

LDI Tree: A Hierarchical Representation for Image-Based Rendering

Chun-Fa Chang Gary Bishop Anselmo Lastra
University of North Carolina at Chapel Hill

ABSTRACT

Using multiple reference images in 3D image warping has been a challenging problem. Recently, the Layered Depth Image (LDI) was proposed by Shade et al. to merge multiple reference images under a single center of projection, while maintaining the simplicity of warping a single reference image. However it does not consider the issue of sampling rate.

We present the LDI tree, which combines a hierarchical space partitioning scheme with the concept of the LDI. It preserves the sampling rates of the reference images by adaptively selecting an LDI in the LDI tree for each pixel. While rendering from the LDI tree, we only have to traverse the LDI tree to the levels that are comparable to the sampling rate of the output image. We also present a progressive refinement feature and a “gap filling” algorithm implemented by pre-filtering the LDI tree.

We show that the amount of memory required has the same order of growth as the 2D reference images. This also bounds the complexity of rendering time to be less than directly rendering from all reference images.

CR Categories: I.3.3 [Computer Graphics]: Picture/Image Generation - Viewing Algorithms; I.3.6 [Computer Graphics] Methodology and Techniques - Graphics data structures and data types; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism.

Additional Keywords: image-based rendering, hierarchical representation

1. INTRODUCTION

The 3D Image warping algorithm [14] proposed by McMillan and Bishop uses regular single-layered depth images (which are called *reference images*) as the initial input. One of the major problems of 3D image warping is the disocclusion artifacts which are caused by the areas that are occluded in the original reference image but visible in the current view. Those artifacts appear as tears or gaps in the output image. In Mark’s Post-Rendering Warping [11], the techniques of splatting and meshing are proposed to deal with the disocclusion artifacts. Both splatting and meshing are adequate for post-rendering warping in which the current view does not deviate much from the view of the reference image.

However, the fundamental problem of the disocclusion arti-

CB#3175 Sitterson Hall, Chapel Hill, NC 27599-3175, USA.
{chang, gb, lastra}@cs.unc.edu <http://www.cs.unc.edu/~ibr>

facts is that the information of the previously occluded area is missing in the reference image. By using multiple reference images taken from different viewpoints, the disocclusion artifacts can be reduced because an area that is not visible from one view may be visible from another. When multiple source images are available, we expect the disocclusion artifacts that occur while warping one reference image to be eliminated by one of the other reference images. However, combining multiple reference images and eliminating the redundant information is a non-trivial problem, as pointed out by McMillan in his discussion of inverse warping [15].

Recently, the Layered Depth Image (LDI) was proposed by Shade et al. [19] to merge many reference images under a single center of projection. It tackles the occlusion problems by keeping multiple depth pixels per pixel location, while still maintaining the simplicity of warping a single reference image. Its limitation is that the fixed resolution of the LDI may not provide an adequate sampling rate for every reference image. Figure 1 shows two examples of such situations. Assuming the two reference images have the same resolution as the LDI, the object covers more pixels in reference image 1 than it does in the LDI. Therefore the LDI has a lower sampling rate for the object than reference image 1. Similar analysis shows the LDI has a higher sampling rate than reference image 2. If we combine both reference images into the LDI and render the object from the center of projection of reference image 1, the insufficient sampling rate of the LDI will cause the object to look more blurry than it looks in reference image 1. When we render the object from the center of projection of reference image 2, the excessive sampling rate of the LDI might not hurt the quality of the output. However, processing more pixels than necessary slows down the rendering.

In this paper, we present the *LDI Tree*, which combines a hierarchical space partition scheme with the concept of the LDI. It preserves the sampling rate of the reference images by adaptively selecting an LDI in the LDI tree for each pixel. While rendering from the LDI tree, we only have to traverse the LDI tree to the levels that are comparable to the sampling rate of the output image. Because each LDI also contains pre-filtered results from its children LDIs, progressive refinement is easy to implement. The pre-filtering also enables a new “gap filling” algorithm to fill the disocclusion artifacts that cannot be resolved by any reference image.

The amount of memory required has the same order of growth as the 2D reference images. Therefore the LDI tree preserves an important feature that image-based rendering has over traditional polygon-based rendering: the cost is bounded by the complexity of the reference images, not by the complexity of the scene.

2. RELATED WORK

2.1. Inverse Warping

The image warping described in [14] is a forward warping process. The pixels of the reference images are traversed and warped to the output image in the order they appear in the reference images. Some pixels in the output image may receive more than

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SIGGRAPH 99, Los Angeles, CA USA
Copyright ACM 1999 0-201-48560-5/99/08...\$5.00

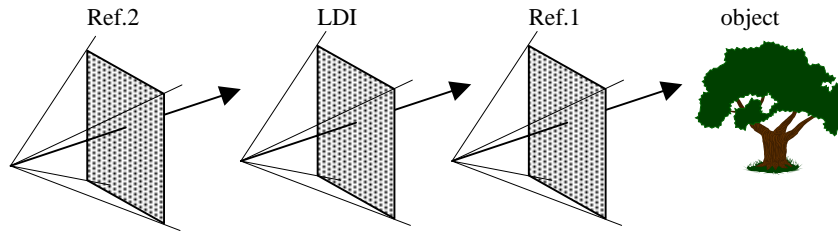


Figure 1: The LDI does not preserve the sampling rates of the reference images.

one warped pixel and some may receive none, which causes artifacts.

In [15], McMillan proposed an inverse warping algorithm. For each pixel in the output image, searches are performed in all reference images to find the pixels that could be warped to the specified location in the output image. Although epipolar geometry limits the search space to a one-dimensional line or curve in each reference image and a quadtree-based optimization has been proposed in [10], searching through all reference images is still time consuming.

2.2. Layered Depth Image

Another way to deal with the disocclusion artifacts of image warping is to use the Layered Depth Image (LDI)[19]. Given a set of reference images, one can create an LDI by warping all reference images to a carefully chosen camera setup (e.g. center of projection and view frustum) which is usually close to the camera of one of the reference images. When more than one pixel is warped to the same pixel location of the LDI, some of them may be occluded. Although the occluded pixels are not visible from the viewpoint of the LDI, they are not discarded. Instead, separate layers are created to store the occluded pixels. Those extra pixels are likely to reduce the disocclusion artifacts. However the fixed resolution of the LDI limits its use as discussed previously in section 1.

Lischinski and Rappoport used three parallel-projection LDIs to form a Layered Depth Cube [9]. Max's hierarchical rendering method [12] uses the Precomputed Multi-Layer Z-Buffers which are similar to the LDIs. It generates the LDIs from polygons and the hierarchy is built into the model.

2.3. Volumetric Methods

The LDI resembles volumetric representations. The main differences between an LDI-based representation and 3D volume data are discussed in [9].

Curless and Levoy presented a volumetric method to extract an isosurface from range images [3]. The goal of their work, however, was to build high-detail models made of triangles. The volume data used in that method is not hierarchical and it relies on a run-length encoding for space efficiency.

There has also been work related to octree generation from range images [1][2][8]. However the octree that is generated in those methods is used to encode the space occupancy information. Each octree cell represents either completely occupied or completely empty parts of the scene.

The multi-resolution volume representation in the Hierarchical Splatting work [6] by Laur and Hanrahan can be considered as a special case of the LDI tree in which the LDIs are of 1×1 resolution. It is however built from a fully expanded octree (which is called a pyramid in their paper). The octree to be traversed during the rendering is also predetermined and does not change with the viewpoint.

2.4. Image Caching for Rendering Polygonal Models

The image caching techniques of Shade et al. [18] and Schaufler et al. [17] use a hierarchical structure similar to the LDI tree. Each space partition has an imposter instead of an LDI. The imposter can be generated rapidly from the objects within the space partition by using hardware acceleration. However, the imposter has to be frequently regenerated whenever it is no longer suitable for the new view.

In contrast, the information stored in the LDI tree is valid at all times. By generating the LDI tree from the reference images instead of the objects within the space partitions, the LDI tree can be used for non-synthesized scenes as well.

3. LDI TREE

The LDI tree is an octree with an LDI attached to each octree cell (node). The octree is chosen for its simplicity but can be replaced by the other space partitioning schemes. Each octree cell also contains a bounding box and pointers to its eight children cells. The root of the octree contains the bounding box of the scene to be rendered¹. The following is pseudo code representing the data structure:

```
LDI_tree_node =
  Bounding_box[X..Z, Min..max]: array of
    real;
  Children[0..7]: array of pointer to
    LDI_tree_node;
  LDI: Layered_depth_image
```

All LDIs in the LDI tree have the same resolution, which can be set arbitrarily. The height (or number of levels) of the LDI tree will adapt to different choices of resolution. In general, a lower resolution results in more levels in the LDI tree. Ultimately, we can make the resolution of the LDI be 1×1 which makes the LDI tree resemble the volume data in the Hierarchical Splatting [6].

Note that each LDI in the LDI tree contains only the samples from objects within the bounding box of the cell. This is sometimes confusing because the LDI originally proposed by Shade et al. combines the samples from all reference images.

For simplicity, we use one face of the bounding box as the projection plane of the LDI. Orthographic projection is used and the projection direction is perpendicular to the projection plane.

An example of the LDI tree is shown in Figure 7 by viewing the bounding boxes from the top. The following sections discuss the details of constructing the LDI tree from multiple reference images and of rendering a new view from the LDI tree.

¹ For outdoor scenes, background textures can be added to the faces of the bounding box. The bounding box can be extended with little overhead if most of the space is empty.

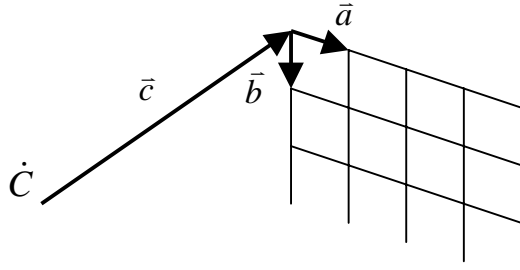


Figure 2: The camera model.

3.1. Constructing the LDI Tree from Multiple Reference Images

The LDI tree is constructed from reference images by warping each pixel of the reference images to the LDI of an octree cell, then filtering the affected LDI pixels to the LDIs of all ancestor cells in the octree.

In 3D image warping, each pixel of the reference images contains depth information which is either stored explicitly as a depth value or implicitly as a disparity value. This allows us to project the center of the pixel to a point in the space where the scene described by the reference images resides.

We observed that the sampling rate or the "quality" of a pixel of a reference image depends on its depth information. For example, if (part of) a reference image represents a surface that is far away, then those pixels that describe that surface do not provide enough detail when the viewer zooms in or walks toward that surface. Conversely, warping every pixel of a reference image taken near an object is wasteful when the object is viewed from far away.

We characterize the reference image by a pinhole camera model using the notation adopted by McMillan [14][15]. Figure 2 illustrates the camera model. \dot{C} is the center of projection. Each pixel of the reference image has coordinates (u, v) and the vectors \vec{a} and \vec{b} are the bases. Each pixel also contains the color information and a disparity value δ . When a pixel is projected to the 3D object space, we get a point representing the center of the projected pixel and a "stamp size." The center is computed as:

$$\dot{C} + (u\vec{a} + v\vec{b} + \vec{c}) / \delta \quad (1)$$

and the stamp size S is calculated by:

$$\begin{aligned} S &= S_X \times S_Y \\ S_X &= |\vec{a}| / \delta \\ S_Y &= |\vec{b}| / \delta \end{aligned} \quad (2)$$

To simplify our discussion, we do not consider the orientation of the object surface from which the pixel is taken. We also ignore the slight variation of stamp size at the edges of the projection plane.

An octree cell is then selected to store this pixel. The center location determines which branch of the octree to follow. The stamp size determines which level (or what size) of the octree cell should be used. The level is chosen such that the stamp size approximately matches the pixel size of the LDI in that cell.

After an octree cell has been chosen, the pixel can then be warped to the LDI of that cell. The details of the warping are described in [11]. Usually, the center of the pixel will not fall exactly on the grid of the LDI, so resampling is necessary. This is

done by splatting [20] the pixel to the neighboring grid points. In this paper we use a bilinear kernel. Four LDI pixels are updated for each pixel of a reference image. More specifically, the alpha values that result from the splatting are computed by:

$$P_X = B_X / N_X$$

$$P_Y = B_Y / N_Y$$

$$\text{Kernel}(d, s) = 1 - \frac{d}{s}$$

$$W_X = \begin{cases} \text{Kernel}(|X_i - X_c|, \frac{S_X}{P_X}), & S_X > P_X \\ \text{Kernel}(|X_i - X_c|, 1) * \frac{S_X}{P_X}, & S_X \leq P_X \end{cases} \quad (3a)$$

$$W_Y = \begin{cases} \text{Kernel}(|Y_i - Y_c|, \frac{S_Y}{P_Y}), & S_Y > P_Y \\ \text{Kernel}(|Y_i - Y_c|, 1) * \frac{S_Y}{P_Y}, & S_Y \leq P_Y \end{cases} \quad (3b)$$

$$\text{alpha} = W_X W_Y \quad (3)$$

where B_X and B_Y are the sizes of the LDI projection plane (which is a face of the bounding box). N_X and N_Y are the resolutions of the LDI. S_X and S_Y are as defined in equation 2. (X_c, Y_c) is the center of splatting in the selected LDI and (X_i, Y_i) is one of the grid points covered by the splatting. The conditions in equations 3a and 3b guarantee that the splat size will not be smaller than the LDI grid size, which represents the maximal sampling rate of the LDI.²

A pixel also contributes to the parent cell and all ancestor cells of the octree cell that was initially chosen. This is done by splatting the pixel to the LDIs of all the ancestor cells. The result is that the LDI of a cell contains the samples within its descendants filtered down to its resolution. Therefore, later in the rendering stage, we need not traverse the children cells if the current cell already provides enough detail.

We classify the pixels in the LDI tree into two categories: *unfiltered* and *filtered*. The unfiltered pixels are those that come from the splatting to the octree cell that was initially chosen for a reference image pixel. Those pixels that come from the splatting to the ancestor cells are classified as filtered, because they represent lower frequency components of the unfiltered pixels. Note that an unfiltered pixel may be merged with a filtered pixel during the construction of LDI tree. The merged pixel is considered as filtered because better-sampled pixels are in the LDIs of some children cells of the current octree cell.

The classification of unfiltered and filtered pixels is necessary for rendering the output images (as described in section 3.2). Imagine that a cell contains unfiltered pixels of a surface area that is only visible from one of the reference images. When the cell and its children cells are processed during the rendering, we must warp its unfiltered pixels but not its filtered pixels that are filtered from the children cells.

² It is similar to how the subpixels are prefiltered in supersampling for antialiasing.

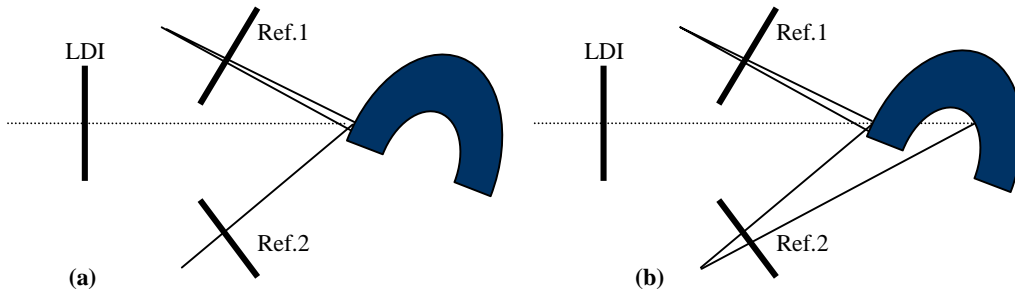


Figure 3: Illustrations of pixels that are warped to the same pixel location in an LDI. (a) Two pixels from reference image 1 and a pixel from reference image 2 are taken from the same region of a surface. Blending is used to combine their contribution to the LDI pixel. (b) One of the pixels from reference image 2 is taken from a different surface. A separate layer in the LDI is created to accommodate its contribution to the same LDI pixel.

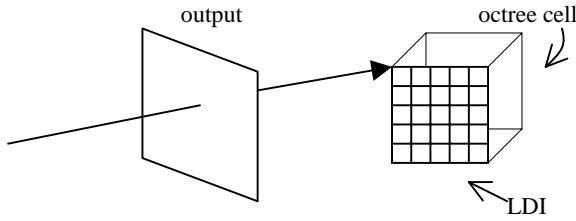


Figure 4: To estimate the range of stamp size for all pixels in the LDI, the corners of the bounding box are warped to the output image.

An LDI pixel may get contributions from many pixels of the same surface. They may be neighboring pixels in the same reference image, or pixels in different reference images that sample the same surface. The contributions from those pixels must be blended together. Figure 3a shows an example of those cases. An LDI pixel can also get contributions from many pixels of different surfaces. In those cases, we assign them to different layers of the LDI pixel. Figure 3b shows an example of those cases. To determine whether they are from the same surface or not, we check the difference in their depth value against a threshold. We select the threshold to be slightly smaller than the spacing between adjacent LDI pixels, so that the sampling rate of a surface that is perpendicular to the projection plane of the LDI can be preserved.

3.2. Rendering the Output Images

We render a new view of the scene by warping the LDIs in the octree cells to the output image. The advantage of having a hierarchical model is that we need not render every LDI in the octree. For those cells that are farther away, we can render them in less detail by using the filtered samples that are stored in the LDIs higher in the hierarchy.

To start the rendering, we traverse the octree from the top-level cell (i.e. the root). At each cell, we first perform view frustum culling, then check whether it can provide enough detail if its LDI is warped to the output image. If the current cell does not provide enough detail, then its children are traversed. An LDI is considered to provide enough detail if the pixel stamp size covers about one output pixel. Therefore the traversal of the LDI tree during the rendering will adapt to the resolution of the output image. Note that we do not calculate the pixel stamp size for each individual pixel in an LDI. Because all the pixels in the LDI of an octree cell represent samples of objects that are within its bounding box (as shown in Figure 4), we can estimate the range of stamp size for all pixels of the LDI by warping the LDI pixels that

correspond to the corners of the bounding box. The corners of the bounding box are obtained by placing the maximal and minimal possible depth at the four corner pixel locations of the LDI. We use equation 2 to compute the stamp size with the vector \vec{a} and \vec{b} of the output image and the disparity value δ obtained from the warping. Note that a special case exists if the new viewpoint is within the octree cell. When this happens we consider the cell as not providing enough detail and the children are traversed.

The pseudo code for the octree traversal follows:

```
Render (Octree) {
1. If outside of view frustum,
   then return;
2. Estimate the stamp size of the LDI
   pixels;
3. If LDI stamp size is too large or the
   viewer is inside the bounding box then {
4.   Call Render() recursively for each
   child;
5.   Warp the unfiltered pixels in LDI to
   the Output buffer; }
6. else {
7.   Warp both unfiltered and filtered
   pixels in LDI to the output buffer; }
}
```

Note the difference in step 5 and step 7 of the pseudo code. As mentioned in section 3.1, each LDI in the octree contains both unfiltered and filtered pixels. When we warp both the LDI in a parent cell and the LDI in a child cell, the filtered pixels in the parent cell should not contribute to the output because the unfiltered pixels in the child cell already provide better sampling for the same part of the scene.

One feature of the original LDI is that it preserves the occlusion compatible order in McMillan's 3D warping algorithm [13][14]. However this feature is compromised in the LDI tree. Although the back-to-front order can still be obtained within an LDI and across LDIs of sibling cells of the octree, we cannot obtain such order between LDIs of a parent cell and a child cell. This causes problems when unfiltered samples exist in both parent and child cells. In addition, the warped pixels are semi-transparent due to the splatting process. Therefore, we need to keep a list of pixels for each pixel location in the output buffer. We implement the output buffer as an LDI. At the end of the rendering, each list is composited to a color for display. The details of the compositing are discussed next.

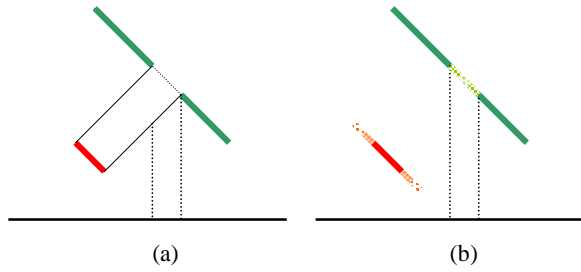


Figure 5: This example shows the different results of gap filling from the meshing method and the method presented in this paper. (a) The meshing method. (b) The gap filling method using filtered samples.

3.3. Compositing in the Output Buffer

Given a list of semi-transparent pixels, we sort the pixels in depth and then use alpha blending starting from the front of the sorted list. An exception is that two pixels with similar depth should be merged first and their alpha values summed together before they are alpha-blended with the other pixels. That is because they are likely to represent sampling of the same surface.

Therefore, the pixel merging is also performed in the output LDI, which is similar to the pixel merging in the LDI of the octree cell as discussed in section 3.1. The difference is that a single threshold value of depth difference does not work anymore because the pixels can come from different levels of the LDI tree. This difficulty is solved by attaching the level of octree cell where the pixel comes from to each pixel in the output LDI. The threshold value that is used for that level of octree is then used to determine whether two pixels in the output LDI should be merged.

3.4. Progressive Refinement

As discussed in section 3.2, the traversal of the LDI tree during the rendering depends on the resolution of the output image. The simplest method to create the effect of progressive refinement is to render the LDI tree to a low-resolution output image first, then increase the resolution gradually. However, this method does not utilize the coherence between the renderings of two different resolutions.

To utilize the coherence between two renderings, we can tag the octree cells that are traversed in the previous rendering and skip them in the current rendering. Note that some filtered pixels may have been warped to the output buffer if they are from the leaf nodes of the subtree traversed in the previous rendering³. Those pixels must also be tagged so they can be removed from the output buffer if the leaf nodes in the previous rendering become interior nodes in the current rendering.

3.5. Gap Filling

When we construct the LDI tree from many reference images, chances are we have eliminated most of the disocclusion artifacts. However, it is possible that some disocclusion artifacts still remain. We propose a two-pass algorithm that uses the filtered pixels in the LDI tree to fill in the gaps in the output image. The algorithm consists of the following steps:

1. The first pass is to render the output image from the LDI tree as discussed in section 3.2.

2. A stencil (or coverage of pixels) is then built from the output image.
3. Render the output image from the LDI tree again. But in this pass, splat only the filtered pixels.
4. Use the stencil from step 2 to add the image from step 3 to the image from step 1.

The stencil from step 2 allows the filtered pixels to draw only to the gaps in the output image from step 1. This assumes that the output image would be completely filled if no disocclusion artifact occurred.⁴

Our gap filling method produces different results from the meshing method described in Mark’s Post-Rendering 3D Warping [11]. Figure 5 shows an example of the gap that is caused by a front surface occluding a back surface. In the meshing method, the gaps are covered by quadrilaterals stretching between the front surface and the back surface (figure 5a). In contrast, our gap filling method splats the filtered samples from surfaces that surround the gap in the output. As shown in figure 5b, the back surfaces make more contribution to the gap than they do in the meshing method. If we do not have additional surface connectivity information in the original reference image, we believe the methods like ours that are based on the filtering of existing samples are more robust.

3.6. Analysis of Memory Requirement

Although a complete, fully expanded LDI tree may contain too many LDIs to be practical for implementation, it is worth noting that only a small subset of a complete LDI tree is used when it is constructed from reference images.

When we construct the LDI tree from reference images, we add a constant number of unfiltered LDI pixels to the octree cell chosen for each pixel of reference images. We also add $O(h)$ filtered LDI pixels to the ancestor cells, where h is the number of ancestors. That means the amount of memory taken by the LDI tree grows in the same order as the amount taken by the original reference images, only if h is bounded.

We can further assume that h is bounded because the maximal height of the LDI tree exists. Let L be the longest side of bounding box of the scene, N be the resolution of an LDI, d be the smallest feature in the scene the human eyes can discern at a minimum distance, and H be the maximal height of the LDI tree. Then we have:

$$H = \left\lceil \log_2 \frac{L}{N \times d} \right\rceil$$

Although we do not include the memory overhead for maintaining the octree, we also do not include the possible saving in memory when pixels are merged in the LDIs. The experimental results will be presented later in this paper to show that amount of memory indeed grows at a slower rate than the number of reference images.

3.7. Rendering Time

An advantage that image-based rendering has over traditional polygon-based rendering is that the rendering time does not grow with the complexity of the scene. That advantage is still preserved in the rendering from the LDI tree, even though more layers of LDIs must be rendered. Let us consider the worst case in which we need to render every pixel in the LDI tree. As discussed

³ See line 7 of the pseudo code in section 3.2.

⁴ See previous footnote ¹ for special cases such as the windows in the video and figure 11.

previously, the number of pixels grows in the same order as the original reference images. Therefore the time complexity of rendering from the LDI tree is of the same order as warping all reference images in the worst case. Because larger cells are used for farther objects, the worst case rarely happens and usually much fewer pixels in the LDI tree are rendered. The experimental results are presented in the next section.

4. RESULTS

We implemented the LDI tree on a Silicon Graphics Onyx2 with 16 gigabytes of main memory. The machine has 32 250 MHz MIPS R10000 processors but we did not exploit its parallel processing capability in our implementation.

We tested our program with a model of the interior of Palladio's Il Redentore in Venice [16]. The reference images are generated by ray tracing using the Rayshade program [5]. Each reference image has 512×512 pixels and 90-degree field of view. Figure 6 shows one of the reference images.

In synthesized scenes, an LDI can be generated directly by ray tracing [19]. We do not include it in our framework because it does not apply to the reference images acquired from non-synthesized scenes, such as the depth images that are acquired by a laser range finder.

Figure 7 shows the top view of the bounding boxes of the LDI tree after two of the reference images are processed. Each cell has an LDI of 64×64 resolution. The left face of each cell is also the projection plane of its LDI. Note that the cells near the center of projection of a reference image have more levels of subdivision. Figure 8 shows a new view rendered from the LDI tree. We disabled the gap filling to let the disocclusion artifacts appear in blue background color. Figure 8 has severe disocclusion artifacts because only four reference images from the same viewpoint are used. Figures 9 and 10 show the same view but with 12 and 36 reference images (from 3 and 9 viewpoints) respectively. Figure 11 is generated from the same LDI tree as figure 10 but with the gap filling enabled.

The memory usage of the LDI trees is shown in chart 1. The first reference image consumes about 30 Mbytes (MB) of memory. About 15 MB is the overhead of the octree. The resampling and filtering (described in section 3.1) generates about 5 LDI pixels for each input pixel. As more reference images are added, the growth of the memory size slows. The last 60 images add less than 1 MB per image in average. Note that the growth of the memory size does not stop completely. That is because more detail near each new viewpoint is still being added to the LDI Tree.

Chart 2 shows the rendering time for various numbers of reference images. Each line represents the rendering times along the path for a given number of reference images. The priority in our experiment is the correctness. Therefore little optimization and hardware acceleration were used to speed up the rendering. For example, the splatting operation is implemented completely in software simulation.

Chart 3 shows the growth of the (averaged) rendering time when the number of reference images increases. It shows that the rendering time grows even slower than the size of memory because some unnecessary details added from additional reference images are not processed during the rendering.

5. CONCLUSION AND FUTURE WORK

Using multiple reference images in 3D image warping has been a challenging problem. This paper describes the LDI tree, which

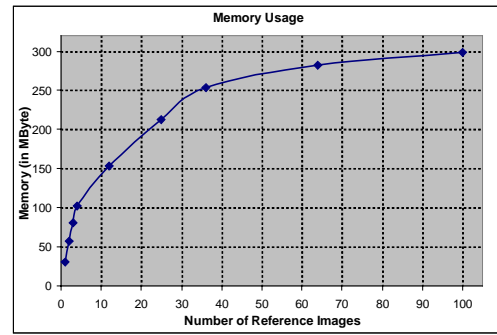


Chart 1: The memory usage of LDI trees.

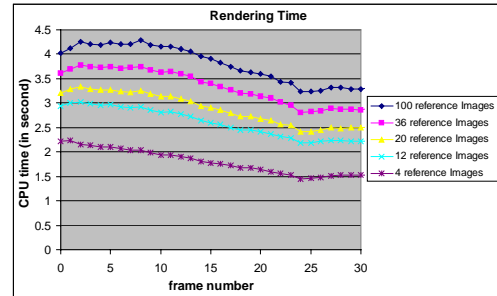


Chart 2: The rendering time.

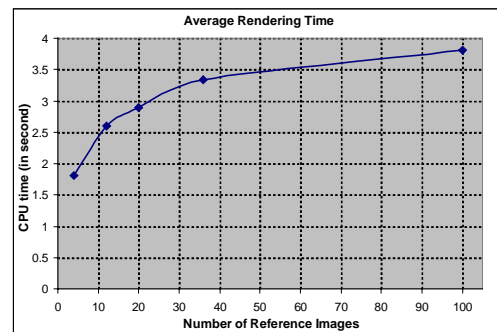


Chart 3: The average rendering time per frame.

combines multiple reference images into a hierarchical representation and preserves their sampling rate of the scene. The LDI tree allows the efficient extraction of the best available samples for any view and uses filtered samples in the hierarchy to reduce the rendering time. The filtered samples also enable the gap filling method presented in section 3.5.

We have assumed that each pixel of reference images provides only the color and depth information. No surface normal or orientation information has been considered. A direction for future work is to incorporate the surface orientation into our framework, for use in the splatting and the calculation of stamp size.

When a surface is sampled in multiple reference images, we should be able to get better sampling of the surface than what we can get from any single image. How to explore this type of cross-image supersampling is another direction of future work.

Like the original LDI, pixels that fall into the same pixel location and have similar depth values are merged together. That is based on the assumption that the surface is diffuse and little view-dependent variance can occur. How to extract view-dependent properties of the surface is yet another direction for future work.

6. ACKNOWLEDGEMENTS

We thank David McAllister for generating the reference images used in this paper, Nathan O'Brien for creating the excellent model of Il Redentore and the permission to use it, and the SIGGRAPH reviewers for their valuable comments. This work is supported by DARPA ITO contract number E278 and NSF MIP-9612643. Generous equipment support was provided by the Intel Corporation.

7. REFERENCES

- [1] C. H. Chien, Y. B. Sim and J. K. Aggarwal. Generation of Volume/Surface Octree from Range Data. The Computer Society Conference on Computer Vision and Pattern Recognition, pages 254-60, June 1988.
- [2] C. I. Connolly. Cumulative Generation of Octree Models from Range Data. Proceedings, Intl' Conf. Robotics, pages 25-32, March 1984.
- [3] Brian Curless and Marc Levoy. A Volumetric Method for Building Complex Models from Range Images. In Proceedings of SIGGRAPH 1996, pages 303-312.
- [4] Steven J. Gortler, Radek Grzeszczuk, Richard Szeliski and Michael F. Cohen. The Lumigraph. In Proceedings of SIGGRAPH 1996, pages 43-54.
- [5] Craig Kolb. Rayshade.
<http://www-graphics.stanford.edu/~cek/rayshade/>.
- [6] David Laur and Pat Hanrahan. Hierarchical Splatting: A Progressive Refinement Algorithm for Volume Rendering. Computer Graphics (SIGGRAPH 91 Conference Proceedings), volume 25, pages 285-288.
- [7] Marc Levoy and Pat Hanrahan. Light Field Rendering. In Proceedings of SIGGRAPH 1996, pages 31-42.
- [8] A. Li and G. Crebbin. Octree Encoding of Objects from Range Images. Pattern Recognition, 27(5):727-739, May 1994.
- [9] Dani Lischinski and Ari Rappoport. Image-Based Rendering for Non-Diffuse Synthetic Scenes. Rendering Techniques '98 (Proc. 9th Eurographics Workshop on Rendering).
- [10] Robert W. Marcato Jr. Optimizing an Inverse Warper. Master's of Engineering Thesis, Massachusetts Institute of Technology, 1998.
- [11] William R. Mark, Leonard McMillan and Gary Bishop. Post-Rendering 3D Warping. Proceedings of the 1997 Symposium on Interactive 3D Graphics, pages 7-16.
- [12] Nelson Max. Hierarchical Rendering of Trees from Precomputed Multi-Layer Z-Buffers. Rendering Techniques '96 (Proc. 7th Eurographics Workshop on Rendering), pages 165-174.
- [13] Leonard McMillan. A List-Priority Rendering Algorithm for Redisplaying Projected Surfaces. Technical Report 95-005, University of North Carolina at Chapel Hill, 1995.
- [14] Leonard McMillan and Gary Bishop. Plenoptic Modeling. In Proceedings of SIGGRAPH 1995, pages 39-46.
- [15] Leonard McMillan. An Image-Based Approach to Three-Dimensional Computer Graphics. Ph.D. Dissertation. Technical Report 97-013, University of North Carolina at Chapel Hill. 1997.
- [16] Nathan O'Brien. Rayshade - Il Redentore.
<http://www.fbe.unsw.edu.au/exhibits/rayshade/church/>
- [17] Gernot Schaufler and Wolfgang Stürzlinger. A Three-Dimensional Image Cache for Virtual Reality. In Proceedings of Eurographics '96, pages 227-236. August 1996.
- [18] Jonathan Shade, Dani Lischinski, David H. Salesin, Tony DeRose and John Snyder. Hierarchical Image Caching for Accelerated Walkthrough of Complex Environments. In Proceedings of SIGGRAPH 1996, pages 75-82.
- [19] Jonathan Shade, Steven Gortler, Li-wei He and Richard Szeliski. Layered Depth Images. In Proceedings of SIGGRAPH 1998, pages 231-242.
- [20] Lee Westover. SPLATTING: A Parallel, Feed-Forward Volume Rendering Algorithm. Ph.D. Dissertation. Technical Report 91-029, University of North Carolina at Chapel Hill. 1991.



Figure 6: One of the reference images.

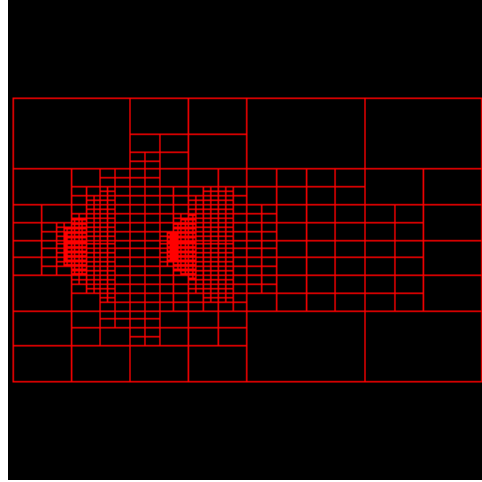


Figure 7: Top view of the octree cells after combining two reference images.



Figure 8: A new view generated from four reference images (at the same position).



Figure 9: A new view generated from 12 reference images (at three different positions).



Figure 10: A new view generated from 36 reference images (at 9 different positions).



Figure 11: A new view generated from 36 reference images. Gap filling is enabled.