Grupo de Tratamiento de Imágenes
Dpto. de Señales, Sistemas y Radiocomunicaciones
E.T.S.I Telecomunicación
UPM‡

Institut de Traitement des Signaux
Département d'Electricité
Faculté des Sciences et Techniques de l'Ingénieur
EPFL†

Implementation of MPEG-4's Subdivision Surfaces Tools

Yannick Maret

Tutors

Prof. Touradj Ebrahimi†
Prof. Francisco Morán‡
Prof. Narciso García‡

February 2003

Objectives

The MPEG (Moving Picture Experts Group) of the ISO (International Organisation for Standardisation) has produced several standards so far, of which MPEG-1 and MPEG-2 deal with the efficient coding of video (including any associated audio) and MPEG-4 (ISO/IEC 14496) with that of interactive, multimedia content. All these standards are divided into several 'parts', namely video, audio, systems, etc. Unlike in MPEG-1 and MPEG-2, in MPEG-4 a RS (Reference Software) is developed to demonstrate how a decoder could be implemented, and is released as one of the Parts of the standard after an extensive testing for maturity — but not necessarily for efficiency! At the beginning of 2003, ISO will release MPEG-4 Part-16: AFX (Animation Framework eXtension), which contains a set of tools produced by the SNHC (Synthetic/Natural Hybrid Coding) subgroup. Among these tools, higher order approximations to 3D surfaces, such as NURBSs (Non-Uniform Rational B-Splines) and SSs (Subdivision Surfaces), have deserved special attention, as they allow to compactly represent piecewise smooth 3D shapes, which are the ones most frequently encountered in practice.

The objectives of this Project, which Yannick Maret will work on at the GTI-UPM: *Grupo de Tratamiento de Imágenes - Universidad Politécnica de Madrid,* with Francisco Morán and Narciso García as his advisors, are the following:

- To gain a general understanding on 3D surface modelling paradigms, and a more specific one in what concerns SSs.
- To implement (in C or C++) a software decoder for the two kinds of SSs considered in the AFX toolset: "plain" SSs and wavelet/detailed SSs. This decoder will evidently be inspired in the one which deals with SSs inside MPEG-4's current RS, but will hopefully be simpler and more efficient, yet integrated seamlessly with the rest of the RS.
- To design a "demo" to illustrate how the (possibly view-dependent) hierarchical transmission of a 3D shape coded thanks to SSs compares to the progressive 3D mesh transmission considered in MPEG-4 version 2.
- If time permits, to propose improvements on AFX's SSs-based tools, which could be taken into account for their inclusion in future versions of MPEG-4.

Abstract

This work is about the implementation of a MPEG-4 decoder for subdivision surfaces, which are powerful 3D paradigms allowing to compactly represent piecewise smooth surfaces. This 'study' will take place in the framework of MPEG-4 AFX, the extension of the MPEG-4 standard including the subdivision surfaces. This document will introduce, with some details, the 'theory' of subdivision surfaces in the two forms present in MPEG-4: 'plain' and detailed/wavelet subdivision surfaces. It will particularly concentrate on wavelet subdivision surfaces, which permit progressive 3D mesh compression.

The first chapter begins with an overview of MPEG-4, more particulary of MPEG-4 Visual. Then, the AFX extension and the role of subdivision surfaces within it are introduced. The second chapter presents the mathematical representation of discrete surfaces and some associated properties, enabling a 'smoother' explanation of subdivision surfaces.

The third chapter introduces 'plain' subdivision surfaces. Some of their properties, such as hierarchical relationships or convergence, are also explored. Ultimately, an example (based on Loop scheme) and the MPEG-4 implementation are discussed.

The fourth chapter describes wavelet subdivision surfaces. It also gives a general overview of wavelet theory and its relation with subdivision surfaces. In continuation, a brief summary of different 3D mesh compression methods (both progressive or not) is done. A section then explains, more adequately, wavelet details compression through a hierarchical tree partitioning algorithm. It permits to introduce MPEG-4 wavelet subdivision surfaces implementation, and to propose the addition of a new (and simpler) AFX tool for progressive 3D mesh compression (without view-dependency, and correcting some discrepancies of the current tool concerning progressive transmission).

A fifth chapter presents the data structures and the program general structures. It also describes some algorithms used in the actual implementation.

The last chapter is a summary of the work done. It also states some of the future work that could be done in this domain, and ends with a short conclusion.

Contents

Notations

$\mathcal{S}$	represents a set;
$\{s_i \in \mathcal{S}\}_i$	explicits an unordered set;
$(s_i \in \mathcal{S})_i$	explicits an ordered set, a sequence or a vector;
$[s_i \in \mathcal{S}]_i$	explicits a circular sequence, that is the last element of set is 'followed' by the first one and vice et versa;
$\lvert\mathcal{S}\rvert$	gives the cardinal of a set;
$\mathbf{p}$	represents a N-D point or vector;
$\mathbf{M}$	represents a matrix.

Chapter 1

MPEG-4

This chapter presents the MPEG-4 standard and the role of AFX[1] and subdivision surfaces within it. First, an overview of MPEG-4 followed by a more detailed description of the visual part is given. Then, AFX is presented and finally, the subdivision surfaces are introduced. This chapter is highly inspired by the following papers [6][11][9], which treat each particular point in more details.

1.1 Introduction

MPEG-4 is an ISO/IEC standard developed by MPEG (Moving Picture Experts Group), the committee that also developed the standards known as MPEG-1 and MPEG-2. These standards made interactive video on CD-ROM, DVD and Digital Television possible. MPEG-4 is the result of another international effort involving hundreds of researchers and engineers from all over the world. MPEG-4 (formally: "ISO/IEC 14496") was finalised in October 1998 and became an *international standard* in the first months of 1999. The fully backward compatible extensions under the title of MPEG-4 'version 2' were frozen at the end of 1999, to acquire the formal *international standard status* early in 2000. Several extensions were added since and work on some specific items are still in progress.

One important thing to mention about MPEG standards, is that they address the decoder side of the application and never the encoder, which enables companies to develop their own competitive compression engines.

MPEG-4 was created using the framework of *interactive* applications: graphics/audio, multimedia or digital TV. It is a set of compression formats and streaming technologies for multimedia content, which can operate over a large spectrum of bitrates. Since its beginning, it was created to address the following issues: interoperability, transport layer independence, decompression of rich media, interactivity, scalability and profiles. MPEG-4 goals are threefold:

- for the *authors*: multimedia contents with reusability, content protection;
- for the *service provider*: transparency for the network, quality of service;

[1] extension (also know as "part-16") of the standard containing, among others enhancements, the subdivision surfaces

- for the *end-user*: higher level of interaction with the content.

They are achieved by providing tools for:

- compact representation of visual/audio, audiovisual objects (natural or synthetic);
- composition of these objects into scenes;
- multiplexing and synchronisation of data.

MPEG-4 is thus divided in several parts (corresponding to the above toolboxes): systems (describing the two last points), visual and audio (covering the first one). Each of them describes a great number of tools, but in order to allow effective implementations of the standard, subsets of MPEG-4 toolboxes have been identified. These subsets are called *profiles* and limit the "tools set" a decoder has to implement for a specific application. Additionally, one or more *levels* have been set for each of these profiles, restricting the computational complexity.

Each version of MPEG-4 is backward compatible with precedent versions. That is, existing tools and profiles from any version are never replaced in subsequent versions; technology is always *added* to MPEG-4 in the form of new profiles. There also exist extensions adding functionalities the original MPEG-4 did not have, or enhancing existing ones. There are currently different ones already completed[2] or underway:

- IPMP extension (access control);
- the Animation Framework eXtension, AFX;
- Multi User Worlds;
- Audio extensions.

1.2 Visual Standard

In MPEG-4, visual objects can either be of natural or synthetic origin. The tools for objects of natural origin are first described. Then, the ones of synthetic origin (which AFX is part of) are presented, with a little more detail.

1.2.1 Natural Textures, Images and Video

The tools for representing natural video in the MPEG-4 visual standard provide standardised core technologies allowing efficient storage, transmission and manipulation of textures, images and video data for multimedia environments. These tools allow the decoding and representation of atomic units of image and video content, called "video objects". An example of a video object could be a talking person (without background and sound), which can then be composed with other audio-visual objects to create a scene.

1.2.2 Coding of Textures and Still Images

Efficient Coding of visual textures and still images (e.g. to be mapped on animated meshes) is supported by the visual texture mode of the MPEG-4. This

[2] the last three of the list

mode is based on a zerotree wavelet algorithm that provides very high coding efficiency over a very wide range of bitrates. It also provides spatial, quality scalabilities and arbitrary-shaped object coding. The wavelet formulation provides scalable bitstream coding in the form of an image resolution pyramid for progressive transmission and temporal enhancement of still images.

1.2.3 Synthetic Objects

Like those of other computer graphics specifications, MPEG-4 synthetic objects are organised in a scene graph based on VRML97 [8], which is a direct acyclic tree where nodes represent objects and branches their properties, called fields. As each object can receive and emit events, two branches can be connected by the means of a route, which propagates events from one field of one node to another field of another node. As any other MPEG-4 media, scenes may receive updates from a server that modify the topology of the scene graph. MPEG-4 supports the following visual synthetic objects:

- parametric descriptions of synthetic face and body;
- static and dynamic mesh coding with texture mapping;
- texture coding for view dependent applications.

Mesh Coding Standards

MPEG-4 capabilities for mesh coding include:

- efficient generic coding of 3D polygonal meshes. The coded representation is generic enough to support both manifold and non-manifold meshes;
- incremental representation enables a decoder to reconstruct a number of faces in a mesh proportional to the number of bits in the bit stream that have been processed;
- error resilience enables a decoder to partially recover a mesh when subsets of the bitstream are missing and/or corrupted;
- level of detail scalability enables a decoder to reconstruct a simplified version of the original mesh containing a reduced number of vertices from a subset of the bit stream. Such simplified representations are useful to reduce the rendering time of objects which are distant from the viewer. It also enables less powerful rendering engines to render the object at a reduced quality.

View-Dependent Scalability

The view-dependent scalability enables to stream texture maps, which are used in realistic virtual environments. It consists in taking into account the viewing position in the 3D virtual world in order to transmit only the most visible information. Only a fraction of the information is then sent, depending on object geometry and viewpoint displacement. This fraction is computed both at the encoder and decoder side. This approach greatly reduces the amount of transmitted information between a remote database and a user, given that a back-channel is available. This scalability can be applied both with DCT (Discrete Cosine Transform) and wavelet-based encoders.

Note that the view dependent scalability of 'basic' MPEG-4 is limited to mesh textures. The AFX standard addresses this issue by providing tools like wavelet subdivision surfaces, which permit to have view dependency at mesh geometry level.

1.3 The Animation Framework eXtension, AFX

The Animation Framework extension (AFX - pronounced 'effects') provides an integrated toolbox for building attractive and powerful synthetic MPEG-4 environments. The framework defines a collection of interoperable tool categories that collaborate to produce a reusable architecture for interactive animated contents. In the context of AFX, a tool represents functionality such as a BIFS node, a synthetic stream, or an audio-visual stream.

AFX utilises and enhances existing MPEG-4 tools, while keeping backward-compatibility, by offering:

- higher-level descriptions of animations (e.g. inverse kinematics);
- enhanced rendering (e.g. multi-texturing, procedural texturing);
- compact representations (e.g. piecewise curve interpolators, subdivision surfaces);
- low bitrates animations (e.g. using interpolator compression);
- scalability based on terminal capabilities (e.g. parametric surfaces tessellation);
- interactivity at user level, scene level, and client-server session level;
- compression of representations for static and dynamic tools.

Compression of animated paths and animated models is required for improving the transmission and storage efficiency of representations for dynamic and static tools.

1.3.1 Subdivision Surfaces in MPEG-4

Why Subdivision Surfaces?

The MPEG-4 standard [10] already included in its version 2 tools for the efficient coding of 3D surfaces, but they were based in the simplest approximation of a surface: the one resulting from tiling it with planar facets. The problem with linear approximation such as these is that hundreds of thousands of elements (vertices, edges, facets) are easily needed to obtain a reasonable approximation accuracy. Moreover, the editing or animation of a polygonal mesh is cumbersome, because its vertices are semantically unrelated, and must therefore be moved individually. This is why most CAD (Computer-Aided Design) and 3D modelling commercial applications still use curved patches of the NURBS (Non-Uniform, Rational, B-Spline) family for tiling 3D surfaces. Patches provide a compact, convenient method to generate piecewise smooth, higher-order surfaces from relatively few control points, whose movement deforms locally the surface. However, patch control grids must be perfectly regular, so modelling objects of arbitrary topology with NURBS introduces non-trivial patch-stitching and curve-trimming difficulties. Furthermore, when modelling intri-

cate surfaces, a large number of tiles (either curved or planar) are needed to describe high-frequency regions, so patches may not provide a much more compact solution than polygons.

Subdivision surfaces [24][5] establish a bridge between polygons and patches, as subdivision schemes define simple and efficient mechanisms to derive a smooth surface from an initial control polyhedron of arbitrary topology. Indeed, Subdivision surfaces are defined as the limit of a refinement process of both the connectivity and the geometry of a planar mesh which recursively splits each of its elements (usually its facets) into several ones. If this process is carefully designed, the mesh tends to a limit surface as smooth as NURBS-based ones. Subdivision is extremely useful for approximating and manipulating a surface at different levels of detail. Subdivision surfaces define inherently a multiresolution model of the limit 3D surface, as the vertex positions at different subdivision levels are (locally) hierarchically related. It is thus easy to perform large-scale edits, in which the movement of a few vertices of a coarse control mesh drags a wide area of the surface, as well as minute detail modifications, in which only a few vertices of the finest meshes are displaced.

Subdivision Surfaces in MPEG-4

Subdivision surfaces in the AFX toolbox, released as "part-16" of the MPEG-4 standard at the beginning of 2003, come in two flavours, depending on whether the positions of the new vertices appearing after each subdivision step may be modified or not before splitting the mesh again. In both cases, a piecewise smooth limit surface is obtained by subdividing the initial control mesh. But in the "basic subdivision surfaces setting", that limit surface is completely defined by the initial control mesh, which is simply smoothed, whereas in the "detailed/wavelet subdivision surfaces scenario", the aim is to approximate a particular target surface with an increasing accuracy thanks to the 3D details added to the predicted new vertices at each step of the subdivision process.

The MPEG-4 subdivision surfaces tools will be presented in more details in chapters 3 and 4.

Chapter 2

Discrete Surfaces

This chapter introduces the theoretical setup of the subdivision surfaces. The first section presents elementary concepts in discrete 3D models representation, which is followed by some definitions that will be mainly used during the algorithms presentation. For more information, one can refer to [4] (computer graphics) or [16] (algebraic topology).

2.1 Introduction

Many 3D shapes can be modelled by a (piecewise) smooth surface and can be approximated, in its simplest form, by a planar faceted *mesh*, or *discrete surface*. A mesh is described by two distinct settings: its *topology* and its *geometry*. The first one represents an *abstract graph* onto which the second[1] is mapped. Note that with the same topology, one can describe different surfaces as shown in figure 2.1. It means that the topology is not directly linked to the surface but is simply a means to describe the relationship between the 3D points forming the geometry.

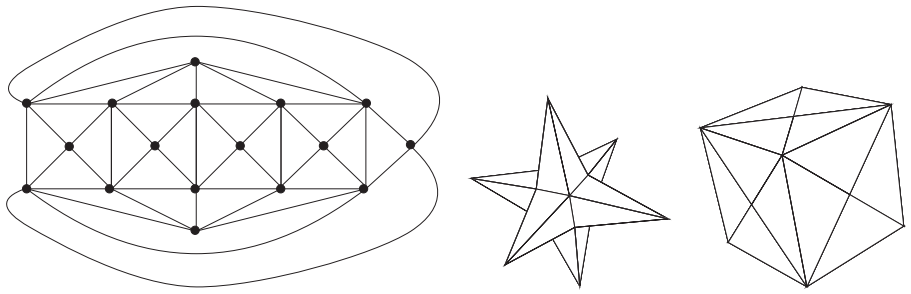

Figure 2.1: *An abstract graph (a) and different geometry mappings, (b) and (c).* While the abstract graph is the same, the two models represented are quite different!

[1] points in the 3D space

2.2 Surfaces Representation

A discrete surface $\mathcal{S}$ can be expressed as an ordered set of three components:

$$\mathcal{S} = (\mathcal{V}, \mathcal{E}, \mathcal{P}) \tag{2.1}$$

Where:

- $\mathcal{V}$ is a set of vertices $i = 0, \ldots, N-1$ where N is the total number of vertices;
- $\mathcal{E}$ is a set of vertices couple $\{i, j\} \in \mathcal{V}^2$, signifying that the vertices i and j are connected together, thus forming an *edge*;
- $\mathcal{P}$ is set of points $\mathbf{p}_i \in \mathbb{R}^3$ linked to each vertex $i \in \mathcal{V}$.

The model geometry is all included in the set $\mathcal{P}$, while the model abstract graph $G(\mathcal{V}, \mathcal{E})$ is represented by the two sets $\mathcal{V}$ and $\mathcal{E}$. There are conditions on $G(\mathcal{V}, \mathcal{E})$ to be actually an abstract graph, which are shown in figure 2.2 and listed below:

- there are no loops and the graph is simple;
- no 'dangling' edges are allowed[2].

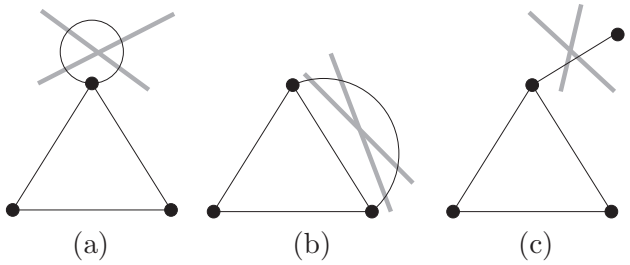

Figure 2.2: *Conditions to be an abstract graph.* That is, no loop (a), simple (b) and no dangling edge (c).

2.3 Definitions and Properties

2.3.1 Facets

A *facet* or *polygon* $\mathcal{P}_j$ is a circuit of $G(\mathcal{V}, \mathcal{E})$ that does not enclose any vertex. It can be represented by a circular sequence of vertices $v_k \in \mathcal{V}$ and can be written as:

$$\mathcal{P}_j = [v_k \mid (v_{k-1}, v_k) \in \mathcal{E}]_k \,. \tag{2.2}$$

For example the facet **0** of the abstract graph shown in figure 2.4 can be written as $\mathcal{P}_0 = [5, 6, 7]$. The set $\{\mathcal{P}_j\}_j$ (or simply $\mathcal{P}$) contains all the facets of the graph $G(\mathcal{V}, \mathcal{E})$.

Note that not all circuits of an abstract graph are necessarily a facet, since one can want a mesh with 'holes'.

[2] i.e each edge must belong to a facet at least (in fact, at most two if the mesh is manifold)

Theoretically one can use any kind of facets to represent a shape. Practically, things are quite different and most meshes have an abstract graph composed either with *triangles* (by far, the most common occurrence) or *quadrilaterals*.

There are reasons for this seemingly poverty of choices. First of all, triangles vertices lie always on the same plane (which is not necessarily the case for facets with more than 3 vertices), which favours greatly the use of triangles. Then, most hardware graphic units only process triangles (which enables them to greatly simplify their logical units, thus reducing cost), which is not too restrictive because any other facets can be triangulated and thus be represented by a set of triangles.

A great advantage of quadrilaterals over triangles is that they permit to better capture the symmetry of human made objects (e.g. a cube) as exemplified in figure 2.3. However, the rest of this document will mostly treat about triangular meshes, although some comments will be made about quadrilateral meshes...

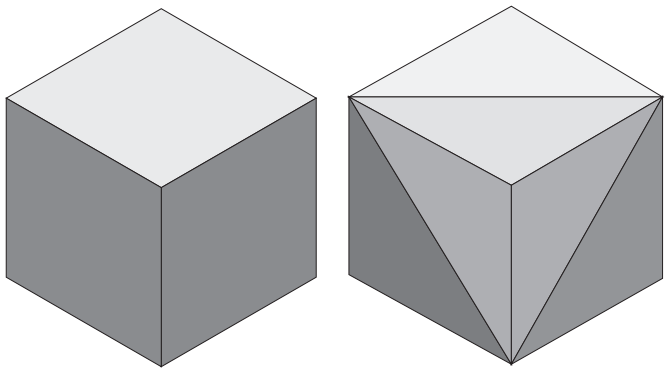

Figure 2.3: *Triangular meshes against quadrilateral meshes.* Figure (a) shows a cube mapped on a quadrilateral abstract graph, while figure (b) presents the same cube geometry but mapped on triangular abstract graph. Both have been rendered in "facet shading mode", and the difference of quality is quite clear. Although by using more sophisticated shading schemes it could improve, the triangular mesh will never be as good as the quadrilateral one.

2.3.2 1-Ring

The *1-Ring* $\mathcal{R}_i$ associated to the vertex i is given by the set:

$$\mathcal{R}_i = \{r_k \mid (i, r_k) \in \mathcal{E}\}_k \,. \tag{2.3}$$

A 1-Ring can be ordered if each facet incident to the vertex i has one or two neighbouring facet(s) incident to the vertex i. In the *ordered 1-Ring*, each neighbour of r_k, as in the above definition, belongs to the same facet. The ordered 1-Ring can be represented by a circular sequence (for interior vertices) or by an ordered set (for boundary vertices). An interesting property of the ordered 1-Ring is that for an interior vertex the first and last elements also

belong to the same facet, thus allowing 'easy' boundaries detection. Figure 2.4 shows the 1-Ring (b) associated with the vertex 5 of the abstract graph (a), the ordered 1-Ring can be written as $\mathcal{R}_5 = [0, 6, 7, 8, 4, 2]$.

2.3.3 Valence and Crease Degree

The *valence* of a vertex i is the number of elements in its associated 1-Ring, i.e. $|\mathcal{R}_i|$.

One can also define the *crease degree* of a vertex, which is the number of facets in the 1-Ring. For an interior vertex, the crease degree and the valence are equal, but for a border vertex the crease degree is one unit less than the valence.

2.3.4 Interior and Boundary Edges

An edge belonging to one and only one facet is said to be a *boundary*, while one shared by two facets is labelled *interior*. A mesh without any border edge is called *closed* and one with at least one border edge is said to be *open*. For example the meshes in figure 2.1 are closed while the one in figure 2.7 is open.

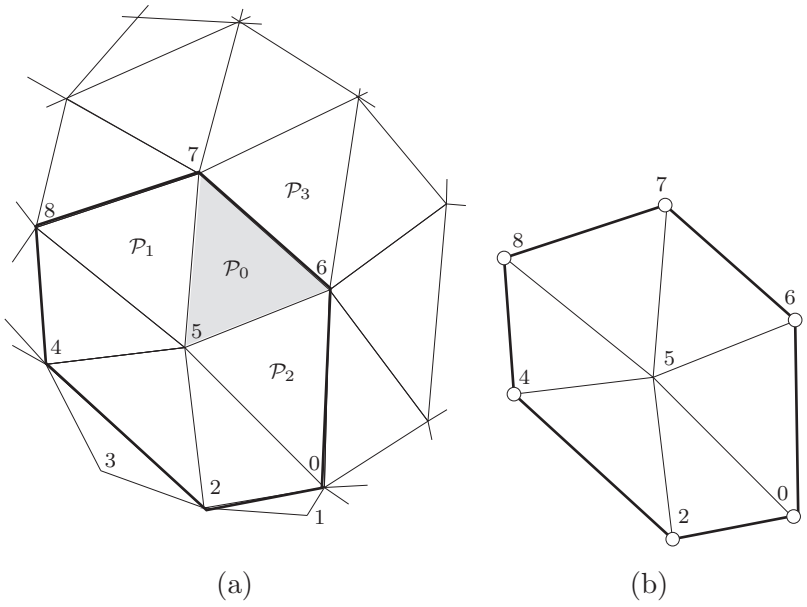

Figure 2.4: *Example of 1-ring.* Figure (a) shows and abstract graph while figure (b) shows the 1-ring associated with vertex 5.

2.3.5 Extraordinary and Regular Vertices

A vertex can be *regular* or *extraordinary* depending on its valence. This denomination comes from the regular tiling of a plane (cf. figure 2.5) by an abstract

planar graph. Regular tiling with triangles gives a valence of 6 for interior vertices and 4 for border ones; while with quadrilaterals theses values are of 4 and 3, respectively.

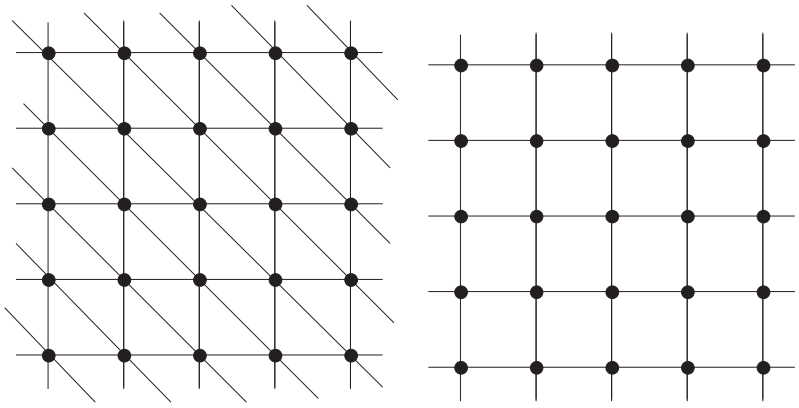

Figure 2.5: *Tiling of the plane.* Figure (a) shows the regular tiling of the plane with triangles while the figure (b) shows the same but with quadrilaterals.

2.3.6 Manifoldness

A mesh can be *manifold* or not, this propriety is very important and summarised as follows (and illustrated on figure 2.6):

- a manifold mesh cannot have edges shared by more than two facets;
- and every 1-ring of the mesh can be ordered.

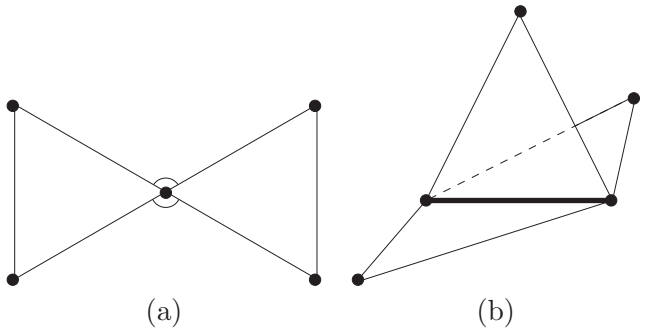

Figure 2.6: *Non-manifold meshes.* Figure (a) shows a non-manifold vertex (the 1-Ring cannot be ordered), while figure (b) presents a non-manifold edge (edge shared by more than two facets).

2.3.7 Orientability

A discrete surface $\mathcal{S}$ is said to be orientable if the normals associated with the facets can be defined consistently. That is, if one can 'walk' counter-clock wisely

on the boundaries (for open mesh) of $\mathcal{S}$ and constantly having the surface on his left. This process can also be extended to the facets of the discrete surface. It is the reason why vertices are usually ordered in a counter-clock wise manner when listing them to describe a facet or a ordered 1-ring. For example, a Moëbius ring is not orientable as it can be seen on figure 2.7.

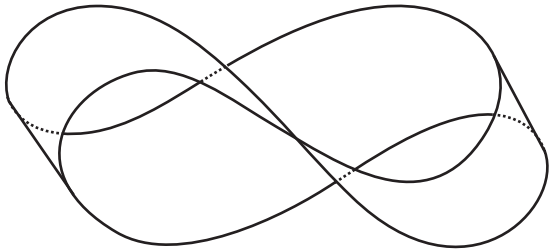

Figure 2.7: *The Moëbius ring* (ring with only one face!) is a non-orientable surface.

2.4 Mesh Computer Representation

One way to store the abstract graph of mesh is to use the *indexed faces set* method. In this method, the graph is stored facet by facet[3], by listing (generally counter-clock wisely) the vertices of each facet. Each vertex v is represented by a number[4], which points to the corresponding coordinate triple in a coordinate list. This representation is generic enough to permit coding meshes that are:

- manifold or not;
- open or closed;
- made of different types of facets (providing the facets are separated by a 'marker', e.g. -1).

[3]which makes sense since 3D rendering is usually done facet by facet

[4]a positive integer ranging from 0 to $N-1$ where N is the total number of vertices

Chapter 3

'Plain' Subdivision Surfaces

The first part of this chapter furthers the introduction of subdivision surfaces from chapter 1. It is followed by a section detailing 'plain' subdivision surfaces and some of it properties. Finally, the corresponding MPEG-4 tool is presented.

The presentation of 'plain' subdivision surfaces is done through an example, the method chosen is Loop's scheme [13], which is an approximating scheme (thus permitting to present both edge and vertex rules). To have an idea of how interpolating schemes work, one can refer to the well-known butterfly scheme [2]. A theoretical analysis of the limit surface is first presented, followed by an extension of the 'basic' Loop's scheme able to produce piecewise smooth surfaces. Finally, MPEG-4 subdivision surfaces are introduced.

While many concepts introduced here are not really necessary for the implementation of the MPEG-4 subdivision surfaces tools, they describe a few key points that help us understand the ideas behind subdivision surfaces.

3.1 Introduction

The basic idea behind subdivision surfaces is to recursively refine both the topology and geometry of a discrete surface towards obtaining a desired limit smooth surface (no more discrete). From the *base mesh* $\mathcal{S}^0$ (also called *initial control hull*), a sequence of finer meshes $\mathcal{S}^i = \left(\mathcal{V}^i, \mathcal{E}^i, \mathcal{P}^i\right)$ is obtained. The (piecewise) smooth surface is the limit of the refinement process and is denoted $\mathcal{S}^\infty$. Figure 3.1 shows an example of subdivision surfaces.

'Plain' subdivision surfaces permit a designer to easily produce a visually pleasing[1] model. First, an initial mesh is manually created with associated tags 'describing' the subdivision methods of its different pieces. The designer can control the results directly, modify the initial mesh and tags accordingly to the limit surface she/he aims at. To retrieve the limit surface, one needs: the base mesh, the tags, and the minimal number of subdivision steps to apply[2]. Note that 'plain' subdivision surfaces were developed in a *geometry framework*.

[1] generally it means smooth, hence a high number of polygons

[2] for obtaining a visually 'pleasing' approximation of the limit surface

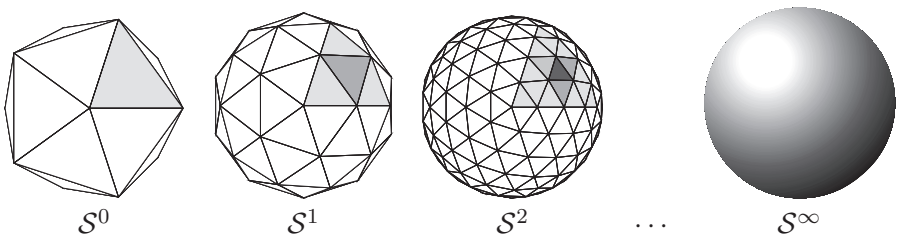

Figure 3.1: *Modelling of a sphere with 'plain' subdivision surfaces.* The left mesh is the initial control hull, the two first subdivision steps are presented in the centre and the limit smooth surface (the targeted sphere) is shown on the right.

3.2 Subdivision Surfaces: 'Modus Operandi'

All subdivision methods can be decomposed in two steps. The *splitting step*, making the topology richer, is common to all (primal) methods. They mainly differ only in the *refinement step*, where the vertex positions are modified. The next two subsections present these two steps for for primal schemes in some details. Dual schemes will not be furthered more than the example shown in figure 3.2 as they are far less common than primal ones, though they are conceptually very interesting and hold some very interesting properties such as an inherent vertices hierarchy, for more information, one can refer to [25].

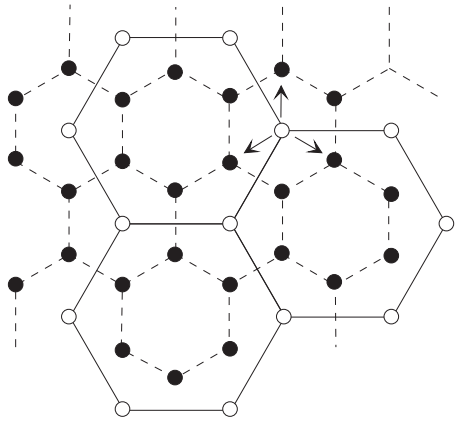

Figure 3.2: *Hexagonal dual scheme.* This figure shows a possible hexagonal dual scheme. In dual scheme, new facets are created around old vertices. Note the arrows, showing that each old vertices 'produces' three news ones during the subdivision step.

3.2.1 Facets Splitting Step

In this step, the abstract graph is made richer by splitting each polygon into several others. There are various ways of proceeding, the most common being

the *midpoint splitting* where each edge is split in two by the adjunction of a new vertex at its midpoint. On the nomenclature side, the newly created vertices are labelled as *odd* while the old ones (from the previous subdivision step) are called *even*. Examples of splitting step for triangular and quadrilaterals meshes are shown if figures 3.3 and 3.4, respectively.

In the case of triangular abstract graphs, the three new vertices created by splitting the edges of a triangle are connected together to form *four* new triangles (as shown in figure 3.3). The number of vertices at a given step i is then given by the simple relation:

$$\left|\mathcal{V}^i\right| = \left|\mathcal{V}^{i-1}\right| + \left|\mathcal{E}^{i-1}\right|, \tag{3.1}$$

and similarly the number of faces is:

$$\left|\mathcal{P}^i\right| = 4^i \cdot \left|\mathcal{P}^0\right|. \tag{3.2}$$

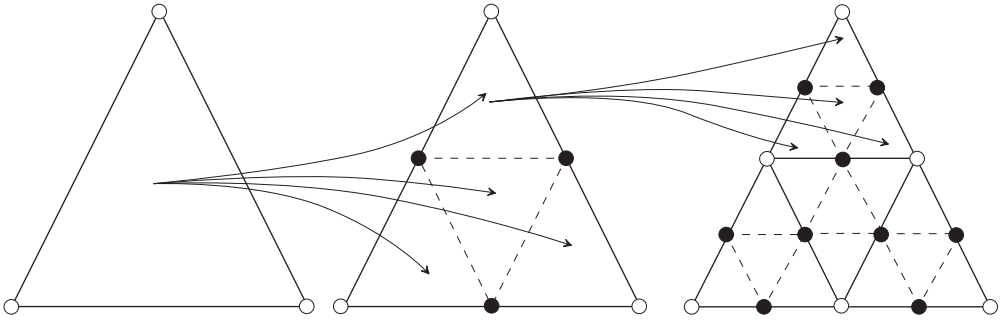

Figure 3.3: *Primal Subdivision Abstract graphs splitting (triangles).* This figure shows a triangular abstract graph splitting. White circles indicate vertices from the precedent step, and the ones splitting edges are marked by a black dot. Note that facets are split in primal schemes, while it is vertices in dual ones.

Now, regarding quadrilateral abstract graphs: another new vertex is added inside the quadrilateral, to which the new edge vertices are connected in order to form *four* new quadrilaterals (as shown in figure 3.4). The number of vertices at a given step i is then given by the simple relation:

$$\left|\mathcal{V}^i\right| = \left|\mathcal{V}^{i-1}\right| + \left|\mathcal{E}^{i-1}\right| + \left|\mathcal{P}^{i-1}\right|, \tag{3.3}$$

and the number of faces is given by the same relation as for triangular meshes.

A very interesting property of the splitting step is that all added vertices are regular. It means that original extraordinary vertices become more and more isolated in an otherwise regular meshing, as it can be seen in figures 3.3 and 3.4 (the original extraordinary vertices are separated by a growing number of regular ones). Note that the splitting step only operates on the sets $\mathcal{V}$ and $\mathcal{E}$, the topology of the model.

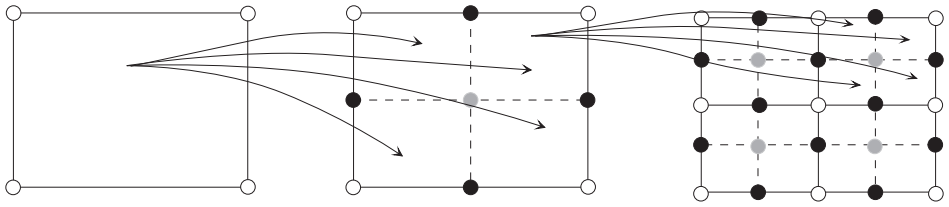

Figure 3.4: *Primal Subdivision Abstract graphs splitting (quadrilaterals).* This figure shows a quadrilaterals abstract graph splitting. White circles indicate vertices from the precedent step, the ones splitting edges are marked by a black dot, the ones splitting facets are shown in grey. Note that facets are split in primal schemes, while it is vertices in dual ones.

3.2.2 Refinement Step

The refinement step aims at smoothing the surface (i.e. to reduce the angle between adjacent facets) and only operates on the set $\mathcal{P}$, that is, the geometry of the model. There are two principal schemes in the refinement step: *interpolating* and *approximating*. The main difference between them is that in interpolating schemes, even vertex positions remain fixed while they are moved in approximating ones.

In plain subdivision surfaces, there are two different sets of rules: *vertex rules* and *edge rules*. The first ones modify even vertex positions (in approximating schemes) while the second give odd vertex positions. The positions are given using weighted sums of neighbouring vertex positions at the precedent level of subdivision, through the help of weighting masks named *stencils*: an example for Loop's method (which is approximating) is given in figure 3.5.

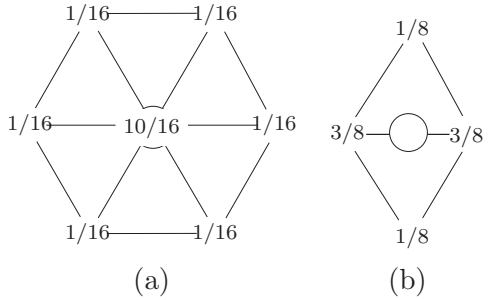

Figure 3.5: *Loop's subdivision stencils.* Figure (a) shows the stencil for regular interior vertices and figure (b) the one for interior edges.

3.2.3 Complexity

It is important to realise that subdivision is an exponential process, in terms of computational complexity and memory requirements, since the number of polygons is multiplied by four at each subdivision step (as shown by equation

3.2). Fortunately, few steps are usually needed to obtain a visually smooth surface, while an 'infinity' of subdivision step are actually required to obtain the real smooth surface.

3.3 Hierarchical Relationships in Primal Subdivisions Surfaces

Let us first define the terms *offspring* and *descent*. The offsprings of an element c of a hierarchical set $\mathcal{H}$, are elements $c_i \in \mathcal{H}$ directly 'issued' from p (like a parent-children relationship). The descent of an element c is the transitive closure of the relation "is offspring of", that is: a set recursively including all the offsprings. Figure 3.6 gives an example of hierarchical set, for instance, a has three offsprings: b, c and d; and ten descendants: b to k.

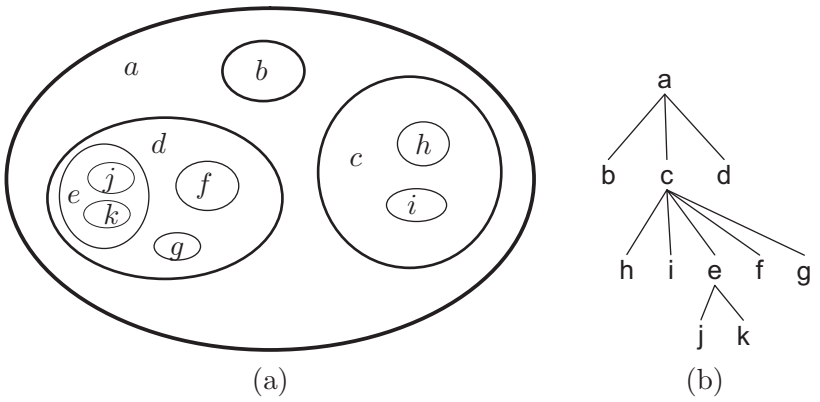

Figure 3.6: *Hierarchical Set*. A hierarchical set can be seen in (a), and the corresponding tree is presented in (b).

As shown in figures 3.1 (highlighted triangles) and 3.3, subdivision surfaces inherently exhibit a *facets-based hierarchy* (for primal schemes). This hierarchy is organised like a *forest* of *trees*, with each base mesh facet being the root of one tree and the last subdivision level facets being their leaves! In a facets tree, every node has exactly 4 offsprings (except for the leaves, which have none).

Moreover, subdivision surfaces also define an *edges-based hierarchy* (at least for triangular and quadrilateral meshes), as shown in [12]. In the subdivision of the two adjacent triangles shown in figure 3.7(a,b), it is quite evident that the two edges ('0' and '1'), resulting from the splitting of the highlighted edge in figure 3.7(a), are its offsprings. What is less evident is that the two other edges ('2' and '3') are also part of its offsprings; the reason being that, locally, they are the only ones being topologically 'akin' to the edge in figure 3.7(a). Note with this definition, each edge in figure 3.7(b) "is offspring" of one and only one edge on figure 3.7(a). With some precaution, this offsprings definition can be extended for quadrilateral meshes, as shown in figure 3.7(c,d). As for the facets-based hierarchy, the edges-based hierarchy is also organised like a forest of trees, with each base mesh edge being the root of one tree and the

last subdivision level edges being their leaves! In an edges tree, a node (that is not a leaf) could have 3 or 4 offsprings, depending on wether the corresponding edge lies on a boundary or not (respectively).

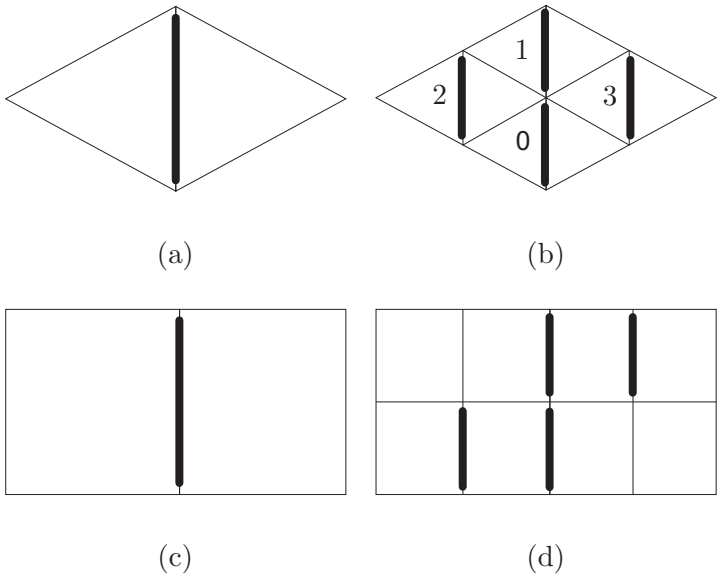

Figure 3.7: *Edges hierarchy.* This figure shows the hierarchical relationship between edges in triangular (a,b) and quadrilateral meshes (c,d).

It is possible to define a *'vertices'-based hierarchy* by remarking that odd vertices at subdivision level $i+1$ are one to one onto edges at subdivision level i (for triangular meshes). And since edges, at different subdivision level, are hierarchically related (as shown above), so are the odd vertices! But it is important to note that this vertices-based hierarchy does not take in account at all base mesh vertices: while the vertices hierarchy is organised like a *forest* of *trees*, the trees roots are the odd vertices issued from the first subdivision step. The last subdivision level odd vertices are the tree leaves. This hierarchy will be used for the "set partitioning sorting algorithm" as shown in [12] and section 4.4.

3.4 Convergence

One way to analyse the existence and the tangent plane continuity of the limit surface is to write the subdivision rules into *local subdivision matrices*. This subsection will only present the method for regular vertices, but for irregular ones the idea is the same (if a bit more involved).

Let $\mathbf{P}^i$ be a column vector as $\mathbf{P}^i = \begin{pmatrix} \mathbf{p}^i & \mathbf{p}^i_0 & \mathbf{p}^i_1 & \dots & \mathbf{p}^i_5 \end{pmatrix}^T$. Where $\mathbf{p}^i$ is the position of a given vertex during the subdivision step i and the $\mathbf{p}^i_j$ are the positions of the vertices contained in its corresponding 1-ring $\mathcal{R}^i$.

It is then possible to represent the evolution of the vector $\mathbf{P}^i$ (as shown in

the figure 3.8) in a matrix form. After one subdivision step[3]:

$$\mathbf{P}^{i+1} \quad = \quad \mathbf{S} \cdot \mathbf{P}^{i}, \tag{3.4}$$

where (for Loop's scheme):

$$\mathbf{S} \quad = \quad \frac{1}{16} \cdot \begin{pmatrix} 10 & 1 & 1 & 1 & 1 & 1 & 1 \\ 6 & 6 & 2 & 0 & 0 & 0 & 2 \\ 6 & 2 & 6 & 2 & 0 & 0 & 0 \\ 6 & 0 & 2 & 6 & 2 & 0 & 0 \\ 6 & 0 & 0 & 2 & 6 & 2 & 0 \\ 6 & 0 & 0 & 0 & 2 & 6 & 2 \\ 6 & 2 & 0 & 0 & 0 & 2 & 6 \end{pmatrix}$$

The first row represents the vertex rule of Loop's scheme (as presented in figure 3.5a), while the following rows are the edge rules corresponding to each new vertex in the new 1-ring (cf. figure 3.5b).

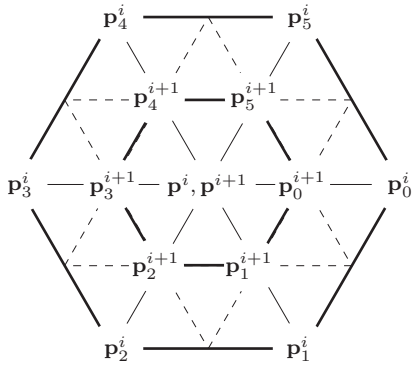

Figure 3.8: *Evolution of the 1-Ring:* two generations of the 1-Ring around the same vertex are shown.

Equation 3.4 can be reiterated from $\mathbf{P}^0$ for k subdivision steps[4]:

$$\mathbf{P}^{k} \quad = \quad \mathbf{S}^{k} \cdot \mathbf{P}^{0}. \tag{3.5}$$

It is evident that the components of the vector $\mathbf{P}^k$ will tend to the same value if the scheme is well defined (the 1-ring positions $\mathbf{p}_j^k$ should shrink around the analysed vertex position $\mathbf{p}^k$). A way to analyse further the convergence of the vector $\mathbf{P}^k$ is to rewrite equation 3.5 using the eigenvectors and eigenvalues of the matrix $\mathbf{S}$, that is:

$$\mathbf{P}^k = \mathbf{S}^k \cdot \mathbf{P}^0 = \sum_{i=0}^{6} \lambda_i^k \mathbf{e}_i \mathbf{q}_i, \tag{3.6}$$

[3]note that $\mathbf{p}_j^i$ always represent the position of the vertex in the 1-ring at the current subdivision level!

[4]The notation could be a bit confusing: $\mathbf{P}^k$ signifies the vector $\mathbf{P}$ at subdivision step k, while $\mathbf{S}^k$ stands for the square matrix $\mathbf{S}$ to the power of k

where λ_i are the eigenvalues of $\mathbf{S}$ (ordered such that $\lambda_0 \geq \lambda_1 \geq \ldots \geq \lambda_6$), $\mathbf{e}_i$ the corresponding eigenvectors and $\mathbf{q}_i$ the expression of $\mathbf{P}^0$ in the basis formed by the $\mathbf{e}_i$.

It is now obvious that $\mathbf{P}^k$ converges to a non trivial point if, and only if, at least one eigenvalue is equal to 1 and the others ones are strictly inferior to 1. In fact it was shown in [25] that the subdivision scheme is convergent and affine invariant if λ_0 is equal to 1 and the remaining eigenvalues are strictly inferior to 1. It is also possible to analyse the tangent plan continuity by looking at the equation 3.6 with only the subdominant eigenvalues.

One interesting insight given by the above analysis[5] is that the limit position (in the limit smooth surface) of a given vertex is only determined by $\mathbf{P}^0$ (i.e. the positions of a few neighbouring vertices and itself). Subdivision surfaces schemes are thus totally locally determined, which is a very important property. Imagine for a moment the case of non-locally determined subdivision surfaces, where a designer trying to modify the surface at a given location incidently changes the shape of the surface at another unexpected place (or even the whole shape)...

3.5 Tags: a Way to Piecewise Smooth Surfaces

Figure 3.5 shows the stencils used for repositioning the even regular interior vertices and the ones for creating new interior vertices. These kind of rules only permit to produce smooth surfaces[6] (figure 3.10b), but one may want to have sharp edges (figure 3.10d) or corners (figure 3.10f) in order to be able to model a wide variety of objects. To represent these shapes with subdivision surfaces, the idea is to divide the mesh in several smooth parts, called *patches*, that meet in sharp edges or *creases*.

One method to create these creases is to tag edges of the initial control hull, as proposed in [1][7]: that is, to tag them as *crease* or not. Note that, boundary edges are 'automatically' tagged as crease, so the presented rules will not make mention of boundary or interior conditions, only that of crease or not. A *crease line* is a sequence of crease edges that separates two smooth patches, the crease line itself ought to be smooth while the 'separation' between patches should be sharp. To achieve this, the crease edge tags define, in turn, associated tags on the vertices, that is:

- a vertex incident to one and only one crease edge is tagged as *dart*;
- a vertex incident to exactly two crease edges is tagged as *crease*;
- a vertex incident to more than two crease edges is tagged as *corner*.

A crease vertex is a vertex which is along the crease line, its position should be independent on the geometry of the neighbouring patches but not on the geom-

[5] that could have been (more easily) found by others means

[6] closed or open

[7] beside piecewise smooth surfaces (that were already introduced in other papers), this article addresses the continuity problem that can appear next to extraordinary boundary vertex with Loop's original rules. It also enables the designer to exactly control the shape at corner (that can be convex or concave) by specifying a crease angle. However, for brevity sake, these two improvements will not be presented here.

etry of the crease line. A dart vertex ends a crease line (merging it smoothly into a patch), its position must depend at the same time on the geometry of the crease line and of the smooth piece. At a corner vertex several crease lines are joined, the position of this vertex depends only on the crease lines geometry.

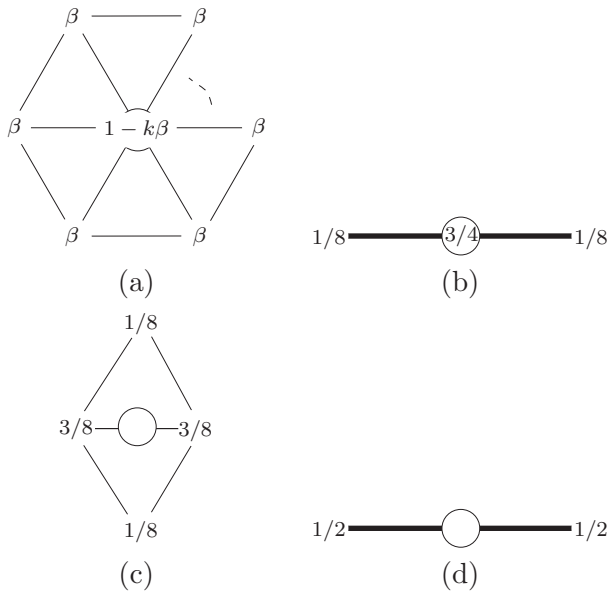

Figure 3.9: *Tagged Loop's vertex and edge stencils* (crease edges are represented with a ticker line). (a) is used to reposition a vertex that is not on an edge or tagged as corner; (b) to reposition a vertex on a crease edge. (c) is used to split an untagged edge with one dart end; and finally (d) to split a crease edge (without end tagged as dart).

Figure 3.9 shows the stencils used for the piecewise smooth Loop subdivision surfaces. For β, Loop proposed (k is the crease degree of the vertex):

$$\beta = \frac{1}{k}\left(\frac{5}{8} - \left(\frac{3}{8} + \frac{1}{4}\cos\frac{2\pi}{k}\right)^2\right) \tag{3.7}$$

Warren latter proposed to have (which amounts to having the same value for $k = 3$ and $k = 6$ than Loop's β):

$$\beta = \begin{cases} 3/16 & \text{if } k = 3 \\ 3/(8k) & \text{otherwise} \end{cases} \tag{3.8}$$

Vertex rules are the following:

- in absence of tag, or if the vertex is tagged as dart, stencil a is used;
- if a vertex is tagged as crease the stencil b is applied;
- at a corner, the vertex position is never moved.

Edge rules are the following:

- if the edge is not a crease edge, stencil c is applied;

- if a crease edge is split there are two cases:
  - if no edge ends is a dart vertex then the stencil d is used;
  - otherwise the stencil c is applied (allowing the crease line to blend smoothly into the surface).

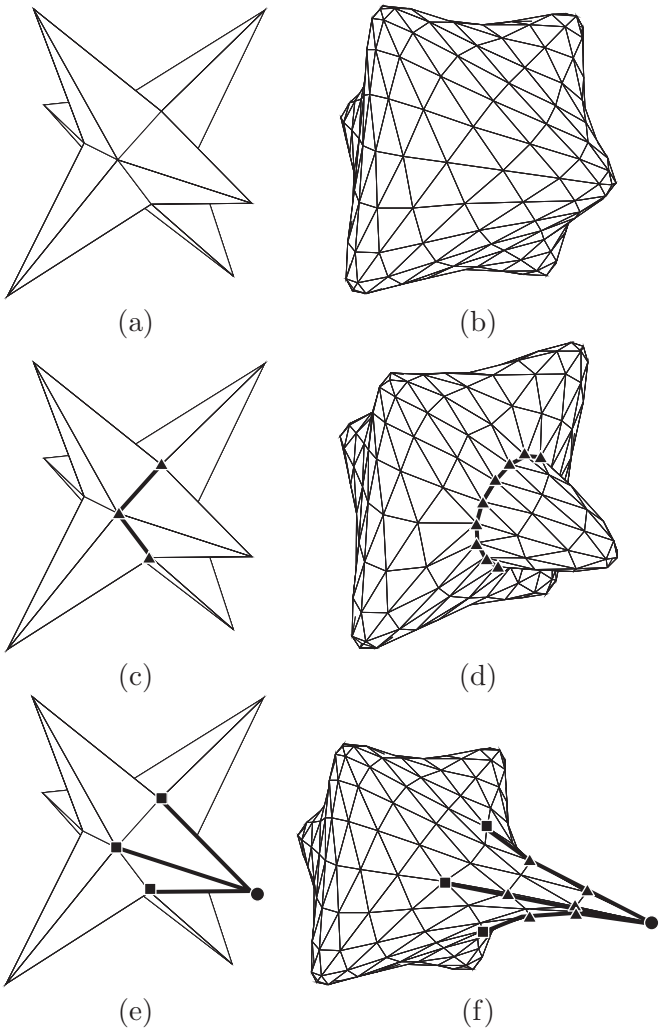

Figure 3.10: *Tags example.* The initial control hull is on the left, the edges tagged as crease are noted with thicker lines. The associated tags on the vertex are shown with dot (corner), triangle (crease) or square (dart). The mesh after two subdivision steps is shown on the right.

3.6 MPEG-4 Implementation of Subdivision Surfaces

As already seen in chapter, MPEG-4 (through its AFX extension) provides a toolbox for 'plain' subdivision surfaces. This toolbox is represented by a new node in the scene graph tree: `SubdivisionSurface`, which is based on an

already existing node (`IndexedFaceSet`). Different subdivision surfaces scheme are supported, that is:

- for triangular meshes: *Loop*, *modified butterfly* and *midpoint*;
- for quadrilateral meshes: *Catmull-Clark*;
- for arbitrary meshes: *extended Loop*.

All schemes supports tags (crease, corner and dart), and additional information about corners could be added through the `Sector` node. Loop scheme is based (with some minor modification) on [1], that is with continuity correction and exact geometry control at corner. Butterfly follow the modified butterfly introduced in [25], with support of crease, dart and corner tags. While the extended Loop was a work proposed by the company Superscape, and gives a total control on the resulting triangulation of the final surface.

Chapter 4

Wavelet Subdivision Surfaces

This chapter introduces *wavelet subdivision surfaces*, which are more or less an extension of 'plain' subdivision surfaces. First some differences between wavelets and 'plain' subdivision surfaces are introduced. Then wavelet transforms and their general properties are presented followed by a rapid presentation on the link existing between subdivision surfaces and wavelets. Then, a more 'in depth' presentation of efficient wavelet coefficients coding is done. Finally, the wavelet subdivision surfaces node within MPEG-4 AFX is introduced.

4.1 Introduction

The followed path of *wavelet subdivision surfaces* is the reverse of the 'plain' subdivision surfaces: a very fine mesh, approximating a possibly complex surface[1], is recursively simplified to find a sequence of coarser meshes. During this simplification process, different sets of wavelet coefficients, permitting to retrieve the finer meshes, are created. To reconstruct the very fine mesh, one needs the coarsest mesh (or *base mesh*) and the wavelet coefficients associated with each subdivision level.

Moreover, wavelet subdivision surfaces could be used to enhance 'plain' subdivision surfaces, with which it is not possible to have fine details on the final mesh. That is, wavelets subdivision surfaces provides a framework in which one can create a synthetic object and performs a fine scale edit adding details as fine as the last subdivision levels permits (which is not possible with 'plain' subdivision surfaces).

Wavelet subdivision surfaces are akin to *signal processing*, which makes some conceptual difference with 'plain' subdivision surface, at least in the way things are presented (as it can be seen below for the refinement step).

4.1.1 Refinement Step (in Wavelets Subdivision Surfaces)

First, even vertex position are set to the values they had on the precedent level of subdivision, and odd vertex positions to $\mathbf{0}$. Then a *low-pass filter* is applied to the *base mesh* (also given through the form of a weighting mask). Another

[1] e.g. obtained by '3D scanning' of a physical shape

mesh, *detail mesh*, is created (with the same topology) with odd vertex positions having the value of the wavelet coefficients and the even ones being set to **0**, to which a *high-pass filter* is applied. Finally the vertex positions of the two meshes are added to form the final mesh for this subdivision step. Note that the result after the low-pass filtering the base mesh can be exactly the same as for 'plain' subdivision surfaces if the wavelet used is derived from the method used in the 'plain' scheme. A low-pass filter using Loop's wavelet is shown in figure 4.1 where it can be seen that the used filter is exactly equivalent to Loop's subdivision surfaces. In wavelet nomenclature, the splitting step together with the first setting of vertex positions is called *up-sampling*.

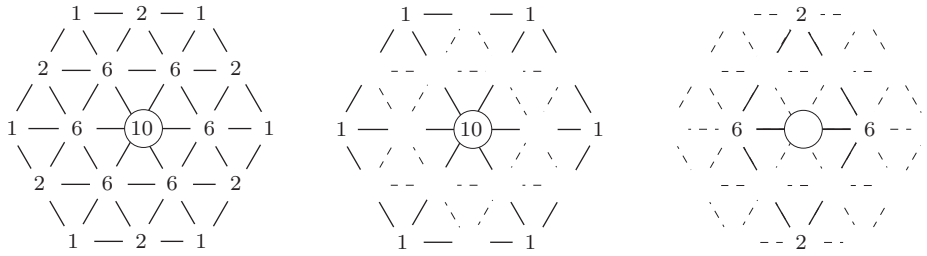

Figure 4.1: *Loop's wavelet low-pass reconstruction filter* (to not surcharge the figure the coefficients c are the numerators of the fraction $c/16$). Figure (a) shows the Loop's low-pass filter. Figure (b) and (c) explicit the same filter but centred on an even or odd vertex, respectively (the coefficients falling on odd vertices are hidden).

4.2 Overview of Wavelets

One step of a wavelet transform on a discrete data set s_i^0 produces two new data sets which together contain the same number of elements as the original set. The first one contains the *scaling coefficients* s_k^1, which defines a lower resolution[2] representation of the original data. And the other one contains the *wavelet (or detail) coefficients* c_k^1, which define the necessary information to retrieve the original data from the low resolution data set. The wavelet coefficients could, thus, be seen as the coding of the details lost while passing to a lower resolution. The *wavelet transform* recursively performs the above operation on the low resolution data set. After N transformation steps, one has N wavelet coefficients sets $\{c_i^j\}_j$ and one scaling coefficients set s_i^N, but the total amount of coefficients still being the same than in the original data set. These different sets are also called *subband analysis*. Figure 4.2 shows a generic wavelet transform, also called *analysis*. The scaling and wavelet coefficients are obtained through the recursive application of the same two *analysis filters* **H**

[2]lower resolution in the sense that the global 'shape' is equivalent but with less elements, thus fewer details

and **G**, respectively. The ↓ signs shows the decimation step (suppression of some samples) applied after each filtering in order to have the same amount of elements[3].

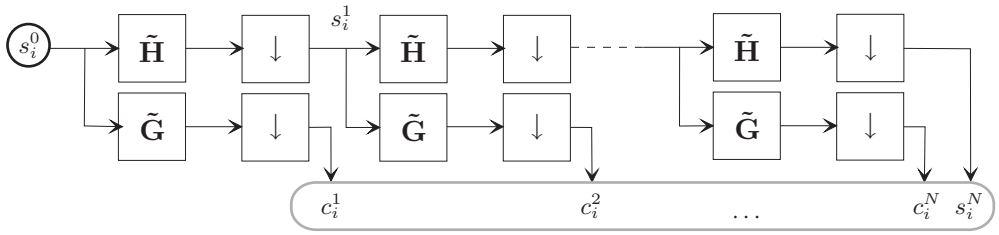

Figure 4.2: *Forward wavelet transform*

The inverse transform (also called *synthesis*) is essentially the same as the forward transform but with the inverse operations. A generic inverse transform is shown in figure 4.3. The *synthesis filters* $\tilde{\mathbf{H}}$ and $\tilde{\mathbf{G}}$ are used to invert the effect of the analysis ones, the up-sampling (↑) step adds zero elements where samples have been suppressed during the analysis phase.

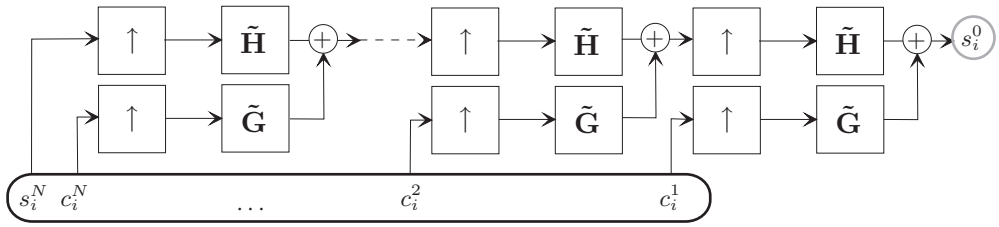

Figure 4.3: *Inverse wavelet transform*

4.2.1 'Analysis' of Wavelets

Beside the mathematic framework in which they are often presented, wavelet transform can be seen from others different points of view, which can help to better understand the reason of their 'successes' by highlighting a few key points.

For example, one can see wavelets as a *predictive data coding process* where the couple up-sampling and synthesis filter $\tilde{\mathbf{H}}$ represents a *predictive model*, and the the *prediction errors* are the data obtained after the application of the synthesis filter $\tilde{\mathbf{G}}$. The main advantage of wavelet over other predictive coding is the *hierarchical representation* of the transformed data inherently presents in the wavelet transform (which is the necessary element to enable the use of the genial SPiHT[4] algorithm), while hierarchical representation is not always available in a predictive data coding scheme.

[3]Note that the filters **H** and **G** must have some special properties so that no information is lost during the decimation step...

[4]see section 4.4.2, or [20][5]

It can also be seen as a *filters bank*, which in fact they are. The difference, and advantage, between a wavelet filter bank and a generic filters bank is that the filters in the wavelet one are linked to a *multiresolution analysis* (see the next section).

One can also add that wavelets are a powerful tool for *data decorrelation*, it means they can capture the essential features of the data set in few important scaling and wavelet coefficients. The reason of this prodigy lies in properties shared by most organised data set[5]:

- the first one could be called *local scale invariance*, meaning that they tend to have locally the same characteristics from any scales they are looked at;
- the second (which is in fact derived from the first one) states that if a certain characteristic is not present in a certain area at a given scale, it will likely (i.e. with probabilities ≈ 1) also not be present in the corresponding area at finer scales.

Following the first property, wavelets are then an appropriated analysis tool[6] for organised data sets since they use refinable basis function (meaning that some overall characteristics are kept common between all scales). The second property of organised data sets is important because it enables the use (in conjunction with wavelets) of the SPiHT algorithm in a efficient way.

Finally, one can have the best analyse tool 'in the world' but if it cannot be used in finite time (and preferably in the shortest possible time) this fabulous tools will not be very attractive. Luckily, the fast wavelet transform permits to compute a wavelet transform in a time *linear* with the length of the data (and also linear with the length of the filter).

4.2.2 Multiresolution Analysis Framework

There are two basic ingredients to set a multiresolution analysis. The first one is a *infinite chain of nested linear function spaces*, and the second one is an *inner product*:

$$\mathcal{V}^0 \subset \mathcal{V}^1 \subset \mathcal{V}^2 \subset \dots \tag{4.1}$$

$$\langle f, g \rangle, \forall f, g \in \mathcal{V}^j (j < \infty). \tag{4.2}$$

The inner product is used to define the orthogonal complement $\mathcal{W}^j$ of $\mathcal{V}^j$:

$$\mathcal{W}^j := \left\{ f \in \mathcal{V}^{j+1} | \langle f, g \rangle = 0, g \in \mathcal{V}^j \right\}. \tag{4.3}$$

Then $f \in \mathcal{V}^{j+1}$ can be uniquely represented as $f^{j+1} = f^j + h^j$ with $h^j \in \mathcal{W}^j$.

The *scaling functions* are the basis of $\mathcal{V}^j$, and are written $\phi_i^j(\mathbf{x})$. And the *wavelets* are the basis of $\mathcal{W}^j$ and are denoted $\psi_i^j(\mathbf{x})$.

The *low-pass filter* $\mathbf{H}$ associating the scaling functions between to adjacent scales is defined as:

$$\phi_i^j = \sum_i h_i^j \phi_i^{j+1}. \tag{4.4}$$

[5] e.g. pictures, music, video, 3D-mesh, etc.

[6] provided that the adequate wavelet is used, i.e. with properties akin to the analysed data set

This means that each scaling function can be written as a linear combination of scaling functions on the next finer level. The *high-pass filter* **H** is defined equivalently as:

$$\psi_i^j = \sum_i g_i^j \psi_i^{j+1}. \tag{4.5}$$

There are different types of wavelets: the *fully orthogonal wavelets* are the strictest form of wavelets where $\langle \psi_i^j(\mathbf{x}), \psi_{i'}^{j'}(\mathbf{x}) \rangle = 0 \ \forall (i,j) \neq (i',j')$.

It was shown that it was impossible to have at the same time locally supported, fully orthogonal and symmetric wavelets. It is why the *semi-orthogonal wavelets* were introduced, $\langle \psi_i^j(\mathbf{x}), \psi_{i'}^{j'}(\mathbf{x}) \rangle = 0 \ \forall j \neq j'$.

The least restrictive form of wavelets are the *bi-orthogonal wavelets* where the orthogonality is totally dropped and $\mathcal{W}^j$ is only some complement of $\mathcal{V}^j$ in $\mathcal{V}^{j+1}$.

4.2.3 Subdivision Surfaces and Wavelets

The *First generation wavelets* define the scaling and wavelet functions through the dilatation and contraction of some mother function. In conjunction with Fourier transformation (where dilatation and contraction become algebraic operation) it enables to 'easily' create a setup for multiresolution analysis. The problem is that this construction is not easily adaptable to non-trivial topology (e.g. manifolds) or irregularly sampled data sets. It is the reason of the apparition of the logically termed *second generation wavelets*, in which the scaling and wavelet functions are defined with other means[7].

The work in [19] first showed the construction of wavelets on a sphere with the help of subdivision surfaces. Subdivision surfaces were then formally linked to wavelets with the fundamental work presented in [14], which first showed that it was possible to build a multiresolution analysis over a surface of arbitrary topology type by using subdivision surfaces. Finally, in [21] it was shown that wavelet on subdivision surfaces as defined in [14] are a special case of the *lifting scheme*. The lifting scheme is a very smart way of building a multiresolution analysis, where one starts with a very simple one (like lazy or Haar wavelets[8]) and enhance it through 'lifting'.

Decimation in Meshes

In regularly sampled data set (in 1D or 2D), the decimation step is quite obvious: data are usually 'suppressed' each two samples. The situation is a bit more complex for non-trivial meshes because mesh connectivity is usually not regular. However a mesh presenting a *subdivision surfaces connectivity* can be consistently decimated as shown in figure 4.4. Subdivision surfaces connectivity means that the abstract graph is equivalent to one which would have been

[7] note that also prevent the use of the Fourier transform, which anyway could not have been extended to all cases covered by second generation wavelets

[8] which can be defined on surfaces of arbitrary topological type

through various steps (at least one) of subdivision[9]. As meshes abstract graphs do not 'naturally' present this connectivity, meshes are usually remeshed before the wavelet forward transform. Several techniques exist, for more details one can refer to [3] (for automatic remeshing), or [15] (for partially user driven remeshing). Note that remeshing is somewhat reasonable in piecewise smooth surfaces: the mesh being a linear approximation of a target surface, there are no particular reasons against the existence of another linear approximation as good[10], but with the desired connectivity.

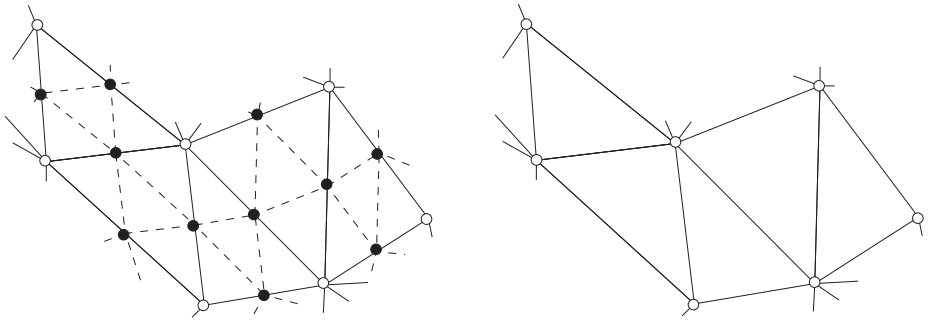

Figure 4.4: *Mesh Decimation.* In (a), a mesh with subdivision surfaces connectivity is presented: white circles represent the vertices to be kept after decimation, and black dots the ones to be suppressed. While (b) shows the same mesh after decimation.

Also note that quadrisection as presented in section 3.2.1 is actually a form of upsampling.

Computation of Details Coefficients

There are different ways of computing the details coefficients for meshes. One can use the lifted transformation for schemes like butterfly (as shown in [19]), where finite filters can be found for both reverse and forward transforms. However, there is no lifting scheme corresponding to Loop wavelet (i.e. the analysis and reconstruction filters cannot be both finite). Thus one need to resolve a linear equations system (linking the details coefficients and the coarser mesh with the finer mesh through inverse transform) in order to actually find the coarser mesh and the detail coefficients, as shown in [12]. This makes the Loop wavelet analysis of meshes computationally more complex than for butterfly wavelet (where both forward and reverse time can be done in time linear with the number of vertices). However the synthesis complexity is only slightly more important than for butterfly.

[9] that is, most vertices are regular, while a few isolated ones (corresponding to base mesh vertices) are irregular, cf. figure 3.3.

[10] with same number of vertices, equivalent approximation error, etc.

4.3 Short Mesh Compression Roundup

This section introduces 'pointers' to some of the current state-of-the-art mesh compression methods.

Compression for mesh can be done on the two independent components of a mesh (cf. section 2.2): the geometry, and the abstract graph (or connectivity information). The compression on the geometry is quite 'classical'[11], while the one applied on the abstract graph is a bit more 'original'. It is, mainly, methods applied for compression of the connectivity information that are introduced in this short roundup.

4.3.1 General Considerations on Compression

There are two main families of data compression: *lossless* and *lossy*. With the first family, data can be integrally retrieved after transmission and decompression, while with the second one only an approximate version of the data can be retrieved. Of course, lossy compression is only usable where to have the exact data is not necessary: e.g. sounds, images, 3D meshes intended to be visualised. That is, where the data will be 'interfaced' with human senses that are not sensitive (or not very) to small variations in the data set.

Compressed data transmission can also be classified (independently of lossy or not) in two categories: progressive or not. In a progressive compression scheme, it is possible to decompress a subset of the compressed data and still be able to recuperate the data (in a coarser way, for more details, see section 4.4.1); while all the compressed data are needed in a non-progressive scheme.

4.3.2 Progressive Compression Methods

Progressive Mesh

This method [7] is based on the principe of *vertex split* and *edge collapse* (where two triangles, three edges and one vertex are added or suppressed from the mesh, respectively), shown in figure 4.5. During the compression step, the original mesh is recursively simplified through a sequence of edge collapses to find the base mesh: a coarser version of the original mesh. The decompression step is the dual operation: an inverse sequence of vertex splits is applied to the base mesh in order to obtain the original mesh. Vertex positions can be interpolated as a continuous function from their initial common location to their respective final ones, which enable visually smooth transition between each edge collapse or vertex splits. The progressive granularity proposed by this method is maximal, since one can stop the decompression after any one of the vertex split steps, hence controlling the mesh enrichment vertex by vertex. According to its author, this method allows to code meshes with around $35\,bit/vertex$ for a good approximation of the input mesh.

[11] wavelet transform for decorrelation, predictive techniques, quantisation, entropy coding, etc.

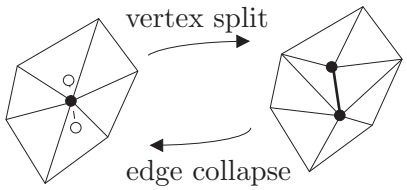

Figure 4.5: *Vertex Split and Edge Collapse.* This figure presents the duality existing between the operations of edge collapse and vertex split.

Compressed Progressive Mesh

This relatively simple triangular abstract graph compression method [17] baptised 'edgebreaker' is based on an edge conquest method. At each stage, compression produces an op-code describing the topological relation between the current triangle and the boundary of the remaining part of the mesh. Decompression uses these op-codes to reconstruct the entire incidence graph. The paper claims that mesh compressed with this method require as less as 1.5 $bit/vertex$ to compress the connectivity information. Also, because edgebreaker's compression and decompression are independent of the vertex locations, they may be combined with a variety of geometry-compressing techniques that exploit topological information about the mesh to better estimate vertex locations.

Progressive Geometry Compression

This method [12] is based on wavelet subdivision surfaces. First the input mesh is remeshed in order to have a mesh with subdivision surfaces connectivity. Then the manifold mesh is simplified through wavelet analysis, to finally find a coarse mesh (or base mesh). The main contribution of this paper is the proposition of an edge-based details hierarchy (cf. section 3.3), which enables the use of the beautiful SPIHT algorithm (cf. section 4.4). The total bitrates can be estimated to be around $10 - 15\, bit/vertex$ for a good approximation of the input mesh.

Note that the method proposed in [5] is very similar, the main difference being that it has a face-based details hierarchy and that it does not need explicit remeshing of the input mesh.

4.3.3 Non-Progressive Compression Methods

Many of the progressive methods mentioned above need to transmit a base mesh and then information to retrieve the finer meshes. But they do not address the particular task of compressing the base mesh. It is where single resolution compression methods are interesting.

Topological Surgery

In this method [23], a spiralling triangle tree is constructed to cover the input mesh (like peeling an orange, as shown in figure 4.6) and in such a way that the connectivity information can be very compactly coded. A vertex spanning tree (interlocked with the triangle tree) is also created to define a vertices traversal order, which enables the uses of predictive techniques to code the geometrical information[12], thus around $8 - 12\, bit/vertex$ only are needed to code the coordinate. The total bitrates (geometry and connectivity information) can be estimated to be around $20\, bit/vertex$ for a good approximation of the input mesh.

(a) (b) (c)

Figure 4.6: *"Orange Peeling"*. Figure (a) shows the original mesh, (b) the 'peeled' one and (c) the spiraling triangles tree.

Note that this method was has adopted, practically without modifications[13], for the MPEG-4 version 2 (1999) standard.

Triangle Mesh Compression

This method [22] sweeps the mesh with an edge conquest mechanism which generates (for each conquered vertex) a valence code; and an exception code in presence of boundaries (or other irregularities of the mesh). The ordered list of those valences and exception codes is then entropy coded to yield extremely low bit counts for connectivity information (as low as $1.5\, bit/vertex$). In fact, this technique is currently considered to be the best single resolution 3D mesh coding technique in terms of compression ratio, especially for mostly regular meshes. According to the authors, this method allows to code mesh with around $15 - 20\, bit/vertex$ for a good approximation of the input mesh.

4.4 Efficient Compression of Detail Coefficients in Meshes

Embedded zerotree wavelet coding, introduced by Shapiro [18] is a very effective and computationally simple technique for image compression. Later, Said and Pearlman [20] extended this technique and gave an alternative explanation of its excellent performance, based on set partitioning in hierarchical trees. This

[12] because unknown vertex locations can be estimated based on the ones of previously visited vertices

[13] the only significant addition was the one of an error resilience technique

subsection will first summarise this explanation and 'upgrade' it to mesh coding, to then introduce its place in MPEG-4 wavelet subdivision surfaces tool.

4.4.1 Progressive or Embedded Mesh Transmission

The idea in progressive transmission is to transmit a mesh in such a way that the bitstream (in which the mesh details are coded) can be stopped at any point and the mesh can still be decompressed and reconstructed. Moreover, the more bits are transmitted the less 'different' is the decoded mesh from the original mesh. This is the strong point of this transmission method: a single bitstream can be used for various bitrates; that is, without needing to code the mesh details again.

The basic idea is to first transform the original mesh $\mathcal{S}$ with an unitary hierarchical subband transformation Ω (e.g. a wavelet transform), that is:

$$\mathcal{C} = \Omega(\mathcal{S}). \tag{4.6}$$

The transformed model $\mathcal{C}$ is composed of two parts (cf. figure 4.2): the base mesh $\mathcal{B}$ and the sets $\mathcal{D}^j$ of detail coefficients d_i^j (corresponding to subdivision level j). Note that this section is only about the coding of the detail coefficients and not about the base mesh coding (which is supposed to have been already coded[14] and transmitted). After reception and decoding, the *decoded detail sets* are denoted $\hat{\mathcal{D}}^j$ (which could be exactly or approximately $\mathcal{D}^j$), and finally the reconstructed mesh is noted as $\hat{\mathcal{S}}$.

First let us remark that the transform being unitary, its Euclidean norm is invariant. It means that the mean square error between the geometry of $\mathcal{S}$ and $\hat{\mathcal{S}}$ is the same than between d_i^j and $\hat{d}_i^j$ (as the base mesh is supposed to be transmitted without error). This means that the greatest coefficient of $\mathcal{D}^j$ should be transmitted first, in order to have an embedded bitstream.

Detail Vectors Handling

For a mesh, detail coefficients are vectors. But, as the different components of a coefficient are 'independent', there are different possibilities to deal with this 'problem': separate streams for each component of the vector, components interleaving in one stream or vector 'scalarisation' (cf. [5]). In the chosen method (i.e MPEG-4), the SPiHT algorithm is applied independently on each component of the detail vectors. This means that on the following explanation of the algorithm, one has to read "component [x,y,z] of the details coefficient" instead of 'coefficient'.

Binary Representation of Coefficients

Moreover, since the coefficients d_i^j are real numbers, they need to be transformed into a fixed point representation. First, the maximal absolute value d_{max} of all

[14] cf. section 4.3.3

the coefficients in $\left(\mathcal{D}^j\right)_j$ is found:

$$d_{max} = \max_{i,j} \left| d_i^j \right| . \tag{4.7}$$

Then, each detail d_i^j is linked to an integer c_i^j (i.e. quantised), and coded on $N+1$ bits[15]:

$$c_i^j = \left\lfloor \frac{\left(2^N - 1\right) \cdot d_i^j}{d_{max}} \right\rfloor , \tag{4.8}$$

where $\lfloor \bullet \rfloor$ denotes the *floor* operation, and where the binary representation of the c_i^j is:

$$c_i^j = (-1)^s \cdot \left(b_{N-1} \cdot 2^{N-1} + b_{N-2} \cdot 2^{N-2} + \ldots + b_1 \cdot 2 + b_0 \right) , \tag{4.9}$$

where s and the b_k are bits of the encoded representation.

Note that some information is lost during this step: during the floating point coding of d_{max}; and essential under the details quantisation process (floor operation and choice of N). This makes the SPiHT coding of real numbers a lossy compression method.

Naive Embedded Bitstream

A naive way of constructing an embedded bit stream containing the quantised coefficient c_i^j is to transmit these coefficients bit plane by bit plane[16]. That is, transmitting first all s, then all b_{n-1}, etc. (as shown in figure 4.7). It would effectively produce the desired embedded bitstream.

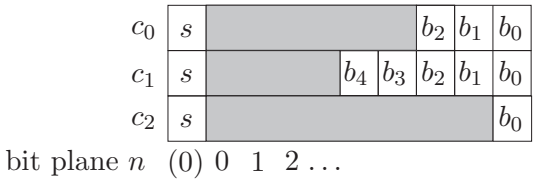

Figure 4.7: *Bit plane organisation.* This figure shows the bit plane organisation of some integer c_i, the greyed zone means bits with value 0.

The reason of the poor performance[17] of that naive algorithm lies in the following reasons: the wavelet decorrelation property (cf. section 4.2.1) means that a great deal of the transmitted bit were actually *zeros*. Moreover offspring coefficients have great probabilities to be actually smaller than their parents, which could be used to improve this naive algorithm. The SPiHT algorithm uses these properties (and the *odd vertices hierarchy*) to define a very smart way of transmitting the d_i^j coefficients.

[15] i.e. one sign bit and an absolute value ranging from 0 to $2^N - 1$

[16] note that, the parameters *max* and n need to have been previously transmitted

[17] poor in the sense: "something much better exists"

4.4.2 Set Partitioning in Hierarchical Trees

The basic idea behind SPiHT is to start transmitting the bits concerning a coefficient only when this one is significant (that is, not to transmit the greyed zone in figure 4.7). It also means that an order has to be set through a sorting algorithm. One of the mean features of the SPiHT algorithm is to not explicitly transmit the d_i^j coefficients order. Instead, it is based on the fact that in any algorithm, the execution path is based on the comparison results at its branching points.

Let first define the significance S (at a bit plane n) of a given set $\mathcal{T}$ of coefficients d_i^j:

$$S_n(\mathcal{T}) = \begin{cases} 1, & \max\limits_{d_i^j \in \mathcal{T}} d_i^j \geq 2^n, \\ 0, & \text{otherwise.} \end{cases} \tag{4.10}$$

If $S_n(\mathcal{T}) = 1$, it simply means that at least one coefficient d_i^j in the set $\mathcal{T}$ is significant (i.e. at least equal, in absolute value, to 2^n).

The beautiful idea of the SPiHT algorithm is to partition the coefficients in such a way that subsets expected to be significant only contain one element, and subsets expected to be insignificant contain a large number of elements. When an expected insignificant subset turns out to be significant it is in turn partitioned the same way. As both encoder and decoder use the same set partitioning algorithm, only the significance test results (on subsets and coefficients) need to be transmitted for actually retrieve the coefficients order.

Wavelet Subdivision Surfaces Hierarchical Trees

The problem yet to solve is the partition choice. As already shown in section 3.3, subdivision surfaces inherently define hierarchical structures. One of these structures can be used to define a canonical (i.e. independent of the significance) coefficients order. Moreover, as it was explained in section 4.2.1, the wavelets transform tends to locally yield self-similarities within subband. That is, if a coefficient is not significant until a certain bit plane n, there are important probabilities that the corresponding offspring coefficients would not be significant in the previous bit planes.

It is then straightforward that the hierarchy used is the 'vertices'-based hierarchy, since wavelet coefficients are linked to odd vertices. A first possible logical choices for initial partitions are the actual trees of the hierarchy. As there are great probabilities that the tree roots would be significant for the first bit plane, let us decide that initial subsets expected to be significant are the tree roots and the initial subsets expected to be insignificant are their descents. The subsequent partitioning is now quite evident: if an expected insignificant subset is found significant then it is separated in a subset expecting to be significant (containing the 'root' of the descent) and in an another subset expected to be insignificant (the descent of this new root).

Coding/Decoding Algorithm

Since the order in which subsets are tested for significance is important, a practical implementation of the aforementioned ideas stores these subsets in three ordered lists (as presented in [21]):

- LIS: List of Insignificant Sets;
- LIC: List of Insignificant Coefficients;
- LSC: List of Significant Coefficients.

While the LIC and LSC lists stores individual coefficients, LIS could either represent $\mathcal{D}$ or $\mathcal{L}$. To differentiate these two possible entries it is said that the first one is of type A and that the second one is of type B.

Finally, let us define some sets that will be used in the SPiHT algorithm:

- $\mathcal{O}(c)$ is a set of vertex indices of all offsprings of node c;
- $\mathcal{D}(c)$ is the transitive closure of the relation "is offspring of" (whole set of descent of c);
- $\mathcal{L} = \mathcal{D}(c) - \mathcal{O}(c)$;
- $\mathcal{H}$ contains every tree roots (node at the highest pyramid level);

The complete coding algorithm is presented below. To find the corresponding decoding algorithm, one has to replace *output* by *input*. Note that refinement bits are not transmitted for coefficients that have been just found significant in the previous sorting pass, as a coefficient first significant at a certain bit p plane means that the corresponding bit b_{N-p} has a value of 1 in its binary representation (it is thus totally deterministic).

It is also possible to reduce the decoded mesh error for each sign or refinement bit transmitted. The idea is to always set the value of the current decoded coefficient to the mean between the maximum and the minimum values it could take in the future. There are two cases: the initialisation (when the sign bit is received) and the refinement. For the initialisation, the maximum value could be $2^n - 1$ and the minimum 2^{n-1}, which gives an initial coefficient value of:

$$c^n = \frac{2^n - 1 + 2^{n-1}}{2} = 3 \cdot 2^{n-2} + 1/2, \tag{4.11}$$

where n corresponds to the one in algorithm 1. When updating this mean value (during the refinement pass) the maximum or the minimum values could change in function of the received bit. If the refinement bit is 0, then the maximum possible value would change with respect to the precedent bit plane: $c^n_{max} = c^{n+1}_{max} - 2^{n-1}$; if the bit is 1, then the minimum value would change with respect to the precedent bit plane: $c^n_{min} = c^{n+1}_{min} + 2^{n-1}$. Meaning that the coefficient is updated to:

$$c^n = c^{n+1} + \begin{cases} -2^{n-2} & \text{if refinement bit is 0;} \\ +2^{n-2} & \text{if refinement bit is 1.} \end{cases} \tag{4.12}$$

Embedded Bitstream Property

A SPiHT bitstream structure for a mesh containing two trees (x and y) could be (for the first bit plane):

$$y_0 x_0 y_1 y_2 x_1 x_2 x_3 y_3 y_4 y_5 y_6 x_4 x_5 x_6 x_7 x_8 x_9 y_7 x_{10} y_8 y_9 y_{10} x_{11} y_{11} x_{12} \dots$$

Algorithm 1 *SPiHT*

Initialisation
output N
$n = N$
$LSC = \emptyset$
$LIC = \mathcal{H}$
$LIS = \mathcal{H}$ (type A)
Sorting Pass
for $c \in LIC$
    output $S_n(c)$
    if $S_n(c) = 1$
        then move c at the end of the LSC and **output** its sign
for $c \in LIS$
    if type$(c) = A$
        then
            output $S_n(\mathcal{D}(c))$
            if $S_n(\mathcal{D}(c)) = 1$
                then for $k \in \mathcal{O}(c)$
                    output $S_n(k)$
                    if $S_n(k) = 1$
                        then add k to the end of the LSC and output its sign
                        else add k to the end of the LIC
                if $\mathcal{L}(c) = 0$
                    then move c at the end of the LIS (type B)
                    else remove entry c from the LIS
        else
            output $S_n(\mathcal{L}(c))$
            if $S_n(\mathcal{L}(c)) = 1$
                then add each $k \in \mathcal{O}(c)$ to the end of the LIS (type A)
                    remove c from the LIS
Refinement pass
for $c \in LSC$ (except those included in the previous sorting pass)
    output the n-th most significant bit of $|c|$
Quantisation-step Update
$n = n - 1$
if $n \neq 0$
    then go to *Sorting Pass*

This stream is used when the SPiHT algorithm is initialised with both roots of the trees x and y. It is also possible to run two instances of the algorithm, one initialised with x root, and the other with y if one has access to two separate streams (one for each tree):

$$x_0 \quad x_1x_2x_3 \quad x_4x_5x_6x_7x_8x_9 \quad x_{10}x_{11} \quad x_{12}\ldots$$

$$y_0 \quad y_1y_2 \quad y_3y_4y_5y_6 \quad y_7 \quad y_8y_9y_{10}y_{11}\ldots$$

What makes it possible is that only the relative order within a tree is important. In fact, even the SPiHT initialised with all forest works 'independently' on each tree, only the fact that the bitstream is interleaved makes it work in an interleaved way. This property can be used to create a view dependent transmission[18] (where only the trees of actual interest would be transmitted), as each tree corresponds to a particular area of the base mesh.

4.4.3 Vector Decorrelation

In [12][5], it was remarked that the detail vectors were highly correlated with the surface normal at the corresponding location. This correlation could be suppressed by defining each detail vector in a local Frenet frame, that is by having local referential $\left(\hat{x}_F(c), \hat{y}_F(c), \hat{z}_F(c)\right)$ corresponding to each coefficient c, with $\hat{x}_F(c)$ being aligned with the local normal. This mean that the local $x_F(c)$ coordinate will be far more important than the local coordinates $y_F(c)$ and $z_F(c)$, which could both be coded on less bits.

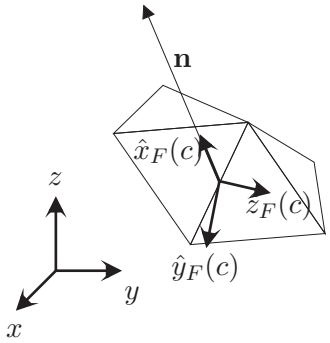

Figure 4.8: *Local Referential*

4.5 MPEG-4 Wavelet Subdivision Surfaces

As already seen in chapter 1, MPEG-4, through its AFX extension, provides a toolbox for 'wavelet' subdivisions surfaces. These tools can receive a bitstream and use it to enhance a base mesh, which can be a simple faces set node or a subdivision surfaces node. This makes the wavelet subdivision surface node a very versatile tool. That is, it can be used to enhance a 'plain' subdivision surface by adding details to its (piecewise) smooth surface; or it can be used as a mesh compression tool. The node supports different wavelets:

- Loop (as introduced in [12]);
- butterfly;
- midpoint.

The two last wavelet types could be also used in lifted mode or not. Moreover, the details could be transmitted in local or global coordinates (cf. section 4.4.3).

[18] cf. section 4.5.1

4.5.1 View Dependency

The wavelet subdivision surfaces tool has been specially tailored to be used in view-dependent applications. It can be seen in the way the bitstream is transmitted and stored: each tree could be transmitted separately bit plane by bit plane. This has a lot of advantages: for instance the client can receive the bits corresponding to certain specific areas of the mesh (e.g. where the user is currently looking at) and still display it, it also enables the transmission of meshes too big to fit in the client memory (or too complex to be processed), which can be used in mobile application (for example). However, there are also overheads in this method: information needs to be transmitted concerning a particular tree bit plane size. Which means a lot of 'meaningless' bits are sent when used in 'simple' compression mode (i.e. without view-dependency enabled).

4.5.2 Really Embedded?

Moreover, there is also another 'problem': the different components of the coordinates are totally independent (which certainly leads to a certain flexibility).

In the idea of SpiHT algorithm, the bit transmission order corresponds to the impact they have on the mean square error. But with the wavelet subdivision surfaces node of MPEG-4, especially in local coordinate mode, this is not the case. Imagine an instant that we are in local coordinate mode with $\max|x| \gg \max|y|, \max|z|$: logically x components will be coded on more bits than y or z. With current MPEG-4 bit plane order, it means that the first transmitted bit plane corresponds to the most significant bits of x component, but also to the most significant bits of y and z components, which have a totally different order of magnitude! However, with the current specification it is possible to hack the bitstream (on server side) in order to be able to transmit information such as SPiHT was intended to do, but at a great price (more bits to be transmitted, bitstream more 'tortuous').

4.5.3 Proposed Extension

In the light of the limitation of the current wavelet subdivision surfaces when used in 'simple' compression mode, we[19] propose to extend them, perhaps in a future version of AFX.

The bitstream should be composed of bit planes containing the whole forest, where each bit plane is stored in one contiguous table (for each component). Before the transmission of a bit plane, the number of bits in the plane should be also transmitted (to put in contrast with the actual view dependent MPEG-4 method where the size of the bit planes is transmitted for each tree in the bit plane).

Moreover, the second problem could also be solved by giving a relative order to each bit plane: for example, by aligning the least significant bits of each component on the same last bit plane.

[19] this proposition is based on a discussion thread over wavelet subdivision surfaces, which included K. Tack, F. Morán Burgos and P. Gioia

Chapter 5

Implementation of the Subdivision Surfaces

This chapter presents the actual implementation of the subdivision surfaces library. First some general choices are introduced, then the data structures used in the library are presented. Finally, some algorithms are explained.

5.1 Introduction

The subdivision surfaces library (SS) is written in `C++`, permitting to easily create a modular program. It is divided in different classes, which relationships are shown on figure 5.1. Classes in the same 'family' are related through inheritance. Classes beginning with '_' are internal, that is: they are not to be instantiated directly when using the library. The library is organised by layers, where each layer adds 'functionalities' to the precedent ones.

5.1.1 'Plain' Subdivision Surfaces

The first set of classes implements the 'plain' subdivision surfaces. Because each of them needs the information of the classes 'below', they are related through inheritance. Classes functions are detailed below:

- The class `_SSsTopo` performs the topological subdivision for all subdivision level $i > 0$, that is creating the graphs $G^i(\mathcal{V}, \mathcal{E})$. It:
  - works on triangles and quadrilateral meshes;
  - can handles boundaries;
  - is based on ordered 1-rings, which construction works on arbitrary mesh type;
  - can detect non-manifoldness or dangling edges;
  - can be easily extend to handle quadrilaterals;
  - caches the topological subdivision hierarchy, vertices type and 1-ring, hence saving computation time in animation mode[1].

[1] if control hull abstract graph is not changed

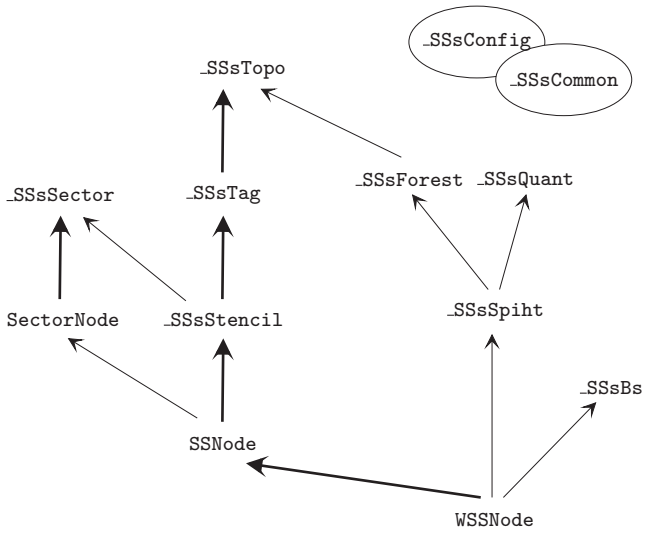

Figure 5.1: *Classes relationships.* Arrows show that a class is dependant of the pointed class, a bold arrow means a `C++` inheritance relationship. The classes `_SSsConfig` and `_SSsCommon` are in fact related to every internal classes, but their relationships are hidden not to burden the figure. The sector related classes are not yet implemented...

- The class `_SSsTag` creates tag tables for all subdivision levels (from initial tag information and vertices types). It:
  - handles MPEG-4 tags entries;
  - caches the tags table for the whole subdivision hierarchy, hence saving computation time in animation mode[2].
- The class `_SSsStencil` create the subdivision mask from the subdivided graphs, the tags and the subdivision type (Loop, butterfly or midpoint). It:
  - handles MPEG-4 stencil rules for Loop, butterfly (modified or not) and midpoint;
  - is independent from geometry (the crease angles are all considered convex and fixed at π/k, or could be overridden through the subdivision sector node), thus it is possible to cache the stencils for the whole subdivision hierarchy, hence saving computation time during in animation mode[3];
  - can create high-pass Loop's mask, so it can also be used for wavelet subdivision surfaces (not yet implemented).
- The class `SSNode` is the subdivision surface node (i.e. the *public class*), which also handles the geometrical subdivision process. It:
  - handles the geometrical subdivision processes for both approximating and interpolating scheme (with optimisations in both cases);

[2] if control hull abstract graph and initial tags are not changed
[3] if control hull abstract graph, initial tags and subdivision rules are not changed

  - handles the subdivision process of textures, colours (not yet implemented);
  - can add details, process lifting and high-pass filtering (not yet implemented) and thus can be used for wavelet subdivision surfaces.

5.1.2 'Wavelet' Subdivision Surfaces

The next set of classes implements the 'detailed' subdivision surfaces. It directly inherits the tools created for the 'plain' subdivision surfaces and adds those necessary to handle wavelet subdivision surfaces, regrouped in the following classes:

- The class `_SSsForest` creates the edges offsprings forest. It:
  - is based on MPEG-4 offsprings and trees ordering;
  - stores the tree in a bi-dimensional table;
  - creates a look-up tables establishing relationship between edges and its corresponding tree, and between trees and their corresponding base mesh edges.
- The class `_SSsQuant` creates the quantisation look-up tables. It:
  - creates two tables: one serving for initialisation (first time a coefficient is significant) and the other to update the coefficients during the refinement pass;
  - permits to the coefficient values to always be the mean value between the maximum and the minimum values they could take in the future.
- The class `_SSsSpiht` performs the SPiHT decoding. It:
  - can perform it on a single tree, the whole forest, or a part of it;
  - can dynamically remove trees from the "forest decoding", in order to perform partial decoding on it;
  - can dynamically add trees to the "forest decoding", in order to perform grouped decoding.
- The class `_SSsBs` performs the generic bitstream decoding. It:
  - can extract unsigned (any number of bits), single bit or float from the stream;
  - can also create a bitstream.
- The class `WSSNode` implements the wavelet subdivision surfaces node. It:
  - can decode the wavelet subdivision surfaces "decoder info" and mesh;
  - can perform mesh reconstruction after each call to wavelet mesh decoder;
  - can perform mesh reconstruction after each/all bit plane(s) and/or after each tree has/have been received (in partial transmission mode, not yet tested);
  - can perform a full transmission for a few bits plane, switch to partial transmission mode then switch back (not yet tested).

5.2 Data Structures

All multi-dimensional data structures use the `STL`[4] *container* `vector`. Note that *Visual C++ 6.0* does not support this feature well: in this case, it is necessary to use an auxiliary library like, for example, *stlport*[5]. There is a main advantage in its utilisation: not to require control of dynamic memory usages. The only disadvantage is its small performance overhead compared to a full custom memory management, which is easily balanced by the protection it gives against memory leaks (which can be a source of 'bugs' very difficult to locate) and its ease of use.

The data structures are indexed from 0 to $N-1$, where N is the size of the data set, which permits to be closer to the actual `C++` code. Not to surcharge the description, the exponents (corresponding to the subdivision level) are not shown.

5.2.1 Vertex Positions

The vertex positions are stored through a `vector` of `coordinate`, which is defined as follows:

$$\mathbf{c}_i = (x, y, z) \tag{5.1}$$

$$\mathbf{C} = \begin{pmatrix} \mathbf{c}_0 \\ \mathbf{c}_1 \\ \vdots \\ \mathbf{c}_N \end{pmatrix} \tag{5.2}$$

The usual operations are defined over the `coordinate` $\mathbf{c}_i$, that is scaling, addition, substraction and dot product.

5.2.2 Vertices

The vertices are not stored as such, they are the indices i of their corresponding position $\mathbf{c}_i$. These indices are used in facets list (or coordinate index) the to describe the abstract graph through the listing of its facets.

5.2.3 Facets

The facets are stored as in MPEG-4: in a `vector` of coordinate indices. The different polygons are also separated by '-1'. The `vector` looks like (where N_p is the number of polygons):

$$\mathbf{P} = \begin{pmatrix} \mathcal{P}_0 & -1 & \mathcal{P}_1 & -1 & \dots & -1 & \mathcal{P}_{N_p-1} & -1 \end{pmatrix} \tag{5.3}$$

The access of a particular element in $\mathbf{P}$ is signaled in the algorithms either by $\mathbf{P}_k$ (for the k^{th} element of $\mathbf{P}$), or $\mathcal{P}_{i,j}$ (for the j^{th} element of the i^{th} polygon).

[4] Standard Template Library, part of `C++` standard

[5] `http://www.stlport.com/`

In algorithm descriptions, the circularity propriety is used when saying that the predecessor of $\mathcal{P}_{i,0}$ is $\mathcal{P}_{i,|\mathcal{P}_i|-1}$ or that the successor of $\mathcal{P}_{i,|\mathcal{P}_i|-1}$ is $\mathcal{P}_{i,0}$.

This structure can be advantageously stored in a contiguous memory block, but one cannot easily access to a particular facet if the mesh is composed of variable size facets.

5.2.4 Ordered 1-Ring

The ordered 1-rings are stored in a "`vector` of `vector`" form. That is:

$$\mathbf{R} = \begin{pmatrix} \mathcal{R}_0 \\ \mathcal{R}_1 \\ \vdots \\ \mathcal{R}_N \end{pmatrix} \tag{5.4}$$

Where $\mathcal{R}_i$ is the 1-ring corresponding to vertex i. A particular ring element access is noted by $\mathcal{R}_{i,j}$, which means element j of the 1-ring corresponding to vertex i. The circularity propriety in case of interior vertices simply means that $\mathcal{R}_{i,0}$ and $\mathcal{R}_{i,|\mathcal{R}_i|-1}$ are neighbours.

5.2.5 Vertex Types and Tags

The vertex types are stored in the `vector` $\mathbf{T}$ of size N, where each element can take any of two values: `interior` or `boundary`. Note that even vertices type never change, permitting to have only one vertex types look-up table for the whole hierarchy.

There are two main families of tags: edge or vertex tags. The edges tags (crease) are represented for each subdivision level by a vector of rings, with each element being a boolean saying if the corresponding edge (in the 'true' 1-rings) is crease or not. For the vertex tags (crease, dart and corner), it is sufficient to have (for each) only one look-up table for the whole hierarchy.

5.2.6 Stencils

Let first define a weighted vertex index $\mathbf{w}_i = (v_i, w)$, where v_i is a vertex index (pointing to the corresponding coordinate $\mathbf{c}$) and w the associated weight.

A mask associated with a vertex v can then be written as:

$$\mathcal{M}_\mathbf{v} = \begin{pmatrix} \mathbf{w}_0 \\ \mathbf{w}_1 \\ \vdots \\ \mathbf{w}_{N_v} \end{pmatrix}, \tag{5.5}$$

where N_v is the number of vertices in the stencil.

5.3 Algorithms

Some algorithms used in the subdivision surface library are presented here. Some comments will also be made concerning their efficiency and the choice of a particular algorithm.

5.3.1 Subdivision

The *topological* and *geometrical* subdivision are done in separate procedures. It permits to separate them and thus be able to cache the result of the topological subdivision. Not to redo the computation of the subdivision topology hierarchy[6] permits to save precious time: this is very important in an animation setup where final meshes must be delivered 'quickly'.

The topological subdivision is divided in three parts: the *main loop*, the *ordered 1-rings* construction and the *splitting step*. The first one is a very simple loop (only presented for coherence sake), the second and third ones are more complex and their pseudo-code can also be found below.

The geometrical subdivision is a very simple process once the the 1-rings and stencils are created; the creation of the last and and a short discussion on the geometrical subdivision follow the topological algorithms discussion.

Main Topological Loop

This loop is very simple: for each subdivision step the 1-rings are created and the polygons are spilt. However there are different subtleties to note: the 1-rings concerning odd vertices are built in the procedure "build 1-rings" while the ones concerning the even vertices are updated in the 'split' procedures, which means that the odd vertices 1-rings do not need to be created for the last subdivision step.

Algorithm 2, *Main Topological Subdivision Loop*
1. *Ordered 1-Rings Construction*
2. **for** $i \leftarrow 0$ **to** $N_{subdivision} - 1$
3. *Triangles Quadrisection*
4. **if** $i < N_{subdivision} - 1$
5. **then** *Ordered 1-Rings Construction*

1-Ring Building

The ordered 1-rings can be used for many things:

- determination if a vertex is on the boundary or is interior;
- an easy construction of the subdivided topology;
- an easy construction of subdivision stencils (even for butterfly);
- easy geometrical subdivision.

Because with the 1-rings, one can easily access (with orientation) to the neighbours of a vertex i: vertices $(\mathcal{R}_{i,j})_j$ or edges $(i, \mathcal{R}_{i,j})_j$. Moreover, all edges of a generic abstract graph can be accessed once and only once through the 1-rings, with the simple loop shown in algorithm 3.

The ordered 1-rings construction is quite simple: first, and for each vertex, the edges (start point A, end point B) of its 1-ring are listed (as shown in figure

[6] which is always the same if no topological change are done to the control hull

Algorithm 3, *Edges Listing Through 1-Rings*

```
1.  for i ← 0 to N − 1
2.      for j ← 0 to |R_i| − 1
3.          if R_{i,j} ≥ i
4.              then do something on the edge {i, R_{i,j}}
```

5.2). Then, they are put back to back, in order to find ordered 1-rings[7]. The 1-rings are ordered in such a way that if the vertices in a facet were ordered counter-clock wisely, then the vertices in the 1-rings would also be ordered counter-clock wisely.

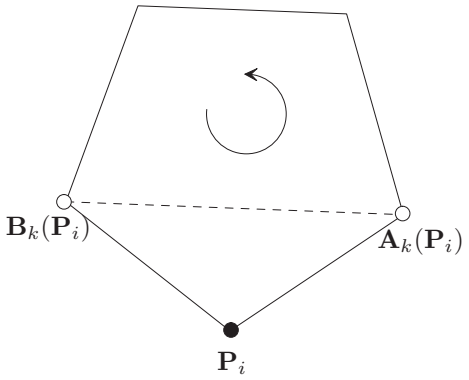

Figure 5.2: *Edges listing.* It illustrates the edges (for 1-rings construction) listing around the vertex $\mathbf{P}_i$, in this example there were already k edges in the list.

The algorithm complexity is linearly proportional to number of edges in the abstract graph and to the mean number of elements in its 1-rings.

Splitting Step

This is the core of the topology subdivision. First a new vertex is created for each old edge, by creating the new 1-rings associated with the even vertices. Then the old indexed faces set and the 1-rings (both old and new) are used to create the new polygons.

The complexity of the edges splitting is linearly proportional to the number of edges (and the mean valence in the mesh), while the polygons splitting is linearly proportional to the number of facets in the abstract graph.

New Triangles and Vertices Ordering

The orders of triangles and vertices generated from the algorithm 5 are always the same (which is pretty normal). That is, if the first triangle is given such

[7] note that the presented algorithm does not reflect the fact that even vertex 1-rings are built in the splitting step!

Algorithm 4, *Ordered 1-Rings Construction*

```
1.  list all edges in the 1-Ring
2.  A = (∅)_{0,...,N}
3.  B = (∅)_{0,...,N}
4.  for i ← 0 to |P| − 1
5.          if P_i ≠ −1
6.             then push back successor of P_i in A(P_i)
7.                  push back predecessor of P_i in B(P_i)
8.  order the 1-rings
9.  for i ← 0 to N − 1
10.         R_i = {B_{i,0}}
11.         push back A_{i,0} in R_i
12.         while A(i) ≠ ∅
13.                 for j ← 0 to |A_i| − 1
14.                         if A_{i,j} = R_{i,|R_i|−1}
15.                            then push back A_{i,j} in R_i
16.                                 push back B_{i,j} in R_i
17.                                 suppress A_{i,j}
18.                                 suppress B_{i,j}
19.                                 exit j loop
20.                         if A_{i,j} = R_{i,0}
21.                            then push front A_{i,j} in R_i
22.                                 push front B_{i,j} in R_i
23.                                 suppress A_{i,j}
24.                                 suppress B_{i,j}
25.                                 exit j loop
26.                         if B_{i,j} = R_{i,|R_i|−1}
27.                            then push back B_{i,j} in R_i
28.                                 push back A_{i,j} in R_i
29.                                 suppress B_{i,j}
30.                                 suppress A_{i,j}
31.                                 exit j loop
32.                         if B_{i,j} = R_{i,0}
33.                            then push front B_{i,j} in R_i
34.                                 push front A_{i,j} in R_i
35.                                 suppress B_{i,j}
36.                                 suppress A_{i,j}
37.                                 exit j loop
38.                 if R_{i,0} = R_{i,|R_i|−1}
39.                    then suppress R_{i,|R_i|−1}
40.                         T_i ← interior
41.                    else T_i ← boundary
```

that its first vertex $v0$ is at index t_0 in coordinate indices list $\mathbf{P}^i$:

$$\mathbf{P}^i = \quad (\ldots \quad v_0 \quad v_1 \quad v_2 \quad \text{-1} \quad \ldots) \tag{5.6}$$

Algorithm 5, *Triangles Quadrisection*

1. *Midpoint splitting*
2. $\tilde{\mathcal{R}}_i \leftarrow \mathcal{R}_i$
3. $\tilde{N} = N$
4. **for** $i \leftarrow 0$ **to** $N-1$
5. **for** $j \leftarrow 0$ **to** $|\mathcal{R}_i| - 1$
6. $v \leftarrow \mathcal{R}_{i,j}$
7. **if** $v \geq i$
8. **then** get idx such that $\mathcal{R}_{v,idx} = i$
9. $\tilde{\mathcal{R}}_{i,j} \leftarrow \tilde{N}$
10. $\tilde{\mathcal{R}}_{v,idx} \leftarrow \tilde{N}$
11. $\tilde{N} \leftarrow \tilde{N} + 1$
12. *Create the four new triangles*
13. **for** $i \leftarrow 0$ **to** $|\mathcal{P}| - 1$
14. $v_0 \leftarrow \mathbf{P}_{4\cdot i}$
15. $v_1 \leftarrow \mathbf{P}_{4\cdot i+1}$
16. $v_2 \leftarrow \mathbf{P}_{4\cdot i+2}$
17. get v_3 such that $\mathcal{R}_{v_0,v_3} = v_1$
18. get v_4 such that $\mathcal{R}_{v_1,v_4} = v_2$
19. get v_5 such that $\mathcal{R}_{v_2,v_5} = v_0$
20.
$$\begin{array}{llll} \tilde{\mathbf{P}}_{i\cdot 16} \leftarrow v_0 & \tilde{\mathbf{P}}_{i\cdot 16+1} \leftarrow v_3 & \tilde{\mathbf{P}}_{i\cdot 16+2} \leftarrow v_5 & \tilde{\mathbf{P}}_{i\cdot 16+3} \leftarrow -1 \\ \tilde{\mathbf{P}}_{i\cdot 16+4} \leftarrow v_1 & \tilde{\mathbf{P}}_{i\cdot 16+5} \leftarrow v_4 & \tilde{\mathbf{P}}_{i\cdot 16+6} \leftarrow v_3 & \tilde{\mathbf{P}}_{i\cdot 16+7} \leftarrow -1 \\ \tilde{\mathbf{P}}_{i\cdot 16+8} \leftarrow v_2 & \tilde{\mathbf{P}}_{i\cdot 16+9} \leftarrow v_5 & \tilde{\mathbf{P}}_{i\cdot 16+10} \leftarrow v_4 & \tilde{\mathbf{P}}_{i\cdot 16+11} \leftarrow -1 \\ \tilde{\mathbf{P}}_{i\cdot 16+12} \leftarrow v_3 & \tilde{\mathbf{P}}_{i\cdot 16+13} \leftarrow v_4 & \tilde{\mathbf{P}}_{i\cdot 16+14} \leftarrow v_5 & \tilde{\mathbf{P}}_{i\cdot 16+15} \leftarrow -1 \end{array}$$

Then at the next subdivision step, the triangles generated by the original triangle will be numbered from index $4 \cdot t_0$ (with v_0) in $\mathbf{P}^{i+1}$ and be ordered in the following manner (cf. figure 5.3 for abstract graph):

$$\begin{array}{rlllllllll} \mathbf{P}^{i+1} = & (\ldots & v_0 & v_3 & v_5 & \text{-1} & v_1 & v_4 & v_3 & \text{-1}\ldots \\ & & v_2 & v_5 & v_4 & \text{-1} & v_3 & v_4 & v_5 & \text{-1} \quad \ldots) \end{array} \tag{5.7}$$

The same property holds for any subdivision step, for example for subdivision step $i+2$, the triangles generated by the original triangle will be numbered from index $16 \cdot t_0$ (with v_0) in $\mathbf{P}^{i+2}$ and will be ordered in the following manner:

$$\begin{array}{rlllllllll} \mathbf{P}^{i+2} = & (\ldots & v_0 & v_6 & v_8 & \text{-1} & v_3 & v_7 & v_6 & \text{-1}\ldots \\ & & v_5 & v_8 & v_7 & \text{-1} & v_6 & v_7 & v_8 & \text{-1}\ldots \\ & & v_1 & v_9 & v_{11} & \text{-1} & v_4 & v_{10} & v_9 & \text{-1}\ldots \\ & & v_3 & v_{11} & v_{10} & \text{-1} & v_9 & v_{10} & v_{11} & \text{-1}\ldots \\ & & v_2 & v_{12} & v_{14} & \text{-1} & v_5 & v_{13} & v_{12} & \text{-1}\ldots \\ & & v_4 & v_{14} & v_{13} & \text{-1} & v_{12} & v_{13} & v_{14} & \text{-1}\ldots \\ & & v_3 & v_{10} & v_7 & \text{-1} & v_4 & v_{13} & v_{10} & \text{-1}\ldots \\ & & v_5 & v_7 & v_{13} & \text{-1} & v_{10} & v_{13} & v_7 & \text{-1} \quad \ldots) \end{array} \tag{5.8}$$

Those vertex orders are important and will be used to easily create the offspring tree needed for the wavelet subdivision surfaces. Also note, that the triangles are locally ordered in counter-clock wisely way (as shown in figure 5.3b).

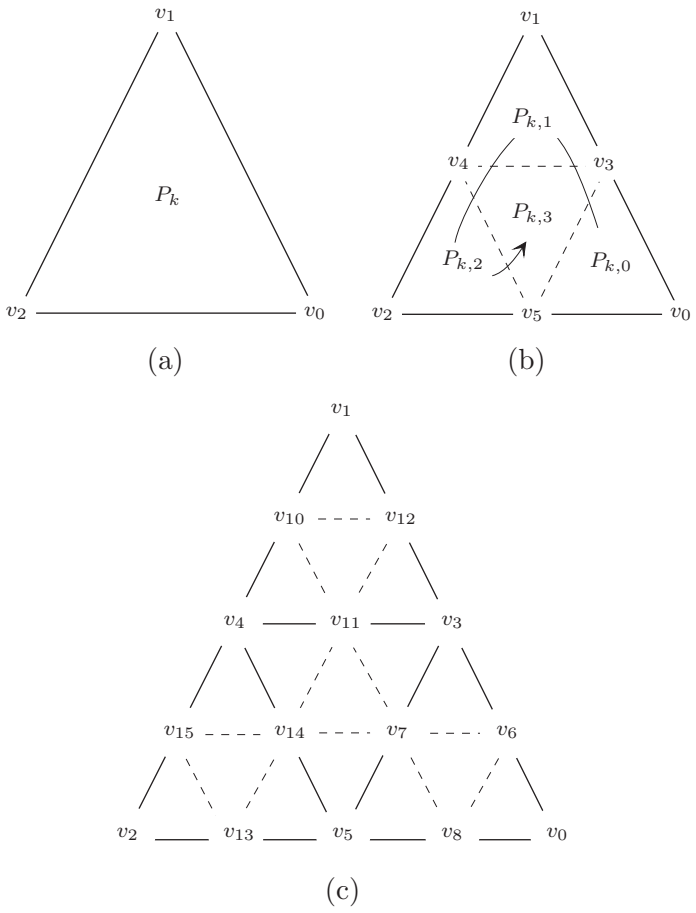

Figure 5.3: *Three generations of splitting step.* The figure (a) shows a given triangle at the subdivision step i. Figures (b) and (c) shows the triangles generated from this original triangle at step $i+1$ and $i+2$, respectively. The v_i represent the vertex indices at the corresponding level.

5.3.2 Stencil Creation

To update the geometry for level $i+1$, a stencil is created for each vertex (of the subdivision level $i+1$). These stencils give which vertices (of the subdivision level i) are taken into account (and their corresponding weights) in order to compute the position of each new odd and even (for approximating scheme) vertices. They are easily found (even for butterfly) by using the ordered 1-rings, which permits to find the neighbouring vertices of an edge (or a vertex) and their relative orientation.

The odd stencil creation uses as main loop the algorithm 3, the stencil for edge $\{i, \mathcal{R}_{i,j}\}$ is created using the 1-rings $\mathcal{R}_i$ and (for butterfly) $\mathcal{R}_{\mathcal{R}_{i,j}}$.

Loop's Stencils

It is quite obvious how one build the stencil for a Loop's even interior vertex i, since the stencil is one to one with the 1-rings and the vertex itself. When edge $\{i, \mathcal{R}_{i,j}\}$ is split (i.e. for odd vertices), all stencil vertices can also be directly from the 1-ring $\mathcal{R}_i$, as shown in figure 5.4.

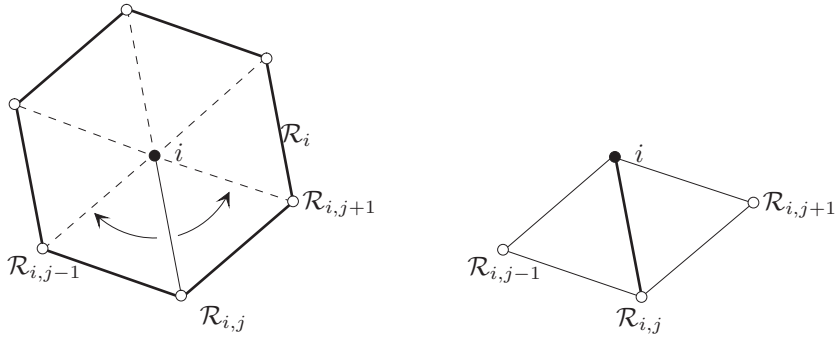

Figure 5.4: *Construction of Loop's stencil with 1-rings.*

Butterfly Stencils

Stencils construction for butterfly scheme is a bit more complex than for Loop's case. The greatest difference is that for building the stencil for edge $\{i, \mathcal{R}_{i,j}\}$, one needs the 1-ring $\mathcal{R}_i$ *and* $\mathcal{R}_k$ (with $k = \mathcal{R}_{i,j}$), as shown in figure 5.5. What costs more, computationally speaking, is to actually find the index l of the 1-ring $\mathcal{R}_k$, such that $\mathcal{R}_{k,l} = i$.

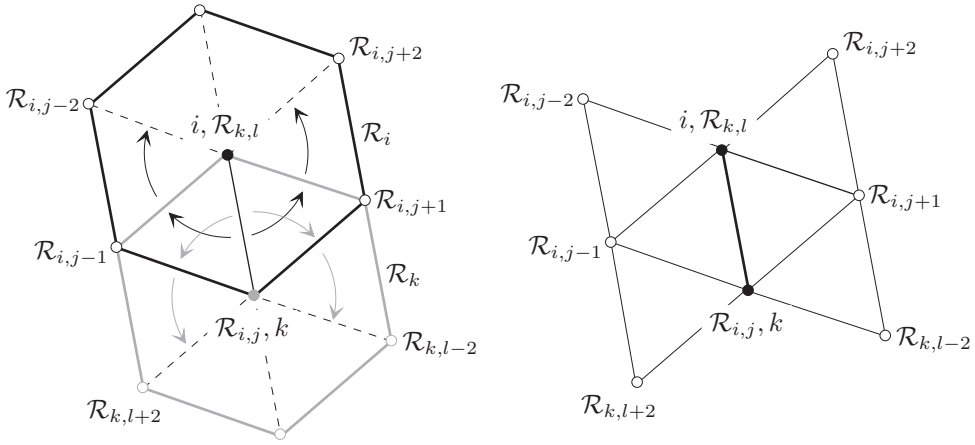

Figure 5.5: *Construction of butterfly stencil with 1-rings.* On the left, the 1-rings for the vertices i and $\mathcal{R}_{i,j} = k$ are shown in black and grey, respectively. The built stencil is shown on the right.

5.3.3 Geometrical Refinement

Once the stencils are built for each vertex v of the subdivision level $j+1$, it is very easy to create the new coordinates $\mathbf{C}^{j+1}$. It is sufficient to 'apply' the stencil on the old coordinates $\mathbf{C}^{j}$. For interpolating schemes, there is no need of a temporary coordinate since the even vertex positions are not modified, while for approximating ones the new coordinates must be stored in a temporary buffer, in order not to replace the even vertices coordinates.

5.3.4 Offsprings Trees Construction

To apply the `SPiHT` algorithm on a subdivision surfaces mesh, one needs to have the *edges offsprings trees*. The aim of this section is to construct these trees.

First of all, some data structures need to be chosen. On the figure 5.3, it could be seen that the edges at level i are one-to-one with the odd vertices at level $i+1$. Let us use the notation e^i_j for the edges at level i, with j being the index of the corresponding vertex at level $i+1$ (going from N_i to $N_{i+1}-1$: that is, the number of vertices in the mesh at level i and $i+1$, respectively). Since all vertices get different indices j, so do the edges using this scheme, it is then sufficient to use the index j of an edge (i.e. the vertex index) in the tree structure.

For example, on figure 5.3a, the edge $\{v_0, v_1\}$ corresponds to the vertex $v3$, and thus would be noted e^i_{v3} (figure 5.3b) and its offsprings are the edges corresponding to the vertex indices v_6, v_{11} and v_{13}, i.e. $e^{i+1}_{v_6}, e^{i+1}_{v_{11}}$ et $e^{i+1}_{v_{13}}$ (figure 5.3c).

This seemingly complex choice holds two important advantages:

- first, the storage of such trees is memory efficient and very simple (see figure 5.6);
- there is no need of a proxy to translate edges at subdivision level i onto their corresponding vertices at subdivision level $i+1$, since the data structure inherently contains this information.

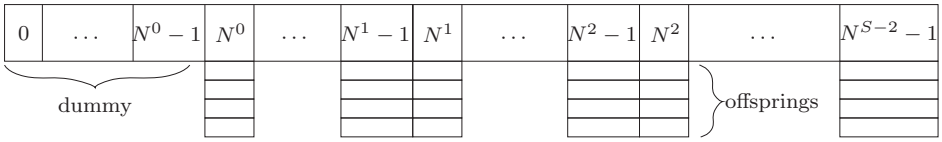

Figure 5.6: *Offsprings trees storage structure.* In this figure, the N^i correspond to number of vertices at subdivision level i and S correspond to the total number of subdivisions. Note that there is no edge corresponding to the vertices 0 to N^0-1, and that edges on the last subdivision level $(S-1)$ do not have descendants and are not represented.

MPEG-4 Edges Offsprings Ordering

The MPEG-4 standard specifies rules for ordering the offsprings and the trees permitting to have offsprings trees independent of the subdivision algorithm

used. The order of the *first* apparition of the edges in the coordinate index $\mathbf{P}^0$ gives the trees transmission order. The first level of offsprings gives a certain 'topological' orientation for each offsprings tree. That orientation needs to be conserved along all offsprings in the corresponding tree. The MPEG-4 rules for offsprings orientation (on the first level) are shown in figure 5.7, the thicker edges shown in (b) represent the offsprings of the thicker edge marked in (a), while the associated number gives the offsprings ordering. The rules are quite simple:

- the 'topological' orientation of '0' and '1' is given by the facts that the left triangle appears before the right one (in the coordinate index) and that its vertices are listed as shown by the arrow (here counter-clock wisely);
- while the 'topological' orientation of '2' and '3' is simply given by the fact that the triangle on the left appeared first.

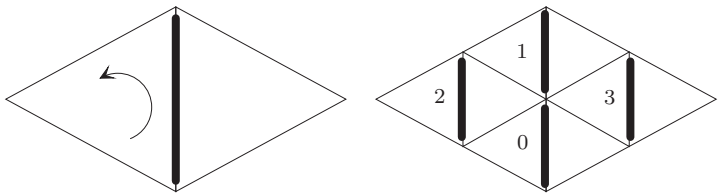

Figure 5.7: *MPEG-4 offsprings ordering.* The triangle on the left appears first in the coordinate index and its vertices ordering is shown by the arrow.

Algorithm

The first time an edge appears, a local offsprings orientation tag is associated with it. It describes the way its offsprings need to be oriented. The tag is formed by two parts:

- the tag linked with the '0' and '1' orientation can take two values. Namely, P(arallel) and A(nti-parallel) depending if the offsprings are ordered in the same direction (or not) as are listed the triangle edges[8];
- the tag linked with the '2' and '3' orientation can also take two values. Namely, R(ight) and L(eft) depending if '2' appears on the right (or left) of the triangles edges[8].

This local orientation and the order of apparition of the vertices (given is equation 5.8 and shown on figure 5.8) are used to build the offspring trees. First, there is an initialisation step where the first layer of offspring is built, which uses the coordinate index produced by the second subdivision step. It permits to find the vertices v_3 to v_5, corresponding to the base mesh edges of triangle P_k. It is also possible to find the edges corresponding to the second subdivision step triangles (the four $P_{k,i}$): edges v_7 to v_{14}, which are the offsprings of the base mesh edges.

[8]these tags are always associated with the edge appearing first in the coordinate index list and not to the second one

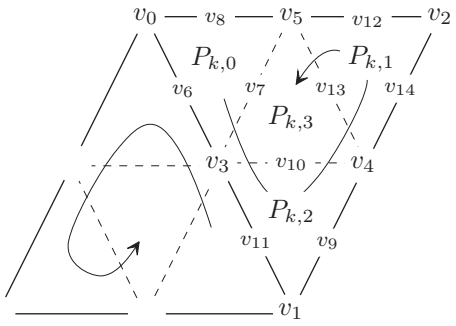

Figure 5.8:

There are two cases in building the offsprings list of v_3 (for the two other edges the description needs to be changed accordingly):

- if the offsprings list of v_3 is empty then set it to (v_6, v_{11}, v_{13}) and associate the local orientation PL to all offsprings;
- if the offsprings list of v_3 is not empty then add at its end v_{13} with local orientation AR.

Note that the local orientation is AR, in the second case, since when the list is not empty it means that the edge corresponding to v_3 has already appeared earlier in the coordinate index and also because the edge corresponding to v_{13} will appear first, in the next subdivision level, associated with triangle $P_{k,2}$, as shown in figure 5.8. The algorithm 6 presents a way of implementing the first level offspring construction. Figure 5.8 shows how the other offsprings level must be handled, and algorithm 7 presents a way of doing it. We see that with this construction local orientation could only be AR or PL (it is not possible to have AL or PR).

Algorithm 6, *Offspring Construction 1*

1. get $(v_3, v_6, v_{11}, v_{13})$
2. **if** $\mathcal{O}(v_3) = \emptyset$ (the right 'main' triangle is first in the coordinate index)
3. **then**
4. $orient(v_3) = PL$
5. $\mathcal{O}(v_3) = (v_{11}PL, v_6PL, v_{13}PL)$
6. **else** $orient(v_3) = AR$
7. $\mathcal{O}(v_3) = (\bullet, \bullet, v_{13}AR)$
8.

5.4 Demonstration Program

The demonstration program is based on `RawView`, a 3D viewing program developed at the LTS (EPFL, Laboratory of Signal Processing) by Nicolas Aspert[9].

[9]for more information: `Nicolas.Aspert@epfl.ch`

Algorithm 7, *Offspring Construction 2*

1. get $(v_3, v_6, v_{11}, v_{13})$
2. **if** $\mathcal{O}(v_3) = \emptyset$ (the right 'main' triangle is first in the coordinate index)
3. **then if** $orient(v_3) = PL$
4. **then** $\mathcal{O}(v_3) = (v_{11}PL, v_6PL, v_{13}PL)$
5. **else** $\mathcal{O}(v_3) = (v_6AR, v_{11}AR, \star, v_{13}AR)$
6. **else if** $orient(v_3) = PL$
7. **then** $\mathcal{O}(v_3) = (\bullet, \bullet, \bullet, v_{13}AR)$
8. **else** $\mathcal{O}(v_3) = (\bullet, \bullet, v_{13}PL)$

This program has been modified in order to incorporate the subdivision surfaces library. There is a command-line based interface permitting to chose the input file and the subdivision surfaces methods to apply to it, then the result is shown on the screen. Moreover, the base mesh edges are highlighted with little red spheres. The main interest of the program is the wavelet subdivision surfaces mode, where three files are taken as entry: base mesh, decoder configuration and bitstream. Then one can settle a bandwidth limitation and see in realtime the mesh reconstruction.

Chapter 6

Summaries and Conclusion

6.1 Results

This work produced a versatile subdivision surfaces library, which can be used in various applications (either MPEG-4-compliant or not). The two main subdivision surfaces tools of MPEG-4 were implemented in `C++`. The supported functionalities are:

- for 'plain' subdivision surfaces:
  - Loop, butterfly and midpoint subdivision scheme;
  - tags (crease, dart and corner);
  - animation functions;
- for wavelet subdivision surfaces:
  - midpoint and butterfly wavelets;
  - MPEG-4 stream input;
  - view-dependency;
  - lifting.

Moreover a command-line test program was created, where one can apply different subdivision schemes to a base mesh and directly visualise the result. The main test program functionalities are:

- highlighting, on final mesh, of vertices corresponding to base mesh ones;
- possibility to enter crease, dart and corner tags at runtime;
- bandwidth limitation for wavelet subdivision surfaces;
- refinement process visualisation for wavelet subdivision surfaces.

The most important contribution of this work is the clarification of the MPEG-4 standard it has helped to[1]. Some clarifications to the ('plain') subdivision surfaces node were proposed, and accepted (the list is not exhaustive):

- clarification on tags overriding;
- addition of butterfly and midpoint schemes (to be consistent with wavelet subdivision surfaces);
- clarification of the special rules (near extraordinary border vertices) for Loop.

[1] note that this clarification was done in conjunction with K. Tack (IMEC, Belgium) and F. Morán (UPM, Spain) who supervised the clarification process

However, the most important clarification process was done for the wavelet subdivision surfaces node, where a lot of important changes were done (the list is not exhaustive):

- change in the bitstream loops order;
- clarification of the quantisation process (conversion of the SPIHT integer in detail floats);
- clarification of the edge offsprings rules;
- correction of the lifting process (suppression of Loop scheme for a possible lifted scheme);
- correction of some local coordinate details.

Moreover, this work also proposed the addition of new MPEG-4 tool for progressive 3D mesh compression, which would performs far better for non view-dependent applications.

6.2 Future Work

There is still a lot to do concerning the created library:

- add a sector node;
- add Loop wavelet;
- add local coordinate support;
- extensive test of view-dependency;
- integration in MPEG-4 reference software.

Also, the proposed MPEG-4 extension needs to be studied in more detail, in order to propose a competitive tool. Finally, there is still a lot to discover in the domain of 3D mesh compression, which is only at its hatching process.

6.3 Conclusion

Even if the main goal of the project was implementation of MPEG-4 subdivision surfaces, this paper can be usefully utilised to introduce subdivision surfaces to people not specialised in the particular fields of 3D modelling or 3D compression. Moreover, the created library can be used in a specialised 3D decompression program or can be integrated in a complete MPEG-4 3D-player.

On a personal point of view, I learned many things with this project. Especially about subdivision surfaces, which were a totally unknown field to me or even OpenGL and `C++`. Also, working in another country was a very appealing experience, even if it was not always easy. I would also like to thank all the people that helped me in this "spanish experience", especially: Carmen, Caroline, David, Francisco, José, María Jose, Marie Carmen, Nicolas, Sophie, Tore, Touradj and many others...

Madrid, February 2003,

Yannick Maret

Bibliography

[1] H. Biermann, A. Levin and D. Zorin *Piecewise-smooth subdivision surfaces*, ACM SIGGRAPH'00 proc., 113-120, July 2000

[2] N. Dyn, D. Levin, *A Butterfly Subdivision Scheme for Surface Interpolartion with Tension Control*, ACM Transcations on Graphics, Vol. 9, No. 2, pp. 160-169, April 1990

[3] M. Eck, et al. *Multiresolution Analysis of Arbitrary Meshes*, ACM SIGGRAPH'95 proc., pp. 173-182, August 1995

[4] J. D. Foley et al. *Computer Graphics: Principles and Practice*, 2nd edition in C, Addison-Wesley, 1997

[5] F. Morán, *Hierachical modelling of 3D objects with subdivision surfaces*, Ph.D. thesis, Universidad Politécnica de Madrid, October 2001

[6] F. Morán et al., *Subdivision Surfaces in MPEG-4*, IEEE ICIP proc. 2002, vol. III, pp. 5-8, September 2002

[7] H. Hoppe, *Progressive Meshes*, ACM SIGGRAPH'96 proc., pp. 99-108, August 1996

[8] ISO/IEC 14772-1, *The Virtual Reality Modeling Language (VRML)*, `http://www.web3d.org/fs_specifications.htm`, 1997

[9] ISO/IEC 14496, *Overview of the MPEG-4 Standard*, N4668, March 2002

[10] ISO/IEC 14496-2, *Coding of Audio-Visual Objects: Visual*, December 2002

[11] ISO/IEC 14496-16, *FDIS of ISO/IEC 14496 Part 16: Animation Framework eXtension (AFX)*, N5397, December 2002

[12] A. Khodakovsky, P. Schröder and W. Sweldens: *Progressive Geometry Compression*, ACM SIGGRAPH 2000 proc., pp. 271-278, July 2000

[13] C. Loop, *Smooth Subdvision Surfaces Based on Triangles*, Master thesis, University of Utah, August 1987

[14] M. Lounsbery, et al. *Multiresolution Analysis for Surfaces of Arbitrary Topological Type*, ACM Transactions on Graphics, No. 1, pp. 34-73, January 1997

[15] A. Lee et al., *MAPS: Multiresolution Adaptive Parameterization of Surfaces*, ACM SIGGRAPH'98 proc., pp. 95-104, July 1998 Jarek

[16] W. S. Massey, *Algebraic Topology: an Introduction*, 8th ed., Springer-Verlag, 1989

[17] J. Rossignac, *Edgebreaker: Connectivity compression for triangle meshes*, IEEE Transactions on Visualization and Computer Graphics 5-1, pp. 47-61, March 1999

[18] J.M. Shapiro, *Embedded image coding using zerotree of wavelets coefficients*, IEEE Transactions Signal Processing, vol. 41, pp. 3445-3462, December 1993

[19] P. Schröder and W. Sweldens, *Spherical Wavelets: Efficiently Representing Functions on the Sphere*, ACM SIGGRAPH 95 proc., pp. 161- 172, August 1995

[20] A. Said, W.A. Pearlman, *A New Fast and Efficient Image Codec Based on Set Partioning in Hierarchical Tree*, IEEE Transactions on Circuits and Systems for Video Technology, Vol. 6, June 1996

[21] W. Sweldens, *The Lifting Scheme: a Construction of Second Generation Wavelet*, July 1995, revised November 1997

[22] C. Touma and C. Gotsman, *Triangle mesh compression*, Graphics Interface'98 proc., pp. 26-34, June 1998

[23] G. Taubin and J. Rossignac, *Geometric Compression Through Topological Surgery*, ACM Transaction on Graphics 17-2, pp. 84-115, April 1998

[24] D. Zorin, *Subdivision and Multiresolution Surface Representation*, Ph.D. thesis, CalTech, September 1997

[25] D. Zorin, P. Schröder et al., *Subdivision for Modeling and Animation*, ACM SIGGRAPH 2000 course notes, July 2000