

# 國立交通大學

## 多媒體工程研究所

### 碩士論文

利用最小周長函數之三維網格骨架擷取

3D Mesh Skeleton Extraction using Minimum Slice  
Perimeter Function

研究生：紀弘軒

指導教授：莊榮宏 教授

中華民國九十八年十月

利用最小周長函數之三維網格骨架擷取  
3D Mesh Skeleton Extraction using Minimum Slice Perimeter function

研究生：紀弘軒

Student : Hong-Xuan Ji

指導教授：莊榮宏

Advisor : Jung-Hong Chuang

國立交通大學  
多媒體工程研究所  
碩士論文



Submitted to Institute of Multimedia Engineering  
College of Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Computer Science

October 2009

Hsinchu, Taiwan, Republic of China

中華民國九十八年十月

# 3D Mesh Skeleton Extraction using Minimum Slice Perimeter function

Student: Hong-Xuan Ji

Advisor: Dr. Jung-Hong Chuang

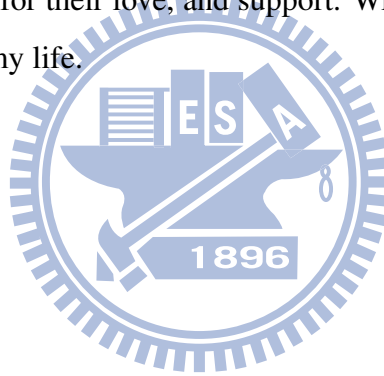
Institute of Multimedia Engineering  
College of Computer Science  
National Chiao Tung University



We propose a novel algorithm to extract curve-skeletons from 3D meshes using the minimum slice perimeter (MSP) function. The MSP function is a scalar surface function to measure the local volume information. Through the MSP function, we can find a potential skeleton position to each vertex. We can transform the input mesh into a so called skeleton mesh whose shape is close to a skeleton by moving each vertex to its corresponding potential skeleton positions. After we obtain the skeleton mesh, we apply the LOD simplification to reduce the skeleton mesh. Our LOD simplification aims to preserve the global shape of the skeleton mesh, rather than local shape and features in traditional LOD simplification. The proposed algorithm is simple yet effective in generating reasonably good curve skeletons that have no branches due to surface noise.

## Acknowledgments

I would like to thank my advisors, Professor Jung-Hong Chuang and Sai-Keung Wong, for their guidance, inspirations, and encouragement. I am also grateful to my senior colleague, Tan-Chi Ho, for his advices and suggestions on my thesis research. And then I want to express my gratitude to all my colleagues in CGGM lab: Jau-An Yang discusses with me on research issues and helps me figure out the questions, Cheng-Min Liu and Ta-En Chen help me to solve mathematics problems and give me necessary informations, Tsung-Shian Huang help me understand much rendering knowledge, and all my junior colleagues' kind assistances. Lastly, I would like to thank my parents for their love, and support. Without their love, I could not pass all the frustrations and pain in my life.



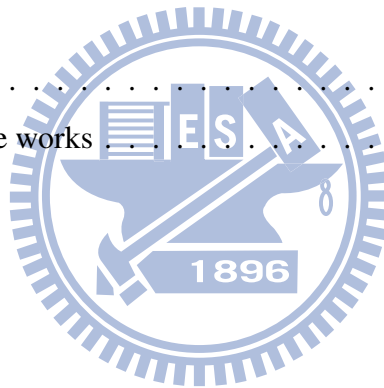
---

# Contents

---

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Contribution . . . . .	3
1.2	Organization of the thesis . . . . .	3
<b>2</b>	<b>Related Work</b>	<b>4</b>
2.1	Medial axis/surface . . . . .	4
2.2	Skeleton algorithms . . . . .	6
2.2.1	Volumetric approaches . . . . .	7
2.2.2	Geometric approaches . . . . .	10
<b>3</b>	<b>Minimum Slice Perimeter Function</b>	<b>12</b>
3.1	Introduction . . . . .	12
3.2	Error of MSP Slices . . . . .	15
<b>4</b>	<b>Skeleton Extraction using MSP Function</b>	<b>21</b>
4.1	Overview . . . . .	22
4.2	Skeleton Mesh Computation . . . . .	22
4.2.1	Repulsive Force Field Modeling . . . . .	23
4.2.2	Deriving skeleton mesh . . . . .	23
4.3	Skeleton Mesh Simplification . . . . .	27

4.3.1	LOD framework . . . . .	27
4.3.2	Error metrics . . . . .	29
4.3.3	LOD Constraint . . . . .	32
4.3.4	Replacement of the skeleton vertex . . . . .	34
<b>5</b>	<b>Results</b>	<b>35</b>
5.1	MSP Computation Results . . . . .	35
5.2	Preprocessing Time . . . . .	38
5.3	Skeleton Mesh Representation . . . . .	40
5.4	Skeleton Results and comparison . . . . .	43
<b>6</b>	<b>Conclusions</b>	<b>49</b>
6.1	Summary . . . . .	49
6.2	Limitations and Future works . . . . .	50
	<b>Bibliography</b>	<b>51</b>



---

# List of Figures

---

2.1	Medial Axis/Surface [CSM07] . . . . .	5
2.2	Medial axis by Grass-fire analogy [ABE07] . . . . .	5
2.3	Medial axis computation using Delaunay triangulation [RT95] . . . . .	6
2.4	Directional thinning method . . . . .	7
2.5	The model to generate the electrostatic field. [GAHY96] . . . . .	9
3.1	MSP slicing representation. . . . .	13
3.2	MSP results with color-coding. . . . .	14
3.3	Terminal Deviation Explanation . . . . .	15
3.4	MSP Slicing by the normal influence. . . . .	15
3.5	Slice error distribution with color-codings . . . . .	16
3.6	Normal Cone Representation. . . . .	17
3.7	Comparison of slice error distribution of horse model. . . . .	19
3.8	MSP/Refined MSP distribution charts for the horse model. . . . .	19
3.9	Distribution charts of the slice error for the horse model. . . . .	20
4.1	System overview. . . . .	22
4.2	Repulsive force field of a cow model [CSYB05] . . . . .	24
4.3	Locating the skeletal position . . . . .	25
4.4	Transformed Skeleton Mesh using the repulsive force model. . . . .	26

4.5	Pseudo code of LOD simplification . . . . .	28
4.6	Slice error distribution with color-coding. . . . .	31
5.1	MSP computation results of different amounts of slices . . . . .	36
5.2	MSP distribution of models . . . . .	37
5.3	Cone-refined MSP distribution of models . . . . .	38
5.4	Skeleton meshes of horse model . . . . .	40
5.5	Skeleton meshes of camel model . . . . .	40
5.6	Skeleton meshes of Raptor model . . . . .	41
5.7	Skeleton meshes of Fertilty model . . . . .	41
5.8	Skeleton meshes of Armadillo model . . . . .	41
5.9	Skeleton meshes of Heptoroid model . . . . .	42
5.10	Skeleton meshes of Dancing Children model . . . . .	42
5.11	Skeleton meshes of Gargoyle model . . . . .	42
5.12	Skeleton results of horse model . . . . .	43
5.13	Skeleton results of Camel model . . . . .	44
5.14	Skeleton results of Dog model . . . . .	44
5.15	Skeleton results of Raptor model . . . . .	44
5.16	Skeleton results of Elk model . . . . .	45
5.17	Skeleton results of Fertilty model . . . . .	45
5.18	Skeleton results of Octopus model . . . . .	45
5.19	Skeleton results of Armadillo model . . . . .	46
5.20	Skeleton results of Heptoroid model . . . . .	46
5.21	Skeleton results of Dancing children model . . . . .	46
5.22	Skeleton results of Neptune and Dancer . . . . .	47
5.23	Skeleton results of horse model by using different saturation . . . . .	47
5.24	Skeleton results of Fertilty model by using different saturation . . . . .	48
5.25	Skeleton results of Camel model by using different saturation . . . . .	48



6.1 Spatial Coherence refined skeleton mesh. . . . . 51

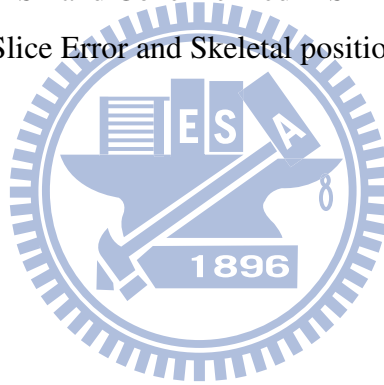


---

# List of Tables

---

5.1	MSP computation time of different amount of slice . . . . .	37
5.2	Computation time of MSP and Cone-Refined MSP . . . . .	39
5.3	Computation time of Slice Error and Skeletal positions . . . . .	39

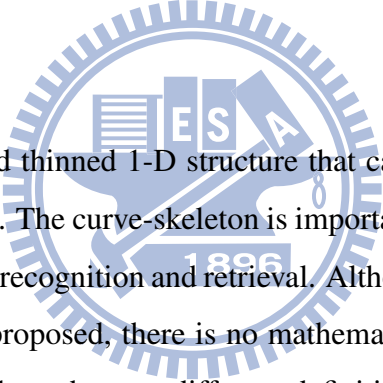


# CHAPTER 1

---

## Introduction

---



Curve-skeleton is a compact and thinned 1-D structure that captures the topological structure and global shape of a 3-D model. The curve-skeleton is important in many applications, such as the animation, morphing, model recognition and retrieval. Although several desirable properties of the 3D skeletons have been proposed, there is no mathematical definition formulated. Due to that, there exist many algorithms that use different definitions, functions, and heuristics to compute skeletons. To be a reasonable skeleton, there are several properties that need to possess, including the homotopy (topology-preserving), centeredness, thinness, component-wise, the smoothness and connected. Corea et al. discuss the properties of the curve-skeleton and conduct a great survey on the skeleton extraction algorithms in the recent ten years [CSM07].

The extraction of curve-skeleton have been researched for more than ten years. There are various skeleton extraction algorithms that apply different skeleton properties and operate in different spaces. The algorithms of volumetric approach operate in voxel space. The thinning algorithm generate a skeleton by iteratively removing boundary voxels [KR89, GB90, SC94]. The algorithms of filed function approach analyze the force field and find the potential skeleton nodes as the extremes of the field [BSB<sup>+</sup>00, AHY94]. Bitter et al. not only construct the

---

distance field but propose the penalized values on the distance field [BKS01] to generate more centered skeleton when using Dijkstra algorithm to connect the skeleton nodes. Because the computation of distance field is non-linear, the electrostatic field functions are proposed to simulate the distance field function [AHY94]. In the continuous space, several methods have been proposed to generate skeletons by directly processing the mesh, for example, approaches based on Reeb Graph [TVD08], Voronoi diagram [DS06], and Laplacian smoothing [ATC<sup>+</sup>08]. By using Laplacian smoothing, one obtains a zero-volume mesh which looks like a skeleton and then simplify the mesh to generate the skeleton [ATC<sup>+</sup>08]. To generate a skeleton during the model reconstruction process is a special approach in the continuous space [SLSK07].

In the previous work, many skeleton extraction algorithms are sensitive to noise and easily generate many unwanted branches, especially the algorithms based on the medial axis. There are still several algorithms that are noise insensitive, such as the potential field approach [AC97, AHY94] and the one using Laplacian smoothing [ATC<sup>+</sup>08].

In this thesis, we propose a skeleton extraction algorithm that applies the MSP function and operates on the mesh structure. The MSP function is a scalar surface function that represents the local volume information of the model. For each vertex, we compute its MSP value, and on the slice from which the MSP value is derived we use Repulsive Force Model [CTK00] to find a proper position where we expect the skeleton passes through. We then move each vertex to its corresponding skeleton point and thus perform the original mesh into a skeleton-like mesh that is nearly zero-volume. Next, we simplify the skeleton mesh by the LOD simplification. In the LOD simplification, metrics are designed to preserve global shape of the skeleton mesh rather than the surface detail and features normally used in the traditional LOD simplification. Properties of MSP function is used to design the error metrics and the constraint. Through the carefully crafted metrics and constraints, our LOD simplification preserve the global shape of skeleton mesh and generate reasonably good skeletons.

## 1.1 Contribution

The contributions of our skeleton extraction can be summarized as follows:

- Produce reasonably good results with no the unwanted branches.
- Invariant under isometric transformations.
- Take less than ten seconds to extract skeleton in the LOD simplification and less than three minutes to compute the MSP functions of the 3D mesh whose face number is about ten thousand.

## 1.2 Organization of the thesis

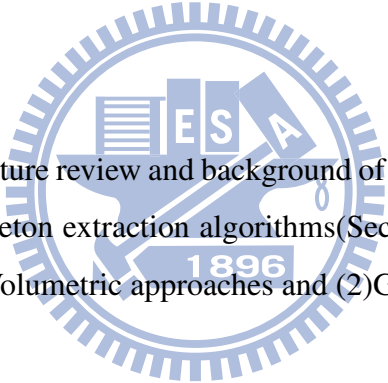
The following chapters are organized as follows. Chapter 2 gives the literature review and the background knowledge. Chapter 3 introduces the minimum slice perimeter (MSP) function and its properties. Chapter 4 describes our skeleton extraction algorithm, including the overview, the skeleton mesh computation, and the LOD simplification framework. Chapter 5 shows the experimental results of our skeleton extraction algorithm and comparison to Laplacian smoothing algorithm [ATC<sup>+</sup>08]. Finally, we summarize our skeleton extraction algorithm and discuss its limitations and future works in Chapter 6.

## CHAPTER 2

---

# Related Work

---



In this chapter, we give the literature review and background of the skeleton included the medial axis(Section 2.1) and other skeleton extraction algorithms(Section 2.2). These algorithms can be classified into two types:(1)Volumetric approaches and (2)Geometric approaches.

### 2.1 Medial axis/surface

Medial axis [Blu67] was proposed by Blum as a tool in image analysis. It is a kind of data structure to describe the topology of different objects. The Medial axis(MA) of the 2-D shape  $M$  is a set of a curve segments that is defined by the centers of all inscribed discs which have at least two closest points on the boundary of the shape  $M$ . The medial axis of 3D object are the 2D planes defined from the centers of the inscribed balls, as show in Figure 2.1. Therefore, the medial axis of the 3-D object is also called the medial surface. The skeleton is a curve segments defined by the centers of the maximal inscribed balls. So the skeleton is a subset of the medial axis.

To compute the medial axis, we can apply the grass-fire analogy [LL92]. Firstly, the object

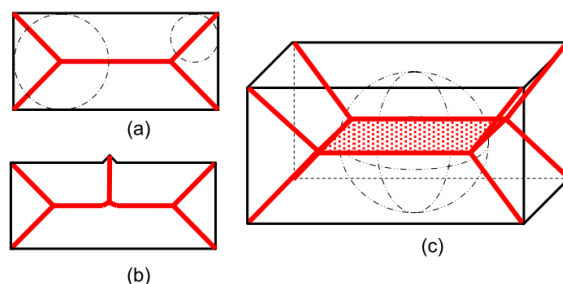


Figure 2.1: Medial Axis/Surface [CSM07]

are entirely full of grass and we ignite the grass on the boundary at same time. Then the medial axis is the set of locations where the fire fronts meet each other. In other words, the medial axis is the set of points that have at least two closet points on the boundary as shown in Figure 2.2. For a geometric shape in  $R^k$ , the medial axis of the shape must be dimension  $k - 1$ .

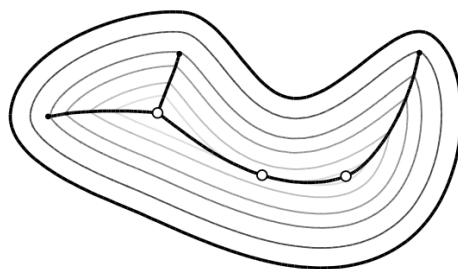


Figure 2.2: Medial axis by Grass-fire analogy [ABE07]

Using the Voronoi diagram to compute the medial axis is a popular approach [OI92, OK95, ACK00]. Voronoi diagram(VD) is a special kind of decomposition of a metric space determined by distances to a specified discrete set of objects in the space. Besides the Voronoi diagram, the Delaunay triangulation which is the dual graph of the VD is often used to compute the medial axis accompanied with VD too. Reddy [RT95] propose a Voronoi-based medial axis algorithm that generate the Voronoi diagram(VD) by sampling on the boundary of the object and check the delaunay edge of the VD whether it is positioned completely inside the object. If so, this edge is a part of the medial axis , as shown in Figure 2.3(a)(b). Figure 2.3(d) shows the medial surface of a box. We know the medial axis is complicated and gets some redundancy. For some

applications such as model retrieval, we do not need the exact medial axis. Therefore, there are many algorithms proposed to compute the approximated medial axis (AMA) that is more practical for some specific applications [DZ04, YBM04]. Dey [DZ04] proposes two criteria to remove some Voronoi cells which are the less important. By removing redundant cells, we get an approximated medial axis by the simplified Voronoi diagrams.

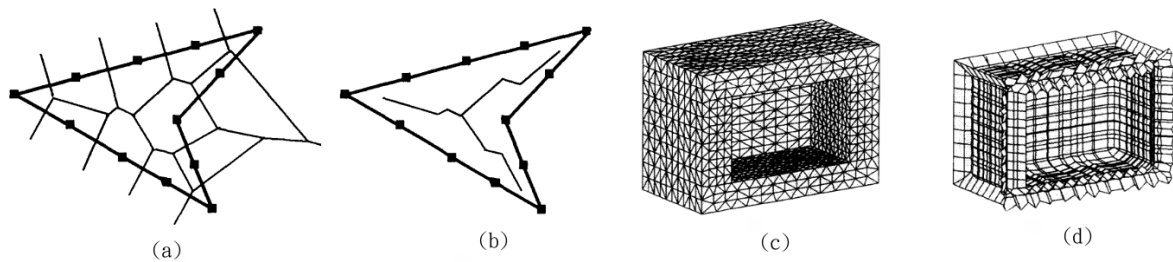


Figure 2.3: Medial axis computation using Delaunay triangulation [RT95]

The major disadvantage of the medial axis/surface is that it is sensitive to the small changes on the object's boundary. An example is shown in the Figure 2.1(b). The little change on the boundary cause the large change on the medial axis. Hence, to design a noise free algorithm or to filter the noise out is the one of the most important goals in these algorithms. We should also be careful to the influence of noise and understand the intrinsic sensitivity to noise in all the medial axis algorithms.

## 2.2 Skeleton algorithms

This chapter gives the review of the skeleton extraction algorithms. We classify the algorithms into two categories, including the volumetric approaches and geometric approach. The algorithms of volumetric approach operate in the discrete space and the geometric approaches are in the continuous space.



### 2.2.1 Volumetric approaches

Many skeleton algorithms are proposed for discrete 3-D data in volumetric approach. We categorize the volumetric algorithms into three types: boundary thinning method, distance field method, and the potential field method.

**The Boundary Thinning Method** generates a skeleton by iteratively removing the boundary voxels of the object until the required thinness is achieved. The criteria for removing boundary voxels are based on the simple points [Mor81]. Removing voxels of such points will not change the topology of the object. At each removing operation, the neighbourhood of the voxel will be localized and checked whether or not the voxel is simple points. The topology preserving conditions are implemented as a series of templates. At runtime, any boundary voxel that fits any one on the templates will be removed. The templates are usually the square cubic of size  $3 \times 3 \times 3$ , as shown in Figure 2.4(a).

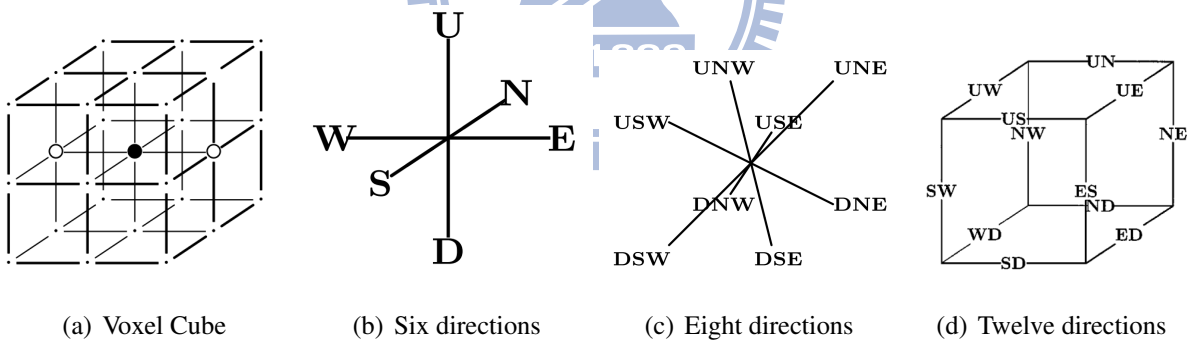


Figure 2.4: Directional thinning method

The most important factor in the boundary thinning method is the analysis of the simple points and building the removing templates. The basic requirement of simple point is that the number of the connected components and the holes of the object do not change after removing the voxel. There are much research work have been proposed on the simple point [KR89, Kon97, SC94, LB01]. At each removing operation, if we find all the simple points on the boundary and remove all of them at a time, we may lead to a disconnected object. Because

the neighborhood of these simple points might overlap and the neighborhood condition of the simple points changes after the deletion, we cannot remove all the simple points on the boundary at a time. Therefore, there are several types of thinning methods based on how simple points are detected and removed. The first method is to classify the  $3 \times 3 \times 3$  voxel cube into different directions and divide the removing operation into several sub-iterations according to the number of directions, such as six directions [GB90, PK98], eight directions [PK99a], and twelve directions [PK99b, LB01]. Figure 2.4 shows the different directions of the thinning method. The second method is to classify the voxel cube into several sub-fields and remove accordingly [BA95, MWL02].

**Distance Field Method** uses the distance from boundary to compute skeleton. The distance field  $D(P)$  is defined as the minimum distances between the interior voxels  $P$  to the boundary  $B(O)$  of the object  $O$  in Equation 2.1. The skeleton algorithms generate the distance field of the object first. After that, most of the distance field-based algorithms can be divided into two steps, including find the ridge points and then connect these points together. The ridge voxels of the distance field mean the local minimum, local maximum, and the saddle point. The ridge points are also the candidate nodes of the skeleton and it locates at the center within the object. Most of the distance field methods attempt to find these ridge voxels and connect the points together to generate a skeleton.

$$D(P)_{P \in O} = \min_{N \in B(O)} (d(P, N)) \quad (2.1)$$

Bitter et al. [BKS01, BSB<sup>+</sup>00, SBB<sup>+</sup>00] propose the penalized distance field, which add more penalty costs to the voxels that is more closer to the boundary. After locating the ridge points, they use the penalized distance field to connect the ridge points together by using the Dijkstra algorithm and generate the entire skeleton. Employing minimum spanning trees accompanied with distance field is another approach [WDK01]. The distance field can also be used as a priority function in a thinning algorithm [Pud98]. Couprie et al. [CCZ07] use the distance field to calculate the bisector function and remove voxels according to the bisector

function in the thinning algorithm.

**Potential Field Method** uses another form of field function to simulate the properties of the distance field and compute the skeleton. Because of the non-linear nature of shortest distance computation, some algorithms use potential field functions [AC97, AHY94] to replace the functions of distance field. The potential of interior points is the sum of the potentials generated from the boundary of the object. The local extremes of the potential field function is also located at the center of the object, like the distance field function. We can compute the skeleton by connecting the ridges of the potential field as well.

Grigorishin et al. [AHY94, GAHY96] propose the electrostatic field function to generate the potentials inside the object and the local extremes of the field function are the part of the skeleton. They model the boundary of the object as a set of unit charges and the voxels inside the object as the conductor, as shown in Figure 2.5. Any interior points can be affected by the boundary charges and the amount of the force influenced by the boundary charges is called the potential value which is proportioned to the inverse distance from the boundary.

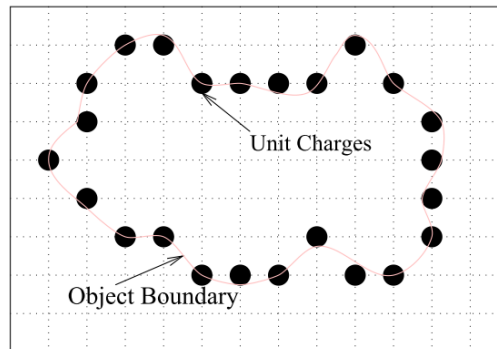


Figure 2.5: The model to generate the electrostatic field. [GAHY96]

Ahuja et al. [AC97] propose the Newtonian potential to compute the skeleton. The Newtonian potential of an interior point  $p$  due to a boundary charge  $c$  is defined as the  $\frac{1}{Dist(p,c)}$ , where  $Dist(p,c)$  is the Euclidean distance between  $p$  and  $c$ . The total potential of a point  $p$  due to all the boundary charges  $B$  is defined as  $\sum_{c \in B} \frac{1}{Dist(p,c)}$ . Besides the Newtonian potential, they also

define the Newtonian force by the partial differential to the Newtonian potential. The Newtonian force gives the direction to the local extremes and we can compute the skeleton by tracing the Newtonian force.

The repulsive force function [CSYB05, LWM<sup>+</sup>03, CTK00] is a kind of potential field function. It assumes the boundary carries electric charges and each point inside the object get a directional repulsive force from the charges on the boundary. The force is attenuated with the distance from the boundary charges. The repulsive force has the similar effect of Newtonian force. By tracing the direction of the repulsive force, we can easily trace the entire skeleton. Wu et al. [WML<sup>+</sup>03] add the visibility test into the computation of the repulsive force. Since not all the interior points should be affected by all the boundary charges, some boundary charges are blocked from the interior points.

Because the intrinsic properties of the potential field method, it is more insensitive to the noise than the distance field method. The algorithms take into account the entire boundary area that creates a averaging effect, so it's less sensitive to noise. By controlling the sampling density on the boundary, they can achieve a trade-off between accuracy and performance.

### 2.2.2 Geometric approaches

The algorithms use the information of the polygonal meshes or scattered point sets to generate skeleton. The algorithms have no clear classifications and standard schemes to generate a skeleton. We introduce the different geometric algorithms briefly as follows.

Using Voronoi diagram is a popular approach to generate medial axis(see Section 2.1). In order to extract the skeleton from the medial axis, Dey et al. [DS06] propose to compute the medial geodesic function(MGF) and its gradient for each medial axis facet. They mark the edges of the medial axis facets with negative divergence of MGF as skeleton edges. Wu et al. [WML<sup>+</sup>06] propose the domain connected graph to compute skeleton. They build the Voronoi diagram that use the vertices to be samples and find the domain balls that is non-intersecting maximal inscribed balls between all Voronoi balls. Then they produce the coarse skeleton path

by connecting the center of the domain balls and use the repulsive force accompanied with the active contour model to refine the skeleton path.

Tierny et al. [TVD08] use Reeb graph to generate a coarse skeleton and refine the Reeb graph using the constrictions to obtain a skeleton. They use the geodesic distance from the nearest feature point to be the mapping function and create the Reeb graph. Then they propose the discrete contour on the Reeb Graph. By analyzing the discrete contour to detect the topological change of the model, they construct the Reeb graph. After finishing the first-stage Reeb graph, they use the constriction of the model to update the Reeb graph and obtain the skeleton.

Sharf et al. [SLSK07] propose to use the deformable model to capture the shape of the object and generate the skeleton during the reconstruction process. The deformable model is originally used to reconstruct the model surface from a point set. The deformable models consist of multiple competing fronts which evolve inside the object in a coarse-to-fine manner. They track the centers of the fronts during the process and filter the resulting arcs, that are connected from the centers, to obtain a curve skeleton of the object.

Shapira et al. [SSCO08] propose the shape diameter function(SDF) and apply it to the segmentation and skeletonisation. They use the fact that SDF approximates twice the distance to the medial axis in most points on the mesh surface. Therefore, they project surface points in an inward normal direction to a distance which is half of their SDF value to generate skeleton points. Then they apply moving least squares (MLS) method to obtain a skeleton curve.

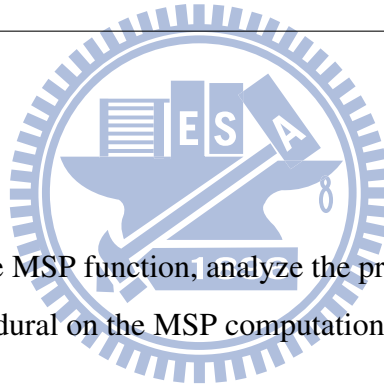
Au et al. [ATC<sup>+</sup>08] apply the Laplacian smoothing to contract the input model into a zero-volume mesh. Then they simplify the zero-volume mesh into a 1-D skeleton by iteratively collapsing the edge with the minimum cost. Tagliasacchi et al. [TZCO09] propose to apply the concept of generalized rotational symmetry axis(ROSA) to extract skeleton from point sets. The skeleton can be regarded as a generalized rotation symmetry axis. By computing the ROSA, the algorithm can handle incomplete point clouds which means some data is missing. They find a cutting plane of each vertex and compute the position of ROSA by the points of the cutting plane. Then they move the point clouds to the positions of ROSA and recover the joint part by Laplacian smoothing. Finally, they apply thinning to generate a skeleton.

## CHAPTER 3

---

# Minimum Slice Perimeter Function

---



In this chapter, we introduce the MSP function, analyze the properties, quantify the MSP error, and develop a refinement procedural on the MSP computations.

### 3.1 Introduction

The *Minimum Slice Perimeter* (MSP) function [HJWC09] defined on a surface vertex is a scalar function aims to measure the local volume. For a manifold surface  $M$ , the MSP function at a surface point  $p$ ,  $f(p) : M \rightarrow R$ , is defined as the minimum perimeter of the planar slices passing through  $p$  and parallel to the surface normal at  $p$ . The 3D plane passing through  $p$  and parallel to the surface normal  $n_p$  at  $p$  is defined by the Equation 3.1.

$$n_p^\perp \cdot x = n_p^\perp \cdot p, \quad (3.1)$$

where  $n_p^\perp$  is perpendicular to  $n_p$ . For each vertex in the mesh  $M$ , the intersection of the plane and  $M$  results in a slice, computed by plane-edge intersection test using a face propagation process. The face propagation process start from the face that incidents to the vertex and find the plane-edge intersection on the face. After finding the first intersection of the face, we iteratively propagates the plane-edge intersection computation to the adjacent faces. The process terminates when the propagating face meets the starting face. We record plane-edge intersection and compute the perimeter of the slice which is the accumulated distance between the plane-edge intersections on the slice.

By generating different slicing planes along the vertex normal, we can get a set of slices and the minimum perimeter of the slices serves as the measure of local volume. We call the slice with minimum perimeter as the *MSP slice* at the vertex. The MSP function is effective for representing the volume information since the perimeter of the planar slice is directly related to the area of the planar slice. Figure 3.1 explains the concepts and show the slice of minimum slice perimeter with thick dots. Figure 3.2 shows the MSP results with color-coding from blue,green,to the red for increasing MSP value.

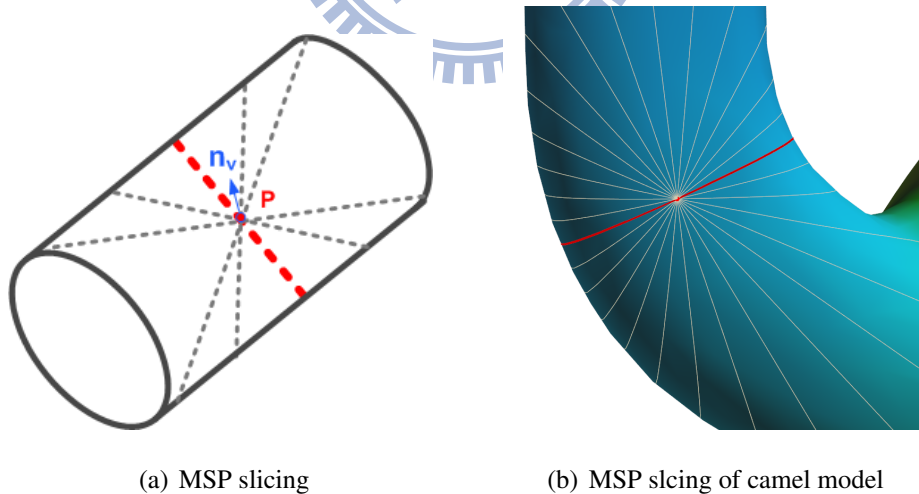
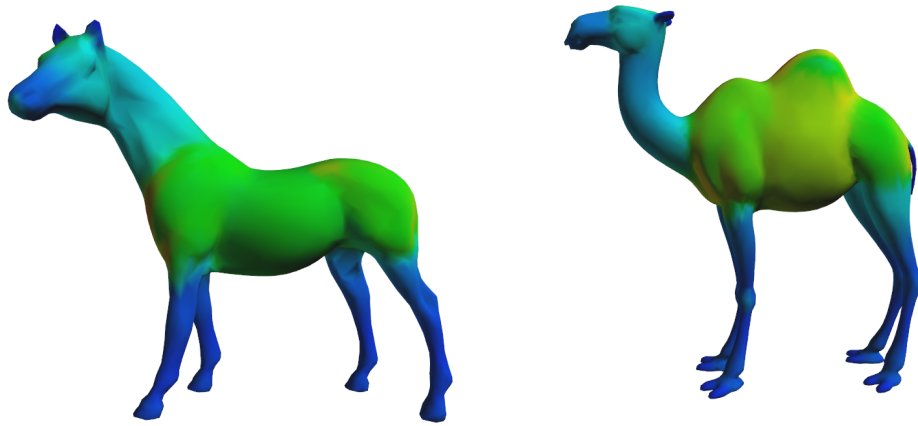


Figure 3.1: MSP slicing representation.



(a) MSP result of horse model

(b) MSP result of camel model

Figure 3.2: MSP results with color-coding.

The MSP slice cuts through the model and divide it into two parts. It is reasonable to say that the MSP slice cuts through the model and intersects with the skeleton at a point inside the slice. The centroid of the slice can be a proper candidate of the skeleton node;but the centroid may lie outside for concave slices. An alternative is the local minimum derived by using Repulsive force model, as shown in the next chapter.

The MSP function exists an intrinsic situation at the terminal part of the input model. The MSP value at a terminal part of a model is usually much larger than that in the other parts. As shown in Figure 3.3, the vertex  $P$  is located at a terminal part of the cylinder and the slice direction in MSP computation is parallel to the long axis of the cylinder. This results in a drastic change of the MSP value when moving a point from the body part to a terminal part. And the slice direction is much inappropriate at the terminal parts. Although the MSP value at point  $P$  is still a valid volume measure, we intuitively expect the slice direction of  $P$  to be perpendicular to the long axis of the cylinder for the purpose of skeleton extraction. We call the intrinsic situation as terminal deviation(TE).



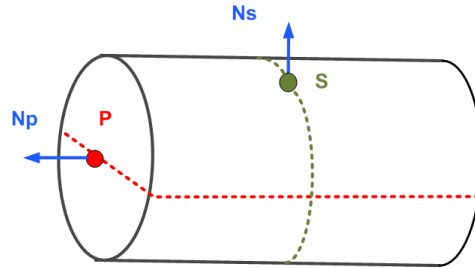
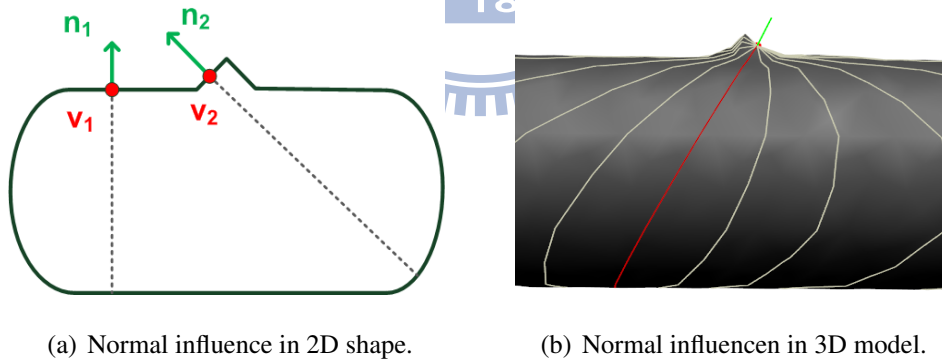


Figure 3.3: Terminal Deviation Explanation

## 3.2 Error of MSP Slices

In the MSP computation, we use the vertex normal to be the slicing direction. But the vertex normal is a local information and can be varied easily by the surface fluctuation. As shown in Figure 3.4(a), the MSP at  $v_1$  is computed properly, while the normal of  $v_2$  is deviated by the surface detail and represents in a wrong slicing direction. The similar situation in 3D is shown in Figure 3.4(b).



(a) Normal influence in 2D shape.

(b) Normal influence in 3D model.

Figure 3.4: MSP Slicing by the normal influence.

In order to alleviate the problem, a smoothing on MSP value is necessary. We first design an error measure on the MSP function and then propose a smoothing operation. The slice error  $E_s$  of a slice is defined to quantify how far is the skeleton node found in the slice deviated from the skeleton.

We denote the MSP slice of vertex  $v$  as  $MS(v)$  and  $V_{MS(v)} = \{p_1, p_2, \dots, p_n\}$  as the set of

points lying on  $MS(v)$ . The slice error  $E(v)$  of the  $MS(v)$  is defined as the sum of the included angles between  $N_S(v)$ , the normal of  $MS(v)$ , and  $N(p)$ , the normal of  $MS(p)$  for all vertices  $p$  on the  $MS(v)$ , as shown in Equation 3.2.

$$E_s(v) = \sum_{p \in V_{MS(v)}} \arccos \left( \frac{N_S(v) \cdot N_S(p)}{\|N_S(v)\| \|N_S(p)\|} \right) \quad (3.2)$$

The slice error  $E_s$  indicates that if the minimum slice normal of the vertices on the MSP slice about the same, this slice is a slice on which a good skeleton point can be derived. Figure 3.5 show the distribution of slice error over the mesh. We can clearly recognize that the slice error on the terminal parts or on the surface details is often large. The slice error reflect the parts where the MSP computation is not consistent with our expectations.

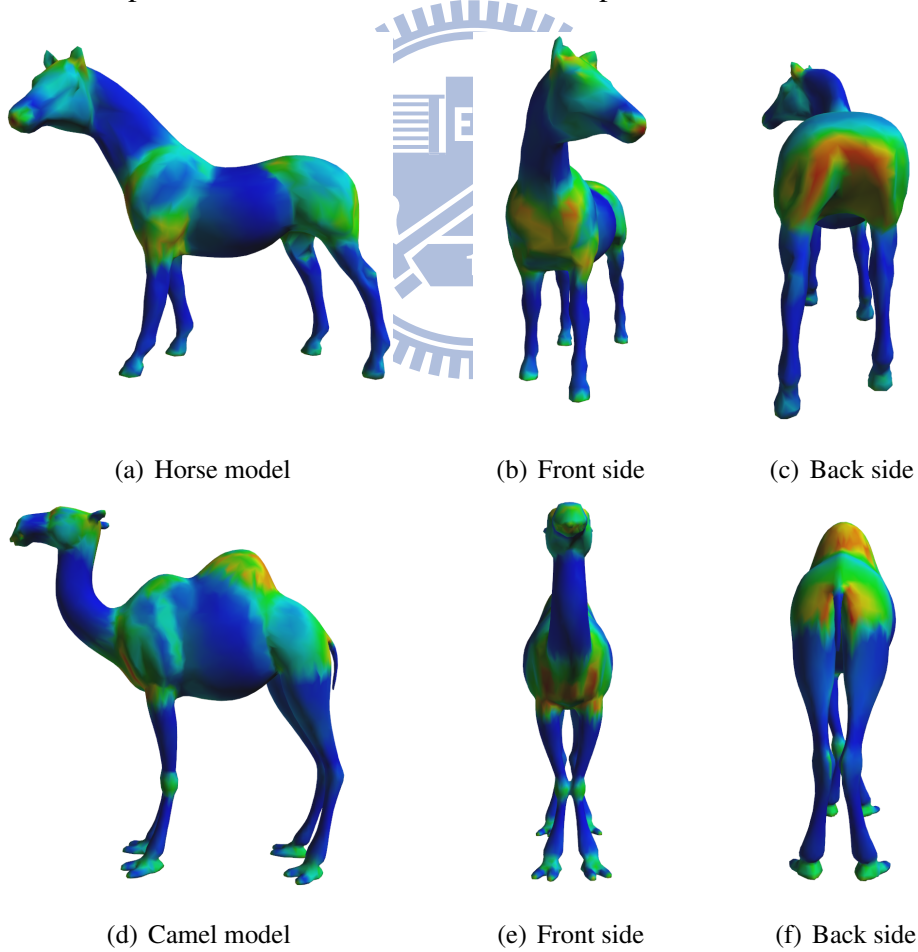


Figure 3.5: Slice error distribution with color-codings

In order to alleviate the influence of surface fluctuations that perturb the computation of MSP function, we need to apply a smoothing procedure on the parts with large slice error. Instead of applying the smoothing operation on the mesh, we are going to smooth the MSP deviation and obtain more accurate MSP slices. We propose to use more samples in the MSP computation of each vertex rather than the single direction sample of the vertex normal. We can obtain the more accurate MSP information by increasing the samples at each MSP computation.

To use more samples in the MSP computation at each vertex, we first create a normal cone that use the vertex normal to be the center axis and vary the angle of the cone according to the value of slice error. Then we randomly create the samples inside the cone. As shown in Figure 3.6(a), the green dash lines enclose a cone with angle  $\theta$ , the red line is the axis of the normal cone, and the gray lines are the sample directions.

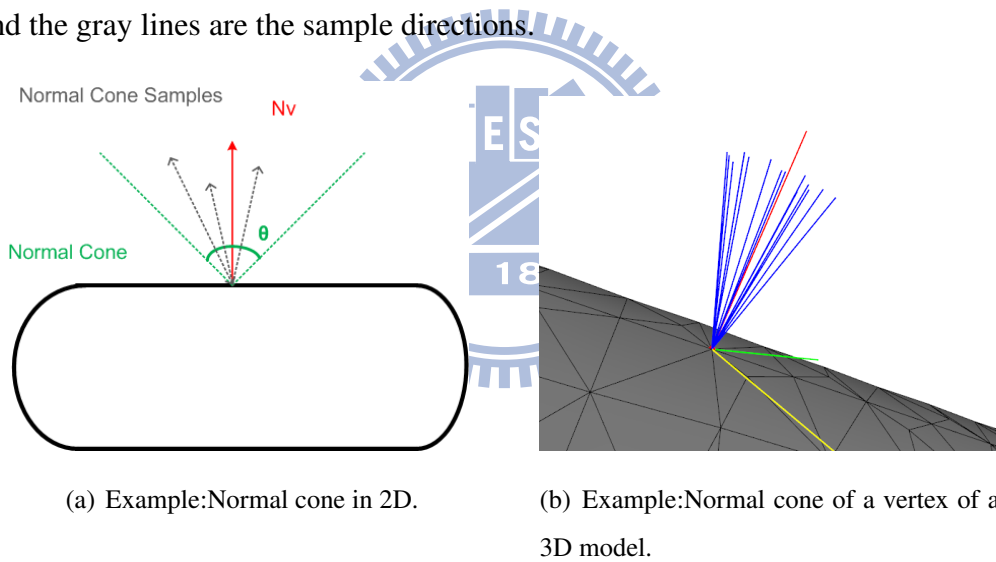


Figure 3.6: Normal Cone Representation.

To generate evenly distributed samples inside the cone, we apply the method proposed in [Gre03]. We first take a pair of independent canonical random numbers  $\xi_x$  and  $\xi_y$  from 0.0 to 1.0. Then map the two random values into the two angles of the spherical coordinates using the Equation 3.3. To limit the angle  $\theta$  to be smaller than the cone angle, we create a direction sample  $(\theta, \phi)$  of the spherical coordinates.

We create a local axis system that use the surface normal to be the Y axis. As shown in

Figure 3.6(b), the green is the local X axis and the yellow lines is the local Z axis. We use this local axis systems to be the axes of the spherical coordinates. Therefore, we can generate a sample by mapping the spherical coordinates  $(\theta, \phi)$  to the Cartesian coordinate system using the Equation 3.3.

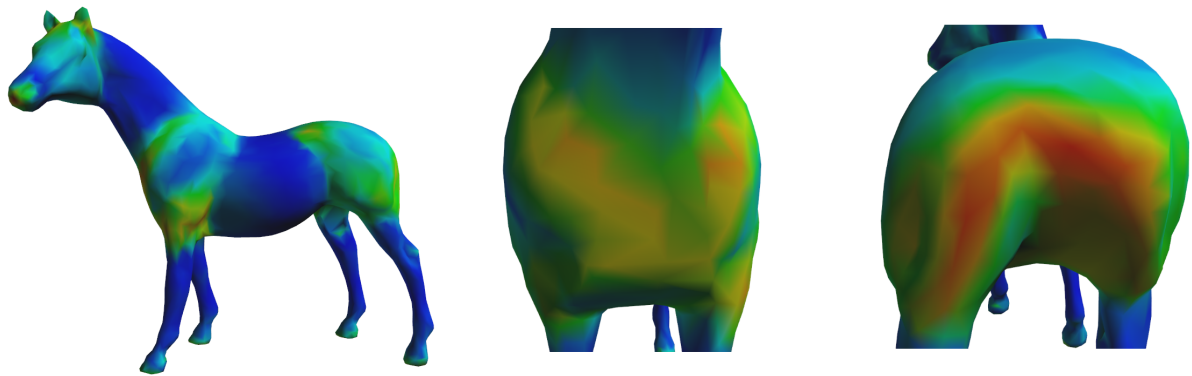
$$(2 \arccos \sqrt{1 - \xi_x}, 2\pi\xi_y) \rightarrow (\theta, \phi) \quad (3.3)$$

$$S_c = Rot_q(\phi, Y_{local}) \cdot Rot_q(\theta, Z_{local}) \cdot N \quad (3.4)$$

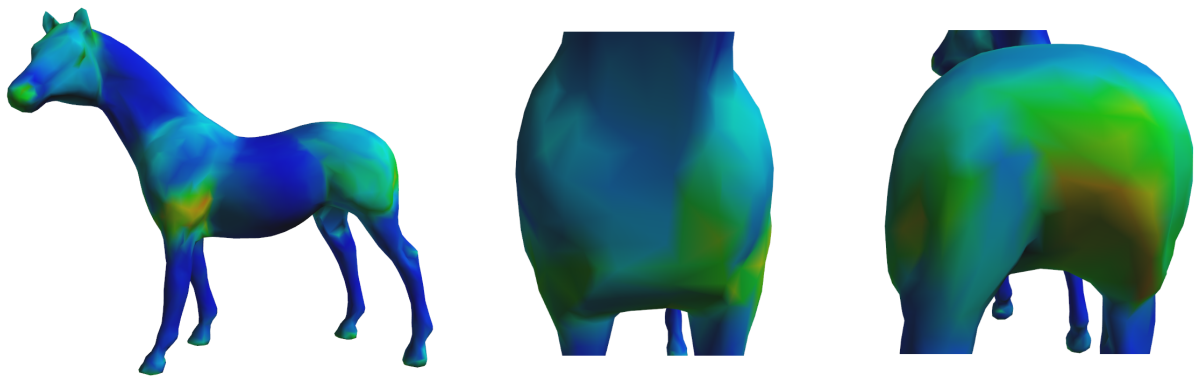
where the  $Rot_q(\theta, Axis)$  is the rotation matrix that rotates around the *Axis* in  $\theta$  degree, which is the quaternion rotation scheme.

For each vertex, we will generate several samples inside its normal cone and use the samples to compute MSP function. By increasing the samples in the MSP computation, we can obtain a smoothing effect and alleviate the influence of surface fluctuation. We call the process as the cone refinement of the MSP function. Figure 3.7 shows the distributions of slice error before and after the cone refinement. The slice error after the cone refinement is much lower than the original one, especially the terminal part.

We figure the MSP values of vertices into charts in Figure 3.8 and the slice error into charts in Figure 3.9. The x-axis of the chart is the vertex index of the horse model which is spatial continuous. We can find out that the distribution of the refined MSP is elaborate more stable than the one of the original MSP and the slice error after the cone refinement is generally smaller than the original one.

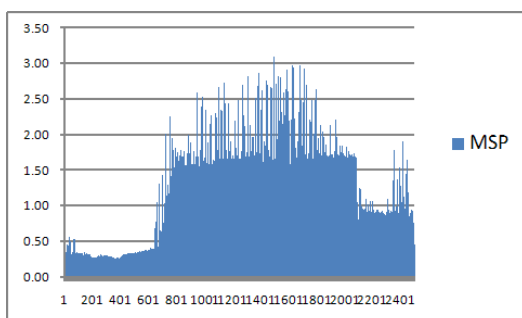


(a) Slice error distribution

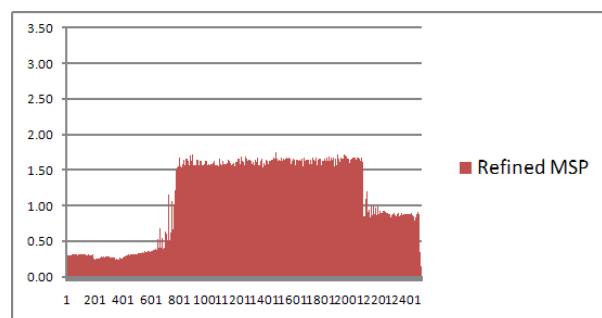


(b) Slice error distribution(with cone refinement)

Figure 3.7: Comparison of slice error distribution of horse model.

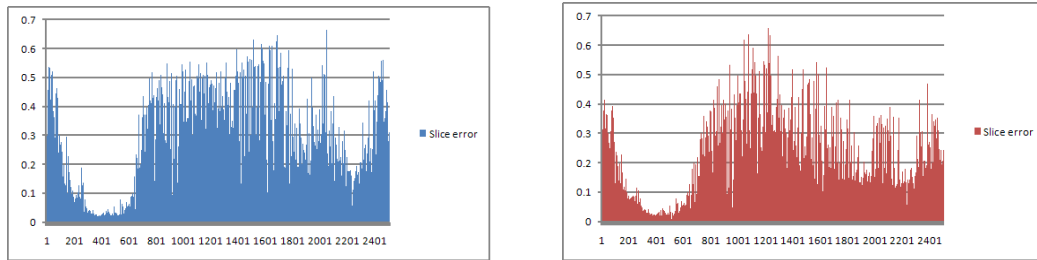


(a) MSP distribution of vertices.)



(b) Refined MSP distribution of vertices.

Figure 3.8: MSP/Refined MSP distribution charts for the horse model.



(a) Slice error distribution

(b) Slice error distribution(after cone refinement)

Figure 3.9: Distribution charts of the slice error for the horse model.

We use the value of slice error to control the angle of cone. The angle of the normal cone is proportional to the slice error. If the slice gets more error, we use larger normal cone to refine it. Because the computation cost of normal-cone refinement is large, we also use the slice error to decide whether to apply the cone refinement. We use 0.25 to be threshold to decide whether to apply normal-cone refinement and change the angle of cone linearly from 25 degree to 55 degree according the normalized slice error from 0.25 to 1.0.

---

# Skeleton Extraction using MSP Function

---



We develop a novel skeleton extraction algorithm that produces compact and reasonably good skeletons for closed 3-D meshes. We apply the MSP computation to each vertex and find a proper skeletal position in the MSP slice of the vertex. We then transform each vertex of the original mesh to its corresponding skeletal position and form a skeleton-like mesh whose shape is expected to close to the skeleton. We denote the skeleton-like mesh as skeleton mesh in the following chapters. Finally, we apply a LOD simplification to reduce the skeleton mesh to a curve skeleton.

## 4.1 Overview

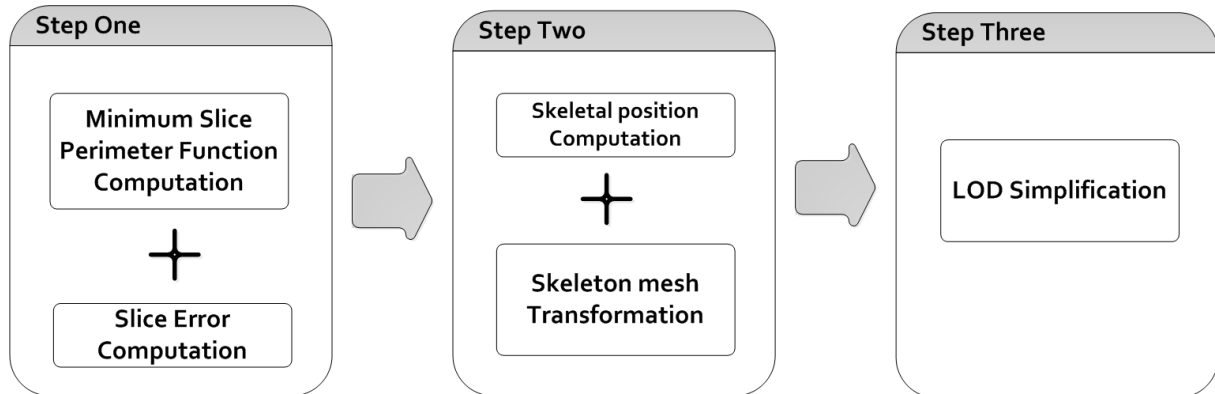


Figure 4.1: System overview.

Our skeleton extraction algorithm can be divided into three steps, as shown in Figure 4.1. For each vertex of the original mesh, we first compute its MSP value and the slice error in Step one. In Step two, we locate a skeletal point inside the MSP slice of each vertex and transform the vertex into its corresponding skeleton point to form a skeleton mesh. After obtaining the skeleton mesh, we apply a LOD simplification to simplify the mesh to form a curve skeleton in Step three.

## 4.2 Skeleton Mesh Computation

Having computed the MSP value for each vertex, we now have a MSP slice for each vertex and its perimeter as the MSP value. Next, we compute a so called skeleton point inside the MSP slice, that is expected to be close to the skeleton. A simple choice is the center of the MSP slice which, however, may lie outside the slice for concave models. Another more meaningful choice is the local minimum derived by applying the Repulsive Force field to the MSP slice. After a skeleton point is derived for each MSP slice, we transform the original mesh to a mesh, called *skeleton mesh*, by moving each vertex to its corresponding skeletal point. The skeleton mesh is



in general close to the skeleton except in the regions where the MSP slices have higher slice error.

### 4.2.1 Repulsive Force Field Modeling

Repulsive Force model is a potential field approach for skeletonization [CSYB05, CTK00]. The key idea is that it assumes the boundary of an object carries electric charge and each point inside the object gets a directional repulsive force from the charges on the boundary. In other words, every force samples distributed over the boundary gives a directional repulsive force to the points and the force is attenuated with the distance between them. The directional force applied to points inside the object forms the repulsive force field. On the repulsive force field inside the object, we can locate the position of the local minimum by tracing the force direction.

The repulsive force applied to an interior point  $p_i$  from the force sample  $b_j$  is defined as  $F(p_i, b_j) = \frac{\text{normalized}(p_i - b_j)}{r_{ij}^2}$ , where  $r_{ij} = \|p_i - b_j\|$ . The total potential results from all the force samples at point  $p_i$  is calculated by summing up all the forces applied from all the force sample  $b_1, b_2, \dots, \text{and } b_n$  to  $p_i$ , as shown in Equation 4.1.

$$F(p_i) = \sum_{j=1}^n F(p_i, b_j). \quad (4.1)$$

After computing the repulsive force field, the local minimums of the field are found by tracing some points inside the object along the direction of the repulsive force field. Figure 4.2 shows the example of the repulsive force field [CSYB05] and the arrow means the repulsive force direction.

### 4.2.2 Deriving skeleton mesh

The skeletal point inside a MSP slice can be derived in six steps as shown in Figure 4.3. First, for each slice point we create a moving sample, which is at the position displaced slightly from the slice point in the inward normal direction. Second, uniform samples on the MSP slice are derived and serve as the force samples of the repulsive force model. Third, we iteratively trace

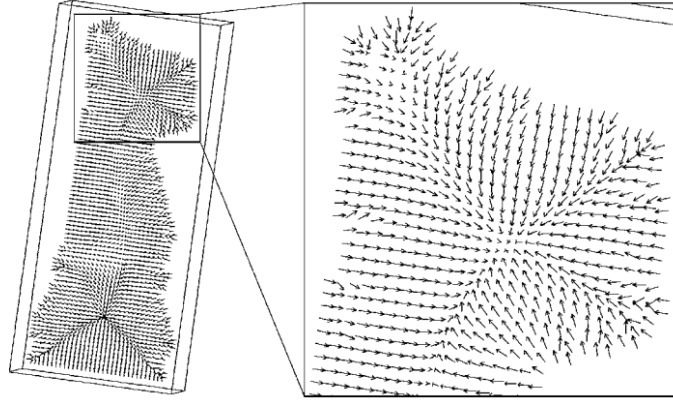


Figure 4.2: Repulsive force field of a cow model [CSYB05]

each moving sample along the direction of the force field until it arrives a local minimum. Thus, for a moving sample  $p$  we stop the tracing until  $\|F^{i+1}(p)\| > \|F^i(p)\|$ , where  $F^i(p)$  is the force applied to  $p$  at the iteration  $i$ . Fourth, when all the moving samples stop their movement, we merge the moving samples that are spatially close to each other into a point and regard it as a local minimum. There may exist several local minimums inside a MSP slice, but the skeleton is expected to intersect the MSP slice at only one position. Hence for a surface point  $p$ , we need to find the most representative local minimum to be the skeletal position of this slice. We choose the local minimum that are converged by the most moving samples to be the skeletal position of the slice. Other local minimums may be generated by the surface detail and can be discarded.

After obtaining all the skeletal points, we can obtain the skeleton mesh by moving each vertex to its corresponding skeletal position. Figure 4.4 shows the skeleton point for each vertex (with color coding based on its slice error) and the skeleton mesh. If a vertex has smaller slice error, the derived skeleton point shall have higher chance to be close to the skeleton.

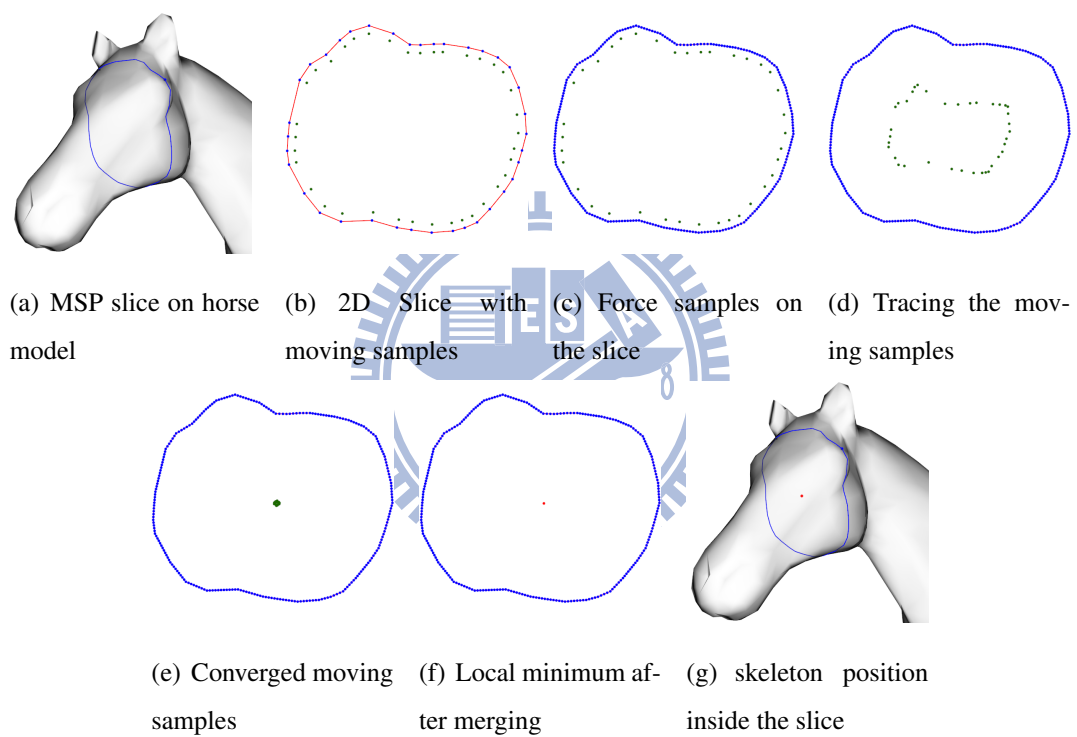
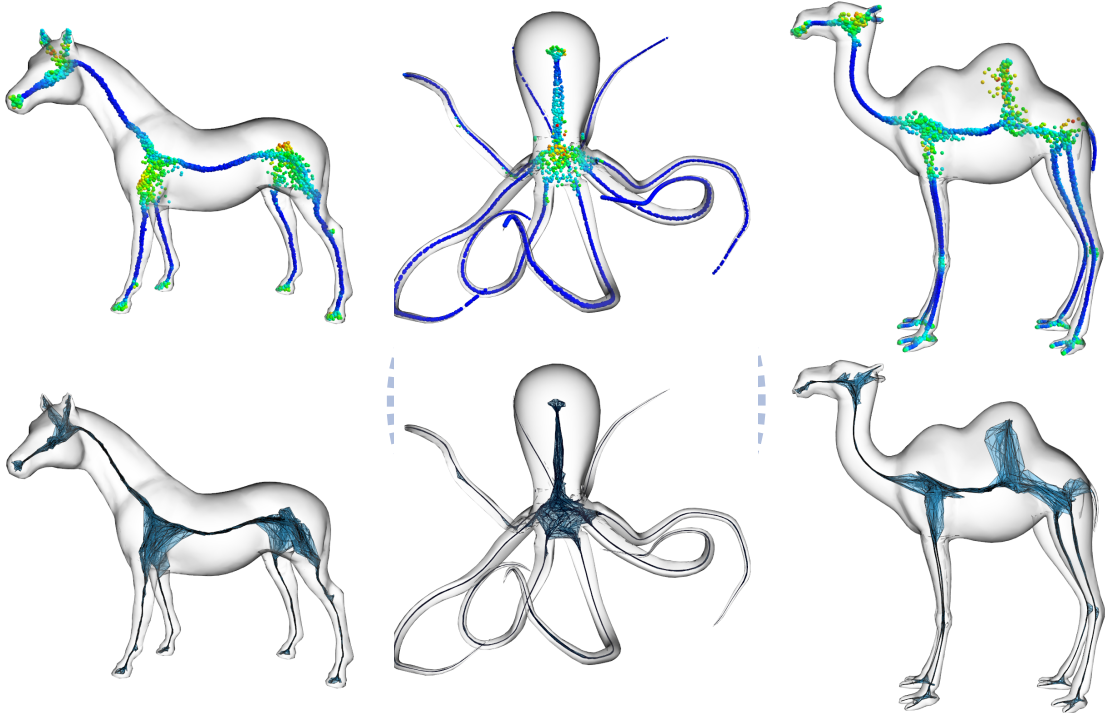


Figure 4.3: Locating the skeletal position



(a) Horse model

(b) Octopus model

(c) Camel model

Figure 4.4: Transformed Skeleton Mesh using the repulsive force model.

## 4.3 Skeleton Mesh Simplification

### 4.3.1 LOD framework

The skeleton mesh transformed from the original mesh maintains the same connectivity of the original mesh and is generally close to the skeleton except in some regions where the MSP slices have higher slice error. Based on this observation, we propose a LOD simplification scheme that reduces the skeleton mesh to a curve skeleton that resembles the global shape of skeleton mesh. Our LOD simplification is different from the traditional one. The proposed simplification aims to preserve the global shape of the skeleton mesh while the tradition one aims to preserve the local shape and feature of the mesh.

Our LOD simplification is a greedy approach. Edges are collapsed based on their simplification cost. Two error metrics are proposed and their weighted sum is used to represent the simplification cost of each edge. At each LOD operation, the edge with lowest cost is selected and then first tested to see if it is ready to be a skeleton edge. If the edge has adjacent triangles, it is collapsed as usual. If the edge has no adjacent triangles, it is 1-D and generally a skeleton edge. However, in order to remove those 1-D edges that are the result of surface noise, such a 1-D edge needs to be tested using the proposed constraint to see if the edge is the result of surface noise. If so, the edge is collapsed. If not, we still push the edge back into the bottom of the edge queue for future consideration since conditions associated with the constraint may be updated by the collapsing of its adjacent edges.

When we pop a 1-D edge  $e$  from the edge queue, we will apply the same constraint test on it and collapse it if it is a branch of noise. If the conditions associated with the constraint of the end vertices of  $e$  won't be changed or updated anymore, the edge must be a skeleton edge and we will put the edge into the list of skeleton edges. On the implementation, we will check the adjacent edges of the edge  $e$  whether they are all 1-D edge and its saturation are all larger than a user-specified threshold. If so, the conditions of its end vertices won't be changed anymore and the edge must be a skeleton. If not, we will push the edge back to the edge queue. When

the edge queue is empty, we finish the computation of skeleton extraction. By the control of the error metrics and the constraint, we can preserve the global shape of the skeleton mesh and generate reasonably good skeletons after the LOD simplification. Figure 4.5 shows the pseudo code of the LOD simplification.

#### Pseudo Code of the LOD Simplification

```

1. Obtain the skeleton Mesh  $M$ .
2. Compute the cost  $c$  for each edge  $e$  in  $M$ .
3. Push each edge into the Priority queue  $Q := \{(e, c)\}$ .
4. List of skeleton edge  $L := \{\emptyset\}$ 
5. While  $Q$  is not empty
  (a) Pop the edge  $e$  with minimum cost.
  (b) Test the  $e$  using the LOD constraint
      If  $e$  is not a 1-D skeleton edge
      {
          Collapse the edge.
      }
      Else
      {
          Compute the  $SE$  of  $e$ .
          if saturation  $SE$  of  $e$  is below a user-specified threshold
          {
              Collapse the edge. //  $e_i$  is a branch due to noise
              Update  $SE$  of the adjacent edges of  $e$ .
              If any adjacent 1-D edges of  $e$  is in the list  $L$  and
              their saturations are below a user-specified threshold
              {
                  Collapse the edges into  $Q$ .
                  Remove the edges from  $L$ .
              }
          }
          Else
          {
              If the adjacent edges of  $e$  are all 1-D edges and
              their saturations are all larger than a user-specified threshold
              Put  $e$  into the list of skeleton edge  $L$ .
              Else
              Push the  $e$  back to the bottom of  $Q$ .
          }
      }
  }

```

Figure 4.5: Pseudo code of LOD simplification

During the LOD simplification, for a vertex  $v$ , we record the set of all vertices that had been reduced to  $v$ . We call this set as the merging set  $S_M(v)$  of  $v$ . Initially, the merging set of a vertex contains only itself. After an edge  $e = (u, v)$  is collapsed to  $w$ , the merging set of  $w$ ,  $S_M(w)$ , will be the union of the  $S_M(u)$  and the  $S_M(v)$ . The merging set of a vertex  $w$  represents the vertices on the original mesh that have been finally reduced to  $w$  in the LOD simplification.

### 4.3.2 Error metrics

In order to preserve the global shape of the skeleton mesh, we propose two error metrics: the compactness metric  $E_c$  and the slice error metric  $E_s$ .

**Compactness metric** The skeleton represents the global shape of the original mesh and a skeleton node stands for a part of surface area. We intend to utilize the MSP function and expect a skeleton node to represent a set of vertices that have similar volume information. We know that each vertex on the original mesh corresponds to one skeleton node and there exists a skeleton node inside each MSP slice. In the ideal case of the MSP computation, such as on the cylinder model, the points on the same MSP slice possess the same MSP slice, which means that the points have the same MSP values, the same slice normal, and zero distance between its corresponding skeleton points. So the points on the MSP slice should correspond to the same skeleton node.

In the reality, the points on a MSP slice generally don't have the same MSP slice. Hence, there must exist some differences between the MSP slices for the points on the MSP slice. However, the points on a MSP slice are generally expected to correspond to a same skeleton position. We design a metric that aims to merged the edges that have the similiar MSP values, similar MSP slice normal, and the small edge length distance. By using this three properties, we design the so called compactness metric, which is the product of three terms in the Equation 4.2.

For a edge  $e = (u, v)$ , we use the  $\frac{\max(MSP(u),MSP(v))}{\min(MSP(u),MSP(v))}$  to measure the similarity of the MSP values of the  $u$  and  $v$ . When the difference between the MSP values at the two nodes is large,

the possibility of the two nodes that correspond to the same skeleton node is lower and therefore the cost of simplifying the edge is larger.  $\frac{\|e\|}{\|e_{max}\|}$  measures the normalized distance between  $u$  and  $v$ . The larger distance it has, the lesser possibility that these two vertices correspond to the same skeleton node. We use  $\arccos(N_s(u) \cdot N_s(v))$  to measure the difference between the slice normal, where  $N_s(u)$  is the slice normal of  $u$ . When the angle of the two MSP slices is larger, the MSP slices are more different and the two vertices are less likely to be correspond to the same skeleton node. So we use these three terms to define a error metric ,aiming to force edges that have lesser difference of MSP value, and slice normal, and smaller edge length between its two end vertices to be collapsed earlier.

To compute the three terms, we extend each end vertex to include all the vertices in its merging set. Because a vertex of the skeleton mesh might be merged before and the position of it might be also updated, we need to find a proper value to be the node's information. Because the merged vertices of each vertex are all recorded in the merging set, we propose to use the averaged value of the all merged vertices in its merging set to be the value of the current vertex. The  $MSP(u)$  is the weighting sum of the MSP values of the vertices in the merging set of  $u$  by the weighting of slice error. The  $N_s(u)$  is also the weighting-sum slice normal of the vertices in its merging set.

$$M_c(e) = \left( \frac{\max(MSP(u), MSP(v))}{\min(MSP(u), MSP(v))} \right)^2 \cdot \left( \frac{\|e\|}{\|e_{max}\|} \right)^2 \cdot \frac{\arccos(N_s(u) \cdot N_s(v))}{\pi} \quad (4.2)$$

**Slice Error metric** As discussed in Section 3.2, we quantify the error of a MSP slice to measure how close its skeleton point is to the skeleton. Figure 4.6 shows the skeleton points derived from the MSP slices and their associated error coded in color ranging from blue to red for increasing error.



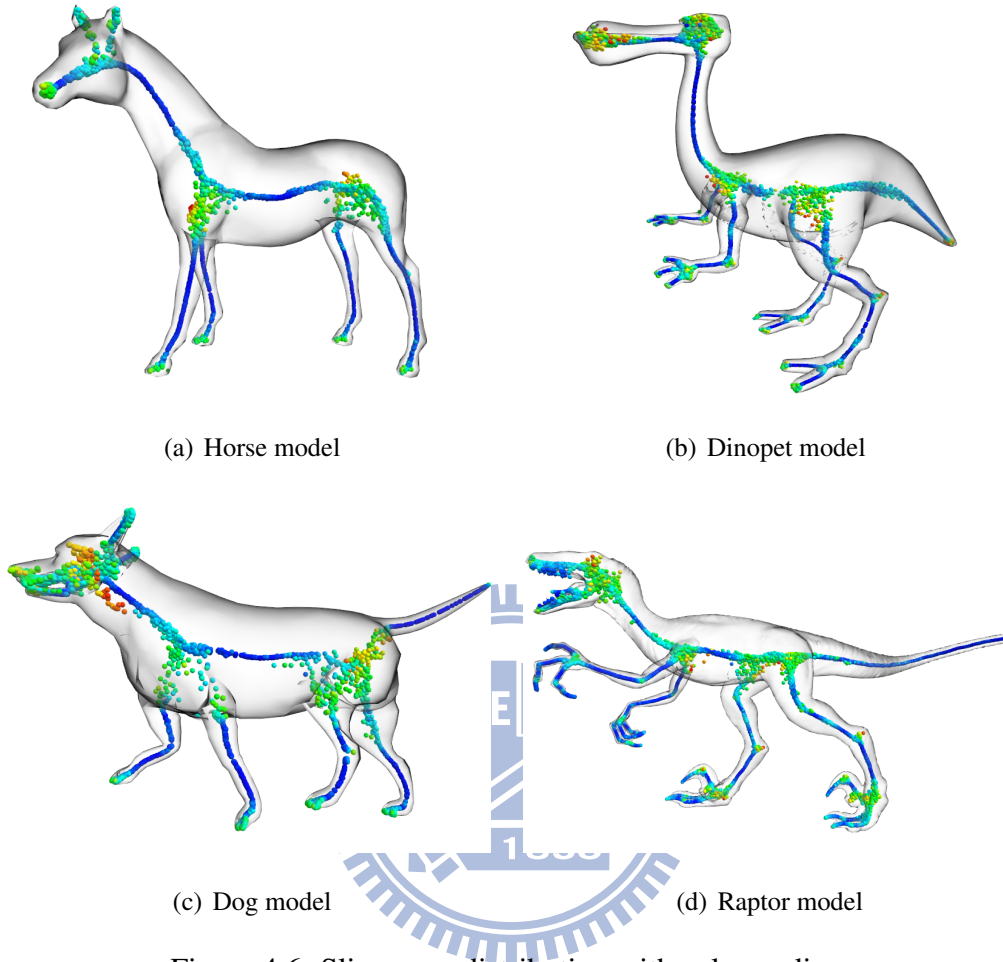


Figure 4.6: Slice error distribution with color-coding.

In chapter 3, we have already defined the slice error  $E_s(w)$  associated with a vertex  $w$  as

$$E_s(w) = \sum_{p \in V_{MS(w)}} \arccos \left( \frac{N_S(w) \cdot N_S(p)}{\|N_S(w)\| \|N_S(p)\|} \right),$$

where  $V_{MS(w)}$  is the set of all points on MSP slice  $MS(w)$  of  $w$  and  $N_S(w)$  is the normal vector of the MSP slice of vertex  $w$ . The slice error metric  $M_s$  for an edge  $e = (u, v)$  is defined as the averaged slice error of node  $u$  and node  $v$ . We intend to collapse the edges that have larger slice error before edges with smaller slice error. The slice error of a vertex can be defined as the averaged slice error of the nodes in its merging set. Furthermore, the slice error metric  $M_s(e)$  of an edge  $e = (u, v)$  can be defined as the averaged slice error of the nodes in the union of the merging sets of  $u$  and  $v$ , as shown in the Equation 4.3.

$$M_s(e) = \frac{1}{T} \cdot \sum_{w \in S_M(u) \cup S_M(v)} E_s(w) \quad (4.3)$$

where the  $E_s(w)$  is the normalized slice error of the node  $w$ ,  $E_s(w) = \frac{E(w) - E_{min}}{E_{max} - E_{min}}$  and  $E_{max}$  is the maximum slice error between the all vertices in the skeleton mesh, and  $E_{min}$  is the minimum slice error.  $T$  is the size of the  $S_M(u) \cup S_M(v)$ .

**Total Cost** The total cost function  $M$  is the weighting sum of the slice error metric and the compactness metric:

$$M(e) = M_c(e) + \lambda(1 - M_s(e)) \quad (4.4)$$

We intend to let compactness metric dominate the whole simplification sequence due to its capability on the global shape preserving. So  $\lambda$  is usually set to 0.1.

### 4.3.3 LOD Constraint

In order to preserve the global shape of the skeleton mesh, we need to design a proper LOD constraint to maintain the topology information and delete the unwanted branches. During the simplification, if an edge has no adjacent triangles, it is a 1-D edge and the edge is generally a skeleton edge. Since the noise on the surface may result in 1-D edges that don't capture any prominent features, we need to detect and remove such edges during the simplification. We call these edges as the branches of noise or branch in short.

We propose the saturation for 1-D edges to quantify the possibility that the edge is a skeleton edge, not a branch. We define the saturation  $SE(e)$  of the edge  $e = (u, v)$  as the ratio of the saturations of  $u$  and  $v$ .

$$SE(e) = \frac{\min(Sat(u), Sat(v))}{\max(Sat(u), Sat(v))}$$

where  $Sat(u)$  and  $Sat(v)$  are defined as the surface areas on the original mesh covered by vertices in the merging set of  $u$ ,  $S_M(u)$ , and vertices in  $S_M(v)$  respectively.

We assume that a skeleton edge has a property that its two end vertices have similar surface area corresponds to them. Hence, for an edge, if it is 1-D and its saturation value  $SE$  is almost equal to 1.0, the edge is a skeleton edge and must be constrained and cannot be simplified. Otherwise, the edge should be simplified. Even though the surface area corresponding to a skeleton node in the final skeleton is spatially connected. For the skeleton edges that capture small prominent features, such as the ear, the surface area of this feature is relatively much smaller than the head. Hence the saturation values of this kind of edges are small. But this kind of edges are normally considered as edges in the final skeleton. So the skeleton edge for small prominent feature can not be distinguished from the branches of noise by using saturation value alone.

But the MSP values of the two end nodes which is connected by the skeleton edge of small feature is much different. Besides, the MSP values of the two end nodes of a noise branch do not be that different. Therefore, we can use the MSP ratio to distinguish these two type edges and we add the MSP ratio term to compensate this weakness of keeping branches of small features, as shown the MSP term in the Equation 4.5. Through the term of the MSP ratio, it increases the value of the edge saturation and compensate the small value of the area ratio. Hence, we can use the edge saturation  $SE$  to detect noise branches and also keep the meaningful skeleton edges in the final skeleton during the LOD simplification.

$$SE(e) = \left( \frac{\max(MSP(u), MSP(v))}{\min(MSP(u), MSP(v))} \right)^2 \cdot \frac{\min(Sat(u), Sat(v))}{\max(Sat(u), Sat(v))} \quad (4.5)$$

where  $MSP(u)$  and  $MSP(v)$  are defined by the slice-error-weighting sum of the MSP values of the nodes in the merging set  $S_M(u)$  and  $S_M(v)$  respectively.  $Sat(n_a)$  is the saturation of skeleton node  $u$ , which is defined by the surface area enclosing by the merging set of  $v$ .

During the simplification, the edge  $e$  with lowest cost are popped from the queue. For each popped edge, we will test the edge to see if it is part of skeleton by the LOD constraint. If the edge had become 1-D skeleton edge and the edge saturation  $SE$  of edge  $e$  is larger than a threshold, the edge will be considered as the candidate of skeleton edge. If the saturation of the skeleton edge will not be changed anymore, this skeleton edge must be a skeleton edge and

we will push this edge into the list of skeleton edges. Otherwise, we will push the candidate skeleton edge back to the edge queue for future consideration. If the edge has not become 1-D edge or the edge saturation  $SE$  of a 1-D edge is below than a threshold, the edge needs to be collapsed. When the edge queue is empty, we finish the skeleton extraction.

#### 4.3.4 Replacement of the skeleton vertex

When an edge is collapsed, two end vertices are merged to a new vertex. We intend to find a new position of the new vertex and merge the information of the original two end vertices. Hence, we propose to use the merged vertices in its merging set to find a proper position of the new position. But there exist different amounts of slice error in the merged vertices of its merging set. So we add the factor of slice error in the node replacement to update the new vertex to a better position.

When collapsing  $e = (u, v)$ , a proper position where  $u$  merges to is the weighting sum of the vertex positions in the merging set of the  $u$  and  $v$ . Equation 4.6 defines the update position after the edge  $e = (u, v)$  collapsed.

$$U(e) = \frac{1}{\sum_{w \in S_M(u) \cup S_M(v)} Weight(w)} \cdot \sum_{w \in S_M(u) \cup S_M(v)} Weight(w) \cdot Pos(w), \quad (4.6)$$

where

$$Weight(w) = 1.0 - E_s(w)$$

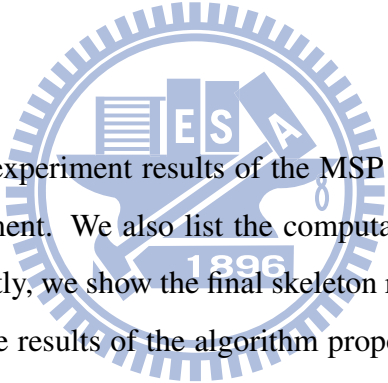
and  $E_s(w)$  is the normalized slice error of node  $w$ .

# CHAPTER 5

---

## Results

---



In this chapter, we present the experiment results of the MSP functions, cone refinement, and the repulsive force field refinement. We also list the computation time of the MSP functions and skeleton node location. Lastly, we show the final skeleton results and compare the skeleton results of our algorithm with the results of the algorithm proposed by Au [ATC<sup>+</sup>08]. We also present some experiments with different parameters in the LOD simplification.

All experiments are performed on Intel Core 2 Duo CPU E8400 3.0GHz with 3G ram, using NVIDIA Geforce 9600GT graphic hardware.

### 5.1 MSP Computation Results

For each vertex of the input model, we generate a number of slices that cut through the model along the vertex's normal. We choose the slice with minimum perimeter, which is the so called MSP slice. And the perimeter of the MSP slice is the MSP value of the vertex. The number of slices is a trade off between accuracy and computation cost. The larger number of slices we sample, the more precise information we will get. But it also spends more computation

time. We compare the MSP results that use 5 slices, 15 slices, and 25 slices for each MSP computation, as follows.

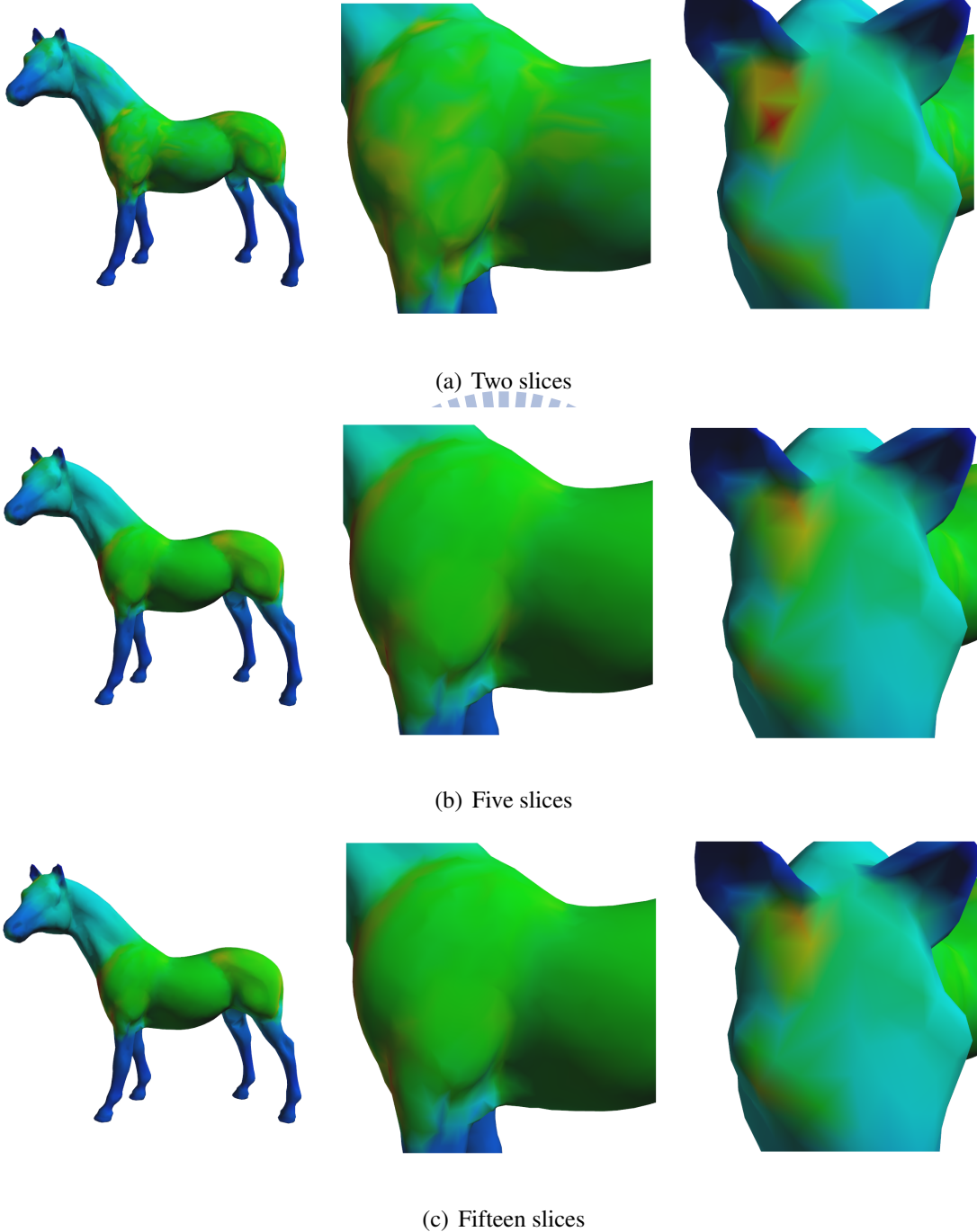


Figure 5.1: MSP computation results of different amounts of slices

We can clearly recognize that the more slices we use to compute MSP, the more accurate result we could obtain. But if we use more slices to compute the MSP, we also take more time to compute as shown in Table 5.1.

Slice Num	2 slices	5 slices	15 slices
Time(sec)	2 sec	3 sec	12 sec

Table 5.1: MSP computation time of different amount of slice

We show the several MSP results in Figure 5.2 and the corresponding MSP result after the cone refinement in Figure 5.3. As the following results show, we alleviate the influences of the surface fluctuation and get more reasonable MSP distributions after the cone refinement.

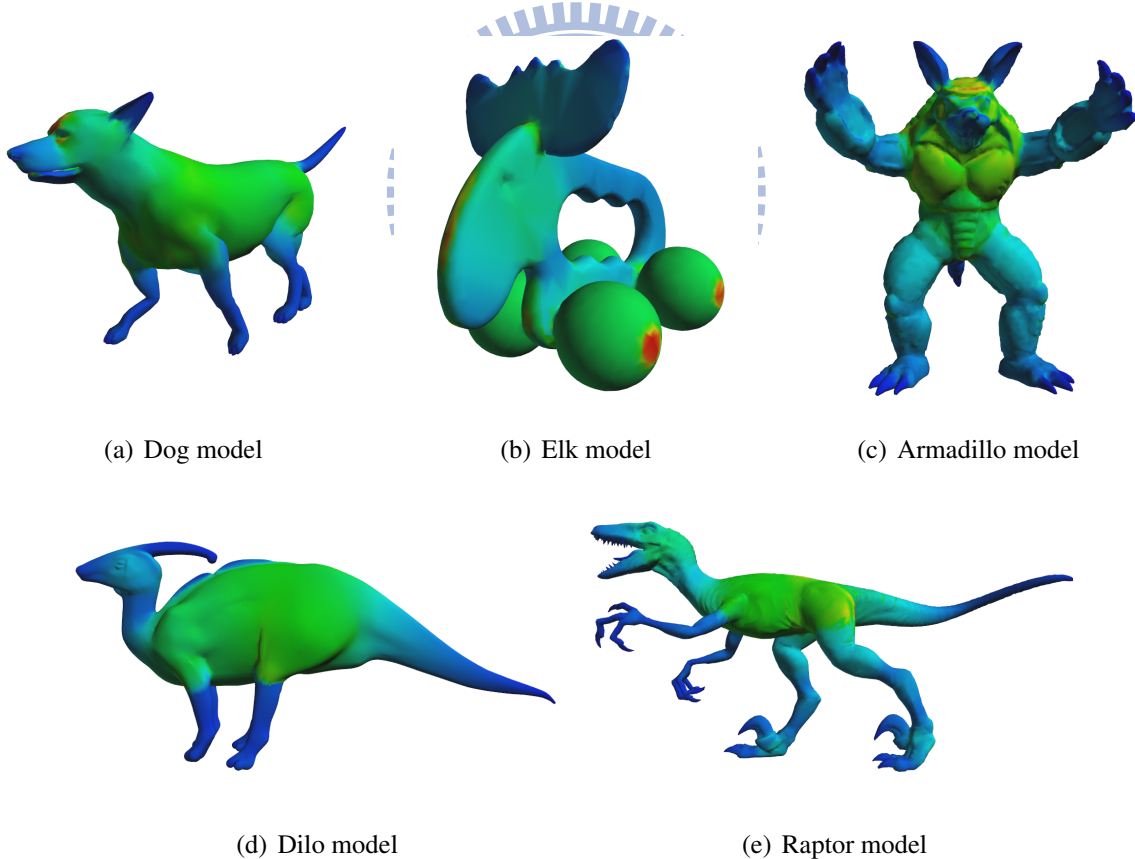
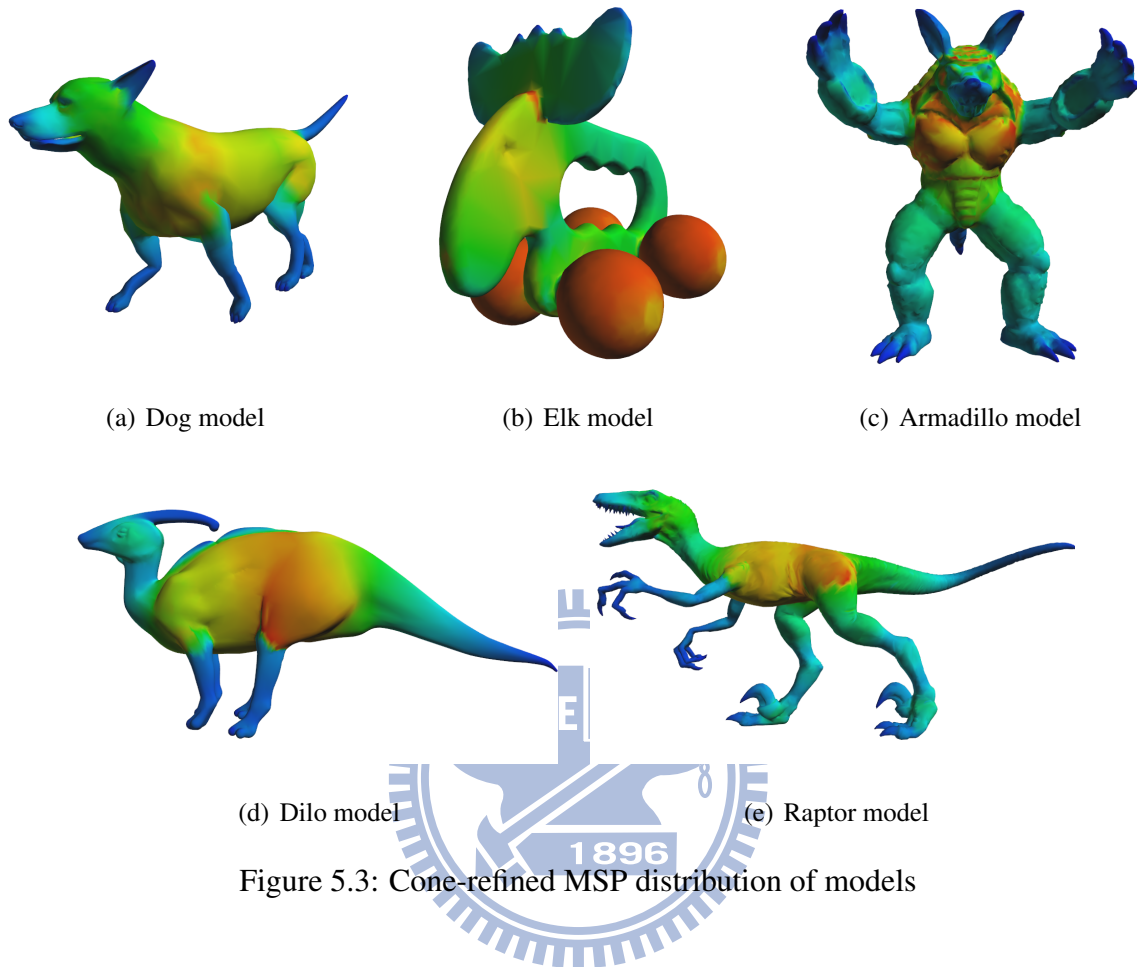


Figure 5.2: MSP distribution of models



## 5.2 Preprocessing Time

In this section, we list the computations time of the MSP functions and skeleton node location as follows. We use 25 slices for each MSP computation and 15 samples in the normal cone at the cone refinement step. The normal-cone refinement is only applied to the vertices whose slice error is larger than 0.25. It's why that the time of the cone refinement of the Octopus model is much shorter than others.



Model	face num.	vertex num.	MSP computation(sec.)	Cone-Refined MSP(sec.)
Horse	5000	2502	4	24
Dog	8136	4070	8	57
Camel	10000	5000	12	67
Octopus	10508	5262	7	16
Raptor	17808	8904	26	109
Fertilty	19010	9499	49	335
Neptune	22444	11218	43	311
Dancing child	25002	12487	74	729
Armadilo	29186	14595	116	921

Table 5.2: Computation time of MSP and Cone-Refined MSP

Model	Slice error computation(sec.)	Local minimum location by RF model
Horse	1	3
Dog	1	5
Camel	1	11
Octopus	1	5
Raptor	1	11
Fertilty	1	31
Neptune	2	18
Dancing child	2	52
Armadilo	4	185

Table 5.3: Computation time of Slice Error and Skeletal positions

Through the tables above, we know the computation time of cone refinement is the most costly. If we can improve the efficiency of the cone refinement or find another approach to refine the MSP function, we may save more computation cost.

### 5.3 Skeleton Mesh Representation

In this section, we show the skeleton meshes of different steps, including the skeleton meshes of MSP function, the ones of the cone-refined MSP function, and the ones of the cone-refined MSP function after the repulsive force model processing. The skeleton meshes of the MSP/Cone-refined MSP function are generated by moving the vertices of the input mesh to the centers of the MSP slices. The third type is the cone-refined skeleton mesh after the operations of the repulsive force model.

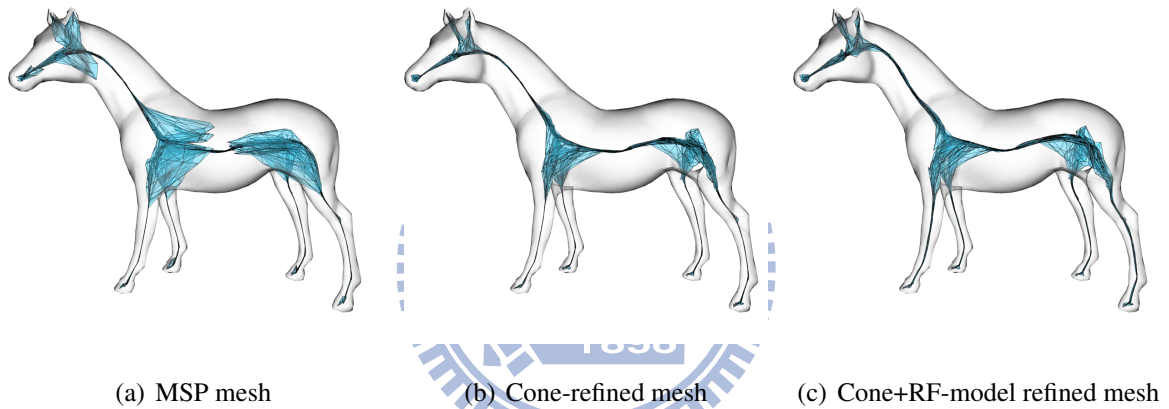


Figure 5.4: Skeleton meshes of horse model

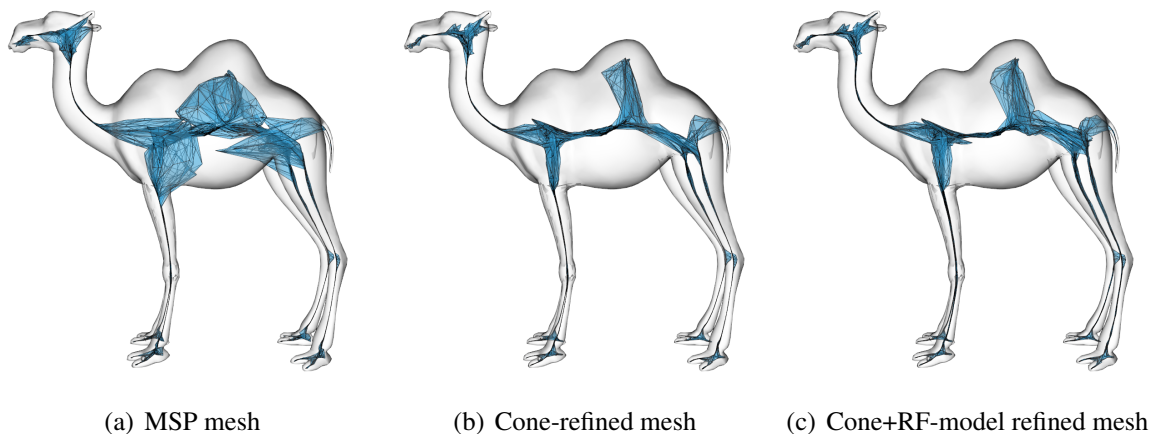


Figure 5.5: Skeleton meshes of camel model

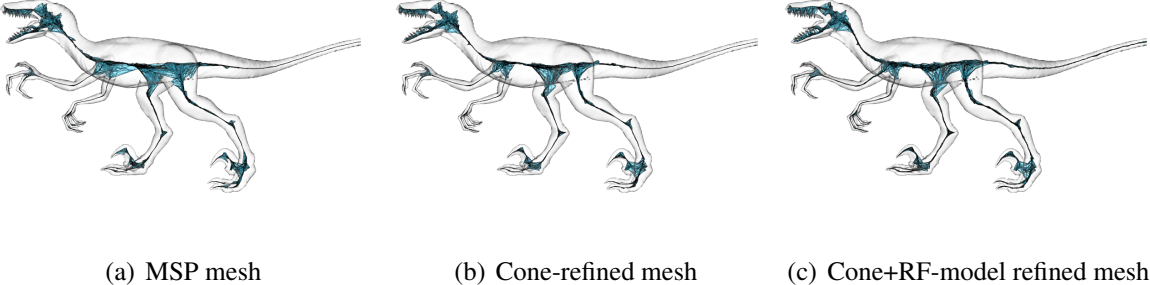


Figure 5.6: Skeleton meshes of Raptor model

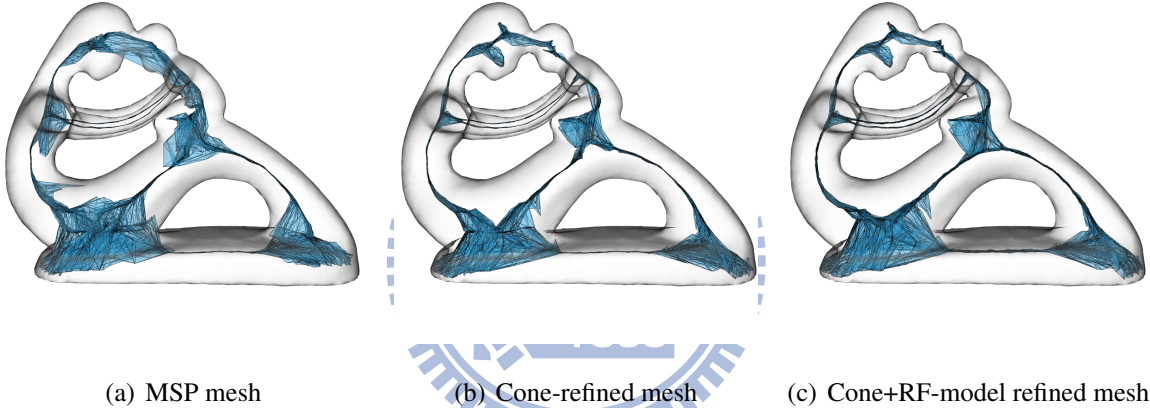


Figure 5.7: Skeleton meshes of Fertility model

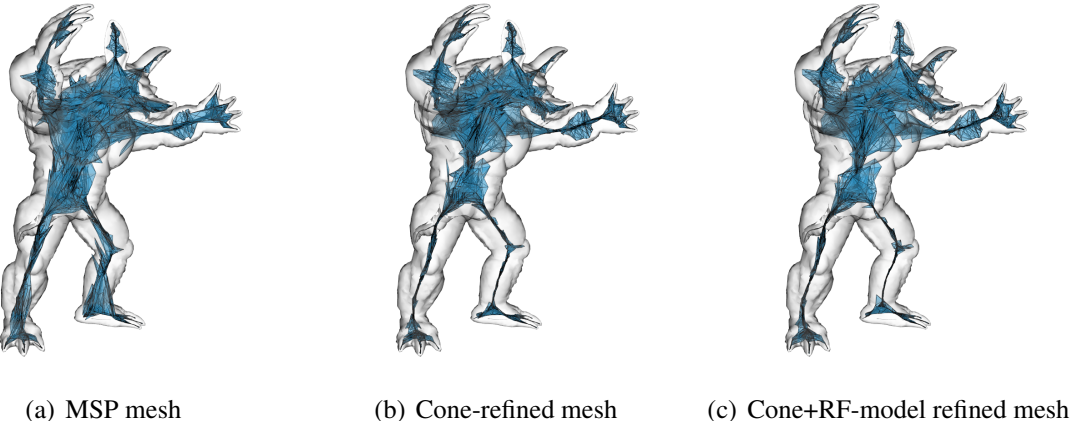


Figure 5.8: Skeleton meshes of Armadillo model

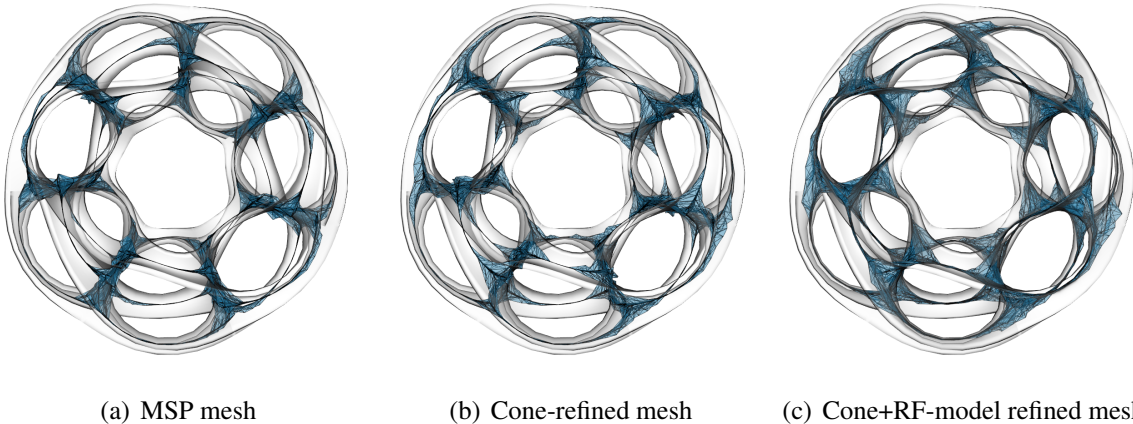


Figure 5.9: Skeleton meshes of Heptoroid model

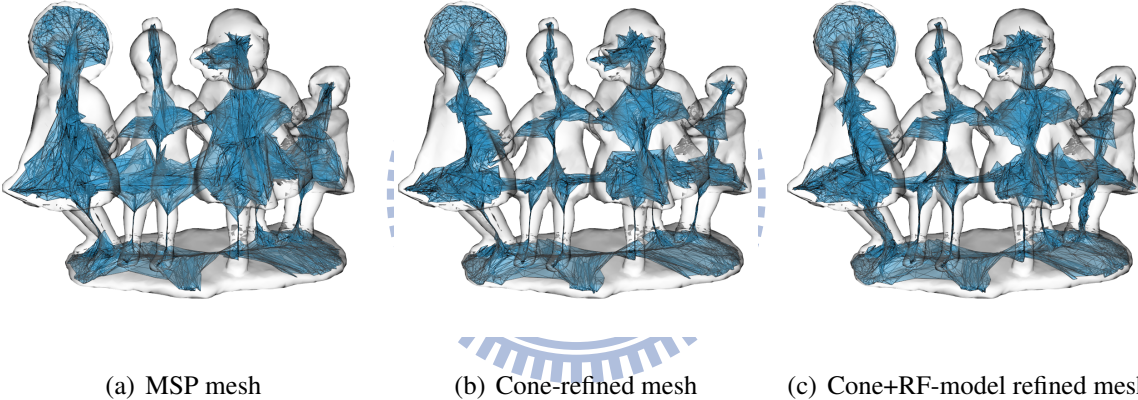


Figure 5.10: Skeleton meshes of Dancing Children model

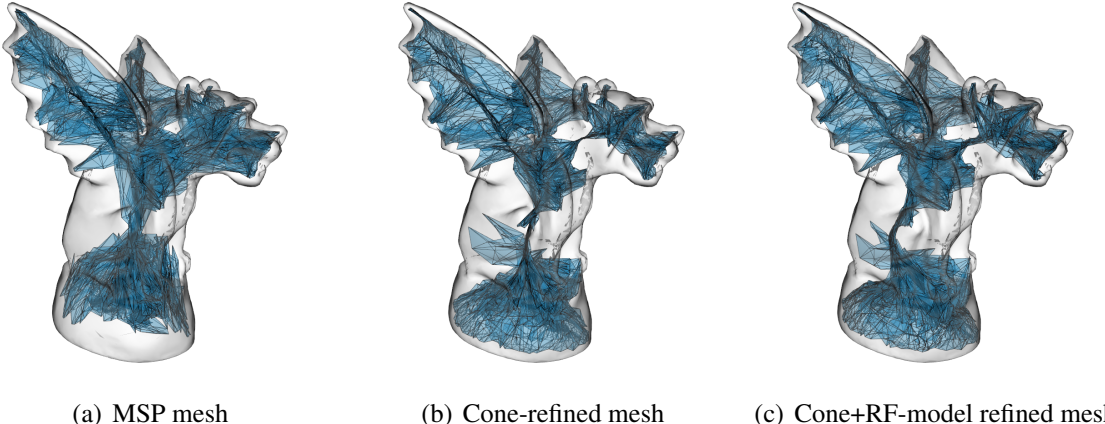


Figure 5.11: Skeleton meshes of Gargoyle model

We clearly recognize that the cone-refined skeleton mesh is more skeleton-like than the skeleton mesh of the original MSP. Furthermore, the skeleton meshes after the processing of RF model capture more detailed surface information than the cone-refined ones. For some models of non-cylinder shape, the skeleton mesh is much similar to the media axis. Such as the dancing child model, gargoyle model, and the body part of armadillo model. Their skeleton meshes seem like the medial axis and it will result in more simplified skeleton. In order to improve it, we need to change the measurement of MSP function or process the skeleton mesh directly to obtain more skeleton-like mesh in the future.

## 5.4 Skeleton Results and comparison

In this section, we compare our skeleton results to the ones proposed by Au[ATC<sup>+</sup>08] as follows. Au et al. propose to apply Laplacian smoothing to obtain a zero-volume mesh and use LOD simplification to get a skeleton. Besides, we use more intuitive method to obtain a nearly zero-volume mesh and apply more complicated LOD simplification to generate a skeleton. There are several similarities between our algorithm and the one of Au. We compare the skeleton results as follows.

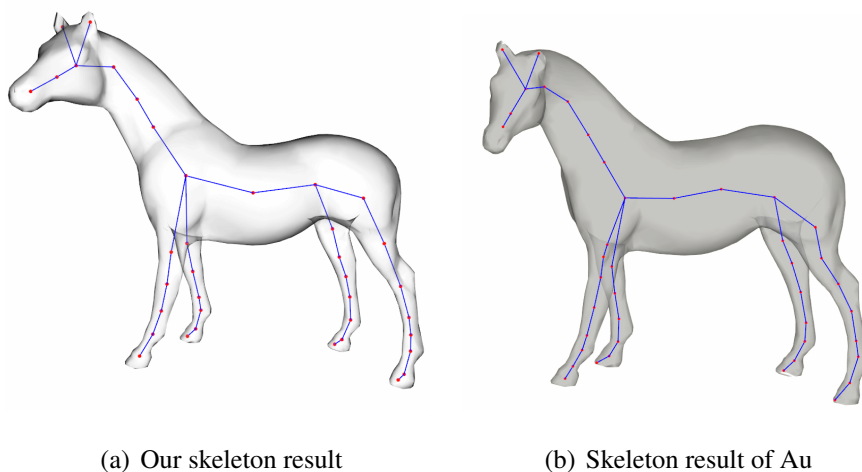
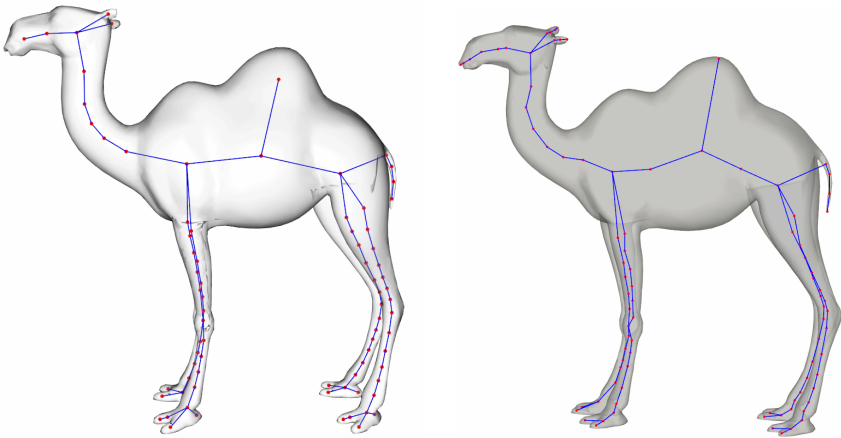


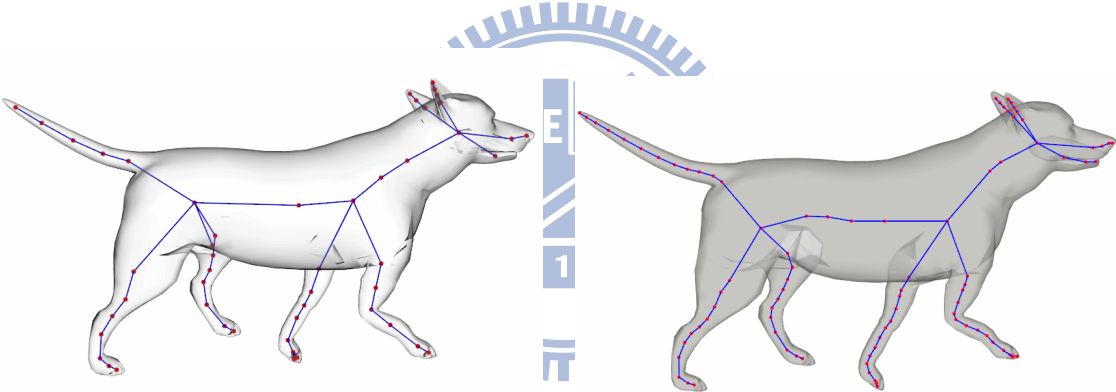
Figure 5.12: Skeleton results of horse model



(a) Our skeleton result

(b) Skeleton result of Au

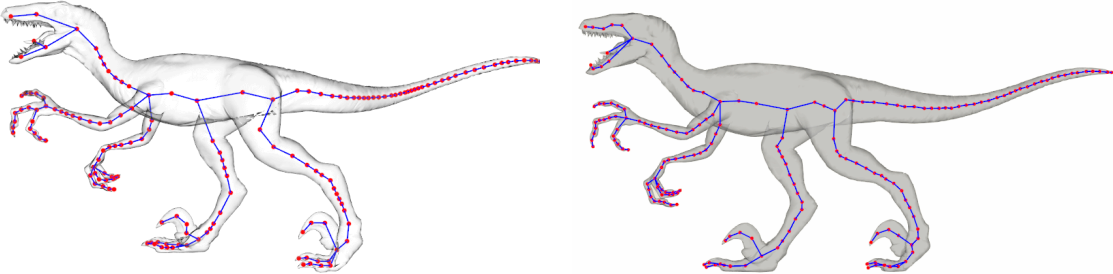
Figure 5.13: Skeleton results of Camel model



(a) Our skeleton result

(b) Skeleton result of Au

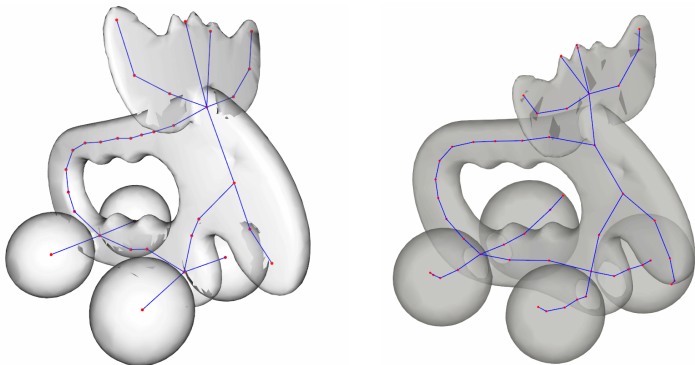
Figure 5.14: Skeleton results of Dog model



(a) Our skeleton result

(b) Skeleton result of Au

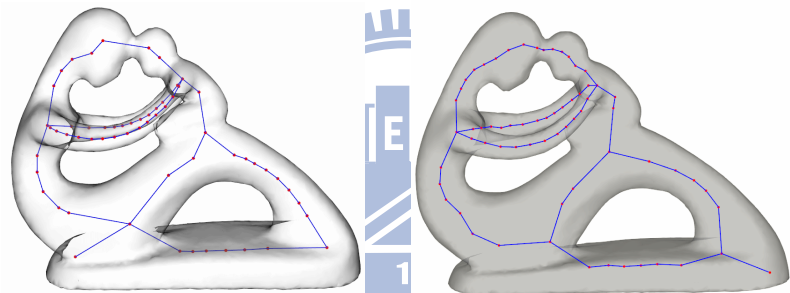
Figure 5.15: Skeleton results of Raptor model



(a) Our skeleton result

(b) Skeleton result of Au

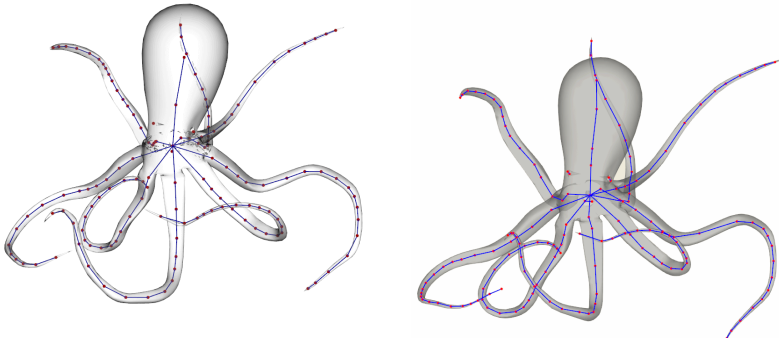
Figure 5.16: Skeleton results of Elk model



(a) Our skeleton result

(b) Skeleton result of Au

Figure 5.17: Skeleton results of Fertilty model



(a) Our skeleton result

(b) Skeleton result of Au

Figure 5.18: Skeleton results of Octopus model

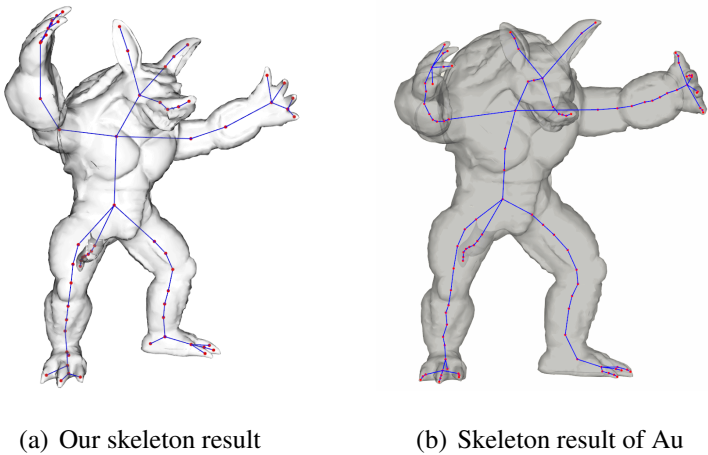


Figure 5.19: Skeleton results of Armadillo model

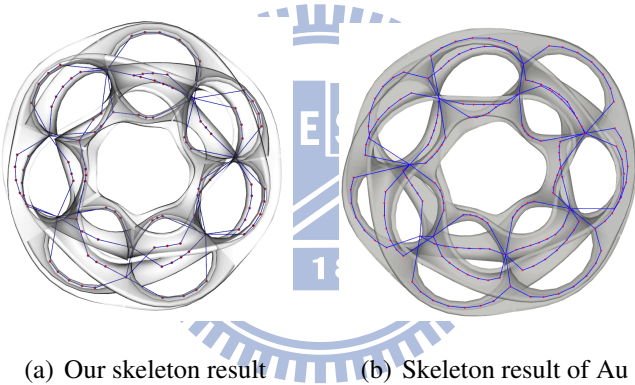


Figure 5.20: Skeleton results of Heptoroid model

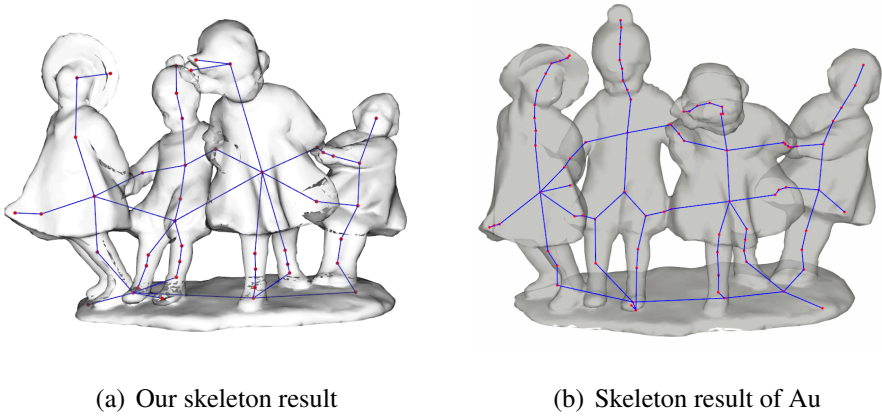


Figure 5.21: Skeleton results of Dancing children model



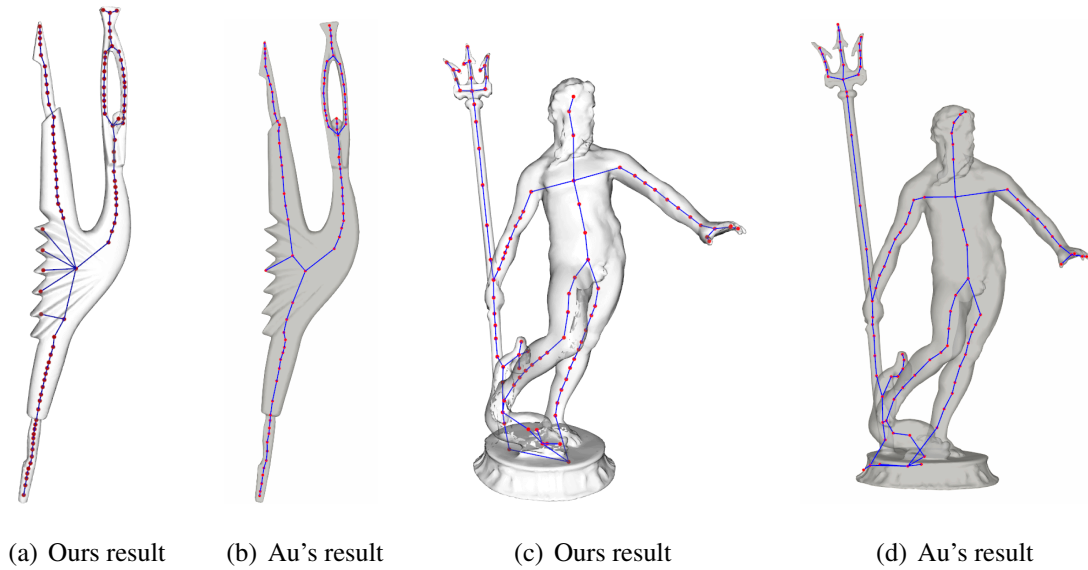


Figure 5.22: Skeleton results of Neptune and Dancer

We get a more coarse skeleton compared to the results of the Au's algorithm. Because of the limitation of MSP function, the skeleton mesh exists several medial-axis regions. It results in a coarse skeleton after the LOD simplification. From the view of applications, there is no difference between our skeletons and the Au's.

The only parameter in our system is the saturation of the edge. By controlling the value of the saturation, we can remove some unwanted branches and control the fineness of skeleton. Figure 5.23 to 5.25 show the skeleton results by using different threshold of saturation.

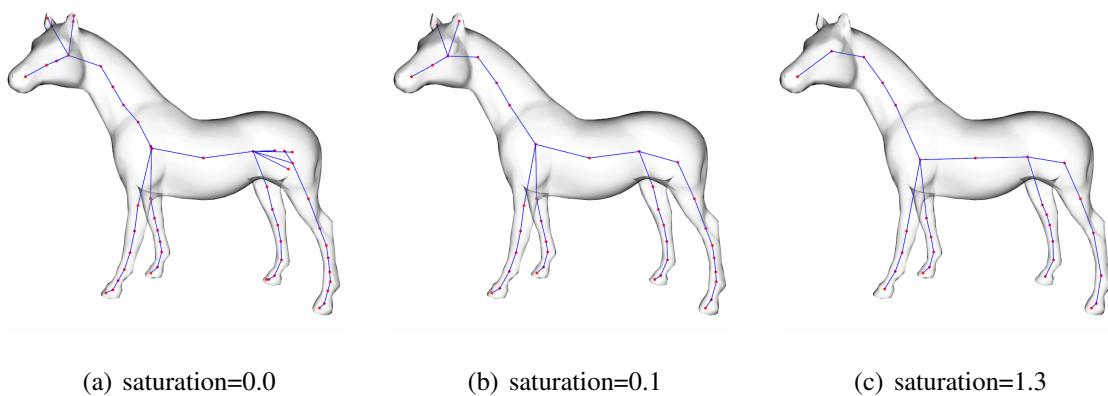


Figure 5.23: Skeleton results of horse model by using different saturation

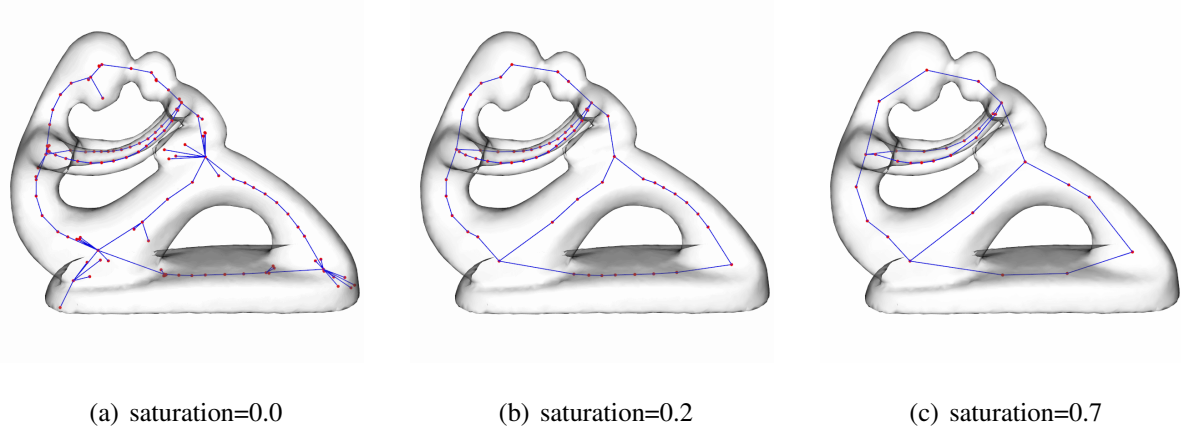


Figure 5.24: Skeleton results of Fertilty model by using different saturation

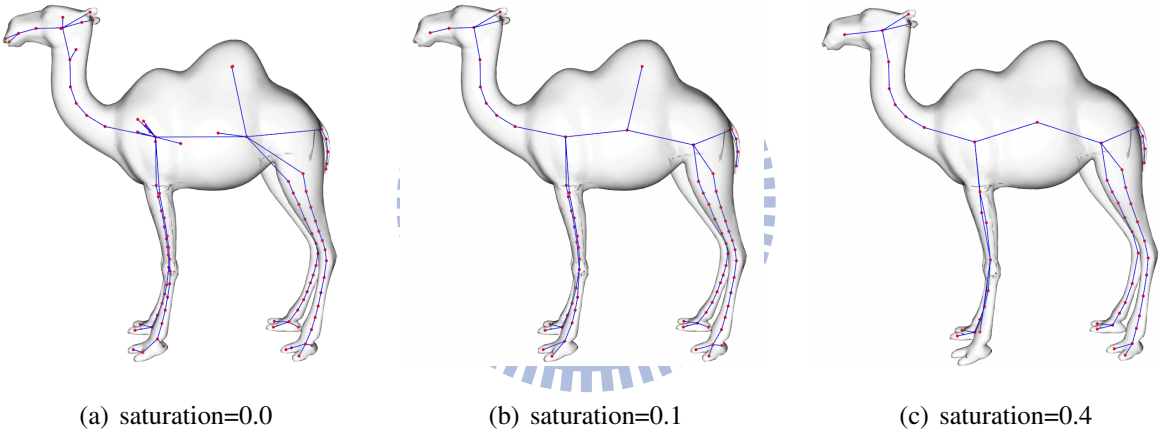


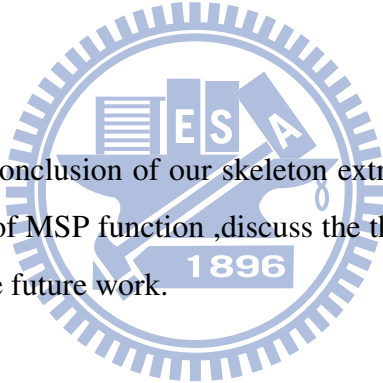
Figure 5.25: Skeleton results of Camel model by using different saturation

## CHAPTER 6

---

# Conclusions

---



We give a brief summary and conclusion of our skeleton extraction algorithm in this chapter. We also analyze the properties of MSP function, discuss the limitations of our system, and propose several directions of the future work.

### 6.1 Summary

We propose a novel algorithm to generate skeletons from the triangle meshes. We apply the MSP function to assist the skeleton extraction. The MSP function is a scalar surface function to measure the volume of a model. From the MSP function, we know the MSP slice is a candidate region that skeleton may pass. We compute the MSP slice for each vertex and perform the Repulsive force model to locate the potential skeletal position within the MSP slice. The Repulsive force model is a kind of skeletonization algorithm of potential field approach. We lower the searching space into 2-D slice and we only need to find a local minimum of the force field within each slice.

Because some MSP slices are perturbed due to the local fluctuations or terminal deviation,

we develop the method to estimate the error of each MSP slice and apply the normal-cone refinement to get a better slice. After obtaining the skeletal positions of each vertex, we transform the input mesh into skeleton mesh by moving each vertex to the corresponding skeletal position. Although the skeleton mesh is nearly zero volume, but the connectivity is still same with the input mesh. We apply LOD simplification to simplify the skeleton mesh until we obtain the 1-D skeleton. The LOD simplification of our system is dedicated to preserve the skeleton shape of the skeleton mesh, rather than local shape and features in traditional LOD simplification. By the properties of MSP function, we develop the constraints and error metrics to preserve the topology of the input model. After finishing the LOD simplification, we apply the skeleton structure filtering on the final skeleton to remove unwanted branch and combine joints. We generate a skeleton and finish our algorithm.

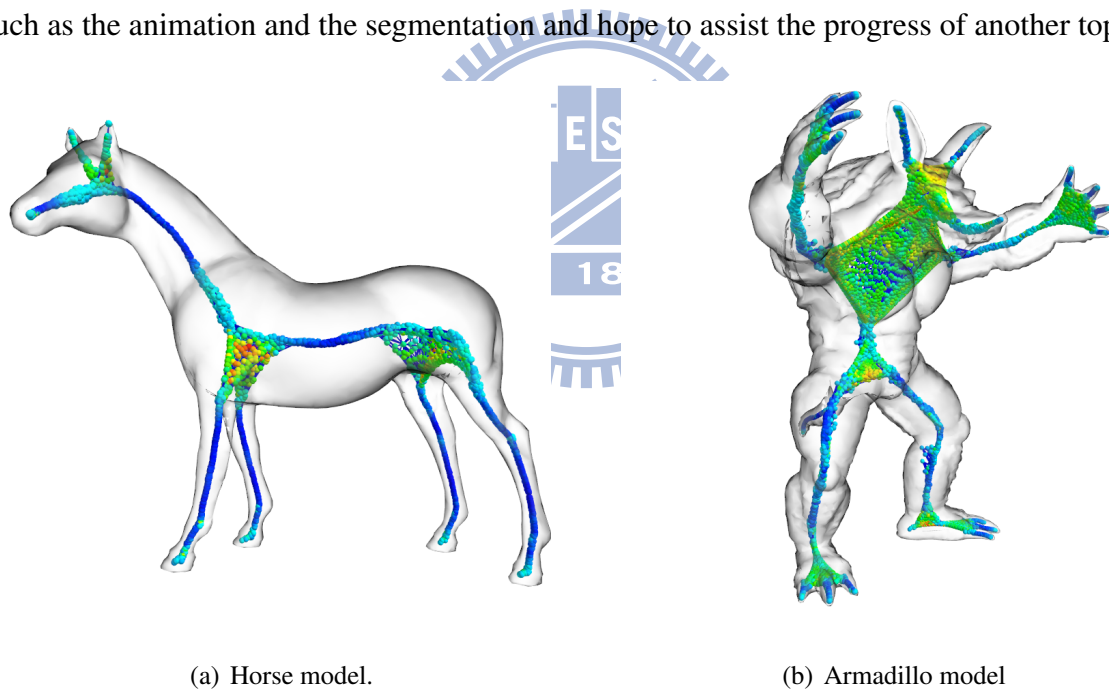
## 6.2 Limitations and Future works

Our skeleton extraction algorithm is capable to generate the compact skeleton for the triangle-mesh models, especially the models of articulated shape. For example, the animal models and the human models are articulated. Generally speaking, the articulated models are composed by the generalized cylinder components. For such models, the MSP function can provide reasonable information and generate more skeleton-like mesh. But for the models of plane shape or the models that own much prominent features such as the dancing-children and the gargoyle models, the MSP function is limited to the local features and can not provide global information. So that the skeleton meshes of these models are much similar to the medial axis. Through this kind of skeleton mesh, we will generate more coarse skeleton. Furthermore, because the MSP function can handle closed models only, we are not able to extract any open meshes to generate skeletons.

In the future, we would like to enhance our skeleton extraction algorithm from two directions. The first direction is to change the measurement of MSP functions to improve the quality of skeleton mesh. The MSP function measure the local volume by the slice with minimum

perimeter. But the slice with minimum perimeter does not imply that the MSP slice must exist an skeleton node. If we can enhance the MSP functions, we can obtain more accurate skeleton mesh and better skeleton results.

The second direction is to apply some refinements on the skeleton mesh directly. We could use the spatial coherence of the skeleton mesh to update the skeleton nodes of low confidences. We update the nodes of low confidence close to the adjacent nodes of high confidences. This could be achieve more thinned skeleton mesh. Figure 6.1 show our testing results of the skeleton mesh which had been refined using the spatial coherence. Clearly, there exist some problems of the skeleton mesh. Such as the design of error metrics and constraints. Besides the two directions of the future work, we would also like to apply our skeleton results to some applications such as the animation and the segmentation and hope to assist the progress of another topic.



(a) Horse model.

(b) Armadillo model

Figure 6.1: Spatial Coherence refined skeleton mesh.

---

# Bibliography

---

- [ABE07] D. Attali, J.-D. Boissonnat, and H. Edelsbrunner. Stability and computation of medial axes: a state of the art report. In B. Hamann T. Möller and B. Russell, editors, *Mathematical Foundations of Scientific Visualization, Computer Graphics, and Massive Data Exploration*. 2007.
- [AC97] N. Ahuja and J.-H. Chuang. Shape representation using a generalized potential field model. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(2):169–176, 1997.
- [ACK00] N. Amenta, S. Choi, and R. K. Kolluri. The power crust, unions of balls, and the medial axis transform. *Computational Geometry: Theory and Applications*, 19:127–153, 2000.
- [AHY94] G.H. Abdel-Hamid and Y. H. Yang. Multiresolution skeletonization: An electrostatic field-based approach. In *ICIP (1)*, pages 949–953, 1994.
- [ATC<sup>+</sup>08] K. C. Au, C. L. Tai, H. K. Chu, D. C. Cohen-Or, and T. Y. Lee. Skeleton extraction by mesh contraction. In *ACM SIGGRAPH 2008*, pages 1–10. ACM, 2008.
- [BA95] G. Bertrand and Z. Aktouf. Three-dimensional thinning algorithm using subfields. volume 2356, pages 113–124. SPIE, 1995.
- [BKS01] I. Bitter, A. E. Kaufman, and M. Sato. Penalized-distance volumetric skeleton

- algorithm. *IEEE Transactions on Visualization and Computer Graphics*, 7(3):195–206, 2001.
- [Blu67] H. Blum. A transformation for extracting new descriptors of shape. *Models for the Perception of Speech and Visual Form*, pages 362–380, 1967.
- [BSB<sup>+</sup>00] I. Bitter, M. Sato, M. Bender, K. T. McDonnell, A. Kaufman, and M. Wan. Ceasar: Accurate and robust algorithm for extracting a smooth centerline. Washington, DC, USA, 2000. IEEE Computer Society.
- [CCZ07] M. Couprie, D. Coeurjolly, and R. Zrouf. Discrete bisector function and euclidean skeleton in 2d and 3d. *Image Vision Comput.*, 25(10):1543–1556, 2007.
- [CSM07] N. D. Cornea, D. Silver, and P. Min. Curve-skeleton properties, applications, and algorithms. *IEEE Transactions on Visualization and Computer Graphics*, 13(3):530–548, 2007.
- [CSYB05] N.D. Cornea, D. Silver, X.S. Yuan, and R. Balasubramanian. Computing hierarchical curve-skeletons of 3d objects. 21(11):945–955, October 2005.
- [CTK00] J. H. Chuang, C. H. Tsai, and M. C. Ko. Skeletonization of three-dimensional object using generalized potential field. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11):1241–1251, 2000.
- [DS06] T. K. Dey and J. Sun. Defining and computing curve-skeletons with medial geodesic function. In *Proceedings of the fourth Eurographics symposium on Geometry processing*, pages 143–152, Aire-la-Ville, Switzerland, 2006. Eurographics Association.
- [DZ04] T. K. Dey and W. Zhao. Approximating the medial axis from the voronoi diagram with a convergence guarantee. *Algorithmica*, 38:387–398, 2004.

- [GAHY96] T. Grigorishin, G. Abdel-Hamid, and Y. H. Yang. Skeletonization: An electrostatic field-based approach. Technical report, 1996.
- [GB90] W. Gong and G. Bertrand. A simple parallel 3d thinning algorithm. volume i, pages 188–190 vol.1, Jun 1990.
- [Gre03] R. Green. Spherical harmonic lighting: The gritty details. *Archives of the Game Developers Conference*, March 2003.
- [HJWC09] T. C. Ho, H. X. Ji, S. K. Wong, and J. H. Chuang. Mesh skeletonization using minimum slice perimeter function. In *Proceedings of CGW*, 2009.
- [Kon97] T. Y. Kong. Topology-preserving deletion of 1's from 2-, 3- and 4-dimensional binary images. In *DGCI*, pages 3–18, 1997.
- [KR89] T. Y. Kong and A. Rosenfeld. Digital topology: Introduction and survey. *Computer Vision, Graphics, and Image Processing*, 48(3):357–393, December 1989.
- [LB01] C. Lohou and G. Bertrand. A new 3d 12-subiteration thinning algorithm based on p-simple points. *Electronic Notes in Theoretical Computer Science*, 46:33 – 52, 2001. IWCIA 2001, 8th International Workshop on Combinatorial Image Analysis.
- [LL92] F. Leymarie and M.D. Levine. Simulating the grassfire transform using an active contour model. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(1):56–75, 1992.
- [LWM<sup>+</sup>03] P. C. Liu, F. C. Wu, W. C. Ma, R. H. Liang, and M. Ouhyoung. Automatic animation skeleton construction using repulsive force field. In *In Pacific Graphics*, pages 409–413, 2003.



- [Mor81] D. Morgenthaler. Three-dimensional simple points: serial erosion, parallel thinning, and skeletonization. Technical report, Computer Vision Lab, University of Maryland, 1981.
- [MWL02] C. M. Ma, S. Y. Wan, and J. D. Lee. Three-dimensional topology preserving reduction on the 4-subfields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(12):1594–1605, 2002.
- [OI92] R. Ogniewicz and M. Ilg. Voronoi skeletons: Theory and applications. In *in Proc. Conf. on Computer Vision and Pattern Recognition*, pages 63–69, 1992.
- [OK95] R. L. Ogniewicz and O. Kbler. Hierarchic voronoi skeletons, 1995.
- [PK98] K. Palágyi and A. Kuba. A 3d 6-subiteration thinning algorithm for extracting medial lines. *Pattern Recognition Letters*, 19(7):613 – 627, 1998.
- [PK99a] K. Palágyi and A. Kuba. Directional 3d thinning using 8 subiterations. In *DGCI*, pages 325–336, 1999.
- [PK99b] K. Palágyi and A. Kuba. A parallel 3d 12-subiteration thinning algorithm. *Graphical Models and Image Processing*, 61(4):199–221, 1999.
- [Pud98] C. Pudney. Distance-ordered homotopic thinning: a skeletonization algorithm for 3d digital images. *Comput. Vis. Image Underst.*, 72(3):404–413, 1998.
- [RT95] J. M. Reddy and G. M. Turkiyyah. Computation of 3d skeletons using a generalized delaunay triangulation technique. *Computer-Aided Design*, 27(9):677–694, 1995.
- [SBB<sup>+</sup>00] M. Sato, I. Bitter, M. A. Bender, A. E. Kaufman, and M. Nakajima. Teasar: Tree-structure extraction algorithm for accurate and robust skeletons. In *Proceedings of the 8th Pacific Conference on Computer Graphics and Applications*, page 281, Washington, DC, USA, 2000. IEEE Computer Society.

- [SC94] P. K. Saha and B. B. Chaudhuri. Detection of 3-d simple points for topology preserving transformations with application to thinning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(10):1028–1032, 1994.
- [SLSK07] A. Sharf, T. Lewiner, A. Shamir, and L. Kobbelt. On-the-fly curve-skeleton computation for 3d shapes. *Comput. Graph. Forum*, 26(3):323–328, 2007.
- [SSCO08] L. Shapira, A. Shamir, and D. Cohen-Or. Consistent mesh partitioning and skeletonisation using the shape diameter function. *Vis. Comput.*, 24(4):249–259, 2008.
- [TVD08] J. Tierny, J.-P. Vandeborre, and M. Daoudi. Enhancing 3d mesh topological skeletons with discrete contour constrictions. *The Visual Computing.*, 24(3):155–172, 2008.
- [TZCO09] A. Tagliasacchi, H. Zhang, and D. Cohen-Or. Curve skeleton extraction from incomplete point cloud. *ACM Transactions on Graphics, (Proceedings SIGGRAPH 2009)*, 28(3):1–9, 2009.
- [WDK01] Mi. Wan, F. Dachille, and A. Kaufman. Distance-field based skeletons for virtual navigation. In *Proceedings of the conference on Visualization '01*, pages 239–246, Washington, DC, USA, 2001. IEEE Computer Society.
- [WML<sup>+</sup>03] F.-C. Wu, W. C. Ma, P. C. Liou, R. H. Laing, and M. Ouhyoung. Skeleton extraction of 3d objects with visible repulsive force, 2003.
- [WML<sup>+</sup>06] F. C. Wu, W. C. Ma, R. H. Liang, B. Y. Chen, and M. Ouhyoung. Domain connected graph: the skeleton of a closed 3d shape for animation. *The Visual Computer*, 22(2):117–135, 2006.
- [YBM04] Y. Yang, O. Brock, and R. N. Moll. Efficient and robust computation of an approximated medial axis. In *In Proceedings of the ACM Symposium on Solid Modeling and Applications*, pages 15–24, 2004.