# EDITING, RETRIEVAL, AND DISPLAY SYSTEM OF ARCHEOLOGICAL INFORMATION ON LARGE 3D GEOMETRIC MODELS
# 大規模３次元モデル上の情報の編集・表示システム

by

Yasuhide Okamoto
岡本 泰英

A Master Thesis
修士論文

Submitted to
the Graduate School of Information Science and Technology
the University of Tokyo
on February 7, 2006
in Partial Fulfillment of the Requirements
for the Degree of Master of Information Science and Technology
in Computer Science

Thesis Supervisor: Katsushi IKEUCHI　池内 克史
Professor of Computer Science

## ABSTRACT

Recent advancement in modeling technologies enables us to obtain 3D models with an extremely large number of polygons. One of the promising directions for utilizing these 3D models is to provide computer graphics models with various types of archeological information. Unfortunately, however, such huge models cannot be handled easily on common PCs because of limits to performance of the hardware.

This thesis proposes an editing, retrieval and display system for archeological information on huge 3D models. This system can deal with the information related to target objects, which stored in a variety of formats, on 3D geometric surfaces. For example, it enables us to 1) associate extra information with models, 2) edit and retrieve this information from models, and 3) browse and view the information efficiently and intuitively. For defining target regions on 3D models, users can quickly select specific regions from huge models by using an extended Lazy Snapping method, and can make associations and access associated information with easy mouse actions. In the display process, we achieved highly interactive rendering of huge 3D models composed of millions of polygons by adopting an efficient 3D rendering algorithm using multi-resolution meshes.

## 論文要旨

近年、モデリング技術の向上により非常に大規模な 3 次元モデルの生成が可能となっている。そのようなモデルを有効に活用する方法として、モデルに関する付加的な情報を幾何情報と共に提供するというニーズが生まれている。しかし、得られた大規模モデルを一般の PC で利用するには、ハードウェアの制約により従来の手法では扱えない場合が多い。そのためそれらを扱うための新たな枠組みが必要である。本論文においては大規模な 3 次元モデルと他フォーマットの情報との関連付け、及びそれらの情報の閲覧作業を可能にするシステムの提案を行う。

本システムでは対象モデルに関連する様々なフォーマットの情報の 3 次元モデルへの関連付け・編集及び検索・閲覧の作業を直感的に行うことを可能としている。関連付け操作におけるモデルの領域定義においては、Lazy Snapping 法を利用した領域分割により、大規模なモデルにおいても現実的な処理時間を実現し、情報とモデルの関連付け・アクセスなどは簡単なマウス操作によって容易に行えるインタフェースを提案する。また、3 次元モデルの描画処理に関しては多重解像度メッシュを用いた効率的な手法を用いることにより、大規模モデルのリアルタイムな描画を実現した。

# Acknowledgements

Thanks to Prof.Katsushi Ikeuchi for giving many advices. Thanks also to Takeshi Oishi and Jun Takamatsu for providing a great deal of knowledge and some kinds of advices in my research topics, programming technique, and private problems. And to Joan Knapp for proof-reading my thesis written in my poor English. Also to the National Research Institute for Cultural Properties (NASRIPT) in Tokyo for providing various kinds of data. In addition to them, I would like to thank to everyone researching with me in the Ikeuchi laboratry for listening to my trivial annoyances and giving helping hand to me.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

## 1.1 Background

In the field of computer vision, there is an application that converts real objects into digital 3-Dimensional models for the preservation of cultural properties ([13]). The accurate 3D models are very useful. For example, they are used for digital storage of real objects, analysis of their geometric shapes, and production of multimedia contents. These applications have contributed to the activity of preserving and displaying precious objects such as [14] and [19].

To obtain 3D geometric shapes from real objects, there are three steps: scanning, alignment, and merging. In the scanning step, we use several types of laser range sensors to measure a surface geometry of objects from different points to cover all surfaces. After scanning, each piece of range data obtained from many viewpoints has its own coordinate system. At this step, we align coordinate system of each range image to the global coordinate. We use a simultaneous alignment method, developed in our group. This method avoids the accumulation errors by estimating all range data simultaneously [24]. The last step is merging. All range data are converted into a single surface to remove the redundancy and fill holes. We employed octrees to represent a volumetric implicit-surface representation with signed distance field [25]. After obtaining 3D geometric model, we texture map by the novel method that estimates parameters by comparing reflectance values in range data with color images photographed by a digital camera [17].

We obtained 3D models from real objects by those methods. Recently, the improvement of 3D modelling technologies, such as resolving power of laser sensors and processing power of workstations, have enabled us to obtain large 3D models in fine detail. This advancement of resolution has caused an increase in the size of the models. For example, we can obtain the model of Bayon temple which has over thousands of million polygons.

However, there are processing problems in increase of data. Such huge models cannot be rendered with traditional rendering methods, even if we uses modern workstations. In addition, the expansion of the data size causes other processing faults. It becomes difficult to treat such large models rapidly by simple methods on common computers. To overcome these limits, we must propose more efficient algorithms to treat huge models.

The increase in data is not only limited to geometric data. During many years of archeological research, the researchers accumulated a large amount of various kinds of data. This included historic data such as the origin and background of the objects, engineering data such as materials and architectural styles, and annotation data such as the condition of the deterioration, destruction, and the plan and progress of restoration. Since this research continues, the amount of accumulated data will be increasing. Additionally, the types of data are represented by a wide variety of formats; for example, text documents, images, numerical values, and other formtat of data. Many database systems to manage such data efficiently have been proposed. However, researchers dealing with these kinds of data are not necessarily familiar with operating database applications, and they store data in traditional forms such as lists and tables.

Therefore, we propose a system that allows users to embed information to 3D surfaces and to access to it through the 3D model. Our system allows users to interactively and easily treat data recorded in any format on 3D models.

In this thesis we propose a system that can render huge 3D models and offer users a database that uses the models as an index to information.

## 1.2 Contribution

Our proposed system offers three main contributions:

- An efficient rendering system. Our proposed system allows users to view very huge 3D models in real time with high interactivity even if the polygons of 3D models are over hundreds of millions.

- The ability to embed a wide variety of data in 3D surfaces. If users find a part on a surface that they want to mark or associate with other files, in viewing the target 3D model, users can assign the data to the region on the surface. Then, users can select specified regions on 3D surfaces by simple operations.

- The ability to retrieve and display the assigned information on the 3D models. Users can access to the information by clicking specified regions in viewing the model. Additionally, the provider of data can easily edit and remove the assigned data.

## 1.3 Related Works

The rendering of huge meshes has been the a “hot topics” in computer graphics because of increase in mesh size.

One type of solutions is the level-of-detail method for huge mesh rendering. In [22] it was proposed to remove the polygons whose projected size is too small to affect the rendering image. In addition, triangle mesh reduction that merges smaller triangles to form larger ones was presented in [11], [12], and [16]. Although these methods can render at higher speed than previous methods, the reduction of each triangle needs a high CPU load when preprocessing and rendering.

To render huge models, new types of rendering primitive were also proposed, i.e. points[18]. The images rendered by points have poorer quality than those rendered by triangles, but they do not need several processes, as triangle rendering does. They can render more quickly than triangle rendering and save the necessary amount of data. QSplat[20] adopts the multiresolution hierarchy of points and changes the set of rendered points depending on the view. Another method is Sequential Point Trees [26], which proposes more efficient point rendering via an algorithm suitable for graphics processing unit (GPU) processing.

Recently, GPU processing power has been improving. However, the rendering pipeline on GPU is not optimized for point, but for polygon rendering. Therefore the efficient polygon rendering methods for huge meshes have been proposed again. In [7], a patch-based hierarchy is proposed to reduce the size of the hierarchy to less than a triangle-based hierarchy. This idea relaxes the complexity of reduction of triangles compared with previous level-of-detail methods.

There are some applications that associate 3D models and related information. However, most of those focus simple 3D models with a small number of triangles such as CAD models, and are unavailable for massive meshes. The system proposed in [6], locates the related information on the 3D model obtained by scanning. Information displayed includes analysis of research such as graphs and charts, and many kinds of images and pictures. Some web applications have been proposed that map information and notes on landmarks to a 3D map of the earth; i.e. Nasa World Wind [3] and Google Earth [2]. These applications propose 3D geometry information, texture images, and embedded information as landmarks through networks.

## 1.4 Thesis Overview

We present an overview of our proposed method in chapter 2. We explain details of the construction of 3D meshes with multi-resolution hierarchy and rendering algorithm in chapter

3. When assigning information to 3D surfaces, users need to select a specific region as indexes to information. In chapter 4, the algorithm of efficient selection is described. In chapter 5, we explain the design details of our proposed system and how to use it. Finally, we evaluate the performance of our system in chapter 6, and conclude this thesis in chapter 7.

# Chapter 2

# System Overview

In this thesis, we propose a system that can render very large meshes with a very large number of triangles and treat information whose format has a wide variety on the 3D models that are viewed.

At the design of the system, we split its functions into three parts, which are 1) rendering of 3D models, 2) assigning of information to 3D models, and 3) displaying of information assigned to 3D models while viewing the model. Users can edit and view information on our system by using those functions. In those functions, we developed efficient algorithms to avoid processing delays and reduction of usability even if the size of models and the amount of information are very large.

In the following section, we present an overview of those procedures in our system.

## 2.1 Rendering System

On our system, users can view a 3D model of various objects. Users can freely change the position and direction of the viewing camera by interactive interface during rendering.

Our system enables us to view very huge 3D meshes interactively even if the size of meshes is larger than the size of the system memory, which is called “out-of-core”. To realize it, we developed an efficient algorithm in time and space. For efficient rendering, we develop view-dependent multi-resolution rendering based on the method described in [7] and [28]. By using this rendering method, our system can render each frame by loading and processing a requisite minimum size of data. This method converts an input mesh into a multi-resolution mesh that has a set of small patches at different levels of resolution and hierarchy of the patches. When rendering, we traverse the multi-resolution hierarchy from the top down, find the patches that can be rendered on screen with less screen errors, and load and render only

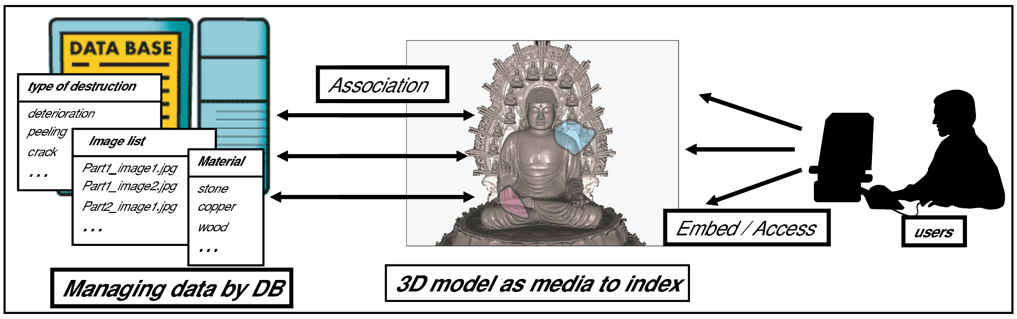


Figure 2.1: The 3D models are used as index. On our system, users can assign and display information on 3D models when viewing them. Assigned information is managed in the database system, and our system can efficiently retrieve and access it.

patches. The traversal of hierarchy needs less processing load than processing of an original mesh in simple format because the size of patch-based hierarchy we use is dramatically smaller than that of original format. Additionally, we can reduce more processing load by skipping the traversals of patches with fine resolution and invisible patches on screen.

We describe the detail of the algorithm in chapter 3.

## 2.2 Assignment of Archeological Information

The second main function of our system is assignment to 3D models of information such as archeological information related to the object.

It is difficult to deal with information recorded in a wide variety of formats interactively and intuitively. To solve it, our proposed system enables users to assign information to 3D models and retrieve it from the models assigned to.

First, we introduce the assignment of information. Initially, users specify regions to which they hope to assign information on 3D surfaces and to use as an index to information. After that, users locate information on the specified regions. Users can do these operations very easily by mouse actions like drag and drop.

We propose a simple selection method similar to lasso. When selecting regions, we use the result of 3D segmentation of the target model to select accurately. However, it is difficult to select all kinds of regions by a simple mouse action. For example, if users hope to select a region with a very complex border, they cannot rapidly draw the border and select it by dragging the mouse. Additionally, we must compute the selected regions on 3D surfaces so rapidly that users cannot make their selections if the size of 3D model is very large. To resolve this problem, we develop the novel selection method that is based on the Lazy Snapping [21] extended to 3D models. To select regions, users need to draw only several markers on “foreground” regions and “background” regions. We add some extensions to the method to efficiently select regions on huge 3D models. Additionally, we propose a supplementary method for the case in which users do not obtain desired regions. We describe these methods in chapter 4.

Users locate information on specified regions. We assume that the formats of information are text documents, numerical values, images, and links to web pages. We pick that information from tables on the database systems or files on *Explorer*©, and locate it on surfaces by a simple drag and drop action. The dropped region is efficiently identified by using index images which represent the ID of specified regions on screen. The assigned information is registered and managed in the database system. We explain this in chapter 5.

## 2.3 Display of Archeological Information

By assignment of information, users can construct a new database system associated with 3D models, and they can utilize 3D models as an interfaces to information.

At the assignment step, the information embedded in the 3D model is stored in a database system, and can be pick out when users want to view it. If users hope to view information related to a part on a surface, they move a mouse cursor and double-click the part. Then, the information browser opens and displays the items assigned to the region. This browser use *InternetExplorer*© components, so users can access related web pages and browse images. The assigned files unsupported by the browser can also be opend by associated applications. In addition, on this browser users can edit labels and captions of the region and items assigned to it and remove items. The search of information is efficiently processed by the functions of existing database system. Therefore we do not discuss the function of database system in this thesis.

We present details of this function in chapter 5.

# Chapter 3

# Rendering Algorithm

We developed a novel algorithm for interactive view-dependent rendering based on the method described in [7] and [28]. This algorithm enables our system to simplify meshes view-dependently, cull occlusion faces, and render out-of-core mesh data. This method uses a hierarchy for multi-resolution rendering to partition the model. Each node of hierarchy contains a precomputed simplified patch of the original model. In rendering time, the hierarchy is traversed coarse-to-fine to select patches of the appropriate resolution given the view parameters as shown in figure 3.1. The resulting system can interactively render high quality views of out-of-core models on current commodity workstations.

In this chapter, we describe the details of the preprocessing and rendering of huge meshes.

## 3.1 Data Structure and Preprocessing

For view-dependent rendering, the meshes must be converted into a form that can compactly represent the step of refinement or coarsening and be efficiently extracted. To realize it, we use not the points or triangles but patches of surfaces, which are sets of points and triangles as primitives of the multi-resolution hierarchy. This approach enables us to reduce the effort for construction and traversing of the multiresolution hierarchy, and the size of resulting data structure more efficiently than the approach using points or triangles as primitives. In particular, in the case of very huge models, the size of hierarchy based on vertices or triangles explode, and the massiveness complicates the constraint for keeping consistency in multi-resolution. As a result, it is difficult for common computers to construct and render it efficiently. But the size of patch-based hierarchy is smaller and at the same time the constraint for consistency is also more simple than with vertex- or triangle-based hierarchy, and so we need less time and disk space for construction and rendering.

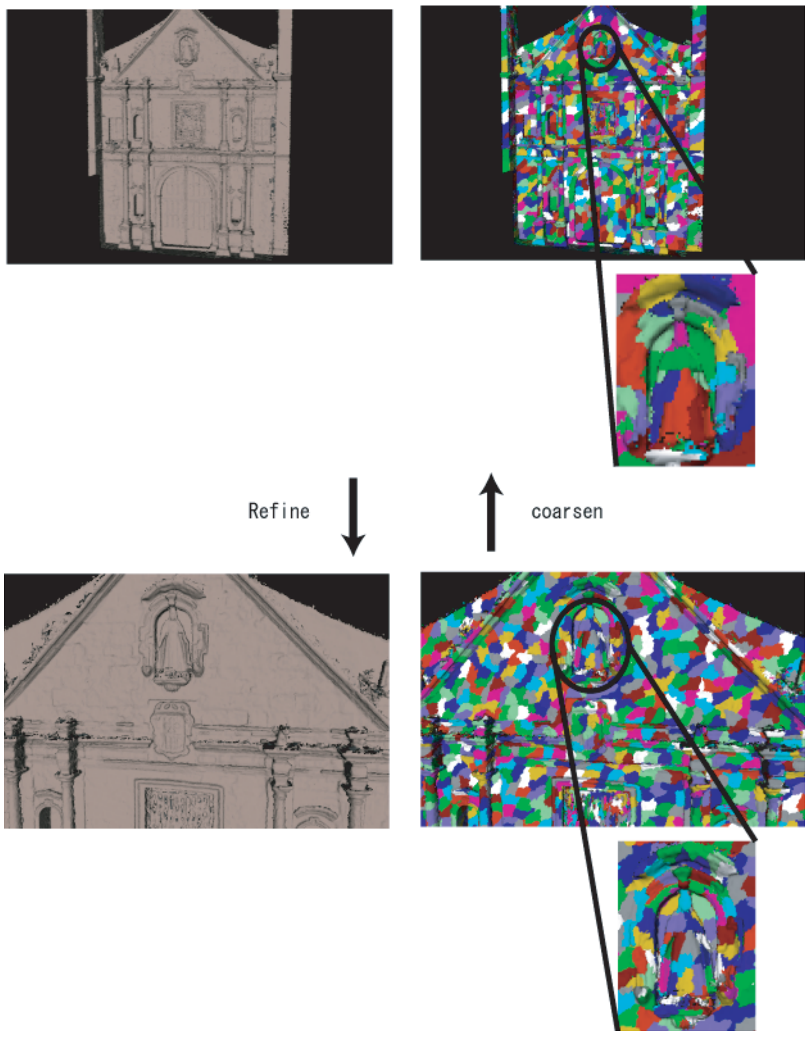


Figure 3.1: This figure shows view-dependent patch-based rendering. The right images represent the rendered patches in left scenes. The patches of the bottom scene are smaller and more refined than those of the upper scene. By controlling the density of patches depending on viewpoint, we can efficiently render any models even if it has millions of triangles.

In our algorithm, first we recursively decompose the input mesh into a pair of patches and construct the hierarchy in top-down manner. Next we compute the simplified meshes for each node by performing a bottom-up traversal of the hierarchy constructed at the first step.

### 3.1.1 Mesh Partitioning

In the first step, we decompose the input mesh into a pair of patches. In the partitioning process, a graph-cut algorithm is utilized for regularly distributing triangles into two patches. We recursively split a mesh into two patches in top-down fashion, and construct the hierarchy of meshes from an input mesh as shown in figure 3.2. When we recursively partition an input mesh, we proceed to partitioning of the patch at the next level and refine the hierarchy if the size of partitioned patch (the size means the number of triangles in the patch) exceed a predefined maximum number $N$. Otherwise, we simply stop refining the hierarchy, backtrack and continue with the rest of unpartitioned patches. This recursive partitioning continues until the size of all leaves of hierarchy is under $N$. We set 4000 to the maximum number $N$ for leaf nodes. The end result is a binary forest of small patches, and a set of triangles associated with leaf patches that cover the original mesh. The hierarchy, rather small since each node typically contains a few thousand triangles, is for efficiency reasons maintained in main memory.

In figure 3.3, we describe the pseud code of the partition algorithm. In this function, we recursively split an input mesh and record the tree structure. The final results of mesh partitioning are recorded in a repository.

We adopted graph-cut algorithm as a partitioning method proposed in [15], which enables us to split the mesh into two patches as evenly as possible, and construct balanced hierarchy, which results in saving of the data size. At the step of decomposing mesh by graph-cutting, the triangles of the input mesh correspond to the vertices of the graph, and the edges of the mesh correspond to the edges of the graph. Each edge of the graph has a weight which is the sum of the differences of colors and normals between neighboring triangles comparing the edge. By this assignment of weight, we can distribute the triangles that are close based on geometric and photometric features. The decomposition based on geometric and photometric attributes of triangles splits the mesh into meaningful parts. The geometric and photometric decomposition can be the result of 3D over segmentation described in chapter 4 of an input mesh.

### 3.1.2 Construction of Multi-resolution Hierarchy

The next step is the simplification of partitioned patches and a completion of the multi-resolution hierarchy. This resulting hierarchy contains surface representations by recursively

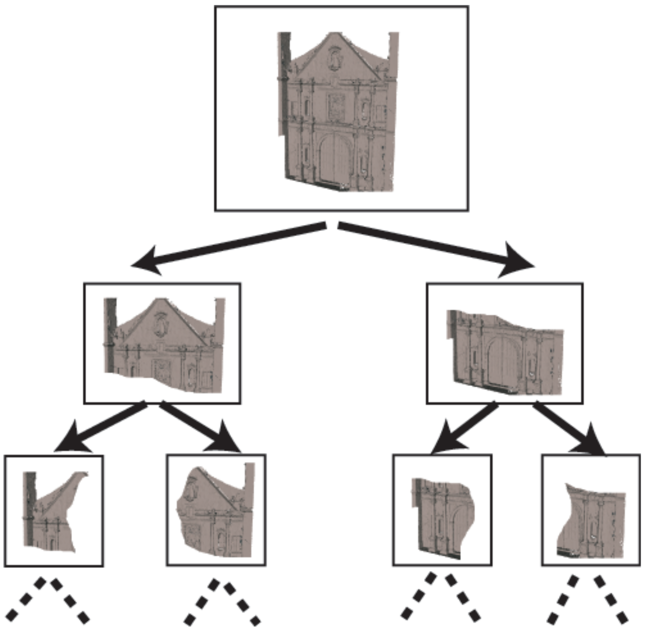


Figure 3.2: The patch-based hierarchy is constructed by recursive partitioning of the mesh. This hierarchy need less space and load than triangle- or vertex-based hierarchy.

```
TreeNode * MeshPartition(Patch $*p$)
{
    if (PatchSize($p$) $<$ $N$)
    {
        $n$ = MakeLeafNode();
        WriteRepository($n$, $p$);
        return $n$;
    }
    else
    {
        ($p_l$, $p_r$) = SplitMesh($p$);
        $n_l$ = MeshPartition($p_i$);
        $n_r$ = MeshPartition($p_r$);
        return MakeNonLeafNode($n_l$, $n_r$);
    }
}
```

Figure 3.3: Mesh partition algorithm. $MakeLeafNode()$ and $MakeNonLeafNode()$ return the pointer to node of tree. $SplitMesh()$ splits $p$ and returns two split patches. $WriteRepository()$ writes a patch with the node's pointer associated with it on the patch repository.

associating to each non-leaf patches a fixed triangle count simplification of the portion of the mesh contained in its two children. This procedure is efficiency done by proceeding in bottom-up fashion. For the leaf nodes, we retain all the fidelity of the original mesh and a representation for each leaf node is directly produced from the partitioned patches in the first step. For non-leaf nodes, we merge patches associated to the two children in a single mesh and simplify the merged mesh so that each node contains less than a predefined number of triangles. In this simplification procedure, we must constrain the edges close to the boundary not to be collapsed for the purpose of keeping the consistency between other patches.

**Merging and Simplification**

As preprocessing of simplification of each patch, we merge pathes of the children nodes. In this step, we combine two arrays storing vertices and triangles into one array, and update indices of triangle array.

After merging, we simplify the patch into one with a predefined number of triangles. On our application, we use Garland's method for simplifying meshes, described in [9], which uses a quadric error metric. This method can simplify meshes while keeping mesh quality in appearance based on geometric features of the surfaces. And it can simplify to the mesh whose number of involved triangles is a predefined number.

**The Constraint of Simplification**

Simple application of the above simplifying method cannot guarantee the quality of multi-resolution mesh. When switching rendering patches from patches at the different levels, the correct connectivity along borders of patches at different simplification levels may be not kept and holes or unnatural connection may appear close to patches' border. To avoid this inconsistency between patches at different levels, we must constrain the simplification of borders of patches to produce exactly matched patches. The border constraint in the simplification process is carried out in the following way.

When constructing a simplified representation for each node, the edges along the borders of patches are of following two possible kinds. (a) Some borders are adjacent to other patches. (b) The other borders are original borders that are ones of the original mesh. The edges in the state of (a) must not be simplified in constrained simplification; otherwise, the nonconformity between neighbor patches arises on the boundary between patches. The edges in the state of (b) do not need the above constraint since the simplification of these edges does not affect neighbor patches. In cases of boundary, to avoid the distortion of original borders, we must allow only the edges whose vertices are along original borders to be collapsed. And we do not allow the edges to be collapsed when one vertex is on the original border but the other is not.

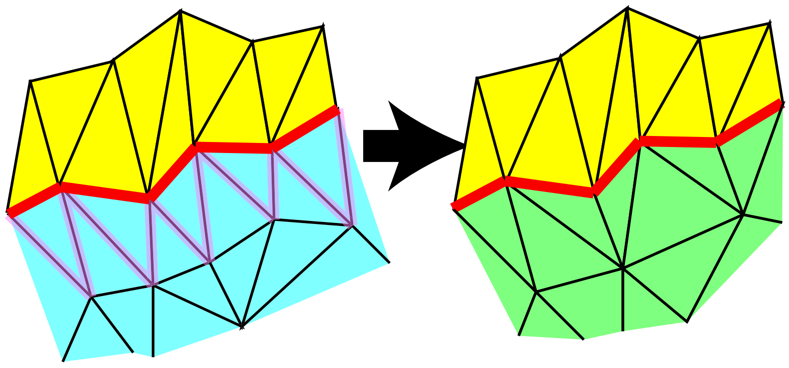


Figure 3.4: The constraint of simplification guarantees the consistency of connection between patches at different levels. The red line is the boundary edges between two patches (left image). When the green patch is simplified, the patch is constrained not to collapse the edges along boundary edges, which are red edges and pink edges. By this constraint, two patches maintain consistency after simplification (right image).

In the process of mesh simplification, we can give the above constraint to a mesh in the following way. Our simplifying method using a quadric error metric keeps a queue of collapsing edges and quadric error values in the form of a heap, and sequentially collapses edges from the top of the queue and updates the heap of edges. And the collapsing and updating procedures are continued until the number of triangles in the mesh is a predefined number. If we constrain simplification of a mesh according to the above rule, we count constrained edges out of the collapsing queue before starting the simplifying process. The initial candidate edges to be collapsed are on the condition that (1)both vertices are not on any borders of a current patch, or (2)both vertices are on the borders in the state of (b). In this way, we can generate the simplified meshes that maintain the above constraint. We show simplification under this constraint in figure 3.4.

In this constraint rule, we ensure that each mesh composed by a collection of small patches arranged as a correct hierarchy generates a globally correct surface triangulation. These constraints have little effect on overall simplification quality since the vertices are constrained when on the boundaries between other patches. After simplification of each patch, we store each simplified patch in the repository on secondary memory.

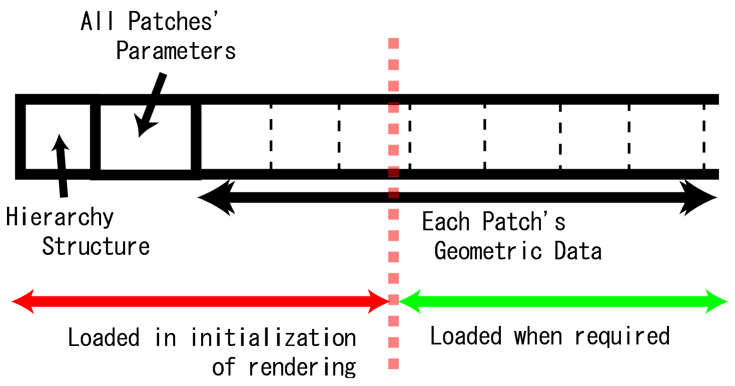

All Patches' Parameters
Hierarchy Structure
Each Patch's Geometric Data
Loaded in initialization of rendering
Loaded when required

Figure 3.5: The data structure of final format is recorded as above. First, the hierarchy structure is recorded. Second, the parameters of each node are recorded, such as center, radius of the bounding sphere. Finally, the geometric data of vertices and triangles of each patch are recorded. When initialization of rendering, our system loads the two parts and the front parts of geometric data. The rest geometric data is sequentially loaded when required.

### 3.1.3 Conversion into Final Format

At the final step of construction, we convert the hierarchy structure and simplified patches into a final format that is appropriate to be rapidly extracted and rendered.

At the first of converting step, we refine mesh of each patch into the cache coherent form. On current graphics processor, the attributes of vertices are temporally stored on the vertex cache when rendering. However, the size of vertex cache is limited to very small even if on the latest graphics architecture. That is why rearrangement of vertices and triangles to coherent form is often done for vertex cache efficiency. The most efficient approach of rearrangement for modern graphics units is indexed stripification of geometry. The topology of mesh is normally represented by polygon or triangle lists which are sequence of lists of vertices belonging to the polygons or triangles. This type of topology representation is very simple to be handled, but inefficient to be rapidly load into cache memory. This reason is that while a vertex is loaded every time of rendering belonging triangles, it may be sporadically located in the triangle list, which means the sporadically located vertices must be loaded and deleted on cache at many times. The strip topology is represented by the lists of collections of continuous stretch of triangles called as “strip” which each vertex is placed. The strategy of stripification is discussed in [8] and [23]. In our implementation of the application, we use the library [1] for rearrangement.

After mesh rearrangement, we store the mesh of the multi-resolution hierarchy in the optimal format. This format is described in figure 3.5. In this format, first, we describe the hierarchy represented by the ID number of each node. Second, we write the property of each node, such as the center, radius of the bounding sphere, error value, and pointer to the position of geometric data of associated patch. Finally, we write the geometric data, such as arrays of vertices and topologies of all patches. The geometric data of each patch is located in depth-first order in multi-resolution hierarchy for minimizing page faults and optimizing data loading. In rendering time, the hierarchy is traversed in depth-first order, and data loading also proceeds in same order. That is why depth-first alignment of geometric data is efficient to quickly load in rendering.

In figure 3.1.3, we describe the pseudo code of constructing the hierarchy. We recursively traverse the tree structure built in mesh partitioning, and merge, simplify, and convert patches into the optimized format. Each resulting patch is recorded on temporary files. With every simplification, we update patches in the repository. After all patches are produced by this function, the file in the final format is written from temporary files.

## 3.2 Multi-Resolution Rendering

The our developed algorithm of view-dependet rendering is based on a top-down traverse and refinement of the constructed hierarchy. At first we traverse the hierarchy constructed in the above section, and determine the minimum set of required patches whose simplification errors on screen do not affect the rendering quality. After that, we locate them in graphics memory, and render. In addition to this, we render some index images when users fix a camera position which are used to select specific regions described in chapter 4.

In this section, we describe the detail of rendering algorithm.

### 3.2.1 Traversing of Hierarchy

For variable resolution rendering, we implement simple stateless top-down traversals of the binary trees.

Before rendering, we load data of the multi-resolugion hierarchy and properties of all nodes in it, described in the section 3.1. This data is stored at the top of the file.

The process of traversing patch-based hierarchy does not affect the whole of the rendering process since the size of the hierarchy and time of traversing it are $O(\frac{n}{m}\log\frac{n}{m})$, where $n$ is the number of an original mesh and $m$ is the average number of patches in the hierarchy. Moreover, we can skip the process of the subtrees which are not obviously rendered; i.e., the patch are rendered out of the viewport (frustum culling), or the patch is on the back face of a rendered mesh (backface culling). When we perform recursive traversals of hierarchy, we

```
void ConstructHierarchy(TreeNode $*n$)
{
   **if** (IsLeaf($n$))
   {
      $p$ = ReadRepository($n$);
      Rearrange($p$);
      WriteFinalFormatToTmpFile($n$, $p$);
   }
   **else**
   {
      ConstructHierarchy(LeftChild($n$));
      ConstructHierarchy(RightChild($n$));

      $p_l$ = ReadRepository(LeftChild($n$));
      $p_r$ = ReadRepository(RightChild($n$));
      $n$ = Merge($p_l$, $p_r$);
      Simplify($p$);
      WriteFinalFormatToTmpFile($n$, $p$);

      WriteRepository($n$, $p$);
      EraseRepository(LeftChild($n$));
      EraseRepository(RightChild($n$));
   }
}
```

Figure 3.6: The algorithm of constructing the multi-resolution hierarchy. After finishing this function, we write a final rendering file based on temporary files. *LeftChild*() and *RightChild*() return the pointer of nodes of children. *ReadRepository* reads the associated patches from the repository, and *EraseRepository* removes them. *WriteFinalFormat*() writes the patches to temporary files in final format.

test if the current node is visible or not, by checking the location of bounding sphere of the current node on a camera coordinate, and the cone of normals of the associated patch against the current view volume. If the current node is visible, we skip its processing and that of the entire subtree and continue traversing the hierarchy. If the node is potentially visible, we test whether its patch is an accurate enough representation by measuring its saturated screen space error. If so, we can render the associated patch; otherwise we recursively traverse the node's children. The upper bound of the screen space error of rendered nodes is usually use 1 (pixel). However, when computers run with low graphics capability, we can control the rendering performance by setting the upper bound to a higher value.

The rendered patches found by the traversal are listed in the rendering list. When rendering, we load that data from the pool of patches' geometric parameters in an input file by another thread if it does not exist in a main memory.

### 3.2.2 Data Extraction and Rendering

The geometric data of all patches is stored posterior to the hierarchy data and property of each node. The data is stored in the format described in section 3.1. Before rendering and after loading hierarchical data, we read some patches close to the top of hierarchy. This initial loading enables us to smoothly start rendering. In rendering time, we load data of required patches to main memory for efficiency in memory space. If we find unloaded patches during traversals, we list the node ID on the loading list, and use ancestors of the required nodes that have already been loaded. While the main thread processes rendering, another thread is running for loading. The patches of the nodes in the loading list are loaded by this loading thread, and the rendering image is updated as soon as required patches are loaded. This loading strategy keeps the rendering speed at high frame rate. If users change a viewing direction drastically, we need to load many patches which are rendered on screen from that direction. In this case, we set the patches required to be rendered in predefined quality in the loading list, and render patches with low resolution. Since the patches with lowest resolution are loaded at initialization of rendering, we can continue rendering even if the loading of required patches is late for rendering. Additionally, to reduce such data fault, we also load the patches of parents and children of required patches by loading thread.

At the same time of loading of required patches, to avoid over-occupation of memory space, we release the geometric data of unreferenced nodes in a certain period of time. The loaded patches are registered in the list of loaded patches. In this list the unreferenced time of each patch is recorded by releasing thread at regular intervals. If one patch is used for rendering, the unreferenced time of it and the patches close to the node in the hierarchy is updated to 0. If the releasing process finds a patch whose unreferenced time is over the predefined time, the patch is removed from main memory. The release of data is processed when the loading thread is idle. In this way, we realize efficient rendering with truly essential memory.

# Chapter 4

# Region Selection by Mouse Strokes

When users assign some types of information to 3D models, they may need to define specific areas on the surfaces to be assigned to and to be used as indices. Some kinds of information are available only on the defined areas. However, on very huge 3D models with millions of polygons, it is very difficult to select specific areas efficiently. That is because users cannot draw accurate boundary lines on very complex surfaces by mouse strokes, and simple algorithms cannot efficiently and accurately select desired areas. To define these specific areas, we implement an interface to enable users to do it easily and interactively. In our system, we propose several segmentation methods, which include extended version of Lazy Snapping [21] method to 3D surfaces based on [29].

In this chapter we describe processes of the interactive segmentation algorithm.

## 4.1 Base Selection Method

At first, we introduce the base method for selecting specified regions.

The most general method for selection objects in computer applications is enclosing them by mouse dragging similarly to lasso. The 2D regions selected by lasso are projected on 3D space, and we determine the selected 3D regions. We use the selection of 3D regions by using the result of 3D pre-segmentation. Additionally, the selecting results affect the upper and lower patches in the hierarchy.

### 4.1.1 2D Selection Procedure

Our system allows users to select regions by simple and interactive strokes. Users can add and remove regions by a stroke similar to the lasso. Users draw freehand curves by mouse dragging

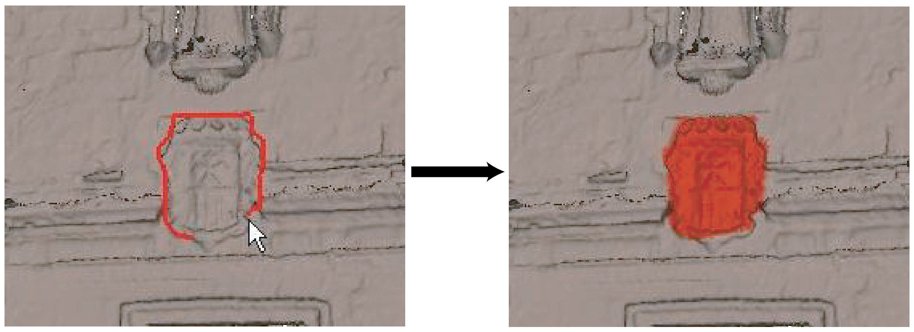


Figure 4.1: This figure shows the selection operation by lasso and the result. Users can select the desired region by dragging and drawing a red marker, as shown in this figure.

to selectively add and remove regions to be select. After selecting by lasso, the 3D regions on the surface corresponding to enclosed 2D regions are selected (details are described in the next subsection). Additionally, users can edit regions many times from different viewing angles. By using this simple selection method, users can select the region not rapidly but faithfully to their intention, even if the region has a complex boundary. We also allow users to modify the segmentation results, if they are not satisfied with them, using the following semi-automatic segmentation method.

### 4.1.2 3D Selection Procedure

After users select 2D regions by lasso, we reflect the selecting results to the target model.

When selection of a specific region is finished, the region is highlighted on the renderer. If users are satisfied with it and finalize marking, we reflect the result in a 3D model. Then, we back-project the 2D region selected by lasso onto the 3D space and assigned specified IDs to selected vertices.

However, if the vertex is in the space of a selected 2D region but the 3D segment which the vertex belongs to in is not rendered in the selected 2D region on screen space, the vertex is not selected. This constraint enables us to avoid selecting vertices which are apparently away from the parts on 3D models users hope to select. For example, the surface occluded by another surface and the surface normal whose direction is away from view direction, which are not rendered on screen, are not regions likely to be selected by users, so we do not select the vertices on these surfaces. To solve this problem, we use the result of 3D segmentation before the selection procedures. On our system, we use the “index image” of rendered patches in the multi-resolution hierarchy as the result of segmented 3D model that is rendered by IDs

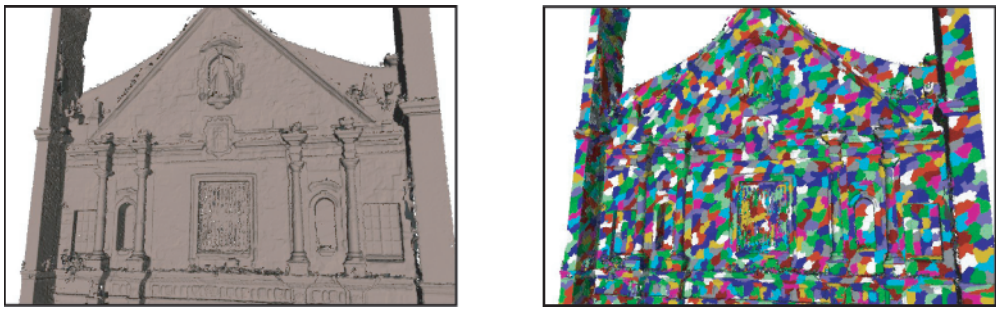


Figure 4.2: Pre-segmentation of the 3D model. We can obtain this result by using an index image in which the IDs of patches are rendered on the screen. This image is rendered on the back buffer.

of rendered patches instead of object color. The mesh partition in construction of multi-resolution is done based on geometric and photometric features of surfaces. Therefore, by using this result, we also narrow the vertices to be selected based on the features of the target model. The index image is rendered on a background buffer at the time of rendering when users fix a camera position. By the use of index images, the system can rapidly specify the rendered patches on screen. We select only vertices belonging to the patches with the ID included in the selected region on the index image.

We also select vertices of the ancestors' and children's patches to cover cases where the rendered resolution level change is dependent on viewpoints. If this procedure is not done, we cannot recognize the correct selected region when we change viewpoints and rendered patches changed. Since we store data of the hierarchy structure, this procedure can be easily done by traversing of the hierarchy.

## 4.2 Lazy Snapping Selection

The selection by lasso is very simple, but users cannot select a complex region quickly and accurately by the method. Therefore, our system allows users to select regions by semi-automatic procedure like the Lazy Snapping method.

Lazy Snapping is the system of 2D image segmentation that enables users to split images by easy mouse strokes. Users draw two types of lines on an input image by dragging the mouse cursor, one type of lines is drawn on the area users want to set as foreground, and the other is drawn on the background area. After users finish drawing a line, the system automatically draws boundary lines and splits an imput image into foreground areas and background areas. If users are not satisfied with segmentation results, they can add lines corresponding to foreground or background until they get desired results. In this system, to automatically segment 2D images, a novel interactive graph cut algorithm is used.

### 4.2.1 Segmentation by Graph Cut

This image segmantation problem equals a binary labeling problem. In the labelling problem, the image is represented by a graph $G = (V, E)$ where $V$ means the set of all nodes and $E$ means the set of all connectivity of adjacent nodes. The goal of this labeling problem is to assign all $V$ to either label $F$ representing foreground or label $B$ representing background with minimum energy. In the case of 2D images, $V$ corresponds to pixels of an input image, and $E$ corresponds to relationships with connections between four or eight neighbor pixels.

The labeling problem is to assign a unique label $x_i$ for each node $i \in V$, i.e. $x_i \in foreground(= 1), background(= 0)$. The solution $X = x_i$ can be obtained by minimizing a Gibbs energy $E(X)$ [10]:

$$E(X) = \sum_{i \in V} E_1(x_i) + \lambda \sum_{(i,j) \in E} E_2(x_i, x_j) \tag{4.1}$$

where $E_1(x_i)$ means energy, encoding the cost when the label of node $i$ is $x_i$ , and $E_2(x_i, x_j)$ denoting the cost when the labels of adjacent nodes $i$ and $j$ are $x_i$ and $x_j$ respectively.

After users draw two types of lines, the pixels intersecting the foreground and background lines are defined as foreground seeds $F_{seed}$ and background seeds $B_{seed}$.

In equation 4.1, $E_1$ means color similarlity of each node, indicating if it belongs to the foreground or background. $E_1$ is computed in the following way. At first, the colors in $F_{seed}$ and $B_{seed}$ are clustered by K-means method. Each mean color of the foreground and background clusters are represented by $\{C_n^{F_{seed}}\}$ and $\{C_n^{B_{seed}}\}$. For each node $i$, the minimum distance to foreground and background clusters is computed from the color $C(i)$. The minimum distance to foreground clusters is denoted as $d_i^F = \min_n \|C(i) - C_n^{F_{seed}}\|$, and to background as $d_i^B = \min_m \|C(i) - C_m^{B_{seed}}\|$. From these equation, $E_1(x_i)$ is defined as follows:

$$\begin{cases} E_1(x_i = 1) = 0 & E_1(x_i = 0) = \infty & \forall i \in F_{seed} \\ E_1(x_i = 1) = \infty & E_1(x_i = 0) = 0 & \forall i \in B_{seed} \\ E_1(x_i = 1) = \frac{d_i^{F\,seed}}{d_i^{F\,seed} + d_i^{B seed}} & E_1(x_i = 0) = \frac{d_i^{B seed}}{d_i^{F\,seed} + d_i^{B seed}} & \forall i \in U \end{cases} \tag{4.2}$$

where, $U$ means nodes intersecting without any markers, which is represented by $U = V \setminus \{F_{seed} \cup B_{seed}\}$. The first equation guarantees that all nodes on $F_{seed}$ are always on foreground. The second equation guarantees that all nodes on $B_{seed}$ are on background. The third equation describes the case of nodes that have a label with similar colors to foreground or background.

$E_2$ represents the energy of the gradient along the boundaries. $E_2$ is defined as the color gradient between $i$ and $j$ in the following equation:

$$E_2(x_i, x_j) = |x_i - x_j| \cdot g(C_{ij}) \tag{4.3}$$

Here, $g(c)$ and $C_{ij}$ are denoted as $g(c) = \frac{1}{c+1}$ and $C_{ij} = \|C(i) - C(j)\|^2$ which is the L2 norm of the RGB difference between pixel $i$ and $j$. If node $i$ and $j$ are assigned to the same label, $E_2$ is zero. This shows that $E_2$ is a penalty term when adjacent nodes are assigned to different labels; the more similar the colors of the two nodes are, the larger $E_2$ is, and the less likely the edge is on the boundary. To solve the problem of minimizing $E(X)$, the min-cut/max-flow algorithm described in [5].

When $E(X)$ is minimized, the set of nodes labeled $x_i = 1(foreground)$ provides the most likely foreground region for users. Users can utilize the resulting segmentation to change the background of the image, to recognize the specific object, and for other purposes. If users are not satisfied with the result, they can add and remove markers and update the result of the segmentation.

### 4.2.2 Extended Lazy Snapping (3D Image Segmentation)

Next, we describe the extension of Lazy Snapping to the version for 3D images. This interactive 3D segmentation method is based on [29]. The 3D segmentation is processed in three steps. The first step, 2D over segmentation involves preprocessing for efficiency of segmentation algorithm. The remaining steps are image segmentation and final 3D segmentation, which cuts specific regions from an input 3D model.

#### Over Segmentation of 3D and 2D images

In the case of Lazy Snapping, similarly to the base selection method, we also need 3D pre-segmentation. We can obtain the result by using the index image of IDs of rendered patches.

Next, we split a rendered 2D image into small clusters. This over-segmentation is done for the efficiency of the graph-cut algorithm described in section 4.2.1. When building a graph $G$, it is less complex to use small clusters than to use pixels. In this case, the nodes of $G$ correspond to small clusters, and the edges of $G$ are the set of all connections between adjacent clusters as shown in figure 4.3. $F_{seed}$ and $B_{seed}$ are also assigned to clusters instead of pixels.

This over segmentation is done when users stop a camera position on the renderer and a rendered image is fixed. We perform this segmentation by using watershed algorithm [27] shown in figure 4.4. After this 2D over segmentation, we form a graph $G(V, E)$. Every edge

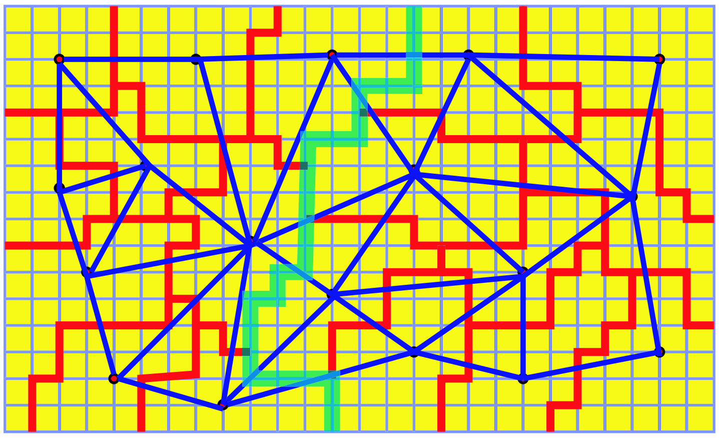


Figure 4.3: This image describes over segmentation. For efficiency, the graph $G$ is constructed from small regions obtained by pre-segmentation instead of pixels.

in $E$ has a weight function $C_{ij}$, which is set to the difference of mean color between adjacent clusters used in Equation (4.3). To do this 2D over segmentation and construction of a graph $G(V,E)$ requires several seconds. To avoid delay of processing by this wasted time, we run this process on background of a main thread that processes user interactions such as drawing markers.

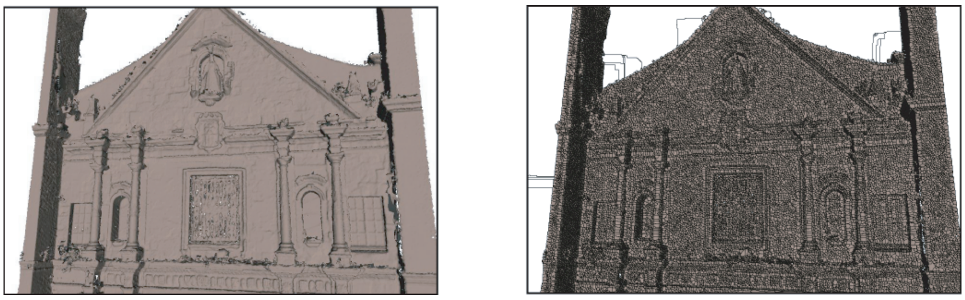


Figure 4.4: Over segmentation of the 2D image as an initialization of segmentation. For the 2D over segmentation, we use the watershed algorithm.

**2D and 3D Selection by Lazy Snapping**

After users fix a camera position, they can draw stroke markers representing foreground or background. When users finish drawing a marker, the system starts an image segmentation procedure. This segmentation corresponds to the graph-cut segmentation described in 4.2.1.

The formulas used in this step are almost same as those used in 4.2.1, but they modify the several points for extension to 3D models. In $E_1(x_i)$ of Equation (4.2), the color of each node is used. In the 3D Lazy Snapping method, we also use normals for energy function. The modified energy function $E'_1(x_i)$ is as follows:

$$E'_1(x_i) = w_c * E^c_1(x_i) + w_n * E^n_1(x_i) \tag{4.4}$$

where $w_c$ and $w_n$ are weight functions for color and normal terms. $E^c_1(x_i)$ equals $E_1(x_i)$ in equation (4.1). and $E^n_1(x_i)$ is the energy function for normals in which $d^F_i$ and $d^B_i$ is represented by the following equations: $d^F_i = \min_n \|1 - (N(i) \cdot N^{F_{seed}}_n)\|$, $d^B_i = \min_n \|1 - (N(i) \cdot N^{B_{seed}}_n)\|$. $N(i)$ means average normal of segment $i$, and $N^{F_{seed}}_n$ and $N^{B_{seed}}_n$ are initial average normals of foreground and background. This extension of the energy function enables us to segment rendering images based on geometric features of rendered objects.

Additionally, $E_2(x_i, x_j)$ in equation (4.3) must also be modified similarly. The modified penalty function $E'_2(x_i, x_j)$ is denoted as below forms:

$$E'_2(x_i, x_j) = |x_i - x_j| \cdot \{w_c * g(C_{ij}) + w_n * g(N_{ij}) + w_d * g(D_{ij})\} \tag{4.5}$$

In this equation, $w_c$, $w_n$, $w_d$ are weight functions, and $N_{ij}$ and $D_{ij}$ mean the differences of normal and projected depth between segments $i$ and $j$.

The normal $N_i$ of each segment is calculated in the following way. The result of 3D over segmentation is rendered on a background buffer when users fix the camera position. As 2D segmentation is done and the graph $G(V, E)$ is constructed, we specify 3D segments included in each 2D segment. The normal $N_i$ is calculated as a weighted average normal included in the 3D segment, which is stored in rendering hierarchy structure. In the case of depth value $D_i$, we can easily obtain it. The depth values of all pixels are also output on a back buffer, and $D_i$ of each segment is calculated as the average value of pixels belonging to the segment. $N_{ij}$ and $D_{ij}$ are obtained from $N_i$ and $D_i$.

After image segmentation, the segmentation result is also displayed on renderer as the case of base selection method. If users are satisfied with it and finalize marking, we reflect the result in a 3D model as described in section 4.1.2. Otherwise, users continue the drawing markers until they obtain the desired result. We show the process of drawing markers and segmentation results in figure 4.5.

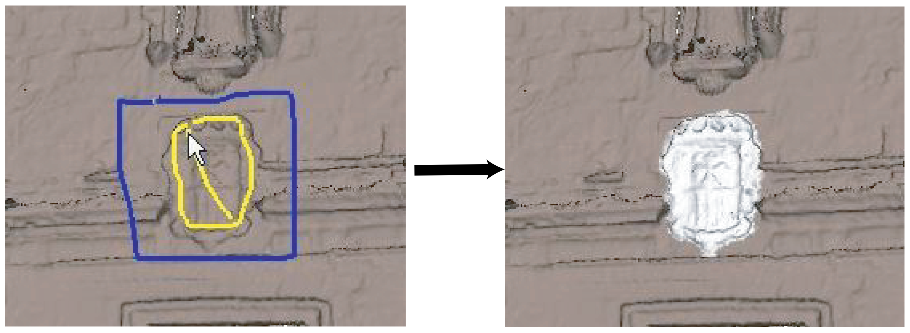


Figure 4.5: This figure shows the result of Selection by 3D Lazy Snapping. Yellow markers represent the foreground region, and blue markers represent the background region. In the result images, the relief on the facade is selected.

As the results of this segmentation, users may obtain larger or smaller regions than they hoped to select. In the case, users can modify the range of selected regions by using the lasso method described in section 4.1.

## 4.3 Extension Method

We think that Lazy Snapping method is not always available, and it is not necessarily the case that the method is intuitive for every user. The reasons is because users must draw multiple lines in order to select a small region, and because it is difficult for novices to foretell a resulting region after drawing strokes. Therefore, we propose extended method that allows users to select regions intuitively by simple strokes like a lasso tool and fast and accurately by using Lazy Snapping algorithm.

### 4.3.1 Robust Lasso Tool

Our proposed selecting method has combined a traditional lasso tool and Lazy Snapping. That is why users can obtain desired regions by using a lasso tool at one time. In the case to select by Lazy Snapping interface, users cannot obtain them at one stroke, on the other side, they can accurately obtain desired regions. Our proposed method combines the intuition of a lasso tool and accuracy of the Lazy Snapping.

This selecting method proceeds the following procedures. At first, users draw a heavy marker along the region that users want to select, as shown in figure 4.6. This operation is intuitive and easy like a lasso tool, but a heavy marker allows users to more easily and speedy

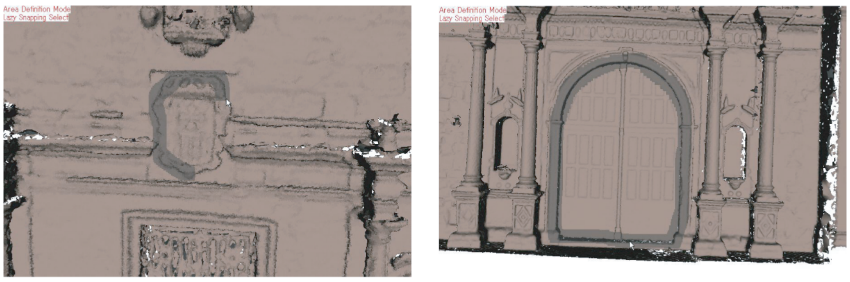


Figure 4.6: This figure shows the selection operation by a heavy lasso. Users track the boundary of the desired region by drawing a grey heavy marker. Users can easily and fast do it because of the robustness of the heavy marker.

track the boundary of a desired region than a lasso bool.

Nextly, after users finish tracking boundary, our system automatically draws a yellow marker and a blue marker which mean foreground and background. In figure 4.7, the drawn markers are shown, the yellow marker is drawn along the inside of a heavy marker, and the blue marker is drawn along the outside. By this automatic drawing and applying Lazy Snapping algorithm, users can obtain a desired region which is close to the one that they want to select. After that automatic drawing, our system displays the resulting region selected by graph-cut algorithm of Lazy Snapping as shown in figure 4.8.

Lastly, if users are not satisfied with the boundary of the resulting region, users can add yellow and blue markers same as Lazy Snapping and adjust the boundary.

By this method, users can obtain desired regions fast, robustly and accurately.

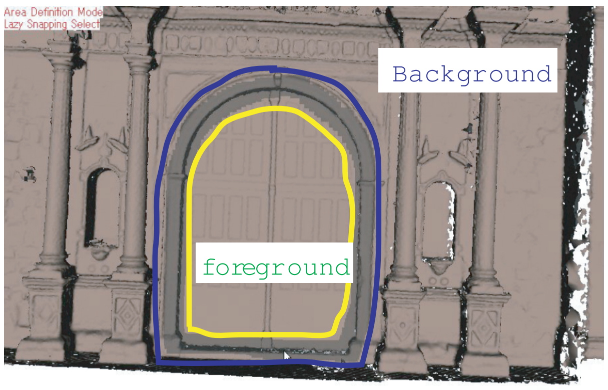


Figure 4.7: This figure shows the automatic drawing markers representing foreground and background. Our system draws a yellow marker inside a heavy line, and a blue marker outside of a heavy line.

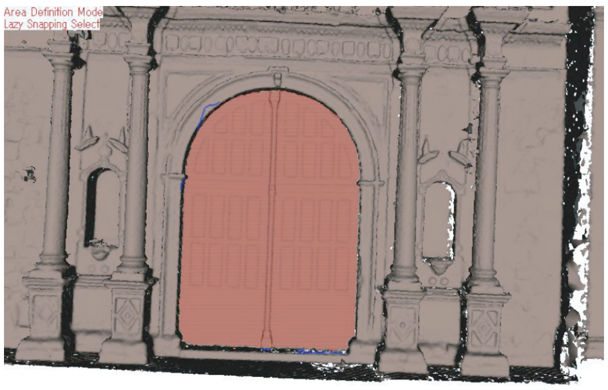


Figure 4.8: This figure shows the resulting region selected by our proposed method.

# Chapter 5

# Design of Interface

The most unusual feature of our system is the function of interactive construction of the database which combines a large amount of information with huge 3D models. The construction of the database is composed by selecting specified regions used as indices and assigning information that is a file or an item in other database. Both selecting and assigning can be done by simple mouse actions on the renderer.

Also, users can view information in the constructed database. Users can view the 3D model in real time, and at the same time they can access the item located on surfaces of the object.

In this chapter we describe the design of interfaces and how to operate our system.

## 5.1 Assignment of information

For assignment of information to 3D models, firstly users specify the desired regions. Users can select two selection modes, which one is 3D Lazy Snapping mode, and the other is selection by lasso. 3D Lazy Snapping selection is described in section 4, and selection by lasso is a simple method of enclosing the desired region by mouse dragging.

### 5.1.1 Region Selecting Operation

Viewing a 3D model on the renderer and finding the region which users hope to use as an index, users fix a camera position and choose the selection mode.

In the case of base selection by lasso, users draw freehand curves along the boundary line of the region users want to select. In this mode, users can draw a red marker by dragging while holding the left mouse button and enclosing the region with the marker. Releasing the mouse button, our system proposes region to be selected. Users can determine the resulting region by double-click. Or they may cancel the result and re-select. In addition to selecting, in this mode users can cut unwished parts from already specified regions. This cutting by lasso can be used by dragging with holding the right button down and drawing blue markers. By using cutting markers, users can modify specified regions that are selected inaccuracy because of the limits of the segmentation algorithm or operational mistakes, and can remove regions no longer required.

In the case of 3D Lazy Snapping, users can draw yellow markers that signify foreground regions by dragging while holding the left mouse button down, and blue markers that signify background regions by dragging while holding the right button down. When users stop drawing a marker and release a button, our system exhibits the result of computing of segmentation. If the result does not satisfy users, they can add markers again and update the segmentation. Otherwise, as in the case of the base selection method, users double-click on the window and determine the segmented region.

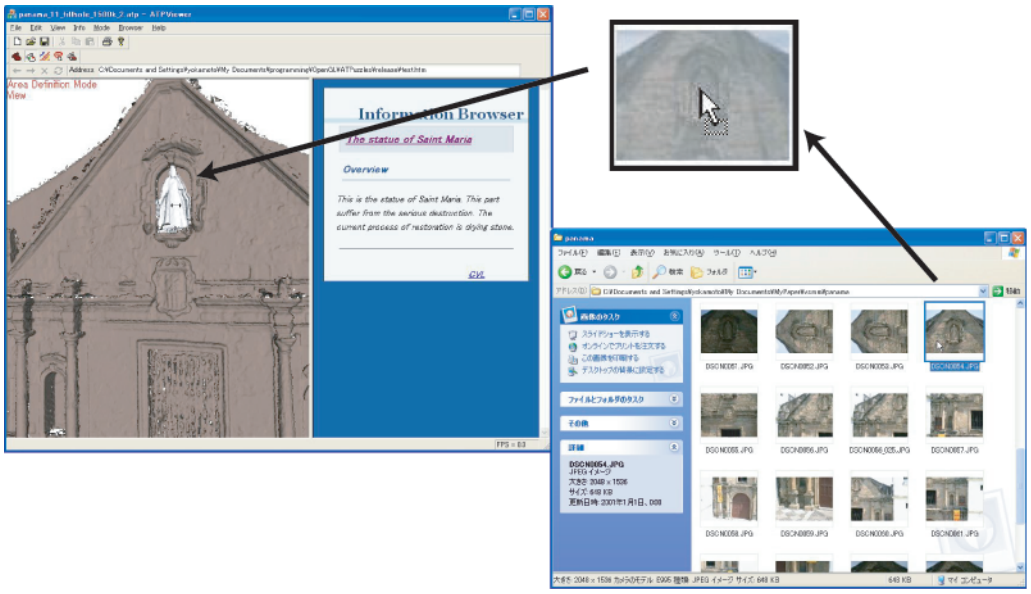
### 5.1.2 Assignment Operation

After selecting regions, users associate information with the 3D model. On our system, we assume the format of information to be text documents, numerical values, image files, and links to web pages and other files. Users can pick data in those formats from the database in advance, or they can select files stored in secondary memory. The associated data is stored in the database and is mapped to the target 3D model.

Our system displays tables of data in the database in the dialog window so users can select the data they want to associate to the model. Users can select the item and move it from the table to the target region by drag-and-drop. After that, the dropped items are copied to the database of the target 3D model.

In the case assigning data in secondary memory, our system allows users to pick the data in any file format from an *InternetExplorer*$^{©}$ window by drag-and-drop as shown in figure 5.1. In this assigning mode, assigned data is not copied to the database entirely; rather, only the address of the data is stored in database. Provided that the format of assigned data is a text document or an image file, our system automatically generates and stores the text header as the caption of the item, or the thumbnail of the image.

After assigning data, the label of each item is automatically stored in the database. The resulting database and the other additional data, such as labels, captions, and thumbnail, are recorded in the same location as the file of the target 3D image.

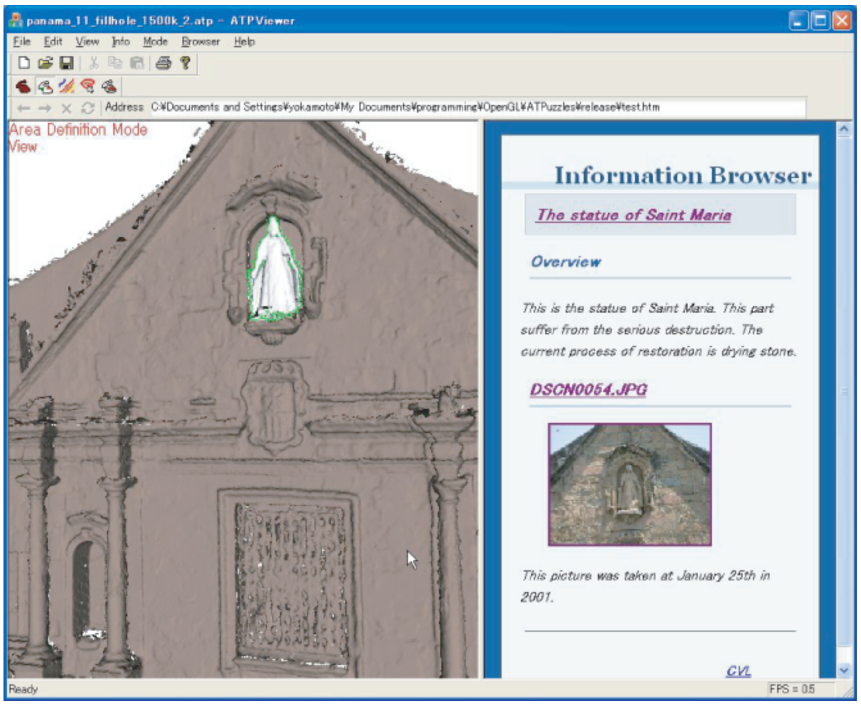

Information Browser

*The statue of Saint Maria*

*Overview*

*This is the statue of Saint Maria. This part suffer from the serious destruction. The current process of restoration is drying stone.*

Figure 5.1: This figure shows the procedure of information assignment to a 3D model. Users can assign items by drag-and-drop from *InternetExplorer*© windows or database tables to specified regions. We allow users to assign the item in any format.


panama_11_fillhole_1500k_2.atp - ATPViewer
File Edit View Info Mode Browser Help
Address C:¥Documents and Settings¥yokamoto¥My Documents¥programming¥OpenGL¥ATPuzzles¥release¥test.htm
Area Definition Mode
View
Ready
FPS = 0.5

Information Browser

The statue of Saint Maria

Overview

This is the statue of Saint Maria. This part suffer from the serious destruction. The current process of restoration is drying stone.

DSCN0054.JPG

This picture was taken at January 25th in 2001.

CVL

Figure 5.2: The right part of the system window is the information browser. This displays assigned information. By clicking links to web pages and image files, users can jump the linked pages on this browser.

## 5.2 Display and Edition of Information

After associating information with a 3D model, users can access it through the model on the renderer. Using a mouse cursor on the surface of the embedded information, the user can preview the data displayed. This preview helps users know contents assigned to the region. If they find the desired region when viewing the target 3D model, users can open the browser on this system and display the embedded information on it by double-click on the region as shown in figure 5.2.

The information browser is implemented to support HTML format. The assigned information page is also proposed in the format with thumbnails of assigned image files. Therefore, if the information includes links to a site on the web, users can view the linked pages on the browser by clicking the links. The system also allows linking to files whose file format is not HTML by opening the application associated with the file format.

Each assigned item has a label and caption that lets users have a detailed explanation of the data and its relationship with the 3D model. The labels and captions can be editted through the browser. Additionally, editing can be used to remove assigned items.

# Chapter 6

# Evaluation

In this chapter, we evaluate the performance of our rendering system and database system. In the evaluation for rendering, we verify the rendering frame rates, and the memory consumption. And we test the usability of our system in operations of selecting 3D regions and assigning information.

## 6.1 Application Domain

### 6.1.1 Mercede Church

Our main target is to build a system for Mercede Church in Casco Antiguo district of Panama City.

This historic district is listed in the World Heritage catalog as “Panama Viejo and Historic District of Panama”. Founded in 1519 by the Spanish conquistadors as a fort city, it is the oldest settlement on the Pacific coast of the Americas. It was laid out on a rectilinear grid and marks the transference from Europe of the idea of a planned town. Mercede Church was built in Panama Viejo in the 17th century.

In the middle of 17th century, Panama Viejo was attacked by pirates, and greatly damaged. It was abandoned by its inhabitants, who replaced it with a new town near the current Panama city.

The historic district in current Panama City has as its center a main plaza that contains historically and administratively important buildings such as the Presidential Office and Cathedral. Mercede Church is also one of important buildings in Casco Antiguo. This church was saved in its original form and moved to the new town. It is now one of the few buildings built in Panama Viejo age, and the one of the historically most important buildings.

### 6.1.2 Restoration Project

Recently Mercede Church has been faced with a critical problem. The deterioration of the stone of wall, which many of those important buildings are made of, has become apparent. The stone is deteriorating mechanically from its weight and chemically by climate condition such as rain, wind, and sunlight. To avoid derioration and save these buildings, activity for restoration started by the cllaboration between National Research Institute for Cultural Properties (NASRIPT), Tokyo, and Instituto Nacional De Cultura (INAC). Many researchers from Panama and Japan are perticipating in this project and are planning and carrying forward the restoration of Mercede Church. In February 2002, we acquired 3D data of the church facade as a part of the restoration project for the purpose of obtaining a 3D model of the whole of the facade. In this activity, researchers have accumulated various kinds of data such as the conditions of deterioration and plans for restoration formatted by text, images, and so on. Additionally, our research group also scanned the 3D shape of the church facade, and built the 3D models. We used a Cyrax2400 range scanner [4] to scan surfaces, and obtained 26 range images. After aligning and merging the range images, we found that an obtained final model of Mercede Church has 1,168,813 vertices and 2,064,537 polygons.

## 6.2 Implementation

We implemented the system using a novel rendering methods described in chapter 3; additionally we have optimized our implementation for efficiency in processing, rendering, and saving memory. We have implemented and verified our system on commodity PC, for rendering on an ATI Radeon 9800 XT, processing on a 3.2 GHz Intel Pentium4 with 2GBytes of RAM. Our system runs on Windows XP. It has been implemented using C++ with OpenGL. In construction of the multi-resolution meshes, we split patches by using the METIS graph partitioning library [15], and we optimally rearrange the order of the triangles by NvTristrip library [1]. And we simplify patches by using Garland's simplification method (QSlim) [9]. And the interface of my application is implemented by MFC. For the browser displaying information we use $InternetExplorer^{©}$ components and it supports HTML format.

In the following sections, we describe how we verified the performance. We verify the processing speed and memory efficiency in preprocessing and rendering. Also, we test the usability of region selection and assignment of information.

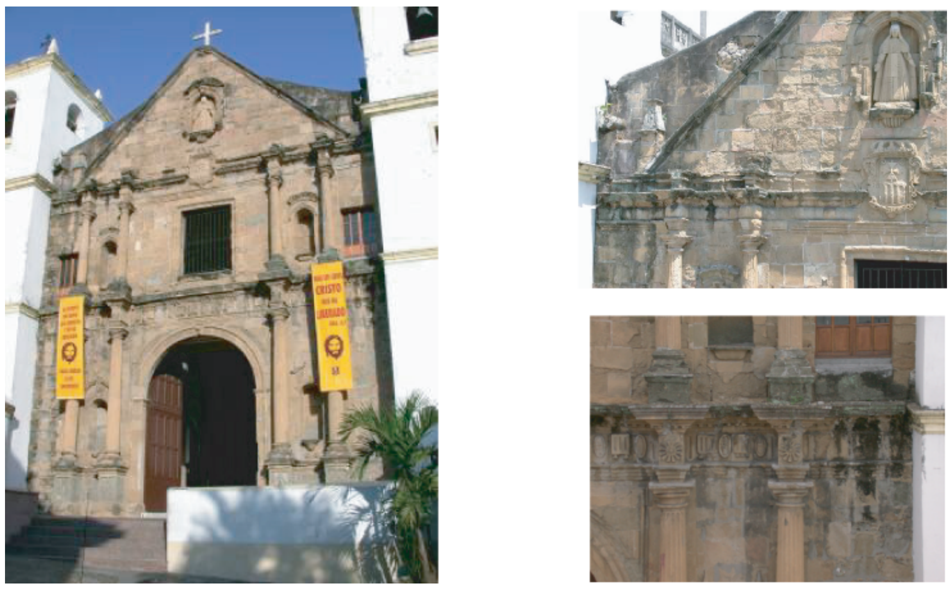


Figure 6.1: The left picture shows Mercede Church, one of the oldest buildings in Panama City. The facade of this church has deteriorated and been cracked by wind and rain, as shown in the right images.

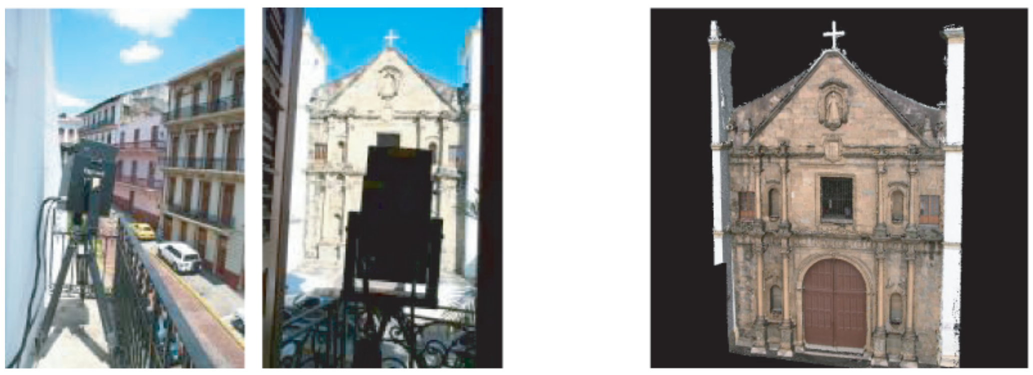


Figure 6.2: The left two images shows the scenes of scanning of Mercede Church in 2001. The right image is the rendering result of the scanning model with textured color.

| Model name | Vertices | Triangles | Processing Time[sec] | IO[sec] | Total time[sec] |
|---|---|---|---|---|---|
| Stanford Bunny | 35947 | 69451 | 12.6 | 0.452 | 13.1 |
| Armadillo | 172947 | 345944 | 73.2 | 0.999 | 74.2 |
| Happy Buddha | 543652 | 1087716 | 281 | 1.09 | 292 |
| Mercede Church | 605579 | 1148939 | 316 | 23.3 | 340 |
| Nara Daibutsu | 1561868 | 3109824 | 788 | 30.1 | 818 |
| Kawara | 2966475 | 5593696 | 1530 | 65.3 | 1595 |
| Bayon Face | 3031197 | 5922790 | 1592 | 49.4 | 1641 |
| Bayon Towers | 5001122 | 9343426 | 2874 | 181 | 3055 |

Table 6.1: Preprocessing performance: The size of input mesh and constructing time for each model.

## 6.3 Rendering System

We verify the performance of preprocessing and rendering for huge models. To evaluate them, we use the Mercede Church and other huge models. They cannot be interactively rendered by traditional polygon rendering, but we can easily render them regardless of the size in our rendering system on commodity PCs. Additionally, we compare the performance of the method that release unused data with that does not release, and we observe the difference of the efficiency in processing speed and memory space.

### 6.3.1 Preprocessing

First, we show the results of preprocessing in table 6.3.1. The construction of the mesh that has about one million triangles costs less than ten minutes. Also, the construction for mesh with ten millions triangles needs about one hour. These results are within the allowable range in terms of practicality, but the processing time will explode as the larger mesh is processed. However, the construction of patch-based hierarchy can be computed in parallel such as partitioning and simplification. The reason is why each patch can be processed independent of other patches' results by constraint of borders. We can implement the construction algorithm optimized for parallel computing, and can construct the multi-resolution models of those input meshes. This is a future work.

### 6.3.2 Rendering

We show the results of rendering some large models in table 6.3.2. From these results, while the rendering frame rate decreases as the size of an input mesh is larger, its average frame rate maintains over 10 fps for larger meshes whose original model has about ten millions polygons. The percentages of loaded data size represent the rates to the entire data size. In the case of the smaller meshes, the percentages are high since the hierarchy of multi-resolution is smaller and the data access can reach all patches. In the case of larger ones, the percentages are low since all data need not to be loaded and accessed. Additionally, we observe the numbers of rendered triangles per second are between 15M and 30M units, and exceed 100M units at peak points.

Next, we evaluate the memory efficiency of our system. In figure 6.3.2, we show the chart representing transition of frame rates and the size of used memory. The red lines represents the performance of the case by using release of unused data, and green's represents that of the case by not using it. In this experiment, we use the 3D model of Bayon Towers (the bottom of table 6.3.2) whose original mesh has about ten million polygons. The upper chart represents the transition of frame rates when viewing the model by two methods at same path. And the bottom one is the transition of the size of loaded patches' data then. We observe that in both cases the frame rates decrease when the loaded data size increase. But both frame rates are transitting between 10 and 25 fps, and the performance in both cases is not so different. On the other sides, we observe that the memory performance in both case has great difference. The memory size in the green case monotonically increases and reach over 400 Mbytes since the loaded data is not released. While it, the memory size in the red case decrease at some points and it appears to fall into about 200 MBytes. From this result, we verified that our system can save the memory space without the loss of rendering speed except at the time that the viewpoint drastically changes and data loading increases.

## 6.4 Usability Study

We believe that our system helps users efficiently deal with 3D models and other information. To test the usability such as selection of our system, we have conducted a usability study. In particular, to evaluate selecting methods, we compare the time efficiency and accuracy of base selecting method using lasso, 3D Lazy Snapping method, and our proposed selection method.

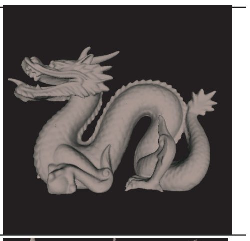
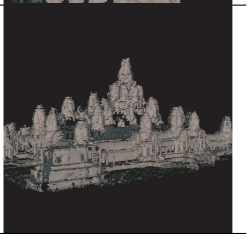
| model | input polygons | average frame rates | average loading size (percentage) | rendered triangles (units/sec) | |
|---|---|---|---|---|---|
| Dragon | 871,414 | 31.2fps | 27.3MB (95.2%) | 16154K |  |
| Mercede Church | 1,148,939 | 32.8fps | 46.3MB (85.9%) | 17900K | |
| Nara Daibutsu | 3,109,824 | 24.7fps | 94.3MB (74.7%) | 18947K | |
| Bayon Face | 5,922,790 | 13.8fps | 123MB (49.6%) | 29923K | |
| Bayon Towers | 9,343,426 | 16.4fps | 193MB (44.3%) | 25384K |  |

Table 6.2: The rendering performance

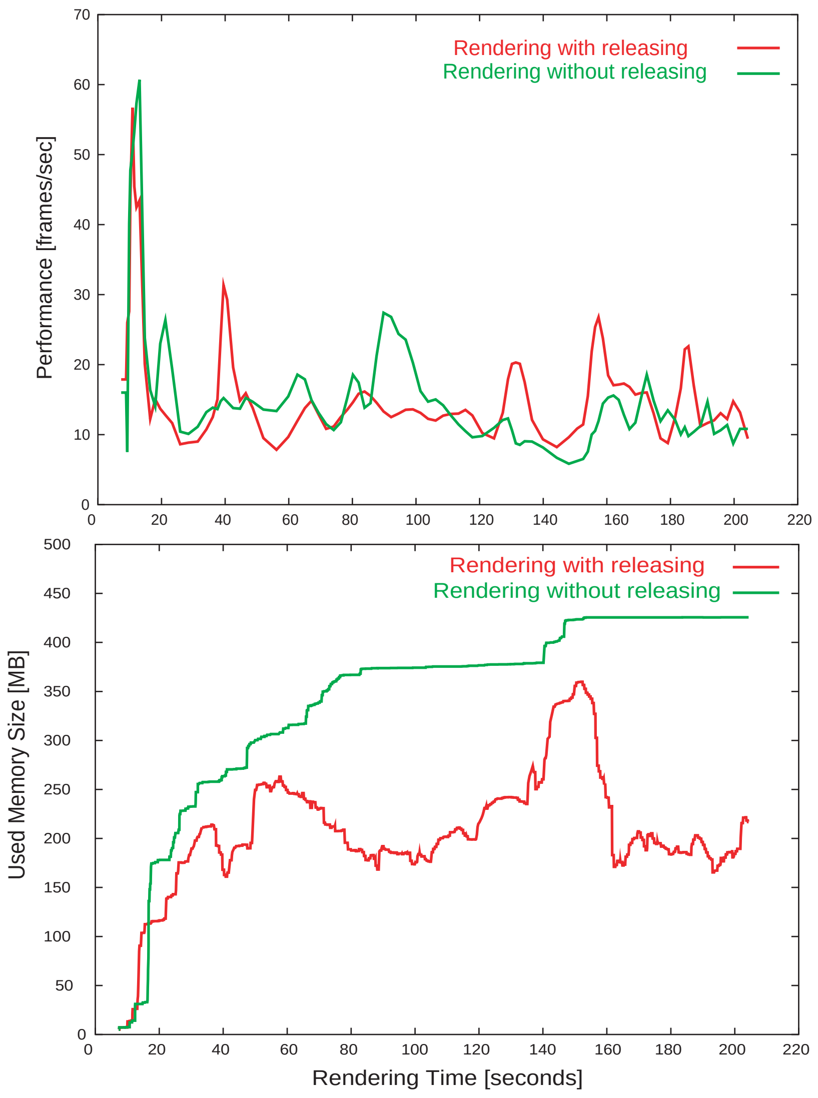


Figure 6.3: The efficiency of memory space. The upper chart shows the transition of the frame rate, and the bottom one shows the size of using memory.

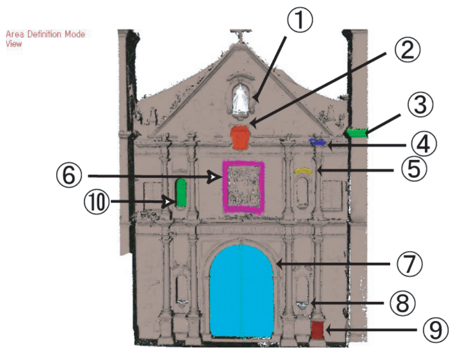


Figure 6.4: These regions are selected in the user study. Users must select those regions in order.

### 6.4.1 Methodology

To evaluate selecting methods, we select eight subjects. Two subjects are novices that have little experience with tools dealing with 2D and 3D image. And other two are experts who are good at using mouse strokes and operating images. The other four people are intermediate users who have experiences with such tools but are not experts. Before we give subjects the instruction session on our system operation.

The study of selecting operations have two times. We give subjects ten region selecting tasks as shown in figure 6.4.1. Subjects do the assignment by three selecting methods which are base selecting method, 3D Lazy Snapping method, and our proposed method. The order of selecting methods is alternated between subjects, with half of the subjects using the base method first, and the other half using the 3D Lazy Snapping and our proposed method to verify that there is an ordering effect.

We record the subject's operation by a video camera, and save the selecting results. By using the resulting data, we evaluate the time to be cost and the quality of selecting results.

### 6.4.2 Results

In this subsection, we introduce users' review, and evaluate the operating time and error of their selecting results.

#### Users' Review

After tests, we interviewed subjects about the usability of all selecting methods.

As far as the base method and Lazy Snapping go, almost of subjects thought that 3D Lazy Snapping method was easier and preferred 3D Lazy Snapping to the base selecting method. Only one subject did not think so, and sait that both methods is not so different in ease of use. Those who preferred 3D Lazy Snapping felt less pressure and spending less time when selecting by the method than the lasso selection.

However, there are many subjects who are not satisfied with Lazy Snapping. The most major one of the reasons is because they could not foresee the resulting region by Lazy Snapping. Comparing the Lazy Snapping method with our proposed method, almost all of subjects selected the latter method. The major reason is because they could robustly track the boundary by heavy markers and obtain good selecting results by the Lazy Snapping algorithm.

**Evaluation of Time and Error**

We evaluate the time spent on selecting operations and error which are difference between users' selecting regions and correct regions.

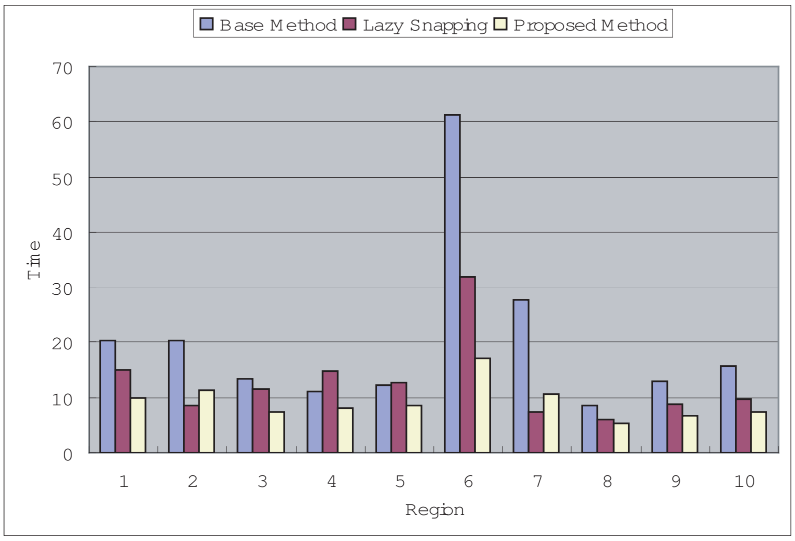
Firstly, see figure 6.4.2. The upper of the figure shows average operating time that each subjects spent on selecting one region by three selecting methods. This figure shows that the time processed by base method is longer than that by Lazy Snapping method in the case of almost regions. In particular, in the case of regions whose IDs are 2, 6, and 7, the time advantage is prominent. Those regions have peculiar boundaries or large areas, that is why the results of selection by Lazy Snapping are better than that of other cases. On the other hand, the cases of the regions whose IDs are 4 and 5 have little advantage or need more time. The reason is because these regions are very small and have vague boundaries. In those regions, the Lazy Snapping is not necessarily available. Comparing the time processed by our proposed method with others, we can figure that the average times are smaller than those of other methods. In particular, in the case of region 7 the time by our method is about half of that by Lazy Snapping. However, in the cases of region 2 and 7, the time by our method is over others. This result shows that selection by our proposed method can cut back on operating time on the regions that need longer time by Lazy Snapping, and it has disadvantage in the case of regions that need a little time by Lazy Snapping, but the increasing rate is not so large.

Nextly, see the lower one of the figure 6.4.2. This chart shows the average operating time spent in selecting one region by using three methods by each subjects. From this chart, we figure that Lazy Snapping is smaller than base method in the case of almost subjects except subject 4. The subject 4 is a novice computer user, and could not handle Lazy Snapping skillfully. Comparing the time spent by our proposed method with others, it further reduces than time by Lazy Snapping. In particular, in the case of subject 4 the time falls below the time by the base method. On the other hand, in the case of subject 6, it exceeds the time by

Lazy Snapping. The subject 6 is advanced-level computer user and the time by Lazy Snapping is short enough, so we think that the time difference between time by Lazy Snapping and that by our method in his case is within the margin of error.

Finally, see the figure 6.4.2. This scattering diagram shows the relationship between time and error of selecting operations by three selecting methods. The average time and error in the case of each region is 100. And the lower value both parameters are, the better the evaluation results are. The error is evaluated by the count of mis-selected vertices. Observing the distributional condition, we figure that the most effective selecting method is our method. The distribution of samples of ours and Lazy Snapping is very analogical in the view of error, but our method has an advantage in the view of time. The base method has a disadvantage in both time and error.

From users' interview and these experimental results, we are sure that our proposed method is the most efficient selecting operation of the three methods.



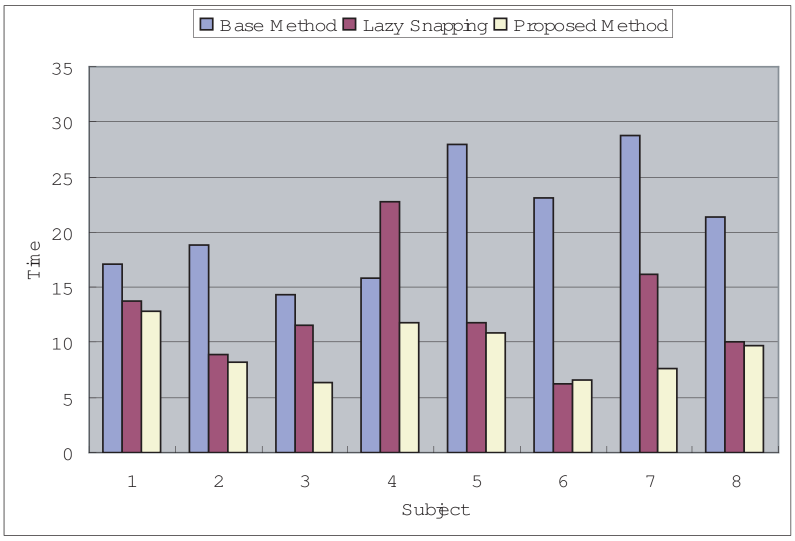


Figure 6.5: These charts illustrates the average time of selecting operations. The upper one shows the average spent time on selecting for each subject. And bottom one is the time for each region.

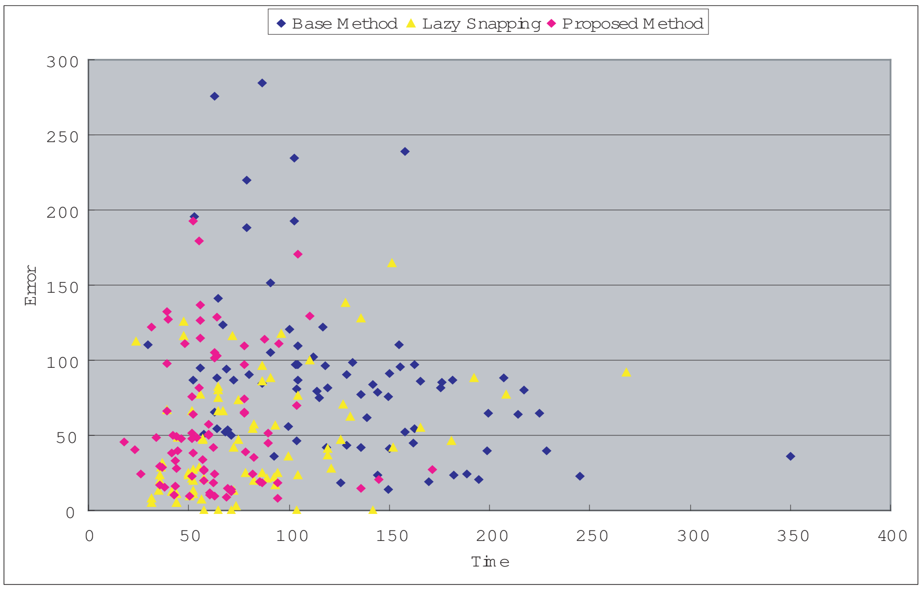


Figure 6.6: This chart shows the relation between operating time and error of resulting regions. The average time and error of each selecting operation equal 100. The lower each value is, the better the result is in the case of both time and error.

# Chapter 7

# Conclusion

In this chapter, we conclude this thesis, describe its contribution, and propose future work.

## 7.1 Summary

In this thesis, we propose a system that can render models obtained by scanning with very huge meshes, assign a wide variety of information to them, and present the information to users with target models.

In the rendering, we adopted a multi-resolution hierarchy based on small patches. The patch-based hierarchy minimizes the size of it and processing load during construction and rendering. For construction of the hierarchy, we use a graph-cut algorithm to split an input mesh into small patches based on geometric features.

Before assignment of information, users need to specify the region on surfaces to which they want to assign information and use as an index for retrieving. The selection operation is easy and interactive. By using the hierarchical structure for selection, we can determine the specified region on a huge model very rapidly. Additionally, we allow users to interactively select information by the 3D Lazy Snapping method and extended method.

Assignment information to specified regions is also done very easily by drag-and-drop actions from data tables or *InternetExplorer*© windows. For displaying the information, we use a browser that supports HTML format and allows the user to jump to other web pages as well as text files and images. Additionally, users can edit the assigned information, can rename labels, add text, and remove items.

From the evaluation results of our system, it can render very large mesh at enough high frame rates for users to interactively view and it achieves in almost cases at least over 10 frames per second independently of the size of the original mesh. Also, it can save memory space without significant decreasing of frame rates by releasing data of unused patches in LRU strategy. Additionally, we prove that our proposed selecting method that combine Lazy Snapping with a lasso tool is more efficient selecting tool than a lasso tool and Lazy Snapping selection.

## 7.2 Contribution of This Thesis

The major contributions of this thesis are:

- Efficient rendering system: our proposed system provides an efficient rendering environment on which users can view 3D models and change the viewpoint freely, interactively, and smoothly, even if the meshes have hundreds of millions of polygons.
- Interactive assignment of information: we enable users to assign information in a wide variety of format to 3D models. The operation of the assignment is easy and interactive for users in viewing the target 3D model. Users do not need precise operations but can select specified regions on 3D surfaces by interactive methods like Lazy Snapping. Also, the assigned data can be easily moved by drag-and-drop.
- Display and editing of assigned information: the assigned information can be presented to users on the 3D models, where they can access it by clicking assigned regions. The access of data is efficiently done by using functions of the database. Additionally, users can easily edit and remove the assigned information.

## 7.3 Future Works

Finally, we describe future work on our proposed system.

We propose to improve the current system so that users can interact with it even more easily and efficiently than they can do at present.

One future goal is to support access over the network to share data among many people. If it can be shared on the network, it could be copied to local machines. The data size of huge meshes is obviously very large, so to copy huge models need high costs in time and space. But with network accesss, many users could access the constructed models, view 3D models, and learn their history. Also, groups of users could share the work of assigning and editing information. To achieve this, we hope to improve the rendering method without need of over-frequent data communication.

Additionally, we want to improve the interaction of viewing 3D models. On our proposed system, users can move their viewpoints by mouse dragging. Users can change the movement patterns of translating, rotating, and zooming; by manipulating mouse buttons. However, this operation is not intuitive for some users, so they cannot move their viewpoint as they would like to. Therefore, we hope to find an easier interactive method and implement it.

Finally, we hope to arrive at a more sophisticated designe for assigning and displaying information.

# References

[1] http://developer.nvidia.com/object/nvtristrip_library.html.

[2] http://earth.google.com.

[3] http://worldwind.arc.nasa.gov/index.html.

[4] http://www.leica-geosystems.com/corporate/en/lgs_405.htm.

[5] Y. BOYKOV and V. KOLMOGOROV. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. In *Energy Minimization Methods in Computer Vision and Pattern Recognition*, 2001.

[6] M. Callieri, P. Cignoni, F. Ganovelli, G. Impoco, C. Montani, P. Pingi, F. Ponchio, and R. Scopigno. Visualization and 3d data processing in david's restoration. In *IEEE Computer Graphics and Applications*, volume 24(5), pages 16–21, 2004.

[7] Paolo Cignoni, Fabio Ganovelli, Enrico Gobetti, Fabio Marton, Federico Ponchio, and Roberto Scopigno. Adaptive tetrapuzzles: Efficient out-of-core construction and visualization of gigantic multiresolution polygonal models. In *ACM Trans. on Graphics (SIGGRAPH 2004)*, volume 23(3), pages 796–803, 2004.

[8] F.Evans, S.S.Skiena, and A.Varshney. Optimizing triangle strips for fast rendering. In *Visualization 96*, pages 319–326, 1996.

[9] Michael Garland and Paul S. Heckbert. Surface simplification using quadric error metrics. In *Proceedings of SIGGRAPH 97*, pages 209–216, 1997.

[10] S. Geman and Geman. D. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 721–741, 1984.

[11] H HOPPE. Progressive meshes. In *Proceedings of SIGGRAPH 96, Computer Graphics Proceedings*, pages 99–108, 1996.

[12] H HOPPE. View-dependent refinement of progressive meshes. In *Proceedings of SIGGRAPH 97, Computer Graphics Proceedings*, pages 189–198, 1997.

[13] Katsushi Ikeuchi. Modeling from reality. In *Proceedings of Third International Conference on 3D Digital Imaging and Modeling (3DIM)*, 2001.

[14] Katsushi Ikeuchi, Kazuhide Hasegawa, Atsushi Nakazawa, Jun Takamatsu, Takeshi Oishi, and Tomohito Masuda. Bayon digital archival project. In *Proceedings of Virtual Systems and Multimedia*, pages 334–343, 2004.

[15] George Karypis and Vipin Kumar. Multilevel k-way partitioning scheme for irregular graphs. *Journal of Parallel and Distributed Computing*, 1998.

[16] G. KLEIN, R. LIEBICH and W STRASSER. Mesh reduction with error control. In *IEEE Visualization '96*, pages 311–318, 1996.

[17] Ryo Kurazume, Ko Nishino, Z. Zhang, and Katsushi Ikeuchi. Simultaneous 2d images and 3d geometric model registration for texture mapping utilizing reflectance attribute. In *Proceedings of 5th Asian Conference on Computer Vision*, 2002.

[18] M. Levoy and T. Whitted. The use of points as a display primitive. Technical report, University of North Carolina at Chapel Hill Technical Report TR 85-022, 1985.

[19] Marc Levoy, Kari Pulli, Brian Curless, Szymon Rusinkiewicz, Dave Koller, Lucas Pereira, Matt Ginzton, Sean Anderson, James Davis, Jeremy Ginsberg, Jonathan Shade, and Duane Fulk. The digital michelangelo project: 3d scanning of large statues. In *Proceedings of SIGGRAPH 2000*, pages 131–144, 2000.

[20] Szymon Rusinkiewicz. Marc Levoy. Qsplat:a multiresolution point rendering system for large meshes. In *Proceedings of ACM SIGGRAPH 2000, Computer Graphics Proceedings*, pages 343–352, 2000.

[21] Yin Li, Jian Sun, Chi-Keung Tang, and Heung-Yeung Shum. Lazy snapping. In *ACM Trans. Graph*, volume 23(3), pages 303–308, 2004.

[22] D. Luebke, M. Reddy, J. Cohen, A. Varshney, B. Watson, and R. Huebner. *Level of Detail for 3D Graphics*. Morgan Kaufmann, 2002.

[23] M.Gopi and David Eppstein. Single-strip triangulation of manifolds with arbitrary topology. In *Proceedings of EUROGRAPHICS 2004*, volume 23(3), 2004.

[24] Takeshi Oishi, Ryusuke Sagawa, Atsushi Nakazawa, Ryo Kurazume, and Katsushi Ikeuchi. Parallel alignment of a large number of range images. In *Proceedings of the 4th International Conference on 3D Digital Imaging and Modelling*, 2003.

[25] Ryusuke Sagawa, Ko Nishino, M. D. Wheeler, and Katsushi Ikeuchi. Processing of range data merging. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2001.

[26] Carsten Dachsbacher. Christian Vogelgsang. Marc Stamminger. Sequential point trees. In *Proceedings of ACM SIGGRAPH 2003, Computer Graphics Proceedings*, pages 657–662, 2003.

[27] L. Vincent and P. Soille. Watersheds in digital spaces: An efficient algorithm based on immersion simulations. In *IEEE Transactions on Pattern Analysys and Machine Intelligence*, volume 13(6), pages 583–598, 1991.

[28] Sung-Eui Yoon, Brian Salomon, Russell Gayle, and Dinesh Manocha. Quick-vdr: Interactive view-dependent rendering of massive models. In *Proceeding of IEEE Visualization*, 2004.

[29] Xiaoru Yuan, Hui Xu, Minh X. Nguyen, Amit Shesh, and Baoquan Chen. Sketch-based segmentation of scanned outdoor environment models. In *Proceeding of the 2nd Eurographics Workshop on Sketch-Based Interfaces and Modeling*, pages 19–26, 2005.