Progressive 3D mesh generation

Nghiem, Viet Nam

2010

Nghiem, V. N. (2010). Progressive 3D mesh generation. Master's thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/41536>

<https://doi.org/10.32657/10356/41536>

**NANYANG
TECHNOLOGICAL
UNIVERSITY**

PROGRESSIVE 3D MESH
GENERATION

PROGRESSIVE 3D MESH GENERATION

NGHIEM VIET NAM

NGHIEM VIET NAM

SCHOOL OF COMPUTER ENGINEERING

2010

2010

PROGRESSIVE 3D MESH GENERATION

NGHIEM VIET NAM

School of Computer Engineering

A thesis submitted to the Nanyang Technological University
in partial fulfillment of the requirement for the degree of
Master of Engineering

2010

Abstract

One important aspect for any 3D graphics application is 3D shape representation. In every graphics application, there always exists a trade-off between complexity and performance. The more realistic 3D representation comes at a price of high computational complexity. In recent years, a new type of 3D shape representation, i.e. *progressive mesh*, has been received much attention since it provides a nice balance between complexity and performance. In this thesis, we focus on generating progressive meshes in two scenarios: *from a given complex 3D textured model*, and *from a set of calibrated and segmented multi-view images*.

For the first scenario, we consider the problem of building progressive representation of a complex 3D model with texture information in a memory-efficient way. Simplification is the most commonly used method to generate progressive meshes from a complex model. In the first part of the thesis, we introduce a new method to extend the quadric error metrics for simplifying complex models with texture attributes. Different from the conventional approaches for 3D textured model simplification, which minimize the summation of geometric errors and texture errors, our method separates the texture error minimization from the geometric error minimization so as to reduce the memory requirement. The experimental results compared with other methods show its effectiveness.

For the second scenario, we study the problem of reconstructing progressive 3D models from a set of multi-view images in a rate-distortion optimized manner. 3D reconstruction from multi-view images has been a very active research area in the past few years. Most of previous studies focus on how to reconstruct 3D models as close to their real-world objects as possible, and rely on a subsequent simplification process to generate progressive meshes for practical usage. In this thesis, we directly generate progressive meshes from the reconstruction process. From a set of calibrated and segmented wide baseline images,

our developed system is able to reconstruct a set of optimal progressive 3D meshes that achieve minimum reconstruction distortion at any reconstruction rate within a certain range. To the best of our knowledge, for wide baseline multi-view reconstruction, there is no prior work addressing similar issues.

Acknowledgments

First and foremost, I would like to express my sincere appreciation and gratitude to my supervisor, Associate Professor Cai Jianfei, for his persistent support and invaluable advice in my research and preparation of this report. It has been a great pleasure to have him as my supervisor. I would also like to express my gratitude to Associate Professor Zheng Jianmin, for his invaluable guidance, advice, and comments to my research. Finally, I would like to express my appreciation to my fellow graduate students in the Center for Multimedia and Network Technology for the helpful discussions.

Contents

Abstract	i
Acknowledgments	iii
List of Figures	vi
List of Tables	viii
1 Introduction	1
1.1 Thesis Organization	5
2 Literature Review	6
2.1 Triangular Mesh	6
2.1.1 Texture Mapping	8
2.1.2 Progressive Mesh	9
2.2 3D Reconstruction from Multi-view Images	10
2.2.1 Visual Hull	10
2.2.2 Triangulation	11
2.2.3 Remeshing	13
2.3 PDE-based Surface Reconstruction	15
2.3.1 Mean Curvature Normal Estimation	17
3 Progressive Mesh Generation Using Memory-efficient Simplification	18
3.1 Simplifying Complex Models with Attributes	18
3.2 Simplification Based on Quadric Error Metrics	21
3.2.1 Simplifying Models without Attributes	21
3.2.2 Simplifying Models with Attributes	22
3.3 Memory Efficient Simplification Based on QEM	23

3.3.1	Geometric Error Minimization	24
3.3.2	Attribute Error Minimization	25
3.3.3	Attribute Discontinuities	28
3.4	Experimental Results	29
3.5	Discussions	32
4	Progressive 3D Reconstruction from Multi-view Images	36
4.1	System Overview	36
4.2	Initial Mesh Generation	38
4.3	Mesh Optimization Using PDE-based Surface Flow	39
4.3.1	Distortion Measurement	40
4.3.2	Computing Distortion Gradient	43
4.3.3	Mesh Regularization	43
4.3.4	Solving PDE-based System	44
4.3.5	Boosting Performance by Using GPU	44
4.4	Progressive Mesh Refinement via Local Adaptive Subdivision	45
4.5	Texture Fitting	47
4.6	Experimental Results	51
5	Conclusions and Future Work	57
5.1	Conclusions	57
5.2	Future Work	58
	References	60
	Publications	65

List of Figures

2.1	Example of a horse model.	7
2.2	Illustration of progressive representation.	9
2.3	Four silhouette images from input image set.	11
2.4	Four different views of a reconstructed visual hull.	12
2.5	Illustration of triangulated surfaces. a) Triangulated surface using marching cube based algorithm [1]. b) Triangulated surface using Poisson surface reconstruction algorithm [2].	14
2.6	Irregular and regular surfaces. a) Irregular surface from Poisson surface reconstruction [2]. b) Regular surface after applying remeshing algorithm [3].	15
2.7	Mean curvature normal estimation.	17
3.1	Edge collapse operation $(V_3, V_4) \rightarrow V_n$ in R^3	24
3.2	Local region affected after each edge collapse in R^3	25
3.3	Construction of mean value coordinates in R^3	27
3.4	Edge collapse operation $(V_4, V_5) \rightarrow V_n$ at a complex region in R^3	29
3.5	Close view of the simplified model.	31
3.6	Simplifying the <i>horse</i> model of 25000 faces to 2500 faces.	33
3.7	Simplifying the <i>bull</i> model of 28000 faces to 6000 faces.	34
3.8	Simplifying the <i>camel</i> model of 30000 faces to 7000 faces.	35
4.1	System flowchart.	37
4.2	Illustration of volume elements (voxels) and triangulated surface of a visual hull.	39
4.3	An example of sampling around one vertex.	42
4.4	An example of the local adaptive subdivision scheme.	46

4.5	2D texture map of a 3D surface.	50
4.6	Interpolation of a point in a triangle.	50
4.7	Visual comparison of the reconstructed <i>Venus</i> models: (a) initial coarse mesh (generated in the first major step) with 2454 vertices and 4904 triangles, (b) optimized coarse mesh , (c) refined mesh using our proposed system with 7377 vertices, (d) optimized fine mesh using [4] with 9810 vertices, (e) refined mesh using our proposed system with 9837 vertices, and (f) one of the input images.	52
4.8	Visual comparison of the reconstructed <i>Pot</i> (top row) and <i>Bowl</i> (bottom row) models: (a) optimized coarse meshes with 5308 vertices and 10612 triangles for <i>Pot</i> and 5296 vertices and 10588 triangles for <i>Bowl</i> , (b) optimized fine meshes using [4] with 29886 vertices for <i>Pot</i> and 10631 vertices for <i>Bowl</i> , (c) results of our proposed system with 29884 vertices for <i>Pot</i> and 10639 vertices for <i>Bowl</i> , and (d) two input images for <i>Pot</i> and <i>Bowl</i> respectively.	53
4.9	The average per-sample distortions of the reconstructed <i>Venus</i> models at different number of vertices.	54
4.10	The average per-sample distortions of the reconstructed coarse <i>Venus</i> model with 2887 vertices and 5770 triangles at low sampling resolution and different optimization iterations.	55
4.11	Visual results of 3 models before and after texture fitting: <i>Venus</i> with 9837 vertices (top), <i>Pot</i> with 29884 vertices (middle), and <i>Bowl</i> with 10639 vertices (bottom).	56

List of Tables

3.1	The results of memory requirement.	30
-----	--	----

Chapter 1

Introduction

An ultimate purpose of computer graphics is to produce a virtual world that looks real. In every graphics application, there always exists a trade-off between complexity and performance. The more complex the representation of the virtual world, the lower the performance of the graphics application. It is well-known that the representation of 3D virtual objects is one of the most important aspects in graphics applications. Choosing the right 3D shape representation can boost the performance of an application significantly.

A few years ago, a new type of 3D shape representation was introduced by Hoppe [5], i.e. *progressive mesh*, to tackle the tension between complexity and performance. The underlying principle of progressive meshes is to provide a flexible representation of 3D shape in such a way that the size of 3D shape can be adjusted according to application requirements. The flexibility of this representation has been exploited in many applications such as dynamic rendering, compression, transmission, etc. This high demand also drives more research studies on how to efficiently generate progressive meshes.

The most common way to generate progressive meshes is through simplification, which takes a highly detailed model as its input, and removes least important details in an iterative manner. A large number of simplification algorithms such as [6, 7] have been proposed in literature. Most of the algorithms focus on simplifying geometric data only.

CHAPTER 1. INTRODUCTION

Among these algorithms, the one proposed by Garland and Heckbert [8] gives the best trade-off between processing time and visual quality. Another nice feature of the method in [8] is its simplicity. Specifically, by defining the distortion model based on *quadric error metrics* (QEM), their method can be easily implemented.

Real 3D models contain not only geometric data but also appearance data. However, only a few techniques [9, 10, 11] consider surface attributes. In [9], Garland et al. proposed a natural extension to extend their QEM model to include surface attributes. The main drawback of the natural extension is that it dramatically increases the memory size required to store the error metrics with the increase of the number of attributes. To overcome the drawback of the natural extension, Hoppe [10] proposed to extend the quadric error metric by defining both geometric distortion and attribute distortion based on geometric correspondence in 3D space. Although it reduces the memory requirement, its performance is not as good as Garland’s method in [9]. Following a different direction, Sander et al. [11] proposed a different method to generate a set of simplified models, while considering texture distortion minimization of all the simplified models. Although this method can achieve very good performance of texture mapping, it requires a long processing time.

To generate a progressive mesh, it is assumed that an initial high-quality 3D model is available, to which the simplification process is applied. In practical applications, such an assumption is not always valid. In other words, an initial 3D model has to be created first either manually or via a 3D reconstruction process. Manually creating 3D models requires a huge effort from modelers, and it’s very difficult to make the artificial models look real. An alternative way is to capture real-world objects using 3D scanners. The drawback is that 3D scanners are expensive and not easy to set up.

With the easy capturing of 2D images, a very promising way is to reconstruct 3D models from a set of 2D multi-view images. In the past few years, 2D-to-3D reconstruction has received much attention, and so far there have been various 3D reconstruction

methods proposed in literature. Based on the camera settings, 3D shape reconstruction from multi-view images can be categorized into two main groups: small baseline reconstruction and wide baseline reconstruction. The cameras in small baseline reconstruction are placed close to each other so as to obtain the stereo information. In such a setting, the number of cameras needed is typically large in order to capture the entire scene. In contrast, the number of cameras needed for wide baseline reconstruction is much less at the cost of placing cameras farther apart, where stereo information is not available.

The quality of reconstructed 3D models can be evaluated based on two main criteria: reconstruction distortion and reconstruction rate. Reconstruction distortion tells how much the difference is between a reconstructed model and its corresponding real-world 3D object, while reconstruction rate conveys how compact the representation of a reconstructed model is.

Reconstruction distortion has been the main issue in most of the previous multi-view reconstruction techniques. We refer readers to the nice surveys by Seitz et al. [12] and Dyer [13]. Most of the previous multi-view reconstruction algorithms focus on how to reconstruct the 3D model as close to the real object as possible. In other words, little attention has been paid to the issue of reconstruction rate. Reconstruction models generated by the state-of-the-art multi-view stereo algorithms [12] or obtained from 3D scanners [14] are typically of huge volume, which are not suitable to be directly used in practical applications. Simplification methods such as [15] need to be applied as a post-processing to not only reduce the size of the reconstructed models but also obtain the progressive representation of the reconstructed models.

Such a fine-to-coarse processing, i.e. high quality multi-view reconstruction followed by simplification, is resource-inefficient since it over-creates numerous vertices or 3D points at the beginning and then removes many of them in the simplification stage. Here one fundamental question is: *can we directly consider the reconstruction rate in the multi-view reconstruction instead of relying on the subsequent simplification to control the rate?*

CHAPTER 1. INTRODUCTION

Moreover, considering that different applications have different rate requirements, it is highly desired to have a multi-rate 3D reconstruction, i.e. we only need to reconstruct the 3D model once and the representation can be truncated to fit different rate requirements with different reconstruction quality. This progressive feature is very useful for the applications of progressive transmission and rendering.

In this thesis, we first consider a simple scenario of generating progressive meshes, i.e. simplifying a given high-quality 3D mesh. In particular, we develop a new method to extend the QEM model to compute both geometry and texture coordinates for each level of progressive representation. Our basic idea is to separate the texture error minimization from the geometric error minimization so as to reduce the memory requirement. Specifically, we apply the original QEM model to compute the optimal geometric coordinates while using the mean value coordinate to calculate the texture coordinates of the region affected by simplification. Experimental results show that our approach can significantly reduce the memory requirement without sacrificing the visual quality of 3D models.

In the second part of the thesis, we consider a more difficult scenario of generating progressive meshes, i.e. with only a set of multi-view 2D images as the input instead of a high-quality 3D mesh. In particular, we propose a framework for rate-distortion optimized progressive 3D reconstruction from a set of calibrated and segmented wide baseline images. Specifically, a coarse triangular mesh is first built up and then optimized by using the state-of-the-art PDE-based mesh optimization method [16, 4], which minimizes the distortion at a certain rate. We then keep applying a local subdivision approach to add new vertices to the most distorted triangle until reaching the target reconstruction rate. For each local subdivision, the newly added vertices and their adjacent vertices are optimized using the same PDE-based optimization method. Experimental results show that our proposed system significantly outperforms the state-of-the-art PDE-based reconstruction method [4].

1.1 Thesis Organization

The rest of the thesis is organized as follows. Chapter 2 reviews the background of the most popular 3D shape representation, i.e. 3D triangular mesh, together with its flexible progressive representation, and briefly introduces some key components of a 3D reconstruction system as well as the state-of-the-art PDE-based surface reconstruction algorithm. In Chapter 3, we focus on the mesh simplification process to generate progressive representation of a 3D model with texture attributes in a memory-efficient manner. Chapter 4 describes the proposed framework to generate optimal progressive meshes from multi-view images. We show how to introduce the rate-distortion concepts into a 3D reconstruction problem. Finally, in Chapter 5, we conclude the thesis and discuss some future research directions.

Chapter 2

Literature Review

In this chapter, we explain some concepts which are crucial to the understanding of the following chapters. In the first section, we discuss in details the most common representation of 3D shape, i.e. *triangular mesh*, together with its flexible progressive representation. In the last two sections, we cover some background concepts which are used in the following chapters, including visual hull, triangulation, remeshing, and the state-of-the-art PDE-based surface reconstruction method.

2.1 Triangular Mesh

Currently polygonal meshes are the most popular representation of polyhedral objects in 3D computer graphics and solid modeling. This is because polygonal meshes provide a piecewise linear approximation to shape, based on which the visibility and colors of most pixels can be interpolated across the polygon's surface. Moreover, most model representations (spline, implicit surface, volume iso-surface) can be converted with arbitrary accuracy to a polygonal mesh. Informally, a polygonal mesh can be defined as a collection of vertices, edges, faces, and additional attributes. The faces can be in the form of triangles, quadrilaterals, or other simple convex polygons. Since other types of polygons can be triangulated into triangles and triangular meshes are efficiently supported

CHAPTER 2. LITERATURE REVIEW

2.1.a: With geometry data only

2.1.b: With geometry data and texture data

Figure 2.1: Example of a horse model.

in hardware, triangular meshes are the most common representation for many applications ranging from video games to visualization of medical, scientific, and CAD data sets. In our work, we use triangular meshes as our object representation. An example of triangular meshes is shown in Fig. 2.1.

Before proceeding further, we first clarify the terminology and notation that are used throughout the thesis.

- **Triangular mesh:** A triangular mesh M consists of a set of vertices V and an abstract simplicial complex K , i.e. $M = (V, K)$. The abstract simplicial complex K contains all topological information about M i.e. how each vertex is connected to one another. The complex K is composed of three types of subsets: a set of vertex indices, a set of edges, and a set of faces. Faces in a triangular mesh are all triangles. The vertex set V contains position data of all vertices and it defines the geometry of M .
- **Vertex:** A vertex v is a position (x, y, z) along with other information such as color, normal vector, and texture coordinates

- **Edge:** An edge $e = \{i, j\} \in K$ is the connection between vertex i and vertex j
- **Face:** A face $f = \{i, j, k\} \in K$ is a closed set of three edges $\{i, j\}$, $\{j, k\}$, and $\{k, i\}$. In rendering applications, the order of three edges is important because it determines the normal vector of the face.
- **Neighboring vertex:** Vertex i is a neighbor of vertex j if they are connected by an edge.
- **1-ring neighborhood:** The 1-ring neighborhood of a vertex is a set of vertices that are connected to it by edges.
- **Valence:** The valence of a vertex is the number of its neighboring vertices.
- **Boundary vertex:** A vertex is a boundary vertex if its 1-ring neighborhood does not form a complete ring around it.
- **Boundary edge:** An edge is a boundary edge if it is shared by a single face only.

2.1.1 Texture Mapping

Besides geometry and connectivity data, a mesh may contain attribute data. There are several attribute data types such as normals, colors, texture coordinates. These data are usually associated with vertices or corners of the faces [17]. A real mesh would require every 3D detail of its surface. Since representing data in 3D space requires much more storage than in 2D space, texture mapping is used as an efficient means to describe 3D details in 2D space.

The basic idea of texture mapping is to relate a 2D image to a 3D mesh. An example of a textured mesh is shown in Fig. 2.1. 2D images are usually considered as the texture domain, and 3D meshes are considered as the geometry domain. Texture mapping defines a mapping function from the texture domain to the geometry domain, where each vertex

is projected to a point $p_i = (u, v) \in R^2$ in the texture domain. Each point p_i in the texture domain represents a texture coordinate. The value of (u, v) ranges from $(0, 0)$ to $(1, 1)$, which corresponds to the bottom left corner and the top right corner of the texture respectively.

2.1.2 Progressive Mesh

The basic idea of progressive mesh representation is to generate a set of approximations of a 3D object $M_0, M_1, \dots, M_n$, where M_0 is the coarsest level of the 3D object, and M_n is its finest level. At the coarsest level, the mesh contains only a small amount of data, hence its size is relatively small and its accuracy is very low. To increase the level of the accuracy, an additional amount of refinement data is needed. The more the amount of refinement data, the better the approximation of the 3D object.

Figure 2.2: Illustration of progressive representation.

Fig 2.2 illustrates a typical process to construct progressive representation of a 3D object. There are two common techniques to generate progressive meshes: simplification and refinement. The main objective of a simplification process is to reduce the size of a 3D object while roughly maintaining its visual quality. There have been a large number

of simplification techniques in literature such as vertex decimation, quadric error metrics based simplification, reverse simplification, image-driven simplification, etc [18]. The final output of a simplification process is the coarsest representation of a 3D object.

Another progressive mesh generation technique starts with a very coarse mesh, and adds more and more details to the coarse mesh iteratively in such a way that it maximizes the similarity to the original 3D object. The common way to refine a mesh is through a subdivision process, which can be categorized into global subdivision and local subdivision. A global subdivision process adds details to overall surface of a 3D object, whereas a local subdivision process adds details to certain important regions on the surface of a 3D object. Among the four most popular subdivision schemes (Butterfly, Loop, Catmull-Clark, and $\sqrt{3}$) [19], we choose the Loop subdivision scheme as its implementation is fast and efficient.

2.2 3D Reconstruction from Multi-view Images

The complexity of a multi-view 3D reconstruction system depends much on the quality of its reconstructed output. A high-quality reconstructed output requires a fairly complex system, which consists of a number of separate supporting algorithms. In the following sections, some of the main algorithms are introduced in details.

2.2.1 Visual Hull

Visual hull is one of the most basic reconstructed results from a multi-view 3D reconstruction process. It is created by the shape-from-silhouette method first introduced by Aldo Laurentini [20]. It's assumed that input images are segmented and calibrated so that they can be thresholded into binary images, which are called silhouette images, with 1 indicating the object and 0 indicating the background. An example of silhouette images is shown in Fig. 2.3.

Figure 2.3: Four silhouette images from input image set.

Back-projection of each silhouette image forms a generalized cone that is called a silhouette cone. Intersection of a set of silhouette cones forms the visual hull of an object. The geometrical meaning of the visual hull of an object is that it defines the upper bound of the approximation of 3D representation of the object such that substituting the 3D object with its visual hull does not affect any silhouette. Fig. 2.4 shows an example of a reconstructed visual hull.

One important point about the visual hull representation is that it only recovers convex geometry information. Recovering other information such as concave information and connectivity information from visual hull is not possible. The accuracy of the visual hull reconstruction depends not only on the 3D sampling resolution of the reconstruction algorithm, but also on the number of input silhouette images. A high sampling resolution together with a large number of silhouette images can achieve a good approximation of the real object.

2.2.2 Triangulation

Since the visual hull contains merely geometry information, it's usually followed by a triangulation process to have connectivity information. The result of a triangulation process is a polygonal mesh, which contains a set of vertices and a set of triangles.

Among the existing triangulation algorithms, marching cubes is one of the most popular ones. It's first introduced by Lorensen et al. [21]. The algorithm proceeds through

Figure 2.4: Four different views of a reconstructed visual hull.

each volume element (voxel), creating an imaginary cube from eight neighboring locations. The algorithm then determines which edges of the cube are intersected by the true object surface, and creates triangular patches which divide the cube between regions inside the object surface and regions outside the object surface. A surface representation is reconstructed by connecting all the triangular patches from all cubes. Because marching cube based algorithms process each voxel separately, they are highly sensitive to noise. When applying the marching cube based algorithms to a visual hull, the reconstructed surface is fairly rough.

A better choice to triangulate a visual hull is to use an algorithm that considers all the voxels at the same time. Therefore, the reconstructed surface is highly resilient to

data noise. One of such algorithms is the Poisson surface reconstruction algorithm [2].

Fig. 2.5 shows a visual comparison of reconstructed surfaces using a more robust marching cube based algorithm [1] and using the Poisson surface reconstruction algorithm [2]. The reconstructed surface using the Poisson surface reconstruction algorithm is much smoother than the one using the marching cube based algorithm.

2.2.3 Remeshing

The surface obtained from the triangulation process is highly irregular. Some regions of the triangulated surface are oversampled and may contain redundant vertices. There's a need to improve the mesh quality in terms of vertex sampling, regularity, and triangle quality. Such a process is called *remeshing*.

There are many remeshing algorithms in the literature [22]. Each remeshing algorithm is designed to meet certain requirements, which are highly application-dependent. In general, they can be classified into five groups:

- *Structured remeshing* replaces an unstructured input mesh with a structured one.
- *Compatible remeshing* generates a new set of meshes from a set of input meshes having a partial correspondence between them. In this way, all the meshes have a common connectivity structure, well-shaped polygons, good approximation to the input meshes, and maintain the correspondence.
- *High quality remeshing* generates a new discretization of the original geometry of input meshes, such that the new meshes possess the following properties: well-shaped elements, uniform or isotropic sampling, and smooth gradation sampling.
- *Feature remeshing* creates a new mesh of an input mesh, such that the new mesh preserves the original sharp features of the input mesh.

CHAPTER 2. LITERATURE REVIEW

- *Error-driven remeshing* optimizes the trade-off between the complexity and the accuracy of output meshes.

Among the five remeshing algorithms, high quality remeshing has a number of properties that ensure numerical stability and reliability of computations for subsequent mesh processing such as simulation, rendering, interactive free-form shape modeling, compression, fairing, and smoothing. Triangles of output meshes from high quality remeshing have an aspect ratio close to 1. Vertex sampling is locally uniform in all directions. If the sampling density is not uniform, it varies in a smooth manner.

2.5.a: Marching cubes

2.5.b: Poisson surface reconstruction

Figure 2.5: Illustration of triangulated surfaces. a) Triangulated surface using marching cube based algorithm [1]. b) Triangulated surface using Poisson surface reconstruction algorithm [2].

Due to these properties of high quality remeshing, we use it as the remeshing method in our multi-view 3D reconstruction system. In particular, we use the remeshing algorithm introduced by Alliez et al. [3], which is called the interactive geometry remeshing algorithm. In the remeshing process, an input mesh is first globally parameterized to

2.6.a: Triangulated surface

2.6.b: Regular surface

Figure 2.6: Irregular and regular surfaces. a) Irregular surface from Poisson surface reconstruction [2]. b) Regular surface after applying remeshing algorithm [3].

produce a set of 2D maps. These maps are combined to produce a control map, indicating the desired sampling distribution, which can be controlled by users. The control map is then sampled using an error-diffusion sampling technique originally designed for image half-toning, followed by relaxation using the umbrella operator. The samples from the control map are triangulated, optimized, and finally outputted as a new mesh. Fig. 2.6 illustrates a visual example of the remeshing output, in which the input mesh on the left is highly irregular, and the regular mesh on the right is generated by the interactive geometry remeshing algorithm.

2.3 PDE-based Surface Reconstruction

The PDE-based surface flow is a gradient descent flow that is governed by the Euler-Lagrange equation with a certain underlying energy function. In our context, the energy function is the one that measures the geometry distortion of all the reconstructed points

CHAPTER 2. LITERATURE REVIEW

on the surface [23, 24]:

$$E(S) = \int_S G(\mathbf{p}) \, \partial S, \quad (\text{Eq. 2.1})$$

where $G(\mathbf{p})$ is a function that measures the geometry distortion of point $\mathbf{p} = (x, y, z)^T$ on the surface.

To minimize the energy function, we need to solve the equation of $E'(S) = 0$, which is typically hard to solve directly for most computer vision applications. One general technique is to solve the PDE in a numerical way:

$$E'(S) = -\frac{\partial S}{\partial t}, \quad S(0) = S_0, \quad (\text{Eq. 2.2})$$

where S_0 is the initial surface, and the variable t is the time marching parameter. The solution is obtained by starting with an initial surface S_0 and then searching for a stable state of Eq. 2.2, i.e. $\frac{\partial S}{\partial t} = 0$.

Since the complete form of Eq. 2.2 is very complex, a simplified version is typically used [23]:

$$\frac{\partial S}{\partial t} = (G H + \nabla G_{\mathbf{N}}) \mathbf{N}, \quad S(0) = S_0, \quad (\text{Eq. 2.3})$$

where H is the mean curvature, G is the non-negative, monotonically decreasing function that measures the geometry distortion of the reconstructed surface, and the scalar $\nabla G_{\mathbf{N}}$ is the derivation of G along the normal vector $\mathbf{N}$. Basically, Eq. 2.3 drives each point on the surface to evolve along its current normal direction. The first term in Eq. 2.3 is a smoothing term for reducing the mean curvature of the surface, and the second term moves the surface in the direction of reducing the surface error.

Specifically, starting from a simple initial surface S_0 , the surface is incrementally updated as

$$S(\mathbf{p})^{t+\Delta t} = S(\mathbf{p})^t + \frac{\partial S(\mathbf{p})^t}{\partial t} \Delta t, \quad (\text{Eq. 2.4})$$

where $\frac{\partial S(\mathbf{p})^t}{\partial t}$ computed by Eq. 2.3 controls the evolving direction and the time step Δt affects the evolution speed. Note that when updating the surface, the time step Δt must

satisfy the Courant-Friedrichs-Lewy stability criterion [25], i.e. the moving speed of each point on the surface must be strictly restrained by the minimum detail in the system.

Figure 2.7: Mean curvature normal estimation.

2.3.1 Mean Curvature Normal Estimation

When expanding Eq. 2.3, the first term contains a vector $H\mathbf{N}$, which is known as the mean curvature normal vector. Since the mean curvature normal is recomputed many times for each point on the surface, its estimation is expected to be of low complexity. An efficient method to compute the mean curvature normal vector of the surface is proposed by Desbrun et al. [26]:

$$H\mathbf{N} = \frac{1}{\sum_j (\cot\alpha_j + \cot\beta_j)} \sum_j (\cot\alpha_j + \cot\beta_j)(\mathbf{P} - \mathbf{Q}_j), \quad (\text{Eq. 2.5})$$

where $H\mathbf{N}$ is the mean curvature normal vector at vertex $\mathbf{P}$, $\mathbf{Q}_j$ is one of neighboring vertices of $\mathbf{P}$, and α_j, β_j are two angles facing the same edge connecting $\mathbf{P}$ and $\mathbf{Q}_j$, as illustrated in Fig. 2.7.

Chapter 3

Progressive Mesh Generation Using Memory-efficient Simplification

With the fundamental concepts covered in the previous chapter, we are now ready to look into the details of our first work. In this chapter, we propose a memory-efficient method to generate progressive representation of a complex 3D object via a simplification process. Our proposed method extends the quadric error metrics for simplifying a 3D object with texture attributes. Different from the conventional approaches for textured 3D model simplification, which minimizes the summation of geometric errors and texture errors, the proposed method separates the texture error minimization from the geometric error minimization so as to reduce the memory requirement.

Before discussing the details of the proposed method, we first cover some key concepts about simplification of 3D models with attributes in Section 3.1. An important simplification method, which our proposed method is built upon, is introduced in Section 3.2. The proposed method is elaborated in Section 3.3, which is followed by the experimental results in Section 3.4. The last section summarizes the chapter with some discussions.

3.1 Simplifying Complex Models with Attributes

To achieve an acceptable level of realism, scanning and acquisition systems often generate 3D models that are far more complex than actually required for real applications. These

CHAPTER 3. PROGRESSIVE MESH GENERATION USING MEMORY-EFFICIENT SIMPLIFICATION

3D models usually contain both geometric data and appearance data. Geometric data consists of vertex coordinates and connectivity information. Appearance data, also known as surface attributes, describes additional information about the models, such as colors, textures, and surface normals. Because the size of an original model is huge, it is not suitable for some applications such as networking applications, mobile applications, and real time rendering. There is a need to design a more flexible representation of a 3D model so that it can be used for a wide range of applications. Given a 3D model, a typical method is to use a simplification process to remove least important details from the 3D model iteratively. As a result, a set of meshes is generated, which is called progressive meshes. This set provides a range of approximations to the original model at different sizes.

The goal of a model simplification algorithm is to reduce the model complexity by removing a number of model elements while maintaining the shape and appearance of the model. There are three types of model elements: vertex, edge, and face. Simplification algorithms can be classified based on the type of chosen elements: vertex removal, face removal, or edge collapse. Among these three types, edge collapse is most popular due to its efficiency and simplicity.

There have been a large number of simplification algorithms [6, 7]. Most of the algorithms focus on simplifying geometric data only. Among these algorithms, the one proposed by Garland and Heckbert [8] gives the best trade-off between processing time and visual quality. Another nice feature of the method in [8] is its simplicity: by defining their distortion model based on quadric error metrics, they make the implementation of their method much easier.

Although ready-to-use 3D models contain not only geometric data but also appearance data, only few techniques consider surface attributes of models [9, 27, 10, 28, 11]. In [9], Garland et al. introduced a natural extension of the distortion model in [8] by extending their quadric error metrics to include surface attributes. The main drawback of the

CHAPTER 3. PROGRESSIVE MESH GENERATION USING MEMORY-EFFICIENT SIMPLIFICATION

natural extension is that it increases the memory size of the error metrics dramatically, according to the formula $(4 + m)(5 + m)/2$ [10], where m is the number of additional attributes to be included. If we include texture coordinates in the distortion model, i.e. $m = 2$, the number of coefficients required to represent each quadric error metric is 21. To overcome the drawback of the natural extension, Hoppe [10] introduced a new way to extend the quadric error metric by defining both geometric distortion and attribute distortion based on geometric correspondence in 3D space. They define attribute distortion as a difference in attribute values of a vertex and its projected image onto the neighboring surface. By doing so, they reduce the number of coefficients required for each quadric error metric to $(11 + 4m)$. If we include texture coordinates in the new error metrics, the number of coefficients is only 19, saving 9.5% of memory requirement as compared to Garland’s method.

Following a different direction, Sander et al. [11] proposed a method to generate a set of simplified models, while considering texture distortion minimization of all the simplified models. The texture distortion model is defined as a texture stretch metric. Minimizing the stretch metrics gives optimal texture attributes of all vertices at all the simplification levels. The quality of the texture map is very good. However, the main disadvantage is that it requires a very long processing time.

In some environments such as mobile devices, memory is a very limited resource and it is worth trading off a certain level of accuracy for memory efficiency. With this consideration in mind, we introduce in our system a novel extension of QEM to save up to 52.4% of memory requirement while attaining an equivalent level of visual quality as compared to the natural extension of QEM. The output of our algorithm is a sequence of models at different levels of details and all the models share a common texture parameterization as in [11].

3.2 Simplification Based on Quadric Error Metrics

A simplification algorithm generally shares a common structure: the first part to determine which simplification operator to use and the second part to decide the priority for the simplification operator. Most common simplification operators are vertex grouping, vertex removal, face removal, and edge collapsing. Among these operators, edge collapsing is most popular due to its progressiveness nature. Each edge collapse contracts its two vertices to a new vertex and removes the collapsed edge and its two incident triangles. Hence there are two main issues in a simplification algorithm based on edge collapses: (1) the priority to apply edge collapses, and (2) the position and attributes to assign to the new vertex. A common approach is to define a cost model to measure the cost incurred by each edge collapse. The position and attributes of the new vertex is determined by minimizing the cost model.

3.2.1 Simplifying Models without Attributes

The cost of each edge collapse is defined as the error caused by each collapse. There are two types of errors: one caused by geometric data and the other caused by appearance data. The most popular cost model to measure geometric error is introduced by Garland et al. [8], i.e. Quadric Error Metric. The QEM model defines on each original face a quadric $Q^i(\mathbf{v})$ equal to the squared distance of vertex $\mathbf{v}$ to the plane containing the face. For each vertex, the geometric error is calculated as the sum of quadrics of neighboring faces weighted by face area A_i :

$$Q(\mathbf{v}) = \sum Q^i(\mathbf{v}) \cdot A_i. \quad (\text{Eq. 3.1})$$

In a 3D space, a plane equation can be represented as $\mathbf{n}^T \mathbf{p} + d = 0$ where $\mathbf{n}$ is a unit normal of the plane. The squared distance from vertex $\mathbf{v}$ to the plane is

$$Q(\mathbf{v}) = (\mathbf{n}^T \mathbf{v} + d)^2 = \mathbf{v}^T (\mathbf{n} \mathbf{n}^T) \mathbf{v} + 2(d \mathbf{n}^T) \mathbf{v} + d^2, \quad (\text{Eq. 3.2})$$

which can be rewritten as a quadratic function $\mathbf{v}^T \mathbf{A} \mathbf{v} + 2\mathbf{b}^T \mathbf{v} + c$ where $\mathbf{A}$ is a symmetric 3x3 matrix, $\mathbf{b}$ is a column vector of size 3, and c is a scalar [8]. Thus, the geometric error of each vertex is represented as a quadric $Q(\mathbf{v}) = F(\mathbf{A}, \mathbf{b}, c)$.

After each edge collapse $(\mathbf{v}_i, \mathbf{v}_j) \rightarrow \mathbf{v}_n$, vertex $\mathbf{v}_n$ is assigned a new quadric that is equal to the sum of $Q_i(\mathbf{v})$ and $Q_j(\mathbf{v})$, i.e. $Q_n(\mathbf{v}) = Q_i(\mathbf{v}) + Q_j(\mathbf{v}) = (Q_i + Q_j)(\mathbf{v})$ where $(Q_i + Q_j) = F(\mathbf{A}_i + \mathbf{A}_j, \mathbf{b}_i + \mathbf{b}_j, c_i + c_j)$. The position of vertex $\mathbf{v}_n$ is determined by minimizing $Q_n(\mathbf{v})$, which is obtained by solving the linear system $\mathbf{A} \mathbf{v}_n = -\mathbf{b}$.

3.2.2 Simplifying Models with Attributes

In the original QEM model, Garland et al. only considers geometric error. In their later work, they introduced a natural extension of the QEM model to include surface attributes [9]. Their idea is to generalize the distance of a vertex to a plane from R^3 to R^{m+3} where m is the number of additional attributes. The new error model can be rewritten in a form of a quadric $Q(\mathbf{v}) = \mathbf{v}^T \mathbf{A} \mathbf{v} + 2\mathbf{b}^T \mathbf{v} + c$ where $\mathbf{A}$ is an $(m+3) \times (m+3)$ matrix, $\mathbf{b}$ is an $(m+3)$ -vector, and c is a scalar. The size of the quadric Q would grow non-linearly with the number of additional attributes, and the total number of coefficients needed to store the quadric Q is $(m+4)(m+5)/2$. When we consider the texture attributes only, i.e. $m = 2$, it needs totally 21 coefficients to store each quadric. Another drawback of the natural extension is that when considering more attributes, the time complexity also grows non-linearly as to solve the linear system $\mathbf{A} \mathbf{v}_n = -\mathbf{b}$. The advantage of this approach is that when solving the linear system; it obtains the new position as well as the new attributes at the same time.

In [10], Hoppe introduced a new extension to the original QEM model by considering geometric errors and attribute errors separately $Q(\mathbf{v}) = Q_{geo}(\mathbf{v}) + Q_{att}(\mathbf{v})$. They first project the vertex $\mathbf{v}(v_{geo}, v_{att})$ onto its neighboring surface to get its projection $\mathbf{v}'(v'_{geo}, v'_{att})$. The geometric error is the squared distance from v_{geo} to v'_{geo} in R^3 , and the

attribute error is the squared deviation between v_{att} and v'_{att} . By separating geometric errors and attribute errors, the number of coefficients to store each quadric Q is reduced to $11 + 4m$ [10]. In case of texture mapping, the number of coefficients needed is only 19. Although this method can reduce the number of coefficients required, simplified models generated by Garland's method have a better vertex distribution than Hoppe's method.

Both the approaches can find the position and attributes of the new vertex at the same time. However, the resultant attributes can have invalid values, i.e. they can have some out-of-bound values. To ensure all the attribute values to be valid, we calculate all the attribute values explicitly.

3.3 Memory Efficient Simplification Based on QEM

In this system, although we consider the case of texture mapping only, our method can also be applicable to other types of attributes such as vertex colors and surface normals. The inputs of our system are a 3D model and its texture parameterization. Besides geometric data including vertex positions (x, y, z) and triangles (j, i, k) , each vertex contains an additional attribute, i.e. texture coordinate (u, v) .

To minimize the memory requirement of quadric error metrics, instead of minimizing the summation of geometric errors and attribute errors, we minimize the geometric errors first and then minimize the attribute errors later. We can consider it as a process of one edge collapsing followed by a process of texture parameterization.

An important part of a simplification algorithm is to find a cost model for each edge collapse and solve the minimization problem for the cost model. When considering model attributes, the cost model is considered as a sum of a geometric error and an attribute error, i.e. $Q_v = Q_{geo} + \lambda Q_{att}$ where λ is a weighted value. In [9], the relationship between the geometric error Q_{geo} and the attribute error Q_{att} is unknown. In [10], even though they specify the relationship explicitly, the value of the weighted value λ has

Figure 3.1: Edge collapse operation $(V_3, V_4) \rightarrow V_n$ in R^3 .

to be specified manually. The resultant values of attributes in both approaches can be out-of-bound as they are solved at the same time as finding the optimal position of the new vertex.

With those considerations, we propose an approach to solve the problem of finding λ and handling invalid values of attributes. The general idea is to find the minimum of geometric errors and then determine the attribute values that result in the minimal attribute errors.

3.3.1 Geometric Error Minimization

Our approach uses the edge collapse technique as the simplification operator as shown in Fig. 3.1. The edge $E(V_3, V_4)$ is collapsed to a single vertex V_n , and the collapse eliminates two incident triangles and three adjacent edges. It also generates a new vertex V_n and the main issue here is how to find the optimal position of the new vertex. To compute the cost of each edge collapse, we use the QEM model introduced in [8]. As

Figure 3.2: Local region affected after each edge collapse in R^3 .

in Fig. 3.1, the cost of edge $E(\mathbf{V}_3, \mathbf{V}_4)$ is the sum of geometric errors of two vertices $(\mathbf{V}_3, \mathbf{V}_4)$, in other words

$$C_{E(\mathbf{V}_3, \mathbf{V}_4)} = Q(\mathbf{V}_3) + Q(\mathbf{V}_4), \quad (\text{Eq. 3.3})$$

where $C_{E(\mathbf{V}_3, \mathbf{V}_4)}$ is the error incurred when removing the edge $E(\mathbf{V}_3, \mathbf{V}_4)$. We can find the optimal position of the new vertex by minimizing the cost function $C_{E(\mathbf{V}_3, \mathbf{V}_4)}$. If the optimal position is not found, the mid-point of edge $E(\mathbf{V}_3, \mathbf{V}_4)$ is used instead [8].

The optimal position of the new vertex ensures that the geometric error of the edge collapse $(\mathbf{V}_3, \mathbf{V}_4) \rightarrow \mathbf{V}_n$ is minimum. And the number of coefficients to store each quadric in this process is only 10.

3.3.2 Attribute Error Minimization

After finding the optimal position of the new vertex to minimize the geometric errors, we then find the optimal attribute values to minimize the attribute errors caused by the

CHAPTER 3. PROGRESSIVE MESH GENERATION USING MEMORY-EFFICIENT SIMPLIFICATION

edge collapsing.

When an edge is collapsed to a new vertex, it affects the local region of the surface as shown in Fig. 3.2. The affected surface region is displayed in dash lines when collapsing the edge $E(\mathbf{V}', \mathbf{V}'')$ to create a new vertex $\mathbf{V}_n$. In the geometric space R^3 , only the position of vertex $\mathbf{V}_n$ is affected. However, in the attribute space R^2 the texture coordinates of vertices $\mathbf{V}_1, \mathbf{V}_2, \dots, \mathbf{V}_7, \mathbf{V}_n$ will be changed to minimize the texture distortion caused by the edge collapse.

We can consider the minimization of texture distortion as a texture parameterization problem. Here we have to find the new texture coordinates of vertices $\mathbf{V}_1, \mathbf{V}_2, \dots, \mathbf{V}_7, \mathbf{V}_n$. In [11], Sander et al. considers the problem of finding new texture coordinates as an energy minimization problem. This approach can produce a very good result; however it requires a long processing time. This drawback makes it impractical for some applications in small and medium devices.

With the observation that the geometry outside the region $\mathbf{V}_1, \mathbf{V}_2, \dots, \mathbf{V}_7$ is unchanged, we can consider the texture coordinates outside this region unaffected. We only have to construct a linear system representing the relationship within the region $\mathbf{V}_1, \mathbf{V}_2, \dots, \mathbf{V}_7$ and solve the linear system to find a new set of texture coordinates. We use the mean value coordinates [29] to form the linear system. Although we can also use other methods, using mean value coordinates is a better choice as it can handle non-convex regions.

The linear system is a set of equations showing the relationship between a vertex and its neighboring vertices in the texture space R^2 . Considering vertex $\mathbf{V}_n$ in Fig. 3.2, this vertex is surrounded by vertices $\mathbf{V}_1, \mathbf{V}_2, \dots, \mathbf{V}_7$. There is a relationship between the texture coordinates (U_n, V_n) of vertex $\mathbf{V}_n$ and texture coordinates of its surrounding vertices:

$$U_n = \frac{\sum_{i=1}^7 \omega_i U_i}{\sum_{i=1}^7 \omega_i}, \quad \text{and} \quad V_n = \frac{\sum_{i=1}^7 \omega_i V_i}{\sum_{i=1}^7 \omega_i}, \quad (\text{Eq. 3.4})$$

Figure 3.3: Construction of mean value coordinates in R^3 .

where (U_n, V_n) is the texture coordinate of vertex V_n , (U_i, V_i) is the texture coordinate of neighboring vertex V_i , and ω_i is a mean value coordinate representing the relationship between V_n and V_i .

We use the formula introduced by Floater [29] to calculate the value of ω_i . In Fig. 3.3, the mean value coordinate between vertex V_n and vertex V_5 is found using

$$\omega_5 = \frac{\tan \frac{\alpha}{2} + \tan \frac{\beta}{2}}{r_{n5}}. \quad (\text{Eq. 3.5})$$

We can use the same process to construct equations for other vertices $V_1, V_2, \dots, V_7$ to form a linear system. Solving that linear system gives a set of new texture coordinates, which generate a new mapping from the 2D attribute space to the 3D geometric space.

In [30, 31], the mean value mapping can be categorized in the group of discrete harmonic mappings. This mapping type can minimize the Dirichlet energy of the mapping. In other words, when mapping a triangle from the attribute space to the geometric space, it will cause distortion to the angles of the triangle. Using the mean value mapping can minimize the angle distortion.

3.3.3 Attribute Discontinuities

When generating new texture coordinates, we assume that vertex attributes are continuous all over the surface. In other words, every vertex is assumed to have a single attribute value. However, there are some vertices that have more than one attribute values such as boundary vertices, which are caused by discontinuities in the attribute space. We treat these special vertices differently. In [9], Garland et al. suggested one solution that is to use boundary constraints during the simplification, i.e. they add a large penalty error to each boundary edge. However, this suggestion may not work in all the cases.

In [10], Hoppe introduced a wedge-based structure to handle the case where a vertex may have multiple attribute values. They partition each complex vertex as a set of wedges, in which each wedge is a set of incident faces of the vertex and the vertex has the same attribute value within each set. The quadric of a complex vertex is the sum of all wedge quadrics and each wedge quadric is the area-weighted sum of all face quadrics in the wedge set:

$$Q^v(\mathbf{p}, s^1, \dots, s^k) = \sum_i Q^{wi}(\mathbf{p}, s^i), \quad (\text{Eq. 3.6})$$

where $Q^v(\mathbf{p}, s^1, \dots, s^k)$ is the quadric of vertex v , Q^{wi} is the quadric of wedge i , and $s^1, \dots, s^k$ are k attributes of vertex v . To find the new vertex position and all the wedge attributes, they have to minimize the quadric $Q^v(\mathbf{p}, s^1, \dots, s^k)$. Because each complex vertex has a constraint, i.e. all the wedges share the same 3D vertex position, minimizing this quadric Q^v is a difficult problem.

In our approach, when there are some vertices consisting of multiple attribute values, we first divide each complex region into a set of simple region where each vertex within a simple region has only one attribute value. We then flatten each simple region separately using the boundary-free parameterization method proposed by Desbrun et al. [32]. Fig. 3.4 illustrates one example of such a region where $V_2, V_3, \dots, V_7$ each have multiple

Figure 3.4: Edge collapse operation $(V_4, V_5) \rightarrow V_n$ at a complex region in R^3 .

attributes. For each edge collapse operation $(V_4, V_5) \rightarrow V_n$, we first perform the geometric error minimization to find the new geometric position of V_n . To find multiple attribute values of V_n , we have to divide the complex region $(V_2, V_1, V_0, V_7, V_8, \dots, V_{14})$ around V_n into simple regions, i.e. $R_1(V_0, V_1, V_2, V_3, V_n, V_6, V_7)$ and $R_2(V_2, V_3, V_n, V_6, V_7, \dots, V_{14})$. There are two important properties of these simple regions: they are boundary regions, and each vertex inside these regions has only one attribute. We then use Desbrun's method to minimize distortion of texture mapping of each simple region. We then obtain all attribute values of those multiple attribute vertices.

3.4 Experimental Results

We carried out the experiment with three 3D models: the horse model with 12502 vertices, the bull model with 14002 vertices, and the camel model with 15002 vertices. The memory requirement of each model is presented in Table 3.1. In the experiment, we used the floating point type double to store each coefficient. Memory storage presented in

CHAPTER 3. PROGRESSIVE MESH GENERATION USING MEMORY-EFFICIENT SIMPLIFICATION

Table 3.1 is measured in megabit (Mb). While Hoppe’s method can reduce the memory requirement to 90.5% as compared to Garland’s method, our method can reduce further to 47.6% as compared to the natural extension, i.e. we can save about 52.4% of memory storage. Besides texture attributes, our method can be applied for other types of attributes (colors, surface normals) without increasing memory requirement. This improvement is significant for mobile applications where the memory resource is very limited.

Method	Memory Needed (Mbits)			In Percentage
	Horse 12502 vertices	Camel 15002 vertices	Bull 14002 vertices	
In [9]	16.02 Mb	19.23 Mb	17.95 Mb	100%
In [10]	14.50 Mb	17.40 Mb	16.24 Mb	90.5%
Ours	7.63 Mb	9.16 Mb	8.55 Mb	47.6%

Table 3.1: The results of memory requirement.

To evaluate the visual quality of results of our method, we carried out the experiment for four cases:

- Calculating the new texture coordinates of V_n using a simple averaging function: for an edge collapse $(V_1, V_2) \rightarrow V_n$, the new texture coordinates of V_n is calculated as $T_n = (T_1 + T_2)/2$ where T_1 , T_2 , and T_3 are texture coordinates of V_1 , V_2 , and V_n accordingly.
- Applying the mean value coordinate to calculate the new texture coordinate of V_n only, and keeping texture coordinates of neighboring vertices of V_n unchanged
- Applying the mean value coordinate to calculate the new texture coordinate of V_n and its neighbors
- Calculating the new texture coordinate of V_n using Garland’s approach

3.5.a: Updating V_n only 3.5.b: Updating V_n and its neighbors

Figure 3.5: Close view of the simplified model.

The implementation of Garland’s method can be found in a mesh processing system called meshlab [33]. We simplified the house model down to 10% of the original size and presented the experiment results in Fig. 3.6. The top model is the original model to be simplified. Fig. 3.6.c shows the simplified model using a simple averaging approach. Because this approach does not minimize texture distortion, the quality of the result is very poor. Fig. 3.6.d shows the result of our first approach, where only the texture coordinate of V_n is updated. Fig. 3.6.e shows the result of our second approach, where the texture coordinates of V_n and its neighbors are updated. As shown in Fig. 3.5, the white lines in our second approach are smoother than the ones in our first approach. This is because texture coordinates of the neighboring vertices of V_n are also updated to minimize the texture distortion. The result of Garland’s approach is shown in Fig. 3.6.b. The experiment results show that our method can achieve an equivalent level of quality as Garland’s method. We can see that 2D texture images are fitted smoothly on the 3D model surface because texture distortion is minimized during the simplification process.

More experimental results for the bull and camel model are presented in Fig. 3.7 and Fig. 3.8 respectively.

3.5 Discussions

The main advantage of our method is its memory efficiency. Our method can save up to 52.4% of memory requirement as compared to the natural extension of the quadric error metric model. The experiment results show that our method can attain the equivalent level of visual quality as compared to the method in [9]. While previous methods find the position and attribute values of a new vertex simultaneously by minimizing the summation of both geometric errors and attribute errors, our method finds the optimal position of a new vertex first to minimize geometric errors and then find the optimal attribute values to minimize attribute errors. By doing so, we can also solve the out-of-range problem, which is found in the previous two methods [9, 10].

Our current method uses different texture parameterization methods to minimize texture distortion, i.e. we use the mean value coordinate method to find the texture coordinates of inner vertices and use the boundary-free parameterization method to find the coordinates of those on the boundary. At the moment, there is no single parameterization method that could handle both boundary regions and non-convex neighboring regions; therefore we combine different methods in our solution. Using different texture parameterization methods could affect the visual quality at boundary vertices. It would be interesting to develop a single texture parameterization method to be able to flatten both inner vertices and boundary vertices, which would lead to further performance improvement.

CHAPTER 3. PROGRESSIVE MESH GENERATION USING MEMORY-EFFICIENT SIMPLIFICATION

3.6.a: Original

3.6.b: Garland [9]

3.6.c: Averaging

3.6.d: Updating v_n only

3.6.e: Proposed

Figure 3.6: Simplifying the *horse* model of 25000 faces to 2500 faces.

CHAPTER 3. PROGRESSIVE MESH GENERATION USING MEMORY-EFFICIENT SIMPLIFICATION

3.7.a: Original

3.7.b: Garland [9]

3.7.c: Averaging

3.7.d: Updating v_n only

3.7.e: Proposed

Figure 3.7: Simplifying the *bull* model of 28000 faces to 6000 faces.

CHAPTER 3. PROGRESSIVE MESH GENERATION USING MEMORY-EFFICIENT SIMPLIFICATION

3.8.a: Original

3.8.b: Garland [9]

3.8.c: Averaging

3.8.d: Updating v_n only

3.8.e: Proposed

Figure 3.8: Simplifying the *camel* model of 30000 faces to 7000 faces.

Chapter 4

Progressive 3D Reconstruction from Multi-view Images

In the previous chapter, we discussed a simple case to generate progressive meshes, i.e. simplifying a given 3D model. In this chapter, we consider a more challenging case, i.e. generating progressive 3D meshes from a set of 2D images. Fig. 4.1 shows the system flowchart. The whole system consists of several main modules, each of them will be discussed separately in the following sections.

The rest of the chapter is organized as follows. We give a system overview in Section 4.1. The process to generate the initial mesh is discussed in Section 4.2. We discuss the optimization of the initial mesh in Section 4.3 and the generation of the progressive mesh in Section 4.4. How to recover the texture information of the mesh is described in Section 4.5. Finally, Section 4.6 shows the experimental results.

4.1 System Overview

Our work addresses the problem of wide baseline reconstruction. The primary inputs of our system are a set of n segmented color images $I = \{I_i | i = 1 \cdots n\}$ and a set of associated projection matrices P_i . The cameras are placed in a circle in which the 3D object is at the center. All the cameras are calibrated to obtain the projection matrices when

CHAPTER 4. PROGRESSIVE 3D RECONSTRUCTION FROM MULTI-VIEW IMAGES

Figure 4.1: System flowchart.

capturing the input images. In addition, we assume that the surface to be reconstructed is a Lambertian surface, i.e. the brightness of the surface is independent of the observer’s angle of view, which is a common assumption for most algorithms [12]. The output of our system is the reconstructed 3D object represented by triangular meshes. The mesh representation is a very popular format since it facilitates efficient computations on many local properties such as normal vectors, mean curvature, etc.

Our objective is to implement a rate-distortion optimized progressive 3D reconstruction. Specifically, given a certain reconstruction rate, we want to achieve the minimum reconstruction distortion. On the other hand, given a certain reconstruction distortion,

we want to find the smallest reconstruction rate. Similar to [34], we define the reconstruction rate as the number of vertices used for the reconstructed 3D model. The definition of the distortion will be discussed in detail in Section 4.3.1.

Our system consists of the following three major steps:

- *Step 1: Initial mesh generation.* Use the method in [20] to reconstruct a visual hull from a set of segmented images together with the projection matrices. The visual hull is then triangulated to obtain a surface representation by using Poisson surface approach [2].
- *Step 2: PDE-based mesh optimization.* Optimize the input mesh using the PDE-based surface flow technique that is similar to [4, 16, 23]. The output of the mesh optimization is an optimized mesh with the same number of vertices as the input. The mesh is optimal in the sense that it achieves the minimal distortion at the given rate.
- *Step 3: Progressive mesh refinement using local adaptive subdivision.* Repeatedly use local subdivision to add new vertices to the most distorted triangle if the target rate has not been met. After each refinement, go back to Step 2 to optimize the newly added vertices and their neighboring vertices.

4.2 Initial Mesh Generation

We generate a very simple visual hull using the method proposed in [20] with a set of segmented color images and their associated projection matrices. In particular, each color image is pre-segmented into a foreground/background binary image called a silhouette image. Combined with its associated projection matrix, each silhouette image defines a silhouette cone. The intersection of all the silhouette cones forms a set of voxels, which is the visual hull.

4.2.a: volume elements

4.2.b: triangulated surface

Figure 4.2: Illustration of volume elements (voxels) and triangulated surface of a visual hull.

The visual hull is then triangulated to obtain the the surface mesh representation using either the marching cube technique [21] or the Poisson surface approach [2]. We choose the Poisson surface approach as its triangulated surface is smoother. An example of a visual hull and the corresponding triangulated surface is shown in Fig. 4.2.

Because the triangulated surface obtained from the Poisson approach is irregular, a remeshing process [3] is further applied to obtain a regular surface representation. This regularization is necessary because the subsequent PDE-based surface optimization requires a roughly regular mesh to obtain stable results. The size of the output regular mesh is set to a small one so as to have a wide range of distortion levels.

4.3 Mesh Optimization Using PDE-based Surface Flow

Our main goal is to reconstruct 3D models in a rate-distortion efficient way. In other words, given the initial coarse reconstruction with a small number of vertices, it is needed to optimize the positions of all the vertices in order to achieve the minimum geometry

distortion at the current rate. Typically, global optimization is applied for such a surface optimization. In our work, we adopt the state-of-the-art partial differential equation (PDE) based surface flow method [4, 16, 23] with our improved distortion measurement to optimize the coarse mesh. The optimization is not only applied to the coarse mesh, but also to each of the subsequent refined meshes. The PDE-based surface flow method has been introduced in Section 2.3. Here, we focus on the key factor in the PDE equation, i.e. distortion factor G .

4.3.1 Distortion Measurement

One important issue in the PDE system (Eq. 2.3) is to define the function $G(\mathbf{p})$ that measures the geometry distortion of point $\mathbf{p}$ on the reconstructed surface. Since the reconstructed 3D surface is derived from multi-view images and our system operates in a coarse-to-fine manner, the best way to measure the geometry distortion of a surface point is through measuring the color inconsistency of the projection image points in all the visible views. Particularly, we use a similar photo-consistency function as that in [35], i.e.

$$F(\mathbf{p}) = \frac{1}{K-1} \left(\sum_{i=1}^K C_i^2(\mathbf{p}) - \frac{1}{K} \left(\sum_{i=1}^K C_i(\mathbf{p}) \right)^2 \right), \quad (\text{Eq. 4.1})$$

where K is the number of cameras that are visible to point $\mathbf{p}$, and $C_i(\mathbf{p})$ is the color of the projection point on image i for $\mathbf{p}$. Eq. 4.1 is essentially the unbiased estimation of the color variance among different projection points on different images.

Because the surface update operates on vertices, what we need is the distortion measurement for each vertex. Directly using the photo-consistency measure $F(\mathbf{v})$ as the vertex distortion is inappropriate since it does not represent the local surface distortion around vertex $\mathbf{v}$. Thus, to better approximate the surface distortion, we should consider the photo-consistency values of all the sampling points in a small neighboring region around vertex $\mathbf{v}$ for the vertex distortion measure.

Clearly, the vertex distortion depends on two factors: the sampling method and the sampling resolution. There are two major sampling methods [16, 4] reported in literature. In particular, Duan et al. [4] proposed to sample around a small patch on the tangent plane of a vertex. However, sampling on the tangent plane of each vertex does not provide a good approximation to high-curvature regions. An alternative method used in [16] is to choose sample points on an orthogonal lattice of each surface triangle. But for obtaining the photo-consistency value of a vertex, only the three closest adjacent samples are used to compute the barycentric value for the vertex.

To better approximate the surface distortion, in this work, we propose to uniformly sample all the triangles. In particular, for a triangle $(\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3)$, a set of sampling points are generated according to

$$\mathbf{p} = \mathbf{v}_1(1 - m) + \mathbf{v}_2m(1 - n) + \mathbf{v}_3mn, \quad (\text{Eq. 4.2})$$

where $m = \overline{0 : 1}$ and $n = \overline{0 : m}$. By adjusting the step sizes (dm and dn) of m and n , we can control the number of sampling points on each triangle. An example of our sampling method is illustrated in Fig. 4.3.

In addition to choosing the sampling method, we also need to determine the sampling resolution. Duan et al. [4] did not address this issue in their work. A method was suggested in [16, 36], which determines the sampling resolution according to the image resolution. However, this method requires back projection of each image pixel onto the 3D surface in order to find a good estimation of the sampling resolution, where back projection from a 2D space to a 3D space is a time-consuming process.

In our work, we propose a more efficient method to relate the sampling resolution with the image resolution. Particularly, for each triangle on the surface, we forward-project the triangle onto all visible image planes. We calculate the area A_i of each projected

Figure 4.3: An example of sampling around one vertex.

triangle and derive the sampling resolution as

$$dm = dn = \frac{1}{\max\left\{\left\lfloor \sqrt{2 \times f(A_1, A_2, \dots, A_n)} \right\rfloor - 2, 1\right\}}, \quad (\text{Eq. 4.3})$$

where A_i is computed as the number of pixels in each projected triangle, and $f(A_1, A_2, \dots, A_n)$ is a function to control the relationship between the sampling resolution and the image resolution. In this work, we set $f(A_1, A_2, \dots, A_n) = C \times \text{average}(A_1, A_2, \dots, A_n)$, where C is a control factor.

We are now ready to define the distortion measurement. First, we define the distortion of a triangle T_i as the weighted summation of the photo-consistency values of all the sample points in the triangle, i.e.

$$\varphi(T_i) = \sum_{\mathbf{p} \in T_i} \omega_{\mathbf{p}} F(\mathbf{p}), \quad (\text{Eq. 4.4})$$

where $\omega_{\mathbf{p}} = 1/\text{valence}(\mathbf{p}), 1/2, 1$ for $\mathbf{p}$ being a vertex, a sampling point on an edge, and an inner sampling point, respectively. The weight $\omega_{\mathbf{p}}$ is to take into account the repeated counting of the photo-consistency values of the same boundary samples in multiple triangles. Then, the distortion for a vertex is defined as the average distortion of all the

samples in the one-ring neighboring triangles, i.e.

$$G(\mathbf{v}) = \frac{1}{|N_s(\mathbf{v})|} \sum_{T_i \in N_t(\mathbf{v})} \varphi(T_i), \quad (\text{Eq. 4.5})$$

where $N_t(\mathbf{v})$ is the set of one-ring neighboring triangles, $|N_s(\mathbf{v})|$ is the number of neighboring sampling points in $N_t(\mathbf{v})$. The overall surface distortion is measured by the average per-sample distortion:

$$\Psi = \frac{1}{|N_s|} \sum_i \varphi(T_i), \quad (\text{Eq. 4.6})$$

where $|N_s|$ is the total number of samples. Using the per-sample distortion makes the distortion measurement independent of the number of samples in the cases of dense samples.

4.3.2 Computing Distortion Gradient

Another question we need to answer is how to compute the distortion gradient $G_{\mathbf{N}}$ in Eq. 2.3. In this work, we use the central difference to compute the distortion gradient as [16]. Specifically, $G_{\mathbf{N}}$ is computed as the difference between the distortion values of the current vertex when it moves forward and backward along the normal direction:

$$\nabla G_{\mathbf{N}} = \frac{G(\mathbf{v} + \eta \Gamma_{min} \mathbf{N}) - G(\mathbf{v} - \eta \Gamma_{min} \mathbf{N})}{2\eta \Gamma_{min}}, \quad (\text{Eq. 4.7})$$

where Γ_{min} is the minimum edge length of the surface and $\eta \in (0, 1]$ is a control factor. In practice, η is chosen to be small enough so that the small displacement of the current vertex does not affect the visibility and normal vectors of other vertices. Note that when calculating $G(\mathbf{v} \pm \eta \Gamma_{min} \mathbf{N})$, we need to re-sample one-ring neighborhood triangles around the current vertex.

4.3.3 Mesh Regularization

Although the surface flow method moves each vertex closer to the true surface, it degrades the regularity of the mesh. To ensure numerical stability, we need to maintain the mesh

regularity in terms of vertex distribution and edge length ration (the ratio between the maximal and the minimum edge lengths). In our implementation, we use the similar tangential Laplacian operator as that in [37] to regularize the mesh:

$$\frac{\partial \mathbf{p}}{\partial t} = \left(\frac{1}{k} \sum_{i=1}^k \mathbf{v}_i - \mathbf{p} \right) - \left(\left(\frac{1}{k} \sum_{i=1}^k \mathbf{v}_i - \mathbf{p} \right) \cdot \mathbf{N} \right) \mathbf{N}, \quad (\text{Eq. 4.8})$$

where $\mathbf{v}_i$ is a neighboring vertex of the current vertex $\mathbf{p}$, and k is the number of neighboring vertices. The time step ∂t is chosen to ensure the displacement of the current vertex to be bounded by the minimum edge length.

4.3.4 Solving PDE-based System

The surface deformation process contains three main steps, which are repeated until no more distortion reduction is observed:

- Determine the visibility of vertices using the z-buffering technique [38] as in OpenGL.
- Update vertex positions using Eq. 2.3.
- Regularize vertices using Eq. 4.8.

The PDE system produces a coarse mesh that most agrees with all the input images given the current number of vertices. The coarse mesh is used as a starting point for the rate-distortion optimization process that is discussed in the next section.

4.3.5 Boosting Performance by Using GPU

To reconstruct a highly textured model, we have to use high-resolution sampling method, i.e. the sampling resolution of each triangle is set to be approximate to image resolution. Hence, when the input images have high resolution, the number of sampling points on the surface increases in the power of two. The dense sampling becomes the bottle-neck to the performance of the system. Especially, when computing the evolution speed of

each vertex, we have to compute the photo-consistency function and the gradient of the photo-consistency function of each vertex, which requires sampling one-ring neighborhood triangles around each vertex.

In our system, we propose a method to solve this problem by using GPUs as similar to the work of [39] to enable the parallel processing capability for our system, i.e. we can make the system performance independent of the sampling resolution on each triangle. So to compute the photo-consistency value and the gradient value of each vertex, we process all the one-ring neighborhood triangles of each vertex in parallel. Based on the observation that each sample on the triangle can be processed independently, we define an array of all the sampling points and send them to the GPUs for parallel processing. In this way, all the sampling points can be processed at once, hence the performance of the system is mostly independent of the sampling resolution of each triangle.

4.4 Progressive Mesh Refinement via Local Adaptive Subdivision

After applying the mesh optimization to the initial coarse mesh that consists of a small number of vertices, we obtain an optimized coarse mesh. To further reduce the overall surface distortion, more vertices need to be added. Here, the refinement process comes into play.

A common way to refine the mesh is to apply global subdivision to increase the resolution of the mesh. However, such a way does not lead to an optimal solution for rate-distortion efficient 3D reconstruction. This is because a real object generally does not have a uniform detail distribution over its surface. Some regions may have more details than others. Using a global refinement method increases vertices uniformly over the entire surface. As a result, some regions with less details are relatively over-refined and the regions with more details are relatively under-refined.

CHAPTER 4. PROGRESSIVE 3D RECONSTRUCTION FROM MULTI-VIEW IMAGES

Therefore, a good solution is to give higher rates to the regions with more details. In other words, more vertices should be added to the regions with rich details or large distortions. In this work, we consider each triangle as a region and keep adding new vertices to the most distorted triangle until reaching the target rate. Our proposed refinement not only achieves rate-distortion efficient 3D reconstruction at the target rate but also facilitates the generation of an optimal progressive mesh, where the 3D reconstruction is optimal at any rate between the rate for the coarse mesh and the target rate.

In particular, we first compute the distortions of all the triangles according to (Eq. 4.4) to select the most distorted triangle. Then, we apply a local subdivision scheme similar to [40] on the selected triangle. The subdivision scheme introduces three new vertices in the middle of the three triangle edges and links them together. To maintain the manifold topology, three triangles adjacent to the current triangle are subdivided accordingly. Fig. 4.4 shows one example, where the triangle ABC is subdivided by adding three new vertices (N, M, L) and the dash lines indicate newly created edges.

Figure 4.4: An example of the local adaptive subdivision scheme.

After each local subdivision, we apply the mesh optimization method introduced in Section 4.3 to optimize the positions of the newly added three vertices. Considering that the displacements of the new vertices affect their local regions, we also update neighboring vertices of the three new vertices. For example, for the case in Fig. 4.4, in addition to optimizing the positions of the three vertices $\{N, M, L\}$, we also update their six neighboring vertices $\{A, B, C, D, E, F\}$. The updating stops when the distortion of any of the nine vertices cannot be further reduced. Note that besides updating the neighboring vertices we could also update all the vertices, but it would significantly increase the computational complexity. We would also like to point out that, before the mesh optimization, an edge swapping technique [41] is used to mitigate the mesh irregularity caused by the local subdivision. This is to ensure the numerical stability of the PDE-based mesh optimization.

4.5 Texture Fitting

After reconstructing a geometry mesh which is close to the true object, we are now ready to discuss how to recover and represent texture information from multi-view images. There are two main issues in texture recovery, i.e. how to combine information from multiple sources and how to represent the combined information in a discrete form.

In the case of multi-view images, each point on the surface is seen from more than one image. In other words, there are a number of colors for each point on the surface. A typical blending function is defined as following:

$$\mathbf{C}_i = \sum_{j=1}^K \vartheta_j(i) \times \omega_j(i) \times \tau_j(i), \quad (\text{Eq. 4.9})$$

where $\mathbf{C}_i$ is the final color of point i , and K is the number of cameras. $\vartheta_j(i)$ is the visibility of point i in camera j , which is equal to zero if the current point is not seen from camera j , and one otherwise. $\tau_j(i)$ is the color information from camera j . $\omega_j(i)$

CHAPTER 4. PROGRESSIVE 3D RECONSTRUCTION FROM MULTI-VIEW IMAGES

is a spatial blending weight depending on the orientation of point i with respect to camera j . With the assumption that the object surface is close to Lambertian, it is valid to compute spatial blending weights based on only the orientation of each point with respect to each camera. An important property of each blending weight is that it decreases when increasing the angle $\theta_j(i)$ between the current point normal and the viewing vector towards the camera from the current point.

A simple way to compute a spatial blending weight for point i and camera j , $\omega_j(i)$, is to apply the reciprocal of the angle $\theta_j(i)$, i.e. $\omega_j(i) = 1/\theta_j(i)$. The drawback of this method is that fine details in original cameras are blurred in the combined texture. To obtain a better control over the influence of each camera on the appearance of the final texture, Carranza et al. [42] proposed alternative spatial blending weights:

$$\omega'_j(i) = \frac{1}{(\max(\omega_j(i)) + 1 - \omega_j(i))^\alpha}. \quad (\text{Eq. 4.10})$$

This blending function assigns higher priorities to the camera having a smaller angle $\theta_j(i)$. The sharpness value α controls the degree of priorities, i.e. the larger the value of α , the higher the priorities to small angles.

In our work, we use a more intuitive weighting function to blend colors from multiple cameras:

$$\omega_j(i) = \cos(\theta_j(i))^\alpha. \quad (\text{Eq. 4.11})$$

The behavior of this weighting function also assigns higher priorities to smaller angles, in which the degree of the priorities is controlled by the sharpness value α .

Since the surface is continuous, it is not possible to store texture information for all points on the surface. Thus, the surface has to be converted into a discrete form first, i.e. finding the discrete representation of the surface. Considering a surface is comprised of a set of triangles, representing a surface is equivalent to representing a set of triangles. Esteban et al. [43] proposed a method to represent each triangle. It first maps each

CHAPTER 4. PROGRESSIVE 3D RECONSTRUCTION FROM MULTI-VIEW IMAGES

triangle to a 2D triangle in an image, and each 2D triangle is represented as a set of pixels. Once color values of all pixels in each 2D triangle are found, all 2D triangles are packed together in an image. The drawback of this method is that it uses the same size for all 2D triangles, and packing all 2D discrete triangles in an image is a complex task.

In our work, we propose a more efficient method with low complexity to represent texture information of a surface without packing process. There are two main steps in our proposed method. First, we flatten the surface into a 2D space, which is called the texture space. Second, we fill the flattened surface using our spatial blending weights.

The process to flatten a surface into a texture space is called a mesh parameterization process. There have been a large number of mesh parameterization techniques proposed in literature. A more concrete introduction to mesh parameterization techniques can be found in two recent surveys [44, 45]. In our work, we use a parameterization technique proposed by Sheffer et al. [46], which is known as the angle based flattening technique. The advantage of this technique is that it can avoid flip-over problems. An example of a flattened surface is shown in Fig. 4.5. The implementation of the angle based flattening technique can be found in Graphite software (<http://alice.loria.fr/software>).

Once the texture map of the surface is created, it is filled with colors from different images. The color-filling process applies to each triangle. For each triangle, it scans each pixel, finds the 3D corresponding point of the pixel, and uses the blending function to find the final color of the pixel. The correspondence between a 2D pixel and a 3D point is found using barycentric coordinates (α, β, λ) , as illustrated in Fig. 4.6:

$$\alpha = \frac{A_M}{A_M + A_N + A_P}, \quad \beta = \frac{A_N}{A_M + A_N + A_P}, \quad \lambda = \frac{A_P}{A_M + A_N + A_P}, \quad (\text{Eq. 4.12})$$

where A_M, A_N, A_P are the areas of three triangles (V, P, N) , (V, M, P) , and (V, M, N) respectively.

CHAPTER 4. PROGRESSIVE 3D RECONSTRUCTION FROM MULTI-VIEW IMAGES

Figure 4.5: 2D texture map of a 3D surface.

Figure 4.6: Interpolation of a point in a triangle.

4.6 Experimental Results

We carry out several experiments to evaluate the performance of our proposed 3D reconstruction system. We test on three models: *Venus*, *Pot* and *Bowl*. Each model is captured with 22 cameras uniformly placed in a circle where the model is at the center. The 22 captured color images with a resolution of 900×900 are segmented and calibrated in advance, and then used as the input to our proposed 3D reconstruction system.

Fig. 4.7.a-c shows the visual results of the reconstructed *Venus* models at different stages in our system. It can be seen from Fig. 4.7.b that the optimized coarse mesh obtained by the developed PDE-based mesh optimization approximates well on the region with less details, however it cannot well represent the regions with details due to the rate constraint. With only 2454 vertices, Fig. 4.7.b is the best we can get. Note that the result of Fig. 4.7.b is the one obtained after many optimization iterations. With more vertices being added in, the regions with more details are recovered greatly as shown in Fig. 4.7.c and e. The produced mesh in Fig. 4.7.e looks very close to the original surface (Fig. 4.7.f) in terms of geometry distortion.

We further compare our system with the PDE-based surface flow method [4]. To have a fair comparison, we adjust the setting of the remeshing process in Section 4.2 to generate around the same number of vertices as our case in Fig. 4.7.e and then use the PDE-based surface flow method to perform the vertex optimization, which produces a result in Fig. 4.7.d. In other words, our method consists of three steps and starts from a coarse mesh while the other method only uses the first two steps of our system and reaches the target rate at the beginning. Comparing Fig. 4.7.d and e, we can see that the output of the surface flow method [4] does not capture those regions with more details such as the eyes, the hair, the nose and the mouth of *Venus* as well as that using our system. Our reconstructed model in Fig. 4.7.c has similar visual quality as Fig. 4.7.d but with less number of vertices. The visual comparison results for another two models are

CHAPTER 4. PROGRESSIVE 3D RECONSTRUCTION FROM MULTI-VIEW IMAGES

Figure 4.7: Visual comparison of the reconstructed *Venus* models: (a) initial coarse mesh (generated in the first major step) with 2454 vertices and 4904 triangles, (b) optimized coarse mesh, (c) refined mesh using our proposed system with 7377 vertices, (d) optimized fine mesh using [4] with 9810 vertices, (e) refined mesh using our proposed system with 9837 vertices, and (f) one of the input images.

shown in Fig. 4.8, where similar observations can be made. In addition to showing the visual quality improvement at one rate, we generate the rate-distortion curve for each model and compare the rate-distortion curves of different methods. Fig. 4.9 shows the obtained curves. To obtain the rate-distortion curves for the surface flow method [4], we

CHAPTER 4. PROGRESSIVE 3D RECONSTRUCTION FROM MULTI-VIEW IMAGES

Figure 4.8: Visual comparison of the reconstructed *Pot* (top row) and *Bowl* (bottom row) models: (a) optimized coarse meshes with 5308 vertices and 10612 triangles for *Pot* and 5296 vertices and 10588 triangles for *Bowl*, (b) optimized fine meshes using [4] with 29886 vertices for *Pot* and 10631 vertices for *Bowl*, (c) results of our proposed system with 29884 vertices for *Pot* and 10639 vertices for *Bowl*, and (d) two input images for *Pot* and *Bowl* respectively.

generate a set of initial meshes with different number of vertices, then optimize them and record their distortions at the stable state.

Fig. 4.9 clearly demonstrates the superior performance of our proposed system. Compared with the surface flow method [4], our system achieves lower per-sample distortion at not only one rate but also all the rates except the starting rate. For example, our system reduces the per-sample distortion about 70.5% for *Venus* at around 4204 vertices, 76.8% for *Pot* at around 9403 vertices, 47.5% for *Bowl* at around 8365 vertices. The reason that our system significantly outperforms the surface flow method lies in the proposed rate-distortion optimized local adaptive refinement that facilitates nonuniform vertex distribution among different regions. In other words, our system allocates more vertices to the regions with more details and processes it in a rate-distortion optimized way.

CHAPTER 4. PROGRESSIVE 3D RECONSTRUCTION FROM MULTI-VIEW IMAGES

Figure 4.9: The average per-sample distortions of the reconstructed *Venus* models at different number of vertices.

We would like to point out that although the per-sample distortion is independent of the sampling resolution when the number of samples is large, it does matter in the cases of low sampling resolution. Fig. 4.10 shows the average per-sample distortions of the reconstructed coarse *Venus* model at low sampling resolution under different optimization iterations. We study the case of applying the PDE-based mesh optimization on the initial coarse mesh and compare the results with uniform sampling (the same number of samples per triangle) and the proposed adaptive sampling described in Section 4.3.1. For a fair comparison, we adjust the numbers of sample points for uniform sampling and the adaptive sampling to be roughly the same.

It can be seen from Fig. 4.10 that the proposed adaptive sampling method achieves better distortion curve at different iterations. This is mainly because the proposed adaptive sampling facilitates non-uniform sampling that allocates more samples to the triangles corresponding to more image pixels and thus it gives a better approximation to the surface distortion. One might argue that if using high sampling resolution, the inaccurate surface distortion approximation is not a problem any more. However, high sampling resolution comes at the cost of high computation complexity. With the proposed adaptive sampling, we can achieve the same level of surface distortion approximation but with less

Figure 4.10: The average per-sample distortions of the reconstructed coarse *Venus* model with 2887 vertices and 5770 triangles at low sampling resolution and different optimization iterations.

computation.

The final step in the 3D reconstruction process is to recover texture information for each reconstructed model. The texture fitting process takes each final reconstructed model as an input, generates a texture map for each model, and fills color information to each texture map. The final results of the texture fitting process are shown in Fig. 4.11. At this stage, final textured models are ready to be used in other applications.

CHAPTER 4. PROGRESSIVE 3D RECONSTRUCTION FROM MULTI-VIEW IMAGES

Figure 4.11: Visual results of 3 models before and after texture fitting: *Venus* with 9837 vertices (top), *Pot* with 29884 vertices (middle), and *Bowl* with 10639 vertices (bottom).

Chapter 5

Conclusions and Future Work

5.1 Conclusions

In this thesis, we have studied the problem of generating progressive meshes in two scenarios: from a given complex 3D textured model, and from a set of calibrated and segmented multi-view images. In light of results obtained, following conclusions are drawn:

- A new method has been developed to extend the original quadric error metric model to compute both geometry and texture coordinates when generating progressive meshes. Our method can save up to 52.4% of memory requirements as compared to the natural extension of the quadric error metric model without sacrificing the visual quality of the generated progressive meshes. This improvement is especially meaningful for handheld device where the memory resource is very limited.
- We have introduced the rate-distortion concepts and their computations into wide baseline multi-view reconstruction so as to generate optimal progressive meshes directly. To the best of our knowledge, for wide baseline multi-view reconstruction, there is no prior work addressing similar issues.
- We have presented two improvements to the sampling method used in the PDE-based optimization so as to achieve accurate surface distortion approximation with

low complexity. Particularly, a novel sampling technique was proposed to extend the influence of each vertex to its complete neighboring regions, and an efficient sampling resolution estimation was provided to adjust sampling resolution according to image resolution.

- We have proposed a novel rate-distortion optimized refinement algorithm that allocates more vertices to the regions with more details. Our algorithm significantly outperforms the state-of-the-art PDE-based optimization approach at all the reconstruction rates.

5.2 Future Work

Our current work can be extended in the following ways:

- *Incorporating stereo information.* In chapter 4, we only consider one type of data sources, i.e. 2D information stored in 2D images. Besides 2D information, 3D information has also been used in 3D reconstruction. With a large number of cameras available, 3D stereo information can be extracted in addition to 2D information. There have been a large number of works proposed in literature to incorporate 2D and 3D information to reconstruct very high quality models [12]. It would be interesting to extend or modify our proposed algorithm to incorporate both 2D and 3D information.
- *Encoding 3D progressive reconstructed models.* In a distributed environment, 3D objects are reconstructed at a server side, and they are transmitted and used at client sides. The transmission process requires 3D objects to be encoded and decoded efficiently. It is interesting to investigate further the encoding and decoding process of the generated progressive models with consideration to the redundancy between each level of the reconstructed models.

CHAPTER 5. CONCLUSIONS AND FUTURE WORK

- *Generating progressive texture maps for a set of progressive meshes.* Our current work generates a texture map for each final model at a fixed rate. This is obviously not an effective way to generate texture maps for a set of progressive meshes. Because two progressive meshes at two close rates are very similar to each other, their texture maps are expected to be very similar. Our future work aims to generate a progressive texture map which can be used for a set of progressive meshes.
- *Towards real-time 3D reconstruction.* It has always been our ambition to develop a 3D reconstruction algorithm with a very low complexity so as to reconstruct any 3D object with an acceptable level of quality in real-time. Although our current algorithm takes about half an hour to reconstruct a 3D model, we believe that the processing time can be reduced by using GPU and choosing a better time step for the PDE-based optimization.

References

- [1] C. Oztireli, G. Guennebaud, and M. Gross, “Feature preserving point set surfaces based on non-linear kernel regression,” in *Computer Graphics Forum*. Blackwell Publishing, 2009, vol. 28, pp. 493–501.
- [2] M. Kazhdan, M. Bolitho, and H. Hoppe, “Poisson surface reconstruction,” in *Proceedings of the Fourth Eurographics Symposium on Geometry Processing*. Eurographics Association, 2006, pp. 61–70.
- [3] P. Alliez, M. Meyer, and M. Desbrun, “Interactive geometry remeshing,” *ACM Transactions on Graphics*, vol. 21, no. 3, pp. 347–354, 2002.
- [4] Y. Duan, L. Yang, H. Qin, and D. Samaras, “Shape reconstruction from 3D and 2D data using PDE-based deformable surfaces,” *Lecture Notes in Computer Science*, pp. 238–251, 2004.
- [5] H. Hoppe, “Progressive mesh,” in *Proceedings of SIGGRAPH*, 1996, vol. 96, pp. 99–108.
- [6] D.P. Luebke, “A developer’s survey of polygonal simplification algorithms,” *IEEE Computer Graphics and Applications*, pp. 24–35, 2001.
- [7] M. Garland, J. Shade, D. Lischinski, D.H. Salesin, T. DeRose, J. Snyder, F. Sillion, G. Taubin, W. Horn, J. Rossignac, et al., “Multiresolution modeling: survey & future opportunities,” *Computers and Graphics*, vol. 22, no. 6, pp. 27–36, 1998.
- [8] M. Garland and P.S. Heckbert, “Surface simplification using quadric error metrics,” in *Proceedings of SIGGRAPH*. Los Angeles, 1997, vol. 97, pp. 209–216.

REFERENCES

- [9] M. Garland and P.S. Heckbert, “Simplifying surfaces with color and texture using quadric error metrics,” in *IEEE Visualization*, 1998, vol. 98, pp. 263–270.
- [10] H. Hoppe, “New quadric metric for simplifying meshes with appearance attributes,” in *IEEE Visualization*, 1999, vol. 99, pp. 59–66.
- [11] P.V. Sander, J. Snyder, S.J. Gortler, and H. Hoppe, “Texture mapping progressive meshes,” in *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*. ACM New York, NY, USA, 2001, pp. 409–416.
- [12] S.M. Seitz, B. Curless, J. Diebel, D. Scharstein, and R. Szeliski, “A comparison and evaluation of multi-view stereo reconstruction algorithms,” in *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition-Volume 1*. IEEE Computer Society, 2006, pp. 519–528.
- [13] C.R. Dyer, “Volumetric scene reconstruction from multiple views,” *Foundations of Image Understanding*, pp. 469–489, 2001.
- [14] B. Curless, “From range scans to 3D models,” *ACM SIGGRAPH Computer Graphics*, vol. 33, no. 4, pp. 38–41, 1999.
- [15] M. Garland and P.S. Heckbert, “Surface simplification using quadric error metrics,” in *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*. ACM Press/Addison-Wesley Publishing Co., 1997, pp. 209–216.
- [16] N. Birkbeck, D. Cobzas, P. Sturm, and M. Jagersand, “Variational shape and reflectance estimation under changing light and viewpoints,” *Lecture Notes in Computer Science*, vol. 3951, pp. 536–549, 2006.
- [17] M. Okuda and T. Chen, “Joint geometry/texture progressive coding of 3D models,” in *IEEE International Conference on Image Processing*, 2000, vol. 3, pp. 632–635.
- [18] D. Luebke, B. Watson, J.D. Cohen, M. Reddy, and A. Varshney, *Level of detail for 3D graphics*, pp. 121–150, Elsevier Science Inc. New York, NY, USA, 2002.
- [19] L. Kobbelt, “3-subdivision,” in *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, 2000, pp. 103–112.

REFERENCES

- [20] A. Laurentini, “The visual hull concept for silhouette-based image understanding,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 16, no. 2, pp. 150–162, 1994.
- [21] W.E. Lorensen and H.E. Cline, “Marching cubes: a high resolution 3D surface construction algorithm,” in *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*. ACM, 1987, pp. 163–169.
- [22] P. Alliez, G. Ucelli, C. Gotsman, and M. Attene, “Recent advances in remeshing of surfaces,” *Shape Analysis and Structuring*, pp. 53–82, 2008.
- [23] V. Caselles, R. Kimmel, G. Sapiro, and C. Sbert, “Minimal surfaces based object segmentation,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, no. 4, pp. 394–398, 1997.
- [24] O. Faugeras and R. Keriven, “Variational principles, surface evolution, pdes, level set methods, and the stereo problem,” *IEEE Transactions on Image Processing*, vol. 7, no. 3, pp. 336–344, 1998.
- [25] W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery, *Numerical recipes in C*, pp. 834–840, Cambridge Univ. Press Cambridge MA, USA:, 1992.
- [26] M. Desbrun, M. Meyer, P. Schroder, and A.H. Barr, “Implicit fairing of irregular meshes using diffusion and curvature flow,” in *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*. ACM Press/Addison-Wesley Publishing Co. New York, NY, USA, 1999, pp. 317–324.
- [27] J. Cohen, M. Olano, and D. Manocha, “Appearance-preserving simplification,” in *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*. ACM Press New York, NY, USA, 1998, pp. 115–122.
- [28] P. Cignoni, C. Montani, C. Rocchini, R. Scopigno, and M. Tarini, “Preserving attribute values on simplified meshes by resampling detail textures,” *The Visual Computer*, vol. 15, no. 10, pp. 519–539, 1999.

REFERENCES

- [29] M.S. Floater, “Mean value coordinates,” *Computer Aided Geometric Design*, vol. 20, no. 1, pp. 19–27, 2003.
- [30] M.S. Floater and K. Hormann, “Surface parameterization: a tutorial and survey,” *Advances in Multiresolution for Geometric Modelling*, pp. 157–186, 2005.
- [31] A. Sheffer, E. Praun, K. Rose, and I. NetLibrary, “Mesh parameterization methods and their applications,” *Foundations and Trends® in Computer Graphics and Vision*, vol. 2, no. 2, pp. 105–171, 2006.
- [32] M. Desbrun, M. Meyer, and P. Alliez, “Intrinsic parameterizations of surface meshes,” in *Computer Graphics Forum*. Blackwell Synergy, 2002, vol. 21, pp. 209–218.
- [33] P. Cignoni, M. Corsini, and G. Ranzuglia, “Meshlab: an open-source 3D mesh processing system,” *ERCIM News (73)*, pp. 45–46, 2008.
- [34] E. Imre, A.A. Alatan, and U. Gudukbay, “Rate-distortion efficient piecewise planar 3-D scene representation from 2-D images,” *IEEE Transactions on Image Processing*, vol. 18, no. 3, pp. 483–494, 2009.
- [35] K.N. Kutulakos and S.M. Seitz, “A theory of shape by space carving,” *International Journal of Computer Vision*, vol. 38, no. 3, pp. 199–218, 2000.
- [36] L. Zhang and S.M. Seitz, “Image-based multiresolution shape recovery by surface deformation,” in *Proceedings of SPIE, the International Society for Optical Engineering*. Society of Photo-Optical Instrumentation Engineers, 2001, vol. 4309, pp. 51–61.
- [37] Y. Duan, *Topology adaptive deformable models for visual computing*, PhD Thesis, State University of New York at Stony Brook, USA, 2003.
- [38] T. Theoharis, G. Papaianou, and E.A. Karabassi, “The magic of the z-buffer: a survey,” in *Proc. of 9th International Conference on Computer Graphics, Visualization and Computer Vision*, 2001, pp. 379–387.

REFERENCES

- [39] A. Hornung and L. Kobbelt, “Robust and efficient photo-consistency estimation for volumetric 3D reconstruction,” *Lecture Notes in Computer Science*, vol. 3952, pp. 179–190, 2006.
- [40] Y. Duan and H. Qin, “A subdivision-based deformable model for surface reconstruction of unknown topology,” *Graphical Models*, vol. 66, no. 4, pp. 181–202, 2004.
- [41] W. Welch and A. Witkin, “Free-form shape design using triangulated surfaces,” in *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*. ACM New York, NY, USA, 1994, pp. 247–256.
- [42] J. Carranza, C. Theobalt, M.A. Magnor, and H.P. Seidel, “Free-viewpoint video of human actors,” in *ACM SIGGRAPH 2003 Papers*. ACM, 2003, pp. 569–577.
- [43] C. Hernández, *Stereo and silhouette fusion for 3D object modeling from uncalibrated images under circular motion*, PhD thesis, TELECOM ParisTech, France, 2004.
- [44] M.S. Floater and K. Hormann, “Surface parameterization: a tutorial and survey,” *Advances in Multiresolution for Geometric Modelling*, pp. 157–186, 2005.
- [45] A. Sheffer, E. Praun, and K. Rose, “Mesh parameterization methods and their applications,” *Foundations and Trends® in Computer Graphics and Vision*, vol. 2, no. 2, pp. 105–171, 2006.
- [46] A. Sheffer, B. Lévy, M. Mogilnitsky, and A. Bogomyakov, “ABF++: fast and robust angle based flattening,” *ACM Transactions on Graphics (TOG)*, vol. 24, no. 2, pp. 311–330, 2005.

Publications

- [1] V. N. Nghiem, J. Cai, and J. Zheng, “Memory-efficient texture coordinate computation in 3D mesh simplification,” in *International Symposium on Digital Life Technologies*, Taiwan, 2009.
- [2] V. N. Nghiem, J. Cai, and J. Zheng, “Rate-distortion optimized progressive 3D reconstruction from multi-view images,” submitted to *Proceedings of the 18th Pacific Conference on Computer Graphics and Applications*, Hangzhou, China, 2010.