박사 학위논문
Doctoral Thesis

우선순위를 갖는 Marching Cube 8진트리를 이용한 3차원 물체의 효율적 상세화 모델 생성방법

Efficient Level-of-Detail Model Generation of 3D Objects using Prioritized Marching Cube Octree

이 하 섭(李 河 燮 Hasup Lee)

전자전산학부 전산학전공
School of Electrical Engineering and Computer Science,
Division of Computer Science

한 국 과 학 기 술 원
Korea Advanced Institute of Science and Technology

2007

우선순위를 갖는 Marching Cube 8진트리를 이용한 3차원 물체의 효율적 상세화 모델 생성방법

Efficient Level-of-Detail Model Generation of 3D Objects using Prioritized Marching Cube Octree

Efficient Level-of-Detail Model Generation of 3D Objects using Prioritized Marching Cube Octree

Advisor : Professor Hyun Seung Yang

by

Hasup Lee
School of Electrical Engineering and Computer Science,
Division of Computer Science
Korea Advanced Institute of Science and Technology

A thesis submitted to the faculty of the Korea Advanced Institute of Science and Technology in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the School of Electrical Engineering and Computer Science, Division of Computer Science.

Daejeon, Korea
2007. 4. 23.
Approved by

_____
Professor Hyun Seung Yang
Advisor

우선순위를 갖는 Marching Cube 8진트리를 이용한 3차원 물체의 효율적 상세화 모델 생성방법

이 하 섭

위 논문은 한국과학기술원 박사학위논문으로
학위논문심사위원회에서 심사 통과하였음.

2007년 4월 23일

심사위원장 양 현 승 (인)

심사위원 오 영 환 (인)

심사위원 이 흥 규 (인)

심사위원 김 병 국 (인)

심사위원 김 호 준 (인)

DCS
975310

이 하 섭. Hasup Lee. Efficient Level-of-Detail Model Generation of 3D Objects using Prioritized Marching Cube Octree. 우선순위를 갖는 Marching Cube 8진트리를 이용한 3차원 물체의 효율적 상세화 모델 생성방법. School of Electrical Engineering and Computer Science, Division of Computer Science. 2007. 71p. Advisor Prof. Hyun Seung Yang. Text in English.

Abstract

A level-of-detail (LOD) modeling in computer graphics is to control the number of polygons in mesh at run-time to reduce the rendering cost of small, distant, unimportant geometry. In this paper, we propose the marching cube octree as a scheme for representing and generating the mesh of various level-of-details efficiently.

In the surface reconstruction, modified marching cubes are used for reconstructing the initial mesh. Instead of a midpoint which is used in the original algorithm [57], the ratio of adjacent vertex is used for the triangulation. The results are also feasible when low resolution data is given. These marching cubes are used for the leaf nodes of the marching cube octree.

The marching cube octree is constructed using an octree structure with the marching cubes. To support continuous LOD between levels the priority numbering is proposed. And all nodes are numbered by proposed algorithm. Using adjacent node check and priority queue, node priority numbering can traverse both breadth and depth.

The LOD meshes are generated at run-time using proposed efficient algorithm. It triangulates only needed nodes of the marching cube octree to cover up the whole surface of mesh. Using the priority numbers on nodes and flagging, the LOD mesh can be generated by only referencing without floating point operations which the other LOD models need.

빈 면

Contents

List of Tables

List of Figures

Chapter1. Introduction

1.1 Research Background

In computer graphics, there is always the tradeoff between realism and speed. For the realistic scene, the more detailed model is used but the rendering time is slower. For the fast scene generation, the less detailed model is used and the rendering time is reduced but the less realistic the scene is.

In 3D (three-dimensional) graphics systems like virtual reality, mixed reality, arcade games and etc., objects are represented by a set of triangles, called a mesh. Meshes constructed from real 3D objects are widely used. Meshes generated with range sensors are constructed with plenty of triangles to describe complex 3D objects in detail. The objects can be described in detail when many triangles are used, although the rendering process is slow. Objects can be rendered faster when small triangles are used, but they will then be described coarsely.

Although computers are fast becoming more and more powerful, they are still not enough to render complex 3D objects in real time. Because the number of triangles in the model has increased rapidly, the latest hardware can not afford to handle these complex models effectively. For example, in The Digital Michelangelo Project [1], the number of polygons in the raw dataset of Michelangelo's David, the left picture of (Fig. 1.1), is about 2 billions. The lossless compressed size of it is about 32 GB. Another example is the dataset of The Great Buddha Project [2] in the right picture of (Fig. 1.1). The number of polygons is bigger than a million.

Fig. 1.1: Large polygon models

(Left: Digital Michelangelo Project [1], right: Great Buddha Project [2])

We should generate and use meshes that fit the performance of the rendering system. System performance is changed continuously because the number of objects or the complexity of the background scene changed. If we can control the LOD (level-of-detail) of the object, we can construct an effective computer graphics system. LOD modeling consists of the representation of the mesh and the algorithm to generate the mesh of a certain LOD.

1.2 Problem Definition and Issues

The concept of LOD is showed in (Fig. 1.2). There are three rendering meshes of Woman model from Cyberware™. The left one is the less simplified and the right one is most simplified. In the lower part of (Fig. 1.2), when the model is farther from the viewer, the most simplified one is used for rendering.

The more simplified the model is, the fewer triangles used in rendering and the rendering speed is faster.

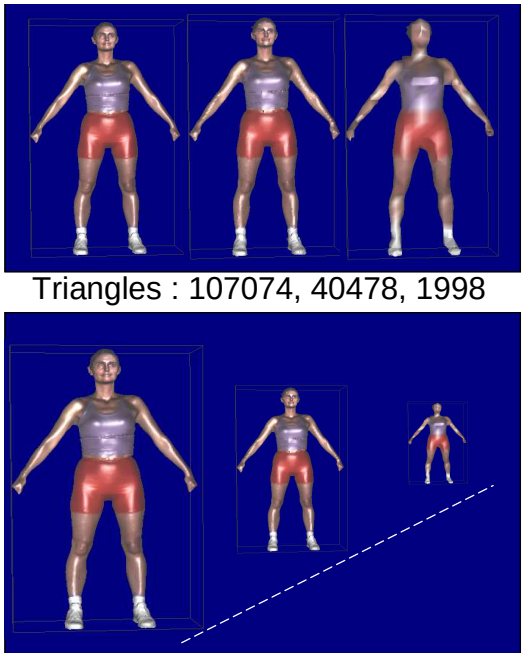


Fig. 1.2: The concept of LOD

(Upper: mesh is simplified, lower: creating LODs of small, distant, unimportant geometry)

The simplified mesh is used not only distant position but also in small or unimportant portions of the scene. When we handle a few high-complexity objects in CAD or large assemblies of many small objects, the simplified model is needed to reduce the rendering cost too.

Controlling the number of polygons in the mesh is known as LOD modeling and it has been a hot issue in computer graphics field for the decade. In the early stage, only the simplification of model just generated from range image had been researched and the result is applied for only static discrete LOD. But recently the control of LOD has been studied and they can adjust LOD at runtime.

For given point set P, normal set N and some LOD number l, the problem is defined

as finding a certain function *Fl* that returns triangle sets satisfying LOD degree *l*.

$$P = \{p \mid p \text{ is input point from range scanner}\}$$
$$N = \{n_p \mid n_p \text{ is normal vector of } p, p \in P\}$$
$$l = [n_0, n_1]\, n_0, n_1 \in \mathrm{N}, n_0 < n_1$$

$$Fl(l, P, N) = \{t \mid t \text{ is triangle satisfying LOD } l \text{ generated from } P, N\}$$

The LOD modeling consists of the representation (data structure) of the mesh and the algorithm to generate the mesh of a certain LOD to implement the function *Fl*.

Instead of range images, if the input data are mesh format, the definition can be changed as follows,

$$V = \{v \mid v \text{ is vertex of mesh}\}$$
$$T = \{t_p \mid t_p \text{ is triangle set of } v, v \in V\}$$
$$l = [n_0, n_1]\, n_0, n_1 \in \mathrm{N}, n_0 < n_1$$

$$Fl(l, V, T) = \{t \mid t \text{ is triangle satisfying LOD } l \text{ generated from } V, T\}$$

Previous methods of LOD modeling are mostly concentrated on a precise description of the original object. The creations of the representations are very complex or expensive and the mesh generation is too slow to be used in practice. Most commercial real-time systems have their own model. In a real-time system, user interactiveness is more important than the precision of the description.

Since the fundamental research [3] on LOD modeling, there have been many issues studied. They are surveyed well in [4]. Some of them are basic issues of computer graphics and some of them are only specific to LOD modeling. The geometric accuracy and the mesh generation time have been researched massively and are relatively important but other issues are not ignorable.

- Geometric accuracy

This is the question “How accurate or similar to the original mesh (initial mesh) geometrically the output mesh generated from LOD modeling is?” There are many geometric attributes from the topology to 3D coordinates of vertices. The mesh simplification method is also important for this purpose. The methodology of simplification is most characteristic feature of LOD modeling. The geometry removal method has the advantage of preserving the geometry information. If a proper fitting algorithm is used, the adaptive subdivision method can also preserve that.

- Visual fidelity

The visual fidelity is the problem of human perception. It is much difficult to measure than geometry. Some methods consider the human’s perception mechanism in graphical devices. When the visual fidelities are similar, the difference of the geometrical accuracy is not significant. The dominant features like the textures are needed to consider in LOD modeling for the visual fidelity.

- Preprocess time

The preprocess time is seemed to be not so important to the application which use the results of LOD modeling. But, for example, when the CAD using model designing that create LOD model and need to see the result several times in middle of the stages, the preprocess time is very important. When the numbers of the objects are large like in museum, the preprocess time is preferred to fast.

- Mesh generation efficiency

The mesh generation is not the same as the rendering time. But it is an important factor in system performance. The LOD mesh generation method is divided into two categories; a discrete (static) LOD and a continuous (dynamic) LOD.

The traditional mesh simplification methods can be applied to the discrete LOD directly. These methods can create multiple versions of every object, each at a different level of detail, in the off-line preprocess time. They can offer proper version of the object by switching the preprocessed models. The more models are, the bigger the memory is used. The memory and the granularity are a trade-off relation in this approach.

In a continuous LOD modeling approaches, the continuous spectrum of the LOD of the object can be generated at runtime. It is more important that the algorithm is fast and efficient in that case.

- Error metrics

There is no standard error metrics for LOD modeling. Each approach has its own error metric which has the advantages and the disadvantages. All the metric are a quantitative but the perceptional correctness is not guaranteed. Measurements about a human perception are hard to evaluate. The scope of metric depends on which operator is used in the modeling, local or global.

- Other issues

In practically, it is important to the LOD model can be easily implemented. To easy implementation, the codes of data structures and algorithms must be simple. It is not always easy to code that the idea is conceptually clear.

The complete automation of the LOD control of the scene is the factor to consider too. For this ends, the levels of detail of meshes are determined automatically by many parameters. But many factors like the size of a display device are dependent to various user environments.

1.3 Research Objective and Outlines

In this paper an efficient object model which is used for computer graphic system is developed. A system performance can be better by reducing rendering time. The LOD modeling of 3D object can reduce the rendering time efficiently. Among the issues of researches, our method was focused at the time of the LOD model and mesh generation.

There are three stages for the LOD modeling. Marching cube octree, data structure for LOD modeling, is constructed from range images or initial mesh. Then node priorities are number to each one for a continuous LOD. Finally, LOD meshes are generated using efficient algorithm which is referring only needed nodes at runtime.

In the marching cube octree construction stage, modified marching cubes are used for the nodes of octree. In the original marching cube algorithm [57], the vertices of triangles are placed at midpoint of the edge of the cubes. Instead of a midpoint, the ratio of adjacent vertex is used for the triangulation [5]. The octree with marching cube node is studied in [62] for mesh simplification. They use the original marching cube structures and no consideration for LOD. They can merge the surfaces that triangulated from the particular marching cubes because the normal and the vertices are limited.

After marching cube octree construction, whole nodes are numbered by priorities. The node priority numbering is needed to support continuous LOD between levels. Each level has the marching cube of double size in one direction. Using adjacent node check and priority queue, node priority numbering can traverse both breadth and depth.

LOD meshes are generated at runtime using efficient algorithm. It refers only needed nodes of the marching cube octree to cover up the whole surface of mesh. Using the priority numbers on nodes and flagging, the LOD mesh can be generated without floating point operations which the other LOD modeling should do.

The contributions of this paper are as follows; to support a continuous LOD,

modifying the marching cube octree (vertex ratio on edge of marching cube) and priority numbering the nodes with an algorithm that can simplify flat area more. An efficient LOD mesh generation which refers only needed nodes without floating point operations.

In the next chapter, the previous researches are introduced. Discrete LOD modeling, which is called mesh simplification generally, and continuous LOD modeling are surveyed. The marching cube and an octree are introduced too. In chapter 3, an overview of the system is presented by a flow diagram and a framework diagram. An initial mesh is reconstructed using the modified marching cube algorithm in chapter 4. In chapter 5, the marching cube octree is constructed and its nodes are numbered in priority. Along with the marching cube octree, online LOD mesh generation framework is described and the implemental results are presented in chapter 6. In chapter 7, the discussion with the comparison to other research is presented. Finally, summary and future works are outlined in chapter 8.

Chapter2. Previous Research

2.1 Level-of-detail Modeling

In 1976, Clark [3] presented a fundamental works in a computer graphics field. These are also the basic principles of level of detail. A redundancy for few pixels was addressed and a hierarchical scene graph was proposed for view frustum culling. It can be said a historical origin of computer graphics.

The earliest application of LOD modeling was flight simulators [7]. Several detailed objects were created by hands. In 1990s, lots of researches had burst out. A mesh was simplified by vertex decimation in [8]. A vertex clustering within grid was proposed in [9]. In [10], an optimization based predictive schedulers for selecting LOD was researched. For continuous LOD, progressive mesh was proposed in [26]. After the progressive mesh, the researches became more various.

Vertex hierarchies for view-dependent LOD were studied in [32] [11] [12]. Quadric error metrics measuring simplification error was proposed in [13]. Guaranteed bounds on surface and texture distortion were developed in [14] [15]. In [16] [17], a principled simplification of topology was addressed.

There is well organized book ‘Level of detail for 3D graphics’ [6]. To study more about LOD modeling, it can be a good manual. According to the survey paper [18], there are three main categories of researches on LOD modeling by geometric criteria: 1) Algorithms that adaptively subdivide an existing mesh with multi-resolutional approaches, [24] [25] and 2) Algorithms that remove geometry features such as vertices, edges, etc., approximate the most detailed mesh, and construct parameterization [26] [27]. 3) Algorithms that produce the simplified model with the sampled geometry. The simplification methods are also well surveyed and organized in [19].

2.1.1 Adaptive Subdivision

An adaptive subdivision method starts from a very simple base model. It continues subdividing and adding some details to it recursively. If the method is a discrete LOD modeling, it stops when the error exceed some user tolerance. For a continuous LOD, it saves the details in each step and produces the LOD mesh using them at runtime.

Many adaptive subdivision methods have been researched since the fundamental work by [20]. The concept of the subdivision and adaptation was proposed and it is the basic idea of the computer graphics. About a terrain data, In the 3D flight simulator application, the triangle bintree is used for LOD modeling of a terrain in real-time is proposed in [21].

The wavelet decomposition approach is very useful because one of its features is multi-resolutional. But a hierarchical decomposition is needed and is not so efficient computationally. The wavelet approaches that using regularly gridded meshes are proposed in [22] [23].

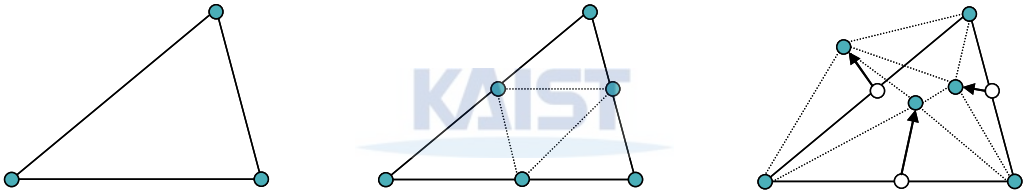


Fig. 2.1: Wavelet decomposition approach

In the field of the continuous LOD modeling, an adaptive subdivision and analysis method that applies wavelet-based multi-resolution analysis to an arbitrary topology surface was proposed [24] [25]. The basic concept of wavelet-based decomposition approaches is showed in (Fig. 2.1). This method can perform smooth parameterization at any LOD and can be applied to adaptive simplification, compression, progressive transmission and editing [28] [29] [30]. The wavelet-based method makes some of these advantages possible. Nevertheless, this method renders the making of the base mesh expensive and slow. Too many triangles are

needed and generated when resolving small local features.

To overcome these drawbacks, a new algorithm, MAPS, was proposed [31]. The MAPS algorithm uses hierarchical simplification, defined by vertex removal, flattening and retriangulation, to induce a parameterization of the original mesh over a base mesh. Although this method can reduce the complexity of the base mesh formulation and resolve small features well, it cannot support view dependency rendering and collision detection, which are important in computer graphics systems.

2.1.2 Geometry Removal

A geometry removal method starts from the original model. It continues selecting geometries to be deleted and removing. This geometry can be vertices or faces or edges or combinations of them. For a discrete LOD, it stops when the produced mesh meets the user-dependent degree of error. If it is a continuous LOD modeling, the reverse operation of the remove is stored and used for generate the LOD meshes.

Merging coplanar or almost coplanar facets and re-triangulating approaches are proposed in [32] [33] [34]. The merging is done by a co-planarity test. By bounded approximations and more robust re-triangulations, the 'superface' improves these approaches in [35].

The vertex, edge or face decimation methods remove the geometries iteratively under local error criteria. These approaches preserve mesh topology generally. The 'mesh decimation' method removes the vertices by a distance or angle criterion in [8]. The resulting holes are patched by re-triangulation. The candidate vertex is chose by a local error test. The simplification methods by collapsing edges [36] [37] [38] and faces [39] are also researched.

The decimation approaches supporting the global error control have been proposed. The 'simplification envelope' method presents bounded error control using the offset surfaces in [14]. The heuristic approaches of evaluating the global error when the vertices are removed and re-triangulated are proposed in [40] [43] [37] [36]. These methods are all under an

incremental simplification framework. The edge flipping while modifying re-triangulated patches for improving approximation accuracy are proposed in [41] [42]. These decimation approaches are extended to 3D simplicial decompositions (tetrahedral sets) in [44] [45] [46].

The vertex clustering approach that clusters vertices and merges into a new representative vertex is proposed in [9]. This method is efficient but can not preserve the topology and a shape of small area. The qualities of visibility and geometry have been improved in [47]. There is some perceptual degradation in this approach. For improving visual quality, couples of edges in the cluster are merged in [48]. To evaluate an error efficiently to these approaches, the 'quadric error metric' is proposed in [13]. This method can simplify disconnected or non-manifold meshes.

An energy function is defined in the 'mesh optimization' [49], to measure the quality of each simplified meshes. Collapsing, swapping or splitting is allowed on mesh edges to optimize the energy function. The geometry of the lowest increase in the function is removed.

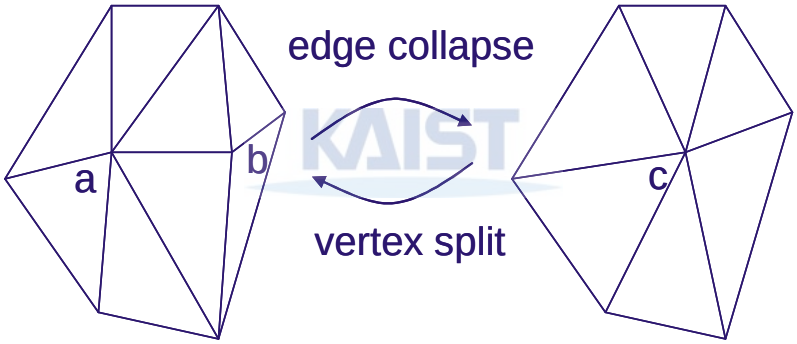


Fig. 2.2: Progressive meshes

For the continuous LOD modeling, another algorithm called 'progressive mesh' was proposed that makes the new mesh by defining the edge collapse and the vertex split operation, which illustrated in (Fig. 2.2), and applying these to the detailed mesh [26]. In addition, a new format was developed for saving and transmitting the triangulated geometric model [27]. The Progressive Mesh method can be applied to adaptive simplification, compression, progressive

transmission and view dependency rendering [50]. The model generation is relatively slow, however, because the simplification is based on the energy function.

Triangle strips with edge collapse operator is used for real-time LOD mesh generation in [51]. The view-dependent meshing of part of the whole data is proposed in [52]. The whole data is in external memory and only part of it is loaded.

2.1.3 Sampling

A sampling method starts from the original model. It samples the geometry and fits the sampled data. For the discrete LOD modeling, the number of the sampled or the size of sampling grid is fixed. It can be controlled when the continuous LOD modeling is applied.

Randomly new vertices are inserted and then moved on the mesh surface to be displaced over maximal curvature areas. The old vertices are deleted and the new re-tiled mesh is generated from the new vertices in [54].

An intermediate octree representation is used for producing simplified boundary representation of mesh in [55]. An intermediate hierarchical representation based on voxels with adaptive surface fitting was proposed in [56] [16].

2.2 Surface Reconstruction

There are several ways of generating mesh from multiple range data. Using the divide-and-conquer method, the marching cube algorithm positions the cubes consisting of eight vertices on the surface and proceeds cubes to make polygons in [57]. Several range data are combined to generate a single mesh using registration of range images and mesh zippering in [63]. The volumetric representation including a weighted signed distance function was defined and used to make the mesh in [59]. Mesh generation methods from the range images proposed so far can be classified into the following two.

One type deals with each point of the range image one by one, and the other adds up the meshes into one whole mesh after constructing the mesh from each of the scanned range images in [63] [57]. The latter makes use of the connectivity and topology between the points to easily make the mesh from one range image. But to generate a whole mesh from these meshes such methods as high-cost zippering are needed. The former one deals with all point independently without zippering. This method can easily add new points to the mesh but has difficulty estimating the connectivity and topology of the points.

Taking advantage of these two methods, we propose here an improved algorithm that exploits information from range scanning based on the former method. We use the marching cube to generate mesh from a range image. The marching cube was proposed and used to construct 3D models from medical images like CT and MRI images in [57]. It is used to construct 3D meshes from unorganized points in [58].

2.2.1 Marching Cube

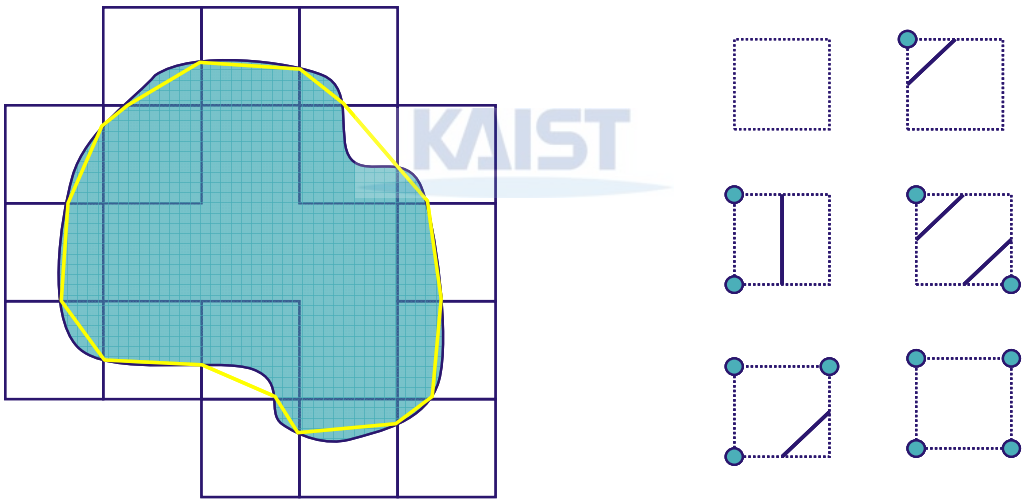


Fig. 2.3: Marching squares (2D)

Marching cube was proposed in [57] and it is an algorithm of a surface construction from 3D medical data. The 3D medical data are volumetric data which are sometimes called voxels. The surface generated from this algorithm is an isosurface. Isosurface is a surface of a constant value corresponding to isocontour in 2D. The output of the algorithm is used for a visualization of 3D data. The ambiguity problem of the marching cube is solved in [90]. The algorithm modified for a surface construction of range data in [58].

The algorithm proceeds through the data field with an imaginary cube. In 2D data, the cube is corresponding to the square. For convenient we call it 'marching square'. The right side of (Fig. 2.3) is the configuration of the marching square. Considering symmetry and rotations, the total number of the configuration is only 6. A dark circle on the vertex means an 'in' configuration which is inside of the area and a vertex without a circle means an 'out' configuration which is outside of the area.

The left side of (Fig. 2.3) is an example of the algorithm. For the area surrounded by a curved line, we can get a boundary approximation. This is the collection of the line segment which is produced by each imaginary marching square. A three-dimensional analog of the boundary line segments is a surface triangle. And the marching cubes are the three-dimensional analog of the marching squares.

Considering symmetry and rotations, the total number of the unique configuration of marching cubes is only 15. The marching cubes are illustrated in (Fig. 2.4). The vertex that covered by a small sphere is corresponding to an 'in' configuration which is inside of the 3D object. The vertex without any sphere is corresponding to an 'out' configuration. That is outside of the object.

The surface approximation is gathered by the triangles generated from each marching cube. The surface of the object is usually curved one except the object is CAD product. The result of marching cube algorithm is the set of triangles which forms the whole surface of the object. This set is called 'mesh' in computer graphics.

The advantage of the marching cube is efficiency and parallelism. The marching cube is computationally efficient because it produces the triangles only by referring a look-up table. The operation in each cube is local, so the error in the cube doesn't affect the others. The relatively higher cost operation like zippering [63] isn't needed in the marching cube.

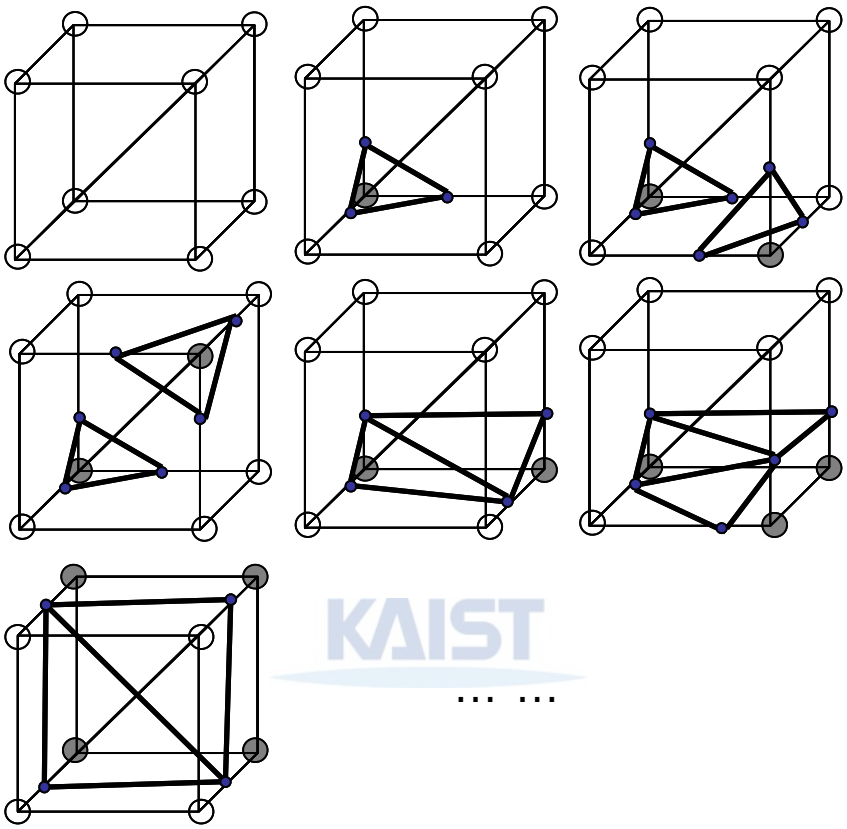


Fig. 2.4: Marching cubes (3D)

We have two problems to run marching cube. The first, we get in/out configuration of the vertex in the marching cubes and the second, the edge intersection between the edge in the marching cube and the surface of the object. In the original marching cube algorithm [57], the input data is array of binary data like CT image. So the in/out configuration is automatically set by the input. And the intersection is fixed to the mid-point of the edge.

2.3 Spatial Indexing

There are several indexing methods for object in space of computer graphics. A mesh (grid) is commonly used for surface data. A quadtree and an octree are for partitioning a 2D and 3D space. A R-tree and UB-tree are used for efficient storing and retrieving of spatial data.

- Mesh (Grid)

A mesh (grid) is a spatial indexing with 2D surface of space using set of cells. Each cell is identified uniquely. The cells can be a square or rectangular, triangular or hexagonal. A hexagonal or other grids including diamond shaped are discussed in [64]. The space-driven or data independent approach in opposition to data-driven method is proposed in [65].

- Quadtree

It is a tree data structure whose internal nodes have up to four children. It used for partitioning a 2D space. The space is subdivided into four quadrants or regions recursively. The Quadtree, also known as Q-tree, was proposed in [66].

The quadtree is useful in image representation, spatial indexing and efficient collision detection in 2D. A view dependent rendering (view frustum culling) of a terrain data is also effective with the quadtree. It is also efficient to storing sparse data like some matrix.

- R-tree

A R-tree is a tree structure that uses the minimum bounding rectangle (MBR) or the bounding box. It is similar to B-trees and researched in [67] [68]. The R-tree stands for range tree. An internal node is identified using MBR contains all entries of its children node. The maximum number of one node should be defined. The R-tree is improved by prioritization in [69]. The R* tree which supports point and spatial data at a time with a few overhead is proposed in [70].

- UB-tree

A UB-tree stands for a universal B-tree and B-tree for a balance tree. It is a balanced tree and can store and retrieve multidimensional data efficiently. It is b+ tree, whose data is stored only in leaf nodes, and the information is stored in Z-order. The UB-tree was proposed in [71].

2.3.1 Octree

An octree is a tree data structure whose internal nodes have up to eight children. It partitions a 3D space by recursive subdividing into eight octants or subspaces, showed in (Fig. 2.5). The octree used for generating isosurface was proposed in [72].

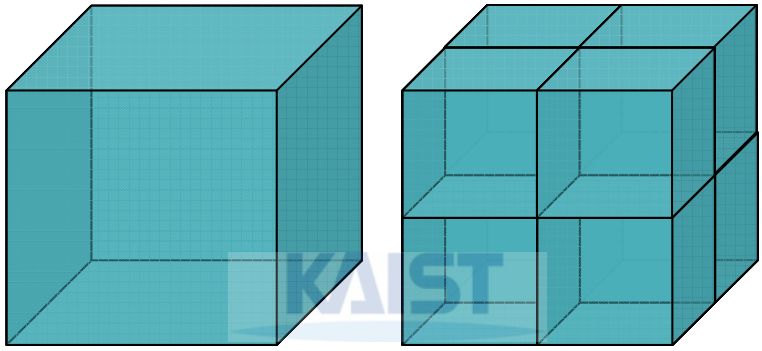


Fig. 2.5: Spatial octree

The octree is very efficient structure in 3D space of the computer graphics. It is useful in the spatial indexing and the collision detection is efficient in 3D with it. The view dependent renderings based on it have been also researched.

- Collision detection with octree

The collision detection is very fundamental problem in the applications of computer graphics like CAD, VR. About 3D collision detection, there is a well surveyed paper [73]

including not only octree but also other methods.

In the approaches of collision detection, the octree is used for spatial partitioning representations. It bounds volumes hierarchically in [74] [75] [76]. At the first, the primitive of the octree is a cube in [77] [78] [79]. The sphere is proposed for the primitive of the collision detection with octree in [80] [81] [82]. And the method whose the branching is 13 instead of 8 is studied in [83]. The modified octree is used in [84].

- View dependent rendering (view frustum culling) with octree

View dependent rendering is also called 'view frustum culling' because it means removing objects that located outside the viewing frustum from the rendering process. A basic concept was proposed in [3] like the level-of-detail concept.

The octree as spatial subdivision used for the view dependent rendering is proposed in [85]. Only the polygons within the viewing frustum are rendered. The hierarchical z-buffer method is researched in [86]. A z-pyramid is used for visibility test and the invisible nodes of an octree hierarchy are deleted. A mixed octree/quadtree structure is developed in [87]. This approach considers not only the location of the observer but also the direction. The octree in the view dependent rendering is improved for 3D navigation, the visibility octree is proposed in [88]. An octant test which can halve the number of the test plane is developed in [89].

Chapter3. System Overview

3.1 Overview

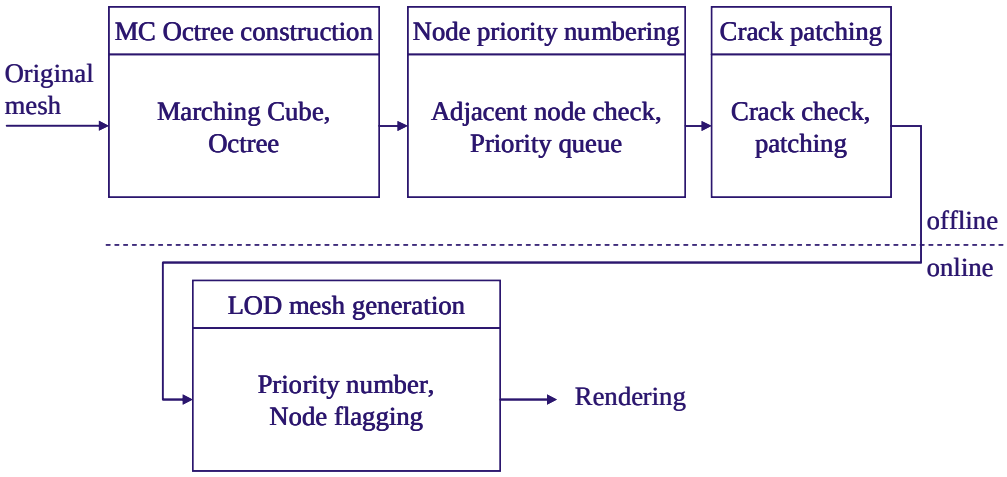


Fig. 3.1: Marching cube octree flow diagram

The input of system is an original mesh or range images. The input data can be obtained from a range scanner directly. Marching cube nodes (terminal leaf nodes) are created from these data. The creation methods are slightly different according to the data. If the input data are the range image, the signed distances are used for configurations of the nodes. If the inputs are the mesh, the configurations of the nodes are calculated using intersection test.

In the marching cube octree construction phase, a spatial octree with the marching cube nodes is constructed. The octree is created with top-down approach by subdividing space recursively but the node configurations – vertices in/out and intersection ratio – are filled up with bottom-up approach from the leaf node which is created from the input data.

In the node priority numbering phase, the priorities of LOD are numbered in each node for a continuous LOD. The numbering is proceeded by adjacent node check with a priority queue. After the priority numbering, the LOD array is generated for fast LOD mesh generation in online.

In the crack patching phase, pre-calculated crack patched vertices are stored. The marching cube nodes which the crack occurs at in run-time are checked and the crack patched vertices are stored.

After the off-line preprocessing, the LOD mesh is generated in run-time. The LOD mesh generation algorithm of this paper is efficient by reference only needed nodes to generate the whole surface of the LOD mesh. It runs with the priority number and the node flagging.

3.2 Framework

The diagram in (Fig. 3.2) describes the relation among the classes. We first get CDataPoint class data from the input data. CreateFromRage / CreateFromMesh module converts this to CCubeNode class data. From CCubeNode class data, MakeTree module makes the marching cube octree using COctCube class data structure. CreateLod module calculates the LOD metrics and adjusted vertices of the octree. SortLod module does the priority numbering and makes the LOD array. Using the octree and the LOD array, MakeMesh module creates the LOD mesh and CalcNormal module re-calculates the normal of the vertices which adjusted because of crack patching.

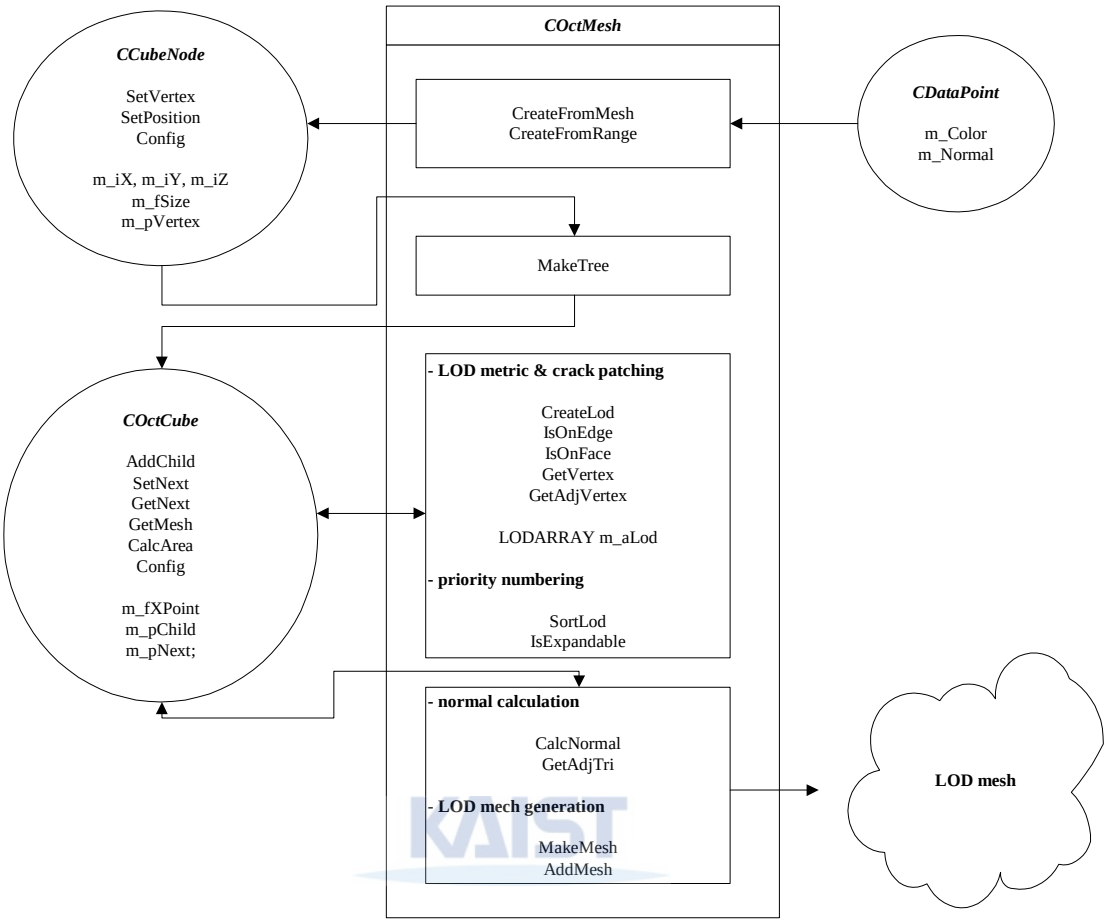


Fig. 3.2: Marching cube octree framework

Chapter4. Surface Reconstruction

4.1 Introduction

Our algorithm was designed to rapidly construct the LOD model and generate the LOD mesh. We approximated the 3D object conceptually with sampling range data in many resolutions. We used the octree and the marching cube to represent the LOD model. The operation necessary for the construction of the marching cube octree is relatively simple. We used the octree that naturally supports progressive transmission, view dependency rendering and collision detection. We did not implement these features yet but octree-based view-defendant rendering has been carried out efficiently by [53]. We can construct the LOD model directly from the range data by using the marching cube data structure, if the Marching-Cube algorithm is applied for the initial mesh generation. Our algorithm directly generates the LOD mesh by referencing only the needed nodes of the tree.

4.2 Marching Cube

The Marching-Cube algorithm for medical images like MRI and CT was proposed [57]. The cube, which includes the in/out configuration of each of the 8 vertices, is classified into 14 distinct cases. Triangles are created automatically for each case. The vertices of the triangles are at the midpoints of the cube's edges. The Marching-Cube algorithm can also be used to generate the mesh from the range data [58].

To generate triangles, we use the sign and the ratio converted from the signed distance of the cube's vertices. The signed distance was proposed for surface reconstruction from unorganized points [58]. It is defined as the distance between the vertex P and the closest range point multiplied by +-1, depending on which side of the surface P is. The sign of the vertex is defined as the sign of the signed distance. The ratio of the edge is defined as the ratio

of the absolute value of one vertex's signed distance to that of the others. We can generate triangles naturally using a low resolution by choosing the proportional point instead of the midpoint for the vertex of the triangles. The comparison of these two approaches is show in (Fig. 4.1). The midpoint is used in (a) and the proportional point is in (b).

(a) (b)

Fig. 4.1: Marching cube: low resolution input data

(a) midpoint and (b) intersection ratio

Using an octree for the representation of 3D object is not new idea. The isosurface generation using marching cubes and octree traversal was proposed in [60]. This study describes efficient creation of octree based representation. But we applied modified marching cube configuration to octree structure, thus can construct hierarchies and implement level-of-detail.

We define a marching cube in this study as the set of signs and colors for each of the 8 vertices, and of ratios for the 12 edges. A marching cube octree is defined as a spatial octree whose nodes are marching cubes. Since the dimension of the root node is known, we can determine the relative position and the size of any node in the octree. A null node is defined

for all edges of the corresponding marching cube that do not intersect with the surface. Thus, all vertex signs are the same. The null node has no child nodes and null points. Finally, the nodes of the marching cube octree correspond only to the region of the surface.

For given point set *P*, normal set *N* and some cube *C* containing vertex v_1, …, v_8, we define the closest point *p*, distance *d* and sign *s* for the vertex configuration *vc* and cube node *CN* as follows:

$$p_n = \text{the closest point to } v_n,\ p_n \text{ is in } C,\ p_n \in P$$
$$d_n = \|v_n - p_n\| \text{ (euclidian distance)}$$
$$s_n = \frac{(v_n - p_n) \bullet n_{p_n}}{\left|(v_n - p_n) \bullet n_{p_n}\right|},\ n_{p_n} \text{ is normal of } p_n,\ n_{p_n} \in N$$
$$(+1: \text{outside}, -1: \text{inside})$$
$$CN = \{vc_m \mid vc_m = s_m \times d_m,\ m \in [1,8]\}$$

By the Marching-Cube algorithm, triangles can be generated easily using *CN* and look-up table.

To apply the marching cube algorithm, the state of the vertices of each cube must be determined. Signed distance has been proposed as a method of determining vertex states [58]. The signed distance *f*(*p*) from point *p* in a 3D coordinate to the known mesh *M* is distance from *p* to the closest point *z* on *M* multiplied by sign *s*. If *p* is inside *M*. *s* is +1, otherwise –1. When data is input from a 3D scanner, an estimated *M* is used because the real *M* is unknown. A tangent plane is calculated for each point *p*, which is approximated to *M*, and the signed distance from *p* to the tangent plane is used instead of the actual signed distance. The approximated signed distance is defined as follows:

$$f(\vec{p}) = dist_i(\vec{p}) = (\vec{p} - \vec{o}_i) \cdot \vec{n}_i .$$

Here, o_i is the center of the tangent plane and n_i is the normal vector of the tangent plane.

The tangent plane is calculated by using the relation between adjacent points because topology information is not used. The available directions of tangent plane are two in this method, so an optimization process is used to choose direction. But we use the topological structure of the data because it is assumed that the input data is a 3D range image. Thus, any point has 4 adjacent points except for the boundary because 3D range images form a lattice. 4 triangles are drawn connecting these points as in (Fig. 4.2). Each triangle's normal vector is calculated by the cross product. The tangent plane normal is the average of these 4 normal vectors and the center point of the tangent plane is the value of the center input point.

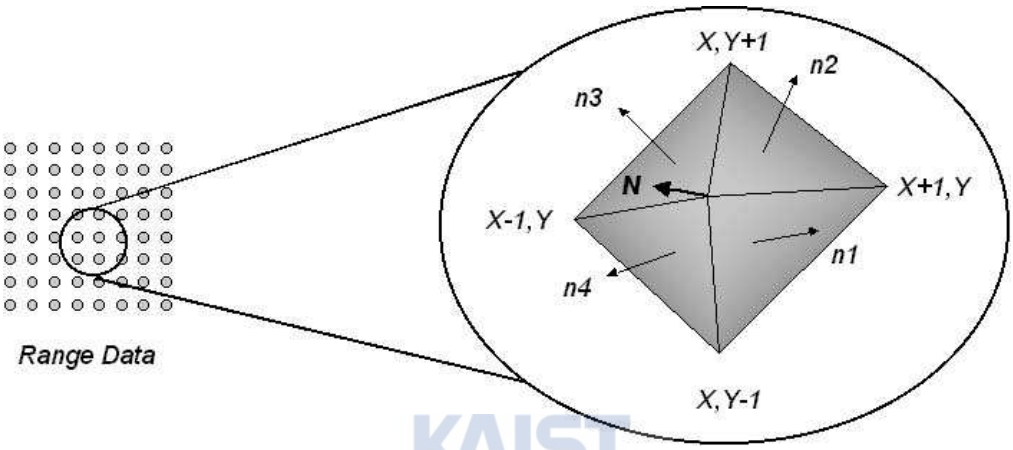


Fig. 4.2: Normal vector calculation of ranger data

Recently the range scanners support the polygon format like 'PLY' as their output. Even when the range image is not used as input, we can run the marching cube algorithm.

First, the vertex in mesh is regarded as the range point and the normal of the vertex is regarded as the normal of the point. Then we apply these to the signed distant algorithm.

Second, the in/out of the vertex of the marching cube is calculated from the cube and the mesh. The intersection on the edge of the marching cube is also calculated from these.

4.3 Algorithm

In this section, we describe about basic data structures and classes of range data, cube node and marching cube node that defined in the previous chapter.

4.3.1 Range Data

DataPoint
Color m_Color Point3D m_Normal

This is a class for the points of input range data defined as point set *P*, normal set *N*. The class Point3D contains coordinates x, y and z. The color variable m_Color is for the example attribute of a data point. The '=' operator is overridden for input data.

If the output of the range scanner is a mesh format file like PLY or if we want to convert other mesh format file to our model's input, the conversion is needed. We took the vertex of the mesh as the point and the average of this vertex's adjacent triangles' normal as the normal of this point, m_Normal.

4.3.2 Cube Node

CubeVertex
ix, iy, iz //index dist //distance flag //sign Color color //color attribute

CubeNode
SetVertex SetPosition Config

m_iX, m_iY, m_iZ; //index
m_fSize; //size of cube node
CubeVertex *m_pV //vertices

This is a class for a cube node *CN* defined as above. The distance and the sign are defined in the structure CubeVertex. SetVertex module is used for input vertex and SetPosition module determines where the node is located. For each range data point inside the node,

Config module examines whether it is the nearest point to each vertex and determine the sign of the vertex. Config module will be called from CreateFromRange module or CreateFromMesh module.

4.4 Results

When the input is the range data, the intermediate results can be showed. The resolution of the initial mesh is determined by user. The resolution means the number of the marching cubes in one of axis (X, Y or Z). For example, if the resolution is 256, the total number of the marching cubes is 256*256*256. Thus, the more the resolution is, the smaller the cubes are and the more detailed the mesh describes. But there can be lots of holes if the range data aren't dense. So the resolution of the marching cubes can not be bigger than that of the range data and by the experiments, the double is proper.

Fig. 4.3: Marching cube: cube's resolution comparison

The marching cube's resolutions are (a) 60 and (b) 200.

(Fig. 4.3) illustrates the comparison results by the resolution. The resolution of (a) is 60 and the number of polygons in mesh is 8926. The resolution is 200 and the number of the polygons is 62124 in (b). The dark spots in the object are the holes that the marching cube can

not located at. When the resolution is higher, the more detailed mesh is obtained but there exist many holes.

(Fig. 4.4) shows the front and the side view of the implemental result. The input data are generated from mesh models instead of range scanner. The generated range images are from front, left 45 and 90 degrees. The front and left side of the model are well constructed but the rear and right side are not so well yet. The total model is constructed when the range data of all directions are given. Because the marching cube method is a local operation and don't need additional zippering, the range data of uncompleted area are added to continue reconstructing the model. Thus, the intermediate results like (Fig. 4.4) are useful to determine needed range images to reconstruct the whole model.

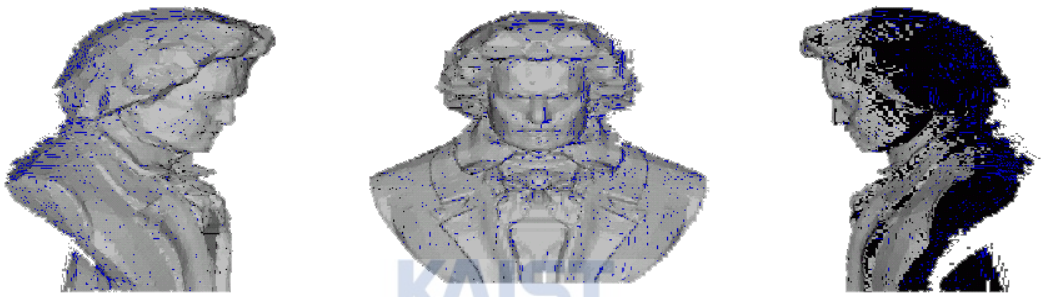


Fig. 4.4: Surface reconstruction: Beethoven

Chapter5. Level-of-detail Mesh Modeling

5.1 Marching Cube Octree

The cube data structure of the Marching-Cube algorithm is basis of the marching cube octree. Instead of signed distance [58], the marching cube octree utilize the ratio on each edge for generating triangles.

For given position vector w, vertex sign s_n and edge ratio er_m and the marching cube node MC containing vertex v_1, …, v_8 are defined as follows:

$$s_n = \begin{cases} true & \text{if } v_n \text{ is inside.} \\ false & \text{if } v_n \text{ is outside.} \end{cases}$$

$$er_m = \begin{cases} 0 & \text{if adjacent vertex's sign is same.} \\ i, i \in (0,1) & \text{otherwise} \end{cases}$$

$$MC = \{w\} \cup \{s_n \mid n = 1, \cdots, 8\} \cup \{er_m \mid m = 1, \cdots, 12\}$$

Then for given some marching cube node r, the following recursive definition is the marching cube octree T_r.

$$T_r = \begin{cases} r & \text{if } r \text{ is leaf node.} \\ r \cup \bigcup_{s \text{ is } r\text{'s child}} T_s & \text{otherwise} \end{cases}$$

The creation of the marching cube octree starts with marching cube nodes of the most detailed resolution (leaf node) and merged into parent's level as (Fig. 5.1). For leaf marching cube nodes, vertex sign is the same as corresponding cube node and edge ratio is obtained from simple divide calculation.

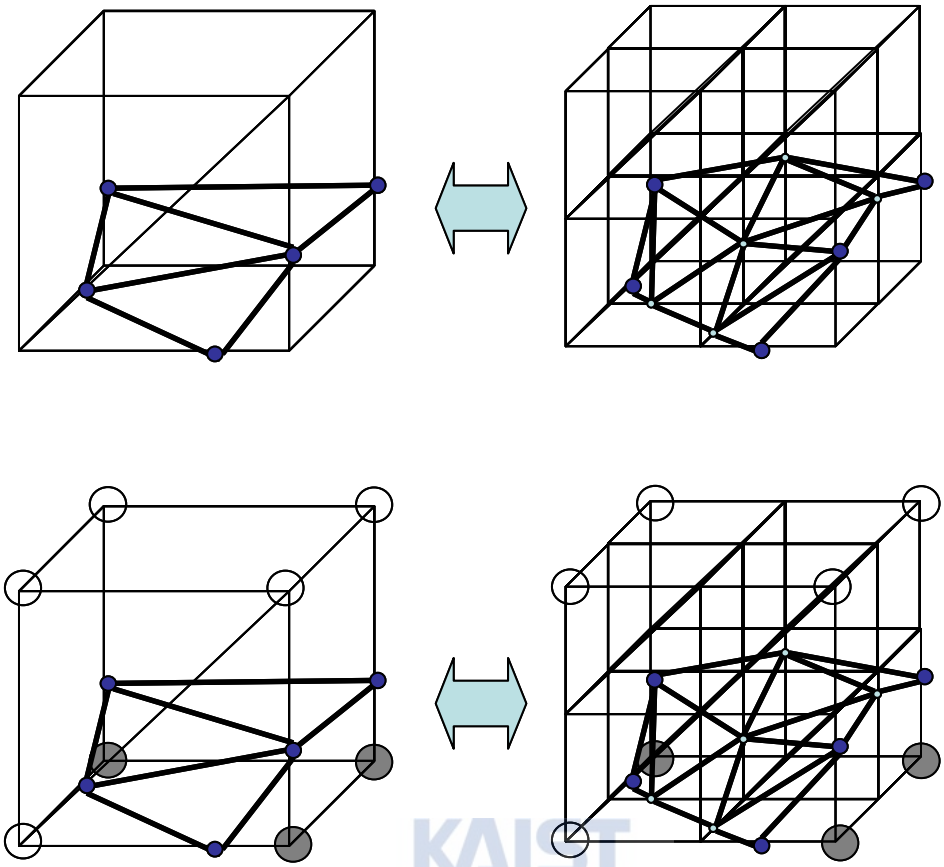


Fig. 5.1: Creation of the parent node

For internal nodes, a parent node can be constructed by referencing its child nodes. The decision on the sign of the vertex is classified into four cases (Fig. 5.2):

1) If a corresponding vertex exists in the child node, the sign of vertex ‘A’ is the same as that of child node.

2) Else if an adjacent vertex exists in the child nodes, the sign of vertex ‘B’ is the same as the adjacent vertex of the child node.

3) Else if a diagonal vertex exists in the child nodes, the sign of vertex ‘C’ is the same as the diagonal vertex of the child node.

4) Else if no corresponding, adjacent and diagonal vertex exists in the child nodes, the sign is the same as that of the center vertex (vertex D).

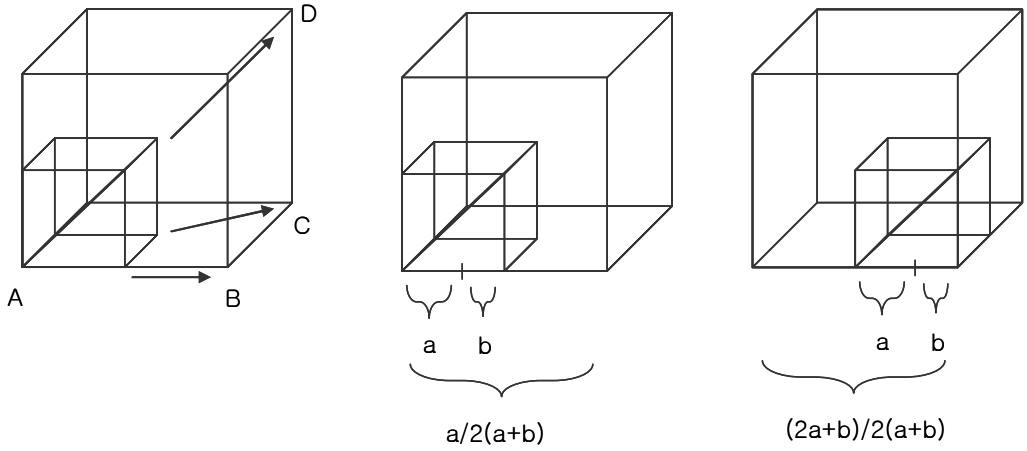


Fig. 5.2: Conversion of the marching cube

In case 2), several adjacent vertices can exist in the child nodes, but their signs will all be the same. The color value is copied in the same manner. The ratio is calculated easily by extending the vertex that has the triangle’s vertex in it (Fig. 5.2). The ratio of the vertex that does not have the triangle’s vertex in it is unnecessary.

The advantage of containing the ratio on edges instead of the distance of vertices is simplifying crack patch problems by sharing the same vertex of triangle generated from the different level marching cube node. Using the origin and the size of node, the parent-children relationship can be calculated but the pointer data structure is used for the efficiency of implementation. The intermediate data structure of our algorithm shows in (Fig. 5.3). The right part of the (Fig. 5.3) is the cube configuration figure.

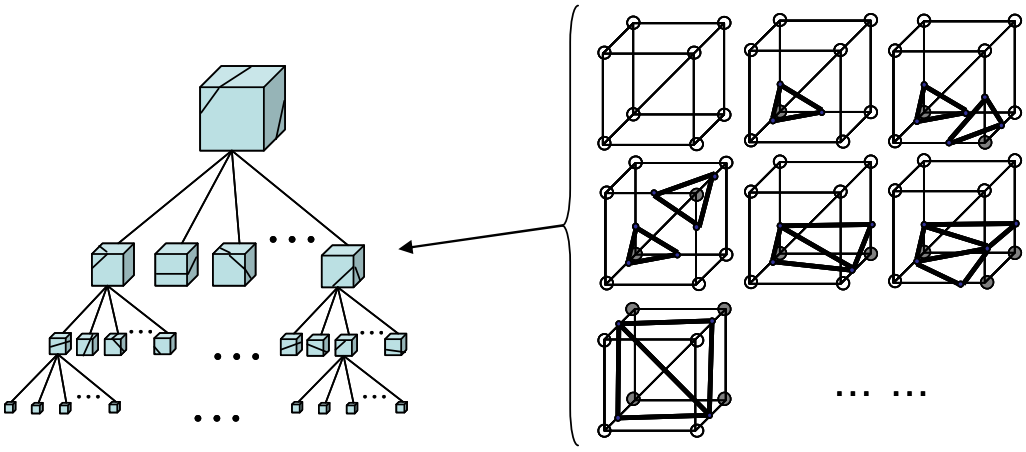


Fig. 5.3: Marching cube octree

5.2 Node Priority Numbering

To control the LOD of the mesh, we assign a priority number to all the nodes. The mesh generated from all nodes of one level has exactly twice the resolution of the mesh from of the parent nodes' level. Thus, only the LOD of the transition from one level mesh to next is needed to be considered. The priority numbering determines which node expands first in the same level.

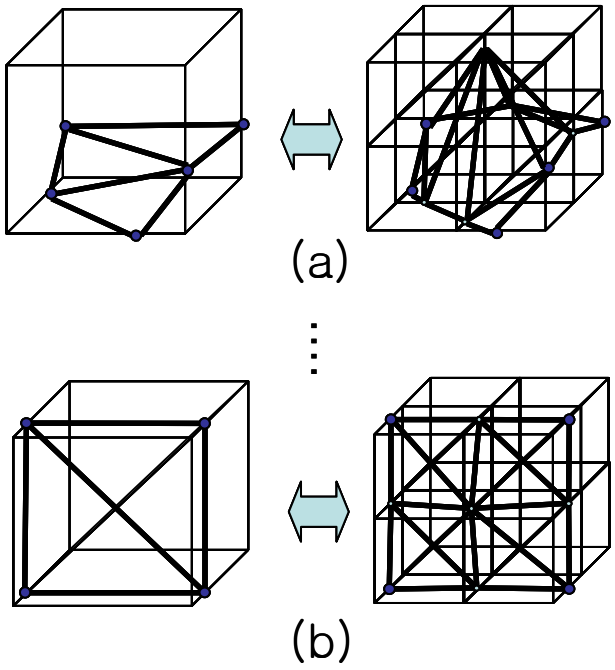


Fig. 5.4: Node priority

The priority of node (a) is higher than (b).

We use the area difference as the LOD metric. For given marching cube node q and r, the area difference M_r and the triangulation function $F_{MC}(q)$ by Marching-Cube algorithm are defined as follows:

$$F_{MC}(q) = \{t \mid t \text{ is triangle generated using Marching - Cube algorithm from } q\}$$

$$M_r = \sum_{s \text{ is } r\text{'s child}} \sum_{area} F_{MC}(s) - \sum_{area} F_{MC}(r)$$

The higher the area difference is, the more detailed is the description of the local feature and the higher the priority is. To describe the 3D object with fewer triangles, the higher priority node expands first in the same level. The example of node priorities comparison is illustrated in (Fig. 5.4).

But to describe the complex parts of the object in detail, we improved some features. This algorithm starts from the root node. The expanded nodes are marked as ‘marked’. When the level differences between some node and its all surrounding nodes are less than 2, that node can be expanded. The node of the biggest the area difference - LOD metric - in the expandable nodes set is inserted in LOD array. Then this node is marked as ‘marked’. This algorithm can be implemented effectively using the priority queue (sorted list). The pseudo code of the node priority numbering is as followed.

```
Insert root node to Slist
(Slist is sorted list by LOD metric)
do
  for node n (n is the 1,2,...,m-th node of Slist)
    if all n's adjacent node is 'marked' or none
      Insert n to LOD array
      mark n's children as 'marked'
      Insert n's children to Slist
      exit loop
until Slist is empty
```

The example of the priority numbering helps the understanding of this algorithm. It is illustrated in (Fig. 5.7) and explained in section 5.4.3.

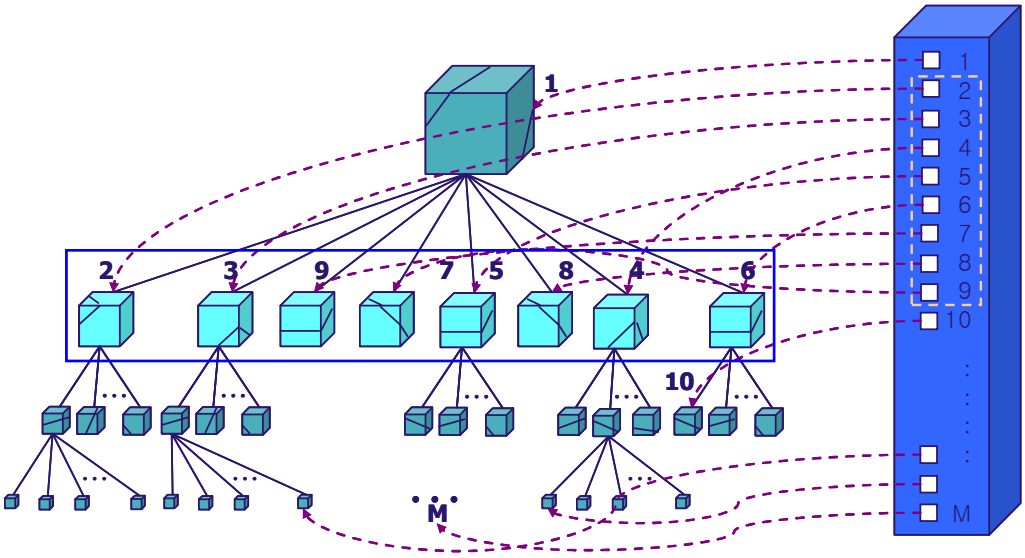


Fig. 5.5: Marching cube octree with LOD array

To generate the LOD mesh rapidly, we save this priority number in a referencing array, the LOD array. The number N means the N-th node to be expanded. Thus, its child nodes are triangulated, as shown in (Fig. 5.5).

5.3 Crack Patching

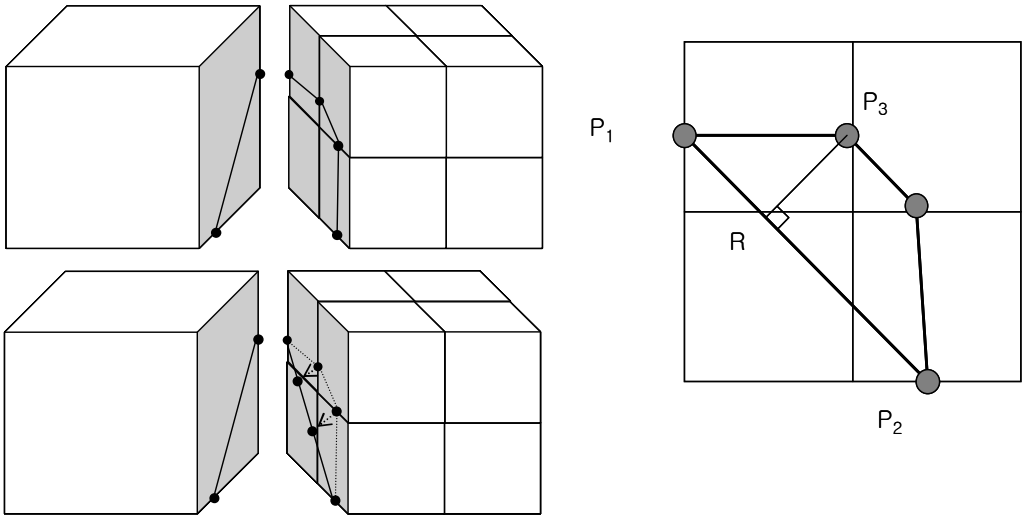


Fig. 5.6: Crack patching

Cracks are generated at the interfaces of nodes with varying levels (the left figure of (Fig. 5.6)). This is a common problem with adaptive subdivision algorithms. Crack patching algorithm was proposed in [62] and we use that in our algorithm. This algorithm can apply the case that adjacent node's level difference is one. For fast generation of the LOD mesh, thus we calculate all compensation points before the mesh is generated. Only the compensation points of crack-happen nodes are calculated and saved. In the right figure of (Fig. 5.6), the compensation point R of $P3$ is a cross point of $P1P2$ and a perpendicular line to $P1P2$ containing $P3$.

$$\overrightarrow{P_1P_2} // \overrightarrow{P_1R}$$
$$\overrightarrow{P_1P_2} \perp \overrightarrow{RP_3}$$

$$P_1 \equiv (x_1, y_1, z_1), P_2 \equiv (x_2, y_2, z_2), P_3 \equiv (x_3, y_3, z_3), R \equiv (X, Y, Z)$$

$$x_2 - x_1 = d(X - x_1)$$
$$y_2 - y_1 = d(Y - y_1)$$
$$z_2 - z_1 = d(Z - z_1)$$

$$(x_2 - x_1)(X - x_3) + (y_2 - y_1)(Y - y_3) + (z_2 - z_1)(Z - z_3) = 0$$

$$X = 1/d(x_2 - x_1) + x_1$$
$$Y = 1/d(y_2 - y_1) + y_1$$
$$Z = 1/d(z_2 - z_1) + z_1$$

The crack is able to patch by moving point *P*3 to *R* and the others to the corresponding point.

5.4 Algorithm

5.4.1 Octree Cube Node

OctCube : CubeNode
AddChild SetNext GetNext

GetMesh CalcArea Config //set internal node's configuration
OctCube* m_pChild //child nodes OctCube* m_pNext //pointer to next (sibling) m_fXPoint //distance ratio on edges

This is a class for the marching cube node *MC* and additional pointers to make an octree. AddChild module adds the child node and SetNext module makes a connection to the sibling node. GetNext module is implemented for retrieval. GetMesh module triangulates the node by referencing the lookup table in Marching Cubes algorithm. CalcArea module calculates the sum of the areas of triangles made by GetMesh module. This is implemented for the LOD metric calculation.

For internal nodes, a parent node can be constructed by referencing its child nodes. The decision on the sign of the vertex is classified into four cases (Fig. 5.2). The attributes are copied in the same manner. The ratio is calculated easily by extending the vertex that has the triangle's vertex in it (Fig. 5.2). This is implemented in COctCube.Config module.

5.4.2 Making Octree

OctMesh
OctCube* m_pRootCube //pointer to root node m_nLevel //level of octree
CreateFromMesh CreateFromRange MakeTree //make tree and set node's configuration

As an initial input for our model, CreateFromRange module makes the cube nodes from the range data. If the input is mesh, CreateFromMesh module is used. From the created cube nodes, MakeTree module constructs the octree consists of the marching cube nodes. This calls Config module of the class COctCube in (6).

The octree is created with top-down approach by subdividing space recursively. But the internal node's configurations which contains vertices in/out and intersection ratio are filled up with bottom-up approach from the leaf node. The leaf nodes are created from the input data.

5.4.3 Priority Numbering

LODARRAY
OctCube* node metric

PriorityNumbering
LODARRAY m_aLod //LOD array
CreateLod IsOnEdge IsOnFace GetVertex GetAdjVertex SortLod IsExpandable

To make a controllable LOD model, we calculated the metrics of each node. CreateLod module is implemented for this purpose. The adjusted vertex for crack patching is pre-calculated and stored in runtime for speed up. IsOnEdge module, IsOnFace module, GetVertex module and GetAdjVertex module are implemented for the pre-calculation.

In COctMesh.SortLod module, this algorithm starts from the root node. The expanded nodes are marked as ‘marked’. When the level differences between some node and its all surrounding nodes are less than 2, that node can be expanded. The node of the biggest the area difference - LOD metric - in the expandable nodes set is inserted in LOD array. Then this node is also marked as ‘marked’. IsExpandable module is implemented for this examination.

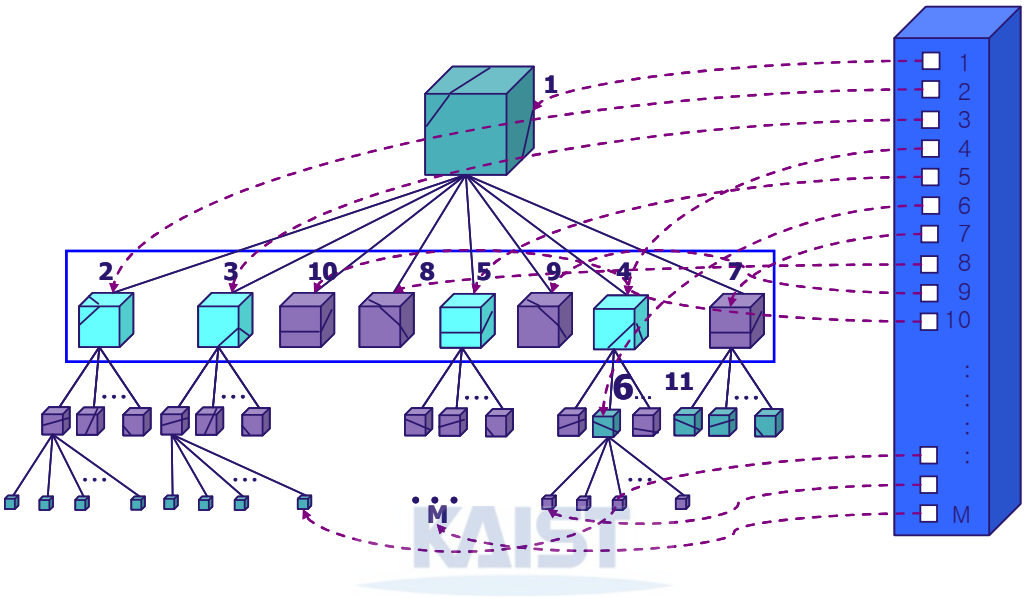


Fig. 5.7: Node priority numbering

Here is the example of priority numbering in (Fig. 5.7). The algorithm begins with putting the root node to the priority queue Q. Then the root node is inserted to LOD array with number ‘1’. The children of the root node are inserted to Q. The LOD number 2, 3, 4 and 5 nodes in the same level are inserted consequently. Then the LOD number 6 node is inserted to LOD array because its LOD metric is bigger than any other nodes in Q and the adjacent nodes of it are all marked as ‘marked’. The algorithm continues until Q becomes empty.

The insertion operation of the priority queue is the same problem ‘insert into sorted list’. The complexity analysis of this operation is explained in the discussion chapter.

5.4.4 Crack Patching

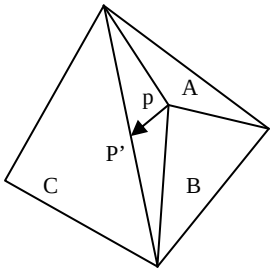


Fig. 5.8: Calculation of the normal of the adjusted vertex

The adjusted vertex for the crack patching is pre-calculated in CreateLod module. If the vertex p is moved to p' for crack patching (Fig. 5.8), the normal of p' is needed to calculate newly. For given adjusted vertex *p'* and triangle *A*, *B*, the normal of *p'* is as follows.

$$n(p') = \frac{n(A') + n(B') + 2 * n(C)}{4} = \frac{n(A') + n(C)}{2} \qquad \because n(A') = n(B')$$

CalcNormal module calculates the normal of the adjusted vertices using above method. GetAdjTri module is implemented for find the adjacent triangle of the adjusted vertex. (ex. Triangle C for vertex p' (Fig. 5.8)).

CrackPatching
CalcNormal //re-calculate normal GetAdjTri

Chapter6. Run-time Framework for Level-of-detail Mesh

6.1 Level-of-detail Mesh Generation

Using the algorithm mentioned above, we constructed the marching cube octree presentation of a certain mesh. In this section, we consider generating the LOD mesh using this representation. We generate the LOD mesh rapidly and efficiently by referencing the LOD array.

The LOD of the LOD mesh is controlled by using the LOD array. In the LOD array, there is a triangulated node sequence of all nodes in the marching cube octree. When the next node is triangulated, the number of triangles in the mesh is increased. We check each node already expanded in its sub-tree on the reverse order sequence from given priority number's node. The pseudo code is written as follows:

```
For all node n (n's priority number is i, i-1, ... , 2, 1)
  If all n's child nodes are 'expanded',
    continue
  Otherwise,
    triangulate n's child nodes except 'expanded'
    mark n as 'expanded'
end of loop
```

The flag 'expanded' are no related to that of node priority numbering.

If the number of triangles in the mesh is given, we convert this input to the priority

number by using the accumulation table. The accumulation table contains a triangle increment by the priority order triangulation. Thus, we can get the approximated priority inversely from the number of triangles. The final data structure of our algorithm shows as followed.

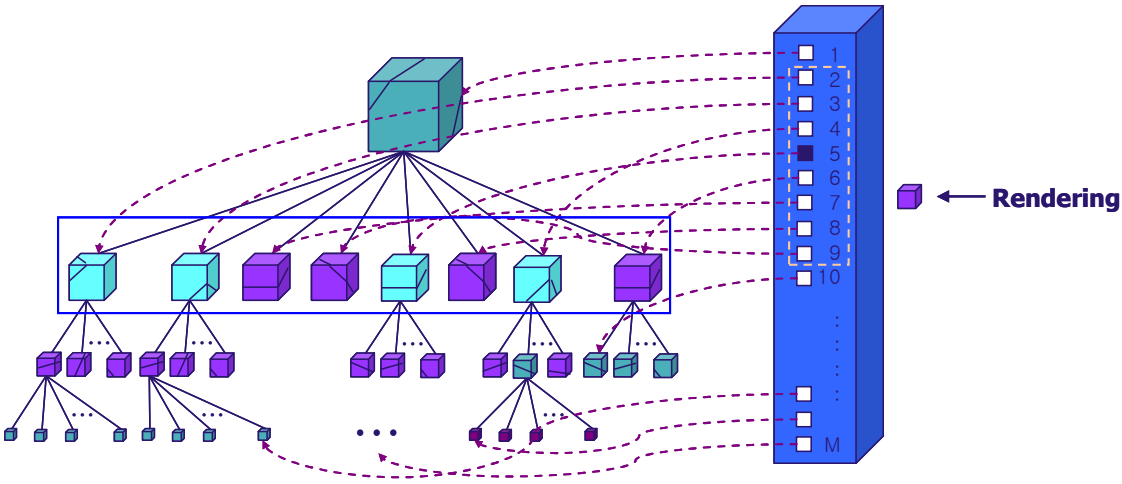


Fig. 6.1: Direct generation of the LOD mesh

6.2 Algorithm

6.2.1 Mesh Generation

The LOD of the mesh is controlled by referencing the LOD array. In the LOD array, there is the sequence of the node to be triangulated. On the reverse order from the given LOD's node to the first node, we examine whether that node's child nodes are all expanded or not. If they are all expanded, it means that the whole area of the object is covered by triangles. COctMesh.MakeMesh module is implemented by this method and AddMesh module makes the mesh structure of OpenGL for rendering.

LODMeshGen
MakeMesh AddMesh

Here is the example of the LOD mesh generation. Let's the input LOD number is 66. First, the children nodes of the LOD 66 node are triangulated and the LOD 66 node is marked as 'expanded'. Then the LOD 65 node's children nodes are expanded and the LOD 65 node is marked. Next, the children nodes of the LOD 64 node are triangulated except the LOD 66 node. And the LOD 64 node is marked. The LOD 63, 62 node are traversed by a similar method. When the LOD 61 node is visited, the LOD 70, 68, 69, 67 node are triangulated.

From the LOD 60 node to the LOD 52 node, the algorithm proceeds in the same way. When the LOD 51 node is visited, the algorithm is continued without any triangulation because the children of the LOD 51 node – the LOD 52, 53, 60, 58, 55, 59, 54 and 57 node - are all marked as 'expanded'. The 'expanded' mark of a node *N* means "the area covered by the node *N* is triangulated by some nodes in the sub-tree whose root node is *N*".

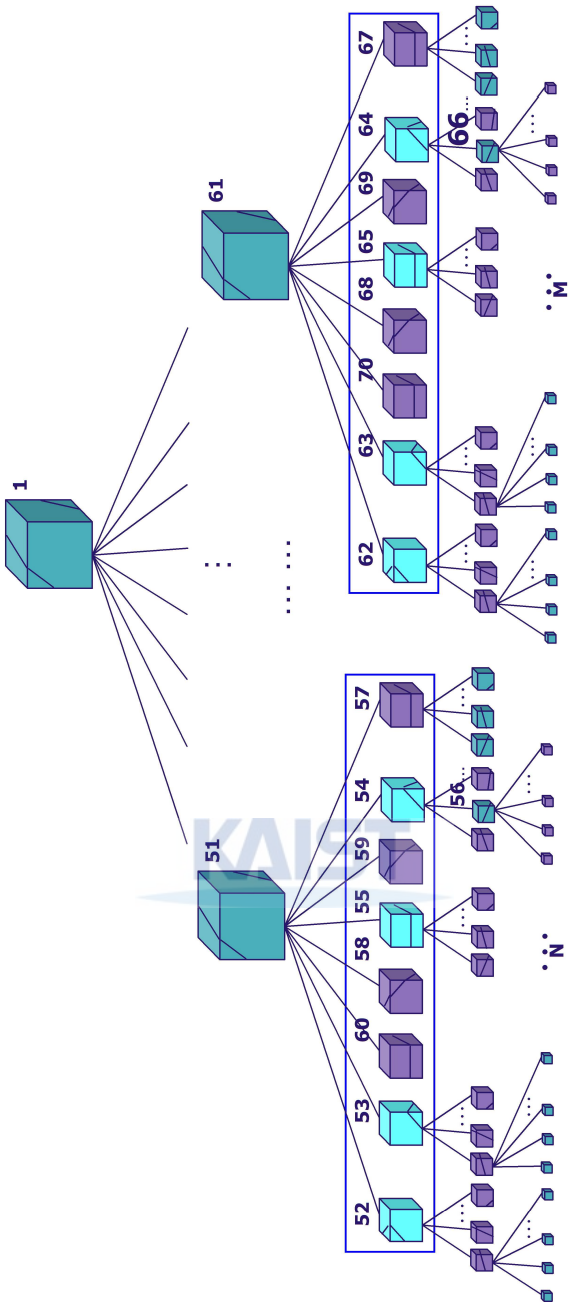


Fig. 6.2: LOD mesh generation example

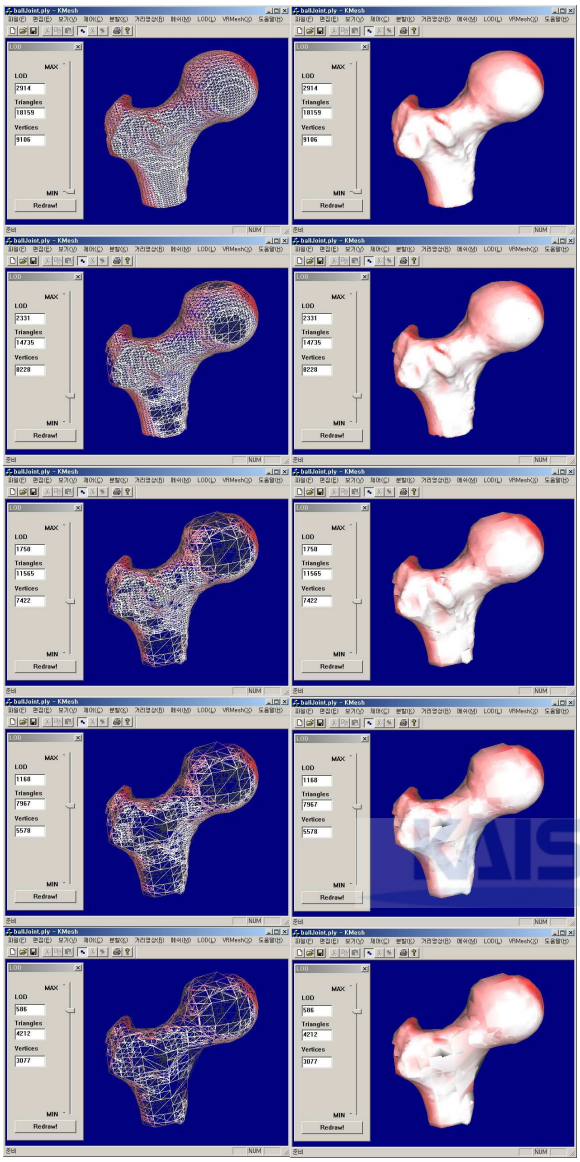
6.3 Results

The hardware specification of implement environment is as follows. The computer is Intel™ Pentium® 4 CPU 2.40GHz. The RAM is 1GB. OS is Microsoft™ Windows® Pro v2002 SP2. The system is implemented under Visual C++® 6.0 with OpenGL.

Recently most of range scanners provide the polygon format data as the results like 'PLY' files. So we generate the range image data from them. The vertices are used as the points and the normal vectors are calculated from neighbor triangles' normal.

The 'ball joint bone' (Fig. 6.3) and the 'Venus statue' (Fig. 6.4) meshes are used for experimental data and they can be downloaded from Cyberware™ homepage. The results are showed with the LOD decreased by 20%. The meshes of the left side are wire frames and of the right side, rendered meshes. The numbers of triangles and vertices are showed in the table on the right.

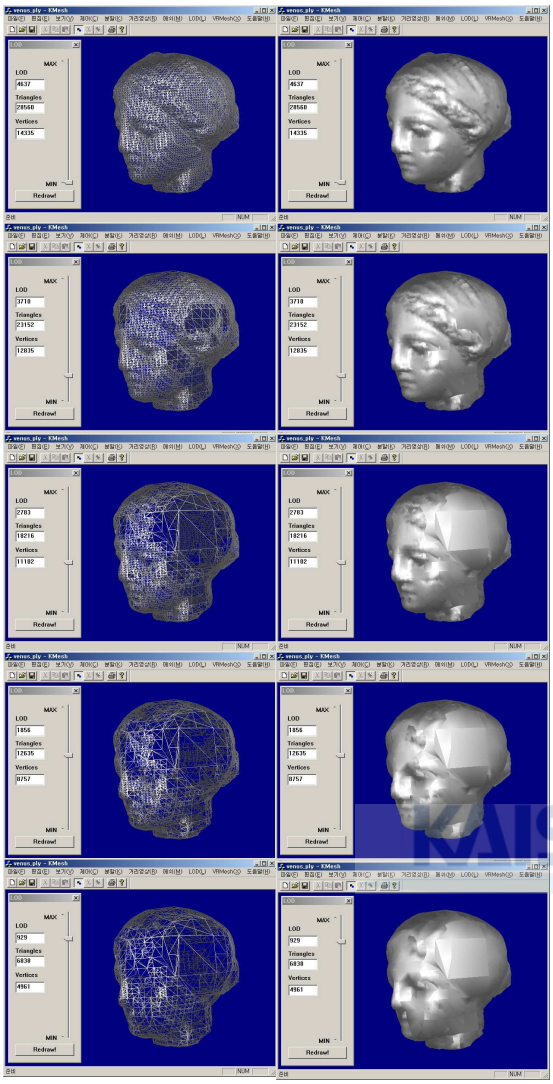
These results show the marching cube octree model is feasible for a LOD representation. In flat area, the number of the polygons is fewer. And in the area have curvatures, the number is more relatively. The upper right part (ball part) of the ball joint bone and the head's side of the Venus statue are coarse in low LOD. But the neck-like part of the ball joint bone and an eye, nose, mouth part of the Venus statue is detailed.



LOD (%)	triangles	vertices
100	18159	9106
80	14735	8228
60	11565	7422
40	7967	5578
20	4212	3077

Fig. 6.3: LOD mesh generation results: ball joint bone

(Dataset courtesy of Cyberware™)



LOD (%)	triangles	vertices
100	28560	14335
80	23152	12835
60	18216	11102
40	12635	8757
20	6838	4961

Fig. 6.4: LOD mesh generation results: Venus statue

(Dataset courtesy of Cyberware™)

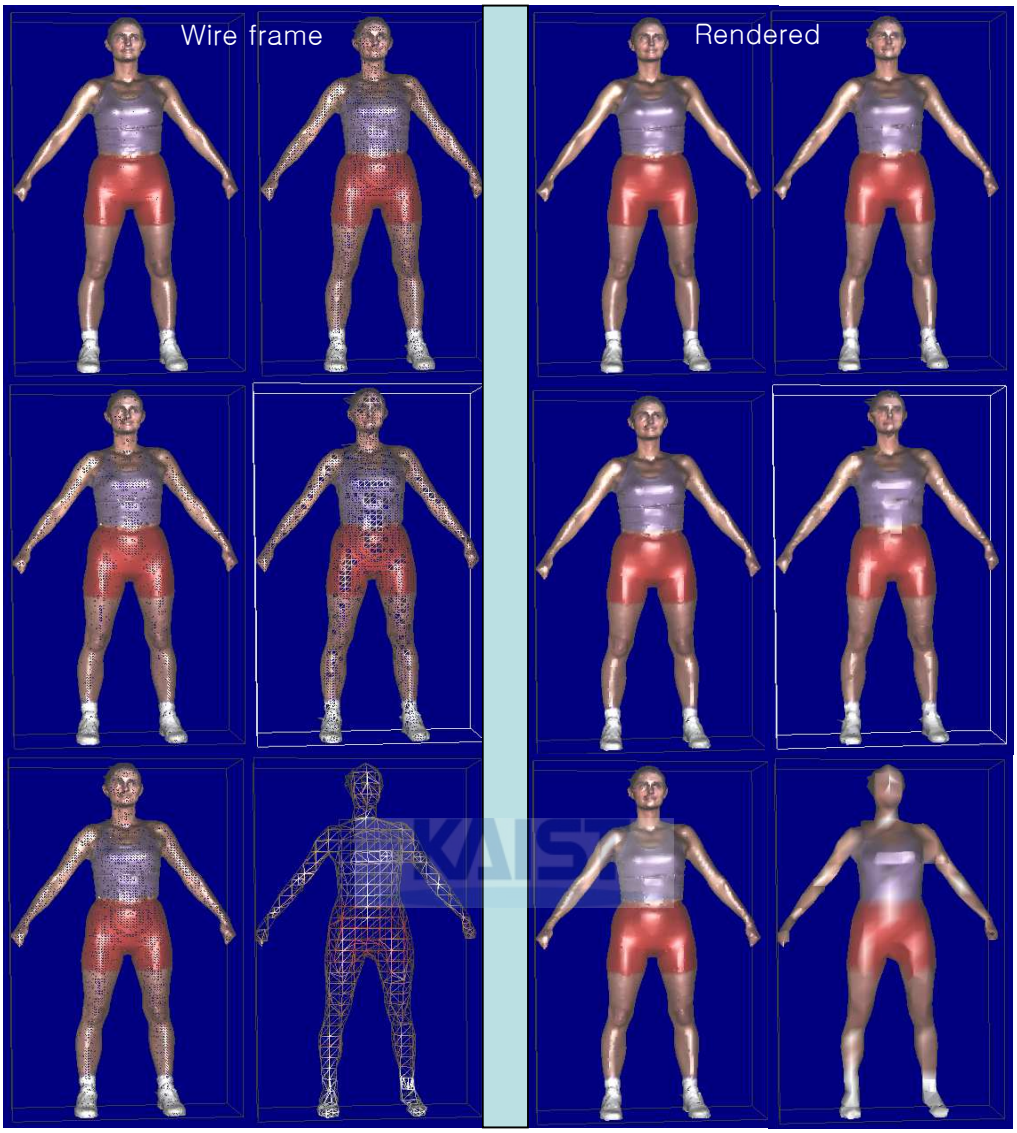


Fig. 6.5: LOD mesh generation results: colored woman body

(Dataset courtesy of Cyberware™)

Woman					
LOD (%)	triangles	vertices	LOD (%)	triangles	vertices
100	107074	53802	40	40478	30849
80	81157	53134	20	22037	14138
60	56117	43703	1	1998	1193

Table 6.1: Numbers of triangles, vertices in examples of woman body

The ‘Woman body’ meshes are also experimental data of Cyberware™ homepage. The results are showed with the LOD decreased by 20%. The meshes of the left side are wire frames and of the right side, rendered meshes in (Fig. 6.5). The numbers of triangles and vertices is showed in (Table 6.1). These results show our model keeps the attributes like colors in any LOD.

Among the other attributes, the texture coordinates are important in coarse meshes. Even when a very simplified model is used, the curvature or some featured part can be show naturally by texture mapping. So the attributes preserving is useful feature of the LOD modeling.

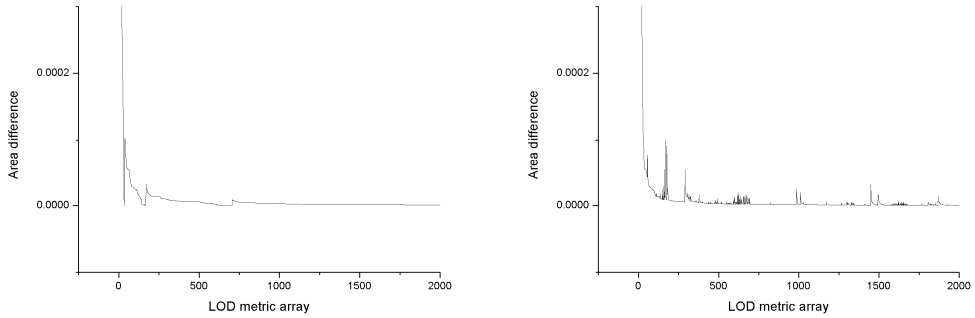


Fig. 6.6: LOD metric graph

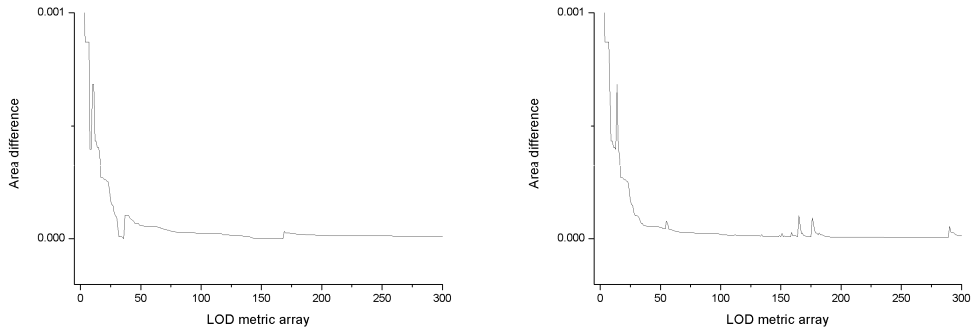


Fig. 6.7: LOD metric graph (part)

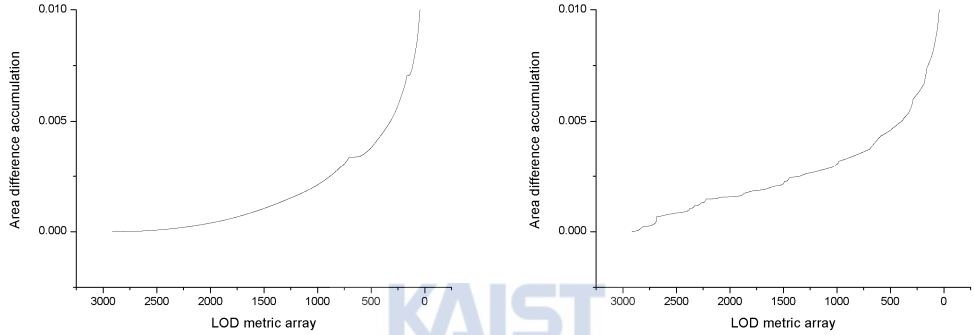


Fig. 6.8: LOD metric graph (accumulation)

LOD metric – the difference between the sum of a node's triangles area and the sum of its all child node's triangles area – data are used for the priority numbering. The graph in (Fig. 6.6) is LOD metric data of 'ball joint bone' mesh after the priority numbering. The left graph is the case of previous level-based algorithm [61] and the right graph is of improved. The left graph shows level discontinuity. They mean our method is beyond level constraint and more continuous model. The difference is clearer in (Fig. 6.7) which is the part of data less than 300 of LOD.

The accumulation graph is showed in (Fig. 6.8). This shows the improved algorithm is more robust about level discontinuity. But the quality of LOD modeling is the problem of a human perception. So the results are not judged easily. The ‘visual fidelity’ is another key issue of the LOD modeling.

Chapter7. Discussion

In this chapter, the proposed model is comparing to the other models. The multiresolution analysis and the progressive meshes are the break-through researches in the field of the continuous LOD modeling. Each model is a representative method for an adaptive subdivision and a geometry removal. The comparison table is itemized as follows.

	Time (Modeling)	Time (LOD Mesh)	Space	Collision detection	View dependency	Attribute preservation	Progressive transmission	Crack problem
M.A.	$O(n^2)$	F.P.	O(n)	O(n)	O(n)	No	Yes	Patching
P.M.	$O(n^2)$	F.P.	O(n)	O(n)	**O(log n)**	**Yes**	Yes	Free
M.C.O.	**O(n log n)**	**Ref.**	O(n)	**O(log n)**	**O(log n)**	**Yes**	Yes	Patching

Table 7.1: Comparison to related works

M.A. : Multiresolution Analysis, P.M. : Progressive Meshes, M.C.O. : Marching Cube Octree / F.P. : floating point operation, Ref. : referencing

- Time (Modeling)

M.A. starts from base mesh that is the simplest. When one triangle is divided and new vertices are moved, it calculates the moving points through whole initial mesh. So it's complexity is $O(n^2)$. The overhead of crack patching is through all vertices. So an overall complexity is not changed.

P.M. references all edges whenever it picks the edge to be collapsed. P.M. uses energy function optimization over whole vertices in this step, so it's time complexity is $O(n^2)$. The edge collapse operator is a reverse of the vertex split, thus one vertex is added to one operation.

The proposed algorithm references each range datum just once and calculates the configuration of octree. The crack patching is local calculation through whole node. In the node priority step, the priority queue is always sorted. The insertion to the priority queue is the same problem as "insert one entity to sorted list." The complexity of the insertion is O(log n). Then this operation proceeds over whole nodes. Thus, our algorithm's time complexity is O(n log n). But the unit of the operation is not a vertex, a node which contains several vertices. So the time is slightly faster than a vertex based operation.

- Time (LOD Mesh Generation)

In online mesh generation phase, M.A. starts from base mesh. To reach a certain LOD it divides triangles and moves the new vertices serially. It's time complexity is O(n) but each dividing and moving step needs floating point operation.

P.M. uses edge collapse or vertex split operation serially, too. Thus, it's complexity is also O(n) and these operation needs floating point operation, too.

Our model references only needed nodes to cover whole surface of mesh, so its complexity is O(n). But there's no float point calculation because ours needs only referencing operations to generate triangles.

For all three models, the traversal time is regardless compare to a triangulation. All models need to follow the links or pointers.

- Space

Because M.A. needs the space for base mesh and coefficients of vertex moving, its

space complexity is O(n). The space of pre-calculated crack patching vertex is smaller than that of base mesh and coefficients.

The space complexity of P.M. is also O(n). It needs the space for base mesh and coefficients of edge collapse and vertex split. the space for the links which determined what vertex is split into two vertices is smaller than that of coefficients and the base mesh.

Our algorithm uses the octree for data structure. The number of internal nodes is less than the external nodes. In case of complete binary tree, the difference is 1. The number of the internal nodes of complete octree is calculated as follows.

$$
\begin{aligned}
&I_n : \text{the number of internal nodes in octree} \\
&E_n : \text{the number of external nodes in octree} \\
&E_n = 8^n \\
&I_0 = 8^0 \quad I_1 = 8^0 + 8^1 \\
&I_n = 8^0 + 8^1 + \cdots + 8^{n-1} \\
&8I_n = 8^1 + \cdots + 8^{n-1} + 8^n \\
&7I_n = 8^n - 1 \\
&\therefore I_n = \frac{1}{7}(E_n - 1)
\end{aligned}
$$

The space complexity for external nodes is linear to the total number of vertices of initial mesh and is O(n). The space for octree structure is smaller than the leaf nodes. Thus, the space complexity of ours is O(n).

- Collision detection

Collision detection is very famous and important problem in 3D computer graphics. M.A. and P.M. have no advantage for collision detection. They must search all vertices to find a collision. So the complexity of collision detection of them are O(n). Because our model use 3D spatial octree, the collision is detected in O(log n).

- View dependency rendering

For fast rendering of 3D scene, view dependency rendering is very useful. M.A. can not support any special feature. All vertices must be searched to check view dependency. So its complexity is O(n). P.M. presents some features to speed up the view dependency rendering. It is some kind of refinement to calculate view dependency [32]. With this overhead its view dependency complexity is O(log n). The special octree used in our model is also very useful data structure for view dependency. The complexity of our model is O(log n).

- Attributes preservation

The attributes like color, texture and normal are necessary in computer graphics. Because M.A. starts from the simplest base mesh, there is no attributes preservation method. But P.M. starts from the initial mesh that is the most complex. Thus, it can preserve attributes. Our model is the same as P.M. The experimental result shows in (Fig. 6.5).

- Progressive transmission

All of them present progressive transmission. M.A. transmits the base mesh first and moving coefficients sequentially. P.M. also transmits base mesh first then coefficients next. In our model, breadth first traversal is used for progressive transmission.

- Crack problem

Crack problem must occur whenever the level idea is used. So M.A. and ours have problem and present solution for it. The solution is similar to each other. It is to move detailed level's vertex to patch the crack.

Chapter8. Conclusion

8.1 Summary

In the paper, the improved method of LOD modeling using the marching cube octree was proposed. We create the representation easily and efficiently by using the marching cube features. We can take advantage of the octree representation in a 3D graphics system. Our LOD model can support adaptive simplification, progressive transmission, view dependency rendering and collision detection. By using the marching cubes, our LOD mesh generation algorithm becomes efficient without floating point operations.

In the surface reconstruction, modified marching cubes are used for reconstructing the initial mesh. Instead of a midpoint which is used in the original algorithm [57], the ratio of adjacent vertex is used for the triangulation. The results are also feasible when low resolution data is given. These marching cubes are used for the leaf nodes of the marching cube octree.

The marching cube octree is constructed using the octree structure with the marching cubes. To support continuous LOD between levels, all nodes are numbered by priorities. Using adjacent node check and priority queue, node priority numbering can traverse both breadth and depth.

The LOD meshes are generated at run-time using proposed efficient algorithm. It triangulates only needed nodes of the marching cube octree to cover up the whole surface of mesh. Using the priority numbers on nodes and flagging, the LOD mesh can be generated by only referencing without floating point operations which the other LOD models need.

The contributions of this paper are as follows; to support a continuous LOD, modifying the marching cube octree (vertex ratio on edge of marching cube) and priority numbering the nodes with an algorithm that can simplify flat area more. An efficient LOD

mesh generation which refers only needed nodes without floating point operations. The proposed model can collision detection efficiently in the complexity of O(log n).

8.2 Future Work

In proposed model, the marching cube is divided into halves in the next level. If we control the size of the next level's cube and formulate it, we can construct a more detailed continuous LOD model. In that case, the fastness and simplicity of the modeling may be lost. And the advantage of sampling simplification is no longer available.

The marching cube octree LOD model can be extended to simplify only viewing area. Using the feature of an octree, it is thought that the model can be developed efficiently. View-dependent LOD modeling can be applied under extra large models with external storages. An octree is efficient data structure for collision detection in 3D graphics. With proposed model, it is thought that the LOD of collision detection can be modeled.

Summary (in Korean)

컴퓨터 그래픽스 분야에서의 상세화 모델링(Level-of-detail modeling)이란 실시간에 mesh을 이루는 다각형의 개수를 조절하는 것을 말한다. 상세화 모델링은 작거나 멀거나 중요하지 않은 위치의 rendering 비용을 줄이기 위해서 꼭 필요하다. 이 논문에서는 marching cube 8진트리를 이용한 상세화 모델을 제안하는데 이것은 다양한 상세화 정도의 mesh를 효율적으로 생성하기에 적합한 방법이다.

표면 재구성 단계에서는 기존의 marching cube 알고리즘을 조금 수정하여 초기 mesh를 생성하게 된다. 기존의 marching cube에서 사용하던 중간점 대신 이웃한 꼭지점의 비율을 이용하여 삼각형을 생성한다. 이 방법은 입력 데이터의 해상도가 낮아도 만족스러운 결과를 보여준다. 이 단계에서 사용된 marching cube는 marching cube 8진트리의 말단 노드가 된다.

Marching cube 8진트리는 marching cube를 가지고 8진트리를 구성한 것이다. 모든 노드는 각 단계간에 상세화 정도를 연속적으로 해주기 위해서 우선순위가 매겨진다. 우선순위를 매길 때는 주변 노드 검사와 우선순위 queue를 사용하여 트리의 노드를 방문하게 된다. 이 때 단계가 다른 인접한 두 노드 사이에 생기는 틈새를 메우기 위한 계수도 미리 구해둔다.

구성된 marching cube 8진트리를 바탕으로 논문에서 제안된 효율적인 알고리즘에 의해 실시간으로 상세화 mesh를 생성한다. 이 때 mesh의 표면을 구성하는 노드만 참조해서 삼각형을 만들게 된다. 우선순위와 표시하기(flagging)를 이용하는 이 알고리즘은 다른 방법에서 필요한 부동소수점연산이 필요 없고 단지 참조만으로 구성되기 때문에 효율적이다.

References

[1] M. Levoy, K. Pulli, B. Curless, S. Rusinkiewicz, D. Koller, L. Pereira, M. Ginzton, S. Anderson, J. Davis, J. Ginsberg, J. Shade and D. Fulk. The Digital Michelangelo Project: 3D Scanning of Large Statues. Computer Graphics, 131-144. 2000.

[2] D. Miyazaki, T. Ooishi, T. Nishikawa, R. Sagawa, K. Nishino. The Great Buddha Project: Modelling Cultural Heritage through Observation. Modelling from reality (Kluwer Engineering And Computer Science International Series), 181-193. 2001.

[3] J. Clark. Hierarchical geometric models for visible-surface algorithms. Communications of the ACM, 19, 547-554. 1976.

[4] D. Luebke. A developer's survey of polygonal simplification algorithms. IEEE Computer Graphics & Applications, 21(3):24--35, May/June 2001.

[5] H. Lee, H. Yang, Wavelet-Based Level-Of-Detail Representation of 3D Objects, Journal of KISS (Korea Information Science Society): Computer Systems and Theory, 29(3, 4), pp 185-191, April 2002.

[6] D. Luebke, B. Watson, J. D. Cohen, M. Reddy, A. Varshney. Level of Detail for 3D Graphics. Elsevier Science Inc. 2002.

[7] M. Cosman, R. Schumacker. System Strategies to Optimize CIG Image Content. Proceedings Image II Conference (Scotsdale, Arizona), 463-480. 1981.

[8] W. Schroeder, J. Zarge, and W. Lorensen, Decimation of triangle meshes, Computer Graphics 26, 65-78. 1992.

[9] J. Rossignac, Jarek, P. Borrel. Multi-Resolution 3D Approximations for Rendering Complex Scenes, Geometric Modeling in Computer Graphics, 455-465. 1993.

[10] T. Funkhouser, C. Sequin. Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments. In Proceedings of the 20th Annual Conference on Computer Graphics and interactive Techniques SIGGRAPH '93. 247-254. 1993.

[11] D. Luebke, C. Erikson. View-dependent simplification of arbitrary polygonal environments. Computer Graphics (Proc. SIGGRAPH '97), pages 199-208, 1997.

[12] J. Xia and A. Varshney. Dynamic view-dependent simplification for polygonal models. In Proc. IEEE Visualization '96, pages 327-334, 1996.

[13] M. Garland and P. Heckbert. Surface simplification using quadric error metrics. In Proc. of SIGGRAPH 97, pages 209-216, August 1997.

[14] J. Cohen, A. Varshney, D. Minocha, G. Turk, H. Weber, P. Agarwal, F. Brooks and W. Wright. "Simplification envelopes." Proc. Computer Graphics 1996 (SIGGRAPH `96), pp.119-128, 1996.

[15] J. Cohen, M. Olano, D. Manocha. Appearance-preserving simplification. In Proceedings of the 25th Annual Conference on Computer Graphics and interactive Techniques SIGGRAPH '98. ACM Press, New York, NY, 115-122. 1998.

[16] T. He, L. Hong, A. Varshney, and S. Wang. Controlled topology simplification. IEEE Trans. on Visualization & Computer Graphics, 2(2):171-183, 1996.

[17] J. El-Sana and A. Varshney. Topology simplification for polygonal virtual environments. IEEE Trans. Visualization Comput. Graphics, 4:133-144, 1998.

[18] C. Erikson. Polygonal Simplification: An Overview. Technical Report TR-96-016, University of North Carolina - Chapel Hill, 1996.

[19] P. Cignoni, C. Montani, R.Scopigno. A comparison of mesh simplification algorithms. Computer and Graphics 22 (1), pp.37-54, 1998

[20] E. Catmull. J. Clark. Recursively generated B-spline surfaces on arbitrary topological surfaces. Computer-Aided Design, 10(6), 350-355, 1978.

[21] M. Duchaineau, M. Wolinsky, D.E. Sigeti, M.C. Miller, C. Aldrich, and M.B. Mineed-Weinstein. Roaming terrain: Real-time optimally adapting meshes. In Proceedings IEEE Visualization'97, pages 81-88, 1997.

[22] M. Gross, O. Staadt, and R. Gatti. Efficient triangular surface approximations using wavelets and quadtree data structures. IEEE Trans. on Visual and Comp. Graph., 2(2):130-144, June 1996.

[23] D. Hebert and H. Kim. Image encoding with triangulation wavelets. Proceedings SPIE, (2569(1)):381-392, 1995.

[24] M. Lounsbery. T. Derose, J. Warren. Multiresolution Analysis for Surfaces of Arbitrary Topological Type. Transactions on Graphics, 16(1), 34–73. 1997.

[25] M. Lounsbery. Multiresolution Analysis for Surfaces of Arbitrary Topological Type. doctoral dissertation. University of Washington, Seattle, WA, 1994.

[26] H. Hoppe. Progressive Meshes. Computer Graphics, 30, pages 99-108. 1996.

[27] J. Popovic, H. Hoppe. Progressive simplicial complexes. Computer Graphics, 31, pages 217-224. 1997.

[28] M. Eck, T. Derose, T. Duchamp, H. Hoppe, M. Lounsbery, W. Stuetzle. Multiresolution Analysis of Arbitrary Meshes. Computer Graphics, 29, pages 173–182. 1995.

[29] A. Certain, J. Popovic, T. Derose, T. Duchamp, D. Salesin, W. Stuetzle. Interactive Multiresolution Surface Viewing. Computer Graphics, 30, pages 91–98. 1996.

[30] D. Zorin, P. Schroder, W. Sweldens. Interactive Multiresolution Mesh Editing. Computer Graphics, 31, pages 259–268. 1997.

[31] A. Lee, W. Sweldens, P. Schröder, L. Cowsar, D. Dobkin. MAPS: Multiresolution Adaptive Parameterization of Surfaces. Computer Graphics, 32, pages 95-104. 1998.

[32] M. Haemer and M. Zyda. Simplification of objects rendered by polygonal approximations. Computers & Graphics, 15(2):175-184, 1991.

[33] A. Kalvin, C. Cutting, B. Haddad, and M. Noz. Constructing topologically connected surfaces for the comprehensive analysis of 3D medical structures. SPIE Vol. 1445 Image Processing, pages 247-259, 1991.

[34] P. Hinker and C. Hansen. Geometric optimization. In IEEE Visualization '93 Proc., pages 189-195, October 1993.

[35] A. Kalvin and R. Taylor. Superfaces: Polygonal mesh simplification with bounded error. IEEE C.G.&A., 16(3):64-77, 1996.

[36] A. Gueziec. Surface simplification inside a tolerance volume. Technical Report RC 20440, I.B.M. T.J.Watson Research Center, 1996.

[37] R. Ronfard and J. Rossignac. Full-range approximation of triangulated polyhedra. Computer Graphics Forum (Eurographics'96 Proc.), 15(3):67-76, 1996.

[38] M. Algorri and F. Schmitt. Mesh simplification. Computer Graphics Forum (Eurographics'96 Proc.), 15(3):78-86, 1996.

[39] B. Hamann. A data reduction scheme for triangulated surfaces. Computer Aided Geometric Design, 11(2):197-214, 1994.

[40] M. Soucy, D. Laurendeau. Multiresolution surface modeling based on hierarchical triangulation. Computer Vision and Image Understanding, 63(1):1-14, 1996.

[41] A. Ciampalini, P. Cignoni, C. Montani, and R. Scopigno. Multiresolution decimation based on global error. The Visual Computer, 13(5):228-246, June 1997.

[42] C. Bajaj and D. Schikore. Error bounded reduction of triangle meshes with multivariate data. SPIE, 2656:34-45, 1996.

[43] R. Klein, G. Liebich, and W. Straßer. Mesh reduction with error control. In R. Yagel and G. Nielson, editors, Proceedings of Visualization '96, pages 311-318, 1996.

[44] K. Renze and J. Oliver. Generalized unstructured decimation. IEEE C.G.&A., 16(6):24-32, 1996.

[45] P. Cignoni, C. Montani, E. Puppo, and R. Scopigno. Multiresolution Representation and Visualization of Volume Data. Technical Report C97-05, Istituto CNUCE C.N.R., Pisa, Italy, January 1997.

[46] B. Hamann and J. Chen. Data point selection for piecewise trilinear approximation. Computer Aided Geometric Design, 11:477-489, 1994.

[47] K. Low and T. Tan. Model simplification using vertex clustering. In 1997 ACM Symposium on Interactive 3D Graphics, pages 75-82, 1997.

[48] M. Reddy. Scrooge: Perceptually-driven polygon reduction. Computer Graphics Forum, 15(4):191-203, 1996.

[49] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Mesh optimization. In ACM Computer Graphics Proc., Annual Conference Series, (Siggraph ’93), pages 19-26, 1993.

[50] H. Hoppe. View-Dependent Refinement of Progressive Meshes. Computer Graphics, 31, pages 189–198. 1997.

[51] M. Shafae and R. Pajarola, DStrips: Dynamic Triangle Strips for Real-Time Mesh Simplification and Rendering, Proc. Pacific Graphics 2003, pp. 271-280, 2003.

[52] C. DeCoro and R. Pajarola, "XFastMesh: Fast View-Dependent Meshing from External Memory," Proc. IEEE Visualization 2002. 363-370. 2002.

[53] D. Luebke, C. Erikson. View-Dependent Simplification of Arbitrary Polygonal Environments. Computer Graphics, 31, pages 199-208. 1997.

[54] G. Turk. Re-tiling polygonal surfaces. In Edwin E. Catmull, editor, ACM Computer Graphics (SIGGRAPH ‘92 Proceedings), volume 26, pages 55-64, July 1992.

[55] C. Andujar, D. Ayala, P. Brunet, R. Joan-Arinyo, and J. Sole’. Automatic generation of multiresolution boundary representations. Computer Graphics Forum (Eurographics’96 Proc.), 15(3):87-96, 1996.

[56] T. He, L. Hong, A. Kaufman, A. Varshney, and S. Wang. Voxel-based object simplification. In IEEE Visualization ’95 Proceedings, pages 296-303. IEEE Comp. Soc. Press, 1995.

[57] W. Lorensen, H. Cline. Marching Cubes: A High Resolution 3D Surface Construction Algorithm. Computer Graphics, 21(4), pages 163-170. 1987.

[58] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, W. Stuetzle. Surface Reconstruction from Unorganized Points. Computer Graphics, 26(2), pages 71-78. 1992.

[59] B. Curless, M. Levoy. A Volumetric Method for Building Complex Models from Range Images. SIGGRAPH 96 Conference Proceedings, Annual Conference Series, pp. 303-312, August 1996.

[60] J. Wilhelms, A. Gelder, Octrees for Faster Isosurface Generation, ACM Transactions on Graphics, 11(3), 201-227. 1992.

[61] H. Lee, J. Lee, H. Yang. Real-time LOD: Marching-cube-and-octree-based 3D Object Level-of-detail Modeling. The 8th International Conference on Virtual Systems and MultiMedia Proceedings, pages 634 – 643. 2002.

[62] R. Shekhar, E. Fayyad, R. Yagel, J. Cornhill. Octree-Based Decimation of Marching Cubes Surfaces, IEEE Visualization, pages 335-342. 1996.

[63] G. Turk, M. Levoy. Zippered Polygon Meshes from Range Images. Computer Graphics, Vol. 28, Number Annual Conference Series, pages 311-318, July 1994.

[64] K. Sahr, D. White, and A. Kimerling. Geodesic Discrete Global Grid Systems. Cartography and Geographic Information Science, 30(2), 121-134. 2003.

[65] P. Rigaux, M. Scholl and A. Voisard. Spatial Databases - with application to GIS. Morgan Kaufmann, San Francisco, 2002.

[66] R. Finkel, J. Bentley. Quad Trees: A Data Structure for Retrieval on Composite Keys. Acta Informatica 4 (1): 1-9. 1974.

[67] A. Guttman: R-Trees: A Dynamic Index Structure for Spatial Searching, Proc. 1984 ACM SIGMOD International Conference on Management of Data, pp. 47-57. 1984.

[68] Y. Manolopoulos, A. Nanopoulos, A. Papadopoulos, Y. Theodoridis. R-Trees: Theory and Applications, Springer, 2005.

[69] L. Arge, M. Berg, H. Haverkort, K. Yi. The Priority R-Tree: A Practically Efficient and Worst-Case Optimal R-Tree, Proc. 2004 ACM SIGMOD international conference on Management of data, pp. 347-358. 2004.

[70] N. Beckmann, H. Kriegel, R. Schneider, B. Seeger. The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles. SIGMOD Conference, 322-331. 1990.

[71] R. Bayer and V. Markl. The UB-Tree: Performance of Multidimensional Range Queries. Technical Report TUMI9814, Institut fur Informatik, TU Munchen, 1997.

[72] J. Wilhelms, V. Gelder, A octree for faster isosurface generation. ACM Trans. Graph. 11, 201-227. 1992.

[73] P. Jimenez, F. Thomas, C. Torras. 3D collision detection: A survey. COMPUTER GRAPHICS (PERGAMON). Vol. 25, no. 2, pp. 269-285. Apr. 2001

[74] M. Moore, J. Wilhelms. Collision detection and response for computer animationr3. In Proceedings of the 15th Annual Conference on Computer Graphics and interactive Techniques R. J. Beach, Ed. SIGGRAPH '88. ACM Press, New York, NY, 289-298. 1988.

[75] M. Moore. A Flexible Object Animation System, Masters Thesis, University of California, Santa Cruz, Computer & Information Sciences, Santa Cruz, California, March, 1988.

[76] K. Hamada, Y. Hori. Octree-based approach to real-time collision-free path planning for robot manipulator. ACM96-MIE, p. 705-10. 1996.

[77] N. Ahuja, R. Chien, R. Yen, N. Bridwell. Interference detection and collision avoidance among three dimensional objects. I Annual National Conference on AI, August, Stanford University, p. 44-8. 1980.

[78] V. Hayward. Fast collision detection scheme by recursive decomposition of a manipulator workspace. In Proceedings of the IEEE International Conference on Robotics and Automation, San Francisco, CA, vol. 2, p. 1044-9. 1986.

[79] J. Klosowski. Efficient Collision Detection for Interactive 3D Graphics and Virtual Environments. Doctoral thesis, State Univ. of New York at Stony Brook, 1998.

[80] P. Hubbard. Interactive collision detection. Proceedings of the IEEE Symposium on Research Frontiers in Virtual Reality, vol. 1, p. 24-31. 1993.

[81] P. Hubbard. Approximating polyhedra with spheres for time-critical collision detection. ACM Transactions on Graphics, 15(3):179-210, 1996.

[82] I. Palmer, R. Grimsdale. Collision detection for animation using sphere-trees. Computer Graphics Forum. Vol. 14, no. 2, pp. 105-116. 1995.

[83] Y. Liu, S. Arimoto, H. Noborio. A new solid model hsm and its application to interference detection between moving objects. Journal of Robotic Systems;8(1), p. 39-54. 1991.

[84] S. Bandi, D. Thalmann. An adaptive spatial subdivision of the object space for fast collision detection of animating rigid bodies. Eurographics'95, Maastricht, p. 259-70. August 1995.

[85] B. Garlick, D. Baum and J. Winget. Interactive Viewing of Large Geometric Databases Using Multiprocessor Graphics Workstations. Siggraph Course Notes (Parallel Algorithms and Architectures for 3D Image Generation). p. 239-45. 1990.

[86] N. Greene, M. Kass, G. Miller. Hierarchical Z-buffer visibility, Proceedings of the 20th annual conference on Computer graphics and interactive techniques, p.231-238, September 1993.

[87] Y. Wang, H. Bao, Q. Peng. Accelerated walkthroughs of virtual environments based on visibility preprocessing and simpli"cation. Computer Graphics Forum, 17(3):187-194. 1998.

[88] C. Saona-Vazquez, I. Navazo, and P. Brunet. The visibility octree: A data structure for 3d navigation. Computer & Graphics, 23(5):635-644, 1999.

[89] U. Assarsson, T. Moller. Optimized view frustum culling algorithms for bounding boxes. J. Graph. Tools 5, 1, 9-22. 2000.

[90] S. V. Matveyev. Approximation of isosurface in the marching cubes: Ambiguity problem. In Visualization '94, pages 288-292, October 1994.