國立交通大學

資訊科學與工程研究所

碩 士 論 文

結 合 置 換 貼 圖 之 網 格 模 型 表 示 法

Mesh Representation with Displacement Mapping

研 究 生：林奕均

指導教授：莊榮宏　教授

中 華 民 國　九 十 五　年 十 月

結合置換貼圖之網格模型表示法
Mesh Representation with Displacement Mapping

研 究 生：林奕均　　Student：Yi-Chin Lin

指導教授：莊榮宏　　Advisor：Jung-Hong Chuang

國 立 交 通 大 學
資 訊 科 學 與 工 程 研 究 所
碩 士 論 文

A Thesis

Submitted to Institute of Computer Science and Engineering

College of Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Computer Science

October 2006

Hsinchu, Taiwan, Republic of China

中華民國九十五年十月

結合置換貼圖之網格模型表示法

研究生：林奕均　　　　指導教授：莊榮宏 博士

國立交通大學資訊科學與工程研究所

摘　　要

置換貼圖是一個以純量高度函數來表示模型表面上的幾何細節的技術。結合置換貼圖的顯像技術在近幾年來因為繪圖硬體的效能提升，已經能夠達到即時顯像的要求。而基於這些現有的結合置換貼圖的即時顯像技術，我們發展出一個新的化簡模型的機制，可以產生一個新的網格模型表示法，以一個化簡過後的基底網格模型結合一張置換貼圖來表示一個以大量的多邊形去建構幾何細節的原始模型。原始模型的整體形狀被轉化成化簡後的基底網格模型，而在原始模型表面上的那些幾何細節資料則以置換貼圖的型態保存下來。於是我們可以利用結合置換貼圖的即時顯像技術去繪製化簡過後，多邊型數量較少的的基底網格模型，並且加上置換貼圖來重現原始模型的顯像效果。

Mesh Representation with Displacement Mapping

Student: Yi-Chin Lin Advisor: Dr. Jung-Hong Chuang

Department of Computer Science
National Chiao Tung University

ABSTRACT

Displacement mapping is an technology to represent the surface details as a scaler height-field function. Thanks to the advancement of graphic hardware, rendering with displacement mapping came to real-time in recent years. Based on the new rendering methods for displacement mapping we develop a new simplification framework to combine the coarse base mesh with a displacement map to be a new mesh representation. The global shape of the original mesh is approximated by the coarse base mesh and the detail componenets are captured by the displacement map. And this mesh representation can be rendered in real-time performance using the per-pixel ray-casting methods for displacement mapping[9][10][22][25].

Acknowledgments

I would like to thank my advisors, Dr. Jung-Hong Chuang, for his inspirations, guidance and patience for the past years.

I especially thank Tan-Chi Ho for giving many concrete ideas of my work and helping me completing this thesis. Thank Chih-Chun Chen for conducting many parts of my research in simplification. Thank Wen-Yeh Lee for taking care of my health and my philosophy. Thank Horng-Shyang Liao for helping me dealing with the awkward Latex compiler.

I extremely thank all my justice companies, Chao-Wei Juan, Yong-Cheng Chen, Roger Hong, Yu-Shuo Lio, Chia-Lin Ko, Ji-Wen Chon, Yung-Cheng Chen, who watch the Gransazers and Justirisers with me every day. May the god of star be with you, my friends. Transform!!! Finally I have to thank the most cute girl Ya-Ching Chiu in our lab for curing my woman-panic-disorder.

Contents

Abstract **1**

Acknowledge **3**

List of Figures **7**

1 Introduction **1**

1.1 Brief of displacement mapping 1

1.2 Motivation 3

1.3 Contributions 4

1.4 Thesis Organization 5

2 Related Work **7**

2.1 Mesh Representations 7

2.2 Multiresolution Mesh Representation with Displacement Map 8

2.3 Real-Time Rendering of Displacement Map 10

3 Mesh Representation with Displacement Mapping **13**

3.1 Approach overview 13

3.2 Mesh simplification 15

3.2.1 Growing a bounding shell 15

3.2.1.1 Self-intersection 17

3.2.2 Simplifying the base mesh 18

3.2.2.1 Progressive meshes 19

3.2.2.2 Quadric error metric 20

3.2.2.3 Constraints 21

3.2.2.4 Conclusion 24

3.3 Mesh parameterization 24

3.3.1 Motivation 24

3.3.2 Parameterization method 26

3.4 Build the displacement map 28

3.5 Rendering with displacement mapping 29

3.5.1 Preprocessing before rendering 29

3.5.2 Per-pixel ray-casting framework 30

4 Experimental Results **33**

4.1 Result of a typical displaced model 33

4.2 The adjustment of offset scale 38

4.3 Approximate surface details of arbitrary models 41

4.3.1 Teeth 41

4.3.2 Venus head 44

4.3.3 Bunny 47

4.3.4 Armadillo 50

5 Conclusion and Future Work **53**

5.1 Conclusion 53

5.2 Future work 54

Bibliography **57**

List of Figures

1.1 An example of displacement mapping to an arbitrary model. [9]. 2

1.2 An example of extracting the surface details in [18]. 3

2.1 An example of deformation and reconstruction [1]. 10

3.1 The flow chart of our work 14

3.2 The view ray miss the correct intersection point 16

3.3 The simplification is bounded in shell volume 16

3.4 The self-intersection after growing vertices b and c.[4] 18

3.5 Edge collapsing and vertex splitting [14]. 19

3.6 PM sequence [16]. 20

3.7 The height-field overlapping after simplification 22

3.8 The definition of a triangle slab 23

3.9 The under-sampling of non-optimized parameterization 25

3.10 The mean value coordinate 27

3.11 The ray-intersection searching scheme of [9] 31

4.1 Reconstruct tiled plane by our method 34

4.2 The rendering differences of tile plane in view 1 34

4.3 The rendering differences of tile plane in view 2 35

4.4 The rendering differences of tile plane in view 3 35

4.5 The displacement map of input model as club 36
4.6 The rendering differences of club in view 1 36
4.7 The rendering differences of club in view 2 37
4.8 The rendering differences of club in view 3 37
4.9 Results with different offset scales of the bounding shell 40
4.10 Results of teeth 41
4.11 Rendering teeth in view 1 42
4.12 Rendering teeth in view 2 42
4.13 Rendering teeth in view 3 43
4.14 Rendering teeth in view 4 43
4.15 Results of venus head 44
4.16 Rendering venus head in view 1 44
4.17 Rendering venus head in view 2 45
4.18 Rendering venus head in view 3 45
4.19 Rendering venus head in view 4 46
4.20 Results of bunny 47
4.21 Rendering bunny in view 1 47
4.22 Rendering bunny in view 2 48
4.23 Rendering bunny in view 3 48
4.24 Rendering bunny in view 4 49
4.25 Results of armadillo 50
4.26 Rendering armadillo in view 1 50
4.27 Rendering armadillo in view 2 51
4.28 Rendering armadillo in view 3 51
4.29 Rendering armadillo in view 4 52

CHAPTER 1

Introduction

1.1 Brief of displacement mapping

Displacement mapping is a technique to represent geometric details on the surface of 3D models. Since more complicated 3D models are demanded in nowadays computer graphic applications such as animations and games, there are many methods developed to represent or approximate large amount of surface features while reduce the rendering overhead.Texture mapping, bump mapping, and normal mapping have already been included to be the typical functionalities of many computer graphic applications. And displacement mapping is also popularly used in many modeling tools because it encodes the surface details as a height-field map which helps the artists designing detail features more conveniently.

The idea of displacing a surface by displacement mapping was first introduced by Cook[6]. The displacement mapping function $D(p)$ of a point p on a surface applies some scalar-offset distance along the point's normal to p

$$D(p) = P_p + N_p d \tag{1.1}$$

Where P_p is the position of p, N_p is the normal of p, and d is the offset distance. Then a displacement map is the map with each pixel stores the value of d, also noted as the displacement depth, of the surface point p mapping to this pixel. Such a map forms a two-dimensional height function to represent the deformed surface details.

The applications of displacement mapping can be roughly classified into two type, depending on how the displacement map is generated. Popular utility of displacement mapping in modern applications is to add some geometric details on simple-shaped surfaces. In this way the content of displacement map is manually designed by artists or from some images. Fig. 1.1 shows an example of these appications. (a) is a gray-scale depth map, (b) is a normal map, and (c) is the rendering result with normal mapping and displacement mapping.

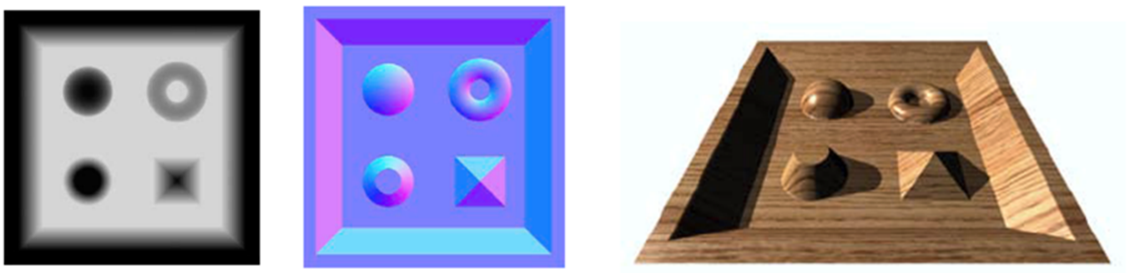

(a) Depth Map (b) Normal Map (c) Rendering result

Figure 1.1: An example of displacement mapping to an arbitrary model. [9].

In contrast to adding extra features to surface, another use of displacement mapping is to store the surface details of arbitrary model in complex shape. This way of using displacement mapping is rare in modeling but common in multi-resolution mesh editing. Separating the small-scale surface details with the model helps us focus on editing the remaining simplified base domain which represents the global shape of the input model. Moreover, to encode the surface details as a displacement map also forms the level-of-detail of the model's features. Fig. 1.2(a) is a simplified base domain, Fig. 1.2(b) is a subdivision smooth domain used to support the displacement mapping, and Fig. 1.2(c) is the reconstruct model from (a)(b) and the stored displacement information [18].

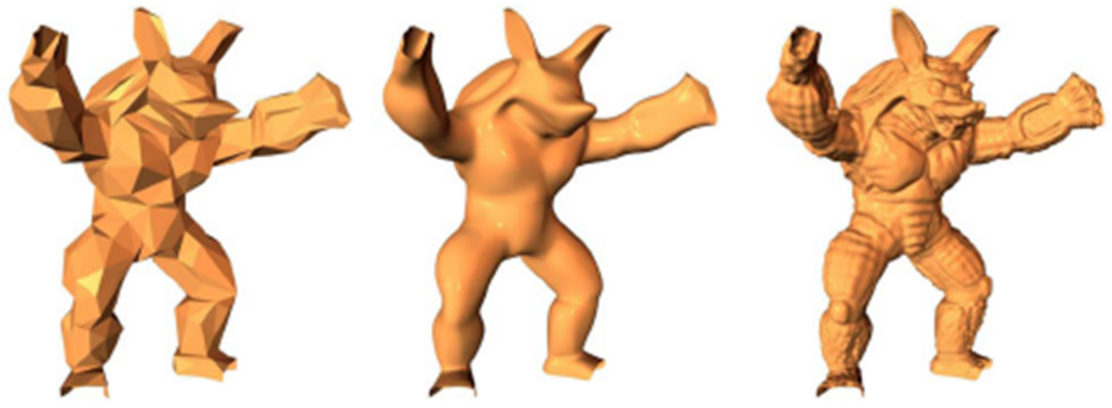


Figure 1.2: An example of extracting the surface details in [18].

1.2 Motivation

In multiresolution mesh editing, there is a reference surface named base mesh to which we map the displacement information to reconstruct the input fine mesh. The previous works tend to make the surface of the base mesh smooth enough to represent the global-shape of original fine mesh and to prevent the undesired distortion on reconstructed surface from applying the displacement map function on base mesh. The general approach is to simplify the input mesh to an arbitrary coarse version and then subdivision this coarse version as deeply as it fits the shape of input fine mesh except the surface details [1][15][17][18][20].

In mltiresolution mesh editing, it is sufficient to use the subdivided surface as the base mesh. But respect to rendering, this base mesh with displacement mapping, the high resolution of subdivided mesh can still be an overhead for standard rendering pipeline. Fortunately, there are developed approaches which utilize pixel-shader's parallel functionality to simultaneously trace each view-ray with the displacement values on base surface and compute the point at which the view-ray intersects the displaced surface [9]. Then we can use this point to compute the lighting effect

Since rendering with displacement mapping in real-time is possible now, naturally it also comes to be possible that we convert a high-detailed model to a coarse version with a displacement map encoding its surface details and render the coarse version with displacement mapping approximate the rendering result of original high-detailed model. We try to develop a simplification framework to build this mesh representation which takes advantage of displacement mapping to reduce the overhead of rendering high-detailed models.

1.3 Contributions

In this paper we proposed a framework to simplify a high-detailed model into a coarse base mesh and extract the original model's surface details as a displacement map. And we add the constraints to prevent simplification from destroying the height-field structure of displacement mapping.

The resulting base mesh with the displacement map which we called displaced base mesh is a new representation of original input mesh. This representation has the following benefits:

1. Large amount of polygons' details are converted to height value of displacement image. The remain mesh topology and parameterization on base domain are simple, make it a compression of input model.

2. The simplified base mesh proposes a global shape and also a tight bounding convex hull of input model.

3. The encoded surface details can be edited and compressed using any image processing method.

4. This mesh representation support real-time rendering methods with displacement mapping.

1.4 Thesis Organization

In chapter 2 we will introduce some previous works related to our work. Chapter 3 describes our framework of building the displaced base mesh. In chapter 4 we show the results of rendering and also analyze the geometric approximation, and chapter 5 is the conclusions and discusses of future work.

CHAPTER 2

Related Work

In this chapter, a review on related works of our method will be given. Section 2.1 briefly introduces kinds of representations of meshes. In section 2.2, we introduce the background of multiresolution mesh representation with displacement map. In section 2.3, we discuss the real-time rendering with displacement mapping using per-pixel ray-casting methods.

2.1 Mesh Representations

The mesh representations are developed for various purposes. Catmull and Clark designed a subdivision scheme to generalize uniform B-Spline knots insertion to meshes of arbitrary topology [2]. Starting with a user-defined mesh of arbitrary topology, it refines the initial mesh by adding new vertices, edges and faces with each step of subdivision following a fixed set of subdivision rules. In the limit, a sequence of recursively refined polyhedral meshes will converge to a smooth surface.

Regular remeshing is the process where an irregular mesh is approximated by a mesh with (semi-)regular connectivity [8]. The simplicity of a regularly remeshed representation has many benefits. In particular it eliminates the indirection and storage of vertex indices and texture coordinates. This will allow graphics hardware to perform rendering more efficiently, by removing random memory accesses and thus improving memory access performance

Gu et al. introduced the geometry images(GIM) which creates the most regular remeshed representation [13]. Their construction converts the surface into a topological disk using a network of cuts and parametrizes the resulting disk onto a square domain. Using this parametrization, the surface geometry is resampled onto the pixels of an image. As an added benefit, techniques such as image compression can be directly applied to the remesh.

Multiresolution geometric models support representation and processing of spatial objects at different levels of detail [1][15][17][18][20]. Such representations have been extensively studied in the literature because of their impact on applications, such as terrain modeling, scientific data visualization, virtual reality, etc. Some constructions of multiresolution mesh representation include a displacement map to record the geometric information extracted from the mesh surface in high resolution. We introduce the works of these kinds of constructions which are similar to ours in the next section 2.2

Color mapping represents the color of surface features regardless of any 3D information. Bump and normal mapping manipulate the input normal of lighting computation to generalize the true lighting effects. Displacement mapping map a scalar height function to the mesh surface to represent surface details of meshes[6]. With exactly encoding the 3D information of mesh surface it provides self-occlusion, self-shadows and silhouette. Some surface features such as bricks, sculpture, textiles can be well approximated by the displacement mapping.

2.2 Multiresolution Mesh Representation with Displacement Map

Krishnamurthy and Levoy show that a detailed model can be represented as a displacement map over a network of B-spline patches [17]. However, they resort to a vector-valued displacement map because the detailed model is not always an offset of their B-spline surface. Also, avoiding surface artifacts during animation requires that the domain surface be tangent-plane $(C1)$

continuous, which involves constraints on the B-spline control points.

Lee et al. proposed a mesh representation called *Displaced Subdivision Surfaces*, which also separates the surface details with a coarse version of the original high-detailed input model [18]. They defined the domain surface using subdivision surfaces since these can represent smooth surfaces of arbitrary topological type without requiring control point constraints.

Hussain et al. enhanced the work of Lee et al.[18] by proposing a new efficient method for defining the smooth domain surface based on $\sqrt{3}$-subdivision scheme [15]. The proposed algorithm not only performs better in terms of the quality of the generated surfaces but is also computationally more efficient and occupies less memory and generates surfaces with more levels of detail due to the specific nature of $\sqrt{3}$-subdivision when the prescribed target complexity of the generated mesh must not be exceeded.

Marinov and Kobbelt used a new decimation scheme for general polygonal meshes (not just triangles) that is based on face merging instead of edge collapsing [20]. This scheme is designed to derive a subdivision control mesh whose structure is properly adjusted and aligned to the major geometric features. This implies that the control vertices of the subdivision surface not only control globally smooth deformations but in addition that these deformations are meaningful in the sense that their support and shape correspond to the characteristic structure of the input mesh.

Botsch and Kobbelt focused on prevent the surface details from unatural distortion after deformation and reconstruction[1][26]. As in Fig. 2.1, a multiresolution deformation of an original surface $\tilde{\mathcal{S}}$ corresponds to change the (smooth) base surface $\mathcal{S}$ into $\mathcal{S}'$ and reconstructing $\tilde{\mathcal{S}}'$ from $\mathcal{S}'$ and the stored detail information.

Botsch and Kobbelt used volume elements enclosed between the different resolution levels to encode the detail information [1].Keeping these displacement volumes locally constant during a deformation of the base surface leads to a natural behaviour of the detail features. The corresponding reconstruction operator can be implemented efficiently by a hierarchical iterative relaxation scheme, providing close to interactive response times for moderately complex models. To solve the same problem, Xu et al. used an affine transformation matrix to encode

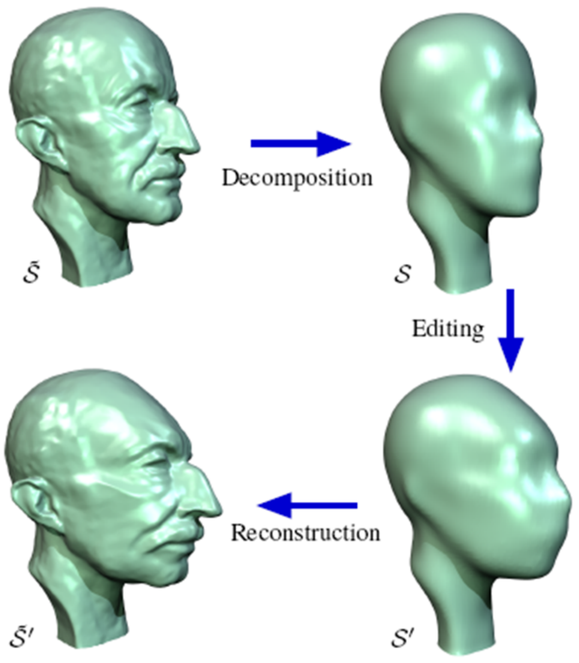


Figure 2.1: An example of deformation and reconstruction [1].

the details between the local frames of each triangle pair in the consecutive levels that share the same connectivity [26].

2.3 Real-Time Rendering of Displacement Map

While *Displaced Subdivision Surfaces* helps maintaining the consistency of surface details in mesh compression, editing, and animation, it doesn't support the real-time rendering. Because the surface details map to the subdivision domain not control mesh, the rendering of *Displaced Subdivision Surfaces* have to tessellate polygons into the subdivision domain's resolution to make the visual effect of displacement mapping. Therefore it is still a burden to the rendering pipeline.

Rendering with displacement mapping comes to be in real-time thanks to several developed methods in recent years. Most of these methods are base on a per-pixel ray-casting architecture with hardware acceleration[9][10][22][25]. Per-pixel ray-cast shades each pixel's illumination by shooting a ray from eye through this pixel's screen position, computing or approximating the point where this ray intersects the displaced surface.

Wang et al. introduced a view-dependent displacement map(VDM) which is a five-dimensinal map indiced by reference surface's texture coordinate, view direction, and the local curvature along view direction [23]. This map stores the precomputed exact displacement effect under a limited number of view conditions. The VDM method is further improved as generalized displacement maps(GDM) to represent non-height-field surface details [24]. Also the GDM approach overcomes both the texture distortion problems of VDM and the computational expense of directly mapped geometry by computing visibility jointly in texture space and object space.

Donelly presented a precomputed three-dimensional distance map for a rendered object can be used for surface extrusion along a given view direction [7]. The cost of a 3D texture and dependent texture fetches' latency make this algorithm not applicable to most real-time applications.

Mapping relief data in tangent space for per-pixel displacement mapping in real-time was proposed in [9][10][22][25]. Welsh defined a offset limit along the view ray to approxmate the intersection point and can simulate a visual motion parallax effect [25]. Policarpo et al. developed linear and binary search schemes on the texture space to iterativly optimize the approximation [9]. Later a supporting of silhouette generation proposed by Oliveira and Policarpo [19]. Sequentially they used multi-layer depth map to encode non-height-field surface details [10]. Tatarchuk applied a dynamic step size of the iterative searching scheme [22]. He adapted the step size to the displaced surface in each iteration to avoid that a sampling on view ray crossover a highly-curved displaced surface.

These methods take excellent advantage of the programmable pixel pipeline efficiency by performing height field-ray intersection in the pixel shader to compute the displacement information. These approaches generate dynamic lighting with self-occlusion, shadows and motion parallax. Unfortunately all of the above approaches exhibit strong aliasing and excessive flattening at steep viewing angles.

CHAPTER 3

Mesh Representation with Displacement Mapping

3.1 Approach overview

The goal of our work is to develop a platform for constructing the displaced base mesh which is a coarse version of input original mesh. The surface details are converted to a displacement map and can be rendered using per-pixel ray-casting methods[25][9][10][22] in real-time. Fig. 3.1 illustrates the flow chart of our work.

To capture the silhouette of original mesh in per-pixel ray casting methods for displacement mapping[9][10][22][25], we need to apply an offset along the outer direction to the surface of original mesh at first. This is done by using the concept of *Simplification Envelopes* developed by Cohen et al.[4]. Then we simplify the offset surface based on the framework of *Progressive Meshes*[14] and uses QEM [12] as our simplification metric. To ensure that the simplified base mesh is valid for displacement mapping, we also introduce two constraints during simplification: (1)The simplified surface can not intersect the surface of original mesh or the bounding shell. (2)The function from simplified surface mapping to the surface of original mesh should maintain a height-field structure. The first constraint is used to guarantee that the simplified

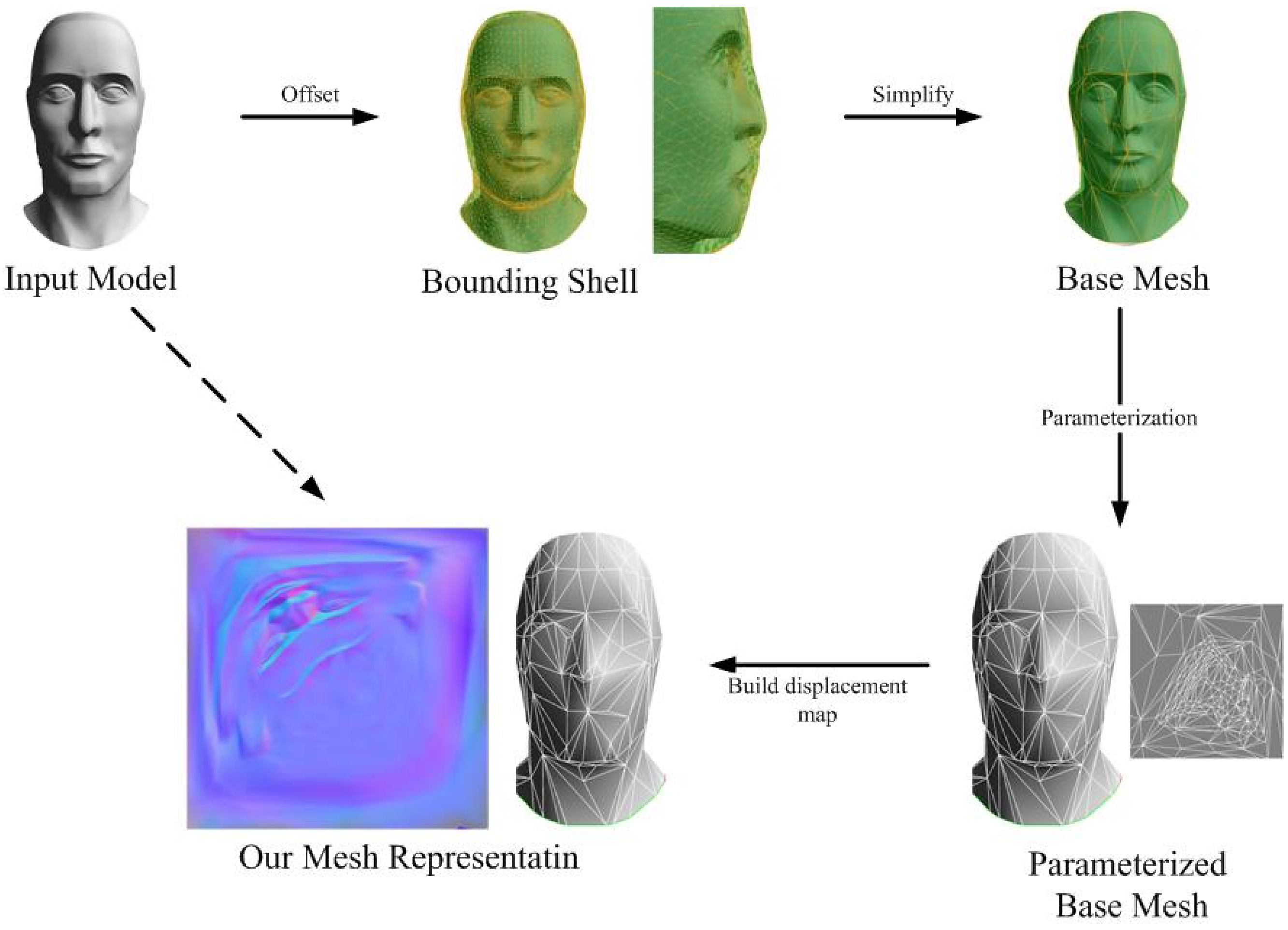


Figure 3.1: The flow chart of our work

base mesh is tightly in the bounding shell volume of original mesh. The second one ensures a one-to-one mapping between simplified base mesh and the original mesh

After we obtain the base mesh in low-resolution, we parameterize it using the method proposed by Yoshizawa et al.[27], which is an area- preserving parameterization. We build the parameterization of base mesh under the assumption that surface with larger area can catch more surface details of original mesh, so that we give it more samples by enlarge its parametric area.

At last we get the base mesh and its parameterization. To build the displacement map, we sample rays from the base mesh surface along the direction of inverse interpolated vertex-normal, computing the distance and the original surface normal of the point at which the ray intersects the surface of original mesh. Then store the original normal vector with the displacement value as a 4-channels luminance to the pixel.

In section 3.2 we will introduce our simplification framework. And section 3.3 describes the parameterization method used in our framework. We will explain how to build the displacement map in section 3.4and at last in section 3.5 we will discuss some preprocessing works before rendering and the rendering scheme[9].

3.2 Mesh simplification

In this section we describe how to build the base mesh to achieve our requirement such that: (1) the base mesh support a coarse global shape of input model. and (2) the base mesh is a coarse version of the input model.

3.2.1 Growing a bounding shell

To capture the silhouette of original mesh in displacement map rendering using per-pixel ray-casting methods[25][9][10][22], the displaced direction must be ”negative” in texture space because the view-ray can not catch the displaced surface signal if it miss the reference surface. As shown in Fig. 3.2(a) , the view ray hit the displaced surface but miss the reference surface. Since we know that the rendering pipeline will not activate if the view ray didn’t hit any triangle of reference surface, the lighting computation of this pixel will be discard and lose the displaced surface signals.

This fault can be avoided by ensuring that a view ray will always hit the reference surface before it hit the displaced surface. The most intuitive solution is to raise the reference surface to totally bound the displaced surface as in Fig. 3.2(b). Since we expect the base mesh surface outside the original mesh surface but not too far away to keep the global shape, we defined a shell volume on the original surface and restricted the base mesh surface inside the shell volume. We build an extra bounding shell to form this shell volume, then we constraints that all

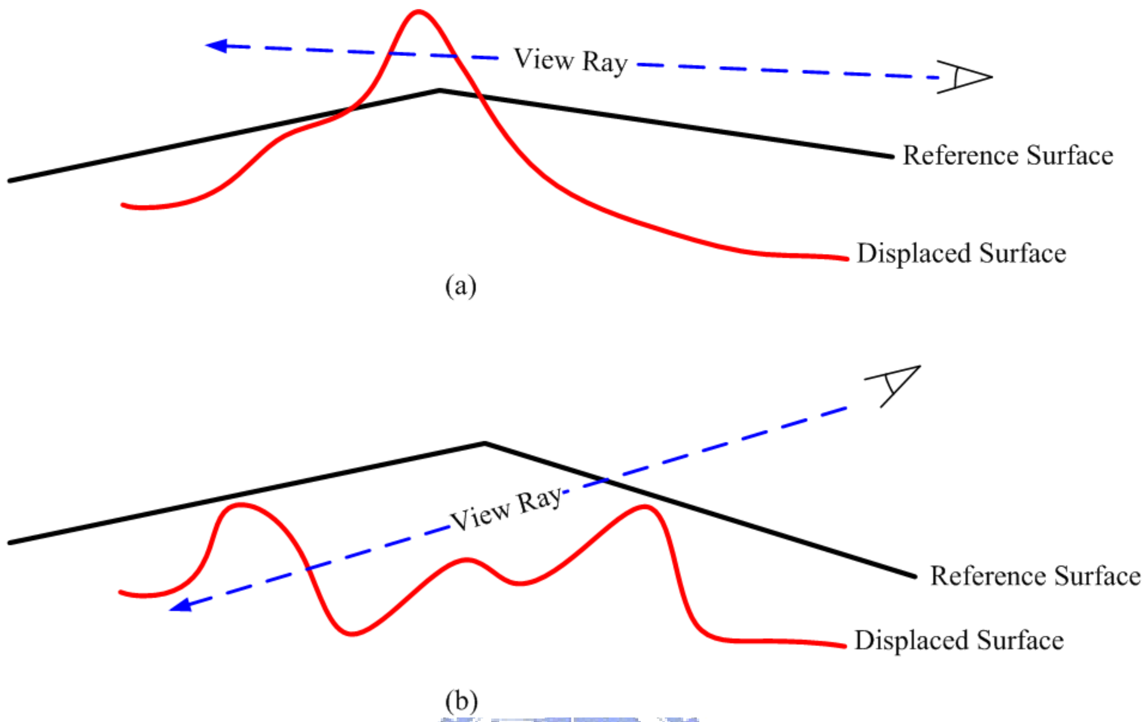


Figure 3.2: The view ray miss the correct intersection point

edit operations toward base mesh must guarantee that the base mesh surface remaining in this shell volume. The relation of original mesh, base mesh and bound shell is illustrated in Fig. 3.3

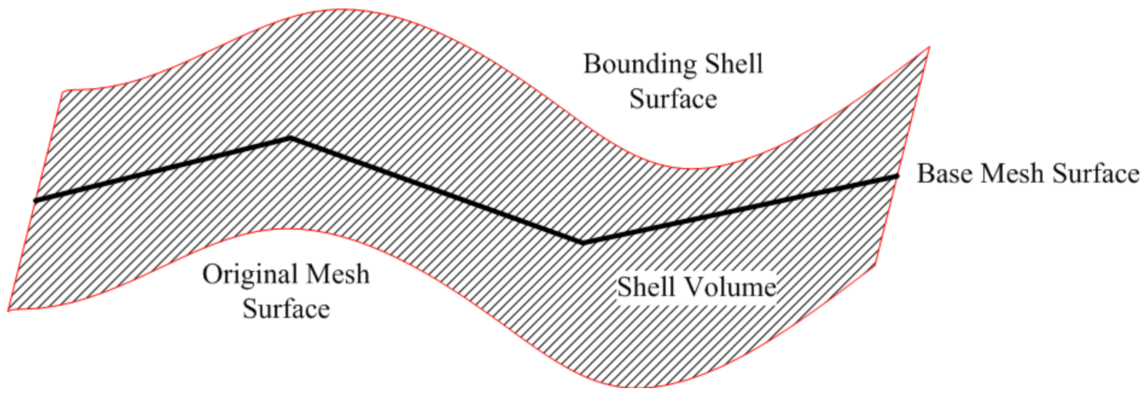


Figure 3.3: The simplification is bounded in shell volume

To construct the bounding shell, we use the concept of *Simplification Envelope* proposed by Cohen et al.[4]. The growing framework is that we iteratively offset all vertices in a very small distance along its normal direction until the grown surface achieve our requirement or there is no vertex can be moved. To maintain a smooth surface after each iteration, we tend to draw a vertex out along the average direction of the normals of the concave curves which cross the vertex.

Initially we assign a user-given parameter γ, which is 4% by default, as a ratio of the diagonal length D of input model's bounding box. Then we set $D\gamma$ as the maximum growing length of our bounding shell. By constraint the base surface inside the bounding shell in the simplification process, $D\gamma$ also implicitly becomes the maximum displacement depth.

Then we give a growing iteration times $\mathcal{K}$ which is 50 by default. For each iteration a global growing step size ϵ is set by $\epsilon = D\gamma/K$. This is the basic offset of each growing iteration. we encourage each vertex to growing the distance ϵ at each iteration such that it tends to achieve the maximum growing length $D\gamma$ after $\mathcal{K}$ iterations.

For each vertex we set the growing direction as the average of vertex normal N_p and a vector E_p. Where E_p is the average of the vertex p's outer edge's vectors, Only the "concave" edges which are within 90^o of the vertex normal N_p is considered. Otherwise if no such edge are adjacent to p, it has a convex neighborhood, than we just offset it the half distance of the global step size ϵ. This will make that: if a vertex has its neighborhood convex throughout whole growing process, than it would grow as half of the distance as a vertex has its neighborhood concave throughout the process.

3.2.1.1 Self-intersection

When growing the outer bounding shell, we have to prevent the surface of bounding shell from self-intersection. The robust method to solve this problem is to compute each triangle prism of bounding shell a Voronoi region, and do not allow the grown vertices to piercing into non-adjacent Voronoi region, as in Fig. 3.4, we can see the grown vertices b^+ and c^+ has pierced into non-adjacent Voronoi region.

But the computation of the three-dimensional Voronoi diagram of triangle prisms is non-trivial. We use an iterative method to avoid self-intersection at each step of a local operation.

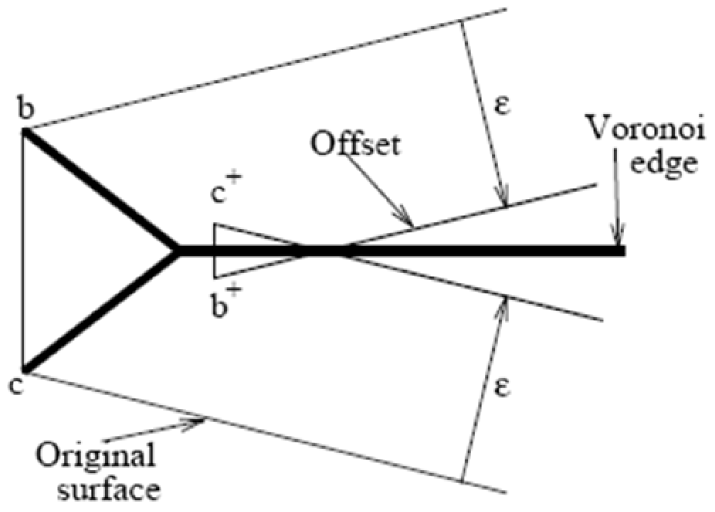


Figure 3.4: The self-intersection after growing vertices b and c.[4]

At each iteration after growing a vertex with a small distance in its normal direction, we test the intersection between its one-ring faces and all other faces. If intersection takes place we delay the growing of this vertex by push it back to its previous position and shorten its growing distance by a given ratio which we currently set it to 0.9. Therefore if a vertex growing is determined to generate self-intersection at the $i \sim i + l$ iterations, than it can be growed at the $i + l + 1$ iteration with the distance $0.9^l \epsilon$ as possible as away.

3.2.2 Simplifying the base mesh

For the initial configuration of base mesh, we set it's surface to be the average of original mesh and bounding shell before simplification. Our simplification framework inherits the typical steps to build a PM[14] representation of base mesh. We use quadric error metric(QEM)[12] as simplification metric to lead the simplified base mesh into a smooth coarse shape of original mesh. Additionally we add two constraints to satisfy our demand for displacement mapping.

3.2.2.1 Progressive meshes

The progressive meshes consists of a base mesh and a sequence of primitive reduction operations used to coarsen the base mesh to represent a multi-resolution triangular mesh. PM is constructed using greedy algorithm by performing a series of primitive reduction of increasing costs on the original mesh, such as *edge collapsing* operation, until no reduction operation is possible. First, all possible edge collapsing operations of the input mesh are collected as candidates, then the *cost* of each candidate is calculated, and finally the edge collapsing operations are performed in the order of increasing cost. After each edge collapsed, the costs of remaining candidates shall be updated, and operations that become invalid will be removed from the candidate list. A cost updating step maybe time consuming and can be delayed using lazy evaluations, as proposed by Garland and Heckbert[12].

Each edge collapsing operation removes one vertex and generally two faces from the mesh. And to refine a simplified mesh back to a higher level of detail, series of inverse operations are performed on the mesh in inverse order of the construction sequence. *Vertex splitting* is the inverse operation of *edge collapsing*, which adds a vertex and generally two faces back to the mesh. Fig. 3.5 shows the behaviors of a vertex collapsing and vertex splitting.

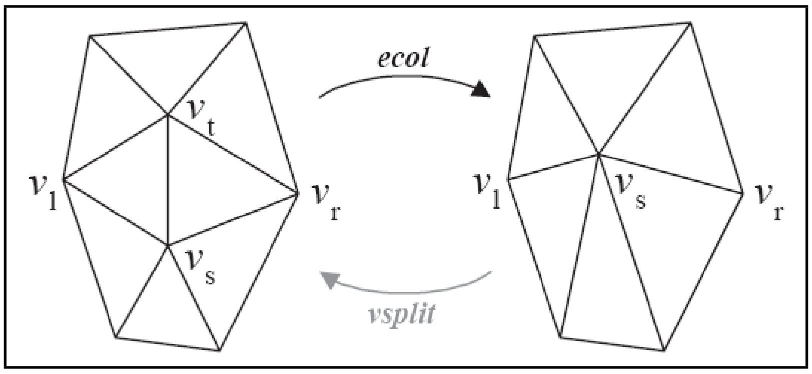


Figure 3.5: Edge collapsing and vertex splitting [14].

Given an input mesh M and a sequence of n vertex collapsing operations, a PM sequence with $n + 1$ levels of detail can be constructed. An example is shown in Fig. 3.6, where a mesh M_5, after five edge collapsing operations, results in a PM of six levels. The mesh of the simplest level is denoted as M_0. a next level of PM can be acquired by performing a edge collapsing operation, and a previous level can be acquired by performing a vertex splitting operation.

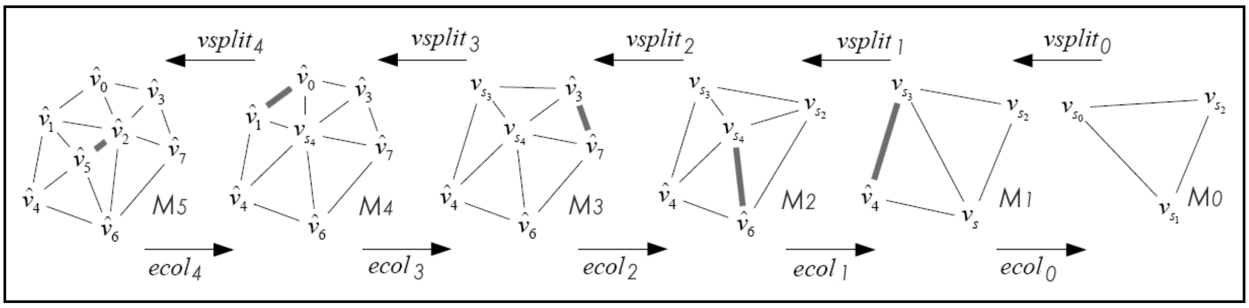


Figure 3.6: PM sequence [16].

3.2.2.2 Quadric error metric

To guide the order of primitive reductions during simplification, we need a metric. The QEM is a metric that considers the geometric error between the simplified surface and the original surface. The QEM evaluates the cost of each vertex collapsing as the distances from the remaining vertex $\overline{V_v}$ to the neighboring faces of the original vertices V_u and V_v. To do this, the first thing is to evaluate the total distance of a vertex v to its neighboring faces by

$$\triangle(v) = \sum_{p \subset neighbor-faces(v)} (p^T v)^2 \tag{3.1}$$

where $v = [v_x\ v_y\ v_z\ 1]^T$, $neighbor - faces(v)$ is the set of v's neighboring faces, and $p = [a\ b\ c\ d]^T$ representing the plane defined by $ax + by + cz + d = 0$ where $a + b + c + d = 1$. Equation 3.1 can be rewritten as the following quadratic form

$$\triangle(v) = \sum_{p \subset neighbor-faces(v)} (vp^T)(pv^T) = \sum_{p \subset neighbor-faces(v)} v^T(p^T p)v = v^T(\sum_{p \subset neighbor-faces(v)} K_p)v \tag{3.2}$$

where K_p is in 4x4 matrix form as

$$K_p = p^T p = \begin{bmatrix} a^2 & ab & ac & ad \\ ab & b^2 & bc & bd \\ ac & bc & c^2 & cd \\ ad & bd & cd & d^2 \end{bmatrix} \tag{3.3}$$

Equation 3.1 depicts that K_p can be used to find the squared distance from a point to the face p, and the K_p of all v's neighbor planes can be summed to present the total distances of v to its neighboring faces. The summed K_p for v is called $Q(v)$, the *quadric error metric* of v, or QEM of v.

To evaluate the cost of the edge collapse $(V_u, V_v) \rightarrow \overline{V_v}$, $Q(V_u)$ and $Q(V_v)$ are summed as $Q(\overline{V_v})$ to approximate the total distances of $\overline{V_v}$ to th neighboring faces of V_u and V_v. By replacing $(\sum_{p \subset neighbor-faces(v)} K_p)$ by $Q(\overline{V_v})$ in Equation 3.2, we get the cost function of edge collapse $(V_u, V_v) \rightarrow \overline{V_v}$ as

$$\triangle(\overline{V_v}) = \overline{V_v}^T Q(\overline{V_v}) \overline{V_v} \tag{3.4}$$

While building the PM sequence, the QEM of each vertex that affected by a edge collapse need to be updated accordingly. The QEM of each remaining vertex during the PM building phase approximates the total distances from the vertex to all the faces collapsed to it.

3.2.2.3 Constraints

In previous simplification works there are some basic constraints such as triangle-flipping, triangle-degeneration, the folding on topology, or additionally the triangle's aspect ratio. But for our requirement, to ensure that the simplified mesh is valid for displacement map construction we add two constraints on the simplification. The first constraint is that the simplified surface can not intersect the surface of original mesh or the bounding shell. When checking an half-edge collapsing for this constraint, we virtually collapse the half-edge and trace the faces F_i in the one-ring neighborhood of the start vertex of the half-edge. If F_i is discarded after collapsing, ignore it, else check if it intersect the surface of original input mesh or the bounding shell.

The other constraint is the height-field overlapping such that: since we want to convert the original mesh surface into a height-field representation, we have to prevent the original mesh surface from height-field overlapping against base mesh surface after simplification.

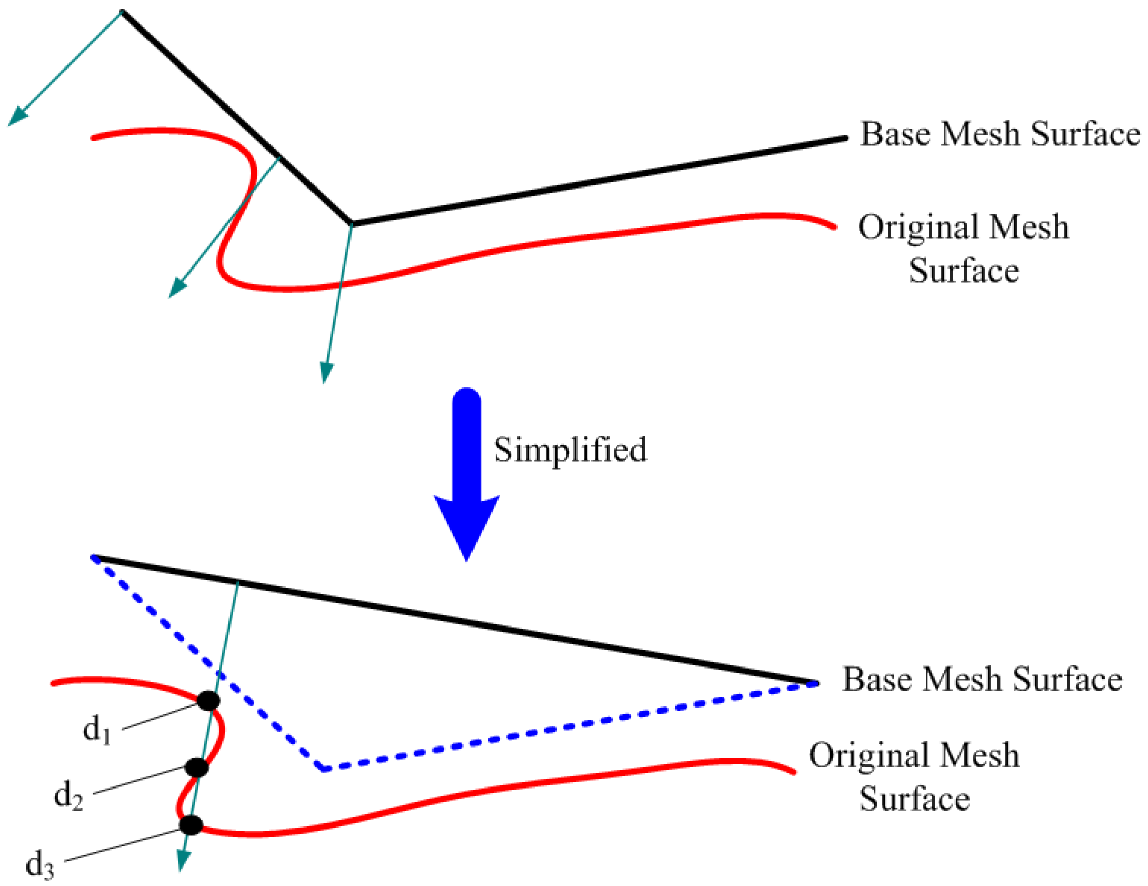


Figure 3.7: The height-field overlapping after simplification

Fig. 3.7 is an example of height-field overlapping after simplification., the arrows represent the displacement mapping directions. After simplifying the base mesh surface, we can see that there is a ray along displaced direction intersects three points on the original mesh surface, result in three displacement depth values mapping to the same point on base mesh surface, ruin the height-field structure.

Such height-field overlapping can be easily identified if there exits a point on original mesh surface whose normal dot the normal of base mesh surface equal to zero. This identifying method is proved to be correct by Collins and Hilton[5]. In order to analyze the height-field structure under a base face of base mesh, we construct a triangle slab of each base face. As in Fig. 3.8, the triangle slab is an volume constructed by extending the base face along its three inverse vertices' normals $(-N_1, -N_2, -N_3)$. The verex normal is computed by average its one-ring adjacent faces's normals. Then we used this triangle slab to determine where the region of original mesh surface are covered by the triangle slab and may be converted to displacement signals mapping to this base face.

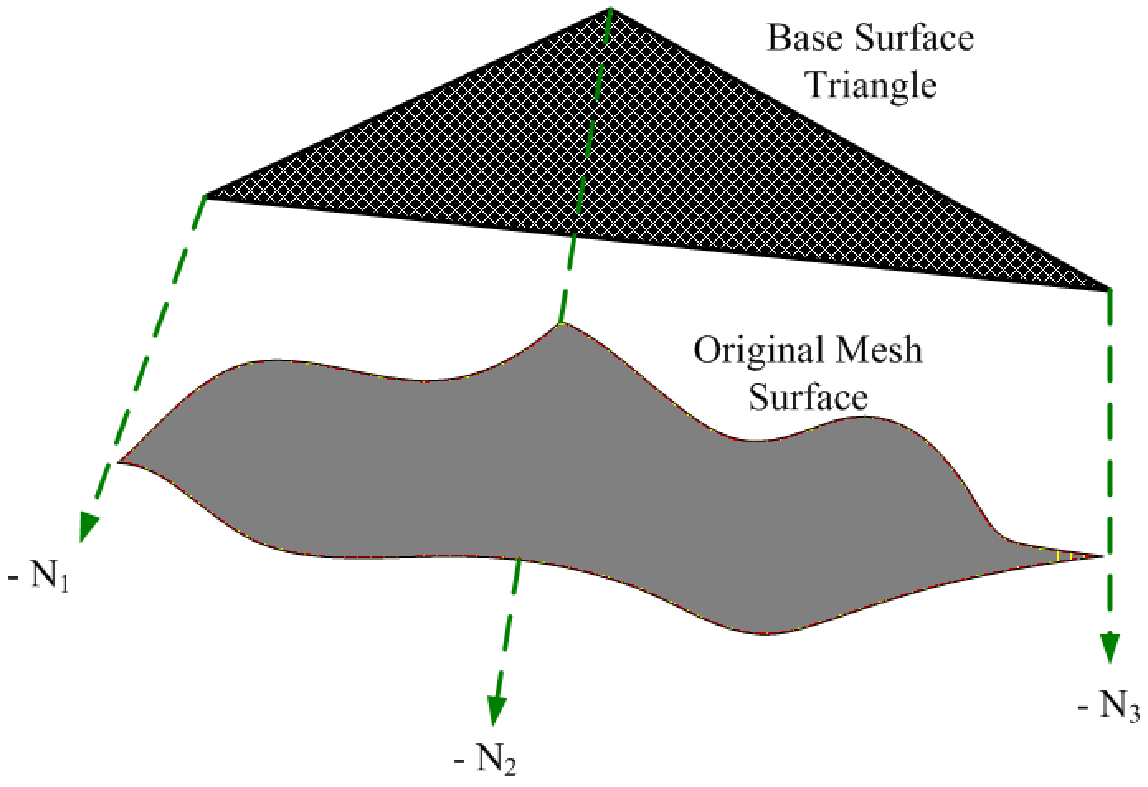


Figure 3.8: The definition of a triangle slab

For the discrete triangular surface of original mesh, because the normal is constant on face but variate cross edges, we compute for each edge of original mesh the two dot values of its adjacent faces. That is, for each edge we have its two adjacent face's normal N_1 and N_2, and the normal of base face N_b. Then the height-field overlapping occurs if the two dot values have different signs as in equation 3.5.

$$(N_1 \cdot N_b)(N_2 \cdot N_b) \leq 0 \tag{3.5}$$

3.2.2.4 Conclusion

In this sectoin we described the framework of generating a simplified base mesh. We first construct the bounding shell with the shell volume to maintain the base mesh's globe shape during and after smplification. For our purpose that to render with per-pixel ray-casting methods[9][10][22][25]. we add one constraint during simplification that the base mesh surface can not cross over the original mesh surface and pierce into the original mesh. For a valid one-to-one displacement mapping from base mesh surface to original mesh surface, we add another constraint that there is no height-field overlapping after each simplifivation step.

3.3 Mesh parameterization

3.3.1 Motivation

Before we build the image of displacement map, we have to construct the correspondence between the 2D domain of displacement map and the 3D surface of base mesh, that is, the parameterization of base mesh. As we will sample the displacement depths according to the image resolution, the parametric area of a base face will determine how many samples to the base face's underlying displaced surface signals. If a base face has its displaced surface containing high-frequency signals(ie. a feature curve on the original surface) but has a small parametric area on 2D domain, it may not get enough samples to catch the high-frequency signal which is called *under-sampling* as shown in Fig. 3.9

The area-preserving parameterization tries to preserve the area ratio between the 3D surface and the parametric domain, and can be used to overcome this drawback. Here we give the assumption that the larger is the base face's area the more displaced surface signals it should

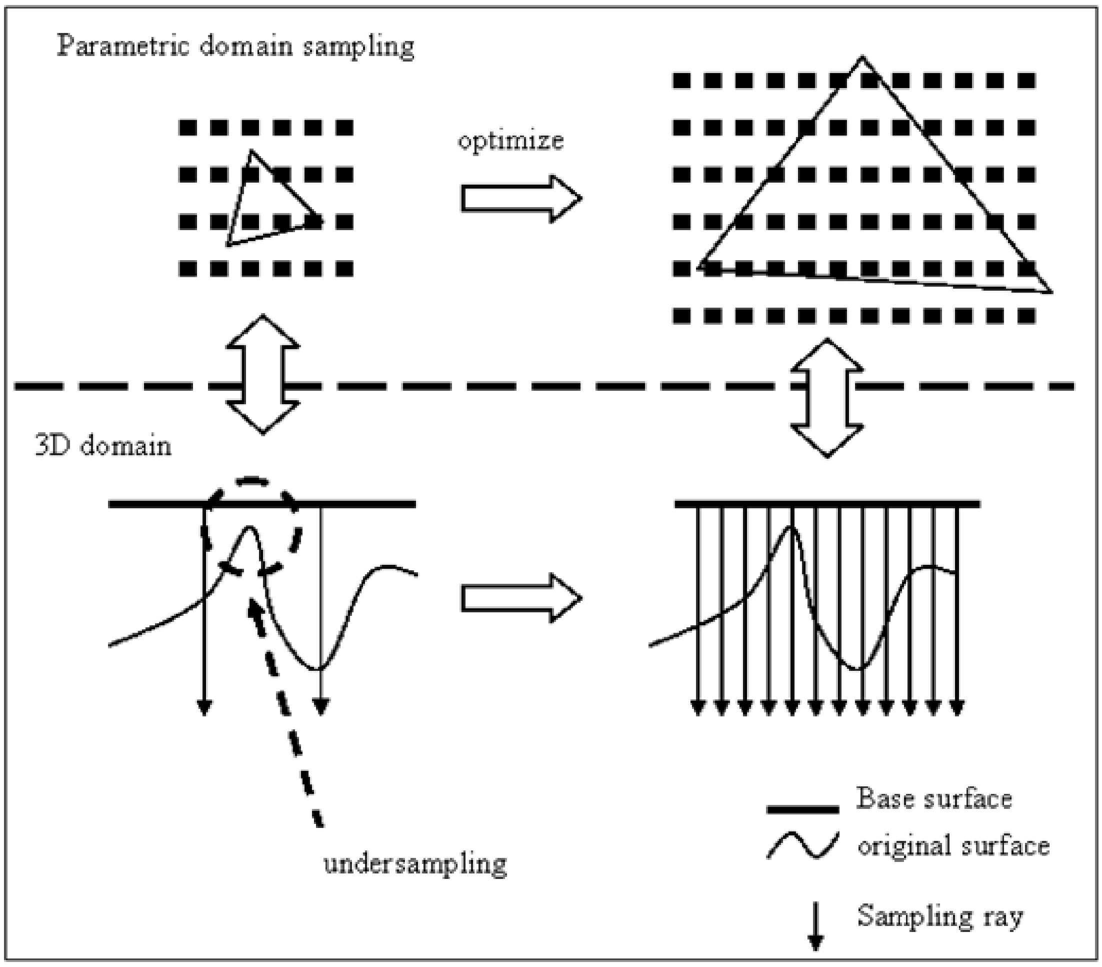


Figure 3.9: The under-sampling of non-optimized parameterization

cover. This assumption is valid when we made the base faces as close to the original surface as possible.

3.3.2 Parameterization method

We use the parameterization method proposed by Yoshizawa et al.[27], and Floater's mean value coordinate[11] as the initial parameterization. This area-preserving parameterization method first applies an intial parameterization to the mesh, then iterativly optimize the parameterization by solving the linear equatino system which is defined by the vertex-vertex relationship. At each improvement iteration, we use the parameterizatin result at previous step to compute the geometric stretches, update the edges weights according to these streches, and use the updated weights to generate the next improved parameterization.

The initial parameterization is obtained by solving a mass-spring system. The first step of the method is to specify the parameter points $\psi(v)$ of the boundary vertices $v \in V_B$. Then, set each interior vertex $v \in V_I$ to be a convex combination of its neighbors. For each interior vertex v, a set of strictly positive convex weights λ_{vw}, $w \in N_v$, is chosen such that

$$\sum_{w \in N_v} \lambda_{vw} = 1.$$

For all interior vertices, the mapping $\psi(v)$, $v \in V_I$, is determined by solving the following linear system of equations:

$$\psi(v) = \sum_{w \in N_v} \lambda_{vw} \psi(w), \quad v \in V_I.$$

This equation can be rewritten as

$$\psi(v) - \sum_{w \in N_v \text{ and } w \in V_I} \lambda_{vw} \psi(w) = \sum_{w \in N_v \text{ and } w \in V_B} \lambda_{vw} \psi(w), \quad v \in V_I,$$

and further as

$$Bx = c.$$

The problem remained is how to determine the value of λ_{vw}. Floter gives these value as follow[11]:

$$\lambda_{vw} = \frac{\tan \alpha/2 + \tan \beta/2}{\|v - w\|} \tag{3.6}$$

The α and β is the two angles show in Fig. 3.10

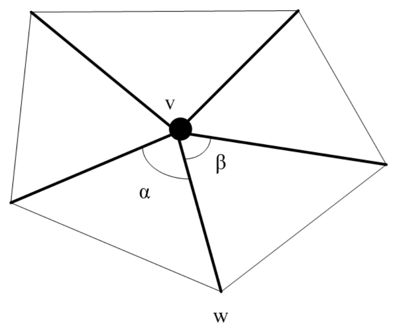


Figure 3.10: The mean value coordinate

We now place emphasis on the optimization stage of parameterization in which we implement [27] to enlarge the parametric area of base faces. The basic idea is to gradually improve the parameterization via changing weights λ_{vw}. The magnitude of λ_{vw} can be viewed as the spring strength of parametric edge vw, larger λ_{vw} tends to pull the parametric positions of v and w together and result in a higher mesh density along the parametric edge vw.

With this property and the requirement that we attempt to enlarge the parametric area with higher geometric stretch. Conceptually we can update each λ_{vw} by dividing it with the stretch of a differential area surrounding w.

In our implementation we add two concerns to smooth the parameterization optimization. First, In equation 3.7 we replace the σ_w by the average of σ_v and σ_w, this new divisor is more appropriate representing the stretch along edge vw. And to each vertex's stretch we apply a transfer parameter η to weaken the power of a single optimization iteration. The modified equation is as follow:

$$\lambda_{vw}^{new} = \frac{\lambda_{vw}^{old}}{(\sigma_v + \sigma_w)/2}, \lambda \in N(v) \tag{3.7}$$

where of a vertex w the stretch σ_w is:

$$\sigma_w = \sqrt{\frac{\Sigma A(T_u)\sigma(U_u)}{\Sigma A(T_u)}}^{\eta} \quad \sigma(U_u) = \sqrt{(\Gamma^2 + \gamma^2)/2} \tag{3.8}$$

The transfer parameter η is simply set as i/K where K is the total iterations and i is the current iteration. Our default number of K is 30. We find that the optimization can generally converged the total stretch of base mesh as η is about $0.7 \sim 0.8$.

3.4 Build the displacement map

The last stage of our framework is to build the displacement map. Given an image of resolution R we compute the displacement depth of each pixel on the image. For each pixel p with its 2D coordinate (s_p, t_p), through the parameterization of the base mesh we can find the base face $\triangle(P_1, P_2, P_3)$ where (s_p, t_p) maps to and its barycentric coordinate $(\lambda_1, \lambda_2, \lambda_3,)$ corresponding to $\triangle(P_1, P_2, P_3)$. The mapped 3D point P on the base face is then computed by $P = \lambda_1 P_1 + \lambda_2 P_2 + \lambda_3 P_3$.

As for the direction D_p of computing the displacement depth from P. we apply the interpolation of the three vertices' normals in the inverse direction. Given the vertices' normals N_1, N_2, N_3 of the unit length. $D_p = -(\lambda_1 N_1 + \lambda_2 N_2 + \lambda_3 N_3)$. There we have the sampling ray with start point P and its direction D_p. By testing the intersection between this ray and the original surface of input model obtain the intersection depth d and the surface normal $n = (x_n, y_n, z_n)$ of the intersection point. Finally the value of pixel p is the 4-channel tuple of (x_n, y_n, z_n, d).

When computing the intersection between sample-rays and original mesh surfaces, we speed up this process by previously maintaining a rough correspondance between base mesh triangles and the original mesh triangles. As mentioned in 3.2.2.3 we have attached each base triangle a triangle slab and use the bounding volume under this triangle slab to collect the original triangles, see Fig. 3.8. As a consequence, we know that a sample-ray casted from a base triangle will more probably hit the original triangles collected by the triangle slab. Therefore we test the sample-ray with these triangles first, and in the case that the sample-ray miss all these triangles, we test it with all the triangles of orginal mesh.

3.5 Rendering with displacement mapping

3.5.1 Preprocessing before rendering

To render with displacement mapping, we use the method proposed by Policarpo et al.[9] of which we will introduce the main algorithm in next section 3.5.2 to approximate the intersection point of view-ray and the displaced surface using pixel-shader of the graphic hardware. In addition to implement the per-pixel ray-casting algorithm, we add some precomputed per-vertex parameters which are interpolated in rendering pipeline and send them to the pixel-shader. These parameters include the tangent space's basis vectors (T_v, B_v, N_v), where T_v is the tangent vector, B_v is the bi-normal vector , N_v is the vertex normal of vertex v. When transforming the view ray from object space to the tangent space, the tangent space of current pixel-shader is interpolated from the vertices' tangent space bases. By smoothing the tangent space variation through fragments on the base mesh's polygons we can get a continuous rendering result across the edges and the vertices of base mesh.

Moreover, because the ray-tracing process of the rendering algorithm defines each searching step size on the parametric domain. a skew proportion of the base face relative to its parameterization may severely distorts the searching step size and direction. To solve this problem we also precompute a tuple of scale values (S_x, S_y, S_z) which represents the scale factor of transformation from object space to the parametric space. For the same reason of continuous rendering result this is a per-vertex parameter, too.

Actually the tangent space basis and the scale factors can be computed simultaneously as follow: First we compute the parameters on base faces. Given positions of 3 vertices (P_1, P_2, P_3) and the corresponding parametric coordinates (u_1, u_2, u_3). we derive two vectors V_p and V_q in object space:

$$V_p = P_2 - P_1, V_q = P_3 - P_1$$

and the corresponding vectors ν_p and ν_q on the parametric domain is:

$$\nu_p = u_2 - u_1, \nu_p = u_3 - u_1$$

Set the transformation matrix M.

$$M = [T|B|N]$$

We solve M by the following equation transforms V_p and V_q to ν_p and ν_q:

$$\begin{bmatrix} & V_p & \\ & V_q & \\ 1 & 0 & 0 \end{bmatrix} M = \begin{bmatrix} & \nu_p & 0 \\ & \nu_q & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

Then we get the tangent space basis T_v, B_v, and N_v are the unit vectors of T, B, and N respectively. The scale factor S_x is the magnitude of T, S_y is the magnitude of B, and S_z is the maximum displacement depth computed from building the displacement map.

3.5.2 Per-pixel ray-casting framework

We implement the process of per-pixel ray-casting framework[9] which can be conceptually understood as following and illustrated in Fig. 3.11. For each fragment to be rendered:

1. Compute the viewing direction VD as the vector from the viewer to the 3D position of the point on the polygonal surface;

2. Transform VD to the tangent space (defined by the tangent, normal and bi-normal vectors) associated with the current fragment;

3. Use VD' (the transformed VD) and A, the (s_0, t_0) texture coordinates of the fragment, perform a linear searching to compute B, the (s_1, t_1) texture coordinates where the depth exceed the displacement value in the first time;

4. Perform a binary searching starting with A and B to compute the intersection between VD' and the height-field surface;

5. Compute the lighting effect of the fragment using the attributes (e.g., normal, depth, color, etc.) associated with the texture coordinates (s, t) of the computed intersection point.

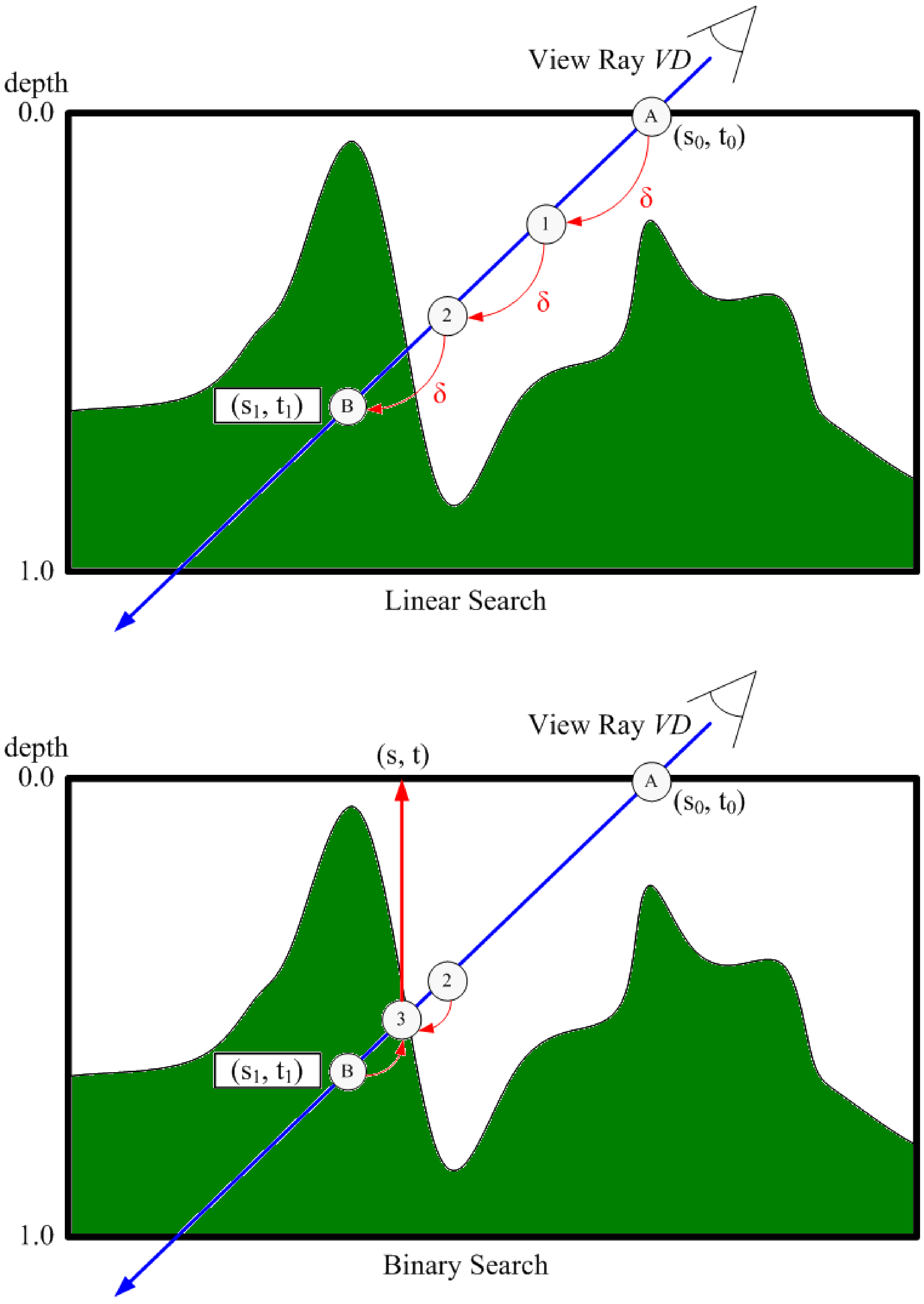


Figure 3.11: The ray-intersection searching scheme of [9]

CHAPTER 4

Experimental Results

In this chapter we present the result of our output displaced base mesh and compare it with original input model. We implement our approach using Visual C++ 7.1 and OpenGL library with shading language GLSL. The image resolution of our displacement map is 1024x1024. The hardware platform we used to run our implementation is a PC with Intel(R) Pentium(R) 4 3.0GHz and 512 MB of RAMS. The graphic card is nVidia GeForce 6800 with 256 MB memory and supports hardware vertex-shader and pixel-shader.

4.1 Result of a typical displaced model

In this section, to validate our approach as a inverse process of mapping a displacement map on a model, we create a reconstructed model by applying a low-frequency displacement map on a plane, tessellate the model sufficiently to represent the displacement features with vertices and polygons. The rendering result of our approach is expected to approximate the result of original model and to recover the displacement map. So we compare the rendering result of original sets and that of our outputs.

Fig. 4.1(a) illustrate the original input model which has 5,00 polygons and is displayed with wireframe, Fig. 4.1(b) shows the base mesh simplified to 12 polygons, and Fig. 4.1(c) is our reconstructed displacement map.

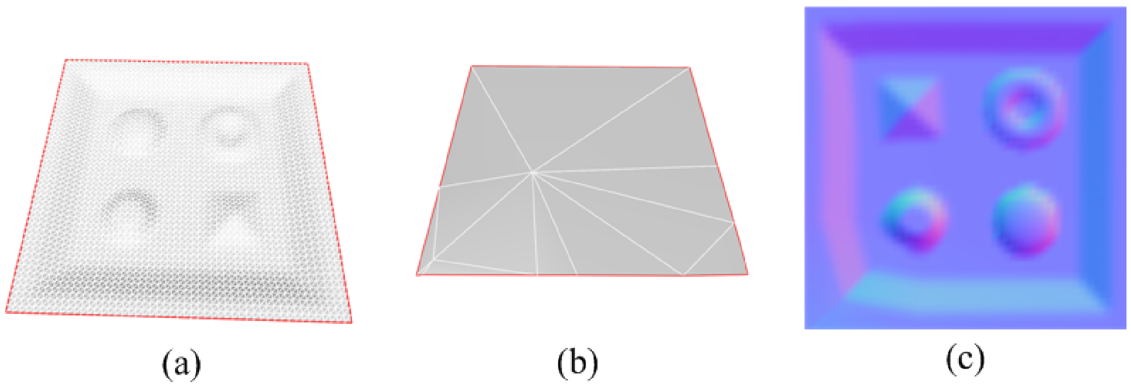


Figure 4.1: Reconstruct tiled plane by our method

Fig. 4.2 to Fig. 4.4 demonstrate a comparison between rendering results of original input model and our output mesh representation. The images in different rows is taken under different view direction. Images in left column are rendered with original model, images in middle column are rendered with our mesh representation, and images in right column are the differences of left two images.

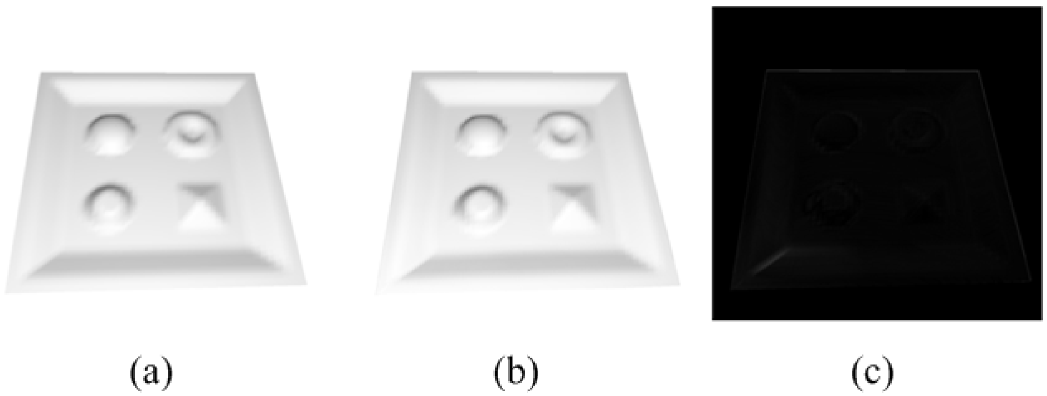


Figure 4.2: The rendering differences of tile plane in view 1

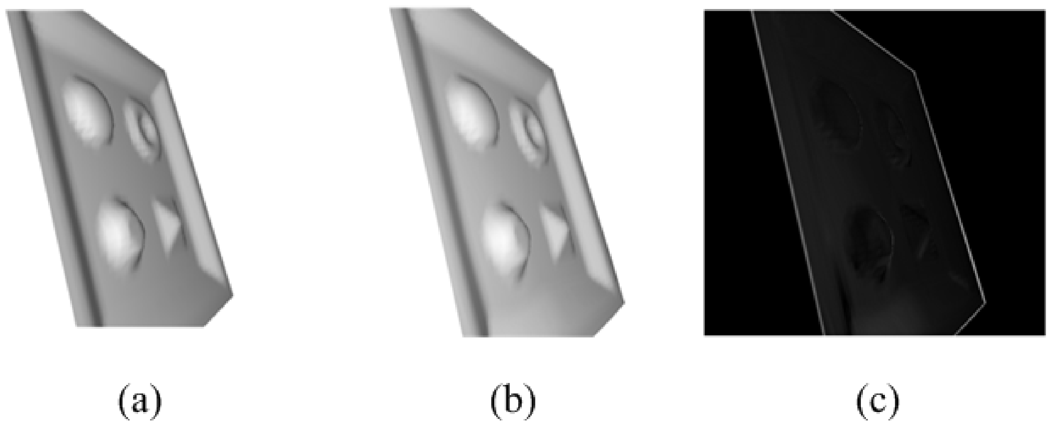


Figure 4.3: The rendering differences of tile plane in view 2

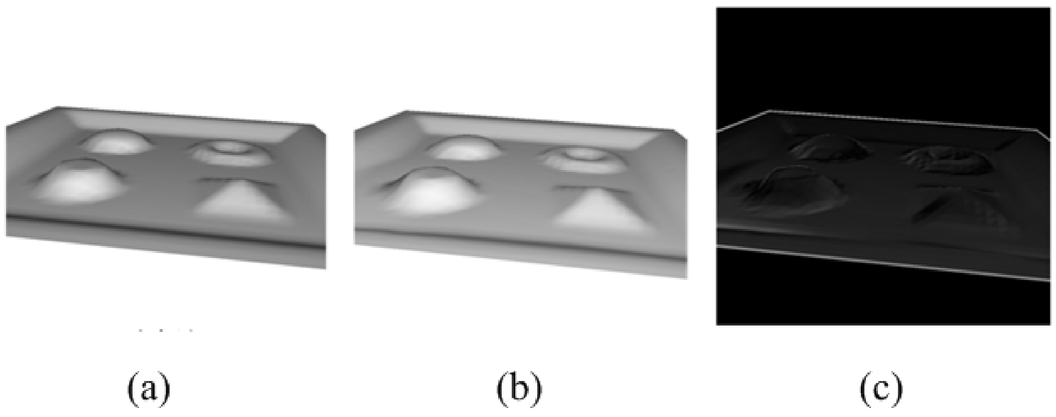


Figure 4.4: The rendering differences of tile plane in view 3

There is another example shown in the following figures from Fig. 4.6 to Fig. 4.8. The input model is a golf club with 83,479 polygons. The base mesh is simplified to 354 polygons and the carved words and grooves on the original surface are converted to displacement signals, see its diaplcement map in Fig. 4.5.

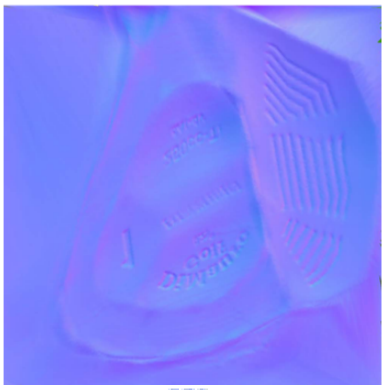


Figure 4.5: The displacement map of input model as club

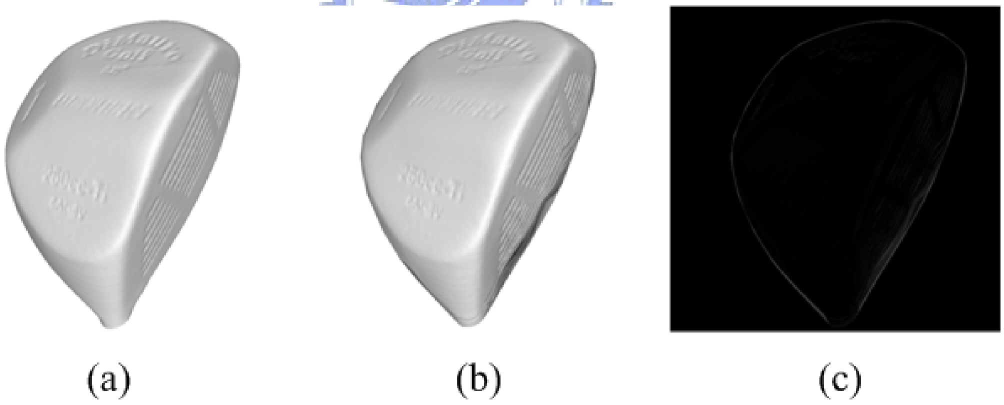


Figure 4.6: The rendering differences of club in view 1

We observed that the error of rendering our mesh representation is larger at the following places: (1) on the base mesh's boundary, (2) near the edges of base mesh, (3) near the silhouette of displaced surface.

Error on the boundary is simply due to offsetting the whole mesh, and error near the edge is the light distortion when the searching of view-ray across a triangle's edge on the texture space.

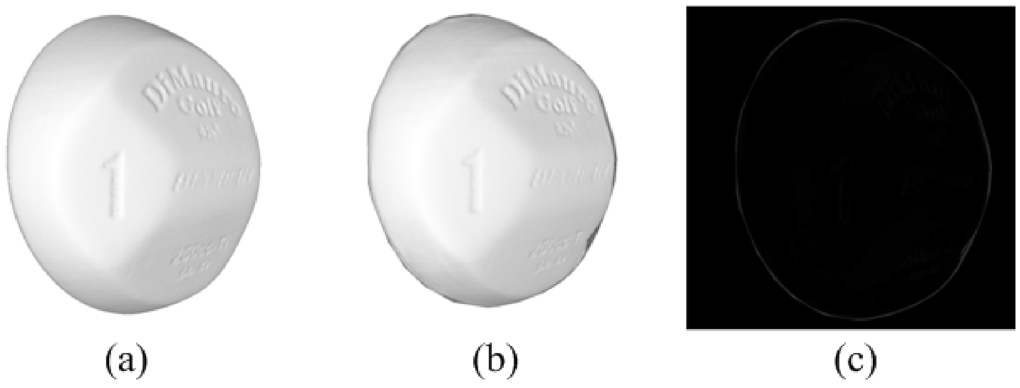


Figure 4.7: The rendering differences of club in view 2

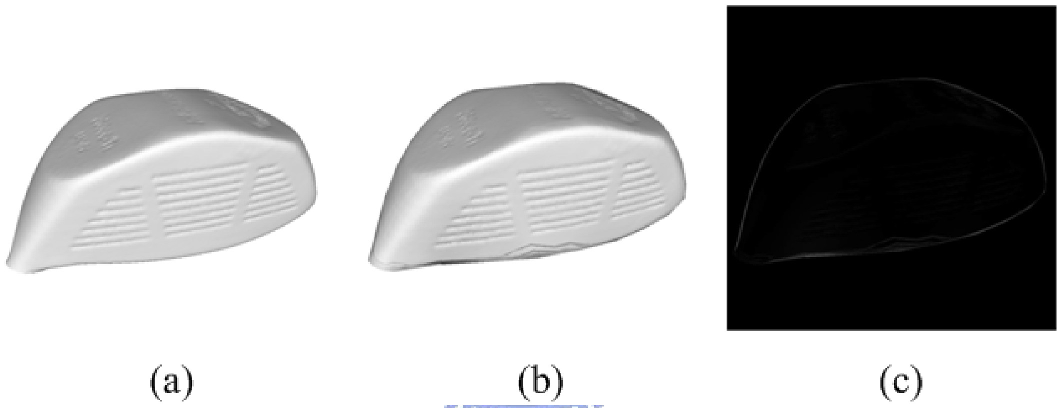


Figure 4.8: The rendering differences of club in view 3

Then near the sihlouette there is the aliasing caused by insufficient searching iterations of the rendering method[9].

Additionally, the differences increases as we view at an angle. The slope of view-ray leads to more aliasing from the larger searching step size which is the same reason of the error condition (3).

4.2 The adjustment of offset scale

In this section we experiment on that how does the given parameter γ, as a ratio of the diagonal length D of input model's bounding box mentioned in section 3.2.1, affect the coarseness, the number of faces, of the simplified base mesh and the rendering quality of the output representation.

We take two measurements into consideration: one is the average difference between interpolated vertex normals and the face normals on base mesh, the other one is the quantization error of displacement depth. The average normal difference δ_m of the simplified base mesh is computed as: For each base face f with its face normal N_f and three vertex normals N_1, N_2, and N_3. Because the normal difference is piece-wise linear on a base face so it is the average of the three differences of its vertex normals.

$$\delta_f = \frac{\|(N_1) - (N_f)\| + \|(N_2) - (N_f)\| + \|(N_3) - (N_f)\|}{3} \tag{4.1}$$

And the average normal difference of base mesh,

$$\delta_m = \frac{\sum_{T_i \in M} \delta_i A(T_i)}{\sum_{T_i \in M} A(T_i)} \tag{4.2}$$

Where $A(T_i)$ is the area of triangle T_i.

These measurement comes from that the rendering schemes of per-pixel ray-casting [9][19][25]tend to trace the view-ray in the texture space regardless of the transformation from object to texture space. These schemes assume that each interpolated normal of fragments is perpendicular to the base face. Hence the visual distortion may rises on the surface of base mesh where there is large normal difference.

Another measurement is the depth quantization error since we output the depth quantized into the range$[0 \sim 255]$. This value is proportion to the tightness of the simplified base mesh. Reasonably we prefer a smaller quantization error which reflects a tighter fitness of the input model and a better configured displacement map.

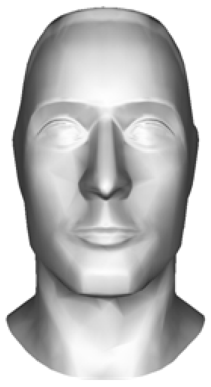
The experiment statics are showed in table 4.1. We can observe that: The face count decreases as given a larger offset scale. This is because of more space of the bounding shell

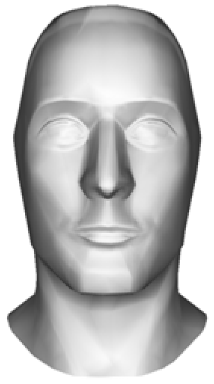
offset scale γ	face count	normal difference	depth quantization error
1%	1515	1.202483	0.000458
2%	778	1.169118	0.000828
3%	559	1.101581	0.001311
4%	396	1.149185	0.001651
5%	358	1.190046	0.001732
6%	308	1.170843	0.002085
7%	270	1.269019	0.002663
8%	259	1.314161	0.002990

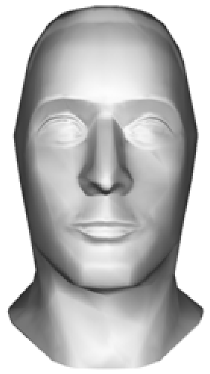
Table 4.1: Statistics of different offset scales

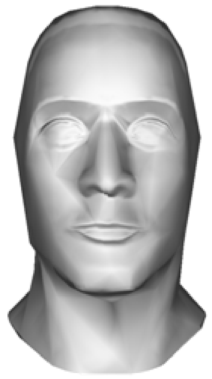
volume for simplification. But the surface of input model limits the decreasing rate as we prevent the simplified surface from cross over the original surface. The normal difference is not totally proportion to offset scale but about to slightly increase. The depth quantization error naturally increase with offset scale.

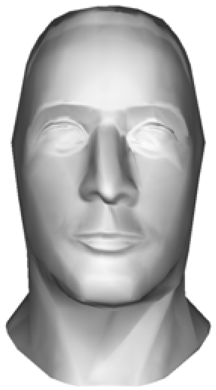
Fig. 4.9 shows the rendering results by give different offset scales γ on the human model with 9,102 faces originally. We found that at the percentage of $\%1$ and $\%2$ the results are better. And at $\%3 \sim \%5$ the results have some visual artifacts but still support a well approximation. At the percentage of $\%6 \sim \%8$ the meshes are simplified too much to remain a smooth base surface such that many visual artifacts appear.

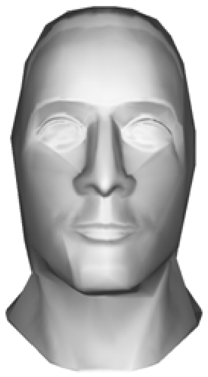

(a) offset = 1%

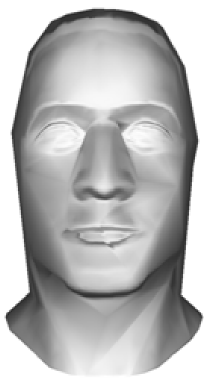

(b) offset = 2%


(c) offset = 3%


(d) offset = 4%


(e) offset = 5%


(f) offset = 6%


(g) offset = 7%

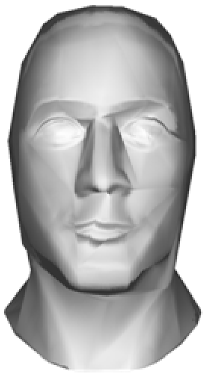
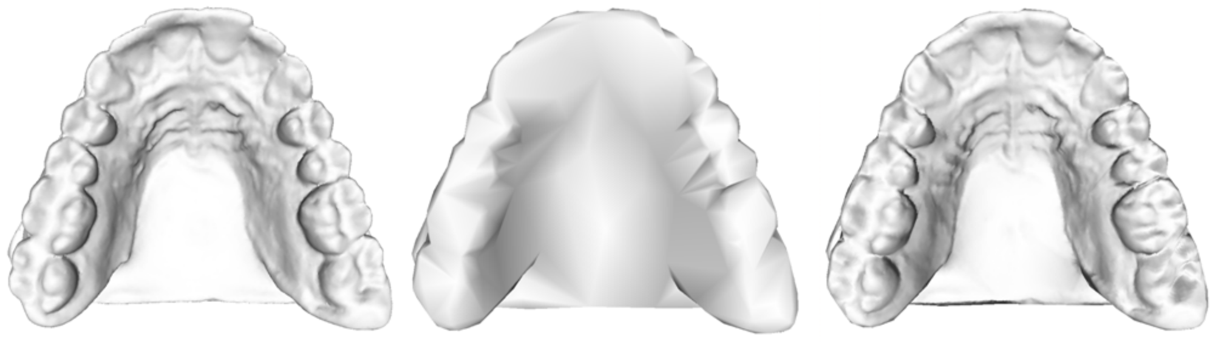

(h) offset = 8%

Figure 4.9: Results with different offset scales of the bounding shell

4.3 Approximate surface details of arbitrary models

Finally we show the result of some input models in arbitrary shapes with large amount of surface details. ”Origin” means the original input model, ”Base” means the simplified base mesh and the notation # means the face count. We present rendering results in different views while the original model at left and our mesh representation at right.

4.3.1 Teeth



(a) Teeth Origin #83k (b) Teeth Base #1020 (c) Rendering result

Figure 4.10: Results of teeth

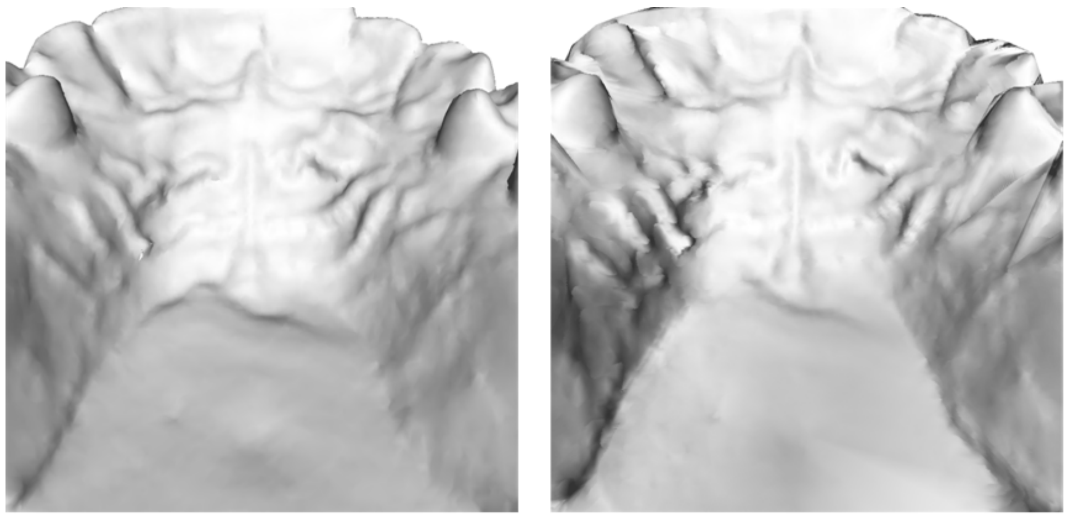


Figure 4.11: Rendering teeth in view 1

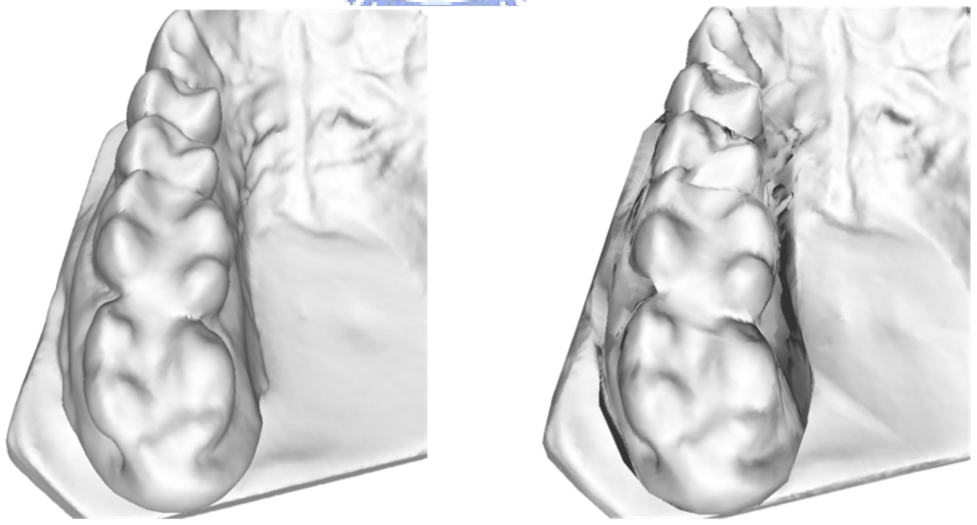


Figure 4.12: Rendering teeth in view 2

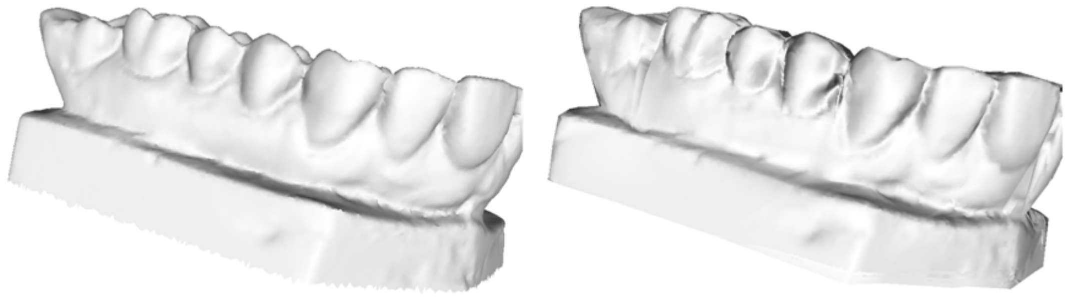


Figure 4.13: Rendering teeth in view 3

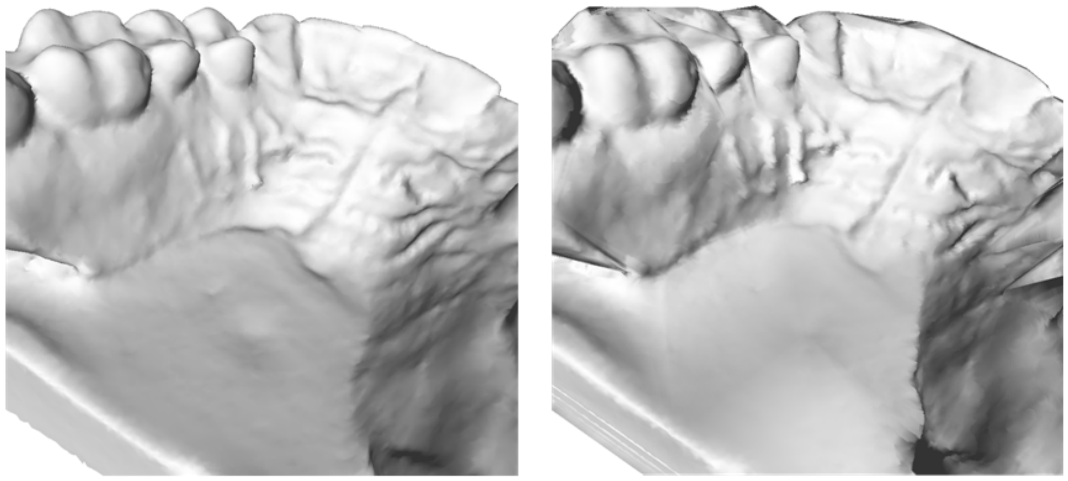


Figure 4.14: Rendering teeth in view 4

4.3.2 Venus head

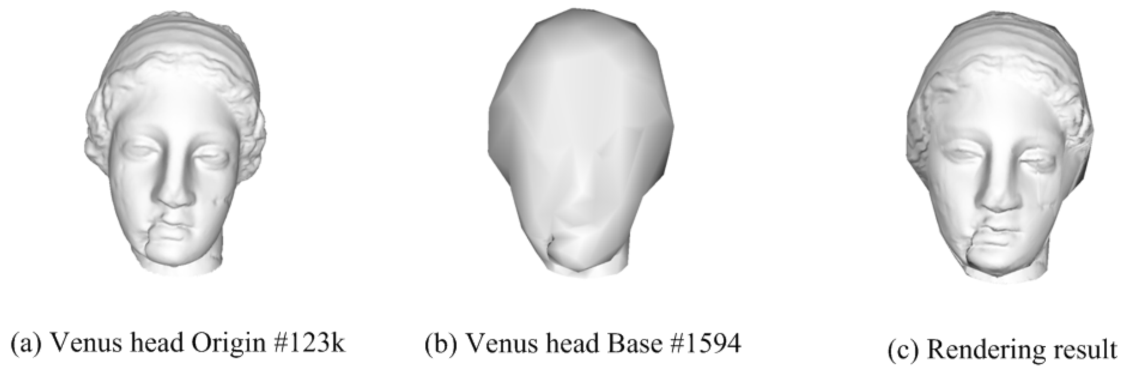


Figure 4.15: Results of venus head

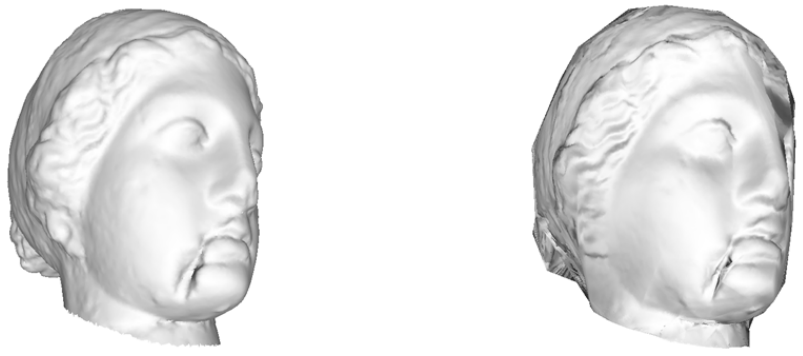


Figure 4.16: Rendering venus head in view 1

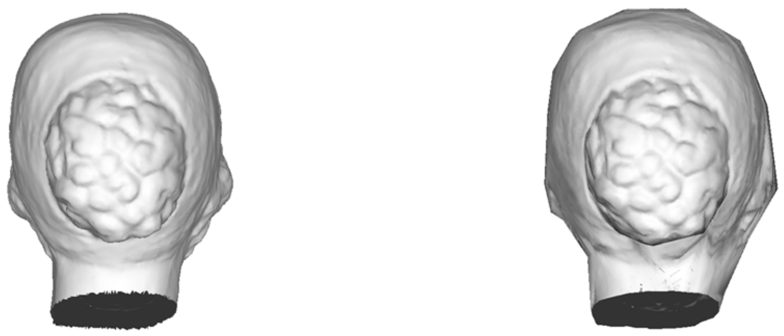


Figure 4.17: Rendering venus head in view 2

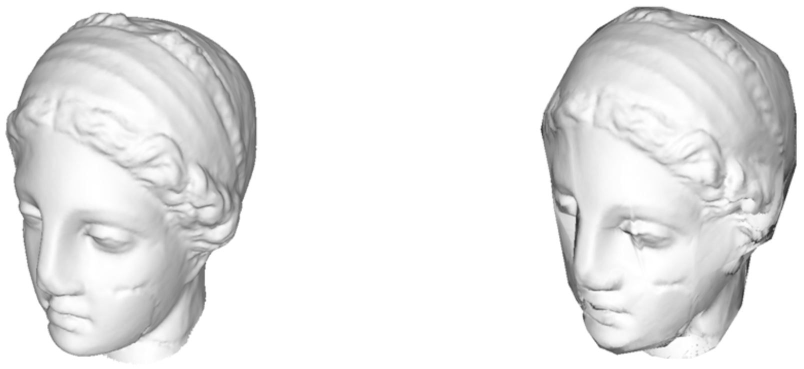


Figure 4.18: Rendering venus head in view 3

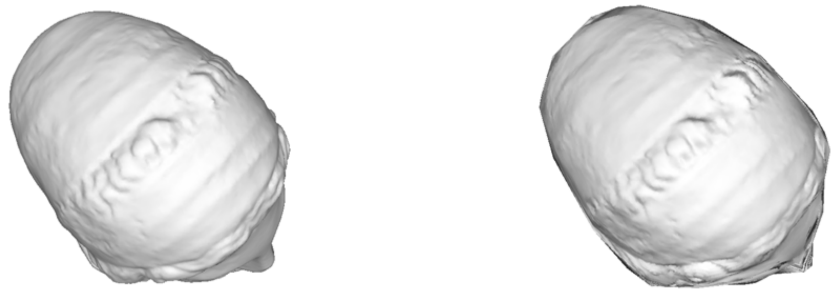
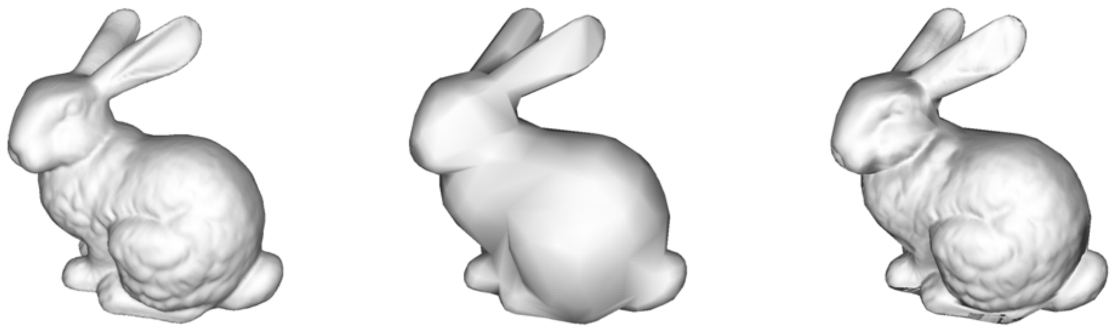


Figure 4.19: Rendering venus head in view 4

4.3.3 Bunny



(a) Bunny Origin #69k (b) Bunny Base #998 (a) Rendering Result

Figure 4.20: Results of bunny

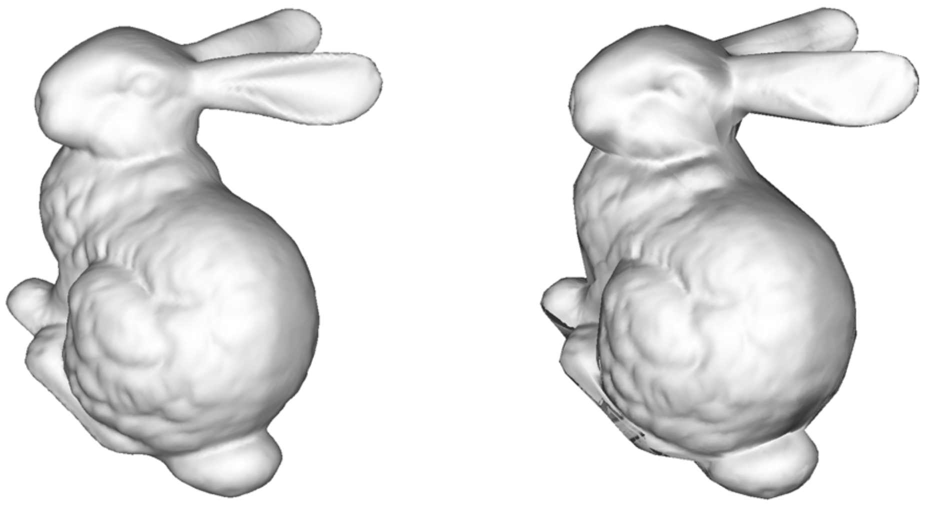


Figure 4.21: Rendering bunny in view 1

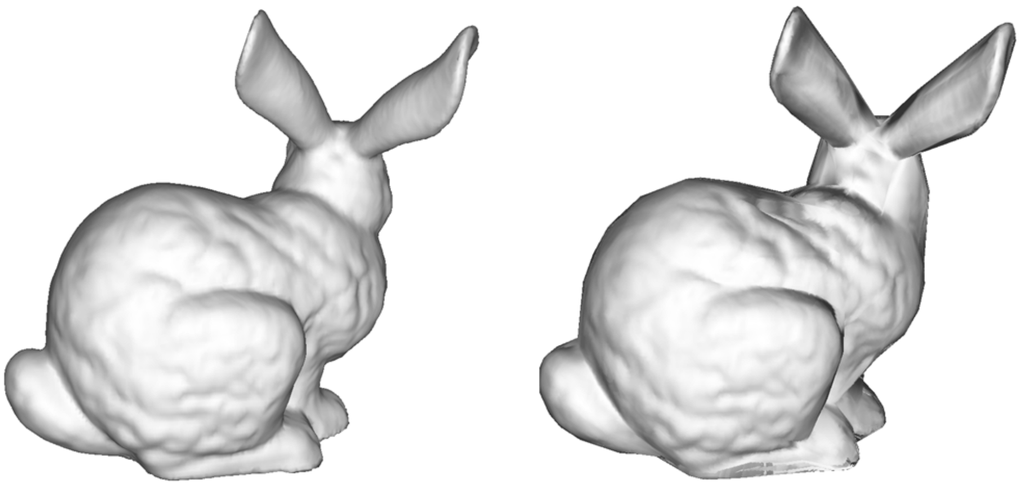


Figure 4.22: Rendering bunny in view 2

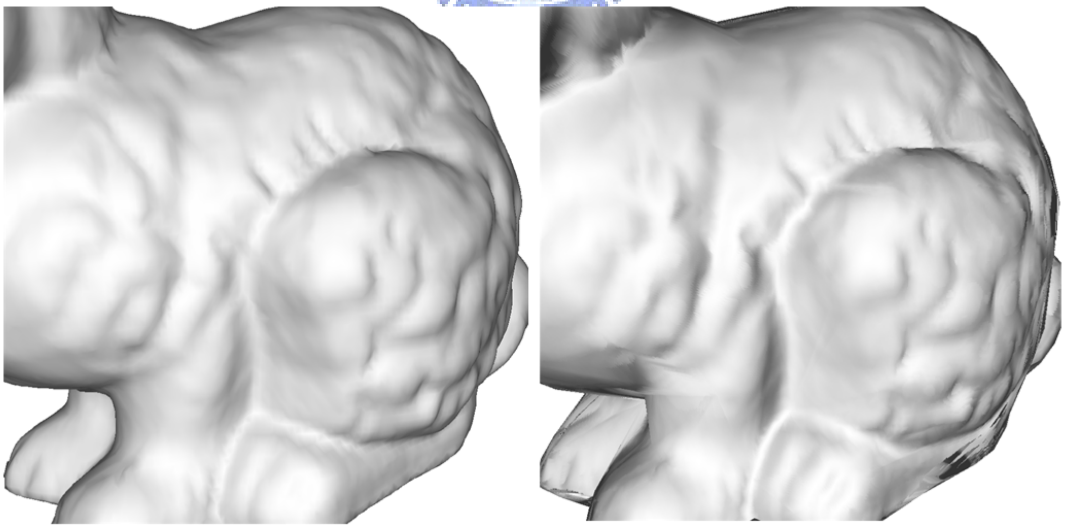


Figure 4.23: Rendering bunny in view 3

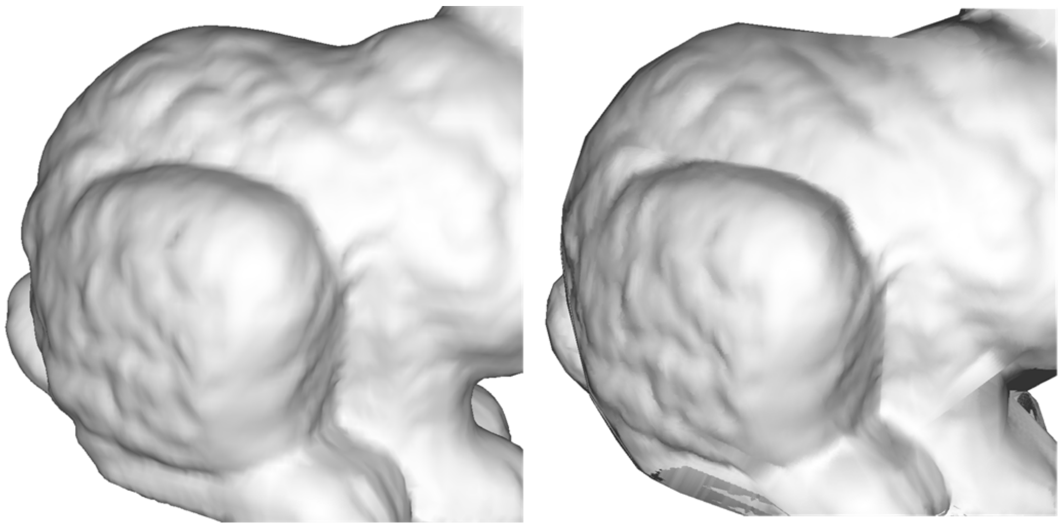


Figure 4.24: Rendering bunny in view 4

4.3.4 Armadillo

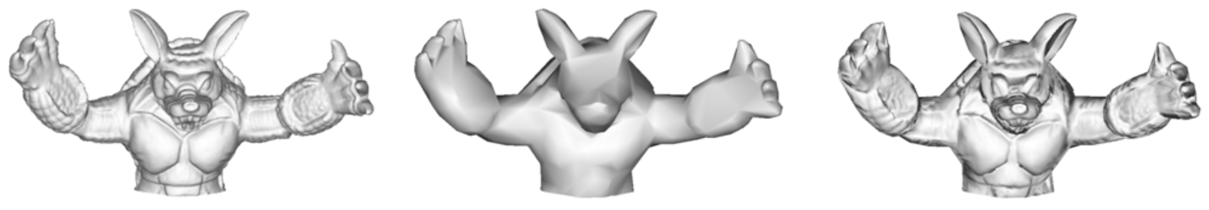

(a) Armadillo Origin #210k (b) Armadillo Base #1343 (c) Rendering result

Figure 4.25: Results of armadillo

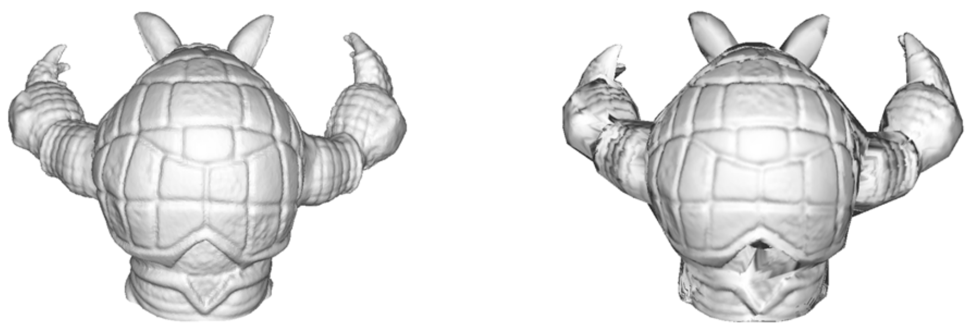


Figure 4.26: Rendering armadillo in view 1

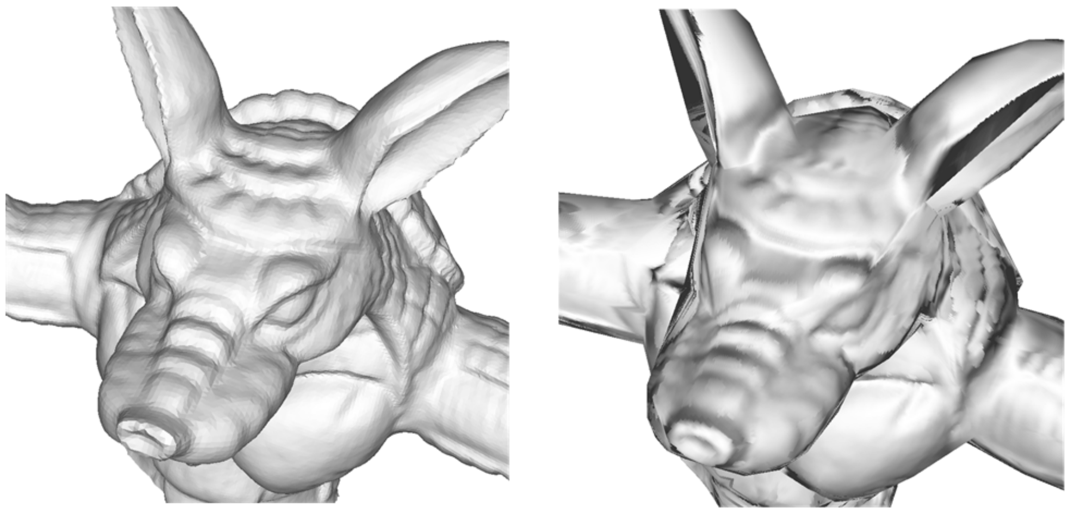


Figure 4.27: Rendering armadillo in view 2

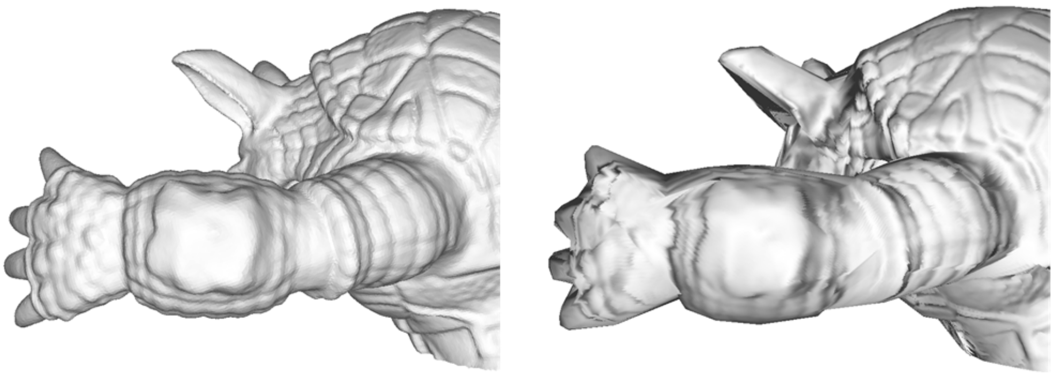


Figure 4.28: Rendering armadillo in view 3

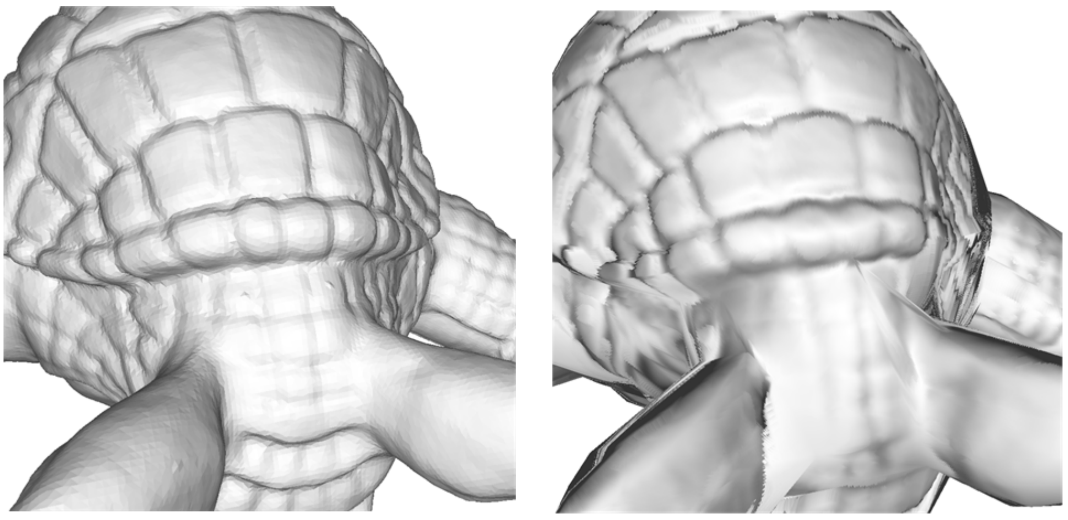


Figure 4.29: Rendering armadillo in view 4

CHAPTER 5

Conclusion and Future Work

5.1 Conclusion

We have developed a platform for constructing a mesh representation with displacement mapping. Our input data is a model with high-detailed surface, and the output is a simplified base mesh of input model with a displacement map to approximate the surface of input model relative to base mesh. Such representation can be rendered using the per-pixel ray-casting methods for displacement mapping[9][10][22][25]. To capture the silhouette of the original mesh, we offset the surface of the original mesh based the concept of *Simplification Envelopes*[4]. Then the mesh is simplified with two constraints for the validation of displacement mapping. We use an area-preserving parameterization to approximate the optimization of a signal distribution on the base surface.

The result mesh representation with the displacement map has the following properties:

1. Large amount of polygons' data is converted to height value of displacement image. The remain mesh topology and parameterization on base domain are simple, make it a compression of input model.

2. The simplified base mesh proposes a global shape and also a tight bounding convex hull

of input model.

3. The encoded surface details can be edited and compressed using any image processing method.

4. This mesh representation support real-time rendering methods with displacement mapping.

5.2 Future work

As regards to a global view of this mesh representation with displacement map. A better approach for using displacement map as a mesh representation is to catch exactly all the surface details of original mesh into displacement map and a rendering method whcih is independent with the curvature of referense surfaces.

The method to precisely catch the surface details is to ensure that every vertex on the original mesh surface maps to exact one texel of the displacement map. This requirement can be satisfied by parameterizing both the original mesh and the base mesh into a same map. But the resolution of this map should be high enough to map each original vertex to a texel. And the mapping constructed by texel correpondance should be a continuous height function for a valid displacement mapping.

For the simplification of base mesh there may be a simplification error metric designed for optimizing the displacement signals referencing to base surface. This error metric will guide the order of simplification sequence to maintain a smooth displaced surface structure mapping to the base face. Moreover, the simplification combined with the surface offsetting can be replaced with a global modeling method in a point-based or volumetric architecture.

In the parameterization stage we apply [27] under the assumption that the amount of signal covered by a base face is proportion to its area. For the geometry of displaced surface has been converted to displacement signals of the base mesh. There is signal-specialized parameterization[21] developed to minimize the stretch of non-linear signals on the base face.

And by the proposed approach[3] we have an alternative choice to parameterize the base mesh in its finest resolution and adapt the distributions of displacement signals after each simplification step.

There are alternative methods which transform surface details into geometry signals in the frequency domaini and use wavelet decomposition to seperate the surface details with meshes' global shape. Analyzing the surface details as frequency signals benefits a level-of-detail construction of the mesh representation with displacement map.

The rendering methods[9][19][25] with displacement mapping generate good rendering results in real time. But since the ray-tracing process is transformed to the texture space not within object space. Rendering distortions caused by this transformation rise when the tracing of view-ray cross from a base face's tangent space into another base face's tangent space. Although we use the linear interpolated transformation to avoid this jiggling on the rendering result, the viewing distortion is still apparent when the tracing is along a winding curve on the surface. An adaptive searching step size and searching direction along the view-ray may be a better solution of this problem.

Bibliography

[1] M. Botsch and L. Kobbelt. Multiresolution surface representation based on displacement volumes. In *Proc. EUROGRAPHICS*, 2003.

[2] E. Catmull and J. Clark. Recursively generated b-spline surfaces on arbitrary topological meshes. In *Computer Aided Design*, 1978.

[3] C.-C. Chen and J.-H. Chuang. Texture adaptation for progressive meshes. In *Proc. Eurographics*, 2006.

[4] J. Cohen, A. Varshney, D. Manocha, G. Turk, H. Weber, P. K. Agarwal, F. P. Brooks, Jr., , and W. V. Wright. Simplification envelopes. In *Proc. ACM SIGGRAPH*, pages 119–128, 1996.

[5] G. Collins and A. Hilton. Mesh decimation for displacement mapping. In *Proc. Eurographics*, 2002.

[6] R. L. Cook. Shade trees. In *Computer Graphics (Proceedings of SIGGRAPH 84)*, 1984.

[7] W. DONNELLY. *Per-Pixel Displacement Mapping with Distance Functions*, chapter 8. M. Pharr, Ed., Addison-Wesley, 2005.

[8] M. Eck, T. DeRose, T. Duchamp, H. Hoppe, M. Lounsbery, and W. Stuetzle. Multiresolution analysis of arbitrary meshes. In *Proc. 22nd annual conference on Computer graphics and interactive techniques*, pages 173–182, 1995.

[9] P. F., O. M. M., and C. J. Realtime relief mapping on arbitrary polygonal surfaces. In *Proc. ACM Symposium on Interactive 3D Graphics and Games.*, pages 155–162, 2005.

[10] P. F., O. M. M., and C. J. Relief mapping of non-height-field surface details. In *Proc. ACM Symposium on Interactive 3D Graphics and Games.*, pages 55–62, 2006.

[11] M. S. Floater. Mean value coordinates. In *Comp. Aided Geom. Design 20*, pages 19–27, 2003.

[12] 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*, pages 209–216, Aug. 1997.

[13] X. Gu, S. J. Gortler, and H. Hoppe. Geometry images. In *Proc. ACM Transactions on Graphics*, pages 355–361, July 2002.

[14] H. Hoppe. Progressive meshes. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 99–108, Aug. 1996.

[15] M. HUSSAIN, Y. OKADA, and K. NIIJIMA. An efficient method for converting polygonal models into displaced subdivision representation. In *Proc. IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, pages 807–816, 2006.

[16] J. Kim and S. Lee. Transitive mesh space of a progressive mesh. In *IEEE Transaction on Visualization and Computer Graphics*, pages 463–480, Dec. 2003.

[17] V. Krishnamurthy and M. Levoy. Fitting smooth surfaces to dense polygon meshes. In *Proc. SIGGRAPH 96, Computer Graphics, Annual Conference Series,*, pages 313–324, 1996.

[18] A. Lee, H. Moreton, , and H. Hoppe. Displaced subdivision surfaces. In *Proc. 27th annual conference on Computer graphics and interactive techniques,*, pages 85–94, 2000.

[19] O. M. M. and P. F. An efficient representation for surface details. Technical Report 351, UFRGS, 2005.

[20] M. Marinov and L. Kobbelt. Automatic generation of structure preserving multiresolution models. In *EUROGRAPHICS*, page 479, 2005.

[21] P. V. Sander, O. J. Gortler, J. Snyder, and H. Hoppe. Signal-specialized parametrization. In *Proc. 13th Eurographics workshop on Rendering.*, pages 87–98, 2002.

[22] N. Tatarchuk. Dynamic parallax occlusion mapping with approximate soft shadows. In *Proc. ACM Symposium on Interactive 3D Graphics and Games.*, pages 63–69, 2006.

[23] L. Wang, X. Wang, X. Tong, S. Lin, S. Hu, B. Guo, and H.-Y. Shum. View-dependent displacement mapping. In *Proc. ACM Transactions on Graphics (TOG).*, pages 334–339, 2003.

[24] X. Wang, X. Tong, S. Lin, S. Hu, B. Guo, and H.-Y. Shum. Generalized displacement maps. In *Proc. Eurographics Symposium on Rendering.*, 2004.

[25] T. Welsh. Parallax mapping with offset limiting: A perpixel approximation of uneven surfaces. Technical report, Infiscape Corporation, 2004.

[26] D. Xu, W. Chen, H. Zhang, and H. Bao. Multi-level differential surface representation based on local transformations. In *Proc. The Visual Computer*, pages 493–505, 2006.

[27] S. Yoshizawa, A. Belyaev, and H.-P. Seidel. A fast and simple stretch-minimizing mesh parameterization. In *Proc. IEEE. Shape Modeling Applications*, pages 200–208, 2004.