Published on October 2010

Image-based Network Rendering of Large Meshes for Cloud Computing

Yasuhide Okamoto[†], ***Takeshi Oishi***[†], ***Katsushi Ikeuchi***[†]

Abstract Recent advances in sensing and software technologies enable us to obtain large-scale, yet fine 3D mesh models of cultural assets. However, such large models cannot be displayed interactively on consumer computers because of the performance limitation of the hardware. Cloud computing technology is a solution that can process a very large amount of information without adding to each client user's processing cost. In this paper, we propose an interactive rendering system for large 3D mesh models, stored on a remote environment through a network of relatively small capacity machines, based on the cloud computing concept. Our system uses both model- and image-based rendering methods for efficient load balance between a server and clients. On the server, the 3D models are rendered by the model-based method using a hierarchical data structure with Level of Detail (LOD). On the client, an arbitrary view is constructed by using a novel image-based method, referred to as the Grid-Lumigraph, which blends colors from sampling images received from the server. The resulting rendering system can efficiently render any image in real time. We implemented the system and evaluated the rendering and data transferring performance

Keywords Huge mesh rendering, Image-based rendering, Network rendering

1. Introduction

Typically, e-Heritage digitization results in mesh models consisting of billions of triangles with high complexities, so it has been difficult to view digitized objects in real time on current consumer computers. First, the current Internet does not have the capability to download such mesh models in real time. Second, the usual PC on the client side cannot render such models in real time. As a solution to these problems, a process called "cloud computing" has been proposed to deal with large-scale information with minimal cost to the user. Cloud computing has server machines to process large data sets and abstract them as "clouds" for client users. Client users can obtain data from the cloud freely and speedily without the expertise and special knowledge they would need to manipulate a large amount of raw data. In this paper, we propose an online rendering system applicable for the cloud computing system.

This paper proposes an efficient rendering system by both model- and image-based rendering as shown in Figure 1. Our system locates original mesh models of cultural assets on a remote server, in order to avoid the channel limitation between the server and the client. The server pre-renders the mesh models from various viewing positions, and stores these images in a repository based on our rendering method referred to as Grid-Lumigraph. At run time, the client sends a request to display the mesh model at a certain view position and specifies the viewpoint parameters. The server sends back the pre-rendered images necessary to calculate the view as well as the sparse mesh model. The client calculates and displays the new view by using the image-based rendering with the set of images and the sparse mesh model from the server.

This paper has the following structure. Section 2 surveys the related work and discusses the benefits and drawbacks of the methods. In Section 3 we describe Grid-Lumigraph, which is the image-based method for reconstruction of arbitrary views from sampling images. In section 4, we describe the construction of an LOD-based model and a repository composed of sampling images. We explain the detail of our proposed server-client rendering system in Section 5, demonstrate and evaluate the system in section 6, and we conclude in section 7.

2. Related work

The Level of Detail (LOD) method is proposed for displaying large mesh models in [14]. LOD methods represent 3D objects with mesh models in multiresolution. The Progressive Mesh presented in [7] records the sequence of the reduction that merges smaller triangles to form larger ones in the mesh structure. The Adaptive Tetrapuzzles proposed in [1] converts the input mesh into a hierarchy structure composed of nodes, containing smaller meshes, referred to as patches with multiresolution. Sending mesh data over the network is very expensive, though.

The LOD methods are also used in point-based representations. QSplat [15] and Layered Point Cloud [4] are point-based rendering systems, which use points as the rendering primitive with a hierarchical data structure. Far Voxels proposed in [5] uses points and voxels with view-dependent color as the rendering primitives in the LOD hierarchy. Those methods can be extended to

† Y. Okamoto, T. Oishi, K. Ikeuchi
3rd Dept., Institute of Industrial Science, University of Tokyo, 4-6-1 Komaba, Meguro-ku, Tokyo 153-8505 Japan

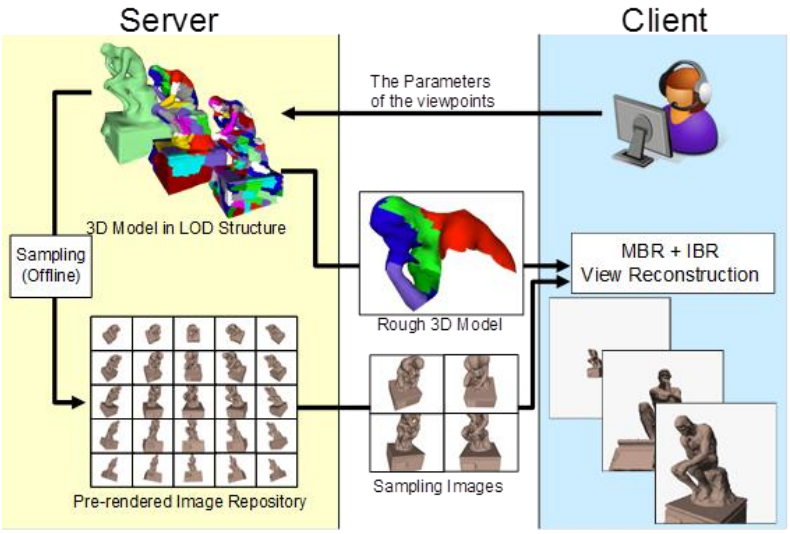


Fig. 1 Overview of proposed rendering system. The server contains geometric data recorded in the format of an LOD hierarchy and the repository of sampling images of the input 3D model. The client system displays arbitrary views reconstructed by using received images and sparse 3D mesh patches.

server-client rendering systems as proposed in [1, 4, 5, 7, 16]. Although this point-based representation is much more compact than the mesh-based, depending on the complexity of the input model, the communication traffic can still be high.

Like the geometry-based methods described above, image-based methods are also extended to a server-client method in [11]. In this approach, the server has the geometric model, and it renders and then sends images corresponding to the requests from clients. Although the server only needs to send an image in real time, rendering at the server is a costly operation, and, if too many requests occur, the server may break down.

Impostor, presented in [2, 9, and 17] is the rendering method on the network using geometric and image information. This system is mainly designed for walkthrough environments such as urban scenes. It assumes that a 3D model is already established on the client, which is not the case in our paradigm.

3. Grid-Lumigraph for image reconstruction

A client machine in our system can display any arbitrary view of the input 3D model from the viewing location and direction that a user chooses, using a new image-based method referred to as the "Grid-Lumigraph." Our Grid-Lumigraph reconstructs a view with only sampling images near current viewpoints and a rough 3D mesh model.

3.1 Grid-Lumigraph

Our Grid-Lumigraph has the same basic idea as light field rendering [12], which reconstructs a view from color values of all rays going through the view. The collection of all rays for this operation is referred to as the "light field."

Constructing a view image using the light field rendering method is depicted in Figure 2 (a). The dotted line indicates one of the new rays necessary for rendering the image to be updated. To calculate the color of the dotted ray, we extract the color values on the nearest rays s1-u1, s1-u2, s2-u1, and s2-u2, and blend them into the new color value assigned to the target ray. This light field method requires dense sampling of s-u pairs; otherwise, the reconstructed image may have blurs and ghosting, as shown in Figure 2 (b).

The Lumigraph [6], shown in Figure 2 (c), utilizes geometric information to remedy this issue. The dotted line indicates a necessary ray that needs to be calculated. The geometric information provides the nearly correct position of the intersection point between the ray and the object surface, which helps use more appropriate rays. This correction does not need a completely precise 3D model. To correct view images, our Grid Lumigraph has a level-of-detail (LOD) 3D mesh model on the server, extract coarse meshes from it, and provides clients with them. This supply of LOD based model enables us to more flexibly and efficiently maintain enough geometric data on clients than Lumigraph.

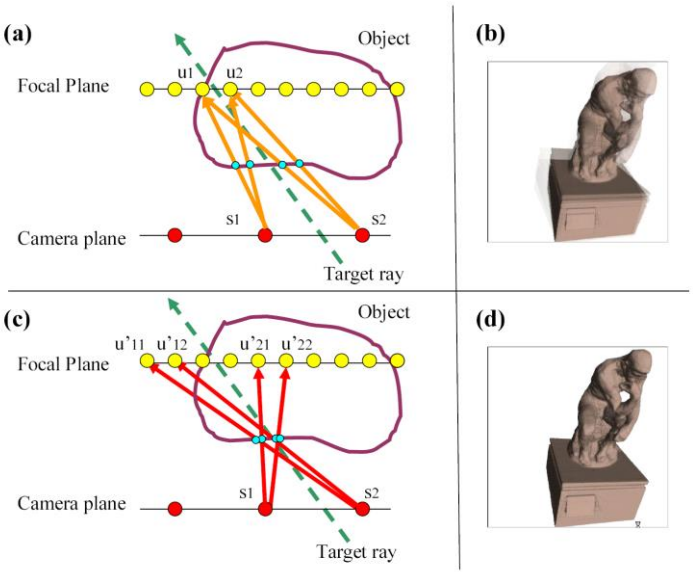


Fig. 2 Correct sampling with geometric information. (a) Extraction colors from sampled colors without geometric information. (b) The result without correction. (c) Extraction with geometric information. (d) The result with correction using geometric information.

Our Grid-Lumigraph projects the sampling images on the 3D mesh model like texture mapping. The image repository stores sampling images at each viewpoint in 3D space. Once a viewpoint is selected by a user at run time, the Grid-Lumigraph chooses the nearest sampling points around viewpoints, extracts images assigned to those points, and then sends and projects those images onto the 3D mesh model on the client. The Lumigraph and our Grid-Lumigraph have the same principle of using image-based rendering with a rough 3D mesh model. The Lumigraph calculates the nearest rays from the light field, while our Grid-Lumigraph determines the nearest images from the image repository constructed of images sampled from each grid point of the object 3D space. Our Grid-Lumigraph only needs texture mapping capabilities and is very efficient for rendering.

3.2 Grid-Lumigraph on GPU

The Grid-Lumigraph can be easily implemented to effectively use the capability of a GPU. For generation of a new image, the Grid-Lumigraph uses sampling images, typically four to twelve, around the viewpoint selected by a user. Then, the Grid-Lumigraph projects each sampling image, one by one, using the projected texture mapping method onto the 3D mesh model. This procedure can be efficiently done by the GPU's texture mapping capability, shown in Figure 3. In this projection, shadow mapping of the GPU is also utilized to avoid mapping rays to back faces of the 3D mesh from the sampling viewpoint. These mapping results are projected and normalized on the current viewing image plane of the user, using the built-in GPU capability.

4. Data construction for Grid-Lumigraph (offline process)

In the offline process, our system generates and stores the pre-rendered image, to be sent and rendered on clients by Grid-Lumigraph, in the repository on a server machine. The Level of Detail (LOD) rendering method extended from Adaptive Tetrapuzzles [1] is employed for rapid construction of the image repository.

4.1 Constructing LOD Hierarchy

Our method to construct an LOD hierarchy is different from that of Adaptive Tetrapuzzles. The Adaptive Tetrapuzzles recursively divide the space and construct a hierarchical structure. The approach of this method is simple to implement and efficient to process. However, the number of triangles of our target 3D models is huge, so the recursive splitting process is time-consuming. To solve this problem, we perform the splitting process not by triangles but by small meshes. Our method defines a voxel space of pre-determined resolution and generates a group of small meshes (Step 1), forms a graph of divided meshes for constructing a tree structure from them (Step 2), and then simplifies the tree structure (Step 3). We prefer this method for space efficiency adapted to the object shape in hand.

Step1: Decomposition into voxel space. We define the voxel space at the finest level. Then, we decompose an input mesh model into smaller meshes using the voxel space. We sort each triangle to a single voxel that

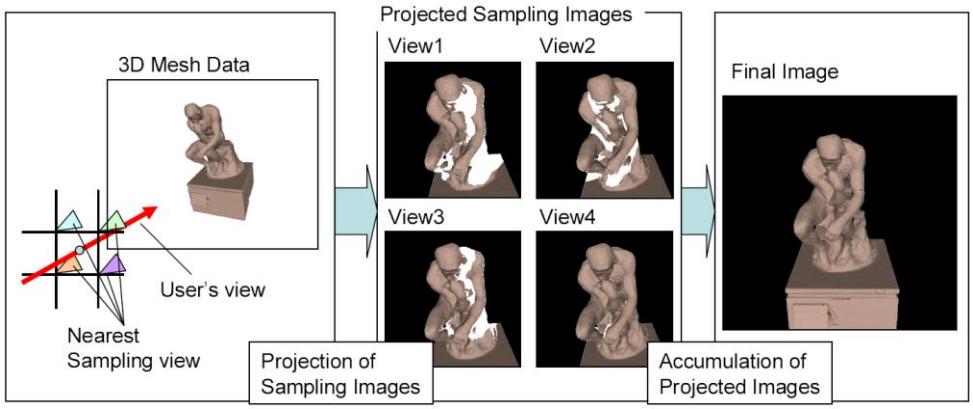


Fig. 3 The reconstruction of images by projective texture mapping using GPU capability. We can efficiently extract each pixel's correct color from sampling images by texture projections. Each nearest sampling image is projected from the sampling point to the geometric data. By accumulating the results in the final buffer, we can obtain the reconstructed image

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 Nv.

Step2: Recursive graph partitioning. We convert a group of meshes, defined in Step 1, into a graph representation, G (V, E), for the construction of an LOD structure in the next step. One vertex in the vertex set, V, of the graph corresponds to one voxel, and one edge in the edge set, E, corresponds to the adjacent relation between two voxels. Thus, in this graph, at most, one vertex has six edges corresponding to six adjacent voxels. For splitting the graph evenly and adaptively for shapes, we assign a weight to each vertex and each edge. Here, a vertex weight is defined as the number of triangles belonging to the mesh. To define edge weight, first, we calculate the mean surface normal vector at each vertex. Then, an edge weight is defined as the inner product of two averaged vectors of the two vertices on the edge.

We recursively partition the graph into a pair of sub-graphs, shown in Figure 4. This recursive partitioning continues until the size of each sub-graph, defined as the number of the belonging triangles, becomes less than a pre-defined number Nl. In our system, this graph partitioning procedure is implemented by using the Metis library [10].

After this procedure, we can obtain a hierarchy structure in which each leaf node has a small sub-graph of the size Nl. Non-leaf nodes contain a larger sub-graph. At each node in the graph structure, we connect all meshes of that node into a continuous mesh patch.

Step 3: Simplification at non-leaf nodes. We simplify a mesh patch at each non-leaf node in the constructed hierarchy. The number of triangles at each node is reduced to a pre-defined number Nn. We implemented this simplification method by using a quadric error metric [3]. This method iteratively collapses edges from the lowest edge, in ascending order of quadric errors, until the number of triangles becomes the desired number. Here the quadric error represents a rough approximation of the distance between original and simplified mesh patches.

Each node holds the node parameters and the geometric data of a simplified mesh patch. The node parameters given by a mesh patch are eight corner positions of the bounding box, the range of surface normal, and the minimum quadric error calculated in simplification. The range of surface normal of the mesh is represented by one 3D vector of the mean normal of the mesh and one scalar value meaning the maximum difference of angle from the mean vector. And the minimum quadric error means the distance between the simplified mesh and the original mesh, and it is used to parameterize the resolution of the mesh rendered on the screen. The geometric data consists of positions and connectivity relations of polygonal vertices in the mesh patch. Node parameters will be used for traversing the hierarchical structure, while the geometric data is used for rendering the mesh patch.

The simplification procedure ensures consistency of boundaries between mesh patches. Inconsistency between mesh patches causes holes and artifacts along the patch boundary on rendering. Simplification is not conducted on edges either along boundaries or directly connected to boundaries.

4.2 Efficient 3D LOD rendering

In order to generate a set of pre-rendered images, our system uses the 3D LOD structure. The rendering process traverses the constructed hierarchy from the root node along the tree structure following the depth-first search strategy. If the process successfully finds a mesh patch that satisfies necessary resolution, the process tracks back following the depth-first search strategy,

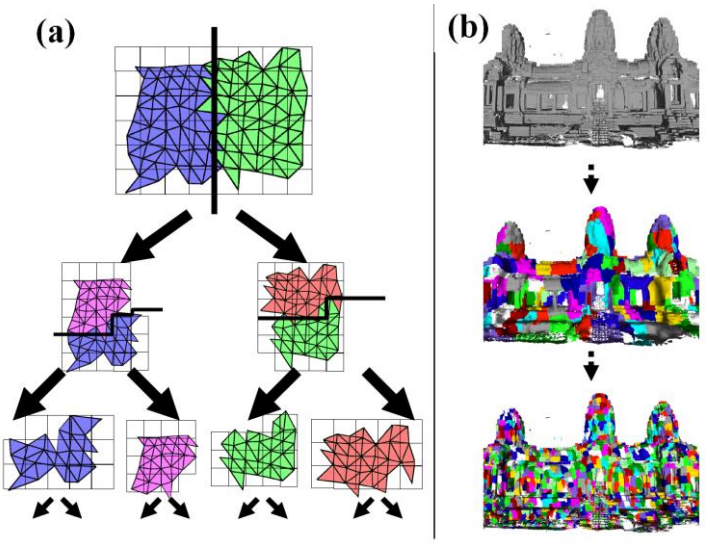


Fig. 4 Construction of Level-of-Detail hierarchy.
(a): Recursive splitting of mesh by graph-partitioning of voxel space.
(b): Results of partitioning into small mesh patches.

while it adds the node to a list of rendering nodes. The necessary resolution is given by the projected quadric error , where ϵ and r are the quadric error at the node, and the distance between a viewpoint and the center of the mesh patch. If the procedure reaches a leaf node, it is also added to the list. After traversing the entire tree structure, the resulting list of rendering nodes is given to the rendering pipeline.

Some exceptional cases occur in the tree traverse, which are out of screen and back-face. An out-of-screen case is given by a node whose bounding box is projected outside of the screen. In a back-face case, all triangles in the mesh patch of traversed node turn away from the viewing direction. We easily judge the back-face case by checking whether the viewing vector intersects with the range of normal. If one of these cases occurs, the traverse operation immediately backtracks along the tree structure following the depth-first strategy.

4.3 Building a sampling repository

The previous section described the method to traverse the LOD structure. In this section, we construct a set of pre-rendered images, referred to as a sampling repository. Later, a group of images from this repository will be sent to a client for image-based rendering.

We sample the viewing space. In our implementation, we assume that the viewing space is defined as a box twice as large as the input 3D model, divided into regularly located sampling viewpoints. The granularity of sampling is empirically decided depending on the density and complexity of the input model. At each viewpoint, we generate an image of the object using the LOD model, in six directions along the axes, positive and negative directions of x, y, and z described in Figure 5. 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 degrees, to capture all rays passing through the viewpoint. If there are no mesh patches to be rendered in a sampling image, we can skip recording the image and register an empty flag instead of an image. Sampling images are stored as JPEGs in the image repository.

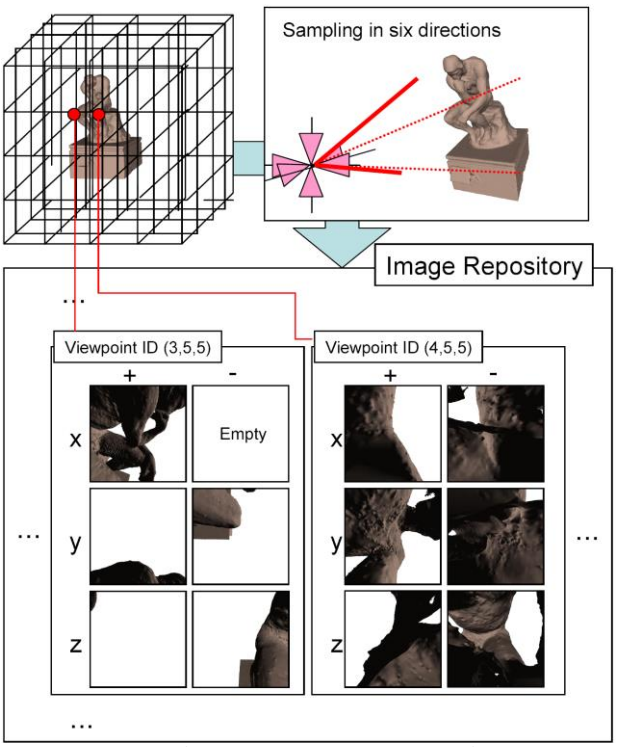


Fig. 5 Sampling from each grid point. Viewpoints for sampling are located on each grid point of the voxel space surrounding the input 3D model. View directions are set along axes x, y, and z. The image repository manages sampling images in a hash table.

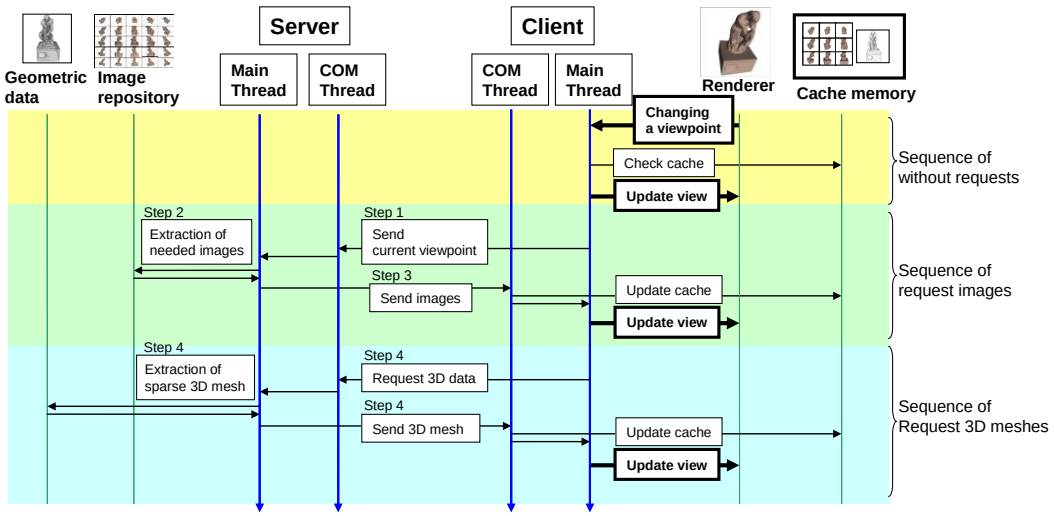


Fig. 6 The sequence chart for the rendering pipeline on our system. If the client's cache has enough data to reconstruct the current view, the client's thread can display the view without requests to the server. Otherwise, the client makes requests for images and 3D meshes to the server. The data communication is managed by COM threads, and it is asynchronously done. The renderer updates the view whenever the data communication is finished.

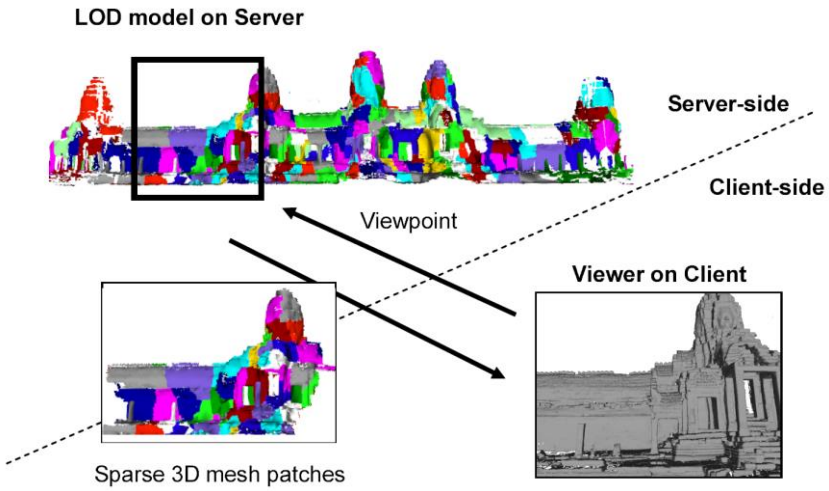


Fig. 7 Communication of the geometric data. The server provides very sparse mesh patches depending on the client's viewpoint. The sparse mesh patches are chosen from the LOD hierarchy structure.

Those images are retrieved using a hash table with the combination of grid point ID and direction index which can be represented by four integers as (xi, yi, zi, d).

5. Rendering system using Grid-Lumigraph through the network (online process)

Our system has a server for image repository and clients for view generation. This section describes the details of data exchange between the server and the clients over the network.

5.1 Protocol for rendering

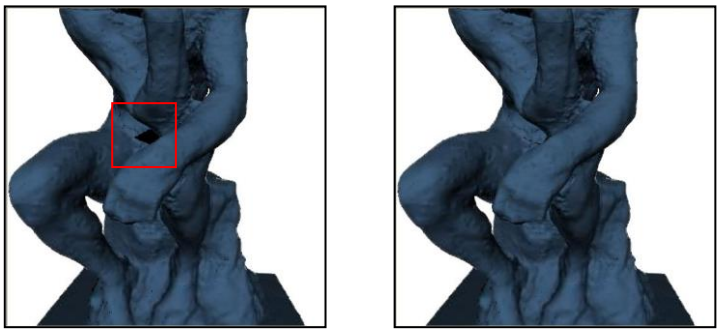
The protocol between a server and clients for rendering is described in Figure 6.

Step 1: 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. Those parameters include the current viewing position and the current viewing direction, which are represented by six floating points.

Step 2: Once the server receives parameters of the current viewpoint, the system determines the set of the nearest sampling points. Then we 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 the viewing direction.

Step 3: The server sends retrieved sampling images to the client. Before sending images, the server checks whether the client has already had the same image by using sent information from the client. The server sends retrieved images only if the client does not have the image.

In addition, the server sends extra images to the client at the same time. In one process of sending images, the server sends not only images nearest to viewing



(a) Rendering result with holes

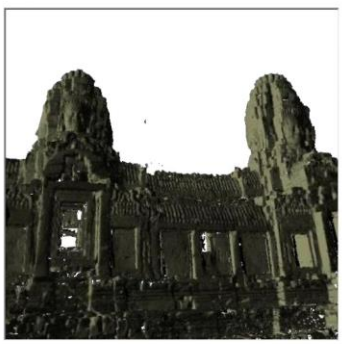
(b) Rendering result after ad hoc filling

Fig. 8 Filling holes. (a) is a rendering result with some holes because of the low sampling granularity. In the interactive time, the filling procedures are done ad hoc by using sparse model data located on the client as shown in (b).

Model	Thinker
Input Triangles	1,742,122
Sampling Grid	16
Average fps	59.5

Model	Bayon Face
Input Triangles	5,922,790
Sampling Grid	16
Average fps	48.1

Model	Menandro's house
Input Triangles	10,388,120
Sampling Grid	32
Average fps	33.7



Model	Bayon 3 towers
Input Triangles	18,132,893
Sampling Grid	32
Average fps	38.5

Fig. 9 The parameters of input 3D models and rendering results on the clients. Some white holes in "Bayon Face" and Menandro's house are originally contained in the input models, and those are unscanned parts.

directions at the nearest sampling point (for example, the x direction at a point), but also images in other directions at the points (for example, the y and z directions at the point). The client saves received images as cache in local memory. The stored sampling images in the cache are managed in the manner of least recent used (LRU).

Step 4: The server sends the sparse 3D model to the clients. The server has the sparse 3D mesh model formatted in the LOD hierarchy. For the initial request, the server sends the entire model, which is composed of coarse mesh patches extracted from around the top part of the hierarchy data structure. For the later requests, it sends the corresponding mesh patches visible from the current viewpoints, described in Figure 7. The clients request new mesh patches, if necessary. In particular, when zooming in toward the object, the number of

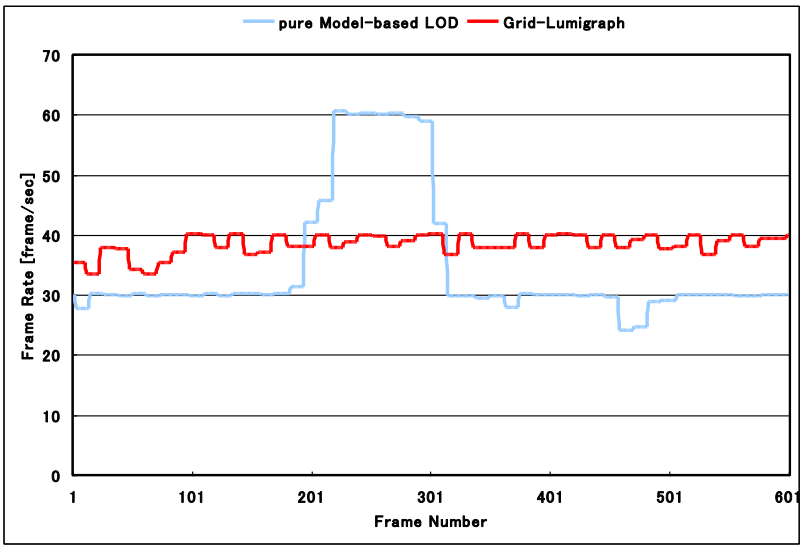


Fig. 10 Rendering speed (frames per second) of pure model-based LOD method and Grid-Lumigraph.

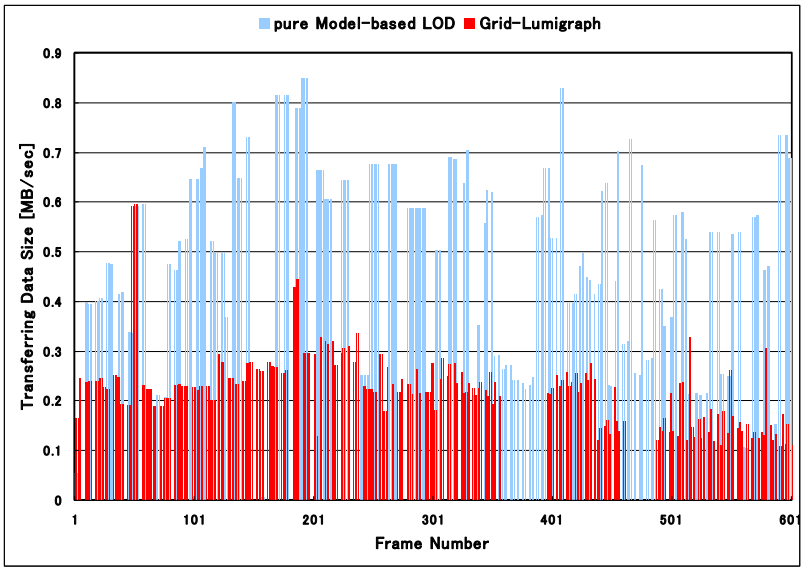


Fig. 11 Transferring data rate (megabytes per second) of pure model-based LOD method and Grid-Lumigraph

vertices displayed is reduced. The client requests new mesh patches in a finer resolution if the projected quadric error of a mesh patch is less than a predefined value.

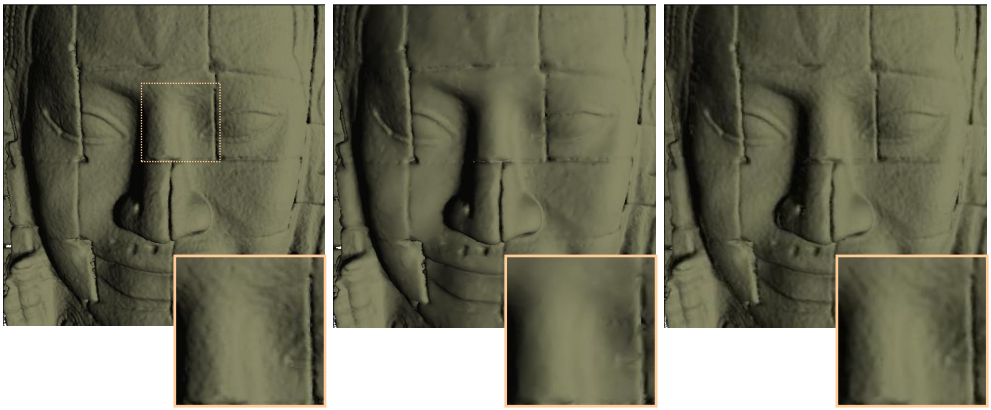
5.2 Additional capabilities for better performance

When there is a rapid change in viewpoints, the server and the client work asynchronously to avoid stalling. The client continues to send the parameters of the viewpoint, while it renders a new image using other images available in cache. The server continues to send sampling images corresponding to the received requests, and the client updates the rendered results whenever new data is received from the server.

In our system, some holes may occur in reconstructed images from sampling images as shown in Figure 8 (a). The main reason is that sampling granularity is lower than the complexity of the input object's shape. If the holes are small, we cover them by calculating shading effect using the surface orientation of the triangles and the light source direction as shown in Figure 8 (b). When the server is idling, the client requests the server to generate the current view from the current viewpoint and send it. The client can instantly update the displayed image by using it.

6. Implementation and results

We implemented and performed the server and client functions on a 2.4GHz AMD Athlon64 X2 PC 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



(a)Highest resolution (ground truth) (b)Pure model-based LOD (c)Grid-Lumigraph

Fig. 12 Comparison of the quality of images rendered by (a) highest resolution model, (b) pure Model-based LOD method, and (c) Grid-Lumigraph. Right bottom images are magnified views

used NVIDIA Cg Toolkit for implementation of details in GPU processing, which include LOD rendering and image projection.

We constructed the LOD structure on the server, setting Nv to 500, Nl to 4000, and Nn to 2000. The size of cache on clients is 50 MB for images and geometric data. The dimensions of sampling images are set to 512 by 512 pixels.

The rendering results on clients are shown in Figure 9. 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 sampling grid in Figure 9. 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, the size of transferred data, and rendering quality of our proposed method with the pure model-based LOD method that renders images by using only geometric data. We used the model of "Bayon Face," moving the viewpoint along a path in a certain time, and evaluated the rendering speed and transferred data.

The result of rendering speed is shown in Figure 10. From this result, we observed that Grid-Lumigraph can constantly render faster than the pure model-based LOD method. The model-based method renders less data according to the close-up scene, but can need more amount of data in other complex scenes. On the other hand, Grid-Lumigraph uses a constant number of images to render views, so the rendering speed is very stable shown in Figure 10.

The result of the data transferring rate is shown in Figure 11. From this, we observed that Grid-Lumigraph can also render views with transfer of less data than pure model-based LOD method. The size of data transferred in the model-based LOD method exceeds 500 kilobytes many times in the sequence. In contrast, the size of transferred data in Grid-Lumigraph can be kept to less than 300 kilobytes. We can say that our system is more applicable for network environments than pure MBR method.

Finally, we show the original image and images rendered by both methods in Figure 12. We decided the resolution of the MBR method and Grid-Lumigraph so that two methods can render images in real time and in almost equal speed. We observed that the image rendered by the model-based LOD were smooth and lost detail due to the simplification. On the other hand, we observed detailed bumpy surfaces in the Grid-Lumigraph image that were generated by projecting pre-rendered images. The pure MBR method is better than Grid-Lumigraph for sharpness of the silhouette, but on the whole, the image rendered by Grid-Lumigraph is more similar to the highest resolution image than pure MBR.

7. Conclusion

We proposed a view-dependent rendering system for large-scale 3D models located on a remote server. In our approach, we use a model-based rendering method for repository generation at the server and a novel image-based method, referred to as the Grid-Lumigraph, for view generation on the client. A model-based rendering system on the server generates rendering images from various view positions using the LOD structure and stores these sampling images in the image repository.

The rendering system on the client displays a requested view using the Grid-Lumigraph, which uses a sparse 3D mesh model and sampling images nearest to the viewpoint. The Grid-Lumigraph, a variation of the light field method, projects blended sampling images using the projective texture mapping method. The Grid-Lumigraph can be implemented effectively on a GPU.

Our system can display very large 3D models, which have a huge number of triangles, in real time with efficient data communication.

For future work, we would like to extend our system to

work on the practical cloud computing system, and evaluate the performance under different network environments. Therefore we will improve the image sampling strategy of Grid-Lumigraph depending on the geometric complexity and access frequency, and develop the compression method for efficient data communication.

Acknowledgments

The research described herein was supported, in part, by the Japanese Ministry of Education, Culture, Sports, Science and Technology. The 3D models of cultural assets were digitized with the cooperation of the Japanese Government Team for Safeguarding Angkor (JSA) and the National Museum of Western Art, Tokyo.

References

[1] P. Cignoni, F. Ganovelli, E. Gobetti, F. Marton, F. Ponchio, and R. Scopigno. Adaptive TetraPuzzles – efficient out-of-core construction and visualization of gigantic polygonal models. ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2004), 23(3):796–803, 2004.

[2] S. Dobbyn, J. Hamill, K. O'Conor, and C. O'Sullivan. Geopostors: A real-time geometry/impostor crowd rendering system. ACM Transaction on Graphics (Proceedings of ACM SIGGRAPH 2005), 24(3):933–933, 2005.

[3] M. Garland and P. S. Heckbert. Surface simplification using quadric error metrics. In Proceedings of ACM SIGGRAPH 97, pages 209–216, 1997.

[4] E. Gobbetti and F. Marton. Layered point clouds: A simple and efficient multiresolution structure for distributing and rendering gigantic point-sampled models. Computers and Graphics, 28(6):2004, 2004.

[5] E. Gobbetti and F. Marton. Far voxels: a multiresolution framework for interactive rendering of huge complex 3d models on commodity graphics platforms. ACM Transaction on Graphics, 24(3):878–885, 2005.

[6] S. J. Gortler, R. Grzeszczuk, R. Szeliski, and M. F. Cohen. The lumigraph. In Proceedings of ACM SIGGRAPH 96, pages 43–54, 1996.

[7] H. Hoppe. Progressive meshes. In Proceedings of ACM SIGGRAPH 96, Computer Graphics, pages 99–108, 1996.

[8] K. Ikeuchi, K. Hasegawa, A. Nakazawa, J. Takamatsu, T. Oishi, and T. Masuda. Bayon digital archival project. In Proceedings of Virtual Systems and Multimedia 2004, pages 334–343, 11 2004.

[9] S. Jeschke, M. Wimmer, H. Schumann, and W. Purgathofer. Automatic impostor placement for guaranteed frame rates and low memory requirements. In Proceedings of the 2005 Symposium on Interactive 3D Graphics and Games, pages 103–110, 2005.

[10] G. Karypis and V. Kumar. Multilevel k-way partitioning scheme for irregular graphs. Journal of Parallel and Distributed Computing, 48(1):96–129, 1998.

[11] D. Koller, M. Turitzin, M. Levoy, M. Tarini, G. Croccia, P. Cignoni, and R. Scopigno. Protected interactive 3d graphics via remote rendering. In Proceedings of ACM SIGGRAPH 2004, pages 695–703, 2004.

[12] M. Levoy and P. Hanrahan. Light field rendering. In Proceedings of ACM SIGGRAPH 96, pages 31–42, 1996.

[13] M. Levoy, K. Pulli, B. Curless, S. Rusinkiewicz, D. Koller, L. Pereira, M. Ginzton, S. Anderson, J. Davis, J. Ginsberg, J. Shade, and D. Fulk. The digital michelangelo project: 3d scanning of large statues. In Proceedings of ACM SIGGRAPH 2000, Computer Graphics, pages 131–144, 7 2000.

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

[15] S. Rusinkiewicz and M. Levoy. QSplat : A multiresolution point rendering system for large meshes. In Proceedings of the 27th annual conference on Computer graphics and interactive techniques, pages 343–352, Jul. 2000.

[16] S. Rusinkiewicz and M. Levoy. Streaming QSplat : A viewer for networked visualization of large, dense models. In Proceedings of the 2001 Symposium on Interactive 3D Graphics, pages 63–68, 2001.

[17] J. Shade, D. Lischinski, D. H. Salesin, T. DeRose, and J. Snyder. Hierarchical image caching for accelerated walkthroughs of complex environments. In Proceedings of ACM SIGGRAPH 96, pages 75–82, 1996.