

Interactive Information Sharing System using Large 3D Geometric Models

YASUhide OKAMOTO,^{†1} TAKESHI OISHI^{†1}
and KATSUSHI IKEUCHI^{†1}

Recent advances in digital archiving technologies allow us to obtain and store many kinds of precious data in large quantities. Additionally, sensing and modeling technologies for 3D digital archiving have also advanced dramatically, so we can obtain dense 3D geometric data of large-scale cultural heritage objects around the world. Thus, in this paper, we introduce a novel system in which ordinary users can access geometric data connected with cultural assets by using dense 3D models as visual interfaces.

Our system has three major functions: (1) to display information associated with the target 3D model in real time, (2) to associate information with a meaningful region of the 3D model, and (3) to provide this system on a network environment. First, we propose a polygon-based method that is more applicable to interactive rendering and information sharing system. Second, regarding our interactive selection tool, this enables users to select complex regions on huge 3D models using graph-cut algorithm. Finally, we propose a rendering method using a client-server model through the Internet to extend the proposed interactive tool to a network environment. Our proposed method is based on a hybrid method that we refer to as “Grid-Lumigraph,” which uses not only model-based data but also image-based data for efficient data transfer and image accuracy.

1. Introduction

In the fields of cultural asset research and protection, digital archiving is widely performed. Because digital technologies have made amazing strides in the past few decades, archaeologists and workers can obtain important information, which includes photographs, text documents, and drawn plans, more quickly and more accurately by using digital devices and software. As a result of digitization, many historical cultural assets have a large amount of important information

accumulated by many years of restoration activities and archaeological research. This digital data can be utilized in a wide range of fields not only for restoration activities, but also for tourism and education by exposure to the public¹⁵⁾.

However, it is difficult to convert the archived data to accessible and sharable form because the materials obtained by many experts specializing in different fields need a high degree of expertise for other people to understand and utilize the data. Additionally, the general database systems managing such data often present the data as text that is incomprehensible to many people, so there is a possibility that important data is lost because it is not understood or evaluated.

On the other hand, 3D sensing and modeling technologies have advanced dramatically in the past few years, so we can obtain more dense 3D geometric data of larger scaled objects than was previously possible. Some of the representative projects for large cultural assets include the following: the Digital Michelangelo Project¹³⁾ and some other projects¹⁾, the Great Buddha Project¹⁰⁾, and the Bayon Digital Archival Project⁹⁾. This 3D archiving is carried out to preserve the accurate shapes of whole or part of the target properties, and the resulting 3D models can be important materials to repair and rebuild them if the target objects face deterioration or destruction.

The obtained highly dense 3D models are also useful for further analysis, as they include all geometric features from the entire target object to the detail as shown in Figure 1. For example, in the case of the Bayon Temple, the scanned 3D model has many kinds of interior object at multiple levels of resolution, which include buildings, towers, and gardens as larger objects, and also statues, pillars, and curved reliefs as smaller objects. Most of the inner objects have important meanings that characterize the entire object, so the model can be used not only for nearly perfect preservation and restoration but also for highly precise archaeological analysis.

Therefore, in this paper, we aim to develop a novel system in which a person without expertise can easily and effectively utilize the data connected with cultural assets by using 3D models as visual interfaces. Our proposed system allows users to use the characteristic parts on the huge 3D models as interfaces with other information, and to freely assign many kinds of information to such 3D parts on dense 3D models, and to view such information and 3D models inter-

^{†1} University of Tokyo

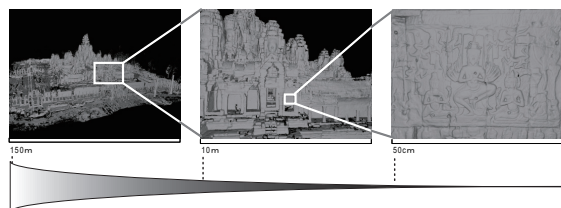


Fig. 1 Interactive viewing of huge 3D models from any viewpoint and with any resolution

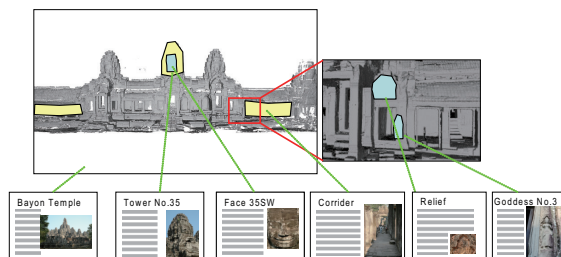


Fig. 2 Usability by relating characteristic parts of 3D models and other information. The middle image: images of 3D models assigned with other information. The upper and bottom rows: attached information

actively as shown in Figure 2. The visualized 3D models, as compared with a text-based interface, enable the general public to understand the significance of historically precious objects intuitively. Moreover, dense 3D models have many meaningful parts in themselves, and those parts can help us understand other kinds of information assigned to them. These meaningful objects can serve as visually intuitive interfaces that provide access to other information for general users. Previous systems have not proposed the use of meaningful surfaces as indices to information or as selection and assignment tools.

This system provides a more direct and intuitive interface than current systems provide. Moreover, to realize the system, we propose three main works as key components as follows:

- Novel real-time rendering methods for interactive display of huge 3D models
- Novel interactive tools for utilizing huge 3D models and other information
- Extension of the rendering system to the network environment for free access

from all over the world

For the realization of such system, first, we need to develop an efficient rendering system to visualize huge 3D models in real time. The obtained model may have millions to billions of polygons whose size is very large and can exceed the amount of memory installed in a computer. To solve it, we propose a polygon-based rendering that uses a multiresolution scheme based on small mesh patches without unconformity. The data structure of the previous patch-based methods, as proposed in Adaptive TetraPuzzles⁴⁾ is not suitable for our region-selecting operations. Therefore, we construct split patches depending on the geometric features to optimize the region-selecting operations on our proposed 3D database system.

Second, we need to develop intuitive handling tools that allow users to easily select specified regions and to assign information to them without too precise operations. The obtained 3D models include very fine shapes that accurately reflect the real target objects, and these can become too complex to be handled easily. The previous graph-cut selecting methods as proposed in^{14), 16)} are only for 2D images and simple 3D point clouds. Therefore, we propose a graph-cut selection method optimized for an LOD based huge 3D data structure. We allow users to select 3D regions accurately by using a patch-based data structure split depending on the geometric features, and we propose a more effective user interface to select regions more quickly and accurately than previous methods allowed.

Finally, we need to propose the extension of the system to a networked environment to share and utilize archived data among many users everywhere. However, the problem of the size of the data in a network environment affects rendering performance more seriously than in a local environment because of the cost of data transfer. To enable low-end clients to access a huge dataset, we propose a hybrid method using both image- and model-based data to render 3D images with low network traffic and high security level for 3D data. The Lumigraph⁷⁾ also proposed a hybrid method, but that method is specialized for a local environment and cannot represent huge, complex objects. Our proposed method, referred to as “Grid Lumigraph”, minimizes the amount of data transfer and to render arbitrary views efficiently by using pre-rendered images sampled from grid

points in the 3D space and polygon-based multiresolution 3D data.

The remainder of this paper is organized as follows. In Section 2, we describe our proposed system, the 3D database system. In Section 3, the real-time rendering methods for huge 3D models in an offline environment are described. In Section 4, we describe the selecting tool that allows users to select complex regions on 3D models by simple operations. In Section 5, the rendering method in the online environment is described. In Section 6, we conclude our research.

2. Overview of Information Sharing System on 3D Models

In this section, we propose a system that enables general users to store, manipulate, and retrieve many kinds of information through very huge 3D models.

We can classify the functions of our system into two parts, which are:

- (1) To assign many kinds of information to specified areas on the 3D model in a flexible way
- (2) To display the 3D model and assigned information in real time

Our system allows users to manipulate the 3D models and related information by simple actions, such as the selection tool described in Chapter 4. They can also assign related information to 3D selected regions as shown in Figure 2.

Moreover, our system helps them browse the 3D models and assigned information by visualizing the assigned regions and information on the 3D model. Our target 3D models are very large and have various features at multiple levels of scaling. Our system can visualize these models, and can seamlessly change the viewing position and direction in real time, so users can access the desired information interactively and quickly.

In this section, we describe these two main functions in detail.

2.1 Assignment System

First, we propose an assignment system that allows users to assign information. 3D models scanned with high density have many parts that have geometric and photometric features as shown in 2. Especially if the target objects are cultural assets, they have unique meanings on each characteristic part at every level of scaling. If there is a statue, there should be information related to it such as the sculptor, the materials, and historical background. By using this function, users select featured regions on the 3D model, and assign such information to them.

For assignment of information to 3D models on our system, users first select the region to be associated with information, and next associate the data file storing the information with the selected region. Here, we describe the operating procedures and interface for assignment of information.

2.1.1 Region Selecting Operation

Users can start region selecting mode whenever they are viewing the 3D model. If a user find a characteristic part to be associated with information when viewing the 3D model, the user can enter the selecting mode by pressing the selection tools button. For selecting operations, our system proposes drawing line tools. However, it is difficult to rapidly and correctly trace the boundaries of all kinds of regions by a simple drawing line tool like a lasso tool, for example, in the case that the region has a very complex boundary.

To enable users to select complex regions, our system proposes three kinds of selection tools: a traditional lasso tool, a mark-up tool like Lazy Snapping, and our proposed graph-cut lasso tool. The mark-up tool and graph-cut lasso tool are based on the graph-cut algorithm, and enable the user to select complex regions with geometric and photometric features by simple operations. These details are described in Section 4.

Additionally, in this mode users can edit the resulting selected regions. If users cannot select a part of a region because it is invisible from their viewpoint, they can change the viewpoint, make a selection, and add it to the region already selected. If they select a region that contains unwanted parts, they can also remove those parts from the selected region. To edit selected regions, our system allows users to use a traditional lasso tool.

2.1.2 Association 3D Regions with Information

After selecting regions, users associate information, such as text documents, image files, and links to web pages, with the 3D model. To assign the data to the 3D model, our system allows users to pick the icon of the file from a *Microsoft Explorer*® window, hereafter referred to as “Explorer”, to the region displayed on the renderer by a drag-and-drop mouse action. In this assigning mode, assigned data is copied to the predefined directory, and registered in the database system as shown in Figure 3. Additionally, users can also describe and add the meta data associated with the selected regions, such as titles, captions,

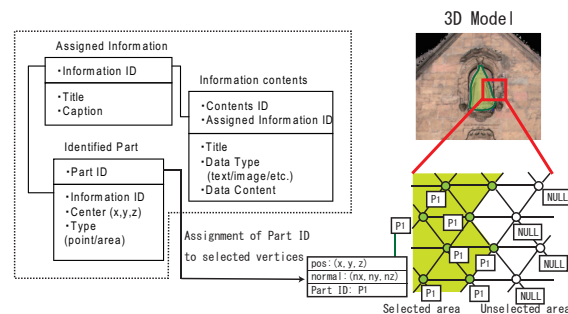


Fig. 3 Database tables for associating information with 3D models and the details of the associating procedure

and thumbnail images.

Next, we explain the detailed scheme of our database system. We use a conventional relational database system for data managing. The skeleton form of our database and relationships between data tables are as shown in Figure 3. The our database is mainly composed of three tables:

- (1) Assigned Information Table, which manages information assigned to each specified 3D region.
- (2) Information Contents Table, which stores the data of information and its meta data.
- (3) Identified Part Table, which records the parameters of the 3D regions specified by the user. The records in this table include Part ID, Information ID, center, and region type.

The Part ID is also recorded in the geometric data as shown in Figure 3; a unique Part ID is assigned to the vertices on each specified region and also recorded in this table when the region is selected. The center is the central point of the region, and the region type represents whether the selected part has only one point or region.

2.2 Display System

Our system enables users to interactively browse the target 3D model and associated information. When users are viewing the model, callout containing the title and information are displayed on the attached regions. Additionally they can browse details of associated information by clicking specified regions.

To quickly render huge scanned 3D models, we use the efficient rendering system proposed in Section 3. By using the rendering method our system can render any 3D models in real time and without memory shortage even if they are so large to exceed the memory size.

In the display system, they can easily view the associated information and associated regions. The callouts with the title label of regions are displayed around assigned regions, and the lines connecting them are drawn as shown in Figure 4. If users place a mouse cursor near the specified regions, the boundaries of those regions are highlighted with bold lines. In addition, the preview window of assigned information is displayed, and its meta data such as title and thumbnail are described. If they find the region to be accessed when viewing the target 3D model, users can open the information browser on this system and display the associated information on it by double-clicking on the region as shown in Figure 4. When regions are displayed and users access to them, the database system can easily retrieve information assigned to the 3D regions from the Information Contents Table shown in Figure 3 by using Part ID as a key variable.

The information browser supports HTML format. The associated information is also formatted in HTML and displayed on it. Additionally, users can also easily edit and remove the meta data included in each 3D region and information, such as a title, captions, and thumbnail images, on the browser, by a “wiki” style editor.

3. Polygon-based LOD Rendering using Graph-Partitioning

In this section, we propose a novel rendering method suitable for huge sized 3D models and our 3D information sharing system based on region selection.

The polygon-based rendering techniques have been used for a long time, and they are supported on recent graphics hardware. However, the rendering performance for huge 3D models can prominently decrease because of the load of the rasterization processing. In addition, it is also difficult to convert the polygon model into an LOD-based hierarchical structure while keeping the accurate connectivity. Many polygon-based LOD techniques have been proposed that preserve the consistency of polygon meshes, such as Progressive Meshes⁸⁾, but such methods cannot work efficiently for huge models.

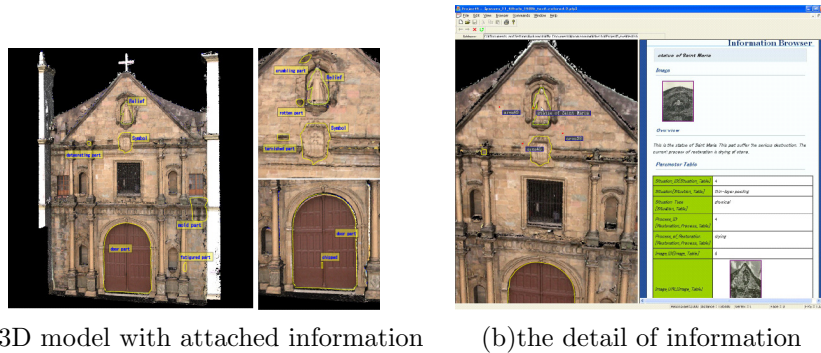


Fig. 4 Display of related information and access to it from 3D models. (a) Interactive display of the 3D model with associated information in real time depending on the viewing scale. (b) Browsing window of the details of associated information. The left window displays a 3D model, and the right window displays the information assigned to the selected region.

So we propose a patch-based LOD method as proposed in Adaptive Tetrapuzzles⁴⁾, whose data structure is composed of small patches and optimized for huge mesh processing. However, Adaptive Tetrapuzzles splits and generates small meshes with a very simple rule, and these simple patches are not suitable for accurate region selection. In our method, by constructing small patches depending on the geometry of the input model, we can enable users to accurately select specific regions on the 3D model at the same time they can view huge 3D models in real time.

3.1 Construction of Hierarchical Structure

The patch-based LOD hierarchy consists of small mesh patches. This approach enables us to reduce the processing load for constructing and traversing the multiresolution hierarchy more efficiently than the approach using one point or one triangle as primitive.

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 generate the simplified meshes assigned to each node, and this process is done by performing a bottom-up traversal of the hierarchy constructed at the first step.

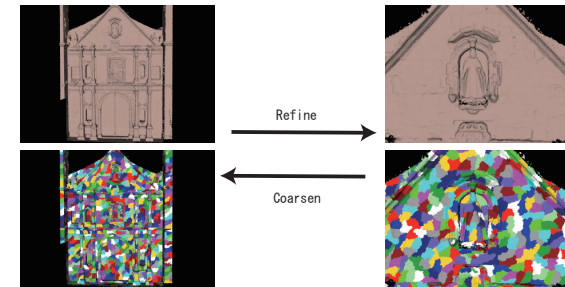


Fig. 5 This figure shows view-dependent patch-based rendering. The right images represent the rendered patches in the left scenes. The patches of the bottom scene are smaller and more refined than those of the upper scene. By controlling the density of patches depending on viewpoint, we can efficiently render any model even if it has millions of triangles.

3.1.1 Mesh Partitioning

In the first step of the constructing process, we decompose the huge input mesh into small patches.

First, we perform over-partitioning of the mesh into small meshes to save processing cost of splitting huge sized meshes. Then, we decompose an input mesh model into smaller meshes using the voxel space, and sort each triangle to a single voxel which contains at least one of its vertices. If the vertices of a triangle belong to multiple voxels, it is sorted to the voxel that contains the most vertices of the triangle. Finally, we assign a group of meshes to the corresponding voxel. We control granularity of the voxel space so that each voxel contains fewer triangles than the predefined value N_v .

After over-partitioning, we perform recursive partitioning of the set of meshes obtained by over-partitioning, and construct a hierarchical structure composed of small patch meshes as shown in Figure 6. We recursively split a set of voxels into two small sets. If the size of the partitioned set (the size is defined as the number of triangles included in the set) exceeds a predefined number N_n , we recursively partition the set again. Otherwise, we release all polygons from all voxels, and continue splitting the mesh into two meshes by polygons. If the number of split mesh is under a predefined number N_l , we simply stop further partitioning of the set of voxels, backtrack, and continue with processing for the

rest of unpartitioned meshes. This recursive partitioning continues until the sizes of all leaves of the hierarchy are under N_l .

To partition the set of meshes depending on the geometry and construct a balanced hierarchy, we use a graph-partitioning algorithm proposed in¹¹⁾. We convert a group of mesh split by over-partitioning into a graph representation $G(V, E)$ whose V correspond to over-partitioned voxels, and E correspond to edges between voxels. Each vertex ($v \in V$) and each edge ($e \in E$) of the graph has a weight. Here, a vertex weight is defined as the number of triangles belonging to it ($W_{vi} = v_i \setminus$). To define edge weight, first, we calculate the surface normal vector at each vertex. We use a mean vector of polygons that each voxel contains, and represent it as \vec{n}_i for vertex i . Then, an edge weight is defined as the inner product of two averaged vectors of the two vertices on the edge ($W_{Eij} = \vec{n}_i \cdot \vec{n}_j$). By this assignment of weight, we can distribute triangles with similar geometric and photometric features into the same part. For implementation, we use the METIS library using the graph partitioning algorithm proposed in¹¹⁾. The geometric- and photometric-based partitioning can be the result of 3D over-segmentation described in Section 4.

In our system, we set 4,000 as the maximum number N_l for leaf nodes. As a result of partitioning, we can obtain a binary tree of small patches and a set of triangles associated with leaf nodes that cover the entire original mesh. So we can construct and process the hierarchy structure more efficiently.

3.2 Construction of Multiresolution Hierarchy

The next step is the simplification of partitioned patches. In this step, we assign all nodes in the hierarchy to a patch whose triangle's count is under the predefined value. This procedure is efficiently done in bottom-up fashion.

To each leaf node, we directly assign the current patch partitioned in the first step because the number of triangles in each leaf node have already been less than N . For non-leaf nodes, we merge two patches associated with the children into a single mesh and simplify the merged mesh so that each node contains less than a predefined number of triangles.

After merging, we simplify the patch to reduce the number of contained triangles to fewer than the predefined number. To simplify meshes, we use Garland's simplifying method, described in⁵⁾, that uses a quadric error metric, which is

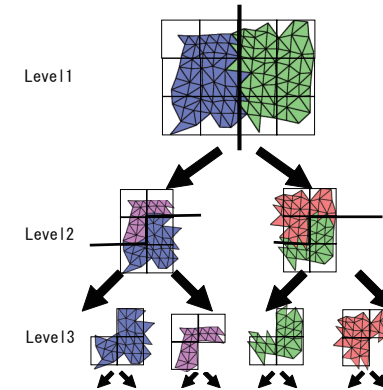


Fig. 6 The patch-based hierarchy is constructed by recursive partitioning of the mesh. This hierarchy needs less space and load than a triangle- or vertex-based hierarchy.

the sum of the distances between the vertex v and adjacent faces f , to inhibit changes of geometrical aspects caused by simplification. By keeping a queue of edges to be collapsed and quadric error values in the form of a heap, and sequentially collapsing the edge from the top of the heap until the number of triangles in the mesh is a predefined number, we can obtain a simplified mesh. After simplification, we attach the maximum quadric error value, which is the value of the last collapsed edge, to the simplified mesh as the error value that represents the resolution of the mesh.

In the above simplification process, we must maintain consistency between different meshes. Otherwise, when switching rendering patches from patches at a different resolution, we cannot keep the correct connectivity along borders of patches and can generate holes or unnatural connections there. To avoid this inconsistency between patches at different resolutions, we must give a constraint to the simplification process, that edges adjacent to other patches are never collapsed.

Finally, we convert the hierarchy structure and simplified patches into a final format that is appropriate to be rapidly extracted and rendered. We rearrange vertices and triangles of each mesh into cache coherent form, compress data, and record a file formatted as Figure 7. In this format, we record a hierarchical

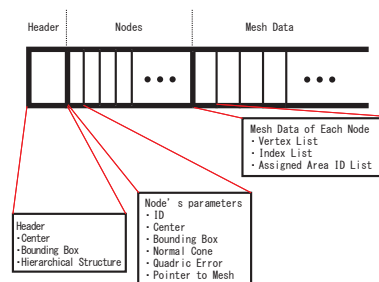


Fig. 7 The data structure of the final format is recorded as above. First, the hierarchy structure is recorded. Second, the parameters of each node are recorded. Finally, the geometric data of vertices and triangles of each patch are recorded.

structure and mesh data in depth-first order to optimize the accessing efficiency when rendering.

3.3 Multiresolution Rendering

Our developed rendering system is based on a top-down traverse and refinement. First we traverse the hierarchy structure constructed in preprocessing, and determine the set of minimum required patches whose simplification errors projected on the screen do not affect rendering quality.

First, in the rendering process, we load data of the multiresolution hierarchy and properties of all nodes as the initialization process. When we perform recursive traversals of the hierarchy, we roughly test whether the current node is visible or not by checking the location of the bounding sphere of the current node on a camera coordinate, and the cone of normals of the patch against the current view direction. If the current node is judged as invisible by the above simple test, we can skip its processing and that of the entire subtree, backtrack, and continue traversing the remaining nodes. If the node is judged as potentially visible, we test whether the resolution of the patch assigned to the node is accurate enough by measuring its saturated screen space error. If so, we can render the patch; otherwise we recursively traverse the node's children. If we find that the patches to be rendered and the geometric data are already loaded, we insert their ID numbers into the rendering list. If the patch and data are not loaded, we insert IDs into the loading list and insert the ancestor node of the patch into the rendering list. This is a temporary action to avoid generating

holes. The upper bound of the screen space error of rendered nodes is usually set to 1.0. However, in computers with low graphics capability, we can control the rendering performance by setting the upper bound to a higher value.

3.4 Evaluation

We evaluated the performance of preprocessing and rendering for huge models by using our proposed polygon-based method. We used three huge models, that could not be interactively rendered by traditional polygon rendering, and found that we could easily render them regardless of the size in our rendering system on commodity PCs.

We implemented and evaluated our system on a commodity PC, for rendering on an ATI Radeon 9800 XT, and 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.

Here, we describe how we evaluated the performance. We measured the processing speed and memory efficiency in preprocessing and rendering.

We show the results of preprocessing and rendering performance for some large models in Table 1.

The construction of a polygon based LOD data structure that has about one million triangles takes less than ten minutes, while the construction of a mesh over ten millions triangles needs more one hour. We can say that those results are within the allowable range in terms of practicality. Additionally, we have room to improve the efficiency of preprocessing by using parallel computing.

From these results, while the rendering frame rate decreases as the size of an input mesh increases, the average frame rate is maintained over 10 fps for larger meshes whose original model has about ten millions polygons. And the frame rate is also maintained around 10 fps when the number of polygons is over ten millions.

The percentages of loaded data size in the table mean the rates relative to the entire data size. In the case of the smaller meshes, the percentages are high since the hierarchy structure is smaller and the data access can easily reach all patches. In the case of larger ones, the percentages are moving down since all data need not be loaded and accessed to render each view. To constrict it, we use the memory management function that replaces data on the memory adaptively


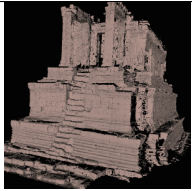
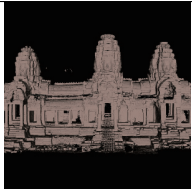
			
model	Bayon Face	South Library	Bayon 3 Towers
Input polygons	5,922,790	9,162,909	18,132,893
Preprocessing time	1592 s	3212 s	6853 s
Average frame rates	13.8 fps	13.0 fps	9.9 fps
Average loaded size	123MB(49.6%)	199MB(47.1%)	379MB(45.4%)

Table 1 The preprocessing and rendering performance for our proposed polygon-based LOD method. This table shows the preprocessing time, rendering frame rate, and loaded data size.

for the access frequency.

4. Region Selection Tools on 3D Database System

In this section, we propose a novel interactive tool to select complex regions on huge 3D models using very simple operations and graph-cut algorithm. The selection using graph-cut has been proposed in Lazy Snapping¹⁴⁾ and¹⁶⁾, but those methods are applicable only for 2D images or for simple point clouds. Our proposed method can be applied to huge 3D model represented by a hierarchical data structure and to enable users to select specific 3D regions very accurately based on the geometric features using split small regions extracted the hierarchical data structure proposed in Section 3. Additionally, we design a sophisticated user interface to select each 3D region very intuitively and speedy.

First, we describe the graph-cut algorithm that splits a 2D image into two regions, propose an extension for huge 3D models, and describe the GUI design. Next, we evaluate the efficiency of our proposed selection tool.

4.1 Interactive Selection Tool

In our system, we propose an interactive selection tool that enables users to easily select complex 3D parts. If you use a conventional tool, such as a lasso tool, to select a specific region, you must accurately trace the boundary line by moving a mouse. However, it is very difficult for general users to select regions

quickly, especially if the regions have very complex boundaries in the 3D models. To select complex 3D regions efficiently, we propose several selection methods that are efficient and interactive for general users by using graph-cut algorithm.

4.1.1 Graph-cut Algorithm for Selection

Various interactive selection tools using graph-cut have been proposed previously. In²⁾, Boykov et al. propose an interactive selecting method on 2D images. The method computes the appropriate labeling for the image from the information of foreground and background defined by users. Lazy Snapping¹⁴⁾ proposed by Li et al. is also a system of 2D image segmentation; they propose some interactive interfaces in addition to Boykov's method. But these methods can be improved to optimize for selecting complex 3D regions. In our system, to accurately and quickly select the specified region from 3D models, we use a novel interactive graph-cut algorithm.

To select 3D regions, our system determines selected regions on the displayed images by computing a binary labeling problem. In the labeling problem, the image is represented by a graph $G = (V, E)$ where V corresponds to pixels of an input image, and E corresponds to connections between eight neighboring pixels. 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.

The labeling problem is to assign a unique label x_i for each node $i \in V$, i.e., $x_i \in \{1, \text{meaning foreground}, 0, \text{meaning background}\}$. The solution $X = x_i$ can be obtained by minimizing a Gibbs energy $E(X)$ ⁶⁾:

$$E(X) = \sum_{i \in V} E_1(x_i) + \lambda \sum_{(i,j) \in E} E_2(x_i, x_j) \quad (1)$$

where $E_1(x_i)$ means energy representing the “unsimilarity” of the labeling of each node, encoding the cost when the label of node i is x_i . The value falls down if the variance between each node and the parent group to which it belongs is small, and it increases otherwise. $E_2(x_i, x_j)$ is the cost of the boundary between F and B when the labels of adjacent nodes i and j are x_i and x_j respectively. We can say that the labeling minimizing this function $E(X)$ is the most appropriate to split the image into two regions.

In the graph-cut method for 3D images, we use surface normals and texture

colors for the energy function, and $E_1(x_i)$ is represented as follows:

$$E_1(x_i) = w_c * E'_1{}^c(x_i) + w_n * E'_1{}^n(x_i) \quad (2)$$

where w_c and w_n are weight functions for texture color and normal terms. The definition of E'_1 is as follows:

$$\begin{cases} E'_1(x_i = 1) = 0 & E'_1(x_i = 0) = \infty & \forall i \in F_{seed} \\ E'_1(x_i = 1) = \infty & E'_1(x_i = 0) = 0 & \forall i \in B_{seed} \\ E'_1(x_i = 1) = \frac{d_i^{F_{seed}}}{d_i^{F_{seed}} + d_i^{B_{seed}}} & E'_1(x_i = 0) = \frac{d_i^{B_{seed}}}{d_i^{F_{seed}} + d_i^{B_{seed}}} & \forall i \in U \end{cases} \quad (3)$$

where U means nodes that are assigned to neither F_{seed} and B_{seed} . The first and second equations guarantee that all nodes on F_{seed} are always in the foreground, and all nodes on B_{seed} are in the background. The third equations describe the distance to F_{seed} or B_{seed} in the case of the node in U which is assigned to the foreground or background. $d_i^{F_{seed}}$ and $d_i^{B_{seed}}$ are differences of colors or normals from those of the initial foreground set or background set. $E'_1{}^c(x_i)$ is the energy function for texture colors, then d_i^F and d_i^B are obtained by comparing the texture color $C(i)$ of the node i with the average color of the initial foreground or background. $E'_1{}^n(x_i)$ is the energy function for surface normals in which d_i^F and d_i^B is represented by the following equations: $d_i^F = \min_n |1 - (\mathbf{N}(i) \cdot \mathbf{N}_n^{F_{seed}})|$, $d_i^B = \min_n |1 - (\mathbf{N}(i) \cdot \mathbf{N}_n^{B_{seed}})|$ where $\mathbf{N}(i)$ means the average normal of segment i . $\mathbf{N}_n^{F_{seed}}$ and $\mathbf{N}_n^{B_{seed}}$ are also the average normals of the initial foreground set and background set.

Additionally, $E_2(x_i, x_j)$ is denoted as follows:

$$E_2(x_i, x_j) = |x_i - x_j| \cdot \{w_c * g(C_{ij}) + w_n * g(N_{ij}) + w_d * g(D_{ij})\} \quad (4)$$

In this equation, w_c , w_n , w_d are weighting factor defined empirically. C_{ij} , N_{ij} , and D_{ij} mean the differences of texture color, normal, and depth between segments i and j .

To obtain the surface normals on the rendered image, we render another image that has the value of the surface normal instead of the pixel color on the background buffer. The difference D_{ij} between clusters i and j is calculated in the same way as normal, by using the depth image rendered on the back buffer. The depth value of each pixel is also extracted from the depth image, and we calculate the mean depth D_i and difference D_{ij} by using extracted values. The normal image and depth image are rendered at the same time the 2D segmentation is

done, when users start selecting operations.

To solve the problem of minimizing $E(X)$, the min-cut/max-flow algorithm described in³⁾ is used. When $E(X)$ is minimized, the set of nodes labeled as $x_i = 1$, which means foreground, provides the most likely region to be selected for users. If users are not satisfied with the result, they can add and remove markers and update the result of the segmentation by repeatedly calculating the above graph-cut problem.

4.1.2 Extension for 3D Segmentation

Next, we describe the extension of selection method using the graph-cut algorithm to the version for 3D images. We extend the 2D selection method to apply to the huge 3D models represented by hierarchical data structure, and enable accurate selections for 3D regions by using small 3D patches composing the LOD hierarchy.

To reflect the selected region to the original 3D model, we project it on the surfaces from the current viewing point and add the information about the region to the vertices located in the projected range. However it is very time-consuming to judge all vertices of the huge 3D model. We need to narrow down the number of vertices to be tested. Moreover, we need to judge whether each vertex is in the region intended by the user, such as occluded surfaces which are not rendered on the screen, because it is obvious that it is not included in the regions likely to be selected by users. So we must not select the vertices on such parts.

To solve these problems, we segment the 3D model into small regions before the selection procedures. The mesh partition is done in the construction of multi-resolution based on geometric and photometric features of surfaces described in 3.1.1, and the rendered image of the model is composed of the set of small surface regions. Therefore, we also narrow down the vertices to be selected based on the features of the target model by using that split model. Not only that, we can accurately select each 3D region using split small meshes because those are split depending on the geometric features in 3.1.1. By using them we can accurately eliminate non-rendered parts not to be selected based on the shape, and can propose resulting regions with high quality to users.

After the segmentation process using the graph-cut algorithm on the screen, our system reflects the result into the original 3D model. We show the process

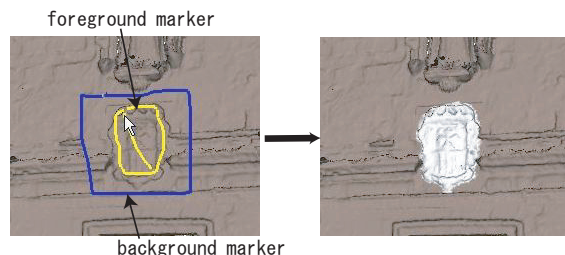


Fig. 8 This figure shows the result of selection by 3D Lazy Snapping. Yellow markers represent the foreground region, and blue markers represent the background region. In the image that results, the relief on the facade is selected.

of drawing markers and segmentation results in Figure 8.

After selection of a region by graph-cut algorithm on the screen, our system assign a unique ID to the selected region, and reflects the result onto a 3D model as shown in Figure 9. Then, we back-project the selected 2D region onto the index image in which each pixel represents the ID of rendered small regions, and specify the IDs of the selected 3D regions from the index image. Next, we back-project the 2D region onto the meshes that belong to the regions selected on the index image. We test whether each vertex in the 3D regions is included in the selected 2D regions on the screen. If the vertex is included, we add the region ID to the vertex as a property of it, and register the relationship to the database system.

The resolution rendered on the index is dependent on the distance between the viewpoint and model. So we can control the granularity of selected regions by the viewpoint. We can select the region in detail if the viewpoint is close to the surface.

We also select vertices of the ancestors' and children's patches to support the case that the rendered resolution level change is dependent on viewpoints. Since we store information about selecting regions in the database, and we store information about each selected vertex in the hierarchy structure, it can be easily extracted by accessing the database and by traversing and rendering.

4.2 Design for Selecting the GUI

A graphical interface like Lazy Snapping allows users to select specified regions

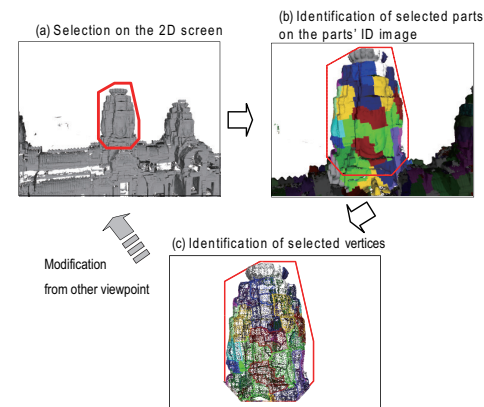


Fig. 9 Selecting operation for 3D models (a) selection an area on the screen, (b) selecting partial patches in the selected area, (c) selecting vertices from those of selected patches.

by only roughly marking foreground and background parts. However, users may not be able to easily estimate the resulting region calculated by graph-cut from the marks they have drawn. If the desired result cannot be obtained, users must draw more marks in order to specify the correct region and need more time.

Therefore, to solve the problem, we propose a more efficient GUI that allows users to select regions intuitively by simple strokes like a lasso tool with the accuracy of the graph-cut method. Our proposed selecting tool combines a traditional lasso tool with graph-cut algorithm, allowing users to specify the desired regions by using this tool only once. This selecting method involves the following procedures. First, users draw a heavy marker along the boundary of the region they want to select, as shown in Figure 10. This operation is intuitive and easy like a lasso tool, but a heavy marker allows users to more easily and speedily trace the boundary of a desired region than they could if they were using a traditional lasso tool.

After users finish tracing the boundary, our system automatically sets a foreground seed and a background seed to the offline buffer of the rendered image as shown in the left of Figure 11. The foreground marker is drawn along the inside of the heavy marker, and the background marker is drawn along the outside. By this automatic drawing and applying the graph-cut algorithm, users can obtain

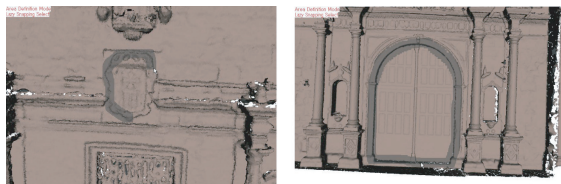


Fig. 10 This figure shows the selection operation by a heavy lasso. Users track the boundary of the desired region by drawing a grey heavy marker. Users can do this easily and quickly because of the robustness of the heavy marker.

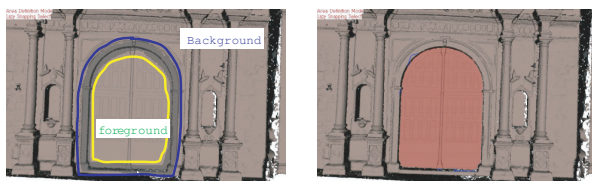


Fig. 11 The left figure shows the automatic drawing markers representing foreground and background. Our system draws a yellow marker inside a heavy line, and a blue marker outside a heavy line. The right figure shows the resulting region selected by our proposed method.

a desired region whose shape is close to that of the marker they draw. Finally, our system displays the resulting region selected by the graph-cut algorithm, as shown in the right of Figure 11.

By using this method, users can obtain desired regions quickly, accurately, and intuitively.

4.3 Usability Study

In this section, we evaluate our proposed selection tool on a 3D database system. To evaluate selecting methods, we compare the processing time and accuracy between the conventional lasso tool, the markup tool indicating foreground and background, and our proposed selection tool.

In our usability study, we select eight subjects. Two subjects were novices that had little experience with applications dealing with 2D and 3D images. Two were experts who were good at operating computers and usually used 3D applications. The other four people were intermediate users who had experience using 3D tools

but were not experts. Before the test, we gave all subjects an instruction session in using the selection tools in our system.

We gave subjects tasks to select ten regions. Subjects did the assignment by three selecting methods: a lasso tool, a markup tool, and our proposed method. To avoid having the results affected by the order, the order of the selection tools used in the test was varied among subjects: four of the subjects first used the lasso tool, two subjects first used the markup tool, and the remaining two subjects first used our proposed method.

We evaluate the time spent in selecting each region, and we evaluated accuracy by measuring the difference between users' selection regions and the correct regions (see Figure 4.3). The left figure shows the average time that all subjects spent in selecting a region by each selection tool. This chart shows that the times in the case of the lasso tool were longer than the case of the markup tool for almost all regions. In particular, for the regions whose IDs are 2, 6, and 7, the time advantages were higher than others. Next, the right chart of the Figure 4.3 shows the average operating time spent by each subject in selecting each region by using the three methods. From this chart, we see that the selecting time by markup tool was smaller than that by the simple lasso tool in the case of almost all of subjects except subject 4, a novice computer user who could not handle the markup tool skillfully. Comparing the time spent by our proposed method with others, our method further reduce the time for almost all subjects, and particularly for subject 4.

Finally, in Figure 4.3, we show in this scattering diagram the relationship between time and error of selection operations by three selecting methods. We define the average time and error in the case of each region as 100 to avoid the difference of difficulty between regions. The lower the value of both parameters, the better the evaluation results are. The error is defined as the ratio of the number of mis-selected vertices to that of all vertices in each region. Observing the distributional condition, we can see that the most effective selection method is our method. Our method is analogous to the markup tool from the standpoint of error, but our method has an advantage in the processing time. The lasso tool has a disadvantage in both time and error.

From these experimental results, we are sure that our proposed method is the

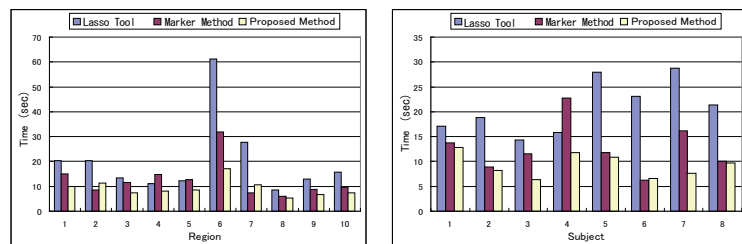


Fig. 12 These charts illustrates the average time of selection operations. The left chart shows the average time spent by all subjects on selection for each region, and the right chart shows the average time spent by each subject per region.

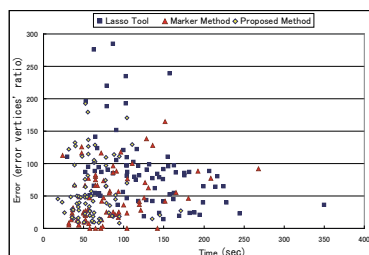


Fig. 13 This chart shows the relation between operating time and error of resulting regions. The average time and error of each selecting operation equals 100. The lower each value is, the better the result is in the case of both time and error.

most efficient selecting tool of the three methods.

5. Hybrid Model/Image Rendering Method in Network Environments

In this section, we propose an efficient rendering system that allows users to view huge 3D models through the network. To render huge 3D models with high quality and low network cost, we propose a hybrid rendering method using model- and image-based data. In The Lumigraph⁷⁾ also adopted a hybrid method, but it only works in a local environment, and it cannot represent complex objects. Our proposed method “Grid-Lumigraph” method samples images more densely than Lumigraph, so it reconstructs arbitrary view images for very huge objects with minimum cost and strong security.

In this section, we propose our rendering system that uses both model- and image-based data, and we describe the details.

5.1 Grid-Lumigraph System

Here, we present the framework of our proposed system for rendering through the network.

To render huge 3D models with appropriate network cost and quality, we propose a novel rendering system using both model-based and image-based rendering as shown in Figure 14.

For rendering on clients, we use the image-based method like Light Field Rendering¹²⁾ because the size of transferred image-based data can be decreased more efficiently than that of model-based data. However, pure image-based method cannot accurately reconstruct arbitrary view images. So we use a small amount of model-based data in combination with image-based data to accurately reconstruct view images, as proposed in the Lumigraph⁷⁾.

Our system locates model-based data represented by the hierarchical structure on a remote server, in order to avoid the channel limitation between the server and the client. Offline, the server pre-renders the mesh models from various viewing positions, and stores these images in an image repository.

At run time, The server sends back the pre-rendered images necessary to calculate the view as well as the sparse mesh model[in response to requests from clients. The client calculates and displays the new view, using image-based rendering with the set of images and the sparse mesh model from the server.

The name of our system, “Grid-Lumigraph,” is derived from the sampling and retrieving pattern of images. In our system, the images are sampled from all grid points of the voxel space that split the viewing box in regular intervals. Previous IBR methods, such as Light Field Rendering and Lumigraph, can require images sampled from fixed planes, which are six faces of the box enclosing the target object. On the other hand, our system has images sampled from all grid points, so we need only images on the grid points nearest to the viewpoint, and we can decrease the number of needed images.

Our Grid-Lumigraph projects the sampling images on the 3D mesh model like texture mapping. Once a viewpoint is selected by a user at run time, the Grid-Lumigraph chooses the nearest sampling points around the viewpoint, extracts

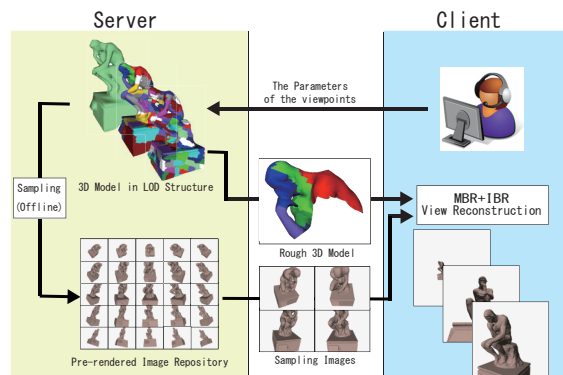


Fig. 14 The overview of proposed rendering system through network

images assigned to those points, and then sends and projects those images onto the 3D mesh model on the client. The Lumigraph calculates the nearest rays from the Light Field by using a 4D function, while our Grid-Lumigraph determines the nearest images from the image repository. So the amount of needed data for reconstruction of one view can be kept to less than that needed by the Lumigraph. Additionally, this procedure can be efficiently done by the GPU's texture mapping capability, shown in Figure 15.

5.2 Data Construction (Offline Process)

We propose an offline process of our system in advance of the run time rendering process.

In the offline process, we prepare certain types of data on the server for the rendering process on clients. Our system constructs LOD geometric data of an input 3D model that is applicable for rapid construction of the image repository, and for efficient transfer of model data. Additionally it generates the pre-rendered images to be sent to clients, and stores them in the repository on a server machine. Here, we describe the details of those preparatory procedures.

5.2.1 Construction of LOD Geometric Data

In the first of the offline procedures, we convert the original 3D model into a format that allows us to handle it in real time. The streamlining of 3D model data enables us not only to transfer itself to clients without overload but also to

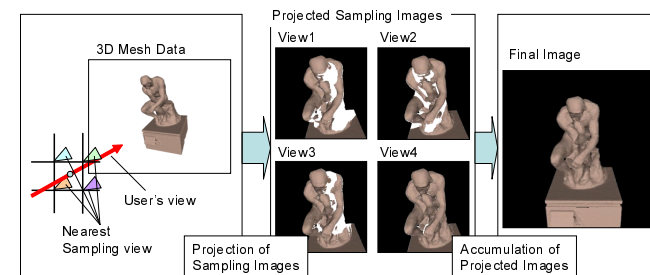


Fig. 15 The reconstruction of images by projective texture mapping using GPU capability. We can efficiently extract each pixel's correct color from sampling images by texture projections. Each nearest sampling image is projected from the sampling point to the geometric data. By accumulating the results in the final buffer, we can obtain the reconstructed image.

construct the database for pre-rendered images very quickly.

To efficiently convert 3D model data, we use a multiresolution data structure based on polygons described in Section 3. The 3D model data stored in the hierarchical format can be efficiently extracted and transferred. While the data size is large, our multiresolution data structure, which is composed of small meshes with a predefined number of polygons, is capable of transferring model data that is partial and has variable resolution. When we want to transfer partial mesh data, we traverse the hierarchy structure and select the set of needed mesh data in the same way as rendering.

5.2.2 Building a Sampling Repository

Next, we construct a set of pre-rendered images, referred to as a sampling repository, using the 3D model with multiresolution constructed in the previous step.

We sample the viewing space around the input 3D model. In our implementation, we assume that the viewing space is defined a box twice as large as the input 3D model that can enclose and cover all aspects of the model. Here, we locate the sampling viewpoints at regular intervals in the box, as shown in Figure 16. Regular sampling is simple, and can be easily implemented and processed. The granularity of sampling is manually decided depending on the density and complexity of the input model.

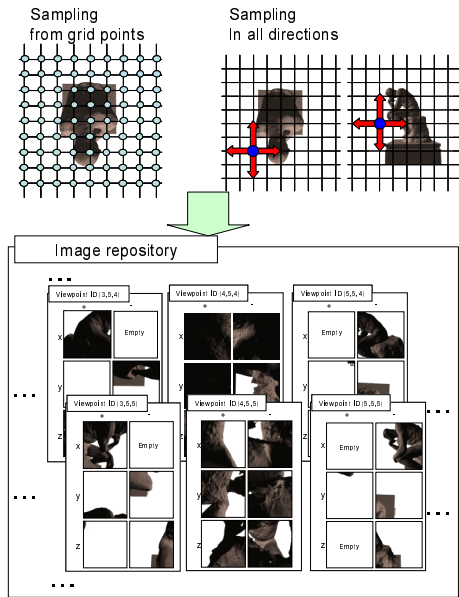


Fig. 16 Viewpoints for sampling are located on each grid point of the voxel space surrounding the input 3D model. View directions are set along axes x, y, and z. The image repository manages sampling images in a hash table.

At each viewpoint, we generate an image of the object. To sample in all directions, we generate images viewed from each point in six directions along the axes, positive and negative directions of x, y, and z described in Figure 16. We record the image viewed in each direction by a direction index that assigns a number from 0 to 5 to the direction. We set each face to one image plane, set the aspect ratio to 1, and set the angle of the field of view to 90 degree, to capture all rays passing through the viewpoint. By these settings, we can completely include all rays passing through the viewpoint in six images.

We register those sampling images in the image repository, where they are stored formatted as JPEGs. Each registered image is managed by using a hash table with the combination of grid point ID and direction index, which can be represented by four integers as (x_i, y_i, z_i, d) , and can be quickly retrieved with those parameters at run time. If there are no mesh patches to be rendered in an

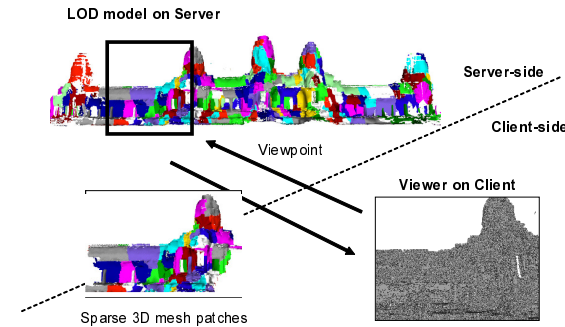


Fig. 17 Communication of the geometric data. The server provides very sparse mesh patches depending on the client's viewpoint. The sparse mesh patches are chosen from the LOD hierarchy structure. In this figure, each part with color in the model is a mesh patch in the LOD hierarchical structure. The server sends only sparse mesh patches displayed in the client's view at run time.

image, we can skip recording the image and register an empty flag instead of an image in the repository.

5.3 View Reconstruction (Online Rendering)

After the offline pre-processing, our system server, which has an image repository, can provide the relevant images and 3D mesh models. The clients for view generation receive this data from the server, and reconstruct arbitrary views by using Grid-Lumigraph. This section describes the details of data exchange between the server and the clients over the network.

When a user requests display of the target 3D model from a particular viewpoint on a client system, the client system sends the parameters of the current viewpoint to the server system, such as viewing position and the current viewing direction, represented by six floating points $(x_p, y_p, z_p, x_d, y_d, z_d)$. Once the server receives parameters of the current viewpoint, the system determines the set of the nearest sampling points. Then, the server can easily retrieve images assigned to each nearest point from the image repository, using a hash key as the calculated grid point's ID, and send them to clients.

The server also sends the sparse 3D model to the clients in addition to the image data. The server has the sparse 3D mesh model formatted in the multiresolution hierarchy. For the initial request, the server sends the entire model, and it sends

only corresponding mesh patches at the later requests, visible from the current viewpoints, described in Figure 17. The resolution of each mesh on rendered images can be obtained by calculating the projected error value of each mesh in the same way as it is checked in multiresolution rendering proposed in Section 3. If the projected error value exceeds the threshold, the client requests new mesh patches. In this transfer step, we set the threshold to less value than that for local rendering described in Section 3, because we need only sparse meshes to correct the image-based rendering result.

5.4 Experimental Results

We evaluate the efficiency of our proposed system in a practical environment by measuring the rendering speed and the rate of data transfer.

We implemented and performed the server and client functions on a 2.4GHz AMD Athlon64 X2 PCs with 4GB RAM, which has GeForce 8800GTS with 512MB of video memory. Our system runs on Windows XP. We used a 1Gbit LAN between the server and the client. We used NVIDIA Cg Toolkit for implementation of details in GPU processing, which include multiresolution rendering on the server and image projection and accumulation on the clients. The dimension of sampling images is set to 512 pixels.

We measured the rendering performance of three huge models using the system, and shows those results in Table 2. Those input models have large numbers of triangles, from one million to twenty million. We constructed the image repositories for those models whose sampling granularity per dimension is 16 or 32, described as grid number in Table 2. In the experiment, all models were efficiently rendered in real time, over 30 fps, on clients, even if the input model was very large.

Additionally, we also evaluated and compared the rendering speed, and data transferring rate. of our proposed method with the model-based LOD method that renders 3D views by receiving and using only geometric data formatted in LOD. We used a model that contained 6,604,908 polygons, moving the viewpoint along a path in a certain time, and measuring the rendering speed and transferred data amount of both methods. And we evaluated and compared the rendering quality of images rendered by both methods.

The left of Figure 18 shows the result of rendering speed. From this result,


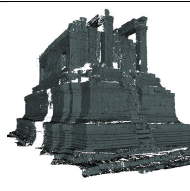
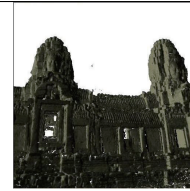
			
model	Bayon Face	South Library	Bayon 3 Towers
Input polygons	5,922,790	9,162,909	18,132,893
Grid number	16	16	32
Average fps	48.1 fps	42.5 fps	38.5 fps

Table 2 The parameters of input 3D models and rendering results on the client.

we observed that Grid-Lumigraph can constantly render faster than the model-based LOD method. The model-based LOD method renders less data according to the close-up scene, but can need more amount of data when rendering complex scenes. On the other hand, Grid-Lumigraph uses a constant number of images to represent the detailed shape, so it can render images at stable speed.

The result of the data transferring rate is shown in the right chart of Figure 18. From this figure, we can observed that Grid-Lumigraph can also render images with transfer of less data than the model-based LOD. The size of data transferred in the model-based LOD method exceeds 500 kilobytes many times in the sequence. On the other hand, the average size of transferred data in Grid-Lumigraph can be kept to less than 300 kilobytes in the sequence. We can say that our Grid-Lumigraph system is more applicable to implement practical systems than pure model-based LOD method.

6. Conclusions

In this thesis, we have proposed a 3D interactive system that makes archived digital information related to huge 3D models accessible to ordinary people. Our system has two major functions: one is the assignment function that allows users to select specified regions on 3D models and assign various kinds of information to their parts. Another is the display function, which enables users to freely view the target 3D model from any viewpoint, and browse the information assigned to the specified regions on the model.

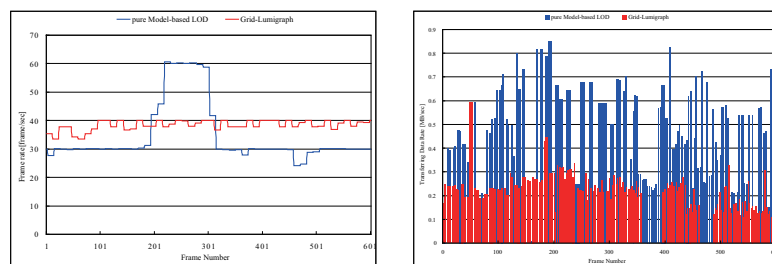


Fig. 18 Comparison the performance of Grid-Lumigraph with pure model-based LOD method. Left chart shows rendering speed (frames per second), and right chart shows transferring data rate (kilobytes per second).

To develop this system, there are technical issues to be resolved. In this thesis, we have presented some technical propositions as follows. First, we have proposed novel multiresolutional rendering methods that have data structures formatted in the hierarchical form to display very huge 3D models. Second, we proposed a novel selection tool using graph-cut algorithm with very intuitive interface. It allows users to select the specified regions on huge 3D models on our system. Finally, we proposed a novel rendering method to extend our proposed 3D system to network environments that will allow users to access it from all over the world. In this paper, we use a hybrid method “Grid-Lumigraph” that combines a model-based rendering method with an image-based method to render images of huge 3D models with minimum data transfer and high quality.

Our proposed system enhances the value of huge 3D models by helping general users easily access and use many kinds of information that could not previously be effectively visualized and utilized.

As future work, we would like to deploy our proposed system in a practical environment, and to extend it to become a more meaningful system. Additionally, we would like to improve Grid-Lumigraph, to use our system practically on general network environments.

References

- 1) Bernardini, F., Rushmeier, H., Martin, I.M., Mittleman, J. and Taubin, G.: Building a Digital Model of Michelangelo’s Florentine Pieta’, *IEEE Computer Graphics*

- and Applications, Vol.22, No.1, pp.59–67 (2002).
- 2) Boykov, Y. and Jolly, M.-P.: Interactive Graph Cuts for Optimal Boundary and Region Segmentation of Objects in N-D Images, *Proceedings of International Conference on Computer Vision 2001*, pp.105–112 (2001).
- 3) Boykov, Y. and Kolmogorov, V.: An Experimental Comparison of Min-Cut/Max-Flow Algorithms for Energy Minimization in Vision, *Energy Minimization Methods in Computer Vision and Pattern Recognition* (2001).
- 4) Cignoni, P., Ganovelli, F., Gobetti, E., Marton, F., Ponchio, F. and Scopigno, R.: Adaptive TetraPuzzles - Efficient Out-of-core Construction and Visualization of Gigantic Polygonal Models, *ACM Transaction on Graphics (Proceedings of ACM SIGGRAPH 2004)*, Vol.23, No.3, pp.796–803 (2004).
- 5) Garland, M. and Heckbert, P.S.: Surface Simplification Using Quadric Error Metrics, *Proceedings of ACM SIGGRAPH 97*, pp.209–216 (1997).
- 6) Geman, S. and Geman, D.: Stochastic relaxation, gibbs distributions, and the bayesian restoration of images, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp.721–741 (1984).
- 7) Gortler, S.J., Grzeszczuk, R., Szeliski, R. and Cohen, M.F.: The Lumigraph, *Proceedings of ACM SIGGRAPH 96*, pp.43–54 (1996).
- 8) Hoppe, H.: Progressive meshes, *Proceedings of ACM SIGGRAPH 96, Computer Graphics*, pp.99–108 (1996).
- 9) Ikeuchi, K. and Miyazaki, D.: *Digitally Archiving Cultural Objects*, Springer (2008).
- 10) Ikeuchi, K. and Sato, Y.: *Modeling from Reality*, Springer (2001).
- 11) Karypis, G. and Kumar, V.: Multilevel k-way partitioning scheme for irregular graphs, *Journal of Parallel and Distributed Computing*, Vol.48, No.1, pp.96–129 (1998).
- 12) Levoy, M. and Hanrahan, P.: Light Field Rendering, *Proceedings of ACM SIGGRAPH 96*, pp.31–42 (1996).
- 13) Levoy, M., Pulli, K., Curless, B., Rusinkiewicz, S., Koller, D., Pereira, L., Ginzton, M., Anderson, S., Davis, J., Ginsberg, J., Shade, J. and Fulk, D.: The Digital Michelangelo Project: 3D Scanning of Large Statues, *Proceedings of ACM SIGGRAPH 2000, Computer Graphics*, pp.131–144 (2000).
- 14) Li, Y., Sun, J., Tang, C.-K. and Shum, H.-Y.: Lazy Snapping, *ACM Transaction on Graphics (Proceedings of ACM SIGGRAPH 2004)*, Vol.23, No.3, pp.303–308 (2004).
- 15) of Japan, N.A.: National Archives of Japan Digital Archive (2005).
- 16) Yuan, X., Xu, H., Nguyen, M.X., Shesh, A. and Chen, B.: Sketch-based Segmentation of Scanned Outdoor Environment Models, *Proceedings of the 2nd Eurographics Workshop on Sketch-Based Interfaces and Modeling*, pp.19–26 (2005).