Eighth Eurographics Workshop on Virtual Environments (2002)
S. Müller, W. Stürzlinger (Editors)

Rail-Track Viewer – An Image-Based Virtual Walkthrough System

Lining Yang, Roger Crawfis
Department of Computer and Information Science
The Ohio State University

Abstract
Complex renderings of synthetic scenes or virtual environments, once deemed impossible for consumer rendering, are becoming available as tools for young artists. These renderings, due to their high-quality image synthesis, can take minutes to hours to render. Nowadays, as the computing power has increased dramatically, the size and complexity of the datasets generated by the super-computer can be overwhelming. It is almost impossible for the visualization techniques to achieve interactive frame rates. Our work focuses on using Image-Based Rendering (IBR) techniques to manage and explore large and complex datasets and virtual scenes on a remote display across the world-wide-web. The key idea for this research is to pre-process the datasets and render key viewpoints on pre-selected paths inside the dataset. We present new techniques to reconstruct approximations to any view along the path, which allows the user to roam around inside the datasets with interactive frame rates. We have implemented the pipeline for generating the sampled key viewpoints and reconstructing panoramic-based IBR models. Our implementation includes an efficient two-phase caching and pre-fetching scheme. The system has been successfully tested on several datasets and satisfying results have been obtained. Analysis of errors is also presented.

Keywords:
Image-based Rendering, Virtual Walkthrough, Virtual Environment, Visibility, Caching

1. Introduction

Complex renderings of synthetic scenes or virtual environment, due to their high-quality image synthesis, can take minutes to hours to render. Ray-tracing or global illumination using a tool such as POVRAY [20] that can render high-quality images however is very time consuming. Nowadays, as the computing power increases dramatically, the size and complexity of the datasets generated can be overwhelming. Building and rendering the geometry of these large datasets are also time consuming. An interactive virtual walkthrough of these large and complex scenes is almost impossible on a low to mid-end system using traditional rendering techniques.

Our goal is to determine a solution for allowing the user to examine and walkthrough the scene from an internal vantage point on a relatively high-resolution display. To achieve this goal, we decided to apply Image-Based Rendering (IBR) techniques as a post-processing tool for any traditional renderer.

{yangl, crawfis}@cis.ohio-state.edu
395 Dreese Lab, 2015 Neil Ave., Columbus, OH 43210, USA

IBR is a new research area in both the computer graphics and visualization community, and offers advantages over the traditional rendering techniques. It does not require building or modeling complicated geometric models. It can utilize real life images and illumination for photo-realistic rendering. Finally, IBR requires a fixed or limited amount of work, regardless of the view or data context. However, this amount of work is fixed to the desired output. As a result, many IBR techniques [9] [10] [11] [12] focus on accurate rendering of relatively low-resolution imageries. Here we explore techniques for large displays having a resolution from 1kx1k to 8Kx3K as in our new video wall.

Our work can be viewed as an extension to QuickTime VR [3] or other panoramic representations [19]. Panoramic imagery allows one to spin around at their current viewing position, but does not allow the user to move forward or backward. We developed a system to allow movement along a linear path in three-dimensions. At any position on this curve, the user can interact with the scene as in a panoramic viewer. We termed this type of viewing, a rail-track view, in that the user can move forward and backward along the "track", while viewing the outside scenery in any direction. Darsa, et al [6] investigated techniques to incorporate information about the depth of the image into the panoramic scene. Depth allows for proper re-projection of the pixels from different viewpoints and provides a sense of motion parallax to give a true three-dimensional sense to the imagery. For efficient rendering, we apply a mesh simplification method to simplify the depth image from every pixel to a more manageable geometric quad-mesh. An intelligent caching and pre-fetching scheme is employed to further improve the rendering speed. We also present and analyze the sources of the errors in our system.

The paper is organized as follows: First we discuss relevant background and previous work in the IBR area. We then present an overview followed by implementation details of our system. Next we discuss the pre-processing and data organization scheme for efficient rendering. A two-phase caching and pre-fetching technique is also presented in this section. We will then discuss the sources and a quantitative measurement of the errors. Finally we give some test results and conclude with future work.

2. Related Work

Recently, a lot of effort has been put into Image-Based Rendering systems. Image-based rendering has the property of a bounded computation according to the input and output image size. This is advantageous over traditional polygonal based rendering where complex scenes or models can have individual polygons smaller than a pixel and therefore constructing and rendering these can be unbounded complicated.

2.1 Plenoptic Function

IBR systems are based on the Plenoptic function. The 7D plenoptic [1] function is defined as the intensity of light rays passing through the camera center at every location defined by V_x, V_y, V_z, at every possible viewing angle θ and ϕ, for every wavelength λ and at any time *t*.

$$\mu = Plenoptic(\theta, \phi, \lambda, V_x, V_y, V_z, t) \qquad (1)$$

Even with faster CPU's and more memory, this function overwhelms modern architecture, making it impractical for interactive applications. In order to make practical use of Image based rendering concepts, this model has to be simplified. McMillan [12] et al simplified the model to a 5D plenoptic function (equation 2) by fixing time and breaking up the wavelength as RGB components.

$$\mu_{\lambda,t} = \mu(\theta, \phi, \vec{V}) \qquad (2)$$

Note that $\vec{V}$ is a vector representation of V_x, V_y, V_z. McMillan's image warping system is based on this 5D plenoptic function. If the user movement is restricted to lie outside of a box, the model can be further simplified to a 4D function such as Lumigraph [9] or Lightfield [10]. A 3D representation of the plenoptic function is constructed in Concentric Mosaics [18] as they restrict the user's movement to lie within a 2D circle. QuickTime VR systems [3] [19] reduce the function to a 2D one by letting the user stand at a fixed point and look around. This research focuses on allowing the user to move on a pre-selected path and look around. Hence it is modeling a 3D slice through the plenoptic function.

Having a continuous plenoptic function, we need to sample and discretize it. Equation 3 is a discretized version of the 5D plenoptic function.

$$\mu_i = \mu_{\lambda,t}(\theta_i, \phi_i, \vec{V}_i) \qquad (3)$$

Obviously we can sample the viewing direction and views. IBR systems can vary in terms of how and when to choose view samples. Some systems choose to sample the viewpoints outside of the scene [4] [7] while others put their sample viewpoints inside the scene, on a path [12] [16], inside a circle [17], or at a fixed point [3]. Systems may choose the reference views a priori, pre-rendering the sampled plenoptic function. While other systems [16] choose view samples on the fly and reconstruct images of new views from these until the image quality degrades to where new reference views are needed.

2.2 Depth Function

For opaque scenes, the depth function ξ is of value for reconstructing the novel views (i.e. views not located at the sample points). Equation 4 describes the distance to the closest object in the scene. Sampling and discretizing this function gives us ξ_i. Equation 3 and 5 provide a framework for IBR representation of a scene.

$$\xi = \xi(\theta, \phi, \vec{V}) \qquad (4)$$

$$\xi_i = \xi(\theta_i, \phi_i, \vec{V}_i) \qquad (5)$$

IBR systems can also differ in how they represent this depth function (geometric information). QuickTime VR [3] and spherical panoramic systems [19] do not store any geometric information. The images for new view-points are reconstructed using implicit geometric relationships. This is adequate for scenes far away from the user at where the user is only allowed to look around or zoom.

2.3 Occlusion and Dis-occlusion

3D warping systems [6] [11] [12] store depth information per pixel. Users can move away from the pre-selected viewpoints and the depth and color information is used to construct novel viewpoints. The occlusion and dis-occlusion problems are not addressed by the 3D warping system, because it only has one layer of the depth information associated with each pixel. As the user moves farther away, holes will appear and information for more than one reference viewpoint is needed to fill these holes. Layered Depth Images (LDI) [17] [2] store several layers of depth and color information for each pixel. When rendering from an LDI, the user can move farther away and expose surfaces that were not visible in the first layer. The previously occluded information can be rendered using information from later layers. The model is inefficient when we have complicated scenes or when the output image has extremely high resolutions. Other systems partition the object space into several slabs along the viewing direction [8] [13]. The data within a slab is projected towards the image plane and then texture mapped onto a quad oriented parallel with the image plane and perturbed by the z-values of its corresponding slab. When reconstructed for a novel view, this can approximate an image warp.

Most of the IBR systems concentrate on accurate rendering of relative low-res imageries. Systems based on 3D warping [2][11][12][17] use per-pixel operation and do not utilize much hardware acceleration. They are not suitable for interactive walkthroughs on very high-resolution displays. Lumigraph [9] and Light-field Renderings [10] need very dense samplings, which is both time consuming and space inefficient. View dependent texture mapping (VDTM) [7] is a typical example utilizing texture hardware to accelerate rendering. However, for one reference viewpoint, the complete viewing direction is not adequately sampled and therefore can not allow head rotation during the fly-through. Cohen-Or et al [5] also looked into ways to pre-compute the geometry and textures on a path and stream the results across the network. They use projective texture mapping for close-by views. Again, their work lacks the ability to allow the user to change the viewing direction during the walkthrough because of the incomplete sampling. Their method focuses on how to compress the resulting textures and efficiently stream them down the network. The closest implementation to ours is Darsa et al's [6], in which they used cubical environmental maps with simplified triangular meshes. Three blending methods were explored for smooth walkthroughs between close-by viewpoints. Our system differs from theirs in that we used cylindrical models instead of cubical ones. Cylindrical models have the advantages of representing texture information homogeneously without discontinuities at the edges like the cubical maps. We have also adopted a multi-layer approach comprised of several depth meshes to better solve the occlusion and dis-occlusion problems. We have also investigated an intelligent two-phase caching and pre-fetching scheme to improve the performance of our system. Darsa's system reports a frame rate of 3.53 FPS on an Infinite Reality Engine using position weighted blending with a 256x256 window, while our system can achieve around 15 FPS using the same blending method with 1kx1k resolution on a low-cost Sun Blade 1000 workstation with an Elite 3D graphics board.

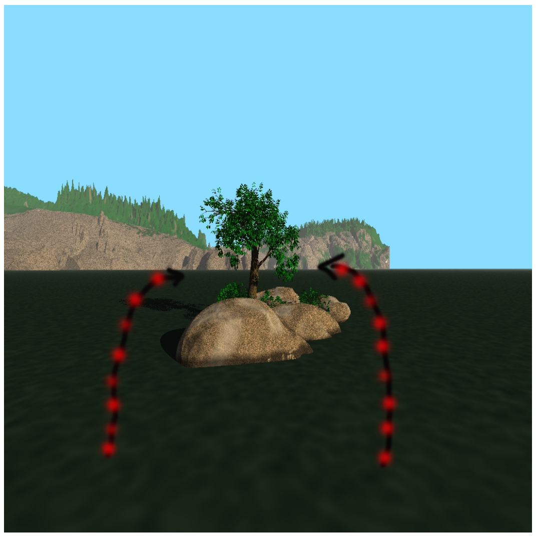
Figure 1: two possible tracks into the Nature dataset we rendered using Povray. The black arrowed curves represent the tracks and red dots represent the pre-selected viewpoints.

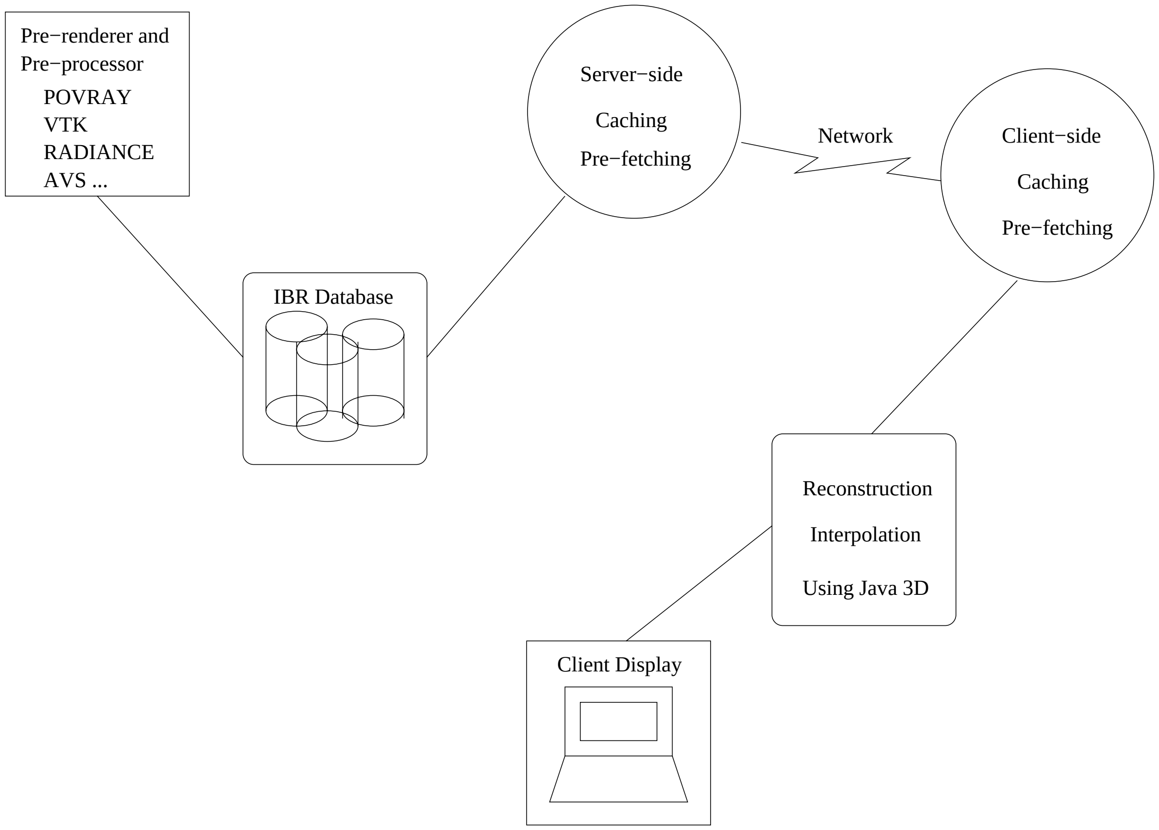

Figure 2. shows the system diagram. First datasets are pre-rendered and pre-processed using appropriate rendering engines, for example, POVRAY, VTK, Radiance and so on and stored on a server-side database. The server keeps a cache and retrieves from the database and stores into the cache according to the client demand then transmit the data across the network to the client. The client keeps its own cache and can reconstruct the results using our layered-panoramic IBR model.

3. Overview

The goal of this research is to interactively manage complex, time-consuming renderings of large datasets. We concentrated our efforts to allow users to roam interactively inside a scene, exploring interesting features with the ability to look around at the same time. We achieve this by restricting the users' movement on a pre-selected path and allow them to look around at any point in both the vertical and horizontal directions. Figure 1 depicts two possible tracks into the so-called Nature dataset we rendered using Povray [21]. The black arrowed curves represent the tracks and red dots represent the pre-selected viewpoints. The user is allowed to move back and forth on this track and change his viewing directions freely. Although some software packages allow discrete jumps from one viewpoint to another, this disturbing teleportation removes the user's focus from smooth walkthrough to one of trying to figure out where they are and regain their orientation or bearing.

Figure 2 describes the idea of our IBR system. First the datasets are pre-rendered and pre-processed using appropriate rendering engines, for example, POVRAY, VTK, Radiance and so on and the resulting imageries and geometry information are stored on a server-side database. The server keeps a cache and retrieves the reference views from the database and stores them into the cache whenever the client requests. The cached data is then transmitted across the network to the client. The client receives the information and stores it into its own cache. Novel views can now be reconstructed using our layered-panoramic IBR model, which will be described later.

4. Visibility Polyhedrons

Moving from one viewpoint in a complex scene to another with the freedom of looking around is a challenging problem. Let's assume that we want to move from V_1 to V_2 in a complex scene. Consider the following definitions.

Definition 1: The **visibility polyhedron** of a viewpoint is the locus of points visible from the viewpoint.

Definition 2: A polyhedron *P* is said to be **star-shaped** if there exists a point z not external to P such that for all points *p* of *P* the line segment $\vec{zp}$ lies entirely within *P*.

Definition 3: The locus of the points *z's* having the above property is called the **kernel** of P.

Theorem 1: The visibility polyhedron of the viewpoint is a star-shaped polyhedron and has a kernel that at least contains the viewpoint**.**

These definitions are extensions from those on polygons [14]. Let's assume that V_1 and V_2 lie inside each other's kernels. The visibility polyhedrons of the two viewpoints are therefore identical. Any points V_3 in between the line segment connecting the two points are also inside that kernel due to definition 2. Therefore, the visibility polyhedron of V_3 is the same as V_1 or V_2's. We can use the combination of these two to represent V_3's as described in equation 6.

$$P_{V_3} = P_{V_1} \bigcup P_{V_2} = P_{V_1} = P_{V_2} \qquad (6)$$

The assumption of two close-by viewpoints lying inside each other's kernels does not always hold. We can approximate V_3's visibility polyhedron from V_1 and V_2's visibility polyhedrons. If V_1 and V_2 are close enough this approximation is quite reasonable. 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 the future works.

We approximate the visibility polyhedron for each viewpoint to make the system more efficient, and texture-map it with the initial rendering. We then use equation (6) to combine the close-by reference views as the user walks between them. In the Next section we will address details of how we implemented our system.

5. Implementation Details

In practice, we use a pre-renderer to generate the imageries and the associated depth information. The depth values are obtained either from the z-buffer or the first intersection points with the viewing rays. In either case, we connect the depth values to form a polyhedron *ZP,* which is a discretized representation of the visibility polyhedron *P*. We need to simplify *ZP* since rendering each pixel with its own depth value is too time-consuming for high-resolution imageries. Therefore, we reduce the geometric complexity by down-sampling the depth buffer into quad-meshes. We call this simplified mesh *QZP*. The reason we chose to use a quad-mesh as our simplified geometry is because of its simplicity and resulting efficiency issues for our caching and pre-fetching scheme. The vertices of each tile of the quad-mesh have their actual depth values. Color and opacity are represented by texture maps. We assume a bilinear function for reconstructing the depth values of the interior points of the tiles, which is not always valid. Therefore, this mesh simplification scheme introduces errors, which will be talked about in more details in a later section.

In order to allow for occluded information at a view to be retained [17], we divide the viewing frustum into several depth ranges, by setting the near and far clipping planes. We call each range a slab QZP_i. A binary opacity mask is assigned to enable per-pixel occlusion. By compositing the slabs, we achieve the complete scene for one viewpoint.

$$discretize(P) = ZP \approx QZP = QZP_1 + QZP_2 + QZP_3 \qquad (7)$$

This will allow the user to move away from the pre-selected viewpoints while revealing the previously occluded information using additional slabs.

Figure 3 Shows depth meshes for one viewpoint. Figure 3(a) is the mesh viewed from the original viewpoint that provides a seamless view. Figure 3(b) shows a side view of the mesh for illustration purpose. Note the individual slab meshes that comprise the visibility polyhedron for this viewpoint.

We now have the simplified visibility polyhedrons, constructed using the slab meshes for both V_1 and V_2 with color as texture maps. How do we smoothly move from one point to another? To walk between two viewpoints we simply combine the two slab-mesh sets. To make the transition smooth, we interpolate the slab image sets of two nearby reference views. The correct way to do the interpolation is to use an over operation on corresponding slabs of the two views and then compute the result of $slab_1$ over the result of $slab_2$ and so on. This can be implemented using a special p-buffer, which is normally not available for consumer-level graphics board. Instead, we compute $slab_1$ for V_1 over slab1 for V_2 and then compute this over $slab_2$ of V_1 and so on. The interpolation requires a weight *w* based on the difference (normalized to be between 0 and 1) between the new view and one of the reference views. The corresponding reference image sets have their opacities modulated by the corresponding weight factor, and then projected in an interleaved fashion in depth sort-order from the new viewpoint. The equation for the resulting color after slab1 image sets are weighted is:

$$C = C_1 * \alpha_1 * w + (1-\alpha_1 * w) * C_2 * (1-w) * \alpha_2 \tag{8}$$

Here C is the final computed color at the new viewpoint, C_1 and C_2 are colors of slab images from the two neighboring reference views, and α_1 and α_2 are alpha values from opacity maps of the slab images. Results show that when using linear interpolation the computed image looks very close to a correct rendering when the new viewpoint is close to a reference view. However, when the new view is located at the middle between reference views, the computed image is darker and transparent looking. The reason for this is due to the fact that when the new viewpoint is near the middle of the two references, C is less than the original C_1 or C_2 colors, and thus appears more transparent. For example, assuming C_1 equals C_2 and both pixels have α's of 1, and our weight w equals 0.5. We have a resulting

$$C = 0.5 * C_1 + 0.5 * C_2 * 0.5 = 0.75 * C_1 \tag{9}$$

An alternative interpolation is to use a nonlinear weighting scheme. We define a non-linear blending equation as:

$$\begin{aligned} C = C_1 * \alpha_1 * w + (1-\alpha_1 * w) * C_2 * \\ (1 - pow(w,b)) * \alpha_2) \end{aligned} \tag{10}$$

Here b is a constant. Let's assume the same example as before with b equaling 2. We have a resulting color of

$$C = 0.5 * C_1 + 0.5 * C_2 * 0.75 = 0.875 * C_1 \tag{11}$$

By using this equation, the dark and transparent problem is alleviated. In practice we choose a value of 3 for b

6. Preprocessing

We could use the whole rendered image as a texture map to map to the simplified depth mesh. However, there are areas of the image that contain no information. Storing and rendering those is a waste of resource. Therefore, we break the resulting image into tiles that are corresponding to the depth mesh. The size of the image tiles and the quad-mesh affects the image loading time, storage, performance and the errors. These will be addressed in a later section. Those image tiles containing no information are detected and labeled as empty tiles. Thus we treat each image tile as a texture map for the corresponding quad. The problem with rendering each image tile as a texture map is that the system has limits on texture binding units. Texture-map thrashing results when too many small image tiles are produced, and interactivity is lost.

To alleviate this problem we group individual image tiles into bigger texture units for rendering. Consider a full image of size W by H, we divide the image into equal sized small tiles $w \ x \ h$ so that each row has W / w tiles and each column has H / h tiles. To remove empty tiles and merge the remaining into a larger texture map, we squeeze each column, removing the empty tiles, and linking the resulting columns of tiles into a one-dimensional tile array and use this as the texture unit. A simple indexing scheme is implemented to quickly calculate the position of the columns needed for our caching and pre-fetching engine.

The IBR rendering engine determines the closest two reference viewpoints on the path. At any given time, only the information of at most two reference viewpoints are needed to reconstruct novel views. Therefore, it is reasonable to have only two reference views in memory at a given time. When the users move to a new path segment, requiring a new reference view, they will encounter severe latency while the needed data for reconstruction is read into memory. Instead of loading just two sampled views, the system loads several views into memory and stores them into a cache. The pre-fetching engine, which is implemented as a separate thread, continually fetches texture maps as the user moves along the track and stores them into the cache. The cache is organized using a Least Recently Used (LRU) rule. The maximum number of views allowed in cache is determined by memory size, disc latency and the view-sampling rate. Our first experiment treats the whole panorama as a caching unit. It alleviated, but did not eliminated, the latency problem. When the user moved quickly along the track, noticeable latency still occurred since the disk reading could not keep up with the demand.

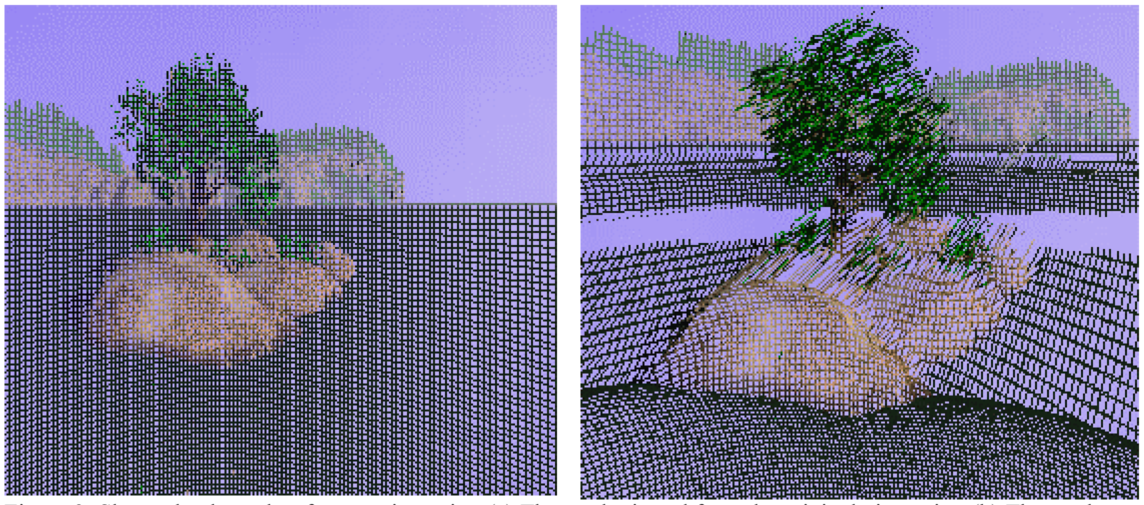
Figure 3: Shows depth meshes for one viewpoint. (a) The mesh viewed from the original viewpoint. (b) The mesh viewed from a viewpoint off of the track for illustration purpose. We can see the slab meshes which comprise the visibility polyhedron for that viewpoint.

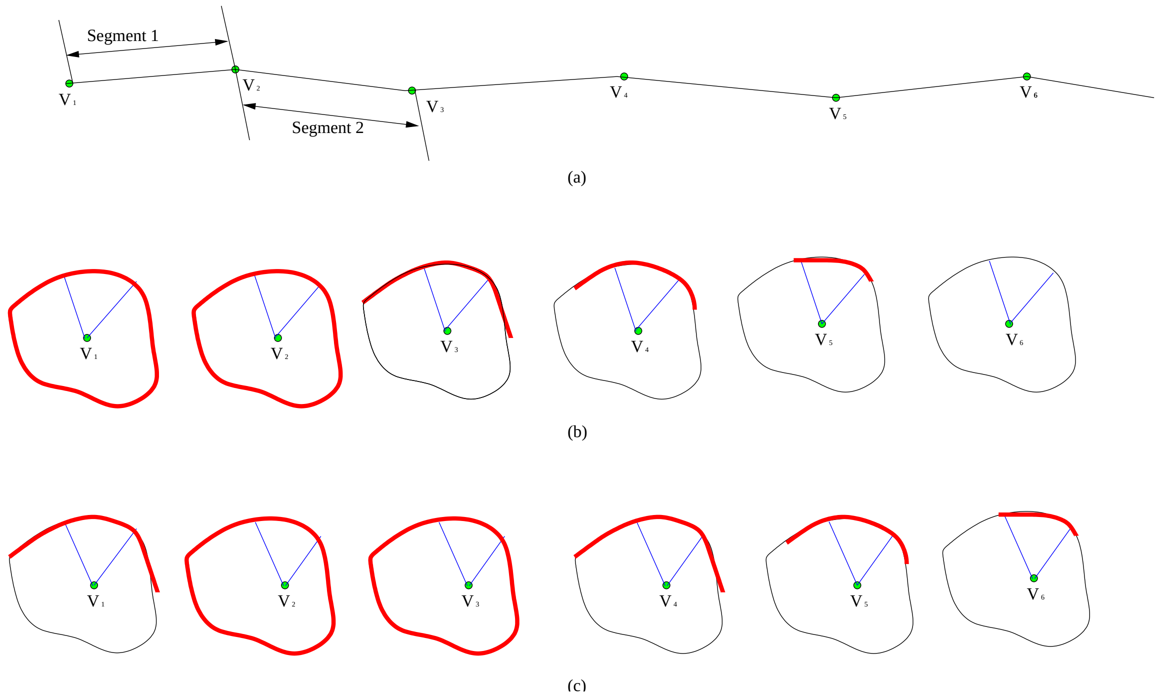

Figure 4 (a) defines a path with V_1 through V_6. A track segment is defined between the two closest-by viewpoints. The current novel viewpoint is on segment number 1 which is between V_1 and V_2. (b) The entire panorama for V_1 and V_2 are loaded into the cache, while only parts of the panoramas (tiles) are loaded in for V_3, V_4 and V_5. And less information is actually loaded for V_5 over V_3. (c) The user moves into segment 2, the novel view lies between V_2 and V_3, we use another thread to load in the remaining information for V_3 and pre-fetch partial information for V_6 which currently has no information in the cache.

Keeping the whole panorama in the memory allows the user to look around in horizontal and vertical directions interactively, but it requires too much memory and disk I/O. By examining the minimal information needed to reconstruct a novel view, we can reduce these demands and increase our pre-fetching length. We consider two scenarios. The first case is when the user heads straight down the track and keeps the view plane normal constant all the time. In this case, only the information within the user's viewing direction is needed. However, for fast movement, more than the two closest reference views are required. The second case is when the user only looks around and never moves on the track. This requires only the information of the whole panoramic of the current two closest reference views. Usually, the user can both move along the track and stop to look around. This means that a fair amount but not all of the panoramic is needed for more than the two closest reference views. Therefore, we have the choice of pre-fetching more of the panoramic of current views or more of the next views along the track. This is determined by the user – whether moving along the track fast or looking around interactively is more important. A compromise was achieved using an adaptive two-phase pre-fetching scheme – one along the track and the other along the viewing direction.

We adaptively reduce the information we pre-fetch as the pre-fetched views are farther away from the current viewpoint. Therefore for the closest two sampled views we load in most of the panorama. As we pre-fetch information of the farther away view samples we pre-fetch less and less of the information into the cache. This is illustrated in Figure 4. Figure 4 (a) defines a path with V_1 through V_6 reference views. A track segment is defined between the two close-by viewpoints. We have labeled these segments 1 through 5. The current novel viewpoint is on segment number 1 which is between V_1 and V_2. V_3, V_4 and V_5 are farther and farther away from the current view. Figure 4 (b) shows that the whole panoramas for V_1 and V_2 are loaded into the cache, while only parts of the panorama (tiles) are loaded in for V_3, V_4 and V_5. When we move to segment number 2, the novel view lies between V_2 and V_3, we use another thread to fetch the remaining information for V_3 and pre-fetch partial information for V_6 which currently has no information in the cache. This is shown in Figure 4 (c).

7. Error analysis

7.1 Depth-Mesh Errors

We consider two sources of errors. The first source of errors comes from down-sampling the depth-meshes. Reconstruction using standard graphics cards utilizes a bi-linear function to determine the depth values in the interior. This assumption, of course, is not always valid. Therefore, some errors are introduced when we reconstruct the reference views and novel views. We can certainly reduce this kind of error by using a finer quad-mesh (smaller tile size). However, this increases the rendering time, image and geometry loading time and storage requirements. Table 1 shows the Mean Squared Error (MSE), the image and geometry loading time, the rendering time (FPS) and the storage requirement using varying tile sizes for the POVRAY rendered Nature dataset. These are also plotted graphically in Figure 5. From the Figure we can see that the MSE decreases almost linearly when we decrease the tile size. However, the loading time increases quite dramatically with smaller tile sizes. The rendering speed decreases with decreased tile size. As tile size increases, the storage first decreases, hits a low value and then increases. The decrease at first is due to the fact that we need to store more depth values when we have more tiles with smaller size. The later increase is because when the tiles are too big, there are fewer empty tiles to remove. This is also the reason why the rendering speed levels off when the size increases to a certain value – we have more empty space to rasterize. After observations, we found that 16x16 or 32x32 tile sizes are good candidates for our application.

	Loading Time(s)	FPS	MSE	Storage (MB)
64x64	0.47	18	11.72	12.6
32x32	0.52	17	8.99	11.1
16x16	0.83	15	6.82	10.0
8x8	1.28	8	4.85	10.1
4x4	3.04	3	2.88	12.0
2x2	9.63	1	0	20.4

Table 1: shows the change of loading time, rendering speed, Mean Squared Error and Storage of one typical viewpoint with different tile size. This is for the Nature dataset. MSE is based on the maximum number of 256.

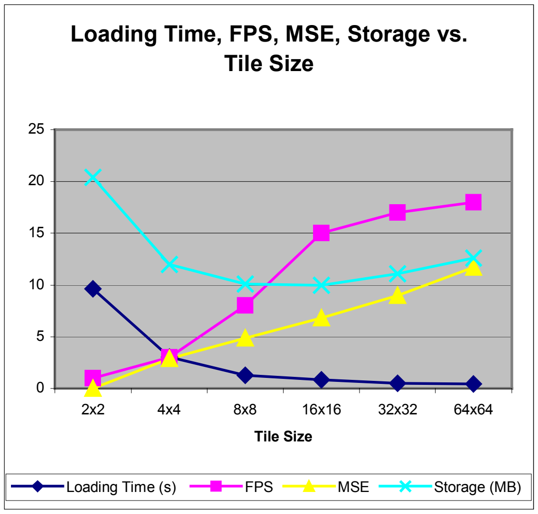

Figure 5: Shows the Mean Square Error (MSE), the loading time, Frame Per Second (FPS) and the storage requirement with different tile sizes for a typical sampled viewpoint of the POVRAY rendered Nature dataset.

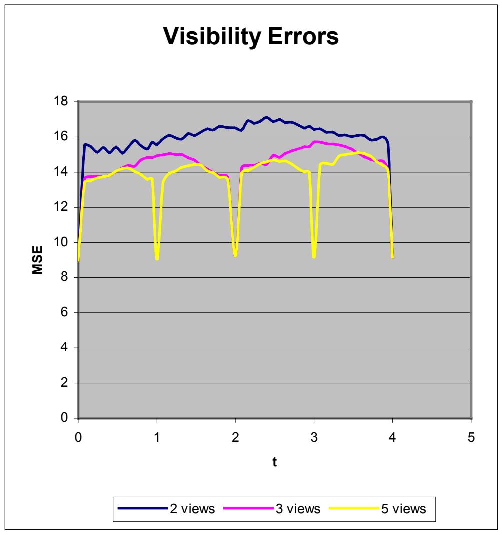

Figure 6: Mean Square Error of interpolated views against actual rendered views. Interpolated views have a tile size of 32x32

7.2 Visibility Errors

Another source of errors are the visibility errors that come from reconstructing the novel view by cross-dissolving the two close-by sampled views. As described previously, if the visibility polyhedrons of the two sampled views coincide with each other perfectly, no visibility errors will occur. Figure 6 shows the MSE of the resulting images for interpolated views against Povray rendered views at corresponding positions along our path. Three different sampling rates are examined and a tile size of 32x32 is used. We can see that the peak errors occur somewhere close to the middle of the two sampled viewpoints for all three cases. This is expected because the visibility errors are most serious in the middle of the two samples. The curve which uses 2 sampled viewpoints has the coarsest sampling rate and therefore has the highest mean square error. The Peak is about 17 out of 256, which is approximately 6.7%. The curve which uses 5 views has the finest sampling rate and peaks out with an MSE of about 14 - 15. The minimum MSE occurs at the sampled viewpoints, which comes entirely from the down-sampling errors. Figures 8 and 9 (see color section) show the interpolated and actually rendered images using the same viewpoint, Figure 10 (see color section) shows the difference image. Here we see the majority of the errors occur on silluoettes where the depth values change substantially.

8. Results and Discussions

Our rail-track viewer was implemented in Java/Java3D. We ran our IBR framework on 2 datasets. The first one is a virtual scene rendered using Povray [21]. The scene is fairly complicated containing trees, bushes, rocks and birds. It requires almost 30 minutes to render one frame using Povray on our dual Pentium II 500MHZ, 512MB SGI visual 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 220MB. The geometry and imagery was broken up into 32x32 quads. The pre-rendering takes about 40 hours on the visual workstation. Our second dataset is the LOX post dataset which Visualization Toolkit (VTK) [22] provides. This dataset simulates the flow of liquid oxygen across a flat plate with a cylindrical post perpendicular to the flow. It is a complex scientific dataset which contains both scalar and vector fields in the data. A rendering was chosen with the post, a slice plane and several stream-polygons. One path was pre-selected going into the stream-polygons region with 23 view samples. Four panoramic layers with a resolution of 512x2048 per layer were pre-computed for each view sample. The reference image database was pre-rendered using VTK and the total size for the image database is 138MB. The geometry and imagery was broken up into 16x16 quads. This database required 9.2 hours to pre-render.

For the Povray dataset, our IBR viewer achieves 10-15 frames per second with a 1kx1k image resolution on the Sun Blade 1000 workstation with dual Ultra-Sparc III 750MHZ, 1 GB of memory and Elite 3D graphics card. For the LOX dataset, we see 15-20 frames per second with a 512x512 resolution on the same platform. Figure 7 (see color section) shows one rendered view for the nature dataset with the quad-meshes represented by white squares and color information as texture maps. Figures 8, 11 and 12 (see color section) show some resulting images from our IBR system.

9. Conclusions and Future Work

This paper presents our framework for allowing the user to examine and walkthrough the scene from an internal vantage point on a relatively high-resolution display interactively. We achieved this goal by limiting the user movement on a track and utilizing IBR techniques. Intelligent data organization, caching and pre-fetching make our system more efficient. We prove by results that our system is suitable for exploring complex virtual environments and large datasets with reasonably small errors.

Triangular meshes would decrease the down-sampling error [6]. We are looking at ways of using triangular meshes to replace fixed size quad-meshes. Even with the empty tile removal scheme, our IBR database still tends to be very large, especially with larger resolution and finer sampling. Future work entails looking at appropriate compression techniques to reduce the data storage requirements.

10. 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] E. H. Adelson, J. R. Bergen, "The plenoptic Function and the Elements of Early Vision," *Computational Models of Visual Processing, Chapter 1*, Edited by Michael Landy and J. Anthony Movshon, The MIT Press, Cambridge, Massachusetts, 1991

[2] Chun-Fa Chang, Gary Bishop, Anselmo Lastra, "LDI Tree: A Hierarchical Representation for Image-Based Rendering," *Proc. SIGGRAPH '99*, pp. 291-298, 1999

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

[4] J-J Choi and Y. Shin, "Efficient Image-Based Rendering of Volume Data," *Proc. Pacific Graphics '98*, pp. 70-78, 1998

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

[6] 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

[7] 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

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

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

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

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

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

[13] K. Mueller, N. Shareef, J. Huang, and R. Crawfis, "IBR-Assisted Volume Rendering," *LBHT IEEE Visualization '99*, pp. 5-8, 1999

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

[15] B. Oh, M. Chen, J. Dorsey and F. Durand, "Image Based Modeling and Photo Editing," *Proc. SIGGRAPH '01*, pp 433 – 442, 2001

[16] H. Qu, M. Wan, J. Qin and A. Kaufman, "Image Based Rendering With Stable Frame Rates," *Proc. IEEE Vis 2000*, pp 251-258, 2000

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

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

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

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

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

[22] http://www.kitware.com/vtk

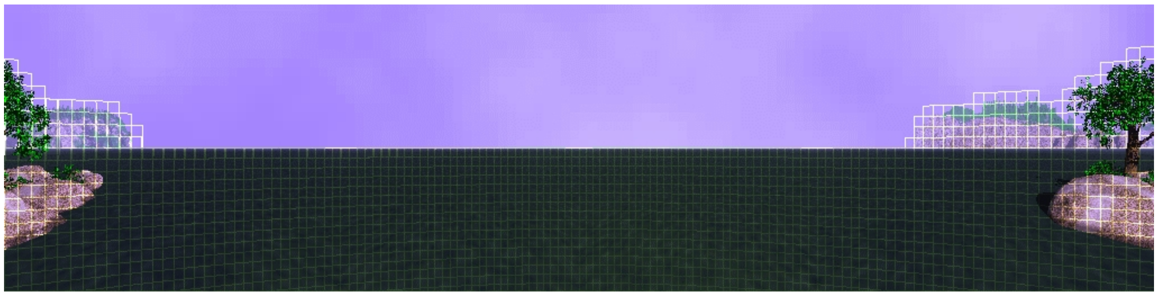

Figure 7: shows one rendered view with quad-meshes represented by white squares and color information as texture maps. This figure also shows our empty tile removal scheme. Any tiles with no useful information won't be reconstructed or rendered.

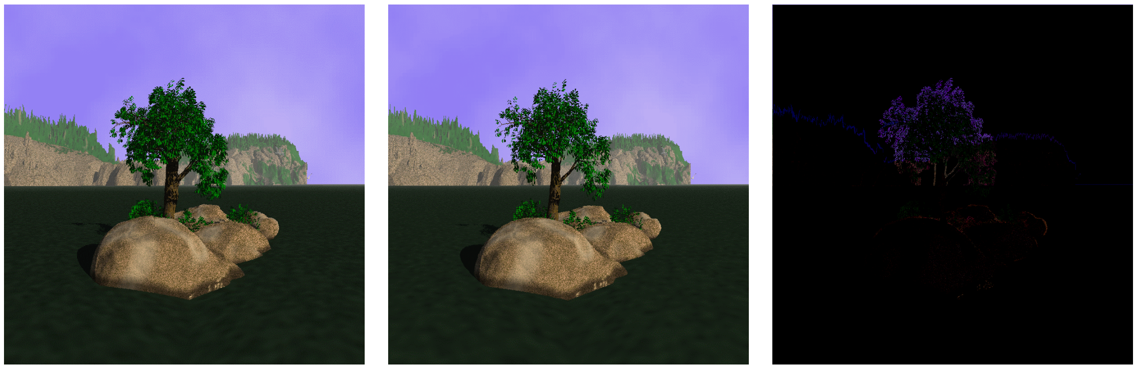

Figure 8: The interpolated image for one viewpoint which is in the middle of the two reference viewpoints
Figure 9: The actual rendered image for the viewpoint at the same position as in Figure 8
Figure 10: The difference image between Figure 8 and Figure 9

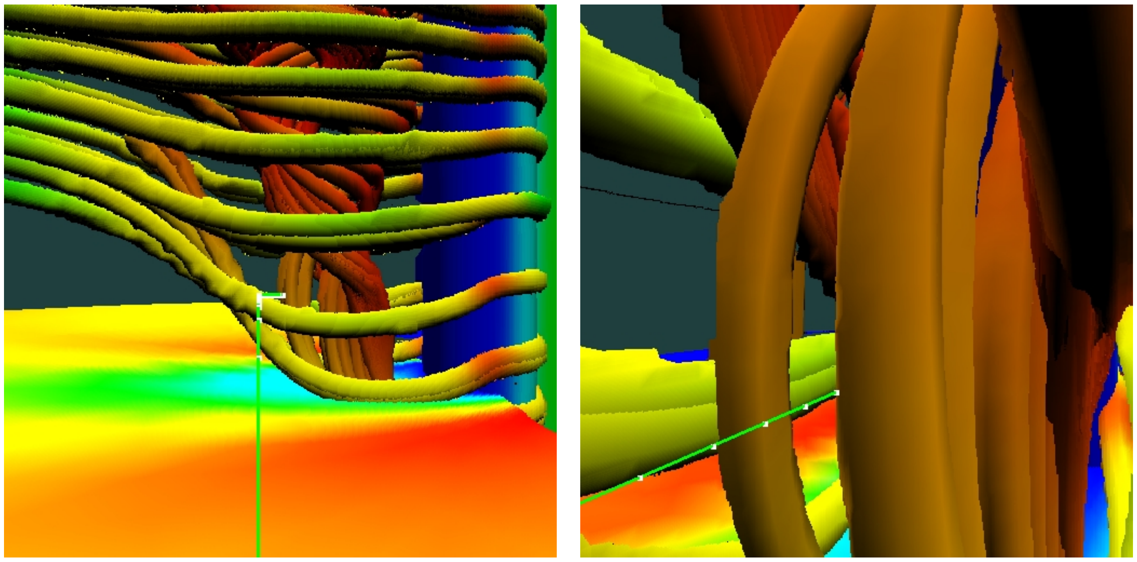

Figure 11: The interpolated image for one viewpoint which is outside of the stream-polygon region of the LOX dataset
Figure 12: The interpolated image for one viewpoint which is inside of the stream-polygon region of the LOX dataset