Occlusion Culling for Image-Based Rendering with Warping

Soner I. Sen[1], Volkan Atalay[1]

[1] Department of Computer Engineering, Middle East Technical University, Ankara, Turkey

Abstract
The basic idea of the proposed algorithm is to reduce the number of depth pixels in Layer Depth Images (LDI) by culling the occluded ones before warping. The method combines the octree spatial subdivision concept with LDI concept, using the implicit geometry stored in depth pixels. The algorithm uses an octree to group depth pixels in a hierarchical way. The proposed algorithm is very effective when used in highly occluded scenes, and the density of the depth pixels should be high enough to get a high performance from the algorithm

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation - Display algorithms; I.3.7 [Computer Graphics]: Three Dimensional Graphics and Realism - Hidden surface removal; I.3.6 [Computer Graphics]: Methodology and Techniques - Graphics data structures and data types

1. Introduction

The fundamental problem of computer graphics is the desire for photorealistic rendering. Although this problem has been resolved for image generation, there is still a large open issue in the area of interactive computer graphics. Dramatic increases in hardware capabilities make little difference as these increases cannot reach a level high enough to resolve this issue via traditional methods; this is why researchers are so interested in this problem.

The traditional methods use 3D geometry for input, and they make approximations while creating the output image. Unless these assumptions are removed from the algorithms, or the error rate is reduced, the output image will not reach the desired photorealistic quality. With traditional methods the solution is to increase the complexity of a scene. It is not possible to render the entire scene at an interactive frame rate where the scene is large, such as huge city models and highly detailed terrains.

To achieve a display of highly complex models at interactive frame rates the approach taken should reduce the scene complexity without loss in the quality perceived by human visualization system. Dividing huge scenes into smaller parts, called portals, and rendering only visible portions is one way of reducing the scene complexity; this method requires a static environment and preprocessing. Another way is to simplify the objects in the scene; this depends on an error metric calculation with respect to the current point-of-view and the objects location. Mesh simplification can be done by storing different levels of details, or on the fly by mesh simplification algorithms [1]. Another method to simplify the complexity of a scene is by eliminating unseen primitives before the rendering pipeline, this is known as "culling techniques" [2]. Back-face culling, view-frustum culling and occlusion culling techniques are the most widely used ones.

Another solution to obtain the desired photorealistic view is via developing new methods for rendering which can remove the approximation steps in traditional rendering methods. The first attempt is to use real photographs with the input models, namely texture mapping. It is the remapping of an image onto a surface residing in the three dimensional scene. Texture mapping still uses some approximations in the rendering stage, wherein lies the aliasing problem. The other problem with this approach is that the rendering speed still depends on the surface the texture is applied to.

Another solution called Image-Based Rendering with Warping (IBRW) is proposed by Leonard McMillan [3]. IBRW produces photorealistic views since it uses real photographs with depth values as input. Images used here are called depth images and each pixel has a color and a depth value associated with it. This technique is independent of scene complexity; but there is a dependency on the resolution of the rendering view itself. McMillan uses a single photograph and IBRW has problems depending on the occluded parts of the scene in the reference image.

A solution to the gaps in the desired image due to the change in visibility is given by Shade et. al [4], by introducing a new concept Layered Depth Images (LDI). LDI can contain multiple depth pixels in each discrete location in the image. A layered depth pixel stores a set of depth pixels along one line of sight in front-to-back order. They can be created artificially by modifying a ray tracer or composing more than one images with depth.

The IBRW has three main stages according to Voicu Popescu [5]. The first stage is finding depth pixels that are visible from the desired view, then the second one is the projection of these depth pixels into the desired view, and the last stage is the reconstruction of the desired view. The first and last stages have no or unsatisfactory solutions, whereas the second stage has an elegant solution proposed by McMillan [3, 18]. The rendering speed of IBRW depends on the number of depth pixels that are input to the second and third stages. In order to obtain interactive frame rates this number should be as small as possible, while being large enough to provide a photorealistic display.

This paper introduces a method to increase the speed in IBRW by reducing the number of depth pixels. The proposed algorithm is a potential solution to the first stage of IBRW. The algorithm uses an octree spatial subdivision of the type commonly used to accelerate ray tracing to create hierarchical groups of classified depth pixels. The bounding values of octree nodes are then checked to see whether the inside depth pixels are visible or not from the desired view.

The remaining part of this paper organized as follows: section 2 is a brief review of previous work done on visibility culling and IBRW, section 3 is a detailed description and outline of the proposed algorithm, section 4 describes the work in progress and section 5 discusses the proposed algorithm.

2. Previous Work

There have been many attempts to obtain the interactivity on highly complex scenes. The common idea of each work is reducing the scene complexity. In this section we first give works done to reduce the primitive count by eliminating the invisible before pipeline, and then we give works done on Image-Based Rendering technique.

The basic idea of reducing the complexity of large scenes is to model the objects of the scene with different levels of detail (LOD). During rendering, the appropriate LOD of the objects of the scene is chosen according to a view-dependent error metric. These methods depend on the limitation of the perception ability of the human visualization system to details. Different LODs can be stored or they can be generated at runtime by mesh simplification algorithms [1].

Another way of reducing the scene complexity is to cull the primitives that cannot be seen from the current viewpoint before the rendering pipeline. Visibility culling methods classify the space into several groups and they use the classification property of each group to determine whether it is appropriate for rendering or not. Backface culling methods [6, 7, 8] avoid rendering of primitives that are not facing to point-of-view. The primitives that are outside of the view-frustum are eliminated by view-frustum culling methods [6, 9, 10]. In comparison of these two culling methods, occlusion culling involves a far more challenging problem.

The optimal occlusion culling method should select only visible primitives, kind of similar to how a z-buffer selects and renders [2]. The idea of occlusion culling methods is selecting and eliminating invisible primitives before sending to the rendering pipeline. Occlusion culling methods are mostly defined for static environments. In static environments, in the preprocessing step, some objects are marked as occluders and the occluded objects are the ones that are in the shadow frustum (occlusion volume) constructed for each of the occluders [11, 12]. For handling of dynamic three dimensional scenes, the method should consider the properties of both the image and the 3D space. Hierarchical Z-Buffer (HZB) [10, 13], and Hierarchical Occlusion Map (HOM) [14] are two kinds of these methods. HZB uses a pyramid of Z-buffers and an octree to classify the scene and remove large parts of the scene with small comparisons. Octree spatial subdivision is used for object-space coherence, Z pyramid for image-space coherence and a list of previously visible octree nodes for temporal coherence [13]. HZB depends on a special hardware since it requires reading back the z-buffer data which is too slow or cannot be done with most of the graphics accelerators. HOM works like HZB, but it is more conservative and requires the precomputation of an occluder database. It has a pyramid of maps like the Z pyramid which contains opacity values instead of depth values.

Wand et. al. [15] define Randomized Z-buffer algorithm for interactive rendering of highly complex environments. The main idea is to reconstruct the image from dynamically chosen surface sample points. They classify scene primitives with octree spatial scene partitioning with respect to their projection factors, generally distance is enough but for better solutions the orientation of primitives should be considered. For reconstruction of the image, first the occluded sample points should be discarded then the image is obtained from interpolation between the visible sample points. The same process can be done by conventional z-buffer rendering in one single step, where the sample density is high enough so that all the pixels of the image are covered by a sample point from a foreground object.

Image-based rendering is one of the approaches used for reducing scene complexity in interactive applications. The very first attempt using real images to reduce scene complexity is texture mapping and sprites. Chen and Williams

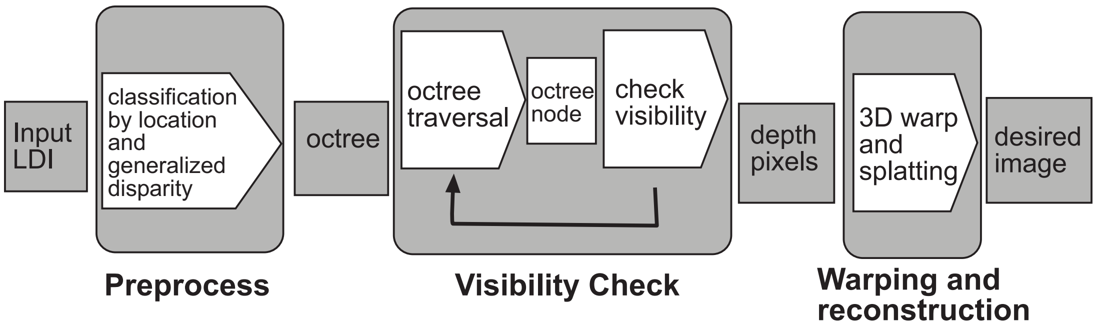

Figure 1: *Overview of the described algorithm*

go one step further in using of real images and they present the idea of rendering from images [16]. Their method uses similar methods of image morphing.

Leonard McMillan and Bishop define the plenoptic function as a complete spherical environmental map from a specific viewpoint at a given time [17]. They introduce a new IBR method with reprojection of depth pixels, 3D warping, Image-Based Rendering with Warping (IBRW). This method relies on the occlusion compatible ordering of depth pixels [17, 18]. Seitz and Dyer [19] describe a simple extension to creation of new views called view morphing, which allows image morphing methods to easily synthesize the changes in viewpoints and 3D effects with the help of user interaction.

IBRW uses a single depth image taken from single viewpoint therefore it cannot reconstruct the parts of the scene which can be seen from desired viewpoint but cannot be seen from the reference viewpoint. Mark et. al. [20] describe the use of multiple input images and creates triangulated depth maps. Shade et. al. [4] introduces Layered Depth Image (LDI) concept, which combines multiple depth images into one data structure.

The rendering quality of the IBRW techniques depends on the reconstruction step. The first attemp and the cheapest method is splatting [3, 20, 21] where the shape and the size of the projected depth and color pixels are approximated by splats. The problem of splatting is the loss of details when the viewer is relatively close to depth pixels where they are taken. A solution for this problem is to store depth images at several levels of detail, and at render time to use the appropriate level to make the projected depth pixels cover only one pixel of the desired image [22]. Another technique is the use of microtriangles. Two microtriangles connect four neighboring depth pixels to make a 3D mesh. The relief textures gives a completely different solution to the reconstruction problem [23]. This method computes an intermediate image of the original image with the desired center of projection, then simply texture maps this newly computed image. The generation of the intermediate image is by factoring the 3D warp into a pre-warp.

The speed of IBRW does not depend on the scene complexity; to speed up IBRW the number of depth pixels should be reduced. Popescu et. al. [24] use a space clipping method to eliminate the unseen columns (because of the horizontal field of view) of the LDI before warping. To do this they built a binary tree recursively, where the root has the maximum and minimum disparities of the entire LDI, and at each step LDI is divided into two equal parts, with a vertical line. While warping, the unseen columns of the LDI are eliminated by a recursive clipping algorithm using the maximum and minimum disparity values stored in the binary tree.

Chun-Fa Chang et. al. combine the hierarchical space partitioning with the LDI concept and introduce LDI tree [22]. LDI tree consists of multiple reference images and preserves their sampling rate by adaptively selecting an LDI in the LDI tree for each pixel. It is an octree which has a LDI at each cell, where each cell has only depth pixels in the bounding box. LDI tree speeds up the rendering time of LDI by not warping the every pixel of a reference image taken near an object when the object is viewed from far away and vice versa.

Chu-Fei Chang et. al. presented a hierarchical image-based rendering method, multi-level image-based rendering, to achieve the progressive refinement by using different resolution images [25]. This method depends on the limitation of the perception ability of human visualization system to details when the object is relatively far away.

Stoev et.al. reduce the number of depth pixels in the LDI by introducing a new image-based data structure multi LDI consisting of several small LDIs instead of one large LDI covering the entire range of view directions [26]. The work of Stoev et. al. differs from the approach described here by the time of depth pixel reducing job. Multi LDI does its job in preprocessing whereas our algorithm does its work on the fly.

Popescu and Lastra introduce vacuum buffer algorithm to use as a new depth pixel selection algorithm for IBRW [27]. This algorithm estimates potentially missed surfaces and depth pixels that surfaces might be located. It uses the empty space information of depth images previously used for building polygonal based models from range data. It is not useful for existing graphics hardware since it requires an extended z-buffer which can store several z-spans for each location.

The algorithm presented here combines the octree spatial subdivision concept with LDI [4] concept, using the implicit geometry stored in depth pixels [3].

3. Our proposal in detail

The basic idea of the proposed algorithm is to reduce the number of depth pixels in LDIs by culling the occluded ones before warping. The algorithm uses an octree to group the depth pixels in a hierarchical way. Then it simply uses the inverse of McMillan's visibility ordering [18] while creating new views to generate the new pixels from front-to-back order. It does the culling of large amount of depth pixels by only comparing the bounding values of octree nodes.

An overview of the algorithm is given in Figure 1. The algorithm consists of three main steps: octree construction, visibility checking, and reconstruction of the depth pixels. In the preprocessing step, the LDI is subdivided into a hierarchy of groups of depth pixels which show similar location in LDI and generalized disparity. After creation of octree structure, the octree is traversed in the inverse of McMillan's occlusion compatible order. Each visited octree node is checked whether it is visible or not. If the node is visible, the associated depth pixels are warped to desired viewpoint. For the reconstruction step, splatting technique is used, because of its simplicity and speed.

3.1. Octree Construction

In ray tracing and volume data rendering approaches octree data structure is used to accelerate the process. This data structure is very effective since it divides the space adaptively according to the scene properties.

In the octree construction step, in preprocessing, the algorithm treats the LDI as if it is a 3D space. The resolution of the LDI is used as x and y axis, for width and height respectively. The generalized disparity values of depth pixels are z values, since depth images encode the depth information as generalized disparities [3]. As an example, the root node has values of width, height, the maximum generalized disparity as the greatest corner, and 0, 0, the minimum generalized disparity as the lowest corner. Table 1 gives the pseudo code representing the data structure:

The octree data structure construction procedure is simple and recursive. The pseudo code of the algorithm is given in Table 2. It begins with the root node enclosing all the LDI and the list of entire depth pixels. Then recursively the following steps are performed starting from the root node: if the number of depth pixels within the octree node or the size of the node is sufficiently small, associate the location of the area of LDI and the count of depth pixels to the node and exit. Otherwise, associate the area and the count of depth pixels, subdivide the octree node, and call the procedure recursively for each of the eight children of the node.

```
OctreeNode =
        BoundingBox[X..Z, min..max] : array of real
        Children[0..7] : array of pointer to OctreeNode
        Count : real
        AreaInLDI[X..Y, min..max] : array of integer
```

Table 1: *Octree Data Structure*

Algorithm 1: Octree Construction Procedure

```
count = number of depth pixels
if count < COUNT_THRESHOLD or
     size < SIZE_THRESHOLD then
  associate depth pixels, area and the count to the node
  exit
else
  associate area and count to the node
end_if
compute child nodes recursively
```

Table 2: *Pseudo code of octree construction procedure*

The ordering of children nodes should be in such a way that the first 4 nodes are the closest ones to the image plane. The ordering can be seen in Figure 2. The reason to order children nodes is to ease the traversal of the octree.

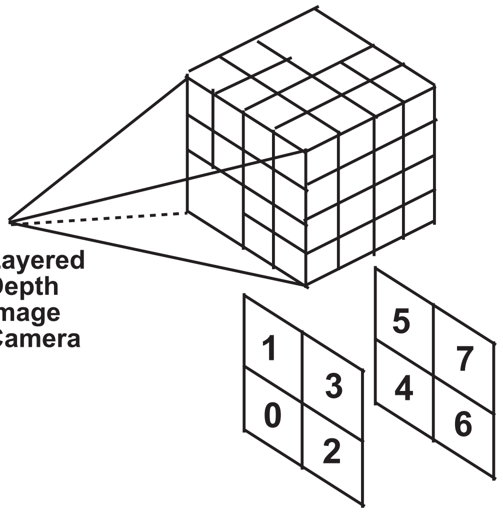

Figure 2: *Order of the octree children nodes*

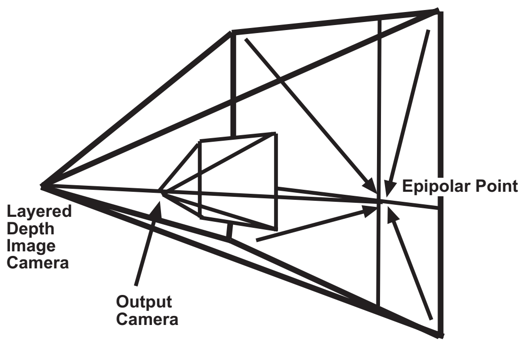

Figure 3: *Occlusion compatible order of warping LDIs (adapted from [4])*

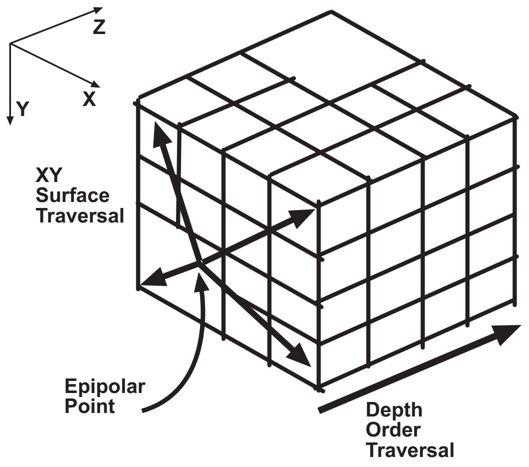

Figure 4: *Traversal of the octree*

3.2. Visibility Check

McMillan reduces the task of determining the visibility to a sorting problem and describes an ordering of depth pixels in a depth image [18]. The occlusion compatible ordering of McMillan works for LDIs if layers are warped in back-to-front order [28], as shown in Figure 3, an informal proof of this ordering is given in [24]. The algorithm presented here uses this ordering scheme; however the inverse of this ordering should be considered to gain from culling.

3.2.1. Travelsal of Octree

Before the traversal, algorithm finds the epipolar point, the projection of the desired viewpoint on to the image plane of reference viewpoint and determines the octree node which encloses the epipolar point, we call this node as the epipolar node. The nodes that include any portion of the isometric lines [18] are marked as isometric nodes. The octree nodes that are candidate to be epipolar or isometric node are the leaf nodes that have the minimum generalized disparity value of the LDI as a boundary value.

The traversal of the octree nodes starts with epipolar node and then it continues in two ways: to the direction of z axis (generalized disparity axis), and on the xy surface, as shown in Figure 4.

The XY surface traversal of the octree guarantees the inverse of McMillan's visibility order. This traversal direction uses the organization of the octree children ordering as a guide. For each visited node, the next node is found by the help of this organization. This traversal direction differs for each of the sheet that isometric lines partition the LDI. We start with the epipolar node, and then we check appropriate sibling nodes. After all appropriate siblings are visited, we go one level upper in the octree, and do the same process for the current node until we reach to the root node. A pseudo code of this traversal direction algorithm is given in Table 3.

After a visible node is reached while traversing the octree in xy surface direction, the nodes that are behind the current node need to be warped to achieve the inverse ordering algorithm of Shade et. al. [4]. To achieve this goal, we need to traverse the octree in detph order. In occlusion compatible order, depth pixels are warped from back to front order, we go from front to back to warp front pixels of the desired image before back pixels.

Octree nodes are organized such that we can easily find the sibling lying behind, as shown in Figure 2. However, after all appropriate siblings lying behind are considered we need to find the octree node, which is exactly the first from the ones behind. We use a ray which starts from a corner of the current node, and has a direction parallel to z axis. The corner of the node is chosen from the 4 corners having maximum generalized disparity value with respect to the sheet we are considering. We advance on the ray by a length of the half of the z-size of the smallest octree node. Therefore, we guarantee that we never miss a node and we reach to the first of the nodes lying behind.

If the level of the reached node is smaller, in other words it is closer to the root node; we only check the visibility of the node and mark it if it is not visible. If it is higher; we warp the current node and traverse its siblings in z direction and then in xy surface direction until we reach to the same level. If it is same; we simply warp the node and continue to the depth order traversal. We only warp the nodes that have same or higher levels in octree, since nodes with smaller level values occupy bigger area in LDI and the remaining parts may not be rendered yet.

3.2.2. Culling

While we are traversing octree nodes we first check the visibility of nodes by simply warping their boundary values.

We keep a buffer, occlusion buffer, with the size of LDI

Algorithm 2: Octree XY surface traversal

```
find the epipolar node
mark all the isometric nodes
for each of the sheets do
  current = epipolar node
  while current != root node do
    if current is marked then
      current = getNextSibling()
      if current == NULL then
        current = getParent()
        DepthTraverse(current)
      end_if
    else
      if visible then
        if current is leaf node then
          MarkAndWarp(current)
          DepthTraverse(current)
          current = getNextSibling()
        else
          current = getFirstChild(current)
        end_if
      end_if
    end_if
  end_while
end_for
```

Table 3: *Pseudo code of octree XY surface traversal algorithm*

resolution to determine whether the pixel is a background pixel or not. Occlusion buffer has all 0s initially which shows a background pixel. We add 1 to the exact location of the warped depth pixel, and add a weighted value to locations of the splats [20]. To decide whether the node is occluded or not, boundary values are warped, and a rectangular bounding box is computed for the eight resulting points. If no value other than 1 inside of the computed bounding box exists, then this node is invisible and it is safe to not warp associated depth pixels.

3.3. Warping and Reconstruction

After finding an octree node visible from the desired viewpoint, we simply warp the associated depth pixels. We use the warping equations given in [3, 5].

Splatting technique is used for reconstruction of the desired image from the warped depth pixels. The exact location of the warped depth pixel is opaque and the other parts are semi-transparent where as going to the edges of the splats it becomes more semi-transparent. Computing the precise size and shape of the splat for a depth pixel is very difficult (expensive); therefore we approximate the size by comparing the generelized disparity values in reference image and desired image, and for the shape we use a rectangle. We prefer rectangle as splat shape, since rendering of rectangles can be accelerated by some of graphics hardware.

4. Work in Progress

In this paper, we present an algorithm of occlusion culling for image-based rendering with warping. The method combines the octree spatial subdivision concept with LDI concept, using the implicit geometry stored in depth pixels.

The work on this method is in progress now. Currently, we are implementing the visibility checking step of the algorithm. The other steps are implemented. The simple IBRW is also implemented as given in [5] for comparing the results of our method. All the implementation of algorithm is done on a simple PC. C++ and DirectX 8.0 are used for programming language. The LDI creation is done by ray-mesh intersection built-in function of DirectX. We are now using LDIs with a simple object, which is constructed by 12 depth images taken from surrounding of the object.

We are planning to test the performance of the algorithm on two types of scenes: densely and coarsely occluded scenes. Examples of these scenes are given in Figure 5. We will check the performance of the algorithm by comparing how many percent of the depth pixels are eliminated on two types of scenes. Furthermore, the percentage of the faults, treating an invisible pixel as visible and vice versa, will be our error metric. These faults can be results of splatting; therefore the visual quality of the desired image will be our main consideration to test the rendering quality of the algorithm.

5. Discussion

Most of the work done on the IBRW is to make the quality of the desired image better, however in order to get a place in interactive computer graphics IBRW needs to be accelerated. There has been a trade off between the speed and quality for all of the approaches in the computer graphics area.

The proposed algorithm carries a concept used for accelerating the traditional geometrical rendering to the IBRW. The simple IBRW does not consider the visibility since McMillan gives an elegant solution to the visibility problem [18]. We try to stick to this ordering while we are traversing the octree structure; however there may be some faults in the order. These faults may arise due to the structure of the octree. The nodes of the octree need not to be the same in size. When a bigger node lies behind the smaller node, the occlusion compatible order of McMillan may not be preserved. However we think these faults are negligible, if we keep the associated depth pixel count stop condition of the octree construction process small enough. The faults in the desired image will not be remarkable by human visualization system, since we use splatting with alpha blending in the reconstruction step.

The proposed algorithm is very effective when used in highly occluded scenes, such as the scene given in Figure 5 (a). The density of the depth pixels should be high to get a high performance from the algorithm.

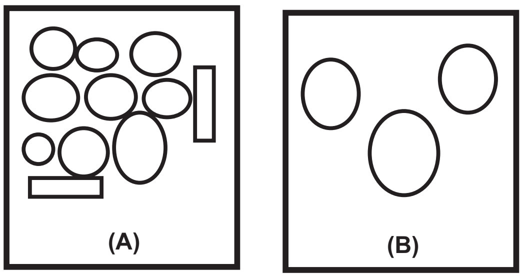

Figure 5: *Sample test scenes top views (a) Highly occluded (b) Coarsely occluded*

References

1. E. Puppo and R. Scopigno. Simplification, LOD, and multiresolution principlas and applications. *Eurographics'97 Turorial Notes*, 1997. 1, 2

2. T. Möller and E. Haines. *Real Time Rendering*. A. K. Peters Ltd., 1999 (Chapter 7). 1, 2

3. L. McMillan. An image-based approach to computer graphics. PhD Dissertation *Technical Report TR97-013, University of North Carolina at Chapel Hill*. 1, 2, 3, 4, 6

4. J. Shade, S. Gortler, L. He, and R. Szeliski. Layered depth images. *SIGGRAPH'98, (computer Graphics Proc.)*, :231, 1998. 2, 3, 4, 5

5. V. Popescu. Forward rasterization: A reconstruction algorithm for image-based rendering. PhD Dissertation *Technical Report TR01-019, University of North Carolina at Chapell Hill*, 2001. 2, 6

6. F. D. Foley, A. van Dam, S. K. Reiner, and J. F. Hughes. *Computer Graphics: principles and practice*. Addison-Wesley Publishing Co., 1990. 2

7. S. Kumar, D. Manocha, B. Garett, and M. Lin. Hierarchical back-face computation. *Eurographics'96, Workshop on Rendering*, :67, 1996. 2

8. H. Zhang and K. E. Hoff III. Fast back-face culling using normal masks. *Symposium on Interactive 3D Graphics '97* :103, 1997. 2

9. U. Assarsson and T. Möller. Optimized view frustum culling algorithms for bounding boxes *Journal of Graphics Tools* **5**(1):9-22, 2000. 2

10. Ned Greene. Occlusion culling with optimized z-buffering. *SIGGRAPH '01 Course Notes* , 2001. 2

11. T. Hudson, d. Manocha, J. Cohen, B. Garett, M. Lin, and K. Hoff. Accelerated occlusion culling using shadow frusta. *13th Annual ACM Symposium on Computational Geometry Proc.*, :1, 1997. 2

12. J. Bittner, V. Havran, and P. Slavik. Hierarchical visibility culling with occlusion trees. *Computer Graphics International '98 Proc.*, :207, 1998. 2

13. N. Greene, M. Kass, and G. Miller. Hierarchical Z-Buffer visibility. *SIGGRAPH '93 Proc*, :231, 1993.

14. H. Zhang, D. Manocha, T. Hudson, and K. Hoff. Visibility culling using hierarchical occlusion maps. *SIGGRAPH '97 Proc.*, :77, 1997. 2 2

15. M. Wand, M. Fischer, I. Peter, F. M. auf der Heide, and W. Strasser. The randomized z-buffer algorithm: Interactive rendering of highly complex scenes. *SIGGRAPH '01 (Computer Graphics Proc)*, :361, 2001. 2

16. S. E. Chen and L. Williams. View Interpolation for image synthesis. *SIGGRAPH '93 Proc.*, :279, 1993. 3

17. L. McMillan and G. Bishop. Plenoptic modeling: an image-based rendering system. *SIGGRAPH '95 Proc.* :39, 1995. 3

18. L. McMillan. Computing visibility without depth. *Technical Report TR95-047, University of North Carolina at Chapel Hill*, 1995. 2, 3, 4, 5, 6

19. S. Seitz and C. Dyer. View Morphing: synthesizing 3D metamorphoses using image transforms. *SIGGRAPH '96 Proc.*, :21, 1996. 3

20. W. R. Mark, L. McMillan, and G. Bishop. Post-rendering 3D warping. *Interactive 3D Graphics '97 Proc.*, :7, 1997. 3, 6

21. W. Mark. Post-rendering 3D image warping: visibility, reconstruction, and performance for depth-image warping. *PhD Dissertation. University of North Carolina at Chapel Hill*, 1999. 3

22. C. Chang, G. Bishop, and A. Lastra. LDI Tree: A Hierarchical Representation for Image-based Rendering. *SIGGRAPH '99 Proc.* :291, 1999. 3

23. M.Oliveira, G. Bishop, and D. McAllister. Relief texture mapping. *SIGGRAPH '00 Proc.*, :359, 2000. 3

24. V. Popescu, A. Lastra, D. Aliaga, M. Oliveira. Efficient Warping for Architectural Walkthroughs using Layered Depth Images. *IEEE Visualization '98 Proc.*, :211, 1998. 3, 5

25. C. Chang, A. Varshney, and Q. J. Ge. Hierarchical Image-based and Polygon-based Rendering for Large-Scale Visualizations. 3

26. S.L. Stoev, I. Peter, and W. Strasser. The multi LDI: an image-based rendering approach for interaction, navigation, and visualization in complex virtual environments. *Technical Report WSI-2000-23, University of Tuebingen*, 2000. 3

27. V. Popescu, and A. Lastra. The vacuum buffer. *Interactive 3D Graphics '01 Proc.*, :73, 2001. 3

28. S. J. Gortler, L. He, and M. F. Cohen. Rendering layered depth images. *Technical Report, MSTR-TR-97-09*, 1997. 5