Geometry Videos
A New Representation For 3D Animations

by

Héctor Manuel Briceño Pulido

Bachelor of Science, Electrical Engineering and Computer Science
Massachusetts Institute of Technology, 1997
Master of Engineering, Electrical Engineering and Computer Science
Massachusetts Institute of Technology, 1997

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2003

© Massachusetts Institute of Technology 2003. All rights reserved.

Author . . . . . . . . . . . . . 
Department of Electrical Engineering and Computer Science
September, 2003

Certified by . . . . . . . . . . . .
Leonard McMillan
Associate Professor
Thesis Supervisor

Certified by . . . . . . . . . . . .
Seth Teller
Associate Professor
Thesis Supervisor

Certified by . . . . . . . . . . . .
Steven Gortler
Associate Professor (Harvard University)
Thesis Supervisor

Accepted by . . . . . . . . . . . .
Arthur C. Smith
Chairman, Department Committee on Graduate Students
Geometry Videos
A New Representation For 3D Animations

by

Héctor Manuel Briceño Pulido

Submitted to the Department of Electrical Engineering and Computer Science on September, 2003, in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Electrical Engineering and Computer Science

Abstract

Animations of three-dimensional computer graphics are becoming an increasingly prevalent medium for communication. There are many sources of 3D animations including physical simulations, scientific visualizations, and classic key-frame animations generated by an artist. There are even computer vision systems available today that are capable of capturing 3D time-varying geometric models. In this research, we develop a new representation for an important class of 3D animations, specifically time-varying manifolds. We call this representation a “Geometry Video.” At present, a viewer of a 3D animation must either have a similar simulation or animation infrastructure to the animation’s producer, or the producer must create a video from a predefined set of viewpoints. Geometry videos provide the ability to encode and transmit a time-varying mesh in a generic, source-independent, and view-independent format.

Geometry videos are created by constructing a global two-dimensional parametrization of a manifold over a rectangular domain. Time sequences of such parametrizations are particularly well-suited to compression using methods akin to video compression. This dissertation develops the techniques necessary to encode and compress arbitrary 3D manifold animations. A system is presented for converting animations into geometry videos as well as compressing and decompressing such representations. We also discusses the problems, design-parameters, and trade-offs associated with building such a system.

Thesis Supervisor: Leonard McMillan
Title: Associate Professor

Thesis Supervisor: Seth Teller
Title: Associate Professor

Thesis Supervisor: Steven Gortler
Title: Associate Professor (Harvard University)
Acknowledgments

If I have seen farther than others, it is because I was standing on the shoulder of giants. *Sir Isaac Newton*

This work would have not been possible without the help, encouragement, support of numerous people. I would like to thank the following people for being part of this journey called life and specifically this trail called thesis:

- My advisor Leonard McMillan, being there for me during the whole process and keeping the bow of the boat facing north. My other committee members, Seth Teller and Steven Gortler for reading, brainstorming, and commenting on the thesis.

- My Mother and Father for believing in me and pushing me through the hard periods. My two sisters and my brother for supporting the cause.

- Rob Jagnow who on short notice proofread the chapters of the thesis, it would have been hard to finish on time without him. Ray Thouis for being the general lab guru and having answers to problems not found on the internet.

- All members, past and present, of the MIT’s Computer Graphics Groups. Including my office mates Osama, Wojciech, Peter for withstanding me and my rambles (related and non-related to research).

- My fellow roommates Robert Zeithammer and Martin Zalezak for the discussions related to the thesis and other issues limiting the progress of the thesis.

- Matthias Müller and Daniel Vlasic for providing me with base animation sequences used in this thesis.

- I would also like to thank the “unknown helper” (alluding to the unknown soldier), all those people who helped but whose memory elude me or I can’t remember how to spell their names.
Dedication

A mi Madre
A mi Padre
A mis Hermanos
A mi Familia
## Contents

1 Introduction .................................................. 11
   1.1 Motivation .................................................. 12
   1.2 Thesis Statement ........................................... 14
   1.3 Contributions ............................................. 14
   1.4 Thesis Overview .......................................... 16

2 Previous Work .................................................. 17
   2.1 Static Mesh Compression .................................... 17
   2.1.1 Maintain Original Connectivity ......................... 18
   2.1.2 Re-sample The Input Mesh .............................. 21
   2.2 Animation Representation .................................. 23
   2.2.1 Survey of Animation Primitives ......................... 23
   2.2.2 Compression of Time-Dependent Geometry ............... 24
   2.2.3 VRML/MPEG4 .............................................. 25
   2.2.4 Representing Animations by Principal Components ...... 26
   2.2.5 Temporal and Spatial Level of Details for Dynamic Meshes .. 26
   2.2.6 Streaming 3D Animations over Lossy Channels .......... 27
   2.3 Summary .................................................. 27

3 Background .................................................... 29
   3.1 Parametrization ............................................. 29
      3.1.1 Local parameterizations ............................ 30
      3.1.2 Global parametrizations ............................ 31
   3.2 Geometry Images ........................................... 35
      3.2.1 Cutting .............................................. 35
3.2.2 Boundary Parametrization ........................................... 37
3.2.3 Interior Parametrization ............................................. 38
3.2.4 Rasterization ......................................................... 39
3.2.5 Compression and Stitching ........................................... 39
3.2.6 Limitations ........................................................... 39
3.3 Summary ................................................................. 41

4 Adapting Geometry Images to Animation .......................... 42
  4.1 Consistency of Cut and Parametrization ....................... 42
    4.1.1 Choice of Parametrization ..................................... 45
    4.1.2 Global Cut ...................................................... 46
    4.1.3 Sampling and Rendering ....................................... 49
  4.2 On Topology and Connectivity Changes ......................... 49
  4.3 Summary ............................................................... 50

5 Adapting Geometry Images to Compression ....................... 51
  5.1 Spatial Coherence .................................................... 52
  5.2 Temporal Coherence .................................................. 54
  5.3 Transformations ........................................................ 55
    5.3.1 Global Transformations ....................................... 56
    5.3.2 Local Transformations ......................................... 57
  5.4 Frame Types .......................................................... 61
  5.5 Summary ............................................................... 64

6 System ................................................................. 65
  6.1 Encoder/Decoder ..................................................... 66
    6.1.1 Pre-process ..................................................... 67
    6.1.2 I-Frames ......................................................... 73
    6.1.3 P-Frames and Transformations ............................... 75
    6.1.4 B-Frames and Blending ....................................... 78
  6.2 Representation ........................................................ 81
  6.3 Parameters ............................................................ 83
  6.4 Adapting to Network Environment ................................. 88
7 Results

7.1 Comparing Shapes and Compression ........................................... 91
7.2 Comparison to Static Approaches .............................................. 93
7.3 Compression Results ................................................................. 97
7.4 Cuts/Parametrization ................................................................. 104
7.5 Predictive Coding ................................................................. 112
    7.5.1 Predictive Coding: P-Frames and Transforms ...................... 114
    7.5.2 Predictive Coding: B-Frames and Blending ......................... 116
7.6 Level of Detail ................................................................. 122
7.7 Timings ........................................................................ 123
7.8 Summary ........................................................................ 123

8 Conclusions and Future Work .................................................. 128

8.1 Conclusions ........................................................................ 128
8.2 Future Work .............................................................. 130

A Wavelet Coefficients .......................................................... 134
List of Figures

1-1 Geometry Video Example ........................................... 15
2-1 Parallelogram Predictor Rule ....................................... 19
2-2 Difference between Connectivity and Parameter Information ........ 22
2-3 Animation Primitives .................................................. 24
3-1 Subdivision Example .................................................... 31
3-2 Types of Parametrizations ............................................. 33
3-3 Geometric Stretch Metric ............................................. 35
3-4 Iterative Cutting Algorithm ......................................... 37
3-5 Geometry Image Pipeline ............................................. 38
3-6 Example of Limitation of Geometry Image Approach .............. 40
4-1 Example of Local and Global Cut .................................... 44
4-2 Sample Global Cut ..................................................... 47
5-1 Compression Using Wavelets vs. DCT ................................ 53
5-2 Global Transformation Snake Example ............................... 57
5-3 Geometry Image Partitioned into Patches ............................ 58
5-4 Local Transformation 1D Example .................................... 59
5-5 Example Weights for Local Transformations ....................... 60
5-6 Local Transformation Snake Example ................................ 61
6-1 Fixing Zero-area Triangles ........................................... 71
6-2 Fixing Corners .......................................................... 71
6-3 I-Frame Encoder/Decoder .............................................. 73
6-4 Regular Triangulation of the Geometry Images .................... 75
List of Tables

6.1 Comparison of MPEG and Geometry Videos .................................. 67
6.2 System Parameters ........................................................................ 89
7.1 Geometry Video Subsystem Timings ............................................. 124
A.1 Villa1810 Wavelet Coefficients .................................................. 134
Chapter 1

Introduction

The problem of representing animated models is of significant importance in the field of computer graphics. There are several applications for animated geometry, such as computer-generated movies, computer games, and recording of physical simulations. Due to the unceasing increase in computational power and memory, it is becoming increasingly easier to acquire such animated models. Some systems, such as Matusik et al [MBR+00], are already able to acquire, construct, and render animated geometry in real-time. For such reasons, the use and sharing of 3D animations could become as common as video is today.

Currently, when users want to exchange or share 3D animations they must either have common simulation infrastructure or a animation system (software and/or hardware), or render a video of the animation from specific viewpoints. The latter approach provides only a limited predetermined view of the animation, while the former requires considerable computation, which might not support interactive rendering rates. Another possibility is to transmit a separate polygonal mesh for each frame of the animation.

In this thesis, I propose a novel representation for 3D animations. This representation can encode 3D animations regardless of how they were generated. The proposed technique builds upon a method that maps 3D meshes (frames in the animation) to 2D images. This mapping introduces some artifacts, which in general are not visually distracting (although for certain meshes some of these artifacts can be unacceptable). We adapt the video compression methods used for video streams to image sequences corresponding to a time-sequence of 3D meshes. By leveraging current video standards, this method can adopt many of the features of current video compression methods. Examples include network streaming,
level of service guarantees, layered coding, high compression rates, among others.

1.1 Motivation

Three dimensional animations are an increasingly important media type. They are becoming more prevalent as computers become more powerful. There are various uses for 3D animations. They are used as teaching aids. The laws of physics can be explained by showing students examples of how objects behave under different forces. The realism of these animations make them more convincing to the student. Simple user interactions such as being able to navigate in 3D space to change the point of view make them more engaging for the student. Similarly, animations can also be used as a visualization tool. Sensor data can be used to generate an animation for visualizing changes in the earth crust. To gain a better understanding of these events, it will be necessary for the simulation to be seen from different viewpoints and by different scientists. Without a visualization, interpreting such data may be impossible.

Movies also make use of 3D animations, both as an end in itself and as a tool for digital effects. There are movies based entirely on synthetic animated characters. These characters are fundamentally human-like in appearance and in action, but have various idiosyncrasies that distinguish them from reality. Computer graphics animations have been used to replace characters in dangerous stunts. The incredible realism currently achievable with modern rendering methods, make computer generated stunts nearly indistinguishable from their real counterpart. These computer generated stunts are used when the stunts are physically unrealizable or dangerous. Computer games also make use of animations to generate realistic scenes and plots; the more realism, the more engaging the game will be. Unlike movies, where the director will fix the viewpoint of the animation, games give the player greater flexibility on changing the viewpoint throughout the game. Furthermore, games have to generate the animations in real time, they do not have the flexibility of spending hours simulating and rendering each a frame that movies have.

These 3D animations are generated using a multitude of techniques. Many synthetic characters are generated from scratch by skilled artists. Their motion is commonly articulated in one of three ways.

Motion capture systems can be used as a motion template for the synthetic character.
Sensors are placed at various joints of an actor and their position is tracked and recorded through time. The person can act the intended motion of the synthetic character. From the position and relation of the sensors placed on the person, the motion can be mapped onto the synthetic character.

Alternatively, the motion of the character can be described by a sequence of key frames. Each key frame describes an important intermediate position in the motion sequence. For example, a person walking might be described with key frames corresponding to a person with the foot in front of him, the foot on the ground, and a frame with the foot behind him. Then, through an interpolation process, all the frames in between can be automatically generated. Sometimes the generated frames are manually tweaked for added realism or to correct the interpolation process.

More recently, physical simulation systems are able to generate realistic motions from a full description of a character (skeleton joints, mass, etc). These motions are generated by simulating the laws of physics and searching for solutions to particular problems (e.g. walking). These motions are not limited to single tasks like running, but can include a combination of tasks like getting into a car.

We would like to devise a system to where these 3D animations, once generated, can be shared or transmitted across the network in a generic format.

This can be achieved by creating a video of the animation sequence. Video compression technology is mature and sharing the resulting video files straightforward. Unfortunately this approach limits interactivity. This reduces the exploration options for the user. Furthermore, all geometrical information is lost when the 3D geometry is flattened onto a 2D video. The user will perceive the geometry only by the “director’s” choice of viewpoint and lighting conditions of the object.

Another way to share animations is to use the approach used for computer games. The animations can be generated in real-time by software. The advantage of this approach, is that the user can now interact with the animation, replaying certain sequences and changing the viewpoint. Unfortunately, some of these simulations cannot be generated in real time; for example, physical simulations may require solving complex equations of tens of variables. Furthermore, the data set required for these animations may be too large. Scientific visualizations may require sensor data from hundreds of sensors will millions of data samples. Additionally, the software needed to “play” these animations will be very
data-specific.

Alternatively, one could use explicit three-dimensional models to represent each frame of an animation. Frames in 3D animations correspond to a 3D object (or a collection of 3D objects). The traditional representation of a 3D object is a polygonal mesh representation. A mesh is represented by a collection of points in 3D space and a collection of faces (set of points) describing a surface over these points. Standard mesh compression techniques can be used to encode and compress each frame in the 3D animation. This approach can be expensive for large meshes. Additionally, this approach does not take advantage of the geometric redundancy that exists between frames. Finally, this approach would lack some of the capabilities that mature video standards have, like support for network streaming and support for variable quality and level of service.

1.2 Thesis Statement

By representing manifold 3D objects using a global 2D parametrization (mapping) it is possible to use existing video techniques to represent 3D animations.

This method has many advantages over current approaches. It supports level-of-detail rendering – the frames can be rendered at a coarse or fine resolutions from the same representation. It is amenable to hardware parallelization and optimization. It provides a representation whose resource and computational requirements for rendering can be known exactly, thus providing level of service guarantees. It achieves high compression by using similar techniques to video encoding. It can be streamed over networks to remote clients. This is a novel representation for 3D animations that allows the leverage and application of many 2D video techniques to the realm of 3D. Figure 1-1 shows example frames from a Geometry Video.

1.3 Contributions

This thesis makes the following contributions:

- I adapt a previous parametrization technique that maps 3D geometries to images for the purpose of representing 3D animations. This process involves cutting a mesh and parametrizing (mapping) it onto a square.
I investigate the tradeoffs between various cuts and parametrizations: having a fixed cut applied to all frames of the animations versus applying a customized cut to each frame; and having a fixed parametrization for the whole animation versus varying the parametrization throughout the animation.

Within this context, I explore how to take advantage of the temporal redundancy that exists throughout an animation to achieve high compression. This affects the choice of cuts and parametrization.

Standard Video encoders make use of “Motion Compensation” techniques to encode frames more compactly. We observe that affine transformations, can serve a similar function in Geometry Videos. In contrast to motion compensation, which involves a search problem, finding the optimal affine transformations (in the least squares sense) has a closed form solution.

I present an entire system for encoding 3D animations as Geometry Videos. I also implement an interactive player to view them.

I discuss the set of parameters used to control the quality of the Geometry Videos, and how to tweak them for various kinds of animations.
• Finally, I compare this technique to other mesh compression and mesh animation methods to validate this approach.

1.4 Thesis Overview

This thesis is organized as follows. In Chapter 2 describes the related work to encoding static meshes and representing time-varying geometries; this highlights the limitations of current approaches.

Chapter 3 presents the concepts and definitions helpful for the understanding of our approach, primarily the Geometry Image algorithm (that maps a 3D mesh onto an image). We also review parametrizations and the geometric-stretch metric used for our parametrizations.

Chapter 4 describes how to adapt the Geometry Image approach to 3D animations, this forms the foundation of Geometry Videos.

Chapter 5 discusses how to compress the 3D animations by taking advantage of the temporal redundancy that exists throughout the animation. We use predictive encoding along with affine transformations for this purpose.

Chapter 6 describes the complete system including all the implementation details and intuition for our choice of parameters.

Chapter 7 presents our results: we compare our technique with other approaches. We compare Geometry Images to other static compression methods and we compare Geometry Videos to another time-varying geometry compression technique. We then proceed to validate our choice of cuts and parametrization for Geometry Videos. We evaluate the performance of different parameters and settings and how they affect compression. We conclude by reporting the time performance of our implementation.

Chapter 8 describes future directions for our work and concludes this thesis.
Chapter 2

Previous Work

In order to formulate a new representation for time-varying geometries, one must be familiar with techniques for compressing static meshes and with alternative methods for compressing time-varying geometries. The former area helps us to understand how we can take advantage of spatial coherence and where the costs and difficulties lie in compressing static meshes. The later points to techniques we can use and reveals the difficulties that arise when working with meshes that change throughout time.

This chapter is divided into two sections. Section 2.1 reviews static mesh compression techniques. Section 2.2 looks at the few techniques that have been proposed for encoding time-varying geometries; We conclude by looking at the issues that arise when adapting static mesh-compression schemes to animations.

2.1 Static Mesh Compression

There are many ways to compress static meshes. A typical 3D model consists of a set of vertices and a set of faces. Three-dimensional vertices are commonly specified by coordinates relative to an assumed Euclidean frame and often have other attributes associated with them, like texture coordinates, material indices, color, and surface normals. A face is a set of vertices; without loss of generality we'll consider only triangular faces, i.e. faces with only three vertices.

There are two common views on compressing meshes. The first one, compresses exact meshes, maintaining the connectivity of the original mesh. A second alternative view compresses the object that the mesh represents. In this view, the mesh is just one of many
ways to represent the same object. Therefore, changing the connectivity or re-sampling the input mesh, may not matter as long as it represents the same object [KSS00].

We will first look at some of the methods for encoding meshes while maintaining connectivity. These methods show that while we can encode connectivity information very well, vertex location information still forms the bulk of the compressed information. We then will look at methods that re-sample the input mesh and see how we can further compress vertex location or geometry information.

### 2.1.1 Maintain Original Connectivity

Here we illustrate the previous work on static mesh compression. A common thread on improving the overall compression is choice of a predictor function for predicting vertex positions. This permits the representation of vertex location with less information.

Deering [Dee95] was the first to propose a method for compressing triangular meshes. His approach was motivated by the need to reduce the amount of information sent to the graphics accelerator hardware. His approach assumed generalized triangle strips. Triangle strips are a compact face list representation that minimizes repetition of vertices for describing a list of vertices of adjacent faces; Deering uses special instructions in the strips to reduce the repetition vertex indices. He points out that it is unnecessarily costly to use 32-bit floating-point values for vertex coordinates as they allow the representation of vertex positions from small atoms to large galaxies. He instead suggests 16-bits fixed-point numbers per coordinate, which is visually equivalent for more realistic problems. Currently, it is common for compression schemes to quantize vertex coordinates with anywhere from 10 to 16 bits. More importantly, vertex coordinates can be further compressed by using delta encoding and then Huffman encoding those deltas. To encode the current vertex position, the previous vertex position on the strip is subtracted, and only this difference is encoded.

Taubin and Rossignac [TR98] propose an encoding method called Topological Surgery. It works by building a minimum spanning tree of the mesh connectivity including all vertices to be compressed. Using this tree, it builds a triangle spanning tree. With this and some additional information, it encodes the connectivity and geometry very well. These spanning trees define an ordering of the vertices. Like Deering, Taubin and Rossignac use a predictor to reduce the geometry information. In their case, they use the $K$ previous visited vertices to build a prediction.
To encode the position of $V_n$, one only needs to encode $\epsilon$. The depth of $K$ and the choice of $\lambda$’s influences the compression rate. The $\lambda$’s are chosen as to minimize the least square error of $\epsilon$ over all the vertices.

Kronrod et al. [KG02] point out that these kind of predictors are suboptimal: every vertex only helps in the prediction of $K$ other vertices, which does not leave much freedom in building better prediction trees.

Touma and Gotsman [TG98] presented an alternative encoding method. Their encoding is based on the observation that “vertices incident on any mesh vertex may be ordered i.e. sorted in a clockwise order in a consistent manner.” They use this fact and special instructions to encode a consistent traversal of the mesh. Additionally, they use a more accurate predictor to reduce the encoding for vertex positions. They call this predictor the “parallelogram rule.” The current vertex is predicted by forming a flat parallelogram based on the previous decode triangle. They remove the flat constraint and further improve the prediction by estimating the crease-angle or curvature of the parallelogram based on the previous triangles. This is illustrated in figure 2-1. This method is able to compress vertex coordinates with 8-bit quantization (24 bits total since there are three coordinates per vertex) at about 8 to 10 bits per vertex. This is about a three-to-one compression, considering that a vertex has three coordinates: $x, y, z$.

$$V_n = \lambda_1 V_{n-1} + \lambda_2 V_{n-2} + \ldots + \lambda_K V_{n-K} + \epsilon$$ (2.1)

**Figure 2-1:** Figure taken from [TG98]. “The parallelogram rule with and without crease angle prediction. Filled triangles have already been decoded. Vertex $r^p$ is the prediction of the true vertex $r$ using the parallelogram rule $r^p = v + u - w$, and $rr_r\sigma^p$ is the prediction adopting the crease angles between triangles $(s, v, w)$ and $(s, v, t)$.”
Methods have also been proposed that allow the decompression of a mesh hierarchically (at many resolutions) [Hop96]. This allows the presentation of low resolution versions when the mesh is seen at a distance. More importantly, on networking environments, it allows a user to see a coarse approximation of the mesh quickly; with time, the quality of the mesh is improved as more information is received. This additional feature has been implemented with little cost on the overall compression ratios. These methods work by starting with a coarse mesh and encoding the addition of some geometric primitive (e.g. edges) to increase the number of vertices and triangles. This opens new possibilities in building predictors. One can predict the vertex locations on a finer mesh based on the location of vertices on a coarse representation of the whole mesh, therefore making the difference between the predictor and the vertex location smaller. For example, in [PR00], Pajarola et al. build a predictor for a vertex in a finer mesh using all the neighbors within a topological distance of two in a coarser mesh.

These methods can compress vertex position with 8 to 12 bits of quantization per coordinate with 8-15 bits per vertex. Connectivity information is smaller and has been studied more extensively; it is usually compressed at 2.0 bits per vertex, and in very regular meshes, it drops down to 0.1 bits per vertex.

So far we have mostly focused on the predictability of vertex position since this forms the bulk of the compressed mesh data. Another important aspect to consider is the compression of texture coordinates. In general, this is not a problem in animations because the parameters of the texture mappings remain the same. However, it is possible that if large deformations exist on the mesh, a new texture mapping might be desirable. In general, most approaches do not delve deeply into the compression of texture coordinates and propose the use of similar predictors to those used for vertex positions. Isenburg et al. considered the encoding of texture coordinates [IS02]. In their proposed scheme, they obtain compression rates of 5.9 bits per vertex for texture coordinates (two of them) compressed at 10 bit quantization for multiple connected components (similar number of texture coordinates and vertices). For a single large component with multiple texture coordinates per vertices (2.1 on average), they compress texture coordinates at 10 bit quantization at 10 bits per vertex.

Previous approaches are considered lossless after quantization, meaning the exact quantized vertex position is recovered, as is the exact connectivity. Alternatively, if we allow some loss in accuracy of the vertex position, greater compression can be achieved.
Karni and Gotsman developed an alternative method for encoding vertex positions [KG00]. Instead of looking at differences, they approached the encoding from another perspective by applying spectral (frequency) analysis to mesh data. Spectral analysis is similar to the Fourier analysis already used in JPEG for image compression which is well understood and achieves good compression. They compute the eigenvectors of the Laplacian Matrix (derived from the connectivity information) to obtain a spectral basis to represent vertex positions. The vertex positions are encoded as a weighted sum of these basis functions. This is a progressive approach. The bases are already ordered by importance, so utilizing more bases results in a finer reconstruction. They encode the connectivity using standard techniques. Vertex positions are recovered with a small loss due to precision errors, but more significantly because the coefficients are quantized. This is a different progressive scheme, in that it keeps the original connectivity at all times, but the vertex positions are refined as more basis functions are used in their computation. Due to the computational expense of calculating the basis functions, the original mesh is partitioned into smaller charts, and the approach is applied to each individual chart. At low bit-rates, the quality of this approach is similar to that of [TG98] while only requiring one quarter the information.

2.1.2 Re-sample The Input Mesh

Better compression rates can be achieved by accepting some loss in the vertex positions or by approximating the input mesh with another mesh. In this section, we look at two methods that achieve higher compression rates by re-sampling the original mesh. This corresponds to “sacrificing” the original connectivity.

Khodakovsky et al. [KSS00] provide a nice argument for discarding original connectivity information. The classical mesh representation describe mesh objects as being composed of geometry (vertex position) and connectivity information (face list). They propose a slightly different view. The propose to view geometry information as two parts: a different geometry information and geometry parameter information:

The parameter information captures where the samples are located within the surface while the geometry information captures the geometry independent of the sample locations used. So far parameter and geometry information were treated together.
Figure 2-2 illustrates the difference between parameter and connectivity information.

![Figure 2-2](image)

**Figure 2-2**: Figure taken from [KSS00]. Three spheres with 2562 vertices. The sphere on the left is semi-regular (mostly valence 6 vertices), has no parameter and very little connectivity information. The figure in the middle has the same connectivity as the one on the left but has more parameter information. The figure on the right is irregular - it has much parameter and connectivity information.

It is possible to re-parametrize a mesh without changing the connectivity. This could yield better compression. But, if while one reparametrizes the mesh, one changes the connectivity to make the mesh semi-regular or regular, then many techniques like wavelet analysis are easier to apply and can yield better results.

There are many techniques for re-parametrizing meshes [EDD+95, LSS+98, LMH00, GVSS00]. Here we just point to the work by Lee et al. In [LSS+98], they present a method to generate a smooth semi-regular parametrization from 2-manifold meshes of arbitrary genus. It consists of an irregular base mesh, which can be thought out as forming charts of the original mesh; and a subdivision method to subdivide in a regular fashion each face (chart) of the base mesh.

Khodakovsky et al. [KSS00] use this method and apply wavelet compression to the vertex location to achieve extremely high compression results. By regularizing and smoothing regions of the mesh, it is easier to apply and get good results from wavelet analysis; the vertex locations are more correlated and can be better predicted.

Gu et al. [GGH02] propose a similar approach except that they work with fully-regular meshes. They devise an algorithm that can map a mesh of arbitrary genus onto a square. This allows them to use a single parametrization or chart for the whole mesh. They sample the original mesh at regular intervals in the parametrization space to obtain a 2D grid of vertex locations that correspond to the original mesh. This 2D grid of \((x, y, z)\) positions forms a “Geometry Image” of the input mesh, where \((x, y, z)\) are analogous to the (red, green, blue) channels of an image. Standard image wavelet compression is then applied to this “image” to compress the vertex locations. This method needs no explicit connectivity
information, since the mesh is fully regular and has no parameter information since it is sampled at uniform grid points.

This approach is very recent and there are still some remaining open questions. For example, although great care is taken to minimize distortion or stretching during the parameterization, the mapping onto a fixed square (as opposed to free boundary) creates distortion at the boundaries. However, for the models presented the results are impressive.

2.2 Animation Representation

In this section we delve deeper into prior work more directly related to this dissertation. We start with a survey of animation primitives and then go into the related work in the area of representing time-varying geometries. We then look at some related work in the areas of network transmission and streaming of time-varying geometries. Finally, we compare and explore the adaptation of static mesh compression techniques to the dynamic setting.

2.2.1 Survey of Animation Primitives

Lengyel [Len99] provides an excellent overview of the major techniques used in computer animation. To build a representation and to encode time-varying geometries, it is useful to know the range of methods used to generate them. He describes five techniques or primitives of animations: Affine Transforms, Free Form Deformations, Key-Shapes, Weighted Trajectories, and Skinning (See Figure 2-3).

**Affine Transforms** This is probably the most common primitive. Each logical part of a body or animation is transformed by a different affine transform which captures translation, rotation, scaling, and skewing. This transform changes through time.

**Free-Form Deformations** This technique resembles a scaffolding built around an object. By deforming the scaffolding the object is deformed accordingly by interpolation. The joints or lattice points of the scaffolding form the parameters that change with time.

**Key-Shapes** This decomposition technique represents the object as a weighted sum of basis functions. The weights are the parameters that vary with time. This approach is similar to Principal Component Analysis or Karhunen-Loeve transform, applied to more general data. Since the resulting animation is a weighted sum of basis functions, it can only capture a limited class of interpolations (effectively linear combinations of the data).
Weighted Trajectories In this scenario a small subset of moving points or dot trajectories are used to deform a larger base mesh.

Skinning This is widely used technique where each vertex is “attached” to a bone of a skeleton. The motion is determined by moving the “bone” coordinates.

The goal of our research is to represent an arbitrary animation without knowledge of where it came from or how it was made. There are many techniques for generating animations. Although it is possible to determine the technique used for an arbitrary sequence of time-varying geometry, the problem is very difficult. Therefore, we want a representation that is flexible and does not depend on the animation method.

2.2.2 Compression of Time-Dependent Geometry

The idea of the compression scheme proposed by Lengyel [Len99] is to capture these animation techniques and primitives from the source data. It encodes each frame by applying some transformations to a base mesh and encoding the difference between this transformed base mesh and the target mesh. When there is a perfect match, the residual or difference would be zero. This is a very hard problem if we want to consider all possible kinds of transformation and the source of the animation is unknown.

Lengyel proposes a solution to a smaller problem. For simplicity, he only considers affine
transforms and divides the vertices into clusters. Vertices may belong to multiple clusters, each with an associated blending weight. For each cluster, an affine transform is computed to minimize the root mean square error between the cluster of vertices in the target shape and the transformed (affine) cluster from the base shape (or reference shape). A residual between the target shape and the transformed base shape is computed and encoded. The final representation consists of a base shape, cluster allocation (including weights for vertices belonging to multiple clusters), a set of affine transforms for each frame, and residual (error) vertices for each frame. His prototype uses a greedy clustering algorithm.

Lengyel proposes several additional optimizations. For example, it is possible to use the previous frame as a reference frames in the compression process. When using the first frame or a single frame as reference to all other frames, there is a large correlation between the residuals of different frames; in this case, the residual of the previous frame is used to predict the residual of the next frame. Finally, the affine transformation coefficients and residual for each frame are quantized. It is important to note that even after all these transformations, there is still much information in the residuals, and this forms the bulk of the compressed information.

This approach achieves a wide range of compression rates depending on the quantization of affine coefficients and vertex residuals, ranging from 2 to 33 times that of the raw data.

Lengyel's work shares many of the same goals as our approach. We want a representation for time-varying geometry that is generic, that can be applied to any input, and that can be streamed, just like voice or video.

2.2.3 VRML/MPEG4

Existing video compression and multimedia standards such as MPEG4 [Koe02] and VRML [BCM96] include support for animations. They support animations in two ways: with interpolating functions and with specific support for face and body animations.

These standards and other research like “Morph-Nodes,” [ABM00] provide sophisticated interpolating functions. MPEG4 supports special face animation descriptions. Base animations can be loaded with motion described through few parameters.

Unfortunately, they assume that most animations are based on key-frames and rigid body animations. For the special cases of face and body animations, it is restricted to what can be represented with the infrastructure. This restricts the kind of deformations
that they can handle and severely limits the amount of compression they can achieve. It
does not provide support to compress non-transformation (e.g. residual) changes that occur
throughout time.

2.2.4 Representing Animations by Principal Components

An alternative way to represent computer graphics animations was presented by Alexa et al.
[AM00]. They use principal component analysis to compute a basis from which the vertex
locations at any frame can be computed. First they align all the frames to a reference frame
(usually the first frame); then, for each frame they compute the affine transformation that
matches (in the least squares sense) the current frame to the base frame. Then singular
value decomposition is used on these transformed frames to form a basis. Each frame is
encoded as basis coefficients and an inverse transform. They found that using every other
or every fifth frame is sufficient to build a good basis.

There are a couple of drawbacks to this approach. As suggested by Lengyel [Len99],
there are non-linear deformations that cannot be expressed exactly by basis functions. More
importantly, there are no methods to correct for errors – no residual to encode. This limits
the quality of animations it can encode. Also, because it relies on having all the information
at once to build the basis, it is hard to use in an on-line environment.

2.2.5 Temporal and Spatial Level of Details for Dynamic Meshes

The work by Shamir et al. [SP01, SBP00] for representing dynamic meshes is very flexible.
It has support for level-of-detail in space - the algorithm can approximate the geometry of
a mesh within some error tolerance with a simpler mesh. More importantly, they extend
this support in the time domain, including handling changes in connectivity and changes in
topology.

In general, level of detail is implemented by a decimation and reconstruction process. A
coarse mesh can be thought of a fine meshes whose vertices have been decimated. Conversely
a fine mesh can be thought of a coarse mesh that has been refined by adding new vertices.
The decimation of a vertex will depend on the local geometry around that vertex. In a
flat area, a vertex can be removed without changing the reconstruction error or the final
geometry of the object. The main problem with time-varying geometry is that given a fixed
reconstruction error, the reconstruction mesh will differ through time due to changes in the
geometry of the object.

The algorithm proposed by Shamir et al. works by building what they call a T-DAG (Time Directed Acyclic Graph). The nodes in the graph encode vertex decimation in the mesh, the can also represent vertex position updates. Nodes have additional information that determines at what times they are active or enabled. The graph can be traversed in many ways, resulting in different reconstructions of the mesh. The traversal is dictated by the reconstruction error or tolerance desired by the user.

The method is also incremental, meaning that the construction of the previous time step does not depend on a future time step, so it could be used in an on-line environment.

Their main focus is supporting level-of-detail throughout time and handling changes in connectivity and topology. These are difficult problems. They do not focus on compression. There is no mechanism to compress the vertex changes between frames. Their work is complementary to Lengyel’s [Len99], so it is possible to combine their work to produce a more compact encoding.

2.2.6 Streaming 3D Animations over Lossy Channels

Al-Regib et al. [ARARM02] propose a method based on Compressed Progressive Meshes [PR00] for streaming 3D animations over the network. Initially they send a coarse version of the mesh, a coarse texture, and its immediate changes. Later, they send commands on how to update the mesh and texture. These commands are enhanced with error-correcting codes to protect against packet loss and corruption. The amount of protection is dependent on the importance of the data (or amount of improvement it provides).

We believe that many of the ideas in Al-Regib’s work can be applicable to ours. More importantly, we can also borrow many other techniques directly from mature video streaming research since our “Geometry Image” representation are much closer to video data than previous animation representations. Therefore, it is highly likely that our proposed representation would stream well over networks.

2.3 Summary

The straightforward adaptation of static compression methods to the dynamic setting is limited. In practice, all these methods can be adapted by using transformation techniques
as presented in [Len99] to reduce the amount of temporal information. The transformation
techniques presented by Lengyel encode changes between frames by predicting the current
frame from some transformation of a reference frame and encoding the difference or resid-
ual. Static compression methods that maintain the connectivity of the mesh typically have
less compression than methods that resample the mesh; the compressed residuals will still
be large with these methods. Encoding each frame statically is possible, but would be pro-
hibitive in some cases, due to the large amounts of information, and inappropriate since we
know there is much temporal coherence in the computer graphics animations.

Two methods stand out: Progressive Geometry Compression and Geometry Images.
Both of these methods have very high compression rates because they reparametrize and
resample the input mesh, and apply wavelet analysis to the vertex coordinates. The
reparametrization is performed in a way that allows the re-sorting of the geometric in-
formation so that it can be processed and compressed effectively with wavelets.

Our work builds largely on the work of Geometry Images and Lengyel to create a
new representation for 3D animation that compresses very well. We use Geometry Images
because it can map a sequence of time-varying geometries to a sequence of images and
hence it allows the applications of many of the well-know video compression and processing
techniques. Furthermore, decoding a Geometry Image can be done very efficiently. That
said, I admit that the application of these techniques to Progressive Geometry Compression
[KSS00] should be explored in the future as it is also very promising.

We believe that for most problems and animations, re-sampling the original mesh is ac-
ceptable. We note that there are cases where this might be unacceptable, and our approach
will not be applicable.
Chapter 3

Background

This chapter presents concepts and terminology helpful for understanding of our approach.

We propose to use Geometry Images [GGH02] as the representation for each mesh in a 3D animation sequence. This representation has important advantages, including ease of implementation and support for multi-resolution representations of the input mesh.

A Geometry Image represents any manifold surface in 3D space as unit-square parameter space. A Geometry Image for a mesh is obtained by computing a global parametrization of the input mesh, which can then be resampled to obtain a fully regular mesh representation.

This chapter is organized as follows. In section 3.1, we explain the concept of global parametrization, and describe how it can be used to represent arbitrary objects. We describe two methods for parametrizing and remeshing geometries. We also explain in detail the geometric-stretch metric [SSGH01], the parametrization metric used by the Geometry Images algorithm. In section 3.2, we explain in more detail, the other aspects of the Geometry Images relevant to our representation.

3.1 Parametrization

A parametrization specifies a curve or surface, etc., by means of one or more independent variables which are allowed to take values in a given specified range.

We are interested in an alternative representation for animated and deforming meshes. These meshes will form the frames of the 3D animations we want to represent. We can think of meshes as a piecewise linear approximation to a surface. Alternatively, we can represent these surfaces as a function of two parameters. This is a parametrization.
We are concerned with parametrizing 2D surfaces embedded in 3D space. To do this, we use a bijective mapping of points \((x, y, z) \in \mathbb{R}^3\) to \((u, v) \in \mathbb{R}^2\). This type of parametrization has many applications. It can be used for texture mapping where we map a 2D image across a surface to enhance its appearance. It can also be used for remeshing or resampling a model. Given a mesh, we compute a parametrization of its surface. We can then evaluate this parametrization at regular intervals to obtain a regular connectivity representation of the input mesh. Regularity is important because it “orders” the information for ease of processing and compression. Furthermore, if we can resample a mesh into a regular rectangular grid, then we can use standard video processing techniques to represent 3D animations compactly. A single regular sampling of the mesh is preferred because it simplifies processing and does away with many boundary conditions and special cases needed by other algorithms. We are interested in the best way to resample the models corresponding to each of the frames of an animation.

We can broadly classify the parametrization of meshes into two kinds: Local and Global. In a local parametrization, the input mesh is partitioned into patches, and each patch is parametrized individually. In a global case, a single parametrization is used for the whole mesh.

### 3.1.1 Local parameterizations

Models can be represented as a collection of patches, each of which can be parametrized locally without effect from other patches. With local parametrizations detailed version of the patch can be generated. The advantage of this representation is that it is easy to use for modelling and editing of objects. The user only edits a small set of control points to modify a section of the surface. The patch is usually evaluated via subdivision. The patch is divided into smaller patches, and the control points for these patches are some polynomial function of the control points in the larger patch. Surfaces defined in this manner are called subdivision surfaces. Figure 3-1 shows an example of a phone modelled using subdivision surfaces.

This representation allow us to resample each patch easily. Unfortunately, the layout of the patches is not regular and hence it is harder to achieve a regular sampling over the whole mesh. The other drawback is that not all meshes are specified in this manner. We want our representation to work with arbitrary meshes of unknown origin. In this situation, a
pre-processing step is required to create and fit a subdivision surface for an arbitrary mesh. This problem has been studied in the past [EDD+95, LLS01]. The disadvantage of such approaches is that special care must be taken at the boundaries of the patches. It also makes the processing more difficult as each patch must be processed individually.

3.1.2 Global parametrizations

It is easier to obtain a regular remeshing of an object if we are provided with a global parametrization. There are no patches, or connectivity constraints involved with a global parametrization. One single function is defined over the whole surface of the mesh. There is one major obstacle to this goal: the topology of the input mesh. The domain of the parametrization \((u, v)\) is most conveniently defined within a unit square. If the mesh is not topologically equivalent to a disk, it is not possible to map the surface of the mesh to a unit square.

There are two approaches to overcome this obstacle. The first approach is to map the object to a parametrization defined over a topologically equivalent object. For example, we could parametrize all genus-1 objects over a toroid. We do not follow this approach since we are interested in parametrizing meshes into a regular grid for the purposes of applying image and video processing techniques.

The second approach to parametrize objects, which are not topologically equivalent to a disk, is to make cuts on the object and then unwrap the object onto a plane. The process of opening a mesh of arbitrary topology has been well-studied [DS95, VY90, LPVV01, EHP02]. For example, to unwrap a torus, we would make two cuts: cutting it transversally unwraps the torus into a cylinder and an additional cut unwraps the cylinder onto a sheet.
description of the algorithms used to open a mesh is provided in section 3.2.1.

Given that we have a mesh that is topologically equivalent to a disk. The next step in the process is to parametrize this mesh onto a square. Since the parametrization mapping is bijective, we can then sample the parametrization in a grid fashion to obtain a representation of the input mesh with regular grid connectivity. There are many ways to obtain a parametrization of a surface. A early method was suggested by Tutte [Tut63], who proposes a barycentric mapping. Once the boundaries of the mesh are fixed onto a convex boundary (in our case a square), the internal vertices are placed such that they are located in the average of their neighbors – this guarantees that the function is bijective.

Floater [Flo97] proposed an alternative parametrization; he uses Tutte’s approach as a basis of his parametrization. The drawback of the barycentric mapping is that is does not locally preserve the shape of the triangles. This means that equilateral triangles in the input mesh could get mapped to a long skinny triangle in the parametrization. Floater suggests a angle-preserving conformal parametrization that locally approximates the shape of the triangles of the original mesh. Unfortunately, this parametrization does not conserve the area of the triangles. Large triangles in the input mesh could be mapped to small triangles in the parametrization in order to preserve the shape. An alternative approach is to construct an area-preserving parametrization.

Ideally, a parametrization that is both area-preserving and angle-preserving is desired. The resampled mesh should closely approximate the input mesh. A sampling that is uniform over the input mesh surface will best approximate the geometry. Preserving the area of the triangles in the parametrization would result in equal sampling density over the input surface. Preserving the angles of the triangles in the parametrization result in locally equal sampling in both orthogonal directions over the surface. Achieving a parametrization that is both angle-preserving and area-preserving is impossible (e.g. a gaussian surface); the best we can do is to approximate it. The geometric-stretch metric of Sander et al. [SSGH01] strives for exactly this goal. Geometry Images use an iterative algorithm that minimizes the average geometric-stretch of the parametrization.

Figure 3-2 shows three different parametrizations of a cut turtle mesh. The first parametrization is an approximation of an area-preserving mapping. Notice that the shape of the shell is distorted on the right side of the parametrization. Sampling this area regularly results in different sampling rates in one direction than on the orthogonal direction. The
middle parametrization is a conformal Floater parametrization. The shape of the shell is preserved. There are two clusters on the bottom left corner that correspond to the rear flippers; locally, the shape of the triangles is preserved, but in order to achieve this, the size of the triangles is smaller. Under this parametrization these flippers would be undersampled. The last parametrization shown is one that minimizes the geometric-stretch. Notice that the shape of the shell is mostly well-preserved and the flippers on the lower left are not undersampled.

**Figure 3-2:** Different kinds of Parametrizations: (a) shows the turtle being parametrized. The turtle is an open mesh with a boundary (shown in red). The boundary of the mesh forms the boundary of the parametrizations. (b) shows an area-preserving (approximation) parametrization. (c) shows its Floater angle-preserving parametrization. (d) shows the parametrization minimizing worst geometric stretch.

**Geometric-Stretch**

We now describe in more detail the geometric stretch metric of Sander et al. Recall that a parametrization is a bijective mapping of points \((x, y, z) \in \mathbb{R}^3\), call this surface \(S\), to \((u, v) \in \mathbb{R}^2\), call this domain \(D\). Let \(f\) be a function that maps points \(D\) to \(S\): \((u, v) \rightarrow [f^x(u, v), f^y(u, v), f^z(u, v)]\).

The Jacobian of the function \(f\) is:

\[
J_f(u, v) = \begin{bmatrix}
\frac{\partial f^x}{\partial u} & \frac{\partial f^x}{\partial v} \\
\frac{\partial f^y}{\partial u} & \frac{\partial f^y}{\partial v} \\
\frac{\partial f^z}{\partial u} & \frac{\partial f^z}{\partial v}
\end{bmatrix} = [f_u(u, v)f_v(u, v)]
\]

The "stretch" or distortion of the mapping from \(D\) to \(S\) is determined by examining the singular values \(\Gamma\) and \(\gamma\) of this Jacobian matrix. The minimum and maximum length of a
unit vector in $D$ mapped onto $S$ correspond to the length of $\gamma$ and $\Gamma$ respectively. They correspond to:

\[
\Gamma(u, v) = \sqrt{\frac{1}{2} \left( (a_f + c_f) + \sqrt{(a_f - c_f)^2 + 4b_f^2} \right)} \text{ maximum singular value} \tag{3.2}
\]

\[
\gamma(u, v) = \sqrt{\frac{1}{2} \left( (a_f + c_f) - \sqrt{(a_f - c_f)^2 + 4b_f^2} \right)} \text{ minimum singular value} \tag{3.3}
\]

where

\[
\begin{pmatrix}
  a_f(u, v) & b_f(u, v) \\
  b_f(u, v) & c_f(u, v)
\end{pmatrix} =
\begin{pmatrix}
  f_u f_u & f_u f_v \\
  f_v f_u & f_v f_v
\end{pmatrix} = J_f^T J_f = M_f(u, v) \tag{3.4}
\]

$M_f(u, v)$ is called the metric tensor of $f$ at $(u, v)$. From the singular values $\Gamma$ and $\gamma$, Sander et al. describe two norms that capture the average and worst case stretch or distortion:

\[
L_2(u, v) = \sqrt{\frac{1}{2} (\Gamma^2 + \gamma^2)} \quad \text{and} \quad L_\infty(u, v) = \Gamma \tag{3.5}
\]

The $L_2$ can also be expressed using the metric tensor as $\sqrt{\frac{1}{2} tr(M_f)}$. The geometric stretch for the whole surface is equal to $L_2$ integrated over the surface:

\[
E_f(S) = \int \int_{(u,v) \in D} (L_2(u, v))^2 dA_S(u, v) \tag{3.6}
\]

Since we are working with triangular meshes which are piecewise linear, the Jacobian $J_f$ is constant over each triangle. In this case, the integral can be written as a sum:

\[
E_f(S) = \sum_{\Delta_i \in D} \left( \frac{1}{2} tr(M_f(u_i, v_i)) A_s(\Delta_i) \right) \tag{3.7}
\]

In the next section we describe the Geometry Image algorithm which is based on a global parametrization that minimizes the geometric-stretch.
3.2 Geometry Images

In order to construct a Geometry Image, the original manifold mesh must be cut in order to form a mesh that is topologically equivalent to a disk. The process works in three stages. First, a sufficient number of cuts are made to unwrap the mesh. Second, the mesh is parametrized onto a square domain. Finally, the parametrization is sampled to obtain a 2D grid of surface points on the original mesh. I now describe these steps in more detail.

3.2.1 Cutting

The mesh must be cut before it can be parametrized onto the unit square. The cuts, besides unwrapping the mesh into a surface topologically equivalent to a disk, also help to improve the parametrization by traversing various extrema, or high-stretch triangles, thus improving the $L_2$ (average) geometric-stretch of the parametrization. The cuts proceed in two stages. The first stage finds an initial set of cuts necessary to unwrap the mesh and the second stage augments the initial cut to improve the parametrization.

The algorithm used to cut a mesh of arbitrary topology in order to map it onto a disk is similar to Erickson’s [EHP02]. It operates as follows: The first stage is divided in two phases, the first phase removes triangles and some edges in order to identify loops, which must be cut in order to make the mesh topologically equivalent to a disk. The second phase removes extraneous edges.

If the initial mesh is manifold, the first phase starts by removing a seed triangle. While
there are boundary edges (i.e. edges belonging to only one triangle) not in $B$, the algorithm removes a boundary edge and its corresponding triangle. Its two remaining edges remain even if there are no triangles associated with them. The order of triangle and edge removals are based on the geodesic distance to the seed triangle. Once all triangles have been removed, a network of edges connecting all vertices remains.

The second phase removes the dangling edges. The algorithm iteratively removes edges connected to a single vertex. This means that any subtrees of the network of remaining edges will be removed, leaving just the cycles in the network. There will be $2g$ cycles in this network for meshes of genus $g$. These loops identify a cut that will unwrap a mesh of arbitrary genus to one topologically equivalent to a disk. For the special case of genus-0 meshes, the remaining network of edges will be the empty set. For this case, the initial cut can be refined by an arbitrary set of connected edges.

The second stage of the cutting algorithm augments the initial cut to improve the parametrization. Gu et al. [GGH02] found through experimentation that the parametrization quality (in terms of the geometric-stretch $L_2$ metric) can be improved if cuts traverse various extrema or regions of high geometric-stretch in the mesh. Starting from the initial cut, the second stage works as follows: The current cut mesh is parametrized onto a circle using Floater's parametrization [Flo97] (It is mapped onto a circle to avoid boundaries degeneracies). Floater's conformal parametrization is used because it "identifies" these extrema well. If a geometric stretch parametrization were used, it would smooth out and redistribute the stretch hiding the extrema. The triangle with the worst geometric stretch ($\Gamma$) becomes the extremum. The initial cut is augmented with a branch (set of edges) from a vertex of this extremum triangle to the boundary via the shortest path. Cuts are repeatedly added to the initial cut until the average geometric-stretch ($L_2$) of the parametrization can no longer be decreased with additional cuts.

For genus-0 meshes an additional optimization is added. The initial cut was based on an arbitrary triangle In order to start with a real extremum, a cut is made from the original initial cut (two arbitrary edges) to the first extremum triangle. Only the two edges closest to the first extremum triangle are kept and will form the new initial cut.

Figure 3-4 shows iterations of the cutting algorithm on a sample mesh.
Figure 3-4: Iterative Cutting Algorithm: We start a short cut. The resulting parametrization has high distortion at the center; The worst under-sampled area is near the horns. A cut is then added from the tail to the horns and the resulting mesh is parametrized. This new parametrization has high distortion at the hoof of the cow. A cut is added from the hoof, etc. The algorithm stops when adding a new cut would increase the average geometric-stretch.

3.2.2 Boundary Parametrization.

Once the cut is defined, the mesh needs to be parametrized onto the 2D domain. The first step in this process is to map the boundary of the mesh (induced by the cuts for closed objects) to the boundary of the 2D square of the parametrization. A vertex in the original mesh will be duplicated at two points on the boundary if it lies in the middle of a cut or will be replicated $n$ times if it lies at the intersection of $n$ cuts. The latter vertices are called cut-nodes. In order to prevent geometric discontinuities across the cut when the mesh is resampled, samples on both sides of the cut must lie on the same location when reconstructed. For this reason, it is helpful to place cut-node vertices at specific grid locations on the boundary of the parametrization so that they can be reconstructed with greater precision. Furthermore, to guarantee that both boundaries between the cut-nodes are sampled properly we enforce that the equal lengths of perimeter are allocated for segments between cut-nodes in the parametrization.

Lastly, two degenerate cases have to be handled: zero-area triangles and corner edges. If a triangle has all three of its vertices mapped to one side of the parametrization it will have zero area. This occurs when two of its edges are mapped on the boundary. To prevent this degenerate case, the triangle is split by introducing a vertex at the mid-point of the non-boundary edge. To prevent T-junctions, the same process is applied to the triangle on the other side of the edge. A second degenerate case occurs when an edge on the mesh is
mapped to an edge on the boundary of the parametrization that spans a corner. This case is prevented by inserting a vertex at the corner, thereby splitting the triangle on this edge in two. To ensure correspondence, the same procedure is applied on the triangle corresponding to the other side of the edge.

Figure 3-5: Pipeline. (a) start with a cut of the input mesh. (b) the boundaries (edges) of the cut map to the boundary of the parametrization. (c) we raster scan the parametrization and interpolate the points. (d) we compress, decompress and rebuild the points. The cut or seam will open if the decoding is lossy. (e) we use the boundary information to close seams.

3.2.3 Interior Parametrization.

After defining the parametrization of the boundary vertices, the parametrization of the interior is computed so as to minimize the geometric-stretch metric. This metric penalizes undersampling in the parametric domain and yields parametrizations that nearly uniformly samples the mesh surface. The algorithm for minimizing geometric-stretch works in a hierarchical manner. It starts by building a coarse representation of the mesh generated by a progressive mesh simplification. It adjusts the location of the vertices to minimize their geometric stretch. Through a series of vertex split operations, new vertices are added. After each addition, the location of the new vertices and their neighbors are adjusted. This process is repeated until all vertices are added back into the parametrization.
3.2.4 Rasterization

The last step in the generation of Geometry Images is the rasterization step. The parametrization is sampled at grid locations \((x/n, y/n)\) for \((0 \leq x, y \leq n)\) where \(n\) is the height and the width of the geometry image. These locations are trilinearly interpolated from the vertices of the triangle they belong to. This process generates a 2D grid of \((x, y, z)\) points over the original mesh. The name Geometry Image is derived from visualizing each coordinate plane \((x, y, z)\) as colors \((\text{red, green, blue})\) and hence we have an image representing a surface.

Furthermore, a level-of-detail rendering of this geometry image can be simply produced by decimating the original geometry image. By picking every other sample in the geometry image, a coarse version of the original mesh can be obtained. To avoid cracks in the boundaries at lower resolutions, the boundary edges have to be mapped in such a way that sampling of the boundary in the decimated geometry image returns cut-nodes residing at grid locations. This is achieved by generating the boundary mapping at the lowest intended resolution and then super sampling the parametrization at the highest resolution.

3.2.5 Compression and Stitching.

The geometry image representation naturally lends itself to compression. Gu et al. compress the geometry image for a mesh by first quantizing the each vertex coordinate to 12 bits. The \((x, y, z)\) positions are then translated and scaled to fit inside the unit cube. The geometry image is then compressed using 2D wavelets based on Davis' implementation \([\text{Dav}]\). If lossy compression is applied, the corresponding boundary samples will not necessarily match after decoding. This will lead to visible cracks where the cuts were made. This is addressed by storing additional sideband information about the boundary and the cut.

3.2.6 Limitations

There are many advantages of being able to parametrize the surface onto one chart: It arranges the data in a compressible manner; it can be easily manipulated and parallelized; it supports level of detail; it provides an implicit parametrization for other surface properties such as textures and normals. Unfortunately, there are limits on the types of surfaces that can be parametrized globally without having regions of high distortion. This limit is a
function of the topology and curvature of the surface. The cutting algorithm and the parametrization attempt to minimize this surface stretch, but the layout of the regions of high distortion may be such, that the system maybe unable to reduce the high stretch in one region without increasing stretch in other regions. Furthermore, the algorithm requires manifold geometry.

Moreover, mapping the surface to a square introduces some distortion at the corners. Having a fixed aspect ratio also imposes a limitation on the unwrapping of the surface. Figure 3-6 shows the parametrization of a snake. Due to the “aspect-ratio” of the snake body, there is high distortion at the boundaries. This becomes a problem for textures, as these areas might not be adequately sampled. It is possible to alleviate the problem by smoothing or filtering the points near the boundary or by changing the parametrization metric or bias. This is sensible in a static context. In a dynamic context, the answer is not obvious, as we would need to calculate the effect of this smoothing over all frames.

![Image](Figure 3-6: Limitations: one of the drawbacks of using a single parametrization, is that regions of high-distortion might be unavoidable. In this case, the boundary of a very long object has high stretch.

There are potential solutions to this problem, among them chartification (breaking the mesh into smaller local parametrizations) and allowing the boundaries to be free. We use this algorithm due to its advantages in compressibility and ease of processing. For now, we acknowledge that Geometry Videos may not be the optimal representation for all objects.
3.3 Summary

We have presented a review of parametrizations and of the Geometry Image algorithm. We have described in detail how the cutting algorithm works, and defined the geometric stretch metric, and the notions of distortion. The Geometry Images algorithm forms the foundation for our 3D animation representation. In the next chapter, we will look into how to adapt this algorithm to work well with multiple meshes corresponding to frames in an animation.
Chapter 4

Adapting Geometry Images to Animation

In this chapter we discuss the basic issues that arise when applying the Geometry Images approach to animation. It presents modifications that allow the algorithm to work well with time-varying geometries; we call this representation Geometry Videos. We consider reconstruction quality, texture mapping and compressibility of these changes. In the next chapter, we look at how we can compress these Geometry Videos by taking advantage of the spatial and temporal coherence that exists in the animation.

The straightforward way to implement 3D animation encoding with Geometry Images is to apply the algorithm individually to each frame. However, this implies that each frame of the animation has a different cut and parametrization. The alternative approach is to apply a single cut and parametrization to all the frames in the animation. We assume that all frames of the animation have the same connectivity and topology. This chapter is divided into two sections. Section 4.1 motivates our choice of a single cut and parametrization. It also presents our algorithm for choosing a good cut that works well with all frames of the animation. In section 4.2, we discuss the limitations of our approach and present some alternatives.

4.1 Consistency of Cut and Parametrization

Computing and applying a different cut and parametrization to each time-step in a 3D animation has many disadvantages. These disadvantages stem from the fact that the Geometry
Images for each frame lack correspondence. Corresponding coordinates in the Geometry Image of each frame correspond to a different parts the animated mesh. Alternatively, we can compute a single cut and parametrization and apply it to all frames in the animation, assuming that each mesh has the same connectivity. In this section, we first describe the disadvantages of using individual cuts and parametrizations for each frame. Then we discuss the benefits of using a single parametrization. We then present our algorithm for computing a single cut from the animation. We also present how we choose our parametrization. We conclude by mentioning some details from the Geometry Image’s approach that differ when representing an animation instead of a static mesh.

Using different cuts and parametrizations from frame-to-frame in an animation does not allow us to reuse much information. The chosen parametrization defines how each frame is sampled. Once the frames are sampled we can fix a texture map to the sampling. Since the sampling is regular, the texture coordinates of the sampled mesh will be the grid coordinates of the unit square. Different parametrizations for each frame, requires separate texture maps for each frame. This can also lead to visible artifacts such as the apparent “slippage” of the texture across the surface and variations in resolution.

Besides texture map information, there are other attributes that can remain constant on the mesh throughout the animation, i.e. material id and lighting coefficients. These would also have to be encoded for each frame if different parametrizations were used for each frame.

Moreover, there is no correspondence between Geometry Image samples from different frames if we use different parametrizations. This lack of correspondence also makes it difficult to extract any temporal coherence from the sequence of Geometry Images. It also makes it difficult to compare the same sampled vertex between different frames, making it hard to predict the location of future samples.

Finally, separate parametrization for each frame of the animation complicate the maintenance of stitching sideband information associated with every cut. The Geometry Images for closed objects are formed by cutting them along some path, which is mapped to the boundary of the unit square. Lossy compression can widen these cuts, making them more visible. We therefore use the sideband stitching information to force the samples on opposite sides of the cuts to match up. This removes any visual gaps. However with a different cut on each frame, we are forced encode and process this sideband information for every
We can overcome this last limitation by enforcing a common cut among all frames of the animation but computing a new parametrization for each frame. Unfortunately, this does not solve the other associated problems. It would still need a different texture map for each frame and it would still be difficult to find correspondence between samples in sequential Geometry Images. Furthermore, we find that the cut selection has a strong effect on the final parametrization. Since the cut defines most of the boundary of the parametrization, the resulting mapping often exhibits very little difference between the parametrizations of different frames using a common cut. Figure 4-1 shows an example of different parametrizations using a single cut and using a cut obtained from each frame.

Figure 4-1: Three different frames from Cow sequence. Cut and Parametrized in two ways: using a local cut and parametrization; and using a global cut and the first frame for parametrization.
We have found that using a fixed cut and parametrization for all frames works well and has very few drawbacks. It allows the use of a single texture map for all frames in the animation since all vertices have the same texture coordinates. The same is true for other attributes like lighting coefficients.

Samples in the geometry images for each frame will correspond to the same location in the mesh. This allows us to predict the location of samples in future frames based on their location in previous frames just by looking at the samples in the Geometry Images, which have greater coherence.

The application of a single cut implies we need sideband information for only one cut. The stitching operation applied to each frame is the consistent. Furthermore, it is possible to optimize this operation as a preprocess. This is important if we are implementing or optimizing this system in hardware, because we would only need to compute and transmit this information once.

Choosing a single cut and parametrization for all frames in the sequence does not work well if the features of the model change drastically from frame-to-frame (i.e. a six-legged spider morphing into four-legged horse). It is possible that there does not exist a single parametrization that works well on all frames. For example, if major deformations occur at different parts of the object in different frames. In our sample data sets we have not found such scenarios. We propose to handle these infrequent situations the same way that we handle changes of topology as discussed in Section 4.2.

Next, we describe how we choose a parametrization given a single cut consistent for the entire animation. Then we describe how we optimize the choice of the cut.

4.1.1 Choice of Parametrization

Given a single cut, we want a parametrization that minimizes the errors across all frames. The error metric we consider is the surface approximation error of the reconstructed mesh from the Geometry Image and the original mesh.

We also want to support level of detail representation of mesh models. We want to be able to create an approximate (lower resolution) mesh of from a given Geometry Image. This is important when dealing with large meshes because it allows us to use a simplified model when the mesh is far away or when detail is not necessary. The way we generate simplified models is by sub-sampling, or more specifically decimating, the Geometry Image. For this,
we want a parametrization which uniformly samples the input surface. We need uniform sampling so that we don’t have unpredictable effects when we decimate the Geometry Image.

The Geometry Image’s algorithm has these properties for a single frame. Thus, we need to modify it to take all frames into account or to determine a single reference frame in our animation whose parametrization works well for all frames.

The choice of a cut has a major effect on the final parametrization. Recall that the cut selection process works iteratively (see Section 3.2.1), where at each stage it makes a cut from the boundary to the face with the largest geometric stretch. In animations without large deformations, these regions of large stretch tend to occur around the same locations of the mesh in every frame. The boundary of the cut mesh is then mapped to the boundary of the unit square. The parametrization then tweaks the internal location of the vertices to minimize the average geometric stretch of the faces. Notice that the boundary is primarily determined by the cut. It still varies depending on the mesh because the lengths along the border will be proportional to the length of the edges in the mesh, but the vertices that belong to the border will be fixed by the cut.

Therefore, once the cut is fixed, only the internal vertices need to be parametrized. We find that for animations without large deformations (deformations that occur on a large percentage of the surface area), that using an arbitrary parametrization still yields good results. The intuition is that most of the regions of high geometric stretch have been traversed by the cut; hence the deformations do not substantially alter the average stretch. We currently use the first frame of our animation as the reference frame for the parametrization. Our experiments validate the effect of the cut on the parametrizations. The reconstruction error using the same cut but different reference frames for the parametrization does not vary much.

We now need a cut that traverses the regions that, when parametrized, have high geometric stretch. This is the topic of the next section.

4.1.2 Global Cut

We would like to find a cut that works well across all frames. This means that when we apply the same parametrization to each cut mesh, the parametrized mesh does not have much geometric stretch and will be sampled approximately uniformly.

One possible approach to find such a cut is to use brute force. That is, we look at
the cuts generated by using the algorithm in Geometry Images [GGH02] for each frame. We parametrize all frames using each cut, and select the cut that has the least average reconstruction error. This was the initial approach we evaluated. We found that frames that showed all the important features of the object (i.e. a human in anatomical position, or a standing cow) worked best, but only slightly better than less optimal frame. This is because the cuts in each frame usually traversed the same important features (although sometimes in a different order). Moreover, cuts tend to pass through all regions of high-distortion. All these areas were usually present and reached when the cutting algorithm was applied to any frame. The drawback of this approach, besides being computationally expensive, is that it will miss areas of high-distortion that vary from frame-to-frame. Figure 4-2 shows an example where this can happen, where the eyes and the tail are regions of high-distortion, but they occur in different frames.

![Figure 4-2: Global Cut. (a) shows the cuts of turtle with its tail sticking out. (b) shows the cuts of turtle now with its eyes sticking out but tail retracted; tail now does not have much distortion. (c) shows a reference turtle with the global cuts built from (a) and (b). Notice that it traverses both the base of the eyes and the tail. (d) shows the reconstruction of the tail using the cut from the (b) (the red mesh is the original frame). (e) shows the reconstruction of the tail using the global cut (c).](image)

In order to identify these potentially difficult areas, we propose a global cutting algorithm that considers all frames of an animation. This cutting algorithm works by applying the single-frame cutting algorithm simultaneously to all frames. Recall that the single-frame cutting algorithm works iteratively by cutting, computing the Floater parametrization on the resulting mesh, finding the vertex next to the face of highest distortion, and then making the shortest cut from that vertex to the current boundary. The algorithm stops when additional cuts do not improve the average geometric stretch. The problem then becomes how to find the face of highest distortion and how to compute the "shortest-path" to the boundary when all the frames are being considered.

In the single frame case, the vertex of highest distortion is located by computing the
geometric-stretch [SSGH01] for each face on the Floater parametrization. A vertex of the face with the highest stretch is then selected. The geometric-stretch metric implicitly measures the distortion introduced between parameter space and object space (e.g. how circles in the parameter space would map into the object space; the distortion is measured by the magnitude of the major-axis of the resulting ellipsis).

For the global case, we have to compare or average the maximum stretch on many frames. We also do not want a single (very distorted) to dominate algorithm. Therefore, for each frame, we normalize the stretch metric by the maximum stretch seen in that frame. The stretch of each face is then the average of the normalized stretch of that face in all frames.

We compute the shortest distance from the vertex next to the face with largest stretch to the boundary by using Dijkstra's algorithm on a proxy mesh. This non-realizable proxy mesh has the same connectivity of our animation, but the length of its edges is the average length of that same edge for all frames of the animation. This method typically yields good results.

The single-frame algorithm stops when the addition of a cut would increase the average distortion. This is the average of the average distortion for each face on each frame. The average distortion of a face considers a circle in the parameter space and the ellipses in object space, and integrates over all $\Theta$ the distance between the center of the ellipsis and the point on the ellipsis in the $\Theta$ direction. This measurement is already normalized by the surface area in object space. It is comparable between frames. For our global cutting algorithm, we use the average of the average distortion between all frames as our stopping criteria. The algorithm stops when an additional cut increases this metric.

Currently for higher-order genus objects (genus greater than 1), we apply the algorithm from [GGH02] to the first frame to produce the minimal cuts necessary to unwrap the surface into a 2-manifold disk. At this point, we apply our global cutting algorithm to improve upon this cut.

The global cut does not always do better than picking a cut and parametrization from a single frame either by hand or by exhaustive search. However, the difference is small. The advantage is that it does consistently well; this reduces human-intervention and computation time. Additionally, it has a unique advantages over using a cut from just one of the frames – namely that it can identify and traverse regions of high distortion that occur at different
4.1.3 Sampling and Rendering

Once we have determined a cut and parametrization, we can transform each animated mesh of the animation sequence into a Geometry Image. The collection of the Geometry Images is called a Geometry Video. The conversion works by sampling over a grid the original vertex values mapped onto the parametrization. By using the same parametrization, the samples in the same coordinates in different frames will correspond to the same region in the original surface.

The mesh resulting from the geometry image will have regular connectivity. It will look like a grid. In [GGH02], they diagonalize each quad in the grid along the shortest diagonal. This increases the reconstruction quality slightly over triangulating all quads with the same diagonal. For animations, we use a fixed arbitrary regular triangulation for all the frames. Changing the triangulation of the grid between frames would yield popping or jittering artifacts between frames where the diagonalization changes.

4.2 On Topology and Connectivity Changes

Our preferred parametrization is fixed and computed from a single arbitrary frame. Furthermore, our algorithm assumes all meshes in the animation have the same connectivity. This presents a problem if the input mesh changes connectivity or topology or where a single parametrization cannot produce a good sampling on all the frames.

Handling connectivity changes is not an inherent problem of Geometry Images. It is possible to compute a parametrization that takes into account different frames of different connectivity and build a consistent parametrization that encompasses the different configurations. This would be easy for small or simple connectivity changes. For example, the subdivision of a triangle on the mesh surface could correspond to the subdivision of a triangle in the parametrization. For more drastic connectivity changes, where computing this parametrization would be difficult or impossible, we could apply a re-meshing algorithm (like [PSS01]) to all the frames to generate frames of equal connectivity. Then we could apply our algorithm on an animation with consistent connectivity.

On the other hand, changes in topology generally also force changes in the cuts which
make re-using or adapting a parametrization next to impossible. In this case, we introduce new cuts and parametrizations for the animation. Similarly, if a single parametrization does not sample all frames well, then we might also be forced to use more than one parametrization. Approximate thresholds can be introduced within the encoder to handle these special cases (This is a current limitation of the system). Our proposed solution is similar to the approach used at scene changes in standard video compression. We would need to ensure that the transition between the two frames with different parametrizations is smooth. 3D morphing techniques could be used for this purpose. The problem of transitioning between parametrizations is not addressed in this thesis.

4.3 Summary

In this chapter we have presented modifications to the Geometry Images approach that adapt them to time-varying geometries. We have described the problems that occur when using different cuts and parametrizations throughout an animation. We propose to use a single cut and parametrization for all frames. We have presented a global cutting algorithm that considers all frames in the animation. However, currently, we use an arbitrary reference frame for our parametrization, this works in practice because the choice of a cut has the greatest impact on the parametrization. We concluded by pointing out the limitations of our approach.

In the next chapter, we explore how to best compress the sequence of Geometry Images produced by using the algorithm proposed in this chapter.
Chapter 5

Adapting Geometry Images to Compression

As in video, there is a large amount of temporal coherence in 3D animations. Animations are often described as smoothly varying mathematical functions (see section 2.2.1) of the vertices. In cases where they are not described by mathematical formulae points tend to move together because of surface continuity. This implies that it should be possible to represent or encode the motion and deformation with few parameters; the remaining change (i.e. the small error that is not encoded by the initial parameters) can be encoded as a residual.

This approach is similar to the approach used in MPEG [Gal91] for video encoding. Where reference frames are encoded independently of others, and remaining frames are encoded by predicting the current frame as a function of the previous (and/or future) frames and then encoding the difference. The key aspect of this compression method is the prediction function. For MPEG, the image is divided into blocks and each block is predicted by finding the best matching block in the reference frame. For 3D animations, this prediction function can be based on a more accurate function of the change between frames.

In the previous chapter, we presented some of the issues and techniques needed to best utilize the Geometry Image's approach for animation. We proposed to utilize the same cut and parametrization for all the frames in the animation. This yields correspondence of vertices across difference frames. For instance, the vertex in position (2,3) on the Geometry
Image for frame one corresponds to the vertex in position (2,3) on the Geometry Image for frame two. We now have a sequence of Geometry Images corresponding to the sequence of time steps in the animation. This sequence of Geometry Images could be encoded using standard video encoders (with minor changes to support more than 256 colors or 8-bit values), but greater compression can be achieved if we take into account that Geometry Images correspond to manifold surfaces in 3D space.

In this chapter, we present how to compress a sequence of Geometry Images by taking advantage of the knowledge that they correspond to 3D geometry. In section 5.1, we revisit the spatial compression of Geometry Images. Gu et al. [GGH02], used wavelets for compression; we compare wavelets with Discrete Cosine Transforms (DCT) used by traditional image and video encoders. In section 5.2, we explore the temporal compression issues and introduce predictor functions.

In section 5.3, we continue the exposition of predictor functions and propose to use affine transformations for 3D animations. Sections 5.3.1 and 5.3.2 describe in detail two ways to apply the predictor function: globally and locally. Finally, section 5.4, describes how we encode frames – similar to MPEG – with a few specific differences.

5.1 Spatial Coherence

Most standard video compressors (i.e. MPEG) use Discrete Cosine Transforms (DCT) as the basis of their encoders. Gu et al. [GGH02] used wavelets for compressing Geometry Images. In this section, we re-evaluate the choice of transforms for encoding single Geometry Images, as well as sequences of Geometry Images.

The DCT transform can be performed over the whole Geometry Image, or in smaller blocks like in MPEG. First we note that the Spectral Compression approach of Karni and Gotsman [KG00] uses spectral analysis similar to DCT for encoding the vertex positions of meshes. Although Spectral Compression is not the same as encoding Geometry Images using DCT, it uses the same frequency domain analysis as DCT; the fact that Geometry Images using wavelets outperform this method in our rate-distortion experiments, indicating that wavelets are well suited for Geometry Images.

An alternative comparison involves encoding the Geometry Images with both DCT and wavelets. The advantage of wavelets stem from their multi-resolution and localized
Multi-resolution algorithms work at different scales of the signal; small details will be handled at a fine scale and will not affect the encoding of other regions. In contrast, DCT has global support, and all coefficients are likely to be affected by localized details. Partially for this reason and implementation efficiency, most video compression applications use Blocked-coded DCT. In this case, the image is split into fixed $N \times N$ regions and each region is encoded individually. Unfortunately, this approach can lead to blocky artifacts near the boundary of the regions. These artifacts are usually addresses with pre-filtering and/or post-processing techniques.

Figure 5-1 shows the comparison of the Geometry Image of a cow being compressed with wavelets and DCT. We can see some “block” artifacts of DCT from the lighting on the bottom of the cow’s eye. In all of our tests wavelets encoded geometry images better than DCT.

There are many choices of wavelet basis functions that can be used to encode Geometry Images. We tried 16 different families of wavelets and we found that wavelets with large support did better than wavelets with small support. Wavelets with large support are able to use more information at each scale and, for smooth images, they will work better.

The other advantage of wavelets over block-based DCT (and even over global DCT)
is that they naturally lend themselves to level of detail support. We can recreate coarser
resolution versions of meshes by simply decimating (picking every other sample in both
dimensions) the Geometry Images. Due to the multi-resolution support of wavelets a low
resolution version of an image can be produced with lower computational overhead.

5.2 Temporal Coherence

There is a great deal of temporal coherence in 3D animations. Animations are often de-
scribed with smoothly varying mathematical functions (see section 2.2.1) of the vertices. In
an ideal world, we would be able to express the animation by the mathematical formulae
used to generate them. Unfortunately, without prior knowledge from the simulation sys-
tem it is computationally difficult to “derive” these generating formulae. Additionally, the
data may have originated from a computationally expensive process or from a hand-tuned
animation process without a physical foundation. We prefer to encode mesh animations
without knowledge of the generating process.

We can approximate the animation motion with a simple process, and only encode the
difference between our best guess and the actual change. This is commonly called predictive
encoding. In MPEG, video frames are predicted as a function of either the previous, or the
previous and the future frames. The function used for images in this case is a direct block
copy. The image is partitioned into 16x16 macro-blocks. The each 16x16 macro-block is
predicted from the best matching window in the previous (or reference) frame. For 3D
animations, or specifically for sequences of Geometry Images, this would not work well. A
block copy with Geometry Images would correspond to predicting the location of some patch
of the mesh from the location of another patch. For example, in an animation of a running
horse it would be highly unlikely that the location of the leg in one frame correspond to the
location of any other body part in the reference frame.

It is clear that a more geometrically-based predictor is desired. Our predictor approach
is similar to MPEG in that we use reference frames as a predictor. We differ in the kind
of predictor function used. In the next section we present our choice of predictor function.
This will be followed in section 5.4 by a description of how frames are encoded.
5.3 Transformations

The best prediction function would be exactly the one used to generate the animation. For example, for some character animations, we could derive a skeleton from the first frame and use a transformed version of this skeleton to skin and predict future frames. For animations with free form deformations, we could predict the scaffolding used on a reference frame to produce the current frame. For a physical simulation of an earthquake, it would be hard to guess the function without previous knowledge of how it was computed.

Although it is hard to compute exactly the mathematical transformations (if they exist or can be represented compactly), we can effectively approximate them. We propose to use a simple affine transformation as our predictor function. We can determine a transformation to represent changes between frames as a first approximation. Then we only need to encode the difference, or delta between this approximation and our target frame. This is similar to the Predictive frames in MPEG, with the difference that we don’t search for an appropriate motion vectors, but rather solver for an optimal affine transformations. Finding the best motion vectors in MPEG is a hard problem and the solution usually involves an exhaustive or hierarchical search. In contrast, finding an optimal affine transformations is a well-defined problem with a closed-form solution (in the least-squares sense). This is possible because we know the exact motion and correspondence of the object - not a 2D flat vision of the world or motion. This idea of using transformations as a form of predictors is not new; Lengyel [Len99] mentions it in the direct context of compression of time-varying geometry.

Affine transformations directly represent common rigid-body motions (rotations, skewing and scaling effects) found in animations, and they are compact to represent. As a first order approximation, the motion of objects can be described as translations and rotations. Translations, rotations, skew, and scale changes are all affine transformations and can be represented with a transform with 12 degrees of freedom in three dimensions; they can be represented compactly with 12 floating point numbers. We use the residual or difference between the predictor and the current frame to compensate for any motion or change not represented by limiting ourselves to affine transformations. We find that the Geometry Image of residuals is typically smooth like those of the actual frames; thus Geometry Images of residuals compresses well.

We propose to apply affine transformations as predictors in one of two ways: globally
or locally. The first approach uses one transformation applied to the whole reference frame to build a predictor of the current frame. We view this as a *global* transformation. These transformations capture the high-level changes between frames. Alternatively, we would like to capture more local changes. For example, in a running sequence, only the arms and legs may move while the torso and head remain mostly fixed. These scenarios motivates us to look for a more *local* transformation. Geometry Images have the property that samples adjacent to each other in the image each are adjacent to each other in the reconstructed mesh. Therefore, we also look into transforming patches of the Geometry Image individually; this can allow us to predict the motion or changes of different parts of the surface independent of each other. We now look at these two kinds of transformations in more detail.

### 5.3.1 Global Transformations

A single affine transformation with translation and rotation can capture the rigid-body motion of the object through space. Changes within an object's shape can be approximated by scale and skew changes. With predictive encoding, we encode a frame as a transformation applied to a reference frame plus its residual or error. We compute the global transformation between a reference Geometry Image $g_{iprev}$ and the target $g_{itarget}$ in the following manner:

We can think of Geometry Images as a list of 3D points. Let $g^{x}$, $g^{y}$, and $g^{z}$ be all the $x$, $y$, and $z$ coordinates of the Geometry Image respectively. Let $g_i$ be an $N^2 \times 4$ matrix, where $N$ is the width of the Geometry Image. The $i$th row of $g_i$ contains $[g^{x}_i, g^{y}_i, g^{z}_i, 1]$.

We compute $T$, the affine transformation, as the least square's solution that minimizes:

$$||g_{iprev} \ast T^T - g_{itarget}||$$  \hspace{1cm} (5.1)

$T$ will be a 4x4 matrix whose last row is $[0, 0, 0, 1]$. The 12 entries in the first 3 rows represent the translation, rotation, scaling, and skewing in the $x$, $y$, and $z$ axes. Computing this transformation can be done on sub-sampled Geometry Images to reduce computational time. The regular sampling of Geometry Images make the sub-sampled version a good approximation and hence also a good reference for computing transformation $T$.

The residual Geometry Image will be:
\[ g^i_{\text{predictor}} = g^i_{\text{prev,global}} = g^i_{\text{prev}} * T^T \]  

\[ g^i_{\text{residual}} = g^i_{\text{predictor}} - g^i_{\text{target}} \]

The residual \( g^i_{\text{residual}} \) is encoded using the same mechanisms used to compressed regular Geometry Images. This transformed reference frame is smooth and subtracting it from another smooth function (the current frame) also, generally results in a smooth function. Thus Geometry Images of the residuals tend to be smooth. The residual Geometry Image has spatial coherence and compresses well with wavelets. Figure 5-2 shows an example of the application of Global Transformations.

Finally, affine transformations are naturally supported by common graphics hardware. The operations mentioned above are common in different stages of the rendering pipeline. Hence this process is well suited for hardware implementation with minimal or no change to current high-end graphics cards.

### 5.3.2 Local Transformations

Unfortunately, global transformations are poorly suited for capturing large local changes. In the method described above, a single transformation is applied to the reference frame. We would also like to be able to capture local changes in the geometry. From the example in Figure 5-2, we can see that the predictor (red snake in (b)) cannot capture the bending of the tail in the snake body; despite doing its best to approximate the target frame (blue snake in (b)).
Recall that contiguous regions in the Geometry Images correspond to contiguous regions in the reconstructed mesh. Therefore, it is possible to capture local changes by breaking the Geometry Image into patches and applying separate transformations to each patch. For simplicity we divide the Geometry Image into \( \frac{N}{\sqrt{nb}} \times \frac{N}{\sqrt{nb}} \) square blocks, where \( N \) is the width of the Geometry Image and \( nb \) is the number of blocks. Figure 5-3 shows an example of what blocks can represent in a Geometry Image. In our initial implementation, we applied the same algorithm as global transformations to each patch. However this approach introduced visually apparent boundary effects. Since the transformations of the patches are computed independently of their neighbors and we are minimizing the total least-square difference per patch, the boundaries of the patches are rarely aligned. These boundary effects lead to noticeable artifacts in the residual Geometry Image, which in turn leads to poor compression.

Non-overlapping blocks are only one way to partition the Geometry Image. The advantage of this partition is that it is simple to compute and easy to process. An alternative partitioning might look at curvature and try to determine significant parts of the object. Such partitions would probably be more efficient and yield better compression. These regions could be described compactly with a Geometry Image representation because pixels adjacent in the Geometry Image are adjacent in the reconstructed mesh; this is unlike the traditional mesh representation where vertices are unordered.

![Figure 5-3](image)

**Figure 5-3:** (a) Geometry Image. (b) Partitioning of Patches. (c) Two views of reconstructed Geometry Image showing the correspondence of patches.

We can reduce the boundary-induced artifacts between the transformed patches by in-
cluding overlapping neighborhoods in the computation of the optimal patch transformations. Transformations for adjacent patches will be more similar because they are computed using a subset of common vertices. Unfortunately, the boundaries of the transformed patches still do not match and artifacts are still visible in the residual Geometry Image.

The solution to the problem involves blending the neighboring vertices around the patches to reduce or eliminate the boundary effects. This is a sensible solution; the skinning technique used for 3D animation uses blending near skeleton joints to produce smooth transition between the skinning of different bones.

![Diagram showing local transformations and blending](image)

**Figure 5-4:** 1D example of local transformations. (a) Reference Frame. (b) Target Frame. (c) Best Match using global transformation. (d) Local transformations of patches and their reconstruction. (e) Local transformation of patches using neighboring sections. The discontinuity in the reconstruction is smaller. (f) Same as (e) but blending neighboring information. Notice that the discontinuity in the reconstruction is mostly smoothed out.

We have applied the previous two techniques to compute the transformations for each patch. By using overlapping neighbor vertices, we are able to produce more similar transformations and patch boundaries that are closer to each other, and we apply the transformation both to the patch vertices and to some neighboring set of vertices; We keep the transformed position of the patch vertices and blend the position of the neighboring vertices with their neighboring transformed patches. We use decaying weights, so that vertices farther away from a patch have a smaller effect on the neighboring patch than closer vertices. Figure 5-4 shows a simple 1D example of local transformations and blending.

The resulting Geometry Image $g_i$ is derived as follows. Let $g_i(u, v)$ be the vertex in position $(u, v)$.

$$g_i(u, v) = g_{i\text{ predictor}}(u, v) + g_{i\text{ residual}}(u, v)$$  (5.4)
\[ g_{\text{predictor}}(u, v) = g_{\text{transformed}}(u, v) = \frac{\sum_{p \in \text{patches}} w_p(u, v) \cdot g_{\text{prev}, \text{local}}(u, v) \cdot T_p^T}{\sum_{p \in \text{patches}} w_p(u, v)} \quad (5.5) \]

where \( p \) is a patch. A point \((u, v)\) is in the patch region \( p_{\text{region}} \) if \( u_{\text{start}} \leq u \leq u_{\text{end}} \) and \( v_{\text{start}} \leq v \leq v_{\text{end}} \). \( w_p(u, v) = 1 \) if \((u, v) \in p_{\text{region}}\) and \( w_p(u, v) = f(\text{dist}) \) if \((u, v)\) is \( \text{dist} \) distance from \( p_{\text{region}} \). Where \( f \) is a monotonically decreasing function from 1 to 0.

The union of all the patch regions covers the whole Geometry Image.

Geometry Images have regular connectivity and strive for uniform sampling of the target frame. Because of this, we can efficiently compute a good approximation of the neighbors and weights for each patch in the Geometry Image except for the few patches on the boundary of the Geometry Image. The neighbors of the vertices that lie on the boundary of the Geometry Image can be anywhere else in the boundary of the Geometry Image and not directly opposite to them. Figure 5-5 shows two examples of blending weights \((w_p)\) for patches – one of them of a patch on the boundary. The relationship of the boundary vertices will be dictated exclusively by the mapping and cut of the original mesh. Their location can be computed from the small sideband information needed to “stitch” or fuse the cut. Besides being computationally efficient, the layout of the vertices and weights for the patches will have very good memory cache behavior.

Figure 5-5: (a) Weights for patches not on boundary. (b) Weights for a patch on the boundary. The neighbors to the boundary vertices can lie anywhere on the Geometry Image; their location will be a function of the cut used and how the cut was mapped onto the boundary.

Figure 5-6 shows the previous global transformation example (see Figure 5-2) but using local transformations. Notice that the predictor (red snake in (b)) more closely resembles the reconstructed frame, resulting in a smoother Geometry Image for the residual.
**5.4 Frame Types**

In the previous section we discussed two different ways to build a predictor from a reference frame. In this section we present two different ways to use predictors. The encoder for geometry videos is very similar to that of an MPEG encoder. Like MPEG, much of the compression is achieved by predicting frames from previous and/or future frames. Like MPEG, we compress a mesh or a frame as one of three types: “I-Frames” are frames that are encoded independently of any other frame. “P-Frames” are encoded as the difference between a prediction based on a previous frame and the target frame. The third type of frames are “B-Frames,” which as based on a prediction, like P-Frames, but the prediction is based on both previous and future frames. We now describe each frame type in greater detail:

**I-Frames**  The geometry image is encoded independently of any other information. We encode the geometry image using wavelets. The compression in this case is derived completely from the spatial coherence. I-Frames are also useful in a lossy networking environment, since they provide a natural synchronization point – they are self-contained.

**P-Frames:** An alternative way to encode a geometry image is to use a transformation of the previous geometry image as a predictor. We use either the global or local affine transformation methods described in Section 5.3 and the previous frame as the reference frame. Any reference frame can be used, but using the previous reference frame typically...
yields a reference frame that is most similar to the current frame and hence yields a good
predictor. The difference between the target frame and the transformed reference frame
(predictor) is the residual Geometry Image. This Geometry Image is then encoded in much
the same manner as I-Frames. This residual Geometry Image turns out to be fairly smooth,
so that we can effectively use the same wavelet basis functions for compressing it.

Unlike motion vectors in MPEG, it is possible to reverse the playback if global affine
transformations are used for predictors. This kind of transformation is invertible. Therefore,
it is possible to implement reverse playback storing only the transformations and residuals,
instead of the full Geometry Images. To play the previous frame, we would subtract the
residual to the current frame and apply the inverse global transformation. This only applies
if we are using global transformations for our predictors; it is a much harder problem, if not
impossible, if we use local transformations.

B-Frames Predictors can also be based on future frames. For even greater flexibility,
they can be based on some combination of both a future and a previous reference frame.
This frame type is commonly used in MPEG video encoding. They are advantageous for
many reasons: they compress well, they allow easy reverse playback support, and they can
efficiently encode occlusions and disocclusions. Although there is no analogous effect in
3D, occlusions and disocclusions can be thought of as features that appear or disappear
on objects. For example, a mutant growing a second head. We also support B-Frames as
a way to obtain temporal information from the future. In this case, the future reference
frame can capture the new features as they appear.

To encode a frame as a B-Frame, we use the same process as the P-Frames plus an
additional step. First, we compute two transformations: one from the previous reference
frame to the current frame and another from the future reference frame to the current frame.
We now have two predictors for the current frame. We would like to combine these two
predictors to build a single predictor for the current frame. We build a single predictor using
a blending function. We compute a blend between these two predictors that minimizes the
distance, in the least squares' sense, to our target frame.

The general form of the blending function is:

$$g_{\text{target}} = \alpha \cdot g_{\text{prev}}^{\text{transformed}} + \beta \cdot g_{\text{next}}^{\text{transformed}} + \gamma + \epsilon$$

(5.6)
Where \( g_{\text{target}} \) is the frame we are encoding, \( g_{\text{prev}}^{\text{transformed}} \) and \( g_{\text{next}}^{\text{transformed}} \) are the transformed reference frames, \( \gamma \) is a constant, and \( \epsilon \) is the residual error. We want to choose \( \alpha, \beta, \gamma \) to minimize the error \( \epsilon \). We do this for each image plane, i.e. for \( x, y, \) and \( z \). Considering only the \( x \) plane, equation 5.6 can be written as:

\[
\begin{bmatrix}
\vdots \\
\begin{bmatrix}
x_{\text{target}} \\
\vdots
\end{bmatrix} &= 
\begin{bmatrix}
x^{\text{transformed}}_{\text{prev}} & x^{\text{transformed}}_{\text{next}} & 1
\end{bmatrix} 
\begin{bmatrix}
\alpha_x \\
\beta_x \\
\gamma_x
\end{bmatrix}
+ \begin{bmatrix}
\vdots
\end{bmatrix} 
\end{bmatrix}
\]

(5.7)

the solution to those parameters for the \( x \) plane, that minimizes \( \epsilon \) is:

\[
\begin{bmatrix}
\alpha_x \\
\beta_x \\
\gamma_x
\end{bmatrix} = (A_x^T A_x)^{-1} A_x^T B_x
\]

(5.8)

where \( A_x = \begin{bmatrix}
x^{\text{transformed}}_{\text{prev}} & x^{\text{transformed}}_{\text{next}} & 1
\end{bmatrix} \) and \( B_x = \begin{bmatrix}
\vdots
\end{bmatrix} \).

For greatest flexibility, we can blend the previous and future reference frame either globally or locally. In the local case we use the same patches and weights \((w_p)\) as local transformations. Local blending is useful if we want to extract features from different reference frames.

The resulting Geometry Image \( g_i \) derived from local blending is computed in the following manner. Let \( g_i(u, v) \) be the vertex in position \((u, v)\).

\[
g_i(u, v) = g_{i,\text{predictor}}(u, v) + g_{i,\text{residual}}(u, v)
\]

(5.9)

\[
g_{i,\text{predictor}}(u, v) = \frac{\sum_p w_p(u, v) \ast (\alpha_p \ast g_{i,\text{prev}}^{\text{transformed}}(u, v) + \beta_p \ast g_{i,\text{next}}^{\text{transformed}}(u, v) + \gamma_p)}{\sum_p w_p(u, v)}
\]

(5.10)

where \( p \) is a patch. A point \((u, v)\) is in the patch region \( p_{\text{region}} \) if \( u_{\text{start}} \leq u \leq u_{\text{end}} \) and \( v_{\text{start}} \leq v \leq v_{\text{end}} \). \( w_p(u, v) = 1 \) if \((u, v) \in p_{\text{region}} \) and \( w_p(u, v) = f(\text{dist}) \) if \((u, v)\) is \text{dist} distance from \( p_{\text{region}} \). Where \( f \) is a monotonically decreasing function from 1 to 0. The union of all the patch regions covers the whole Geometry Image. \( g_{i,\text{prev}}^{\text{transformed}} \) and \( g_{i,\text{next}}^{\text{transformed}} \) are computed using local transformations according to equation 5.5.

B-Frames work well for key-frame animations, a common animation technique. In key-
frame animations many frames are interpolated from a small set of selected key-frames; the interpolation is not an exact interpolation of the some key-frames, but usually an interpolation of some parameter between the two key-frames, i.e. rotation. B-Frames can generate good predictors by blending the transformed versions of the key-frames. In these scenarios, better compression will be achieved if the reference frames in the encoded animation correspond to the actual key-frames of the input animation; the difficulty lies in discovering which frames are key-frames on an animation of unknown source.

There are two major disadvantages with the use of B-Frames: added memory footprint and added latency. The first drawback stems from the fact that we now have to store two reference frames. This is not generally a problem for desktop machines, but could be a problem for small wireless devices. The second drawback, added latency, is caused by the need to decode a future reference frame before the current frame can be decoded and the additional overhead of blending the two reference frames.

These frames types correspond to the frame types used in MPEG with the changes outlined above. The major advantage of our approach is that all the knowledge and techniques that are used for encoding video streams using the aforementioned frame types can be directly applied to encoding 3D animations. With these three possible frame types we can encode a stream of Geometry Images very compactly.

### 5.5 Summary

In this chapter we have looked at how we compress a sequence of Geometry Images. We revisited how to take advantage of the spatial coherence that exist in Geometry Images. We then addressed temporal coherence. By using predictive coding it is possible to reduce the information necessary to encode a Geometry Image. The key observation is that unlike video, computing the predictors between frames is less computationally intensive (no searching) than computing motion vectors for standard video. The residual Geometry Image obtained from this process is also very smooth and compresses well. Finally, we looked at the different ways we can use predictors. We can encode frames using a combination of predictors based on both previous and future frames. In the next chapter, we bring together all of the elements of the compression system and present the implementation details of a system to encode, represent, and replay arbitrary manifold 3D animations.
Chapter 6

System

In this chapter the implementation details of the Geometry Video pipeline are described. The system is divided into two parts: an encoder and a decoder. We have implemented two different decoders, one for making error measurements and comparisons and another one for real-time playback of the geometry videos. In the previous chapters we have presented the theory and design decisions related to how to adapt and compress a sequence of Geometry Images when encoding a 3D animation. Here we’ll look at the actual details and controls of a complete working system.

In section 6.1, the whole compression system is presented which is similar to a standard video compression system (MPEG [Gal91]). Next, the system is described bottom up, from a simple model to a more complex model by progressively adding complexity and enhancements. We present the encoder and decoder at the same time because much of their functionality is the dual of the other; furthermore, the encoder has a built-in decoder to model what the decoder sees. In section 6.2, we describe how different components (like the Geometry Images and transformations) are represented in our system, and provide design rationale. In section 6.3, we explain our choice of compression controls and their effect on the system. We conclude in section 6.4 by explaining how our choice of implementation is well-suited for lossy (i.e. networking) environments and suggesting how to choose parameters for various environments.
6.1 Encoder/Decoder

The encoder for geometry videos is very similar to that of an MPEG encoder. Like MPEG, much of the compression is achieved by predicting frames from previous and or future frames. Like MPEG, we compress a mesh or a frame as one of three types: I-Frames, P-Frames, and B-Frames. I-Frames are frames that are encoded independently of any other frame. P-Frames are encoded as the difference between a prediction based on a previous reference frame and the target frame we are encoding. The third type of frames are B-Frames, which as based on a prediction, like P-Frames, but the prediction is based on both previous and future reference frames. These future frames can only be encoded as I-Frames or P-Frames.

Although we are encoding images, they are different from your traditional video images. Geometry Images are a 2D mapping corresponding to actual points on a manifold embedded in 3D space and are very smooth. In contrast to MPEG, we use wavelets to encode Geometry Images. They achieve higher compression than the Discrete Cosine Transforms used in MPEG and they don’t suffer from the block artifacts. Furthermore, by using wavelets, it is easy to allow for spatial scaling of the geometry images and add error correction for lossy environments.

We form our inter-frame predictions differently from MPEG. In MPEG, the frame is divided into blocks and predictions are made for each block. The prediction for a given block of pixels is based on finding a block of pixels of the same size on another frame that closely matches the given block. It the most computational intensive part of the video encoding. Block-based motion compensation assumes that motion in a video can be approximated a translation across frames. However, because pixels in Geometry Images correspond to 3D vertices, we are able to use a more geometrical-based predictor. We use affine transformations to capture the rigid body motion of the object in space as well as small changes in the shape of the object. For larger changes in the object, we can compute different affine transformations on different parts of the object; we call these local transformations. For simplicity, we divide the Geometry Image into equal-sized square patches that correspond to different parts of the original mesh. Table 6.1 summarizes the differences between MPEG and Geometry Videos.

For rapid prototyping, the encoder and decoder was implemented in Matlab 6.5, and later a decoder was implemented in C++ for interactive viewing and rendering.
The encoding process can be broken down into two parts. The first part, a pre-processing step, converts a 3D animation into a sequence of Geometry Images. The second part is the actual encoding of the Geometry Video. Each frame in the Geometry Video is encoded as one of three types of frames: I, P, or B-Frames. P-Frames are built using predictors, or in our case affine transformations of a reference frame. B-Frames build on top of P-Frames and are computed by blending two predictors. These transformations and blending can be done either globally, using one transformation and/or blend for the whole Geometry Image, or locally, using one transformation and/or blend for each patch of the Geometry Image. We explain the encoding process in this order: the pre-processing, I-Frames, P-Frames and transformations, and B-Frames and blending. We leave the discussion of the exact choice of frames and parameters for section 6.3

6.1.1 Pre-process

The pre-processing step generates a sequence of Geometry Images that are well-suited for compression. We use the same cut and parametrization for all the meshes in an animation (see chapter 4). For this, we run a pre-processing step that computes the cut and parametrization to be used for all meshes in the animation. Should the genus of the mesh change or should the cut/parametrization not yield good results, we can detect it and "reset" the cut and parametrization; this is analogous to a scene change on a video. Additional care is needed to make sure there is a smooth transition between two frames of meshes with a different cut and/or parametrization. We leave such functionality for future work and limit ourselves to manifold animations that do not change topology or connectivity.

The very first thing we need to compute is the geometry image size. This will have a

<table>
<thead>
<tr>
<th>Encoding</th>
<th>MPEG</th>
<th>Geometry Videos</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Discrete Cosine Transform +</td>
<td>Wavelets + Embedded Zero</td>
</tr>
<tr>
<td></td>
<td>Variable Length Coding +</td>
<td>Tree encoding + Arithmetic Coding</td>
</tr>
<tr>
<td></td>
<td>Run Length Coding</td>
<td></td>
</tr>
<tr>
<td>Prediction</td>
<td>Motion Compensation on blocks</td>
<td>Affine Transforms (global or on blocks)</td>
</tr>
<tr>
<td>B-Frames</td>
<td>Prediction based on previous, future or average of the motion compensated reference frames</td>
<td>Prediction based on linear combination between transformed previous and future reference frame.</td>
</tr>
</tbody>
</table>

Table 6.1: Comparison of MPEG and Geometry Videos
great impact on the performance of the system. It will:

- Affect how the boundaries of the cut mesh are mapped onto the geometry image.
- Determine the number of vertices of the decompressed mesh. A larger image will result in lower performance because more vertices have to be processed.
- Characterize the fidelity and reconstruction quality of the resulting mesh. The larger the image size, the more sampling and greater fidelity.
- Affect the compression of the geometry image

The user specifies the resolution of the Geometry Image used each animation. As a heuristic, we pick an image whose resolution is about 5 times greater than the number of vertices as the original mesh. This is reasonable for meshes with low number of vertices (less than 10000). For bigger meshes, we can obtain good results with image sizes containing the same or slightly more vertices. Most of our results are computed with Geometry Images of size 256x256.

The domain of our Geometry Images is currently restricted to a square. Sander et al. [SSGH01] has shown that using free-boundaries (non-square) Geometry Images produce better reconstruction with less distortion at the boundaries. The trade-off is higher complexity in the processing of such non-square Geometry Images; Compression would also be affected by the fact we don’t have a regular square grid of values; it is harder to take advantage of the spatial coherence in the image. Lastly, Geometry Images with free-boundaries would be harder to parallelize and optimize for hardware implementation. Rectangular Geometry Images would be a compromise to square Geometry Images. This would require analysis of the input to select the right aspect ratio. Although we have not tried this, we believe the gains would be small and only applicable to a sub-class of animations.

Given the image size, we then want to compute the cut before parametrizing the animation sequence. We can compute this either on a single frame or based on all frames (globally). If we use a single frame, the choice of frame is important. This frame can be chosen either from some heuristic or by a brute force consideration of all choices. We use a global cutting algorithm (see Section 4.1.2) because it gives us consistent results and can find better cuts for certain animations than using only a single frame.
Algorithm 6.1.1: FINDGLOBALCUT(Meshes(1..N))

Where each Mesh has $N_F$ faces and $N_V$ vertices, all with the same connectivity
Pick one seed triangle and remove it from all Meshes(1..N)

while there are edges $e$ adjacent to only one triangle $t$

  do Remove $e$ and $t$ from all Meshes(1..N) 

enddo
Handles meshes with genus > 1

while there is a vertex $v$ with degree 1 on edge $e$

  do Remove $v$ and $e$ from all Meshes(1..N)

enddo

Cut ← Remaining edges and vertices

if Cut is empty

  then Cut ← two arbitrary connected edges

end if

MeshesCut(1..N) ← APPLYCUT(Meshes(1..N), Cut)

while true

  Floater(1..N) ← FLOATERPARAMETRIZATION(MeshesCut(1..N))

  % GeometricStretch averages the normalized $L_{\infty}$ norm of each mesh over all meshes
  $L_{\max}(1..N_F), L_{2avgOld}$ ← GEOMETRICSTRETCH(MeshesCut(1..N), Floater(1..N))

  % Compute Stretch after the cut
  $badt$ ← index of $\max(L_{\max})$
  NewCut ← Cut + shortest path from $badt$ to Cut

  MeshesNewCut(1..N) ← APPLYCUT(Meshes(1..N), NewCut)

  FloaterNew(1..N) ← FLOATERPARAMETRIZATION(MeshesNewCut(1..N))
  $L_{2avgNew}$ ← GEOMETRICSTRETCH(MeshesNewCut(1..N), Floater(1..N))

  if $L_{2avgNew} > L_{2avgOld}$

    then break

  end if

  MeshesCut(1..N) ← MeshesNewCut(1..N)

  Cut ← NewCut

end while

output (Cut)

Algorithm 6.1.1 presents the global cutting algorithm which produces a list of cuts. Each cut is a list of vertices. For genus-0 objects, these cuts form a tree; for higher n-genus ($n > 0$) objects these cuts form a cyclic graph containing $2n$ cycles with subtrees.

The last step is the actual parametrization. Ideally, we would like to optimize the
parametrization by considering all frames. However, we do not have a global parametrization algorithm and instead compute the parametrization based on an arbitrary frame in the animation sequence (the first frame) using the global cut. We have found that the parametrization is primarily determined by the cut, with little dependence on the frame choice; hence, picking an arbitrary frame with a global cut for the parametrization yields satisfactory results. We make the first frame the reference mesh.

We cut the reference mesh using the global cut to create an open mesh to be parametrized. We keep a map of pairings or mates for each vertex that was cut for mapping the edges to the boundary of the image and for computing the stitching information.

The next step involves computing and preparing the boundary of the parametrization. The length of the boundary edges of the parametrization (those at the boundary of the square) should be proportional to the length of the edges in the mesh (those at the cuts).

We want the cut-node vertices (those vertices where a cut starts or meets another cut) of the cut boundary to lie on grid locations on the boundary of the Geometry Image. If cut-nodes do not lie on grid locations, they will not be sampled and hence it will be difficult to seal or fuse the boundary at these locations. This implies that the sum of the lengths of the edges between cut-nodes must be an integer value. To determine the distance between all cut-nodes on the boundary of the Geometry Image we first normalize the lengths between the cut-node vertices and multiply them by the perimeter of the image. To obtain an integer value, we take the floor function of their lengths. To make sure it adds up to the perimeter of the image, we then iteratively increment the lengths of the edges (in order of minimum percentage change) until they add up to perimeter. Notice that we increment two edge lengths at a time since each edge on the boundary has a complementary pair.

The last step in mapping the boundary choosing the vertex that will be mapped to position (0,0) on the Geometry Image. We find, like Gu et al. [GGH02] that mapping cut-nodes to this location produces unsatisfactory results. We therefore map a vertex that is furthest away from two adjacent cut-nodes to this location.

The mapping of the cut boundary onto a square can produce two degenerate cases:

The first degeneracy is zero-area triangles. If two edges of a triangle are mapped onto the border of the Geometry Image, the triangle will be flatten and have zero-area. This is fixed by splitting in half the edge that is not mapped to the boundary. Now you will have two triangles and they will have non-zero area when parametrized (See Figure 6-1).
Figure 6-1: If a cut passes through two edges of a triangle, the triangle will be mapped to a flat triangle on the boundary of the geometry image. This is fixed by splitting the edge that is not mapped.

The second degeneracy result from unmapped corners. This can happen when the mapped edge on the Geometry Image spans a corner. We solve this by splitting the edge (and its complementary pair) on the input mesh (See Figure 6-2).

Figure 6-2: If an edge is mapped around a corner, the edge has to be split; otherwise, the corner will not be mapped.

We record the edges that are split, so that this operation can be performed on all frames of the input animation. This is necessary to maintain geometric coherence between the frames. The boundaries for each frame should be mapped in the same way.

We also use this boundary mapping to produce the sideband stitching information. Since we are using lossy compression (wavelets), cracks will be visible at the boundary of the cut. For this, the stitching information provides a correspondence between complementary-pair vertices on the boundary of the geometry image. We can then use this correspondence information to make sure that at reconstruction time, these vertices have the same value.

We build a parametrization that minimizes the average $L_2$ geometric-stretch (See Section 3.1.2). There are two inputs to build the parametrization: an opened mesh (topologically equivalent to a disk) and the mapping of the boundary of the mesh to the boundary of the parametrization. The parametrization algorithm is outlined in section 3.2.3 and it minimizes the average geometric-stretch of the parametrization. We run these inputs through the parametrization code used in [GGH02] to obtain a mapping of vertices from the prepared
cut mesh to vertices inside a unit square ($x, y$ locations).

The steps of the pre-processing stage are summarized in algorithm 6.1.2. With this technique we can generate consistent Geometry Images for each mesh in the 3D animation. We now proceed to explain how we can encode this sequence of consistent Geometry Images.

Algorithm 6.1.2: PRE-PROCESS($Meshes(1..N)$)

\[
\begin{align*}
\text{imagesz} & \leftarrow \text{manually select image size} \\
Meshes_{\text{Cut}}, Cuts & \leftarrow \text{FINDGLOBALCUT($Meshes(1..N)$)} \\
Meshes_{\text{CutBoundary}}, BoundaryMap, Cuts_{\text{Boundary}} & \leftarrow \text{MAPBOUNDARY($Meshes_{\text{Cut}}, \text{imagesz}$)} \\
\text{comment: Cuts}_{\text{Boundary}} & \text{ may add additional vertices to fix corners and zero area triangles} \\
\text{Parametrization} & \leftarrow \text{PARAMETRIZE($Meshes_{\text{CutBoundary}}, BoundaryMap$)} \\
\end{align*}
\]

With a cut and a parametrization we can then proceed to quickly map each mesh in the animation into Geometry Images. This can be done very rapidly. Cutting, preparing the borders and parametrizing the meshes for each time-step in the animation can be done very quickly once we have one reference mesh that has been cut and parametrized. For all other meshes, we can use the same connectivity as the reference mesh and duplicate and manipulate their vertices as necessary. We rasterize all the meshes using the same parametrization.

The rasterization step is simple. We map the $<x, y, z>$ values of each vertex in the mesh as the $<\text{red}, \text{green}, \text{blue}>$ color values of the parametrization. We rasterize the parametrization using linear interpolation and sample it at grid coordinates. This step could be trivially accelerated using commodity graphics hardware.

The final output of the pre-processing step is a sequence of Geometry Images corresponding to the meshes at each time-step in the animation, and sideband information used to fuse the cut when the Geometry Images are reconstructed.

We now progressively describe the encoding process, starting with a method that encodes each frame independently and ending with a system that uses predictors to achieve higher compression rates.
6.1.2 I-Frames

The simplest encoder would be one that encoded each frame independently of each other. The wavelet coder would take advantage of the spatial coherence that exists in the geometry image of each frame. Because each frame is encoded independently, this would not exploit the temporal coherence that exists throughout time. This is exploited using predictors as will be shown in the later subsections.

The pipeline for each frame is shown in Figure 6-3. We use the cuts computed in the preprocessing step to open the mesh_input, generating mesh切割boundaries. These cuts include both the initial cuts and any cuts we use to fix the degeneracies in resulting in mapping the boundaries. The cut information is also used to generate the stitch sideband information. Since we make the same cut on all frames, we only have to record stitch information once.

![Figure 6-3: I-Frame Encoder/Decoder](image)

The next step is the Rasterization. In this step we convert the open mesh切割boundaries into a Geometry Image (gi). This steps takes as input the parametrization, which is a mesh with the same connectivity of the mesh切割boundaries, with vertices on the unit square (no z information). There is a one to one correspondence between vertices in the parametrization and the mesh切割boundaries. We use the location < x, y, z > information of the vertices in mesh切割boundaries to color the vertices in the parametrization. We stretch the parametrization to match the size of the geometry image. Lastly we sample the grid location in the geometry image from the parametrization.
We use the wavelet encoder from Ng Mow-Song [MS02]. This is an integer-based encoder. We adapt the encoder to work with floating point numbers by multiplying the input by a large integer factor, and dividing the result when is decoded. This factor will depend on the size and location of the vertices in the animation. For our animations vertices tend to lie near the origin and the objects are no larger than one unit. We use a scale factor of $2^{20}$ to multiply the floating point vertex locations before encoding them. The wavelet encoder will not have higher precision than 20 bits, so this is satisfactory for us.

The decoder reverses each of the steps of the encoder. First, it decodes the image using the wavelet decoder. We divide by $2^{20}$ since the input was pre-multiply by that factor. The next step is the inverse rasterization ($Rasterize^{-1}$). In this step we simply map each pixel of the Geometry Image into vertex locations in 3D space. The $<x, y, z>$ position of the vertex will correspond to the $<red, green, blue>$ color of the pixel in the Geometry Image. There are many ways to triangulate the geometry image. We choose to triangulate it with a regular grid as shown in Figure 6-4. The two corners of the regular triangulation are triangulated differently to prevent degeneracies when we reconstruct the border. It is important to use the same triangulation throughout the animation; otherwise popping artifacts might be seen when the triangulation changes. Note that Gu et al [GGH02] use a different triangulation of the geometry image for static meshes. They connect all the vertices with their immediate north, south, east and west neighbors. Then for each square, they place the diagonal (with direction either north-west, or north-east) that spans the shortest euclidian distance between two opposite corners (there are only two pairs). We find that this optimization helps to reduce reconstruction error, but only slightly. In order to use it in an animation, this optimization should be computed considering all frames. We do not currently use this optimization.

The lossy nature of the wavelets will cause the complementary-boundary vertices not to fall on top of each other. This will be seen as a crack on the rendered mesh. To solve this, we use the stitch information to make sure that complementary-pair points in the boundary of the geometry image mesh correspond to each other. This is done by computing their average, and moving both complementary-pair points to the average location. For low-bitrates, more sophisticated algorithm that take into account the curvature of the mesh should be used. Gu et al. [GGH02] use an error diffusion process to further reduce artifacts that can be found near the borders.
Figure 6-4: (a) Regular Triangulation of the Geometry Images. Note that only two quads (corners) are triangulated differently (flipped). (b) shows a close up of a rendered Geometry Image border without flipping the corners. (c) shows a close up of a Geometry Image border with flipped corners.

We flip the triangles at two corners to avoid mapping two edges of the same triangle onto the border. Since border edges in a Geometry Image tend to represent straight edges in the re-projected geometry, mapping two edges of one triangle to the border leads to long, skinny, and possibly zero-area triangles.

It is important to note that the Geometry Image mesh has a different number of vertices and connectivity than the original mesh\textsuperscript{input}. The connectivity of the Geometry Image mesh is fully regular. The goal of the geometry image is to approximate the mesh\textsuperscript{input} as close as possible. There are two sources of losses in this system. First is the parametrization and rasterization stage. The rasterization step samples surface locations that may not correspond exactly to original vertex locations in mesh\textsuperscript{input}. Furthermore, there might be triangles that may not be sampled at all if they are small or do not span any grid point. The second source of loss is the wavelet coder. We use lossy wavelet compression to achieve high compression ratios at a very small loss of reconstruction quality.

6.1.3 P-Frames and Transformations

In order to take advantage of the temporal coherence that exists between frames, we use predictive encoding. This method works by transforming a reference frame to build a prediction of the current frame, and then only encoding the difference between this prediction and the actual current frame. Frames encoded in this manner are called "P-Frames".

The encoding of these frames is similar to I-Frames. We run the input mesh through the Cut and Rasterize stages to produce a Geometry Image, \(g_i\). At this point, want to compute a predictor for this Geometry Image based on a reference frame, in our case the previous frame. The predictor is built using one of two methods: Global Transformations or
Local Transformations. We use affine transformations for our predictor function. We finish describing the encoding and decoding process before explaining how the transformations are computed.

The current Geometry Image, $g_i$ can be expressed as the sum of the predictor Geometry Image, $g_{i\text{predict}}$, and a residual Geometry Image, $g_{i\text{residual}}$. The residual Geometry Image is smooth. In general, the $g_{i\text{residual}}$ will have the largest values in the location of the mesh that changed in a non-affine way, or could not be fitted by transforming the previous reference frame. Finally, we run this residual Geometry Image through the same last step of the I-Frame encoder and we compress it using a wavelet encoder.

The decoding process is straightforward. We decode the input to generate a residual Geometry Image. Using the transformations sent by the encoder, we transform the last reference frame ($g_{i\text{prev}}$), this becomes $g_{i\text{predict}}$. We add the residual to this predictor to generate the geometry image corresponding to the current frame. The boundary stitching is then performed so that no cracks can be seen on the rendered mesh. The decoded Geometry Image can then be used as a reference frame for another predictor. The inverse rasterization is the same as explained in the previous subsection.

It is important to note that the encoder also decodes its output stream to make sure any reference frames it uses to build predictors will be the same ones used by the decoder. Figure 6-5 shows a block diagram of how P-frames are encoded using one global transformation. We now describe in more detail how to apply and compute global and local transformations.

**Global Transformations**

To build a predictor using global transformations and a reference frame we need to compute a single affine transformation. We compute the global transformation $T$ by computing the least squares solution to:

$$\|g_{i\text{prev}} * T^T - g_i\| = g_{i\text{residual}}$$

$$\text{(6.1)}$$

The encoder outputs both the transformation $T$ and the compressed residual Geometry Image $g_{i\text{residual}}$. 

76
Local Transformations

Global transformations often fail to effectively capture local changes in the mesh. For this reason, we also support local transformations. The idea of local transformations is to partition the Geometry Image into overlapping patches and compute an affine transformation for each patch. To avoid boundary artifacts on the predicted Geometry Image we blend the points where the patches overlap. For simplicity and fast computation we use square patches on the Geometry Image (See figure 6-6).

The first step in using local transformations is partitioning the Geometry Image and computing the weights for the patches and the weights for the overlapping vertices. We divide the a Geometry Image of size $N \times N$ into $\frac{N}{\sqrt{nb}} \times \frac{N}{\sqrt{nb}}$ non-overlapping squares, where $nb$ is the number of blocks or patches. We define an overlap distance as how much we want the patches to overlap. For our results we set the overlap distance to be the same as the width of the patch; although the weights are small ($0.1$) for points that are 70% overlap distance away.
Figure 6-6: Example Blending Weights: (a) Color key for partitioning of Geometry Image. (b) Reconstructed Cow showing the correspondence of patches. (c) Patch weights including overlapping into adjacent regions; blue arrows show some neighboring patches to the bottom left corner patch.

To set the weights of the overlapping points, we use a flood-fill algorithm to find the points that are $i$ distance away from the patch for $1 \leq i \leq \text{overlap}$. We set the weight of vertices that are $i$ distance away using the error function (see Figure 6-7). For patches interior to the Geometry Image this is very simple to compute. For patches on the boundary of the Geometry Image, we have to “wrap around” the boundary. Due to how the cut is mapped on the boundary, the neighbors to vertices on the boundary may not lie directly on opposite side, but could lie anywhere in the boundary of the Geometry Image. We build an adjacency matrix of grid connectivity augmented by the neighbor relationship of vertices on the boundary. This adjacency matrix is used to direct a flooding algorithm for all patches. Figure 6-6 shows the patch weights of a Geometry Image.

We compute the affine transformation for each patch using all the vertices that have non-zero weight for that patch. The advantage of using the overlapping vertices for calculating the patch transformation is that it will lead to transformations that are more similar to the neighboring patches, reducing the distance of the vertices near the boundary of the patches. The predictor Geometry Image $g_{\text{predict}}^i$ is then computed as (same as equation 5.5):

$$g_{\text{predict}}^i(u, v) = g_{\text{transformed}}^i(u, v) = \frac{\sum_{p \in \text{patches}} w_p(u, v) \times g_{\text{prev,local}}(u, v) \times T_p^T}{\sum_{p \in \text{patches}} w_p(u, v)}$$  (6.2)

6.1.4 B-Frames and Blending

Predicting frames based on previous frames allows us to encode frames more compactly. We can augment this predictor by using information from the future. Here a reference frame in
the future is encoded independently as an I-Frame or based on a previous frame and decoded before the current frame. We can then predict the current frame using both a previous and a future frame. This kind of frame exists within MPEG and is called “B-Frame.”

Using a predictor based on future frames allows us to capture features that have not yet appeared. B-Frames can also be useful for encoding key-frame animations. Here an animator specifies a few key frames and the animation system interpolates the frames in between based on parameters from the key-frames.

Supporting B-Frames does not require many changes to the encoder and decoder. Essentially the encoder works the same as the P-Frame encoder, but with two small changes. First, we need to rearrange the order that frames are transmitted, so that when we decode B-Frames we have “seen” both the previous and future frame. This also implies that the system needs buffers to store at most two reference frames (as opposed to one in the P-frame). Second, we need to combine information from the previous and future frames.

We have chosen a predictor similar to the predictor used with P-Frames. Our B-Frame predictor is shown in Figure 6-8. We compute two P-Frame-like predictors and then combine them. The two P-Frame predictors can use either global or local transformations. The difference with P-Frames is that the reference frame of one of the predictors will be based on a future frame.

We combine the two predictors using a simple linear combination of the predictors plus a constant. As with P-Frames, we support two kinds of combinations or blending: Global and Local. With global blending, there is just one linear combination of the predictors, this can be useful if there is only one feature that changes. If many features change, as with P-Frames, local blending can capture different features from the predictors. With
Figure 6-8: The B-Frame predictor blends two predictors: one based on a transformation of a previous frame and one based on the transformation of a future frame.

Local blending, we compute a linear combination of the predictors on a per-patch basis as with local transformations. This allows us to interpolate different features from different predictors.

The general form of the blending function is:

\[
g_{\text{target}} = \alpha * g_{\text{prev, transformed}} + \beta * g_{\text{next, transformed}} + \gamma + \epsilon \tag{6.3}
\]

where \( g_{\text{target}} \) is the frame we are encoding, \( g_{\text{prev, transformed}} \) and \( g_{\text{next, transformed}} \) are the predictors built using a previous and a future reference frame, \( \gamma \) is a constant, and \( \epsilon \) is the residual error. We want to choose \( \alpha, \beta, \) and \( \gamma \) to minimize the error. We do this for each image plane, i.e. for \( x, y, \) and \( z \). For the global blending case, the constant \( \gamma \) is unnecessary. For the local blending case the constant \( \gamma \) is necessary to make the patches more aligned and hence reduce patch boundary artifacts on the residual Geometry Image. For local blending, we compute \( \alpha_p, \beta_p, \gamma_p \) for each patch \( p \) and form the predictor by blending and adding each patch together:

\[
g_{\text{predictor}}(u, v) = \frac{\sum_p w_p(u, v) * \left( \alpha_p * g_{\text{prev, local}}(u, v) + \beta_p * g_{\text{next, local}}(u, v) + \gamma_p \right)}{\sum_p w_p(u, v)}
\]

\[
g_i(u, v) = g_{\text{predictor}}(u, v) + g_{\text{residual}}(u, v) \tag{6.4}
\]
We can choose different constraints for the linear combination. For example, we can make $\beta$ equal $1 - \alpha$, or set $\gamma$ to 0. For flexibility, we allow $\alpha$ and $\beta$ to be independent of each other and allow $\gamma$ to be non-zero. For each plane, the solution to those parameters is:

$$
\begin{bmatrix}
\alpha_x \\
\beta_x \\
\gamma_x
\end{bmatrix} = (A_x^T A_x)^{-1} A_x^T B
$$

(6.5)

where $A_x = \begin{bmatrix}
\vdots \\
\text{transformed} \\
\text{transformed} \\
\vdots
\end{bmatrix}$, $B_x = \begin{bmatrix}
\vdots \\
x_{\text{target}}
\end{bmatrix}$, and $x_{\text{prev}}$, $x_{\text{next}}$.

Note that this is just one possible way to compute a prediction of the current frame based on previous and future frames. For example, the blending can be non-linear; it could interpolate some high-degree polynomial between the two predictors. Another possibility is to compute the blending and the transformation of the reference frames all at once. This is a more complex problem, and area for future work.

With these three possible frame types we can encode a stream of geometry images very compactly. In the next section, we discuss in detail how different parameters are represented. In section 6.3 we explain how we choose the different parameters.

6.2 Representation

In this section, we describe how different parameters are represented in the system.

Wavelets We use the Embedded Zerotree Wavelet implementation of Mow-Song [MS02]. We do not change any of its representation. To encode the Geometry Image, we encode each "color" or coordinate plane independently and store them sequentially in a tar-like file. Because this representation is embedded (hierarchical), we can decode the Geometry Image using any prefix of the wavelet representation, although this will result in an image with less fidelity.

We have a separate data stream which contains the rest of the parameters of our animation. This file contains all the frame encoding information including stitching information, transforms, and blend weights.
Stitch  We use a very simple flexible encoding for stitching information. We want to encode correspondences between the vertices at the boundary of the Geometry Image. This information is used to fuse the cut made to unwrap the mesh. When points in a mesh are cut, they are replicated, and hence they appear multiple times on the boundary of the Geometry Image. A Cut-node will either appear once if it lies at the beginning of a cut, or more than twice if it lies at the intersection of multiple cuts. Our encoding is a simple list of stitching structures. We have two kinds of stitching structures.

1. Sets containing the points on the boundary that correspond to the same vertex. This correspond to Cut-nodes that lie on the intersection of cuts.

2. Complementary-pair runs. Complementary-pair runs encode the correspondence of all points between two cut-nodes. Points between two cut-nodes on the same side of the cut boundary will be adjacent to each other on the boundary of the Geometry Image. Complementary-pair runs are thus encoded as the location of the first point on one side of the boundary, the location of the first point on the other side of the boundary and the run length.

All points, since they lie on the boundary, are encoded using their distance around the perimeter (clockwise) of the Geometry Image relative to a fixed point (0,0). For example, a 256 x 256 Geometry Image will have a perimeter of length 1020, which only requires 10 bits to index each point along the boundary of the Geometry Image. This is a simple encoding used for prototyping. The stitching information can be generated from a tree of cuts in addition to 2n cycles for objects of genus n and can be represented much more compactly than this.

Transformations  The transformation coefficients are quantized to achieve some additional compression. We encode transformation coefficients using 17 bits \((T_q)\) of precision. The transform coefficients are normally between -1 and 1. We allow scaling and distortion, and use a least squares approximation to compute the transform coefficients. Without major scale changes, we see that coefficients have a range between -2 and 2. We have found that encoding the transform coefficients as 17-bit floating point values (15 significant bits and two exponent bit) works well, and does not affect compression. For global transformation, we find that transformations coefficients can be quantized down to 11 bits; any
transformation change caused by the lack of precision will be "absorbed" by the wavelet encoder when the residual is encoded. For local transformations, because we want to avoid patch boundary mismatches, precision is important. Using heavily quantized coefficients for local transformations causes the boundaries of the patches to be less precisely aligned; The residual Geometry Image has to compensate for this, and hence it is harder to compress. We currently use 17 bits for local transformation coefficients. Each transformation is represented as a $4 \times 4$ matrix with four degrees of freedom in three dimensions; therefore we only need to encode 12 coefficients per patch when using local transformations.

We have not looked at compressing local inter-frame transformation coefficients. The transform coefficients for adjacent patches seem to be correlated so it should be possible to achieve some compression even by using a simple delta encoding.

**Blend Weights** B-frames are encoded using a blend of the transformation of the previous reference frame and the transformation of the future reference frame. We use a very flexible linear blending function that incorporates a constant and allows the sum of the weights for each reference frame be different than one. We have two weights, $\alpha$, $\beta$, and a constant $\gamma$; and we apply these to each axis independently. Therefore we have 9 coefficients for global blending. For local blending will have 9 coefficients per patch. We encode the coefficients as a 11-bit floating-point values (nine significant bits and two exponent bit). Again, as in local transformations, when using local blending it is important to have moderate precision with the constant term $\gamma$. The rationale is that mismatches between the patches leads to residual Geometry Images that are less smooth, and hence harder to compress.

Most of the representations used for our data structures are simple. Since this information is very small compared to the wavelet data, we have not thoroughly explored further compression options, although it is clear that greater compression could be achieved. For example, the transformation coefficients would be good candidates for delta encoding them based on the coefficients of the previous frames.

### 6.3 Parameters

There are many parameters that can be adjusted to control compression. Depending on the application requirements different set of parameters might be chosen. In this section,
we discuss some of the trade-offs in our choice of parameters and alternatives.

This is our current choice of parameters. In many cases, it is possible to search by brute force to determine an optimal parameter-setting.

**Image Size (imagesz)**

The choice of image size has multiple effects in the encoding pipeline. It affects:

- The rasterization step. The more samples the parametrization has, the less likely there are to be faces that are not sampled.

- Wavelet compression. Typically, the larger the image, the more bits it will require. Even with multi-resolution encodings, larger image size will consume more bits unless the wavelet filters match the image resize filters.

- Vertex processing time. A larger image means more vertices to process.

- Spatial scaling possibilities. The image size has to be of a particular size \((2^{\text{levels}}-1 \times c+1)\) and the boundary calculated in the lowest supported resolution for the Geometry Image to easily support spatial scaling or level-of-detail.

As mentioned in Section 6.1.1, We currently hand-pick the image size.

**Choice of Frame Type** The choice of frame types or frame schedule will have a great impact on the total compression of the stream.

Predictive frames allow many frames to be represented at a fraction of the bits of independent frames. If compression was all we wanted, we would use mostly predictive frames. However, increasing the frequency of predicted frames makes it difficult to support operations that require non-sequential data access. For example, if we want to seek to a random frame, we must decode all other frame that it depends on. In the case of many P-frames in a row, it implies decoding all of these. Editing operations are also more complex, as reference frames may have to be updated, and hence all the dependent frames will also be affected. Our representation is targeted mainly for viewing purposes.

Independent frames are important for synchronization and to serve as good reference frames for nearby predicted frames. Synchronization is necessary in networking/lossy (broadcast) environments. If much information is loss, the only hope for the decoder is
to wait for the next independent frame to start with a clean slate again. In these cases, one may use more independent frames than usual to make the stream more robust to errors.

In order to get the most out of the predictive frames, we want our prediction to be as close as possible as our target frame; in other words we want to minimize the residual or error we need to encode. For this, it is important that the reference frames serve as good predictors. P-frames and I-frames can serve as reference frames for both P-frames and B-frames. We mentioned previously that in order to achieve the best compression we want to use mostly predictive frames; at the same time, the distance between the reference frames (or length of runs of predictive frames) should not be too large. If the distance is too large, then the reference frames won't serve as good predictors for our target frames. The distance between the reference frames will, however, depend on how much the animation changes, and influences how well we predict with frames. The problem of choosing optimal I-Frames is analogous to detecting scene changes in traditional video compression.

In addition to choosing a frame schedule, there is also a bit-allocation budget. Predictive frames in general will require less bits to achieve the same reconstruction quality since they start with a prediction of the target frame. Nevertheless, the ratio of bits for encoding predictive frames can vary substantially: very few bits if the transformed reference frame matches exactly the target frame; if a bad or anomalous predictor is formed and there is a large residual error, P-Frames may require even more bits than its equivalent I-Frames. Ideally, we want the bit allocation be dependent on the frame we are encoding. Most applications will prefer a constant quality level. In these cases, frames that are harder to compress will be allocated more bits. This can usually be achieved with a multi-pass encoder. In the first pass, we can compress the animation using default parameters. In subsequent passes, we can refine these parameters to make the quality more similar over time.

Certain low computational power applications have added restrictions. For example, we might have a limitation on the average throughput of the application; the application might only be able to process x bytes per unit of time. This might impose additional constraints on how the bits are allocated between frames. It might be possible that the initial frames are very hard to compress. We would normally allocate many bits to this section of the animation. But if our throughput is limited, we may not be able to do this.

Our encoder currently does not take into account these additional constraints due to
the complexity of the bit-allocation problem. We encode the animation by hand-picking four parameters: the number of P-frames between I-frames \((P_{ratio})\), the number of B-frames between P-frames or I-frames if there are no P-frames \((B_{ratio})\), the ratio of bits between P-frames and I-frames \((P_{bitrate})\), and the ratio of bits between B-frames and I-frames \((B_{bitrate})\). This is a common compromise that is widely used in video compression.

**Wavelets** The choice of wavelet parameters affects the overall compression of the system. We use the Embedded Zerotree Wavelet implementation of Mow-Song [MS02]. The advantage of this specific encoding is that it minimizes the reconstruction error for any given bit-rate. Furthermore, any prefix of the encoding will achieve minimal reconstruction error (for this algorithm) for the given bit quota [Say00, p. 486]. Because of this, we do not have to worry about trimming and quantizing wavelet coefficients.

For the depth or number of levels of the wavelet pyramid, we use the maximum possible. This is useful for both compression and for spatial scaling. If a low resolution version of the mesh is acceptable, we do not have to decompress the whole wavelet tree to obtain the required reconstruction information.

The wavelet function also affects the compression. We have found that the Geometry Images are very smooth. Therefore large support functions are well suited to them. We currently use the "Villa's 18/10 filter" as our wavelet function choice ([TVC96], see appendix A). It is also possible to use brute force to choose our wavelet function. For example, if the residual of a predictive frame is of high-frequency – a mesh having goose-bumps – a smaller support wavelet might be more appropriate. We have not found such cases in our test data sets.

**Transformations** The motion compensating transformations are key to building good predictions. They modify the reference frame to form a close match of the target frame. We use affine transformations because they correspond to the general motion of objects and can be encoded compactly (12 coefficients). We find that Geometry Images are able to extract the spatial coherence that exist in meshes. Geometry Images tend to be very smooth and compress well. To obtain a smooth residual we look for a smooth predictor.

We can use one single transformation to form a predictor or we can subdivide the reference Geometry Image into patches and compute one transformation per patch. Using
one global transformation will be computationally simple for the client or decoder. In general it produces good results if there is not much deformation within the mesh, as it effectively captures the motion of the body in space. For more significant changes in the mesh, local transformations are more appropriate. Local transformation works on patches of the Geometry Image. For simplicity and ease of implementation, we divide the Geometry Image into equally-sized square patches.

There are two parameters we can control when computing and using local transformations: The overlap to compute the local transformation and the overlap to blend the local transformation. We currently use the same value for both parameters and use an overlap that is equal to the width of the patch.

The first parameter, $T_{overlap}$, controls what points the transformation is trying to match between frames. Using points on adjacent patches generates transformations that are more similar to the transformations of surrounding patches. This makes the transformed patches easier to blend. Even though we do not currently compress transform coefficients, using more neighboring points for local transformation will lead to more similar transformations, and hence more compressible data. The tradeoff of using more neighboring points for computing the local transformation is that the transformation will not fit the patch area as tightly, as it will be influenced by the neighboring patches. This tradeoff is small in comparison to the possibilities of patch boundaries not matching or not being close to each other.

The second parameter, $T_{overlap}$, helps to alleviate this problem — it allows the blending of vertex positions on the overlapping portions of the patches. This blending reduces the patch boundaries mismatch that occurs when patch transformations are computed independently of each other. If there is too much blending, the transformed patch will be overly influenced by its neighbors and will not effectively predict well the current patch. In contrast, if there is too little blending, the patch boundary will not be smooth; this can reduce the compression of the residual and possibly leading to artifacts on the reconstructed mesh.

Another factor that influences the choice of global versus local transformations will be the desired bit rate or bit budget. At very small bit rates where there is little bit budget to compress the residual Geometry Image, local transformations will at least, in the least squares sense, more closely approximate the final mesh than one single global transformation.
**Blending**  
Blending tradeoffs are very similar to transformation tradeoffs. We use blending to combine a predictor based on a previous reference frame and a predictor based on a future reference frame when implementing B-Frames. We have few options how to blend the predictors: we can use global or local blending, and we can use dependent or independent blending coefficients.

The first choice is influenced by the same issues as local versus global transformations: the amount of change within the object. One global blend will interpolate between the two predictors using the same interpolating factor or blending for all the areas in the predictors. Unless the interpolation factor of all the areas is similar, local blending will typically provide a better predictor. The real advantage of local blending is being able to capture areas of change from different predictors. For example, if there are two object features – one present on the future reference frame but not on the previous reference frame and another feature that is present on the previous reference frame but not on the future reference frame – then local blending will be able to capture different degrees of change of these features in the target frame. Similarly to transforms, we can also control how much of the neighborhood we use to blend the patches from the two different predictors ($B_{overlap}$).

The second choice we can make is how independent the blending coefficients will be. We currently use the most flexible blending equation with two independent blending coefficients for the predictors ($\alpha$ and $\beta$ in equation 6.3) and a constant ($\gamma$). For global blending the constant is unnecessary due to the fact that a constant in the residual Geometry Image will be trivially absorbed by the wavelet encoder. For local blending this constant factor is important because it will minimize mismatch between the patches. This mismatch is not absorbed by the wavelet encoder easily and will thus hinder compression. The choice whether to make the blending factors $\alpha$ and $\beta$ independent or not will depend more on the animation. For animations without major deformations or scale changes, it is probably unnecessary to make them independent. We currently use the most flexible approach at the expense of extra blending weight coefficients (three vs. two).

### 6.4 Adapting to Network Environment

Throughout this thesis we have borrowed many video encoding techniques to implement a 3D manifold animation encoder. Although we have not extended our mechanism to a
networking and lossy environment, it is clear that many of the same techniques used for adapting video standards to networking environments can be applied here. This is an important advantage of our proposal.

When adapting Geometry Videos to a networking environment, the first step to consider is what kind of information is lossy and what kind of information must be communicated to a decoder in a lossless fashion. If this latter information is large, then it will be hard or inefficient to adapt a scheme to a networking environment. The information structure of Geometry Videos are analogous to the information structure of MPEG. The only difference is the additional sideband information needed in Geometry Videos to fuse the cuts of the Geometry Images. This sideband information is also used to generate the overlap of patches at the boundary of the Geometry Images when local transformations or local blending is used. All other information has a counterpart in standard video encoding: The wavelet coefficients of the Geometry Image or residuals is analogous to DCT coefficients; the transformations and blending coefficients are similar to motion vectors in MPEG.

Besides the actual representation of data for transmission across the network, other changes to the encoding process apply. The most important change is the choice of param-

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>imagesz</td>
<td>The larger the image size the higher the possible reconstruction quality; although this comes at a price of extra processing.</td>
<td>256 x 256</td>
</tr>
<tr>
<td>$P_{ratio}$</td>
<td>Ratio of P-Frames to I-Frames. The more P-Frames the more compressed the stream will be. Comes at the price of robustness.</td>
<td>$\infty$</td>
</tr>
<tr>
<td>$B_{fratio}$</td>
<td>Ratio of B-Frames to P-Frames and/or I-Frames. Similar effect as $P_{fratio}$, except that B-Frames can be skipped as no other frames depend on them.</td>
<td>10 (when used)</td>
</tr>
<tr>
<td>$P_{bitrate}, B_{bitrate}$</td>
<td>Determines size of P-Frames and B-Frames.</td>
<td>0.3</td>
</tr>
<tr>
<td>nb</td>
<td>Number of blocks to divide the Geometry Image.</td>
<td>16</td>
</tr>
<tr>
<td>$T_q$</td>
<td>Quantization of Transform coefficients in bits, can use less.</td>
<td>17</td>
</tr>
<tr>
<td>$T_{overlap}$</td>
<td>How much neighborhood per patch we use to compute transformations.</td>
<td>64</td>
</tr>
<tr>
<td>$T_{overlap}$</td>
<td>How much we blend each patch with its neighbors when using local transformations.</td>
<td>64</td>
</tr>
<tr>
<td>$B_{overlap}$</td>
<td>How much we blend each patch with its neighbors when we blend two reference frames.</td>
<td>64</td>
</tr>
<tr>
<td>$B_q$</td>
<td>Quantization of blending coefficients $(\alpha, \beta, \gamma)$ in bits.</td>
<td>11</td>
</tr>
</tbody>
</table>

Table 6.2: System Parameters
eters. For non-networking environments, frequent use of P-Frames yield good compression results. In lossy networking environments, small losses in frames will be propagated to P-Frames that depend on these degraded reference frames. For that reason, P-Frames are used more conservatively in lossy environments, or more sophisticated error-recovery or error-concealment approaches have to be used. Compression can still be achieved by the use of B-Frames, since no other frames depend on them, they can be skipped or lost without further effects on the video or animation stream.

6.5 Summary

In this chapter we have presented our system for encoding arbitrary 3D manifold animations. We have described the implementation of the system and our design decisions. The system has been tested with a several 3D animations. At this point, we have identified many features of the encoder that would be useful for a considerable range of animations. Like any video encoder, a very sophisticated version of our encoder would do a thorough analysis of the animation in order to make optimal encoding decisions. In the case where analysis is difficult, a brute-force approach could be used. Many of the techniques here are not very computationally intensive, such that time permitting, it would not be prohibitive to do an exhaustive search.

In the next chapter we present our results. We compare Geometry Images with static mesh encoders; We compare our approach with another 3D animation compression method based on principal component analysis; and finally we look in detail at the performance of different components and parameters of our system.
Chapter 7

Results

We have implemented an encoder and decoder for Geometry Videos. In this chapter we compare our implementation with other methods, and we explore the usefulness and validity of many of the proposed techniques. In Section 7.1, we present our methodology for comparing meshes, animation, and compression. In Section 7.2, we compare the Geometry Image method to two other static compression methods to form a baseline. In Section 7.3, we compare the baseline Geometry Video technique to an animation representation based on principal component analysis. At this point we conclude our comparison to other methods and proceed to look at the effects of various parameters and components of our system. In Section 7.4 we consider our choice of cuts and parametrizations. In Section 7.5 we evaluate the performance of predictive coding, in particular we look at transformations and blending. In Section 7.6, we present some discussion on level of detail support for Geometry Images that also applies to Geometry Videos. In Section 7.7, we present time measurements for the encoding and decoding process. Finally, in Section 7.8, we conclude with a summary of our results.

7.1 Comparing Shapes and Compression

Similarly to Khodakovsky et al. [KSS00], we compare two shapes with an $L^2$ norm. We will use this metric to evaluate the reconstruction quality of each frame in animation sequences compressed with our system. The $L^2$ distance $d(X,Y)$ from shape $X$ to shape $Y$ is defined
as the root mean square of the distance of the points in surface $X$ to the surface $Y$:

$$d(X, Y) = \left( \frac{1}{\text{area}(X)} \int_X d(x, Y)^2 \, dx \right)^{1/2} \quad (7.1)$$

Where $d(x, Y)$ is the minimum Euclidean distance from the point $x$ to the surface of the shape $Y$. The distance defined by equation 7.1 is not symmetric, so we use the maximum of $d(X, Y)$ and $d(Y, X)$ for the $L^2$ distance. This $L^2$ distance is usually normalized by the diagonal of the axis-aligned bounding box for the mesh. For animations, we normalize each frame's $L^2$ norm by the bounding box of the first frame; otherwise what may appear as improvements in reconstruction error throughout the animation maybe simply due to large changes in the bounding box diagonal in that frame.

Our $L^2$ shape comparison algorithm is based on the GTS Library [Pop01]; the results are consistent with the commonly used Metro Tool [CRS98] widely used in related literature. Since we are approximating an integral, we use 1/200th the length of the diagonal for our $dx$ (delta step).

The $L^2$ norm is a measure of quality. We are also interested in a measure of compression. For our results, we use bits per vertex (bpv) as our measure of compression. We always consider the number of vertices of the original mesh when computing the bits per vertex (bpv) of a Geometry Image. Vertices are assumed to be 3 coordinates of 32-bit floating point numbers; hence they are 96 bpv uncompressed. Therefore, compressing a mesh at 1 bpv would correspond to a 96x compression over the uncompressed format.

For animation comparison measurements we do not include the cost of connectivity information. For Geometry Images the mesh connectivity overhead is zero since its fully regular and implied by the global parametrization; all we need to know is the dimensions of the image to determine the connectivity. For the original mesh, the connectivity information can range anywhere from 0.1 to 2.0 bits per vertex (or more) depending on the irregularity of the connectivity. More importantly, we are working with animations whose connectivity does not change; therefore, the connectivity information is only transmitted once, and its cost can be amortized over the lifetime of the animation. Therefore, in the limit, the connectivity information per frame would be zero.
7.2 Comparison to Static Approaches

Geometry images are competitive with other static mesh compression approaches. Figure 7-1 shows rate-distortion curves for a few models using two other compression methods: the Touma and Gotsman (TG) algorithm [TG98] and Spectral Compression [KG00]. The TG algorithm achieves compression by quantizing the residuals from a vertex predictor (from 8 to 12 bits, see figure 2-1). Spectral Compression (for overview see end of Section 2.1.1) represents vertex positions as a linear combination of bases. It achieves compression by quantizing the bases’ coefficients and reducing the number of bases used. For our results, we take into account the entropy of the coefficients in computing the total bit-rate. All the measurements were done using Geometry Images of size 256 × 256.

Figure 7-1 show the rate distortion curves for four models. Static compression methods based on quantized residuals or delta encoding usually cannot support very low bit-rates. Also, note that the Geometry Image algorithm works better with a larger number of vertices. With more vertices, we have a larger bit-budget to encode the object. For lower bit-rates it might be useful to use a smaller Geometry Image size. However, with fewer vertices the object will be represented less faithfully. The Turtle model has many more vertices than the Cow model and it is smoother. The Cow model has large triangular faces approximating the surface. The Geometry Images technique is competitive with other static mesh compression methods and forms a reasonable base for developing a 3D animation encoding.

Since the Geometry Image algorithm resamples the input mesh, Geometry Images will never achieve a perfect reconstruction ($L^2$ distance of zero). In our examples, we see that at 10bpv the rate-distortion curves for Geometry Images are nearly flat. At this point the compressed Geometry Image quality is close to the original uncompressed Geometry Image and we are only seeing errors due to re-sampling. In order to improve reconstruction using the same parametrization, a larger image size would be needed. An alternative is also to use a different parametrization. In figure 7-2 we show the types of artifacts generated by using a Geometry Image representation. There are two kinds of artifacts that are inherent in the representation: sampling artifacts and parametrization artifacts. The sampling artifacts occur when extrema points are not sampled, e.g. the tips of fingers. This can be seen in

---

$^1$The implementations for the TG encoder and the Spectral Compression encoder were obtained from Technion - Israel Institute of Technology
Figure 7-1: Rate Distortion Curves: comparing Geometry Images to Touma and Gotsman (TG) encoder, and Spectral Compression

Figure 7-2.f. There are also parametrization artifacts that result from mapping the mesh onto a square. Some faces may be mapped to long skinny faces in the parametrization. Figure 7-2.k shows the results of such parametrizations on the arm of the dancer; this artifact results from having a high number of skinny faces where there were few faces in the original mesh. There are also artifacts associated with compression, namely boundary separation and smoothing. After compression, adjacent boundary edges of the reconstructed mesh may separate to form a visible seam. This seam is eliminated by using stitching sideband information to seal the seam. This seam can be small even at high compression. Figure 7-2.1 shows the size of the seam for a Geometry Image encoded at 4bpv. In all of our examples, we use stitching information to seal the seam by simply averaging complementary vertices on the border (see Section 6.2.) The other kind of artifact resulting from compression is smoothing of features; this can be seen on the face of the dancer in Figure 7-2.1.

Geometry Images outperform the TG algorithm and Spectral compression algorithms in
compressing static meshes. There are some artifacts associated with Geometry Images; for most cases we find these artifacts acceptable, in other cases these artifacts can be addressed with different parametrizations or larger image sizes. In the next section we compare how our Geometry Videos approach compares to an animation compression technique.
Figure 7-2: Comparison of Different Artifacts in Geometry Images: (a) Frame 1 of Dance sequence. The relative $L^2$ reconstruction error is given underneath and is in multiples of $10^{-4}$ the size of the bounding box diagonal. (b) Reconstruction from uncompressed Geometry Image. (c) Reconstruction from Geometry Image compressed at 4 bits per vertex; compression causes smoothing of features. (d) Reconstruction from Touma and Gotsman (TG) Algorithm using 8 bit quantization. (e) Original hand close-up. (f) Resampling artifact of Geometry Image, if unacceptable, either a different parametrization or a large Geometry Image needs to be used. (g) Hand from Geometry Image at 4bpv. (h) Hand from TG representation at 8bit quantization. (i) Cut used to generate parametrization. (j) Parametrization used to generate Geometry Images. (k) Parametrization artifact caused by large skinny triangles in parametrization. (l) Reconstruction of Geometry Image at 4bpv without stitching the boundary (compare to (c)); the unseamed seam would be small even at low bitrates.
7.3 Compression Results

In this section we compare the baseline Geometry Video approach (no predictive coding) to an animation encoding scheme proposed by Alexa: "Representing Animations by Principal Components" [AM00]. We shall refer to this method as PCA from here on. We use three data sets for comparing this method to our approach and later for evaluating different parameters of our approach:

- **Cowheavy.** This is a sequence of frames depicting a cow mesh being tossed around in space. The cow is subject to extreme deformations. The mesh has 2904 vertices in each frame.

- **Dance.** This is a swing dance sequence based on skinning a skeleton animated with motion capture data. The mesh of the dancer has 7061 vertices.

- **Snake.** This is a sequence of a snake also being tossed around in space; the snake coils and uncoils itself. It can be viewed as major body deformations. If this were to be simulated with a skeleton, it would need many joints as the bending occurs throughout the length of the snake. The snake mesh has 9179 vertices.

Figures 7-3, 7-4, 7-5 show every 10th frame of the Cowheavy, Dance and Snake sequences respectively.

We reimplemented a version of Alexa’s PCA method in Matlab 6.5 for our comparison. PCA uses principal component analysis to compute a basis from which the vertex locations at any frame can be computed. First it aligns all the frames to a reference frame (the first frame) by computing the affine transformation. This transformation will try to match the transformed frame to the reference frame in the least square’s sense. The reference frame along with all the transformed frames are combined into a large $3M \times N$ matrix, where $M$ is the number of vertices (3 coordinates per vertex) and $N$ is the number of frames. This matrix is decomposed into its principal components. Each column of this matrix can be approximated by some linear combination of the principal components or basis. Compression is achieved by using only the first $n$ basis vectors.

Figure 7-6 shows the results of our comparison. It is not perfect comparison for several reasons. It is hard, without being unfair, to quantify the exact bitrate for the PCA algorithm. The compressed size will be determined by the number of selected bases (each being
three 32-bit floating point values, or 96 bits per vertex), the basis coefficients (number of bases times number of frames), the inverse transformations for each frame, and the number of vertices. This information can also be further compressed. For example, figure 7-7 show what the first 3 bases for each animation look like. It is clear that these bases should be compressible using standard mesh encoders (e.g. the Touma and Gotsman encoder [TG98]), although it is not clear what the effect of the errors and/or quantization of the compressed basis might be.

The bit-rate for PCA will be determined mainly by the ratio of number of basis vectors used to the total number of encoded frames. For uncompressed basis vectors, if this ratio is one, then the animation will be encoded at approximately 96bpv (three 32-bit floating point numbers per vertex). For our comparison, we chose a moderate number of bases. For example, for the dance sequence of 181 frames, using PCA with 16 bases implies a base encoding of approximately 8bpv (16 basis vectors/181 frames * 96bpv \approx 8bpv).

Under this scenario, Geometry Images outperforms PCA when compared using the $L^2$ distance error. This error metric is misleading, because for most frames, the quality of the PCA encoding is better than the quality given by Geometry Images (as can be seen on the right frames of Figure 7-6). PCA has a hard time matching the exact motion of the object, so, for the most part, the error is caused by misalignment.

Nevertheless, there are other sources of more critical errors. For example, in the dance sequence when the dancer swings his arm around his back, there are major artifacts in the PCA encoding. This occurs when there are insufficient number of basis vectors to encode the motion. In this case, this even occurs when 32 bases are used for the encoding. Principal component methods have a hard time expressing non-linear deformations\(^2\). More importantly, there are no methods to correct for errors as there is no explicit residual to encode. This limits the quality of animations it can encode. Additionally, because it relies on having all the mesh information over the length of the animation to build the basis vectors, it is hard to use in an online environment. Geometry Images, on the other hand suffer from smoothing and loss of details at low bit-rates, but do very well in capturing the motion and encoding any residuals.

\(^2\)As suggested by Lengyel [Len99]
Frames 1, 11, ..., 81

Figure 7-3: Sample Original Frames from Cowheavy Sequence
Frames 1, 11, 21, ..., 61

Frames 71, 81, ..., 111

Frames 121, 131, ..., 181

Figure 7-4: Sample Frames Original from Dance Sequence
Figure 7-5: Sample Original Frames from Snake Sequence
Figure 7-6: PCA and Geometry Images Comparison: We compare Alexa’s [AM00] to Geometry Images. Objects on the right shows the reconstruction using Geometry Images, the original mesh, and the reconstruction using PCA. The reconstructed meshes are color-coded (not all by the same factor) to show the relative magnitudes of the error from the original mesh. Note the scaling distortion in the PCA reconstruction in (d). The basis meshes for the PCA encoding are uncompressed.
Figure 7-7: Examples of PCA basis: First three basis of each PCA decomposition of each sequence. These basis are compressible.
7.4 Cuts/Parametrization

In this section we explore the effect of using a global cut and a single parametrization for generating the sequence of Geometry Images to be compressed. In order to exploit temporal coherence that exists in the animations, we would like to apply the same cut and parametrization on all frames in order to have a correspondence between all frames in the sequence of Geometry Images. This correspondence allows us to construct predictors and compute residuals to compactly represent the animation. Alternatively, we could apply a cut and a parametrization specific to each frame and encode each frame independently, but this approach has substantial drawbacks for data compression as discussed in Section 4.1.

First, we build an intuition of how the cutting algorithm and the parametrization work by looking at a simple example. We look at a simple sequence of a bouncing ball, generated using keyframe animation. There are only two different kinds of keyframes: a round ball and a compressed ball (frame 71). The ball mesh has 5552 vertices. Figure 7-8 shows every 10th frame of the sequence.

Figure 7-9 shows the different cuts of frames in the ball sequence as chosen by our automated routine designed to minimize the distortion of the parametrized mesh. Figure 7-9.a shows the cut from Frame 1; it is generated by first making a cut from one pole of the ball to the other. An additional cut is made from the equator to one of the poles. Figure 7-9.b shows the cut for Frame 71, the compressed ball. In this case there is more distortion at the equator than at the poles. The first cut starts at one point on the equator and continues over the pole to the opposite side of the equator. The two additional cuts go from other points on the equator to the pole. The global cut for this sequence is practically identical to the cut from frame 1 since 90 out of the 100 frames are identical to frame 1.

For this particular example, the choice of frame for the parametrization has a large impact on the reconstruction. Figure 7-10.a shows the parametrization resulting from applying the global cut to frame 1. This yields a very good reconstruction as can be seen in Figure 7-10.d. The parametrization resulting from applying the global cut to frame 71 is shown in Figure 7-10.b and the reconstruction is shown in 7-10.e. This frame is a poor choice for a parametrization due to the large number of compressed faces near the equator; their location in the parametrization is evident and will yield poor sampling in this area. Finally we apply the cut from frame 71 to frame 1 and look at the resulting parametrization in...
Figure 7-10.c. The reconstruction quality shown in 7-10.f is only slightly worse than using the global cut (Figure 7-10.d). In this example the parametrization has a large effect due the fact that there are no major features in the model and the two possible frames that can be used for the parametrization vary greatly.

Frames 1, 11, ..., 91

Figure 7-8: Sample Original Frames from Ball Sequence

Figure 7-9: Cuts of Ball Sequence: (a) Cut of Frame 1 of Ball sequence. A first cut is made from one pole to the other; a subsequent cut is made from the second pole to a point on the equator (b) Cut of Frame 71. Since this sphere is flattened there is more distortion at the equator than at the poles in the initial parametrization, the first cut goes from one side of the equator to the other; the next two cuts go from other locations on the equator to the pole. (c) The Global Cut which considers all the frame is very similar to the cut of frame 1 as 90 of the 100 frames in the sequence look like frame 1.

Our next experiment looks at the effect of choosing a cut and parametrization for different frames in the animations. Figure 7-11 shows the reconstruction error for the Cow, Dance and Snake sequences using both a particular cut and parametrization for each frame.
versus using a single global cut and parametrization based on the first frame for all frames. For the Cow and Snake sequence, the global cut performed similar to choosing a cut and parametrization for each frame. For certain frames the global cut performed better than a customized cut based on that frame – this might seem counter intuitive. Recall that the cutting algorithm is a greedy algorithm; it iteratively identifies a triangle with the worst distortion in the parametrization and makes a cut from a vertex in that triangle to the current boundary. It is possible for this algorithm to be stuck in a local minimum. But even in these cases the difference in reconstruction error is small.

The Dance example, on the other hand, has other unique parametrization properties. First notice that the global cut performs about the same as or slightly worse than using a cut and parametrization from a specific frame as indicated in Figure 7-11.c. There are two frames that stand out in this experiment: frame 116 has the worst distortion and frame 131 has the best reconstruction by a wide margin when using the specific cut for each frame. To understand the difference in quality, one can look at the parametrizations for each frame. The parametrization using the cut from frame 116 is show in Figure 7-11.e. The fingers (located on the top middle part of the parametrization) and the top of the head (located in the upper left corner) are distorted. This is not the case for the other parametrizations (d,f) based on a global cut and on a cut from frame 131.

To see what factors are responsible for the variations in reconstruction quality due to our choice of cuts, we applied the cut from frame 131 to frame 116 and used frame 116 for its parametrization. Similarly, we applied the cut from frame 116 to frame 131 and used frame 131 for its parametrization. The results were that the best reconstruction was achieved with the cut from frame 131. Furthermore, we achieved consistently better results using the cut from frame 131 and the parametrization from frame 1, as illustrated in the Figure 7-11.c. Both fixed cut plots in this figure are fairly flat; this is due to the fact that there is not much body deformation in the sequence. The other sequences (Snake and Cow) have larger variance.

Although the choice of global cut was suboptimal in the dance sequence, using a brute-force approach to discover the best cut among all the frames would be very expensive. To do this, one would have to compute the cut for each frame, compute a parametrization, and compute the error for all frames. Using only one frame to judge the quality of the cut would not be sufficient as there is typically much more variance in the reconstruction quality than
that found in the Dance sequence. Furthermore, the global cut tends to have less variance because it can take all frames into account and make cuts into regions of high distortion that occur in different frames.

Next, we look at our choice of parametrization. Our hypothesis is that the cut has a major influence on the parametrization. The input to the parametrization is a cut mesh with a mapping of the cut boundary to the parametrization boundary. The parametrization attempts to optimize the location of the internal vertices to minimize the average geometric-stretch. The cuts will presumably have reached regions of high-stretch bisecting them to reduce this stretch. Our experiment consists of looking at the reconstruction error by fixing the cut and computing the parametrization using different frames.

Figure 7-12 shows the result from this experiment. The vertical shows the $L^2$ distances (reconstruction error), normalized by the diagonal of the first frame. On the x-axis we have the choice of frame used for the parametrization. The reconstruction quality for each frame across different parametrizations (y-axis) is similar; this supports our hypothesis.

The parametrization from frame 141 in the Dance sequence stands out as a particularly bad parametrization. Figure 7-13 shows the parametrization for that specific example and the resulting artifacts. The red square shows how the fingers on the left hand are badly parametrized. The green oval shows the parametrization of the fingers of right hand which is better. Figure 7-13 (c) and (d) show the reconstruction of the two hands. In our experience such cases occur infrequently. In order to avoid these anomalies it is possible to try a few different parametrizations from various frames.

In general we have found that the following approach gives acceptable results: compute a global cut based on all frames and pick an arbitrary frame as a reference for the parametrization. One might improve upon our results by perhaps trying a few different parametrizations or computing a parametrization based on all frames.
Figure 7-10: (a) Global Cut applied to Frame 1 and resulting parametrization. (b) Global Cut applied to Frame 71 and resulting parametrization. (c) Cut from Frame 71 applied to Frame 1 and the resulting parametrization. (d) Geometry Image from (a) applied to frame 1 and reconstruction compressed Geometry Image. The reconstructions are color coded by with the color from the Geometry Image. The compressed Geometry Image size is given in bytes and the relative $L^2$ reconstruction error is given in multiples of $10^{-4}$ the size of the diagonal bounding box. (e) Geometry Image from using the cut from frame 71 for parametrization (b), applied to frame 1 and reconstruction compressed Geometry Image. Using the parametrization from frame 1 yields better results because it does not have skinny faces near the equator, in contrast to frame 71. (f) Geometry Image from (c) applied to frame 1. In the ball example, the parametrization has a greater effect than the cut because there are no significant features and the reference frames vary greatly (there are a large number of skinny faces near the equator in frame 71).
Figure 7-11: (a,b,c) Cow, Snake, and Dance sequences encoded at 8 bits per vertex (2904, 7061, and 9179 bytes per frame respectively) using a per frame cut and parametrization vs. using a global cut based on all frames and a parametrization based on the first frame. (d) shows the parametrization of frame 1 using the global cut of the dance sequence. (e,f) show the cut/parametrization of frame 116 and 131 of the dance sequence; these frames correspond to the highest and lowest distortion.
Figure 7-12: The choice of cut has a significant effect on the parametrization. This figure shows the L2 distance of each sequence cut using their global cut, and parametrized using different reference frame. Notice that the error for each frame is about the same for all parametrizations (except the dance sequence using frame 141 for the parametrization, see Figure 7-13 for more detail).
Figure 7-13: Examples of a poor parametrization from global cut choice from the Dance data set. (a) shows the parametrization for frame 141 of the Dance sequence using the global cut. (b) shows a zoomed window of the parametrization of the fingers of the left hand which are compressed near the top (the parametrization of the fingers of the right hand is circled and not compressed). (c) shows the reconstruction of the distorted hand. (d) shows the reconstruction of fingers of the right hand which have a better parametrization (better sampling). (e) shows the original left hand. (f) shows original right hand.
7.5 Predictive Coding

In this section we examine the advantages and impact of predictive coding. Specifically, we find that there is significant temporal coherence in an animation and that we can halve the reconstruction error for the same bit-rate using predictive coding.

First we look at our simple bouncing ball sequence. Figure 7-14 shows the reconstruction errors of the bouncing ball sequence encoded using only I-Frames versus using P-Frames with global transformations at various bit rates. For the predictive coded sequences, the first frame is encoded as an I-Frame and the remaining frames as P-Frames. The size of the P-Frames is 30% the size of the I-Frame. For the I-Frame only sequence, the reconstruction error is the same for frames 61-68 as the only change of the ball is translation. For frames 69-71 the ball is compressing as it hits the floor. The reconstruction error is smaller because the majority of the surface has less curvature. For frames 72-78, the ball is decompressing and the reconstruction error returns to that of the first frame.

The translation and scaling is captured by the predictor function in the predictive encoded sequences since the ball is only undergoing rigid-body deformations. As expected, the deltas are used to improve the reconstruction quality of the ball (rather than compensating for any deformations not captured by the predictor function). The reconstruction error for the predictive encoding decreases with each frame. In this example, a sequence encoded using P-Frames with 1388 bytes per frame has the better reconstruction error than the same sequence encoded using only I-Frames at 5552 bytes per frame.

Figure 7-15 shows more complex sequences ("Cowheavy", "Dance" and "Snake") encoded at different bit-rates. The base sequence is one encoded using only I-Frames (i.e. not using any temporal information) at 8 bpv. We compare that to the same sequence encoded using P-Frames with local-block transformations encoded at 2, 4, and 8 bpv. P-Frames with local-block transformations yield the best results. As a reference, we also show the same sequences encoded using B-Frames with local transformations and blending at 8bpv. B-Frames provide additional flexibility for controlling frame-rate, because they can be discarded by the decoder with only minimal effects. Unlike P-Frames which may depend on other P-Frames, no frames are dependent on B-Frames. The size of each P-Frame or B-Frame is 30% the size of each I-Frame. For the predictive coded sequences we place one I-Frame every 19 P-Frames or B-Frames; this allows us to see the major benefits of
Figure 7-14: (a) Ball sequences encoded using I-frames only and P-frames at various rates. The ball translates for most frames; In frames 68-71 the ball is squished at it compresses when hitting the floor; In frames 71-78 the ball decompresses. As expected, since there are no deformations of the ball in frames 60-68, the quality of each frame improves in the predictive case as the translation is captured by the global transformation and the deltas are used to improve the quality. (b,c,d) shows the reconstruction of frame 60, 68, and 71 from using predictive coding at 694 bytes per frame (1bpv).

predictive coding while at the same time resetting the encoder regularly if the current prediction is not effective. All the sequences were encoded with an image size of 256 x 256. Local transformations and blending were computed with a block size of 64 x 64 and overlap distance of 64; this implies that there are 16 patches that are transformed separately.

We find that using predictive coding allows us to encode the sequences at half the bit-rate (4bpv) and still get better reconstruction error than the same sequence encoded using only I-frames at 8bpv. We now describe a few interesting sequences.

The predictive coder when applied to the Cow sequence, exhibits problems on frame 66. At this point in the sequence, the cow is deflating instead of being tossed around. The patches for the local transformation are big relative to these changes (or conversely, there are major changes within the patches), so the local transformations have a hard time representing this deformation.
The Dance sequence shows fairly stable reconstruction error as shown in Figure 7-15.b. This sequence of a dancer has little deformation; it has a limited number of joints. We note that B-Frame encoding oscillates with a period of 20 frames. P-Frames depend on the previous frame; B-Frames on the other hand, depend on the previous and next I-Frame or P-Frame. In these examples, the reference frames are in location 1, 21, 41, etc. The quality of the predictor for B-Frames will vary with the distance to its reference frame; thus the B-Frames located in the center between the reference frames will typically have the lowest quality reconstruction. Compare frame 2 of the B-Frame encoded sequence to frame 11.

The Snake sequence exhibit the most variation in reconstruction quality. Unlike the Dance sequence, which has a fixed skeleton, the snake bends at all locations of its body; this can be thought of as an skeleton with an infinite number of joints. Such deformations are difficult to model with affine transformations. Also note that we are using 16 patches for the local transformations. Increasing the number of patches could help with compression by allowing finer granularity transformation of the reference frame; but this would come at the expense of encoding more transformation coefficients.

In the next two subsections we look at transformations and blending.

7.5.1 Predictive Coding: P-Frames and Transforms

In this section we explore the advantage of local versus global transformations. We also look at the effects of quantizing the transform coefficients.

Figure 7-16 show the sequences encoded with different P-Frames parameters. The baseline, is the animation encoded using only I-Frames. We show an encoding using exactly the previous frame, we call this “P-Frame Identity;” this gives an idea of how much motion and or deformation exists between frames. We compare two encodings: P-Frames using Global transformations vs. P-Frames using Local transformations. We find that P-Frames with Local transformations generally outperform other schemes. However, if the changes are small, using a global transformation or even the identity transformation can yield similar results with less computational overhead.

The Cow sequence (Figure 7-16.a) shows very little difference between different P-Frame parameters. The changes between frames are small enough that there is little difference in the residuals using different P-Frame parameters. We also note that the identity transformations performs very well. For the majority of the Cow sequence there is one dominant
motion, which is the global motion of the cow being tossed around. Frames 60-80 are the exception where the cow is lying on the floor and deforming; this is purely a local deformation.

The Dance sequence (Figure 7-16.b) has more temporal information as can be seen by the reconstruction quality of the Identity transformation. There is little difference between the global and local transformations because the major motion is the rotation of the whole dancer. Although there is independent motion in the arms and the legs, this motion is small enough to be captured by the residual.

The Snake sequence (Figure 7-16.c) has similar results except for frames 81-100. Here the snake exhibits a twisting motion from the middle of the body at it stretches.

We next explore the effect of quantizing the transform coefficients. We look at both global and local transformations.

Figure 7-17 shows the sequences encoded using global P-Frames. Each sequence is shown compressed with 11 bit and 17 bit quantized transformation coefficients. For all the sequences, the difference in reconstruction quality is negligible. We do note that there is a difference between the predictors generated using 17-bit coefficients (better) versus 11-bit coefficients (worse), but this difference is absorbed by the residual, which still compresses well.

Our initial hypothesis was that heavy quantization (11 bits) of transformation coefficients with local transformations would be detrimental to compression. The argument was that the predictor would have boundary artifacts derived from transforming each patch of the reference frame with quantized coefficients. We found that is not the case. Figure 7-18 shows the animation sequences encoded using Local P-Frames using 11-bits and 17-bits for the transform coefficients. The difference is minimal, we had expected a larger difference. The reason for the small effect of using 11-bit coefficients is twofold: First, the use of the neighborhoods for computing the transformation for the patches makes the initial unquantized transformations for the patches more similar to each other. Secondly, the blending of neighborhoods around the patches allows for a smooth transition around patch boundaries, even if these don’t match as well.(e.g. the greater misalignment being the result of quantizing the transformations).
7.5.2 Predictive Coding: B-Frames and Blending

In this section we look at B-Frames, in particular we look at the choice of global blending versus local blending as described in section 5.4. Figure 7-19 show the sequences encoded using both local and global blending. Global blending in general has only slightly worse reconstruction than local blending. Local blending is advantageous for sequences where there are major deformations, not just motion. The snake sequence (Figure 7-19.c) exhibits such behavior. Frames 61 and 81 form the reference frames for all the B-Frames in between. Local transformations/blending performs better for such frames than global transformations/blending. Figure 7-19.d shows the reference frames and the target frame. Due to how the body of the snake is bent, it is very hard with a single affine transformation to try to match the reference frames to the target frame. Figure 7-19.e shows what the predictor (green) for the target frame (red) looks like using a global transformation and blending. Similarly, Figure 7-19.f shows what the predictor (green) for the target frame (red) looks like using a global transformation and blending; notice that local transformations were able to match the tail of the predictor to the target frame.
Figure 7-15: Cow, Dance, and Snake Sequences using Local P-Frames and Local B-Frames encoded at different bit-rates. Note that the quality is better at half the bit-rate, and that the quality is twice better for the same bit-rate.
Figure 7-16: Cow, Dance, and Snake Sequences encoded at 8 bits per vertex (2904, 7061, and 9179 bytes per frame respectively) using different P-Frame options. Sequences encoded using Local Transformations always do better. The objects in the right show some of the original frames in the sequences to illustrate the difficulty in capturing certain motion and/or deformations.
Figure 7-17: Quantization of Global Transformation Coefficients: Cow, Dance, and Snake Sequences encoded at 8 bits per vertex using Global P-Frames with different quantization for the coefficients.
Figure 7-18: Quantization of Local Transformation Coefficients: Cow, Dance, and Snake Sequences encoded at 8 bits per vertex using Local P-Frames with different quantization for the transformation coefficients. Due to the blending of neighbors of each patch, and the use of the neighbors to compute the transformation for each patch, there is little effect to the reconstruction quality due to the quantization of the transformation coefficients.
Figure 7-19: Cow (a), Dance (b), and Snake Sequences (c) encoded at 8 bits per vertex using different B-Frame options. There is not much difference between global and local except for few frames in the snake sequence (c).
7.6 Level of Detail

Level of detail is naturally supported by Geometry Images and hence by Geometry Videos. It is simple to construct a coarse version of the mesh by simply decimating (removing every other pixel in each direction) the Geometry Image. Figure 7-20 shows the first frame from each of the sequences reconstructed from their original Geometry Image of $257 \times 257$. This size corresponds to upsampling a Geometry Image of size $65 \times 65$ twice. The rationale is that if cut-nodes lie on a grid location on the boundary of the $65 \times 65$ Geometry Image, then it will also lie in a grid location on the boundary of the same parametrization sampled at $129 \times 129$. Note that images of size $65 \times 65$ will have 64 edges per side; images of size $129 \times 129$ ($2x - 1$) have 128 edges per side; each edge in the lower resolution image gets mapped to two edges in the higher resolution image. The next frame shows the result from decimating the Geometry Image used to compute the first frame (i.e. on with size $129 \times 129$). The next frame is the result of applying this same process to the previous frame. As expected, the coarse meshes are less detailed than the original. Note the ear of the Cow and head of the Snake. The facial features of the dancer disappear in the coarse mesh.

We currently use a parametrization that tries to uniformly sample the input mesh. This is important for level-of-detail support in the context of texture mapping. Texture coordinates are computed implicitly with Geometry Images – the texture coordinate correspond to the (u,v) position of the “pixel” in the Geometry Image. The points in the decimated (coarser) Geometry Image will have the right texture coordinate, but the span between texture coordinates will be larger, and hence larger patches of the texture will be copied. The texture map will be more accurate if the finer sampling is uniform; otherwise the texture will seem to shift when there are no intermediate texture coordinate to compensate for the non-linearity.

The more precise definition of Level-of-detail support implies that regions of interest can have greater detail than other non-interesting regions. This is supported by combining different resolution versions of the same Geometry Image. Figure 7-21 shows a simple example. The upper right quadrant of the Geometry Image corresponds mostly to the head of the cow. We can combine a fine version of this quadrant with a coarse version of the Geometry Image to obtain a Cow that has more detail on the head. The example shown was constructed by hand. Additional care must be taken at the transition areas from fine
to coarse detail, as the connectivity has to be adjusted. This is a simple operation due to the grid-like connectivity of Geometry Images.

### 7.7 Timings

In this section we briefly report on the performance of our system. Table 7.1 reports the performance of different algorithms.

The encoder was developed in Matlab 6.5 for ease of prototyping. It has not yet been ported to C++. The first four measurements concern the encoder, in particular the parametrization pre-process. Once a cut has been chosen and a parametrization computed, we can encode 3D animations at a rate of few frames per second for an image size of 256 × 256, and much faster with a smaller image size. The encoding time is dominated by two factors: the wavelet encoding of the Geometry Images and the cutting and sampling of the frames in the 3D animation. For example, the encoder will encode 100 frames of the Cow Sequence with an image size of 64 × 64 in 6.24 seconds: 1.69 seconds are spent cutting the meshes of each frame and preparing them to be sampled, 1.90 seconds are spent sampling the input meshes; and finally 1.66 seconds will be spent encoding the resulting Geometry Images for each frame. The wavelet encoder/decoder is a research prototype and its performance can be improved. For example, the wavelet basis is not hard-coded and it can support many different wavelet basis functions. The cutting algorithm is implemented in Matlab; and the sampling (rasterization) could be done in hardware.

We can decode Geometry Videos at interactive rates for small image sizes (64). The decoding speed is dominated by the wavelet decoder. The beauty of Geometry Videos is that the performance for a Geometry Video of size 64 × 64 and will be the same (assuming same file size) regardless of the actual original mesh that was encoded.

### 7.8 Summary

In this chapter we have evaluated the design decisions, the choice of parameters and the performance of our system. We compared Geometry Images to static mesh compression

---

3 Windows XP, Dell Dimension 4550, Pentium 4 at 3.06 GHz, 533MHz front side bus, 1 GB DDR SDRAM

4 Linux, Pentium 4 at 2.4 GHz, 533MHz front side bus, 4 GB DDR SDRAM
Table 7.1: Geometry Video subsystems timings

schemes and compared Geometry Videos to an animation representation based on principal component analysis. Figure 7-1 show that Geometry Images are competitive to other static mesh compression methods and Figure 7-6 show that Geometry Videos perform better than PCA for encoding animations.

We evaluated our choice of cut and parametrization; although we have shown that our choice is not always optimal, in general it produces acceptable results. Figure 7-2 show the resulting artifacts that we can expect from this encoding. Figure 7-10 shows that sometimes the choice of parametrization makes a difference in the reconstruction quality; but for our more complex sequences Figure 7-12 shows that the choice of parametrization may not make such a difference.

We also evaluated at the usefulness of predictive coding. For our animation sequences using Local P-Frames reduces our reconstruction error by half for the given bit-rate. Figure 7-15 demonstrates this result for each of the sequences. We explored the utility of Local P-Frames versus Global P-Frames. We found that Global P-Frames work fairly well. In cases where there is a lot of deformation within the mesh then Local P-Frames will work better. Figure 7-16.c shows how this is the case for the Snake sequence.

We looked at the quantization effects of the transformation coefficients. Figure 7-18 and 7-17 show our results which were surprising in that 11-bit quantized transformations did only slightly worse than 17-bit quantized transformations.

We also compared local and global B-Frames. Figure 7-19 show that the reconstruction using Global and Local B-Frames only makes a different in a few frames of the Snake sequence. Again, we found that unless there are major deformations, global transformations
perform nearly as well as local transformations.

Many of the predictor building techniques we have presented here can be applied to other mesh compression mechanisms. The advantage of our approach is that because of the uniform sampling and being able to take advantage of the spatial coherence in orthogonal directions, residuals compress well.

This approach also makes describing a partitioning of the mesh simpler because vertices adjacent on the geometry image will be adjacent in the reconstructed mesh. In our approach we only need the overlap size and the boundary map to describe the partitioning.

Geometry Videos also support level-of-detail modelling. Although we have not utilized this feature, it is simple to implement. We showed a few examples of how it works. Finally, we reported various time measurements for different stages and steps of the system. Although a Geometry Video can be generated quickly (i.e. few minutes for computing a single cut and parametrization), generating a global cut can be more time consuming. Much improvement in speed can be achieved by implementing parts of the encoding subsystems in C++ instead of Matlab.

Our results demonstrate a system that is able to compress animation sequences well. The most important aspect is that it uses many of the same techniques already used in mature video encoding standards. This method can quickly leverage this knowledge to bring the representation of 3D animations to the current video performance standards.
Figure 7-20: Level-of-Detail Frames: First frame of each sequence rendered from a Geometry Image of size 257 x 257, 129 x 129, and 65 x 65. To compute a Geometry Image that can support multiple levels of detail, the boundary mapping has to be computed at the lowest intended resolution, in this case at a size of 65 x 65.
Figure 7-21: Level-of-Detail: by decimating the geometry image (a) we obtain a lower resolution version (b); We can combine the two to support a multi-resolution version of the mesh (c); care must be taken at the transition edges, this is a simple operation due to the regularity of the mesh.
Chapter 8

Conclusions and Future Work

We have introduced a new animated mesh representation, called Geometry Video. We have described an algorithm to create geometry videos from animated meshes. Our algorithm is based on the Geometry Image construction algorithm from Gu et al [GGH02]. By extending the algorithm to take advantage of frame-to-frame coherence we demonstrated the similarity of our approach to that of standard 2D video encoding and we showed how we can apply many of the video compression techniques to 3D animations.

8.1 Conclusions

This thesis began with a review of static mesh compression techniques and a review of methods to encode time-varying geometries. We found that Geometry Images are a good mesh representation. They have the advantage that they do not need explicit connectivity information (they have grid connectivity); they have a global parametrization that can be used for other attributes besides geometry, like textures, normals and other surface-related parameters; and they are smoothly varying and thus easily compressed.

Geometry Images form the foundation of our animation compression approach. The following enhancements to Geometry Images were developed to adapt them for representing animation sequences. In order to take advantage of the temporal coherence between frames it was concluded that a single cut and parametrization is needed to be used throughout the animation. A global cutting algorithm was presented to find a good cut for the animation. Unlike static mesh compression methods like Geometry Images, the global cut considers all frames in the animation. The global cut has the potential of reaching areas that would have
high-distortion than if the cut was based on only a single frame.

A parametrization maps the points on the surface of a manifold onto the unit square. This mapping is used after the mesh has been cut. We adopted the geometric-stretch metric as our measure of distortion. We motivate why this is a good metric to minimize in a parametrization for Geometry Videos. Ideally we would like global parametrization. We currently use the first frame of the animation. Our results show that this yields satisfactory results except in few rare occasions.

Once a cut and a parametrization is found, we can generate a sequence of Geometry Images by sampling the parametrization over a grid. This sequence of generated Geometry Images will have correspondence between frames, so it will be suitable for temporal compression.

We then presented methods for compressing such animation sequences. We described how predictive encoding, similar to that used in video compression, can be applied to compress this sequence of Geometry Images. Of importance, is the fact that these images correspond to geometries, and hence using the same techniques as video would be inefficient. We presented local and global transformation techniques to build good predictors. We use an affine transformation for our prediction function. These transformations do a good job of predicting rigid body transformations and approximating simple deformations in the animations. We also showed how blending between frames can also be used as a predictor. Frames in the animation are encoded as a transformation of the previous frame plus a residual. We find that the wavelets used to encode and compress Geometry Images work very well for encoding the residuals in Geometry Videos.

In order to validate our approach, we built a complete system to encode and decode 3D manifold animations as Geometry Videos. This system has different parameters to control the quality of the encoded animation. We discussed the intuition behind different parameter settings and heuristics on how to apply them to various kinds of animations.

The base of our proposed method, Geometry Images, compares favorably to static mesh compression techniques. Our method for compressing animations can halve the reconstruction error of a 3D animation in comparison versus encoding each frame individually without taking advantage of temporal coherence.

Overall, our scheme is very similar to that used in standard video encoders. The key advantage of this approach is to leverage all of the accumulated knowledge on 2D Video
encoding to allow 3D animation to become another natural media.

8.2 Future Work

In this section, we conclude this thesis by looking at possible extensions to Geometry Videos and new applications of this technique.

**Improved Video Compression Techniques**  The first extension or topic to explore is the application of other video techniques or optimization. For example there is a large amount of research in scheduling (choosing the frame type for each frame) and bit allocation (how many bits to allocate to each frame). These techniques could be easily ported to Geometry Videos. Perhaps a better understanding of the effects of compression on reconstruction might be necessary for its application.

There are many techniques for robust transmission and streaming of video over networks. This is a young field in the realm of geometry encoding. The application of these techniques to Geometry Videos, would allow the fast deployment of streaming 3D animations.

**Improved Transformations**  Our inter-frame predictors are based on affine transformations because an optimal least-squares fit can be computed in a closed form. Other transformations may be better suited for particular animations, for example free form deformations (see Section 2.2.1). Having a wide variety of transformations to choose from can lead to better inter-frame predictors. The trade-off is the computation required to choose them and the complexity required to apply them.

Compression of local transformations is another area for further research. One possible way to compress them is to use hierarchical transformations: For example, use one global transformation as an initial predictor, and then use local transformations to further refine the predictor. Alternatively, Alexa [Ale02] proposes a transformation space where transformations can easily be manipulated and interpolated. This space may be better suited for compressing transformations.

**Chartification**  In order to apply local transformations we divided the Geometry Image into a number of equal-sized blocks. In essence we subdivide the object in order to transform each part separately. It may be better to divide the object into more meaningful patches.
For example, Zuckerberger et al. [ZTS02] proposes a decomposition algorithm that identifies meaningful parts of the object (e.g. limbs on a human mesh). Lengyel [Len99] also proposes a decomposition of the mesh for the purpose of compression; his approach uses a greedy algorithm to cluster faces depending on their time-trajectory. One advantage of using Geometry Images, is that a continuous surface patch in object space can be simply described as a bounded patch in the image space; in the classical mesh representation, a patch is usually represented as a set of vertex indices or face indices which would not necessarily be continuous.

**Improved Parametrization** The parametrization defines how the surface will be sampled. We look for a parametrization that minimizes the geometric-stretch metric of the mapping from image space to object space. We find that computing the parametrization using an arbitrary frame yields satisfactory results because the choice of a cut has the greatest impact on the parametrization (we compute the cut using a global algorithm). It is likely that for animations with major deformations this approach would yield unsatisfactory results. A parametrization that considers all frames may be more appropriate.

We have also explored parametrizations that can yield better reconstruction. For example, it is possible to “snap” the internal vertices of the parametrization to grid locations, so some vertices in the original mesh are sampled exactly. This did not yield much better reconstruction, counter to our intuition (it is sampling original vertices exactly); although in the limit it improves the reconstruction quality, in general, it hurts compression, reducing the final quality. This is another area for further exploration.

We have only considered one metric for the parametrization. Depending on the application, other metrics may be more important to minimize. For example, Sander [San03] suggests building parametrizations optimized for final rendering, i.e. the error metric would “measure” the pixel value error of the rendered object.

**Improved Geometry Image Compression** Currently each X-Y-Z coordinate plane of the Geometry Image is compressed separately (i.e. the red, green and blue channels are compressed independently). There is still some coherence left between different coordinate planes. It should be possible achieve greater compression by changing the “color” model (similar to converting from RGB space to a chroma-luminance model). A similar approach
to this is called "local frame coordinate" optimization proposed by Zorin et al. [ZSS97]. The idea is that locally each face is aligned with the $xy$ plane and most of the information will reside on the $z$-coordinate plane.

Additionally, the compression of the wavelet encoder can be improved. Most compression schemes will make assumptions about the boundaries, either using mirroring or replication techniques to generate points that would otherwise lie beyond the boundary. In our case, we know how the boundary "wraps around." This information can be useful to improve the compression of the information near the boundaries of the Geometry Image. The disadvantage of this approach could be the added complexity to the wavelet algorithm.

**Overcoming Remaining Limitations** One of the limitations of Geometry Videos is the handling of connectivity changes and topology changes within the animation. We assume constant connectivity throughout the 3D animation. Handling connectivity changes should be a manageable problem, at the very least a solution is a consistent remeshing of all frames in the animation. A larger problem is handling topological changes. Topological changes can cause large changes in the necessary cut. It might be hard to smoothly transition between frames processed by significantly different cuts and/or parametrizations. We have proposed some solutions to this problem (see Section 4.2) and further research might bring different solutions to this limitation.

Our approach involves a resampling of the original mesh. This resampling facilitates the application of standard video encoding techniques. If the original mesh connectivity is essential, it would be interesting to explore using Geometry Images as a first order predictor for the *original* geometry. The idea would be to use the Geometry Videos approach to encode the animation, but keep the original parametrization and connectivity. With that information, it possible to reverse sample the Geometry Video to obtain the approximate location of the original samples with the original connectivity. Then, one could encode the residuals between the re-sampled Geometry Video and the original vertex locations. This approach retains some of the advantages of Geometry Videos: small size, level of detail support, etc... and maintains the original connectivity of the mesh. It would be interesting to see if it is more efficient than using predictors and encoding residuals directly on the original mesh.
Other Applications  Other more exciting applications of Geometry Videos include ray tracing of dynamic scenes. The regular grid connectivity of Geometry Videos could allow the implementation of very fast searching algorithms. The transformations used for the predictors could also be used to predict light changes, reducing the amount of computation.

Another interesting application is in the generation of special effects. For example, if one were to super-sample the Geometry Image to eight times its size, high detail bump maps could be applied to them, either repeating small bump-maps or using very large bump-maps. This is easier to apply with Geometry Images, because they have the same shape as textures, and super-sampling classical geometry would involve sub-division which is more complex. These effects could be applied to animations in the same manner.

In this thesis I have introduced a new representation for animations. I have implemented a system that uses this representation and shows the promise of this approach. This new media type has considerable potential for further development.
Appendix A

Wavelet Coefficients

We found the Villa1810 filter to give us the best results when compressing geometry images. 18/10 filter from Villasenor's group [TVC96]

<table>
<thead>
<tr>
<th>Synthesis</th>
<th>Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.544158682436510e-04</td>
<td>2.885256501123136e-02</td>
</tr>
<tr>
<td>-2.7271962909995984e-06</td>
<td>8.244478227504624e-05</td>
</tr>
<tr>
<td>-9.452462998353147e-03</td>
<td>-1.575264469076351e-01</td>
</tr>
<tr>
<td>-2.528037293949898e-03</td>
<td>7.67904884691438e-02</td>
</tr>
<tr>
<td>3.083373438534281e-02</td>
<td>7.589077294537618e-01</td>
</tr>
<tr>
<td>-1.376513483818621e-02</td>
<td>7.589077294537619e-01</td>
</tr>
<tr>
<td>-8.56611833165798e-02</td>
<td>7.67904884691436e-02</td>
</tr>
<tr>
<td>1.633685405569902e-01</td>
<td>-1.575264469076351e-01</td>
</tr>
<tr>
<td>6.233596410344172e-01</td>
<td>8.244478227504624e-05</td>
</tr>
<tr>
<td>6.233596410344158e-01</td>
<td>2.885256501123136e-02</td>
</tr>
<tr>
<td>1.633685405569888e-01</td>
<td>-8.566118833165885e-02</td>
</tr>
<tr>
<td>-8.566118833165885e-02</td>
<td>-1.376513483818652e-02</td>
</tr>
<tr>
<td>3.083373438534267e-02</td>
<td>9.452462998353147e-03</td>
</tr>
<tr>
<td>-2.528037293949898e-03</td>
<td>-2.7271962909995984e-06</td>
</tr>
<tr>
<td>9.544158682436510e-04</td>
<td></td>
</tr>
</tbody>
</table>

Table A.1: Villa1810 Wavelet Coefficients
Figure A-1: Synthesis and Analysis filters.
Bibliography


