7. International Immersive Projection Technologies Workshop
9. Eurographics Workshop on Virtual Environments
J. Deisinger, A. Kunz (Editors)

A Panoramic Walkthrough System with Occlusion Culling

Lining Yang and Roger Crawfis

Department of Computer Science, The Ohio State University, Columbus, OH 43210, USA

Abstract

The interactive and high quality rendering of very complex scenes in a virtual environment is very difficult if not impossible to achieve. This research extends our previous EGVE paper that presents a panoramic walk-through system, allowing the user to move inside the datasets on a pre-defined track and look around interactively in both the horizontal and vertical directions. The interactivity is achieved by using Image-Based Rendering ideas to pre-compute the partial renderings for the reference viewpoints chosen on the track and store them on the server so that the client can retrieve and reconstruct novel views using the appropriate information. In this paper, we present simple and efficient methods to partition the scene into depth layers to avoid occlusion and dis-occlusion problems. We also present a novel track-dependent occlusion cullion algorithm to efficiently cull away unnecessary information. The system is tested using several scenes and provides real-time walkthroughs on even the most challenging scenes.

Keywords:
Image-based Rendering, Occlusion culling, Virtual Walkthrough

Categories and Subject Descriptors (according to ACM CSS): I.3.2 [Computer Graphics]: Graphics System; I.3.7 [Computer Graphics]: Virtual Reality; I.3.7 [Computer Graphics]: Visible Line/Surface Algorithms

1. Introduction

Complex renderings such as ray-tracing, global illumination, soft shadows and anti-aliasing can take a long time to render even simple scenes. Very complex scenes, such as those appearing at POVRAY's [20] competition site, can take hours or even days to finish rendering one frame. Therefore, interactivity for this quality of a virtual environment is not possible.

Without interactivity, the end-user has a much more difficult time in building a mental model of the environment. One of our primary goals was to provide a very low-latency framework that allows the user to stay focused on the scene, rather than the limitations of any system. Since interactive rates were impossible for complex renderings of virtual scenes, this implies that the resulting rendering or imagery needs to be pre-computed and more readily available. We apply novel Image-Based Rendering (IBR) ideas to achieve our goal. With reference views on a pre-defined path, we store the resulting imagery on a server, from which the client can request and build appropriate IBR models to reconstruct novel views.

Our goals for this project are:

i. To pre-compute information such that the resulting visualizations are accurate at many different viewpoints.
ii. Utilize efficient image-based rendering (IBR) techniques as a cognitive tool to move from one accurate view to another.
iii. Allow for extremely high-resolution imagery in the interactive IBR framework.
iv. Separate the costly storage of the pre-computed imagery from the resulting viewing platforms over high-speed networks.
v. Provide a general enough framework, such that many different rendering packages can be integrated into our framework.
vi. Finally, guarantee interactive frame-rates regardless of the data/scene complexity.

Our IBR viewer, unlike other per-pixel based models [10][11], can utilize 2D texture hardware to accelerate the rendering, allowing for interactive frame rates on relatively large displays (over 1Kx1K). Our viewer, which we term a rail-track viewer, allows the user to move inside the scene on user-selected paths and view the outside scenery in any viewing direction. It is an extension to QuickTime VR [1] or other panoramic based models [17] that incorporate depth information.

In our previous research [18], we discussed our overall system architecture. The rail-track viewer partitions the scene into layers and combines the neighboring viewpoints to reconstruct the viewpoint in between. As an extension, this paper concentrates on demonstrating how to divide the scene into layers by a simple but efficient algorithm. We also present in this paper a novel texture culling technique to eliminate unnecessary information introduced by using multiple layers.

Our contributions presented in this paper are:

(1) A texture streaming client/server architecture for panoramic walkthrough
(2) View-dependent layers to better address occlusion and dis-occlusion problems.
(3) A simple and efficient way to partition the scene into panoramic depth layers
(4) View-dependent IBR database compression, considering possible occlusion/dis-occlusion along the track segments.

The paper is organized as follows: First we discuss relevant background and previous work in the IBR area. We then present an overview which summaries our system and previous work. We also present our simple approach to partition the scene into layers. Next we discuss data management that includes an occlusion culling algorithm. Finally we present some test results and conclude with future work.

2. Related Work:

A lot of effort has been put into designing more accurate IBR systems. This is because IBR has a very important advantage over the traditional geometry-based rendering systems in that it has a bounded computational cost according to the input image resolution.

QuickTime VR [1] is probably the earliest effort of IBR research. It has the ability to allow the user to look around horizontally and vertically (QuickTime VR only allows 360 degrees in horizontal directions and spherical panoramic systems [17] allow for both horizontal and vertical directions). A QuickTime VR system is simple and very efficient because it only uses implicit geometry relationships to reconstruct the scene. However, it also restricts the user to sit at the pre-defined viewpoint. It projects everything to the same radius cylinder. Darsa et al [4] suggests a way to introduce depth information into the cubical panoramic system to allow for a walkthrough. They use three blending methods to reconstruct the views between two pre-defined viewpoints. Cohen-Or et al [2] introduces the idea of pre-defining a path and precomputing the geometry and textures for the sampled viewpoints on the path. They use the texture mapping hardware to reconstruct the novel views in between. Both of these systems do not address the occlusion and disocclusion problems as described in the Layered Depth Image paper [15]. That is, when the user moves away from the pre-selected viewpoints, some features that were previously occluded in the original viewpoint can become visible. Without multiple layers of depth [6][15], these systems require several viewpoints to fill in the holes. A dense sampling is needed for this purpose, which increases the database size and puts more burden on storage and network transmissions and loading time. By utilizing multiple layers and culling away unnecessary information, our system can achieve more efficiency in this sense.

Most of the previously introduced IBR systems concentrate on accurate renderings of relatively low-resolution imageries. These systems use per-pixel warping, as described in [10][11][12][15][17]. Hardware texture mapping is not utilized and therefore the performance is not very fast for larger image resolutions. They are not suitable for our purpose, which is interactive management and walkthroughs of large datasets and complex scenes on a high-resolution (over 1kx1k) display. Examples such as the LumiGraph [8] and Light-field Rendering systems [9] usually sample the viewing parameters very densely, requiring large storage spaces. There are some systems that utilize the texture hardware to accelerate the rendering performance, such as the previously mentioned Darsa et al [4] and Cohen-Or [2]'s work. The View Dependent Texture Mapping (VDTM) [5] is a typical example of using 2D texture hardware to accelerate the renderings. However, they do not sample the viewing direction adequately to allow for panoramic viewing. Our system [18] on the other hand, allows the user to move back and forth on a pre-defined track, with full rotational viewing directions.

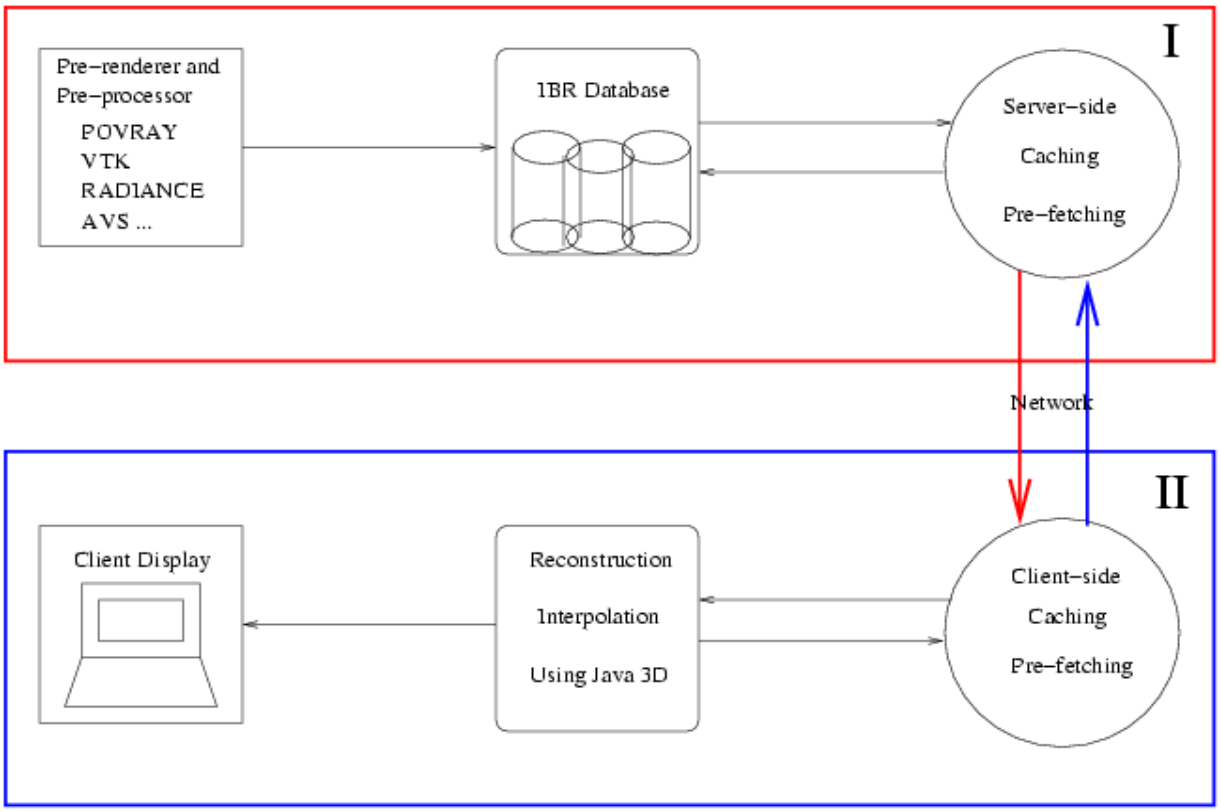

Figure 1: System diagram – a two-phase pipeline is used. The first step uses different rendering engines to pre-render the datasets. The resulting geometries and imageries are pre-processed to a more manageable format and stored on a disk or the server. Whenever the client needs the data, it sends the necessary information across the network to the server and the server retrieves the related data from the database and sends it down the network to the client. Both the server and the client maintain their own cache for fast data access.

3. Overview

In our previous research [18], we described a system to interactively manage complex, time-consuming renderings on a relatively large display. We allow the user to move inside the dataset on a pre-selected path and look around in both horizontal and vertical directions. The system consists of a two-phase pipeline as in Figure 1. The first step allows the user to select his/her own rendering engines to pre-render the datasets. A depth image along with the color image is acquired for each reference viewpoint. The resulting geometries and imageries are then pre-processed to a more manageable format and stored on the disk of the server.

In the second step, whenever the client needs the new information for a view, it sends the request across the network to the server and the server retrieves the related data from the database and passes it across the network to the client. Both the server and the client maintain their own cache for fast data access. The client can then reconstruct novel views when the data is ready.

The IBR model we use on our client side is an extension of the QuickTime VR type of viewers, which lets the user move on the pre-defined track. To achieve this, we incorporated the depth values into the IBR systems as proposed by Darsa et al [4]. In our previous work, based on visibility polyhedrons [13] we derived the theories of how to combine close-by reference views sampled on the track to reconstruct the information for the novel viewpoints. We then described a scheme to partition the scene into several depth ranges, which we call slabs, by setting the near and far clipping planes of our pre-render. A binary opacity mask is assigned for each pixel to enable per-pixel occlusion. The reason we adopted depth slabs was to address occlusion and dis-occlusion problems described in the previous work section.

This works similar to Layered Depth Images [15]. However, we obtained improved rendering performance, with View Dependent Texture Mapping and 2D texture hardware. A front to back or back to front compositing of the slabs gives us the complete description of one viewpoint. To move smoothly from one viewpoint to another, we use a non-linear interpolation scheme to combine the close-by reference viewpoints for reconstructing the novel views. By doing this, we guarantee that at the vantage viewpoints, our IBR renderer can achieve accurate renderings while the in-between views exhibit only small errors. Our previous paper [18] provides a detailed description of the slab representation and interpolation scheme.

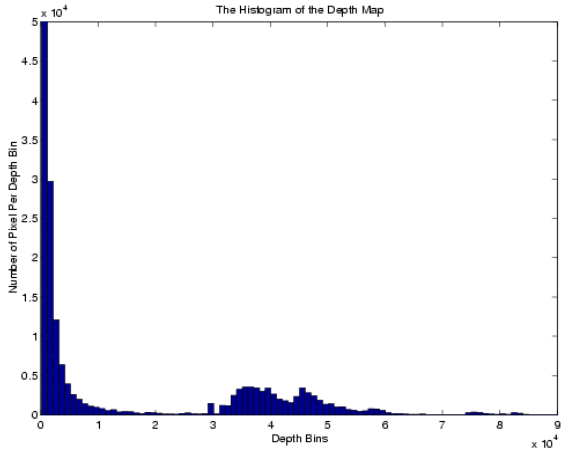

Figure 2: shows the histogram of the depth map for the Nature dataset

Decoret et al [6] finds separating planes by grouping the objects according to their distances. However, it is not trivial to group the objects in a very complicated scene. We divide the scene into the slabs by analyzing the histogram of the depth map of the entire scene. Figure 2 shows the histogram of the depth map for the Nature dataset [21]. By analyzing the figure, we can see that most of the depth values appear in the range of 0 – 10000 with the largest peak at about 0 – 1000. Another peak appears between 30000 – 60000. We therefore set the slab ranges to be 0 – 1000, 1000 – 10000 and 10000 – 100000 respectively. This is a crude algorithm for selecting these partitions, but works well in practice and is easily incorporated into the existing renderers.

Using slabs we can better address occlusion and dis-occlusion problems. However, it also saves information which are never visible on the local track segment. This will hurt the storage, loading and rendering performance. In this paper, a track dependent occlusion culling algorithm is utilized to remove this unnecessary data and improve the rendering performance. This will be addressed in detail in the next section.

4. Texture Removal Using Track Dependent Occlusion Culling:

The slab representation is used to better address the occlusion and dis-occlusion problems. As the user moves away from the reference viewpoint, previously occluded information can be rendered using later slabs. However, the problem with partitioning and pre-rendering scenes into several slabs is that it produces unnecessary information. Consider the example in Figure 3. In this example, we have three reference viewpoints on the track segment: V_1, V_2 and V_3. Objects O_2, O_3 and O_4 are occluded by Object O_1 for V_1 but are rendered and stored in $slab_2$. O_2 is visible from V_2 and O_4 is visible from V_3. Hence, when the user moves away from V_1 towards V_2, the information stored in slab2 of V_1 is used to represent O_2. Likewise for O_4. However in this example, O_3 is not visible from any viewpoint along the track segments. Without occlusion culling we would still render O_3 and store the result in $slab_2$ as part of the texture map, which is unnecessary. This unnecessary data affects the storage cost, network transmission time and the rendering performance. A conservative track dependent occlusion-culling scheme is thus developed to remove these occluded textures.

We call this algorithm track dependent occlusion-culling because we need to consider current and neighboring viewpoints for the algorithm. How much information is occluded depends on the sampling rate of the reference views on the pre-selected track. Durand et al [7] introduced an algorithm that combines multiple viewpoints into one cell. Occlusions of the objects are calculated for the whole cell. They introduced the extended projection operators. The extended projection for the occluder is the intersection of the views within the cell, while the extended projection for the occludee is the union of the views within the cell. To calculate the occlusion, we need to compare the extended projections of the occluders and occludees. As pointed out in our previous paper [18], the depth slabs are down-sampled to quad-meshes and the textures are partitioned into tiles accordingly. We therefore perform our occlusion culling algorithm for all the tiles in all slabs. We want to determine whether potential tiles in the later slabs are occluded by those in the previous slabs for both the current and the neighboring two viewpoints. Therefore in our algorithm, we consider current and neighboring viewpoints as a cell and texture tiles of the early slabs to be occluders and texture tiles of the later slabs to be occludees.

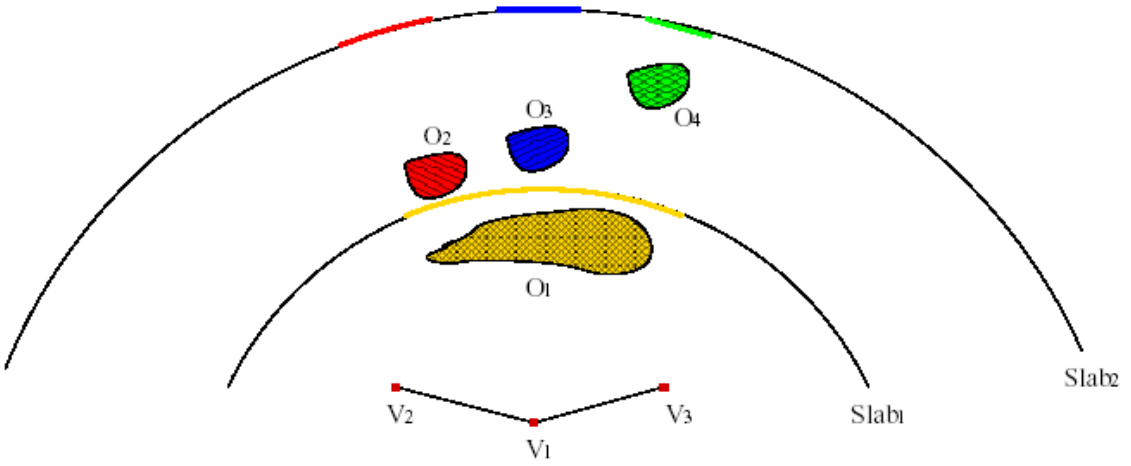

Figure 3. Objects O_2, O_3 and O_4 are occluded by Object O_1 for V_1 and therefore are rendered and stored in the $slab_2$. O_2 is visible for V_2 and O_4 is visible for V_3. However O_3 is not visible from any of the viewpoints along these track segments.

The algorithm works as follows. For each reference viewpoint, we first build an occlusion map and fill the map with the opacity values of the projected first slab texture tiles. We treat the slab textures in a front to back order, considering each tile in each slab to see whether it is occluded. The occlusion is performed by comparing the extended projections of the occluders: texture tiles from the previous slab and the extended projections of the occludees: texture tiles from the later slab. Figure 4 (a) and (b) show the extended projections of occluder tiles and occludee tiles, with regard to the current viewpoint V_1 and its neighboring viewpoints V_2 and V_3. If the extended projection of the occludee falls within that of the occluders, the occludee is not visible. In practice, we use the opacity map of the first slab as our starting occlusion map and then convolve (average) the window in the occlusion map with the size of extended projection of the occludee. If the average is 1, all pixels in this range are fully opaque, and the occludee tile is occluded.

For easier calculation, we make several conservative simplifications. According to [7], for non-flat tiles, the depth of the occluder is the maximum depth of the tile, while the depth of the occludee is the minimum depth of the tile. For all the occluder tiles, we chose the slab depth which is larger than any maximum tile depth as another conservative simplification. By taking the minimum depth of the occludee tile and the slab depth, we can consider them as flat tiles and therefore we have a setup as in Figure 4 (c).

Each tile in our system has a width of w. Considering the projections of V_2 and V_3, we now need to convolve (average) an extended area with a width of $w + s_1 + s_2$ in the opacity map to see if the result equals 1. Considering the 2D case, s_1 can be calculated using the following equation.

$$s_1 = \frac{h_2 - h_1}{h_2} \times d_{21} \tag{1}$$

In which h_1, h_2 are the distance from the viewpoint to $slab_1$ and $slab_2$ respectively. To calculate d_{21}, consider Figure 4 (d).

$$|d_{21}| = |l| - |l'| \tag{2}$$

While

$$|l| = (\vec{Q} - \vec{V}_1) \bullet \vec{x} \tag{3}$$

and

$$|l'| = \frac{|h|}{\tan\beta} \tag{4}$$

Here

$$\cos\beta = \frac{(\vec{Q} - \vec{V}_2) \bullet \vec{x}}{|\vec{Q} - \vec{V}_2|} \tag{5}$$

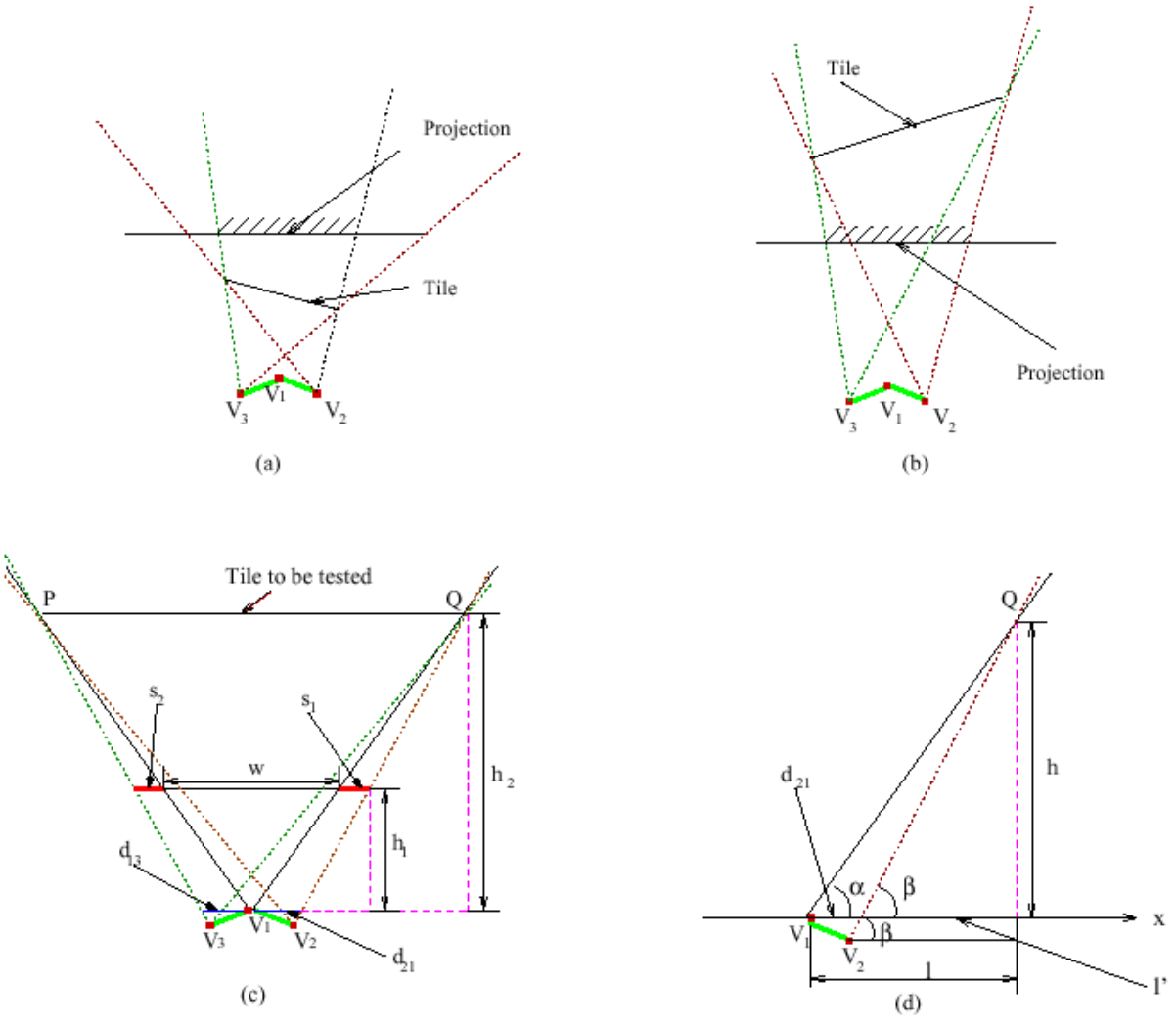

Figure 4: (a) and (b) the extended projections for occluder and occludee tile respectively, with regard to three viewpoints on the track. (c) The occlusion culling setup after conservative simplification of depth. h_1 is the distance from the viewpoint to $slab_1$ and h_2 is the largest depth of the occludee tile. (d) Geometry used to calculate d_{21}

A similar equation can be used to calculate s_2. If the averaged opacity value of the enlarged window is one, we mark the tile as an empty tile and do not store the geometry and color information. If the value is less than one, the tile is not occluded and we update the occlusion map. We treat all the tiles in one slab and continue to the next one until all the slabs are processed. As pointed out in [3][12][19], we can use a method similar to α-acceleration, which lowers the opacity threshold to less than one to cull the tiles more efficiently without degrading the quality of the rendering results too much.

		$Slab_1$			$Slab_2$			$Slab_3$		
		Before	After	Reduction	Before	After	Reduction	Before	After	Reduction
α=1.0	Tiles	3228	3228	0	1556	352	77.4%	708	197	72.2%
	Size (MB)	13.38	13.38	0	6.69	1.544	77%	3.09	0.92	70.3%
α=0.9	Tiles	3228	3228	0	1556	307	80.7%	708	161	77.3%
	Size (MB)	13.38	13.38	0	6.69	1.34	80.0%	3.09	0.75	75.6%
α=0.8	Tiles	3228	3228	0	1556	283	81.9%	708	143	89.8%
	Size (MB)	13.38	13.38	0	6.69	1.23	81.7%	3.09	0.647	89%

Table1. Reduction Rates for Track Dependent Occlusion Culling using different α values for Castle Dataset

		$Slab_1$			$Slab_2$			$Slab_3$		
		Before	After	Reduction	Before	After	Reduction	Before	After	Reduction
α=1.0	Tiles	1864	1864	0	408	398	2.5%	249	249	0
	Size (MB)	7.717	7.717	0	2.057	2.01	2.3%	1.03	1.03	0
α=0.9	Tiles	1864	1864	0	408	382	6.4%	249	246	1.2%
	Size (MB)	7.717	7.717	0	2.057	1.93	6.2%	1.03	1.02	1%
α=0.8	Tiles	1864	1864	0	408	367	10%	249	242	2.9%
	Size (MB)	7.717	7.717	0	2.057	1.87	9.1%	1.03	1.01	2%

Table 2. Reduction Rates for Track Dependent Occlusion Culling using different α values for Nature Dataset

We tested our occlusion culling algorithm on the castle [22] dataset. The scene is partitioned into three slabs. After culling, we can reduce the information in the second slab by 77% from 6.69 MB to 1.54 MB. The storage requirement for the third slab is reduced by 72% from 3.09 MB to 0.92 MB. This is without the α-acceleration. With α set to 0.9, the reduction rates are 80% and 77% for the second and the third slab. The reduction rates reach 82% and almost 90% when the α value is set to 0.8. This is shown in Table 1. The rendering results using different α values are shown in Figure 5. The red circles indicate artifacts caused by decreasing the α threshold. From the figure we can see that the rendering quality doesn't degrade substantially if we set a reasonable α value. The results show that the track dependent occlusion culling is quite efficient for this dataset. It can reduce the storage requirement and decrease the network transmission time. The pre-fetching efficiency and rendering performance can also be improved because with a fixed amount of cache memory and the occlusion technique, we can cache more viewpoints along the track.

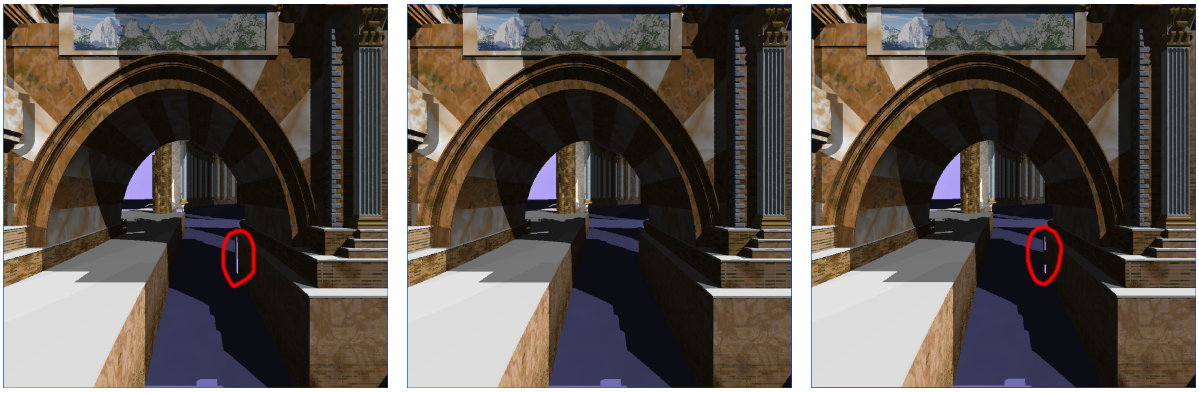

Figure 5. Shows the rendering images with track dependent occlusion culling with α values equal to 1, 0.9 and 0.8 respectively. Notice the artifacts indicated by the red circle when decreasing the α threshold.

Another benefit is that it can reduce the rendering cost/overhead that is caused by increasing the number of slabs. More slabs can address the occlusion/dis-occlusion problem better. Using the occlusion culling technique, less information will be left for later slabs after culling. Therefore, increasing the number of slabs does not affect the rendering speed too much.

The efficiency of the algorithm is highly dataset-dependent. For the Nature dataset [21] we tested, the scene is more open and therefore our slab representation does not have much occluded information in the first place. We can only cull about 2 percent without α-acceleration and 10% with α-acceleration. This is shown in Table 2. Even though for some cases not too much information is occluded, we should note one thing for our system. The occlusion culling is performed as a pre-processing step and therefore there will not be any performance penalty during the run-time.

From the results, we can see that our track-dependent occlusion culling can help us efficiently cull away unnecessary information and therefore save storage space and network resources as well as improve performance. The rendering quality is not degraded due to this.

5. Results:

We implemented the image-based framework as a proof of concept to manage complex renderings at high image resolutions and interactive frame rates using only off-the-shelf texture mapping hardware. Our system is implemented in Java / Java3D. Almost all renderers can be configured to pre-compute the image layers. We use POVRAY in the results shown here.

We tested our system on three POVRAY datasets. One is the "Nature" dataset from the POVRAY quarterly-competition website [21]. The scene contains trees, bushes, rocks and birds and is quite complicated. It takes about 10-20 minutes to render one 1kx1k frame using POVRAY on our Pentium 4 2GHZ, 2GB Dell Precision 530 workstation. One path was chosen for this scene with 20 sampled viewpoints along the track. Three panoramic layers with a resolution of 1024x4096 per layer were pre-computed for each view sample. The total size for the image database after pre-processing was only 220MB. Without empty tile removal and track-dependent occlusion culling it would require over 1.2GB. Another dataset is the "Castle" dataset from the same source [22]. This dataset is even more complicated and takes several hours to generate one 1024x1024 frame. We chose a path with 10 reference viewpoints. Three panoramic layers with a resolution of 1024x4096 per layer were pre-computed for each view sample. We also tested a Night [22] dataset which was obtained from the same source on the web.

We tested our IBR walkthrough system on two workstations. On our Sun Blade 1000 with dual UltraSparc III 750 MHZ, 1GB of memory and an Elite 3D graphics card with 64MB of video memory, we achieve a frame rate of 10-15 frames per second for the three datasets. With our Dell Precision 530 workstation with Pentium 4 2GHZ, 2GB of memory and Nvidia Geforce 4 Titanium 4600 graphics card with 128MB of video memory, we achieve a frame rate of 20-30 frames per second for all datasets. Some test results are shown in Figures 6.

6. Conclusions and Future Work:

In this paper we presented extensions to our previous paper [18]. Compared to our previous work, we use the occlusion culling algorithm to make the system more efficient and present a way to better partition the scene into layers.

With the existing data management techniques, our database still tends to be very large, especially with larger resolution and finer sampling. However, this is not something that we did not expect. We trade storage for rendering efficiency and low-cost texture streaming. This allows us to render extremely large resolution and offers an advantage over other high-quality IBR techniques.

Again our contributions include a texture streaming client/server architecture for panoramic walkthroughs and using view-dependent layers to better address occlusion and dis-occlusion problems. A simple, however efficient, partitioning of the scene into layers is used. We also use a track-dependent IBR database compression scheme considering possible occlusion culling along the track segments.

We need to point out here that how we sample the predefined track was not addressed in this paper in detail. The sampling rate can be either user defined, chosen by the database designer or at fixed intervals. It can also be controlled by the maximum mean squared errors allowed for the reconstructed scenes along the track, which will be one of our areas of future work.

Future work also requires us to look at appropriate tile compression techniques to further reduce the data storage and network transmission overheads.

7. Acknowledgements:

We would like to thank the Department of Energy ASCI Program for generous support for this project. Additional support was provided through an NSF Career Award (#9876022). Equipment was provided by the ASCI project and by NSF grants (#9818319) and (#9986052).

References:

[1] S. E. Chen, "QuickTime VR – An Image-Based Approach to Virtual Environment Navigation," *Proc. SIGGRAPH '95*, pp. 29-38, 1995

[2] D. Cohen-Or, Y. Mann and S. Fleishman, "Deep Compression for Streaming Texture Intensive Animations," *Proc SIGGRAPH '99*, pp. 261-268

[3] J. Danskin and P. Hanrahan, "Fast Algorithms for Volume Raytracing," *Proc. 1992 Workshop Volume Visualization*, pp. 91-98, 1992

[4] L. Darsa, B. Costa, and A. Varshney, "Navigating static environments using image-space simplification and morphing," *1997 Symposium on Interactive 3D Graphics*, pp. 25-34, 1997

[5] P. Debevee, Y. Yu and G. Borshukov, "Efficient View-Dependent Image-Based Rendering with Projective Texture Mapping," *In 9th Eurographics Rendering Workshop*, Vienna, Austria, June 1998

[6] X. Decoret, G. Schaufler, F. Sillion, J. Dorsey, "Multi-layered imposters for accelerated rendering," *Proc. Eurographics '99*, pp. 145-156, 1999

[7] Frédo Durand, George Drettakis, Joëlle Thollot, Claude Puech, "Conservative visibility preprocessing using extended projections", *Proc Siggraph '00*, pp 239 - 248

[8] S. Gortler, R. Grzeszczuk, R. Szeliski, and M. Cohen, "The Lumigraph," *Proc SIGGRAPH '96*, pp. 43-54, 1996

[9] M. Levoy and P. Hanrahan, "Light Field Rendering," *Proc. SIGGRAPH '96,* 1996.

[10] W. Mark, L. McMillan, and G. Bishop, "Post-Rendering 3D Warping," *1997 Symposium on Interactive 3D Graphics*, pp. 7-16, 1997

[11] L. McMillan and G. Bishop, "Plenoptic Modeling: An Image-Based Rendering System," *Proc. SIGGRAPH '95*, pp. 39-46, 1995

[12] K. Mueller, N. Shareef, J. Huang, and R. Crawfis, "High-quality splatting on rectilinear grids with efficient culling of occluded voxels," *IEEE Transactions on Visualization and Computer Graphics*, vol. 5, no. 2, pp. 116-134, 1999.

[13] F. P. Preparata, M. I. Shamos, "Computational Geometry, An Introduction," Chapter 1, Springer-Verlag New York Inc, 1985

[14] P. Rademacher and G. Bishop, "Multiple-Center-of-Projection Images," *Proc. SIGGRAPH '98*, pp. 199-206, 1998

[15] J. Shade, S. Gortler, Li-Wei He, and R. Szeliski, "Layered Depth Images," *Proc. SIGGRAPH '98,* pp 231-242, 1998

[16] Heung-Yeung Shum, Li-Wei He, "Rendering With Concentric Mosaics," *Proc. SIGGRAPH '99*, pp. 299-306, 1999

[17] R. Szeliski and H. Y. Shum, "Creating Full View Panoramic image Mosaics and Texture-Mapped Models," *Proc. SIGGRAPH '97*, pp 251-258, 1997

[18] L. Yang and R. Crawfis, "Rail-Track Viewer, an Image Based Virtual Walkthrough System", *in 2002 Eurographics Workshop on Virtual Environment*, pp. 37-46, Barcelona, Spain, May, 2002

[19] H. Zhang, D. Manocha, T. Hudson, K. E. Hoff, "Visibility culling using hierarchical occlusion maps", *Proc. SIGGRAPH '97*, pp 77-88, 1997

[20] http://www.povray.org

[21] http://www.irtc.org/stills/1998-06-30.html

[22] http://www.irtc.org/stills/1999-02-28.html

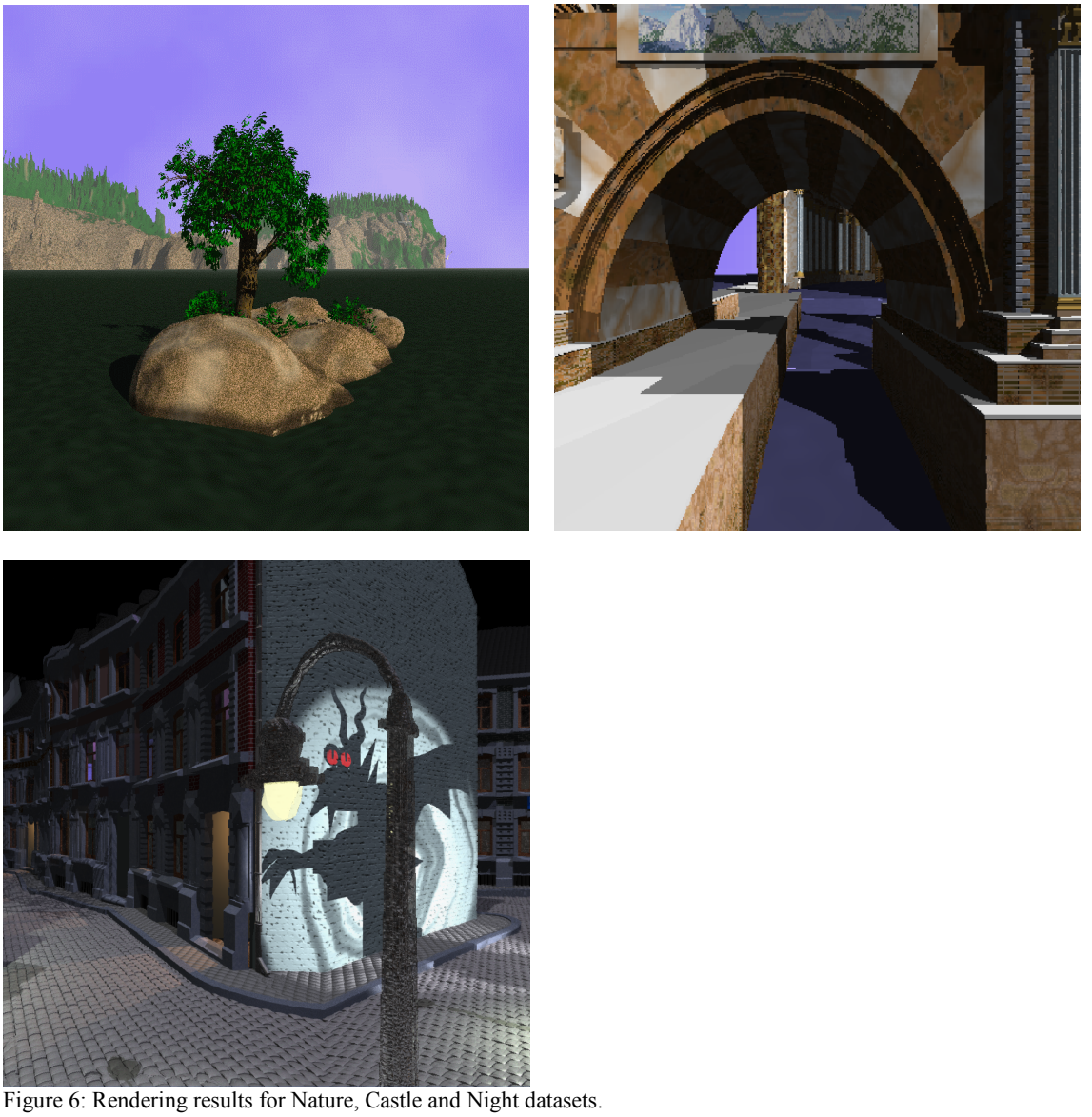

Figure 6: Rendering results for Nature, Castle and Night datasets.