COMPRESSION OF 3D MODELS WITH NURBS

THÈSE N° 2770 (2003)

PRÉSENTÉE À LA FACULTÉ SCIENCES ET TECHNIQUES DE L'INGÉNIEUR

Institut de traitement des signaux

SECTION D'ÉLECTRICITÉ

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES TECHNIQUES

PAR

Diego SANTA CRUZ DUCCI

ingénieur électricien diplômé EPF
et de nationalité chilienne et italienne

acceptée sur proposition du jury:

Prof. T. Ebrahimi, directeur de thèse
Prof. A. Baskurt, rapporteur
Prof. H. Bunke, rapporteur
Prof. B. Macq, rapporteur
Dr. G. Taubin, rapporteur
Prof. D. Thalmann, rapporteur

Lausanne, EPFL
2003

To my beloved Joëlle and Dorian.

The significant problems we face cannot be solved at the same level of thinking we were at when we created them.

Albert Einstein

Acknowledgments

The process carrying out this research and writing this thesis has been a wonderful adventure and a very exciting time. Nevertheless, it has not been without its share of pain and I could not have achieved this work without the help, support and encouragement of many around me.

First of all I would like to thank Prof. Murat Kunt who persuaded me of doing a PhD and offered me a position in the lab. I would like to thank Prof. Touradj Ebrahimi, my thesis supervisor, for the many initial discussions on the subject, for the freedom he gave me in carrying out this research and his unconditional support. He has also provided me with great opportunities to participate in other projects, such as the JPEG 2000 standardization process, where I have learned a lot. I am grateful to the members of my jury, Prof. Atilla Baskurt, Prof. Horst Bunke, Prof. Benoît Macq, Prof. Juan R. Mosig, Dr. Gabriel Taubin and Prof. Daniel Thalmann, for accepting to be part of the committee and for reading the manuscript.

I am also grateful to all my colleagues that have helped me carry out this work. In particular to Nicolas Aspert, with which the collaboration on developing many 3D tools has been invaluable. Also thanks to Davy Jacquet for developing some initial tools during his student project.

For all their friendship, fruitful discussions and countless unforgettable moments I would like to thank all the Signal Processing Laboratory members. I am also grateful to the secretaries of the laboratory, Marianne Marion, Fabienne Vionnet, Isabelle Bezzi and Corinne Degott, for their help and their patience, and to our system administrator, Gilles Auric, for his logistic support.

My sincere gratitude goes to my wife, Joëlle, to whom I dedicate this thesis together with my son, Dorian. Without her support and encouragement I would have never been able to carry out this PhD. Thanks for always being there, comprehensive and caring. Thanks also to Dorian, for being a lovely child and for his long nights of uninterrupted sleep.

Finally, thanks to my parents who have provided me with great opportunities and encouraged in all my undertakings with great love.

Contents

List of Figures

List of Tables

Version abrégée

Avec les récents progrès de l'informatique et des télécommunications, les modèles 3D sont de plus en plus utilisés dans les applications multimédia. La visualisation, les jeux, le divertissement et la réalité virtuelle comptent parmi les exemples les plus répandus. Dans le domaine du multimédia les modèles 3D ont été traditionnellement représentés comme des maillages polygonaux. Cette représentation plane par morceaux, peut être vue comme l'analogue des images *bitmap* pour les surfaces 3D. Comme les images *bitmap*, ils jouissent d'une grande flexibilité et sont particulièrement bien adaptés pour décrire des informations acquises depuis le monde réel, comme par exemple, lors d'un processus de balayage. Ils souffrent, cependant, des mêmes limitations, notamment une résolution limitée et un grand espace de stockage.

La compression de maillages polygonaux est un domaine de recherche très actif depuis une décennie et des algorithmes de compression efficaces permettant de réduire fortement les besoins en place de stockage, ont été proposés dans la littérature. Cependant, cette description bas-niveau de formes 3D a une performance limitée. Une compression plus efficace devrait être possible avec l'usage de primitives de plus haut niveau. Cette idée a été extensivement explorée dans le contexte du codage à base de modèles de l'information visuelle. Dans une telle approche, lors de la compression de l'information visuelle une représentation de plus haut niveau (par ex. un modèle 3D d'une tête parlante) est obtenue par analyse. Ceci peut être vu comme un problème de projection inverse. Une fois cette tâche accomplie, les paramètres du modèle résultants sont codés à la place de l'information originale. Il est communément admis que si le module d'analyse est suffisamment efficace le coût total de codage (dans le sens débit distorsion) en sera largement réduit.

La performance relativement basse et la haute complexité des méthodes d'analyse existantes (mis à part des cas spécifiques où une connaissance *a priori* de la nature des objets est disponible), a empêché un large déploiement des techniques de codage basées sur une telle approche. Le progrès dans le domaine de l'infographie (*computer graphics*) a néanmoins changé cette situation. En effet, de nos jours, un nombre croissant d'images, vidéos et contenu 3D sont générés par procédés de synthèse au lieu de provenir d'un appareil de capture, tels une caméra ou un scanner. Cela signifie que le modèle sous-jacent dans le stade de synthèse peut être utilisé pour améliorer les performances de codage sans avoir besoin de recourir à un module d'analyse hautement complexe. En d'autres mots, ce serait une erreur que de vouloir essayer de compresser une description bas-niveau (par ex. un maillage polygonal) alors qu'une description de plus haut niveau est disponible dans le processus de synthèse (par ex. une surface paramétrique). Cela est, cependant, ce qui est couramment fait dans le domaine du multimédia, où des descriptions de modèles 3D de haut niveau sont convertis en maillage polygonaux, ne serait-ce que par manque d'un format standard pour le codage de ceux-ci.

Par ailleurs, la façon dont nous consommons l'information audiovisuelle est en train de changer. A l'opposé des anciennes applications et une grande partie des actuelles, l'interactivité est en train de devenir un élément clé de la manière dont nous consommons l'information. Dans le cadre de la

présente dissertation, cela signifie que, lorsque nous codons un information visuelle (par ex. une image ou une vidéo), des considérations évidentes par le passé telles que la sélection des paramètres d'échantillonage n'est plus aussi évidente qu'avant. En effet, à l'instar d'un environnement interactif où la résolution d'affichage effective peut être contrôlée par l'utilisateur à travers un zoom, il n'y a pas de choix optimal clairement défini pour les paramètres d'échantillonage. Cela signifie qu'à cause de l'interactivité, la représentation utilisée pour coder la scène devrait permettre l'affichage des objets dans une large gamme de résolutions et, idéalement, jusqu'à l'infini.

Une façon de résoudre ce problème serait l'utilisation d'un suréchantillonage extensif. Néanmoins, cette approche est irréaliste et trop coûteuse à implanter dans beaucoup de situations. L'alternative serait d'utiliser une représentation indépendante de la résolution. Dans le domaine du modelage 3D, lesdites représentations sont couramment disponibles lorsque les modèles sont crées par un artiste sur un ordinateur.

Le sujet de cette dissertation est précisément la compression de modèles 3D dans une forme de plus haut niveau. Le codage direct sous une telle forme devrait délivrer une performance débit distorsion améliorée tout en procurant un large degré d'indépendance de résolution. Il n'y a pas eu, jusqu'à ce jour, de travaux majeurs sur la compression efficace de telles représentations, telles que les surfaces paramétriques. Cette thèse propose une solution pour combler ce vide.

Une variété de représentations 3D de haut niveau existe, parmi lesquelles les surfaces paramétriques sont un choix répandu parmi les designers. Dans la famille des surfaces paramétriques, les B-Splines rationnelles non-uniformes (NURBS) jouissent d'une grande popularité étant donné qu'une large gamme d'outils basés sur les NURBS sont couramment disponibles. Récemment, les NURBS ont été ajoutées dans le *Virtual Reality Modeling Language* (VRML) ainsi que son descendant de nouvelle génération le *eXtensible 3D* (X3D). Les bonnes propriétés des NURBS et leur utilisation largement répandue nous ont conduit à les choisir comme forme sous laquelle les modèles seront codés.

Le but principal de cette dissertation est la définition d'un système de codage des modèles 3D NURBS avec une distorsion garantie. La base du système est la modulation par impulsion et codage différentiel (DPCM) codée entropiquement. Dans le cas de NURBS, garantir la distorsion n'est pas évident, dès lors que certains de ses paramètres (par ex. nœuds) ont une influence compliquée sur la distorsion totale de la surface. A cette fin, une analyse détaillé de la distorsion est effectuée. En particulier, des relations jusqu'alors inconnues entre la distorsion des nœuds et la distorsion de la surface résultante est démontrée.

L'efficacité de la compression est recherchée à chaque stade et des codeurs entropiques simples mais néanmoins efficaces sont définis. Le cas particulier de surfaces fermées et dégénérées avec des point de contrôle dupliqués est adressé et un codage simple et efficace est proposé pour compresser les relations de duplication.

Les aspects de l'encodeur sont aussi analysés. Des prédicteurs optimaux ayant une bonne performance sur une large classe de modèles sont trouvés. Des techniques de simplification, ayant un coût négligeable sur la distorsion, sont aussi considérées pour une meilleure efficacité de compression.

La transmission sur des canaux présentant un taux d'erreur non négligeable est aussi considérée est une extension de résilience aux erreurs est définie. Le train de données est morcelé en codant indépendamment des petits groupes de surfaces et en insérant les marqueurs de resynchronisation nécessaires. Des stratégies simples pour atteindre le niveau désiré de protection sont proposées. La même extension sert aussi pour l'accès aléatoire et le réordonnancement à la demande du train de données.

Abstract

With recent progress in computing, algorithmics and telecommunications, 3D models are increasingly used in various multimedia applications. Examples include visualization, gaming, entertainment and virtual reality. In the multimedia domain 3D models have been traditionally represented as polygonal meshes. This piecewise planar representation can be thought of as the analogy of bitmap images for 3D surfaces. As bitmap images, they enjoy great flexibility and are particularly well suited to describing information captured from the real world, through, for instance, scanning processes. They suffer, however, from the same shortcomings, namely limited resolution and large storage size.

The compression of polygonal meshes has been a very active field of research in the last decade and rather efficient compression algorithms have been proposed in the literature that greatly mitigate the high storage costs. However, such a low level description of a 3D shape has a bounded performance. More efficient compression should be reachable through the use of higher level primitives. This idea has been explored to a great extent in the context of model based coding of visual information. In such an approach, when compressing the visual information a higher level representation (e.g., 3D model of a talking head) is obtained through analysis methods. This can be seen as an inverse projection problem. Once this task is fulfilled, the resulting parameters of the model are coded instead of the original information. It is believed that if the analysis module is efficient enough, the total cost of coding (in a rate distortion sense) will be greatly reduced.

The relatively poor performance and high complexity of currently available analysis methods (except for specific cases where *a priori* knowledge about the nature of the objects is available), has refrained a large deployment of coding techniques based on such an approach. Progress in computer graphics has however changed this situation. In fact, nowadays, an increasing number of pictures, video and 3D content are generated by synthesis processing rather than coming from a capture device such as a camera or a scanner. This means that the underlying model in the synthesis stage can be used for their efficient coding without the need for a complex analysis module. In other words it would be a mistake to attempt to compress a low level description (e.g., a polygonal mesh) when a higher level one is available from the synthesis process (e.g., a parametric surface). This is, however, what is usually done in the multimedia domain, where higher level 3D model descriptions are converted to polygonal meshes, if anything by the lack of standard coded formats for the former.

On a parallel but related path, the way we consume audio-visual information is changing. As opposed to recent past and a large part of today's applications, interactivity is becoming a key element in the way we consume information. In the context of interest in this dissertation, this means that when coding visual information (an image or a video for instance), previously obvious considerations such as decision on sampling parameters are not so obvious anymore. In fact, as in an interactive environment the effective display resolution can be controlled by the user through zooming, there is no clear optimal setting for the sampling period. This means that because of interactivity, the representation used to code the scene should allow the display of objects in a

variety of resolutions, and ideally up to infinity.

One way to resolve this problem would be by extensive over-sampling. But this approach is unrealistic and too expensive to implement in many situations. The alternative would be to use a resolution independent representation. In the realm of 3D modeling, such representations are usually available when the models are created by an artist on a computer.

The scope of this dissertation is precisely the compression of 3D models in higher level forms. The direct coding in such a form should yield improved rate-distortion performance while providing a large degree of resolution independence. There has not been, so far, any major attempt to efficiently compress these representations, such as parametric surfaces. This thesis proposes a solution to overcome this gap.

A variety of higher level 3D representations exist, of which parametric surfaces are a popular choice among designers. Within parametric surfaces, Non-Uniform Rational B-Splines (NURBS) enjoy great popularity as a wide range of NURBS based modeling tools are readily available. Recently, NURBS has been included in the Virtual Reality Modeling Language (VRML) and its next generation descendant eXtensible 3D (X3D). The nice properties of NURBS and their widespread use has lead us to choose them as the form we use for the coded representation.

The primary goal of this dissertation is the definition of a system for coding 3D NURBS models with guaranteed distortion. The basis of the system is entropy coded differential pulse coded modulation (DPCM). In the case of NURBS, guaranteeing the distortion is not trivial, as some of its parameters (e.g., knots) have a complicated influence on the overall surface distortion. To this end, a detailed distortion analysis is performed. In particular, previously unknown relations between the distortion of knots and the resulting surface distortion are demonstrated.

Compression efficiency is pursued at every stage and simple yet efficient entropy coder realizations are defined. The special case of degenerate and closed surfaces with duplicate control points is addressed and an efficient yet simple coding is proposed to compress the duplicate relationships.

Encoder aspects are also analyzed. Optimal predictors are found that perform well across a wide class of models. Simplification techniques are also considered for improved compression efficiency at negligible distortion cost.

Transmission over error prone channels is also considered and an error resilient extension defined. The data stream is partitioned by independently coding small groups of surfaces and inserting the necessary resynchronization markers. Simple strategies for achieving the desired level of protection are proposed. The same extension also serves the purpose of random access and on-the-fly reordering of the data stream.

1 Introduction

1.1 Context

With recent progress in computing, algorithmics and telecommunications, 3D models are increasingly used in various multimedia applications. Examples include visualization, gaming, entertainment and virtual reality. In the multimedia domain 3D models have been traditionally represented as polygonal meshes. This piecewise planar representation can be thought of as the analogy of bitmap images for 3D surfaces. As bitmap images, they enjoy great flexibility and are particularly well suited to describing information captured from the real world, through, for instance, scanning processes. They suffer, however, from the same shortcomings, namely limited resolution and large storage size.

The compression of polygonal meshes has been a very active field of research in the last decade and rather efficient compression algorithms have been proposed in the literature that greatly mitigate the high storage costs. However, such a low level description of a 3D shape has a bounded performance. More efficient compression should be reachable through the use of higher level primitives. This idea has been explored to a great extent in the context of model based coding of visual information. In such an approach, when compressing a video captured by a camera, for instance, an analysis module attempts to model the scene under consideration as a set of (generally) 3D models. This can be seen as an inverse projection problem. Once this task is successfully fulfilled, instead of coding the image sequence from the video, represented as a set of pixels and their motions, the parameters of the model extracted from the scene are coded. It is believed that if the analysis module is efficient enough, the total cost of coding (in a rate distortion sense) will be greatly reduced.

The relatively poor performance and high complexity of currently available analysis methods (except for specific cases where *a priori* knowledge about the nature of the objects is available), has refrained a large deployment of coding techniques based on such an approach. Progress in computer graphics has however changed this situation. In fact, nowadays, an increasing number of pictures, video and 3D content are generated by synthesis processing rather than coming from a capture device such as a camera or a scanner. This means that the underlying model in the synthesis stage can be used for their efficient coding without the need for a complex analysis module. In other words it would be a mistake to attempt to compress a low level description (e.g., a polygonal mesh)

when a higher level one is available from the synthesis process (e.g., a parametric surface). This is, however, what is usually done in the multimedia domain, where higher level 3D model descriptions are converted to polygonal meshes, if anything by the lack of standard coded formats for the former.

On a parallel but related path, the way we consume audio-visual information is changing. As opposed to recent past and a large part of today's applications, interactivity is becoming a key element in the way we consume information. In the context of interest in this dissertation, this means that when coding visual information (an image or a video for instance), previously obvious considerations such as decision on sampling parameters are not so obvious anymore. To be more clear, consider the following example. In a conventional digital video coding problem, the sampling (i.e. number of pixels in the image) mostly depends on the display resolution. Knowing the resolution of the display, there is no need to acquire and then to code a signal with sampling characteristics beyond those that the display is capable of reproducing.

In an interactive environment, this is not true anymore. Even using a conventional display with, for instance, a resolution of 300×400 pixels, one could request to examine more closely (and therefore display) an object in a scene. This means that because of interactivity, the representation used to code the scene should allow the display of objects in a variety of resolutions, and ideally up to infinity.

One way to resolve this problem would be by extensive over-sampling. But this approach is unrealistic and too expensive to implement in many situations. The alternative would be to use a resolution independent representation. In the realm of 3D modeling, such representations are usually available when the models are created by an artist on a computer.

1.2 Approach

The scope of this dissertation is precisely the compression of 3D models in higher level forms. As explained above, the direct coding in such a form should yield improved rate-distortion performance while providing a large degree of resolution independence. There has not been, so far, any major attempt to efficiently compress these representations, such as parametric surfaces. This thesis proposes a solution to overcome this gap.

A variety of higher level 3D representations exists, of which parametric surfaces are a popular choice among designers. Within parametric surfaces, Non-Uniform Rational B-Splines (NURBS) enjoy great popularity as a wide range of NURBS based modeling tools are readily available. Recently, NURBS has been included in the Virtual Reality Modeling Language (VRML) [113] and its next generation descendant eXtensible 3D (X3D) [114]. The nice properties of NURBS and their widespread use has lead us to choose them as the form we use for the coded representation.

The approach presented in this dissertation draws its origins from the vast literature available in 3D polygonal mesh coding and the related field of image compression. Since the landmark paper of Deering published in 1995 [15] the field of polygonal mesh coding has seen intense activity, leading to standardization in MPEG-4 [49] and very efficient algorithms in the past few years [2, 56, 64, 109].

The primary goal of this dissertation is the definition of a system for coding 3D NURBS models with guaranteed distortion. In the case of NURBS, guaranteeing the distortion is not trivial, as some of its parameters (e.g., knots) have a complicated influence on the overall surface distortion. To this end, a detailed distortion analysis is performed. In particular, previously unknown relations between the distortion of knots and the resulting surface distortion are demonstrated.

Compression efficiency is pursued at every stage and simple yet efficient entropy coder realizations are defined. The special case of degenerate and closed surfaces with duplicate control points is addressed and an efficient yet simple coding is proposed to compress the duplicate relationships.

Encoder aspects are also analyzed. Optimal predictors are found that perform well across a wide class of models. Simplification techniques are also considered for improved compression efficiency at negligible distortion cost.

Transmission over error prone channels is also considered and an error resilient extension defined. The data stream is partitioned by independently coding small groups of surfaces and inserting the necessary resynchronization markers. Simple strategies for achieving the desired level of protection are proposed. The same extension also serves the purpose of random access and on-the-fly reordering of the data stream.

1.3 Main contributions

The main contributions of this dissertation can be summarized as follows.

- Definition of an efficient coding procedure for knot vectors, separating multiplicity and value information.
- Derivation and demonstration of a bound relating the knot quantization error to the resulting parametric surface distortion.
- Definition of a simple yet effective entropy coding procedure for the prediction errors of the various parameter types (knots, coordinates and weights).
- Definition of an efficient entropy coding procedure to compress the duplicate point information of closed and degenerate surfaces.
- Derivation of a detailed distortion analysis for each parameter type leading to the definition of global quantizer settings for a balanced distribution of distortion among the data types.
- Derivation of predictors for optimal compression performance that behave uniformly on a wide class of models.
- Thorough evaluation of rate-distortion performance leading to the definition of heuristics for optimal, in the rate-distortion sense, quantizer settings. These heuristics complement the derivation of the global quantizer settings.
- Definition of a decomposition of the data stream in independent segments to achieve error resilience and random access capabilities.

1.4 Outline

This dissertation is organized as follows. Chapter 2 is dedicated to a review of the existing means for describing the shape of 3D objects. We discuss surface modeling techniques including polygonal meshes and parametric surfaces, with special attention on B-Splines, as well as solid modeling techniques such as boundary representations (B-Reps) and constructive solid geometry. Other surface modeling techniques, such as subdivision and implicit surfaces are also reviewed. Chapter 3 is dedicated to a review of the state-of-the-art in 3D model coding. In particular the coding of 3D polygonal meshes, the most researched topic in 3D compression, is reviewed at length with special attention to fixed rate coding. Prior to this, a brief review of the relevant results of information theory is included.

Chapter 4 presents the main contributions of this dissertation. A coding system for NURBS surfaces, trimmed or not, is presented. A detailed distortion analysis is performed to derive guaranteed distortion bounds and the experimental rate-distortion performance is assessed. Chapter 5 discuss encoder techniques that can be used to achieve optimal compression performance, as well as extensions to operate in error prone environments. Chapter 6 presents several application scenarios demonstrating the viability and interest of the proposed system. Conclusions and an outline of some interesting future research directions are presented in Chapter 7. Finally, the demonstration of the surface distortion bound given a knot quantization error is provided in Chapter A.

2 3D model representation

2.1 Introduction

This chapter is dedicated to a review of the different schemes used to represent the shape of three-dimensional (3D) objects, and their associated properties. The development of the techniques used to represent the 3D models started out of a necessity in the computer aided geometric design community. Later, many of the methods have been adopted and extended in the more general computer graphics field.

The representation of 3D objects can be divided into two main categories: *surface modeling* and *solid modeling*. Surface modeling deals with the problem of representing two-dimensional surfaces embedded in the three-dimensional space. These surfaces might or might not define a volume. Solid modeling extends on the techniques of surface modeling to deal with the representation and manipulation of volumes completely surrounded by surfaces, such as a cube, a sphere or much more complicated objects such as airplanes, buildings, the human body, etc.

This chapter is organized as follows. Section 2.2 briefly introduces polygonal meshes and some of the associated topological concepts. Section 2.3 introduces the general concept of parametric curves and surfaces, while Section 2.4 introduces in more detail one of the most common parametric surface forms, B-Splines, including regular tensor product as well as trimmed surfaces. Other surface representation techniques are briefly described in Section 2.5. Section 2.6 describes the major model properties such as normals, texture coordinates and color. Section 2.7 introduces some of the concepts and special constraints of solid modeling as well as some of the most commonly used techniques. Finally conclusions are drawn in Section 2.8.

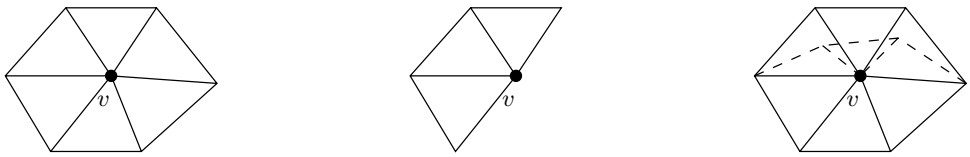
2.2 Polygonal meshes

A *polygonal mesh* [7, 23] describes a surface as a set of connected polygonally bounded planar surfaces. More formally, a mesh $\mathcal{M} = (\mathcal{V}, \mathcal{F})$ is represented by a set $\mathcal{V}$ of points in $\mathbb{R}^3$, called *vertices*, together with a set $\mathcal{F}$ of n-tuples of ordered vertices from $\mathcal{V}$, called *faces*. The n-tuple

$\{v_1, \ldots, v_n\}$ of a face defines a polygon with edges (v_1, v_2),...,(v_{n-1}, v_n) and (v_n, v_1). Each face should form a planar polygon. Clearly, adjacent polygons share some of their vertices and edges. We call the neighborhood $N(v)$ of a vertex v the set of vertices that share an edge with v. It is also referred to as the *1-ring* of v. The number of edges incident on a vertex is referred to as the *valence* or *degree* of the vertex.

A *triangular mesh* is a mesh in which all the faces are triangles. Any polygonal mesh can be transformed into a triangular mesh by triangulating each polygonal face.

A mesh is said to be a *2-manifold*, or simply a manifold, if for every vertex v the faces incident on v are homeomorphic to a disk, or a semi-disk if v is on a *boundary*. A manifold mesh will thus be locally planar at all vertices. Figure 2.1 shows examples of manifold and non-manifold cases. An alternative, more abstract definition for triangular meshes is as follows: a triangular mesh is manifold if for each vertex v in V, the vertices of $N(v)$ can be sorted in a list $\{n_1(v), \ldots, n_k(v)\}$ such that for each couple $(n_i(v), n_{i+1}(v))$ the edges $(n_i(v), v)$ and $(v, n_{i+1}(v))$ belong to a same face. In a manifold, each edge is either shared by two faces or belongs to one face only. They are referred to as *internal* and *external* edges, respectively. The closed loops of external edges form the boundaries of the manifold. A manifold without external edges is said without boundary.



(a) manifold **(b)** boundary manifold **(c)** non-manifold

Figure 2.1: Examples of manifold classification for the center vertex v.

Each face has an *orientation* that is determined by the order of the vertices in the n-tuple describing it, which defines an up and a down side for the face. If the order is reversed, the orientation is also reversed. Two faces f_1 and f_2 have a *consistent orientation* if for each shared edge its vertices v_1 and v_2 appear in opposite orders in the n-tuples describing f_1 and f_2. In other words, two faces are consistently oriented if their up sides are oriented in the same direction. A mesh is said *orientable* if all of its faces can be consistently oriented. Figure 2.2 shows a simple orientable manifold. Many systems only deal with orientable manifold meshes.

Although polygonal meshes can accurately describe any objects with planar surfaces, such as drawers, boxes, building exteriors, etc., they can only approximate curved surfaces. However, this approximation can be made arbitrarily close to the curved surface being modeled by using small enough polygons. However, the storage space requirements can increase dramatically as well as the time it takes to run certain algorithms. Furthermore, a polygonal mesh approximation of a curved surface that appears sufficiently close at one scale, will inevitably appear too coarse at sufficiently finer scales.

2.3 Parametric curves and surfaces

The coordinates of the set of points on a surface, or a portion of it, embedded in 3D space can be represented by equations. These can be given either in implicit or parametric form. In the former, the surface is defined by an implicit equation that is satisfied by the points of the surface. In the

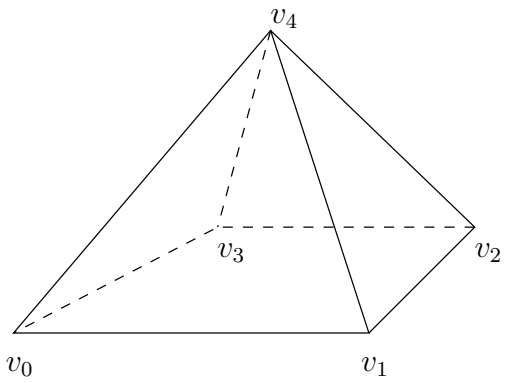


Figure 2.2: Example orientable manifold with five vertices $(v_0, \ldots, v_4)$, four faces (three triangles plus one quadrilateral) and eight edges.

latter, the coordinates of the points on the surface are functions of two independent variables. A surface $\mathbf{S}$ could be defined as

$$\mathbf{S} = \{(x, y, z) \quad | \quad f(x, y, z) = 0\}$$

in implicit form and as

$$\mathbf{S}(u, v) = \big(x(u, v), y(u, v), z(u, v)\big)$$

in parametric form, were u and v are the independent variables or *parameters*.

The implicit form makes it easy to determine if a point is on the surface, and if not, on which side it is located. However, the implicit form does not lend itself to compute the points on the surface in a simple way, when drawing for instance, and even less so to local modifications of the shape. Furthermore, it is very difficult to model free-form objects using the implicit form. Conversely, the parametric form naturally allows to compute the points on the surface by simply evaluating the coordinate functions within the parameter domain. In addition, many forms of parametrization allow local modifications of the surface to be carried out in a fairly straightforward way, as we will see later. However, the problem of determining if a given point is on the surface is not straightforward, but is typically less of a concern since it arises less often.

In a very similar manner, a curve in 3D space can be defined parametrically as

$$\mathbf{C}(u) = \big(x(u), y(u), z(u)\big)$$

where only one parameter, u, is required. This can be regarded as a mapping from an interval of $\mathbb{R}$ into Euclidean $\mathbb{E}^3$, which is more typically expressed as

$$\mathbf{C}(u) = \sum_{i=0}^{n-1} f_i(u)\mathbf{P}_i \qquad u_{\min} \leq u \leq u_{\max}$$

where $\mathbf{P}_i$ is the set of n *control points*, collectively referred to as the *control polygon*, and $\{f_i(u)\}$ the set of parametric functions. For a curve in 2D space, the third coordinate is left out. Note that while curves in 2D space can also be expressed in implicit form, it is not possible to do so for curves in 3D space, except as intersections of two surfaces defined in implicit form. In the following we will concentrate on surfaces, however all remarks can be transposed to curves by analogy.

A parametric surface can be regarded as a mapping from a region of $\mathbb{R}^2$ into the Euclidean $\mathbb{E}^3$ space. There are several ways to perform this mapping. One of the most widely used is *tensor product surfaces*. They are defined as

$$\mathbf{S}(u,v) = \sum_{i=0}^{n-1}\sum_{j=0}^{m-1} f_i(u)g_j(v)\mathbf{P}_{i,j} \qquad \begin{matrix} u_{\min} \le u \le u_{\max} \\ v_{\min} \le v \le v_{\max} \end{matrix} ,$$

where $\mathbf{P}_{i,j}$ is the set of $n \times m$ control points, which are collectively referred to as the *control net*, and $\{f_i(u)\}$ and $\{g_j(v)\}$ the sets of parametric functions. Note that the domain of definition in $\mathbb{R}^2$ is rectangular. For this reason, tensor product surfaces are often referred to as *rectangular* patches. The control points are topologically arranged in a rectangular grid, and can be organized in a matrix of n rows and m columns to obtain the following matrix formulation

$$\mathbf{S}(u,v) = [f_i(u)]^T[\mathbf{P}_{i,j}][g_j(v)] ,$$

where $[f_i(u)]$ and $[g_j(v)]$ are length n and m column vectors, respectively, and ^{T} denotes the transpose operator. Each of the m columns of the $[\mathbf{P}_{i,j}]$ matrix blends the n parametric functions of u, $\{f_i(u)\}$, with one parametric function of v. Analogously, each of the n rows of $[\mathbf{P}_{i,j}]$ blends the m parametric functions of v, $\{g_j(v)\}$, with one parametric function of u.

Although strictly speaking any function could be used to generate a parametric surface only polynomials and rational functions, as well as piecewise variations of these, are employed in modeling. While being the most flexible approach, arbitrary functions are too general and no useful properties for surface modeling can be derived. On the contrary, piecewise polynomial and rational functions possess a set of very useful properties, while still allowing for great flexibility in the shape, as we will see.

A natural way to derive a parametric surface from polynomial functions would be to express it in power basis form. Thus $f_i(u) = u^i$ and $g_j(v) = v^j$. Although straightforward in formulation, this representation has several major drawbacks. While the power basis form can be efficiently evaluated using Horner's method

$$\sum_{i=0}^{n-1} a_i u^i = ((\ldots((a_{n-1}u + a_{n-2})u + a_{n-3})u + \ldots)u + a_1)u + a_0 ,$$

a slight variation of the value of some a_i can induce a large variation of the result, for large i, which would lead to a sensibly different geometric shape. Furthermore, the control points have no geometric relationship to the actual shape, making it very difficult to interactively create and modify surfaces. For these reasons, other schemes have been derived, which are explained in the following sections.

2.4 B-Splines

Tensor product parametric surfaces are obtained by interpolating or approximating a set of control points using a given set of parametric functions. As was mentioned, the power basis for polynomial functions is not adequate, but other piecewise polynomial bases exist that are much more suitable. Among them, the B-Spline functions, which stand for Basis Spline, provide a very flexible framework on which parametric surfaces can be built. Polynomial as well as rational constructions are possible, allowing not only to model free-form surfaces, but also to precisely describe common analytical shapes such as spheres, cylinders, ellipsoids, etc. Rational B-Splines have become the standard for curve and surface description in computer graphics. They are commonly known as *NURBS*, which

stands for Non-Uniform Rational B-Splines*. NURBS have been recently added as an extension [30, 113] to the Virtual Reality Modeling Language [47] and are a proposed part of its successor, the eXtensible 3D (X3D) standard [114]. They are also part of the widely used OpenGL™ 3D rendering API (Application Programming Interface) [119]. Besides B-Splines other suitable bases do exist, such as Bernstein polynomials and the Bézier curves and surfaces they generate [21, 29]. However, the B-Spline functions form a unifying foundation and Bézier surfaces are a special case of B-Splines, as explained in Section 2.4.2. Thus we limit our coverage to B-Splines only.

The theory of B-Splines was first suggested by Schoenberg [93, 94] and Riesenfeld [87] and Gordon and Riesenfeld [28, 29] applied the B-Spline basis to curve and surface definitions. Versprille [111] was the first to discuss rational B-Splines. In the following sections we give an overview of B-Splines and their application to tensor product surfaces. A more detailed introduction can be found in [21], [20] and [88] while an in-depth coverage can be found in [80] and [22].

2.4.1 B-Spline functions

The basis functions used to define B-Spline curves and surfaces are the B-Spline functions. They are piecewise polynomial functions defined as follows. Given a non-decreasing sequence of r real numbers $U = \{u_0, u_1, \ldots, u_{r-1}\}$, the ith B-Spline function of pth degree (i.e. order $p+1$), $N_{i,p}(u)$, is recursively defined as

$$N_{i,0}(u) = \begin{cases} 1 & \text{if } u_i \leq u < u_{i+1} \\ 0 & \text{otherwise} \end{cases} \tag{2.1a}$$

$$N_{i,p}(u) = \frac{u - u_i}{u_{i+p} - u_i} N_{i,p-1}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}(u) \,, \tag{2.1b}$$

where the 0/0 quotient is defined to be zero.

The u_i values are referred to as *knots* and U is the *knot vector*. Note that consecutive knots can have equal values: if u_i through u_{i+k-1} are all equal, u_i is said to have *multiplicity* k. The unique values in the knot vector are referred to as *breakpoints*. The semi-open interval $[u_i, u_{i+1})$ is called the ith knot span.

The B-Spline functions have the following properties:

1. for a knot vector with r values there are $n = r - (p+1)$ basis functions of degree p;
2. local support: $N_{i,p}(u)$ is non-zero only in the interval $[u_i, u_{i+p+1})$;
3. non-negativity: $N_{i,p}(u) \geq 0 \; \forall \; u \in [u_i, u_{i+p+1})$;
4. in the ith knot span $[u_i, u_{i+1})$ at most $p+1$ of the B-Spline functions of degree p are non-zero, namely $N_{i-p,p}, \ldots, N_{i,p}$;
5. partition of unity: $\sum_{j=0}^{n} N_{j,p}(u) = \sum_{j=i-p}^{i} N_{j,p}(u) = 1 \; \forall \; u \in [u_i, u_{i+1})$;
6. for $p > 0$, $N_{i,p}(u)$ attains exactly one maximum value;
7. the n $N_{i,p}(u)$ functions defined on a given knot vector U are linearly independent.

The B-Spline basis functions are smooth functions. In fact, all the derivatives of $N_{i,p}(u)$ exist at the interior of a knot span. At a knot u_j, $N_{i,p}(u)$ is $p - k_{i,j}$ times continuously derivable, where $k_{i,j}$ is the multiplicity of u_j, as seen by $N_{i,p}(u)$. Note that, since $N_{i,p}(u)$ is identically zero outside

*As pointed out by Piegl and Tiller [80], in the early 90's some researchers joked in that the acronym stands for Nobody Understands Rational B-Splines!

the interval $[u_i, u_{i+p+1})$, the multiplicity $k_{i,j}$ as seen by $N_{i,p}(u)$ is the multiplicity of u_j in the set $\{u_i, \ldots, u_{i+p+1}\}$, and not in the set U. Using C^i notation, the B-Spline functions are C^∞ continuous everywhere except at knot values, where they are $C^{(p-k_{i,j})}$ continuous.

The first derivative is calculated as

$$N'_{i,p}(u) = \frac{p}{u_{i+p} - u_i} N_{i,p-1}(u) - \frac{p}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}(u) \tag{2.2}$$

and the kth derivative is

$$N^{(k)}_{i,p}(u) = p \left(\frac{N^{(k-1)}_{i,p-1}(u)}{u_{i+p} - u_i} - \frac{N^{(k-1)}_{i+1,p-1}(u)}{u_{i+p+1} - u_{i+1}} \right) . \tag{2.3}$$

The following non-recursive formulation is also given in [80] for $k \leq p$

$$N^{(k)}_{i,p}(u) = \frac{p!}{(p-k)!} \sum_{j=0}^{k} a_{k,j} N_{i+j,p-k}(u) , \tag{2.4}$$

where

$$\begin{aligned}
a_{0,0} &= 1 \\
a_{k,0} &= \frac{a_{k-1,0}}{u_{i+p-k+1} - u_i} \\
a_{k,j} &= \frac{a_{k-1,j} - a_{k-1,j-1}}{u_{i+p+j-k+1} - u_{i+j}} \qquad j = 1, \ldots, k-1 \\
a_{k,k} &= \frac{-a_{k-1,k-1}}{u_{i+p+1} - u_{i+k}} .
\end{aligned}$$

Note that the denominators above can become zero, in which case the quotient is considered to be 0. For $k > p$, $N^{(k)}_{i,p}(u) \equiv 0$.

Now that we have established the continuity properties of B-Spline basis functions let's consider $\mathcal{V}$, the vector space of all the piecewise polynomials of degree p that are C^{r_j} continuous at u_j and C^∞ continuous elsewhere; where $\{u_j\}$, $0 \leq j \leq l$, is a strictly increasing set of breakpoints. Then the B-Spline basis functions of pth degree defined on the knot vector

$$U = \{\underbrace{u_0, \ldots, u_0}_{p-r_o}, \underbrace{u_1, \ldots, u_1}_{p-r_1}, \ldots, \underbrace{u_l, \ldots, u_l}_{p-r_l}\}$$

form a basis of $\mathcal{V}$. Therefore, any piecewise polynomial function with given continuity constraints can be expressed as a linear combination of B-Spline functions over an appropriately defined knot vector. This justifies the statement that B-Splines form a unifying foundation for piecewise polynomial and rational curves and surfaces.

Although knot vectors can be an arbitrary sequence of non-decreasing real values, some classes provide additional properties to the B-Spline functions. The following classification is typically found in the literature:

clamped if the knot vector is of the form

$$U = \{\underbrace{a, \ldots, a}_{p+1}, u_{p+1}, \ldots, u_{r-p-2}, \underbrace{b, \ldots, b}_{p+1}\} ;$$

uniform if all the knots are uniformly spaced, that is $u_{i+1} - u_i = c$, $0 \leq i \leq r-2$;

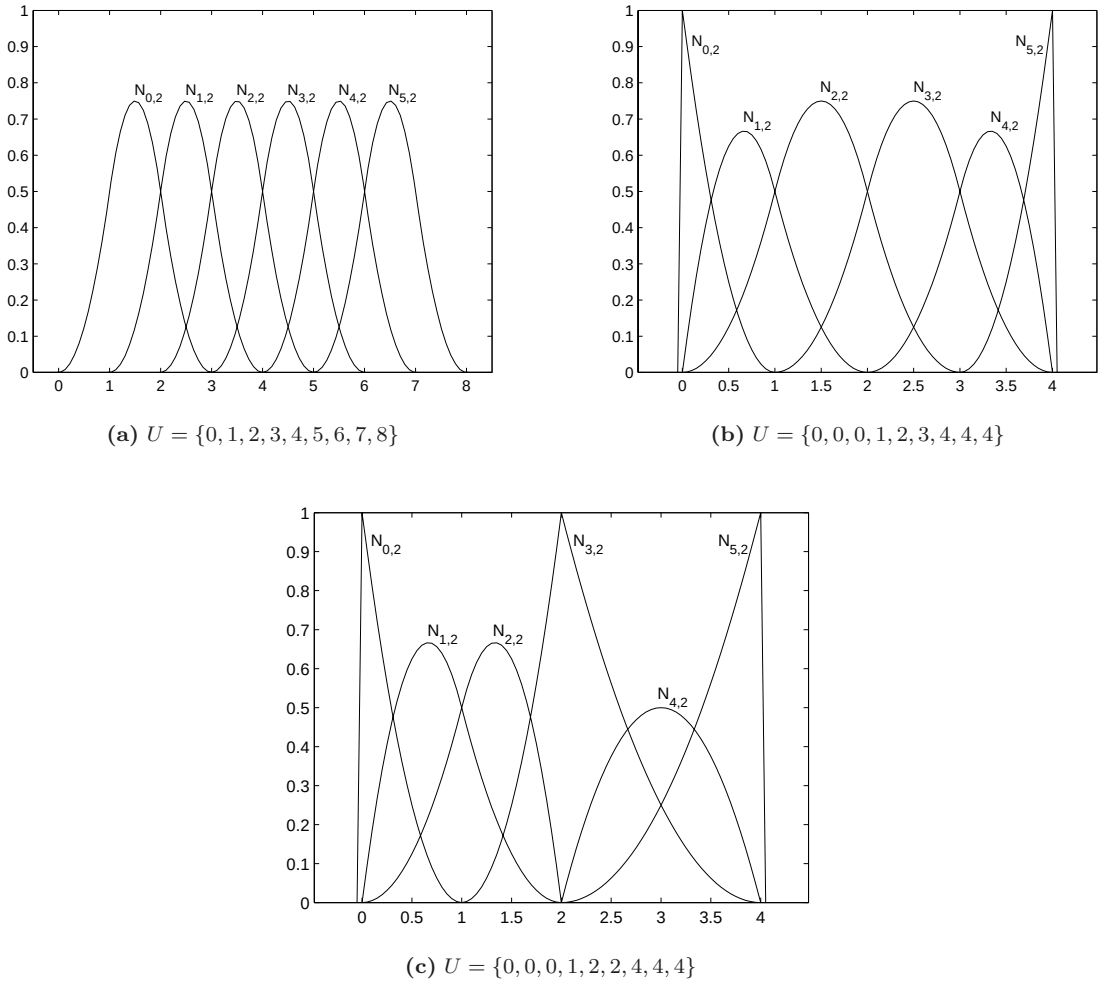
clamped uniform if the knot vector is clamped and the interior knots are uniformly spaced;

periodic if the knot vector is non-clamped uniform.

Depending on the authors, clamped knots are often referred to as *non-periodic* or *open*. In the remaining of this dissertation we shall only use the *clamped* term, since the other ones are less intuitive and misleading at times. Note that the notion of clamped is with respect to the degree of the B-Spline functions the knot vector defines.

When a knot vector is clamped B-Spline functions have the following additional properties:

$$N_{0,p}(a) = 1 \text{ and } N_{i,p}(a) = 0 \text{ for } i \neq 0$$
$$N_{n-1,p}(b) = 1 \text{ and } N_{i,p}(b) = 0 \text{ for } i \neq n-1.$$



(a) $U = \{0, 1, 2, 3, 4, 5, 6, 7, 8\}$

(b) $U = \{0, 0, 0, 1, 2, 3, 4, 4, 4\}$

(c) $U = \{0, 0, 0, 1, 2, 2, 4, 4, 4\}$

Figure 2.3: The quadratic B-Spline functions for different knot vectors U.

Figure 2.3 shows the shape of the quadratic B-Spline functions for various types of knot vectors. Figure 2.3(a) shows the periodic knot vector. As can be seen, all B-Spline functions are translated

versions of the first one, hence the term periodic. Figure 2.3(b) shows the typical clamped uniform knot vector, while Figure 2.3(c) shows a similar knot vector but where one of the internal knots has multiplicity 2. At that knot the continuity is reduced from C^1 to C^0.

2.4.2 B-Spline surfaces

Given the knot vectors $U = \{u_0, \ldots, u_{r-1}\}$ and $V = \{v_0, \ldots, v_{s-1}\}$ a polynomial tensor product B-Spline surface of degree p and q in the u and v directions, respectively, is defined as

$$\mathbf{S}(u,v) = \sum_{i=0}^{n-1}\sum_{j=0}^{m-1} N_{i,p}(u)N_{j,q}(v)\mathbf{P}_{i,j} \qquad \begin{matrix} u_p \le u \le u_{r-p-1} \\ v_q \le v \le v_{s-q-1} \end{matrix}, \tag{2.5}$$

where $n = r-(p+1)$ and $m = s-(q+1)$. The rational counterpart is obtained by defining a polynomial surface in projective $\mathbb{P}^4$ space and projecting it to Euclidean $\mathbb{E}^3$ space. Denoting homogeneous coordinates by the ^{w} superscript, a B-Spline rational surface of degrees p and q is obtained as

$$\mathbf{S}^w(u,v) = \sum_{i=0}^{n-1}\sum_{j=0}^{m-1} N_{i,p}(u)N_{j,q}(v)\mathbf{P}^w_{i,j} \qquad \begin{matrix} u_p \le u \le u_{r-p-1} \\ v_q \le v \le v_{s-q-1} \end{matrix}, \tag{2.6}$$

where $\{\mathbf{P}^w_{i,j}\}$ are the control points in projective space. Projecting this to Euclidean space one obtains

$$\mathbf{S}(u,v) = \sum_{i=0}^{n-1}\sum_{j=0}^{m-1} R_{i,j}(u,v)\mathbf{P}_{i,j} \qquad \begin{matrix} u_p \le u \le u_{r-p-1} \\ v_q \le v \le v_{s-q-1} \end{matrix}, \tag{2.7}$$

where

$$R_{i,j}(u,v) = \frac{N_{i,p}(u)N_{j,q}(v)w_{i,j}}{\sum_{k=0}^{n-1}\sum_{l=0}^{m-1} N_{k,p}(u)N_{l,p}(v)w_{k,l}} \tag{2.8}$$

are the rational basis functions. The p and q subscripts of $R_{i,j}$ are omitted for brevity. Here $w_{i,j}$ is the homogenizing coordinate of $\mathbf{P}^w_{i,j}$, while $\mathbf{P}_{i,j}$ are the corresponding affine coordinates. That is, if $\mathbf{P}_{i,j} = (x_{i,j}, y_{i,j}, z_{i,j})$ then $\mathbf{P}^w_{i,j} = (x_{i,j}w_{i,j}, y_{i,j}w_{i,j}, z_{i,j}w_{i,j}, w_{i,j})$. The $w_{i,j}$ quantity is called the *weight* of control point $\mathbf{P}_{i,j}$. Note that the polynomial form of Eq. (2.5) is a special case of the rational form of Eq. (2.7), where all the weights are equal. Strictly speaking, rational surfaces defined as above are not tensor product once they are projected into Euclidean space, but the term is retained nevertheless.

If the weights are restricted to be positive, the rational basis functions are always well defined (i.e. the denominator is non-zero) over the entire parametric domain of the surface. Furthermore, the properties enumerated above for the univariate B-Spline functions carry on in an analogous fashion to the bivariate rational functions, which provides NURBS surfaces with very desirable properties, outlined below. While some authors have used zero and/or negative weights [88], in practice only positive weights are used. In fact, they are enforced by standards such as VRML [113] and X3D [114].

The main properties of NURBS surfaces with positive-only weights are:

1. projective and affine transformation invariance;

2. strong convex hull property: if $(u,v) \in [u_{i_0}, u_{i_0+1}) \times [v_{j_0}, v_{j_0+1})$, then $S(u,v)$ is in the convex hull of the control points $\mathbf{P}_{i,j}$, $i_0 - p \le i \le i_0$ and $j_0 - p \le q \le j_0$;

3. local control: if $\mathbf{P}_{i,j}$ or $w_{i,j}$ is modified it affects the surface only for $(u,v) \in [u_i, u_{i+p+1}) \times [v_j, v_{j+q+1})$;

4. $\mathbf{S}(u,v)$ is $C^{(p-k)}$ continuous in the u direction at u knots of multiplicity* k, $C^{(q-k)}$ continuous in the v direction at v knots of multiplicity k and C^{∞} continuous everywhere else;

5. if both the U and V knot vectors are clamped the NURBS surface interpolates the four corner control points ($\mathbf{P}_{0,0}$, $\mathbf{P}_{n-1,0}$, $\mathbf{P}_{0,m-1}$ and $\mathbf{P}_{n,m}$)

6. if both knot vectors are of the form $U = \{\underbrace{0,\ldots,0}_{p+1},\underbrace{1,\ldots,1}_{p+1}\}$ and $V = \{\underbrace{0,\ldots,0}_{q+1},\underbrace{1,\ldots,1}_{q+1}\}$ the resulting NURBS surface is a Bézier surface [80] with the same control points (i.e. the B-Spline functions reduce to the Bernstein polynomials).

In general a NURBS surface does not interpolate its control points, except at places where the knot multiplicities lead to C^0 continuity in both directions. A very attractive property of NURBS is their local control, since the modification of a control point only affects the surface shape in its neighborhood. Therefore, it is possible to obtain large surfaces and their shape can be easily controlled. Also, the number of control points is decoupled from the degree of the NURBS, making it feasible to use large control nets to obtain complex shapes without numerical instabilities or major complexity problems. This is to be contrasted to the popular Bézier surface for which the number of control points is directly related to its degree. In that case multiple adjacent patches are needed to obtain moderately complex shapes, requiring extra inter-patch constraints in the control point placement to ensure the desired surface smoothness at the patch borders. In addition, it is relatively easy to add knots and control points without altering the NURBS shape, using the well known knot insertion algorithm (see Section 2.4.4). The extra knots and control points are typically used to reduce the neighborhood affected by the modification of some control point or weight.

As we have seen above the parametric continuity of a NURBS surface can be adjusted with the knot vectors. However, this continuity does not always reflect the characteristics of the surface itself and the definition of *geometric continuity* is necessary. We use G^i notation to denote geometric continuity, instead of C^i. A surface is said to be G^1 continuous if a unique tangent plane exists at every point. More generally, geometric continuity can be defined as in [21, sec. 18.7]: a surface is G^i continuous if, for every point, a C^i parameterization exists on its neighborhood. A NURBS surface which is C^i continuous will therefore be G^i continuous in general, except at particular locations where that continuity can be raised by special alignment of neighboring control points or lowered by using multiple coincident control points.

Surface derivatives

The parametric derivatives of a purely polynomial NURBS can be obtained by directly deriving its B-Spline functions to obtain

$$\frac{\partial^{k+l}}{\partial^k u \partial^l v}\mathbf{S}(u,v) = \sum_{i=0}^{n-1}\sum_{j=0}^{m-1} N_{i,p}^{(k)}(u) N_{j,q}^{(l)}(v)\mathbf{P}_{i,j}. \tag{2.9}$$

For surfaces with U and V clamped knot vectors the above expression simplifies to

$$\frac{\partial^{(k+l)}}{\partial^k u \partial^l v}\mathbf{S}(u,v) = \sum_{i=0}^{n-k-1}\sum_{j=0}^{m-l-1} N_{i,p-k}(u) N_{j,q-l}(v)\mathbf{P}_{i,j}^{(k,l)}, \tag{2.10}$$

*For NURBS surfaces, as well as curves, the relevant multiplicity of a knot u_j is the maximum multiplicity with respect to all non-zero basis functions, which is simply the total multiplicity of u_j in U.

where $N_{i,p-k}(u)$ and $N_{j,q-l}(u)$ are evaluated on the knot vectors

$$U^{(k)} = \{u_k, \ldots, u_{r-k-1}\},$$
$$V^{(l)} = \{v_l, \ldots, v_{s-l-1}\},$$

and

$$\mathbf{P}_{i,j}^{(k,l)} = \begin{cases} \mathbf{P}_{i,j} & k = 0 \text{ and } l = 0, \\ \frac{p-k+1}{u_{i+p+1}-u_{i+k}} \left(\mathbf{P}_{i+1,j}^{(k-1,l)} - \mathbf{P}_{i,j}^{(k-1,l)} \right) & k > 0, \\ \frac{q-l+1}{v_{j+q+1}-v_{j+l}} \left(\mathbf{P}_{i,j+1}^{(k,l-1)} - \mathbf{P}_{i,j}^{(k,l-1)} \right) & l > 0. \end{cases}$$

Note that the derivatives are purely polynomial NURBS surfaces as well. In particular, the first derivatives are given by

$$\mathbf{S}_u = \frac{\partial}{\partial u}\mathbf{S}(u,v) = \sum_{i=0}^{n-2}\sum_{j=0}^{m-1} N_{i,p-1}(u) N_{j,q}(v) \frac{p}{u_{i+p+1} - u_{i+1}} (\mathbf{P}_{i+1,j} - \mathbf{P}_{i,j}), \tag{2.11}$$

$$\mathbf{S}_v = \frac{\partial}{\partial v}\mathbf{S}(u,v) = \sum_{i=0}^{n-1}\sum_{j=0}^{m-2} N_{i,p}(u) N_{j,q-1}(v) \frac{q}{v_{i+q+1} - v_{i+1}} (\mathbf{P}_{i,j+1} - \mathbf{P}_{i,j}). \tag{2.12}$$

For rational NURBS surfaces the derivatives in Euclidean $\mathbb{E}^3$ space involve denominators to high powers and are not NURBS surfaces. Nonetheless, Piegl and Tiller [80] give a rather simple expression for the first derivative. Given the surface in homogeneous space as $\mathbf{S}^w(u,v)$ let $\mathbf{A}(u,v)$ be the surface obtained by considering only the first three coordinates of $\mathbf{S}^w(u,v)$ and $w(u,v)$ the homogenizing coordinate of $\mathbf{S}^w(u,v)$. The first derivative of $\mathbf{S}(u,v)$ is then given by

$$\mathbf{S}_\alpha(u,v) = \frac{\mathbf{A}_\alpha(u,v) - w_\alpha(u,v)\mathbf{S}(u,v)}{w(u,v)}, \tag{2.13}$$

where the α subscript denotes the partial derivative on either u or v.

Examples

Figure 2.4 shows a simple biquadratic polynomial B-Spline surface defined over uniform clamped knot vectors. As it can be seen the resulting surface approximates the control net and has at least C^1 continuity at all points. Figure 2.5 shows the effect that the translation of a control point has on the resulting surface. As can be seen the surface is only modified in the neighborhood of the control point, as required by property 3 above. All points of the surface in that region are translated in the same direction as the control point. The amount of translation is maximum close to the modified control point and diminishes as we move farther from it. Figure 2.6 shows the effect of weight modification. If the weight of a control point is increased all the surface points within its neighborhood are pulled toward it. On the other hand, if it is decreased the surface points are pulled away from the control point. Figures 2.6(c) and 2.6(d) demonstrate the effect of negative weights. As can be clearly seen the use of negative weights breaks the convex hull property 2 above. Furthermore, negative weights produce a very irregular parametrization: in the center of the region affected by the control point a small portion of the parametric domain is mapped to a large portion of the surface. Negative weights can also introduce zero roots in the denominator of Eq. (2.8), leading to evaluation problems and numeric instabilities. Because of this, negative weights are avoided in practical systems, as previously mentioned.

Figure 2.7 shows a complex surface modeled as a single NURBS patch. The NURBS is cubic in v, the direction of the axis, and quadratic in u, the perpendicular direction. The clamped non-uniform

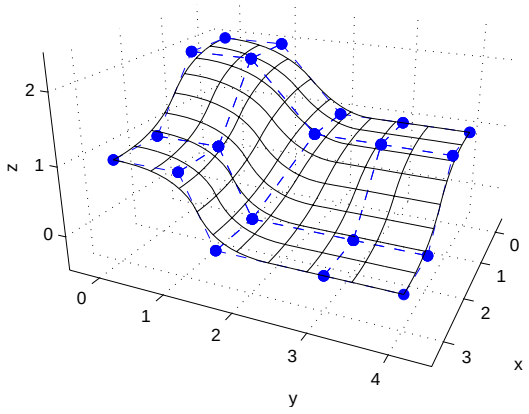


Figure 2.4: A simple biquadratic polynomial B-Spline surface with knot vectors $U = \{0, 0, 0, 1, 2, 2, 2\}$ and $V = \{0, 0, 0, 1, 2, 3, 3, 3\}$. The dashed lines and solid bullets show the control net and points.

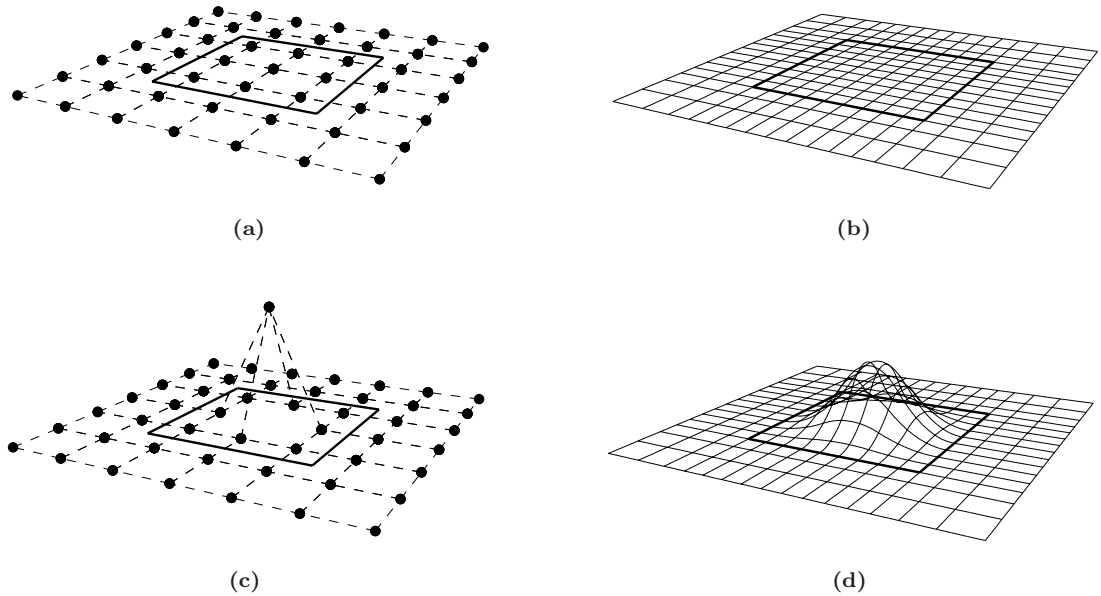
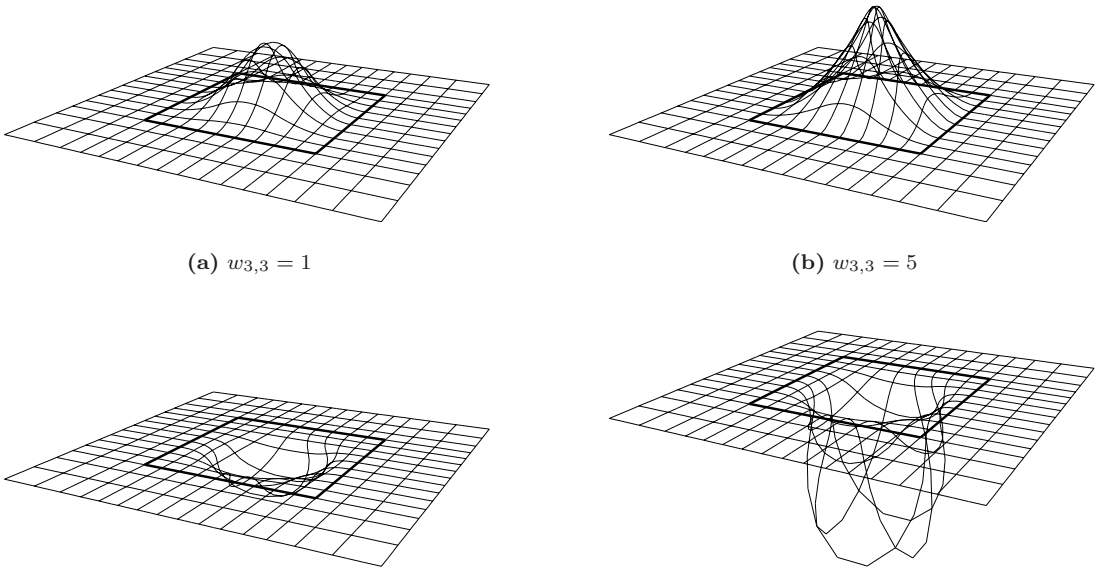


Figure 2.5: The effect of control point modification on a biquadratic NURBS surface, where all control points have unity weights. The thick lines delimit the region affected by the modification of the central control point. The (a) and (c) plots show the control nets while (b) and (d) the resulting surfaces.



(a) $w_{3,3} = 1$ **(b)** $w_{3,3} = 5$

(c) $w_{3,3} = -0.2$ **(d)** $w_{3,3} = -0.5$

Figure 2.6: The effect of weight modification on the surface of figure Figure 2.5(c). The thick lines delimit the region affected by the modification of the central weight.

knot vectors are

$$U = \{0, 0, 0, 0.25, 0.25, 0.5, 0.5, 0.75, 0.75, 1, 1, 1\}$$
$$V = \{0, 0, 0, 0, 0.05, 0.1, 0.124, 0.15, 0.2, 0.225, 0.25, 0.3, 0.325, 0.348,$$
$$0.423, 0.494, 0.564, 0.585, 0.618, 0.684, 0.805, 1, 1, 1, 1\}$$

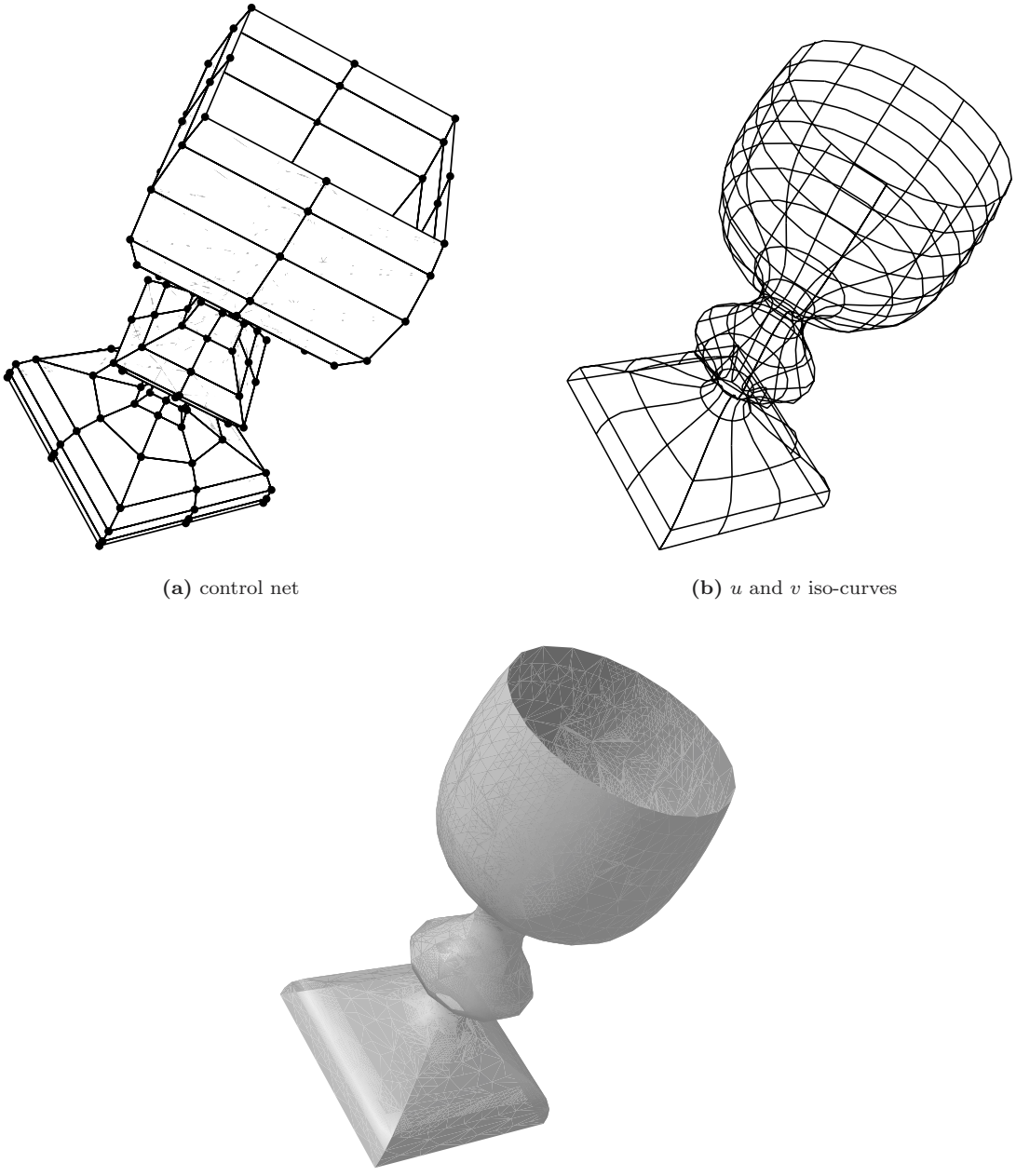
This example shows the flexibility of the NURBS formulation: while the top of the goblet is a surface of revolution, the bottom is a square base exhibiting sharp angles. The internal knots of U are all of multiplicity two, leading to C^0 continuity in the parameter space. However, geometrically speaking, the surface is G^∞ continuous in u at the top of the goblet (circle) and G^0 continuous at the bottom (square). This difference is solely due to the special placement of the control points and their weights. The homogeneous coordinates of the control points along the top of the goblet are

$$\begin{pmatrix} 1 \\ -0.24 \\ 0 \\ 1 \end{pmatrix}, \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ -0.24 \\ -0.24 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \\ -0.24 \\ 1 \end{pmatrix}, \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 0.24 \\ -0.24 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 0.24 \\ 0 \\ 1 \end{pmatrix}, \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 0.24 \\ 0.24 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \\ 0.24 \\ 1 \end{pmatrix}, \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ -0.24 \\ 0.24 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ -0.24 \\ 0 \\ 1 \end{pmatrix},$$

while along the bottom they are

$$\begin{pmatrix} 0 \\ -0.339 \\ 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 \\ -0.170 \\ -0.170 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ -0.339 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 \\ 0.170 \\ -0.170 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 \\ 0.339 \\ 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 \\ 0.170 \\ 0.170 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 0.339 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 \\ -0.170 \\ 0.170 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 \\ -0.339 \\ 0 \\ 1 \end{pmatrix}.$$

These control points form a square, at the bottom as well as at the top. However, the ones at the top have weights of $1/\sqrt{2}$ at the corners, which change the shape from a square into a circle. Finally note that the surface is closed in the u direction, since the first and last control points along the u direction are always equal and that U is clamped.



(a) control net

(b) u and v iso-curves

(c) shaded rendering

Figure 2.7: The surface of a goblet modeled as one truly rational NURBS patch. The patch is cubic in the direction of the height and quadratic in the other, perpendicular, one.

Remark on end knot values

As a concluding remark let us note that for a given knot vector U the values of the first and last knots, u_o and u_{r-1}, have no influence on the shape of the surface $\mathbf{S}(u,v)$. Likewise for the first and last knots of V, v_0 and v_{s-1}. In fact, each function $N_{i,p}(u)$ is defined exclusively by the knots $\{u_i,\ldots,u_{i+p+1}\}$ and thus $N_{0,p}(u)$ is the only function affected by u_o. Although $N_{0,p}(u)$ is non-zero only on the interval $[u_o,u_{p+1})$, the surface is defined over $[u_p,u_{r-p-1}]$ and thus $[u_p,u_{p+1})$ is the interval of interest here. If we look at the recursive formula of Eq. (2.1) for $N_{0,p}$, the only non-zero functions of degree $p-1$ over $[u_p,u_{p+1})$ are $N_{1,p-1}$ through $N_{p,p-1}$. Therefore the left term vanishes and, since $N_{1,p-1}$ through $N_{p,p-1}$ are not defined by u_0, the value of u_0 has no effect on $N_{0,p}$ over $[u_p,u_{p+1})$. Thus, the value of u_0 does not affect the surface $\mathbf{S}(u,v)$. Analogously, the values of u_{r-1}, v_0 and v_{s-1} do not affect $\mathbf{S}(u,v)$ either.

2.4.3 B-Spline curves

For completeness let us now provide the definition of NURBS curves. As expected, a pth degree rational B-Spline curve is defined in projective space as the polynomial curve

$$\mathbf{C}^w(u)=\sum_{i=0}^{n-1}N_{i,p}(u)\mathbf{P}_i^w \qquad u_p\le u\le u_{r-p-1}. \tag{2.14}$$

Projecting to Euclidean $\mathbb{E}^3$ space yields

$$\mathbf{C}(u)=\sum_{i=0}^{n-1}R_i(u)\mathbf{P}_i \qquad u_p\le u\le u_{r-p-1}, \tag{2.15}$$

where

$$R_i(u)=\frac{N_{i,p}(u)w_i}{\sum_{k=0}^{n-1}N_{k,p}(u)w_k} \tag{2.16}$$

are the rational basis functions. As previously, the p subscript of $R_i(u)$ is omitted for brevity. All the properties of NURBS surfaces enumerated in the previous section carry over to NURBS curves by analogy and therefore they are omitted here. Let us just note that given a NURBS surface $\mathbf{S}(u,v)$ and a fixed $\bar{v}$, $\mathbf{S}(u,\bar{v})$ is a NURBS curve of parameter u and control points $\{Q_i^w\}$, where

$$\mathbf{Q}_i^w=\sum_{j=0}^{m-1}N_{j,q}(\bar{v})\mathbf{P}_{i,j}^w,$$

and analogously for fixed $\bar{u}$.

The derivatives are computed in a form analogous to that of surfaces. For a purely polynomial curve $\mathbf{C}(u)$ with a knot vector $U=\{u_0,\ldots,u_{r-1}\}$ the first derivative is

$$\mathbf{C}'(u)=\frac{d}{du}\mathbf{C}(u)=\sum_{i=0}^{n-2}N_{i,p-1}(u)\frac{p}{u_{i+p+1}-u_{i+1}}(\mathbf{P}_{i+1}-\mathbf{P}_i), \tag{2.17}$$

where $N_{i,p-1}(u)$ is evaluated on the knot vector $\{u_1,\ldots,u_{r-2}\}$. For a rational curve the first derivative is given by

$$\mathbf{C}'(u)=\frac{\mathbf{A}'(u)-w'(u)\mathbf{C}(u)}{w(u)}, \tag{2.18}$$

where $\mathbf{A}(u)$ is the polynomial curve obtained by considering only the first three coordinates of $\mathbf{C}^w(u)$ and $w(u)$ is the homogenizing coordinate of $\mathbf{C}^w(u)$.

2.4.4 Fundamental NURBS algorithms

In the above sections we have introduced the basic definitions and concepts of NURBS, as well as some examples. Now we introduce some of the fundamental NURBS algorithms that are required for the development of the subsequent chapters.

Knot insertion

Knot insertion refers to the process of adding a new knot to a given knot vector of a curve or surface and recalculating the set of control points so that the shape is not changed, either geometrically or parametrically. This is one of the most important NURBS algorithms since it allows to perform such basic operations such as curve or surface subdivision and addition of control points to increase the flexibility of shape control. We first overview the algorithm for curves and then apply it to surfaces.

Let $\mathbf{C}^w(u) = \sum_{i=0}^{n-1} N_{i,p}(u)\mathbf{P}_i^w$ be a NURBS curve defined on the knot vector $U = \{u_0, \ldots, u_{r-1}\}$. Let $\bar{u} \in [u_k, u_{k+1})$ be the knot to be inserted in U, $\bar{U} = \{u_0, \ldots, u_k, \bar{u}, u_{k+1}, \ldots, u_{r-1}\}$ the resulting knot vector and $\bar{N}_{i,p}(u)$ the B-Spline functions defined on $\bar{U}$. The vector space defined by $\{N_{i,p}(u)\}$ is embedded in the one defined by $\{\bar{N}_{i,p}(u)\}$, which has one more dimension. Therefore, expressing $\mathbf{C}^w(u)$ in terms of $\bar{N}_{i,p}(u)$ amounts to a change of basis and the curve is not changed, either geometrically or parametrically. Note that inserting a knot vector outside the parametric domain of definition of the curve, that is outside $[u_p, u_{r-p-1}]$, makes no practical sense and leads to an undetermined problem as the curve is extended on one of its sides.

Let $\mathbf{Q}_i^w$ denote the new control points. Thus

$$\mathbf{C}^w(u) = \sum_{i=0}^{n-1} N_{i,p}(u)\mathbf{P}_i^w = \sum_{i=0}^{n} \bar{N}_{i,p}(u)\mathbf{Q}_i^w.$$

As demonstrated in [80, pp. 141–144] the new control points can be easily obtained by

$$\mathbf{Q}_i^w = \alpha_i \mathbf{P}_i^w + (1 - \alpha_i)\mathbf{P}_{i-1}^w, \tag{2.19}$$

where

$$\alpha_i = \begin{cases} 1 & i \leq k - p \\ \frac{\bar{u} - u_i}{u_{i+p} - u_i} & k - p + 1 \leq i \leq k \\ 0 & i \geq k + 1 \end{cases}.$$

As can be seen only p new control points need to be computed, the other ones remaining unchanged. Note that, if a knot $\bar{u}$ is inserted multiple times so that its resulting multiplicity is p, one of the new control points lies on the curve at $u = \bar{u}$. More precisely, if u_l is the first knot larger than $\bar{u}$ (i.e. $\bar{u} = u_{l-1} < u_l$) or the last one if none exists, then $\mathbf{C}^w(\bar{u}) = \mathbf{Q}_{l-p-1}^w$. Doing this subdivides (i.e. splits) the curve in two pieces. Furthermore, if each internal knot is inserted so that its multiplicity is p, then the NURBS curve becomes a piecewise Bézier one.

Knot insertion on NURBS surfaces is simply realized by knot insertion as explained above. For example, to insert a new knot in U, the algorithm above is carried out for each one of the m columns of the control point matrix. Similarly, to insert a new knot in V it should be carried out for the n rows of the control point matrix. Like curves, NURBS surfaces can be subdivided by inserting knots with the appropriate multiplicities.

Knot removal

Knot removal is the inverse process of knot insertion. Typical applications are to piece together piecewise Bézier curves into one NURBS curve and to clean up knot vectors after repeated knot

insertion and control point modification during interactive design, to obtain the simplest possible representation. Clearly, it is not always possible to remove a knot from a NURBS curve or surface without altering it, either geometrically or parametrically. Let U be the knot vector of a NURBS curve $\mathbf{C}^w(u)$, with control points $\mathbf{P}_i^w$ and where the knot u_l has multiplicity k. Let $\bar{U}$ be the knot vector resulting of removing t times the knot u_l from U, with $1 \leq t \leq k$. We say that u_l is t times removable if $\mathbf{C}^w(u)$ has a precise representation on the basis of the B-Spline functions defined on $\bar{U}$, where $\mathbf{Q}_i^w$ are the new control points.

We know that at u_l $\mathbf{C}^w(u)$ is at least $C^{(p-k)}$ parametrically continuous. However, if the control points are properly placed, the curve can well be C^h continuous, where $h \geq p-k+t$. Therefore, if we can verify that the first $p-k+t$ derivatives of $\mathbf{C}^w(u)$ are continuous at u_l, then u_l is effectively t times removable. Note that it is necessary to verify this in the projective space, since $\mathbf{C}(u)$ can be continuous even if $\mathbf{C}^w(u)$ is not. Verifying this amounts to applying Eq. (2.19) t times in reverse and ensuring that no impossible equality arises. That is, we verify that $\mathbf{C}^w(u)$ can be obtained from a NURBS curve defined on the knot vector $\bar{U}$ by inserting u_l t times. Solving this set of equations yields the new control points $\mathbf{Q}_i^w$.

The removal of a u (v) knot of a NURBS surface is carried out by repeating the procedure outlined above for each of the m columns (n rows) of the control point matrix. If the removal procedure fails on one of the columns (rows) then the knot is not removable for the surface. Details of knot removal algorithms for curves and surfaces can be found in [108].

Reparameterization

Given a NURBS curve $\mathbf{C}(u)$ defined on $u \in [a,b]$ and a function $f(t)$ one can express the curve in terms of the parameter t by posing $u = f(t)$ and obtain the new parametric curve $\bar{\mathbf{C}}(t)$. It is clear that the two curves are parametrically different but geometrically identical. If $f(t)$ is a piecewise polynomial or rational function and fulfills the conditions

- $f'(t) > 0$ for all $t \in [c,d]$ (i.e. $f(t)$ is strictly increasing) and
- $a = f(c)$ and $b = f(d)$,

or

- $f'(t) < 0$ for all $t \in [c,d]$ (i.e. $f(t)$ is strictly decreasing) and
- $a = f(d)$ and $b = f(c)$,

then the new curve will be a piecewise polynomial or rational curve that can be expressed as a NURBS curve (if $f(t)$ is decreasing then the curve is reversed). If the degree of $\mathbf{C}(u)$ is p and that of $f(t)$ is p' then $\bar{\mathbf{C}}(t)$ is of degree $p'p$. We should point out that, if $f(t)$ is rational, then the curve will be geometrically changed in projective space, nevertheless its projection to Euclidean space remains unmodified.

Typical applications of reparameterization are internal point mapping (i.e. forcing a curve to assume particular points at particular parameter values), modification of end derivatives and modification of end weights. The general procedure to carry out reparameterization is quite involved and the interested reader is referred to [80, sec. 6.4]. One particularly interesting case is when $f(t)$ is a linear function of the form $\alpha t + \beta$. In this case the control points of $\bar{\mathbf{C}}(t)$ and its degree are the same as those of $\mathbf{C}(u)$ (i.e. they remain unchanged). The new knot vector $T = \{t_i\}$ is obtained by applying the inverse of $f(t)$ to the original knots u_i. The only effect of this reparameterization is to change the parameter bounds and multiply all the derivatives of the curve by α. Thus, any NURBS curve can be reparameterized so that its knot vector occupies the $[0,1]$ interval. Reparameterization can be applied to any of the two parameters of NURBS surfaces in much the same manner.

Weight normalization

This is not a NURBS algorithm per se but an interesting remark on Eq. (2.8): if all the weights of a NURBS surface are multiplied by some non-zero constant the surface remains unchanged in Euclidean space, either parametrically or geometrically. Note, however, that the surface is changed in projective space. This property can be used to normalize the weights of a surface to the $(0, 1]$ interval, provided that the original weights are all positive. The same remark applies to NURBS curves.

Tessellation

In many situations, such as display, it is necessary to evaluate points on a NURBS surface. This can be carried out by repetitive use of knot insertion in order to obtain, for a surface of p and q degrees in u and v, u and v knots of multiplicities p and q, respectively. At the intersection of these knots the control point will lie on the surface. A more geometric interpretation of this procedure is the equivalent Cox-de Boor algorithm [80]. Although perfectly valid, this procedure is not adequate for display on modern graphics hardware, where it is more beneficial to exploit the highly optimized triangle primitives. In particular, knot insertion requires considerable amounts of memory for a large number of points and uses floating-point instructions which cannot, in general, be offloaded to a graphics co-processor.

To exploit triangle primitives of modern hardware, NURBS surfaces can be converted to triangular meshes with a given tolerance through a procedure known as *tessellation*. Many NURBS tessellation algorithms can be found in the literature [62], where the tolerance is given either as the maximum allowable distance from a triangle to the "true" NURBS surface or as the maximum allowable triangle side length. Depending on the algorithms these tolerances can be expressed either in the object's 3D space or in the projected 2D screen space.

2.4.5 Trimmed NURBS

One of the limitations of NURBS surfaces is that their topology is rectangular, which is a direct consequence of using a tensor product formulation. This is apparent when modeling some shapes that require several coincident, or *degenerate*, control points. As an example Figure 2.8 shows a sphere octant described by a NURBS surface. The control net has nine control points, where the three ones at the top are coincident. As it can be seen by the iso-curves the parameter space is severely distorted near that region. Such degenerate control points are allowed by the NURBS surface formulation, but their manipulation and evaluation is not without problems.

Even more so, this topological limitation imposes a considerable hurdle on solid modeling systems, where it is necessary to model the result of surface intersections and Boolean operations (see Section 2.7.1). The resulting shapes are often difficult to express using the rectangular topology. As a solution Casale [10] proposes trimmed surfaces, which allow to cut away portions of a parametric surface.

A trimmed NURBS surface consists of two things: a tensor product NURBS surface as defined in Section 2.4.2 and a properly ordered set of trimming curves that define which portions of the parameter rectangle are to be excluded from the domain of definition. The trimming curves could be of any form, but clearly NURBS curves are most desirable. Given a surface $\mathbf{S}(u, v)$ let the N trimming curves be defined as

$$\mathbf{C}_k(h) = (u_k(h), v_k(h)) = \sum_{i=0}^{n_k-1} \frac{N^k_{i,l_k}(h) w^k_i}{\sum_{j=0}^{n_k-1} N^k_{j,l_k}(h) w^k_j} \mathbf{P}^k_i \qquad k = 0, 1, \ldots, N-1 \tag{2.20}$$

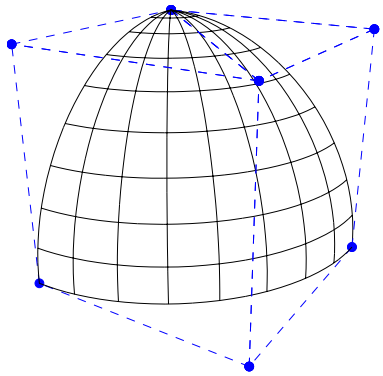
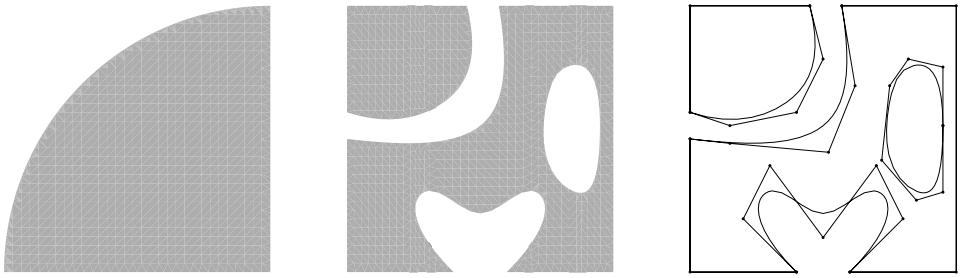


Figure 2.8: The iso-curves and control net of a sphere octant described as a biquadratic NURBS surface.

where $\mathbf{P}_i^k$ are control points in the (u, v) domain, w_j^k their associated weight, and the N_{i,l_k}^k are degree l_k B-Spline functions defined over the knot vector

$$H_k = \{h_0^k, \ldots, h_{m_k-1}^k\}.$$

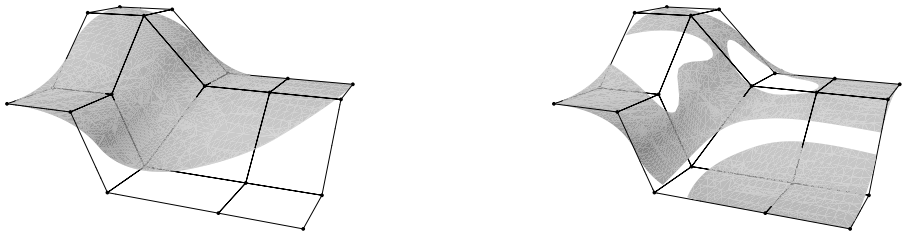
The curves $\mathbf{C}_k(h)$ are all properly oriented forming $M < N$ number of loops. When marching along a curve in a loop as indicated by its direction, the valid portion of the domain of definition is always on the same side. Figure 2.9 shows two examples of trimming domains, with one and three loops. Figure 2.9(b) shows that the loops can be nested to define holes. Figure 2.10 shows the result of applying the previous trimming domains to a simple NURBS surface.



(a) simple domain (1 loop) **(b)** complex domain (3 loops) **(c)** curves of complex domain

Figure 2.9: Trimming domains defined by NURBS curves.

The fundamental algorithms of Section 2.4.4 also apply to trimmed NURBS, with the exception of reparameterization and tessellation. If the u parameter of a NURBS surface is reparameterized with the function $u = f(t)$, the u coordinate of the trimming curves also needs to be reparameterized. For the kth trimming curve the u coordinate is given by $u_k(h)$ and needs to be reparameterized as $t_k(h) = f^{-1}(u_k(h))$, where f^{-1} is the inverse of the function f. Clearly this will lead to a NURBS curve only if $f(t)$ is a linear polynomial or rational function. In the special case of a linear reparameterization with $f(t) = \alpha t + \beta$ the u coordinates of the $\mathbf{P}_i^k$ control points of the trimming curves need to be transformed by the $t = (u - \beta)/\alpha$ relation. Of course, the same remarks



(a) trimming domain of Figure 2.9(a) **(b)** trimming domain of Figure 2.9(b)

Figure 2.10: Trimmed NURBS surfaces resulting of applying the trimming domains of Figure 2.9 to the surface of figure Figure 2.4.

apply to the v parameter of a NURBS surface. Tessellation of trimmed NURBS surfaces requires modified algorithms that take into account the trimming regions and are thus more complex. Many such algorithms have been developed with varying degrees of complexity and accuracy, of which [69, 81, 82] are good examples. [85] considers the issue of jointly tessellating multiple trimmed NURBS surfaces.

2.5 Other surface modeling techniques

Besides polygonal meshes and tensor product surfaces many other surface modeling techniques have been developed, some of which we briefly review here.

2.5.1 Other parametric surfaces

Among parametric surfaces we have only addressed rectangular tensor product ones. Although very flexible, they suffer from two main drawbacks: cumbersome design of complex shapes and rectangular topology. The design issue can be addressed by techniques such as Hierarchical B-Splines [24]. The rectangular topology is, however, inherent in tensor product formulations and other domain types are required to overcome them. Triangular patches have no topological limitation and are thus better suited. Among them, Bézier ones have been thoroughly studied. As pointed out by Farin [21], they were originally developed by de Casteljau in 1959 as the first generalization of Bézier curves to surfaces, although those results remained largely unknown for more than fifteen years. Since the basis functions of Bézier curves, the Bernstein polynomials, are defined in terms of one-dimensional barycentric coordinates they are easily extended to two dimensions in a triangular setting. For degree n the can be expressed as

$$B^n_{i,j,k}(\mathbf{u}) = \binom{n}{i\,j\,k} u^i v^j w^k = \frac{n!}{i!j!k!} u^i v^j w^k, \qquad i+j+k=n, \tag{2.21}$$

where $\mathbf{u} = (u, v, w)$ is the vector of barycentric coordinates of the parametric triangle domain*. The formulation of an nth degree Bézier triangle is then trivial and yields

$$\mathbf{S}(\mathbf{u}) = \sum_{i+j+k=n} \mathbf{P}_{i,j,k} B^n_{i,j,k}(\mathbf{u}), \tag{2.22}$$

where the $\mathbf{P}_{i,j,k}$ control points are arranged in a regular triangular grid. All the geometric properties of tensor product Bézier surfaces, or their equivalent NURBS formulation, remain valid for Bézier triangles. More details on Bézier triangles and their application to geometric modeling can be found in [21]. Although Bézier triangles have seen an important development since the early days of surface modeling, NURBS surfaces remain the preferred method in existing modeling systems, despite their topological limitations.

Unlike multivariate Bernstein polynomials, the development of multivariate B-Spline functions is rather involved. Dahmen, Micchelli and Seidel [14] proposed triangular B-Splines that have been extended by Qin and Terzopoulos [84] to triangular NURBS, which enjoy many of the properties of their tensor product counterparts. Despite the theoretical development of triangular NURBS, their application has not seen major developments and remains a curiosity, due to the complicated domain partitioning that can be required and their time-consuming evaluation.

As a final remark, let us note that triangular meshes can be thought of as linear parametric surfaces, were each triangle is parameterized in terms of its (local) barycentric coordinates. However, no inter-patch constraints are enforced and the surface is only G^0 continuous.

2.5.2 Subdivision surfaces

Another approach to generating smooth surfaces is *subdivision surfaces*. They are based on the following basic procedure: start with a 2-manifold polygonal mesh and iteratively apply a refinement, or subdivision, procedure. In general, the faces of the polygonal mesh need not be planar. If the subdivision procedure is well chosen a smooth surface is obtained in the limit. The first subdivision surface schemes were proposed by Catmull and Clark [11] and Doo and Sabin [17]. The rules of subdivision are similar in both schemes. In the regions of the mesh which are topologically rectangular, they can be considered as the control net of a B-Spline patch and the refined mesh is obtained by subdividing the patch into four sub-patches. In the other regions of the mesh the rule is adapted in an ad-hoc manner. These algorithms will therefore generate a surface which is a B-Spline surface everywhere, except at a limited number of *extraordinary* points. The scheme of Catmull and Clark generates bicubic B-Splines while that of Doo and Sabin generates biquadratic B-Splines. However, in both cases the limit surface is only G^1 continuous at the extraordinary points, even though one could expect G^2 continuity for the former.

Peters [79] proposes a slightly different method (also explained in [21, sec. 19.4]), referred to as *surface splines*. The original arbitrary mesh is refined once to obtain a mesh of quadrilaterals. Following this a corner cutting procedure is applied and triangular and/or rectangular Bézier patches are fitted on the resulting mesh. Each vertex of the original mesh has two *blend ratios* associated to it that can be used to vary the curvature of the resulting surface in its neighborhood. If triangular or a mix of triangular and rectangular patches are employed the resulting surface is G^1 everywhere. If only rectangular patches are employed the resulting surface is G^1 except at vertices with an even number of incident edges exceeding four. Depending on some choices of the corner cutting procedure the resulting surface can approximate or interpolate the original mesh.

*Although three variables appear the polynomials are bivariate only, since the barycentric coordinates always sum up to one, thus $u + v + w = 1$.

Loop [66] proposes a similar but simpler algorithm that requires only one refinement step and no corner cutting. As usual, the refinement step is used to isolate the irregularities of the mesh. Then a quad-net is generated for each vertex of the resulting mesh, from which four quartic Bézier triangles are derived that ensure tangent plane continuity at their edges. In regular regions of the mesh, each set of four triangle patches reduces to a biquadratic rectangular patch. The resulting surface is therefore a parameterizable surface that has G^1 continuity.

More recently, Zorin, Schröder and Sweldens [121] proposed a modified Butterfly subdivision scheme which is smooth on irregular meshes, unlike the original Butterfly scheme [18]. As the Catmull-Clark and Doo-Sabin schemes it is only based on subdivision, but unlike them it is interpolative by construction.

The main advantage of subdivision surfaces is the capability to model arbitrary topologies, something that is not possible to do with rectangular parametric patches without resorting to degenerate control nets. However, the price paid is that, in general, it is not possible to obtain an analytic expression of the surface (e.g., in Catmull-Clark, Doo-Sabin and modified Butterfly schemes). Not being able to derive an analytic expression complicates many design and analysis operations and it can make difficult the realization of some precise shapes such as quadrics. Even in the cases where it is possible to derive an analytic expression, as in the Loop and Peters schemes above, the precise relation between the shape of the initial mesh and the resulting shape is not trivial, also complicating some design operations.

2.5.3 Implicit surfaces

The principle of implicit surfaces were briefly explained in Section 2.3. Here we will explain in some more detail the particular types of implicit surfaces. An implicit surface is generally defined as

$$\mathbf{S}_f = \{\mathbf{P} \in \mathbb{R}^3 | f(\mathbf{P}) = 0\}, \tag{2.23}$$

where $f : \mathbb{R}^3 \to \mathbb{R}$ is the function defining the surface. The following types of implicit surfaces can be distinguished.

Algebraic surfaces satisfy Eq. (2.23) where f is a polynomial. They have been used in graphics and modeling, although the shapes they can model are restricted and there is no direct relationship between the polynomial coefficients and the resulting shape.

Quadrics are algebraic surfaces where the polynomial is of degree two. They represent the familiar ellipsoid, sphere and cylinder as well as more exotic shapes such as hyperboloids, elliptic and hyperbolic paraboloids, etc. While they can suffer from some of the shortcomings of algebraic surfaces, the type of shape and its characteristics (e.g., sphere radius) can be rather easily derived from the polynomial coefficients.

Superquadrics are a generalization of quadrics allowing the representation of a wider surface family [120]. Their equation is similar to that of quadrics but involve non-integer exponents. For example a superellipsoid, centered on the origin and whose axes are aligned to the coordinate axes, is given by

$$f(\mathbf{P}) = f(x, y, z) = \left[(x/a_1)^{2/\epsilon_2} + (y/a_2)^{2/\epsilon_2}\right]^{\epsilon_2/\epsilon_1} + (z/a_3)^{2/\epsilon_1} - 1.$$

where $\epsilon_{1,2} > 0$. The $\epsilon_{1,2}$ parameters determine the "deviation" from a quadric, if they are unity a quadric is obtained.

Hyperquadrics are an extension [61] of superquadrics where

$$f(\mathbf{P}) = f(x, y, z) = \left(\sum_{j=1}^{n} |a_j x + b_j y + c_j z + d_j|^{\epsilon_j} \right) - 1,$$

and where $\epsilon_j > 0$ and $n \geq 3$. The surface is enclosed in the convex envelope defined by the set of planes $a_j x + b_j y + c_j z + d_j = \pm 1$. As the ϵ_j coefficients increase the surface merges with its envelope. An advantage of hyperquadrics is that additional terms can be added that modify the surface "semi-locally", which is desirable for surface matching algorithms. The display of the surface is, however, far from straightforward, in particular for large n.

With the exception of quadrics and superquadrics, implicit forms are rarely used for modeling. Both these forms accept, however, parametric forms, and quadrics can be represented by NURBS. The algebraic surfaces and hyperquadrics are, however, much more suitable for surface matching algorithms [75]. As it was previously noted, the display of implicit surfaces is not trivial.

2.6 Model properties

In addition to geometric shape, 3D models have many other position dependent characteristics, which are collectively referred to as *model properties*. In some cases, as for example in polygonal meshes, the position of the vertices is also considered a property. Besides this, the most common model property is the normal vector. Normal vectors define the direction and orientation of the tangential plane in each position of the surface and are essential for the lighting calculations when rendering a shaded model, as well as in many other operations. In implicit models defined as in Eq. (2.23) the normal vectors are obtained as $\mathbf{N} = [\partial f/\partial x, \partial f/\partial y, \partial f/\partial z]$, which is easily evaluated. For parametric surfaces the derivation is also rather straightforward: given a surface $\mathbf{S}(u, v)$ its normal vector is

$$\mathbf{N}(u, v) = \frac{\partial \mathbf{S}(u, v)}{\partial u} \times \frac{\partial \mathbf{S}(u, v)}{\partial v}.$$

It can happen that both partial derivatives are zero for some (u, v), even if the surface has a normal defined at that point. In that case other means need to be used to compute the normal. For NURBS the expression $\mathbf{N}(u, v)$ can be of high degree and rather expensive to evaluate.

For polygonal meshes, the normal vectors are typically explicitly given. If not, they can be estimated, but having accurate normal vectors is essential for a good quality rendering when the mesh is coarse. Normals can be bound to different elements [7], as follows:

face normals: one normal is associated with each face, or polygon, of the mesh;

vertex normals: one normal is associated with each vertex of the mesh;

corner normals: one normal is associated with each corner of each face.

The most compact representation is obtained with face normals. However, they need to be interpolated between adjacent faces to obtain a G^1 rendering of the model. Vertex normals is the most widely used type. They are still fairly compact and allow a G^1 rendering of the surface. Within a face the normal is typically interpolated from the normals at its vertices. Vertex normals, however, cannot model sharp edges (as between the sides of a cube). This can be achieved with corner normals, which are the most flexible but least compact normal representation for polygonal meshes. Most systems support face and vertex normals, as is the case in VRML [47]. Corner normals are, however, not widely supported.

Other common and important model properties are texture coordinates and color. Texture coordinates are 2D coordinates that define how an image or texture is to be mapped, as a decal, onto the surface [see 23]. Texture coordinates can be bound to vertices or corners. Color is typically represented as an RGB triplet and can be bound to faces, vertices or corners.

2.7 Solid modeling

The techniques presented in the previous sections allow to describe the shape of surfaces in 3D space. These surfaces, however, are not necessarily closed and therefore do not necessarily bound a volume. In many applications it is necessary to describe closed volumes, or solids, as well as being able to distinguish inside from outside. Doing so allows to carry out, among others, physical based simulations. The term *solid modeling* collectively refers to the techniques that consider closed volumes. In the following we will review some of the basic concepts of solid modeling systems. A more detailed introduction can be found in [23].

2.7.1 Regularized Boolean set operations

No matter how the solid objects are represented it is desirable to be able to combine pairs of objects to create more complex ones. One of the most intuitive and popular methods for doing this is through Boolean set operators: union ($\cup$), intersection ($\cap$) and difference ($-$). On 3D solids they are conceptually realized by the straightforward application of those operators to the (infinite) sets of space points in the volume of the solids. Applying these Boolean operators to solids does not, however, always yield solids. As a trivial example consider the intersection of two adjacent cubes: it is a two-dimensional square, which is not a solid. To avoid such cases regularized Boolean set operators are used. If we distinguish between interior and boundary points of a solid (boundary points are not necessarily part of the object), we can define the *closure* of an object as the union of its interior and boundary point sets. A regularized operator op^* between objects A and B is then defined as

$$A\ op^*\ B = \text{closure}(\text{interior}(A\ op\ B)),$$

where op can be any of $\cup$, $\cap$ or $-$. With this definition the degenerate artifacts that can occur with plain Boolean set operators are avoided and the result is always a valid solid.

2.7.2 Boundary representation

Boundary representations, or *B-reps*, describe an object in terms of its boundary surfaces: vertices, edges and faces. Regularized Boolean operators can be applied on the B-reps to construct more complex objects. In the simplest case the faces are restricted to planar polygons and the representation is thus a polygonal mesh. This restriction allows for simpler implementations, but the resulting B-reps are, in general, only capable of approximating the solids being modeled. Some systems allow B-rep meshes with curved faces, for which tensor product NURBS surfaces are widely used. Trimmed NURBS surfaces are generally used to handle the result of the Boolean operations. However, the intersection of NURBS surfaces can lead to very high order implicit trimming curves [74] (e.g., degree 32 for biquadratic patches and 162 for bicubic). An overview of possible strategies to handle these intersections in a reasonable way is given in [74]. Among them we can mention the use of piecewise parametric polynomial approximations to the trimming curves.

Many systems restrict the possible B-reps to 2-manifolds. However, in some cases Boolean operators on 2-manifolds can generate objects which are not 2-manifolds and special steps are

required to avoid such situations. Arbitrary 2-manifolds, either with or without boundary, obey Euler's formula [73]. Given a mesh with V vertices, E edges and F faces, the *Euler characteristic* χ is defined as

$$\chi = V - E + F. \tag{2.24}$$

Euler's formula states that χ is a constant that depends only on the topology of the 2-manifold surface. For an orientable (see Section 2.2) 2-manifold B-rep of C connected components, with H boundary loops (i.e. holes on the surface) and G holes passing through the object, it is given by

$$\chi = 2(C - G) - H. \tag{2.25}$$

If an object has a single component then G is known as its *genus**; for multiple components G is the sum of the genera of its components. Any orientable 2-manifold of genus G, without boundary, can be continuously deformed (i.e. is homeomorphic) into a sphere with G handles. As an example, a sphere has genus zero while a torus has genus one. Hence, they have Euler characteristic 2 and 0, respectively. In the non-orientable case Euler's formula becomes

$$\chi = 2C - G - H. \tag{2.26}$$

One of the main results involving Euler's formula states that two 2-manifolds are homeomorphic if and only if they have the same number of boundary loops, they are both orientable or non-orientable, and they have the same Euler characteristic [73, Theorem 11.1]. An orientable 2-manifold of genus zero with one component and no holes, and thus homeomorphic to a sphere, is often referred to as a *simple mesh*†.

2.7.3 Constructive solid geometry

In *constructive solid modeling* (CSG) simple primitives are combined by means of regularized Boolean set operators. The sequence of operators is stored in a tree, its leaves being the simple primitives. Unlike B-reps, the result of an operation is not computed as another, more complex, object. Algorithms that need to evaluate the final object (i.e. the tree's root) can do so by walking the tree in depth-first order without explicitly combining the primitives. Some CSG systems do, however, store a B-rep approximation to make some operations, such as on-screen display, more efficient.

The type of primitives supported depend on the system. Some of them use simple solids such as parallelepipeds and quadrics. The advantage is that the primitives being solids, any combination will always yield a solid. Other systems use half-spaces as primitives. For example a cube can be defined as the intersection of six planar half-spaces. Half-spaces not being solids, their combinations do not always yield half-spaces and special steps must be taken to ensure the validity of the CSG model. Despite this, half-spaces are a popular choice for CSG primitives due to the simplicity of handling operations such as slicing.

2.8 Conclusions

In this chapter we have reviewed various popular schemes for the representation of 3D objects. Of the possible techniques, polygonal meshes, parametric surfaces and subdivision surfaces appear

*More formally, the genus of a surface can be defined as the maximum number of non-intersecting closed loops along which the surface can be cut without disconnecting it [115].

†Note, however, that in some texts the definition of a simple mesh allows for one boundary loop, in which case it is homeomorphic to a disk.

as the only comprehensive solutions to describe free-form objects. The other techniques can only describe a limited set of shapes or are not adequate for modeling purposes.

While simple and flexible, polygonal meshes are not capable of accurately representing smooth surfaces. If at a given scale a mesh can approximate sufficiently well a smooth curved surface, it will not be sufficient at a finer scale where the non-smoothness will become apparent. Of parametric and subdivision surfaces, only the former are capable of exactly modeling analytic shapes, such as quadrics, in addition to free-form ones. We have also seen that despite their topological limitations tensor product and trimmed NURBS surfaces are a comprehensive solution which have become a popular choice in modeling tools.

In the later sections we have also described the representation of the model properties and the main issues of solid modeling in connection with the previously described surface modeling techniques. Trimmed parametric surfaces can represent the resulting shape of a chain of solid modeling operations, although not without some non-trivial, yet solvable, problems.

3 3D model coding

3.1 Introduction

This chapter is dedicated to a review of previous work in 3D model coding, including a review of fundamental results and techniques in entropy coding and rate-distortion theory. We pay particular attention to non-progressive, or single-rate, mesh coders but also include a discussion of progressive ones. In addition we discuss the compression of non-manifold meshes as well as parametric surfaces.

This chapter is organized as follows. In Section 3.2 we review the fundamental notion of entropy in information theory and describe two practical realizations of entropy coding: Huffman and arithmetic coding. In Section 3.3 we introduce the concept of quantization and the main results of rate-distortion theory. We also introduce a meaningful distortion for 3D models and relate previous results to it. The long Section 3.4 is dedicated to polygonal mesh coding, reviewing from early methods, such as Geometry Compression of Deering, to state-of-the-art methods such as Angle Analyzer of Lee, Alliez and Desbrun. Included in the discussion is also a brief review of progressive coding methods and techniques to handle non-manifold meshes. Section 3.5 is dedicated to the compression of parametric surfaces. Finally, conclusions are drawn in Section 3.6.

3.2 Entropy coding

Consider an alphabet of symbols $\mathcal{A}_X = \{\alpha_1, \alpha_2, \ldots\}$, usually finite in size, where we denote the number of symbols as $\|\mathcal{A}_X\|$. A message composed of a sequence of symbols from this alphabet can be trivially transmitted using $B = \lceil \log_2 \|\mathcal{A}_X\| \rceil$ bits per symbol. However, depending on the statistical properties of the alphabet, and the correlation between subsequent symbols, one can usually code the message in less than B bits per symbol, and often considerably less. The process of coding in such a way is referred to as *entropy coding*, which we explain in the following sections. More details on entropy coding and related concepts can be found in many books, such as [53, 77, 107].

3.2.1 Entropy

If we disregard the interdependencies between the different symbols of a message, we can consider each of them as the occurrence of an independent discrete random variable X with alphabet $\mathcal{A}_X$. Each symbol x from the alphabet occurs with probability $f_X(x)$ (i.e., f_X is the probability mass function). The amount of information carried by the occurrence of a symbol x is clearly related to $f_X(x)$ and we can define the quantity

$$I_X(x) = -\log_2(f_X(x)),$$

referred to as *information content*. Clearly $0 \leq I_X(x) < \infty$. The occurrence of a likely symbol does not transmit much information and has an information content which is close to zero. Conversely, the occurrence of a highly unlikely symbol carries a lot of information and its information content will be high. From this, we can define the *entropy* $H(X)$ as the expected value of the information content, that is

$$H(X) = E[I_X] = \sum_{x \in \mathcal{A}_X} -f_X(x) \log_2 f_X(x).$$

The entropy is always non-negative and bounded above by $H(X) \leq \log_2 \|\mathcal{A}_X\|$. The upper bound is only reached for a uniform distribution, where each symbol has the same probability $f_X(x) = 1/\|\mathcal{A}_X\|$.

The entropy, as defined above, of a stationary random process $\{X_n\}$ is usually infinite, since $\{X_n\}$ has infinite extent and thus the amount of information conveyed is infinite. A more useful definition is the *entropy rate*, which can be thought of as the average entropy per symbol. It is defined with the aid of the m-th order entropy

$$H^{(m)}(\{X_n\}) = \frac{1}{m} H(\mathbf{X}_{0:m}) = \frac{1}{m}(H(X_0) + H(X_1|X_0) + \cdots + H(X_{m-1}|X_{m-2}, \ldots, X_0)),$$

where $\mathbf{X}_{0:m}$ denotes the vector of random variables $(X_0, \ldots, X_{m-1})$. The first order entropy is then the same as the entropy of any random variable of the process, and the m-th order entropy is bounded by it. The entropy rate is defined as

$$H(\{X_n\}) = \lim_{m \to \infty} H^{(m)}(\{X_n\}).$$

Since the m-th order entropy is a monotonically decreasing function of m and bounded below by zero, the entropy rate always converges. Note that for stationary memoryless sources the first order entropy, and hence the entropy of a random variable of the process, equals the entropy rate. In general one refers to the entropy rate simply as entropy, the rate concept being implied from the context.

The relevance of entropy to coding stems from Shannon's noiseless source coding theorem [107, sec. 2.1.3]. This theorem states that the average number of bits per symbol required to code a source $\{X_n\}$ without error is always larger than the entropy rate $H(\{X_n\})$, and that this bound can be approached arbitrarily close as the complexity of the coding scheme is allowed to grow without limit. Conversely, if the code expends an average of less than $H(\{X_n\})$ bits per symbol, the probability of an error approaches 1 as the size of the message grows (i.e., the message cannot be losslessly encoded with an average rate of less than $H(\{X_n\})$ bits per symbol).

Clearly, a fixed-length code with $B = \lceil \log_2 \|\mathcal{A}_X\| \rceil$ bits per symbol cannot approach the entropy rate, except for the trivial case of uniformly distributed memoryless sources. To approach the limit requires the use of variable length coding, where shorter codes are assigned to likely symbols and longer codes to rarely occurring ones. The following two sections introduce two of the most popular variable length coding algorithms: Huffman and arithmetic coding.

Before closing this section, let us note that for continuous random variables the entropy is infinite, since each particular value occurs with zero probability. A more useful quantity is the *differential entropy* that is defined as

$$h(X) = -E[\log_2 f_X(X)] = -\int f_X(x) \log_2 f_X(x) dx,$$

where f_X is the probability density function (PDF) of the continuous random variable X. Unlike entropy for discrete random variables, the differential entropy can be negative. It can be thought of as a relative measure of the average information content, where $h(X)$ is zero when X is uniformly distributed in the interval $[0, 1]$.

3.2.2 Huffman coding

Consider the code where each symbol $x \in \mathcal{A}_X$ is assigned a separate codeword c_x, which is a string of $\|c_x\|$ bits. The code must be uniquely decodable, in that the concatenation of codewords must not create any ambiguity on the coded message. A sufficient and necessary condition for unique decodability [107, sec. 2.2] is that the codeword lengths satisfy

$$\sum_{x \in \mathcal{A}_X} 2^{-\|c_x\|} \leq 1. \tag{3.1}$$

The goal is to find a code (in general it is not unique) that minimizes the average code rate $R = \sum_{x \in \mathcal{A}_X} f_X(x)\|c_x\|$. As demonstrated in [107, sec. 2.2], for any distribution of X there always exists a uniquely decodable code with a rate that satisfies

$$H(X) \leq R < H(X) + 1. \tag{3.2}$$

Huffman developed an algorithm to find an optimum code satisfying Eq. (3.1), in the sense that it minimizes the rate. Huffman's algorithm for code construction for an alphabet $\|\mathcal{A}_X\|$ can be outlined as shown below. Once the code is constructed the encoding procedure becomes a simple

Algorithm 3.1: Huffman code construction

$K \leftarrow \|\mathcal{A}_X\|$
Order the elements of $\mathcal{A}_X = \{\alpha_0, \ldots, \alpha_{K-1}\}$, such that $f_X(\alpha_{i-1}) \leq f_X(\alpha_i)$
if $K = 2$ **then**
  $c_{\alpha_0} \leftarrow$ "0" and $c_{\alpha_1} \leftarrow$ "1"
else
  Create the new alphabet $\mathcal{A}_{X'} = \{\alpha'_1, \alpha_2, \ldots, \alpha_{K-1}\}$
  $f_{X'}(\alpha'_1) \leftarrow f_X(\alpha_0) + f_X(\alpha_1)$
  $f_{X'}(\alpha_i) \leftarrow f_X(\alpha_i)$ for $i \geq 2$
  Invoke the code construction algorithm on the reduced alphabet $\|\mathcal{A}_{X'}\|$
  $c_0 \leftarrow c_{\alpha'_1}$ with "0" appended
  $c_1 \leftarrow c_{\alpha'_1}$ with "1" appended
end if

table lookup operation. The decoding procedure is more involved but can also be optimized as a table lookup operation.

While Huffman coding can achieve a rate close to the entropy in many cases, in general it cannot guarantee a rate closer than implied by Eq. (3.2). Its limitation stems from the fact that each symbol is coded with a separate codeword. Since these codewords must be at least 1 bit long each,

a Huffman encoded source has a code rate which is no less than 1 bit per symbol, even if its entropy rate is much smaller. An extreme example is that of binary sources, where Huffman coding cannot achieve any compression. This limitation can be overcome by blocking the source output. Each block of m source symbols is assigned a codeword, instead of one codeword per symbol. This way, the inefficiency is distributed among m symbols and the bounds on the achievable code rate become

$$H(X) \leq R < H(X) + \frac{1}{m}.$$

Therefore, the entropy rate can be approached arbitrarily close with a sufficiently large m. However, the number of codewords that have to be maintained in memory grows exponentially with m, as $\|\mathcal{A}_X\|^m$, and limits the attainable code rate in practical implementations.

So far we have only considered memoryless sources. To exploit the redundancy between successive source outputs two approaches are possible. One is to apply blocking as above, with the same practical limitations. The attainable code rate is then bounded by the m-th order entropy, instead of the entropy rate, as

$$H^{(m)}(\{X_n\}) \leq R < H^{(m)}(\{X_n\}) + \frac{1}{m}.$$

Another option is to construct a different Huffman code for each of the conditional probability distributions $f_{X_m|\mathbf{X}_{1:m}}(\cdot, \mathbf{x}_{1:m})$. That is, a different Huffman code is selected to code x_i based on the previously coded $x_{i-m+1}, \ldots, x_{i-1}$ source outputs. This approach also requires $\|\mathcal{A}_X\|^m$ codewords ($\|\mathcal{A}_X\|^{m-1}$ separate codes of $\|\mathcal{A}_X\|$ codewords each) to be stored and thus the order m of the model is also limited in practice.

In the above we have assumed that the probability distribution of the source is known to the coder and decoder. If it is not known, it can be estimated at the coder by counting the occurrences of each symbol before applying Huffman's algorithm. The resulting code is then transmitted as overhead information prior to the coded message. This requires, however, to scan the entirety of the source output, or a large part of it, before any information can be coded. Another option, known as adaptive Huffman coding, is to estimate the probability distribution as the source outputs are coded and to modify the Huffman code accordingly at regular intervals. This estimation and code modification is performed at both the coder and decoder. While attractive at first, adaptive Huffman solutions are usually avoided since they significantly increase the complexity of the coder and decoder by requiring the periodical invocation of the Huffman code construction procedure, a significant task in itself. Furthermore, since the probability distribution estimates can only be based on previously coded source outputs, the coding efficiency is often inferior than that of static Huffman coding.

3.2.3 Arithmetic coding

As we have seen above, codes which assign separate codewords to each symbol, such as Huffman, are not efficient for sources with entropies below 1 bit per symbol. This limitation is lifted by another class of coders collectively known as *arithmetic coders*. They are based on recursive interval subdivision, which was devised by P. Elias shortly after Shannon's original publication on information theory [see 107, sec. 2.1.4]. For a source $\{X_n\}$ with distribution f_X the algorithm associates the message $\mathbf{x}_{0:n} = (x_0, \ldots, x_{n-1})$ with an interval $[c_n, c_n + a_n) \subseteq [0, 1)$. It can be stated as shown below, where F_X is the cumulative distribution, given by

$$F_X(\alpha_i) = \sum_{j=0}^{i-1} f_X(\alpha_j). \tag{3.3}$$

Algorithm 3.2: Elias coding

$c_0 \leftarrow 0$, $a_0 \leftarrow 1$
for $n = 0, 1, \ldots$ **do**
  $c_{n+1} \leftarrow c_n + a_n F_X(x_n)$
  $a_{n+1} \leftarrow a_n f_X(x_n)$
end for

The algorithm starts with the unit interval. At each iteration it partitions the current interval into K disjoint sub-intervals whose length is proportional to the probabilities of each symbol (i.e., the i-th sub-interval has length $a_n f_X(\alpha_i)$), where K is the size of the alphabet $\mathcal{A}_X$. The sub-interval corresponding to symbol x_n is selected as the new interval and it goes to the next iteration. Since successive intervals are nested, that is $[c_{n+1}, c_{n+1} + a_{n+1}) \subseteq [c_n, c_n + a_n)$, any number in the interval $[c_n, c_n + a_n)$ represents the message $\mathbf{x}_{0:n-1}$. The coded message is then formed by the fractional bits of any such number expressed in binary. An example of this algorithm is shown in Figure 3.1 for a binary source. Note that, since subsequent intervals are nested, the coded message can be decoded one symbol at a time in a manner very similar to the encoding procedure above.

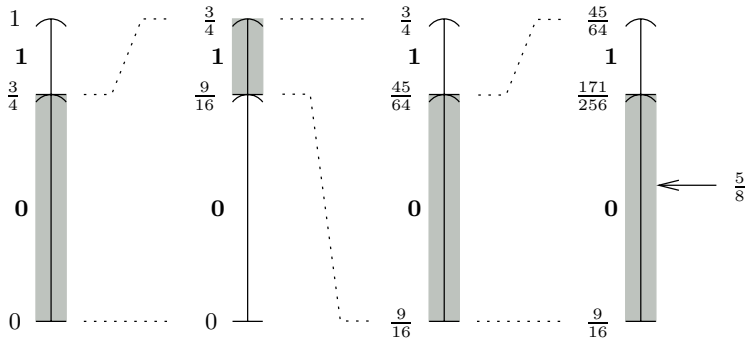


Figure 3.1: Elias coding example: the string "0100" from a binary source with $p(0) = \frac{3}{4}$ and $p(1) = \frac{1}{4}$ can be coded as the fraction $\frac{5}{8}$ (i.e., 0.101 in binary).

The average number of bits required to code m source outputs is $mH(X)$, if we disregard the overhead required to signal the length of the resulting codeword. Most often, a very large number m of source outputs will be coded together making this overhead negligible. Unlike Huffman coding, the complexity of the coding and decoding procedures are not affected by m, making such uses feasible. Therefore, Elias coding can produce a code with a rate that is arbitrarily close to the source entropy $H(X)$ [see 107, sec. 2.1.4].

If we compare with Huffman coding, Elias coding can produce extremely efficient codes whatever the entropy of the source and there is no need to resort to costly blocking schemes. Elias' algorithm, however, requires infinite precision arithmetic and thus is not a practical one. In fact, Elias' algorithm did not see any practical realization for many years. Nevertheless, finite precision realizations are possible with the introduction of some approximations, at the cost of a very small loss in coding efficiency. These realizations are collectively known as *arithmetic coders*. The main change is that the interval size a_n must be rescaled whenever it becomes too small to avoid any risk of underflow. Whenever a rescaling occurs, the base c_n of the interval must also be rescaled accordingly. A practical realization of arithmetic coding for K-ary alphabets is given by Witten, Neal and Cleary [118].

So far we have only considered memoryless sources. However, with Elias' algorithm it is trivial to exploit the redundancy between successive source outputs. For an order p model, it suffices to consider the conditional distribution $f_{X|\mathbf{X}_{n-p:n}}(x_n, \mathbf{x}_{n-p:n})$ instead of the *marginal* distribution $f_X(x_n)$ in Algorithm 3.2 and Eq. (3.3). The code rate will then approach the conditional entropy $H(X|\mathbf{X}_{0:p})$, which is never larger, and often smaller, than the order p entropy $H^{(p)}(\{X_n\})$. Since the symbols can be decoded one at a time, the decoder can select the appropriate conditional probability at each iteration of the decoding procedure and thus the scheme remains realizable. This kind of scheme is usually referred to as *conditional* or *context dependent* arithmetic coding, where a *context* is the vector $\mathbf{x}_{n-p:n}$ of past coded symbols.

While very effective, arithmetic coding is more complex than Huffman coding. Whereas the latter involves only a table lookup operation, the former requires several additions and multiplications for each coded symbol. One of the most expensive operations is the multiplication required to update the interval size a_n and base c_n. In the case of binary arithmetic coders, where the symbols are 0 and 1, the multiplications can be approximated by a sum, if the size of the interval a_n is always kept close to one. The update formulas become as shown below, where f'_X is a properly scaled version of the distribution f_X.

Algorithm 3.3: Approximated interval updating for binary sources

if $x_n = 0$ **then**
  $a_{n+1} \leftarrow f'_X(0) \approx a_n f'_X(0)$
  $c_{n+1} \leftarrow c_n$
else $\{x_n = 1\}$
  $a_{n+1} \leftarrow a_n - f'_X(0) \approx a_n(1 - f'_X(0)) = a_n f'_X(1)$
  $c_{n+1} \leftarrow c_n + f'_X(0)$
end if

This approximation is the basis of the Q-coder [78] and its very successful derivatives, the QM-coder [77, 96] and MQ-coder [107]. The QM-coder is used in the JPEG [77, 112], JBIG [46] and MPEG-4 [48, 49] visual coding standards, while the MQ-coder is used in the more recent JPEG 2000 [107] and JBIG-2 [51] standards. This family of coders always maintains the interval size a_n in the $[0.75, 1.5)$ interval. Although this approximation might seem crude, the loss in coding efficiency is usually in the order of 0.5% to 1%, and does not exceed 3% in the worst case of a uniform distribution, thanks to some additional optimizations. Another arithmetic coder based on this approximation is the Z-coder [8], while the ELS coder [117] uses an alternative approximation based on logarithms. Even though they are still more computationally demanding than Huffman coding, these arithmetic coders require only a few basic operations per coded symbol.

While the use of a binary coder might seem restrictive, let us point that any K-ary alphabet can be coded with a binary coder, by using conditional coding on the $\lceil \log_2 K \rceil$ bits of the binary form of each symbol [see 107, sec. 2.3.2]. The number of binary distributions f_X is the same as would be necessary for the K-ary coder and the coding efficiency is not affected.

Although up until now we have assumed that the distribution f_X is known, an adaptive scheme is easily integrated into the Elias coding algorithm and its practical incarnations. At each iteration, the current estimate $\tilde{f}_{X,n}$ can be used instead of the unknown distribution f_X. The estimate can be obtained by counting the occurrences of past coded symbols or by an implicit relationship between the frequency of renormalization of the interval size a_n and the binary probability distribution. All the practical coders cited above are adaptive ones. The Witten et al. and ELS coders use explicit symbol counting, while the Q, QM, MQ and Z coders use an implicit estimation that is computationally cheaper.

3.3 Quantization and distortion

In the above we have seen how outputs of a source with finite alphabet can be losslessly compressed. However, in many situations the source outputs are from an infinite and uncountable alphabet, such as $\mathbb{R}$, and entropy coding cannot be directly applied (the source entropy is infinite). Such is often the case in geometric models, where vertex coordinates and other model properties are given as 32 bit or 64 bit floating point data that, for all practical purposes, can be assimilated to real valued variables in an interval of $\mathbb{R}$. Even if the source has a finite alphabet, the compressed rate that can be achieved by entropy coding alone is usually larger than what is desired. In such cases, it is necessary to reduce the precision of the source data to obtain a lower compressed rate, at the expense of a distortion between the original and coded data. The coding process becomes, thus, lossy.

Quantization is the process by which the precision of the source is reduced, be it from an infinite to a finite alphabet, or from a large finite alphabet to a smaller one. For a source random process $\{X_n\}$, discrete or continuous, quantization assigns an index $q = Q(\mathbf{x})$, $q \in \mathbb{Z}$, to the data vector $\mathbf{x}$. In general, many different vectors $\mathbf{x}$ are assigned the same index q. The inverse process, *dequantization*, assigns a data vector $\hat{\mathbf{x}} = \overline{Q^{-1}}(q)$ to the codeword q, which is desired to be close to the original data vector $\mathbf{x}$ in order to minimize distortion. The end-to-end process

$$\hat{\mathbf{x}} = \overline{Q^{-1}}(Q(\mathbf{x}))$$

is often also referred to as quantization and $\hat{\mathbf{x}}$ as the quantized data. Note that while $\overline{Q^{-1}}$ is a one-to-one function, Q is a many-to-one function and $\overline{Q^{-1}}$ is not the inverse of Q, which does not exist. Quantization is thus an irreversible process. The many-to-one relationship of Q reduces the entropy of the source, at the expense of some distortion, and thus allows for higher compression.

In the above we have loosely referred to distortion. *Distortion* is the measure of dissimilarity, or error, between the original data vector $\mathbf{x}$ and its quantized form $\hat{\mathbf{x}}$. Depending on the application, different measures of distortion are desirable. A popular choice in coding applications is the mean squared error (MSE)

$$\mathcal{E}_{\text{MSE}} = E[(X - \hat{X})^2],$$

where $\hat{X}$ is the quantized version of the random variable X. In some other applications, the maximum error

$$\mathcal{E}_{\max} = \sup|X - \hat{X}|$$

is a more appropriate measure. When dealing with MSE measures the distortion D is often expressed as the signal-to-noise ratio (SNR) in dB, defined as

$$SNR_D = 10 \log_{10} \frac{\sigma^2}{D}$$

for a signal of variance σ^2.

Quantization is the basis of lossy compression. In general, the coarser the quantization the larger the distortion between the original and coded data but the lower the compressed rate. This trade-off between distortion and compressed rate is studied by *rate-distortion* theory. In the following sections we will review some of the basics of quantization and rate-distortion theory. A more detailed treatment of the matter can be found in [53, 77, 107].

3.3.1 Rate distortion

The trade-off between rate and distortion can be characterized by the *rate-distortion function*, which gives the minimum rate $R(D)$ that is required to code a source with distortion D. The rate-distortion function is convex, continuous and monotonically decreasing on the interval $(0, \bar{D})$, where

$\bar{D}$ is the value above which $R(D) = 0$. Therefore, the inverse function does exist and is called the *distortion-rate function*, $D(R)$.

For an independent and identically distributed (IID) process X with variance σ^2 and an MSE distortion measure, the rate-distortion function is lower bounded by the *Shannon lower bound* given by

$$R_L(D) = h(X) - \frac{1}{2}\log_2 2\pi e D \quad \text{for} \quad D < \sigma^2.$$

Note that for the trivial case of $D \geq \sigma^2$, $R(D) = 0$, since fixing $\hat{X} = E[X]$ yields $E[(X - \hat{X})^2] = \sigma^2 \leq D$. Henceforth, we will assume $D < \sigma^2$. The rate is maximum for a Gaussian process* and is given by

$$R(D) = \frac{1}{2}\log_2 \frac{\sigma^2}{D}.$$

Therefore, the rate-distortion function $R(D)$ of any IID process can be bounded by

$$R_L(D) \leq R(D) \leq \frac{1}{2}\log_2 \frac{\sigma^2}{D},$$

which can also be expressed for the distortion-rate function as

$$D_L(R) = \frac{1}{2\pi e} 2^{2h(X)} 2^{-2R} \leq D(R) \leq \sigma^2 2^{-2R}.$$

The Shannon lower bound for often used distributions can be found in [53, 107]. Furthermore, it is known that, for small distortions D, and hence large rates R, the Shannon lower bound is tight and thus $D(R) \cong D_L(R)$, when R is large.

For correlated processes (i.e. sources with memory) the rate-distortion function is extremely difficult to compute and its treatment is omitted here. The interested reader is referred to [53, appendix D]. However, it is easily seen that the rate-distortion function for such processes will always be smaller than the one of a memoryless process with the same distribution.

3.3.2 Scalar quantization

Scalar quantization is the simplest form of quantization, where each source data sample is quantized independently. The domain $\mathcal{D}$ of the source data samples, usually $\mathbb{R}$, is partitioned into M disjoint intervals $\mathcal{I}_q$ with

$$t_0 < t_1 < \cdots < t_{M-1} < t_M, \quad q = 0, 1, \ldots, M,$$

where

$$t_q = \inf(I_q) = \sup(I_{q-1}).$$

Clearly $t_0 = \inf(\mathcal{D})$ and $t_M = \sup(\mathcal{D})$. The scalar quantizer maps all values in $\mathcal{I}_q$ to a value $\hat{x}_q$ in that interval. The values t_q are thus the decision boundaries, or thresholds, for the $\hat{x}_q$. Given the number of *quantization levels* M and a distortion measure the goal is to choose the t_q and $\hat{x}_q$ values so that the distortion is minimized. The rate R of a scalar quantizer is $\lceil \log_2 M \rceil$ and, if M is not a power of two, can approach $\log_2 M$ arbitrarily close, as m gets large, by jointly coding m quantized values.

For an MSE distortion the necessary conditions for an optimal scalar quantizer are

$$t_q = \frac{\hat{x}_{q-1} + \hat{x}_q}{2} \tag{3.4}$$

*Given all distributions with variance σ^2, the differential entropy h is maximum for the Gaussian distribution.

and

$$\hat{x}_q = E[X|X \in \mathcal{I}_q]. \tag{3.5}$$

Thus, the $\hat{x}_q$ should be the centroids of the quantization intervals and the thresholds t_q the midpoints between the reconstructed values $\hat{x}_q$. These conditions are also sufficient if the source has a log-concave PDF (i.e., $\log f_X$ is a concave function). Uniform, Laplacian and Gaussian distributions all fall under this class.

A quantizer fulfilling the above conditions is thus an optimal scalar quantizer and is called a *Lloyd-Max scalar quantizer*. In general, the solution cannot be expressed in closed form but can be found by iterative numerical algorithms : the Lloyd and Max algorithms [see 107, sec. 3.2.1]. The rate-distortion performance of a Lloyd-Max quantizer can be approximated as

$$D(R) \cong \varepsilon^2 \sigma^2 2^{-2R} \tag{3.6}$$

for large R, where ε^2 is a function of the source PDF (e.g., ε^2 is 1, 4.5 and $\sqrt{3}\pi/2 \cong 2.721$ for uniform, Laplacian and Gaussian PDFs, respectively). For low values of R this approximation is pessimistic and lower MSE distortions are achieved than predicted by Eq. (3.6).

These results can be improved if one allows for entropy coding of the quantizer output. The Lloyd-Max quantizer optimizes the distortion under the only constraint of having M quantization levels and does not take into account the entropy of the quantizer output. If one optimizes the quantizer under the constraint $H(\hat{X}) \leq R$ an offset is introduced in Eq. (3.4) that depends on the probabilities $P(X \in \mathcal{I}_q)$ and therefore the thresholds t_q are no longer the midpoints between the $\hat{x}_q$. Conversely, Eq. (3.5) remains valid and the optimum $\hat{x}_q$ are the centroids of the $\mathcal{I}_q$. Like in the Lloyd-Max quantizer an iterative algorithm can be used to solve the entropy constrained case [see 107, sec. 3.2.3]. Since the quantizer output is entropy coded M can be as large as necessary, and $M = \infty$ is the optimum for unbounded distributions (e.g., Gaussian and Laplacian).

For large $H(\hat{X})$ (i.e. small MSE) the optimum entropy coded scalar quantizer is the *uniform quantizer*, where all intervals are of equal size Δ and the reconstruction values are the midpoints of these intervals. Letting the quantizer indices take values in $\mathbb{Z}$ we have

$$\mathcal{I}_q = \begin{cases} [q\Delta - \frac{\Delta}{2}, q\Delta + \frac{\Delta}{2}) & q > 0 \\ (-\frac{\Delta}{2}, \frac{\Delta}{2}) & q = 0 \\ (q\Delta - \frac{\Delta}{2}, q\Delta + \frac{\Delta}{2}] & q < 0 \end{cases}$$

and

$$\hat{x}_q = q\Delta.$$

Under the assumptions that the entropy coding is effective, hence $R \cong H(\hat{X})$, and that Δ is small (i.e. small MSE) the MSE distortion is $\Delta^2/12$ and the rate-distortion function can be approximated as

$$D(R) \cong \frac{1}{12} 2^{2h(X)} 2^{-2R}.$$

Comparing to the Shannon lower bound in Section 3.3.1, entropy coded uniform scalar quantization differs by a factor of only $\pi e/6$, at high rates. In SNR terms the difference is 1.53 dB. At lower rates, the optimal entropy coded scalar quantizer is not uniform anymore. However, a uniform scalar quantizer with centroid reconstruction values (i.e., as in Eq. (3.5)) is nearly optimal [107, sec. 3.2.4].

Comparing to the results of the Lloyd-Max quantizer one can easily see that an entropy coded uniform scalar quantizer performs better. Hence, at high enough rates there is little interest in using scalar quantizers other than uniform, provided that the use of entropy coding is allowed. Another advantage of the uniform scalar quantizer, which can be relevant in some applications, is that the

maximum error $\mathcal{E}_{\max}$ is bounded by Δ and thus it is easily controlled. For zero mean PDFs, a small improvement in the rate-distortion performance of the uniform scalar quantizer can often be obtained by increasing the width of the interval $\mathcal{I}_0$. Such quantizers are often referred to as *deadzone uniform scalar quantizers*. Although this modification increases the distortion, the gains in $H(\hat{X})$ are high enough to compensate for that loss. This quantizer can be easily implemented as

$$q = Q(x) = \begin{cases} \text{sign}(x)\lfloor \frac{|x|}{\Delta} + \xi \rfloor & \frac{|x|}{\Delta} + \xi > 0 \\ 0 & \text{otherwise} \end{cases},$$

where $\xi < 1$ controls the width of $\mathcal{I}_0$ and

$$\text{sign}(x) = \begin{cases} 1 & x > 0 \\ 0 & x = 0 \\ -1 & x < 0 \end{cases}.$$

For $\xi = 1/2$ the plain uniform scalar quantizer is obtained, where all intervals have width Δ. For $\xi = 0$ the width of $\mathcal{I}_0$ is 2Δ. This is an interesting and widely used case, where the quantizer indices q' for a lower rate quantizer with $\Delta' = 2^p\Delta$, $p \in \mathbb{N}$, are obtained by right bit-shifting the quantizer indices q (i.e., $q' = \lfloor 2^{-p} q \rfloor$).

As we have seen, entropy coded uniform scalar quantization is the best possible scalar quantizer at high rates. However, it falls short by 1.53 dB of the Shannon lower bound. Comparing the definition of the scalar quantizer at the beginning of this section to the general quantizer definition of Section 3.3, one can see that scalar quantization is the special case where the data vectors $\mathbf{x}$ are of length $m = 1$. The 1.53 dB gap can be diminished by allowing m to grow and thus by jointly quantizing m data samples. This is referred to as *vector quantization*. In this case the intervals $\mathcal{I}_q$ become regions in m-dimensional space and are not constrained to be m-dimensional cubes anymore, which is the main limitation of scalar quantizers. As m is allowed to grow without bound the performance of a vector quantizer gets arbitrarily close to the Shannon lower bound. The complexity of vector quantizer encoders is, however, very high as m gets moderately large and they cannot be exploited to their full potential in practical applications. A more detailed introduction to vector quantization can be found in [107, sec. 3.4] while an extensive treatment can be found in [27].

3.3.3 Differential pulse code modulation

The rate-distortion performance of the scalar quantizer established in the previous section assumes that the source $\{X_n\}$ is an IID process. If it is not IID the Shannon lower bound will be lower. However, scalar quantization alone cannot exploit the source correlation and the performance will be governed by the marginal PDF as if $\{X_n\}$ was IID. Several techniques exist to exploit this correlation. One is to use a vector quantizer, as it was briefly explained in the previous section. Another is to use a scalar quantizer followed by conditional entropy coding of the quantization indices. However, the number of conditional distributions that need to be tracked can be prohibitively large for small distortions (i.e., large number of quantization levels) and simplifications of the conditional model are necessary that degrade the achievable performance. Yet another technique is to use decorrelating transforms, such as the discrete cosine transform (DCT) [52, 77, 107] or discrete wavelet transform (DWT) [72, 107]. The resulting data of these transforms is nearly decorrelated, hence nearly IID, and it is possible to proceed with entropy coded scalar quantization. In many cases conditional entropy coding is applied, with a simplified conditional model, to exploit the remaining correlation.

A fourth possibility is to predict a value from the past coded ones and to code the prediction error. Let $\mu_n = \mu(\mathbf{x}_{0:n})$ be this predicted value and $e_n = x_n - \mu_n$ the prediction error. Since $\{X_n\}$ and $\{E_n\}$ are equivalent data sequences carrying the same information, $H(\{X_n\}) = H(\{E_n\})$. However, if the predictor $\mu(\mathbf{x}_{0:n})$ is appropriately chosen the first order entropies will usually obey $H^{(1)}(\{E_n\}) \ll H^{(1)}(\{X_n\})$. A non-conditional entropy coder will thus be more effective if applied on $\{E_n\}$ than $\{X_n\}$. Furthermore, if conditional entropy coding is used, it is simpler and more effective to use a simplified conditional model on $\{E_n\}$ than on $\{X_n\}$.

When quantization is introduced in this predictive scheme it becomes known as *differential pulse code modulation* (DPCM). To avoid the accumulation of successive quantization errors the prediction must be based on the quantized values. DPCM is thus governed by the following equations

$$\begin{aligned}
\mu_n &= \mu(\hat{\mathbf{x}}_{0:n}) \\
e_n &= x_n - \mu_n \\
q_n &= Q(e_n) \\
\hat{e}_n &= \overline{Q^{-1}}(q_n) \\
\hat{x}_n &= \hat{e}_n + \mu_n.
\end{aligned}$$

Where the q_n are the quantization indices that are transmitted, after being entropy coded if desired, and $\hat{e}_n$ and $\hat{x}_n$ the quantized values of the prediction error and source data, respectively. The equivalent block diagram is shown in Figure 3.2.

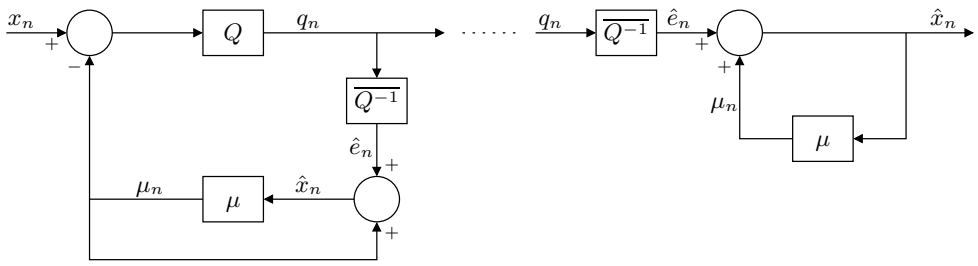


Figure 3.2: Block diagram of a DPCM coder

An important characteristic of DPCM is that the quantization error of the prediction error equals that of the source data, i.e., $\hat{e}_n - e_n = \hat{x}_n - x_n$. Therefore, $\{X_n\}$ and $\{E_n\}$ have the same distortion, which is independent of the predictor μ. The optimal predictor is the one that minimizes the variance of the prediction error and thus is the conditional mean of X given the previous (quantized) values of the samples used in prediction. This optimal predictor cannot be found in practice because the conditional mean depends on the quantized values. By assuming a high rate we can approximate $\hat{x}_n \cong x_n$ and obtain a conditional mean. In practice the predictor is often restricted, for the sake of simplicity, to be a linear combination of the past quantized values. Nevertheless, a linear predictor will only be optimal if $\{X_n\}$ is Gaussian, but can provide a fairly good approximation for other distributions. For Gaussian Markov-p processes, DPCM with a linear predictor and entropy coded uniform scalar quantization can deliver a rate-distortion performance that differs from the Shannon lower bound by 1.53 dB at high rates, as in the IID case. At low rates, however, the performance of the predictor degrades because of the large distortion introduced by the quantization error and the overall rate-distortion of the DPCM coder is severely affected. As such, the applicability of DPCM is limited to high and medium rates.

3.3.4 Distortion in 3D models

The distortion measures used in the development of the previous sections are based on sample to sample distances. When dealing with the geometry of 3D models the samples are the vertex positions in space. However, the relevant distortion is the distance between the surfaces defined by the vertices and not the distance between the vertices themselves. A meaningful distance between two surfaces $\mathcal{S}$ and $\hat{\mathcal{S}}$ can be defined as the Hausdorff distance [13]. First we define the point to surface distance

$$d(\mathbf{P}, \hat{\mathcal{S}}) = \min_{\hat{\mathbf{P}} \in \hat{\mathcal{S}}} \|\hat{\mathbf{P}} - \mathbf{P}\|, \tag{3.7}$$

where $\|\cdot\|$ denotes the usual Euclidean, L-2, norm. The Hausdorff distance is then defined as

$$D_{\max}(\mathcal{S}, \hat{\mathcal{S}}) = \max_{\mathbf{P} \in \mathcal{S}} d(\mathbf{P}, \hat{\mathcal{S}}). \tag{3.8}$$

This distance is usually asymmetric, in that $D_{\max}(\mathcal{S}, \hat{\mathcal{S}}) \neq D_{\max}(\hat{\mathcal{S}}, \mathcal{S})$. An example depicting such a situation is shown in Figure 3.3, where $D_{\max}(\mathcal{S}, \hat{\mathcal{S}}) < D_{\max}(\hat{\mathcal{S}}, \mathcal{S})$. A symmetric distance can be defined as

$$D_{s,\max}(\mathcal{S}, \hat{\mathcal{S}}) = \max\left[D_{\max}(\mathcal{S}, \hat{\mathcal{S}}), D_{\max}(\hat{\mathcal{S}}, \mathcal{S})\right]. \tag{3.9}$$

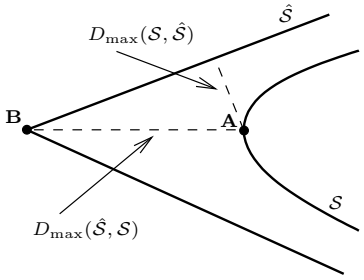


Figure 3.3: Example of non symmetric surface distances: $D_{\max}(\mathcal{S}, \hat{\mathcal{S}}) < D_{\max}(\hat{\mathcal{S}}, \mathcal{S})$.

The one-sided and symmetric Hausdorff distances provide a maximum, worst case, distortion measure between two surfaces. A one-sided MSE-like distortion can be defined as

$$D_{\text{MSE}}(\mathcal{S}, \hat{\mathcal{S}}) = \frac{1}{|\mathcal{S}|} \oint_{\mathbf{P} \in \mathcal{S}} d(\mathbf{P}, \hat{\mathcal{S}})^2 dS, \tag{3.10}$$

where $|\mathcal{S}|$ is the area of $\mathcal{S}$. A symmetric version can be defined analogously to Eq. (3.9).

Consider a polygonal mesh defined by vertices $\mathbf{v}_i$, $i = 0, 1, \ldots, M-1$, their coded counterparts $\hat{\mathbf{v}}_i$ and the surfaces $\mathcal{S}$ and $\hat{\mathcal{S}}$ defined by $\{\mathbf{v}_i\}$ and $\{\hat{\mathbf{v}}_i\}$, respectively. Unfortunately, there is no simple relationship between the sample to sample MSE distortion

$$D_{\text{MSE}}(\{\mathbf{v}_i\}, \{\hat{\mathbf{v}}_i\}) = \frac{1}{M} \sum_{i=0}^{M-1} \|\mathbf{v}_i - \hat{\mathbf{v}}_i\|^2$$

and the surface to surface distances $D_{\text{MSE}}(\mathcal{S}, \hat{\mathcal{S}})$ and $D_{\text{MSE}}(\hat{\mathcal{S}}, \mathcal{S})$. Depending on the shape of the polygons and the valence of the vertices the sample to sample MSE distortion can be smaller or larger than the surface to surface distances. However, one can expect the same order of magnitude for the two measures and therefore the rate-distortion results previously derived provide a useful approximation for polygonal meshes. If one considers the maximum distance, the sample to sample distance provides an upper bound to the surface to surface distances of Eqs. (3.8) and (3.9).

Since the Hausdorff based MSE and maximum distances are defined on the surfaces themselves, and not samples, they remain applicable to polygonal meshes were there is not a one-to-one correspondence between vertices of $\mathcal{S}$ and $\hat{\mathcal{S}}$. Such cases arise often when surface simplification steps are involved in the coding process. Hence, the Hausdorff based distance is a more general distortion measure.

Unfortunately, while Eq. (3.7) can be computed in exact form on polygonal meshes that is usually not the case with Eqs. (3.8) and (3.10). Therefore, it is necessary to resort to point sampling approaches to obtain an approximation of the maximum and MSE distances. Furthermore, if $\hat{\mathcal{S}}$ has a large number of polygons the exhaustive search for the minimum in Eq. (3.7) demands a very large number of computations. Cignoni, Rocchini and Scopigno [13] and Aspert, Santa-Cruz and Ebrahimi [3] propose methods to efficiently compute the Hausdorff based distances for triangular meshes. Since all polygonal meshes can be converted to triangular ones with no geometric distortion, those methods can provide close approximations of $D_{\max}(\mathcal{S}, \hat{\mathcal{S}})$ and $D_{\mathrm{MSE}}(\mathcal{S}, \hat{\mathcal{S}})$ for any polygonal mesh.

3.4 Coding of polygonal meshes

As explained in the previous chapter, polygonal meshes are one of the most popular means to model 3D surfaces. Although very flexible in the kind of shapes that can be modeled, polygonal meshes often require a very large storage space, in particular for high quality models. The compression of polygonal meshes has therefore been a topic of much research in the past decade and continues to be so. The early compression methods, such as that of Deering [15], were mainly focused on speeding up the transfer of model data from the CPU to the graphics board, for rendering purposes, across a bus limited in bandwidth. Such methods have to be of low complexity so as to be easily decodable by the hardware on the graphics board and therefore they only obtain modest compression ratios. Very rapidly though, more effective compression methods tailored for transmission over the Internet or similar networks started being proposed and are still the main focus of research in the field. A fairly recent review can be found in [45, Part IV]. Other, somewhat dated, reviews can be found in [7] and [89].

The coding process of polygonal meshes can usually be divided into two complementary, yet fairly independent, components: connectivity and geometry. Connectivity coding deals with the topology of the mesh, or in other words the adjacency relations between the polygons. On the other hand, geometry coding deals with the position in space, or coordinates, of each vertex and optionally the normals, colors or any other model properties. In general, geometry coding will exploit the connectivity information to increase the compression efficiency. In the following sections we will review the previous art in the coding of polygonal meshes, following a more or less chronological order. Let us note that many of the compression methods that have been proposed, and which are explained below, deal with triangular meshes exclusively. In fact, any polygonal mesh can be converted in a triangular mesh without distortion by triangulating each non-triangular face. Furthermore, each so introduced edge can be specially signaled and the original polygonal mesh can be recovered by simply removing these edges after decoding. Therefore, considering triangular meshes is sufficient, although it is often not optimal.

Polygonal mesh coders can be classified as *progressive* or *non-progressive* (also referred to as *single-rate*). A progressive coder will code a coarse version of the original mesh and will progressively add more detail until the full detail mesh is recovered. This is convenient in situations where the transmission of the whole compressed model takes considerable time: the receiver can quickly display a coarse version of the model and refine it over time as more data is received. Conversely, single-rate

coders code the whole model as a unit. In general, single-rate coders deliver higher compression efficiency than progressive ones since they can better exploit the redundancy in the data. Although at first it might seem that only progressive coders should be of interest, single-rate coders are useful in many situations and also serve to encode the coarse base mesh of progressive ones. In what follows, we discuss single rate coders in detail and then provide a brief overview of progressive ones in Section 3.4.12.

3.4.1 Uncompressed meshes

Prior to the overview of the compression methods, let us briefly cite the uncompressed formats as a reference point. The VRML standard [47] defines a textual format for polygonal meshes, named `IndexedFaceSet`. The (x, y, z) spatial coordinates of all the vertices in the mesh are given in an array, named `coord`. Each face is then defined by the ordered list of zero based indices of its vertices, given in an array named `coordIndex`. Since the number of vertices of each face is variable, a dummy vertex with index -1 is used to indicate the end of each list. Hence, the connectivity and geometry information is embodied in the `coordIndex` and `coord` arrays, respectively. Optionally, normal vectors can be specified as bounded to faces or vertices, but not corners. Figure 3.4 shows an example of a simple polygonal mesh described in VRML. As can be seen, VRML is not a compact format but it is very flexible and the model data is easily modified.

```
IndexedFaceSet {
    coord Coordinate {
        point [
            0 0 0
            0 2.5 0
            3.2 0 0
            4.1 3.7 0
            1.1 2.9 4
        ]
    }
    coordIndex [
        0 1 3 2 -1
        1 0 4 -1
        3 1 4 -1
        2 3 4 -1
        0 2 4 -1
    ]
}
```

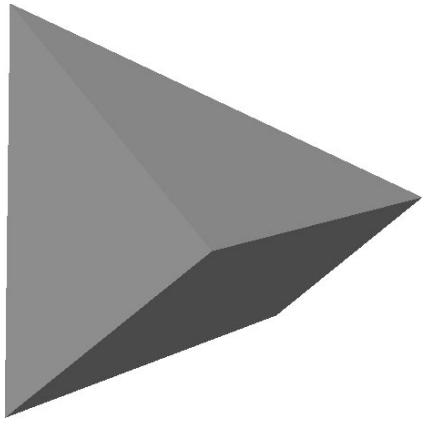


Figure 3.4: VRML description of an irregular pyramid (left) and the rendered model (right).

Analogous canonical binary representations can also be defined. For a model with V vertices the vertex indices can be coded on $\lceil \log_2 V \rceil$ bits each. Additionally $\lceil \log_2(N-2) \rceil$ bits are required per face to code the number of vertices of each face, where N is the maximum number of vertices per face (each face has at least three sides, hence the -2). The total number of bits required to code the connectivity of a model with F faces is therefore

$$F\lceil \log_2(N-2) \rceil + \lceil \log_2 V \rceil \sum_{i=0}^{F-1} n_i,$$

where n_i is the number of vertices of the i-th face. For triangular meshes the first term vanishes

and the second one becomes $3F\lceil \log_2 V \rceil$. Recalling Euler's formula, Eq. (2.24), we have

$$V - E + F = \chi, \tag{3.11}$$

where E is the number of edges and χ the Euler characteristic. Typical meshes have one component and just a few holes and handles (i.e. low genus), hence their Euler characteristic χ is small in magnitude. Letting E_E and E_I be the number of external and internal edges, respectively, we have, for triangular meshes, $E_I = (3F - E_E)/2$ and thus $E = E_E + E_I = 3F/2 + E_E/2$. If we further assume the holes to be comparatively small, which is typically the case, and since we assume few holes, $E_E \ll E_I \approx E$ and hence $E \approx 3F/2$. Substituting into Eq. (3.11) yields $V \approx F/2 + \chi$ and since χ is close to zero, $2V \approx F$. Therefore, in a typical triangle mesh, there are roughly twice more triangles than vertices*. The coding cost becomes then approximatively $6V\lceil \log_2 V \rceil$. Therefore, the space required for connectivity coding in such an uncompressed canonical form grows as $\mathcal{O}(V \log V)$. Such a non-linear behavior is clearly undesirable as it does not scale well to large models.

The vertex positions are usually stored as 32 bit floating point numbers (i.e., IEEE single precision format). The 8 bit exponent of a single precision floating point number covers from the size of the known universe down to the size of sub-atomic particles. The 24 bits of mantissa provide a precision in excess of one in 16 million. It is clear that the encoding of vertex positions in individual geometric objects does not require such a precision, nor such a dynamic range†. Even if some storage formats allow such a precision, it corresponds to noise and has no parallel in any real world object. This precision can be therefore reduced through quantization without loss of relevant data (although there is a loss of information in the strict sense). The usual solution is to apply uniform quantization on each of the vertex coordinates. The axis aligned *bounding cube* that encloses the geometric object is partitioned into small cubes. The quantized position of each vertex is the center of the partition cube it occupies. For a bounding cube of side length L and quantization of each coordinate into n bits, the partition cubes are of side $\Delta = L/2^n$. Hence, the maximum distortion for the position of any vertex is $\sqrt{3/4}L/2^n$. Table 3.1 shows the maximum distortion for varying number of bits and object scales. While 8 bits appears as insufficient for most purposes, 10 to 12 bits provide enough precision for many modeling applications and 16 bits is more than necessary for the vast majority of applications. For example, 16 bits provide a precision comparable to the feature size of a Pentium 4 (0.13 μm) or a cell in a human body ($\sim$10 μm) and are more than enough to precisely locate a car in Manhattan or a door in the Titanic.

		8 bits	10 bits	12 bits	16 bits	20 bits	24 bits
Pentium 4 die	(14 mm)	47.4 μm	11.8 μm	2.96 μm	0.19 μm	11.6 nm	0.72 nm
Human body	(1.8 m)	6.08 mm	1.52 mm	0.38 mm	23.8 μm	1.49 μm	92.9 nm
Titanic	(269 m)	90 cm	22.7 cm	5.7 cm	3.6 mm	0.22 mm	13.9 μm
Manhattan	(21.6 km)	73.1 m	18.2 m	4.57 m	28.5 cm	17.8 mm	1.11 mm

Table 3.1: Maximum distortion for uniform quantization of geometric objects of various scales as a function of the number of quantized bits.

A canonical binary representation of geometry information will therefore often require between 30 and 36 bits per vertex and will rarely exceed 48 bits (i.e. 10, 12 or 16 bits per coordinate, respectively). Comparing with the results for triangular meshes above, the cost of connectivity

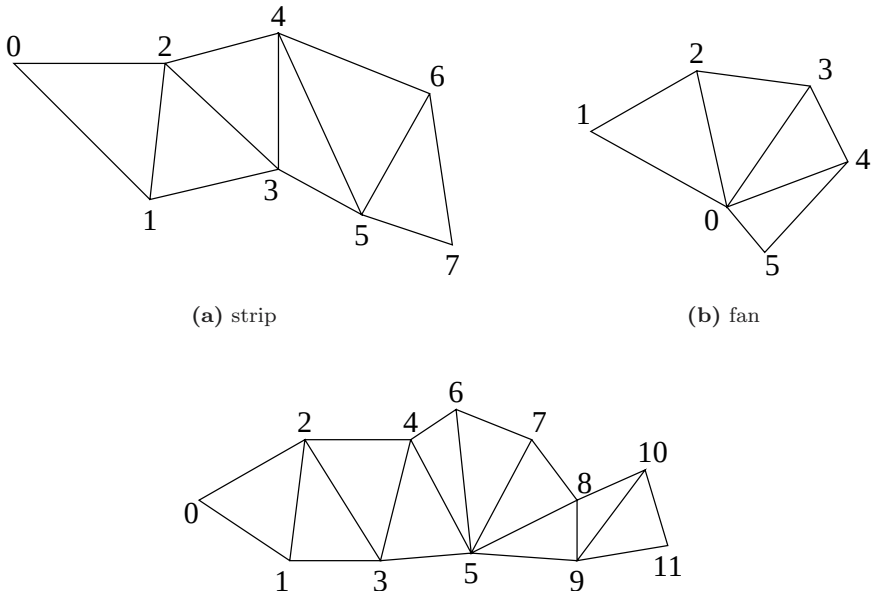
*Following an analogous development, quadrilateral meshes have roughly as many vertices as faces.

†Note however, that in a complex virtual world the scale of different objects can differ by several orders of magnitude. Since this difference in size appears across objects it is not required to have an extremely high precision to represent each of them.

coding grows much rapidly than geometry with model size. For example the connectivity cost is slightly larger than that of geometry for models with 32, 64 and 256 vertices, when coordinates are quantized to 10, 12 and 16 bits, respectively. For a model with 10'000 faces the connectivity cost is more than 20 times larger than that of geometry, if a 12 bit quantization is used. This demonstrates the interest on performing an efficient compression of connectivity.

3.4.2 Geometry Compression

As mentioned above, the first compression methods proposed for polygonal meshes were particularly tailored to the problem of in-memory storage and rendering. Popular 3D toolkits, such as OpenGL [119], make use of triangle strips and fans to reduce the number of times a vertex needs to be transferred to and processed by the graphics hardware. As shown in Figure 3.5, each new vertex is combined with one of the free edges of the previous triangle to form a new one. Depending on which edge the new triangle is attached, two configurations are possible: the *triangle strip* and the *triangle fan*. In the former, the edge on which the new triangle is formed alternates between "left" and "right", while in the latter the new triangle is always attached to the "right" edge. Triangle strips are common on mostly regular meshes, such as those generated by tessellation of parametric models, while triangle fans typically arise at irregular vertices. Both of these forms are supported by OpenGL. A more general format, the *generalized triangle strip*, uses an extra bit per triangle to specify on which of the two free edges the triangle should be formed and allows to merge triangle strips and fans. While these structures reduce the number of times a vertex's coordinates need to be specified, they are limited by the fact that past vertices cannot be referenced. In other words, the connectivity cannot be losslessly encoded. On the other hand, the decoding engine is extremely simple and only two vertices need to be stored at any time.



(a) strip **(b)** fan

(c) generalized strip

Figure 3.5: Vertex ordering in a triangle strip, triangle fan and generalized triangle strip.

In the *Geometry Compression* scheme proposed by Deering [15], this restriction is partially lifted

by the use of a queue to store vertices. A new vertex can be optionally pushed into the queue so that it can be referenced later. The resulting structure is referred to as a *generalized triangle mesh*. The size of the queue required to be able to avoid sending vertices more than once is clearly dependent on the model size and hence unlimited *a-priori*. Bar-Yehuda and Gotsman study this relationship [5] and conclude that a queue of size $12.72\sqrt{n}$ is sufficient for an arbitrary triangular mesh with n vertices and that $1.6\sqrt{n}$ is a lower bound. In Deering's scheme the queue size is limited to 16, to limit the amount of memory required, and thus 4 bit indices are required to reference past vertices. The coordinates are uniformly quantized and coded using a predictive scheme and a modified Huffman code. The predictor is simply the value of the previous vertex. The compression of normals (vertex bounded) is more involved, as the orientation of the unit length normal is coded using spherical coordinates instead of the Cartesian xyz values. This allows to reduce the number of bits used to represent a normal since an almost uniform distribution of quantized orientations is obtained, which would not the case if the xyz values were used. Vertex bounded colors can be specified as RGBα values. They are coded in a way very similar to coordinates.

The final coded bitrate depends on the ability to exploit as much as possible the vertex queue to avoid resending previous vertices. The construction of a generalized triangle mesh that fulfills this requirement is a non-trivial problem. Chow [12] proposed an algorithm that locally constructs the generalized triangle mesh by starting with triangle strips adjacent to the edges and covers the object's mesh in a spiraling pattern, taking care not to overflow the vertex queue. Using this algorithm they obtain an average ratio of 0.67 between the number of explicitly coded vertices and the total number of triangles. As a reference, this ratio is 3 for independent triangles, 1 for infinitely long generalized triangle strips and $\approx$0.5 for a generalized triangle mesh on an infinite regular mesh.

As reported in [15], bitrates in the range of 25 to 30 bits per triangle can be obtained for typical models with quantizations of 30 to 36 bits per vertex (i.e., 10 to 12 bits per coordinate) and 12 to 14 bits per normal. This amounts to a compression factor between 2 and 3 to 1 with respect to the canonical encoding defined previously, quantized to the same number of bits. This modest performance is largely due to the simplicity of the scheme and to the lossy coding of the connectivity information. The compression factor is, however, about 8 or 9 to 1 relative to the primitives used for rendering with triangle strips. Hence, a good reduction in the bandwidth requirements for rendering is obtained. Deering's architecture has been chosen as the compressed data format for the Java3D API [100].

3.4.3 Topological Surgery

If a one-component mesh contains no holes, the coding of the connectivity amounts to the coding of the graph that has the vertices as nodes and the face edges as edges, which is referred to as the *vertex graph*. If the mesh is of genus zero (i.e. has no handles), the vertex graph is a planar graph. An example is shown in Figure 3.6. In an equivalent way, the graph, referred to as the *face graph*, that has faces as nodes and the edges shared by adjacent faces as graph edges can also be used to encode the connectivity. In fact, the face graph is the dual of the vertex graph. In the case of triangular meshes the face graph is referred to as the *triangle graph*. From a theoretical point of view Tutte [110] enumerated all the possible structures that a planar triangular graph can assume and shows that the encoding requires 3.245 bits per vertex. Hence, the entropy of the connectivity of simple triangular meshes is 3.245 bits per vertex, or alternatively 1.623 bits per triangle, if all possible meshes are considered equally probable. Taubin and Rossignac's *Topological Surgery* [105, 106], provides a practical way to efficiently compress the vertex graph for 2-manifold orientable triangular meshes, along with extensions to deal with holes, handles, n-sided faces and non-orientable meshes. Additionally, Topological Surgery also compresses vertex positions and other properties such as

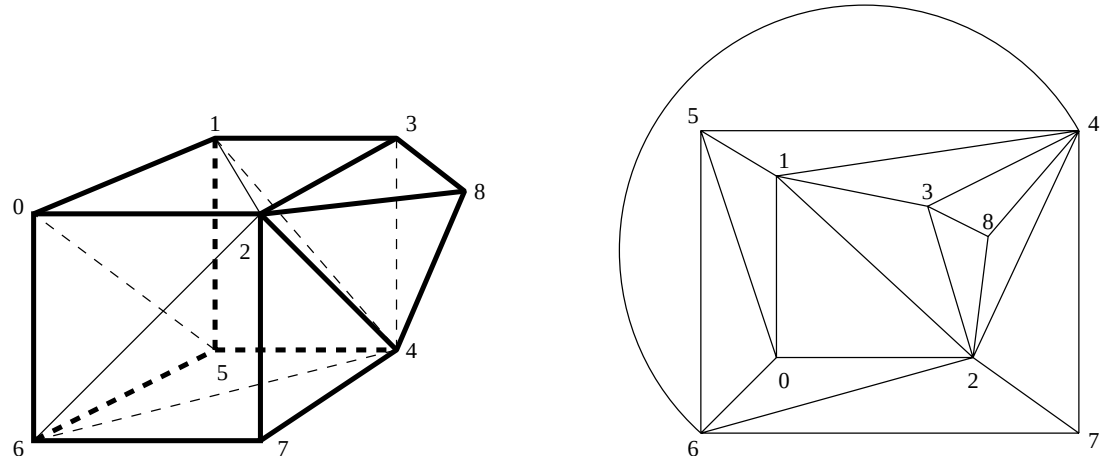


Figure 3.6: The triangle mesh of the union of a cube and an irregular tetrahedron: (a) mesh with labeled vertices (sharp edges are bold, hidden edges are dashed) and (b) the corresponding vertex graph.

normals, colors and texture coordinates. We start by describing the connectivity coding algorithm for meshes homeomorphic to a sphere (i.e., orientable 2-manifolds with no holes and no handles) and then introduce the extensions to deal with more complex meshes.

Topological Surgery encodes the vertex graph as two interlocked trees. A spanning tree of the vertex graph is constructed and the mesh is cut along the edges of this tree, the *cut edges*. The resulting mesh has all the vertices on its boundary and is referred to as a *simply connected polygon* by Taubin and Rossignac. An example is shown in Figure 3.7 for the mesh of Figure 3.6. Since the mesh is 2-manifold and has no holes, each cut edge corresponds to two border edges of the simply connected polygon. The triangle graph of a simply connected polygon is actually a binary tree, the *triangle spanning tree*. Furthermore, the simply connected polygon can be viewed as generalized triangle strips that are joined at *branching* triangles. The triangle spanning tree is thus composed of triangle runs that end at either branching or leaf triangles. Topological Surgery encodes this tree by starting at a leaf, that becomes the root, and recursively encoding each triangle run. A triangle run is encoded as its length plus a bit indicating if it ends in a leaf or branching triangle. The triangle spanning tree is, however, not sufficient to recover the simply connected polygon. In fact, within each triangle run it is necessary to know to which free edge, "left" or "right", the next triangle is to be attached. This is encoded by the *marching pattern* as one bit per triangle, which is then entropy coded.

Obviously, the simply connected polygon is not sufficient to reconstruct the original mesh, since it is necessary to establish which pairs of edges of the former must be stitched together to close the simply connected polygon into the mesh. This information is contained in the triangle spanning tree that was used to cut the mesh. Topological Surgery encodes this tree in the same way as the triangle spanning tree, except that an extra bit per run is used to signal if the new run is the last run starting at the branching node. This extra bit is not required for the triangle spanning tree because it is a binary tree. Thus, the encoded vertex and triangle spanning trees along with the marching pattern, plus an offset to relate the roots of both trees, are enough to recover the connectivity of the original mesh. Table 3.2 shows the encoding of the triangle tree and simply connected polygon

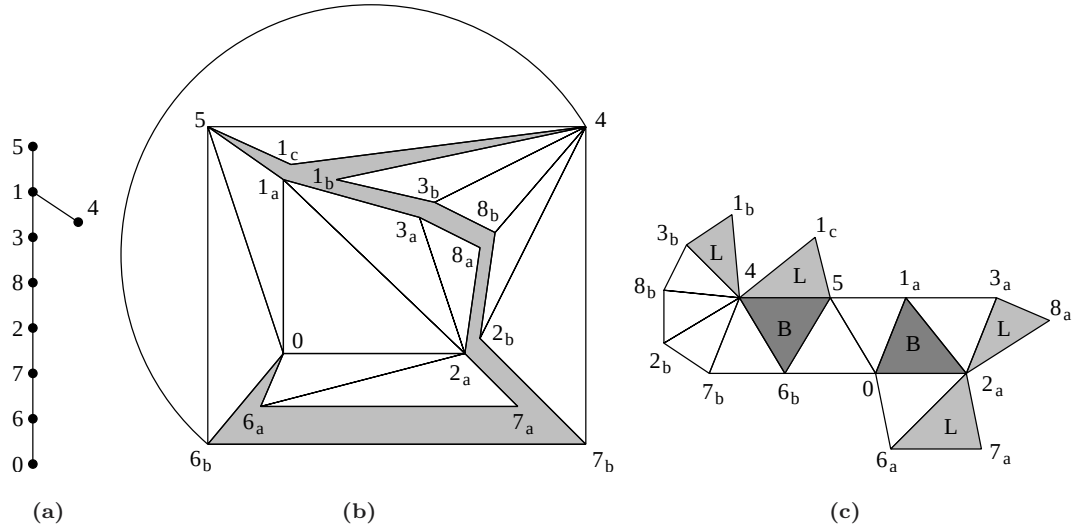


Figure 3.7: Topological Surgery applied to the mesh of Figure 3.6: (a) the vertex tree, (b) the cut mesh (cut area grayed), (c) the cut mesh "flattened" as a simply connected polygon (branching and leaf triangles are labeled as "B" and "L", respectively).

of Figure 3.7, where the triangle tree has been rooted on the leaf triangle that has the vertex 1_b.

	code	run code format
Vertex tree	(1,0,1), (6,1,0), (1,1,1)	(`length`, `endsInLeaf`, `isLastBranch`)
Triangle tree	(5,0), (1,1), (3,0), (2,1), (2,1)	(`length`, `endsInLeaf`)
Marching pattern	LLLL, , LR, R, L	

Table 3.2: Coding of the vertex and triangles trees and marching pattern of Figure 3.7. The triangle tree is rooted on vertex 1_b.

Clearly, the compressed rate will depend on the triangle spanning tree used to cut the mesh. The longer the triangle runs and the fewer branching triangles, the higher the compression ratio. Taubin and Rossignac propose several methods to construct efficient vertex spanning trees. They obtain the best results by a method that resembles the process of peeling an orange, which can be described as follows. Starting at a root vertex, concentric rings are constructed by considering the triangles that are adjacent to the previous ring. Each ring is then cut open and they are joined together in a spiral.

If the original mesh has non-zero genus, then cutting along the vertex tree does not yield a simply connected polygon. In fact, cutting in such a way reduces the Euler characteristic by one, from χ to $\chi - 1$. Due to the non-zero genus $\chi < 2$ and thus the cut mesh has a Euler characteristic less than one and cannot be a simply connected polygon. Because of the handles of the original mesh, the cut mesh presents some cycles and additional cut edges are necessary to break them. Making $2 - \chi$ cuts along *jump edges* is thus necessary to obtain a simply connected polygon. These jump edges are the edges that belong to neither the vertex nor the triangle tree. The start of each jump edge is coded in the triangle tree by treating each regular (i.e., non-branching and non-leaf) triangle incident on a jump edge as being a branching triangle having one run of length zero starting at the jump edge. For leaf triangles, two extra bits encoded in the marching pattern are required to

identify jump edges. The other end of jump edges is encoded in an extra table as the number of edges separating the two vertices in the boundary of the mesh cut by the vertex tree.

If the original mesh has one or more boundaries it is possible that after cutting through the vertex tree disconnected meshes are obtained. Topological Surgery avoids this by ensuring that all but one of the edges of each boundary loop are included in the vertex tree. The remaining edge is treated as a jump edge. If the mesh is non-orientable, the simply connected polygon is always orientable and no special steps are required to encode it. The orientation, and thus the loop traversal order, can, however, change across jump edges and thus one extra bit per jump edge is used to signal such changes.

The scheme above can only handle triangular meshes. In order to handle arbitrary polygonal meshes, the scheme presented in [105] is used which can be described as follows. Each polygonal face is triangulated without adding new vertices. Edges of the original mesh are referred to as *polygonal* edges, while edges introduced by the triangulation are referred to as *non-polygonal*. The vertex tree is constructed on the original mesh, and thus contains only polygonal edges. In this way all the non-polygonal edges will be interior edges of the simply connected polygon. Hence, one extra bit per interior edge is required to recover the polygonal mesh.

Vertex positions are uniformly quantized and then coded in vertex tree traversal order. Predictive coding is used with a linear predictor using the K most recent ancestors in the vertex tree. The coefficients of the linear predictor are chosen as to minimize the MSE and coded as overhead information. The prediction errors are then entropy coded. Properties (e.g., normals, colors, etc.) are coded in an analogous way after quantization. For vertex and face bound properties the vertex tree and triangle tree traversal order is used, respectively. For corner bound properties a corner tree is constructed based on the order in which corners are visited during decompression and used for the encoding order. Quantization of normals is done in a non-rectilinear way [105] somewhat similar to that of Deering's method [15].

In [105] Taubin et al. report compressed connectivity rates ranging from 1 to 8 bits per triangle, for models of more than 100 triangles. Typically, the compression efficiency is larger for larger models, since longer runs occur. Conversely, in the cases where "good" vertex trees cannot be found, the compression efficiency does suffer due to the short triangle runs and high number of branching triangles. When the geometry is included, compressed rates in the range of 6 to 11 bits per triangle can be obtained with quantizations of 10 to 12 bits per coordinate. Comparing with Deering's method, Topological Surgery achieves compressed rates that are significantly lower even if the fact that normals are not included for the latter is taken into account.

Bossen proposes an alternative way to encode the connectivity information [7]. The main difference is that instead of coding the length of each run in the triangle and vertex spanning trees, a binary variable is used to signal if a tree edge is the end of a run. This and all other binary variables are then compressed with binary arithmetic coding. Alternative ways to code holes and handles are also proposed. This improved format is very similar to what is included in the MPEG-4 standard [49] for the coding of polygonal meshes. Topological Surgery was originally submitted as a binary format for VRML [105].

3.4.4 Triangle Mesh Compression

Touma and Gotsman propose an alternative method to code the connectivity and geometry of 2-manifold triangular meshes, called *Triangle Mesh Compression* [45, 109]. The key to their coding scheme is the following observation: in a triangular mesh which is an orientable manifold without boundary the 1-ring of any vertex can be consistently ordered and furthermore the 1-ring forms a closed loop. The coding algorithm is conceptually rather simple and basically consists of specifying

the valence of each vertex in a predetermined order. This kind of coding algorithm has become known as *vertex* or *valence based coding*, of which Triangle Mesh Compression was the first to be proposed.

Prior to describing the algorithm let us define the following terms.

Free vertex: a vertex not yet coded.

Free edge: an edge not yet coded.

Full vertex: a vertex with no free edges.

Active list: an ordered, cyclic, list of coded but not yet full vertices. We denote it as $\mathcal{L} = \{v_0, v_1, \ldots, v_{l-1}\}$, where l is its length. Since the list is cyclic $v_i = v_{i+kl}$, $k \in \mathbb{Z}$.

Cut border: the loop of edges formed by the pair of vertices v_i, v_{i+1}, $0 \leq i \leq l-1$, of the active list (note that v_l, v_0 is included in this list).

Pivot: the vertex on which all coding operations are performed. It is always the head of the active list.

Let us first assume that the triangular mesh is without boundary and has no handles. Vertices are coded with `add` n commands, where n is their valence. The algorithm is initialized by coding an arbitrary vertex, which becomes the pivot. The active list is initialized to this vertex. Each free edge incident on the pivot is then coded, in counter-clockwise order, by coding the vertex at the other end. In this way, all the vertices in the 1-ring of the pivot are coded. Whenever a vertex v_i in the active list becomes full the edge (v_{i-1}, v_{i+1}) is implicitly coded and v_i is removed from the active list. If the full vertex is the pivot, the next one in the active list becomes the new pivot and coding proceeds as previously. Note that, in this process the triangles are implicitly coded. After each addition of a vertex to the end of the active list the (v_0, v_{l-2}, v_{l-1}) triangle is implicitly coded, with counter-clockwise orientation. Likewise, when a vertex v_i becomes full, and prior to its removal from the active list, the triangle (v_{i-1}, v_i, v_{i+1}) is implicitly coded.

The cut border constitutes the boundary between the coded and not yet coded parts of the mesh, as all vertices in its interior are full. When a free edge incident on the pivot is to be coded, it is possible that the vertex at the other end is already present in the active list (i.e., the vertex has been previously coded). Coding the free edge will cause the cut border to become self-intersecting. This is coded with a `split` n command, where n is the offset in the active list between the pivot and the other vertex of the edge being coded. The active list, hence the cut border, is split as $\mathcal{L}_1 = \{v_0, v_1, \ldots, v_{n-1}, v_n\}$ and $\mathcal{L}_2 = \{v_n, v_{n+1}, \ldots, v_l\}$. The list $\mathcal{L}_2$ is pushed into an active list stack and coding proceeds with $\mathcal{L}_1$. When the active list contains only three vertices and the pivot is full, processing of that active list is terminated after removing the pivot. The next active list is popped from the stack and processing continues. When processing of the last active list terminates the whole triangle mesh has been coded. This process is shown in Figure 3.8 for the mesh of Figure 3.6. The steps are as follows. (a) `add 5`, this new vertex becomes the pivot. (a)→(b): `add 4`, `add 7`, the active list forms the first triangle. (b)→(c): `add 4`. (c)→(d): `add 7`. (d)→(e): `add 4`. (e)→(f): the pivot vertex is full, remove it and make next vertex the new pivot. (f)→(g): `add 5`. (g)→(h): second to last vertex of active list is full, remove it. (h)→(i): the pivot vertex is full, remove it and make next vertex the new pivot. (i)→(j): `add 3`. (j)→(k): `split 2`, push second part of active list in stack. (k)→(l): `add 3`. (l)→(m): the pivot vertex is full, remove it and make vertex the new pivot. (m)→(n): active list exhausted (only 3 vertices left and the pivot vertex is full), pop next from stack. (n)→(o): last active list also exhausted. (p): reconstructed mesh.

The algorithm, as described above, only handles triangular meshes with no holes. Triangle Mesh Compression handles holes by adding a dummy vertex for each hole, that is connected to all vertices on the hole's boundary, thus closing the mesh. These dummy vertices are specially coded by adding a negative sign to their valence. Hence, the dummy vertices can be identified and removed at the decoding stage. In order to handle meshes with handles it is necessary to introduce the *merge* operation. In fact, for non-zero genus meshes, it is possible that a free edge to be coded is incident on a vertex in an active list which has been pushed to the stack. In other words, this free edge connects two disjoint cut borders, which must be merged. This situation is coded through a `merge` i n command, where i is the position in the stack of the active list to be merged and n is the position in that list of the vertex at which they should be merged. If the currently active list is $\mathcal{L} = \{v_0, \ldots, v_l\}$ and the one to be merged is $\mathcal{L}' = \{v'_0, \ldots, v'_n, \ldots, v'_{l'}\}$ the merged list is $\{v_0, \ldots, v_l, v'_n, \ldots, v'_{l'}, \ldots, v_0, \ldots, v'_{n'-1}\}$. After merging, $\mathcal{L}'$ is removed from the stack.

The coding of a typical mesh consists primarily of `add` commands along with some `split`s and very few `merge`s. Furthermore, the vertex valence distribution is typically skewed, with a mean of six. All this is amenable to efficient entropy coding. Touma and Gotsman use Huffman and run-length coding. Using this scheme they typically obtain compressed connectivity rates between 1 and 2 bits per vertex, which amounts to 0.5 to 1 bits per triangle. For highly regular meshes, such as those obtained through a tessellation process, they obtain values as low as 0.2 bits per vertex (i.e., 0.1 bits per triangle). When compared to the Topological Surgery method previously presented, Triangle Mesh Compression achieves a substantial improvement. Compared to Tutte's bound (3.245 bits per vertex), Triangle Mesh Compression yields a much lower coded rate for typical meshes. However, this implies that some pathological meshes will have a coded rate that will exceed 3.245 bits per vertex.

As in Topological Surgery, Triangle Mesh Compression quantizes the coordinates to some number of bits and uses predictive coding to reduce the entropy of vertex coordinate data. The predictor is also linear, albeit more sophisticated. Due to the order in which the connectivity coding proceeds, when a new vertex r is coded forming the triangle (u, v, r) with the u and v vertices of the active list, an adjacent triangle (u, w, v) has already been coded. The two triangles, adjacent on the (u, v) edge, are predicted to form a parallelogram. The predicted vertex position is thus $r' = u + v - w$. This predictor is often referred as the *parallelogram predictor*. The u, v, r and w vertices will, however, be seldom coplanar. Touma and Gotsman recognize this fact and propose to predict the crease angle along the (u, v) edge as an average of the crease angles between sets of previously coded neighboring triangles. A schematic of the parallelogram predictor is shown in Figure 3.9. The prediction errors are entropy coded using a mix of Huffman and fixed length coding. Using this technique, they report compressed rates between 8 and 12 bits per vertex, with an 8 bit per coordinate quantization, for a variety of models. Comparing to Topological Surgery, the geometry coding results are 1.5 times better in average. The overall, connectivity plus geometry, results are 1.7 times better, in average, for Triangle Mesh Compression.

3.4.5 Edgebreaker

Although Triangle Mesh Compression can compress the connectivity of mostly regular meshes very efficiently, there is no known worst case analysis that provides an upper bound for the connectivity coding rate. In fact, the coding cost of a split operation is not fixed and depends on the size of the mesh as $\log_2 V$. Rossignac proposes the *Edgebreaker* [89] connectivity coding scheme that has a guaranteed worst case compressed rate of 2 bits per triangle for genus zero meshes with no more than one boundary. As Triangle Mesh Compression, it is a region growing algorithm. However, it is based on faces instead of vertices.

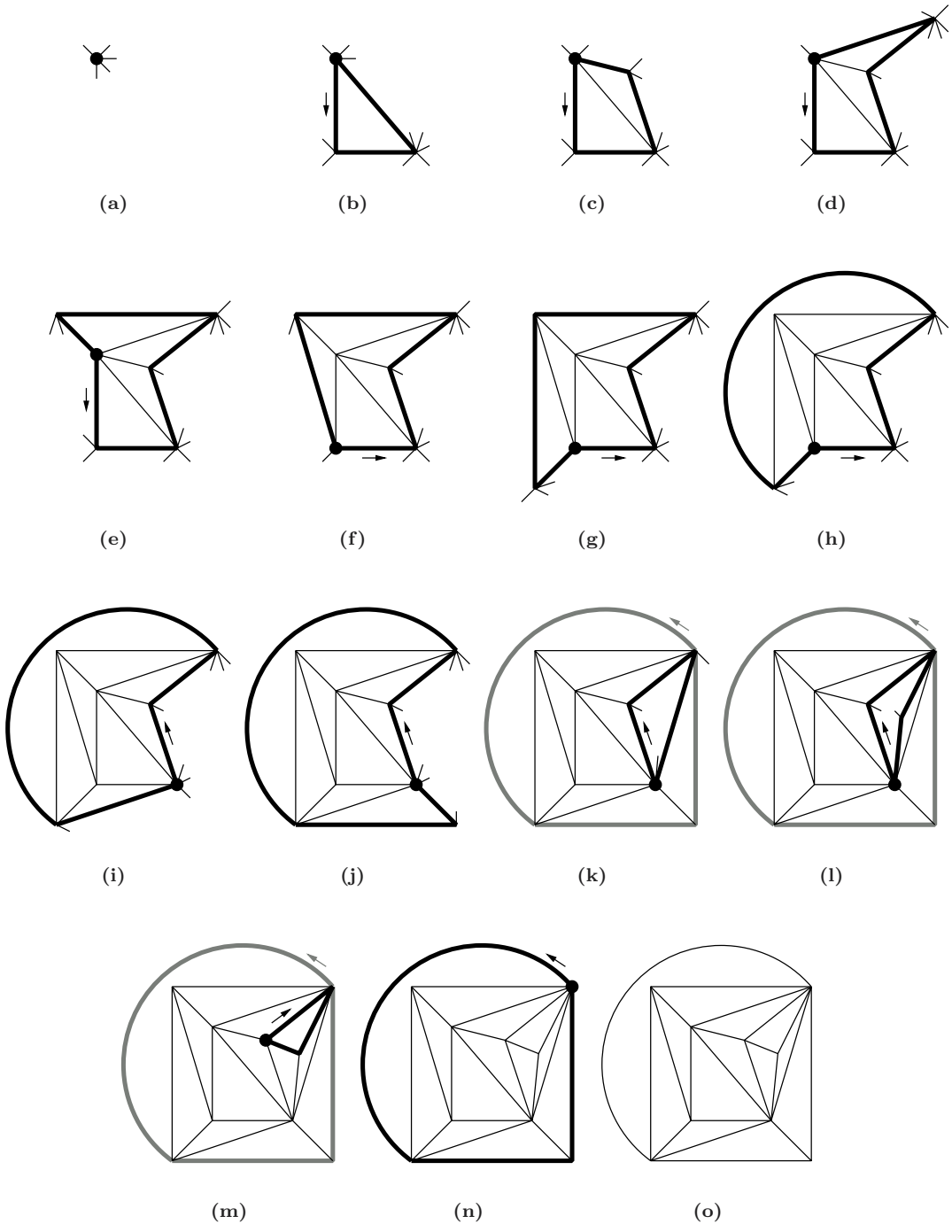


Figure 3.8: Triangle Mesh Compression coding process for the mesh of Figure 3.6, starting at the vertex labeled 1. Thick lines denote the current active list edges, or cut border, and the arrow indicates its orientation. Gray denotes an active list that has been pushed down the stack. The current pivot is shown by the thick dot.

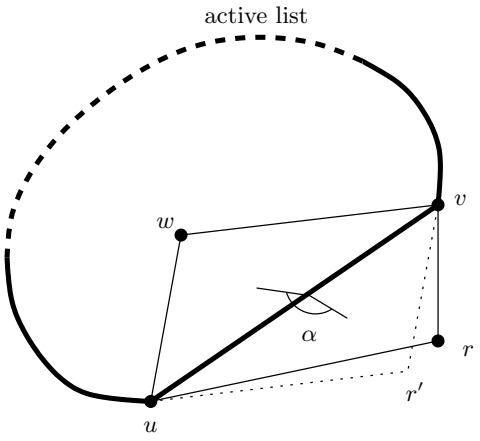


Figure 3.9: Parallelogram prediction as used in Triangle Mesh Compression. The vertex r is predicted as r', so as to form a parallelogram with vertices w, u and v with a crease angle α.

Edgebreaker codes a triangle mesh as a series of operations. Each operation processes one triangle that is removed from the mesh. The set of not yet processed triangles forms one or more connected components, the boundaries of which are called *cut-borders*. The edges of a cut-border form a closed non-intersecting path of the original mesh. Note that cut-borders may share a vertex, but not an edge. Each cut-border has an oriented edge, called the *gate*, on which the next operation will take place. The set of cut-borders is stored in a stack, all operations taking place on the top one. For meshes with boundary, the cut-border is initialized to the boundary loop, of which an arbitrary is designated as the gate. For closed meshes the cut-border is initialized to a single arbitrary edge. At each step Edgebreaker codes the not yet processed triangle that is incident on the gate as a C, L, R, E or S operation and updates the cut-border and gate. These operations are shown in Figure 3.10 and are as follows. Let the gate be the oriented edge (v_1, v_2) of the cut-border and v the vertex of the triangle to be coded that is opposite the gate. If v is not on the cut-border a C is coded. The gate (v_1, v_2) is removed from the cut-border and is replaced by the edges (v_1, v) and (v, v_2), the latter becoming the new gate. If v is on the cut-border several situations can arise. If v precedes v_1 on the cut border an L is coded and the gate is replaced by (v, v_2). If v follows v_2 on the cut-border an R is coded and the gate is replaced by (v_1, v). If v both precedes v_1 and follows v_2 on the cut-border an E is coded. This can only arise if the current cut-border has three edges, in which case it vanishes after the E operation. The empty cut-border is popped from the stack and coding continues with the next one. If v is somewhere else on the cut-border an S is coded and the cut-border split in two, in much the same way as in Triangle Mesh Compression. The second half is pushed down the stack and coding continues with the first one. Note that each operation codes the triangle (v_1, v_2, v), with counter-clockwise orientation, which becomes a processed triangle. Note that, as in the other region growing algorithms, no merge operations can occur for meshes of genus zero homeomorphic to a sphere or disk.

One important point about Edgebreaker is that no index needs to be coded with the S split operation, unlike in Triangle Mesh Compression. This is a direct consequence of encoding the E operation. In fact, an E operation is always required to terminate the first loop created by an S operation. In between an S operation and its matching E other pairs of S and E operations can appear that further split the cut-borders. In this regard, S and E operations can be thought of as opening and closing parentheses, respectively, in the stream of coded operations. Finding the matching E of an S operation is thus straightforward. The split offset is the number of edges removed

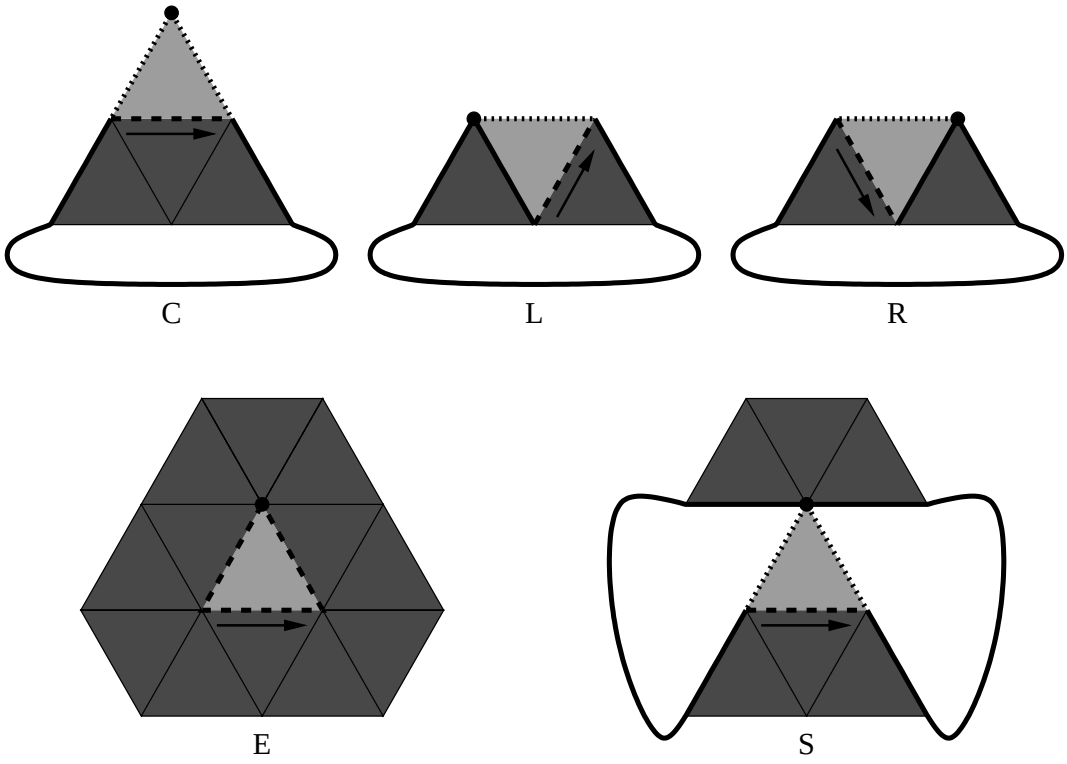


Figure 3.10: Edgebreaker operations. Dark gray denotes processed triangles, light gray the triangle being coded. Thick lines denote the cut-border. Dashed edges are removed from the cut border and dotted ones are inserted. The arrow shows the current gate. The vertex defining the new triangle is shown as the thick dot.

from the cut-border, counting from the S operation up to, and including, its matching E. This can be computed at the decoder by a simple procedure in a preprocessing step as C and S operations increase the edge count by one, L and R decrease it by one and E decreases it by three.

As usual, the stream of operations is entropy coded. A conceptually simple approach would consist in Huffman coding the symbols C, L, R, E and S. The Huffman tables would be computed for the particular mesh being coded and included as overhead information in the coded representation. Rossignac proposes in [89] a simpler approach consisting of prefix codes independent of the mesh that guarantees a worst case compressed rate. Each operation codes exactly one triangle and there is a one-to-one correspondance between C operations and interior vertices. Therefore, for meshes with comparatively few boundary edges, about half the operations are Cs, as there are roughly twice more triangles than vertices and almost all vertices are interior ones. The prefix code proposed in [89] assigns a 1 bit codeword to C and 3 bit codewords to the other operations L, R, E and S. The total coded bitrate is thus approximatively 2 bits per triangle. In the case of meshes without boundary the bitrate is exactly 2 bits per triangle. A more rigorous demonstration can be found in [89]. The CL and CE sequences are impossible for manifold meshes as they would specify an identical triangle twice. Hence, the above code can be improved by using an alternative prefix code for operations immediately following a C: a 1 bit codeword for C, as previously, and a 2 bit codeword for the other operations R and S. Using this scheme a rate of 1.7 bits per triangle is reported [89]. For meshes with a large number of boundary edges the above code is not optimal as the R operation becomes the most frequent one. A more efficient, but similar code, would use 1 bit for R and 3 bits for the other operations. In either case, the coding cost never exceeds 3 bits per triangle.

The coding procedure, as presented so far, is only capable of handling meshes with no more than one boundary and with no handles. Rossignac also describes how to code additional holes and handles. If the mesh has more than one hole, it can happen that the third vertex v of the triangle to be coded lies on the boundary of a hole, instead of being on the cut-border or an internal vertex. In this case an additional operation, M, should be coded to merge the cut-border with the boundary loop of the hole in question. The length of the boundary loop needs to be encoded as well. Coding this extra symbol would require to extend the prefix codes above by using, for example, 4 bits to code S and M operations, which would decrease the compression efficiency of each S. Instead, Edgebreaker uses the same codeword for M and S and stores an *M-table* as overhead information to distinguish between the two. The M-table includes the offsets of M operations in the coded stream together with the lengths of their respective boundary loops. Handles are coded in a similar way. In fact, in non-zero genus meshes the third vertex v of the triangle being coded can lie on a cut-border which is not at the top of the stack. This happens because when the cut-border wraps around a handle a new hole is created, instead of a disconnected component. This situation is handled in Edgebreaker with a slightly different merge operation, M'. No length needs to be coded along M', as that is already known. However, it is necessary to code the position in the stack of the cut-border to be merged and the offset of the vertex on that cut-border where the merge is to be performed. As is the case with M operations, the same codeword is used for S and M' and an *M'-table* is stored to distinguish them. This table includes the corresponding stack index and vertex offset of each M', as well as the offsets of M' operations in the coded stream. In general, since the number of holes and handles is small compared to the total number of triangles of a mesh, the overall impact of M- and M'-tables on the coding efficiency is very small.

If one compares Edgebreaker to Triangle Mesh Compression, the latter can provide much higher compression for regular or semi-regular meshes than the former. Nevertheless, Edgebreaker has a guaranteed worst case behavior that is linear in the number of triangles. In fact, Edgebreaker provides a connectivity rate between 1.7 and 2 bits per triangle, with an absolute worst case of 2 bits

(3 bits if the codewords are badly chosen). On the other hand, Triangle Mesh Compression provides a compressed rate between 0.5 and 1 bit per triangle in general, but can take up to in excess of 2 bits per triangle for badly behaved meshes, with no guaranteed upper-bound. In addition, Edgebreaker's algorithm allows for a simpler implementation.

In [57] an alternative coding of the CLERS string is proposed that yields a worst case rate of 1.83 bits per triangle for meshes homeomorphic to a sphere (or equivalently 3.67 bits per vertex). The scheme uses three sets of alternative prefix codes, somewhat similar to the original ones above. The authors prove that, for an arbitrary mesh homeomorphic to a sphere, at least one of the three sets of prefix codes has a rate that does not exceed 3.67 bits per vertex. It suffices then to select the appropriate set for the mesh being coded and to include a 2 bit code signaling the choice at the beginning. The authors, however, do not provide insight as to which set is appropriate for which type of mesh and a priori the appropriate set can only be selected after constructing the CLERS string. They use the same methods as the original Edgebreaker technique for encoding holes and handles. For this kind of meshes the coding cost becomes $3.67(V + 2G) + 0.83B$, where B is the number of boundary edges, plus the cost of the M- and M'-tables.

With respect to Tutte's bound of 3.245 bits per vertex, the worst case coded rate of this improved coding scheme is fairly close ($\sim +13\%$). This indicates that the algorithm is fairly close to the best possible one from a worst case perspective. However, this also hints to the fact that the encoding is not capable of compressing typically occurring meshes in a highly efficient manner.

3.4.6 Edgebreaker derivatives

The original Edgebreaker decoding algorithm exhibits an asymptotic non-linear time complexity due to the look-ahead procedure that is used to handle sub-sequences generated by S operations. Rossignac and Szymczak [91] propose a new decoding procedure, named *Wrap&Zip*, that exhibits linear time complexity as well as more efficient binary encodings of the CLERS string. The Wrap&Zip decoding algorithm starts by decoding the CLERS string by assigning a new temporary third vertex to the triangles decoded by L, E, R and S operations, instead of immediately identifying which of the vertices of the cut-border is this third vertex. For C operations this third vertex is unique, as is not on the cut-border, and a final label can be immediately assigned to it. The use of these temporary vertices avoids the look-ahead procedure to deal with the S operations. This is the wrapping step. It yields a simply connected polygon, very much like the triangle spanning tree and marching pattern in Topological Surgery. During wrapping, each boundary edge of the simply connected polygon is assigned an orientation that depends on the operation that generated it. The zipping step consists in stitching the pairs of boundary edges that are incident to a common vertex and that are both oriented away from it. This has the effect of merging a temporary vertex with its final label. Only L and E operations can introduce edges that start zipping. While L operations generate only one zip, E operations start recursive zipping. In [91] it is proved that the number of zip operations equals the number of edges in the vertex spanning graph, $V - 1$. Hence, Wrap&Zip decoding has linear time complexity. As such, this algorithm can only deal with simple meshes. [91] also proposes extensions for meshes with holes and handles. Handles are dealt with in a way analogous to Topological Surgery, with two jump-edges per handle (in [91] they are called *glue-edges*). Triangles with jump edges are coded by S* operations, along with the index of the edge to which their left edge, the jump-edge, should be glued. Holes are dealt with S' operations, which are analogous to the M operations of Edgebreaker. As in Edgebreaker, the S* and S' symbols are coded as S, and S*- and S'-tables are used to distinguish among them. An alternative but similar decompression algorithm, named *Zip&Wrap*, is proposed in [90] along with a very simple implementation of the Edgebreaker compression algorithm. The extensions for handling meshes with handles are also presented in the

subsequent papers [67, 92], along with an alternative explanation of Edgebreaker.

Rossignac and Szymczak [91] also propose alternative binary encodings that are, in general, more efficient than those previously reported, but that no longer guarantee an upper bound of 2 bits per triangle. The first proposed scheme codes CC, CS and CR pairs as single symbols (remember that CL and CE pairs cannot occur), in addition to isolated R, E, S and L operations. The codewords for these symbols are prefix codes of 2 or 4 bits. This scheme achieves, in general, rates between 1.3 and 1.6 bits per triangle. The other proposed scheme breaks the CLERS string into words by introducing spaces after each non-C symbol that is followed by a C. The generated words are then Huffman coded, and the corresponding Huffman table included as part of the coded mesh. This scheme is also briefly mentioned in the original Edgebreaker paper [89]. Using such a technique they obtain coded rates of 0.9 bits per triangle, including the Huffman table.

More recently another decoding algorithm for Edgebreaker has been proposed named *Spirale Reversi* [44]. This algorithm works by decoding the CLERS string in reverse order, and reversing the meaning of each operation. In this way no particular procedure is required to handle the split generated by the S operation, avoiding Edgebreaker's look-ahead and Wrap&Zip's zipping. In this reversed decoding a C removes a vertex from the cut-border, while L and R add new vertices to the cut-border. An E operation introduces a new cut-border with three edges and the previous cut-border is pushed to a stack along with its current gate. Finally, the S operation pops the top cut-border from the stack and merges it with the current one. The two cut-borders are merged at the end and start vertices of their gates, respectively. As decoded vertices are sometimes merged, temporary dummy labels are assigned to the vertices introduced by some operations. For meshes with a single boundary Spirale Reversi does not need to compute the length of the boundary by scanning the CLERS string, unlike Edgebreaker and Wrap&Zip, as the boundary is simply the cut-border after the decoding of the last operation of the reversed CLERS string. The coding of additional holes is done in the same way as in Edgebreaker with an M operation. Reverse decoding of M is performed in reverse by merging two of the vertices of the cut-border and splitting the current cut-border, where one of the two halves becomes the hole's boundary. Handles are coded in the same way as in Edgebreaker with an M' operation, except that the counterclockwise offset from the gate to the common vertex is coded in addition to the counterclockwise offset from the common vertex to the gate. The inverse M' operation splits the current cut-border in two by merging the two vertices given by the offsets and inserts this new cut-border in the stack at the position given by the index coded with M'. Having both offsets allows Spirale Reversi to avoid scanning the reversed CLERS string to deduce the length of the resulting new cut-border after the reverse merge. The reverse decoding of the operations involves only fixed time procedures, leading to linear time complexity. The only exception is the M' operation whose decoding requires an $\mathcal{O}(n)$ procedure. However, the number of M' operations is upper bounded by the genus of the mesh, which is typically very low compared to the number of triangles. Hence, the impact of these operations in the overall time complexity is negligible. The binary coding of the reversed CLERS string bears approximatively the same cost as the non reversed one, as its entropy rate cannot be modified by the reversing process.

A more efficient compression of the CLERS string has been studied in [103] that is tailored for mostly regular meshes. The binary codes previously presented always guarantee a worst case rate of 2 bits per triangle or less. However, they do not exploit the redundancy inherent in mostly regular meshes and achieve compressed rates considerably larger than other techniques, such as Triangle Mesh Compression, on mostly regular meshes. A well known consequence of Euler's formula is that the average vertex valences in simple triangular meshes is six. In practice, typical meshes have a large proportion of valence six vertices, typically in the order of 75% or more for large meshes. In [103], Szymczak et al. propose a compression scheme for the CLERS string that achieves 0.811 bits

per triangle for large regular meshes (i.e., meshes of infinite size in which the proportion of valence six vertices tends to 100%), although it gives up the tight worst case rate. Besides the compression step they use the standard Edgebreaker compression algorithm and the Spirale Reversi decompression algorithm. The CLERS string is coded in three parts. For the first part the algorithm predicts if the next operation to be decoded is a C, and a hit/miss binary string is coded that corrects any mispredictions. The predictor is based on the observation that a C operation decodes the last incident triangle on the starting vertex of the gate. Hence, the next operation is predicted as C iff the number of already decoded triangles incident on the starting vertex of the gate is five, since the expected final valence is six. For a mesh with a proportion of valence six vertices tending to one, the entropy of this hit/miss sequence tends to zero. The second part classifies these operations into two groups: C, L or S and E or R. A binary sequence, called CLS/ER, distinguishes between the two groups. With the aid of the hit/miss sequence for C this sequence can be coded as a binary variable with an entropy of 0.311 bits per triangle, as proved in [103]. The third part distinguishes between the L and S of the CLS group or the E and R of the ER group and codes this information with another binary sequence, called LS/ER. Note that the C of the CLS group are already identified by the hit/miss sequence. Since half the operations in the CLERS string are C the LS/ER sequence has an entropy of at most 0.5 bits per triangle. The total entropy of this CLERS encoding is thus 0.811 bits per triangle, for large regular meshes. Given that arithmetic coding yields a rate that approaches the entropy this encoding is realizable. However, no practical realization is provided in [103].

Inspired by the ideas behind the above compression scheme, Szymczak proposes an efficient practical compression of the CLERS string in [101, 102]. The compression procedure relies on context based arithmetic coding. The context for encoding an operation is $5d + n_S$, where d is the degree of the starting point of the gate and n_S is the index of the previously coded operation in the set {C,L,E,R,S} (i.e., the ID of the operation). Remember that the CLERS string is coded backward since the Spirale Reversi decompression algorithm is used. The reported compressed rates vary between 0.6 and 1.2 bits per triangle, depending on the mesh. These rates are very close to the entropy of the vertex valence distribution for the meshes tested, which in some way hints at the optimality of the proposed scheme.

3.4.7 Edgebreaker for polygonal meshes

The above incarnations of Edgebreaker only handle triangular meshes. Although every polygonal mesh can be transformed into a triangular mesh without geometric distortion, it is often desirable to preserve the original polygonal form. Furthermore, it is more efficient to encode the polygonal mesh than an arbitrary triangulated version of it.

To this end, King et al. [58] extend the original Edgebreaker algorithm to handle pure quadrilateral meshes, or *Q-meshes*, and mixed quadrilateral and triangular meshes, or *QT-meshes*. Conceptually, the proposed algorithm encodes quads by splitting them into two triangles through one of their diagonals. The split, however, is always performed along the diagonal that makes the coding operations for both triangles adjacent in the CLERS string. This is ensured by always using the diagonal to the left of the gate. A quad can thus be coded as a pair of Edgebreaker operators. At the decoder it is known that each consecutive pair of decoded triangles must be merged into a quad. In order to derive an efficient encoding of such a CLERS string it is necessary to determine which pair of operators are possible. For a triangle there are five possible combinations of visited and not yet visited vertices and edges when it is to be attached at the gate. These five possibilities correspond to the five Edgebreaker codes C, L, E, R and S. As explained in [58], the number of such combinations for an n-gon (i.e., polygon of n sides) is given by the Fibonacci number $\mathrm{F}(2n-1)$. For

a quadrilateral the number of combinations is thus 13. Hence, only 13 pairs of Edgebreaker operators are possible for quads, among the total of 23 that are possible in an arbitrary triangular mesh. Using a technique analogous to that of [57], namely that the best among four possible prefix codes is chosen to code a mesh, they achieve a worst case coded rate of 2.67 bits per vertex (equivalently 2.67 bits per quad) for any simple Q-mesh with no valence two vertices*. If valence two vertices are allowed, the worst case jumps to 3.07 bits per vertex. These results show that it is indeed more efficient to encode Q-meshes as such, instead of coding an arbitrary triangulation of them. In fact, the worst case values above are well below the absolute minimum worst case established by Tutte of 3.245 bits per vertex for triangular meshes. The authors also prove that the absolute minimum worst case rate for a Q-mesh cannot be below 2.24 bits per vertex. The achieved worst case rates are therefore no more than 19% and 37% larger than the unknown absolute minimum, which is fairly close to optimal but still leaves room for improvement.

For the encoding of QT-meshes the problem of distinguishing between quads and triangles arises. Triangles use single labels, while quads use label pairs. King et al. solve this problem, by introducing a sixth label, T, that precedes the label of a triangle. Hence, triangles are coded as TC, TL, TE, TR or TS label pairs. Typically, QT-meshes contain a large proportion of quads, with few triangles. For such meshes, the proposed technique is most effective, since the additional overhead is introduced only for triangles. An extension to arbitrary polygonal meshes is also proposed. The same rule that is used to split quads can also be used to split arbitrary polygons into triangles. An extra bit is encoded with each label to signal if the triangle is to be merged to the next one or not. Note however, that R and E triangles do not require the extra bit, as they can only occur at the end of a polygon, if valence two vertices are not allowed. The worst case cost of this encoding is 5 bits per vertex using the CLERS encoding of [57].

The experimental results show that the prefix codes yield compressed rates of 1.3 and 2 bits per vertex for Q-meshes, depending on their regularity. These results are compared to the conditional entropy coding of the CLERS labels with a 1 and 3 labels of memory (i.e., context) of random triangulations of the Q-meshes. The results show that the prefix codes consistently outperform both entropy coding schemes for random triangulations, which confirms the theoretical results outlined above. The authors also experiment applying the same entropy coding to the labels of Q- and QT-meshes, instead of using the prefix codes. For Q-meshes the encoding with a 1 label memory yields results slightly inferior to the prefix codes. However, a 3 label memory greatly improves the coded rate, achieving between 0.25 and 0.85 bits per vertex, which is several times better what can be achieved with random triangulations. For QT-meshes, the encoding with a 1 label memory yields considerably inferior results than random triangulations (2.5∼4.1 vs. 2.4∼2.5 bits per vertex). This is due to the overhead incurred in signaling quads vs. triangles. However, a 3 label memory is capable to capture the label interdependencies and a rate between 0.8 and 1.1 bits per vertex is achieved, compared to 1.9 to 2.2 bits per vertex for random triangulations using the same entropy coding.

Another face based algorithm for polygonal meshes is proposed in [60]. While not exactly an Edgebreaker derivative, this algorithm reduces to Edgebreaker for triangular meshes. The key observation, also made in the work above, is that an n-gon might interact with the gate and cut-border in a finite number of ways, namely P_n. They derive the recursive relation $P_n = 3P_{n-1} - P_{n-2}$, where $P_3 = 5$ (triangle) and $P_4 = 13$ (quad). As previously mentioned, P_n also verifies $P_n = \mathrm{F}(2n-1)$. The authors describe a way to obtain the index, between 0 and $P_n - 1$, describing the interaction of the next polygon, as well as a way to derive back the interaction from the index. The encoding algorithm thus proceeds by writing the degree (i.e., number of sides) of the next polygon

*Valence two vertices are rare in practice and undesirable in many applications.

and its interaction index to the stream. The gate and cut-border are updated in a way equivalent to Edgebreaker. They propose prefix codes for Q- and QT-meshes which achieve a worst case of 3.5 and 4 bits per face, respectively. The experimental results achieve rates between 2.2 and 3.4 bits per vertex, depending on the mesh. The theoretical and practical results are inferior than those achieved by the Edgebreaker extension above. Nevertheless, the algorithms are equivalent and only differ in the binary encoding, which happens to be less effective in this case.

3.4.8 Triangle Mesh Compression derivatives

From the review above, it is clear that the best compressed connectivity rates for typical meshes are obtained by Triangle Mesh Compression. In fact, Triangle Mesh Compression has long be considered as the state of the art in connectivity and geometry coding. For a few years no work was published that improved on its results.

Somewhat recently, Alliez and Desbrun [2] have analyzed Triangle Mesh Compression's bottlenecks and improved its connectivity coding results, in particular for non-regular meshes. Their analysis shows that the encoding of `split` commands and their associated offsets seriously hampers the coded rate for non-regular meshes, where the number of split commands is not negligible. The `split` commands arise whenever the cut-border closes a cavity and becomes self-intersecting. This often occurs as the cut-border grows around a dense area (i.e. with many triangles), as shown in Figure 3.11 and is due to the simple order in which subsequent pivots are chosen (i.e., next one in active list). As suggested in the middle of Figure 3.11 selecting a vertex in the interior of the cavity as the next pivot would decrease the chance of closing the cavity. In [2] an adaptive pivot selection is proposed to achieve this goal. When the pivot becomes full, the next pivot is chosen as the vertex with the fewest free edges in the active list. If more than one vertex matches this rule, the average number of free edges of a vertex and its two neighbors in the active list is considered. If there is still a non-unique solution, an average with the four neighbors is considered and so on until there is a unique selection. This criteria often results in selecting vertices inside the cavities. For irregular coarse meshes the authors report a 70% decrease in the number of splits. The other source

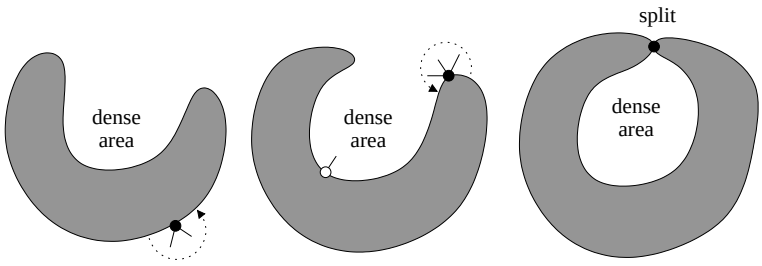


Figure 3.11: Cause for splits in the original Triangle Mesh Compression algorithm.

of decreased compression efficiency is the encoding of the split offsets, which are often large. Alliez and Desbrun propose to encode the offset after sorting the active list's vertices in order of increasing Euclidean distance to the pivot. Using this technique the magnitude of the offset is drastically reduced and often becomes zero. However, this sorting requires that the geometry be encoded interleaved with the connectivity, which might be undesirable in some applications. In such a case, they propose to use the same offset as in Triangle Mesh Compression, but with a sign. This effectively reduces the magnitude of the offsets. The authors also propose an alternative way to handle holes. Instead of adding one dummy vertex to each hole, they add a single *ghost* vertex that is connected to each boundary vertex. The valence of the ghost vertex is coded at the beginning and a special

code is inserted each time the ghost vertex is encountered, instead of the `add` command. Since the ghost vertex is probably non-manifold it is never chosen as a pivot. With some other minor modifications of the decoding algorithm holes are easily and efficiently handled. Using this technique the compressed rate for meshes with a large number of holes is not largely affected. Finally, they use an arithmetic coder instead of the Huffman and run-length coding. The experimental results show that these modifications of Triangle Mesh Compression always increase the compressed rate and particularly so for highly irregular meshes. The rate is reduced by 10% in average and up to 20% in some cases.

Alliez and Desbrun [2] also provide a theoretical study of valence based coding. They prove that the highest possible first order entropy of the vertex valence distribution, subject to the constraint of Euler's formula, is 3.245 bits per vertex. This is exactly the same as Tutte's bound. Hence, this proves that valence based coding is optimal, in the worst case sense, provided that the number of splits is negligible. Furthermore, they verify that the zero order entropy of the valence distribution accurately predicts the coded bitrate, within 2%.

More recently Khodakovsky et al. [55] have extended valence based coding to polygonal meshes. The entropies of the vertex graph and its dual, the face graph, are obviously the same as the information content does not change. Based on this remark, the authors propose that an optimal polygon mesh coder should be dual. Following this proposition, they extend the valence based scheme by coding the degree of a face (i.e., its number of sides), each time the vertices of a not yet coded face are to be processed. Each time a new face is encountered, all its free vertices are coded and added to the active list. Apart from these modifications, the algorithm is basically the same as that of Triangle Mesh Compression. For the pivot selection they use the same heuristics as Alliez and Desbrun above. The face degrees and vertex valences are coded to separate streams. In the case of a triangular mesh the entropy of the face degrees, or of the vertex valences in the dual graph, is zero and thus the corresponding stream can be coded in virtually no space. For the entropy coding the authors use a context based arithmetic coder. The contexts are derived from the degree of the neighboring faces and vertices. The coded rates for polygonal meshes is about the same as the best results for the polygonal version of Edgebreaker [58]. For triangular meshes, regular and irregular, the proposed technique slightly outperforms the scheme of Alliez and Desbrun above. The advantage over other methods is that the same coder is capable of coding arbitrary polygonal meshes and yet it achieves code rates as good as, or even better than, the best coders tailored specifically to triangular or quadrilateral meshes. The authors also theoretically derive the worst case rate as a function of the ratio $r = F/V$ ($r = 2$ and $r = 1$ for triangular and quadrilateral meshes, respectively), under the assumption of a negligible number of splits. The maximum of this function occurs for quadrilateral meshes, $r = 1$, leading to a worst-case rate of 2 bits per edge. This coincides with the minimum number of bits necessary to encode the edges of an arbitrary planar graph as found by Tutte [110]. Hence, this proves that this dual coding approach is optimal, in a worst case sense, for the whole set of possible meshes. Note that this does not preclude the fact that more efficient connectivity encodings can be found for a given corpus of meshes. For the triangular mesh case, $r = 2$, the upper bound becomes 3.245 bits per vertex, as is the case for valence based coding of triangular meshes. The authors also point out, based on the duality remark, that the coded rate should be measured in bits per edge, instead of bits per vertex or face, as the number of edges is the only invariant quantity under duality.

A very similar, independently developed, algorithm has also been recently proposed by Isenburg [41]. Triangle Mesh Compression is extended in the same way as above to handle polygon meshes and context based arithmetic coding is also used for entropy coding. The context formation is, however, a little different. The context for face degrees is deduced from the average valence of the

known incident vertices. Following the duality principle, the context for vertex valences is deduced from the degree of the face for which it is being added. The search for the next pivot is also modified. In fact, the selection algorithm proposed by Alliez and Desbrun [2] explained above is an expensive operation as it requires an extensive search on the active list. Isenburg proposes instead to use the first vertex with the lowest number of free edges around the active list in counterclockwise order. Furthermore, this search is used only if the lowest number of free edges is 0 or 1, otherwise the next vertex in the active list becomes the pivot. Pointers to the potential next pivot vertices can be rather easily maintained with a low impact on the time complexity of the algorithm. The results reported are, in general, slightly better than those of Khodakovsky et al. above. The authors also propose a simple mean of dealing with non-manifold meshes. Each non-manifold vertex is replicated so as to obtain a manifold mesh. Each time a new vertex is coded through an `add` command, an arithmetically coded binary flag is coded signaling if it is a duplicate of a previously coded vertex or not. If it is a duplicate the index of the duplicated vertex is encoded in $\log_2 n$ bits, where n is the number of vertices coded so far. The decoder can thus "un-replicate" the relevant vertices and recover the original non-manifold connectivity. Since the number of non-manifold vertices is typically small, this simple strategy only marginally affects the coded bitrate.

In a further paper, Isenburg and Alliez [42] complement the above connectivity coding scheme with geometry coding. It uses the parallelogram predictor from Triangle Mesh Compression in way tailored to polygonal meshes. The author observes that polygons, as found in typical meshes, are often quasi-planar and convex. Conversely, adjacent polygons often exhibit a crease angle between them. It is therefore more convenient to predict the position of a new vertex from other ones of the same polygon than from neighboring polygons. Hence, the parallelogram rule should be applied, as much as possible, *within* polygons instead of *across* polygons. The traversal order of [41] allows the effective application of this heuristic and more than 80% of the vertices are within predicted, in average. The prediction residuals are coded with a context adaptive arithmetic coder. As the distribution of within prediction errors is much more skewed than that of across predictions, different contexts are used for the two. This avoids "polluting" the skewed distribution with large values that would significantly decrease the compression performance. Using this extension to polygonal meshes, the coded rate is reduced by more than 20% for quantizations between 8 and 12 bits per coordinate. The reported coded rates are 11 bits per vertex on average, ranging from 7.3 to 15.4, for a 10 bit quantization. For triangular meshes, the rates are very similar to those of Triangle Mesh Compression, showing that the majority of the benefit comes from the large percentage of within predictions and not solely from the use of an arithmetic coder.

3.4.9 Angle analyzer

Very recently another, more efficient, encoding algorithm has been proposed by Lee et al. [64], called *Angle-Analyzer*. The proposed algorithm is capable of coding the connectivity and geometry of T-, Q- and QT-meshes. The authors claim that handling higher degree polygons as combinations of triangles and quads should not significantly affect the bitrate as polygonal models typically contain few such faces. The connectivity coding attempts to mix the simplicity of face based coding schemes such as Edgebreaker and Cut-Border Machine (see Section 3.4.10) with the efficiency of valence based coding. They do this by redefining the five basic Edgebreaker symbols and exploiting many of the valence based improvements presented in [2]. The algorithm always operates on the *active gate*, which is an oriented edge. Of the two faces incident on the gate, the one on its left, or *back face*, was previously coded, while the one on its right, or *front face*, is the next face to be coded. The vertices of the front face which are not part of the gate are called the *front vertices*. The gates are stored in ordered lists and a stack of such lists is maintained. At initialization time an arbitrary face is chosen

as the seed face. Its edges are stored as gates in the ordered list at the top of the stack. At each coding step a gate in the list at the top of the stack is selected as active and processed. The symbol specifying how to stitch the front face is coded and the gates incident on the front face replaced by the other edges of the front face. Each time a new front vertex is encountered its geometry is coded. Whenever a gate list becomes empty it is popped from the stack. When the stack is empty all the polygons of the connected component have been coded. As in the other region growing algorithms the gate lists define a cut-border, inside which all faces have been coded.

We refer to already coded and not yet coded vertices as *visited* and *free* vertices, respectively. The symbols for triangles are as shown in Figure 3.12 and are as follows. If the front vertex is free a `C` (create) is output. If the, visited, front vertex can be found by turning clockwise around v_1 or counterclockwise around v_0 a `CW` (mesh clockwise) or `CCW` (mesh counterclockwise) is output, respectively. If the front face is actually a hole an `S` (skip) is output. Otherwise the, visited, front vertex is elsewhere on a gate list and a `J` (join) is coded along with the offset of the front vertex. The offset is coded in the same way as in [2] above, except that all gate lists in the stack are considered instead of just the one at the top. This avoids the need of a merge symbol. In fact, if the front vertex is in the list at the top of the stack a split of that list should be performed. Conversely, if the front vertex is on another list down the stack that list should be merged with the one at the top. The symbols for quadrilaterals are shown in Figure 3.13 and are as follows. If both of the front vertices are free a `C2` (create 2 vertices) is output. If the left front vertex is visited and is found by turning clockwise around v_1 and the right one is free a `CR` is output. For the symmetric case a `CL` is output. If both front vertices are visited a `Mesh` symbol is output. If both front vertices are visited, the left one can be found by turning clockwise around v_1 and the right one by turning clockwise around the left one a `DCW` (mesh double-clockwise) is output. For the symmetric case a `DCCW` (mesh double-counterclockwise) is output. The case where the front face is a hole is handled by the same `S` symbol as for triangles. The remaining case is when one or both of the front vertices are visited and can be found elsewhere in the gate lists. If this occurs a `JQ` (join quad) is output along with two offset values calculated as for triangles. As shown in Figure 3.13, three possible configurations can arise for a JQ: (a) both front vertices are visited and cannot be found by turning around the active gate's vertices; (b) one front vertex is free and the other visited but cannot be found by turning around an active gate vertex; and (c) both front vertices are visited and one of them can be found by turning around a vertex of the active gate. If one the front vertices is visited a -1 offset is coded for it. These three possible configurations do not require different symbols, as the decoder can deduce which one happened by local exploration. Whether to split or merge gate lists is decided in the same way as for triangles. Since a different set of codes is used for triangles and quadrilaterals, no extension is required to handled QT-meshes (the set of symbols to use at each coding step depends on the degree of the front face). Worth noting is also the fact that the symbols above correctly handle meshes with holes and handles and that no extensions are required.

Inspired by the findings of [2], the gate that forms the smallest angle with the following gate along the cut-border is always selected as the active gate. This choice helps in minimizing the number of `J` and `JQ` symbols required and proves to be much more effective than the heuristic used in [2] in achieving this goal. For triangle and quadrilateral meshes the symbol distribution is largely dominated by `C`s and `CR`s, respectively. The symbols and offsets are thus efficiently compressed through an arithmetic coder. The connectivity code generated by Angle-Analyzer is extremely efficient. The reported connectivity bitrates improve on those of Triangle Mesh Compression and the valence based coding of [2] by 38% and 35% in average, respectively. For uniform meshes, the improvement of Angle-Analyzer exceeds 50%.

For geometry compression two methods are proposed. The first one uses the popular parallelo-

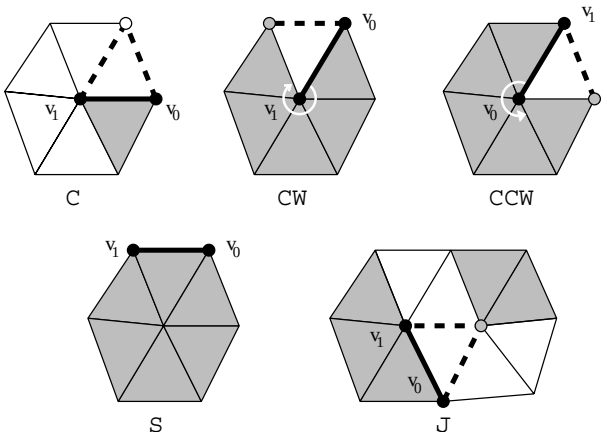


Figure 3.12: Set of symbols used to code triangles. The thick solid line is the current active gate (v_0, v_1). Dashed lines are gates to be inserted. Black, white and gray dots are the gate's vertices, new front vertices and previously coded front vertices, respectively.

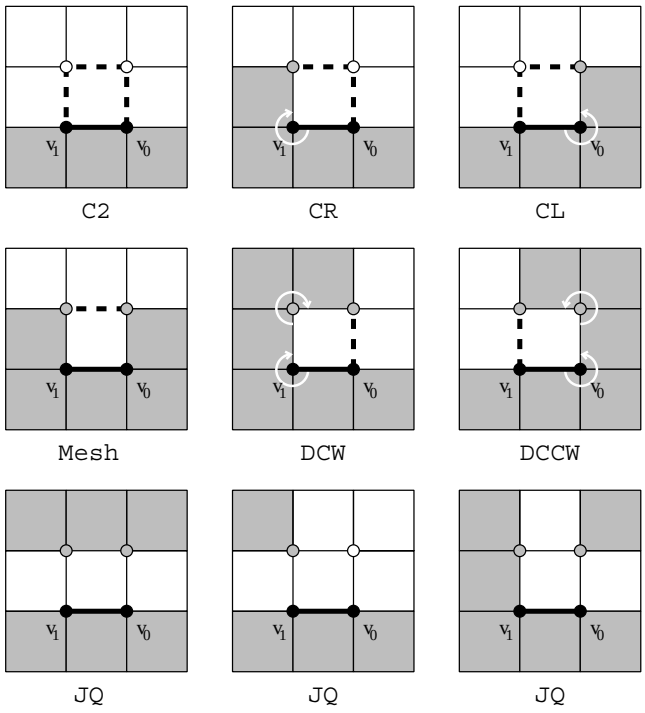


Figure 3.13: Set of symbols used to code quadrilaterals. The thick solid line is the current active gate (v_0, v_1). Dashed lines are gates to be inserted. Black, white and gray dots are the gate's vertices, new front vertices and previously coded front vertices, respectively. The last row shows the three possible configurations that can be coded with a JQ symbol.

gram predictor previously described. However, instead of first quantizing the global coordinates, the prediction errors are quantized in a coordinate system local to each vertex. To avoid drift buildup the standard DPCM feedback loop is used (see Section 3.3.3). As has been proved in [56] (see Section 3.4.12) a local coordinate system allows a better encoding. The active gate is chosen as the local x axis, the back-face normal as the $-z$ axis and the cross product of the z and x directions as the y axis. The second geometry coding method uses three angles. The two angles formed between the sides of the "chamfered parallelogram" at the active gate's vertices and the dihedral angle between the front and back faces. The angle values are uniformly quantized and compressed with an arithmetic coder. No prediction is used for this second method. The coded bitrates achieved by both methods are very similar, depending on the mesh one or the other might be slightly better. Compared against Triangle Mesh Compression these methods improve the coded bitrate by 19% on average, for comparable MSE distortions (as defined in Section 3.3.4). Furthermore, the angle based geometry coder gives smoother results when coarse quantization is used, leading to more pleasant results.

Although the proposed gate selection algorithm makes for a very efficient connectivity code, its implementation can, in our opinion, present portability problems that the authors fail to mention. In fact, calculating the angle, or its cosine, between successive gates in the gate list involves floating point computations. Furthermore, these computations depend on the decoded vertex positions, which also involve floating point arithmetic. However, it is of utmost importance that the decoder is able to exactly reproduce the encoder's computations leading to the gate selection. If this is not ensured, the decoder could choose a different gate in cases where the candidate angles are only slightly dissimilar. Such a mis-selection would lead to a totally corrupted decoded connectivity. Unfortunately it is almost impossible to ensure identical computations at encoder and decoder, due to potentially different Angle-Analyzer implementations, potentially different implementations of mathematical routines in the platform's library and most importantly due to slightly different floating-point arithmetic implementations across CPUs. Because of this, it seems to us that the angle based gate selection heuristic, while very interesting from a theoretical point of view, is not robust enough for a practical use. The authors propose, nevertheless, another heuristic analogous to that of [2]: chose the next gate based on the number of already decoded polygons around the first vertex of the gate (i.e., v_0). While robust, this heuristic is less efficient in avoiding Js and JQs and leads to an increased connectivity bitrate.

3.4.10 Other methods

Although the review of polygonal mesh coding above is fairly exhaustive the field is very rich and thus many interesting methods have been omitted from the above discussion. In this section we provide a brief overview of other popular methods.

Independently of Edgebreaker [89], Gumhold and Strasser [36] proposed a very similar face based coding approach, called *Cut-Border Machine*. It uses five operators: *new vertex*, *connect backward*, *connect forward*, *split cut-border* and *border*. The first three are the same as the C, L and R operators of Edgebreaker, respectively. The fourth corresponds to the S operator, except that the split index is coded (as in Triangle Mesh Compression). Since the Cut-Border Machine encodes the split offset no E operator is required. On the other hand, the border operator is introduced to explicitly handle the case where the gate is a boundary edge. Hence, Edgebreaker's M operation is not required. For non-zero genus meshes the Cut-Border Machine uses the *cut-border union* operator which is the same as Edgebreaker's M', with the same associated index and offset. Additionally, this algorithm proposes a rather efficient way to handle non-orientable meshes, requiring only one bit per split cut-border and merge cut-border operators. The Cut-Border Machine was originally developed

for real-time compression and rendering, in the same line as Deering's Geometry Compression. It was later refined by Gumhold [35] for maximum compression, at the expense of increased time complexity. The refined version uses a context based arithmetic coder, instead of prefix codes, and an alternative way of handling holes. The achieved connectivity rates range between 1.2 to 2.7 bits per vertex for typical meshes and can be as low as 0.3 for some regular meshes. Although it this is not as efficient as Triangle Mesh Compression results are within 36% in average, but with a more efficient implementation in terms of time complexity.

A third class of region growing connectivity coding approaches that we have not mentioned is that of *edge based coding*. This class of methods was first introduced by Isenburg's *Triangle Fixer* algorithm [40]. The basic idea is that the coded region is grown by coding an additional triangle and vertex with a T operator and additional codes are used to stitch the edges of adjacent triangles. As in Edgebreaker and Cut-Border Machine a cut-border is maintained and all coding operations are performed on the gate. A new triangle incident on the gate, and its opposite vertex, are coded with a T operator. The L and R operators stitch the gate with the preceding and following edges in the cut-border, respectively. The S operator stitches the gate with some other edge of the cut-border, splitting it in two. When the cut-border becomes a digon an E operator is encoded terminating the processing of the current cut-border. Note that since the E operator is used, no offset need be encoded for S. Whenever the gate is to be stitched to an edge in another cut-border an M operator is coded, together with the stack index and two offsets, as in Spirale Reversi (see Section 3.4.6). Holes are explicitely handled by coding an H operator, along with the hole's boundary length, whenever the gate is a boundary edge. Decoding is performed in reverse order, in an analogous way to Spirale Reversi. The symbols are coded with an order-3 adaptive arithmetic coder. The reported connectivity rates range between 1.43 and 3 bits per vertex, with as low as 0.77 bits per vertex for some regular meshes. In the same paper the author proposes an extended version, *Triangle Strip Compression*, which is able to efficiently code the precomputed strips of the mesh along with the connectivity. This has the advantage that efficient rendering of the decoded model is straightforward, without requiring the re-computation of efficient strips (which is known to be a complex problem). The Triangle Fixer connectivity coder was later extended by Isenburg and Snoeyink [43] to polygonal meshes. Given its edge based nature, this is achieved by simply replacing the T operator with a F_n operator, where n is the degree of the face being added.

On the geometry coding front, the parallelogram predictor introduced by Touma and Gotsman in Triangle Mesh Compression remains one of the most effective and simpler predictors proposed. While very effective in smooth parts of a model, it performs poorly across sharp creases like often found in CAD models. However, a geometry coder for triangle meshes whose traversal order is driven uniquely by the connectivity cannot possibly know the presence of creases. Kronrod and Gotsman [59] propose to code the connectivity in a way that optimizes the efficiency of the parallelogram predictor. For this they construct a weighted graph, where the weights are the prediction errors across adjacent triangles. Then they build a minimum weight cover tree of this graph. The traversal order is given by this tree, which minimizes the spread of the prediction error distribution. This tree, however, is not sufficient to recover the full connectivity. The boundaries of the cover tree form a set of simply connected polygons that are coded in 2 bits per triangle. While this scheme adversely affects the coded connectivity rate, this loss is more than made up by the improved geometry coding. The entropy of the coded models is reduced, in average, by 45% for CAD models and 10% for smooth models when compared to Triangle Mesh Compression.

Following the widespread use of spectral methods for the lossy coding of images, such as the DCT in JPEG [77], Kami and Gotsman [54] propose the use of spectral transforms to code mesh geometry. While spectral transforms, such as the DCT, on regular 2D quadrilateral grids are well

known they remain largely unexplored on irregular grids as found in polygonal meshes. Kami and Gotsman partition a mesh and compute a spectral transform based on Laplacian operators. In order to keep the computation effort reasonable the size of the sub-meshes should be carefully chosen. Furthermore, all sub-meshes should be of roughly the same size and the number of boundary edges minimal so as to mitigate as much as possible the "blocking" artifacts introduced by the partitioning. As is the case in image coding, the high-frequency components of the resulting spectra are less relevant to the visual quality and can be discarded. For smooth models the reported results are promising, in that very good quality reconstructions are obtained with rates considerably inferior to those of Triangle Mesh Compression (between a half and a third). A nice property is that the coded geometry is progressive, as discarding less high-frequency spectral coefficients improves the reconstructed quality. For non-smooth models, such as those often used in CAD, the sharp creases lead to large high-frequency coefficients that create ringing artifacts if discarded. Note that the mesh connectivity must be available before computing the transform, and therefore connectivity and geometry compression cannot be interleaved in this case.

3.4.11 Coding of non-manifolds

All the connectivity coding algorithms reviewed above require that the input mesh is 2-manifold. While this might appear very restrictive at first, 3D surfaces are in general locally planar and thus the corresponding meshes fulfill this requirement. Nevertheless, non-planarity can often arise at some points of the surface leading to a non-manifold. As an example, two cubes that are adjacent by and edge form a non-manifold mesh. The usual approach is to split the non-manifold vertices in two or more so as to make the surface planar everywhere. An efficient method to do this is given in [33, 34]. Note that the resulting mesh can have several components. After the conversion, any of the above connectivity coding algorithms can be applied on the resulting mesh. This transformation is, however, lossy as the original connectivity cannot be faithfully recovered. Depending on the application, lossy connectivity coding might be undesirable. Furthermore, if the number of non-manifold vertices is large the replicated vertex positions will often lead to a significant increase of the compressed bitrate.

An straightforward way to code the information required to recover the original non-manifold mesh is to specify the indices of the pairs of replicated vertices. This encoding is however very costly and requires $2\log_2 V'$ bits for each pair of vertices, where V' is the number of vertices of the mesh after conversion to manifold. If this information is coded prior to the geometry, no repeated encoding of replicated vertex positions is required leading to some bitrate savings. A more efficient, yet very simple, scheme is used in [41] (see Section 3.4.8) and works as follows. Each time the connectivity coder encounters a new vertex a binary flag is coded indicating if it is a duplicate of a previsouly coded vertex. If it is a duplicate, the index of the duplicated vertex is coded in $\log_2 n$ bits, where n is the number of vertices coded so far, and no geometry is coded. The binary flag is arithmetically coded. Although more efficient, this method can incur a significant overhead in large meshes since the index can be as large as the number of vertices.

Two more sophisticated methods yielding better compression are proposed by Guéziec et al. [32]. They are proposed as extensions to Topological Surgery [106] (see Section 3.4.3) but are easily applicable to other connectivity coding schemes. Each non-manifold, or *singular*, vertex is replicated using the method of [33, 34]. The set of replicas of a vertex is called a *cluster*. The first method, referred to as *stack-based*, is extremely simple and maintains a list of cluster representatives, which is initially empty. When a regular vertex is found a `NONE` command is coded. When the first representative vertex of a cluster is found a `PUSH` command is coded and the vertex inserted at the head of the list. When subsequent representatives of a cluster are found, a `GET(`i`)` command is

coded, where i is the index of the cluster in the list. Finally, when the last representative of a cluster is found, a `POP(`i`)` command is coded, instead of `GET(`i`)`, and the cluster representative is removed from the list. This code requires one bit for regular vertices, which can be further arithmetically coded, and two bits for the other commands. In addition, the `GET` and `POP` commands require at most $\lceil \log_2 m \rceil$ extra bits for the index, where m is the number of non-manifold vertices in the original mesh. Since m is typically much smaller than the total number of vertices, this method incurs an overhead much lower than the above methods, yet being very simple.

The second method, referred to as *variable-length*, extends the previous one by exploiting the fact that adjacent non-manifold edges in the original mesh often form long paths. Derived from the connectivity coding of the manifold mesh a father-child relationship can be established between vertices. For Topological Surgery, this relationship is directly derived from the vertex spanning tree of each component and forms a forest (set of rooted trees). The conversion of paths of non-manifold edges to a set of paths of manifold edges will often lead to a given number of adjacent vertices that are clustered together along two, or more, paths of the father-child forest. This relationship can be thought of as an open zipper. The stack-based approach will independently code each pair of vertices of this "zipper". However, a joint encoding would be much more efficient and can be handled by the so-called *stitch* operation. A stitch specifies one vertex on each side of the "zipper" and a length and can be coded by extending the commands of the stack-based approach, other than `NONE`, with a length. Many of the `GET` and `POP` commands can be replaced by `NONE`, because the necessary information is coded by the stitch. A method for constructing efficient stitches from the father-child relationship is provided.

Using these methods on a set of non-manifold meshes Guéziec et al. [32] report average savings in excess of 8% when non-manifold connectivity is included on Topological Surgery (quantizing coordinates to 10 bits, colors to 6 bits and normals to 10 bits). Although the non-manifold connectivity incurs an overhead, it is more than made up by the savings in geometry and property coding. The stack-based method achieves overhead rates in the range of 3.6 to 10 bits per replica vertex. The variable-length approach often reduces this figure considerably and can reach down to 0.8 bits per replica vertex in some cases. The two methods, which use a common syntax, have been adopted by the MPEG-4 standard [49] as part of the 3D mesh coder.

3.4.12 Progressive methods

In the previous section we have provided an extensive review of single rate mesh coders. In this section we provide a brief overview of progressive coders for completeness. As previously explained in Section 3.4 progressive methods start by coding a coarse version of the mesh and subsequently add more and more vertices until the original mesh is recovered. Most progressive coders obtain the coarse mesh by topological simplification and replay the inverse of the simplification steps along with the corresponding vertex displacements to recover the original, fully detailed, mesh. The different meshes that can be decoded between the coarse and fully detailed ones are referred to as *levels of detail*. (LODs).

The first successful such method, *Progressive Meshes*, was developed by Hoppe [37] one year after Deering's Geometry Compression coder. It defines a single simplification operation: *edge collapse*. It consists in collapsing an edge by merging its two end vertices v_0 and v_1 into a new vertex v'_0. The two triangles incident on the collapsed edge are also collapsed. This operation reduces the vertex, triangle and edge counts by one, two and three, respectively. The inverse operation is the *vertex split*. It splits the vertex v'_0 back into v_0 and v_1, recovering the original local topology, by splitting two of the edges incident on v'_0. The *edge collapse* operations are successively applied to obtain the coarse base mesh. At each step the edge that minimizes a simplification error will be selected. To

recover the original mesh from the base one, each vertex split can be coded as the index of v'_0 and a permutation index to identify the two incident edges to be split. The permutation index requires 5 bits in average*, leading to a connectivity cost of $\lceil \log_2 V \rceil + 5$ bits per vertex. The positions of the new vertices are coded as offsets from the split vertex, which are further Huffman coded. For a 16 bit quantization the geometry coding cost is between 31 and 50 bits per vertex. Compression of Progressive Meshes is further refined by Hoppe [38]. In this improved version bitrates between 60 and 110 bits per vertex are obtained for models with normals and texture coordinates (using 16 bits for position coordinates, 8 bits for normal coordinates and 16 bits for texture coordinates). A nice property of Progressive Meshes is its ability to smoothly interpolate between levels of detail.

Since Progressive Meshes codes only one vertex at a time it generates a finely progressive coding. However, it incurs a considerable bitrate penalty in doing so. Taubin et al. [104] propose a more efficient method, *Progressive Forest Split*, that jointly codes multiple vertex split operations at a time, at the expense of some granularity. The base mesh is encoded using Topological Surgery. At each subsequent step, the refined mesh is obtained by a *forest split* operation, instead of a single vertex split. The forest split operation consists in cutting the mesh along a forest of the vertex graph of the current mesh, splitting the resulting boundaries apart, filling each of the resulting tree boundary loops with simply connected polygons, and finally, displacing the new vertices. The forest can be coded with one bit per edge. The simply connected polygons can be encoded either as in Topological Surgery or using a fixed length coding of two bits per triangle. As the former might not be efficient for small polygons, the most compact encoding is selected for each forest split, and signaled with a one bit. If the forest at each coding step has the maximum possible number of edges, this encoding leads to about 4 bits per new triangle, which compares very favorably to the $\lceil \log_2 V \rceil + 5$ bits per vertex of Progressive Meshes. The vertex displacements can be coded as in Progressive Meshes. A more efficient encoding is, however, achieved by predicting the vertex positions as the output of a global smoothing algorithm. As the authors report, this can decrease the geometry bitrate by as much as 20-25%. The results show that this progressive encoding yields connectivity rates that are only 1.7-2.7 times larger than the single rate results of Topological Surgery. Progressive Forest Split has been adopted by the MPEG-4 standard [49] for the encoding of LODs.

Another scheme that removes a vertex at a time is proposed by Li and Kuo [65]. The simplification step is performed through a *vertex decimation* method. Each time a vertex is removed, its 1-ring is triangulated (with no additional vertices). The vertex to be removed at each step is simply selected as the one that introduces the least distortion. The refinement operation is performed by selecting a region, removing all its interior edges, inserting an interior vertex, and finally, connecting it to all the vertices of the region. This is encoded as a triangle index, that locates the region in the mesh, and an index into a table of topological patterns. This table lists the possible neighborhood patterns after a simplification step. For space reasons it is limited to 231 entries, requiring 8 bit indices, and is able to handle vertices up to valence 10. Higher valence vertices are avoided as never being eligible for simplification. In average, this requires $log_2 V + 6$ bits per vertex, which is comparable to Progressive Meshes. The geometry is differentially coded, by predicting a vertex as the average of its 1-ring. A novel approach is introduced to interleave the geometry and connectivity information. A uniform deadzone quantizer is employed for embedded quantization, instead of the more generally used single uniform quantizer. The progressively finer position approximations are explicitly multiplexed with the progressively finer connectivity to achieve a better progressive rate-distortion performance. In fact, it would not be efficient to provide a finely quantized vertex position when only few polygons have been reconstructed. As is the case for Progressive Meshes, this method provides modest connectivity rates because new vertices are coded one by one.

*The average vertex valence is 6 and the number of permutations of 2 on 6 is 30.

An important improvement of Progressive Meshes, *Compressed Progressive Meshes*, is proposed by Pajarola and Rossignac [76]. It consists on the following modifications. The vertex split operations are coded in batches, instead of one by one, with about 50% of the previously decoded vertices being split in each batch. For each batch, the split vertices are marked with one flag bit while traversing the previously decoded vertices, instead of explicitly encoding the indices. Also, the incident edges to be split are coded as an unordered set, requiring less bits. Inspired by the Butterfly subdivision scheme (Section 2.5.2), a vertex position is predicted as a linear combination of the average position of vertices with topological distance 1 (i.e., its 1-ring) and the average position of vertices with topological distance 2. This leads to two combined equations for each split-vertex operation. Combined with the knowledge that each split-vertex was produced as the mid-point of the collapsed edge during simplification, only one prediction error value is required for the two vertex positions generated by a split-vertex operation. These error values are Huffman coded. Instead of including the Huffman tables, the variance of the distribution in each batch is coded. The Huffman tables are derived at the encoder and decoder from the coded variance under an assumption of a Laplacian probability distribution. The experiments show that the connectivity rate is about 7 bits per vertex of the final mesh. The comparisons with Progressive Forest Split shows an improvement of roughly 50% overall (connectivity + geometry) for a lower geometric distortion.

Khodakovsky et al. [56] introduced a more effective compression scheme, called *Progressive Geometry Compression*, particularly tailored for highly detailed meshes that exhibits a much better rate-distortion behavior. Departing from the previous schemes it uses multi-resolution analysis, as was already applied by [4], and which has been widely applied to image compression. First the input surface is resampled so as to obtain a semi-regular uniform mesh. This process creates a base coarse mesh along with a bijective map between the domain of the base mesh and the input mesh. The base mesh is coded with Triangle Mesh Compression. Then a wavelet transform is applied on the fine semi-regular mesh that outputs the base mesh and a set of wavelet coefficients. For regular (i.e., valence 6) vertices the Loop subdivision (see Section 2.5.2) is used as the low-pass reconstruction filter, which uniquely determines the high-pass filter. For non-regular vertices a modified Loop subdivision is employed. The forward, or analysis, transform is carried out by solving a sparse linear system. The inverse transform is performed by applying the Loop subdivision and the corresponding high-pass filter. As is the case in image coding, the wavelet coefficients exhibit a very skewed distribution. Furthermore, the wavelet coefficients can be organized in trees that correspond to regions of the surface. The coefficients toward the root of a tree represent coarser detail, whereas fine detail is embodied in coefficients closer to the leaves. These trees are efficiently coded using embedded quantization and the Zerotrees concept pioneered by [95]. Instead of expressing the vector valued wavelet coefficients in the global coordinate system, a local coordinate system is used that has the z axis oriented in the surface normal direction. In this way most of the energy of the wavelet coefficients is captured by the z component. Finally, the zerotree codes are further arithmetically compressed. The rate-distortion results show reductions of up to 4 times of the MSE error, as defined in Section 3.3.4, for equivalent bitrate when compared to Compressed Progressive Meshes. It also provides better compression that the Triangle Mesh Compression and MPEG-4 (based on Topological Surgery) single rate coders for the same L_2 distortion. These greatly improved rate-distortion behavior is due to the combination of effective coding through wavelets and zerotrees and the embedded quantization. Note, however, that the original mesh cannot be recovered as there is an intervening remeshing step. In general this is not a problem, since the remeshing error would be of the same order as the quantization error. Nevertheless, some applications might require to recover the connectivity of the original mesh after decompression, in which case this algorithm would not be suitable.

An efficient progressive coder with lossless connectivity, has been recently proposed by Alliez and Desbrun [1]. It uses basically the same simplification as [65], namely vertex decimation coupled with the efficient valence based coding of [109]. However, it avoids coding the locations of the decimated vertices by using a deterministic ordering and triangulation of the 1-rings that depends only on connectivity information available to the decoder. Furthermore, each time a vertex is removed its valence, or equivalently the degree of the face formed by its 1-ring, is output. The decimation is applied by layers. In a given layer the maximum number of non-interfering vertices are decimated. The 1-rings of vertices that can not be decimated are signaled through a skip code. Decimation layers are interleaved with cleaning layers. The cleaning procedure removes 2 out of 3 triangles by removing valence-3 vertices, obtaining a semi-regular mesh on which a further decimation layer can be applied. Predictive coding is used for the geometry, as usual, but the prediction errors are expressed in a local coordinate system in the same way as in Progressive Geometry Compression. The predictor is the same as in [65] above. The results show that this compression algorithm is almost as efficient as that of the best single rate coders, being only 10% worse than those of Triangle Mesh Compression [109] for connectivity and geometry combined. The reported average connectivity bitrate is 3.69 bits per vertex, which is also close to that of efficient coders such as Edgebreaker [57]. Although not compared to other methods, the progressive rate-distortion behavior appears to be as good as that of Progressive Geometry Compression.

3.5 Coding of parametric surfaces

The compression of parametric surfaces, and meshes of such patches, has received relatively little attention compared to that of polygonal meshes. While many of the techniques reviewed in the last section can be applied to meshes of parametric patches, very few studies have proposed algorithms that are specifically tailored for them. In what follows we provide a brief overview of the literature related to parametric surface compression.

DeVore et al. [16] develop a multiresolution wavelet decomposition of box-splines* to express parametric surfaces. An approximation with a reduced number of coefficients is achieved by selecting only the wavelet coefficients above a given threshold. Reissell [86] proposes an alternative wavelet decomposition, using interpolating functions instead of box-splines. The selection of wavelet coefficients is performed in a similar manner, but following a truncated binary tree structure across sub-bands. A similar wavelet decomposition for surfaces of arbitrary topological type is proposed by Lounsbery et al. [68] in a manner related to subdivision surfaces. While all these decompositions explicitly target compression as an application, no rate-distortion analysis or experimental compression results are provided.

Staadt et al. [97] and Gross et al. [31] propose a B-Spline wavelet decomposition as a generic framework for the compression of uniformly sampled non-parametric, parametric and implicit lines, surfaces and volumes. After selection of the relevant wavelet coefficients, progressive lossy coding is applied using uniform quantization and Huffman coding. The results show, however, that for parametric surfaces a very low rate-distortion performance is obtained. In fact, a very poor visual quality reconstruction is attained at moderate compression ratios, requiring post-processing to smooth the reconstructed surface.

In the spirit of Hoppe's Progressive Meshes [37], Stoddart and Baker [98, 99] propose a progressive coding of surfaces as quadrilateral meshes of generalized biquadric B-Splines using edge collapse operations. The use of generalized biquadric B-Splines guarantees a G^1 continuous surface. Unfortunately, no experimental results are provided regarding compression.

*A multidimensional generalisation of B-Splines.

Another wavelet approach is presented by Malassiotis and Strintzis [71]. They also use box-spline derived wavelets to obtain a multiresolution decomposition, although the high-pass sub-bands are not critically sampled. The coding scheme utilizes scalar quantization followed by Huffman coded DPCM. Experimental results are shown for surfaces fitted to small samples of natural data. Although no clear rate-distortion result is presented, they report rates ranging from 1.11 to 3.7 bits per vertex for an unknown quantization.

Very recently, Furukawa and Masuda [25] propose an interesting approach to coding NURBS surfaces. For each surface the knot vectors and boundary curves are coded first. The boundary curve control points are coded as the deviation from the straight segment between the start and end points of the curve. From the boundary curves an approximate surface is generated, using a bicubic Coons patch. The differences between the control points of this approximate surface and the original is then coded. The boundary curve data and prediction error of control points is transformed with the DCT and uniformly quantized. The quantized coefficients are output to a textual file that is compressed with a general purpose file compressor. The reported compression ratios, with respect to the original losslessly compressed textual data, vary between 3 and 10 for good visual quality. Unfortunately the reported results are difficult to interpret as they don't provide any indication if the distortion is absolute or relative to the model bounding box size.

Finally, we should mention that MPEG-4's Animation Framework eXtension (AFX) [9, 50] includes support for NURBS surfaces, in a way similar to VRML [113]. However, the NURBS data is coded using the general Binary Format for Scenes (BIFS), which consists on simple DPCM and arithmetic coding that is used for general data. Unlike for polygonal meshes, MPEG-4 does not currently include a coder specially designed for NURBS surfaces.

3.6 Conclusions

In the first part of this chapter, two areas of interest to 3D model coding have been reviewed, namely entropy coding and rate-distortion theory. The principles behind entropy coding were introduced as well as two widely used realizations: Huffman and arithmetic coding. Huffman coding is computationally efficient but provides limited compression efficiency for low-entropy sources and it is necessary to resort to blocking to approach the entropy limit. Arithmetic coding, while more computationally demanding, can approach the source entropy arbitrarily close. Multiplier free binary arithmetic coders, such as the MQ, reduce the computational requirements while incurring almost no degradation in compression performance. Furthermore, the implementation of higher order models (i.e., conditional coding) and adaptive variations of entropy coders is straightforward for the arithmetic coders. This makes arithmetic coding a very flexible and powerful entropy coder, even if it incurs a higher computational cost when compared to Huffman coding. About rate-distortion theory it has been seen that for small distortions and IID sources, entropy coded uniform scalar quantizers are optimal and therefore there is no need to consider more complex scalar quantizers, such as Lloyd-Max, if entropy coding is allowed. However, its performance falls short by 1.53 dB from the Shannon lower bound (SLB). In fact, it is necessary to resort to much more complex vector quantization schemes to approach the SLB further. We have also briefly mentioned some alternatives for coding non-IID sources, namely: vector quantization, decorrelating transforms (e.g., DCT, DWT) and predictive coding (i.e., DPCM). Finally, we have defined a meaningful distortion for 3D models, based on the Hausdorff distance. Unfortunately, no simple relation exists between the Hausdorff based distortion and the sample to sample distortion used in the development of the rate-distortion theory. However, both distortions can be expected to have the same order of magnitude and thus the findings remain largely applicable, even if not rigorous.

On the second part of this chapter we have provided a thorough review of the state of the art in single-rate polygonal mesh coding, followed by a brief review of progressive polygonal mesh coding and also by a review of the scarce literature available on parametric surface coding. The coding of a polygonal mesh can be considered in two fairly independent parts: connectivity and geometry. As we have seen, a canonical coding of connectivity as a labeled graph has a non-linear cost. However, Tutte's theoretical study derives an upper bound of 3.245 bits per vertex for an unlabeled triangular graph, a linear cost. Geometry information is usually coded by applying global uniform quantization between 8 and 12 bits, higher precisions being typically unnecessary. Early polygonal mesh compression methods, such as that of Deering [15], were geared toward rendering and use only lossy connectivity coding achieving modest compression. Lossless connectivity coding is introduced by Topological Surgery, by Taubin and Rossignac [106], achieving higher compression overall. The connectivity graph is decomposed in two interlocked-trees and run-length coded. The geometry is compressed using entropy coded linear prediction, following the traversal order of one of the connectivity trees. A refined version of Topological Surgery forms the basis to polygonal mesh compression in the MPEG-4 standard. Higher compression is achieved by the later region growing schemes, of which the valence based Triangle Mesh Compression, of Touma and Gotsman [109], provides best results. Connectivity is handled by coding the valence of new vertices and signaling the cases where the region border grows onto itself through special split codes. The valence distribution being skewed, entropy coding is very effective. Connectivity rates range between 1 and 2 bits/vertex are typically obtained, with as low as 0.2 bits/vertex for mostly regular meshes. Geometry is coded using the parallelogram predictor, a particular linear predictor. Because of the region growing connectivity code, better neighbors can be chosen for the predictor leading to 50% better geometry compression as compared to Topological Surgery (e.g., 8 to 12 bits/vertex for an 8 bit quantization). Another, simpler, region growing approach with a guaranteed worst case, Edgebreaker, is proposed by Rossignac [89]. This face based scheme uses five symbols to signal how each new triangle face is to be attached to the coded region. An encoding with a worst case of 3.67 bits/vertex, which is within 13% of Tutte's bound, has been derived. However, only modest rates are achieved on typical meshes. In fact, providing a worst case rate so close to Tutte's bound, limits the achievable rate on typical meshes to be also close to Tutte's bound. The attempt to guarantee a worst case rate is thus questionable. Several other Edgebreaker encodings without worst case guarantees have been proposed, which can achieve rates between 1.2 and 2.4 bits/vertex. We have also reviewed the numerous Edgebreaker decoding algorithms, of which Spirale Reversi, of Isenburg and Snoeyink [44], functions in linear time. We also reviewed extensions of Edgebreaker to polygonal meshes, which are based on deterministic triangulations of the polygonal faces.

Better connectivity rates are achieved by refinements of Triangle Mesh Compression that resort to adaptive traversal algorithms and that also exploit geometric information in the coding of connectivity. Alliez and Desbrun [2] carry out a thorough analysis of Triangle Mesh Compression and achieve a 10% improvement in average. They also prove that the maximum entropy of the vertex valences is the same as Tutte's bound, hinting at the optimality of valence based approaches under the assumption of a negligible number of split codes. Khodakovsky et al. [55] extend the valence based approach, and the above optimality proof, to polygonal meshes. A similar extension is also provided by Isenburg [41]. Both extended schemes also achieve further improvements by the use of conditional arithmetic coding. Finally, the best compression ratios to date are achieved by Angle Analyzer, proposed by Lee, Alliez and Desbrun [64]. It is a region growing method that mixes the face based approach of Edgebreaker with the adaptive traversal of the valence based approaches above. However, unlike the previous methods the adaptive traversal is driven by the connectivity and geometry of the polygonal mesh, instead of connectivity only. The geometry coding uses DPCM

and quantizes the error on a local coordinate system, instead of the classical predictive scheme on globally quantized coordinates. They obtain improvements on connectivity of 35% in average over the one of Alliez and Desbrun [2] and of 19% on geometry over Triangle Mesh Compression.

Most of the above methods include extensions to handle meshes with holes and handles. All of them, however, require that the connectivity be manifold. In Section 3.4.11 we review several methods to losslessly deal with non-manifold meshes, among which the two efficient ones of Guéziec et al. [32]. In Section 3.4.12 we briefly reviewed progressive polygonal mesh coders. Most progressive coders are based on coding, in reverse, the mesh simplifications steps from the original mesh to a coarse version. Progressive Meshes, by Hoppe [37], uses vertex split (i.e., the inverse of edge collapse) operations but has a non-linear coding cost. Its results are improved by Progressive Forest Split, of Taubin et al. [104], by grouping the edge collapse operations in trees and applying techniques similar to Topological Surgery. Better results are obtained with Compressed Progressive Meshes, of Pajarola and Rossignac [76], by coding edge collapse operations in bacthes and a novel geometry prediction rule. Another simplification operator, vertex decimation, is employed by the scheme of Li and Kuo [65]. The encoding of connectivity and geometry takes place in an special interleaved manner that leads to a better progressive rate-distortion performance.

A different approach is taken in Progressive Geometry Compression, by Khodakovsky et al. [55], based on multiresolution analysis and coding techniques inspired by EZW image coding [95]. Another novelty introduced is the use of a local coordinate system, oriented along a tangential plane and its normal, to express the refinement data. The resulting rate-distortion performance is excellent, obtaining better results than popular single-rate coders. The coding algorithm involves, however, a remeshing step and is therefore not suitable for applications that require the preservation of the original connectivity. Another scheme based on the vertex decimation has been recently proposed by Alliez and Desbrun [1]. However, instead of signaling the location of decimated vertices it uses a deterministic ordering that can be replayed at the decoder. The connectivity is coded using a valence based scheme and geometry coding uses the local coordinate system of Progressive Geometry Compression coupled with the predictor of Li and Kuo [65] above. The performance achieved is similar to that of Progressive Geometry Compression, although no remeshing step is involved.

Finally, in Section 3.5, we reviewed the scarce literature available on the coding of parametric surfaces. The few results available suggest that multiresolution analysis provides poor rate-distortion results on parametric models. Only very modest compression can yield good visual quality. The approach of Furukawa and Masuda [25] for NURBS coding is, however, very interesting although the reported compression ratios are not very large.

Parametric surface coding

4

4.1 Introduction

As reviewed in Chapter 2, parametric surfaces are a convenient and popular method to describe computer authored 3D models. Parametric surfaces are particularly popular in the domain of Computer Aided Design (CAD) and virtual character generation. Although such models are usually compact when compared to polygonal meshes they can still be rather large and could benefit from a controlled compression system. In this chapter we propose such a system for lossy coding of parametric models given as NURBS surfaces. We have chosen NURBS, among the multiple parametric forms, because they are widely used in varied domains of geometric modeling. As an example, they have been recently added to the VRML standard [113]. A useful NURBS coding system should fulfill a number of requirements, as follows.

Generic : models are created on a large variety of systems and under different constraints. The resulting data can therefore have different characteristics depending on the type of application, object being modeled, detail of the model, etc. A NURBS compressor should be able to deliver similar efficiency for these different types of models, be they small or large, very smooth or highly detailed.

Efficient : the amount of bits needed to represent a NURBS model should be small so that the coded models can be economically transmitted or stored.

Guaranteed distortion : the amount of distortion introduced when coding a model should be know *a-priori*. For a large part of applications, such as CAD, it is necessary to provide a guaranteed L_∞ distortion as that defines the tolerance up to which the model is defined.

Flexible : the coding algorithm should not restrict the type of model data that it is capable of handling. For example, no restrictions should be made on the type of knot vectors or degree of the surfaces that can be coded. In fact, it is desirable in some applications to recover the original data as closely as possible, so that the model can be modified as if it was the original.

As we have seen in Section 3.5, few techniques have been proposed for the coding of parametric models, let alone fulfilling the above requirements.

This chapter is organized as follows. Section 4.2 provides a brief overview of the storage required for uncompressed NURBS models. Section 4.3 gives a high-level view of the coder structure to aid in the understanding on each part in the later sections. Section 4.4 introduces the techniques used for coding the knot vectors, with a detailed analysis of the distortion bounds. An analogous development is carried out for control point coordinates and weights in Section 4.5. Section 4.6 treats the special case of surfaces that are closed in some parametric direction or that present degeneracies, while Section 4.7 extends the previous techniques to trimmed surfaces. A detailed evaluation of the resulting coding system is provided in Section 4.8. Finally conclusions are drawn in Section 4.9.

4.2 Uncompressed NURBS

Before introducing the coding of NURBS surfaces let us present, as a reference point, how such data is coded in its uncompressed form. In general, each NURBS based commercial package uses its own coding and few standard exchange formats exist. The aging IGES standard [39], widely used in CAD applications, uses a rather arcane form of textual encoding. Recently, an extension [113] has been added to the popular VRML standard to deal with NURBS, in a way similar to that of polygonal meshes presented in Section 3.4.1. Rational surfaces are expressed by control points in Euclidean $\mathbb{E}^3$ space and their associated positive weights, instead of the more general projective $\mathbb{P}^4$ space. Unlike polygonal meshes, no explicit connectivity information is required. The syntax also supports trimmed surfaces, where the trimming curves are in NURBS or piecewise linear form. Related surfaces can be explicitly grouped for joint tessellation in renderers. A simple example is shown in Figure 4.1. As is the case for polygonal meshes the VRML encoding is not compact but it remains flexible and easily modifiable. Note that for reasonably sized surfaces, the storage cost is dominated by that of control points.

```
NurbsSurface {
   uOrder 3
   vOrder 2
   uDimension 3
   vDimension 2
   uKnot [ 0 0 0 1 1 1 ]
   vKnot [ 0 0 1 1 ]
   controlPoint [
                 0 -1 0
                 0.5 -1 0.5
                 1 -0.7 -0.7

                 0 1 0
                 0.5 1 -0.1
                 1 1 0.6
                ]
   weight  [
            1 0.7 1
            1 0.7 1
           ]
}
```

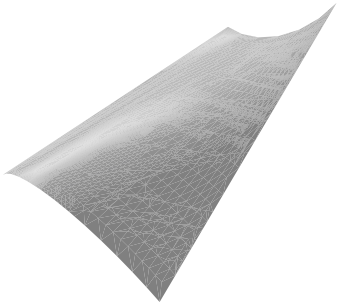


Figure 4.1: Example of a simple NURBS surface in VRML.

More compact binary encodings can be obtained by writing knot values and control point coor-

dinates and weights as 32 bit single precision IEEE floating-point numbers, which is the precision required by VRML implementations. This leads, however, to a usage of 128 bits per control point in addition to the space required for the knot values. Obviously, great savings can be achieved with proper compression of these values, with a controlled amount of distortion, which is the matter of the following sections.

4.3 General coder structure

Figure 4.2 shows a simplified view of the NURBS coder's structure. The basic building blocks are prediction, quantization and entropy coding. As shown in the figure, knots and control points are handled independently, although in a similar manner. The first step is to normalize all input NURBS surfaces. This involves exploiting the properties presented in Section 2.4 to obtain knot vectors of the form $\{0, 0, u_2, \ldots, u_{r-3}, 1, 1\}$ and weights in the interval $(0, 1]$. Note that none of these modifications affect the geometry of the surface. As degenerate and closed surfaces can have multiple coincident control points, the optional "duplicate detection" block, explained in Section 4.6, can be used to improve the compression efficiency.

Prediction and quantization follow the classical DPCM scheme (see Section 3.3.3) to exploit the inherent redundancy in knot and control point values. Since 3D models are usually visualized in an interactive manner where the user can zoom into any part of the object, it is important to guarantee a maximum deviation at any point of the decoded surface. Furthermore, in CAD and CAD-like applications, a guaranteed bound for the maximum deviation is often required. Therefore an L_∞ distortion measure seems more appropriate than the more widely used L_2 distortion measure. The use of DPCM provides direct control over the L_∞ distortion of the coded values that, as we will demonstrate, can be related to the L_∞ distortion of the surface itself. Transform coding, such as DCT or DWT based schemes, has also been considered. However, both control point and knot vector data often exhibit large discontinuities that represent significant features of the surface shape. A transform could introduce large distortions in the neighborhood of these locations, adversely affecting the quality of the decoded model. Additionally, it is often the case that the number of control points in each surface is rather low and some transforms would have trouble exploiting the redundancy in such cases. For these reasons, the choice of DPCM seems more appropriate than transforms.

Entropy coding uses the MQ adaptive arithmetic coder presented in Section 3.2.3 that provides very good compression efficiency while having relatively low computational complexity. The symbols coded by the arithmetic coder are generated by a statistical model that depends on the type of data (knot values, control point coordinates, weights, etc.). Multiple adaptive contexts are used to cater for the different statistics of each type of symbol. The MQ coder also provides a non-adaptive context with a uniform distribution that is convenient to code data of unknown distribution. Its use is roughly equivalent to send data along an uncompressed secondary bitstream, although without any of the synchronization issues that more than one bitstream implies.

4.4 Knot vectors

Although the number of knot values is usually much lower than that of control point coordinates and weights (e.g., the `gnom` model in Figure 4.3 has 612 knots and 6798 control point coordinates), it is also important to achieve good compression for them.

Since the values of the first and last knots are irrelevant, given a knot vector $U \equiv \{u_i\}$ of r elements, we consider only the reduced knot vector $\breve{U} \equiv U - \{u_0, u_{r-1}\}$. The knot vectors define the shape of the B-Spline functions. In addition, the multiplicity of the knots define the continuity

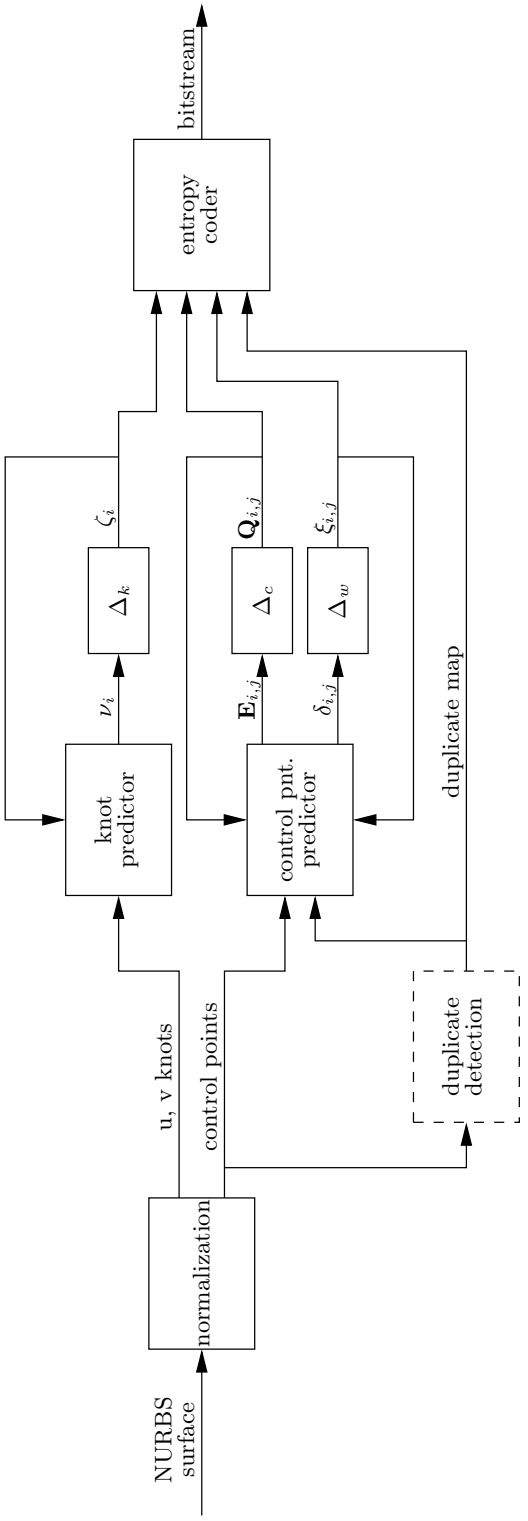


Figure 4.2: Simplified view of the coder structure.

of the surface. It is thus important to be able to preserve the continuity information in the coded representation. To this end we split the reduced knot vector $\check{U}$ into a *multiplicity map* $U^m \equiv \{u_i^m\}$ and a *break vector* $U' \equiv \{u'_i\}$. The latter is the vector of breakpoints (i.e., the unique knot values) while the former specifies the multiplicity associated with each breakpoint, minus 1. More formally

$$u_i^m = h_i - 1 \qquad i = 0, \ldots, r' - 1 \tag{4.1a}$$

$$u'_i = u_{i+\sum_{l=0}^{i} u_l^m} \qquad i = 0, \ldots, r' - 1 \tag{4.1b}$$

where h_i is the multiplicity of the breakpoint u'_i in the reduced knot vector $\check{U}$ and r' is the number of breakpoints in U', or equivalently the number of unique values in $\check{U}$. For knot vectors with no multiple knots the multiplicity map will be a sequence of zeroes and the break vector will be the same as the reduced knot vector.

4.4.1 Prediction and quantization

The break vector is a strictly increasing sequence of values in the $[0, 1]$ interval, since the knot vector is normalized. Most break vectors are uniform or close to it. One can thus expect the difference between consecutive breakpoints to remain constant, or close to constant. We therefore use a predictor that exploits this property: the predicted value $\tilde{u}'_i$ obeys

$$\tilde{u}'_i - \hat{u}'_{i-1} = \hat{u}'_{i-1} - \hat{u}'_{i-2}; \tag{4.2}$$

and thus

$$\tilde{u}'_i = 2\hat{u}'_{i-1} - \hat{u}'_{i-2}, \tag{4.3}$$

where $\hat{u}'_j$ is the previously decoded value of u'_j. The prediction error is therefore

$$\nu_i = u'_i - 2\hat{u}'_{i-1} + \hat{u}'_{i-2} \qquad i = 1, \ldots, r' - 2. \tag{4.4}$$

The first and last breakpoints are always implicit. Hence $\hat{u}'_0 = u'_0 = 0$ and $\hat{u}'_{r-1} = u'_{r'-1} = 1$. In order to compute v_0 we set $\hat{u}'_{-1} = 1/(r' - 1)$, which follows from the supposition of a quasi-uniform break vector and hence minimizes the magnitude of ν_0.

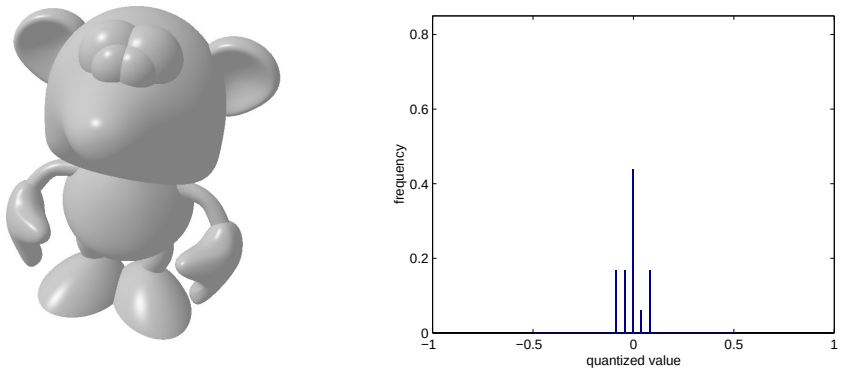
A uniform scalar quantizer is applied on the prediction error. Since we later entropy code the resulting quantization indices, it is not necessary to consider more elaborate scalar quantizers (see Section 3.3.2). Given the quantizer step size Δ_k, the quantization index is $\zeta_i = \langle \nu_i / \Delta_k \rangle$ and the reconstructed prediction error $\hat{\nu}_i = \zeta_i \Delta_k$, where $\langle \cdot \rangle$ denotes the rounding operator. The break value is thus decoded as

$$\hat{u}_i = \hat{\nu}_i + 2\hat{u}_{i-1} - \hat{u}'_{i-2}. \tag{4.5}$$

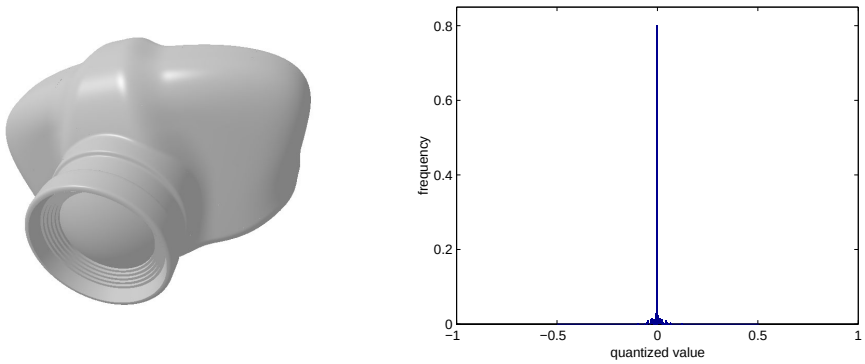
As for any DPCM coder with a uniform scalar quantizer, the coding error for breakpoints, and thus knots, is bounded by $\Delta_k/2$. That is, $|\hat{u}_i - u_i| \leq \Delta_k/2$. We therefore achieve a guaranteed maximum L_∞ distortion for the knot values of $\Delta_k/2$.

Figure 4.3 shows the histograms of the prediction error obtained for the non-uniform break vectors of three models. As it can be seen, the distribution is very skewed and thus effective entropy coding can be easily applied. Note that for uniform break vectors, the prediction error is identically zero.

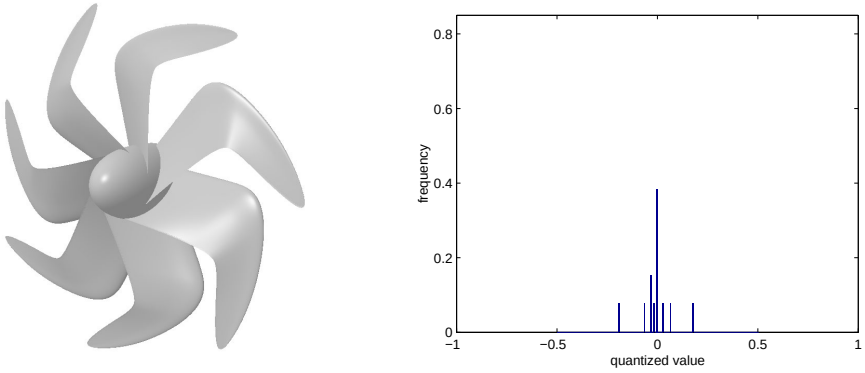
Although the predictor above is very effective it cannot, in general, guarantee that the decoded break vector will be a sequence of strictly increasing values. If the quantization error is large enough, it can happen that $\hat{u}'_{i-1} > \hat{u}'_i$, for some i. This leads to an illegal knot vector that cannot be used to define B-Spline functions. Even the case of equality is not desired, since it modifies the continuity properties of the B-Spline functions, and in some cases could also lead to illegal knot vectors (e.g., if



(a) `gnom`, 26 surface patches, 238 samples in histogram.



(b) `camera`, 32 surface patches, 343 samples in histogram.



(c) `subprop`, 10 surface patches, 91 samples in histogram.

Figure 4.3: Histograms of the breakpoint prediction error for the non-uniform break vectors of various models. The breakpoints of the uniform break vectors are not considered.

some internal knot has a resulting multiplicity larger than the B-Spline degree). An exclusive upper bound on Δ_k that guarantees proper coding is $\min\{u'_{i+1} - u'_i\}$. That implies $u'_i + \Delta_k < u'_{i+1}$ and

$$\hat{u}'_i \leq u'_i + \Delta_k/2 < u'_{i+1} - \Delta_k/2 \leq \hat{u}'_{i+1} \Rightarrow \hat{u}'_i < \hat{u}'_{i+1} \quad \text{for all } i.$$

This upper bound, however, is not tight and larger values of Δ_k rarely lead to illegal decoded knot vectors. In practice the encoder must check that all the decoded breakpoints form a strictly increasing sequence. This is, in principle, computationally simple since the decoder has to calculate the decoded values of each breakpoint for the DPCM predictor anyways. However, in the rare case that the constraint is violated it deems necessary the use of an iterative scheme to find a suitable value for Δ_k. As we will show later, the coding cost of knot vectors is rather small when compared to that of control points. Using a smaller than necessary Δ_k will therefore not affect the overall coding cost in a very significant way.

The use of the DPCM structure often requires an implementation in floating-point arithmetic, given that the input data is in that format. Nevertheless, an equivalent, integer only implementation can also be realized. In such an implementation, each breakpoint is quantized with a uniform scalar quantizer of step size Δ_k prior to any prediction. The predictor is then applied on the resulting quantization indices and no further quantization is required. Such an integer only implementation is computationally simpler. Nevertheless, when breakpoints are quantized care must be taken to ensure the constraint of a strictly increasing decoded break vector. We use, however, a floating-point implementation for uniformity with the predictors used for control point coordinates.

4.4.2 Distortion analysis

The use of DPCM with a uniform scalar quantizer guarantees that the L_∞ distortion on the knot values is no larger than $\Delta_k/2$. The distortion of interest is, however, that of the surface shape, which is of course related to that of knots. The relation between these two distortions is not trivial and, to our knowledge, has not been previously assessed. In Appendix A we develop bounds on the L_∞ distortion of a curve or surface given the quantization error of one breakpoint. It turns out that this distortion is proportional to the quantization error, the maximum distance between pairs of adjacent control points and the knot multiplicity, and inversely proportional to the minimum knot spacings. Detailed formulas are given in Appendix A, along with the demonstration. In the following we apply those bounds to derive the proportionality factor $\bar{D}$ between the knot and surface L_∞ distortions. Furthermore, we use this quantity $\bar{D}$ to set the knot quantization step size in a meaningful way.

Polynomial surfaces

Consider a non-rational surface $\mathbf{S}(u, v)$. Let $\psi(t) = t + \sum_{l=0}^{t} u_l^m$ be the function that maps a breakpoint index t to the index of the last knot corresponding to u'_t. Following the notation used in Appendix A, $D_{\psi(t)}$ is the maximum deviation induced on the curve by the quantization of breakpoint u'_t. Quantizing breakpoint u'_t affects the curve only on the interval $(u'_{t_{\text{inf}}}, u'_{t_{\text{sup}}})$, where t_{inf} and t_{sup} are the indexes of the breakpoints corresponding to knots $u_{\psi(t)-u_t^m-p}$ and $u_{\psi(t)-u_t^m+p}$, respectively. Therefore, on the interval $[u'_t, u'_{t+1})$ only the quantization of breakpoints $u'_{\ddot{t}+1}$ to $u'_{\dot{t}-1}$ affect the surface, where $\dot{t}$ and $\ddot{t}$ are such that $\dot{t}_{\text{sup}} = t$ and $\ddot{t}_{\text{inf}} = t + 1$. The total distortion on the interval $[u'_t, u'_{t+1})$ is thus

$$\sum_{l=\dot{t}+1}^{\ddot{t}-1} D_{\psi(l)}$$

and the total distortion on the overall curve is thus

$$D = \max_{0 \le t \le r'-2} \sum_{l=\dot{t}+1}^{\ddot{t}-1} D_{\psi(l)}.$$

Let $\epsilon \bar{D}_{\psi(l)}$ be the upper bound on $D_{\psi(l)}$ as calculated by one of the formulas in Appendix A. Hence

$$\bar{D} = \max_{0 \le t \le r'-2} \sum_{l=\dot{t}+1}^{\ddot{t}-1} \bar{D}_{\psi(l)}, \tag{4.6}$$

and $\epsilon \bar{D}$ is the upper bound on D.

Rational surfaces

The bounds given in Appendix A, and therefore Eq. (4.6), are only valid for polynomial surfaces. In the case of rational surfaces they only provide a distortion bound in projective space, where the surface is always polynomial, that we need to relate to Euclidean space. Given a distortion D^w in projective space for a surface with control points $\mathbf{P}_{i,j}$ and weights $w_{i,j}$ Tiller [108] provides the following bound for the Euclidean distortion D:

$$D \le \frac{1 + \max\|\mathbf{P}_{i,j}\|}{\min w_{i,j}} D^w.$$

This bound is however not satisfactory. It depends on the maximum norm of control points, which has no relation to the surface shape itself, since a surface can be arbitrarily translated. Furthermore, if all weights are equal (i.e., the surface is polynomial), it does not reduce to the obvious relation $D \le D^w / \min w_{i,j}$. Hereafter we develop a tighter bound that addresses these problems.

Consider two points $\mathbf{P}_1$ and $\mathbf{P}_2$ with weights w_1 and w_2 and a translation vector $\mathbf{T}$. Without loss of generality let $\mathbf{P}_1$ be the point with the smallest weight (i.e., $w_1 \le w_2$). The Euclidean distance between these two points can be bounded as

$$\begin{aligned}
\|\mathbf{P}_1 - \mathbf{P}_2\| &= \left\| \frac{w_1(\mathbf{P}_1 - \mathbf{T})}{w_1} - \frac{w_2(\mathbf{P}_2 - \mathbf{T})}{w_2} \right\| \\
&= \left\| \frac{w_1(\mathbf{P}_1 - \mathbf{T}) - w_2(\mathbf{P}_2 - \mathbf{T})}{w_1} + \left(\frac{w_2 - w_1}{w_1}\right)(\mathbf{P}_2 - \mathbf{T}) \right\| \\
&\le \frac{\|w_1(\mathbf{P}_1 - \mathbf{T}) - w_2(\mathbf{P}_2 - \mathbf{T})\| + |w_2 - w_1| \, \|\mathbf{P}_2 - \mathbf{T}\|}{w_1}.
\end{aligned}$$

The left term of the numerator just above is the distance between the projective points corresponding to $\mathbf{P}_1$ and $\mathbf{P}_2$ translated by $-\mathbf{T}$, where only the first three coordinates are considered (i.e., the homogenizing coordinate is disregarded). The term $|w_2 - w_1|$ is the difference between the homogenizing coordinate of these points. By the convex hull property this bound can be applied to the distortion of rational surfaces, where $\mathbf{P}_1$ and $\mathbf{P}_2$ would be pairs of corresponding control points. The Euclidean distortion D is hence bounded as

$$D \le \frac{D^A_{\mathbf{T}} + D_w \max\|\mathbf{P}_{i,j} - \mathbf{T}\|}{\min w_{i,j}}, \tag{4.7}$$

where $D^A_{\mathbf{T}}$ is the distortion on the first three homogeneous coordinates of the surface translated by $-\mathbf{T}$ and D_w is the distortion of the homogenizing coordinate. The translation vector $\mathbf{T}$ is arbitrary and can be chosen so as to minimize $\max\|\mathbf{P}_{i,j} + \mathbf{T}\|$ and obtain the tightest possible bound. A simple choice which is often close to optimal is the center of the bounding box of control points. Note that

the above bound reduces to the obvious relation $D \leq D^w / \min w_{i,j}$ when applied to polynomial surfaces, since $D^w = D^A_{\mathbf{T}}$ and $D_w = 0$. Note also that this bound reduces to the one of Tiller by using the relations $D^A_{\mathbf{T}} \leq D^w$ and $D_w \leq D^w$.

Therefore, the general expression for $\bar{D}$ that holds for rational surfaces becomes

$$\bar{D} = \frac{\bar{D}^A_{\mathbf{T}} + \bar{D}_w \max\|\mathbf{P}_{i,j} - \mathbf{T}\|}{\min w_{i,j}},$$

where $\bar{D}^A_{\mathbf{T}}$ is calculated by applying Eq. (4.6) to the first three homogeneous coordinates of the surface translated by $-\mathbf{T}$ and $\bar{D}_w$ by applying Eq. (4.6) to the homogenizing coordinate.

Knot quantization step size

Figure 4.4 shows the ratio between the estimated surface distortion to the actual surface parametric distortion. The distortion is estimated as $\bar{D}$ times the L_∞ distortion of knot values. By parametric distortion we mean the distortion $\max\|\mathbf{S}(u,v) - \tilde{\mathbf{S}}(u,v)\|$, that is the maximum deviation of a point on the surface corresponding to a parametric position. Note that, as we will see below, this can differ from the surface distortion as measured by the Hausdorff distance (see Section 3.3.4). As we can see, the distortion is consistently overestimated, by a factor between 3 and 17. This discrepancy comes from the fact that $\bar{D}$ is not a really tight bound and that we use the L_∞ distortion of knots, instead of the per breakpoint error. Nevertheless, the estimate is fairly close to reality.

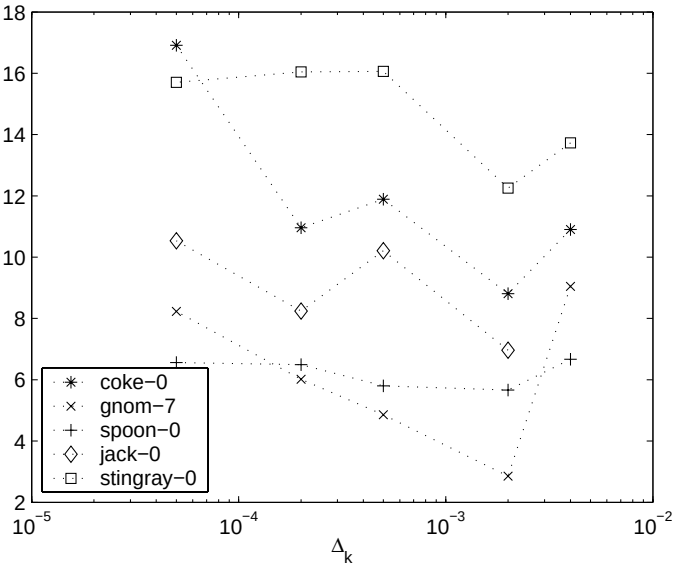


Figure 4.4: Ratio of the surface distortion bound to the actual surface L_∞ parametric distortion, as a function of the knot quantization step size Δ_k, for various surfaces.

Figure 4.5 shows the ratio between the parametric and Hausdorff L_∞ distortions. As it can be seen, the parametric distortion is, in general, between one and ten times larger than the Hausdorff one. This difference can be explained by the fact that the Hausdorff distortion measures the deviation of the surface in the normal direction, while the parametric distortion often occurs in the tangential direction. This is particularly true for the `stingray-0` surface, which is a mostly flat shape. Which distortion measure is appropriate depends on the application. For applications that require the preservation of the parametrization of a curve the parametric distortion is most appropriate. This

is also the case when the surface includes trimming curves. These curves are defined in parametric space and therefore any parametric distortion will affect the placement of the trims on the surface. If, however, the surface does not include any trimming loop the less stringent Hausdorff distance is more appropriate. The definition of a distortion bound for the Hausdorff distance remains, however, an unsolved problem.

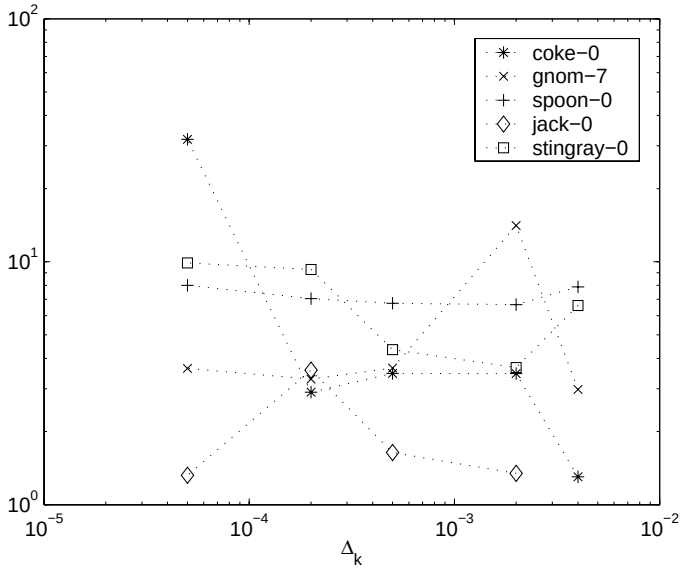


Figure 4.5: Ratio of the parametric to Hausdorff L_∞ distortions for various models and knot quantization step sizes.

Based on the above we will fix the knot quantization step size from $\bar{D}$ and assuming $\Delta_k/2$ as the L_∞ distortion of knot values. The individual contributions of the U and V knot vectors to the overall surface distortion is difficult to determine a priori. Except for the cases of uniform break vectors, in which case the contribution of the knot vector is identically zero, we will assign half the distortion to U and half to V. Let $\Delta'_k/2$ be the maximum allowable L_∞ surface distortion incurred by knot value quantization. We fix the quantization step size as

$$\Delta_k^\alpha \leq \frac{\Delta'_k}{2\bar{D}^\alpha}, \tag{4.8}$$

where Δ_k^α and $\bar{D}^\alpha$ are Δ_k and $\bar{D}$ for U or V, as appropriate. The value Δ'_k can be interpreted as the Euclidean $\mathbb{E}^3$ space equivalent of the quantization step size Δ_k in parametric space. In the cases where either U or V has a uniform break vector, the quantization step size used is the double of that above, since the uniform break vector induces no distortion on the surface.

In our implementation we fix Δ'_k globally for an entire model and apply Eq. (4.8) to obtain a knot quantizer step size for each knot vector of each surface. The $\bar{D}_{\psi(l)}$ values are calculated using the tightest possible bounds as given in Appendix A for the knot multiplicity of breakpoint u'_l (or v'_l).

4.4.3 Entropy coding

As previously mentioned, we entropy code the quantization indices ν_i and multiplicity values u_i^m with the MQ arithmetic coder. In addition to these it is also necessary to code the degree and

knot vector length to have a complete characterization of the B-Spline functions. In the following paragraphs we explain how each of these entities is entropy coded.

Degree, length and type

In a given model with several NURBS surfaces, only a few different B-Spline degrees are used. Most typically only quadratic and cubic forms are used, although higher and lower degrees are occasionally employed. For a B-Spline of degree p we code the n-bit word $p-1$ with a full 2^n-ary model. Using such a model assigns one independent probability in the binary arithmetic coder to each possible value and is thus equivalent to 2^n-ary arithmetic coding. This is shown in Algorithm 4.1. The n-bit word is coded from most significant to least significant bit and the context index for each bit is uniquely determined by its position and the value of the previously coded ones. For an n-bit word 2^n-1 binary contexts are required. In our implementation we set $n=3$, hence covering up to a maximum degree of 8, which is more than is required by VRML's base profile [113] and what is allowed in OpenGL [119]. Entropy coding the degree has the advantage of allowing arbitrary orders while ensuring that no more than 1 bit is required in average in the typical case of only quadratic and cubic B-Splines.

Algorithm 4.1: Code value x with a full 2^n-ary model and a binary arithmetic coder

```
c ← 0
for i ← n − 1 to 0 do
  b ← ⌊x/2^i⌋ − 2⌊x/2^(i+1)⌋ (i.e., ith bit of x)
  Code bit b with context c.
  if b = 0 then
    c ← c + 1
  else
    c ← c + 2^i
  end if
end for
```

The minimum length of a knot vector is always twice the B-Spline function order (i.e., degree plus one), since an order $p+1$ NURBS requires at least $p+1$ control points to be defined. In addition, the knot vector length for Bézier surfaces expressed in NURBS form is always twice the order. Therefore, for a knot vector of length r and a B-Spline of degree p, we code the value $\bar{r} = r - 2(p+1)$. Given that knot vectors can be arbitrarily long, a full model as used for degree coding is not applicable as that would require too many contexts and no meaningful symbol distribution could be derived. Instead, we first signal a coding length group j and then code $\bar{r}$ as an m_j-bit word, each bit being coded with the uniform context of the MQ coder (i.e., uncompressed). The coding group index j is coded using a full 2^n-ary model. The first coding group is reserved exclusively for Bézier knot vectors, for which $\bar{r}=0$, by setting $m_0=0$. Therefore, no extra bits are required for $\bar{r}$ for such knot vectors. In our implementation we use four coding groups, requiring a 2^2-ary model for their index, and we set $m_1=5$, $m_2=10$ and $m_3=15$. The maximum length is thus 32768, more than twice what is required by VRML's base profile. The special handling of Bézier knot vectors allows to efficiently code models made of Bézier surfaces, incurring a coding cost for the knot vector length that approaches zero with the number of surfaces.

The clamped and uniform knot vector types are signaled with two flags. Although not strictly necessary, these flags reduce the overall coding cost and complexity. In the common case of clamped knot vectors the first and last values of the multiplicity map equal the degree p and need not be

coded. For uniform knot vectors, the multiplicity map and prediction error are identically zero and need not be coded either. In the case of non-uniform knot vectors we still signal uniform break vectors with a third flag, in which case only the multiplicity map need be coded. Each of these three flags is coded with its own context of the MQ coder.

Multiplicity map

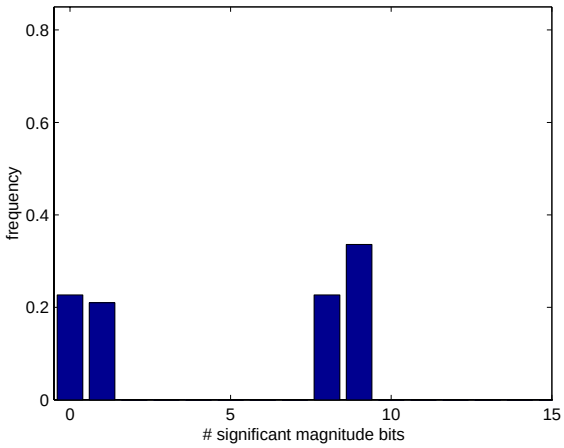
In the case where the knot vector is non-uniform its multiplicity map needs to be coded. We use a model similar to that used in JPEG [77] for DPCM values, which is itself a modified version of the method devised by Langdon [63]. It is shown in Algorithm 4.2. The basic idea is to choose the statistics with which to code the magnitude bits of a value based on its $\log_2$ bin. First the value is signaled as zero or non-zero with the MZERO context. If non-zero the position of its most significant bit (i.e., the $\log_2$ bin) is signaled using contexts MEXP through MEXP+n. Finally, the magnitude bits of the remaining value are coded, if necessary, with a context dependent on the $\log_2$ bin. This modified version needs to code one symbol less for power of 2 values, when compared to Langdon's since $x-1$ is coded instead of x if the value is non-zero. Note that for any knot the multiplicity value u_i^m is never larger than p. For a maximum coded degree of 8, only 6 contexts are required: MZERO, MEXP to MEXP+2, MMAG and MMAG+1.

Algorithm 4.2: Code multiplicity value x

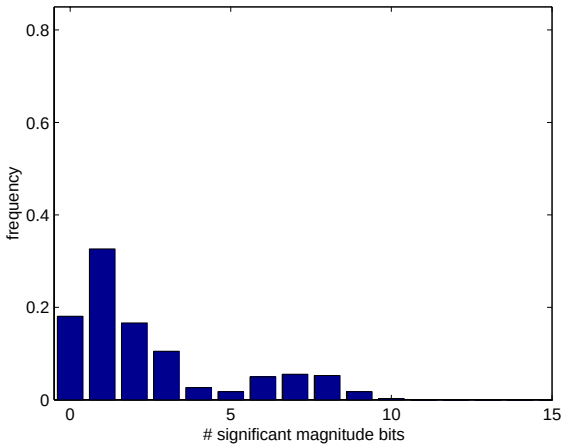
```
if x = 0 then
  Code a 1 with context MZERO
else
  Code a 0 with context MZERO
  n ← ⌈log2 x⌉
  for j ← 0 to n − 1 do
    Code a 1 with context MEXP+j
  end for
  Code a 0 with context MEXP+n
  if n ≥ 2 then
    Code bits n − 2 to 0 of x − 1 − 2^n with context MMAG+n − 2.
  end if
end if
```

Quantization indices

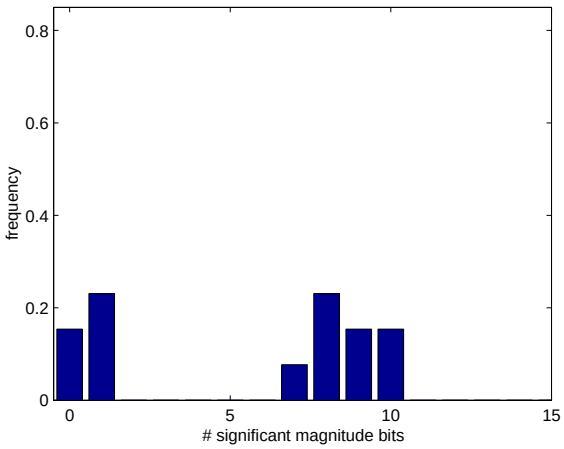
If a break vector is non-uniform (i.e., some quantization indices are non-zero) it is necessary to code the quantization indices. We use bitplane arithmetic coding for this. Given a global knot quantization step size of Δ'_k the quantizer used for the knot vector is $\Delta_k = \Delta'_k / 2^{\lceil \log_2(2\bar{D}) \rceil}$, following Eq. (4.8). The minimum number of bits necessary to code the magnitude of the quantization indices without overflow is $\lceil \log_2(1/\Delta_k + 1) \rceil$, since the range of possible prediction errors is $[-1, 1]$. We signal Δ'_k using exponent mantissa representation as $\Delta'_k = 2^{-\epsilon_k}(1+\mu_k/2^M)$, where $\epsilon_k \geq 1$, $0 \leq \mu_k \leq 2^M - 1$ and M is the number of bits used to signal the mantissa μ_k. The number of magnitude bits is thus $N_k = \epsilon_k + \lceil \log_2(2\bar{D}) \rceil + 1$. The quantity $\lceil \log_2(2\bar{D}) \rceil$ is coded for each surface as a fixed length binary number using the uniform context of the MQ coder. Figure 4.6 shows the histogram of the number of significant bits of $|\zeta_i|$, for the same models of Figure 4.3. As it can be seen the distribution is often bimodal.



(a) `gnom`, 238 samples, mean: 5.1 bits



(b) `camera`, 343 samples, mean: 2.5 bits



(c) `subprop`, 91 samples, mean: 5.5 bits

Figure 4.6: Histograms of the number of significant bits in the quantizer indices of the breakpoint prediction error for various models. Used $\Delta_k = 2 \times 10^{-4}$, hence $\epsilon_k = 13$ and $N_k = 14$. The proportion of non-zero values for (a), (b) and (c) is 77%, 82% and 85%, respectively.

We code $|\zeta_i|$ by bitplanes, from most significant to least significant. As suggested by Figure 4.6, $\max|\zeta_i|$ is usually smaller than $2^{N_k}-1$. Therefore, many of the most significant bitplanes will be identically zero. Let η_k be their number. Instead of explicitly coding all these zero bits separately, η_k is coded using a comma code (η_k 0s followed by one 1), with the dedicated KZBP context. In practice this is more cost efficient and less computationally expensive than explicit coding. For the remaining $N_k-\eta_k$ bitplanes the bits are coded one at time. For each $|\zeta_i|$, the most significant 0 bits up to and including the most significant 1 bit are coded with the KLZERO context. The remaining bits of $|\zeta_i|$ are coded with the KMREF context. Finally, for each non-zero ζ_i the sign is coded with the KSIGN context.

An alternative to the simple coding procedure above is to code the number of significant bits in each $|\zeta_i|$ value with a full 2^n-ary model, and then code the magnitude bits as above. This coding procedure has the potential to better exploit the bi-modality of the distributions shown in Figure 4.6. Figure 4.7 compares the coding rate obtained by these two coding procedures for a large collection of models and various quantization step sizes. The full 2^n-ary model achieves coding rates between 3% and 5% better, in average, than the simple bitplane coder. However, this improvement is below the standard deviation, which is between 6% and 7%. We consider therefore that this modest improvement is not worth the increase in complexity and will henceforth use the simple bitplane coder.

4.5 Control points

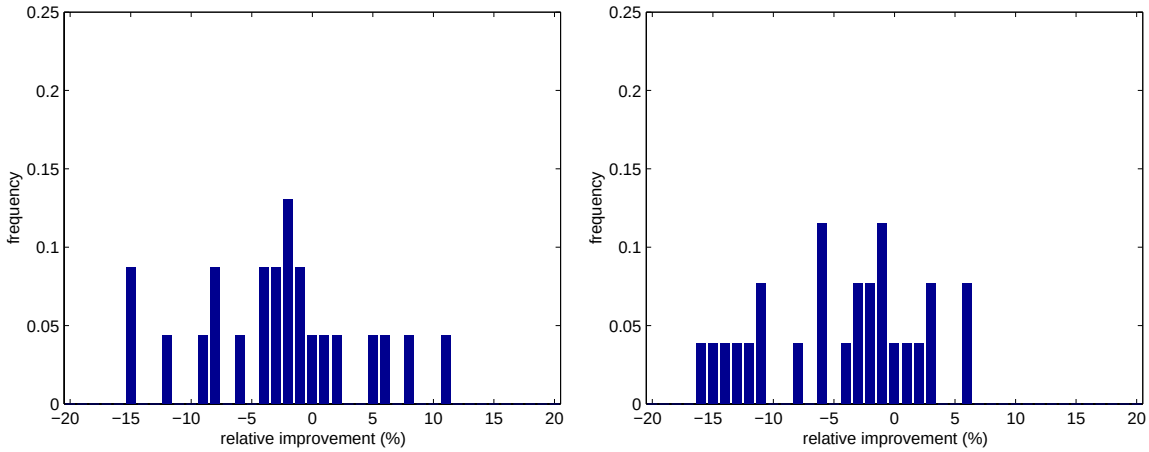
As previously mentioned the coding structure used for control points is similar to that used for knot values, namely DPCM followed by entropy coding. However, the number of control points being typically much larger than that of knots it is worth using a higher complexity predictor and entropy coder for control points.

NURBS control points can be represented either in projective space with homogeneous coordinates or in Euclidean space as affine coordinates plus an associated weight. Figure 4.8 shows the control net obtained by using the affine coordinates of control points (i.e., normal situation) and using the first three homogeneous coordinates as affine ones. This allows to compare the control net in affine space vs. the control net in projective space, by reducing the four dimensions of the latter to three. As it can be clearly seen the structure present in the affine coordinates is much higher than that present in the homogeneous ones. As a consequence, the redundancy in the control point data is easier to exploit on the affine coordinates. We shall thus code the control points using the affine coordinates plus the weight, instead of the homogeneous coordinates. The two forms are, however, equivalent and the entropy rate present in each is exactly the same. Following this separation of affine coordinates and weights, control points and their weights are coded independently, although in a very similar manner.

4.5.1 Prediction and quantization

Control point coordinates

Prior to prediction we need to offset the control points $\mathbf{P}_{i,j}$ to obtain a dynamic range centered on zero that is easier to handle. We use the center of the model's axis aligned bounding box as the offset for all surfaces of the model. Using the same offset for all surfaces has the advantage that coincident control points between different surfaces are still coincident after being offset. This is important as model designers often use coincident control points to ensure geometric continuity between surfaces, although this is not a sufficient nor necessary condition. In order to later characterize the maximum



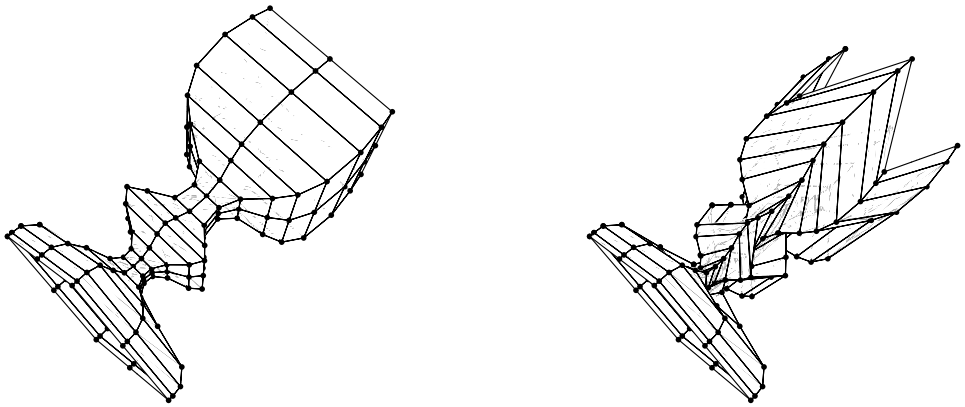
(a) $\Delta_k = 2 \times 10^{-3}$, mean = -2.8%, σ=6.7%

(b) $\Delta_k = 2 \times 10^{-4}$, mean = -4.4%, σ=6.5%

(c) $\Delta_k = 2 \times 10^{-5}$, mean = -3.5%, σ=7.4%

(d) $\Delta_k = 2 \times 10^{-6}$, mean = -2.6%, σ=5.7%

Figure 4.7: Histograms for the coding cost improvement of using a full 2^n-ary model instead of a simple bitplane coder for the breakpoint prediction error.



(a) affine coordinates **(b)** homogeneous coordinates

Figure 4.8: Comparison of the control net obtained using affine coordinates vs. the first three homogeneous coordinates for control points, for the `goblet` model.

magnitude of predicted control points we require to precisely know the dynamic range of offset control points. Let $\mathbf{O} = (x_O, y_O, z_O)$ be this center point. We code its coordinates using an exponent mantissa representation as $\hat{x}_O = 2^{\epsilon_{x,O}}(1+\mu_{x,O}/2^{M_O})$, and likewise for y_O and z_O, obtaining $\hat{\mathbf{O}} = (\hat{x}_O, \hat{y}_O, \hat{z}_O)$. Let 2^L be the maximum dynamic range of the control points of the entire model, across all three coordinates. That is, for $\mathbf{P}_{i,j} = (x_{i,j}, y_{i,j}, z_{i,j})$,

$$L = \log_2 \max\{(\max x_{i,j} - \min x_{i,j}), (\max y_{i,j} - \min y_{i,j}), (\max z_{i,j} - \min z_{i,j})\},$$

where the maximums and minimums are taken across all surfaces of the model. We signal this dynamic range simply as $\hat{L} = \lceil L \rceil$. Let $\mathbf{P}'_{i,j} = \mathbf{P}_{i,j} - \hat{\mathbf{O}}$ be the offset control points. The resulting maximum magnitude for any of the coordinates of offset control points is not larger than

$$2^{\hat{L}-1} + 2^{\epsilon_O - M_O},$$

since the error incurred by coding a coordinate of $\mathbf{O}$ is no larger than $2^{\epsilon_O - M_O}$, where ϵ_O is either $\epsilon_{x,O}$, $\epsilon_{y,O}$ or $\epsilon_{z,O}$, as appropriate.

Control points of each surface are predicted and quantized following a raster scan order. To allow for various kinds of statistics for the model data we use a general linear predictor as the DPCM predictor for the control point coordinates. A point $\mathbf{P}'_{i,j}$ is thus predicted as

$$\tilde{\mathbf{P}}'_{i,j} = \sum_{\substack{k \geq 0 \\ l>0 \text{ when } k=0}} \lambda_{k,l} \hat{\mathbf{P}}'_{i-k,j-l}, \tag{4.9}$$

where $\lambda_{k,l}$ are the predictor coefficients and the $\hat{\mathbf{P}}'_{i,j}$ are the previously decoded control points. The prediction error is thus

$$\mathbf{E}_{i,j} = \mathbf{P}'_{i,j} - \sum_{\substack{k \geq 0 \\ l>0 \text{ when } k=0}} \lambda_{k,l} \hat{\mathbf{P}}'_{i-k,j-l}.$$

As is the case for breakpoints there is no interest in using scalar quantizers other than uniform since we allow for entropy coding. Given a quantizer step size of Δ_c the quantization indices are thus simply $\mathbf{Q}_{i,j} = \langle \mathbf{E}_{i,j}/\Delta_c \rangle$. The decoded control points are trivially obtained as

$$\hat{\mathbf{P}}_{i,j} = \hat{\mathbf{E}}_{i,j} + \sum_{\substack{k \geq 0 \\ l>0 \text{ when } k=0}} \lambda_{k,l} \hat{\mathbf{P}}_{i-k,j-l},$$

where $\mathbf{E}_{i,j} = \mathbf{Q}_{i,j}\Delta_c$. At the top and left ends of the control net the "missing" control points $\hat{\mathbf{P}}'_{i,j}$, where $i < 0$ or $j < 0$, are set by mirroring any known control points across the control net borders. As a special case, for the top-left corner control point the prediction $\tilde{\mathbf{P}}'_{0,0}$ is set to zero, since no control points are known at that stage.

An effective, yet very simple, linear predictor is the parallelogram predictor of Touma and Gotsman [109] (see Section 3.4.4). In this case

$$\lambda_{k,l} = \begin{cases} 1 & k = 1, l = 0, \\ 1 & k = 0, l = 1, \\ -1 & k = 1, l = 1, \\ 0 & \text{otherwise.} \end{cases}$$

Figure 4.9 shows the histograms of the control point prediction error obtained with the parallelogram predictor on the models of Figures 4.3 and 4.10. The `killeroo-lowres` model is a NURBS fitting of a high-resolution 3D scan, while the others are CAD modelings. As it can be seen the distribution is very skewed and therefore linear predictors can provide a prediction error that is amenable to efficient yet simple entropy coding.

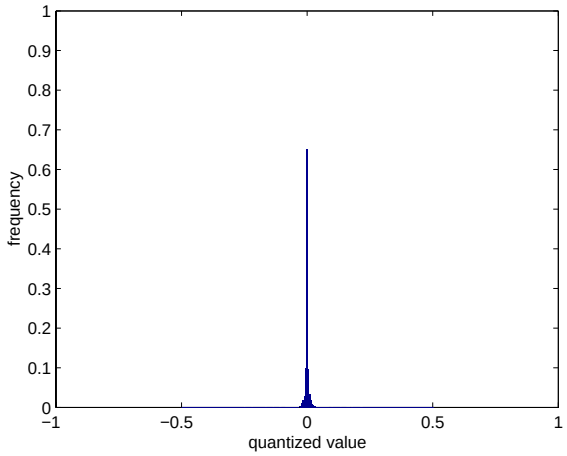
At the top row and leftmost column of the control net the parallelogram predictor reduces to a zero order predictor due to the lack of previous coded values. The prediction error can thus be often improved by the use of a 1D predictor that works across the top row or leftmost column. We therefore provide the choice between zero-, first- and second-order so called "edge" predictors, in addition to the use of the selected linear predictor used for all other control points. As we will later see, the first-order predictor often provides some added compression ratio, while the second order one is rarely of interest.

Quantization in a local basis

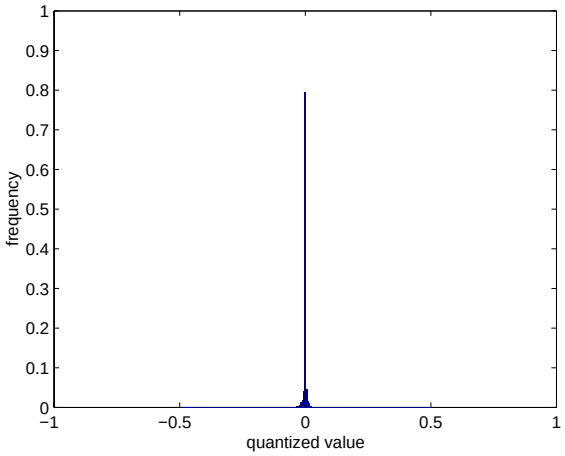
Inspired by the work of Khodakovsky et al. [56] (see Section 3.4.12) we propose a variant of the above DPCM scheme that uses a local coordinate system to quantize the prediction error at each control point, instead of the global coordinate system. Although independently developed, this concept has also been recently applied by Lee et al. [64] to single-rate coding of polygonal meshes. For a point $\mathbf{P}_{i,j}$ we define the local orthonormal basis $(\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3)$ as

$$\begin{aligned} \mathbf{e}_1 &= \frac{\hat{\mathbf{P}}_{i-1,j} + \hat{\mathbf{P}}_{i,j-1} - 2\hat{\mathbf{P}}_{i-1,j-1}}{\|\hat{\mathbf{P}}_{i-1,j} + \hat{\mathbf{P}}_{i,j-1} - 2\hat{\mathbf{P}}_{i-1,j-1}\|}, \\ \mathbf{e}_2 &= \frac{\mathbf{e}_3 \times \mathbf{e}_1}{\|\mathbf{e}_3 \times \mathbf{e}_1\|}, \\ \mathbf{e}_3 &= \frac{(\mathbf{P}_{i-1,j} - \mathbf{P}_{i-1,j-1}) \times (\mathbf{P}_{i,j-1} - \mathbf{P}_{i-1,j-1})}{\|(\mathbf{P}_{i-1,j} - \mathbf{P}_{i-1,j-1}) \times (\mathbf{P}_{i,j-1} - \mathbf{P}_{i-1,j-1})\|}. \end{aligned}$$

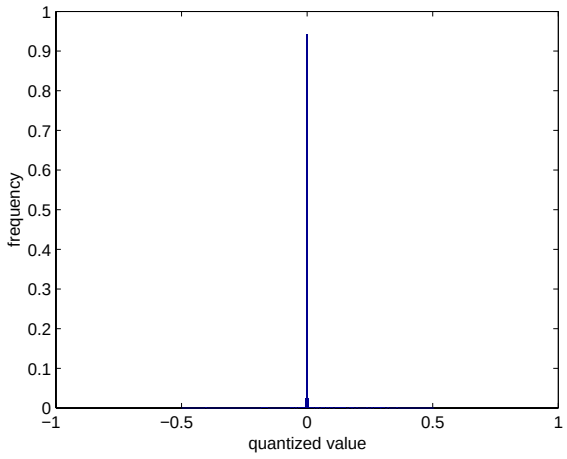
Therefore $\mathbf{e}_1$ and $\mathbf{e}_3$ are set to the directions of the main diagonal and normal, respectively, of the parallelogram formed by the parallelogram predictor. The other basis vector, $\mathbf{e}_2$, is set so as



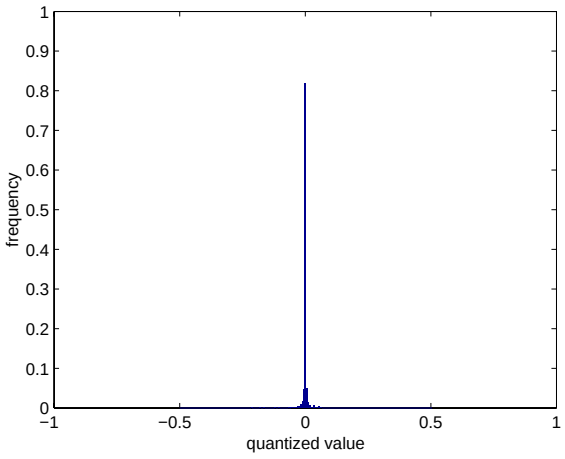
(a) `gnom`, 26 surfaces, 2266 control points.



(b) `camera`, 32 surfaces, 2642 control points.



(c) `killeroo-lowres`, 89 surfaces, 17181 control points.



(d) `scissors`, 7 surfaces, 1002 control points.

Figure 4.9: Histograms of the combined x, y and z prediction errors for the control points using the parallelogram predictor, for various models.

(a) killeroo-lowres

(b) scissors

(c) coke

Figure 4.10: The killeroo-lowres, scissors and coke models.

to obtain a right-handed orthonormal basis. This local basis is introduced in the DPCM loop by projecting the prediction error $\mathbf{E}_{i,j}$ into it and quantizing the resulting coordinates by Δ_c, and then performing the opposite precessing to obtain the dequantized error $\hat{\mathbf{E}}_{i,j}$. The quadrilateral formed by the three points above and the point to be coded $\mathbf{P}_{i,j}$ is expected to be close to planar. Thus, the magnitude of the prediction error along $\mathbf{e}_3$ should be less than that along $\mathbf{e}_1$ and $\mathbf{e}_2$, leading to more efficient entropy coding.

The use of this scheme involves some relatively complex operations to obtain $(\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3)$ at each control point and project the quantized error from the local to the global basis. At the encoder, it is in addition necessary to project from the global to the local basis, which involves the inversion of the matrix $[\mathbf{e}_1; \mathbf{e}_2; \mathbf{e}_3]$. These computations cannot be realistically realized in integer or fixed-point arithmetic, making the use of floating-point arithmetic a requirement. In fact, the use of the local basis places some rather stringent requirements on the numerical precision used at the decoder to implement these operations. Of particular concern are the cases where the parallelogram used to define the local basis is degenerate, or close to. This can occur if $\hat{\mathbf{P}}_{i-1,j} - \hat{\mathbf{P}}_{i,j}$ or $\hat{\mathbf{P}}_{i,j-1} - \hat{\mathbf{P}}_{i,j}$ have a length close to zero or are close to parallel. Nevertheless, such degenerate conditions are entirely legal and actually rather common in NURBS models. Since floating-point arithmetic is required, there is no way to ensure that a decoder is able to exactly reproduce the encoder computations, without any rounding error. Therefore, it is necessary to signal, for each control point, if the local basis is considered degenerate given the minimum precision requirements for a decoder. In such a case, the global basis is used as a fall-back. This signaling requires the coding of extra information, and therefore the gains brought by the local basis should be higher than the cost incurred by the handling of degenerate conditions. We defer the comparative analysis of the global and local bases to Section 4.8.

Weights

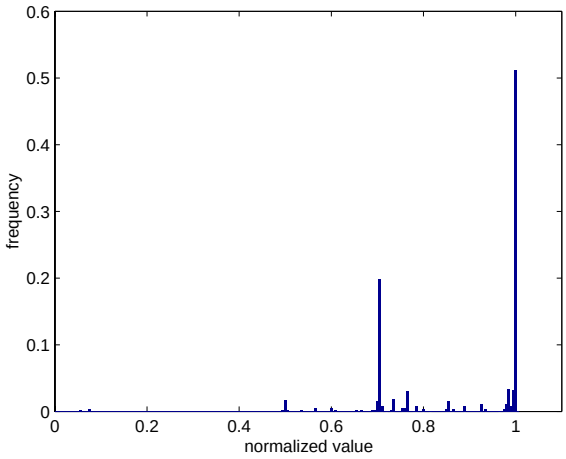
Figure 4.11 shows the histograms of the weight values for various models. As it can be seen, often only very few different weight values are used in a given model (e.g., `camera` and `scissors` models). This would suggest the use of a table lookup approach to coding the weights. On the other hand, such an approach can be complex and not perform well in the case of a large number of different weight values, as in the `coke` model, and therefore lacks flexibility. Furthermore, exploiting local correlation is more difficult with this technique. We take thus the same approach to coding weights as used for control point coordinates, with a linear predictor and uniform quantization later followed by entropy coding.

We have found that the 1D analogous of the parallelogram predictor performs well for weights. In fact, a weight is typically equal to its top or left neighbor in the control net and therefore more complex predictors are typically less effective. The prediction error for a weight $w_{i,j}$ is therefore

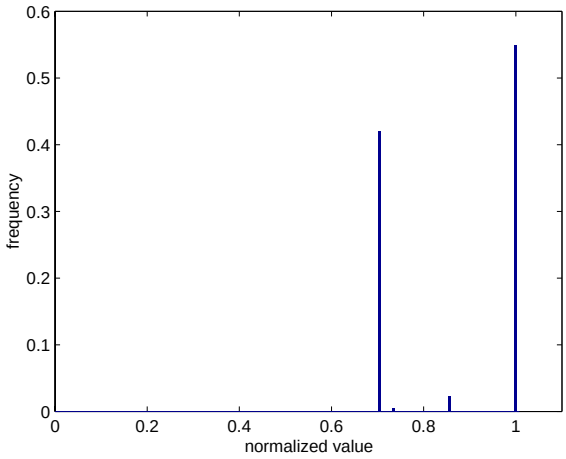
$$\delta_{i,j} = w_{i,j} - \tilde{w}_{i,j} = w_{i,j} - (\hat{w}_{i-1,j} + \hat{w}_{i,j-1} - \hat{w}_{i-1,j-1}),$$

where $\tilde{w}_{i,j}$ is the predicted value and $\hat{w}_{i,j}$ the previously decoded weights. Since the majority of weights equal one we set $\tilde{w}_{0,0} = 1$. Given a quantizer step size Δ_w the quantization indices are $\xi_{i,j} = \langle \delta_{i,j}/\Delta_w \rangle$ and the decoded values $\hat{w}_{i,j}$ are trivially derived as $\hat{w}_{i,j} = \tilde{w}_{i,j} + \hat{\delta}_{i,j}$, where $\hat{\delta}_{i,j} = \xi_{i,j}\Delta_w$.

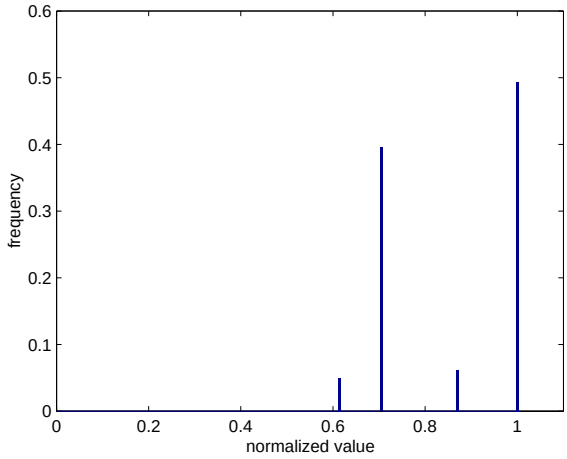
Figure 4.12 shows the weight prediction error histograms obtained for the models of Figure 4.11. Unfortunately, the histograms are not as skewed as those of control point coordinates. Nevertheless, the vary large proportion of zero values still allows for a rather efficient entropy coding.



(a) `coke`, 21 surfaces, 1601 weights

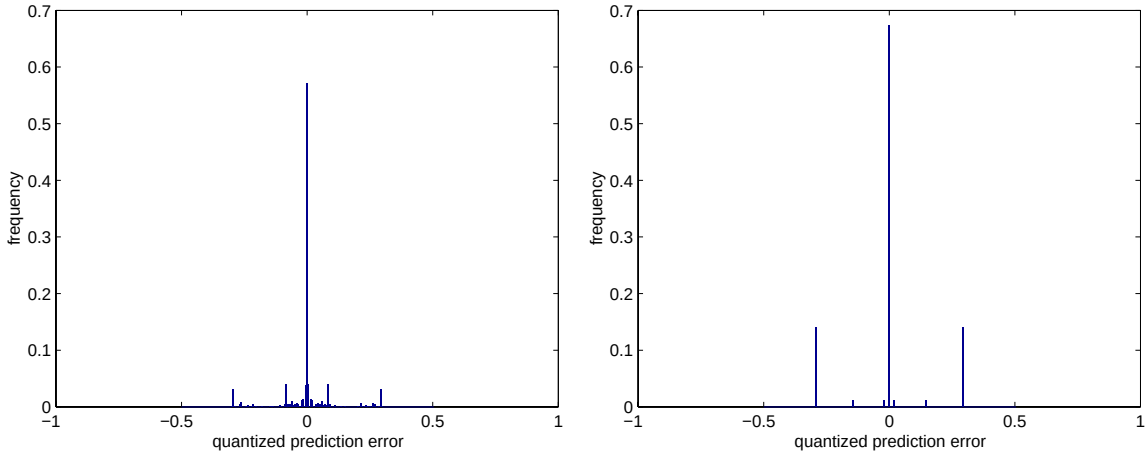


(b) `camera`, 32 surfaces, 342 weights



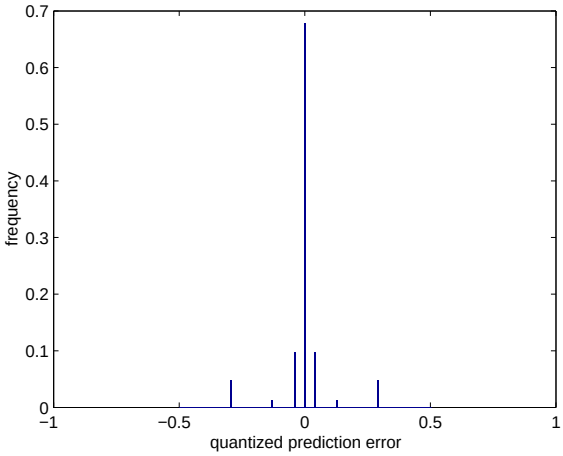
(c) `scissors`, 7 surfaces, 81 weights

Figure 4.11: Histograms of the weight values for various models.



(a) `coke`

(b) `camera`



(c) `scissors`

Figure 4.12: Histograms of weight prediction error for various models.

4.5.2 Distortion analysis

As we previously did for knots, we now look at the surface distortion induced by the quantization of control point coordinates and weights, and use it to derive meaningful values for the quantizer step sizes for each surface of a model. Let us first consider the distortion engendered by control point coordinate quantization alone. It is a well knot fact [see 80, chap. 11] that given quantization error $\bar{\mathbf{P}}_{i,j}$ for control point $\mathbf{P}_{i,j}$ the distorted surface is

$$\hat{\mathbf{S}}(u,v) = \sum R_{i,j}(u,v)\hat{\mathbf{P}}_{i,j} = \sum R_{i,j}(u,v)\mathbf{P}_{i,j} + \sum R_{i,j}(u,v)\bar{\mathbf{P}}_{i,j} = \mathbf{S}(u,v) + \sum R_{i,j}(u,v)\bar{\mathbf{P}}_{i,j},$$

where the $R_{i,j}(u,v)$ are the rational B-Spline functions (see Section 2.4.2). Hence, the distortion is a NURBS surface with identical knot vectors and weights and the quantization errors as control points. By the convex hull property the L_∞ distortion on the surface is limited by the L_∞ distortion on the affine coordinates of the control points, namely $\Delta_c/2$. When the entire model of an object is taken into consideration it does not make sense to provide varying coding accuracies for the various surfaces, since a single object should be represented with the same L_∞ distortion everywhere. We therefore use the same quantizer for all surfaces. We signal a global coordinate quantization step size Δ'_c relative to the object's bounding box approximate size $2^{\hat{L}}$ and set $\Delta_c = \Delta'_c 2^{\hat{L}}$. The L_∞ distortion on any of the affine coordinates of the object's surfaces is no larger than $\Delta_c/2$, while the norm of the Euclidean L_∞ distortion is $\Delta_c\sqrt{3}/2$.

The distortion analysis for weights is more involved as these are present in both the numerator and denominator of the rational B-Spline functions. Nevertheless, we can consider the distortion in the projective space and deduce the one in Euclidean space using the same bound that was derived for the knot case in Section 4.4.2. Consider a control point $\mathbf{P}_{i,j}$, its associated weight $w_{i,j}$, its quantized version $\hat{w}_{i,j}$, and a translation vector $\mathbf{T}$. The norm of the distortion on the first three homogeneous coordinates of the translated control point is

$$D^A_{\mathbf{T}} = \|w_{i,j}(\mathbf{P}_{i,j} - \mathbf{T}) - \hat{w}_{i,j}(\mathbf{P}_{i,j} - \mathbf{T})\| = |w_{i,j} - \hat{w}_{i,j}|\,\|\mathbf{P}_{i,j} - \mathbf{T}\| = D_w\|\mathbf{P}_{i,j} - \mathbf{T}\|.$$

Making use of Eq. (4.7) and the convex hull property, we find that the L_∞ distortion of the surface engendered by quantization of weights alone is not larger than

$$\frac{2\max\|\mathbf{P}_{i,j} - \mathbf{T}\|}{\min w_{i,j}} \max|w_{i,j} - \hat{w}_{i,j}| \leq \frac{\max\|\mathbf{P}_{i,j} - \mathbf{T}\|}{\min w_{i,j}}\Delta_w.$$

As for knots, finding the optimal value for $\mathbf{T}$ that minimizes $\max\|\mathbf{P}_{i,j} - \mathbf{T}\|$ is a difficult problem. Nevertheless, a good enough estimate for our purposes for the minimum of $\max\|\mathbf{P}_{i,j} - \mathbf{T}\|$ is half the diagonal length of the control points' bounding box of the surface being coded, which is easy to compute. As for control point coordinates, we desire the L_∞ distortion induced by weight quantization to be the same for all the surfaces of an object. Hence, we signal a global weight quantizer step size Δ'_w relative to the approximate bounding box size $2^{\hat{L}}$ and set the quantizer Δ_w individually for each surface, using the above bound.

4.5.3 Entropy coding

We entropy code the quantization indices using a bitplane coder, similar in spirit to that used for break vectors, that we explain in the following paragraphs. We distinguish non-rational and rational surfaces by coding a binary flag at the start of each surface. If it is non-rational no weights are coded, as all of them are unity. Also, prior to coding the quantizer indices we code the degenerate basis flag for each control point if the local basis is used to quantize the prediction errors. This flag is coded using its own arithmetic coder context.

In order to prevent overflow of the quantized indices we require to know the dynamic ranges of the coordinate and weight prediction errors. To ease the notation in what follows we define, for an arbitrary 3D point $\mathbf{P} = (x, y, z)$, the function

$$\Theta(\mathbf{P}) = \Theta(x, y, z) = \max\{|x|, |y|, |z|\}$$

that gives the maximum magnitude of the coordinates of $\mathbf{P}$. In Section 4.5.1 we established that the maximum magnitude of any coordinate after offsetting the control points is bounded as

$$\max \Theta(\mathbf{P}'_{i,j}) \leq 2^{\hat{L}-1} + 2^{\epsilon_O - M_O}, \tag{4.10}$$

where $2^{\hat{L}}$ is the approximated size of the model's bounding box and ϵ_O is the exponent of the coded offset for the relevant coordinate. Using the parallelogram predictor the bound on the predicted control points is

$$\max \Theta(\tilde{\mathbf{P}}_{i,j}) \leq 3 \max \Theta(\hat{\mathbf{P}}_{i,j}) \leq 3 \max \Theta(\mathbf{P}'_{i,j}) + 3\frac{\Delta_c}{2}$$

and therefore

$$\max \Theta(\mathbf{Q}_{i,j}) = \max\Big\langle \frac{\Theta(\mathbf{P}'_{i,j} - \tilde{\mathbf{P}}_{i,j})}{\Delta_c} \Big\rangle \leq \Big\langle 4\frac{\max \Theta(\mathbf{P}'_{i,j})}{\Delta_c} + \frac{3}{2} \Big\rangle \leq \Big\langle \frac{2^{\hat{L}+1} + 2^{\epsilon_O - M_O + 2}}{2^{\hat{L}-\epsilon_c}} + \frac{3}{2} \Big\rangle,$$

where ϵ_c is the exponent of the exponent-mantissa representation used to signal Δ'_c and therefore $\Delta_c \geq 2^{\hat{L}-\epsilon_c}$. We set M_O, the number of mantissa bits used to code the model offset $\hat{\mathbf{O}}$, so as to ensure that $\epsilon_O - M_O + 2 - \hat{L} \leq 0$. Hence we require that $M_O \geq \epsilon_O - \hat{L} + 2$. This means that we require the offset to be coded with sufficient precision relative to the model's bounding box size. With this constraint, and knowing that $\epsilon_c \geq 0$, the bound becomes

$$\max \Theta(\mathbf{Q}_{i,j}) \leq 3 \,.\, 2^{\epsilon_c} + 2.$$

Finally, the number of bits N_c used to code a quantization index must not be smaller than

$$\lceil \log_2(\max \Theta(\mathbf{Q}_{i,j}) + 1) \rceil \leq 3 \,.\, 2^{\epsilon_c} + 2 \leq 3 \,.\, 2^{\epsilon_c} + 2^{\epsilon_c} \leq 2^{\epsilon_c + 2}.$$

Therefore, the number of bits required to code the quantization indices without overflow is

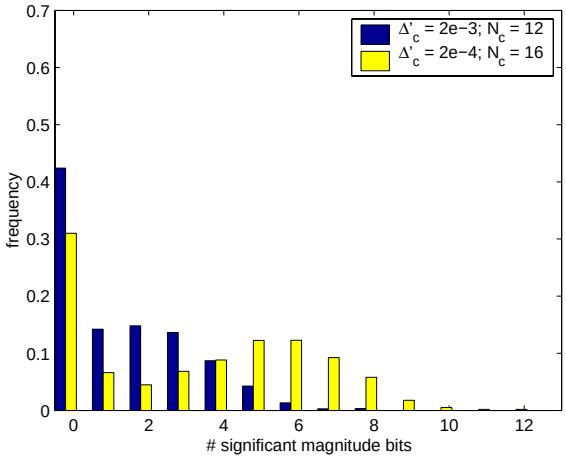
$$N_c = \epsilon_c + 2,$$

when the parallelogram predictor is used. In the above it is implied that we require $\epsilon_c \geq 2$, and thus $\Delta'_c < 1/2$, which is less than the minimum that any realistic compression setting would use.

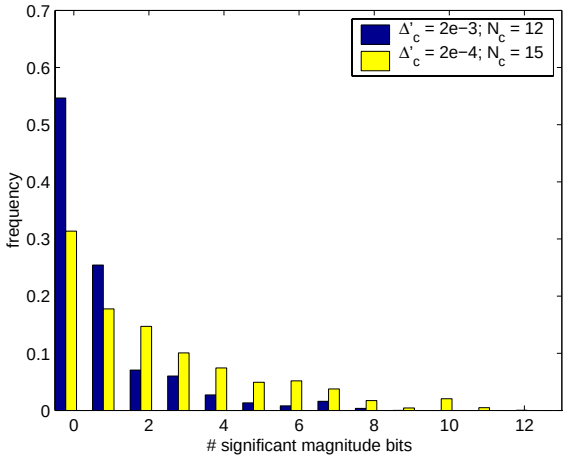
In the general case of an arbitrary linear predictor the number of bits required to avoid overflow will be determined by the sum of the absolute values of the predictor coefficients. Instead of deriving this quantity from them we choose to clip the predicted values to the range of the offset control points $\mathbf{P}'_{i,j}$, as given in Eq. (4.10). In fact, a predicted value outside the bounding box given by Eq. (4.10) does not make sense and can only hurt compression performance. Using this limiting scheme the number of required bits is reduced by 1, yielding $N_c = \epsilon_c + 1$. Independent of this, the use of a local basis for quantizing the prediction errors incurs an extra bit since the magnitude of a coordinate can be increased by up to $\sqrt{3}$.

The bitplane coding of coordinate quantizer indices proceeds in a way similar to that of breakpoints. Each of the x, y and z coordinates is coded in an identical way although using separate sets of contexts for the arithmetic coder, so that their potentially differing statistics can be properly exploited. We code the values from the most significant bitplane $(N_c - 1)$ down to the 0-th bitplane. Figure 4.13 shows the histograms of the number of significant bits of each value obtained on various

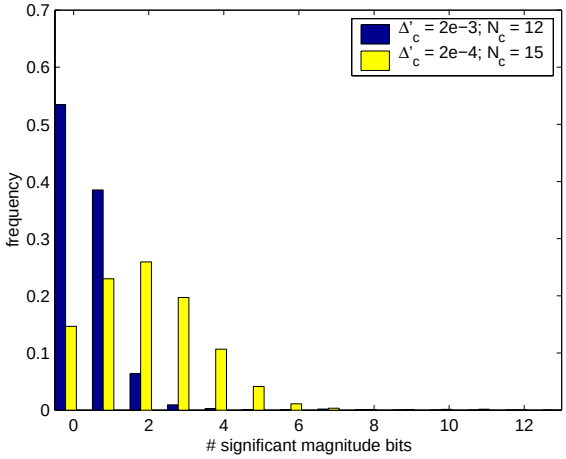
models and for two quantizer step sizes. As we can see, for not very fine quantization step sizes (e.g., 12 magnitude bits or less) the distribution is fairly skewed toward zero and bitplane coding should be very effective. For very fine quantizations (e.g., 16 or 15 magnitude bits) the distribution can sometimes be bi-modal or have its peak at a non-zero value. Given such a fine quantizer, the lower bitplanes will actually be noise-like and such a distribution is to be expected. In any case, the maximum number of significant magnitude bits is always considerably lower than N_c. Although not shown in the histograms, the number of significant magnitude bits along the top row and leftmost column of the control net, and in particular at the top-left corner, is usually much larger than at the other parts of the control net. This is due to the fact that the linear predictor is less efficient at these *border* locations, since there are less samples available on which to base the prediction.



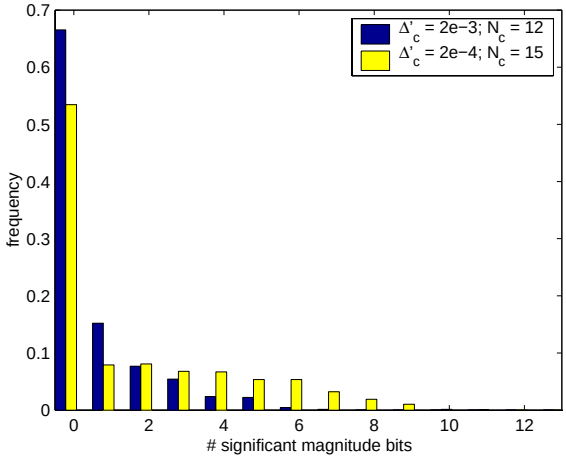
(a) `gnom`, means: 1.5 and 3.4 bits



(b) `camera`, means: 0.9 and 2.3 bits



(c) `killeroo-lowres`, means: 0.6 and 2.1 bits



(d) `scissors`, means: 0.7 and 1.8 bits

Figure 4.13: Histograms of the number of significant magnitude bits in the quantizer indices of the control point coordinate prediction error for various models and quantizer step sizes, using the parallelogram predictor. The quantizer step sizes are given with respect to the exact bounding box size (i.e., not power of two approximated).

As in the breakpoint case we first signal the number of leading all zero bitplanes. Let $\eta_{c,1}$ be the number of most significant bitplanes that are identically zero, except possibly for the top-left corner. Likewise, let $\eta_{c,2}$ be the number of additional such bitplanes when the top column and leftmost row are ignored. In general $\eta_{c,2} > \eta_{c,1}$. We arithmetically code both these quantities using a comma code but with separate contexts (i.e., $\eta_{c,1}$ 0s followed by one 1 with context CBZBP plus $\eta_{c,2}$ 0s followed by one 1 with context CZBP). The remaining uncoded bits are handled by bitplanes, one bit at a time. For each value the most significant 0 bits up to and including the most significant 1 bit are coded. The context depends on the position of the most significant 1 bit of the top and left neighbors not being below the current bitplane (i.e., the neighbors are *significant*). The four possibilities are the top neighbor only, left neighbor only, both neighbors or none. This rather simple context modeling allows to efficiently exploit most of the first order redundancy still present in the values. Inspired by the context modeling of JPEG 2000 [107] the remaining, *magnitude refinement*, bits are coded with one of three contexts. The first magnitude refinement bit of each value is coded with context CMREF1S if the top or left neighbors are significant at the current bitplane and CMREF1NS otherwise. The rest of the magnitude refinement bits are coded with a third context (CMREF2). This special handling of the first magnitude refinement bit allows to exploit the correlation with its neighbors that is usually present in it. Lower magnitude refinement bits do not, however, exhibit such a correlation and it is therefore not worth using more elaborate schemes to code them. Finally, for each non-zero value the sign is coded with the CSIGN context.

Due to differing statistics the top-left value as well as the top row and leftmost column values use a simplified set of contexts to code the most significant bits. The top-left value uses a context CTLLZERO while the other top row or leftmost column values use the CBLZERO context. Hence, no first-order modeling is employed at these locations.

The entropy coding of the quantizer indices of weights follows an identical scheme, using a separate set of contexts, with the only exception that the top-left value is handled as any other top row or leftmost column value. This is done since the predicted weight value at that location is typically accurate. The weights require, however, that we encode the per surface quantizer step size Δ_w. Instead of coding Δ_w for each surface we code the factor used to convert from the global step size Δ'_w. Let $2^{\bar{\epsilon}_w}$ be this factor, rounded to the next integer power of two. From Section 4.5.2

$$\bar{\epsilon}_w = \left\lceil \log_2\left(\frac{d_{\mathrm{BB}}}{\min w_{i,j}}\right) \right\rceil - \hat{L},$$

where the quantity d_{BB} is the diagonal length of the bounding box of the surface's control points, used as an approximation of $\max\|\mathbf{P}_{i,j} - \mathbf{T}\|$ (see Section 4.5.2). We code $\bar{\epsilon}_w$ as a signed fixed length integer with the uniform context of the MQ-coder.

The number of bits N_w required to handle the weight quantizer indices without overflow is easier to derive than for coordinates. Since we always work with normalized weights in the range $(0, 1]$ the range of the prediction error is $[-2, 2]$. The number of required bits is thus $N_w = \epsilon_w + \bar{\epsilon}_w + 2$. If like in the coordinate case we choose to limit the predicted value to be in the range $[0, 1]$ then one less bit is required.

Despite the different statistics of coordinates and weights we have chosen to use the same entropy coder for both of them. The motivations are that the coordinate entropy coder performs well for weights and that the complexity of the implementation is lowered by the use of the same entropy coder, in particular for hardware designs. The scheme requires a total of twelve arithmetic coder contexts per coordinate or weight, leading to a grand total of forty-eight for the entropy coding of control points.

4.6 Degenerate and closed surfaces

NURBS surfaces sometimes form closed shapes, such as cylinders. The common, although not unique, way of obtaining a closed surface is to use clamped knot vectors and make the first and last rows, or columns, of the control net identical. The control net of a closed surface will therefore contain a series of pairs of coincident control points. In a similar manner the modeling of some shapes that do not have tensor product like topology, such as a sphere octant, require multiple coincident control points as shown in one of the examples of Section 2.4.5. In such cases we say that the coincident control points form a *pole*, while in the case of closed surfaces we say that the pairs of coincident control points form a *seam*. Note that, in general, control points in a seam have only one duplicate, while control points at poles have two or more duplicates. In order to preserve these important characteristics of a surface it is necessary to ensure that control points that are coincident in the original model are still coincident in the decoded model. This can only be ensured if the coefficients of the linear predictor are integers. In fact, if a coefficient is not an integer it is probable that the prediction of a control point is not aligned to the quantization grid of a coincident control point, resulting in a different coded position in space, which in turn produces undesirable visual distortions (e.g., cracks). Furthermore, the prediction errors should all be quantized in the same coordinate system, preventing the use of the local basis (see Section 4.5.1). In short, poles and seams of a control net are only preserved if the control point predictor has only integer coefficients, of which the parallelogram predictor is a special case, and the local basis is not used. Besides these restrictions, linear predictors do not work well at poles and the repeated coding of identical control points usually leads to less than optimal compression performance. It is therefore desirable to provide means to efficiently and faithfully code duplicate control points, without placing restrictions on the prediction of control points.

The problem of coding duplicate relationships among control points is equivalent to the problem of handling, in a lossless manner, non-manifolds in polygonal mesh coding. In Section 3.4.11 we reviewed existing techniques, of which the one of Guéziec et al. [32] is the most efficient. A variation of their variable length coding scheme (see Section 3.4.11) could be easily derived and would probably be rather efficient. Nevertheless, we propose an alternative technique which is simpler to encode and models the seams and poles of control nets in a more efficient way, while at the same time being flexible.

First we establish a *duplicate map* for each control net. It lists the control points in raster scan order. For each control point that has duplicates we provide the list containing their indices, also in raster scan order. For control points that are already listed as duplicates of a previous control point no duplicate list is provided. Evidently, no control point coordinate or weight data needs to be coded for duplicate points. The duplicate map is entropy coded with the arithmetic coder, as follows. For each control point that is not a duplicate of a previous one we code a flag signaling if it has any duplicates, using one of four contexts. The one to use is determined by the top and left neighbors having or not any duplicates themselves. This simple context modeling is intended to capture the different statistics of seams and poles. If the current control point, whose position we denote as (i_0, j_0), has any duplicates its duplicate list follows. For each member of this list one bit is coded, with an independent context, telling if it is the last member of the list or not. If the list has more than one member we predict to be at a pole, otherwise we predict to be at a seam. Now we need to code the position of the duplicate in the control net. Let it be (i, j). The row index i is coded as the offset from the one of the previous duplicate in the list. Furthermore, the offset is predicted as the offset used for that previous duplicate, unless we are at the first duplicate of a pole, in which case it is predicted as zero. If the row offset is zero the column index j is coded as the offset from the column index of the previous duplicate, plus one. This offset is itself predicted

in the same way as for the row index. If the row offset is non-zero we code j as the offset from j_0 and the value used to predict future column offsets is set to zero. Note that in the cases where the row or column index is implicit (e.g., the previous duplicate was in the last row of the control net) the row or column index is omitted.

The prediction error e of each offset is coded as follows. One bit is coded, with its own context, signaling if it is zero. If non-zero we code $|e| - 1$ from its most significant to its least significant bit, where we derive the number of magnitude bits from the control net dimensions. The leading 0 bits and the most significant 1 bit are coded with a specific context, while the magnitude refinement bits are coded with the MQ-coder's uniform context. Finally, the sign of e is coded with its own context, if not implied from its maximum and minimum possible values. The context used to code the condition $e = 0$ is different for rows and columns. In addition two sets of these two contexts are used, one when coding a pole and another when coding a seam.

Table 4.1 shows the coding cost per control point incurred by the technique above for various models, and compares it to the cost of coding the coordinates and weights. As it can be seen, the duplicate map's cost is almost negligible when the proportion of control points having or being duplicates is low. When this proportion increases the cost increases as well but it never exceeds 1 bit per control point. The overall coding cost of control points is almost always decreased, often by a noticeable amount. In particular the `lion` model has many surfaces with poles and therefore benefits the most, reaching savings of almost 20%. Table 4.2 shows the same results but compares it to the coding cost of coordinates and weights for a rather large quantization step size (10^{-2} is roughly equivalent to a 7 bit quantization of the original data). As expected, the benefit of duplicate map coding decreases with larger quantization step sizes. Nevertheless, it still remains advantageous for several models, despite the large quantizer. Hence, the coding of the duplicate map is usually beneficial, even in cases where its use is not strictly required because a predictor with integer only coefficients is used and quantization is performed in the global coordinate system.

model	c.p. having/being duplicates (%)	dmap cost (bits/c.p.)	c.p. coord. and weight cost (bits/c.p.) w/o dmap	with dmap	change (%)
`coke`	8.8	0.15	9.01	8.76	−2.8
`subprop`	22.6	0.20	8.59	7.74	−9.9
`killeroo-lowres`	2.2	0.01	5.87	5.83	−0.7
`fairing`	1.1	0.08	6.31	6.34	+0.5
`scissors`	38.1	0.77	6.63	6.56	−1.1
`lion`	48.5	0.90	16.21	13.12	−19.1

Table 4.1: The duplicate map (dmap) coding cost and change in control point (c.p.) coding cost for $\Delta'_c = 2 \times 10^{-3}$ and $\Delta'_w = 2 \times 10^{-3}$, with the parallelogram predictor, for various models.

4.7 Trimmed surfaces

Trimmed NURBS surfaces, as explained in Section 2.4.5, are often employed in modeling, be it in CAD applications or otherwise. A generic system should therefore be able to handle them. In general, most systems describe the trimming curves as NURBS in the parametric space of the surface. In addition, piecewise linear curves are also usually accepted as a compact way to describe simple trims. In what follows we describe how do we code trimming curves. We restrict all trimming curves to be in NURBS form and we do not cater for piecewise linear curves. In fact, the latter can be trivially converted to order 2 NURBS with a clamped uniform knot vector and vice-versa.

model	c.p. having/being duplicates (%)	dmap cost (bits/c.p.)	c.p. coord. and weight cost (bits/c.p.) w/o dmap	with dmap	change (%)
`coke`	8.8	0.15	5.84	5.73	−1.9
`subprop`	22.6	0.20	5.00	4.64	−7.2
`killeroo-lowres`	2.2	0.01	3.01	3.00	−0.3
`fairing`	1.1	0.08	3.54	3.59	+1.4
`scissors`	38.1	0.77	3.91	4.20	+7.4
`lion`	48.5	0.90	10.45	8.74	−16.4

Table 4.2: The duplicate map (dmap) coding cost and change in control point (c.p.) coding cost for $\Delta'_c = 10^{-2}$ and $\Delta'_w = 10^{-2}$, with the parallelogram predictor, for various models.

(a) `fairing`, 15 surfaces, 734 control points.

(b) `lion`, 49 surfaces, 2142 control points.

Figure 4.14: The `fairing` and `lion` models.

Given that our encoding of these type of knot vectors is very efficient it is more compact to code all piecewise linear curves as NURBS.

Prior to coding the trimming curves it is necessary to signal the number of trimming loops of a surface and the number of curves in each of these. Most often a NURBS surface has either zero or one trimming loop, and rarely more. We therefore use one bit to signal the presence of trimming loops and, if there are any, another bit to signal if there is one trimming loop or more. Each of these bits is arithmetically coded with its own context. If there are more trimming loops, their number is coded as a fixed length integer with the MQ-coder's uniform context. The number of trimming curves in each loop is coded in a way analogous to the coding of knot vector lengths. We first code a group index describing the number of bits used to code the value and then code the binary word with the MQ-coder's uniform context. This allows to support a large number of trimming curves, while using only a few bits in the common case of just a few curves.

4.7.1 Knot vectors

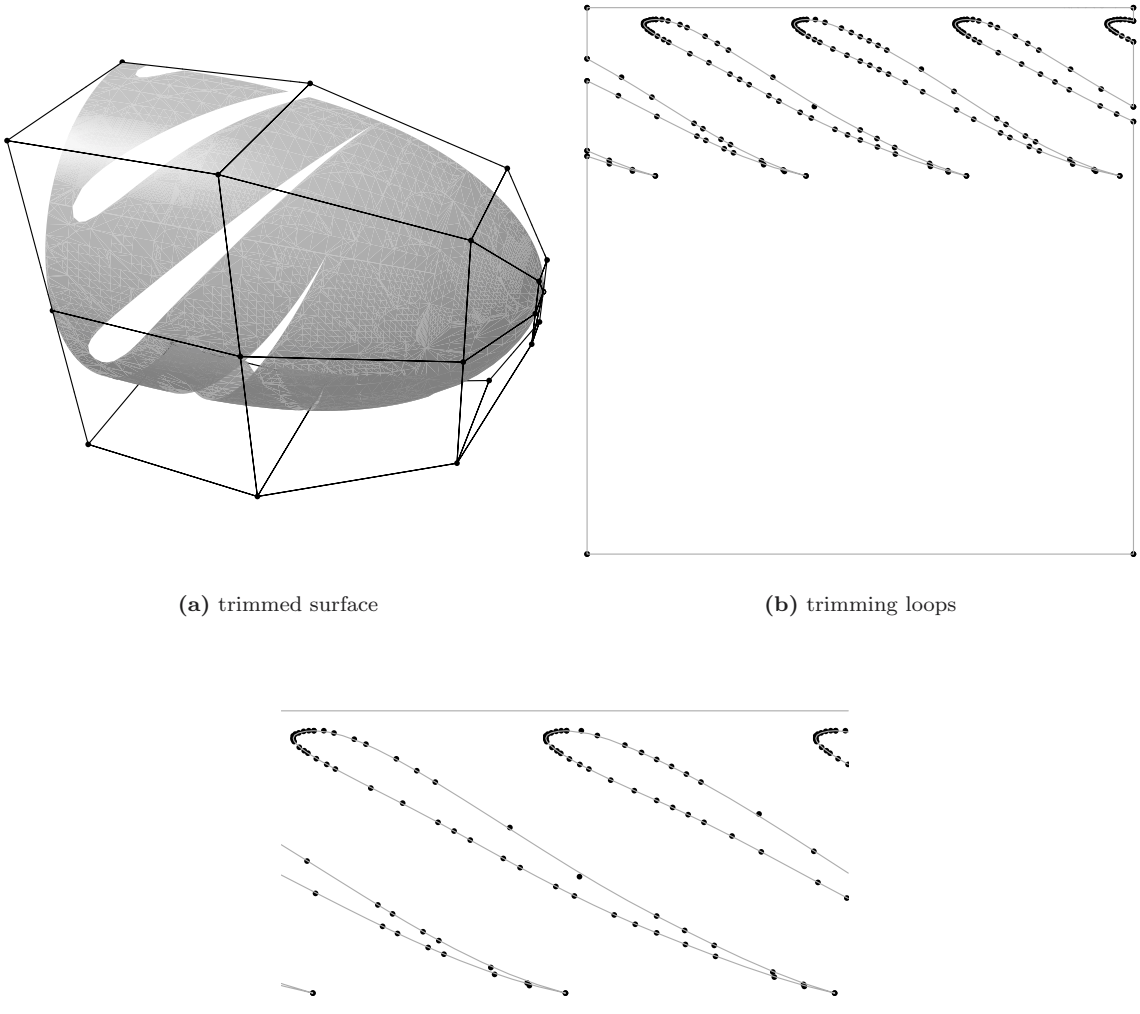
Knot vectors are coded in the same way as those of surfaces, as explained in Section 4.4. The only difference is that we require that all knot vectors be clamped. As the trimming curves in a loop must form a closed curve the end-point of a curve must coincide with the start-point of the subsequent one. In order to ensure this, design systems use clamped curves and make the last and first control point of adjacent curves coincident. This restriction allows us to avoid coding the end control points and automatically guarantees that the decoded trim curves form a closed curve. If, however, a non-clamped trim curve is encountered during the coding process, it can be converted to a clamped one without any geometric or parametric distortion. The sets of arithmetic coder contexts used to entropy code the trim curve knots is independent from that of surfaces to allow for differing statistics and thus better compression.

4.7.2 Control points

The control points are also coded analogously to surface control points. The major difference, however, is that the control points of the trim curve live in the 2-dimensional parametric (u, v) space of the surface. Furthermore, their coordinates are always in the range $[0, 1]$ since the knot vectors of the surface are normalized. As mentioned above, the affine coordinates of the last control point of each curve need not be coded. The weight, however, can differ between the last and first control points of two adjacent curves and therefore needs to be coded. In fact, the end weights of the control polygon of a clamped curve do not affect the end points of the curve itself. The predictor is a 1D linear predictor, since we deal with a control polygon instead of a control net. As particular cases we note the zero-order one, which predicts a control point as being equal to its predecessor, and the first-order one, which predicts constant differences between adjacent control points. The entropy coder is equivalent to that of surfaces, although with the 2D context modeling reduced to 1D. Figure 4.15 shows the trimming curves of a surface of the `subprop` model. As it can be seen, the curves are much more complex than the surface they trim, which is instead rather simple. This is often the case and leads to a coding cost for trimming curves much higher than for surfaces.

4.7.3 Trim distortion in 3D space

Trimming curves live in the (u, v) parametric space of the trimmed surface. Therefore, the distortion bounds that we have established in Sections 4.4.3 and 4.5.1 only relate the distortion of the trim knots or control points to the distortion in the parametric (u, v) space. A deviation of a trim curve



(a) trimmed surface

(b) trimming loops

(c) detail of 2nd trimming loop

Figure 4.15: A trimmed surface of the `subprop` model and its corresponding three trimming loops. The surface has 25 control points, while the trimming curves total 199 control points.

in the parametric space will have as effect the displacement of the surface's borders in 3D space, leading to a potentially visible distortion. Hence, it is important that we obtain bounds that relate parametric to Euclidean distortion. We achieve this by approximating the surface with a first order Taylor expansion. Given a parametric deviation (ρ_u, ρ_v) at a point (u, v) the surface deviation is

$$\mathbf{S}(u_0 + \rho_u, v_0 + \rho_v) - \mathbf{S}(u_0, v_0) \approx \frac{\partial}{\partial u}\mathbf{S}(u, v)\rho_u + \frac{\partial}{\partial_v}\mathbf{S}(u, v)\rho_v = \mathbf{S}_u(u, v)\rho_u + \mathbf{S}_v(u, v)\rho_v,$$

where we use the notation of Section 2.4. Its norm can be bounded as

$$\max\|\mathbf{S}_u\|\,|\rho_u| + \max\|\mathbf{S}_v\|\,|\rho_v|,$$

where we have dropped u and v for clarity. From the above it is clear that the same distortion on the u and v coordinates of trim control points can have very different effects on the resulting 3D distortion. Because of this, we use different quantizer step sizes to code the u and v coordinates of these points that we denote by $\Delta_{t,u}$ and $\Delta_{t,v}$, respectively. Hence, the 3D surface distortion due to control point coordinate quantization is bounded by

$$\max\|\mathbf{S}_u\|\frac{\Delta_{t,u}}{2} + \max\|\mathbf{S}_v\|\frac{\Delta_{t,v}}{2}$$

Instead of explicitly signaling $\Delta_{t,u}$ and $\Delta_{t,v}$ for each surface, we signal a global quantizer $\Delta'_{t,c}$ for the entire model, relative to its approximate bounding box size $2^{\hat{L}}$ and set

$$\Delta_{t,u} = \frac{\Delta'_{t,c}}{2^{\bar{\epsilon}_{t,u}}} \leq \frac{\Delta'_{t,c}2^{\hat{L}}}{2\max\|\mathbf{S}_u\|},$$
$$\Delta_{t,v} = \frac{\Delta'_{t,c}}{2^{\bar{\epsilon}_{t,v}}} \leq \frac{\Delta'_{t,c}2^{\hat{L}}}{2\max\|\mathbf{S}_v\|},$$

where $\bar{\epsilon}_{t,u} = \lceil \log_2 \max\|\mathbf{S}_u\| \rceil + 1 - \hat{L}$ and $\bar{\epsilon}_{t,v} = \lceil \log_2 \max\|\mathbf{S}_v\| \rceil + 1 - \hat{L}$. Note that we assign half the distortion to u and half to v, by lack of a better estimate of the interaction between the two. This guarantees that the L_∞ surface distortion due to the quantization of trim control point coordinates does not exceed $\Delta'_{t,c}/2$. The values $\bar{\epsilon}_{t,u}$ and $\bar{\epsilon}_{t,v}$ are coded for each surface as signed fixed-length integers through the MQ-coder's uniform context, if there are any trims.

For the weight quantization a slightly different bound is used, since given a weight distortion we know the norm of the distortion on the parametric plane but not the individual u and v distortions. If ρ is the deviation in parametric space then the deviation in 3D space is bounded by

$$(\max\|\mathbf{S}_u\| + \max\|\mathbf{S}_v\|)\rho.$$

We signal a global trim weight quantizer $\Delta'_{t,w}$ for all surfaces, relative to $2^{\hat{L}}$ and set the quantizer step size $\Delta_{t,w}$ for each trim curve as

$$\Delta_{t,w} = \frac{\Delta'_{t,w}}{2^{\bar{\epsilon}_{t,w} + \bar{\epsilon}_w}},$$

where $\bar{\epsilon}_{t,w} = \lceil \log_2(\max\|\mathbf{S}_u\| + \max\|\mathbf{S}_v\|) \rceil - \hat{L}$ and $\bar{\epsilon}_w$ is calculated for each trim curve as explained in Sections 4.5.2 and 4.5.3. Here $\bar{\epsilon}_{t,w}$ accounts for the relation between the parametric and 3D space distortions, while $\bar{\epsilon}_w$ accounts for the relation between projective and affine parametric space distortions. Thus, the L_∞ distortion due to trim curve weight quantization does not exceed $\Delta'_{t,w}/2$. The value $\bar{\epsilon}_{t,w}$ is coded for each surface in the same way as $\bar{\epsilon}_{t,u}$ and $\bar{\epsilon}_{t,v}$ above.

Finally, the quantizer step size is set in a similar fashion using the same bound as given above for weights. We signal a global trim knot quantizer $\Delta'_{t,k}$, relative to $2^{\hat{L}}$ as usual, and set the quantizer for the knot vector of each trim as

$$\Delta_{t,k} = \frac{\Delta'_{t,k}}{2^{\bar{\epsilon}_{t,w} + \lceil \log_2 \bar{D} \rceil}},$$

where $\bar{D}$ is calculated for each trim knot vector as explained in Section 4.4.2 and $\lceil \log_2 \bar{D} \rceil$ is coded as explained in Section 4.4.3. Here $\bar{D}$ accounts for the relation between the trim curve parametric and surface parametric distortions.

In order to compute the $\bar{\epsilon}_{t,u}$, $\bar{\epsilon}_{t,v}$ and $\bar{\epsilon}_{t,w}$ quantities we need to know the maximum norm of the partial derivatives $\mathbf{S}_u$ and $\mathbf{S}_v$ of the surface. If the surface is non-rational and clamped the partial derivatives are non-rational NURBS surfaces as well and are given by Eq. (2.11). Their maximum norm is thus straightforward to obtain using the convex hull property. If the surface is rational the partial derivatives are not NURBS surfaces and the same strategy cannot be applied. Nevertheless, Eq. (2.13) gives the derivative in terms of the non-rational surface $\mathbf{A}(u, v)$, obtained by considering only the first three homogeneous coordinates, and the 1D non-rational function $w(u, v)$, obtained by considering only the homogenizing coordinate. From Eq. (2.13) the maximum norm of the derivative is bounded as

$$\max\|\mathbf{S}_\alpha\| \leq \frac{\max\|\mathbf{A}_\alpha(u,v)\| + \max|w_\alpha(u,v)|\max\|\mathbf{S}(u,v)\|}{\min w(u,v)},$$

where α stands for the partial derivative with respect to u or v, as appropriate. Since $\mathbf{A}_\alpha(u, v)$ and $w_\alpha(u, v)$, as well as $\mathbf{S}(u, v)$, are all NURBS their maximum and minimum norm is computed by the convex hull property. Like in Section 4.4.2 a translation can be applied prior to computing $\mathbf{A}(u, v)$ so as to obtain a tighter bound. This does not affect the derivative, since it is translation invariant. Finally, if the surface is not clamped it can be converted to a clamped one, by using knot insertion, for the purpose of computing the maximum derivative. Since knot insertion does not modify the surface, either parametrically or geometrically, the derivative of the original and clamped surfaces are identical.

4.8 Performance analysis

In the previous sections we have described the coding technique and distortion trade-offs for all the constituents of a NURBS model: knot vectors, control points and weights and trimming curves. In this section we provide a detailed experimental analysis of the relation between global quantizer settings and the established distortion bounds as well as the resulting compression performance of the coding system. We consider only the parallelogram predictor, with the first order "edge" predictor, for surface control point coordinates. For trim curve control points we consider only the first order predictor. We postpone the analysis of different predictors till the next chapter.

4.8.1 Distortion measurement

In the following, the parametric L_∞ and L_2 distortions between the original and coded surfaces is measured by sampling both NURBS surfaces at a very large number (approx. 250'000 per model) of regularly spaced positions on the parametric domain and taking the Euclidean distance between the resulting pairs of points. The Hausdorff L_∞ and L_2 distortions, as defined in Section 3.3.4, are obtained by sampling one surface at regularly spaced positions on the parametric domain and searching, for each sample, the closest point on the other surface. Both, the forward and backward Hausdorff distortions are measured and the resulting symmetric measure is reported. The closest point problem is resolved using numerical methods as outlined in [80], namely a combination of Newton-Raphson root finding and Broyden-Fletcher-Goldfarb-Shanno (BGFS) and Nelder and Mead's Simplex minimization algorithms [26]. This combination provides a robust, yet reasonably fast, solution for the problem.

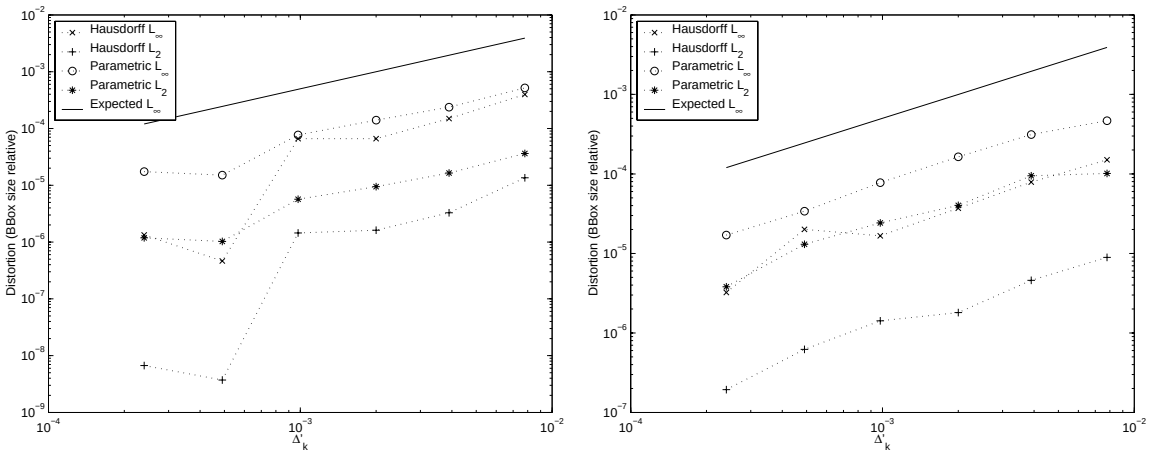
Unfortunately, the above strategy for measuring the parametric distortion cannot be straightforwardly applied to trimmed surfaces. The problem is that the parametric domain is no longer a

square and there is no one-to-one correspondence between positions in the original and coded parametric domains due to the distortion of the trimming curves that define this domain. Therefore, measuring the parametric distortions for trimmed surfaces is an ill-defined problem. We instead resort to only measuring the Hausdorff distortion. Furthermore, since the evaluation of the closest point problem is rather complicated for trimmed surfaces we resort to tessellating the models to a very large number of triangles (approx. 900'000) and measuring the Hausdorff distortion on the resulting polygonal models with the Mesh [3] tool. The tessellation process is performed using the OpenGL NURBS tessellator [119].

4.8.2 Quantizers and rate distortion

Surface knot vectors

In Section 4.4.2 we used the previously established knot distortion bounds to derive a per-surface quantization step size Δ_k from a global (i.e., per-model) step size Δ'_k that guarantees a surface L_∞ distortion not exceeding half this latter value. Figure 4.16 shows the resulting distortions for varying values of the global knot quantizer for two representative models with non-uniform knots. The parametric L_∞ distortion is around 10 times smaller than expected from the value of Δ'_k. This means that the influence of the knot quantization is consistently overestimated, which is in agreement with the observations in Section 4.4.2. The L_2 distortions are, however, very much smaller. The reason for this is twofold. First, on typical models a considerable number of surfaces have uniform break vectors, which are losslessly coded, and hence reduce the overall L_2 distortion. Second, the knot distortion bounds are worst case L_∞ estimations and a considerably smaller L_2 distortion is to be expected.

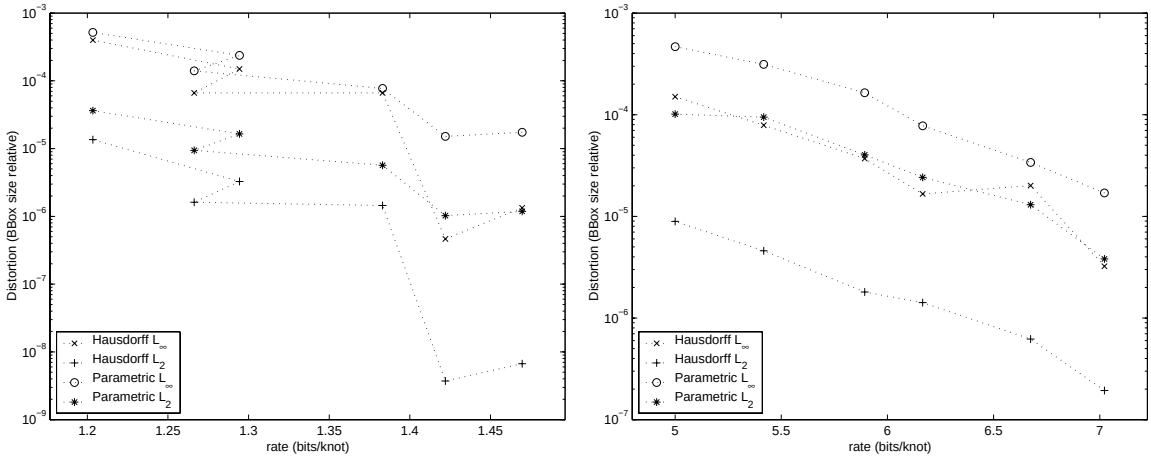


(a) `coke` **(b)** `scissors`

Figure 4.16: Distortion due to knot quantization for varying Δ'_k on various models. The quantizer step sizes are roughly equivalent to 7, 8, 9, 10, 11 and 12 bit quantizations.

Figure 4.17 shows the knot coding rate-distortion functions for the same models. The bitrate is reported as bits per knot. The knot coding includes the B-Spline degrees, knot vector lengths, multiplicity maps and breakpoint values. Surprisingly, the `coke` model has a rather non-smooth behavior: although reducing the quantization step size always reduces the distortion it does not

always increase the bitrate. Nevertheless, the bitrate differences are rather small. The probable cause of such unexpected behavior is the fact that DPCM is a non-linear process and some particular quantizer values yield easier to compress data. The general trend is, however, similar to that of the better behaved `scissors` model. By comparing the plots one can also observe that the knot coding cost for `scissors` is much higher than for `coke`. In fact, the proportion of uniform break vectors, which are much more efficiently coded, in the latter is much higher than in the former.



(a) `coke` **(b)** `scissors`

Figure 4.17: Knot rate-distortion for varying Δ'_k on various models. The quantizer step sizes are roughly equivalent to 7, 8, 9, 10, 11 and 12 bit quantizations.

Finally, Figure 4.18 shows the increase in the overall (i.e., including knots and control points) bitrate when the global knot quantizer step size is changed from $7.8 \times 10^{-3} \approx 2^{-7}$ to smaller values, all other things being kept equal. As it can be seen, the increase is often modest, although non-negligible. Nevertheless, models that have a large proportion of non-uniform break vectors (e.g., `gnom`, `goblet`, `scissors`, `stingray`) can incur a considerable increase when exceedingly small quantizers are used.

Summarizing, applications that must ensure a prescribed parametric L_∞ distortion can rather safely use a quantizer step size 10 to 20 times larger than the maximum allowed distortion. Furthermore, applications for which only the Hausdorff distortions are relevant can use much more aggressive quantization, with a step size often 150 times larger, or even more, than the allowed distortion. Note, however, that parametric distortion is relevant for trimmed surfaces, as previously noted. On the other hand, as shown in Section 4.8.5, the knot coding cost represents only a small proportion of the overall coding cost and it is therefore useless to use an excessively large knot quantizer step size.

Surface weights

As previously explained in Section 4.5.2, the per-surface weight quantizer step size Δ_w is derived from the global quantizer step size Δ'_w using the surface's bounding box diagonal length and minimum weight. Figure 4.19 shows the resulting distortions for varying values of the global weight quantizer. As for knots, the parametric L_∞ distortion is considerably smaller than what is predicted by the bound given in Section 4.5.2, which is based on that of Eq. (4.7). This behavior is not particular to

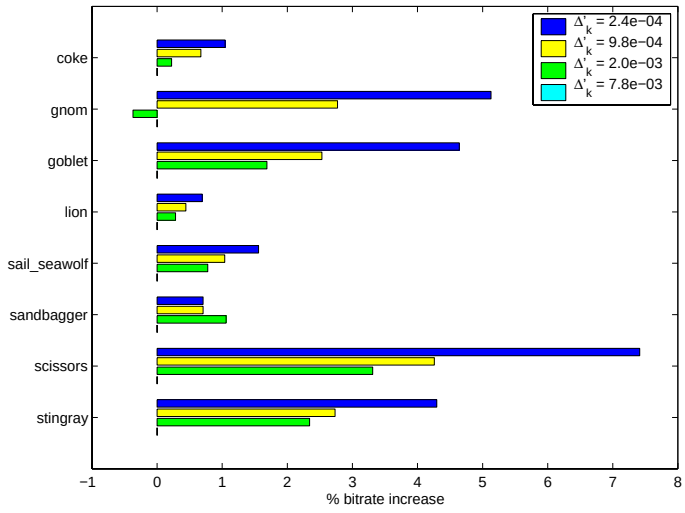


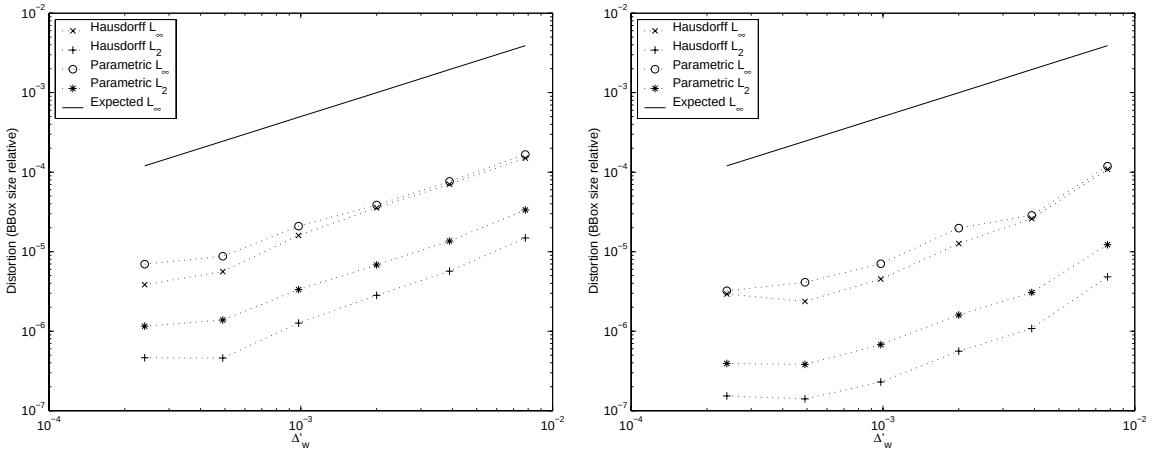
Figure 4.18: Overall bitrate increase arising by reducing the global knot quantizer step size from 7.8×10^{-3}, for various models and $\Delta'_c = \Delta'_w = 7.8 \times 10^{-3}$.

these models and has been observed on all the our test rational models. In fact, Eq. (4.7) usually provides a rather loose bound. Furthermore, in Section 4.5.2 we apply this bound collectively to all weights of a surface. A somewhat tighter bound can be found by applying Eq. (4.7) to each weight independently. For each weight $w_{i,j}$ only the bounding box and weights of the control points affecting the relevant knot spans for $w_{i,j}$ need to be considered, instead of all the surface control points. The overall surface bound would then be the maximum of these per-weight bounds. It is not clear, however, that this much more complex bound would be tighter, as the basic one of Eq. (4.7) is not itself tight. Figure 4.20 shows the corresponding weight rate-distortion curves, where the bitrate is reported as bits per weight of rational surfaces. Both models are well behaved. We can observe, however, that the weight coding cost is much higher for `coke` than for `scissors`. This large difference is easily explained by the histograms shown in Figure 4.12. In fact, `coke` has many more different weight values and its prediction error distribution is wider, leading to higher entropy and thus a reduced compression ratio. Summarizing the above results an application can, in general, use a global weight quantizer step size Δ'_w that is 20 or more times larger than the allowed L_∞ distortion. As shown in Section 4.8.5, the weight coding cost can sometimes be a considerable part of the overall coding cost (e.g., for `coke`) and choosing a suitable value of Δ'_w is important in achieving the highest rate-distortion performance.

Surface control points

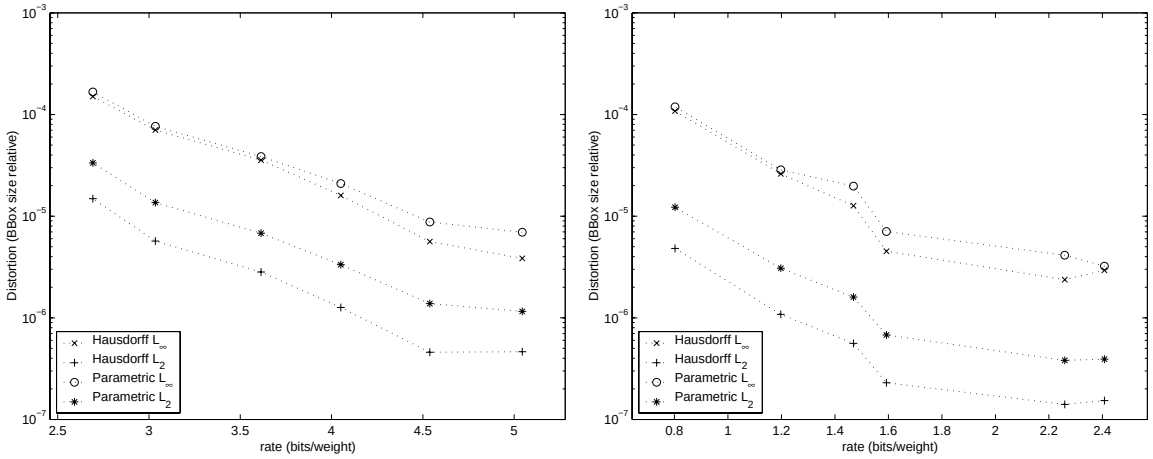
Figure 4.21 shows the overall (i.e., including the distortion induced by knot, weight and control point coordinate quantization) for varying quantizer step sizes. Given a quantizer step size Δ'_c the expected parametric L_∞ distortion is $\sqrt{3}\Delta'_c/2$ *. From the plots, it is clear that the parametric L_∞ distortion is very close to the expected distortion. Furthermore, the Hausdorff L_∞ distortion is almost identical to the parametric one. Taking into account the results on knot and weight induced distortion above, one can note that the largest contributor to the overall distortion is the quantization of control point coordinates. In addition one can note that the parametric and Hausdorff L_2 distortions are

*Given that we work in 3D and that each coordinate has an identical maximum error of $\Delta'_c/2$ the maximum resulting Euclidean error distance is $\sqrt{3}$ larger, or $\sqrt{3}\Delta'_c/2$.



(a) `coke` **(b)** `scissors`

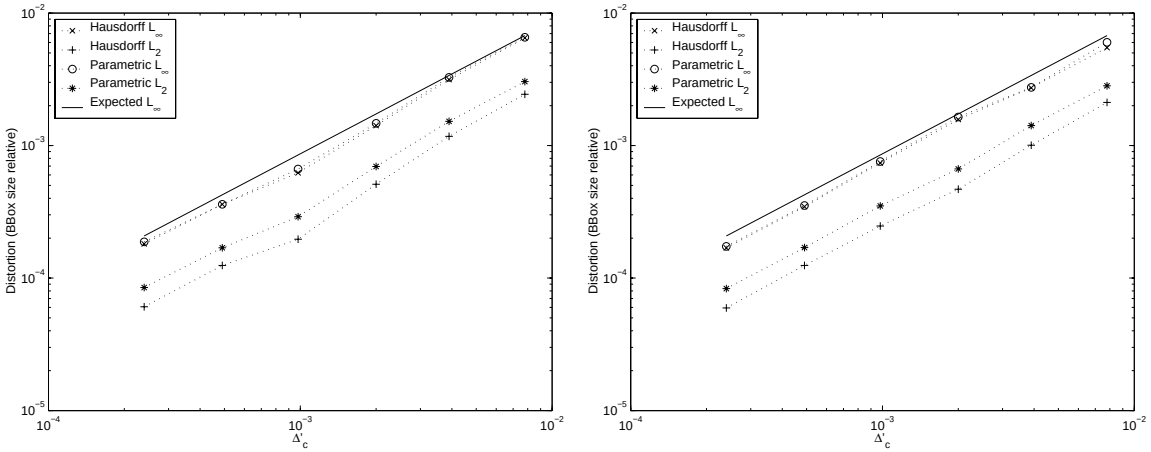
Figure 4.19: Distortion due to weight quantization for varying Δ'_w on various models. The quantizer step sizes are roughly equivalent to 7, 8, 9, 10, 11 and 12 bit quantizations.



(a) `coke` **(b)** `scissors`

Figure 4.20: Weight rate-distortion for varying Δ'_w on various models. The quantizer step sizes are roughly equivalent to 7, 8, 9, 10, 11 and 12 bit quantizations.

consistently 2 and 3 times smaller than their L_∞ counterparts, respectively, for all quantizer step sizes. This has also been observed on all the other test models.

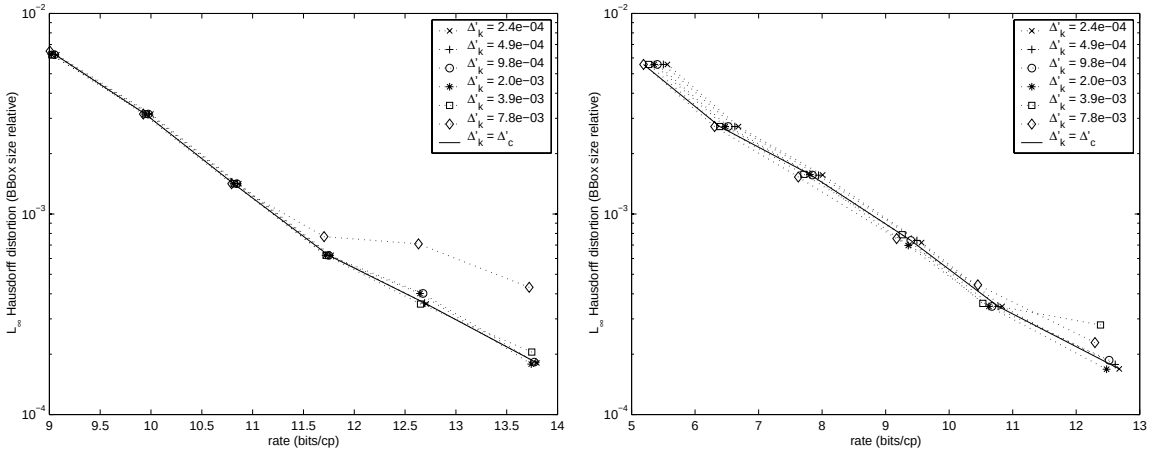


(a) `coke` **(b)** `scissors`

Figure 4.21: Overall model distortion for varying Δ'_c, with $\Delta'_k = \Delta'_c$ and $\Delta'_w = \Delta'_c$, for various models. The quantizer step sizes are roughly equivalent to 7, 8, 9, 10, 11 and 12 bit quantizations.

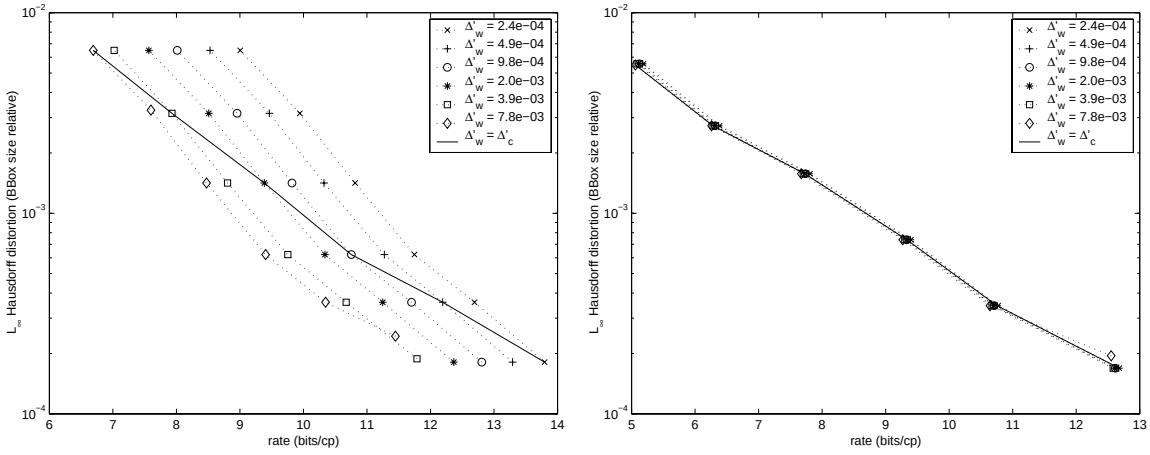
Figure 4.22 shows the corresponding Hausdorff L_∞ rate-distortion curves with a fixed weight quantizer step size and varying control point coordinate and knot quantizers, when using the parallelogram predictor. Worth noting is the fact that the knot quantizer has virtually no effect on the distortion and little effect on the bitrate. A notable exception is the use of large knot quantizers (e.g., $3.9 \times 10^{-3} \approx 2^{-8}$ and $7.8 \times 10^{-3} \approx 2^{-3}$) with small coordinate and weight quantizers (i.e., at high bitrates). As shown in Figure 4.16, in this situation the knot induced distortion becomes comparable to, or larger than, the control point induced distortion leading to a severe degradation of the compression performance. This phenomenon is, however, not observable on the Hausdorff L_2 distortion. Although not shown, the latter is approximately 3 times smaller than the L_∞ distortion for $\Delta'_k = \Delta'_c$ and thus is just a parallel plot. The analogous plots for various Δ'_w values are shown in Figure 4.23. In this case, however, the behavior of `coke` and `scissors` is considerably different. Although the distortion of both models is not affected by a change of Δ'_w, the bitrate of the latter is considerably affected, whereas that is not the case for the former. In the `coke` model all the surfaces are rational and its weight distribution does not lend itself to highly effective compression. On the other hand in `scissors` only one surface is rational, out of seven, representing only 8% of the control points, and its weight distribution is more easily compressed. Nevertheless, in both models the large quantizer $\Delta'_w = 7.8 \times 10^{-3}$ provides the best rate-distortion performance, at all rates. This conclusion also follows from the previous analysis of the weight-only distortion, where we stated that a weight quantizer up to 20 times larger than the desired global L_∞ distortion can be used. As for the varying knot quantizer case, the L_2 distortion shows the same behavior, hence the corresponding plots are omitted. Figure 4.24 shows the distortion and rate-distortion results obtained on the very large `killeroo-hires` model, which is non-rational and has uniform knot vectors only. As it can be seen, this model exhibits the same behavior although it shows much lower bitrates due to the high density of control points.

Visual results for the `scissors` model are shown in Figure 4.25 for two coordinate quantizer step



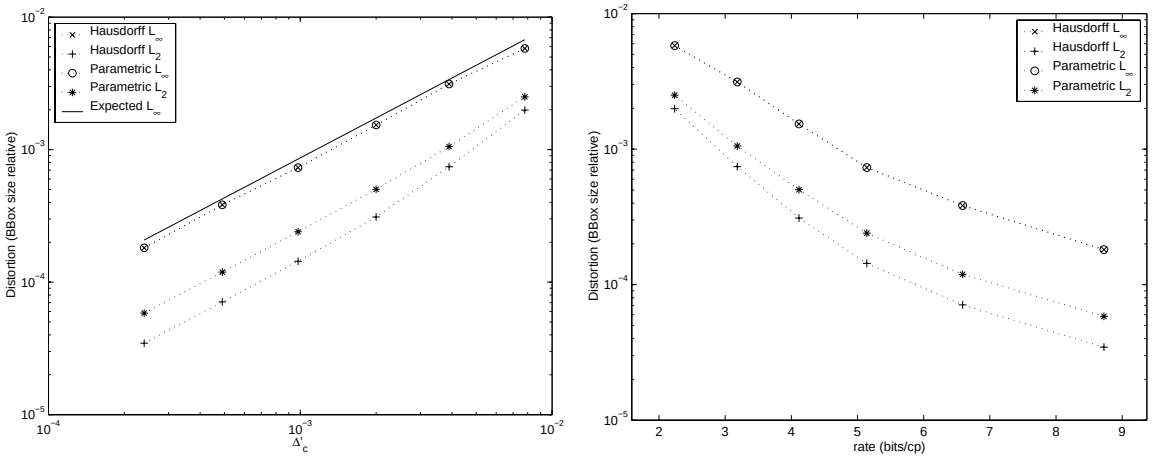
(a) `coke` **(b)** `scissors`

Figure 4.22: Overall rate-distortion for varying Δ'_c and various Δ'_k values, with $\Delta'_w = 7.8 \times 10^{-3}$, with the parallelogram predictor.



(a) `coke` **(b)** `scissors`

Figure 4.23: Overall rate-distortion for varying Δ'_c and various Δ'_w values, with $\Delta'_k = \Delta'_c$, with the parallelogram predictor.



(a) `killeroo-hires` **(b)** distortion (parallelogram predictor)

Figure 4.24: Overall distortion and rate-distortion for the highly detailed `killeroo-hires` model.

sizes, along with the original model. For $\Delta'_c = 3.9 \times 10^{-3}$ some artifacts are visible along the top of the blade, whereas no artifacts are visible for $\Delta'_c = 9.8 \times 10^{-4}$. Figure 4.26 shows a detailed view of the same coded models that exemplifies the so-called "crack" problem. In fact, the larger quantizer step size creates a sufficiently large deviation of the borders of two adjacent surfaces, creating a visible crack that does not exist in the original model. The crack is, however, imperceptible with the smaller quantizer. This crack problem is the reason why the L_∞ distortion is a relevant distortion measure, since it provides an upper bound on the width of the crack. Finally, Figure 4.27 shows the distribution on the model of the Hausdorff distortion along with the corresponding error histogram, as shown by the Mesh program [3]. As it can be seen the error distribution is biased toward zero (i.e., mostly blue and green color shades), while the maximum distortion is very localized (i.e., no visible surface points are shown in red). This demonstrates that the resulting coded surfaces behave nicely, as long as the targeted L_∞ distortion is small enough to prevents cracks.

Trimmed surfaces

In addition to the three quantizer step sizes, Δ'_k, Δ'_c and Δ'_w, analyzed above three extra quantizer step sizes determine the rate-distortion performance for models with trimmed surfaces: $\Delta'_{t,k}$ for trim curve knots and $\Delta'_{t,c}$ and $\Delta'_{t,w}$ for trim curve control point coordinates and weights in parametric space, respectively. As explained in Section 4.8.1, we resort to tessellation of the trimmed models in order to measure the distortion. Unfortunately, the OpenGL NURBS tessellator is not very accurate and sometimes introduces a measurement error comparable to the magnitude of the distortion being measured, hence yielding somewhat unreliable results. As a consequence, the results presented in this paragraph are not very accurate for small distortions (i.e., in the order of 10^{-4}).

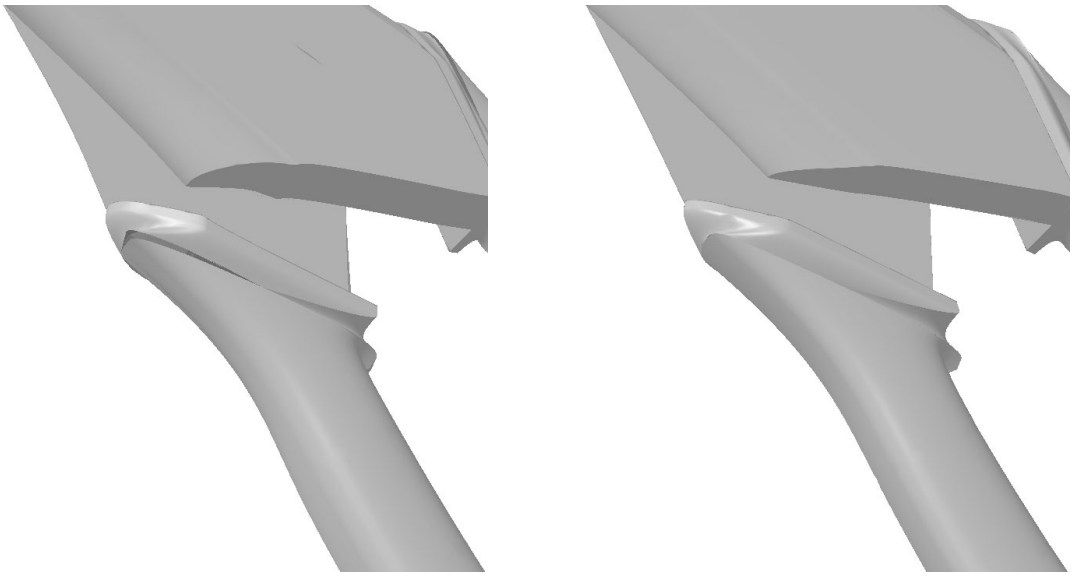
Figure 4.28 shows the distortion induced by the trim curve coding as the control point coordinate and weight quantizer step sizes are varied. We only report the L_∞ distortion as that is the only surface distortion that is relevant for trimming curves. The L_2 surface distortion is not relevant, as it is measured on the entire surface and the coding of the trim curves only affects the borders of the trimmed surface. The corresponding rate-distortion curves are show in Figure 4.29, where the rate is reported as bits per trim curve control point. We can observe that the actual distortion is below the

(a) original

(b) $\Delta'_c = 3.9 \times 10^{-3}$ and $\Delta'_k = \Delta' w = 7.8 \times 10^{-3}$

(c) $\Delta'_c = 9.8 \times 10^{-4}$ and $\Delta'_k = \Delta' w = 7.8 \times 10^{-3}$

Figure 4.25: Visual results for the `scissors` model.



(a) $\Delta'_c = 3.9 \times 10^{-3}$ and $\Delta'_k = \Delta' w = 7.8 \times 10^{-3}$ **(b)** $\Delta'_c = 9.8 \times 10^{-4}$ and $\Delta'_k = \Delta' w = 7.8 \times 10^{-3}$

Figure 4.26: Detail of visual results for the `scissors` model of Figure 4.25.

expected one. The somewhat irregular behavior at small quantizer step sizes is suspected to be an artifact of the OpenGL NURBS tessellator, as explained above. The corresponding rate-distortion curves are shown in Figure 4.29. It can be observed that the cost of trim curve coding is very high, consuming in excess of 15 bits per control point. The reasons for such a high cost, when compared to the surface coding cost, are many. Often, the trimming curves are the product of applying Boolean set operators and are obtained through numerical approximation algorithms. As such, the trimming curves can present a large number of irregularly spaced knots and control points, which are difficult to predict. Furthermore, surfaces are often sensitive to trim curve placement and therefore the conversion from the specified Euclidean space global quantizers to the parametric space quantizers increases the number of magnitude bits to be coded by a considerable amount. This increase is, however, necessary to ensure that the prescribed maximum distortion is not exceeded. In addition, typical models use a very large number of small trimming curves per trimming loop, instead of a few long curves. The cost of coding these independent curves is non-negligible. Nevertheless, we should point out that a large quantity of information is embedded in the trimming curves, as obtaining the same shape without trimming would require a very large number of control points. In this sense, it is to be expected that the coding cost is rather high. Notwithstanding, in the next chapter we propose several encoder strategies to diminish this cost with negligible or no distortion.

Figure 4.30 shows the overall distortion when all the surface and trim curve quantizers are set equal and varied. Figure 4.31 shows the corresponding rate-distortion curves, where the rate is expressed as bits per control point. As it can be seen, the resulting Hausdorff L_∞ distortion is always slightly inferior than the expected, from the quantizer step sizes, distortion. The exception being the `subprop` model for very small quantizer step sizes, where we measure little reduction in distortion as the quantizer step size is decreased. Again, we suspect this to be an artifact of the OpenGL NURBS tessellator.

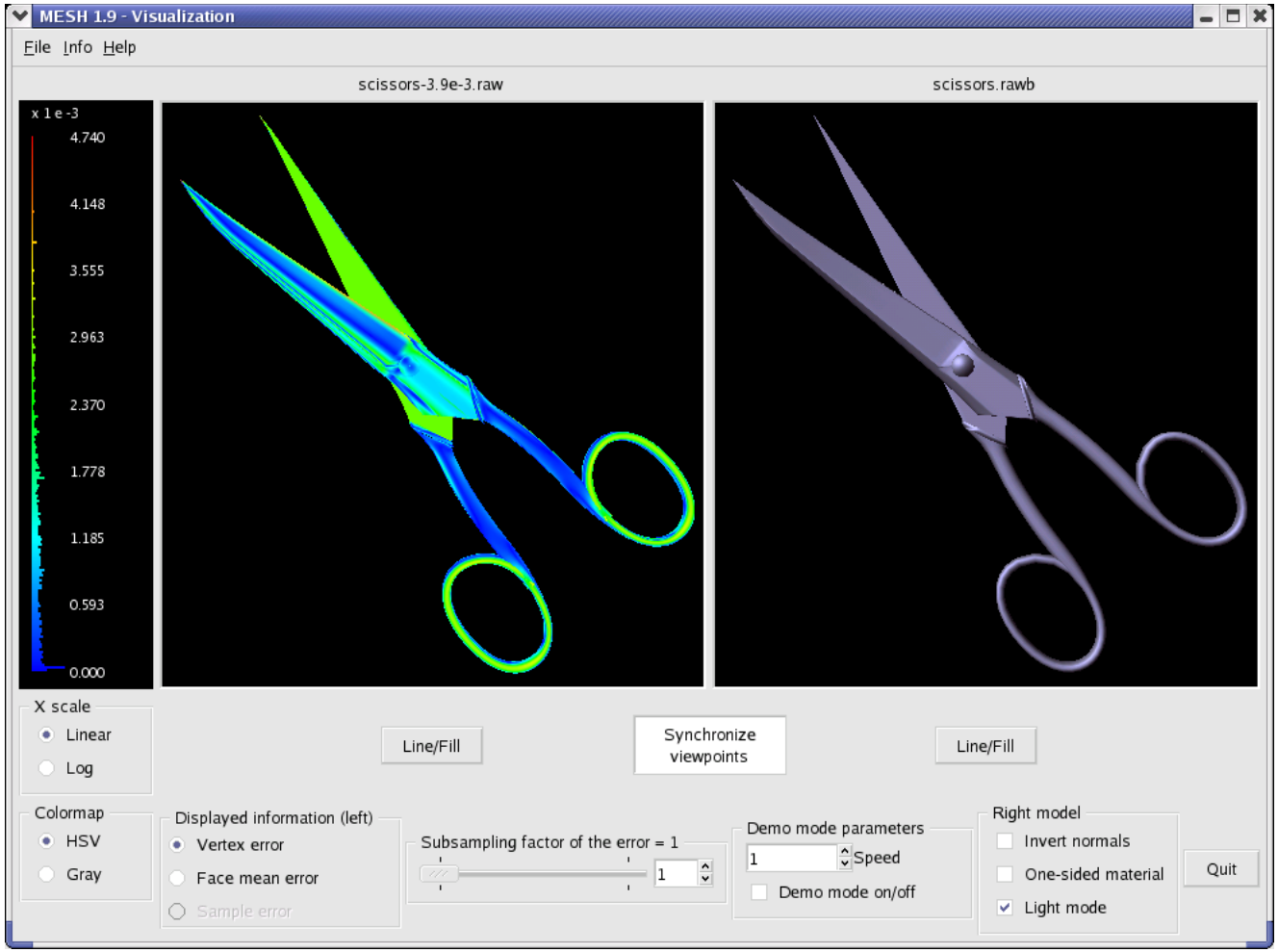
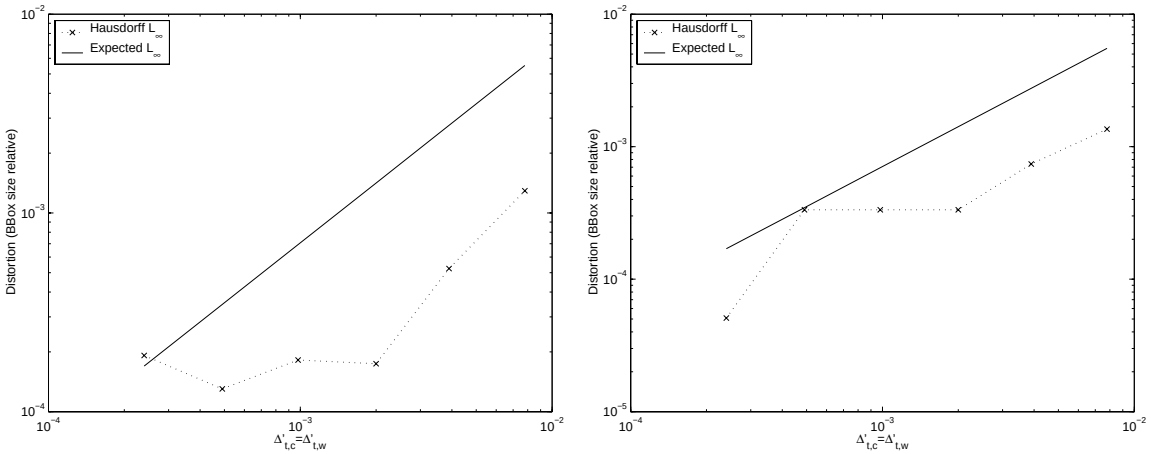
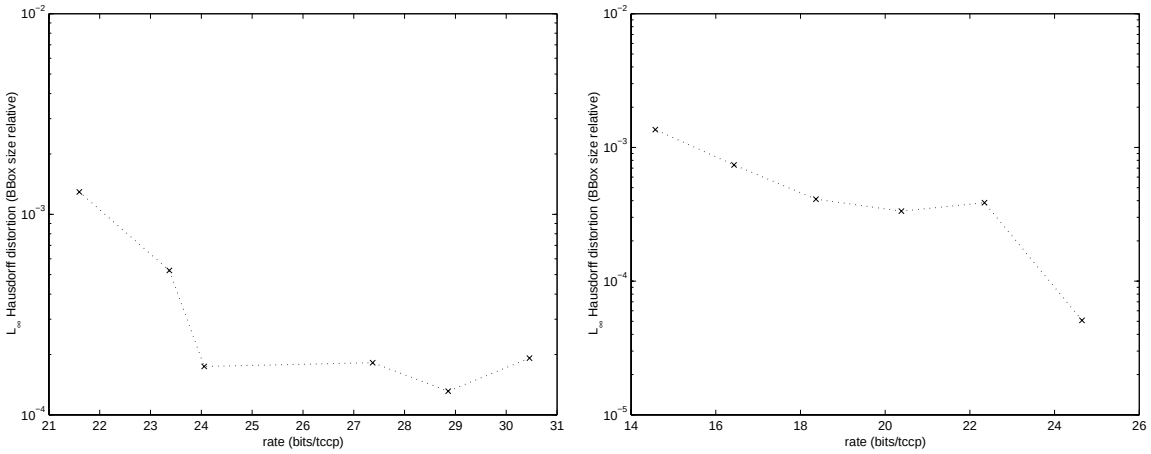


Figure 4.27: Hausdorff distortion distribution for the `scissors` model coded with $\Delta'_c = 3.9 \times 10^{-3}$ and $\Delta'_k = \Delta' w = 7.8 \times 10^{-3}$



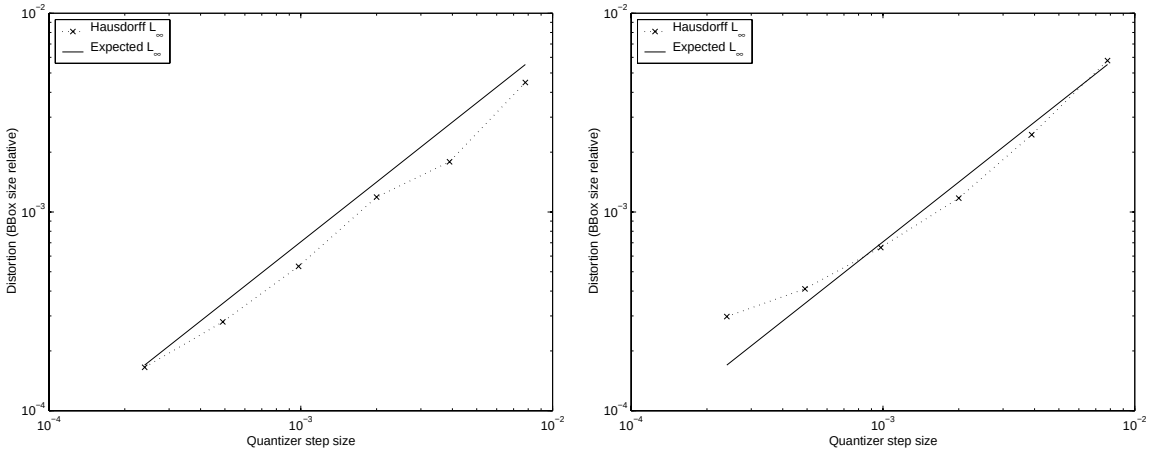
(a) `flashlight` **(b)** `subprop`

Figure 4.28: Trim curve distortion for varying $\Delta'_{t,c}$ and $\Delta'_{t,w}$, with $\Delta'_{t,c} = \Delta'_{t,w}$, and $\Delta'_{t,k} = 7.8 \times 10^{-3}$.



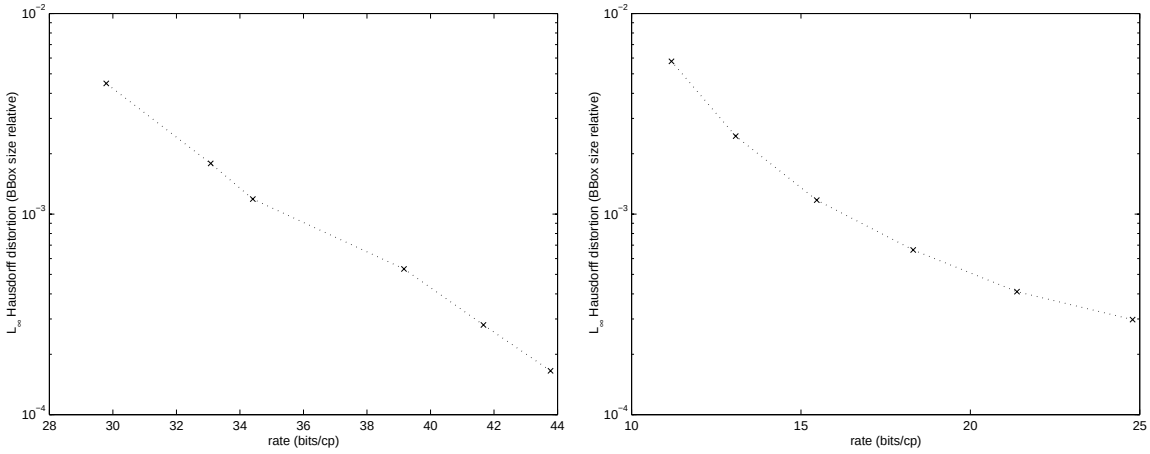
(a) `flashlight` **(b)** `subprop`

Figure 4.29: Trim curve rate-distortion for varying $\Delta'_{t,c}$ and $\Delta'_{t,w}$, with $\Delta'_{t,c} = \Delta'_{t,w}$, and $\Delta'_{t,k} = 7.8 \times 10^{-3}$.



(a) `flashlight` **(b)** `subprop`

Figure 4.30: Overall distortion for trimmed curves, with all quantizer step sizes equal.



(a) `flashlight` **(b)** `subprop`

Figure 4.31: Overall rate-distortion for trimmed models, with all quantizer step sizes equal.

4.8.3 Comparative compression ratios

In Section 4.2 we described how NURBS surfaces are represented in the VRML NURBS extension [113]. The use of NURBS in VRML is appealing since NURBS can represent complex surfaces with a relatively low number of coefficients, in particular when compared to polygonal meshes. Table 4.3 shows the compression ratios, with respect to the original VRML file size, achieved by the proposed NURBS coding, for various quantizer settings. We also include in the comparison the result of compressing the VRML data with the popular `gzip` file compression utility, as it is typically used with VRML. In addition, we also report the results of quantizing knots, coordinates and weights with a step size 2^{-24}. This is equivalent to the relative precision of IEEE single precision 32 bit floating point data, which is the precision required by VRML. Hence we label the column as "nominally lossless", because besides rounding errors introduced in the computations during the coding process the result is as precise as the original VRML model.

The original VRML size is, however, not a good basis for comparison since it can vary considerably depending on the number of whitespace characters used and such textual formatting details. In addition, VRML data is rarely transmitted uncompressed. A more meaningful comparison is shown in Table 4.4, where the compression ratios are expressed with respect to the VRML data compressed with `gzip`. This compression removes a lot of the variability in the original file size and is the common form under which VRML data is transmitted. The results show that average compression ratios between 6 and 8 are obtained for quantizer step sizes that create very little distortion. We can also observe that an average compression ratio of 2 is obtained for nominally lossless data, even if the NURBS coding system has not been specifically designed for such mode of operation. It is also worth noting that the compression efficiency can be increased a little by adjusting the individual quantizer step sizes as shown in the previous sections, instead of setting all of them to the same value.

4.8.4 Global and local quantizer bases

In the previous sections we have presented results by performing the control point coordinate quantization in the global coordinate system. We now compare the performance of quantization in the local basis, as described in Section 4.5.1, and the global basis.

Figure 4.32 shows the Hausdorff L_∞ and L_2 distortions for the both quantization bases. As expected the L_∞ distortion is the same for both bases, on all models. In fact, the quantization on a different basis amounts for a rotation and hence should not modify the L_∞ distortion. On the other hand, the L_2 distortion is, in general, smaller when the quantization is performed on the local basis. For some models, such as `scissors`, it is considerably so. From these results one would be tempted to deduce that the local basis performs better than the global one. Figure 4.33 shows the corresponding rate-distortion curves. Although for the same quantizer the local basis performs better, it incurs an increase in the coded bitrate that, in general, makes its rate-distortion performance inferior to that of the global quantizer. This is particularly so for `coke`. Nevertheless, the local basis shows a slight advantage at high bitrates for `gnom` and `killeroo-hires`, which are smoother models. This slight advantage does not, however, justify the considerable increase in computational complexity that the use of the local basis incurs.

4.8.5 Coded bit distribution

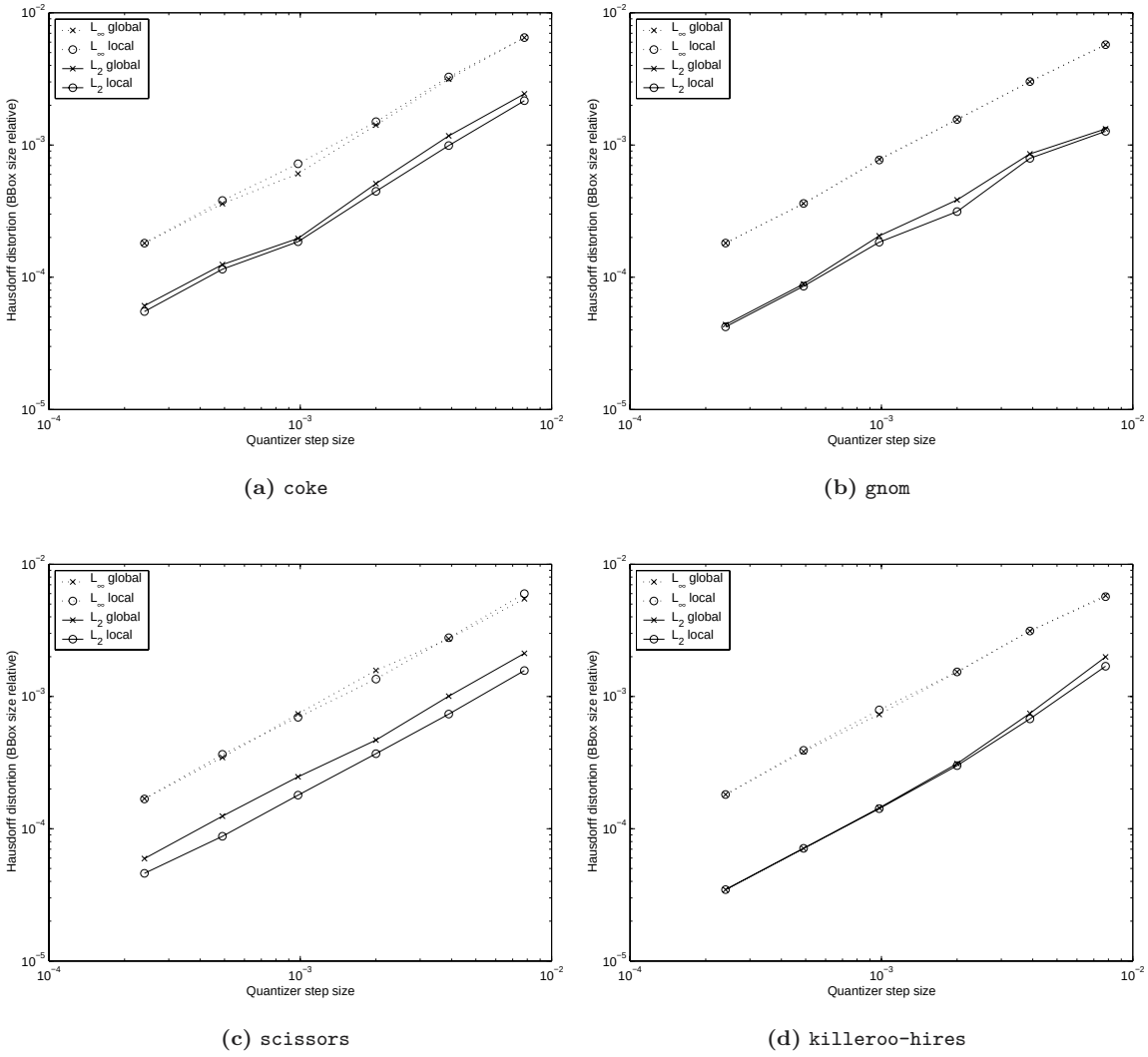
In the previous sections we have reported the overall coded rates, as rate-distortion curves, for the different parameters that define NURBS surfaces. It is however interesting to know how the coded bits are spent for the coding of these different parameters, in other words the coded bit distribution.

model	trimmed	VRML size (bytes)	compression ratio (over VRML size)					
			`gzip`	3.9×10^{-3}	9.8×10^{-4}	2.4×10^{-4}	1.5×10^{-5}	nominally lossless (for 32 bit floats)
`coke`	no	48936	5.5	31.7	23.5	18.4	12.2	6.9
`gnom`	no	72892	4.4	29.4	21.1	16.0	10.7	6.4
`goblet`	no	5719	4.4	22.3	17.4	14.7	11.0	6.4
`killeroo-hires`	no	1843186	2.6	82.4	50.9	30.1	12.9	5.9
`killeroo-lowres`	no	516703	2.8	53.1	31.1	18.7	9.6	5.0
`lion`	no	80171	4.8	25.8	19.3	15.3	10.8	7.2
`male-head`	no	248926	2.8	34.6	21.6	14.4	8.2	4.6
`pencil`	no	67096	7.2	79.1	49.4	40.3	30.0	20.2
`sail_seawolf`	no	144761	2.6	80.0	63.2	49.9	24.2	8.7
`sandbagger`	no	12405	3.7	38.6	24.9	17.9	10.3	5.5
`scissors`	no	27106	4.5	33.8	23.8	18.0	11.6	6.8
`stingray`	no	40044	2.6	60.8	38.0	25.1	12.7	5.9
`camera`	yes	161639	3.2	31.6	23.3	17.9	11.5	6.7
`fairing`	yes	43645	3.7	36.5	25.7	19.3	12.6	7.6
`flashlight`	yes	17076	7.9	26.0	22.5	19.7	15.9	13.5
`officechair`	yes	180212	5.5	33.0	25.5	20.3	14.2	9.0
`subprop`	yes	110563	3.4	30.7	22.0	16.4	10.5	6.5
average			4.2	42.9	29.6	21.9	13.5	7.8

Table 4.3: Compression ratios, over the original VRML file size, for various models. The `gzip` column shows the compressed file size using `gzip` at its maximum setting. The "nominally lossless" column is obtained by setting the quantizers to the relative precision of 32 bit floating point (i.e., 24 bits). The other columns are obtained by setting Δ'_k, Δ'_c, Δ'_w, $\Delta'_{t,k}$, $\Delta'_{t,c}$ and $\Delta'_{t,w}$ to the value shown, and are roughly equivalent to 8, 10, 12 and 16 bit quantizations of the surface points in Euclidean space. In all cases the coding of the duplicate map has been enabled.

model	trimmed	gzip'ed VRML size (bytes)	compression ratio (over gzip'ed VRML size) 3.9×10^{-3}	9.8×10^{-4}	2.4×10^{-4}	1.5×10^{-5}	nominally lossless (for 32 bit floats)
coke	no	8902	5.8	4.3	3.3	2.2	1.2
gnom	no	16575	6.7	4.8	3.6	2.4	1.5
goblet	no	1294	5.0	3.9	3.3	2.5	1.5
killeroo-hires	no	712312	31.8	19.7	11.6	5.0	2.3
killeroo-lowres	no	185602	19.1	11.2	6.7	3.5	1.8
lion	no	16705	5.4	4.0	3.2	2.2	1.5
male-head	no	90470	12.6	7.8	5.2	3.0	1.7
pencil	no	9322	11.0	6.9	5.6	4.2	2.8
sail_seawolf	no	55670	30.8	24.3	19.2	9.3	3.3
sandbagger	no	3328	10.4	6.7	4.8	2.8	1.5
scissors	no	5978	7.5	5.2	4.0	2.6	1.5
stingray	no	15284	23.2	14.5	9.6	4.8	2.3
camera	yes	49907	9.8	7.2	5.5	3.5	2.1
fairing	yes	11746	9.8	6.9	5.2	3.4	2.0
flashlight	yes	2166	3.3	2.9	2.5	2.0	1.7
officechair	yes	32652	6.0	4.6	3.7	2.6	1.6
subprop	yes	32667	9.1	6.5	4.8	3.1	1.9
average			12.2	8.3	6.0	3.5	1.9

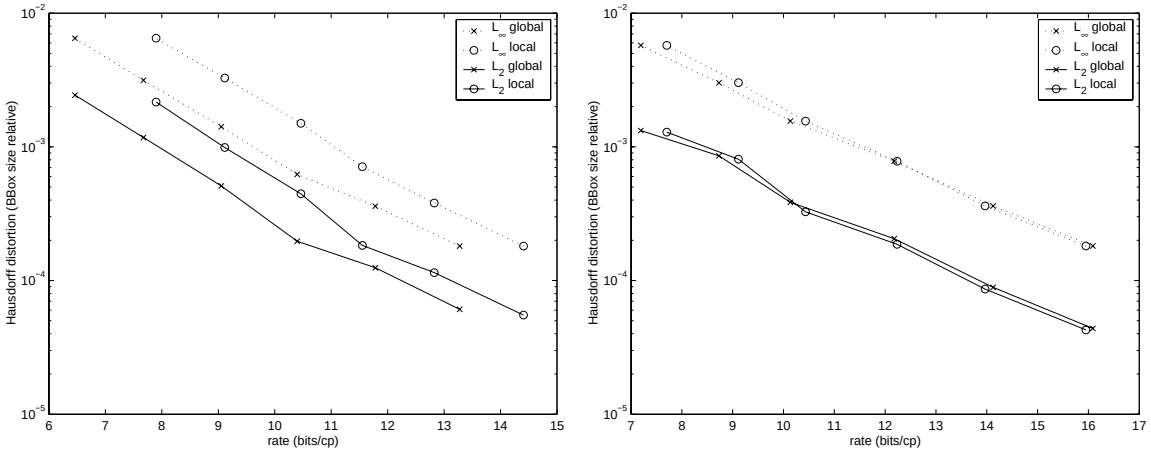
Table 4.4: Compression ratios, over the gzip'ed VRML file size, for various models. The gzip column shows the compressed file size using gzip at its maximum setting. The "nominally lossless" column is obtained by setting the quantizers to the relative precision of 32 bit floating point (i.e., 24 bits). The other columns are obtained by setting Δ'_k, Δ'_c, Δ'_w, $\Delta'_{t,k}$, $\Delta'_{t,c}$ and $\Delta'_{t,w}$ to the value shown, and are roughly equivalent to 8, 10, 12 and 16 bit quantizations of the surface points in Euclidean space. In all cases the coding of the duplicate map has been enabled.



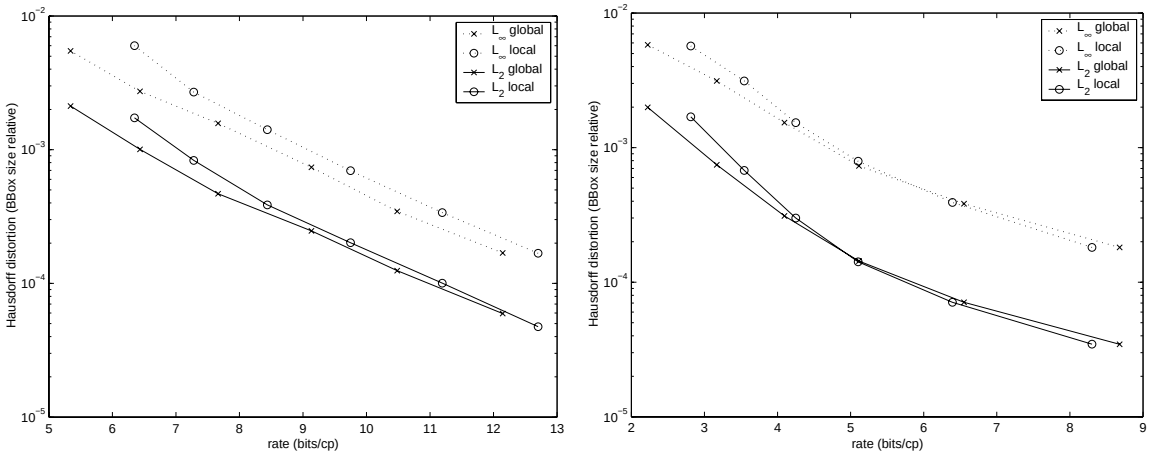
(a) `coke` **(b)** `gnom`

(c) `scissors` **(d)** `killeroo-hires`

Figure 4.32: Distortions for control point quantization in the global and local bases. All the global quantizer step sizes are set to the same value.



(a) `coke` **(b)** `gnom`



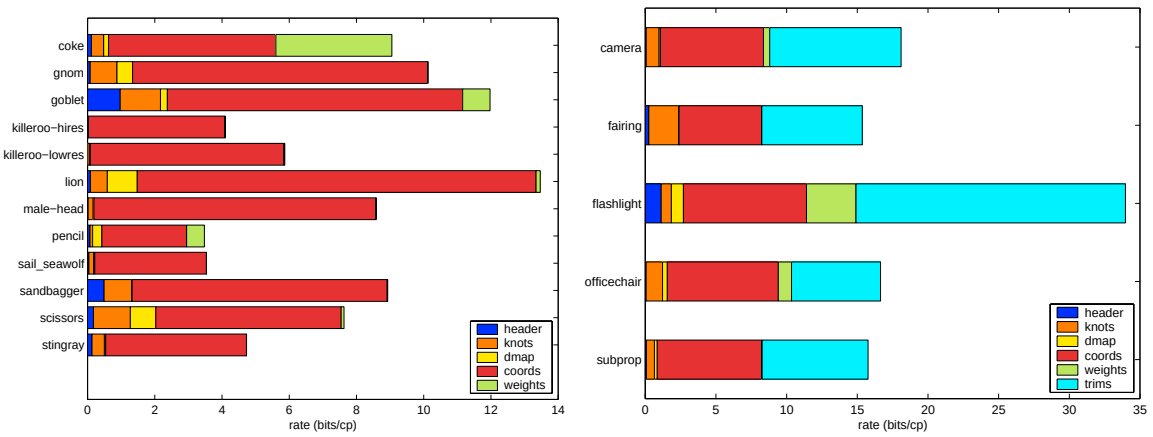
(c) `scissors` **(d)** `killeroo-hires`

Figure 4.33: Rate-distortion for control point quantization in the global and local bases. All the global quantizer step sizes are set to the same value and duplicate map coding is enabled.

Figure 4.34 shows the high-level bit distribution for non-trimmed and trimmed models. The bitstream header is of small fixed size (i.e., independent of the model size) and hence almost negligible for all but the smallest models, such as `goblet` and `sandbagger`. The largest proportion of bits is spent on the coding of control point coordinates, whereas weights take only very little space, except for `coke` as already pointed out in the analysis of weight coding performance. Likewise, knot vectors take a modest proportion of the overall bitrate. This explains why the knot and weight quantizer step sizes have a very limited effect on the coded rate. For trimmed models we can see that the coding of the trimming curves takes about half the overall coded bitrate, the reasons for which have been previously explained.

Figure 4.35 shows the detailed bit distribution for knot vectors and duplicate maps. As expected, the largest proportion of the knot coding cost is taken by the break values. This is particularly true for models that use highly non-uniform knot vectors. On the other hand, models that have uniform knot vectors (e.g., `killeroo-lowres`, `killeroo-hires`, `male-head`) or quasi-uniform ones (e.g., `pencil`) are coded very efficiently, as it is to be expected. Concerning the duplicate map we can observe that the models that have few or no duplicate points spend very little bitrate. On the other hand, models that have a considerable amount of duplicate points spend a considerably larger amount, although still relatively small. Worth noting is the fact that the coding of the offset of duplicate points takes relatively little bitrate, given that many values are possible, demonstrating the effectiveness of the coding scheme.

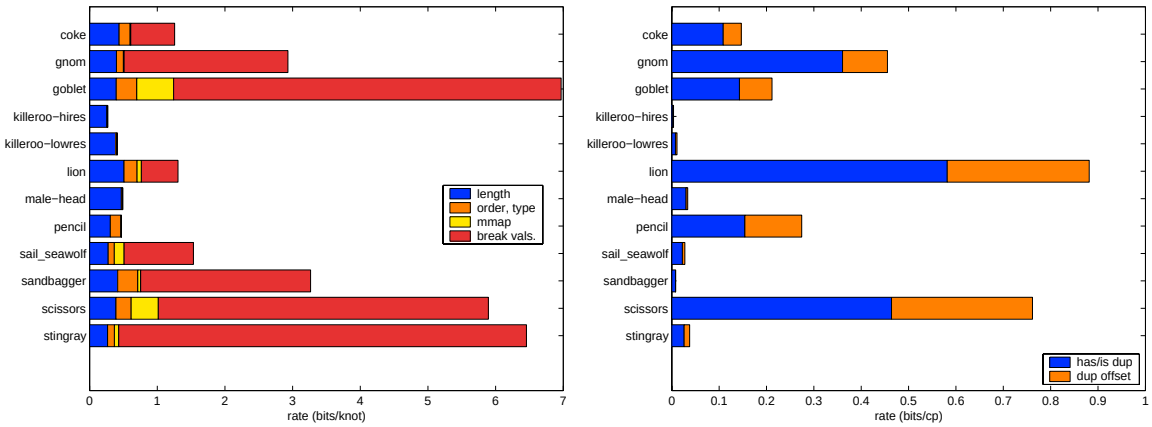
Finally, Figure 4.36 shows the detailed bit distribution for control point coordinates and weights and trimming curves. We can see that a large proportion (50% or more) of the control point coding cost is spent in coding the leading zero bits and most significant one bit (i.e., the significance map). The rest is spent in the sign and magnitude refinement bits which are almost uncompressible, since extremely difficult to predict. As for the trimming curves we can see that most of the coded rate is consumed by the trimming curve control point coordinates and, for rational curves, weights. The trimming loop header, which comprises the number of trimming loops, number of trimming curves in each loop and the per surface quantizer step adjustments, consumes very little space.



(a) non-trimmed models

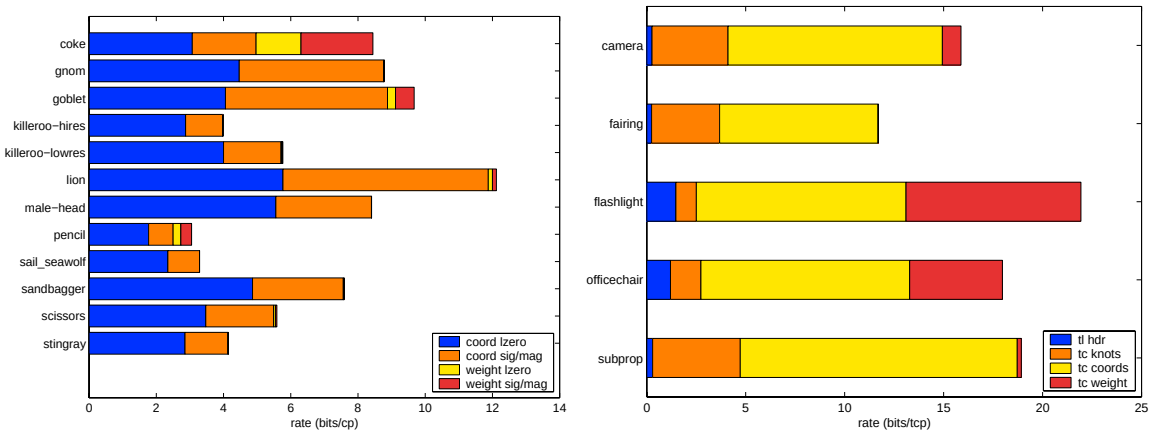
(b) trimmed models

Figure 4.34: Global distribution of coded bits for various models. All global quantizer step sizes are set to 2×10^{-3}.



(a) knot vectors **(b)** duplicate map

Figure 4.35: Detailed distribution of coded bits for knots vectors and duplicate map, for various models. All global quantizer step sizes are set to 2×10^{-3}.



(a) surface control points **(b)** trim curves

Figure 4.36: Detailed distribution of coded bits for surface control points and trimming curves. All global quantizer step sizes are set to 2×10^{-3}.

4.8.6 Comparison with polygonal meshes

Last but not least, we present a comparison of the rate-distortion of the proposed NURBS coding scheme against compressed polygonal meshes. Since NURBS is a compact representation, even if not compressed, it is to be expected that smooth models are more efficiently coded as NURBS than as polygonal meshes. The intent of this comparison is, however, to demonstrate that there is great interest, from a rate-distortion point of view, in transmitting NURBS models in their NURBS form instead of resorting to transmitting a tessellation of these, even if the resulting polygonal mesh is very efficiently compressed.

We have tessellated the NURBS models*, at several precisions, yielding multiple triangular meshes with varying triangle counts. These meshes have then been coded with the Touma-Gotsman (TG) compressor [109], reviewed in Section 3.4.4, at 8, 10 and 12 bit quantizations. Although, Touma-Gotsman compression is currently not the most effective state-of-the-art compressor it provides very competitive results for the mostly regular meshes that are obtained from the tessellation of the NURBS models. Figure 4.37 shows the results for various models. One can observe that, at low bitrates (i.e., high distortion) NURBS provide between 3 and 8 times better compression. At high bitrates (i.e., very low distortion) NURBS provide between 15 and 50 times better compression. Besides being more efficient, compressed NURBS retain most of the resolution independent characteristics of the original models. Furthermore, the coded NURBS models contain the surface normal information implicitly. Although surface normals can be estimated from the surface information in triangular models, accurate rendering would require coding the normal vectors, leading to an even greater advantage for compressed NURBS.

4.9 Conclusions

4.9.1 Summary

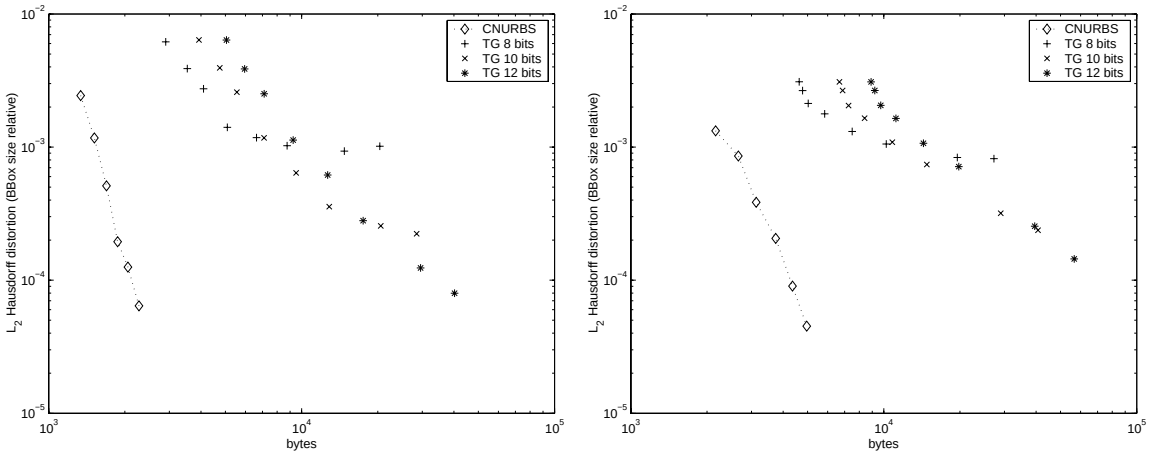
In this chapter we have proposed a new coding system for NURBS models, which is based on prediction coupled with efficient entropy coding. We have also derived bounds that allow to determine *a-priori* the distortion of the resulting coded model. The results shown demonstrate that the coder provides good rate-distortion performance and that compares very favorably to compressed VRML NURBS models, even under the conditions of "nominally lossless" distortion. We have also demonstrated the advantage, from a rate-distortion point of view, of coding NURBS models as such and not transforming them in compressed polygonal meshes.

Several requirements were listed in the introduction of this chapter, of which we can now make the following observations.

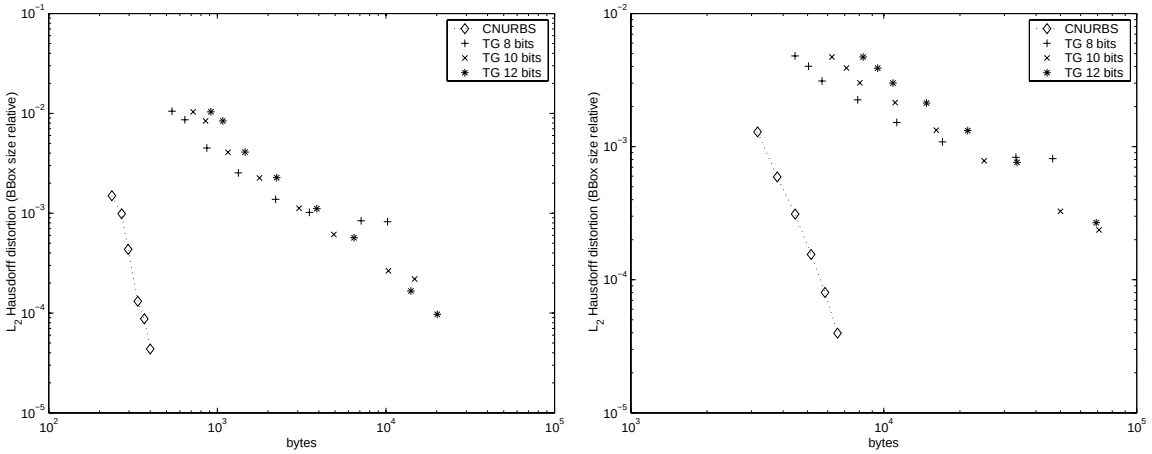
Generic : the proposed coder deals efficiently with the different types of models. The coding has been designed to very efficiently deal with common particular cases, such as clamped uniform knot vectors and one or two different surface degrees, yet it performs efficiently under the more general unconstrained cases. Furthermore, we have reported results for several types of models, finding similar rate-distortion characteristics.

Efficient : the achieved compression ratios for all models are reasonably high. With respect to compressed VRML, we find average compression ratios between 6 and 9 for very low distortion values. For "nominally lossless" (i.e., the relative precision required by VRML, that of IEEE 32-bit floating point numbers) the average compression ratio is two.

*It is worth noting that these models were originally created as NURBS and are not the result of fitting a NURBS representation to a polygonal mesh or some other non-NURBS representation.



(a) `coke` **(b)** `gnom`



(c) `goblet` **(d)** `lion`

Figure 4.37: Rate distortion of compressed NURBS (CNURBS) and the corresponding Touma-Gotsman (TG) compressed triangular meshes coded at 8, 10 and 12 bits, for various tessellation precisions.

Guaranteed distortion : the distortion bounds derived in this chapter allow to determine the quantizer step sizes adequate to ensure a prescribed L_∞ distortion. While the bounds for knots and weights overestimate the distortion, we have found that the overestimate factor is similar for all models. This allows to take them into account, in addition to the distortion bounds, when setting the quantizer step sizes for optimal rate-distortion performance.

Flexible : no arbitrary restrictions have been included in the proposed coding system. The only restrictions are that weights must be positive and that the surface degree is limited to eight. The former is an almost universally accepted restriction in systems dealing with NURBS, and thus has no effect on the applicability of the proposed coder. The latter is somewhat arbitrary, but is larger than the limits imposed by other NURBS systems. The maximum surface degree can, however, be easily increased at the expense of extra contexts for the arithmetic coder, or some slightly increased computationally complexity, at negligible coding cost.

4.9.2 Achievements

Several achievements have been realized in this chapter:

distortion bounds : we have derived bounds for the surface distortion induced by the quantization of knots, control point coordinates and weights. While the bounds for control points are rather straightforward, the one for knots is highly involved and has not been previously reported.

entropy coding : the proposed entropy coding scheme is particularly tailored to each type of coded data and deals effectively with the statistics of the prediction data.

closed and degenerate surfaces : a particular duplicate map coding technique has been proposed that efficiently deals with closed and degenerate surfaces, leading to large bitrate savings in some models.

detailed anaylis : the performance of the proposed coding system has been analyzed in detail. The influence of quantization of each data type on the surface distortion has been assessed and the rate-distortion behaviour analyzed. In addition the distribution of bits from the entropy coder has been studied.

5 Encoder design and extensions

5.1 Introduction

A method to code NURBS models has been defined in the previous chapter and its rate distortion performance analyzed along with optimal trade-offs for the different quantizer step sizes. However, some aspects of the encoding process have been neglected, namely optimal predictors and trimming curve simplification. Furthermore, no mechanism has been included to deal with transmission errors. These topics are the subject of this chapter. They complement the coding system already defined.

This chapter is organized as follows. Section 5.2 analyzes the available choices for linear predictors and their performance for different model classes and provides simple solutions that are optimal or close to optimal. Section 5.3 deals with the optimization of trimming curves that usually consume a very important part of the coded bitrate. Section 5.4 extends the coding system in a simple way to allow for error detection and containment. Finally, conclusions are drawn in Section 5.5.

5.2 Optimal linear predictors

The coder as described in the previous chapter can use arbitrary linear predictors for the control point coordinates. However, the results have been presented only for the parallelogram predictor. Here we look at different predictors and how they affect the coding performance. As different predictors do not affect the distortion we only concentrate on the coded rate.

A common choice in DPCM applications is a minimum variance (MV) predictor [see 52, chap. 6 and 11]. An MV predictor minimizes the variance of the prediction error, and hence its energy, under the assumption of a stationary signal. As lower prediction error energy usually leads to better compression the predictor is close to optimal. Following the formulation of the predictor in Eq. (4.9), the optimal predictor is found by solving, over the causal domain and for the desired number of predictor coefficients, the system of equations

$$r(l,k) = \sum_{\substack{j\geq 0 \\ j>0 \text{ when } i=0}} \lambda_{i,j} r(k-i, l-j) \qquad (k,l) \neq (0,0),$$

where $\{r(k,l)\}$ is the covariance matrix of control point coordinates.

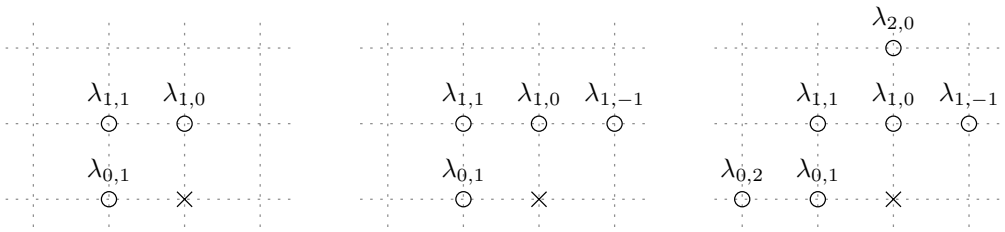
Tables 5.1 and 5.2 show the so obtained MV predictors of order 3, 4 and 6. Figure 5.1 shows the corresponding arrangements on the control point grid. We can observe that the order 3 MV predictor is always close to the parallelogram predictor, for which $\lambda_{0,1} = \lambda_{1,0} = 1$ and $\lambda_{1,1} = -1$. Furthermore, with the exception of the `killeroo` models, $\lambda_{1,-1}$ in the order 4 and 6 MV predictors is close to zero. This implies that there is, in general, little correlation between a control point and its upper-right neighbor. The same remark also applies, although to a lesser extent, to $\lambda_{0,2}$ and $\lambda_{2,0}$ in the order 6 MV predictor. Table 5.3 shows the relative changes in bitrate for the MV predictors with respect to the parallelogram predictor. Surprisingly, the performance of the former are really poor when compared with the latter, with the exception of the `killeroo` models. The `pencil` model is particularly sensible. Inspecting the prediction error data shows that the control nets of that model have rows of control points which are parallel to the yz plane. The prediction error along x will thus be zero for the parallelogram predictor and non-zero, although small, for the MV predictor. The fact that zeros are very efficiently coded explains the drastic bitrate increase. The general failure of the MV predictor is probably due to the fact that the coordinate data is not stationary. A counterexample to this trend is provided by the `killeroo` models. For these models the MV predictor yields a slightly better compression performance. This is probably due to the fact that these models are very large (17181 and 56625 control points, respectively, in 89 surfaces) and obtained by fitting NURBS surfaces to range scanning data of a real-world 3D object. Table 5.4 shows the results of experimenting on these models with higher order MV predictors, where the 8, 10 and 12 closest causal neighbors are considered. As expected the higher resolution version of the model benefits most of large predictors and shows a considerable improvement.

	order 3			order 4			
model	$\lambda_{0,1}$	$\lambda_{1,0}$	$\lambda_{1,1}$	$\lambda_{0,1}$	$\lambda_{1,0}$	$\lambda_{1,1}$	$\lambda_{1,-1}$
`coke`	0.9923	0.9823	−0.9760	0.9923	0.9821	−0.9760	0.0001
`gnom`	0.8616	0.9068	−0.7773	0.8618	0.8337	−0.7456	0.0491
`killeroo-lowres`	0.9663	0.9342	−0.9007	0.9672	0.7050	−0.8058	0.1342
`killeroo-hires`	0.9778	0.9571	−0.9349	0.9785	0.7152	−0.8281	0.1346
`pencil`	0.9978	0.9984	−0.9963	0.9978	0.9991	−0.9963	−0.0007
`stingray`	0.9811	0.9976	−0.9787	0.9811	0.9953	−0.9778	0.0013
`scissors`	0.9077	0.9953	−0.9031	0.9079	0.9885	−0.9004	0.0044

Table 5.1: Minimum variance predictors of order 3 and 4 for various models.

	order 6					
model	$\lambda_{0,1}$	$\lambda_{1,0}$	$\lambda_{1,1}$	$\lambda_{1,-1}$	$\lambda_{0,2}$	$\lambda_{2,0}$
`coke`	1.0044	0.9804	−0.9678	0.0013	−0.0161	−0.0072
`gnom`	0.8655	0.8952	−0.5262	0.0782	−0.1445	−0.2083
`killeroo-lowres`	0.9841	0.7149	−0.7340	0.1321	−0.0522	−0.0452
`killeroo-hires`	0.9889	0.7251	−0.7654	0.1309	−0.0419	−0.0376
`pencil`	0.9978	1.0011	−0.9958	−0.0007	−0.0007	−0.0021
`stingray`	0.8386	1.1521	−0.8247	0.0031	−0.0055	−0.1639
`scissors`	0.8991	1.0297	−0.8584	0.0152	−0.0169	−0.0695

Table 5.2: Minimum variance predictor of order 6 for various models.



(a) order 3 **(b)** order 4 **(c)** order 6

Figure 5.1: The arrangement of the three predictors considered.

model	order 3	order 4	order 6
`coke`	+27.2	+27.2	+33.2
`gnom`	+13.0	+12.8	+15.8
`killeroo-lowres`	−0.2	−0.9	−1.4
`killeroo-hires`	−0.4	−1.2	−1.7
`pencil`	+45.7	+38.6	+49.8
`stingray`	+10.6	+11.1	+3.5
`scissors`	+12.8	+15.3	+22.7

Table 5.3: Percent change in coded bitrate for the minimum variance predictors of Tables 5.1 and 5.2, with respect to the parallelogram predictor.

model	order 8	order 10	order 12
`killeroo-lowres`	−1.4	−1.8	−1.9
`killeroo-hires`	−5.8	−5.7	−6.1

Table 5.4: Perecent change of coded bitrate for high order minimum variance predictors, with respect to the parallelogram predictor, for the `killeroo` models.

Since the MV predictors fail to provide good results for most models we resort to exhaustive search to asses the optimality of the parallelogram predictor. The time required to search the predictor space for the optimum is large and grows exponentially with the predictor's order. In order to keep the search time in a manageable scale we first perform a rather coarse search for all predictors with coefficients in the $[-1.3, 1.3]$ range and whose sum is in the $[0.7, 1.3]$ range. Since the control nets are mainly smooth the optimum predictor should be more integrator-like than derivative-like, hence the restriction on the sum. Besides, all MV predictors sum to values between 0.95 and 1. This coarse search gives the parallelogram predictor, or a very similar one, as the optimum for orders 3, 4, 5 and 6. In a second step we refine this optimum up to a tolerance of 0.005, 0.01, 0.02 and 0.033 for the order 3, 4, 5 and 6 predictors, respectively. Table 5.5 shows the results for the order 6 predictor, for all the models above except `killeroo-lowres` and `killeroo-hires`. It turns up that the parallelogram predictor is always the optimum, except for `stingray`. It is also worth noting that the second best predictor found has, in general, considerably lower performance. In fact, the more CAD-like a model is (i.e., with control nets having sharp edges) the more sensible it is to the predictor chosen. For relatively smooth control nets, such as `gnom` and `stingray`, the exact choice of the predictor does not affect the bitrate in a considerable manner. Although not shown, the results for the order 3, 4 and 5 predictors are analogous. For the `killeroo-lowres` and `killeroo-hires` models the efficiency of the so found optimum predictors is similar to that of the MV predictors.

	optimal order 6 predictor						bitrate change (%)	
model	$\lambda_{0,1}$	$\lambda_{1,0}$	$\lambda_{1,1}$	$\lambda_{1,-1}$	$\lambda_{0,2}$	$\lambda_{2,0}$	best	2nd best
`coke`	1	1	-1	0	0	0	–	10.0
`gnom`	1	1	-1	0	0	0	–	1.4
`pencil`	1	1	-1	0	0	0	–	18.6
`stingray`	0.89941	1.06641	-0.89941	0	0	-0.06641	−0.4	−0.4
`scissors`	1	1	-1	0	0	0	–	7.6

Table 5.5: Optimal predictors, within a 0.033 tolerance, and their bitrate improvement over the parallelogram predictor for various models.

Summarizing, we have shown that the parallelogram predictor is, in general, the optimum linear predictor for all practical purposes. Although this can be surprising at first it is to be expected since control points are arranged in quadrilaterals. Designers apply only limited amounts of deformation on the quadrilaterals, making the parallelogram predictor an optimal one. This is particularly true for CAD models (e.g., `coke`, `pencil`). For models with very dense control nets (e.g., `killeroo-lowres`, `killeroo-hires`) better performance can be achieved by using the MV predictors with large support (i.e., high order), gaining a few percent in the coded bitrate.

5.3 Trimming loop optimization

As we saw in the previous chapter, trimming curves are often complex and incur a high coding cost. In Section 12 we outlined the multiple reasons for this behavior, among which we can find: complex curves obtained by numerical algorithms and large number of short curves per trimming loop. The influence of these two factors can often be reduced, with little or no surface distortion, by curve merging and curve simplification. The former will reduce the number of curves per loop, while the latter will lower the complexity of each curve. We explain these two techniques below.

5.3.1 Curve merging

Within a trimming loop, adjacent curves always have at least G^0 continuity, since they must form a closed loop. The ending point of a curve will thus coincide with the starting point of the next. Two such NURBS curves can always be merged into a single curve, if they have the same degree. Consider the curves $\mathbf{C}_1(u)$ and $\mathbf{C_2}(u)$ of common degree p defined on the clamped knot vectors

$$U_1 = \{\underbrace{a_1, \ldots, a_1}_{p+1}, u_{1,p+1}, \ldots, u_{1,r_1-p-2}, \underbrace{b_1, \ldots, b_1}_{p+1}\},$$
$$U_2 = \{\underbrace{a_2, \ldots, a_2}_{p+1}, u_{2,p+1}, \ldots, u_{2,r_2-p-2}, \underbrace{b_2, \ldots, b_2}_{p+1}\}.$$

We denote the $n_1 = r_1 - p - 1$ and $n_2 = r_2 - p - 1$ control points as $\{\mathbf{P}_{1,i}\}$ and $\{\mathbf{P}_{2,i}\}$, respectively, and $\{w_{1,i}\}$ and $\{w_{2,i}\}$ the corresponding weights. Since the curves are adjacent $\mathbf{P}_{1,n_1-1} \equiv \mathbf{P}_{2,0}$.

$\mathbf{C}_1(u)$ and $\mathbf{C}_2(u)$ can be merged into a single curve $\bar{\mathbf{C}}(u)$ by simply concatenating the knot vectors with an appropriate offset and removing the excess inner knots. The resulting knot vector is

$$\bar{U} = \left\{ \underbrace{a_1, \ldots, a_1}_{p+1}, u_{1,p+1}, \ldots, u_{1,r_1-p-2}, \underbrace{b_1, \ldots, b_1}_{p}, \right.$$
$$\left. b_1 + \frac{u_{2,p+1} - a_2}{c}, \ldots, b_1 + \frac{u_{2,r_2-p-2} - a_2}{c}, \underbrace{b_1 + \frac{b_2 - a_2}{c}, \ldots, b_1 + \frac{b_2 - a_2}{c}}_{p+1} \right\},$$

where c is an arbitrary positive constant. The length of $\bar{U}$ is $r = r_1 + r_2 - p - 2$. The $n = n_1 + n_2 - 1$ new control points are

$$\bar{\mathbf{P}}_i = \begin{cases} \mathbf{P}_{1,i} & \text{if } i \leq n_1 - 1, \\ \mathbf{P}_{2,i-n_1+1} & \text{if } i \geq n_1, \end{cases}$$

and the corresponding weights are

$$\bar{w}_i = \begin{cases} w_{1,i} & \text{if } i \leq n_1 - 1, \\ \frac{w_{1,n_1-1}}{w_{2,0}} w_{2,i-n_1+1} & \text{if } i \geq n_1. \end{cases}$$

Note the special handling of weights for the second curve. If the curves are rational it is necessary that they have coincident end-points in homogeneous space for them to be merged. This, in general, will not be the case. We can, however, renormalize the weights of the second curve to achieve coincidence in homogeneous space, as is done above. This is always possible since the curve end-points are coincident in affine space. The resulting curve is geometrically identical to the original pair of curves and hence the merging of trimming curves incurs no surface distortion.

If $c = 1$ both curves remain parametrically unchanged. However, setting $c = 1$ will often produce a merged curve that does not have good parametrization, since the parameterizations of $\mathbf{C}_1(u)$ and $\mathbf{C}_2(u)$ are *a-priori* arbitrary and unrelated. Most notably, even if the contact of the two curves is geometrically G^1 continuous the merged curve will not necessarily be parametrically C^1 continuous at the joint. If the parametric first derivatives at the end of $\mathbf{C}_1(u)$ and the start of $\mathbf{C}_2(u)$ are parallel the curves have G^1 continuous contact. To achieve parametric C^1 continuity it then suffices to set c so that the norm of the derivatives are equal. The resulting curve will be C^1 continuous, at least.

The derivatives at the end points are

$$\mathbf{C}_1'(b_1) = \frac{p}{b_1 - u_{1,r_1-p-2}} \frac{w_{1,n_1-2}}{w_{1,n_1-1}} (\mathbf{P}_{1,n_1-1} - \mathbf{P}_{1,n_1-2}),$$
$$\mathbf{C}_2'(a_2) = \frac{p}{u_{2,p+1} - a_2} \frac{w_{2,1}}{w_{2,0}} (\mathbf{P}_{2,1} - \mathbf{P}_{2,0}),$$

and therefore setting

$$c = \frac{\|\mathbf{C}_1'(b_1)\|}{\|\mathbf{C}_2'(a_2)\|} = \frac{u_{2,p+1} - a_2}{b_1 - u_{1,r_1-p-2}} \frac{w_{1,n_1-2}}{w_{1,n_1-1}} \frac{w_{2,0}}{w_{2,1}} \frac{\|\mathbf{P}_{1,n_1-1} - \mathbf{P}_{1,n_1-2}\|}{\|\mathbf{P}_{2,1} - \mathbf{P}_{2,0}\|}$$

yields a C^1 joint if $\mathbf{C}_1'(b_1)$ and $\mathbf{C}_2'(a_2)$ are parallel. Note, however, that for rational curves the result will be C^1 in affine space, but not necessarily so in homogeneous space.

At the coder we inspect each trimming loop and successively merge pairs of adjacent curves, as explained above, if they are of equal degree and have G^1 contact. Curves that have only G^0 contact were probably intended to be disjoint by the designer and are therefore not merged. In addition, merging them would leave a knot of high multiplicity that will often be as expensive to code as the clamped knot vectors of the two adjacent curves. Furthermore, there would be no clear choice of c to yield a good overall parametrization. The only exception to this rule are first degree curves, which are actually piecewise linear curves and thus have no inherent G^1 continuity. Note that merging curves with G^1 contact will also create a high multiplicity knot. Its multiplicity can, however, be reduced without distortion through knot removal, as the merged curve is at least C^1 at that knot.

Table 5.6 shows the effects of trimming curve merging on various models. The number of curves is, in general, drastically reduced since many of the small curves have G^1 contact and can be merged. The reduction of knots and control points is less important, since each merge operation removes only $p+2$ knots, of the many in each curve, and one control point. The bitrate savings are, however, non-negligible and sometimes large. Although the bitrate savings due to curve merging are sometimes modest (e.g., `camera`, `fairing`, `subprop`), the resulting longer curves allow for more aggressive simplification, as shown in the next section.

	original			merged			savings (%)	
model	# curves	# knots	# c.p.	# curves	# knots	# c.p.	trim	overall
`camera`	174	1817	1547	60	1507	1433	4.7	2.2
`fairing`	12	458	446	8	450	442	2.4	0.9
`flashlight`	48	180	140	18	120	110	17.8	9.8
`officechair`	305	1307	1037	154	983	886	13.2	4.7
`subprop`	106	986	852	60	851	806	3.3	1.4

Table 5.6: The effects of trim curve merging on various models. The bitrate savings for the trimming curve data alone as well as the overall are shown. No knot removal has been applied on the merged curves. All quantizer step sizes are set to $9.8 \times 10^{-4} \approx 2^{-10}$. The predictor for the trimming curve control points is set to the first order one.

5.3.2 Curve simplification

NURBS curves can be simplified by applying knot removal algorithms. A knot that has little or no influence over the curve's shape is removed, as is its corresponding control point. The other control points are recomputed so as to obtain a curve as close as possible to the original. If at a knot of multiplicity k the curve is C^{p-k+s} continuous the knot is s times redundant and can be removed s

times without distortion. Tiller [108] proposes an algorithm, also given in [80], that performs the knot removal by inverting the knot insertion formulas. Since the problem is overdetermined, applying the inverse formulas from the start and end of the curve yield, in general, different solutions for the control points. If and only if the knot is redundant the solutions are identical. Tiller's algorithm removes a knot if the two solutions are equal, within some small tolerance. This algorithm is hence suitable to remove redundant knots. However, non-redundant knots will not be removed even if an approximate solution to the overdetermined problem exists that incurs a small curve distortion. The simplification power is therefore limited. Note that for rational curves the algorithm is applied in projective space, where the curve is polynomial.

Eck and Hadenfeld [19] overcome the above limitation by considering the approximate solutions that are a linear combination of the two particular solutions obtained from the start and end of the curve. The coefficients for the linear combination are determined so as to minimize a distortion measure. Eck and Hadenfeld propose three parametric distortion measures: *discrete-*L_∞, *discrete-*L_2 and *continuous-*L_∞. The discrete distortions approximate the parametric curve distortion by using the convex hull property, while the continuous one directly minimizes the parametric curve distortion. As usual, we are interested in the L_∞ distortion. For simplicity's sake we use the discrete variant. The continuous one provides tighter bounds and thus has a higher simplification power, but it is highly involved requiring auxiliary functions and iterative solutions. Since the parametric distortion is considered, the parametrization of the simplified curve will be close to that of the original. Knot removal can, in some cases, modify the endpoints of the curve. This is not desirable as that it breaks the G^0 continuity of the trimming loop. To avoid it, we set the constraints on Eck and Hadenfeld's algorithm to ensure parametric C^0 contact at the endpoints. The details of the somewhat involved algorithm and distortion bounds can be found in [19].

The above algorithm solves the problem of removing one knot. Inspired by [70] we rank the knots of a curve by the *discrete-*L_∞ distortion associated with their removal and remove the one with the lowest one. After the removal they are ranked again by the new L_∞ distortions and the procedure is repeated. At each step we keep track of the accumulated distortion on each knot span and stop when the removal of any knot would exceed the allowed L_∞ distortion. As we mentioned above, rational curves are processed in projective space and we need to relate the L_∞ distortion in projective space to the one in affine space. This is done by using the bound provided by Tiller [108], which is described in Section 4.4.2. The maximum allowed L_∞ distortion, or *simplification factor*, is specified relative to the trimming curve coordinate quantizer $\Delta'_{t,c}$. Typically simplification factors below 0.5 are used, so that the simplification distortion is less than the coding distortion. Using a zero simplification factor, or some very small number, will only remove the redundant knots.

Table 5.7 shows the results of applying trim curve simplification on various models with and without prior merging. The simplification factor is 0.2. We can see that simplification is much more effective if combined with merging. In fact, merging curves allow more opportunities for knot removal, as previously disjoint parts are considered together. Furthermore, the constraint of C^0 contact at endpoints will be less stringent as there are fewer endpoints. The achieved overall bitrate savings are, in general, around 10% for virtually no increase in the distortion. Table 5.8 shows the bitrate savings for different values of the simplification factors. The first column uses a very small simplification factor and hence only redundant knots, which were either present in the original curve or introduced by merging, are removed. We see that for many models a good part of the gain is achieved by merging and removal of redundant knots only. We can also observe that most of the possible gains are achieved by a simplification factor of 0.3. Beyond this value the additional gains diminish and the simplification and quantization distortions start being comparable and hence there is the risk of exceeding the maximum specified overall coding distortion. A special case is the `fairing`

model, for which increasing the simplification factor can decrease the bitrate savings. Although the simplification removes more and more knots, the resulting curve is not necessarily easier to compress. Nonetheless, the bitrate savings are close to zero on this model and the counterintuitive behavior is negligible.

	original		simplified		simplified + merged		savings (%)	
model	# knots	# c.p.	# knots	# c.p.	# knots	# c.p.	trim	overall
`camera`	1817	1547	1674	1404	1133	1059	21.7	10.4
`fairing`	458	446	398	386	384	376	0.6	0.2
`flashlight`	180	140	176	136	116	106	26.6	14.7
`officechair`	1307	1037	1237	967	868	771	24.5	8.6
`subprop`	986	852	933	799	685	640	18.4	7.5

Table 5.7: The effects of trimming curve simplification, alone and combined with merging, on various models. The simplification factor is set to 0.2. The bitrate savings are reported for merged and simplification combined. All quantizer step sizes are set to $9.8 \times 10^{-4} \approx 2^{-10}$. The predictor for the trimming curve control points is set to the first order one.

	bitrate savings (%)						
model	1×10^{-4}	0.05	0.1	0.2	0.3	0.4	0.5
`camera`	4.1	9.8	10.1	10.4	11.0	12.0	12.5
`fairing`	0.4	0.3	0.6	0.2	0.1	0.1	1.2
`flashlight`	10.9	14.7	14.7	14.7	14.7	14.7	14.7
`officechair`	7.1	8.3	8.6	8.6	8.8	8.8	8.7
`subprop`	3.0	6.3	6.6	7.5	8.1	8.6	9.2

Table 5.8: Bitrate savings for trim curve combined merging and simplification, for different simplification factors. All quantizer step sizes are set to $9.8 \times 10^{-4} \approx 2^{-10}$. The predictor for the trimming curve control points is set to the first order one.

5.4 Error resilience

Up until now we have assumed an error-free transmission channel, or storage medium for that matter. This is a fairly reasonable assumption when the network layer has strong error protection, such as in TCP/IP. If the network layer does not provide strong error protection the decoded model can be severely affected. The types of errors can be classified as follows [6, 116].

Random errors appear as independent and isolated single bit errors. They are, in general, due to additive noise on the physical transmission channel. These errors are characterized by the *bit error rate* (BER) which is the probability of a single bit being erroneous.

Burst errors appear as a block of erroneous bits. They are due to interference and fading.

Packet losses appear as missing blocks of bits. They are due to packets being undelivered because of network congestion. They are applicable only to datagram network protocols, such as UDP/IP.

Network layer protocols recur to *forward error correction* (FEC) [116] to reduce the channel BER by several orders of magnitude. Depending on the amount of FEC there can be, however, a non-negligible residual BER that can corrupt the bitstream.

Arithmetic coding is very sensible to bit errors. An erroneous bit will, at some point, affect the interval testing and an incorrect symbol will be decoded. All subsequent symbols will also be erroneously decoded as the arithmetic coder gets out of sync. Even if it the arithmetic coder was removed the coefficient bit modeling, which generates the coded symbols, is also sensible to errors and can get out of sync after a bit error. A single bit error will thus render all the following coded data useless. For instance, if the error occurs in the middle of the bitstream half of the model is lost. Even worse, if the error is not detected arbitrary data is decoded and a model that has no resemblance to the original is decoded. Missing packets have a similar effect. In the following sections we present extensions that allow to limit the extent up to which bit errors affect the model data. These extensions are inspired by the error resilience mechanisms present in JPEG 2000 [107] and similar image or video coding standards, which rely on data partitioning. A subset of these extensions also enable other functionality such as random access and on-the-fly reordering, as we later explain. We should note that the proposed extensions are intended only for modest bit error rates (BER), such as the residual errors left by network or channel level FEC.

It should be noted that the bitstream header contains very sensible information: number of surfaces, bounding box size and offset, codec options, predictor coefficients and quantizer step sizes. It is therefore of paramount importance that its transmission be error free. As the size of the header is very small (i.e., between 20 and 40 bytes) it is feasible to apply strong error correcting codes and/or automatic repeat request (ARQ) mechanisms to guarantee proper decoding with only minimal cost. Hereafter it is assumed that the header is transmitted without error.

5.4.1 Data partitioning

As explained above, the major problem with respect to error-prone transmission is that a model is coded as a single unit and a single bit error makes all the subsequent data unusable. A NURBS model has inherent data partitioning, as the different surfaces are independent. Although the surfaces are handled independently by the prediction and coefficient bit modeling stages, they are coded in the same arithmetic coder bitstream. This is in general beneficial, in particular for models with small surfaces, since the adaptive arithmetic coder processes enough data to accurately learn its statistics. For error resilience this is however catastrophic. The arithmetic coder needs therefore to be periodically terminated and restarted. The overall bitstream will be hence made of independent arithmetic coder bitstreams that are concatenated together.

We choose to restart the arithmetic coder every n surfaces, where n is signaled in the bitstream header and typically small. We exploit two special features of the MQ arithmetic coder. With adaptive arithmetic coders the initial probability distribution for each context is typically set to uniform, but with a fast rate of adaptivity. The state machine of the MQ coder has a number of so-called *fast-attack* intermediate states that provide various probability distributions with a fast rate of adaptivity. There are also other intermediate states, that we call *medium-attack*, which provide somewhat slower adaptivity. The rest of the states constitutes the steady state part of the machine and provides accurate probability estimation with slow adaptivity. A context of the MQ coder will start in the fast-attack part of the state machine to quickly converge to the optimum distribution and move on to the medium-attack and steady state parts. Despite this behavior the adaptivity bitrate cost can be considerable if the MQ coder is frequently restarted. To diminish this effect, we initialize each context to the state in the fast-attack or medium-attack parts, as appropriate, that has approximatively the expected symbol distribution. The exact state is determined by experimentally

measuring the typical distribution for each context and refining the result until optimal results are found. Note that these initial states are fixed, in the sense that the settings are hard coded.

The second feature of the MQ coder that we exploit is *termination markers*. The MQ coder never generates 2-byte words in the range `0xFF90` to `0xFFFF`, in hexadecimal. These sequences are denoted termination markers. When the decoder encounters a termination marker it locks into a mode where it behaves as if the bitstream continued with an infinite sequence of 1 bits. In this mode no more bitstream data is consumed. Each time the MQ coder is to be restarted flush its internal state and include a termination marker, just before restarting. Since the termination markers cannot be emulated by coded data, unless a bit error synthesizes one, a decoder can identify the individual bitstream segments generated by the MQ coder. Since there are a fair number of different termination markers we successively use the ones in the `0xFF90` to `0xFFCF` range to signal a sequence index with period 64. When the last surface is coded we use a special marker, `0xFFF0`, to signal the end of bitstream. Note that a termination marker is also included right after the bitstream header and before the first surface. This allows to detect packet lossless at the start of the bitstream. The average cost of including a marker is 3.5 bytes, since two bytes are required for the marker and 1.5 bytes are consumed when flushing the internal MQ coder state.

Whenever an error resilient decoder detects an error it will locate the next bitstream segment by searching for the next termination marker. It can also use the sequence index and end of bitstream markers to detect packet loss and/or markers that have been affected by bit errors.

Table 5.9 shows the overhead created by terminating and restarting the MQ coder at each surface. Even for moderately large quantization the bitrate increase is modest, around 5%. The bitrate increase for `stingray` is negligible, since it has only one surface. The granularity of this data partitioning scheme is determined by the number of surfaces in the original model. Given a BER p, the probability of a N bits long bitstream segment being hit by an error is $1-(1-p)^N$. For small p and $N < 1/(10p)$ this is approximately Np. Hence, the probability of a coded surface being hit by an error increases linearly with the coded length, until a probability close to unity is reached. Large surfaces are therefore more likely to be affected by errors. From the results of the previous chapter an expected bitrate between 10 and 15 bits per control point seems reasonable. If a surface has M control points, the probability of being hit by an error is roughly $15Mp$. If this value is larger than the application's tolerable limit, given the channel's BER p, the encoder should split the surface into two or more subsurfaces so as to meet the tolerable limit. This can be easily performed by repeated knot insertion. If the restart period n is larger than one, the surfaces should be considered in groups of n.

model	quantizer 2×10^{-3}	quantizer 2×10^{-4}
`coke`	4.8	3.4
`gnom`	3.2	1.8
`killeroo-lowres`	3.5	1.1
`pencil`	5.8	2.0
`stingray`	0.2	0.1
`scissors`	2.5	1.8

Table 5.9: Overhead (%) due to arithmetic coder termination and restart, using a restart period of one, for two quantizer step sizes.

5.4.2 Basic error detection

Despite the NURBS coding process removing most of the redundancy in the model data, there is still some residual redundancy that can be exploited to detect errors. A decoder should therefore check the following things to detect errors.

- The decoded multiplicity values should not be larger than the decoded basis function degree.
- The sum of the multiplicity values should not exceed the decoded knot vector length.
- The decoded break values should form a strictly increasing sequence.
- The control net size corresponding to the decoded knot vector lengths and surface orders should be below the maximum size handled by the decoder or specified by the application.
- The decoded weights should all be positive and not larger than one.
- The decoded control point coordinates should be within the bounding box size.
- The offsets in the duplicate map should not refer to positions outside the control point grid.

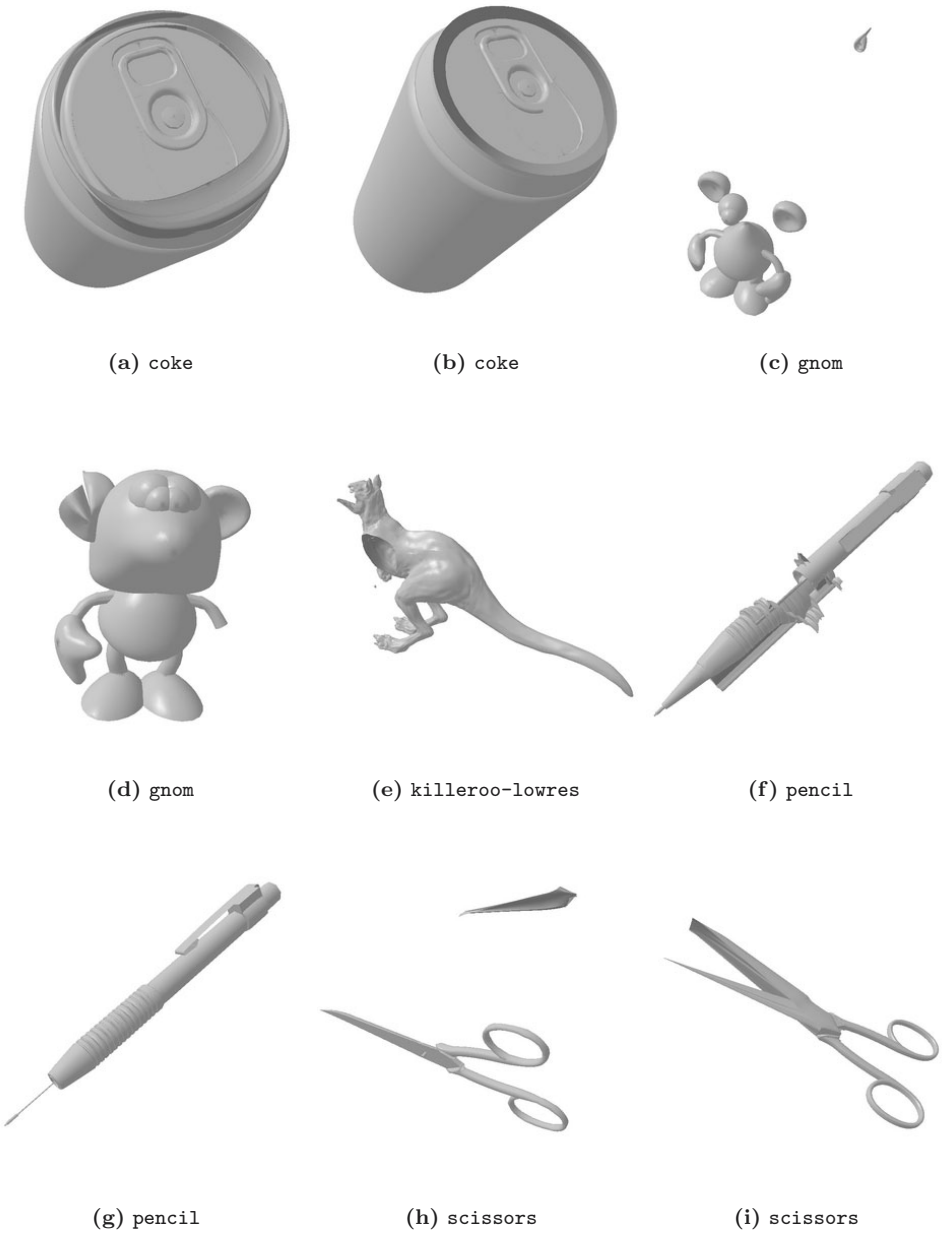
These checks should be applied on both surfaces and trimming curves. The constraint on the control net size can seem rather arbitrary, as there is basically no limit as to how large a surface can be. Nevertheless, the data needs to be decodable in a finite amount of memory that depends on the resources available to the decoder. In addition, as explained in the previous section, the probability of a surface being affected by an error is roughly proportional to its control net size and thus practical sizes are limited. An application should thus specify a maximum control net size. Larger surfaces are handled by splitting them at the encoder.

This very simple error detection technique proves to be surprisingly effective, in particular for surfaces with non-uniform knot vectors, as the most severe errors are detected. Examples of decoded models are shown in Figure 5.2 for a BER of 10^{-4}. The decoder simply drops a surface if it detects an error. We can see that severely distorted surfaces are in general avoided by this simple error detection mechanism. Occasionally, some severe errors go undetected and result in highly distorted surfaces. There is therefore a need for additional mechanisms to detect errors, as we explain in the next section. Nevertheless, we should point that since the length of the bitstream is small for some models (e.g., `pencil`, `scissors`) very often no errors will hit the bitstream and they will be perfectly decoded.

5.4.3 Segment markers

The consistency checks of the previous section allow to detect a good portion of the severe bit errors but are nevertheless insufficient. Better error detection can be achieved by including *segment markers* in the coded data, as is done in JPEG 2000 [107]. A segment marker is inserted at predetermined places by coding the `0101` bit sequence with the uniform context of the MQ-coder. The decoder should decode the same bit sequence in the places where it is expected. If the decoded sequence is not `0101` it means that an error has previously occurred. The sequence of alternating symbols and the use of the uniform context maximize the likelihood of detecting an error.

Segment markers are included at the end of the surface control point data and at the end of the trimming curve data. Including a segment marker in the knot vector data is not useful, since an error in that part will anyhow be detected later and otherwise having the knot data but no control point data available is useless. Likewise for the duplicate map data. As the segment marker is coded with the uniform context, the overhead of each segment marker is approximatively four bits and



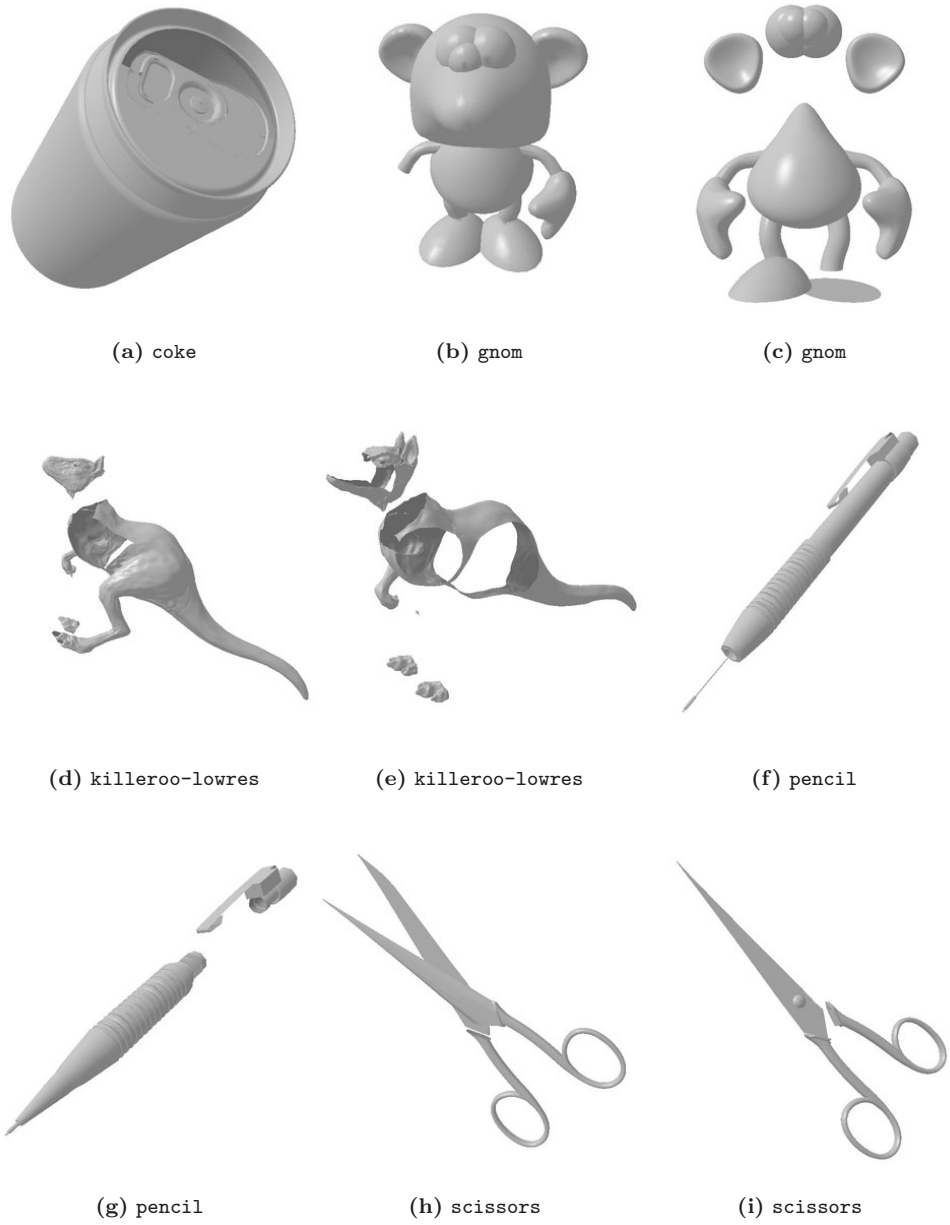
(a) coke **(b)** coke **(c)** gnom

(d) gnom **(e)** killeroo-lowres **(f)** pencil

(g) pencil **(h)** scissors **(i)** scissors

Figure 5.2: Example of decoded models for a BER of 10^{-4}, restart period of 1 and basic error detection only. The quantizer step sizes are set to 2×10^{-3}.

hence negligible. Figure 5.3 shows the results for the same conditions as those in Figure 5.2. As previously, whenever an error is detected the affected surface is dropped. As we can see, all decoded surfaces are correct. Of all the models, `killeroo-lowres` is the most affected. This is the case because given a BER of 10^{-4} and the average coded length of each surface (1182 bits) 12% of the surfaces are hit by an error, in average. In fact a network layer residual BER of 10^{-4} is large for this kind of application but we have used it to demonstrate the effects of error detection.



(a) `coke` **(b)** `gnom` **(c)** `gnom`

(d) `killeroo-lowres` **(e)** `killeroo-lowres` **(f)** `pencil`

(g) `pencil` **(h)** `scissors` **(i)** `scissors`

Figure 5.3: Example of decoded models for a BER of 10^{-4}, restart period of 1 and with segment markers. The quantizer step sizes are set to 2×10^{-3}.

In the above we have used a very simple error concealment strategy: whenever an error occurs the corresponding surface is discarded. Better error concealment is difficult to achieve because of

the lack of interleaving of control point data and the error propagation properties of DPCM. In our coder, the control point data is coded sequentially for each coordinate: all x data, followed by all y data, and so on. If the x, y, z and w data for each control point is interleaved within each bitplane, one can insert a segment marker at the end of each bitplane. A decoder would then be typically able to decode at least the most significant bitplanes and would thus recover approximate prediction error values. The approximation error would however accumulate when the prediction is undone and cause drift in the decoded data. This is particularly true for the parallelogram and minimum variance predictors, which are perfect integrators. It would thus be required to use an alternative predictor for which the influence of an error decays rapidly. Such a predictor, would however affect the decoded rate in a very significant way for CAD-like models (see Section 5.2). Another alternative is to abandon bitplane based coding and code all the bits of a prediction error value together. A segment marker could then be inserted at regular intervals. In the advent of an error, a subset of the control points would be recovered without error. Hence a portion of the surface can still be reconstructed. Giving up bitplane coding would, however, probably incur a considerable penalty in the coded bitrate since the magnitude information of neighbors can not be as effectively exploited.

5.4.4 Bitstream reordering

The data partitioning approach introduced in Section 5.4.1 enables another functionality at no additional cost. Since each group of n surfaces is independently coded, where n is the restarting period, it is possible to directly access each group of surfaces. Furthermore, all of a surface's data is contained in a single independent segment of the bitstream. These segments can therefore be reordered, removed or added to a bitstream before transmission. Only the sequence indices contained in the termination markers, and eventually the number of surfaces in the header, need to be readjusted, which is a fairly trivial operation. The only restriction is that all segments should have been coded using the same parameters. This can be used to optimize the transmission order based on the client's viewing point, so that the larger and closest surfaces arrive first. Another application is random access to large models. When only a subset of the surfaces is to be displayed only those surfaces need be decoded and there is no time wasted decoding data that will be immediately discarded. As the computational complexity of entropy decoding is orders of magnitude higher than scanning for termination markers the approach is worthwhile.

5.5 Conclusions

The performance of linear predictors for control point coordinates has been thoroughly analyzed, where minimum variance and exhaustive search methods are compared against the simple parallelogram predictor used in the previous chapter. It has been shown that the parallelogram predictor is within a few percent of the best possible bitrate. Furthermore, for CAD-like models (i.e., with control nets having sharp edges) the parallelogram predictor is optimal and any other predictor, including minimum variance, suffers from a great degradation in the compression performance. For more natural models having smoother control nets the parallelogram predictor is very close to optimal and achieves bitrates within less than 1% of the optimum. For extremely detailed models having very smooth control nets the minimum variance predictors with large support (e.g., order 8 or even 12) provide the best performance. Nonetheless, the parallelogram predictor is also within a few percent of the optimum bitrate. We can therefore conclude that the parallelogram predictor is optimal for all practical purposes, with the possible exception of highly detailed models with smooth control nets where it is sub-optimal but still a good approximation. Furthermore, using anything

else than the parallelogram predictor on CAD-like models can severely degrade the compression performance.

In the previous chapter we showed that the coding of trimming curves does often consume a very considerable part of the bitrate. We have applied two techniques, namely curve merging and simplification through approximate knot removal, that yield considerable gains in compression efficiency at virtually no additional distortion. Curve merging is performed at G^1 continuous junctions and achieves, in general, modest bitrate savings. It enables, however, much more efficient subsequent simplification. When the two techniques are combined, bitrate savings up to 10%, and sometimes even larger, have been demonstrated for simplification distortions below the quantization distortion. Curve merging does no incur any distortion.

Finally we have extended the coding system with periodic arithmetic coder restarting and termination markers to enable transmission error containment. Error detection is achieved through checks in the residual redundancy on the coded data and insertion of segment markers. These mechanisms are shown to provide effective means to detect and contain transmission errors. The natural granularity of error containment is given by the number and area of the original surfaces and it is necessary to recur to surface splitting at the encoder to improve it. Simple heuristics are provided to asses the required amount of splitting, given the channel bit error rate and the application tolerable surface loss probability. Error concealment strategies more elaborate than simple surface discarding, as demonstrated in examples, are proposed. Finally the application of termination markers to bitstream reordering for optimal viewer dependent transmission and random access of large models has been presented.

6 Applications

6.1 Introduction

The applications of three dimensional content are numerous and compression is an enabling technology in many of them, as the transmission times or storage costs would otherwise be too high. So far NURBS have not seen widespread use outside the computer aided design (CAD) realm, most probably because there has been no widespread standard for their encoding. This situation is however likely to change with the adoption of NURBS in VRML. In this chapter we present several applications and application scenarios where the compression of NURBS enables the effective use of 3D models.

The outline of this chapter is as follows. In Section 6.2 we look at the benefits of compressing VRML NURBS nodes for general applications such as virtual worlds on the Internet. Section 6.3 shows how mixed reality applications can benefit from NURBS compression. Section 6.4 concentrates on the applications in the CAD domain for network accessible models and collaborative design. Section 6.5 examines the possibility of including 3D models in new "super-teletext" services of digital television. Finally, conclusions are drawn in Section 6.6.

6.2 VRML coding

VRML is an established format for describing virtual reality worlds and the 3D objects in them. It has support for a number of primitive shapes, such as spheres, cones, cylinders and boxes. Complex shapes are however described as polygonal meshes. The widespread use of VRML has been therefore hampered by the very large size required to describe reasonably complex models. This issue has been addressed in the recent years by the intensive ongoing research in polygonal mesh coding, as reviewed in Section 3.4. For detailed models the storage requirements are however still high, even if effective compression is applied. Very recently NURBS have been added as an optional functionality to the VRML standard [30, 113] as a means of compactly describing models. While not all 3D shapes are suitable for NURBS based descriptions, a large class of models can be handled in this form. In

addition to being compact, NURBS models require few parameters to be animated and provide automatic scalability of the display accuracy based on viewing distance, available memory and CPU resources, and world complexity. Furthermore, most current 3D modeling programs are NURBS based and hence the models can be readily modified. Although compact, VRML NURBS models can still be rather large and can thus benefit from compression. For example, the Egyptian temple shown in Figure 6.1 has a file size of 638 kilobytes. Using a modem connection at 56 kbps, more than 90 seconds are required for its transmission. Even if compressed with `gzip` nine seconds are required. Even though this time might seem short, it needs to be considered that this object would be one of many in a relatively complex VRML world. Compression of this model to an accuracy of one centimeter, assuming a height of 12 meters, brings the transmission time down to two seconds, a gain of 4.5. Even if a very high accuracy of one millimeter is required the time would be reduced to three and a half seconds, a gain of 2.5. In Section 4.8.3 we have shown that for low distortion, average file sizes six times smaller than `gzip` compressed VRML are obtained. Even for a distortion not exceeding the precision required by VRML (i.e., 32 bit floating point), or in other words nominally lossless compression, the average file sizes are two times smaller.

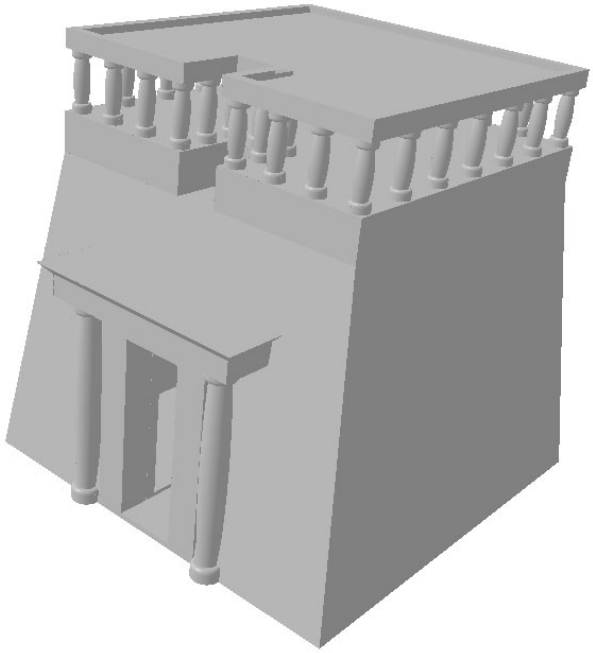


Figure 6.1: Large VRML model made of 362 surfaces and 230 trim loops, with 9827 and 3652 control points, respectively.

NURBS in VRML is however not limited to the description of surfaces. In addition to nodes describing surfaces and trimming loops, the format defines the following extra NURBS nodes:

`NurbsCurve` defines NURBS curves in 3D space,

`NurbsPositionInterpolator` defines a NURBS based 3D interpolation function,

`NurbsTextureSurface` defines a two dimensional texture coordinate interpolation NURBS function, and

`CoordinateDeformer` defines a volumetric NURBS function for shape deformation purposes.

Although not considered in the coding system defined in Chapter 4, trivial extensions can accommodate these nodes. The definition of `NurbsCurve` and `NurbsPositionInterpolator` are analogous to the definition of trimming curves, except that the control points are given in 3D. A `NurbsTextureSurface` is analogous to a NURBS surface, except that the control points are given in 2D. A `CoordinateDeformer` is actually a NURBS volume defined by a three-dimensional tensor product and thus requires three knot vectors, but otherwise is defined as a NURBS surface (i.e., its control points are in 3D). Although appropriate predictors for these data types need to be found all the distortion bounds Chapter 4, developed for curves and surfaces, can be applied by analogy.

Another important aspect when considering the application of NURBS compression to VRML is that the speed and footprint of the decoder needs to be acceptable so that a user agrees to download the necessary browser plug-in. Our naive implementation of the decoder achieves a speed of 90 thousand control points per second on a laptop computer featuring a Pentium II processor clocked at 300MHz. The size of the executable for Linux is 45 kilobytes, which is the typical size of a moderately complex NURBS model and therefore not problematic for the user to download. A properly optimized implementation can probably achieve three times faster decoding for the same footprint. In any case, the decoding speed of the naive implementation is already around three times faster than VRML parsing. The use of compressed NURBS hence leads to an important reduction of both transmission and loading time.

6.3 Mixed reality

In the recent years, mixed reality applications involving 3D objects have gained popularity. We can distinguish two main approaches. Immersive environments insert real world objects in virtual worlds. The inverse is performed in augmented reality where virtual objects are inserted in the real world, typically through semitransparent head mounted displays or by holographic means. In both these environments, a user can inspect the objects in a rather arbitrary way. For example, if a user is immersed in a virtual futuristic city, vehicles will approach from a distant point and will eventually be at a very close viewing distance. In order to optimize the utilization of graphical rendering resources and still provide an accurate rendering, the resolution of the object should be adjusted based on the viewing distance. This calls for resolution independent or progressive formats. NURBS are ideally suited, since they provide natural scalability of the rendering accuracy, which can be easily controlled in the tessellation process. For a large virtual city thousands and thousands of different models would be required to provide a realistic variety. Figure 6.2 shows a futuristic vehicle modeled with NURBS. When compressed with `gzip` the size is 82 kilobytes. If the entire city was stored in a DVD with a capacity of 9 gigabytes and ten percent of the space was devoted to model shape data, the rest of the space being devoted to other data such as textures, animation information, video clips, etc., around 10 thousand such models could be stored. While this number is high, a multitude of different object are required for a realistic city. Furthermore, models which cannot be adequately described as NURBS would require large polygonal models, rapidly diminishing the space available for other models. If the model is compressed to an accuracy of 2 millimeters, for an assumed size of 5 meters, the number of such models which can be stored increases to 80 thousand, a very comfortable number to work with.

An inverse scenario using augmented reality could be a visit of an archaeological site. Some of the objects, such as statues, could be long missing or severely damaged. It is often not desirable, or possible, to rebuild missing items. Using augmented reality with head mounted transparent displays a visitor could be presented with virtual models that are embedded in the real world, so that a convincing and realistic experience can be delivered. The head mounted unit would receive the

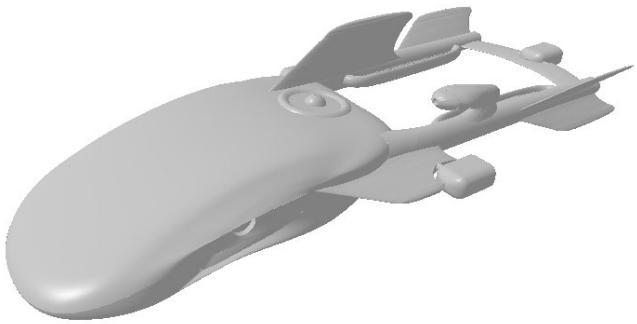
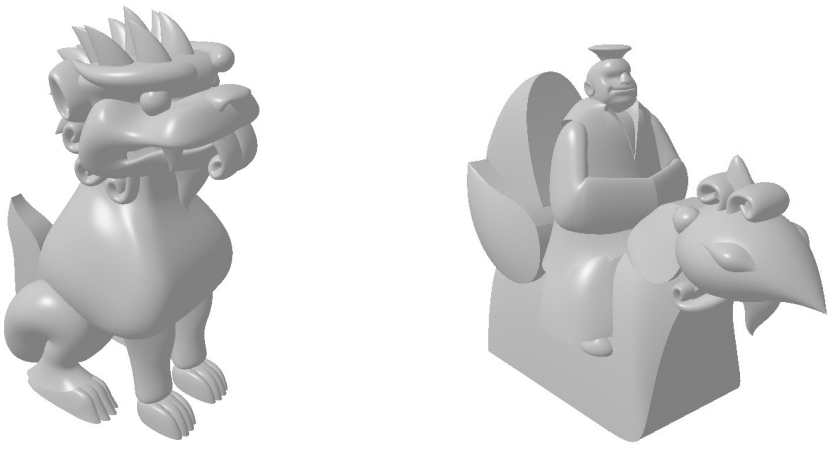


Figure 6.2: Futuristic vehicle made of 137 surfaces and 284 trimming loops, with 7667 and 1743 control points, respectively.

model data through wireless means. In this scenario, resolution independence is also highly desirable and NURBS models are a good fit. Figure 6.3 shows two statues of an hypothetical Chinese garden. The `gzip` compressed sizes are 16 and 8 kilobytes. For small sites it might not be cost effective to install a wireless LAN infrastructure, but a GPRS cellular connection with 32 kbps of available bandwidth is realistic. Receiving the statue models would therefore take between 2 and 4 seconds each, not accounting for eventual texture data network protocol overhead. Receiving the entire collection of statues and other objects for a site could easily take several minutes. Compressing both models to an accuracy of half a centimeter, assuming a height of 2 meters, yields sizes of 3.4 and 2.2 kilobytes. The download time, and thus the financial cost, would be hence reduced by a factor between three and four.



(a) 49 surfaces, 2142 control points **(b)** 47 surfaces, 1310 control points

Figure 6.3: Two Chinese garden statues.

6.4 Computer aided design

In the field of computer aided design (CAD), be it mechanical, industrial or otherwise, NURBS has been the modeling tool of choice for a long time. CAD models are often extremely large since they are modeled in very fine detail. This makes the use of compression particularly attractive for network accessible data. However, CAD applications typically require higher precision when compared to other applications. Figure 6.4 shows a large CAD model of a mountain bike. The control net shown in Figure 6.5 attests of the complexity of the model. The uncompressed data size in VRML is 7.2 megabytes, while the `gzip` compressed data occupies 1.2 megabytes. Applying NURBS compression with a precision of 60 micrometers, for an assumed bike size of 2 meters, results in a file size of 215 kilobytes. A compression factor of five and a half. Even if 32 bit floating point precision is required, and only "nominally lossless" compression is acceptable, the compressed size is 596 kilobytes, twice as short as the `gzip` compressed model, representing half a megabyte of savings.

Figure 6.4: Large CAD model made of 2262 surfaces and 1187 trimming loops, with 90855 and 43500 control points, respectively.

The size reduction achieved enables collaborative design over wide area networks (WAN) such as the Internet. Despite the compression, a fast end to end connection would still be necessary in order to work with acceptable transmission delays, if the whole model was coded as a unit. However, including segment markers allows to independently transmit each (small) group of surfaces. In the event that a remote designer modifies some surfaces only the groups containing modified surfaces need be re-encoded and sent back, allowing to carry out interactive design sessions between remote locations without the need for particularly fast connections. In addition, when the model is transmitted for the first time the ordering can be modified so that the most meaningful parts are received first and designers can start examining the objects while minor details are transmitted in the background.

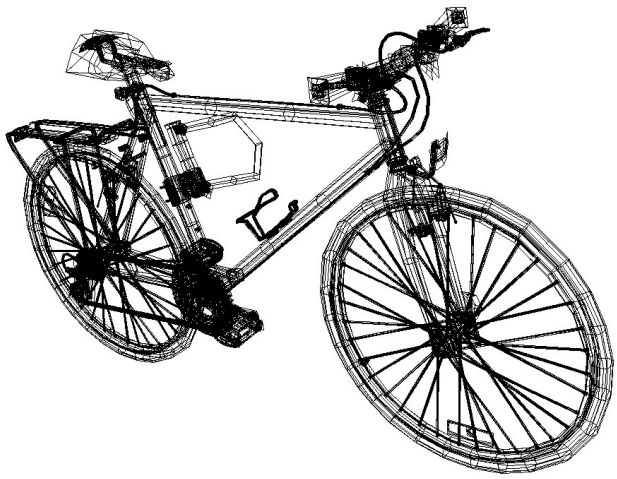


Figure 6.5: Control net of the large CAD model of Figure 6.4.

6.5 Augmented commercials

As already proposed by Bossen [7], 3D models are a compelling way to enhance TV commercials and advertisement in general. Compression is however essential to achieve sufficient model quality in a reasonable transmission time. As an example scenario in the domain of digital TV, Bossen proposes to transmit the 3D model of a product in a side channel while its advertisement plays on the video channel. However, the large size of a polygonal mesh of sufficient detail requires the dedication of an entire 1 Mbps side-channel for the duration of the commercial. In what follows, we propose a similar application involving newer digital TV services, but that is enabled by the much more compact NURBS representation.

In the past years digital TV has gained popularity and widespread deployment is taking place in many places around the world. The family of Digital Video Broadcast (DVB) standards, built on top of MPEG-2, has established itself as the most popular solution. DVB not only allows the broadcast of video and audio but also allows general data. Within DVB, the Multimedia Home Platform (MHP) [83] allows for the delivery of applications and data in what is called *object carousels*. These object carousels continually and periodically broadcast a set of files to receivers, in what could be thought of as a "super-teletext" service. The receivers can cache the files as they are received and display them to the user on demand as if an two way interactive communication existed, with the difference that no return link is required. A video channel typically requires between 3 and 4 Mbps of bandwidth. It is therefore not unreasonable to think that 1 Mbps of bandwidth could be dedicated to an object carousel. One obvious application of this "super-teletext" service is advertisement, as is currently the case with regular teletext service. The advertisements could therefore contain 3D models of the products being proposed. For instance, a car manufacturer could present its latest models in 3D, so that the user might look at them from all possible angles, change color and interior, etc. The fact that the files are continually broadcasted in a cyclic arrangement places restrictions on the sizes of the files sent, as the cycle time should be reasonable. Consider the highly detailed NURBS model of a car shown in Figure 6.6. Its `gzip`'ed VRML size is 861 kilobytes and would require almost seven seconds to be transmitted on the 1 Mbps link, not taking into account the

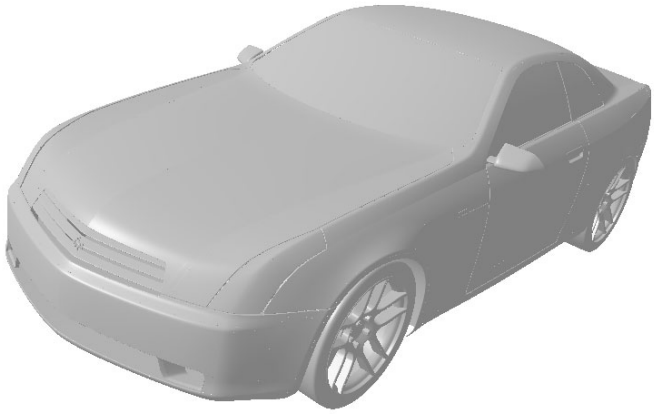


Figure 6.6: Highly detailed model of a car, made of 1057 surfaces and 338 trim loops, with 60704 and 338 control points, respectively.

link's signaling and error correction overhead. While this is not an unreasonable download time on an interactive dedicated link, it is probably not practical in an object carousel, as that would imply large cycle times. When compressed to an accuracy of 2 millimeters, assuming a car size of five meters, the file size is reduced to 90 kilobytes and thus would download in less than a second. Note also that this is comparable to the size of a 1000×1000 JPEG image. The application of NURBS models to advertisement of products is facilitated by the fact that the 3D models used during the industrial design would typically be already in NURBS form and thus readily available.

6.6 Conclusions

Several application scenarios that benefit, or are enabled, by the use of compressed 3-D NURBS models have been reviewed. First we have proposed the compression of VRML NURBS nodes and have shown that interesting compression ratios are achieved in the general case. We have also considered other factors that affect the adoption of such a technology, as are execution speed and decoder plug-in size. We have also reviewed more specific application scenarios and shown how compressed NURBS can be realistically applied. In most applications the driving requirement is compression performance. We have, however, also demonstrated how bitstream reordering and random access can enable remote collaborative design, even in demanding CAD environments. Since the proposed NURBS compression provides guaranteed distortion it is most suited to the field of computer aided design (CAD), where NURBS are the modeling primitive of choice. The applicability to other fields of computer geometry and multimedia is of course limited to the extent to which NURBS descriptions of the models are available.

7 Conclusions

7.1 Summary of achievements

In this chapter we review the most important issues and contributions presented in this thesis. Following this we discuss possible future research topics.

In this thesis a system for the compression of NURBS based 3D models has been presented. The primary objective being the best possible rate-distortion performance for a reasonable complexity we have not only presented a coding system, but also carried out a detailed distortion analysis. The results of this analysis have helped in finding optimal trade offs for the quantizer settings. The major achievements can be summarized as follows.

- Encode knot vectors as two independent entities: a break vector and a multiplicity map. This separation allows to retain the exact continuity properties of the surface, while providing data suitable for predictive coding.

- Derived an expression relating knot quantization error to curve and surface parametric L_∞ distortion. This expression handles the general case of multiple knots and provides relatively simple bounds, given the complexity of the relation.

- Based on the relation of knot, weight and control point coordinate quantization error to surface L_∞ distortion we have derived the necessary relations for meaningful settings of the quantization parameters guaranteeing a maximum overall L_∞ distortion.

- Proposed simple yet efficient entropy coders for each type of prediction error: break values, control point coordinates and weights. In particular the special handling of the uppermost and leftmost columns of the control point grid allows for higher compression efficiency.

- Proposed a simple yet very effective way to code the duplicate points arising in closed and degenerate surfaces. The entropy coder's context modeling efficiently captures the structure of the duplicate map and provides important bitrate savings for models with a significant number

of degenerate or closed surfaces, yet leaving almost unaffected the performance on the rest of the models.

- Experimentally validated the distortion analysis. The global distortion when all individual quantization errors are considered has been assessed and generic rules to achieve optimal rate-distortion performance have been derived.
- It has been demonstrated that quantization in a locally oriented basis, while being beneficial for polygonal meshes, as reported in the literature, is not efficient for NURBS models. In general it leads to severely degraded rate-distortion performance.
- It has been shown that the optimal linear predictor for CAD-like models is the simple parallelogram predictor. This predictor is also quasi-optimal for other model classes. For models with very smooth control nets, such as those obtained by a fitting process, optimal performance can be achieved with minimum variance predictors of large support (i.e., order 8 to 12).
- A method for partitioning the data stream to achieve error resilience has been defined. Processes to achieve error detection and resynchronization have been described. A simple strategy to obtain the maximum surface size given the channel conditions and application requirements has also been described. In addition, the partitioning enables random access and bitstream reordering capabilities. The problem of effective concealment remains, however, unsolved. Notwithstanding some possible solutions are outlined.

7.2 Future directions

The topic of 3D model compression has been an intense field of research for almost ten years. There has been however little research on the coding of parametric models and even less so of NURBS models, despite their popularity in many fields of computer graphics. As this is the first major work addressing this problem many ways remain unexplored. Here follows a list of some of the possible future research topics.

- Coding topology information. In this thesis we have only considered models that are a simple collection of surfaces, without information on how the different surfaces are adjacent. This is typically insufficient in solid modeling systems where a proper B-Rep with topology information is required. While the problem is similar to the encoding of polygonal mesh connectivity, several special cases arising because of curved edges need to be addressed. Encoding the B-Rep topology could also help solve the crack problem that appears at coarse quantization levels. In addition, better overall compression could probably be achieved, as coincident control points between different surfaces need be encoded only once.
- Improved predictors. Adaptive and non-linear prediction schemes could potentially deliver higher compression efficiency. It is not clear, however, that they would be able to significantly outperform the simple parallelogram predictor on CAD-like models, where the parallelogram is the natural predictor. One possible direction worth exploring is to exploit knot vector information for the prediction of control points, as largely spaced break values are often related to largely spaced control points.
- Improved error concealment. Better error protection and concealment strategies could greatly improve the decoded quality in the event of a transmission error. Possible solutions have been outlined in Section 5.4.3 but its performance and related trade offs would need to be assessed.

- Encoding of properties. This has not been considered in this work. However, unlike polygonal meshes, NURBS models typically do not have properties attached to each control point. They typically have an associated tensor-product NURBS function providing color values or texture coordinates. While the proposed NURBS coding scheme can be easily applied to these NURBS functions, the optimality of the different design choices would have to be evaluated and possible extensions proposed.

- Improved entropy coding. Although fairly efficient, the proposed entropy coding method for control point prediction errors does not exploit the potential correlation existing between the different coordinates, nor between coordinates and weights. There is therefore possible room for improvement. It remains to be seen, however, what simple relations provide worthwhile reductions in the entropy leading to improved compression ratios.

- Animated NURBS models. NURBS models are often used in animation, in particular of virtual characters, as the placement of control points can be done in a physiological meaningful way. Modifying the position of a few well chosen control points usually provides the desired animation effect. The encoding of control point movement could be an interesting way to efficiently compress animated models.

- Improved distortion measures. While the parametric and Hausdorff L_∞ and L_2 distortion measures employed in this thesis have meaningful value and have aided in finding good trade offs for the various quantization parameters, they do not always correlate well to visual distortion. In particular, models with very dense control nets but smooth surfaces can exhibit small ripples on the coded surface that are visible on shaded renderings. The problem resides in the fact that although the distortion of the surface is low, the normals rapidly oscillate between different directions. Deriving a distortion measure on the variation of the normal direction could relate well to the visual distortion. The expression of the distortion can however get very complicated and become intractable.

Distortion induced by knot quantization

A

A.1 Introduction

The quantization error incurred by knot values in the coding process modifies the surface shape. The L_∞ distortion on the knot values is easy to derive and amounts to half the quantizer step size, (i.e., $\Delta_k/2$). The relationship between knot quantization error and surface deviation is, however, far from trivial and the authors are not aware of any previous work describing it. In this appendix we demonstrate this relationship by giving upper bounds of the maximum deviation induced by the quantization of one knot, for polynomial curves and surfaces.

Introducing the knot quantization error directly in the formulation of B-Spline functions in Eq. (2.1) leads to difficult to solve equations. Instead of following this route, we solve the problem through knot insertion, as explained below. We first derive the results on curves and then extend them to surfaces.

A.2 Distortion on curves

Consider a polynomial NURBS curve $\mathbf{C}(u)$ of degree p with n control points $\{\mathbf{P}_i\}$ and knot vector $U \equiv \{u_i\}$. We want to know the maximum deviation of the curve incurred by quantizing the knots with value $\breve{u}$ as $\hat{u} = \breve{u} + \epsilon$. Let h be the multiplicity of $\breve{u}$ in U and k its maximum index, so that $u_{k-h+1} = \ldots = u_k = \breve{u}$, $u_{k-h} < \breve{u}$ and $u_{k+1} > \breve{u}$. The knot vector after quantization is thus

$$U' = \{u'_i\} = \{u_0, \ldots, u_{k-h}, \underbrace{\hat{u}, \ldots, \hat{u}}_{h}, u_{k+1}, \ldots, u_{n+p}\}$$

and defines, with the control points $\{\mathbf{P}_i\}$, the curve $\mathbf{C}'(u)$. Hence, the maximum deviation $D_k = \max \|\mathbf{C}(u) - \mathbf{C}'(u)\|$. Since these two curves are defined on different knot vectors it is difficult to evaluate D_k. We solve this problem through knot insertion.

Let $\bar{\mathbf{C}}(u)$ be the curve obtained by inserting knot $\hat{u}$ h times into U, obtaining the knot vector $\bar{U}$ and control points $\{\mathbf{Q}_i\}$. Likewise, let $\bar{\mathbf{C}}'(u)$ be the curve obtained by inserting knot $\breve{u}$ h times into

U', obtaining knot vector $\bar{U}'$ and control points $\{\mathbf{Q}'_i\}$. The maximum deviation D_k is then given by $D_k = \max \|\bar{\mathbf{C}}(u) - \bar{\mathbf{C}}'(u)\|$, since $\bar{\mathbf{C}}(u) = \mathbf{C}(u)$ and $\bar{\mathbf{C}}'(u) = \mathbf{C}'(u)$. Given that $\bar{\mathbf{C}}(u)$ and $\bar{\mathbf{C}}'(u)$ are defined on the same knot vectors (i.e., $\bar{U} = \bar{U}'$) the maximum deviation D_k is bounded by the convex hull property. That is

$$D_k \leq \max \|\mathbf{Q}_i - \mathbf{Q}'_i\|. \tag{A.1}$$

In the following we only consider $\epsilon > 0$, and thus $\hat{u} > \check{u}$, the opposite case being analogous. The augmented knot vectors are therefore

$$\bar{U} = \bar{U}' = \{\bar{u}_i\} = \{u_0, \ldots, u_{k-h}, \underbrace{\check{u}, \ldots, \check{u}}_{h}, \underbrace{\hat{u}, \ldots, \hat{u}}_{h}, u_{k+1}, \ldots, u_{n+p}\}.$$

The well-known formulas for knot insertion are given in Eq. (2.19). Extended to multiple knot insertion and applied to the curves above they yield the following recursive formulas.

$$\mathbf{Q}_{i,j} = \alpha_{i,j}\mathbf{Q}_{i,j-1} + (1 - \alpha_{i,j})\mathbf{Q}_{i-1,j-1}, \tag{A.2}$$

$$\mathbf{Q}'_{i,j} = \alpha'_{i,j}\mathbf{Q}'_{i,j-1} + (1 - \alpha'_{i,j})\mathbf{Q}'_{i-1,j-1}, \tag{A.3}$$

where

$$\mathbf{Q}_i = \mathbf{Q}_{i,h}, \qquad \mathbf{Q}_{i,0} = \mathbf{P}_i, \tag{A.4}$$

$$\mathbf{Q}'_i = \mathbf{Q}'_{i,h}, \qquad \mathbf{Q}'_{i,0} = \mathbf{P}_i, \tag{A.5}$$

$$\alpha_{i,j} = \begin{cases} 1 & 0 \leq i \leq k - p + j - 1, \\ \dfrac{\hat{u} - u_i}{u_{i+p+1-j} - u_i} & k - p + j \leq i \leq k, \\ 0 & k + 1 \leq i \leq n + j - 1, \end{cases} \tag{A.6}$$

and

$$\alpha'_{i,j} = \begin{cases} 1 & 0 \leq i \leq k - h - p + j - 1, \\ \dfrac{\check{u} - u'_i}{u'_{i+p+1-j} - u'_i} & k - h - p + j \leq i \leq k - h, \\ 0 & k - h + 1 \leq i \leq n + j - 1. \end{cases} \tag{A.7}$$

These weighting coefficients enjoy the following properties

$$0 \leq \alpha_{i,j} \leq 1 \qquad \text{and} \qquad 0 \leq \alpha'_{i,j} \leq 1, \tag{A.8}$$

as well as (see [19])

$$\alpha_{i,j} \geq \alpha_{i,j+1} \qquad \text{and} \qquad \alpha'_{i,j} \geq \alpha'_{i,j+1}. \tag{A.9}$$

Finally

$$u'_i = \begin{cases} u_i & \text{for } i = 0, \ldots, k - h \text{ and } i = k + 1, \ldots, u_{n+p}, \\ \check{u} & \text{otherwise.} \end{cases}$$

Note that $\mathbf{Q}_i = \mathbf{Q}'_i$ for $i \leq k - p$ and $i \geq k + 1$. Therefore $\bar{\mathbf{C}}(u) = \bar{\mathbf{C}}'(u)$ for $u \leq \bar{u}_{k-p+1}$ and $u \geq \bar{u}_{k+p+1}$, since a control point $\mathbf{P}_i$ affects a curve only for $u \in [u_i, u_{i+p+1})$. Transposing this back to the original curve yields

$$\mathbf{C}(u) = \mathbf{C}'(u) \qquad u \notin (u_{k-v+1-p}, u_{k-v+1+p}).$$

In other words, the curve deviation is non-zero only between the pth knot before and after the first quantized knot (i.e., u_{k-v+1}).

Let us now analyze where the deviation is non-zero. Consider the point to point distance between the ith control points of these two curves at the jth step in the knot insertion process, namely

$$d_{i,j} = \|\mathbf{Q}_{i,j} - \mathbf{Q}'_{i,j}\|. \tag{A.10}$$

Let $D_{k,j} = \max_i\{d_{i,j}\}$. Note that $d_{i,0} = 0$ and $D_{k,0} = 0$. From Eq. (A.1) and Eqs. (A.4) and (A.5) we have

$$D_k \le \max_i\{d_{i,h}\} = D_{k,h}. \tag{A.11}$$

Inserting Eqs. (A.2) and (A.3) into $\mathbf{Q}_{i,j} - \mathbf{Q}'_{i,j}$ and rearranging we obtain

$$\begin{aligned}\mathbf{Q}_{i,j} - \mathbf{Q}'_{i,j} = (\alpha_{i,j} - \alpha'_{i,j})(\mathbf{Q}_{i,j-1} - \mathbf{Q}_{i-1,j-1}) \\ + \alpha'_{i,j}(\mathbf{Q}_{i,j-1} - \mathbf{Q}'_{i,j-1}) + (1 - \alpha'_{i,j})(\mathbf{Q}_{i-1,j-1} - \mathbf{Q}'_{i-1,j-1})\end{aligned}$$

and therefore

$$\begin{aligned}d_{i,j} &\le |\alpha_{i,j} - \alpha'_{i,j}|\,\|\mathbf{Q}_{i,j-1} - \mathbf{Q}_{i-1,j-1}\| + |\alpha'_{i,j}|d_{i,j-1} + |1 - \alpha'_{i,j}|d_{i-1,j-1} \\ &= |\alpha_{i,j} - \alpha'_{i,j}|\,\|\mathbf{Q}_{i,j-1} - \mathbf{Q}_{i-1,j-1}\| + \max\{d_{i,j-1}, d_{i-1,j-1}\},\end{aligned}$$

since $|\alpha'_{i,j}| + |1 - \alpha'_{i,j}| = 1$ by Eq. (A.8), and finally

$$d_{i,j} \le |\beta_{i,j}|\,\|\mathbf{H}_{i,j-1}\| + \max\{d_{i,j-1}, d_{i-1,j-1}\} \tag{A.12}$$

by defining

$$\beta_{i,j} = \alpha_{i,j} - \alpha'_{i,j}, \tag{A.13}$$

$$\mathbf{H}_{i,j} = \mathbf{Q}_{i,j} - \mathbf{Q}_{i-1,j}, \tag{A.14}$$

where $\mathbf{H}_{i,0} = \mathbf{P}_i - \mathbf{P}_{i-1}$.

Now let us derive bounds for $\beta_{i,j}$ and $\mathbf{H}_{i,j}$. Applying Eqs. (A.6) and (A.7) to Eq. (A.13) and simplifying yields

$$\beta_{i,j} = \begin{cases} 0 & 0 \le i \le k-h-p+j-1, \\ \dfrac{\epsilon}{\epsilon + \breve{u} - u_i} < \dfrac{\epsilon}{\breve{u} - u_i} = \dfrac{\epsilon}{u_{i+p+1-j} - u_i} & k-h-p+j \le i \le \min\{k-p+j-1, k-h\} \\ 1 & k-h+1 \le i \le k-p+j-1 \\ \dfrac{\epsilon}{u_{i+p+1-j} - u_i} & k-p+j \le i \le k-h \\ \dfrac{\epsilon}{u_{i+p+1-j} - \breve{u}} = \dfrac{\epsilon}{u_{i+p+1-j} - u_i} & k-h+1 \le i \le k \\ 0 & k+1 \le i \le n+j-1 \end{cases}$$

Note that $\beta_{i,j} \ge 0$. The case $\beta_{i,j} = 1$ can only arise if $j \ge p-h+2$, which implies $h \ge \lceil p/2 \rceil + 1 \ge 2$. We delay dealing with this particular case until a later point. For all the other cases

$$\begin{aligned}\beta_{i,j} &\le \frac{\epsilon}{u_{i+p+1-j} - u_i} && i \notin [k-h+1, k-p+j-1], \\ \beta_{i,j} &= 0 && i \notin [k-h-p+j, k].\end{aligned} \tag{A.15}$$

We are now in position to give the deviation bound in the case where the knot $u_k = \check{u}$ is simple (i.e., $h = 1$). In this case we have

$$d_{i,1} = \begin{cases} \frac{\epsilon}{u_{i+p} - u_i} \|\mathbf{H}_{i,0}\| & k - p \le i \le k, \\ 0 & \text{otherwise,} \end{cases}$$

by substituting Eq. (A.15) into Eq. (A.12). The deviation bound is thus

$$\boxed{D_k \le D_{k,1} \le \max_{k-p \le i \le k} \left\{ \frac{\|\mathbf{P}_i - \mathbf{P}_{i-1}\|}{u_{i+p} - u_i} \right\} \epsilon \qquad \text{if } h = 1}, \tag{A.16}$$

by Eq. (A.11). Comparing to Eq. (2.17) we see that the deviation bound for $h = 1$ is proportional to the maximum norm, over the relevant interval, of the first derivative of the original curve.

Let us return to the general case $h > 1$. Rearranging Eq. (A.14) from Eq. (A.2) yields

$$\begin{aligned} \mathbf{H}_{i,j} &= \alpha_{i,j}(\mathbf{Q}_{i,j-1} - \mathbf{Q}_{i-1,j-1}) + (1 - \alpha_{i-1,j})(\mathbf{Q}_{i-1,j-1} - \mathbf{Q}_{i-2,j-1}) \\ &= \alpha_{i,j}\mathbf{H}_{i,j-1} + (1 - \alpha_{i-1,j})\mathbf{H}_{i-1,j-1} \end{aligned}$$

and therefore

$$\|\mathbf{H}_{i,j}\| \le \begin{cases} \|\mathbf{H}_{i,j-1}\| & i = 1, \\ |1 - \alpha_{i-1,j} + \alpha_{i,j}| \max\{\|\mathbf{H}_{i,j-1}\|, \|\mathbf{H}_{i-1,j-1}\|\} & 2 \le i \le k, \end{cases}$$

Introducing the fact that $(\alpha_{i,j} - \alpha_{i-1,j}) \in [-1, 0]$ by Eqs. (A.8) and (A.9) and expanding the recursion

$$\|\mathbf{H}_{i,j}\| \le \max_{0 \le j \le \min\{j, i-1\}} \{\|\mathbf{H}_{i-l,0}\|\} \le \max_{0 \le j \le \min\{j, i-1\}} \{\|\mathbf{P}_{i-l} - \mathbf{P}_{i-l-1}\|\}$$

From Eqs. (A.12) and (A.10)

$$d_{i,h} \le \begin{cases} \beta_{i,h}\|H_{i,h-1}\| + \max\{d_{i,h-1}, d_{i-1,h-1}\} & k - h - p + 1 \le i \le k - h \quad \text{or} \quad k - p + h \le i \le k, \\ \|\mathbf{Q}_{i,h} - \mathbf{Q}'_{i,h}\| & k - h + 1 \le i \le k - p + h - 1 \quad \text{if } h \ge \lceil \frac{p}{2} \rceil + 1, \\ 0 & \text{otherwise,} \end{cases} \tag{A.17}$$

where the definition of $d_{i,j}$ has been used in the case $\beta_{i,h} = 1$ instead of Eq. (A.12). Expanding the recursion in the first case above, we have

$$\begin{aligned} d_{i,h} &\le \sum_{t=0}^{h-1} \max_{0 \le l \le \min\{t, t+i+p-k\}} \{\beta_{i-l,h-t}\|\mathbf{H}_{i-l,h-t-1}\|\} \\ &= \sum_{t=0}^{h-1} \max_{0 \le l \le \min\{t, t+i+p-k\}} \left\{ \beta_{i-l,h-t} \max_{0 \le \tau \le l-t+h-1} \{\|\mathbf{H}_{i-\tau,0}\|\} \right\} \\ &\le \max_{0 \le \tau \le \min\{h-1, i-k+h+p-1\}} \{\|\mathbf{H}_{i-\tau,0}\|\} \sum_{t=0}^{h-1} \max_{0 \le l \le \min\{t, t+i+p-k\}} \{\beta_{i-l,h-t}\} \\ &\le \max_{0 \le \tau \le \min\{h-1, i-k+h+p-1\}} \{\|\mathbf{H}_{i-\tau,0}\|\} \sum_{t=0}^{h-1} \max_{0 \le l \le \min\{t, t+i+p-k\}} \{\frac{\epsilon}{u_{i-l+p-h+1+t} - u_{i-l}}\}. \end{aligned} \tag{A.18}$$

Note that none of the $\beta_{i-l,h-t}$ equals 1 in the expansion above. Defining

$$\phi_l = \min\{u_{i+l} - u_i\} \qquad \begin{matrix} k - h - l + 1 \le i \le k - h \\ k - l + 1 \le i \le k \end{matrix}$$

we obtain

$$d_{i,h} \leq \max_{0\leq\tau\leq\min\{h-1,i-k+h+p-1\}}\{\|\mathbf{P}_{i-\tau}-\mathbf{P}_{i-\tau-1}\|\}\sum_{l=p+1-h}^{p}\frac{1}{\phi_l}\,\epsilon. \tag{A.19}$$

For the cases where $\beta_{i,h}=1$ in Eq. (A.17), where $k-h+1\leq i\leq k-p+h-1$, we obtain

$$\mathbf{Q}_{i,h}-\mathbf{Q}'_{i,h}=\mathbf{Q}_{i,i+p-k}-\mathbf{Q}'_{k-h,k-i}$$

by exploiting the fact that many of the $\alpha_{i,j}$ and $\alpha_{i-l,j-l}$ are 1 and 0, respectively. Rearranging the above

$$\mathbf{Q}_{i,h}-\mathbf{Q}'_{i,h}=\alpha_{i,i+p-k}\mathbf{H}_{i,i+p-k-1}+(1-\alpha'_{k-h,k-i})\mathbf{H}'_{k-h,k-i-1}+\mathbf{T}_i \tag{A.20}$$

where $\mathbf{H}'_{i,j}$ is defined analogously to $\mathbf{H}_{i,j}$ in Eq. (A.14) and

$$\mathbf{T}_i=\mathbf{Q}_{i-1,i+p-k-1}-\mathbf{Q}'_{k-h,k-i-1}.$$

Solving for $\alpha_{i,i+p-k}$ and $\alpha'_{k-h,k-i}$ in Eq. (A.20) above yields

$$\mathbf{Q}_{i,h}-\mathbf{Q}'_{i,h}=\frac{\epsilon}{u_{k+1}-u_k}\mathbf{H}_{i,i+p-k-1}+\frac{\epsilon}{\epsilon+u_k-u_{k-h}}\mathbf{H}'_{k-h,k-i-1}+\mathbf{T}_i$$

and therefore

$$d_{i,h}\leq\epsilon\left(\frac{\|\mathbf{H}_{i,i+p-k-1}\|}{u_{k+1}-u_k}+\frac{\|\mathbf{H}'_{k-h,k-i-1}\|}{u_k-u_{k-h}}\right)+\|\mathbf{T}_i\|. \tag{A.21}$$

By recursively expanding $\mathbf{Q}_{i-1,i+p-k-1}$ and $\mathbf{Q}'_{k-h,k-i-1}$ from Eqs. (A.2) and (A.3) and analyzing the resulting expression for $\mathbf{T}_i$ one can demonstrate that

$$\|\mathbf{T}_i\|\leq\max_{k-2h+3\leq i\leq k-h}\{\|\mathbf{P}_i-\mathbf{P}_{i-1}\|\}\sum_{l=p-h+3}^{p}\frac{1}{\phi_l}\,\epsilon\qquad h\geq 3$$

and $\|\mathbf{T}_i\|=0$ for $h\leq 2$. Combining with Eq. (A.21) we have

$$\max_{k-h+1\leq i\leq k-p+h-1}d_{i,h}\leq\max_{k-2h+2\leq i\leq k-h}\{\|\mathbf{P}_i-\mathbf{P}_{i-1}\|\}\sum_{l=p-h+1}^{p}\frac{1}{\phi_l}\,\epsilon.$$

Combining with Eqs. (A.11), (A.17) and (A.19) we finally obtain the following two overall bounds:

$$\boxed{D_k\leq D_{k,h}\leq\max_{k-h-p+1\leq i\leq k}\{\|\mathbf{P}_i-\mathbf{P}_{i-1}\|\}\sum_{l=p+1-h}^{p}\frac{1}{\phi_l}\,\epsilon} \tag{A.22}$$

and

$$\boxed{D_k\leq D_{k,h}\leq\max_{k-h-p+1\leq i\leq k}\{\|\mathbf{P}_i-\mathbf{P}_{i-1}\|\}\frac{h}{\phi_{p-h+1}}\,\epsilon};$$

the former being tighter.

Note that when compared to Eq. (A.16) these bounds are not as tight. In fact, in order to obtain a general bound that holds for $h\geq\lceil p/2\rceil+1$ we had to relax some of the bounds. For the case where $h\leq\lceil p/2\rceil$ we can derive a rather complicated but tighter bound directly from Eq. (A.18) and obtain

$$\boxed{D_k\leq D_{k,h}\leq\max_{k-h-p+1\leq i\leq k}\sum_{t=0}^{h-1}\max_{0\leq l\leq\min\{t,t+i+p-k\}}\left\{\frac{\max_{0\leq\tau\leq l-t+h-1}\{\|\mathbf{P}_{i-\tau}-\mathbf{P}_{i-\tau-1}\|\}}{u_{i-l+p-h+1+t}-u_{i-l}}\right\}\epsilon\quad\text{if } h\leq\lceil\frac{p}{2}\rceil.} \tag{A.23}$$

Note that setting $h = 1$ gives the same bound as Eq. (A.16). All these bounds indicate that the maximum deviation D_k is proportional to ϵ, the maximum distance between pairs of adjacent control points and the knot multiplicity h, and inversely proportional to the minimum knot spacing between $p - h + 1$ and p positions apart.

A.3 Distortion on surfaces

The distortion bounds derived in the previous section are trivially extended to surfaces by considering the distortions induced by quantizations of u and v knots as additive. Consider a polynomial surface $\mathbf{S}(u, v)$ with $n \times m$ control points, of degree p and q and with U and V knot vectors. Let D_k^u be the maximum deviation if knot u_k is quantized as $u_k + \epsilon$ and D_k^v the one if knot v_k is quantized as $v_k + \epsilon$. Let h be the multiplicity of the quantized knot.

If the quantized knot is simple (i.e., $h = 1$) the bound of Eq. (A.16) becomes

$$D_k^u \leq D_{k,1}^u \leq \max_{k-p \leq i \leq k} \left\{ \frac{\max_{0 \leq j \leq m-1} \{\|\mathbf{P}_{i,j} - \mathbf{P}_{i-1,j}\|\}}{u_{i+p} - u_i} \right\} \epsilon \qquad \text{if } h = 1$$

and

$$D_k^v \leq D_{k,1}^v \leq \max_{k-q \leq j \leq k} \left\{ \frac{\max_{0 \leq i \leq n-1} \{\|\mathbf{P}_{i,j} - \mathbf{P}_{i,j-1}\|\}}{v_{j+q} - v_j} \right\} \epsilon \qquad \text{if } h = 1.$$

The more general bounds of Eq. (A.23) become

$$D_k^u \leq D_{k,h}^u \leq \max_{k-h-p+1 \leq i \leq k} \sum_{t=0}^{h-1} \max_{0 \leq l \leq \min\{t, t+i+p-k\}} \left\{ \frac{\max_{\substack{0 \leq \tau \leq l-t+h-1 \\ 0 \leq j \leq m-1}} \{\|\mathbf{P}_{i-\tau,j} - \mathbf{P}_{i-\tau-1,j}\|\}}{u_{i-l+p-h+1+t} - u_{i-l}} \right\} \epsilon$$

$$\text{if } h \leq \lceil \frac{p}{2} \rceil$$

and

$$D_k^v \leq D_{k,h}^v \leq \max_{k-h-q+1 \leq j \leq k} \sum_{t=0}^{h-1} \max_{0 \leq l \leq \min\{t, t+j+q-k\}} \left\{ \frac{\max_{\substack{0 \leq i \leq n-1 \\ 0 \leq \tau \leq l-t+h-1}} \{\|\mathbf{P}_{i,j-\tau} - \mathbf{P}_{i,j-\tau-1}\|\}}{v_{j-l+q-h+1+t} - v_{j-l}} \right\} \epsilon$$

$$\text{if } h \leq \lceil \frac{q}{2} \rceil.$$

Finally the completely general bounds of Eq. (A.22) become

$$D_k^u \leq D_{k,h}^u \leq \max_{\substack{k-h-p+1 \leq i \leq k \\ 0 \leq j \leq m-1}} \{\|\mathbf{P}_{i,j} - \mathbf{P}_{i-1,j}\|\} \sum_{l=p+1-h}^{p} \frac{1}{\phi_l^u} \epsilon$$

and

$$D_k^v \leq D_{k,h}^v \leq \max_{\substack{0 \leq i \leq n-1 \\ k-h-q+1 \leq i \leq k}} \{\|\mathbf{P}_{i,j} - \mathbf{P}_{i,j-1}\|\} \sum_{l=p+1-h}^{p} \frac{1}{\phi_l^v} \epsilon,$$

where

$$\phi_l^u = \min\{u_{i+l} - u_i\} \qquad \begin{matrix} k-h-l+1 \leq i \leq k-h \\ k-l+1 \leq i \leq k \end{matrix}$$

and

$$\phi_l^v = \min\{v_{j+l} - v_j\} \qquad \begin{matrix} k-h-l+1 \leq j \leq k-h \\ k-l+1 \leq j \leq k \end{matrix}$$

Bibliography

[1] P. Alliez, M. Desbrun (2001). Progressive compression for lossless transmission of triangle meshes. In *Proc. of the International Conference on Computer Graphics and Interactive Techniques, SIGGRAPH'01*, pp. 195–202, Los Angeles, CA, USA.

[2] P. Alliez, M. Desbrun (2001). Valence-driven connectivity encoding for 3D meshes. *Computer graphics forum* **20**(3):480–489. Presented at EUROGRAPHICS 2001, 4-7 Sept. 2001, Manchester, UK.

[3] N. Aspert, D. Santa-Cruz, T. Ebrahimi (2002). Mesh: Measuring errors between surfaces using the hausdorff distance. In *Proc. of the IEEE International Conference in Multimedia and Expo (ICME) 2002*, vol. 1, pp. 705–708, Lausanne, Switzerland.

[4] C. L. Bajaj, V. Pascucci, G. Zhuang (1999). Progressive compression and transmission of arbitrary triangular meshes. In *Proc. of IEEE Visualizetion'99*, pp. 307–316, San Francisco, CA, USA.

[5] R. Bar-Yehuda, C. Gotsman (1996). Time/space tradeoffs for polygon mesh rendering. *ACM Transactions on Graphics* **15**(2):141–152.

[6] D. Bertsekas, R. Gallager (1992). *Data Networks*. Prentice Hall, Englewood Cliffs, NJ, USA, 2nd edn.

[7] F. Bossen (1999). *On the art of compressing three-dimensional polygonal meshes and their associated properties*. Ph.d. dissertation, École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland. No. 2012.

[8] L. Bottou, P. G. Howard, Y. Bengio (1998). The Z-coder adaptive binary coder. In *Proc. of the Data Compression Conference, 1998. DCC '98*, pp. 13–22.

[9] F. M. Burgos, M. Kitahara, C. Joslin (2002). *SNHC FAQ 9*. ISO/IEC JTC1/SC29/WG11 N4972.

[10] M. S. Casale (1987). Free-form solid modeling with trimmed surface patches. *IEEE Computer Graphics and Applications* **7**(1):33–43.

[11] E. Catmull, J. Clark (1978). Recursively generated b-spline surfaces on arbitrary topological meshes. *Computer Aided Design* **10**(6):350–355.

[12] M. M. Chow (1997). Optimized geometry compression for real-time rendering. In *Proceedings of IEEE Visualization'97*, pp. 347–354.

[13] P. Cignoni, C. Rocchini, R. Scopigno (1998). Metro: measuring error on simplified surfaces. *Computer Graphics Forum* **17**(2):167–174.

[14] W. Dahmen, C. Micchelli, H.-P. Seidel (1992). Blossoming begets b-spline bases built better by b-patches. *Mathematics of Computation* **59**(199):97–115.

[15] M. Deering (1995). Geometry compression. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques, SIGGRAPH'95*, pp. 13–20, ACM Press, Los Angeles, California.

[16] R. A. DeVore, B. Jawerth, B. J. Lucier (1992). Surface compression. *Computer Aided Geometric Design* **3**(9):219–239.

[17] D. Doo, M. Sabin (1978). Behavior of recursive division surfaces near extraordinary points. *Computer Aided Design* **10**(6):356–360.

[18] N. Dyn, D. Levine, J. A. Gregory (1990). A butterfly subdivision scheme for surface interpolation with tension control. *ACM Transactions on Graphics* **9**(2):160–169.

[19] M. Eck, J. Hadenfeld (1995). Knot removal for B-spline curves. *Computer Aided Geometric Design* **12**(3):259–282.

[20] G. Farin (1992). From conics to NURBS: A tutorial and survey. *IEEE Computer Graphics and Applications* **12**(5):78–86.

[21] G. Farin (1996). *Curves and surfaces for computer aided geometric design*. Academic Press, 4th edn.

[22] G. E. Farin (1999). *NURBS: from projective geometry to practical use*. A K Peters, Ltd., Natick, Massachusetts, 2nd edn.

[23] J. D. Foley, A. van Dam, S. K. Feiner, J. F. Hughes (1996). *Computer graphics: principles and practice*. Addison-Wesley, 2nd C edn.

[24] D. R. Forsey, R. H. Bartels (1988). Hierarchical b-spline refinement. In *Proceedings of the 15th annual conference on Computer graphics, SIGGRAPH'88*, pp. 205–212, Atlanta, GA USA.

[25] Y. Furukawa, H. Masuda (2002). Compression of NURBS surfaces with error evaluation. In *NICOGRAPH International 2002*, Tokyo.

[26] M. Galassi et al. (2002). *GNU Scientific Library Reference Manual*, 1.3 edn. `http://sources.redhat.com/gsl/ref/gsl-ref_toc.html`.

[27] A. Gersho, R. M. Gray (1992). *Vector quantization and signal compression*. Kluwer Academic Publishers, Boston, MA.

[28] W. J. Gordon, R. F. Riesenfeld (1974). B-spline curves and surfaces. In R. E. Barnhill, R. F. Riesenfeld (eds.), *Computer aided geometric design : proceedings of a conference held at the University of Utah*, pp. 95–126, Academic Press, New York, Salt Lake City, Utah.

[29] W. J. Gordon, R. F. Riesenfeld (1974). Bernstein-Bézier methods for the computer aided design of free-form curves and surfaces. *Journal of ACM* **21**(2):293–310.

[30] H. Grahn, T. Volk, H. J. Wolters (2000). Nurbs in VRML. In *Proc. of the Web3D-VRML 2000 fifth symposium on Virtual reality modeling language*, pp. 35–43, ACM Press, Monterey, CA, USA.

[31] M. H. Gross, L. Lippert, O. G. Staadt (1999). Compression methods for visualization. *Future Generation Computer Systems* **15**(1):11–29.

[32] A. Guéziec, F. Bossen, G. Taubin, C. Silva (1999). Efficient compression of non-manifold polygonal meshes. *Computational Geometry* **14**(1-3):137–166.

[33] A. Guéziec et al. (1998). Converting sets of polygons to manifold surfaces by cutting and stitching. In *Proc. of Visualization '98*, pp. 383–390, Research Triangle Park, NC, USA.

[34] A. Guéziec et al. (2001). Cutting and stitching: converting sets of polygons to manifold surfaces. *IEEE Trans. on Visualization and Computer Graphics* **7**(2):136–151.

[35] S. Gumhold (1999). Improved cut-border machine for triangle mesh compression. In *Proc. of the Erlangen Workshop'99 on Vision, Modeling and Visualization*.

[36] S. Gumhold, W. Strasser (1998). Real time compression of triangle mesh connectivity. In *Proc. of the International Conference on Computer Graphics and Interactive Techniques, SIGGRAPH'98*, pp. 133–140, Orlando, FL, USA.

[37] H. Hoppe (1996). Progressive meshes. In *Proc. of the International Conference on Computer Graphics and Interactive Techniques, SIGGRAPH'96*, pp. 99–108, New Orleans, LA, USA.

[38] H. Hoppe (1998). Efficient implementation of progressive meshes. *Computer & Graphics* **22**(1):27–36.

[39] IGES/PDES (1998). *The Initial Graphics Exchage Specification (IGES) Version 6.0 - Draft*. `http://www.iges5x.org/archives/version5x/`.

[40] M. Isenburg (2000). Triangle strip compression. In *Proc. of Graphics Interface 2000*, pp. 197–204, Montreal, Que., Canada.

[41] M. Isenburg (2002). Compressing polygon mesh connectivity with degree duality prediction. In *Proc. of Graphics Interface 2002*, pp. 161–170, Calgary, Canada.

[42] M. Isenburg, P. Alliez (2002). Compressing polygon mesh geometry with parallelogram prediction. In *Proc. of Visualization 2002*, pp. 141–146, Boston, MA, USA.

[43] M. Isenburg, J. Snoeyink (2000). Face Fixer: Compressing polygon meshes with properties. In *Proc. of the International Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 2000*, pp. 263–270, New Orleans, LA, USA.

[44] M. Isenburg, J. Snoeyink (2001). Spirale Reversi: Reverse decoding of the Edgebreaker encoding. *Computational Geometry* **20**(1-2):39–52.

[45] A. Iske, E. Quak, M. S. Floater (eds.) (2002). *Tutorials on Multiresolution in Geometric Modelling*. Mathematics and Visualization. Springer-Verlag, Berling Heidelberg.

[46] ISO/IEC (1993). *ISO/IEC 11544:1993 Information technology — Coded representation of picture and audio information — Progressive bi-level image compression*.

[47] ISO/IEC (1998). *ISO/IEC 14772-1:1998 Information technology — Computer graphics and image processing — The Virtual Reality Modeling Language (VRML) — Part 1: Functional specification and UTF-8 encoding*.

[48] ISO/IEC (1999). *ISO/IEC 14496-2:1999: Information technology — Coding of audio-visual objects — Part 2: Visual*.

[49] ISO/IEC (2000). *ISO/IEC 14496-2:1999/Amd 1:2000: Visual extensions*.

[50] ISO/IEC (2002). *Study on PDAM of ISO/IEC 14496-1 / AMD4*. ISO/IEC JTC1/SC29/WG11 N4627, Jeju.

[51] ISO/IEC JTC 1/SC 29/WG 1 (1999). *ISO/IEC FCD 14492: Information technology — Coded representation of picture and audio information — Lossy/Lossless coding of bi-level images [WG 1 N 1359]*. `http://www.jpeg.org/public/jbigpt2.htm`.

[52] A. K. Jain (1989). *Fundamentals of Digital Image Processing*. Prentice Hall, Englewood Cliffs, New Jersey.

[53] N. S. Jayant, P. Noll (1984). *Digital Coding of Waveforms*. Prentice Hall, Englewood Cliffs, NJ.

[54] Z. Kami, C. Gotsman (2000). Spectral compression of mesh geometry. In *Proc. of the International Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 2000*, pp. 279–286, New Orleans, LA, USA.

[55] A. Khodakovsky, P. Alliez, M. Desbrun, P. Schröder (2002). Near-optimal connectivity encoding of 2-manifold polygon meshes. *Graphical Models* Submitted for publication, `http://www.multires.caltech.edu/pubs/ircomp.pdf`.

[56] A. Khodakovsky, P. Schröder, W. Sweldens (2000). Progressive geometry compression. In *Proc. of the International Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 2000*, pp. 271–278, New Orleans, LA, USA.

[57] D. King, J. Rossignac (1999). Guaranteed 3.67v bit encoding of planar triangle graphs. In *Proc. of the 11th Canadian Conference on Computational Geometry (CCCG)*, pp. 146–149, Vancouver, Canada.

[58] D. King, J. Rossignac, A. Szymczak (1999). *Connectivity Compression for Irregular Quadrilateral Meshes*. Tech. Rep. GVU-GIT-99-36, Georgia Tech, GVU Center, Georgia, GA, USA.

[59] B. Kronrod, C. Gotsman (2000). Optimized compression of triangle mesh geometry using prediction trees. In *Proc. of the 8th Pacific Graphics 2000 Conference*, pp. 602–608, Padova, Italy.

[60] B. Kronrod, C. Gotsman (2001). Efficient coding of nontriangular mesh connectivity. *Graphical Models* **63**(4):263–275.

[61] S. Kumar, S. Han, D. Goldof, K. Bowyer (1995). On recovering hyperquadrics from range data. *IEEE Trans. on Pattern Analysis and Machine Intelligence* **17**(11):1079–1083.

[62] S. Kumar, D. Manocha, A. Lastra (1996). Interactive display of large NURBS models. *IEEE Trans. on Visualization and Computer Graphics* **2**(4):323–336.

[63] G. G. Langdon (1988). *Compression of Multilevel Signals*. Patent Nr. 4,749,983, United States.

[64] H. Lee, P. Alliez, M. Desbrun (2002). Angle-analyzer: A triangle-quad mesh codec. *Computer Graphics Forum* **21**(3):383–392.

[65] J. Li, C.-C. J. Kuo (1998). Progressive coding of 3-D graphic models. *Proceedings of the IEEE* **86**(6):1052–1063.

[66] C. Loop (1994). Smooth spline surfaces over irregular meshes. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques, SIGGRAPH'94*, pp. 303–310, ACM Press, Orlando, Florida.

[67] H. Lopes et al. (2002). Edgebreaker: A simple compression for surfaces with handles. In *Proc. of the seventh ACM symposium on Solid modeling and applications*, pp. 289–296, Saarbrücken, Germany.

[68] M. Lounsbery, T. D. DeRose, J. Warren (1997). Multiresolution analysis for surfaces of arbitrary topological type. *ACM Transactions on Graphics* **16**(1):34–73.

[69] W. L. Luken (1996). Tessellation of trimmed nurb surfaces. *Computer Aided Design* **13**(2):163–177.

[70] T. Lyche, K. Mørken (1987). Knot removal for parametric B-spline curves and surfaces. *Computer Aided Geometric Design* **4**(3):217–230.

[71] S. Malassiotis, M. G. Strintzis (1999). Optimal biorthogonal wavelet decomposition of wireframe meshes using box splines, and its application to the hierarchical coding of 3-D surfaces. *IEEE Trans. on Image Processing* **8**(1):41–57.

[72] S. Mallat (1999). *A wavelet tour of signal processing*. Academic Press, San Diego, CA, 2nd edn.

[73] W. S. Massey (1967). *Algebraic Topology: An Introduction*. Harcourt, Brace & World, Inc., New York, NY.

[74] J. R. Miller (1986). Sculptured surfaces in solid models: Issues and alternative approaches. *IEEE Computer Graphics and Applications* **6**(12):37–48.

[75] J. Montagnat, H. Delingette, N. Ayache (2001). A review of deformable surfaces: toppology geometry and deformation. *Image and Vision Computing* **19**(14):1023–1040.

[76] R. Pajarola, J. Rossignac (2000). Compressed progressive meshes. *IEEE Trans. on Visualization and Computer Graphics* **6**(1):79–93.

[77] W. B. Pennebaker, J. L. Mitchell (1992). *JPEG: Still Image Data Compression Standard*. Van Nostrand Reinhold, New York.

[78] W. B. Pennebaker, J. L. Mitchell, G. G. Langdon, R. B. Arps (1988). An overview of the basic principles of the Q-coder adaptive binary arithmetic coder. *IBM Journal of Research and Development* **32**(6):717–726.

[79] J. Peters (1995). C^1-surface splines. *SIAM Journal of Numerical Analysis* **32**(2):645–666.

[80] L. Piegl, W. Tiller (1997). *The NURBS Book*. Springer-Verlag, Berlin, Germany, 2nd edn.

[81] L. A. Piegl, A. M. Richard (1995). Tessellating trimmed NURBS surfaces. *Computer Aided Design* **27**(1):16–26.

[82] L. A. Piegl, W. Tiller (1998). Geometry-based triangulation of trimmed NURBS surfaces. *Computer Aided Design* **30**(1):11–18.

[83] J. Piesing (1999). The DVB multimedia home platform – "mhp". In *Interactive Television (Ref. No. 1999/200), IEE Colloquium on*, vol. 2, pp. 1–6.

[84] H. Qin, D. Terzopoulos (1997). Triangular NURBS and their dynamic generalizations. *Computer Aided Geometric Design* **14**(4):325–347.

[85] G. V. V. Ravi Kumar, P. Srinivasan, K. G. Shastry, B. G. Prakash (2001). Geometry based triangulation of multiple trimmed NURBS surfaces. *Computer Aided Design* **33**(6):439–454.

[86] L. M. Reissell (1996). Wavelet multiresolution representation of curves and surfaces. *Graphical Models and Image Processing* **58**(3):198–217.

[87] R. F. Riesenfeld (1972). *Application of B-Spline approximation to Geometric Problems of Computer Aided Design*. Ph.d. dissertation, Syracuse University, Syracuse, NY. Also available as University of Utah UTEC-CSc-73-126, March, 1973.

[88] D. F. Rogers (2001). *An Introduction to NURBS: with Historical Perspective*. Morgan Kaufmann, San Francisco, CA.

[89] J. Rossignac (1999). Edgebreaker: Connectivity compression for triangle meshes. *IEEE Trans. on Visualization and Computer Graphics* **5**(1):47–61.

[90] J. Rossignac (2001). 3D compression made simple: Edgebreaker with zip&wrap on a Coner-Table. In *Shape Modeling and Applications, SMI 2001 International Conference on*, pp. 278–283, Genova, Italy.

[91] J. Rossignac, A. Szymczak (1999). Wrap&Zip decompression of the connectivity of triangle meshes compressed with Edgebreaker. *Computational Geometry* **14**(1-3):119–135.

[92] J. Rossignac et al. (2002). *Edgebreaker: A Simple Compression for Surfaces with Handles*. Tech. Rep. GIT-GVU-02-03, Georgia Tech, GVU Center, Atlanta, GA, USA.

[93] I. J. Schoenberg (1946). Contributions to the problem of approximation of equidistant data by analytic functions, Part A: On the problem of smoothing or graduation, a first class of analytic approximation formulas. *Quart. Appl. Math.* **4**(1):45–99.

[94] I. J. Schoenberg (1946). Contributions to the problem of approximation of equidistant data by analytic functions, Part B: On the problem of osculatory interpolation, a second class of analytic approximation formulae. *Quart. Appl. Math.* **4**(2):112–141.

[95] J. M. Shapiro (1993). Embedded image coding using zerotrees of wavelet coefficients. *IEEE Trans. on Image Processing* **41**(12):3445–3462.

[96] M. J. Slattery, J. L. Mitchell (1998). The Qx-coder. *IBM Journal of Research and Development* **42**(6):767–784.

[97] O. G. Staadt, M. H. Gross, R. Weber (1997). Multiresolution compression and reconstruction. In *Proc. of IEEE Visualization'97*, pp. 337–346, Phoenix, AZ, USA.

[98] A. J. Stoddart, M. S. Baker (1998). Progressive splines. In *Proceedings of IMDSP'98 10th Image and Multi-dimensional Digital Signal Processing Workshop*, pp. 295–298, Alpbach, Austria.

[99] A. J. Stoddart, M. S. Baker (1998). Surface reconstruction and compression using multiresolution arbitrary topology G^1 continuous splines. In *Proceedings Fourteenth International Conference on Pattern Recognition*, vol. 1, pp. 788–791, Brisbane, Qld., Australia.

[100] Sun Microsystems (2000). *The Java 3D API Specification, version 1.2*. Palo Alto, CA, USA. `http://java.sun.com/products/java-media/3D/forDevelopers/J3D_1_2_API/`.

[101] A. Szymczak (2002). Optimized edgebreaker encoding for large and regular triangle meshes. In *Proc. of the Data Compression Conference (DCC), 2002*, p. 472, Snowbird, UT, USA.

[102] A. Szymczak (2002). *Optimized Edgebreaker Encoding for Large and Regular Triangular Meshes*. Tech. rep., Georgia Tech, GVU Center, Georgia, GA, USA. `http://www.cc.gatech.edu/fac/Andrzej.Szymczak/papers/prac.pdf`.

[103] A. Szymczak, D. King, J. Rossignac (2001). An Edgebreaker-based efficient compression scheme for regular meshes. *Computational Geometry* **20**(1-2):53–68.

[104] G. Taubin, A. Guéziec, W. Horn, F. Lazarus (1998). Progressive forest split compression. In *Proc. of the International Conference on Computer Graphics and Interactive Techniques, SIGGRAPH'98*, pp. 123–132, Orlando, FL, USA.

[105] G. Taubin, W. P. Horn, F. Lazarus, J. Rossignac (1998). Geometry coding and VRML. *Proceedings of the IEEE* **86**(6):1228–1243.

[106] G. Taubin, J. Rossignac (1998). Geometry compression through topological surgery. *ACM Transactions on Graphics* **17**(2):84–115.

[107] D. S. Taubman, M. W. Marcellin (2002). *JPEG2000: image compression fundamentals, standards and practice*. Kluwer Academic Publishers, Norwell, MA.

[108] W. Tiller (1992). Knot-removal algorithms for NURBS curves and surfaces. *Computer Aided Design* **24**(8):445–453.

[109] C. Touma, C. Gotsman (1998). Triangle mesh compression. In *Proc. of the 24th Conference on Graphics Interface (GI-98)*, pp. 26–34, Morgan Kaufmann, San Francisco, CA, USA.

[110] W. T. Tutte (1962). A census of planar triangulations. *Canadian Journal of Mathematics* **14**:21–38.

[111] K. J. Versprille (1975). *Computer-aided Design Applications of the Rational B-spline Approximation Form*. Ph.d. dissertation, Syracuse University, Syracuse, NY.

[112] G. K. Wallace (1992). The JPEG still picture compression standard. In *IEEE Trans. on Consumer Electronics*, vol. 38, pp. xviii – xxxiv.

[113] Web3D Consortium (2001). *ISO/IEC 14772-1:1997/Amd.1:2002 Information Processing Systems - Computer Graphics The Virtual Reality Modeling Language Part 1 — Functional specification and UTF-8 encoding Amendment 1 — Enhanced interoperability.*

[114] Web3D Consortium (2002). *Information technology – Computer graphics and image processing – eXtensible 3D (X3D) – Part 1: Architecture and Base Components, Final Working Draft.* `http://www.web3d.org/TaskGroups/x3d/specification-milestone4/index.html`.

[115] E. W. Weisstein (2003). Eric weisstein's world of mathematics. `http://mathworld.wolfram.com`.

[116] R. G. Winch (1993). *Telecommunication transmission systems: microwave, fiber optic, mobile cellular radio, data, and digital multiplexing*. McGraw-Hill.

[117] W. D. Withers (1997). A rapid entropy-coding algorithm. *Dr. Dobb's Journal* **22**(4):38–43. `ftp://www.pegasusimaging.com/pub/ELSCODER.PDF`.

[118] I. H. Witten, R. M. Neal, J. G. Cleary (1987). Arithmetic coding for data-compression. *Communications of the ACM* **30**(6):520–540.

[119] M. Woo, J. Neider, T. Davis, D. Shreiner (1999). *OpenGL Programming Guide*. Addison-Wesley, Reading, Massachusetts, 3rd edn.

[120] L. Zhou, C. Kambhamettu (2001). Extending superquadrics with exponent functions: Modeling and reconstruction. *Graphical Models* **63**(1):1–20.

[121] D. Zorin, P. Schröder, W. Sweldens (1996). Interpolation subdivision for meshes with arbitrary topology. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques, SIGGRAPH'96*, pp. 189–192, ACM Press, New Orleans, Louisiana.

Curriculum Vitæ

Name:	Diego Santa Cruz Ducci
Citizenship:	Chilean / Italian
Birthdate:	July 24, 1973
Birthplace:	Santiago, Chile
Marital status:	Married

Contact information

Address:	Rue Chaponnière 7 1201 Genève Switzerland
Phone:	+41 79 419 56 71
Email:	Diego.SantaCruz@epfl.ch

Work experience

- ***March 1998 – present***: research assistant, Swiss Federal Institute of Technology (EPFL), Lausanne, Switzerland
  - Development of image compression, watermarking and 3D model compression algorithms.
  - Project management: technical manager and lead developer of JJ2000 (JPEG 2000 reference software, 4 persons, 1999-2000) and NexImage (advanced image compression, 3 persons, 1998-1999).
  - Technical consultant: Swiss delegate to ISO JPEG 2000 standardization (1998-2000); development of region-of-interest functionality in JPEG 2000 (1998).
  - Teaching: JPEG 2000 instructor at IEEE's International Conference in Image Processing 2001; supervision of 3 full time (4 to 6 months) and 4 part time (4 months) student's projects; teaching assistant for image and video processing course (1 semester).
- ***May 1998 - December 1998***: IT consultant at Cambridge Technology Partners – NatSoft S.A. (CTP), Geneva, Switzerland.
  - Analysis and definition of sales and help desk processes (Bobst S.A. and Phillip Morris, Switzerland).
  - Analysis of a project planification, evaluation and tracking system (UNAIDS, Switzerland).
  - Analysis of a vehicle leasing system (GE Capital Fleet Services, Holland).

Education

- ***March 1998 – May 2003***: PhD student in electrical engineering, Swiss Federal Institute of Technology (EPFL), Lausanne, Switzerland.
- ***October 1992 – March 1997***: *Diplôme d'Ingénieur Electricien EPF*, equivalent of Master of Science in Electrical Engineering.
- ***August 1994 – May 1995***: exchange student at Carnegie Mellon University (CMU), Pittsburgh, PA, USA; exchange fellowship awarded by EPFL.
- ***October 1991 – June 1992***: *Cours de Mathématiques Spéciales*, Swiss Federal Institute of Technology (EPFL), Lausanne, Switzerland.
- ***January 1989 – December 1990***: Baccalaureate, specialization in Engineering, *Sagrado Corazón* College, Montevideo, Uruguay.

Awards

- VCIP 2002 Best Student Paper, Compression of parametric surfaces for efficient 3D model coding, Visual Communications and Image Processing (VCIP) 2002.

Skills

Languages

Spanish:	mother tongue
English:	fluent oral and written
French:	fluent oral and written
Italian:	basic oral

Computing

Operating systems:	Linux, Unix, Windows, MacOS
Programming languages:	C, C++, Java, Perl, Unix shell
Other:	HTML, LaTeX, Matlab, CVS

Publications

Journal papers

- Diego Santa-Cruz and Touradj Ebrahimi. Coding of 3D virtual objects with NURBS. Signal Processing, special issue on image and video coding beyond standards, pp. 1581–1593, vol. 82, no. 11, November 2002.
- Diego Santa-Cruz, Raphaël Grosbois and Touradj Ebrahimi. JPEG 2000 performance evaluation and assessment. Signal Processing: Image Communication, pp. 113–130, vol. 17, no. 1, January 2002.

Conference papers

- Nicolas Aspert, Diego Santa-Cruz and Touradj Ebrahimi. MESH: Measuring Errors between Surfaces using the Hausdorff distance. In Proc. of the IEEE International Conference in Multimedia and Expo (ICME) 2002, vol. 1, pp. 705–708, Lausanne, Switzerland, August 26–29, 2002.

- Diego Santa-Cruz and Touradj Ebrahimi. Compression of parametric surfaces for efficient 3D model coding. In Visual Communications and Image Processing (VCIP) 2002, Proc. of SPIE, vol. 4671, pp. 280–291, San Jose, CA, USA, January 21–23, 2002.

- Raphaël Grosbois, Diego Santa-Cruz and Touradj Ebrahimi. New approach to JPEG 2000 compliant region of interest coding. In SPIE's 46th annual meeting, Applications of Digital Image Processing XXIV , Proc. of SPIE, vol. 4472, pp. 267-275, San Diego, California, July 29–August 3, 2001.

- Franck Leprévost, Raphaël Erard, Touradj Ebrahimi, Martin Kutter and Diego Santa-Cruz. How to Bypass the Wassenaar Arrangement: A New Application for Watermarking. In Proc. on ACM multimedia 2000 workshops, pages 161–164, Los Angeles, CA, USA, October 30–November 3, 2000.

- Diego Santa-Cruz, Touradj Ebrahimi, Joel Askelöf, Mathias Larsson and Charilaos Christopoulos. JPEG 2000 still image coding versus other standards. In Proc. of the SPIE's 45th annual meeting, Applications of Digital Image Processing XXIII , vol. 4115, pp. 446–454, San Diego, California, July 30–August 4, 2000.

- Diego Santa-Cruz and Touradj Ebrahimi. An analytical study of JPEG 2000 functionalities. In Proc. of the IEEE International Conference on Image Processing (ICIP), vol. 2, pp. 49–52, Vancouver, Canada, September 10–13, 2000.

- Diego Santa-Cruz and Touradj Ebrahimi. A study of JPEG 2000 still image coding versus other standards. In Proc. of the X European Signal Processing Conference (EUSIPCO), vol. 2, pp. 673–676, Tampere, Finland, September 5–8, 2000.

- Diego Santa Cruz, Touradj Ebrahimi, Mathias Larsson, Joel Askelöf and Charilaos Cristopoulos. Region of Interest Coding in JPEG2000 for interactive client/server applications. In Proc. of the IEEE Third Workshop on Multimedia Signal Processing (MMSP), pp. 389–394, Copenhagen, Denmark, September 13–15, 1999.

- Maryline Charrier, Diego Santa Cruz and Mathias Larsson. JPEG2000, the next millenium compression standard for still images. In Proc. of the IEEE International Conference on Multimedia Computing and Systems (ICMCS), vol. 1, pp. 131–132, Florence, Italy, June 7–11, 1999.

Non refereed

- Diego Santa Cruz, Touradj Ebrahimi and Charilaos Cristopoulos. The JPEG 2000 Image Coding Standard. In Dr. Dobb's Journal, vol. 26, no. 4, pp. 46–54, April 2001.

Patents

- Partial retrieval of images in the compressed domain. Swedish pat. #9803593-4.
- Entropic encoding method and device. US pat. Application # 09/796455, French pat. application #00.02700.