The WarpEngine: An Architecture for the Post-Polygonal Age

Voicu Popescu, John Eyles, Anselmo Lastra, Joshua Steinhurst, Nick England, Lars Nyland
University of North Carolina at Chapel Hill

ABSTRACT

We present the WarpEngine, an architecture designed for real-time image-based rendering of natural scenes from arbitrary viewpoints. The modeling primitives are real-world images with per-pixel depth. Currently they are acquired and stored off-line; in the near future real-time depth-image acquisition will be possible, and WarpEngine is designed to render in immediate mode from such data sources.

The depth-image resolution is locally adapted by interpolation to match the resolution of the output image. 3D warping can occur either before or after the interpolation; the resulting warped/interpolated samples are forward-mapped into a warp buffer, with the precise locations recorded using an offset. Warping processors are integrated on-chip with the warp buffer, allowing efficient, scalable implementation of very high performance systems. Each chip will be able to process 100 million samples per second and provide 4.8GigaBytes per second of bandwidth to the warp buffer.

The WarpEngine is significantly less complex than our previous efforts, incorporating only a single ASIC design. Small configurations can be packaged as a PC add-in card, while larger deskside configurations will provide HDTV resolutions at 50 Hz, enabling radical new applications such as 3D television.

WarpEngine will be highly programmable, facilitating use as a test-bed for experimental IBR algorithms.

KEYWORDS: Graphics hardware, image-based rendering.

1 INTRODUCTION

Research efforts in interactive 3D computer-graphics have been targeted at providing high-quality, high-resolution images. This goal has proven elusive: renderings that can be mistaken for photographs can usually be obtained only by sacrificing interactivity.

This problem, and the extreme difficulty of modeling natural environments, motivated research on image-based rendering. Although the image-based primitives are novel, conventional polygon-based graphics hardware has been used for the rendering. Few attempts have been made to take advantage of the new image-based primitives with novel hardware.

{popescu, jge, lastra, jsteinhu, nick, nyland}@cs.unc.edu
Sitterson Hall, CB#3175, CS UNC, Chapel Hill, 27599, NC

In this paper, we present the WarpEngine architecture for rendering directly from an image-based representation, specifically from images with per-pixel depth [McMillan95]. The prototype that we plan to build promises high performance at HDTV resolution, as well as extensive programmability to support research in algorithms for image-based rendering.

We first review related work in image-based rendering (and graphics hardware that uses image-based primitives). Then we present the algorithm on which the WarpEngine is based, followed by a detailed architectural description of the machine. We close with proposed future work and conclusions.

1.1 Related Work

The spectrum of image-based approaches ranges from those that exclusively use images to those that re-project acquired imagery onto geometric models.

[Chen95] employed 360° panoramas, stitched together from overlapping photographs. A panorama offers a realistic view of the scene, but the user has a correct 3D perspective only from a single location. However, using inexpensive hardware, the user can view, at interactive rates, outdoor or indoor scenes that are hard to model as a collection of polygons.

Image morphing approaches [Wolberg90, Beier92, Chen93, Seitz96] allow some range of motion but the transformation of the reference views to the desired views is *approximated* by interpolation. To maintain a high update rate, the Talisman architecture [Torborg96] reused portions of rendered images by re-projecting to a new, nearby, view using a 2D warp.

The Lumigraph [Gortler96] and Light Field [Levoy96] densely sample light rays to create a ray database. Unfortunately, the database of rays grows quite large for bigger viewing volumes. [Regan99] describes low-latency rendering hardware for a one-axis light field.

At the other end of the spectrum are methods based largely on geometry. Texture mapping is the most common way to incorporate images in the scene description. The *Façade* system [Debevec96] represents the scene with an approximate geometric model (semi-automatically created) texture-mapped, in a view-dependent fashion, from photographs.

McMillan and Bishop's [McMillan95] method is in the middle of the spectrum, representing the scene as a collection of images that in addition to color also store depth at each pixel. The desired view is generated by a 3D-warp of the depth images.

We have chosen image-based rendering by 3D warping (IBRW) as the basis for the WarpEngine. There are two reasons for this: (1) the storage and bandwidth requirements are manageable, and (2) laser rangefinders [K2T, Beraldin92, Cyra] and other range-acquisition equipment [Kanade99, Minolta] are rapidly improving and appearing on the commercial market. Some instruments can even acquire range in real time. Such a "depth camera" coupled with the WarpEngine will enable extraordinary applications: a spectator of a live event will not be confined to the view of the TV camera; he or she can choose any seat in the arena, and even venture onto the stage or court.

1.2 3D Image Warping

McMillan and Bishop show in [McMillan95] how to compute the desired image coordinates of a depth image sample using the 3D warping equations

$$u_2 = \frac{w_{11} + w_{12} \cdot u_1 + w_{13} \cdot v_1 + w_{14} \cdot d(u_1, v_1)}{w_{31} + w_{32} \cdot u_1 + w_{33} \cdot v_1 + w_{34} \cdot d(u_1, v_1)}$$
$$v_2 = \frac{w_{21} + w_{22} \cdot u_1 + w_{23} \cdot v_1 + w_{24} \cdot d(u_1, v_1)}{w_{31} + w_{32} \cdot u_1 + w_{33} \cdot v_1 + w_{34} \cdot d(u_1, v_1)} \quad (1)$$

where u_2, v_2 are the desired image coordinates, u_1, v_1 the original (reference) image coordinates, the w's are transformation constants obtained from the reference and desired image camera parameters, and $d(u_1, v_1)$ is the generalized disparity at sample (u_1, v_1), which is defined as the ratio between the distance to the reference image plane and $z_{eye}(u_1, v_1)$.

The warping equation is equivalent to the vertex transformation commonly used in computer graphics, but allows one to take advantage of the regular structure of images to perform incremental transformation. The warped coordinates of a sample can be computed with six adds, five multiplies[1], and one divide.

Reconstructing by simply setting a desired image pixel to the color of the sample that warps within its boundary results in holes thus not acceptable. Also, more than one visible sample can warp to the same pixel, and simply discarding all but one sample produces aliasing. Reconstruction is a challenging task when warping images with depth; we analyze it in more detail next.

1.3 Reconstruction

The 3D warping equation is a forward mapping that takes samples from the reference domain and maps them to the destination domain. An inverse mapping (as in conventional texturing) would be ideal. Unfortunately there is no analytically computable inverse for 3D warping (there is a rather costly search procedure described in [McMillan97]).

Reconstruction for IBRW is mainly done in one of two ways [Mark97, McMillan97]: with splats or with a polygonal mesh.

A splat [Westover90] is a representation of the projected shape of the reference sample. The original use for splats was to render transparency for volume rendering; thus the splats were blended in front-to-back order. For IBRW, we do not want to blend samples that are at different depths. Rather we want to overwrite samples that should be hidden and only blend samples that represent the same surface. This is very difficult to do because in IBRW we have no information about surfaces. To prevent samples of hidden surfaces from showing through, the sizes of the splats are overestimated [Shade98], thus overlapping splats may incorrectly erase visible samples, resulting in aliasing.

Good reconstruction can be obtained by connecting the samples of the reference image into a polygonal mesh. Not all samples should be connected, of course; in Section 2.1 we present a simple method for detecting depth discontinuities. With meshing, continuity of the surfaces is maintained where desired, and hardware acceleration increases performance. On WarpEngine, we connect samples, but avoid the overhead of general polygonal rendering.

[1] If w_{34} is non-zero (non-zero translation from reference position) one could save a multiply by dividing all w's by w_{34}.

1.4 Why WarpEngine?

One might ask, why build the WarpEngine if existing graphics hardware can be used for IBRW? One reason is performance; another is efficiency.

Assume that 1280 by 1024 is the targeted resolution and that on average we warp twice the reference samples as the desired resolution. Two triangles need to be rendered for every warped sample. The average number of triangles per second to sustain a frame rate of 30 Hz is

$$N \approx 1280 \times 1024 \times 2 \times 2 \times 30 \approx 157 Mtris/s$$

Neither high-end systems like PixelFlow [Molnar92] or InfiniteReality [Montrym97], nor the rapidly improving PC 3D graphics accelerators, can produce the necessary performance. Moreover, we speculate that it will take years for them to reach this sustained level of performance. Even then, it will take more hardware than on a machine optimized for IBRW. We believe that WarpEngine is more efficient because it takes advantage of the regularity of image-based primitives and of the small screen-size of the warped samples.

More needs to be said to explain the number of reference-image samples required at each frame. This number depends on the scene and on how it is modeled. One must process (on average) more than one reference image sample per desired image location because:

- there are surfaces that are redundantly captured in more than one reference image;
- there are surfaces captured in the reference images that are not visible in the desired image (depth complexity is greater than one);
- there are surfaces that were better sampled in the reference image than in the desired image, which leads to more than one visible sample per desired image pixel.

Two input samples per output pixel is a reasonable lower bound; in practice we have found it difficult to use fewer. With real-time depth-image updates (immediate mode), the number of samples will be determined by the number, resolution and update rates of the cameras.

The most viable alternative to IBRW is to simplify triangle meshes in order to reduce the polygon count, and thus meet our performance goals with conventional graphics hardware (in fact, other members of our team are investigating this approach). However, simplified meshes are less well suited to real-time depth updates because of the pre-processing required.

2 RENDERING ALGORITHM

We wish to treat the depth image as connected (as in the mesh approach) in order to prevent samples of hidden surfaces from showing through. The triangles resulting from the mesh method are very small in screen space; thus scan conversion time is dominated by setup. Instead of conventional scan conversion, we propose simply bilinearly interpolating between connected samples in reference image domain, reducing *per-sample* setup.

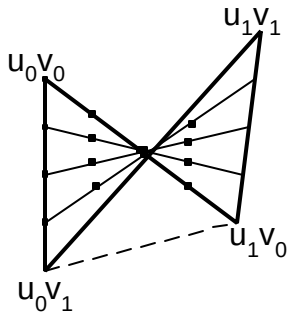

Figure 1. Once warped, the four neighboring reference-image samples u_0v_0, u_1v_0, u_1v_1, and u_0v_1 may form a concave quad. Interpolating in desired-image space produces the sub-samples shown with little squares, which is different from the projection of the surface on the image plane.

When interpolating in reference-image space and then warping, the surface projection is approximated better, but at higher computational cost.

The algorithm is:

For all adjacent, connected samples

Bilinearly interpolate color and depth to obtain subsamples

Warp resulting subsamples to desired image space

Z-composite warped subsamples into the warp buffer

In order to reduce aliasing, we warp into a sub-pixel resolution *warp buffer* (usually 2x2), then filter to produce the final image in the frame buffer. The sub-samples are z-buffered.

The interpolation factor (number of subsamples created in each of *x* and *y* directions) is critical in order to ensure that (1) back surfaces do not show through, and (2) we do not generate too many subsamples. We describe the computation of the interpolation factor in more detail in Section 3.3.

Recall that the reference-image depth information is stored as generalized disparity that is proportional to $1/z_{eye}$, which is linear in image space. Consequently, if the four neighboring samples are planar, the sub-samples resulting from the interpolation are correctly located on the same plane. If the samples are not coplanar, the sub-samples define a general bilinear patch. Adjacent patches exhibit C^0 continuity.

An alternative rendering algorithm, to save the cost of warping the sub-samples (dominated by the inverse computation, see Equation 1) is to *first* warp the reference-image samples and *then* interpolate. This still avoids the triangle setup costs. Just as when interpolation is done in reference-image space, if the original samples lie on a plane, the sub-samples are also on the plane.

Figure 2. The vertical and horizontal depth discontinuities are marked in green and red, respectively. The detection algorithm works well in spite of the great range of distances in the image.

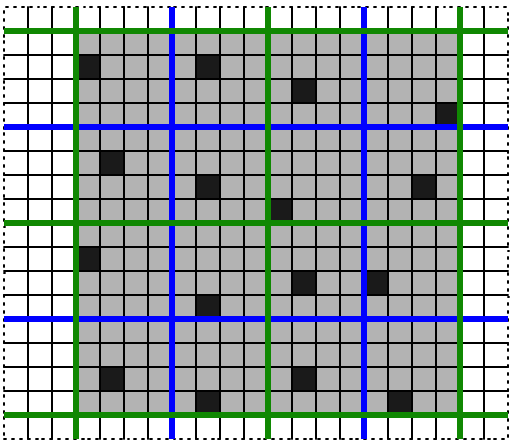

Figure 3. In the warp buffer fragment shown, the green lines define the warp buffer locations. The blue lines delimit the output pixels, which span four warp buffer locations each. The fine black lines show the virtual subdivision of the warp buffer locations corresponding to the two 2-bit offset values. The locations at which these samples warped are shown by black squares and are recorded to a precision of $1/8^{th}$ of a pixel. There is exactly one sample per warp buffer location.

The output pixel is reconstructed using a two-pixel wide kernel, with a half pixel (one warp buffer location) overlap. The kernel is shown in gray. The 16 color samples are weighted according to their position inside the warp buffer location, as modified by the offsets. The reconstruction is equivalent to reconstructing from an 8x8 supersampled buffer that is sparsely populated, without having to explicitly allocate the dense buffer or to search for the locations that are populated.

However the similarity between the two methods ends when the four original samples are not coplanar and the resulting screen-space quad might be concave (resulting in a "bow tie", see Figure 1). Our simulations show that this is a very infrequent case, which usually occurs between silhouette samples that were marked as disconnected anyway.

2.1 Determining Connectivity

We devised a robust and inexpensive way of detecting which samples should not be connected by interpolation, based on the surface curvature (see also [McAllister99]). At every reference image sample we compute the second derivative[2] of the generalized disparity along four directions: E-W, SE-NW, N-S, and SW-NE. If the surface sampled is planar the second derivative is exactly zero. If it exceeds a threshold (which is unique per scene) the samples are marked as disconnected (see Figure 2).

2.2 Reconstruction Using Offsets

We want to render high-quality, antialiased images. Conventional jittered supersampling is not an option because of the *forward-mapping* nature of the warping process; warping produces sub-samples that do not correspond exactly with the centers of warp-buffer locations. Even with a 2x2 warp buffer, aliasing is greater than we wish.

Our proposed alternative is to compute the *(x, y)* location of the warp to a precision higher than that of the warp buffer, and store that more precise location as an offset from the corner of the warp

[2] difference of neighboring differences

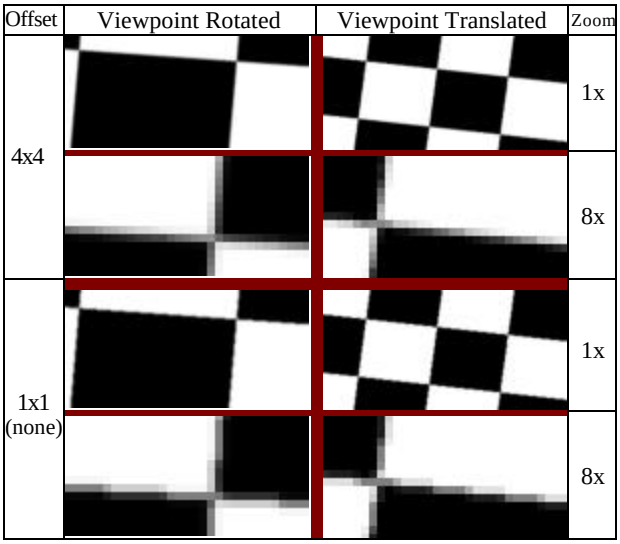

Figure 4. Antialiasing using offsets. These images were rendered from a depth image of a checkerboard. The left column is just rotated, the right also translated. We show both original and 8X zoomed versions. All images used a *2x2* warp buffer. The upper set was rendered with two bits for each of x and y offset. The lower set used no offsets and exhibits more aliasing.

buffer location. The *offsets* are used during reconstruction to obtain better filtering and a higher-quality final image. We have found that a 2-bit offset in each of *x* and *y* (total of 4 bits per warp buffer location) provides good results. This, combined with a 2x2 warp buffer, locates the warped subsamples to within one-eight of a pixel (see Figure 3). The results are illustrated in Figure 4.

Offsets are, of course, not equivalent to higher warp buffer resolution. Although its location is recorded more precisely, only one sample is stored at each warp buffer location. In the expected case, when the sampling resolution of the desired image is within a factor of two of that of the reference images, the 2x2 warp buffer with 4x4 offsets provides a good reconstruction. Outside that interval, other reference images should be used. One could increase the resolution of the warp buffer to accommodate even bigger sampling mismatches, but this comes at a substantial additional cost, not only in memory, but also in warping since more reference image samples must be used.

The offset reconstruction also has good temporal antialiasing properties. Antialiasing by jittered supersampling or coverage-mask-based methods suffer from the problem of collinear sampling locations within a pixel. No matter how the sampling locations are chosen, at least two of them are collinear in jittered supersampling and *k* are collinear when *kxk* subpixel masks are computed. If from one frame to the next all collinear sampling locations move from one side to the other of a slowly moving edge, the change in color of the output pixel is too abrupt. Using 2x2 pixel kernels with 2x2 warpbuffer and 4x4 offsets guarantees 16 intermediate levels (when an edge moves slowly enough). We refer the reader to the conference-proceedings videotape and DVD-ROM, which illustrate the dynamic antialiasing properties of our algorithm. Also the conference-proceedings CD-ROM includes the antialiasing examples. Please look at the images on the CD to see the full effects of the antialiasing. Figures 5, 6, and 10 (at the end of the paper) show images of various test scenes rendered with the WarpEngine simulator.

Figure 5. Eurotown images were made from reference depth images placed on a regular grid. Only the reference images of the current grid-cell were used to render each frame.

3 WarpEngine ARCHITECTURE

3.1 Overview

The hardware architecture must provide sufficient warping power for all required reference-image samples and sufficient bandwidth to the warp buffer.

We decided to partition the reference images into 16x16-sample *tiles* (with a 15x15 payload) and to use these as the basic rendering primitive. Tiles provide several important advantages:

- we can selectively use portions of reference images as needed for adequate sampling and coverage of visible surfaces (Section 3.3);
- one can easily estimate the screen area a tile transforms to, enabling efficient high-level parallelism (Section 3.1.3);
- tiles are small enough that the same interpolation factor can be used for all samples, enabling SIMD low-level parallelism (Section 3.1.1).

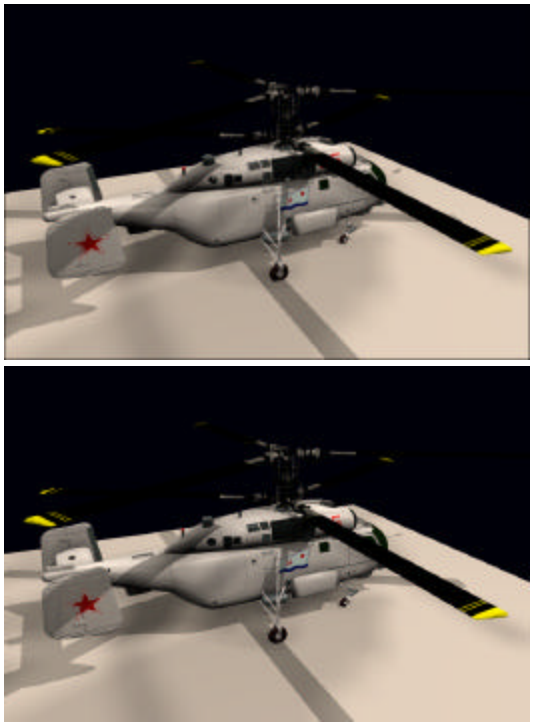

Figure 6. The upper image was rendered on the WarpEngine simulator. For comparison, the lower image was rendered directly from the geometric model.

3.1.1 Warping and Interpolation

All the samples of a tile can be warped and interpolated with the same set of instructions so a SIMD implementation is, we believe, the most efficient. We opted for an array of simple byte-wide processors, similar to the one used in PixelFlow [Molnar92]. For a computation that can be efficiently mapped, a SIMD array provides efficient use of silicon, since control is factored out over all the processors. A large array of simple processors is more easily programmable than a complex pipelined processor. The programmability is necessary for use of the WarpEngine as a research tool.

A SIMD array equal in size to the reference-image-tile maps very efficiently, since the warping calculation is the same for every pixel, with minimal branching required. Nearest neighbor Processing Element (PE) connectivity provides each PE with access to the three other samples needed for interpolation.

3.1.2 Warp Buffer

The biggest design concern was providing sufficient warp buffer bandwidth. We assume the maximum resolution to be HDTV (approximately 2K x 1K pixels) and 60 Hz update rate; we assume again that one needs to use at least two reference-image samples per output pixel. This implies that at least 240 million reference samples per second must be warped. In our simulations, a 2x2 warp buffer resolution required in some cases an average interpolation factor of 4x4. Thus, for each warped reference-image pixel, 16 warped samples are generated, and the warp buffer must process approximately 4 billion warped samples per second. Each sample is about 12 bytes in size (4 bytes RGB; 4 bytes Z-buffer; 4 bytes X and Y values, including offsets). Assuming a depth complexity of two, and that 50% of the hidden samples initially pass the Z-comparison test, an average of 10 byte accesses is required per warped sample. Thus total warp buffer bandwidth is about 40 GigaBytes/sec.

To achieve this enormous warp-buffer bandwidth, a very large number of commodity DRAMs is required (well over 100); similarly, the warping/interpolation processors would require hundreds of pins dedicated to interfacing with the warp-buffer. By placing the warp buffer on-chip, that is, on the same ASIC as the processors that generate the warped samples, very wide and fast memory interfaces can be used.

3.1.3 Region-Based Rendering

With current technology, a single ASIC can provide neither sufficient processing power nor sufficient warp buffer memory[3]. Thus multiple ASICs are required, and some form of high-level parallelism must be employed. Partitioning the warp buffer into contiguous screen regions with each region assigned to an ASIC (screen-space subdivision) is appealing, because the typical 16x16-sample tile intersects only one screen region and therefore needs to be processed by a single ASIC (tiles that overlap region boundaries are assigned to multiple regions). By contrast, with interleaving, each tile would need to be processed by many or all of the ASICs.

For partitioning by screen-space subdivision, primitives must be sorted by screen region[4]. Using tiles as the rendering primitive means that sorting is performed on 256 samples at a time; the number of tiles per frame ranges from a few thousand to a maximum of a few tens of thousands (depending on screen resolution) so the computational and memory burden of sorting is considerably less than for the general polygon-rendering case. By assigning multiple screen regions to each ASIC, a smaller number of ASICs is sufficient; however this requires sorting into buckets corresponding to screen regions [Ellsworth97], because an ASIC must process all primitives in a given region before moving to its next assigned region.

The sort first, sort middle, sort last taxonomy developed to describe object-parallel polygon-rendering architectures [Molnar94], can also be applied to IBR architectures. Sorting by reference-image tiles is sort first from the point of view of reference-image samples, since after a tile has been assigned to a screen region, it is known a priori that its sub-samples will warp to the desired screen region (those that do not can be discarded, since the tile will be assigned to all pertinent regions). In polygon rendering, sort first [Mueller95] is prone to load-balancing difficulties; this is not be a problem for IBR, since reference-image tiles and interpolation factors are chosen to sample the destination image uniformly. We believe that sort first is an attractive approach for the WarpEngine, because it makes scaling of the system relatively painless. Performance is increased by adding additional ASICs, and assigning fewer screen regions to each ASIC. Screen-space subdivision requires a central processor, perhaps the host, which can perform the tile sorting, or a way of distributing these tasks across the multiple ASICs.

3.1.4 Processing Warped Sub-Samples

It is straightforward to build a region-sized on-chip warp buffer with very high performance. Since each warped sample maps to only one location in the warp buffer, the warp buffer can be partitioned, with a *sample processor* assigned to each partition. Very high numbers of samples can be processed by instantiating more sample processors, processing simultaneous streams of warped samples. Load-balancing can be achieved by sub-pixel interleaving the partitions and providing input FIFOs for the sample processors. The region size is determined by the silicon budget for the warp buffer, independently of the number of partitions.

The sample processors are very simple: they combine a new warped sample with the previous contents of the warp buffer location, using a z-compare operation. Since the sample processors' memory interface does not cross chip boundaries, it can be very wide and very fast; thus the sample processors are not bandwidth limited.

3.2 WarpEngine Implementation

Our architecture, the *WarpEngine*, consists of one or more identical *Nodes* (typically 4 to 32); each Node consists of an ASIC and a Tile Cache. The ASIC contains:

- a 16x16 SIMD Warp Array, for warping and interpolating reference-image samples;

[3] As silicon technology improves, a full-sized warp buffer becomes feasible (on an embedded-DRAM process).

[4] For tiles, this is efficiently done by warping the 4 corners, using both the tile's minimum and maximum disparity values; the resulting 8 points define the tile's screen-space bounding box.

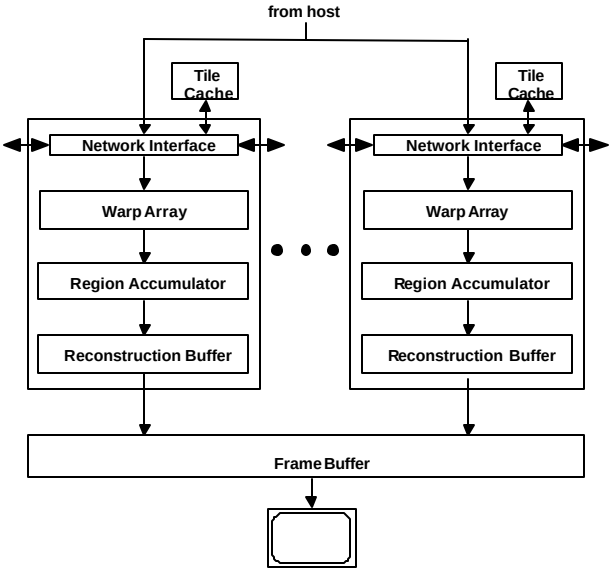

Figure 7. Block diagram of the WarpEngine

- a Region Accumulator, which includes a double-buffered warp buffer for a 128x128 screen region and 4 sample processors for resolving visibility;
- a Reconstruction Buffer, for computing final pixel values;
- a Network Interface, which connects the Nodes together into a high-bandwidth ring, and provides a connection to the host, a connection to each of the Warp Arrays, and a connection to the Tile Cache.

The Tile Cache is a commodity DRAM device; it is used for caching both reference-image tiles and instructions. A double-buffered Frame Buffer receives the final pixel values from the Nodes for display.

The basic operation of the system is as follows (see Figure 7):

- The host determines which reference-image tiles are to be used to compute the destination image, and computes the screen-space bounding box for each of these tiles. For each screen region, the host maintains a *bin*; each bin contains pointers to the tiles whose bounding boxes intersect that screen region.
- For each screen region, the host assigns a Node to be responsible for that screen region. The host sends each tile in the region's bin to the Node. (Tiles are cached in each node's Tile Cache. If a tile is resident in one of the caches, the host instructs the Network Interface to forward it to the appropriate Node. If not, the host must send the tile data to the Node).
- Each tile received by each Node is loaded into the Warp Array, which performs the warping and interpolation calculations for the tile, and forwards the warped samples to the Region Accumulator.
- The Region Accumulator collects the warped samples into its sub-pixel resolution warp buffer.
- After all tiles in the region's bin have been processed, the Region Accumulator swaps its buffers and initializes the visibility buffer, in preparation for processing the next screen region.
- Concurrently with processing the next screen region, the Region Accumulator steals memory cycles to send the previous region's data to the Reconstruction Buffer. The Reconstruction Buffer computes the final pixel values for the region and forwards them to the Frame Buffer.
- After all regions have been processed and the final pixel values calculated and forwarded to the Frame Buffer, the Frame Buffer swaps buffers.

The system can function in *retained* mode, in which there is a fixed set of reference images describing an environment, or *immediate* mode, in which new reference images are being received "on the fly".

3.2.1 Warp Array

The Warp Array (see Figure 8) consists of 256 processing elements (PEs), arranged as a 16x16-pixel array. Each PE consists of a simple byte-wide ALU and 160 bytes of local memory partitioned as: 128 bytes main memory, 16 bytes IO Buffer, 16 bytes Sample Buffer.

A distributed linear expression evaluator provides values of the linear expression Ax+By+C to each PE simultaneously, in byte-serial form (x and y represent the position of the PE in the 16x16 array). It is used for very fast computation of the linear part of the numerator and denominator of the warp-equation expressions (see Equation 1). Each PE includes a byte-wide connection to its neighbor in each dimension. Clock rate for the PE and local memory will be at 300 MHz or more.

The IO Buffer is used for inputting reference-image tiles (from the Tile Cache or host, via the Network Interface) via a 300 MByte/sec interface. The Sample Buffer is used for exporting warped samples to the Region Accumulator, over the sample port, via an on-chip 4.8 GigaByte/sec interface. Access to these buffers may occur simultaneously with accessing of the main memory by the ALU, so that the next tile may be loaded during processing of the current one, and one set of interpolated samples can be computed while the previous set is being output to the Region Accumulator.

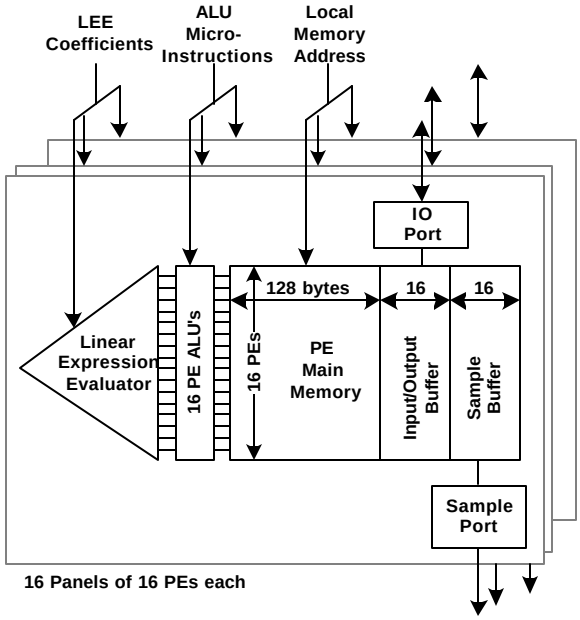

Figure 8. Block diagram of the Warp Array

3.2.2 Region Accumulator

The Region Accumulator (Figure 9) consists of a large SRAM warp buffer (the Region Buffer) and a set of 4 Sample Processors, which combine warped samples into Region Buffer memory.

The Region Buffer contains data for a 128x128-screen region, at 2x2 sub-pixel resolution; a half-pixel wide boundary is added, to allow reconstruction kernels up to two pixels wide. The Region Buffer is partitioned into 4 sections, interleaved 2x2 across the sub-pixel grid.

Each word of Region Buffer memory is divided into three fields. Two double-buffered fields (the RGB/Offset/Present fields) include RGB values, the offsets used for reconstruction, and a *present* bit (used to avoid z-buffer initialization). One buffer is used for accumulating samples for the current region, while the other buffer contains the previous region's values for output.

The third field contains values that are not required for reconstruction and need not be double-buffered. Besides z value, we are reserving space for measures such as the quality of each sample [Mark99]. If the z of two samples are similar, the sample processor gives preference to the better sample. The quality of the sample is derived differently according to the scene. In the context of imperfect registration characteristic to our (and probably all) current depth-image acquisition devices, we obtained better results when we consistently chose the samples of *one* sampling location and used the additional samples from other images just to fill in holes. Synthetic data simulates perfect registration and the quality of the samples was derived from the interpolation factor of the tile it belonged to: the closer the interpolation factor was to 2x2, the higher the quality[5].

A 128-bit wide memory interface provides read/write access to all three buffers in parallel.

Each Sample Processor processes a sample every two clock cycles; this is the maximum possible rate, since 2 Region Buffer accesses (1 read and 1 write) are required for each sample. The Sample Processor is pipelined, so that each computation has several cycles to execute, while sustaining the rate of a sample every two clock cycles. Thus 4 Sample Processors handle an aggregate rate of 2 samples per clock cycle, or 600 million samples/sec at 300 MHz.

The back buffer outputs samples from the previous region to the Reconstruction Buffer, via a shift path that spans all 4 partitions of the Region Accumulator. A small fraction of memory cycles are stolen from the Sample Processors, to feed this scan-out path.

3.2.3 Reconstruction buffer

The Reconstruction Buffer accepts the stream of final warped subsample values from the Region Accumulator, and filters them to produce final pixel values for the 128x128 pixel region. The Reconstruction Buffer includes two scan-line-sized accumulators, and four simple processors. For each RGB/Offset/Present value, each color component is multiplied by a weight from the filtering kernel and added to a sum. Normalization by the sum of weights produces the final pixel value, which is output from the ASIC to the Frame Buffer. The filter kernel is 2x2 pixels in size, with 4x4 sub-pixel resolution. The two 2-bit offset values select the proper kernel element within each sub-pixel.

[5] A 2x2 interpolation factor implies destination image sampling close to reference image sampling, which is desirable.

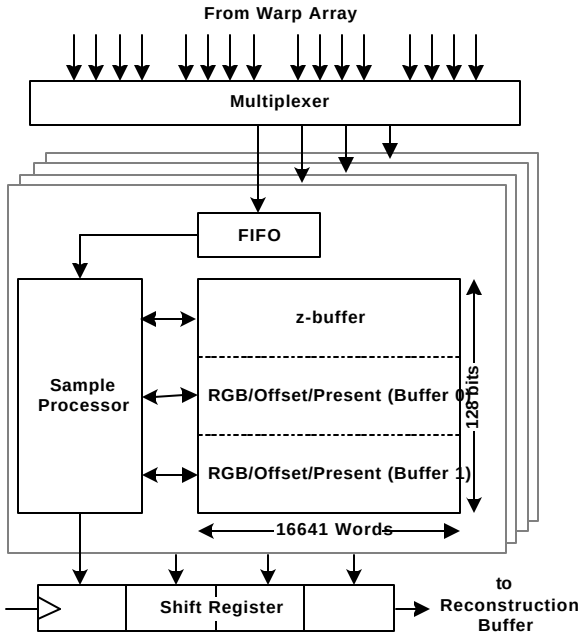

Figure 9. Block diagram of the Region Accumulator

3.2.4 Frame Buffer

The Frame Buffer is a straightforward assembly of commodity DRAMs and programmable parts. It must absorb the full bandwidth of the Reconstruction Buffers on all Nodes, so the peak output rate of the Nodes must be tuned to avoid over-running the Frame Buffer.

3.3 Host and Software

The host is responsible for determining which reference-image tiles will be used to compute the current destination image, for sorting the tiles according to screen region, and for sending the tiles to the WarpEngine Nodes. The host must also determine the interpolation factor for each chosen tile and send instructions to control the warping and interpolation, but these instructions are cached in the Tile Caches and should not represent a significant computational or bandwidth burden for the host.

3.3.1 Retained Mode

Since all reference images are available beforehand, and since the depth discontinuities in the reference images do not depend on the desired view, surface connectivity is estimated as a pre-process. This frees the Warp Array of an additional task at the price of a few additional connectivity bits per reference-image sample.

Determining which tiles are needed for the current image begins with choosing the tiles that are visible. This is done efficiently by subdividing each reference image down to 16x16 tiles in quad-tree fashion and recursively testing whether rectangular sub-images of the image are visible. The visibility test itself is identical to the bucket sorting of tiles: the 4 corners of the sub-image are warped with the minimum and then the maximum disparity of the sub-image. The bounding box of the 8 resulting points is a conservative estimate of the screen area covered by the sub-image. If the sub-image is a tile (a leaf in the quad-tree) it is also assigned to the appropriate screen region bin(s).

Depending on the scene, a large number of tiles can be visible and warping all of them is inefficient. Not all visible tiles are needed for the current frame since some tiles sample the same surfaces. Choosing among the visible tiles is not a trivial task. First one needs to determine which tiles sample the same surfaces and then choose among the several candidate tiles according to a quality metric.

The algorithm we use approximates each visible tile by two triangles. The triangles are transformed, projected and scan-converted according to the desired view. The z-buffer test is fuzzy and when two samples are close, the one that belongs to a better tile wins. A better tile is a tile whose approximating triangles have a desired-image size closer to 16x16, which implies a reference-image sampling close to the destination-image sampling. After all visible tiles are processed the chosen tiles are the tiles that have at least one sample left in the tile-choosing buffer.

Scene		Tiles	Overlap factor	Interpolation Factor
Reading room	w/o t.c.	6246	1.28	2.3x2.5
	w/ t.c.	4863	1.32	2.4x2.6
Eurotown	w/o t.c.	15050	1.23	2.2x2.4
	w/ t.c.	6736	1.44	2.6x2.9
Helicopter	w/o t.c.	9101	1.47	2.5x3.7
	w/ t.c.	4976	1.61	3.3x4.8

Table 1. Simulation results on three test scenes at VGA resolution (see Color Plates 3-5) with and without tile choosing.

Table 1 shows the average number of visible tiles in our simulations and the number of tiles chosen by the algorithm. It also shows the *overlap factor*, the average number of regions a tile mapped to in our simulations. The number of chosen tiles is high due to the tiles that have depth discontinuities and for which the triangle approximation breaks. Such tiles are conservatively chosen since the algorithm cannot establish their potential redundancy. We are currently investigating splitting the tiles that have depth discontinuities into depth-discontinuity-free tiles, whenever possible. We believe that this will reduce the total number of chosen tiles since the algorithm presented above will now eliminate more redundant tiles. The overlap factor also will be lower since tiles with depth discontinuities have an unnecessarily large screen-space bounding box.

The host needs to determine the interpolation factor for each chosen tile. The ideal interpolation factor is the minimum value for which surface continuity is preserved. We first find the maximum changes in disparity along each direction, then we use them to estimate the maximum screen-space distance between two neighboring samples. The maximum one-pixel disparity variation is computed as a pre-process, taking into consideration depth discontinuities. Table 1 shows the average of the interpolation factors used in the simulations shown on the video.

Frame-to-frame coherence can be exploited to minimize bandwidth requirements by storing each rendered tile in the Tile Cache of the WarpEngine Node that rendered it[6]. A large percentage of these tiles can then be used in rendering the same region for the next frame, and many of the remainder can be re-distributed using the Network Interface and used by other WarpEngine Nodes for other regions. Only a relatively small percentage of the tiles will need to be sent from the host; in fact, with a modest number of reference images, it should be possible to cache all the reference-image tiles. A PC's AGP interface should provide plenty of bandwidth for sending missing tiles and pointers to cached tiles[7].

3.3.2 Immediate Mode

For immediate-mode, frame-to-frame coherence cannot be utilized as effectively, since users may wander into areas of the environment that have not been previously sampled in reference images, and the environment itself may indeed be in flux (persons moving, for example). This means that bandwidth requirements from the host will be much higher. In the worst case, it may be necessary on each frame to send every tile from the host to the WarpEngine, and to render every tile.

Within the next few years, we do not expect real-time depth-image acquisition at better than VGA resolution. We can build an immediate-mode system with 20 WarpEngine Nodes that contains a full-screen-sized warp buffer; this means that bucket sorting is not required. Similarly, the low-resolution yields a manageable amount of data. If one data stream provides 640*480 pixels at 30Hz, this is 36,000 tiles/sec or 72 megabytes/sec. For an immediate mode system with four such data streams, a single high-end PC host with an AGP 4X Interface could handle routing tiles to the WarpEngine Nodes. Silicon technology (for the WarpEngine ASIC) and interface technology (for data bandwidth) should scale as depth-image acquisition scales. We are also investigating the possibility of decompressing tiles within the Warp Array.

Another difference is that the PEs will have to compute connectivity information. This is not a serious performance loss since the computation required is simple enough: two adds and a compare for each of the four directions along which connectivity is estimated; a PE can easily get the disparities of the neighboring samples through the closest-neighbor communication paths. Also the host cannot approximate the interpolation factor as described in 3.3.1 since the tile information needed cannot be pre-computed. Our solution is to let the Warp Array estimate the interpolation factor: after all PEs warp their sample, using the inter-PE paths, the Warp Array establishes the maximum distance between consecutive warped samples, in both directions. This produces interpolation-factors that are close to ideal, as described in 3.3.1.

3.4 Performance Considerations

The performance estimates are based on our WarpEngine functional-block-level software simulator. The Warp Array performance was measured with a cycle-accurate simulator. The Warp Engine system has two basic performance limits: the number of tiles per second that can be warped and interpolated, and the number of regions per second for which final pixel values can be reconstructed and forwarded to the Frame Buffer. The first defines the maximum achievable rendering rate, while the second defines the maximum achievable update rate for a given screen-size.

[6] Using a 64-MegaBit SDRAM chip as the Tile Cache, each WarpEngine node can cache up to 4,096 reference-image tiles (each tile contains, 256 pixels, each with 4 bytes of color and connectivity, and 4 bytes of disparity).

[7] AGP 2X presently supports peak data transfer rates of 533 MBytes/sec, with a future 4X extension to 1066 MBytes/sec planned. Actual usable throughputs are 50-80% of the peak rate.

3.4.1 Tile Warping/Interpolation Performance

We have found that the Warp Array will require 1878 cycles to perform a 3D image warp for all samples in a tile, using fixed-point arithmetic as described in [Mark99].

If interpolation is done after warping, the interpolated samples will be computed and output to the Region Accumulator one sample at a time (over the entire tile). Outputting one sample for the entire tile requires 256 clock cycles (one cycle per PE). The time to actually compute each interpolated sample from the warped samples will be significantly less. Table 1 indicates that, on average, interpolation generates about 8 sub-samples, so about 8 * 256 = 2K cycles are required to interpolate and output the warped samples. The Region Accumulator can process up to two samples per clock cycle, assuming decent load-balancing, so it is very likely that the one sample per cycle peak output rate of the Warp Array can be sustained. The total time per tile is therefore about 4K cycles, or about 75K tiles per second, per Node. Table 1 shows a typical overlap factor of less than 1.5, so the net performance will be 50K tiles per second per Node. Table 1 shows that at VGA resolutions 5K tiles are typically required to render a scene; we believe that these numbers extrapolate to higher resolutions. Using these assumptions, we computed the following performance numbers for some typical system configurations:

Screen size	Tiles/ frame	Nodes	Sub-samples/sec	Update rate
640x480	5K	3	307 M	30
1280x1024	20K	16	1.6 G	40
2048x1024	32K	32	3.2 G	50

Table 2. Projected performance of typical system configurations.

These numbers show that 4 Nodes can easily handle VGA output resolution loads and that 32 Nodes make a quite powerful system capable of high update rates at HDTV resolution.

If interpolation is done before warping (which we do not think is necessary), it takes on average 8200 cycles to interpolate and warp the same average number of 8 sub-samples. However, there is enough time for the warped sub-samples to be forwarded to the Region Accumulator so no additional cycles are needed. This indicates that interpolation and then warping is feasible but it requires on average twice as many Nodes for the same performance.

3.4.2 Reconstruction Performance

The Reconstruction Buffer operates on the back buffer. It requires 64K clock cycles to compute final pixel values for a region, which is pipelined with the time to render another region. Only if the next region is assigned fewer than 16 tiles (less than it takes to *cover* the region) will the reconstruction time affect performance.

4 FUTURE WORK

The programmability and high-performance of the WarpEngine will allow us to conduct many experiments. Thus far we have not attempted to generate view dependent effects. If provided with the necessary BRDF information, perhaps as a shader program, the Warp Array could compute the view dependent color [Olano98]. Similarly, one could experiment with changing the original lighting conditions of the reference images.

Tile-choosing is a very important and difficult problem, similar to the visibility and level-of-detail problems in conventional rendering. Our tile-choosing is presently complicated by having to detect and resolve inconsistencies between the samples of the same surface seen in several reference images. Dealing separately with view-dependencies will simplify tile-choosing.

Another challenging problem is encountered at the silhouettes. Since photographs or antialiased renderings are used, the color of a silhouette sample is a blend between the color of the front and back surfaces. When warping the depth image, this blended color persists on both the front and the back surfaces, which are no longer adjacent. To prevent this, we discard the silhouette samples, and rely on other reference images to provide replacement samples. For very thin features however, correct samples cannot be found in any of the reference images, causing the thin features to disappear. This comes at no surprise since, in order to respect the Nyquist sampling-rate criterion, the reference images should sample the scene at least twice as densely as the output image. We could, of course, use higher-resolution reference images, but practical considerations will usually prevent this.

An alternative approach, at least in retained mode, is to detect the thin features and model them with tiles from higher resolution images closer to the objects. For example, a light pole that projects one-pixel wide in the desired image can be extracted from a reference image that sees it as several pixels wide. Detecting the thin features can be done relatively easily using the depth-discontinuity detection method described. For efficient tile utilization, the reference-image coordinates of the samples can be stored explicitly, which allows packing the pole samples on one tile, at the price of more data per tile and slightly longer warping time.

An attractive alternative use for the WarpEngine architectural ideas is in a hybrid geometry/image-based rendering machine, which uses images as impostors to bound the total number of polygons [Aliaga99].

5 CONCLUSIONS

The WarpEngine is a 3D graphics hardware architecture designed specifically for rendering by warping images with depth. It might be argued that warping and then interpolating is equivalent to creating a quadrilateral mesh and rendering it on conventional polygon-rendering hardware. There is some truth to this, but the WarpEngine ASIC is a quad renderer particularly optimized for this application. This is because the quads formed by warping the samples of a reference-image tile are of small and uniform size, and conveniently grouped into square arrays, so a SIMD array provides particularly efficient processing. Furthermore, bilinearly interpolating between warped samples (a forward-mapping) requires minimal setup costs, unlike conventional scan conversion (which is a reverse-mapping), further optimizing the processing of tiny quads. Finally, the SIMD array allows flexible programmability, facilitating experimentation with new algorithms. And integration of the SIMD array with the on-chip warp buffer obviates bandwidth concerns and the use of off-chip memory (except for the frame buffer), and the partitioning facilitates scalability to very high performance levels.

The WarpEngine will be implemented using a single custom ASIC, replicated as necessary to meet the desired resolution and warping performance. We expect the ASIC to measure about 12 mm by 16 mm when fabricated on a 0.18-micron process, and to run at 300 MHz or higher. A small 4-node system could fit on a board inside a PC, while a 32-node system will be in a workstation-sized enclosure. We expect to begin layout of the WarpEngine ASIC later this year.

Building the WarpEngine will provide us with insights applicable not only to IBRW architectures but also to architectures for conventional polygon-based rendering, particularly when rendering small polygons.

We expect that the WarpEngine, coupled with image-based modeling or real-time depth imaging, will render images that look truly photorealistic, leading to a dramatically heightened sense of presence for applications like visual simulation and tele-presence, and enabling entirely new applications such as 3D TV.

ACKNOWLEDGEMENTS

We would like to thank Gary Bishop and John Poulton for their encouragement at early stages of this work, David McAllister for his important contributions to depth-image acquisition, and Henry Fuchs for his useful critique of earlier versions of this paper. Special thanks to Mary Whitton for organizing the SIGGRAPH-submission event here at UNC. Support was provided by DARPA, order number E278, and NSF grant number MIP-9612643.

REFERENCES

[Aliaga99] Aliaga D. and Lastra A., "Automatic Image Placement to Provide a Guaranteed Frame Rate", *Proc. SIGGRAPH '99*, 307-316 (1999).

[Beier92] Beier T. and Neely S., "Feature-Based Image Metamorphosis", *Proc. SIGGRAPH '92*, 35-42 (1992).

[Beraldin92] Beraldin J-A., Rioux M., Blais F., Domey J., and Cournoyer L., "Registered Range and Intensity Imaging at 10-Mega Samples per Second", *Opt. Eng.*, **31**(1): p. 88-94 (1992).

[Chen93] Chen S. and Williams L., "View Interpolation for Image Synthesis", *Proc. SIGGRAPH '93*, 279-288 (1993).

[Chen95] Chen S., " Quicktime VR - An Image-Based Approach to Virtual Environment Navigation", *Proc. SIGGRAPH '95*, 29-38 (1995).

[Cyra] The Cyrax System, in http://www.cyra.com/.

[Debevec96] Debevec P., Taylor C., and Malik J., "Modeling and Rendering Architecture from Photographs: A Hybrid Geometry and Image-Based Approach", *Proc. SIGGRAPH '96*, 11-20 (1996).

[Ellsworth97] Ellsworth D., *Polygon Rendering for Interactive Visualization on Multicomputers*, PhD thesis, University of North Carolina at Chapel Hill, 1997.

[Gortler96] Gortler S., Grzeszczuk R., Szeliski R., and Cohen M., "The Lumigraph", *Proc. SIGGRAPH '96*, 43-54 (1996).

[Kanade99] Kanade T., Rander P., Vedula S., and Saito H., "Virtualized Reality: Digitizing a 3D Time-Varying Event As Is and in Real Time", *Mixed Reality, Merging Real and Virtual Worlds*, Y. Ohta and H. Tamura, Editors. Springer-Verlag. p. 41-57 (1999).

[K2T] *Scene Modeler*, http://www.k2t.com/.

[Levoy96] Levoy M. and Hanrahan P., "Light Field Rendering", *Proc. SIGGRAPH '96*, 31-42 (1996).

[Mark97] Mark W., McMillan L., and Bishop G., "Post-Rendering 3D Warping", *1997 Symposium on Interactive 3D Graphics*, 7-16 (1997).

[Mark99] Mark W., *Post-Rendering 3D Image Warping: Visibility, Reconstruction, and Performance for Depth-Image Warping*, PhD thesis, University of North Carolina at Chapel Hill, 1999.

[McAllister99] McAllister, D., Nyland L., Popescu V., Lastra A., McCue C., "Real-Time Rendering of Real-World Environments", *Rendering Techniques '99, Proc. Eurographics Workshop on Rendering*, 145-160, (1999).

[McMillan95] McMillan L. and Bishop G., "Plenoptic Modeling: An Image-Based Rendering System", *Proc. SIGGRAPH '95*, 39-46 (1995).

[McMillan97] McMillan L., *An Image-Based Approach to Three-Dimensional Computer Graphics*, PhD thesis, University of North Carolina at Chapel Hill, 1997.

[Minolta] *Minolta 3D 1500*, in http://www.minolta3d.com/.

[Molnar92] Molnar S., Eyles J., and Poulton J., "PixelFlow: High-speed Rendering using Image Composition", *Proc. SIGGRAPH '92*, 231-240 (1992).

[Molnar94] Molnar S., Cox M., Ellsworth D., and Fuchs H., "A Sorting Classification of Parallel Rendering", *IEEE Computer Graphics and Aplications*, 14(4), 23-32 (1994)

[Montrym97] Montrym J., Baum D., Dignam D., and Migdal C., "InfiniteReality: A Real-Time Graphics System", *Proc. SIGGRAPH '97*, 293-302 (1997).

[Mueller95] Mueller C., "The Sort-First Rendering Architecture for High-Performance Graphics", *1995 Symposium on Interactive 3D Graphics*, 75-84 (1995).

[Olano98] Olano M. and Lastra A., "A Shading Language on Graphics Hardware: The PixelFlow Shading System", *Proc. SIGGRAPH 98*, (1998).

[Regan99] Regan M., Miller G., Rubin S., and Kogelnik C., "A Real Time Low-Latency Hardware Light-Field Renderer", *Proc. SIGGRAPH '99*, 287-290 (1999).

[Seitz96] Seitz S. and Dyer C., "View Morphing: Synthesizing 3D Metamorphoses Using Image Transforms", *Proc. SIGGRAPH '96*, 21-30 (1996).

[Shade98] Shade J., Gortler S., He L., and Szeliski R., "Layered Depth Images", *Proc. SIGGRAPH '98*, 231-242 (1998).

[Torborg96] Torborg J. and Kajiya J., "Talisman: Commodity Real-time 3D Graphics for the PC", *Proc. SIGGRAPH '96*, 353-364 (1996).

[Westover90] Westover L., "Footprint Evaluation for Volume Rendering", *Proc. SIGGRAPH '90*, 367-376 (1990).

[Wolberg90] Wolberg G., *Digital Image Warping*, IEEE Computer Society Press, Los Alamitos California, 1990.

Figure 10. These images were rendered with the WarpEngine simulator. The reference depth images were created by registering color images with the range information acquired by our laser range finder. The range finder captured data from two positions in the center of the room. We are missing some data, on the ceiling for example.