Simplification and Refinement of Point Cloud Models

ZHANG, HAI NING

A Thesis
in
The Department
of
Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements
for the Degree of Master of Computer Science at
Concordia University
Montreal, Quebec, Canada

December 2004

Your file Votre référence
ISBN: 0-494-04460-8
Our file Notre référence
ISBN: 0-494-04460-8

Abstract

Simplification and Refinement of Point Cloud Models

ZHANG, HAI NING

Point cloud models (PCMs) are 3D point datasets from 3D acquisition device. They are densely sampled from the surfaces of objects. Each point in the PCM consists only of 3D coordinates, sometimes also of normal vector without information of structures, or connectivity. The size of a PCM may be up to several millions. Research reported in this thesis focuses on developing a two-stage technique: to simplify a very dense PCM into compact PCMs according to required compact rate, and on reverse, to refine the compact PCMs at least as dense as or denser than the original.

In stage one, a PCM (only including 3D coordinates), will be simplified into a compact one with an octree and *principle component analysis* (PCA). From the result of PCA, features of the local surface defined by points in a leaf node can be detected. For a specific feature in a leaf node of the octree, corresponding simplification algorithm is applied to resample points from original one. The points in compact PCM also have information of *normal vector and feature*. On stage two, a dense PCM is obtained by refining the compact PCM from stage one with the refinement schemes. Finally, in order to check the results of simplification and refinement, we make two comparisons. The first is between compact PCM and the original PCM in order to *determine if it is good or bad for a compact PCM to represent the original PCM.* The second comparison happens between refined and original PCMs in order to verify how many features are lost during the two stages, and for survived features, how different they are from the ones in the original PCM. For both comparisons, a normalized result is reported.

Acknowledgement

Finally, the thesis is almost finished, and I am enjoying the good fortune of having got academic and non-academic, physical and spiritual help and attention from many people. Here is just the official place to show my appreciation.

My supervisor Professor S.P. Mudur: In last two years, he helped me enter the amazing area of Computer Graphics. Even more, he devoted his precious time to help me in this research. Without his inspiring suggestions, encouragement, and financial support, it was impossible for me to complete this work.

Mr. Hao Zhou, Mr. Sushil Bhakar: From the two doctoral students of my supervisor, helped me a lot with many technical issues about *point cloud models.*

CyberWare: A 3D-scanning device company made available point cloud models on its website. I got all models from their website for my experiments.

Finally, I deeply appreciate the encouragement and support from my family. My wife HAIKUN always gives me love and encouragement. My parents in China always express their pride in me and my work, whenever I called them.

Table of Contents

List of Figures

List of Tables

Chapter 1
Introduction

1.1 Introduction of Point Cloud Models (PCMs)

Recent advances in 3D scanning technology have enabled sampling a tremendous number of points, called *point cloud model* (PCM) with 3D-coordinates (x, y z) from a given object. In the last around 20 years, much work has been done on *point cloud models.* In recent years, this has been becoming a very hot area of research in Computer Graphics.

1.1.1 Reverse Engineering (RE)

Reverse engineering (RE) used to be a nefarious term. It formerly meant making a copy of a product, or the outright stealing of ideas from competitors. In current usage, however, RE has taken on a more positive character and now simply refers to the process of creating a descriptive data set from a physical object. More precisely, it usually starts with acquisition of digital point surfaces sampled data from physical/master model and then developing surfaces on this point data set for other engineering purposes [68].

1.1.2 Typical RE Applications

This has come about over the last fifteen or more years due to the intense parallel development of many different types of three dimensional digitizing devices, and the powerful reverse engineering software that allows the data they produce to be manipulated into a useful form. These applications can be categorized as follows:

- ➢ Creating data to refurbish or manufacture a part for which there is no CAD data, or for which the data has become obsolete or lost.
- ➢ Comparing a fabricated part to its CAD description or to a standard item.
- ➢ Creating 3D data from a model or sculpture for animation in games and movies.
- ➢ Creating 3D data from an individual, model or sculpture for creating, scaling or reproducing artwork.
- ➢ Generating data to create dental or surgical prosthetics, tissue-engineered body parts, or for surgical planning.
- ➢ Architectural and construction documentation and measurement.

Generally, there are two parts to any reverse engineering application: scanning and data manipulation. Scanning, also called digitizing, is the process of gathering the requisite data from an object using devices. It also allows several different scans of an object to be melded together so that the data describing the object can be defined completely from all sides and directions. What comes out of each of these data collection devices is a description of the physical object in three-dimensional space called a point cloud Model (PCM). Manipulation is the process of dealing with given PCM according to application requirement – constructing mesh surface from PCMs, merging several PCMs together to building new PCMs, resampling PCMs for rendering, editing PCMs directly for artwork, compressing and refining PCMs for storage and network transmission, etc.

1.1.3 Point Cloud Models (PCMs)

Point cloud models typically define numerous points on the surface of the object in terms of x, y, and z coordinates. For each triple (x, y, z), it is a surface coordinate of the original object.

However, some scanners, such as those based on X-rays, can see inside an object. In that case, the point cloud also defines interior locations of the object, and may also describe its density.

For use by graphics systems, sometimes, besides geometric information, several attributes associated with object surfaces, such as color, normal vector, and local sampling density, can also be obtained from 3D scanners.

There is usually far too much data in the data set collected from the scanner or digitizer, and some of which might be unwanted noise. Also the data set is typically densely sampled. It is unstructured, lacking connectivity information, and is increasingly becoming very large. Without further processing, the data is not in a form that can be used by downstream applications such as CAD/CAM software or in rapid prototyping. Reverse engineering software is used to edit the point cloud data, establish the interconnectedness of the points in the cloud, and translate it into useful formats such as surface models. The models used in this research are not representative. The practical sizes of PCMs are much larger than them. Interested readers can refer to Digital Michelangelo project at Stanford University Computer Graphics Lab [60] and CyberWare [61] for more details.

In this thesis, the PCMs used for experiments include only surface information without inside data. In order to make our idea more clear, we only consider coordinates information.

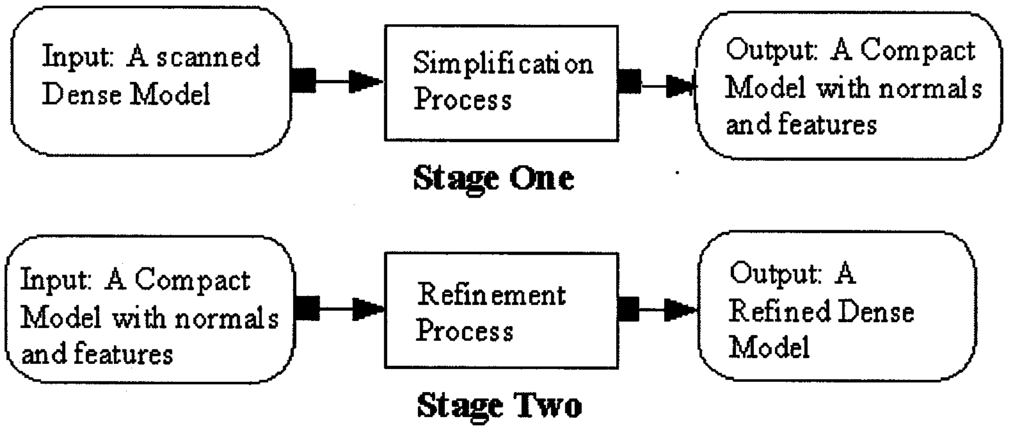
1.2 Statement of Purposes

1.2.1 Challenges and Our Ideas

As mentioned before, Point Cloud Models are usually densely sampled points on the surfaces of the object. It has been proved that using points, as a universal representation is an easy-accepted concept because graphics content has been becoming abundant more and more with extreme geometric complexity. On the other hand, the implementation of effective geometry and graphics processing techniques was never simple, and progressing of computing power is always running after the challenge from computer graphics, especially, for practical applications of PCMs on modeling, shape editing, and visualizing.

Furthermore, to save these extreme large models directly in computer system is storage consuming and not suitable to transmit over the network system.

According to these challenges, we propose to build a system to simplify and refine PCMs in double ways (**Figure 1**).



Input: A scanned Dense Model
Simplification Process
Output: A Compact Model with normals and features
Stage One
Input: A Compact Model with normals and features
Refinement Process
Output: A Refined Dense Model
Stage Two

Figure 1: Dual stages of simplification and refinement

In stage one, the output; a compact model can be used for multi-purposes: rendering and visualization, and model-editing. It is easy to transmit over network system and economic for storage. The density of compact model can be changed according to different requirement. Also, the normal and local surface features are associated with each sample point in compact models. In stage two, the input is the compact model from stage one, and after refinement process according to given density requirement, a refined model is output.

1.2.2 Several Approaches to Represent Data in PCMs

Current point-based representations of data include: bounding balls hierarchy [17], octree hierarchy [57, 016], and progressive implicit surface representations [09, 045].

In this thesis, we prefer to use octree for the hierarchy structure. Apparently, octree is easier to subdivide and it can cover volumetric space more compactly than bounding ball (sphere or ellipsoid).

1.2.3 Overview of the Idea

At stage one of compression, we use the octree to voxelize an input model to build up a hierarchy structure, then Principal Component Analysis (PCA) is applied to each leaf node of the octree. According to the results of PCA, eigenvalues and eigenvectors for a given leaf node, the local surface feature of that node can be decided: flat, creased, or cornered. For the flat area, the eigenvector associated to the smallest eigenvalue must be the normal vector. With neighborhood algorithm of octree, we can decide the normal vector for crease and corner areas as well.

Based on the feature information, we resample the given model from the input data set according to given compact rate. In brief, larger the first eigenvalue, more points are kept in that leaf node. We assigned the normal vector of the leaf node to the re-sampled points in that node. The result of stage on is a compact PCM consisting of resampled points. Each point in a compact has 3D coordinates, feature, and normal vector.

At stage two of refinement, the compact model is input and several parameters for refinement should be set: density or percentage of refinement. Also, we use octree to voxelize the compact model and the criteria of stopping subdivide leaf node is that each leaf node should contain at least 6 points. Based on the points in each leaf node, a neighborhood of points is constructed. Then refinement schemes are applied to the neighborhoods. We then delete the octree structure, and built a new octree with the same parameters. The only difference is that we use a small offset to make sure that the new octree is different from the old one. The refinement process can be applied recursively until the density command is satisfied.

Finally, in order to verify assess of our ideas, we make two comparisons: the compact PCM with the original PCM and the refined PCM with the very original PCM.

1.3 Structure of the Thesis

This thesis includes 7 chapters, and a bibliography. Generally, this thesis is arranged according to the process of experiments.

In chapter 1, we introduce the point cloud model and our objectives in this research. Definition of terms globally for the thesis is also been provided.

In Chapter 2, we survey related research and techniques, which precede and is parallel with the experiments, which we conducted. The survey is related to the experiments, including several aspects, such as rendering and visualization of PCMs, editing of PCMs, up sampling and down sampling of PCMs. As for up sampling and down sampling, it is an exhaustive survey and includes all papers, currently published.

Chapter 3 to chapter 6 describes the experiments and algorithms in this research. These chapters share the same structure. At the very beginning, the purpose of the experiment or algorithm will be given followed by a related brief survey. Then come the theoretical part, including the programming issues. The final section is the summary of the chapter.

Chapter 3 mainly focuses on octree, including definition, data structure, and algorithms, because it is the essential part of all our experiments.

How to get a compact model from the original one is discussed in chapter 4

In Chapter 5 works on how to get a refined model from the compact one. At the beginning of this chapter, subdivision surface is introduced because the refinement schemes are borrowed form subdivision surfaces.

Chapter 6 makes two comparisons: comparing the refined and compact PCMs with the original respectively. The first comparison is to find how well a compact PCM to represent its original. The second PCM is based on features and includes two parts: one is to evaluate how many features are lost during processes, and another is for retained features to evaluate how much difference they are comparing with the corresponding ones in the original PCM.

Chapter 7 includes concluding remarks and future work.

Bibliography includes all the books, papers, notes, and websites, which I have referred for experiments and programming.

1.4 Models Used in Experiments

All the models, used in this research, are downloaded from CyberWare (http://www.cyberware.com/samples), a company focusing on 3D scanning systems and software. On this website, there are a lot of free samples of *point cloud models*. These models are scanned from desktop 3D scanner, so they are a little small compared to models from Stanford.

File format of point cloud models used in this work has to be mentioned. The original point cloud models are saved in PLY file format, which is defined by Stanford Graphics Lab for PLYVIEWER software [61]. In experiments, they have been translated into models, which only contain 3D-coordinates information line for line, with format translators. The command-line translators can be obtained from CyberWare website.

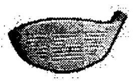
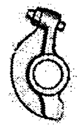
Models	Pictures	Number of points
Rabbit		67,038
Igea Artifact (Venus)		134,345
Dinosaur Sculpture		56,194
Ball Joint		139,306
Golf Club		209,779
Isis		187,689
Hip Bone		530,168
Rocker Arm		40,178

Table 1: 3D point cloud models used in this research. Pictures for models are also from CyberWare's website [61]

1.5 Definition of Terms

In Section 1.2.3, we defined *point cloud models* (PCMs), other terms we need in this research are defined in the following subsections.

1.5.1 Multi-resolution on PCMs

For polygonal meshes the up sampling and down sampling operations are applied to switch between different levels, where "resolution" refers to both vertex density and amount of geometric details. In the case of point cloud models, both concepts of resolution have to be treated separately, since a point-sampled surface representing a low geometric detail still has to provide a sufficiently high sampling density. Hence, hierarchies of different resolution for point cloud models are only to increase the efficiency of filtering operations.

1.5.2 Features on Surfaces

For a local surface area of an object, we can assign one of the following features to describe it: flat, crease or edge, boundary, or corner. Such classification is meaningless when the local area is much larger. We can consider crease as the joint of two pieces of smooth surface, and the corner is two or more crease meet there. Boundary is defined as the boundary of a piece of surface. In our experiment, we only use close models, so there is no boundary features in models. Usually, corner and crease areas need more vertices or points to describe in either mesh or point cloud models. Please see chapter 3 for more details.

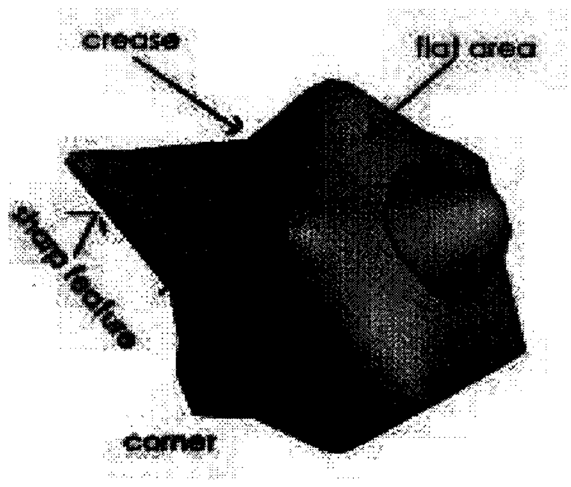


Figure 2: Features on the surface [1]

1.5.3 Subdivision Surfaces and Schemes

Subdivision surface defines a smooth curve or surface as the limit of a sequence of successive refinements. For instance, Figure 3 shows an initial control mesh, and the one after several refinement steps.

[1] Pictures in Figure 2 and 3 are from http://grail.cs.washington.edu/projects/subdivision/, and text in Figure 2 is added by us.

Figure 3: Example of Subdivision Surface: the control mesh (Left) and the refined result.

The method used to refine the mesh is called subdivision scheme, which define the stencil and formulas to insert new vertices and compute new position of old vertices. More details will be provided in chapter 6.

1.5.4 Principal Component Analysis

Principal Component Analysis (PCA) is a tool having been used successfully to analyze empirical data in a variety of fields. Principal component analysis (PCA) is a classical statistical method.

Chapter 2
Survey on Research and Technology Related to PCMs

In this Chapter, we present an overview of research and technology in this area. In particular, we provide a comprehensive survey of for up sampling and down sampling.

The point-sample surface could be in two forms: image range data from laser range scanners, and point cloud models. Image range data consists of regularly spaced samples in a two dimensional domain, with a depth value associated with each sample point. These are sometimes referred to also as depth images. *Point Cloud Models* is irregularly spaced and sampled density may vary considerably over the complete surface of the model.

A rendering process of point sampled surfaces will result in an image, usually a color shaded picture of the 3D model. However, geometry recovery processes will give us many different kinds of models: 3D triangle models, boundary representation of the model in terms of vertices, edges and faces (B- Rep Model), etc.

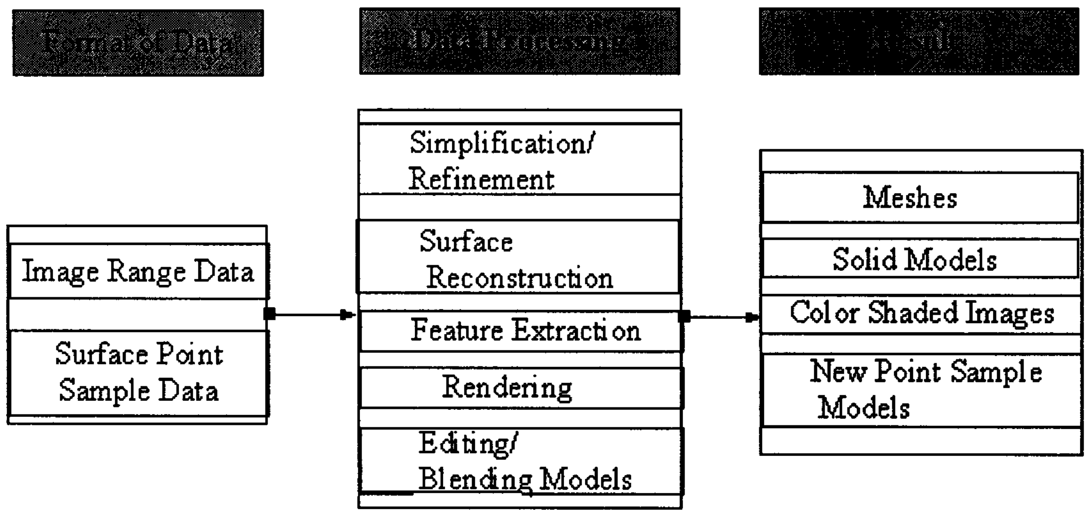


Figure 4: Overview of processing techniques of PCMs

2.1 Surface Reconstruction

Surface Reconstruction is the procedure of constructing a mesh from a given point cloud model. A lot work has been done in this area. Here, only representative papers are surveyed.

➢ **A general way for all applications**: In 1992, Hoppe et al [47] introduced a general algorithm that solves this problem for all applications, rather than giving different algorithms for different applications. The algorithm has two stages. Define a function f where $f(x, y, z)$ estimates the distance from the point (x, y, z) to the unknown surface M. The domain of f is restricted to only a relevant region in 3D space. We will use the zero set $Z(f)$ of f, i.e. the set $\{(x, y, z) : f(x, y, z) = 0\}$, as our approximation of M. To define the function, we calculate a tangent plane $Tp(x_i)$ to M' at x_i. Also we assign an orientation, i.e., a normal vector n_i, to each tangent plane. We also choose a point s_i on each tangent plane to be the *center* of the plane.

Finally we define *f*(*p*) to be the dot product of (p - s_i) and n_i. Finding a consistent way to assign orientations to the tangent planes is a nontrivial problem--this paper uses a graph optimization technique to solve the problem. It uses contouring algorithm [47] to approximate *Z*(*f*) by a simplicial surface. This paper uses a modified version of the *marching cubes algorithm*, which samples *f* at the vertices of a cube, breaks the cube into tetrahedra, and finds the intersection with the zero set inside those tetrahedra. Only cubes intersecting the zero set are visited.

- **Radical Basis Function:** In 2001, Carr et al [65] suggest to use Radical basis Functions (RBF) constructing compact functional description of surface data, making interpolation and extrapolation easy. With RBF, Derivatives (and hence gradients and normals) can be computed analytically. Also, level sets without self-intersection can be extracted. RBFs make mesh simplification and remeshing easier.
- **Reconstructing by parameterization:** In 2001, Floater et al [66] present a way to triangulate a point cloud model, which is assumed to be sampled from a single surface patch. A surface patch is any set that is homeomorphic to a disc in the plane. Now, the basic idea is to Map the points on the boundary of the point cloud to the boundary of some convex polygon D in the plane (a meshless parameterization), then triangulate the parameter domain (for example, use the Delaunay triangulation), and map back to the point cloud to obtain a triangulation of the point cloud.
- **Regular Triangulation:** In 1995, Bajaji et al [62] propose a method dealing with connected, orientable manifolds of unrestricted topological type, given a sufficiently dense and uniform sampling of the objects surface. It is capable of reconstructing both the model and a scalar field over its surface. It uses regular triangulations, power diagrams and weighted alpha-

shapes for efficiency of computation and theoretical soundness. It generates a representation of the surface and the field based on barycentric Bernstein-Bèzier polynomial implicit patches, which are guaranteed to be smooth and single-sheeted.

- **Voronoi diagram and Delaunay Triangulation:** In 1998, Amenta et al [44] present a simple new algorithm for surface reconstruction based on Voronoi diagrams and Delaunay triangulations. Their algorithm computes a piecewise-linear surface, vertex set exactly equal to the sample points, with a provable guarantee: if the original surface was sampled sufficiently densely, then the reconstructed surface lies within a small error tolerance of the original. "Sufficiently densely" is defined locally as proportional to the distance to the medial axis, which incorporates both curvature and proximity to "other" parts of the surface. Dey et al [62] propose an algorithm call concone, also use Voronoi diagram for surface construction. In addition, they can detect under sampling area and features on the surface [62]. Also, they suggest a hole-free algorithm based on the concept of concone [63].

2.2 Rendering and Visualization of PCMs

PCMs themselves usually are modeled as a set of *point primitives*: instead of having just a set of points, we may instead have a set of tangential disks, spheres, quadratic surfaces, or higher-degree polynomial patches. Using point primitives instead of just points helps fill in spaces between points, as well as efficiently rendering small regions of triangle meshes.

2.2.1 Concept of Surfel for PCMs

In 2000, Pfister et al [40] gave out the concept of Surfel (surface element). Surfels are a powerful paradigm to efficiently rendering complex geometric objects at interactive frame rates. *Surfels* are **point primitives** without explicit connectivity, unlike classical surface discretizations.

For a given Surfel, its attributes consist of depth, texture color, normal, and others. This concept is accepted by researchers popularly for rendering PCMs.

Also, Pfister et al [40] suggest a rendering pipeline. In preprocessing stage, an input model is voxelized into a *layered depth cube* (LDC). During rendering, the LDC is traversed top to bottom. For each block, view-frustum culling is applied.

2.2.2 Splatting

QSplat: *QSplat* [17] is a multi-resolution point rendering system. QSplat uses a hierarchy of bounding spheres for visibility culling, level-of detail control, and rendering. Each node (sphere) of the tree contains the sphere center and radius, a normal the width of the normal cone, and optionally a color. The hierarchy is constructed as a pre-process, and is written to disk. For display, when the bounding sphere hierarchy is recursed, invisible nodes are culled.

Currently, for high quality and efficient point-based rendering, a splatting approach is doubtless the best choice since such approaches are supported by modern GPUs [58]. However, if the input model is under-sampled, QSplat will produce visual artifacts on the silhouette and high curvature areas because from the geometric point of view, a surfel set describing a continuous texture function is a simple set of oriented overlapping disks.

2.2.3 Point Set Surfaces

Alexa et al [04] [45] advocate using point sets directly to represent shapes. They use *moving least squares* to up sample and down sample input models according to screen resolution, and also use a bounding sphere hierarchy to arrange point set for culling and view dependence.

2.2.4 Visualization from Statistic view

Kalaiah et al [08] propose to render PCMs from the statistic aspect. Their method is appropriate when the input model contains details heavily or the precision needs are high. Firstly, an input model is voxelized with octree. Then the hierarchy of *Principal Component Analysis* (PCA) is constructed form the octree. Finally, the PCA hierarchy is shown with spatial PCA ellipsoids. With scaling of the intercepts of ellipsoids, they can get a hole-free representation.

2.2.5 Other Miscellaneous Methods

In addition, several other ideas are given to render and visualize the PCMs. Bhakar et al [16] also construct hierarchy on the input model by voxelization, and for each voxel, only one sample are kept. Their idea is suitable for global rendering. Guennehaud et al [02] apply improved subdivision surface schemes (interpolating) to neighborhoods on PCMs. they focus on resolve the problems when we rendering an under sampling model with splating.

2.3 Up Sampling and Down Sampling of PCMs

In the paper [48], an efficient and general multi-resolution framework is proposed for compression of point-based models. They build multi-resolution hierarchy by recursively computing coarser approximations of the point set, exploiting similarities of each attributes by finding a specific order of the samples which minimize entropy (least-squares planes as a local approximation of the surface). To that end, they employ a matching algorithm with finds neighborhoods with high correlation based on distance measures for all attributes we want to compress. Next they compute the hierarchy of details coefficients that describes how to successively reconstruct the original model from the lowest resolution point set. They eventually end up with a set of quantized detail coefficients

Gu et al [50] decompose arbitrary surfaces into completely regular structures called *Geometry Images*, ranged in a 2D array. Their approach enables the encoding of additional appearance attributes using the same implicit surface parameterization. They propose to remesh an arbitrary surface onto a completely regular structure called a geometry image. It captures geometry as a simple 2D array of quantized points. Surface signals like normals and colors are stored in similar 2D arrays using the same implicit surface parameterization—texture coordinates are absent. To create a geometry image, we cut an arbitrary mesh along a network of edge paths, and parameterize the resulting single chart onto a square. For up scaling and down scaling, traditional algorithms for 2D images can be applied directly to the 2D array. *Geometry Images* require a certain mesh topology and a global parameterization, which is very hard to achieve.

Gumhold et al [06] propose methods of detecting features from point clouds and reconstruct them. They first construct neighbor graph on the input point cloud model with k nearest neighbors and Riemannian Graph. Then the point neighborhoods are constructed for *Covariance Analysis*, or *Principal Components Analysis* (PCA). According the eigenvalues obtained from the PCA, (for details, interested readers can refer to chapter 3), features can be detected on local surfaces.

However, the constructing Riemanneian Graph and point neighborhoods are heavily time consuming.

Linsen [13] introduces efficient algorithms for simplification and refinement. For simplification, he associates an information content measure with every input point and subsequently removes points featuring the lowest entropy. His information measure allows for local curvature and RGB color changes. The algorithm is simple and procedures visually appealing results, but gives no guarantees on the density of the output point set. Therefore, extremely non-uniformly distributed input point sets will necessarily result in simplified point sets of insufficient density. Also, even high dense input point clouds may be simplified to prohibitively unevenly distributed point sets. In either case, resampling of the input PCM may be necessary to support any effective further processing. For refinement, content information is still used to make sure a new inserted point with highest information content, and the new inserted point is moved along the surface normal by applying the smoothing operator.

Moenning et al [55] use their Fast Marching farthest point sampling method [56] to represent a new coarse-to-fine point cloud simplification algorithm directly applied on input point cloud models. The algorithm is efficient on time and memory. The result of the algorithm is a model, in which points distributed evenly or uniformly.

Alexa et al [04] uniformly reduce redundancy of a PCM by estimating a point's contribution to *moving least squares* (MLS) representation of the underlying surface. They require that the input model has normal information for each point in the set. This method has two shortages like in [55] and [33]. The point in the result is distributed uniformly without considering the features and there is no guarantee of the absence of insufficiently dense output point sets. For up sampling, they also use MLS from the compact models they get in the simplification stage, so both stages are time-consuming.

Based on [04], in 2003, Alexa et al [45] propose the concept of *Progressive Point Set Surfaces* (PPSS). Given a reference or input point set R, which define a reference surface S_R. The R is reduced by removing points to form a base point set $P_0 < R$, defining a surface that is deferent from S_R. The base point set P_0 is refined by inserting additional pints yielding the set P_1, the refinement operator first inserts points independent of the referent set R. Then the inserted points are displaced so that the difference between the surfaces decreases. This process is repeated to generate a sequence of point set P_i, with increasing size and decreasing difference from the reference surface.

Pauly et al [10] extract features from input point cloud models. Firstly, *k-nearest neighborhoods* are constructed on an input PCM, and *principal component analysis* (PCA) is applied to each neighborhood. They can keep sharp feature very well by examining the variation-scale curve to determine the critical neighborhood size.

Guennebaud et al [02] present a new way to refine an input PCM for rendering. Comparing to *QSplat* [17], when an under-sampling model, their method can produce more smooth result than *QSplat* on the silhouette and in high curvature area. The creativity is that algorithm of interpolating subdivision surface schemes are borrowed for up sampling. Firstly, they construct neighborhood by improved *k-nearest neighborhood* algorithm. The improvement is to discard bad points in the neighborhood to construct polygon fan for refinement. Then on the neighborhood, a new inserted point is defined by a polygon. They interpolate all the original points. However, they can only guarantee C^0 smoothness for the global surface because their interpolating scheme is only based on 1-neighborhood.

Nehab and Shilane [57] present an algorithm to simplify an input model. The idea is to voxelize the model and output one sample for each voxel. The sample for each voxel is chosen according to a probability that decays as its distance to the center of the voxel increases. Finally, a minimum distance between two points is defined to remove some points, which are too close to others. Because there is only one sample per voxel, this is not suitable to keep details and features for multi-resolution. Therefore, this method is only proper for rending models which far from viewer.

Bhakar et al [16] introduce an efficient way to rendering PCMs, randomly resampling according to feature information and silhouettes. Firstly, they apply octree onto the input PCM in order to voxelize the PCM. For each voxel, *Principal Component Analysis* is applied for feature detection. Feature of local surface in that voxel decides the number of points needed for rendering. The greater the first eigenvalue is, the more points should be kept in that voxel. If a voxel is fallen into silhouette, more points also should be kept. The samples are randomly selected from the original model.

In this important paper, Pauly et al [03] introduce, analyze and compare a number of surface simplification methods for point cloud models. First of all, they also use *Covariance Analysis* (or *Principal Component Analysis*) and *moving least squares* (MLS) to estimate features (or geometric properties) on the input model. For clustering simplification, a 3D BSP tree is created to split the input models into parts from the model's centroid. Then the leaves in BSP tree are down sampled according to feature information. Also, they transplant an iterative mesh simplification algorithm—*quadric error metrics* (QEM), to *k-nearest neighborhood* on PCMs. QEM works by contracting edges in original mesh. For the particle simulation method, they use density to control the number of samples; for lower sampled density, more samples are placed.

Chapter 3
Octree and Related Algorithms

3.1 Introduction

An octree is a hierarchy tree data-structure based on a cell with eight children. Each cell of an octree represents a cube in physical space. Each child represents one octant of its parent. For different purpose, the data structure and algorithm to implement the octree will vary.

3.2 Definitions and Data Structure

Several things should be defined to build the octree: the root node (or cell), which usually is a cubic box, and the scheme to subdivide a node into eight. Here, some terms have to be made clear.

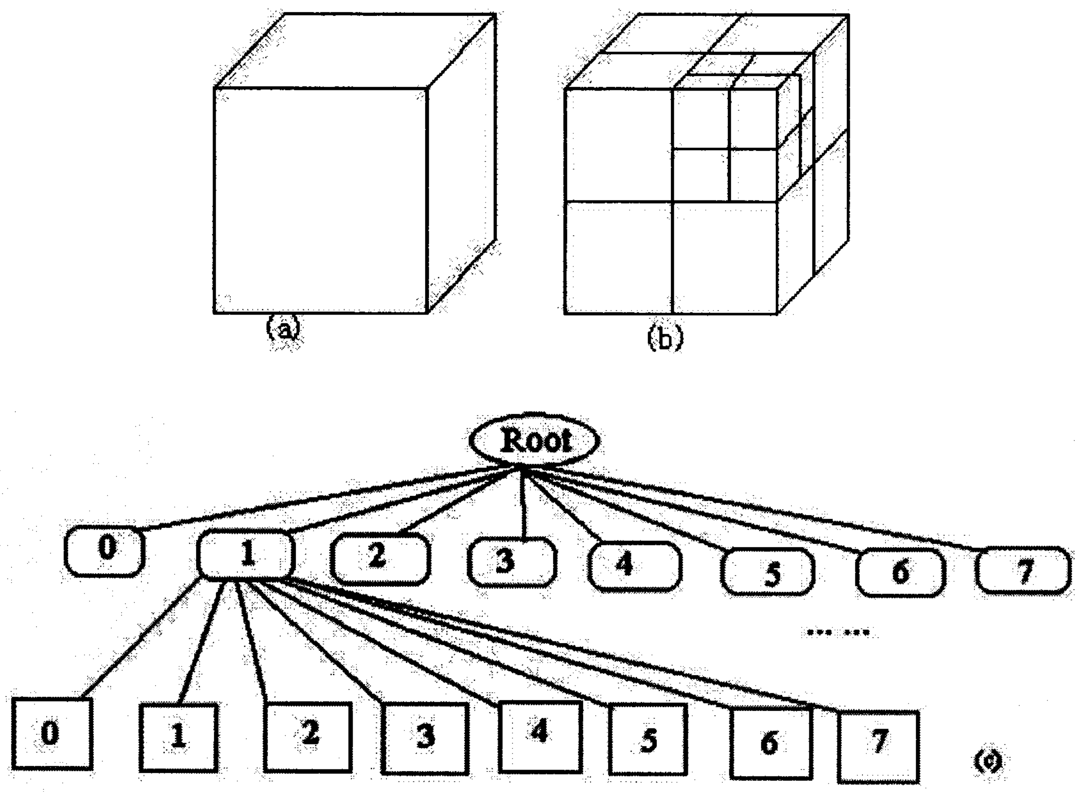


Figure 5: (a) A root node; (b) At the first step, it is divided into 8 children, and at the second step is divided into 8 further; (c) Expressed in tree-like structure of (b)

Root Node[2] is the original box when an octree is built. All the child nodes in the octree are in the root node.

Child nodes are defined by subdividing the parent node into eight according to certain scheme.

[2] Node is also called **octant** in some text.

Leaf node is the node, which has no child node.

Centriod is used to define the center of the node.

Node Size is used to define the size of the node.

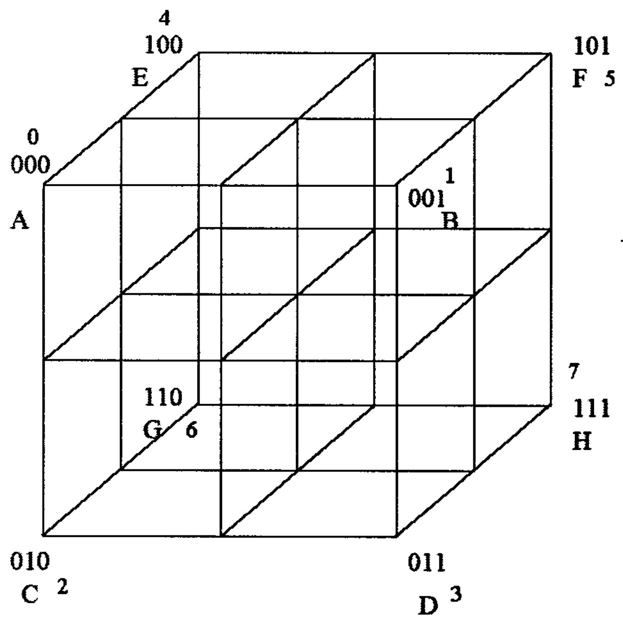
3.2.1 Octree for PCMs

We have to know, in advance, how the point cloud models are managed and manipulated before we focus on octree algorithm. In this research, the point cloud model we are using is very simple, only a set of points with 3D coordinates. Therefore, a *point list* is enough to hold such a PCM. For our purposes, we add more properties to this vector. Firstly, we have to know the maximum and minimum coordinate value in X, Y, and Z directions respectively. This is the critical information to construct the cubic box of octree root.

When a point cloud of object is loaded and a *point list* is constructed, this point list will be attached to the root node of an octree. The cubic box of the root node is defined by the maximum span among X, Y, and Z direction. When a node is subdivided, the point list attached to the node is also subdivided into eight sub lists according to the eight new bounding boxes.

3.2.2 Index Code for Nodes and Directions

For the convenience of algorithms, we give each node a number in decimal and binary. According to the node number in binary, we can also define the eight directions in an octree. This definition is based on a parent node and its eight children node at a certain level.



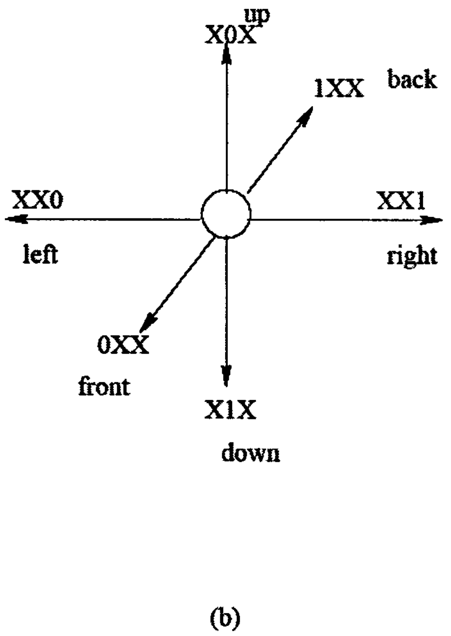


Figure 6: Node number and directions (a) Node number in decimal and binary (b) Direction definition according to node number in binary

3.3 Subdividing a Node

Ideally, a parent-level octree node has eight children, as shown Figure 6. However, it is not valid in an application of PCMs. We have several rules to subdivide octree node for point cloud model.

Point Number Criterion: If the point number in a node is near to the criteria, this node is considered as leaf-node, and will not be subdivide any more. In our experiments, point number criterion is 20.

Bounding Box: The bounding box (cubic box) of a node is divided into eight sub bounding boxes, which together occupy the volume space defined by parent node and share with the same size.

Sub-bounding Boxes: The sub bounding boxes define the boundary to subdivide the point list in the parent-level node.

Discarded Node: After obtaining the eight sub-bounding boxes, and we can use them to build point list for each corresponding child node, if the point list of a child node is empty, this child node will be discard.

According to these rules, it is apparent now that after several steps of subdivision, the hierarchy of octree will get fit to the shape of the given *point cloud model.* Furthermore, the point set in each leaf node defines a local surface patch. If the leaf node is small enough, we can ensure that the surface property in that node is simplex instead of complex. We will have more analysis in the next chapter.

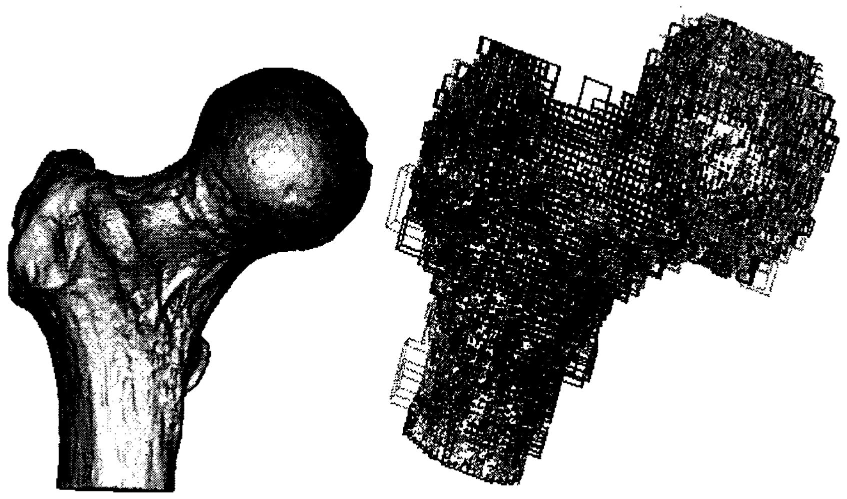


Figure 7: The picture of Jointball model and its octree hierarchy

3.4 Transversal of Octree with Action Parameter

In the hierarchy of an octree, very often, we are required to access each leaf node; so transversal algorithm is critical. Transversal algorithm itself is a simple recursive loop. However, for each recursion, we may have different purpose, so a parameter of action should be given for corresponding computation when recursion loop arrive a target node.

3.5 Neighborhood of Each Leaf Node

For a leaf node, it has at most 26 direct neighbors: 6 face neighbors, 12 edge neighbors, and 8 vertex neighbors. However, considering the local surface of the PCM, vertex neighbors are meaningless to us. Therefore, only face neighbors and edge neighbors are considered to construct leaf node neighborhood.

3.5.1 Find Face Neighbors for a Leaf Node

Each node of an octree has either at most eight or no children. Each of the children represents one of the nodes of the 3D objects. A 3-bit binary vector (as shown in **Figure 6(a)**) index is defined for each of the octants. When a bounding box is subdivided, its sub-bounding boxes are also indexed the same with a 3-bit vector and this three bits are appended to the index of the parent node. Each bounding box has six faces, hence has at most six neighboring blocks, which share a face with it. Each of these six directions can be represented by a 3-bit string D 2 (0XX; 1XX; X0X; X1X; XX0; XX1) as shown in **Figure 6(b)**. Direction vectors 0XX, 1XX, X0X, X1X, XX0, and XX1 represent directions front, back, up, down, left and right respectively.

3.5.2 Definition of Index of a Node

Let $a_1a_2a_3...a_m$ be the index of an octant P found by the above indexing method where $a_i \in (000;001;010;011;101;110;111)$ for $1 \le i \le m$. In this section, we consider the case of face neighbors. For example the node *A* (000) in **Figure 6(a)** has a face shared with node *B* (001) and it is in right (XX1) direction, hence node *A* has a neighbor node present in the same block (the parent node, the same in the following) as *A* is. Now consider for the left (XX0) direction, there is no node in that direction for the node *A* in the same block. We can define *the octree face neighbor presence table* as shown in **Table 3**, where a True entry indicates absence of the neighbor in the same block. We can define a function $\eta(a_i,D)$ from the table. $\eta(a_i,D)$ is False if there is a face neighbor of node $a_1a_2a_3...a_m$ in the parent block $a_1a_2a_3...a_{m-1}$.

3.5.3 Find Complete Index for the Face Neighbors

Now if the neighbor is present in the same parent block (indexed by $a_1a_2a_3...a_m$ then finding its index is trivial. Nodes in the same block have same index except for the last three bits. We can change a_m of the index to get neighbor index. For example, node A(000) and its neighbors in right (XX1), down (X1X) and back (1XX) directions are in the same octant. Consider node *A*(001), neighbors of this node in right, down, and back directions are 001, 101, and 100 respectively. Consider node *A*(001) from Figure, and neighbors of this node in right down and back directions are 001, 010, and 100 respectively. Similarly we can find the indices of neighbors of other nodes.

If for a node $Q(a_1a_2a_3...a_m)$, the neighbor is present in the same octant, we can find the index of it simply by changing a_m. Neighbor of node *B* in right (XX1) direction is not present in the same block as *B* is. If it is present then it is in the right neighbor of parent block. If we add another block as shown in **Figure 6(a)** to the right side of it then octant 000 of the second block will come adjacent to the octant 001 of first block. Hence we can say that, right neighbor of octant 001 is octant 000 of a different block. Neighbor of a node with a_m = 001in right (XX1) direction is in a different octant, if it exists it is the octant 000 of its parent. From **Table 2 and 3** we observer that the neighbor of node with a_m = 001 in XX0 direction is also the same. Hence we can say that left and right neighbors of Q have their last three bits (a_m) same. We can define 24 complete *octree neighbor index table* by copying the entries of partial octree neighbor index (Table 2) table from XX0 column to XX1 column and vice versa. Similarly we copy up (X0X) and front (0XX) entries to down (X1X) and back (1XX) entries respectively and vice versa. We define a function from the following table. Where $n_i = \varphi(a_i, D)$ and n_i is the corresponding binary bits of the neighbor in direction *D*. Function η is used to check the presence of neighbor in same block as the node is.

η	0XX	1XX	X0X	X1X	XX0	XX1
000	100	100	010	010	001	001
001	101	101	011	011	000	000
010	110	110	000	000	011	011
011	111	111	001	001	010	010
100	000	000	110	110	101	101
101	001	001	111	111	100	100
110	010	010	100	100	111	111
111	011	011	101	101	101	101

Table 2: Complete Octree Face Neighbor Index Table

a_m D	0XX	1XX 0	X0X	X1X	XX0	XX1
000	True	False	True	False	True	False
001	True	False	True	False	False	True
010	True	False	False	True	True	False
011	True	False	False	True	False	True
100	False	True	True	False	True	False
101	False	True	True	False	False	True
110	False	True	False	True	True	False
111	False	True	False	True	False	True

Table 3: Octree Face Neighbor Presence Table

3.5.4 Find Edge Neighbors for a Leaf Node

Also, we have to define the edge directions to express the edges in an octant, as shown in Figure 8. For example, the edge with the direction 00X, means that the two end points share the same binary index bits "00-".

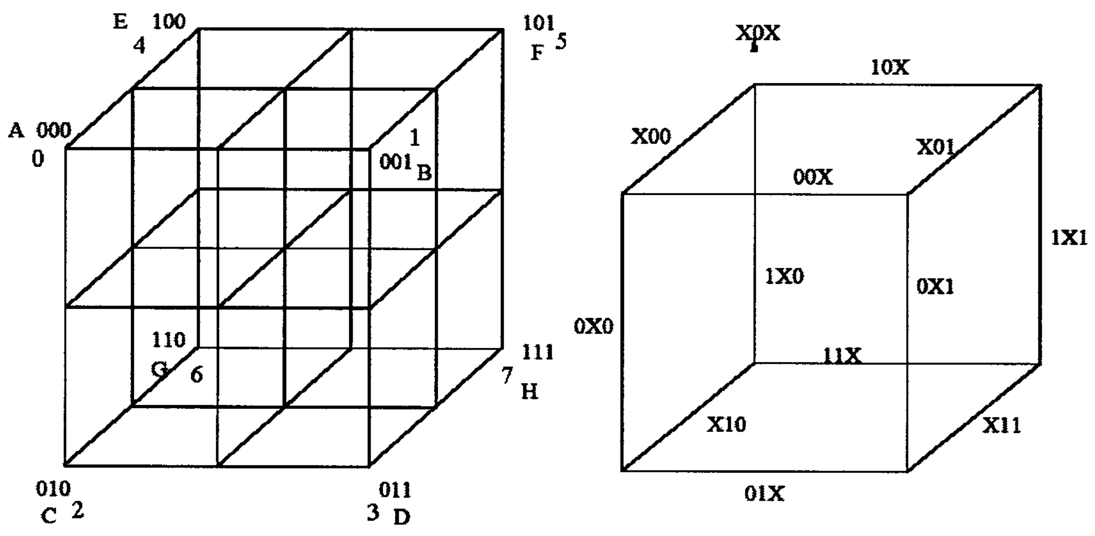


Figure 8: Definition of edge directions

Let $a_1a_2a_3..a_m$ be the index of a node found by the indexing method discussed, where $a_i \in (000,001,010,011,101,110,111)$ for $1 \le i \le m$. The 12 edges of a bounding box are indexed as shown in the **Figure 8**. Here we present an algorithm for finding the edge neighbor of an octree node. The basic approach is same as earlier methods. We first try to find the index of the neighbor node from the given index of a node and direction; then we visit the node. Consider a 3D block as shown in **Figure 8**. There are eight octants indexed by 000, 001, 010, 011, 101, 110 and 111. Consider octant *A* (000), it shares a common edge with node *F* (101). From **Figure 8**, we can see that the edge common in node *A* and *F* is indexed 1X1. Hence the neighbor of node A (000) along the edge 1X1 and the node A have same parent, in other words they are the octants of the same 3D block. Similarly node *D* and *G* shares edge X11 and 11X respectively with node *A*. Hence node *F*, *D* and *G* are the edge neighbors of node *A* and they are present in the same 3D block. Similarly for every other octant there are three octants in the same 3D block, which shares an edge. Hence each node has three of its edge neighbors in the same 3D block. These neighbors can be found just by changing the last three bits of the given index.

Now consider neighbors, which are not present in the same block, node A has its neighbor in 0X1 direction in a different block. If we add a new block to the block shown in **Figure 8**, at front side then the node 101 of the newly added block will become the edge neighbor of node A in 0X1 direction. If we attach the block at left then the node 101 of newly added block will be the edge neighbor of node A in 1X0 direction. Here we can see that for node A both of its edge neighbors in 0X1 and 1X0 directions are octant 101 of their parent blocks. Similarly we can show that for any node its edge neighbors in 0X1 and 1X0 directions have the same last three bits. We can see that directions 0X1 and 1X0 are diagonally opposite to each other.

Node A has its neighbor in 10X direction in a different node. If we attach a new block on top of the block shown **in Figure 8**, node 110 of the newly added block will become the edge neighbor on node A in 10X direction. Direction 01X is diagonally opposite to 10X. We can see that node 110 of a block added at the front side is the edge neighbor of node A in 01X direction. It can be shown that for diagonally opposite edge directions, the neighbors have the same last three bits. Edge neighbor of node A in direction 00X is present in a different block. If we add a block to the existing block along 0XX edge direction then node 110 of the newly added block becomes the neighbor of A in 00X direction. Similarly we can find the neighbors of A in 0X0, X00 directions and diagonally opposite directions. We can find the neighbor index for all of the nodes as we have done for node A. Following table shows how the bits are changed for getting the corresponding bits of edge neighbor index. We define a function $n_i = \Omega(a_i, D)$ from the table where $n_i \in$ (000; 001; 010; 011; 101; 110; 111). If a_i is a bit trio of the index of given node, n_i is the corresponding bit trio of neighbor node. For diagonally opposite directions, the exactly same entry can be found in the table for reasons discussed above.

Ω	X00	X01	X10	X11	0X0	0X1	1X0	1X1	00X	01X	10X	11X
000	011	011	011	011	101	101	101	101	110	110	110	110
001	010	010	010	010	100	100	100	100	111	111	111	111
010	001	001	001	001	111	111	111	111	100	100	100	100
011	000	000	000	000	110	110	110	101	101	101	101	101
100	111	111	111	111	001	001	001	001	010	010	010	010
101	110	110	110	110	000	000	000	000	011	011	011	011
110	101	101	101	101	011	011	011	011	000	000	000	000
111	100	100	100	100	010	010	010	010	001	001	001	001

Table 4: Octree Edge Neighbor Index Table

As there are 12 edges of each octant, there are at most 12 edge neighbors for each node. As shown above only three of them are in the same 3D block hence other nine are in different blocks.

Again we consider node A, edge neighbor of this node along edge 10X is not in the block as it is. If it exists it is in a block above the block containing A. Similarly for direction X01 edge neighbor of node A is in a block above the block containing A. The index of these neighbor nodes can be found by first changing the least significant three bits a_m by $\Omega(a_m, D)$. Now we consider the rest of the bits $a_1a_2a_3...a_m$.

Now consider the case of node 000, the neighbor in direction 00X is not present in the same block. It is not even present in a block, which is a face neighbor of the parent block. The block containing the neighbor is present in a block, which is again an edge neighbor of the parent block in the same direction. In this case we change a_m, and continue with finding the edge neighbor of the parent block $a_1a_2a_3...a_m$ in the same direction.

So here we have three different cases first one is when the neighbor is present in the same 3D block. Second one is when the neighbor node is present in a block, which is a face neighbor of the parent block of the given octant. The third case is when the neighbor is present in a block, which is the edge neighbor of the parent block of the given octant. We define a table, which indicates what action should be taken for finding the edge neighbor index. For the first case we put an entry '***S***' which indicates the neighbor is present in the same block as the given octant. We put the direction of the face neighbor for the second case. For the third case we put '***C***'. We define a function $\Theta(a_i, D)$, which gives the entries of the table.

Θ	X00	X01	X10	X11	0X0	0X1	1X0	1X1	00X	01X	10X	11X
000	C	X0X	XX0	S	C	0XX	XX0	S	C	0XX	X0X	S
001	X0X	C	S	XX1	0XX	C	S	XX1	C	0XX	X0X	S
010	XX0	S	C	X1X	C	0XX	XX0	S	0XX	C	S	X1X
011	S	XX1	X1X	C	0XX	C	S	XX1	0XX	C	S	X1X
100	C	X0X	XX0	S	XX0	S	C	1XX	X0X	S	C	1XX
101	X0X	C	S	XX1	S	XX1	1XX	C	X0X	S	C	1XX
110	XX0	S	C	XX1	XX0	S	C	1XX	S	X1X	1XX	C
111	S	XX1	X1X	C	S	XX1	1XX	C	S	X1X	1XX	C

Table 5: Octree Edge Neighbor Action Table

3.6 Chapter Summary

In this chapter, we have presented the octree and related algorithms, which is used in experiments. Related definitions are given at the beginning: octree, index, and all kinds of nodes. Then, subdivision and transversal algorithm are discussed. Finally, the algorithm about how to construct neighborhood for a leaf node is explained. Data structures and algorithms expressed in this chapter are essential for our experiments.

Chapter 4
Process of Simplification

4.1 Overview of the Idea

As mentioned in chapter 3, we use the octree to divide a surface in an input PCM into small patches. These small patches represent local surfaces of the PCM. As long as the input PCM is dense enough, and the size of leaf node in the octree is small enough, the point set in a leaf node can describe a simplex feature on the surface.

The process of simplification can be expressed in several stages. At the beginning, the input PCM is subdivided in the octree. On each leaf node of octree, *Principal Component Analysis* (PCA) on an each leaf node is applied for constructing an eigen-system. Based on the eigen-system in leaf node, feature for the local surface can be detected. According to different features, we use different algorithms for simplification.

4.2 Features on Surfaces

Traditionally, when we describe a local area on a surface, we put it into one of the following categories: ***flat, crease*** or ***edge, corner, boundary*** [06]. From another view, a complex surface can be considered as a compound of flat patches. In the interior area of each patch, it is flat area; along the joint of two flat areas, it is the crease or edge; and when two or more creases bump into each other, a corner is formed. Boundary is to describe non-closeness of a model. In this thesis, all the models we used are closed. Theoretically, if we only focus on a local area of a surface, which are small enough, we can ensure that the feature in such area is simplex, i.e., the feature in such area is flat, crease, or corner, instead of a compound of them.

For a flat area, if it is ideally flat, it is enough to represent it by a plane defined by one point with the normal; and if it has a little curve there, one or two more points should be add for it (see Figure 9). Keeping more points than that is meaningless for simplification.

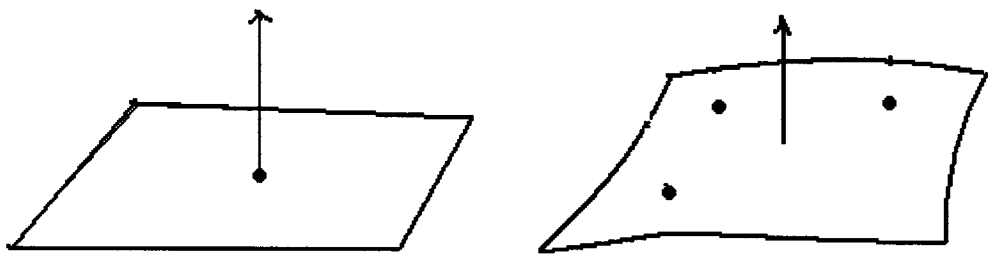


Figure 9: Ideally flat area and the flat area with a little curvature

For a crease area, as mentioned above, a simplex crease can be decomposed as two semi-planes merging together along the crease direction. Therefore, we can use two planes to define the simplex crease, so 6 points are enough to express the crease and more compactly, 5 or even 4 points (sharing 2 points between 2 planes).

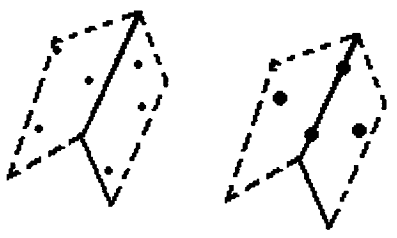


Figure 10: Use 6, 5, or 4 points to define a crease

As for a corner area, it is like a pyramid, so we can just use a tetrahedron to define it simply without considering how many creases meet there; and one vertex of the tetrahedron is the top point of the corner.

4.3 Principal Component Analysis (PCA)

The basic idea of PCA is to describe the variation of a set of multivariate data in terms of uncorrelated (linearly independent) variables, each of which is a particular linear combination of the original variables. The new variables are derived in decreasing order of importance, so that, for example, the first principal component accounts for as much as possible of the variation in the original data. The objective of this analysis is usually to see whether the first few components account for most of the variation in the data. If so, it is argued that they can be used to summarize the data with little loss of information, thus providing a reduction in the dimensionality of the data, which may be useful in simplify later analysis. PCA summarizes the variation in a correlated multi-property into a set of uncorrelated components, each of which is a particular linear combination of the original Variables. *Principal Component Analysis* (PCA) involved a mathematical procedure that transforms a number of correlated variables into a number of uncorrelated variables called *principal components*. The first principal component accounts for as much of the variability in the data as possible, and each succeeding components accounts for as much of the remaining variability as possible. Mathematically, PCA is called *eigen-analysis*. The eigenvector corresponding to the largest eigenvalue is the direction of the first principal component.

4.3.1 Data Set

In our experiments, the data set of input is a cluster of points $P = \{p_0(x_0, y_0, z_0)......p_n(x_n, y_n, z_n)\}$ contained in a leaf node of the octree.

4.3.2 Subtract Mean

For PCA to work properly, you have to subtract the mean from each of the data dimensions. The mean subtracted is the average across each dimension. So, all the *x* values have (the mean of the x values of all the data points) subtracted, and all the y values have subtracted from them, and all the z values have subtracted from them. This produces a data set whose mean is zero. As a result, we get the mean vector $[\overline{x}, \overline{y}, \overline{z}]^T$

4.3.3 Calculate Covariance Value

Covariance is such a measure. Covariance is always measured between 2 dimensions. Therefore, given a 3-dimensional data set (*x, y, z*), we could measure the covariance between the *x* and *y* dimensions, the *x* and *z* dimensions, and the *y* and *z* dimensions. Measuring the covariance between *x* and *x*, or *y* and *y*, or *z* and *z* would give us the variance of the *x*, *y* and *z* dimensions respectively. The formula to calculate covariance can be defined as,

$$\operatorname{cov}(x, y) = \frac{\sum_{i=0}^{n}[(x_i - \overline{x})(y_i - \overline{y})]}{n}$$

$$\operatorname{cov}(x, z) = \frac{\sum_{i=0}^{n}[(x_i - \overline{x})(z_i - \overline{z})]}{n}$$

$$\mathrm{cov}(y,z) = \frac{\sum_{i=0}^{n}[(y_i - \overline{y})(z_i - \overline{z})]}{n}$$

In addition, $\mathrm{cov}(y,z) = \mathrm{cov}(z,y)$, $\mathrm{cov}(y,x) = \mathrm{cov}(x,y)$, and $\mathrm{cov}(x,z) = \mathrm{cov}(z,x)$

4.3.4 Calculate Covariance Matrix

In 3D, the covariance matrix consists of covariance values between any two dimensions, so we have the following definition of covariance matrix,

$$C = \begin{pmatrix} \mathrm{cov}(x,x) & \mathrm{cov}(x,y) & \mathrm{cov}(x,z) \\ \mathrm{cov}(y,x) & \mathrm{cov}(y,y) & \mathrm{cov}(y,z) \\ \mathrm{cov}(z,x) & \mathrm{cov}(z,y) & \mathrm{cov}(z,z) \end{pmatrix}$$

4.3.5 Calculate Eigenvectors and Eigenvalues

In popular way used in most texts, we denote the eigenvalues as $\lambda_0, \lambda_1, \lambda_2$, where $\lambda_0 \le \lambda_1 \le \lambda_2$ and eigenvectors as e_0, e_1, e_2, where e_0, e_1 and e_2 are corresponding to λ_0, λ_1 and λ_2 respectively. Eigenvalues are used to measure the amount of the variation described by each *principal component* (PC).Eigenvectors provide the vectors for the uncorrelated PC. Eigen-System consists of $\{\lambda_0, \lambda_1, \lambda_2\}$, and $\{e_0, e_1, e_2\}$. Here, for convenience of algorithms, all the eigenvalues and eigenvectors are normalized. That means the mode of the eigenvectors is 1. For eigenvalues, we have formula to calculate new eigenvalues $\lambda_0 = \frac{\lambda_0}{\lambda_0 + \lambda_1 + \lambda_2}$, $\lambda_1 = \frac{\lambda_1}{\lambda_0 + \lambda_1 + \lambda_2}$

and $\lambda_2 = \dfrac{\lambda_2}{\lambda_0 + \lambda_1 + \lambda_2}$. In the following, when elements in an eigen-system are mentioned, they refer to normalized ones.

Because the covariance matrix is symmetric, we can use Jacobi Transformation [05] to find eigenvalues and eigenvectors for such a matrix.

4.4 Algorithms of Simplification

4.4.1 How Many Points in Each Leaf Node

Our purpose of using the octree is to voxelize a *point cloud model*, and in each leaf node (voxel), *the feature of local surface should be simplex instead of complex*. If keeping too many points in a leaf node, it is not to a certainty that the local surface in a leaf node is simplex. On the other hand, if the number of points is not enough, the principal component analysis will get meaningless, because it based on statistic theory. Based on the experiments, we found that keeping around 20 points is suitable for most of models.

However, it is inevitable, in the final octree, that there are a few *special leaf nodes*, which contain points much fewer than 20, and are the necessary result of octree subdivision. If the point number in a leaf node is fewer than 13, the eigen-analysis is getting meaningless or collapsing. We also have algorithm for such areas in the following.

4.4.2 Penalty Functions for Detecting Features

After applying PCA to each leaf node of the octree, we get an eigen-system for each leaf node. The eigen-system consists of a three eigenvalues $\{\lambda_0, \lambda_1, \lambda_2\}$, $0 < \lambda_0 < \lambda_1 < \lambda_2 < 1$ and three eigenvectors $\{e_0, e_1, e_2\}$, where e_0, e_1 and e_2 are corresponding to λ_0, λ_1 and λ_2 respectively. Through experiments, we establish a system to detect simplex feature in a leaf node.

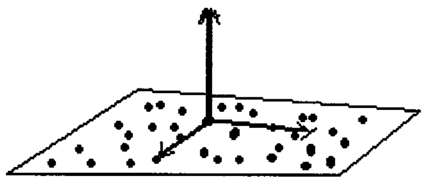
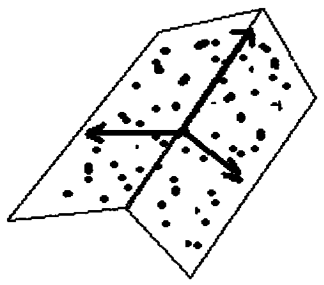
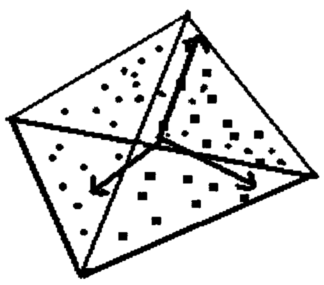
Features	Eigenvalues	Eigenvectors	Picture for Illustration
Flat Area	$\lambda_0 \le 0.09$	e_0 is the normal vector	
Crease Area	$\lambda_2 \ge 0.40$, $\frac{\lambda_0}{\lambda_1} \ge 0.25$	e_2 is the direction of crease	
Corner	$\lambda_0 \ge 0.27$	One of eigenvector is the direction of the corner	

Table 6: Feature penalty based on eigen-system

4.4.3 Simplification of Flat Areas

As mentioned in Table 6, for a leaf node, if the first eigenvalue is less than 0.09, it will be classified as flat area. If the first eigenvalue is extremely small, i.e. near zero, we can say the leaf node is extremely flat, so in this area, only point is enough to express the local surface. However, if the eigenvalue is near 0.09, it means that the local surface in this area is not very flat. To keep such feature, we'd better to keep more points in such area. In order to form the compact model, in experiments, *if the first eigenvalue is greater than 0.03, we keep two points in this flat area*; and *if first eigenvalue is greater than 0.075, three points are kept.*

The normal vector is set the same as the first eigenvector. The flat feature coded as 0 is also attached to each points in compact model.

4.4.4 Simplification of Crease Areas

Just as mentioned in 4.2, a crease is defined as the merge of two smooth areas. For a local surface area with simplex feature, we can simplify it as merging of two planes with C^0 continuity along the crease. Therefore, we can subdivide point set a crease node into two sets, which are held by two planes respectively.

Firstly, from the point set in the crease leaf node, we select 3 points v_1, v_2, v_3 randomly to form a triangle to define plane ***S***. The interior angles of the triangle should greater than 30^o. Then, nearest to the three selected points, we select three more points to verify if this plane defines the half of the crease. If the three later selected points are contained in the plane, we can ensure that this plane is good, otherwise, the plane ***S*** will be discarded and we have to find a new plane just as at the beginning.

After the plane is decided, we can match each point left. If it is contained in the plane, this point is on the half of the crease defined by the plane; otherwise, it is in another half of the crease.

After the point set is classified into two ***P₁*** (containing v_1, v_2, v_3) and ***P₂***, we keep v_1, v_2, v_3 in the compact model. Also, three new points v_4, v_5, v_6 are selected randomly from ***P₂*** to represent another half of the crease. If the interior angles of triangle defined by v_4, v_5, v_6 are greater than 30°, we keep them the compact model, otherwise, we have to discard them and reselect three new points until the criterion are satisfied.

4.4.5 Simplification of Corner Areas

As we mentioned in Table 6, there is one of three eigenvectors, which represent the direction of the corner. By observation, we find that on the reverse direction of the corner, there is no point, as shown in the grey area of Figure 9.

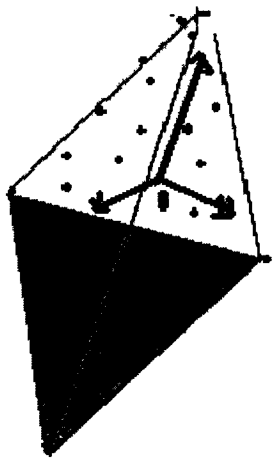


Figure 11: A corner leaf node with eigen-system

After we find the direction of the corner, supposed e_c, using the coordinate system defined by three eigenvectors and the center of leaf node, we get new positions in such system for each point, and e_c is the positive direction of y-axis. We set the *top point* of the corner as the average of three points with maximum values along the e_c. The purpose of choosing average is to cut down the effect of a possible noisy point.

In the new coordinate system, we also find 7 points, which has minimum y-coordinate value. From the 7 points, we choose three to form a triangle, which defines a plane. The interior angles of the triangle should no less than 30°, and the angle between the normal of the plane and e_c should be no greater than 45°. In experiments, we can always find such triangle for the corner leaf node. The three vertices of the triangle and the top point of the corner is the result of simplification, i.e., we keep 4 points for a corner leaf node. We mark the top point as corner-feature and other three as crease-feature.

4.4.6 Deal with of Special Leaf Nodes

For a special area, we use different strategy according to its neighborhood condition. If it is the neighbor of the flat area, we just chose one point from it for compact model. If it is the neighbor of a crease or corner, we merge it to the crease or corner area.

4.4.7 Set Normal Vectors for Compact Model

Information of normal vectors is critical to track the shape and orientation of surfaces. For compact model, it is more important because in some area, the density of points is less than the size of the model, so having normal vectors seems to be the only efficient way to orientate the surface.

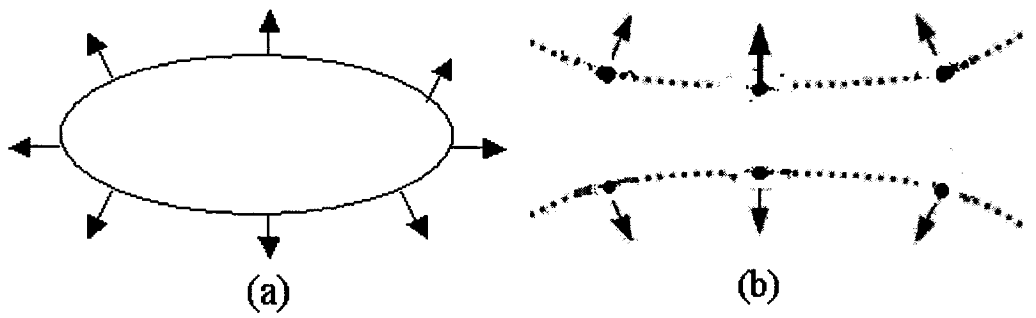


Figure 12: (a) Normal vector tells us the inside and outside of the model (b) In some areas of compact models, normal vector has to be used to separate disjoint parts.

Set Normal for Flat Areas: Just as shown in Table 6, the normal vector for flat area share the same or reverse direction of the first eigenvector.

Set Normal for Crease Areas: We have mentioned that a simplex crease is dealt with as merging of two planes, which can be defined by our algorithm. Therefore, normal vectors of crease-featured points in compact model are the normal (or just with reverse direction) of the plane they belong to.

Set Normal for Corner Areas: For a corner leaf node, one top-point and 3 other points are kept. Apparently, the normal of the top point can be set as corner direction. For other three points, we don't set normal. However, we have enough information to track the shape and orientation of the surface because all neighbors of corner area have normal vectors. When we want to Starting from the leaf node with the normal near to the positive *z*-axis, we can construct octree-leaf-node neighborhood.

Set the first exact normal vector. In the octree, it is trivial to find the leaf node with maximum z-coordinate value. Apparently, the normal for this leaf node is near or the same with positive *z*-axis. For such leaf node, we set one of the eigenvector as the normal, which has the smallest angle with positive *z*-axis direction among the three eigenvectors.

For flat areas, currently the exact direction of the normal vector cannot be decided, and the same problem exists for crease areas. We can use octree-leaf-node neighborhood algorithm to resolve this problem. In the whole octree, we have set the first normal vector for a leaf node with extreme *z*- coordinate. As shown in Figure 13, the right direction of normal should have a smaller angle with the known one. In experiments, we use dot product to evaluate the angle.

After setting normal the first neighborhood, we can construct neighborhood for each leaf node in this neighborhood, and set normal direction. This is a recursive process. Until we cannot find a leaf node with uncertain normal direction, the recursion will continue.

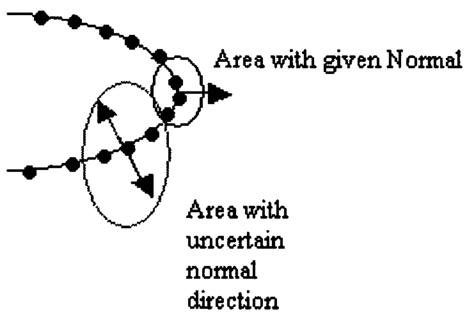


Figure 13: Selection of normal direction for points in ellipse-circled area. For C^1- continuity area, the right direction forms a smaller angle between two normal vectors.

The normal information may not be very accurate, but it is enough for us to orientate operations on compact models.

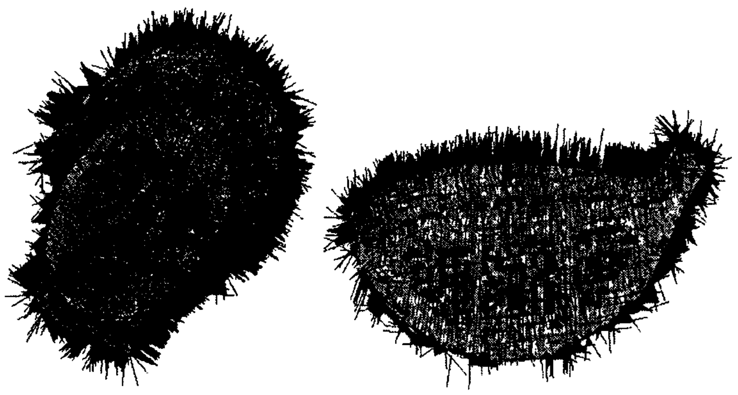


Figure 14: Venus and Club models after setting normal vectors. Normal vectors shown longer at crease and corner areas

4.5 Experiment Results

4.5.1 Simplification According to Compact Rate

As mentioned in theoretical parts, *typically one point is kept for a flat leaf node, 4-6 points for a crease leaf one, and 4 for a corner one.* In order to implement compact more flexible, several strategies are used. For low compact rate, i. e., given compact is greater than 50%, we usually, only simplify flat leaf node, and leave other leaf node unchanged. If compact rate is more than 20% and less than 50%, we used typical way to do simplification. If compact rate is less than 20%, simplification mainly depends on merging simplified flat leaf node together, and keeping 4 points in a crease leaf node. For even higher compact rate, we have to keep one point in a corner, and merge flat areas with crease and corner area, which are the neighbors of flat areas.

In addition, dynamic simplification is implemented because compact rate has to be approximated as accurately as possible. In above, we just implement roughly according to experiences from experiments. Moreover, different PCMs have different features on their surfaces. For examples, the model of CLUB has more flat areas than others, and VENUS has more slim creases on her hair. Therefore, dynamic simplification is necessary to implement accurate compact rate. Usually, we use different strategies according the compact rate domain set above. After simplification, if the compact rate is much higher than the given one, that means more points should be kept in simplified areas, and simplification algorithms will be applied on original PCMs again with keeping one more point for each simplified leaf node. After around 100 leaf nodes are simplified, compact rate will be checked again. If compact rate is much less than given one, we do reverse.

4.5.2 Experiment Results

Model Name	Flat Nodes	Crease Nodes	Corner Nodes	Special Nodes	Total Leaf Nodes	Running Time3
Rabbit	3482	1861	2	211	5556	0: 0: 11: 262
Igea Artifact (Venus)	7275	3657	4	451	11387	0: 0: 23: 473
Dinosaur Sculpture	934	2747	3	141	3825	0: 0: 19: 474
Ball Joint	3717	4079	5	239	8040	0: 0: 30: 180
Golf Club	12393	2084	5	600	15082	0: 0: 49: 565
Isis	8618	3626	4	356	12604	0: 0: 35: 180
Hip Bone	19132	15047	19	1279	35477	0: 1: 36: 180
RocketArm	1582	1067	2	104	2755	0: 0: 7: 436

Table 7: Feature Statistic from PCA

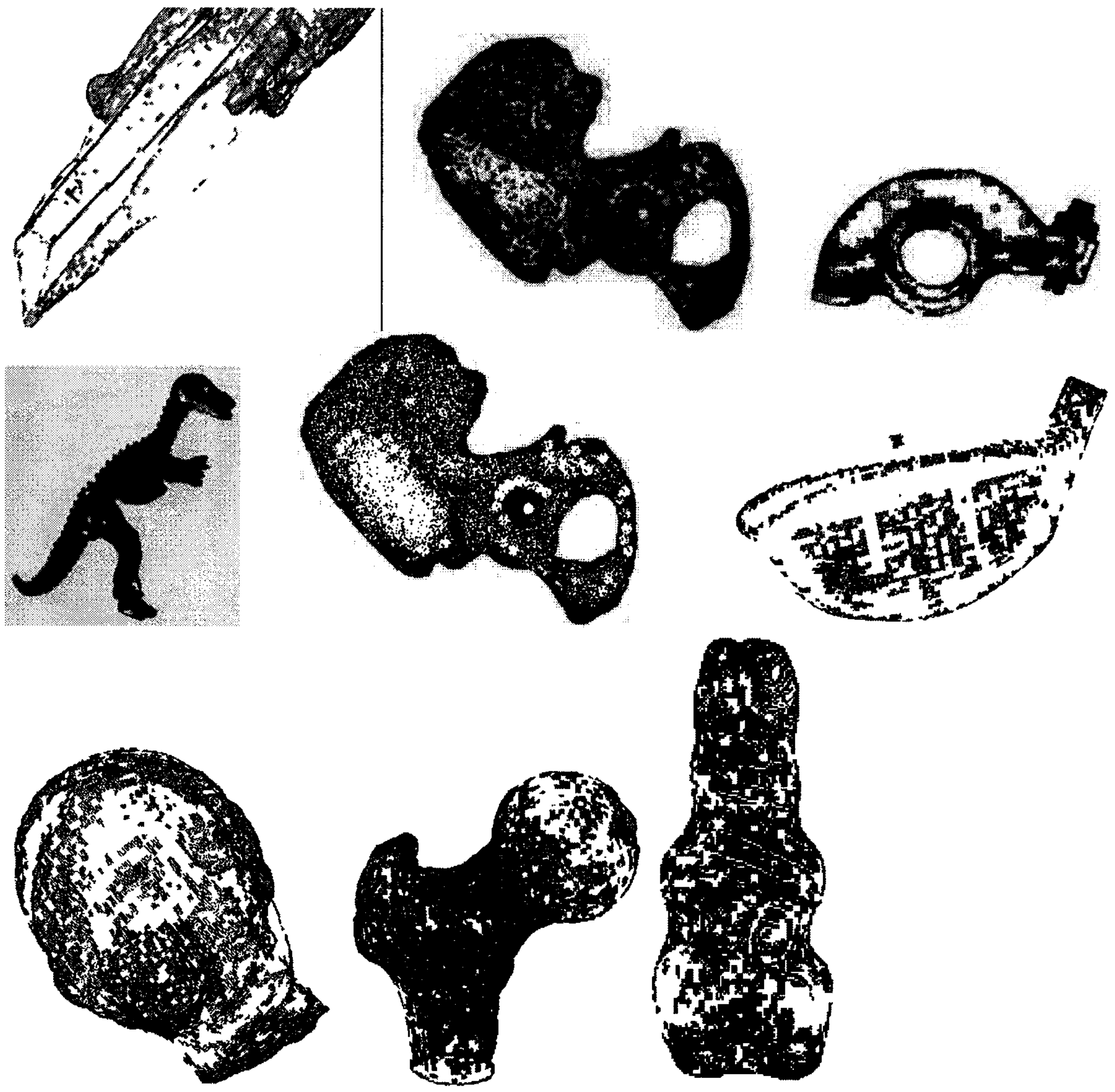
[3] All algorithms are tested on PC with P4 1.6 CPU, and 256M Memory. Running Time Formate, hours: minutes: Seconds: MilliSecond.

PCA on most PCMs work well, except on Dinosaur and Rabbit. From last **Table 7**, we can find that both of them only have several thousand leaf nodes, even they have the real size greater than Club and Jointball. In addition, for Dinosaur, from the picture, we can know that there are tremendous creases on this sculpture models, ribs, head bones, toes, so more points should be sampled from scanning for this model. Comparing to Dinosaur model, Club model was sampled densely during scanning, while it has smaller size, so we can capture very slim features (**Figure 15**).

Model Name	**Rabbit**	**Igea Artifact (Venus)**	**Dinosaur Sculpture**	**Ball Joint**
Given Rate 80%	81.55% 0: :0 20: 089	78.52% 0: :0: 40: 452	80.23% 0: :0: 15: 326	80.12% 0: :0: 35: 756
Given Rate 60%	59.33% 0: :0 30: 125	63.12% 0: :0: 45: 162	62.13% 0: :0: 17: 274	58.96% 0: :0: 39: 879
Given Rate 40%	42.56% 0: :0 37: 452	41.23% 0: :0: 47: 333	35.27% 0: :0: 21: 104	40.25% 0: :0: 42: 274
Given Rate 20%	X	19.85% 0: :0: 52: 745	X	21.35% 0: :0: 47: 352
Given Rate 10%	X	11.79% 0: :0: 59: 102	X	10.23% 0: :0: 50: 221
Total Points	67,038	134,345	56,194	139,306
Model Name	**Golf Club**	**Isis**	**Hip Bone**	**RockerArm**
Given Rate 80%	79.23% 0: 0: 46: 257	83.47% 0: 0: 42: 121	82.57% 0:1: 05: 105	80.12% 0: 0: 4: 221
Given Rate 60%	60.56% 0: 0: 49: 575	57.33% 0: 0: 44: 267	58.54% 0: 1: 20: 288	57.45% 0: 0: 5: 225
Given Rate 40%	38.42% 0: 0: 52: 346	42.12% 0: 0: 47: 512	42.13% 0: 1:56: 237	41.86% 0: 0: 6: 999
Given Rate 20%	22.22% 0: 0: 48: 762	23.75% 0: 0: 49: 108	23.15% 0: 2: 25: 264	45.21% 0: 0: 8: 110
Given Rate 10%	11.76% 0: 1: 05: 277	12.73% 0: 0: 52: 254	11.54% 0: 2: 55: 285	12.33% 0: 0: 9: 117
Total Points	209,779	187,689	530,168	40,178

Table 8: Simplification Results (outputs: compaction rate and consumed time)

For Rabbit and Dinosaur models, higher compact rates were not tested on them, because even at compact rate around 40%, PCA result is difficult to be classified according to **Table 6** already.



According to Sequence: Lower Part of Isis (20%), Hip Bone (60%), RocketArm (40%), Dinosaur (80%), Hip Bone (20%), Golf Club (10%), Venus (15%), Jointball (60%), and Rabbit (60%) Rendered with only OpenGL glVertex3d () command.

Figure 15: Pictures of each PCMs after simplification

4.6 Chapter Summary

In this chapter, the simplification algorithms for PCMs based on octree and PCA is presented in details. Basically, we have obtained the results, which we expect from theoretical analysis. The experiment results on Dinosaur and Rabbit models are very unpleasing because they have a lot of creases and corners on these models, especially on Dinosaur Model, but their density is not high enough according to their dimension and features. Therefore, in the refinement stage, we have no experiment result on PCMs of Dinosaur and Rabbit. In addition, our algorithms did not consider the noise in the PCMs. In short, density is critical for PCA. For example, we can see that the small features on the Club PCMs can be captured because it is dense enough.

Chapter 5
Refinement of Compact Models

5.1 Introduction

In last chapter, we have obtained compact PCM with feature and normal vector tagged on each point. In this chapter, we will try to refine the compact PCM backwards. The density of refined PCM can be assigned in advance. In addition, we can get PCMs as dense as or denser than the original PCMs. The strategies used for refinement compact PCMs are adapted from subdivision surfaces. Pauly et al [54] just pointed out theoretically that subdivision schemes possibly work well to refine PCMs, Guennebaud et al [02] use interpolating subdivision schemes to refine down sampling PCMs for rendering, but interpolating schemes need large stencil for compute new points and if the original models are with noise, interpolating schemes are not good to smooth noise out.

Our schemes are adapted from Loop's [24], which is an approximating schemes working on triangle meshes originally and is very popular in subdivision surfaces. First of all, we should introduce what is subdivision surface, and following that, Loop's schemes will be discussed in details. We also need neighborhood algorithm to select a group of points for subdivision stencil. Finally, programming and the result will be listed.

5.2 Subdivision Surfaces

5.2.1 A Brief Review

In 1974, Chaikin presented 'An algorithm for high speed curve generation' in a speech at Utah University. This algorithm generates a smooth curve that is approximately the shape of a given control polygon. The algorithm works with corner cutting of the control polygon to form a new smoother control polygon. For each clipping of the old of point from control polygon, two new points are found in new control polygon. By repeated application of the corner cutting, the finer control polygon will converge to a smooth curve. Chaikin's rules for corner cutting are simple. On each segment of the control polygon, two new points are found at a distance of 1/4 and 1/3 between the end points. The ratio 1/4 and 1/3 are for each of performing corner cutting on digital computer. In fact, Chaikin's algorithm is the refinement algorithm for quadratic B-splines.

In 1978, Doo and Sabin, and Catmull and Clark published their papers describing their subdivision surface respectively. Doo-Sabin's scheme is the generalization of biquadratic B-spline refinement on arbitrary mesh. Also, it is first time, Doo-Sabin showed the idea of using matrix to analyzing subdivision schemes. This idea is thoroughly exploited by Denis Zorin [18]. Catmull-Clark's scheme is the generalization of uniform bicubic B-spline refinement mentioned above. In 1986, Loop [24] presented his new subdivision scheme in his Master thesis, but the control mesh is triangulated. His scheme is based on linear combination of shifts of a 3-direction box spline.

To that time, it was the first development phase of subdivision surfaces. Until 1995 when Reif [23] gave the analysis of C^1-continuity of subdivision, a lot of questions of continuity at extraordinary vertices are unknown. Since then, there were a lot of new practical schemes appeared, and more work of analysis of subdivision schemes was done.

In 1995, Habib et al. [21], and Peters et al. [22] suggested mid-edge schemes with a little difference respectively. This scheme can be called simplest subdivision scheme, because its mask of refinement is just to find the middle point for each edge.

In 1996, Kobbelt showed his interpolating subdivision scheme, which is the extension of tensor product of four-point scheme, and just named as Kobbelt [25].

In 1990, DYN et al [26] devised a new interpolating scheme—Butterfly scheme, but the final surface is not-C^1-continuous extraordinary points of valance k=3 or k>7. In 1996, Zorin et al [27] gave a modified Butterfly scheme, which guarantees that the scheme produces a C^1-continuous surface for arbitrary mesh.

In 1996, and 1997, two papers [18, 028] analyzing subdivision surfaces appeared. These are some milestone papers for development of subdivision surface because they heavily used math tools to analyze known subdivision schemes. Key features of convergence, smoothness, etc are anatomized thoroughly first time.

Most of time, stationary subdivision schemes can be expressed in matrices, which are a better way to for mathematical analysis. It is not difficult to understand this comparing that we can express B-spline refinement in matrices.

5.2.2 Classification Comparison of Subdivision Surfaces

A lot of subdivision schemes are mentioned above, but only representative ones are listed in the Table.

Scheme	Approximating	Interpolating	Quadrilateral	Triangle	C^1	C^2	Primal	Dual
Midedge	*		*		*			*
Doo-Sabin	*		*		*			*
Catmull-Clark	*		*			*	*	
Loop	*			*		*	*	
Butterfly		*		*	*		*	
Kobbelt		*	*		*		*	

Notes:

1. Approximating and interpolating with respect to the original control mesh point position.
2. Quadrilateral and triangle based on the basic primitive of control mesh.
3. Primal or dual type depending on the splitting rule (faces splitting or edge splitting).
4. Some schemes are difficult to put into a certain category, so they don't appear in the form Classification of Subdivision Schemes

Table 9: Classification for Schemes of Subdivision Surfaces

By observation, we can find that the stencil of approximating subdivision schemes is smaller than interpolating one, because the latter has to make keep tracking on the original points and make out a smooth result (see **Figure 15** and **16**). Generally, approximating schemes only need a *1-neighborhood* stencil, and interpolating ones need a larger neighborhood to cover the stencil.

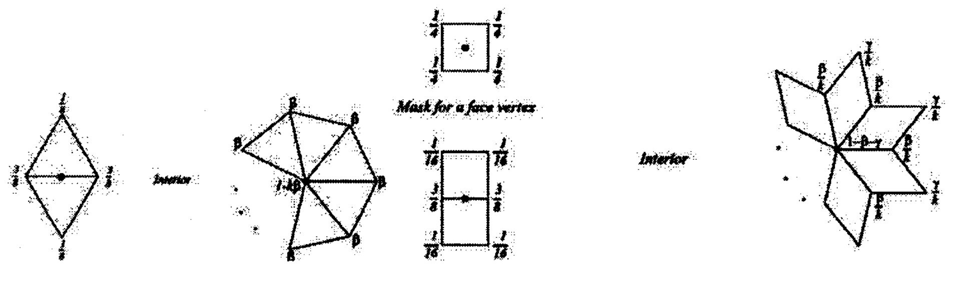


Figure 16: Approximating schemes. Left two: main stencils of Loop Scheme. Rest Three: Main stencils for Catmall-Clark schemes.

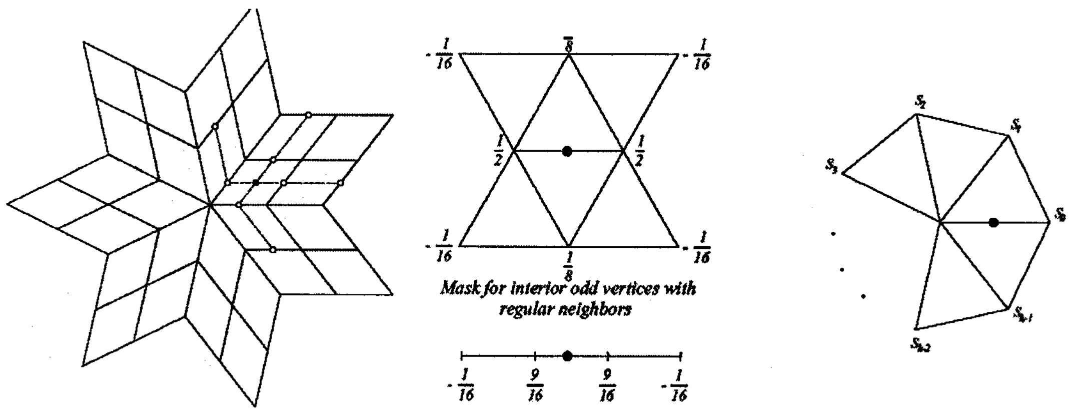

$-\frac{1}{16}$
$\frac{1}{8}$
$-\frac{1}{16}$
$\frac{1}{2}$
$\frac{1}{2}$
$-\frac{1}{16}$
$\frac{1}{8}$
$-\frac{1}{16}$
Mask for interior odd vertices with regular neighbors
$-\frac{1}{16}$ $\frac{9}{16}$ $\frac{9}{16}$ $-\frac{1}{16}$

Figure 17: Interpolating schemes. Left One: stencil of Kobbelt .Right three: Butterfly

Usually, Loop and Catmull-Clark subdivision produce more pleasing surfaces, as these schemes reduce to C^2 splines on a regular mesh. These two schemes appear to the best choices for most applications, which do not require exact interpolation of the initial mesh.

5.2.3 Loop's Subdivision Scheme

The Loop scheme is a simple approximating vertex insertion scheme for triangular meshes proposed by Charles Loop [24]. C^1-continuity of this scheme for valences up to 100, including the boundary case, was proved by Schweitzer [28]. The proof for all valences can be found in [38]. The scheme is based on the three-directional box splines, which produces C^2-continuous surfaces on the regular meshes. The Loop scheme produces surfaces that are C^2-continuous everywhere except at extraordinary vertices, where they are C^1-continuous mesh, for example, by triangulating each polygonal face.

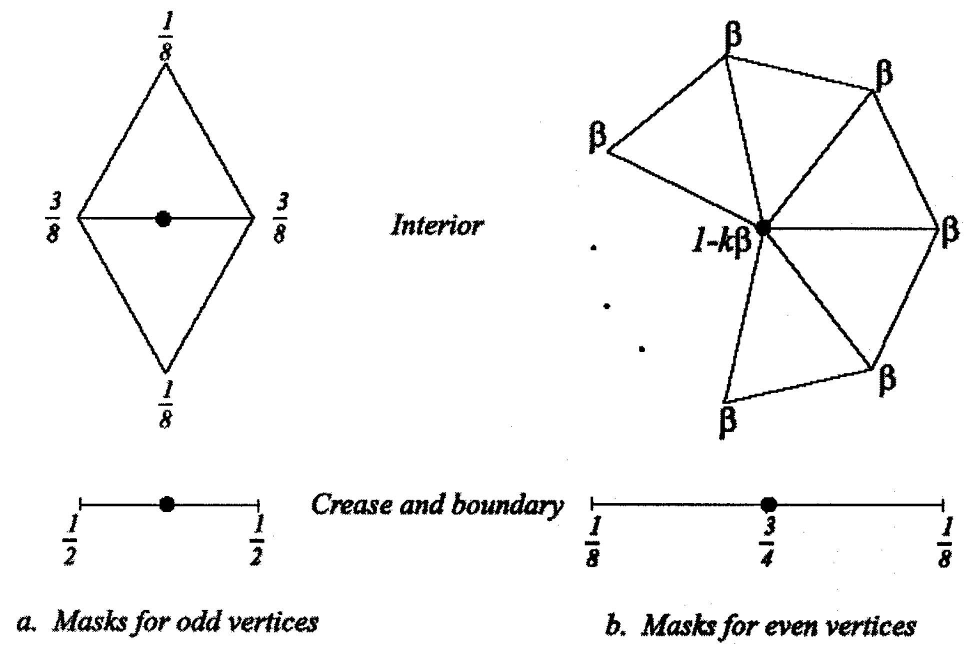


Figure 18: Loop's Subdivision Scheme

Stencils of Loop's scheme are in **Figure 17**. There are two types of stencils. One is for *odd vertices*, which will be inserted newly, and another is even vertices, which have existed already, but we need to calculate the new positions of them. In formula, the scheme can be expressed as,

$P^{j+1} = (p_0 + 3 * p_1 + p_2 + 3 * P_3)/8$ -- for new inserted interior point;

$P^{j+1} = (p_0 + p_1)/2$ --for new inserted crease or boundary point;

$P' = 1 - k * \beta * P + \sum_{i=0}^{k-1} \beta * p_i$ -- for old interior point;

$P' = \frac{3}{4}P + (p_0 + p_1)/8$ --for old crease or edge point.

And, there are two approaches to choose β . One is $\beta = \frac{1}{n}(\frac{5}{8} - (\frac{3}{8} + \frac{1}{4}\cos\frac{2\pi}{n})^2)$ from Loop, and other one is $\beta = \frac{3}{8n}, n > 3$, and $\beta = \frac{3}{16}, n = 3$ from Warren [69].

The rules for computing tangent vectors for the Loop scheme are especially simple. To compute a pair of tangent vectors at an interior vertex, use $t_1 = \sum_{i=0}^{k-1} \cos \frac{2\pi i}{k} p_{i,l}$, and $t_2 = \sum_{i=0}^{k-1} \sin \frac{2\pi i}{k} p_{i,l}$. These formulas can be applied to control points at any subdivision level.

5.2.4 Piecewise Smooth Subdivision Surfaces

However, in application, stationary schemes are not practical. There are several reasons. Firstly, some of schemes are devised on closed or work well if don't consider interpolating boundary points. Secondly, in application, most of models are piecewise smooth instead of smooth on the whole surface. Also, some schemes need more calculation but are suitable for a certain problem. Therefore, there are more extensive problems relating to subdivision surfaces are discussed. In 1994, Hoppe et al [30] discussed how to improve subdivision schemes on piecewise smooth surface. In 1999, BIERMANN et al [31] suggested improved Loop's and Catmull-Clark's schemes for piecewise smooth surfaces.

Piecewise smooth surfaces [30] are the surfaces, which can be constructed from surfaces with piecewise smooth boundary joint together. If the resulting surface is C^1 -continuous at the common boundary of two pieces, this common boundary is a crease. However, if we still want to keep C^1-continuity at the crease, subdivision rules have to be changed for such areas, especially for approximating schemes.

In piecewise smooth surfaces, for the interior smooth areas, there is no change for the rules and stencil of both schemes. Also, the same rules are kept for creases or boundaries.

For areas around corner or crease, the weight is changed for each vertex in the stencil.

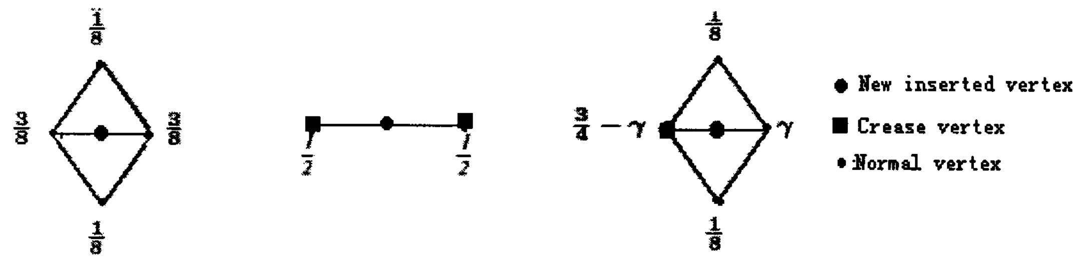


Figure 19: Rules of point insertion in Piecewise Smooth Surfaces according to Loop schemes

In **Figure 19**, first one defines how to insert a new point between two crease points. The second defines how to insert a new point for smooth area. The third defines how to insert a new point for a crease or corner point neighbor. The γ is defined in the following,

$\gamma(\theta) = \frac{1}{2} - \frac{1}{5}\cos\theta$. For a vertex with degree k, the θ following is chosen,

$\theta = \frac{\alpha}{k}$, for convex corners where is the angle between the crease edges meeting at the corner;

$\theta = \frac{\pi}{k} = \theta_k$, for smooth crease vertices;

$\theta = \frac{\pi - \alpha}{k}$, for concave corner vertices.

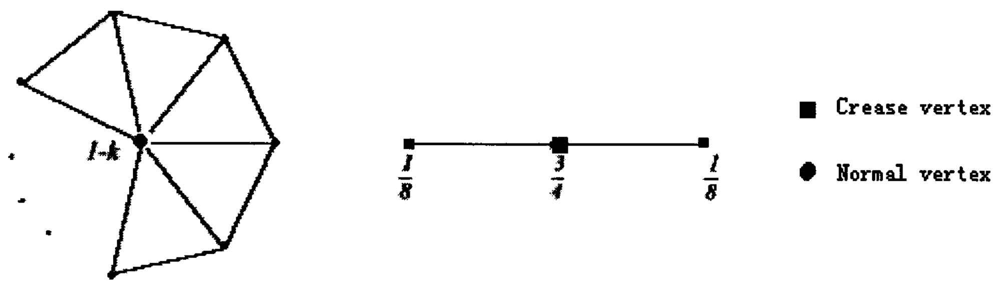


Figure 20: Rules to compute new position of old vertices according to Loop schemes in piecewise smooth surfaces

In implementation, on a completely closed mesh, we have to put tags on vertices to tell the program where the crease is, and where the corner is.

5.3 Refinement Scheme for PCMs

As for the refinement, we firstly construct point neighborhood with the help of Octree. In experiment, we still use octree to subdivide compact models, and for each leaf node, the number of point should be in the range of (4, 12), because, if less than 4, it is impossible to construct triangle-mesh-liked neighborhood. If greater than 12, a neighborhood with high valance will be built, but it is proved by Reif [23] that the higher valance, the more difficult to keep smoothness. After constructing neighborhood, we form a virtual triangle fan with the centroid of neighborhood as center, and apply improved refinement schemes on the virtual triangle fan, which will be discussed in 5.4.2.

5.3.1 Point Neighborhood Algorithm

A neighborhood usually is defined by a centroid, or a center point, and the radius. In our refinement process, the radius, in practice, has been defined by the size of the leaf node. The following are the facts, which have to be considered during neighborhood construction.

Discard point from other side of surface. Because in a non-dense, especially very slim model, for a leaf node, it is very possible to hold points from disjoint parts of the surfaces (see **Figure 12(b)**). With normal information, we got from simplification process; it is easy to remove these unwanted points.

Choose Centroid of the Neighborhood. We choose the centroid in a straightforward way. The average of point set in a leaf node can be defined as $\overline{p} = \frac{1}{n}\sum_{i=1}^{n} p_i$, and the centroid pc is select from the point set in the leaf node, which meeting the requirement of $p_c - \overline{p} <= \min(pi - \overline{p}), i = 1 \ldots n$

Map Point Set into 2D. With the centroid and its normal (we have got it in Chapter 4), a specific plane $P(p_c, \vec{n})$ is defined, which roughly represents the tangent plane of the surface at that area. So we can map the point set onto this plane and, in 2-dimension, get a new point set. This map is one-to-one.

Form the neighborhood in 2D. Now we can get a sorted the point set in 2D according to the angle of two neighbored points. If the angle is less than 10^o or greater than 160^o, this point should be discarded, because it forms a triangle with very sharp corners (**Figure 21**). Here, there is a special condition, that is, if the centroid is a corner-featured point, when choosing points to form a neighborhood, crease points have higher priority because the angle θ, between two creases forming the corner, has to be found (see **Section 5.3**). If unable to find two crease points in the same leaf node, more points should be borrowed from other leaf node neighbors to find two nearest crease points. After we get the 2D point neighborhood, we can get the 3D point neighborhood by map the 2D neighborhood reversely.

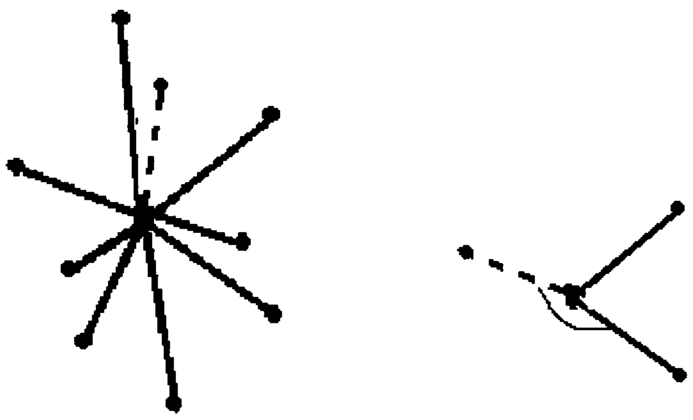


Figure 21: Discarded point (one end of dash segment). Left: too small angle; right: too great angle

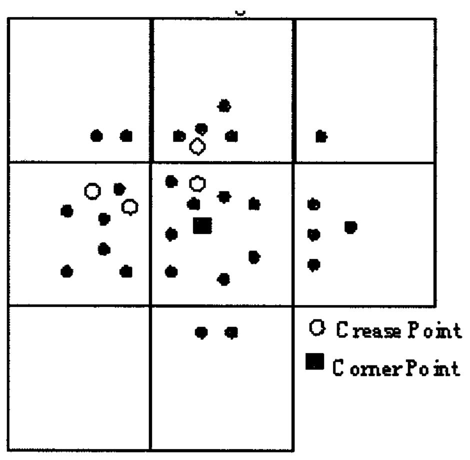


Figure 22: Corner always has at least 2 crease neighbors (illustrated in 2D)

Bad Neighborhood: If a neighborhood contains less than 4 points, it is impossible to form a virtual triangle fan. Therefore, such neighborhood should be discarded. If a neighborhood contains more points than 12, it is also not good for smoothness. In experiment, we have not bumped into such neighborhoods. For a leaf node with a bad neighborhood, it will be merged to one of its neighbors, which contains the least points. Then we can construct new neighborhood on the two leaf nodes.

Stencils for Refinement: If we virtually connect points in the neighborhood to the centroid, and also connect all the points except centroid one after another according to the sorted sequence, we got in Section 5.4.1, we can get a triangle fan shown in Figure 23.

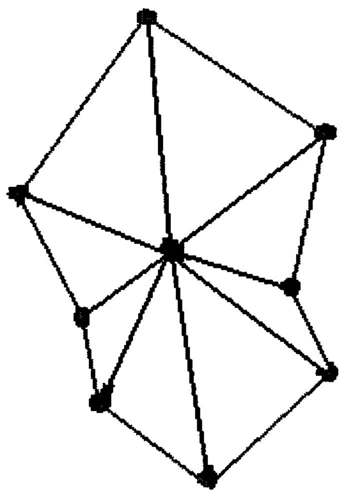


Figure 23: Virtual triangle fan from point neighborhood

Because Loop Scheme works on 1-neighborhood, it is enough to apply subdivision scheme on it. Here, we should have some modification or specification in order to make the Loop scheme work on point neighborhood well. Firstly, we should put tag on the points in the compact model to explicitly mark out boundaries for each smooth piece of surface. Fortunately, PCMs used in experiment is not very large, we can mark out non-smooth creases and corners manually after simplification. Apparently, we can apply interior smooth stencils directly on the *virtual triangle fan*, and the edge and vertex rules for smooth crease vertex also works well. The only problem is on the edge rule of crease and corner neighbors. It is straightforward to calculate θ. For a neighborhood with the corner-featured centroid, according to **Section 5.3.1**, there must be two crease points in the neighborhood. In the point neighborhood with corner point centroid, the angle can be defined in the following; the first crease neighbor p_1, the second crease neighbor p_2, and the corner point $P_{centroid}$ form the angle $\angle p_1 P_{centroid} p_2$. Here, we also assume that all the corners in PCMs are convex.

5.3.2 Offset for Refinement

In the refinement, if we always use the exact bounding cubic box according to the size of the PCMs, gaps will be left among leaf nodes because we always voxelize the PCM almost in the same way even though the leaf node is small enough. Therefore, we have to use an offset to define a different bounding cubic box in the second round of refinement. In order to make the gap from last refinement round falls into the center of the leaf node, the offset is kept in the domain of [NeighborhoodSize/3, NeighborhoodSize/2]. The size of neighborhood is the same with the size of leaf node.

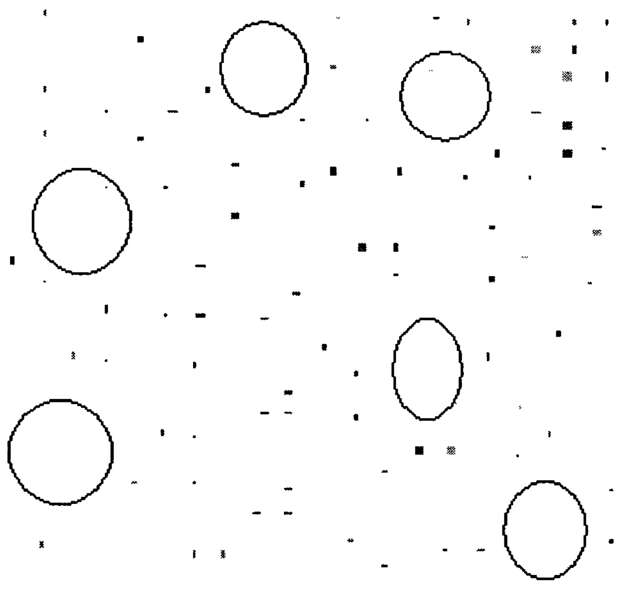


Figure 24: A local surface in Club's top. Without offset, refinement will leave holes, illustrated by circles

5.4 Experiment Results

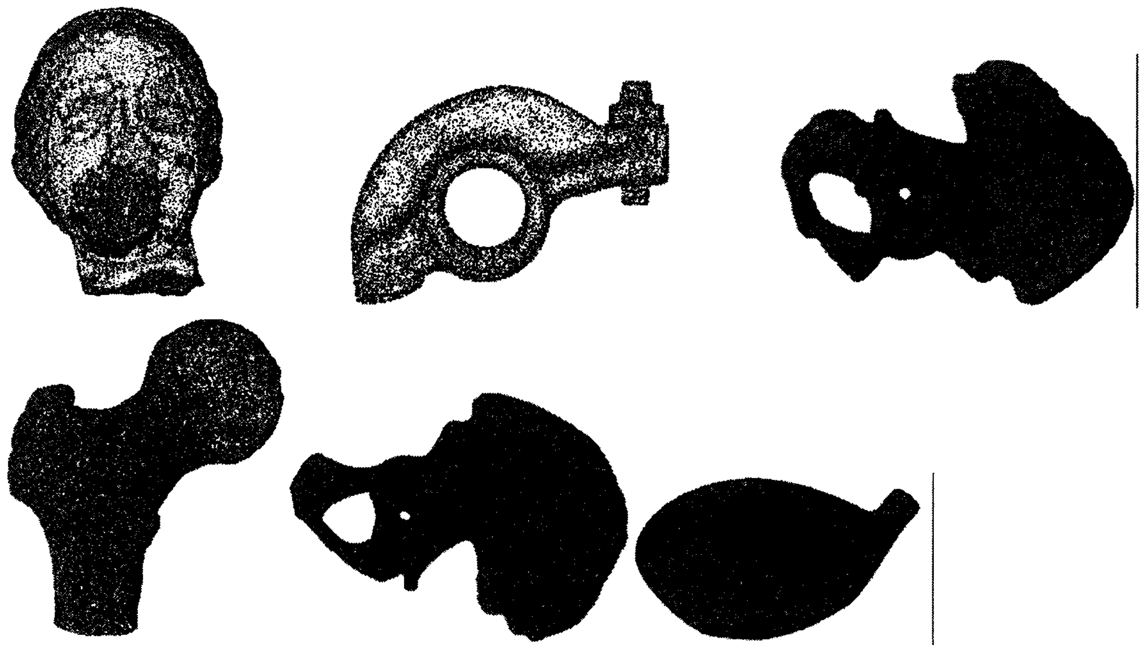
In experiments, we almost tried to refine compact models from any level to any level. In order to avoid holes on the refinement results, we apply at least two rounds of refinement schemes on each model with an offset (**Appendix A**). In fact, most of the following results are two or three rounds refinement. The first round refinement almost happens on all leaf nodes, and two and three rounds mainly focus on making the flat area denser.

Compact Model	Refined Model	Time Consumed	Size of Neighborhood
11.76%	48.42%	0: 0: 56:211	0.23940cm
22.22%	75.34%	0: 0: 46:878	0.15932cm
60.56%	103.75%	0: 0: 39:878	0.05615cm
80.37%	123.43%	0: 0: 42: 563	0.03523cm
Actual Size: 12.2004cm * 8.7345cm*5.9604cm			

Table 10: Results of refinement for Club model

Compact Model	Refined Model	Time Consumed	Size of Final Neighborhood
12.73	56.39%	0: 0: 26: 597	2.4198cm
23.75%	78.56%	0: 0: 32: 505	1.2099cm
42.13%	79.26%	0:0:38:61	0.903cm
58.54%	123.01%	0:0:45:851	0.287cm
82.57%	125.33%	0:0:36:851	0.240cm
Actual Size: 14.101*13.904*15.488			

Table 11: Results of refinement for Hip Bone Model



By Sequence: Venus (25.54%), Rocker Arm (35.87%), Hip Bone (55.39%), Balljoint (54.28%), Hip Bone (120.17%), and Golf Club (123.43%). The ratio is comparing with the point number of original models. All pictures are rendered only with OpenGL command glVertex3d();

Figure 25: Pictures of refinement Results

5.5 Chapter Summary

In this chapter, we mainly explain the experiments of refinement of compact models, which are got from simplification process of Chapter 4. The refinement schemes are borrowed from subdivision schemes on triangle mesh: Loop schemes. Comparing to the work of Guennebaud et al [02], our refinement schemes are more simple and fast. In addition, approximating schemes can cover noise from original PCMs, or errors from simplification process.

Chapter 6
Comparison of Original, Compact, and Refined Models

In the simplification and refinement processes, we just obtained compact and refined PCMs as the results. However, we have to find some approaches to evaluate these outputs. In this chapter, regenerated PCMs (simplified or refined) will be compared with the original ones in order to find how much is lost and how much is retained on the surfaces described by PCMs during the processes.

The algorithms in this research are heavily based on geometry information of PCMs, and the results of refinement and simplification are not orientated directly to rendering, so it is not good idea to purely use rendered pictures of refined and simplified PCMs for comparison. The only reliable way is finding geometry difference between processed PCMs and the original ones.

6.1 Comparing Compact PCMs with the Original One

6.1.1 Theoretical Basis

In the simplification stage, the compact PCMs are obtained by resampling points from the original one. Therefore, the compact PCM shares the same dimension and coordinate-system with the original one. In addition, the compact PCM means that *one point in a **good compact PCM** should be able to represent a cluster of points from the original model.* For PCMs, the only information is 3D coordinates, so checking capacity of representation through 3D coordinates is a reasonable way.

In StudioTools [66], the most basic comparison function is introduced, that is, to compute the distance between two point cloud models, which share some similarity. Given two

PCMs $S_1 = \{p_{1,i}\}, 0 < i < n$, and $S_2 = \{p_{2,i}\}, 0 < i < m$ (m and n is the size of the two PCMs, respectively, and suppose that n<m). At first, we have to decide that given one point from *S*$_2$, how many points from *S*$_1$ correspond to it in average. This is decided by the *capacity of representation*, *num*=$\frac{m}{n}$. If points in *S*$_1$ are distributed evenly, given a point $p_{1,t}(0 < t < n)$ from *S*$_1$, a set of points $P = \{p_{2,i}\}, 0 < i < \| num \|$ can be found from *S*$_2$. Such a set of points is closest to the given point $p_{1,t}, 0 < t < n$ from S_1, for comparison.

6.1.2 Definition of Bias

According to a point $p_{1,t}, 0 < t < n$ in compact model, after a cluster of points $P = \{p_{2,i}\}, 0 < i < \| num \|$ (||*num*|| means the *modulus* of *num*) in original model is found, the value to evaluate goodness of representation can be defined as $\xi = \frac{1}{\| num \|} \sum_{i=1}^{\| num \|} | pi - p^{'} |$. The formula means how the $p_{1,t}, 0 < t < n$ is close to the average of $P = \{p_{2,i}\}, 0 < i < \| num \|$. *For the whole compact model, we can get an average bias as output for* P' .

6.1.3 A Normalized output for the Whole Compact Model

In **Section 6.1.2**, a bias value is defined, and an accumulation for the whole compact PCM can also be made out. The value, however, is not normalized, so just based on that value we can not make sure how well or bad the compact PCM is. Therefore, normalizing the output is necessary.

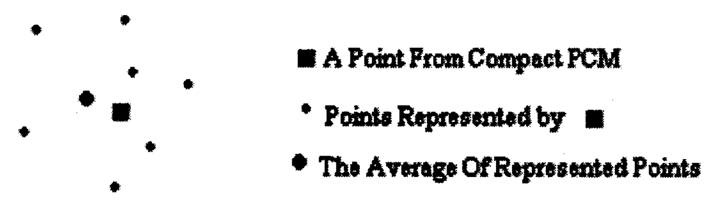

■ A Point From Compact PCM
• Points Represented by ■
● The Average Of Represented Points

Figure 26: A point from compact PCM and points from the original, it represents

Firstly, the bias for each point in compact model should be normalized. In **Figure 26,** we illustrated a point $p_{1,t}, 0 < t < n$ from compact PCM S_1 and a corresponding point set $P = \{p_{2,i}\}, 0 < i <\| num \|$ represented by from the original PCM S_2. The average of P is $\overline{p}$, and a point $p_{2,f}$ from P, which is farthest to the $\overline{p}$ can be found. If the value of $|\overline{p} - p_f|$ is scaled to one (the unit value), the normalized bias $\overline{\overline{\xi}}$ of p' should be $\overline{\overline{\xi}} = \frac{1}{\| num \| * | \overline{p} - p_f |} \sum_{i=1}^{\|num\|} | pi - p' |$ because the bias of p' must fall into the domain of $[0, |\overline{p} - p_f|]$.

Based on $\overline{\overline{\xi}}$, the bias for the whole comparison can be defined as $\delta = \frac{1}{n} \sum_{i=1}^{n} \sqrt{\overline{\overline{\xi}}_i^2}$, and n is the number of points in compact models.

6.1.4 Dynamic Capacity of Representation in Our Algorithm

According to our simplification algorithm, more points are kept for creases and corners, so in simplification process, a point of flat area in compact PCM represents more points than the points of crease and corners. Specifically, *in crease area, a point of compact PCM represents 3—7 points, and in corner area, a point represents around 5 points. However, for flat area, the* *number depends on the compact rate.* According to the capacity of representation, we can find a cluster of points in original model, which has the nearest distance to the corresponding point in compact model.

As for implementation, the octree is used to subdivision the original model, so that when we want to find a cluster of points corresponding to a point in compact model, we just find in an octree leaf node or its neighborhood instead of the whole model.

6.2 Compare Refined Models with Original Ones

For the refined result, the method for compact model is unsuitable, because, refinement means that output is dense, and sometimes even denser than the original model.

The comparison is carried out on two levels. The first is to evaluate how many features have been lost and the second is based on kept features in refined PCM, evaluating how much difference there is between the corresponding features in both PCMs.

If a refined model almost has the same density of the original one, we can also apply PCA on each local surface as we did in simplification process to find local features. The comparison is divided into several steps: Construct Octree on the Original PCM. We can use the same work done in simplification process; Construct the Same Octree on the Refined PCM. This can be got without much difficulty after making the refined model shares the same dimension and coordinate-system. Ideally and also proved in experiments, if the simplification and refinement algorithms are good enough, in the same structure and size of octree, for a leaf node with the same index, there is the same feature inside.

6.2.1 Feature Loss

The first step is to find how many features in refined PCM we have lost with the original PCM as a standard. This is a statistic on difference of numbers of flat, crease, and corner leaf nodes respectively. Please see the Section 6.3.1 for the experiment results. Generally, if there are too many feature losing, we will not evaluate feature accuracy again because that will get meaningless on some degree.

6.2.2 Feature Accuracy

The second step is to do more accurate evaluation for kept features. For flat area, the two first-eigenvalues are compared. The first eigenvalue represent the flatness in the flat leaf node. More great the first eigenvalue, more possibly, this area will become a corner or crease. we keep the difference DF (absolute value) of the first eigenvalue for a flat leaf node, and for the whole PCM an least square root value $\overline{D_F} = \frac{1}{n_f}\sqrt{\sum_{i=1}^{n_f} {D_F}^2}$ (n_f means the number of flat nodes in the whole PCM) can be evaluated. Because the first eigenvalue is normalized already, so the difference or bias DF between two first eigenvalues is also normalized, so does $\overline{D_F}$. In crease areas, as mentioned in simplification process, a simplex crease can be defined by two crossed planes, and the crease can be described numerically with the angle between the two semi-planes. The value of crease angle can be evaluate according to the way in Figure 27, which shows a cross-section of a crease by cutting the crease with a plane, which is perpendicular to both semi-planes of the crease. In the cross-section, the normals of the two planes and the cross-section lines of the crease forms a quadrilateral with two right angles, so the sum of rest two angles A and B is π, denoting A is the angle between two normals and B is the crease angle. For comparison, in two corresponding crease leaf nodes from refined and original PCMs respectively, we evaluate the difference or bias DC of two crease angles. If considering the crease angle from the original PCM Ac is correct, the bias can be normalized as $\frac{Dc}{A_C}$. Also, like what is done for flat leaf nodes, for all the creases in the whole model a normalized bias in least squared root $\overline{D_C} = \frac{1}{n_c}\sqrt{\sum_{i=1}^{n_c}\left(\left(\frac{Dc}{Ac}\right)_i\right)^2}$ (n_c is the number of crease leaf nodes in the whole PCM) can be evaluated.

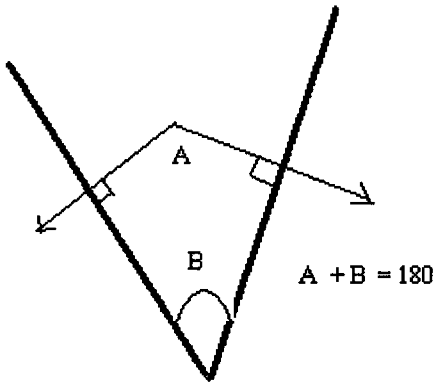


Figure 27: Approach to calculate crease angle

Because a corner is a pyramid-like shape, it can be described with the solid angle. However, it is unnecessary to calculate the exact values of solid angles in refined leaf node and original one. As shown in Figure 28 (Left: Find the tangent plane and map the point set on to the plane, Right: rasterizing the area into grids, the 2D point set occupies), a solid angle is defined by the cross area of a unit sphere with the corner top as the center and a corner. If we map the solid angle onto the plane, which is the tangent plane of the unit sphere at the center of the corner, we will get an area in 2D (shadowed area in the Figure 26). After rasterizing the 2D area, we can evaluate the 2D area with number of grids, which is corresponding to the value of solid angle. As for evaluating the difference of solid angles, we can just base on the difference or bias D_R of the grid number. If considering the crease angle from the original PCM A_R is correct, the bias can be normalized as $\frac{D_R}{A_R}$. Also, like what is done for flat leaf nodes and crease leaf nodes, for all the corners in the whole model, a normalized bias in least squared root $\overline{D_R} = \frac{1}{n_R}\sqrt{\sum_{i=1}^{n_R}\left(\left(\frac{D_R}{A_R}\right)_i\right)^2}$ (n_R is the number of corner leaf nodes in the whole PCM) can be evaluated.

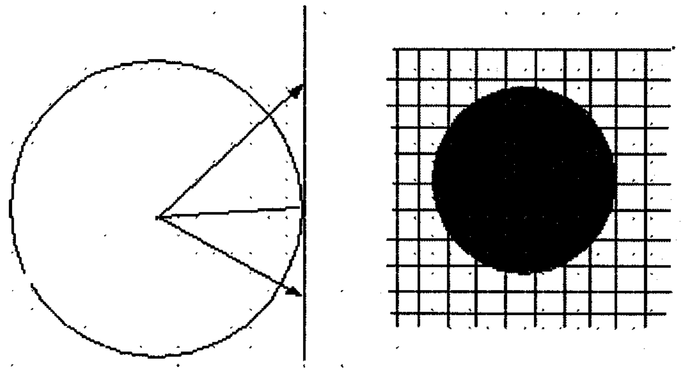


Figure 28: Approach to evaluate solid angle with grids

In order to evaluate the total difference or bias about features between the two models, a formula is defined as $P_F * \overline{D_F} + P_C * \overline{D_C} + P_R * \overline{D_R}$, where P_F, P_C, and P_R denote the percentage of flat leaf nodes, crease leaf nodes, corner leaf node in the whole octree, respectively.

6.2.3 Scales Used in Comparison between Refined and Original PCMs

As mentioned above, we use the two octree hierarchies to voxelize refined and original PCMs at the beginning of comparison. The two octree hierarchies share with the same structure and size. That means the both PCMs should share the same coordinates and dimension. The first requirement is guaranteed definitively, but the second cannot be directly. *From the knowledge of* *subdivision surfaces, the result of approximating schemes will have a little shrink comparing to the control polygon* (see **Figure 29** for details*)*. Our approximating scheme has the same effect on the final results.

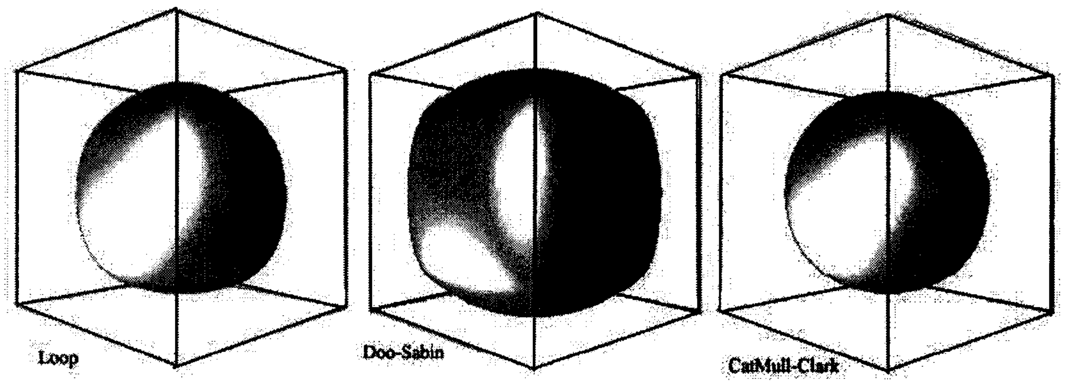


Figure 29: Results of Subdivision Surfaces with approximating schemes[4]

Therefore, for comparison, we have to scale the refined PCM to the same dimension of the original one. For accuracy, we use different scales in x, y, and z directions. It is not difficult to get the scale for one dimension. For example in x-direction, if denoting size of the original and refined PCMs in x-direction as X_o and X_r respectively, the scale applied on refined PCM in x-direction is $\frac{X_o}{X_r}$.

[4] The picture is from [20] in pp. 85

6.3 Experiment Results

6.3.1 Compare Compact PCMs with the Original

In Table 12, the experiment results of comparing compact PCMs with the original are listed for Golf Club and Hip Bone. For each model, the first column is PCMs used for comparison with various densities, the second to the final one are *average capacity of representation* for a flat point, a crease point, corner point, respectively.

When only a little simplification is applied on flat areas, the compact PCMs can still represent the original very well with a slight bias. When the compact rate comes to some degree— around 40%, the error in the result increases dramatically, because we have touched corners and creases. When with a compact rate higher than 40% or so, the error mainly comes from flat area, because we implement higher compact rate by merging flat leaf node together, and a flat point has to represent much more points.

The up limitation of the compact rate with current algorithm is around 10%. From experiments, the up limitation of error is 40% of the original one.

Part I: Comparing Results of Golf Club PCMs

Model	Flat Areas	Creases	Corner	Final Report
79.23%	4	1	1	0.52%
60.56%	8	1	1	0.98%
38.42%	20	7	6	10.73%
22.22%	42	8	6	18.66%
11.76%	74	9	6	33.85%

Part II: Comparing Results of Hipbone PCMs

Model	Flat Area	Creases	Corner	Final Report
82.57%	4	1	1	0.65%
58.54%	7	1	1	0.87%
42.13%	22	6	7	13.25%
23.15%	44	9	7	17.74%
11.54%	78	10	7	36.19%

Table 12: Results of comparison for compact PCMs

6.3.2 Compare Refined PCMs with the Original

In Table 12, some experiments are listed. The accumulation of errors for leaf nodes are in real number, and for creases in radian, for corners in grids measured with centimeter unit.

In Part I of the Table 12, the density of refined PCMs used for comparison is in the range of 65 to 85%. The results are to some degree in mess because of the density is not enough and cause the errors in *principle component analysis*. After constructing the octree on refined PCMs with the same depth-level of the original one, the point number in each leaf node less than 20. In Part

II, the comparison results are much better than in Part I because the refined PCMs are dense enough for *principal component analysis.* For all the models, there is only light or no feature losing. This is an expected result because during simplification, even though higher compact rate required, we keep points in crease and corner areas as many as possible.

Model	Missed Features	Errors of Flatness	Errors of Creases	Errors of Corners	Normalized Report
Hipbone (65.33%)	18 creases 4 corners	54.2016	463.569	1032	23.25%
Isis (74.28%)	20 creases 2 corners	23.5485	76.245	507	32.7452%
Golf Club (87.43%)	5 creases 2 corners	20.1546	32.1584	204	18.1639%
Venus (77.45%)	13 creases 4 corners	17.8526	45.1485	388	22.15362%
Rocker Arm (78.37%)	10 creases 1 corners	6.4573	18.4526	297	29.8549%
Balljoint (72.12%)	15 creases 4 corner	12.3645	60.1284.	482	31.2312%

Table 13: Comparison of refined PCMs with density in range of 60-80% of the Original

Model	Missed Features	Errors of Flatness	Errors of Creases	Errors of Corners	Normalized Report
Hipbone (125.33%)	6 creases 0 corners	26.21	111.6	852	7.31%
Isis (120.28%)	0 creases 0 corners	11.80	26.90	204	4.06%
Golf Club (123.43%)	2 creases 0 corners	16.97	15.46	636	6.99%
Venus (98.90%)	10 crease 0 corners	9.963	27.13	204	9.22%
Rocker Arm (105.37%)	0 creases 0 corners	2.163	7.912	214	3.64%
Balljoint (103.25%)	2 crease 1 corners	5.088	30.26	317	8.25%

Table 14: Comparison of refined PCMs with density near or more than 100% of the Original

6.4 Chapter Summary

In this chapter, we explained the strategies for comparing compact and refined PCMs with the original one as the measure of bias. For compact and refined PCMs, different methods are used because of their different distribution of the points.

As for comparison between refined and original PCMs, the PCA is applied on both of them. As mentioned in Chapter of simplification process, the PCA works well only when the model is dense enough. Therefore, when the refined model is 80% of the original or lower, the result of comparison is very bad or meaningless. In the future, we may find a better way to compare low-level refined model with the original one, because sometimes, only low-level refined PCMs are required.

Chapter 7
Conclusion and Future Work

7.1 Contributions of This Research

The main contributions of this research are the following:

1. Considerable amount of work has taken place in the area of point cloud models and their representations. Because of their vast sizes and the discrete nature of the geometric data in these models, simplification and refinement techniques are of very great importance. They also have their own unique problems as compared to similar techniques for surface models with topological information such as polygon models. As part of this research, we have carried out an exhaustive survey of techniques for simplification and refinement of point cloud models, classified them, and identified the advantages and disadvantages of each of the method. Specifically, problems in the current approaches have been discussed. This survey may serve as a starting point for others interested in the same problem.
2. The other major contribution is the development and implementation of a new feature-based method that detects flat, crease, and corner regions, resamples a dense point cloud model for simplification, and successfully implements Loop's subdivision schemes to refine the point cloud model with piecewise smoothness. This method required the integration of a number of well established algorithms: Octree, Principal Component Analysis, and Subdivision Surfaces. The results from this implementation are also provided showing that reasonable compaction and refinement of point cloud models can be achieved using this method.

7.2 Future Improvements in Asymmetric Use of Compaction Rate

During simplification, we have successfully detected features and kept them in compact models achieving high compaction rates up to 10%. Because the same compact rates are applied to corners and creases as for flat regions, we biased more towards flat areas. Also, based on such strategy, currently the maximum compact rate is around 10% for most of the models. Trying for higher compaction rates would result in losing edge and corner features. In the future, asymmetric strategies among flat areas, creases, and corners should be developed for higher compact rate, for example, up to 5% or even better.

7.3 Improving Octree Related Algorithms

As for the octree algorithms, in experiments, a positive offset has been used to increase the dimension of the initial bounding box of octree to avoid special leaf nodes. More careful studies must be done to arrive at new strategies that decrease the number of special leaf nodes.

7.4 Improving Accuracy of Two Stage Processes

In comparison between compact and original PCMs, especially for higher compact rate, most bias is from flat areas because the high compact rate is implemented mainly by simplifying flat areas. And such a way will not work well if a model has more creases and corners instead of flat areas. That requires more research.

In comparison between refined and original PCM, more bias is given to creases and corners even though higher refinements results are pleasing. The main reason is the way our algorithm of refining point neighborhood has been devised. It can be seen in the experiments that with current algorithm, sometimes, it is difficult to refine local surface areas around corners because at least two crease points have to be found for constructing point neighborhood. In addition, the algorithm supposes the corner is always convex. Finding a more practical algorithm for the corner is necessary for better results.

Bibliography

[01] **P. Bhattacharya**, Efficient Neighbor Finding Algorithms in Quadtree and Octree, Master's Thesis, Indian Institute of Technology, Nov, 2001.

[02] **G. Guennebaud and L. Barthe and M. Paulin**, Real-Time Point Cloud Refinement, Eurographics Symposium on Point-Based Graphics (2004)

[03] **Mark Pauly, Markus Gross, and Leif P. Kobbelt**, Efficient Simplification of Point-Sampled Surfaces

[04] **Marc Alexa, Johannes Behr, Daniel Cohen-Or, Shachar Fleishman, David Levin, and Claudio T. Silva**, Point Set Surfaces, IEEE Visualization 2001

[05] **William H. Press**, Numerical Recipes in C: The Art of Scientific Computing, Cambridge University Press 1992, ISBN: 0521431085

[06] **Stafen Gumhold, Xinlong Wang, and Rob Macleod**, Feature Extraction From Point Clouds, Proceedings, 10th International Meshing Roundtable, Sandia National Laboratories, pp.293-305, October 7-10 2001

[07] **G. Guennebaud, L. Barthe, and M. Paulin**, Real-time point cloud refinement, Eurographics Symposium on Point-Based Graphics (2004)

[08] **Aravind Kalaiah, and amitabh Varshney**, Statistical Point Geometry, Eurographics Symposium on Geometry Processing (2003)

[09] **Marc Alexa** et al., Point Set Surfaces, IEEE Transactions on Visualization and Computer Graphics, Vol. 9, No. 1, January-March 2003

[10] **Mark Pauly, Richard Keiser, and Markus Gross**, Multi-scale Feature Extraction on Point-Sampled Surfaces, Eurographics 2003

[11] **Leif P. Kobbelt, Mario Botsch et al.**, Feature sensitive Surface Extraction from Volume data, ACM SIGGRAPH 2001

[12] **P. Cignoni, C. Rocchini, and R. Scopigno**, Metro: Measuring error on simplified surfaces, The Eurographics Association 1998

[13] **Lars Linsen**, Point Cloud Representation, Technical Report No. 2001-3, Fakultät für Informatik, Universität Karlsruhe, 2001.

[14] **Carsten Moenning, and Neil A. Dodgson**, Fast Marching farthest point sampling for implicit surfaces and point clouds, Technical Report, Cambridge University

[15] **Michael Garland, and Paul S. Heckbert**, Surface simplification using Quadric Error Metrics, SIGGRAPH 97

[16] **Sushil Bhakar, Liang Luo, and S.P. Mudur,** View Dependent Stochastic Sampling for Efficient Rendering of Point Sampled Surfaces, WSCG 2004

[17] **Szymon Rusinkiewicz, and Marc Levoy**, QSplat: A multiresolution Point Rendering System for Large Meshes, ACM 2000

[18] **Denis N. Zorin**, Stationary Subdivision and Multiresolution Surface Representations, Thesis, California Institute of Technology, 1997

[19] Online Notes of Geometry Modeling, http://graphics.idav.ucdavis.edu/education, University of California, Davis

[20] **SIGGRAPH course 2000**, Subdivision of modeling and animation

[21] **A. Habib, and J. Warren**, Edge and vertex insertion for a class of C1 subdivision surfaces, CAGD 16 (4) (1999) 223–247, previously available as a TR, Rice University, August 1997.

[22] **J. Peters, and U. Reif**, The simplest subdivision scheme for smoothing polyhedra, ACM Transactions on Graphics 16 (4) (1997) 420–431.

[23] **REIF,** U. A Unified Approach to Subdivision Algorithms near Extraordinary Points. Computer Aided Geo. Des. 12 (1995), 153–174.

[24] **LOOP, C.** Smooth Subdivision Surfaces Based on Triangles. Master's thesis, University of Utah, Department of Mathematics, 1987.

[25] **KOBBELT, L.** Interpolatory Subdivision on Open Quadrilateral Nets with Arbitrary Topology. In Proceedings of Eurographics 96, Computer Graphics Forum, 409–420, 1996.

[26] **DYN, N., LEVIN, D., and GREGORY, J.** A Butterfly Subdivision Scheme for Surface Interpolation with Tension Control. ACM Trans. Gr. 9, 2 (April 1990), 160–169.

[27] **ZORIN, D., SCHRODER, P., and SWELDENS, W.** Interpolating Subdivision for Meshes with Arbitrary Topology. Computer Graphics Proceedings (SIGGRAPH 96) (1996), 189–192.

[28] **SCHWEITZER, J. E.** Analysis and Application of Subdivision Surfaces. PhD thesis, University of Washington, Seattle, 1996.

[29] **KOBBELT, L.** $\sqrt{3}$ Subdivision. Computer Graphics Proceedings, Annual Conference Series, 2000.

[30] **HOPPE, H., DEROSE, T., DUCHAMP, T., HALSTEAD, M., JIN, H., MCDONALD, J., SCHWEITZER, J., and STUETZLE, W.** Piecewise Smooth Surface Reconstruction. In Computer Graphics Proceedings, Annual Conference Series, 295–302, 1994.

[31] **BIERMANN, H., LEVIN, A., and ZORIN, D.** Piecewise smooth subdivision surfaces with normal control. Tech. Rep. TR1999-781, NYU, 1999.

[32] **ZHANG, H., and WANG, G.** Honeycomb subdivision, Journal of Software, 2002 Vol.13, No.4

[33] **Tony DeRose, Michael Kass and Tien Truong**, Subdivision Surfaces in Character Animation, SIGGRAPH '98

[34] **ZORIN, D., SCHRODER, P. and SWELDENS, W.** Interpolating Subdivision for Meshes with Arbitrary Topology. Computer Graphics Proceedings (SIGGRAPH 96) (1996), 189–192.

[35] **Denis Zorin and Peter Schroder**. A unified Framework for Primal/Dual Quadrilateral Subdivision Schemes. Compu. Aided Des. 2000.

[36] **Luiz Velho, and Denis Zorin**. 4-8 Subdivision. Computer Aided Design. 2001.

[37] **Denis Zorin, Peter Schroder, and Wim Sweldens**. Interpolating Subdivision for Meshes with Arbitrary Topology.

[38] **Adi Levin**, Combined subdivision schemes for the design of surfaces satisfying boundary conditions, Computer Aided Geometric Design, Volume 16

[39] **M. Levoy and T. Whitted**. The use of points as a display primitive. Technical Report 85-022, Computer Science Department, UNC, Chapel Hill, January 1985

[40] **H. Pfister, M. Zwicker, J. van Baar, and M. Gross**, Surfels: Surface Elements as Rendering Primitives, SIGGRPH 2000, Computer Graphics Proceedings

[41] **P. Hebert, D. Laurendeau, and D. Poussart.** Scene reconstruction and description: Geometric primitive extraction from multiple view scattered data. In IEEE Computer Vision and Pattern Recognition 1993, pages 286–292, 1993.

[42] **M. Rutishauser, M. Stricker, and M. Trobina**. Merging range images of arbitrarily shaped objects. In IEEE Computer Vision and Pattern Recognition 1994, pages 573–580, 1994.

[43] **P. Besl and N. McKay**. A method for registration of 3-d shapes. IEEE Transactions on Pattern Analysis and Machine Intelligence, 14(2):239–256, February 1992.

[44] **N. Amenta, M. Bern, and M. Kamvysselis**. A new voronoi-based surface reconstruction algorithm. Proceedings of SIGGRAPH 98, pages 415–422, July 1998.

[45] **Shachar Fleishman,Marc Alexa,Daniel Cohen-Or,Cl'audio, and T. Silva**, Progressive Point Set Surfaces, ACM Transactions on Graphics, Vol. 22, No. 4, October 2003.

[46] **D. Levin**, Mesh-Independent Surface Interpolation, Geometric Modeling for Scientific Visualization 2003

[47] **H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle,** Surface Reconstruction from Unorganized Points Computer Graphics (SIGGRAPH) Volume 26, Pages 19-26, 1992

[48] **M. Waschbüsch, M. Gross, F. Eberhard, E. Lamboray and S. Würmlin,** Progressive Compression of Point-Sampled Models, Eurographics Symposium on Point-Based Graphics (2004)

[49] **T. K. Dey and J. Hudson.** PMR: point to mesh rendering, a feature-based approach. Proc. IEEE Visualization 2002, (2002), 155--162.

[50] **Gu X., Gortler S.J., and Hoppe H.** Geometry images. In proc. SIGGRAPH '02 (2002), pp.355—361

[51] **Hoppe H.**, Progressive meshes. In Proc. SIGGRAPH '96 (1996), pp.99—108

[52] **Botsch M., Wiratanaya A., and Kobbelt L.**, Efficient high quality rendering of point sampling geometry. In Proc. Eurographics workshop on Rendering (2002), pp.53-64

[53] **Mark Pauly, Markus Gross, and Leif P. Kobbelt,** Multiresolution Modeling of Point-Sampled Geometry. ETH Zurich Technical Report, 2002

[54] **T. K. Dey, J. Giesen and J. Hudson.** Decimating Samples for Mesh Simplification, In Proc. 13th Canada Conference on Computational Geometry. Waterloo, Canada, 2001, p.85-88.

[55] **C. Moenning and N. A. Dodgson,** A new point cloud simplification algorithm, 3rd IASTED International Conference on Visualization, Imaging, and Image Processing (VIIP 2003) 8-10 Sep 2003, Benalmádena, Spain

[56] **C. Moenning and N. A. Dodgson,** Fast Marching farthest point sampling for implicit surfaces and point clouds. Computer Laboratory technical Report No. 565, University of Cambridge, UK, 2003

[57] **Diego Nehab and Philip Shilane,** Stratified Point Sampling of 3D Models, Eurographics Symposium on Point-Based Graphics (2004)

[58] **Botsch M., and Kobbelt L.,** High-Quality Point-Based Rendering on Modern GPUs, in 11th Pacific Conference on Computer Graphics and Applications (2003), pp. 335-343

[59] **Facundo Mémoli and Guillermo Sapiro**, Comparing Point Clouds, Eurographics Symposium on Geometry Processing (2004)

[60] http://graphics.stanford.edu/

[61] **CyberWare**, http://www.cyberware.com, http://www.cyberware.com/samples/index.html

[62] **C. Bajaj, F. Bernardini, and G. Xu**. Automatic reconstruction of surfaces and scalar fields from 3D scans. Computer Graphics Proceedings, Annual Conference Series. Proceedings of SIGGRAPH '95 (1995), pp. 109-118.

[62] **N. Amenta, S. Choi, T. K. Dey and N. Leekha.** A simple algorithm for homeomorphic surface reconstruction. Proc. 16th Sympos. Comput. Geom., 2000, 213--222.

[63] **T. K. Dey and J. Giesen**. Detecting undersampling in surface reconstruction. Proc. 17th Ann.Sympos.Comput. Geom., (2001), 257--263.

[64] **T. K. Dey and S. Goswami**. Tight Cocone: A water-tight surface reconstructor. Journal of Computing and Information Science in Engineering, Vol. 3 (2003), 302--307.

[65] **J. C. Carr, R. K. Beatson, J. B. Cherrie, T. J. Mitchell, W. R. Fright, B. C. McCallum, and T. R. Evans**, Reconstruction and Representation of 3D Objects with Radial Basis Functions, Computer Graphics (SIGGRAPH '01) 2001, pp. 67-76

[66] **M. S. Floater and M. Reimers**, Meshless Parameterization and Surface Reconstruction, Computer-Aided Geometric Design (2001), Volume 18, pp. 77-92

[67] http://www.alias.com/eng/support/studiotools/documentation

[68] http://www.acm.uiuc.edu/sigmil/RevEng/

[69] **Warren,** J. Subdivision Methods for Geometric Designng, Unpublished manuscript, November 1995.