Sort-First Parallelism for Image-Based Rendering

Voicu Popescu, Anselmo Lastra, John Eyles
University of North Carolina at Chapel Hill

ABSTRACT

Rendering from images with depth (Image-Based Rendering by Warping, IBRW) is an appealing 3D-computer-graphics technique since it alleviates the modeling bottleneck and promises photorealism at interactive rates. Current, and probably near future silicon technology cannot produce single-chip IBRW renderers that are powerful enough. Thus a high-level parallelism scheme needs to be employed that divides the rendering task among several nodes.

We show that sort-first is the technique best suited for parallel IBRW since it takes advantage of the locality characteristic of depth images. One can easily determine the output-image region to which a contiguous set of depth samples warps. Thus the nodes get the samples pertinent to their region which they fully render independently, eliminating the need of very high bandwidth node-to-node communication paths for sorting the warped samples.

We previously presented the *WarpEngine*, a sort-first architecture for IBRW. In this paper we will measure the performance of sort-first by analyzing the efficiency of the parallelization scheme, the communication costs, and the load balancing for various system configurations and output-image resolutions.

KEYWORDS: graphics hardware, parallel rendering, image-based rendering, 3D image-warping.

1 INTRODUCTION

In recent years, numerous image-based rendering (IBR) methods have been proposed. They are all aimed at reducing the complexity of the modeling task associated with traditional rendering. Another common goal was to produce photorealistic renderings at high refresh rates.

One of the most promising IBR techniques enhances the images with per pixel depth ([McMillan95]). Using the original camera pose one can reproject (warp) the samples to the output (desired) image. Image-based rendering by warping (IBRW) is a complete rendering solution, in that it offers correct 3D views of the scene from arbitrary locations. Also the memory and memory-bandwidth requirements are manageable since the samples in the depth images are reused efficiently. Moreover, advances in rangefinding technology make it possible to reliably acquire high-resolution high-precision depth images, eliminating most of the modeling effort.

In the remainder of this section we will briefly describe IBRW, and show that IBRW requires non-trivial amounts of computation and framebuffer bandwidth. We then review the sort-first, –middle and -last parallel rendering classification. Section 2 argues that sort-first is the most appropriate parallelization scheme for IBRW. Section 3 analyzes the performance of sort-first parallelism using the simulator of our proposed IBRW architecture, the *WarpEngine* [Popescu00a].

1.1 IBRW

The process of image-based rendering by warping consists of three steps: warping, visibility determination, and reconstruction. The scene is modeled with images that have depth in addition to color at each pixel. We refer to these as the *reference* images. In practice, we divide larger images into small rectangular sections (*tiles*) in order to obtain more flexibility in selecting samples, while retaining the coherence and regularity of the image representation [McAllister99].

The warping equation presented by McMillan and Bishop [McMillan95] is equivalent to a conventional transformation, but is structured to be computed incrementally, thus taking advantage of the regularity of the depth images. Correct visibility can be computed either by z-buffering, or by traversing the samples in occlusion-compatible order (back-to-front) [McMillan95].

High-quality reconstruction to create the final image is a major problem. Two methods are used: splatting [Westover90] or representing connected samples as surfaces with a polygonal mesh. Either of these two approaches must use a supersampled framebuffer for high-quality, antialiased results.

1.2 Cost of IBRW

In order to estimate the cost of IBRW we estimate the number of samples that need to be warped each second and the bandwidth to the buffer that stores the warped samples (warpbuffer).

We assume that one needs as a working minimum two samples per output image pixel. This figure obviously depends on how the scene is modeled with depth images and on how efficiently one chooses the samples needed for each frame. We will not describe the current best solutions for these complex problems; for the purpose at hand we believe it is sufficient to point out various reasons why the ideal one sample per output pixel is exceeded:

- *Oversampling*: some surfaces are better sampled in the reference (input) image than in the desired image, which leads to more than one warped sample per output pixel.
- *Depth complexity*: it is impossible to efficiently discard, before warping, *all* samples that are not visible in the current view.

- *Redundancy*: some surfaces are sampled by more than one reference image; not *all* such redundant samples are discarded before warping.

The number of samples that need to be warped each second (with the two samples per output pixel assumption) is:

Output resolution		VGA 0.3 Mpix	XVGA 1.0 Mpix	HDTV 2.0 Mpix
MWarps / s	30 Hz	18	60	120
	60 Hz	36	120	240

Due to differences in depth, the warped samples move apart in the warpbuffer. In order to prevent holes, sub-samples are generated by interpolation between the neighboring warped samples. Also, for quality antialiased output images, the warpbuffer needs to have a higher resolution than the final image. With 2x2 warpbuffer supersampling we found that every sample generates between 8 and 16 subsamples that have to be z-buffered in the warpbuffer. The figure depends on the scene and on how it is modeled (number and placement of depth images). Assuming 8 byte subsamples (color and z), a total depth complexity of two[1], and that 50% of the hidden subsamples are initially visible, there will be on average 10 byte warpbuffer accesses per subsample. This translates to the following warpbuffer bandwidths (16 subsamples per sample):

Output resolution		VGA 0.3 Mpix	XVGA 1.0 Mpix	HDTV 2.0 Mpix
GB / s	30 Hz	2.88	9.6	19.2
	60 Hz	5.76	19.2	38.4

At the present, a single ASIC can provide neither the warping power nor the warpbuffer bandwidth required. Several ASICs need to be used and thus the need for a high-level parallelization scheme to distribute the work.

1.3 Sort-first, -middle and -last taxonomy

A useful taxonomy for describing parallel polygon-rendering architectures is the sort-first, -middle and -last taxonomy formalized in [Molnar94].

An architecture is said to be sort-first if the primitives (polygons, higher order surfaces) are sorted before transformation according to the screen region onto which they map and are then assigned to renderers that completely render their screen region. Sorting the primitives implies pre-transformation computations, which are typically substantially less expensive than fully transforming each primitive. Examples of sorting techniques are computing screen-space bounding boxes of objects composed of many polygons or coarse pre-tessellation of higher order surfaces [Mueller97].

Sort-middle architectures sort fully transformed primitives. The geometry processors, separated from the rasterizers, transform a subset of the primitives and send them to the rasterizers that are each responsible for a portion of the screen. The rasterizers process only the primitives pertinent to their screen region.

Sort-last architectures are composed of several renderers that fully process a subset of the primitives. The final image is obtained by compositing the partial images built by the individual renderers. Sorting occurs after rasterization, as samples (subsamples) of the primitives are composited.

Considering the depth samples as the primitives, the taxonomy can be readily applied to IBRW. Transforming the primitives corresponds to warping the depth samples and rasterization is replaced by interpolation between the warped samples. The next section will investigate the suitability of all three types of architectures for IBRW.

2 SORT-FIRST FOR IBRW

2.1 Why not sort-middle?

For polygon-rendering, a natural place to sort the primitives is after they are transformed (UNC's *Pixel-Planes 5* [Fuchs89], SGI's *RealityEngine* [Akeley95] and *InfiniteReality* [Montrym97]). The transformation burden can be equitably distributed among the geometry processors and the transformed primitives are usually small enough in screen space to map to only one screen region, thus requiring processing by only one rasterizer.

One problem with sort-middle is the high communication cost, proportional to the number of visible primitives. Complex scenes, finely modeled for high-quality renderings, imply a large number of primitives, which in turn translates into large communication costs. This disadvantage is accentuated in the IBRW case when the number of primitives (samples) is at least twice the number of output pixels.

Another problem traditionally associated with sort-middle is poor load balancing among rasterizers. Some regions of the screen may get a large number of primitives. Primitive clumping is due to:

- poor or no visibility culling inside the view frustum (rendering of objects (primitives) that are completely occluded)
- inadequate levels of detail (LOD) (objects that rendered with considerably fewer primitives would have produced the same image)
- scene complexity variation (e.g. the walls of a room require fewer primitives per pixel than a complex piece of furniture).

Numerous efforts have been targeted at eliminating the first two causes. The third cause of primitive clumping cannot be eliminated. An architectural-level solution is to make the screen regions smaller and improve the load balancing by dynamic (or statically-interleaved) assignment of rasterizers to regions.

Translated into the IBRW context, primitive clumping is less of an issue, although it cannot be ignored (Section 3.3). The depth images were acquired from nearby locations so the LOD is well adjusted and depth complexity is low. If all samples that project in the view-frustum are used, they spread evenly in the warpbuffer. If occlusion culling (extended to discarding redundant samples of coincident surfaces) is used, the total number of samples is low and there will be only a few samples per output pixel. Thus, again, the primitives are likely to be well distributed. Since the sampling resolution is constant throughout the depth image (it doesn't decrease as simpler surfaces are encountered), one cannot talk about scene complexity variation

[1] Computed as the ratio between the number of subsamples and the number of warpbuffer locations.

since the modeling paradigm does not offer this flexibility to begin with[2].

In conclusion, the main challenge of a sort-middle IBRW architecture is the high-bandwidth required for the interconnection network that delivers the transformed primitives (warped samples) to the rasterizers that interpolate between the warped samples.

2.2 Why not sort-last?

Sort-last (*PixelFlow*, [Molnar92]) is appealing since the renderers process sets of primitives, independently computing partial solutions which are composited at a cost linear with the number of renderers. Unlike sort-middle, the communication cost does not depend directly on the number of primitives. However this advantage is irrelevant for IBRW since the total number of primitives is proportional to the number of output image pixels anyway. When super-sampling is used for antialiasing, compositing has to be done before the framebuffer is filtered down to create the output image. Thus, in the case of IBRW with a 2x2 super-sampled warpbuffer, the communication cost is 4 times the number of output pixels times the number of renderers used.

Pure sort-last architectures instantiate full framebuffers at each renderer so they are less prone to load imbalance. However, single-chip high-resolution super-sampled framebuffers cannot be currently built because of silicon technology limitations, so region-based rendering has to be employed.

2.3 Why sort-first?

Sort-first has the great advantage of exploiting the frame to frame coherence associated with typical 3D-graphics applications. The view changes little from one frame to the next, so a renderer that is responsible for a certain screen region needs only a few new primitives for the new frame.

To our knowledge no sort-first polygon-renderer has ever been built. Such an architecture was proposed by [Mueller95]. There are serious challenges associated with sort-first in the case of polygon-renderers [Mueller97]. We will next show that these can be overcome in the case of IBRW.

First, one needs to establish a pre-transformation operation to be executed (by the nodes or host) that is inexpensive and accurately predicts the screen location of primitives. In the case of polygons, such an operation is hard to find. Mueller [Mueller00] groups sets of primitives and transforms their bounding box.

In the IBRW case, one can take advantage of the locality of depth images, and inexpensively and accurately predict the screen region to which a set of samples warps. This is easily done for a rectangular subregion of a depth image (tile) by computing the screen bounding box of a frustum that contains all the depth samples of the subregion. Such a frustum is easy to find, and we use the frustum defined by the four rays that go through the corners of the tile and by the planes of minimum and maximum depth in the tile (figure 1). Assuming that the hither plane does not clip the frustum, the bounding box can be easily found by warping the four corners of the tile once with minimum and once with maximum depth, and computing the bounding box of the eight resulting points. If the frustum crosses the hither plane, clipping has to be done first and the same procedure can be applied to the clipped frustum.

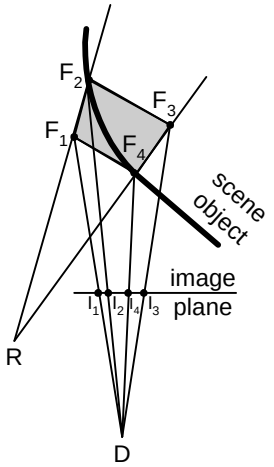


Figure 1. A single tile of the reference image R is shown (2D view). The grey area corresponds to the $F_1F_2F_3F_4$ minimum and maximum depth frustum used to approximate the tile. All the samples of the tile will warp inside the desired-image (D) projection of the frustum $I_1I_2I_3I_4$. One can see that the projection contains pixels to which no tile sample will warp.

An ideal primitive-group pre-transformation operation assigns a certain group only to renderers that need it. That is at least one of the primitives projects inside the screen region of the renderer. The method described deviates from the optimal behavior because:

- the bounding box is the axis-aligned rectangular superset of the warpbuffer locations actually impacted by the frustum projection;
- the frustum is a superset of the depth samples of the tile, as if the tile had, at every location, samples with all possible depths between the tile's minimum and maximum (fig. 1).

The first problem is not so severe. It could be eliminated by projecting and scan-converting the frustum and choosing only the renderers whose regions are impacted, but the gain is too small to justify the scan-conversion of the 12 triangles.

The second problem is important, especially for tiles that cover two or more objects at different depths. The resulting frustum is large due to the large depth variation in the tile (figure 2). We segment such tiles into several groups of similar depth, one for each object sampled by the tile. In retained mode, when the depth images are available beforehand, the tile segmentation can be done as a preprocess. For immediate mode, when the depth images are acquired in real time, the cost of segmenting the tiles on the fly needs to be compared with the savings achieved.

We use 16x16 sample tiles and we found, as will become apparent from the next section, that the pre-transformation operation is very effective.

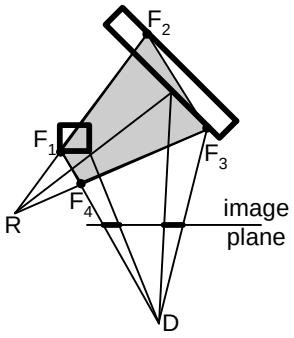


Figure 2. The tile shown spans two objects in the scene (rectangles) that are far apart. The frustum becomes big and its desired-image-plane projection is much larger than the actual projection of the tile (shown with thicker lines).

[2] An alternative to IBRW is to first use the depth images to generate a geometric model of the scene and then render the model on polygon-rendering hardware. Such models might group nearly coplanar samples onto a single textured polygon.

Another problem imputed to sort-first architectures is the difficulty in using hierarchical databases of primitives. Such databases have tree like structures where the internal nodes are modifiers of the objects obtained by aggregating the primitive groups stored at the leaves. Problems arise especially with the replication modifier, when the database needs complex editing.

For the first generation of IBRW hardware we do not target rendering from hierarchical databases because of the problems it introduces.

First, natural-light-source modeling from photographs and image-based-object re-lighting are complex problems not yet solved (research is underway [Yu98] and [Debevec98]). For now we concentrate on prototyping hardware that renders, interactively, complex natural scenes using the lighting originally captured in the photographs.

Second, having all the instances of a replicated object in the image-based representation of a scene has the advantage of providing pre-computed appropriate level of detail for each instance. Also the images provide a partial occlusion-culling solution, which limits the number of samples that have to warped at a frame. In the case of hierarchical databases this advantage is lost and the number of primitives is unbounded, like in polygon rendering.

We conclude that sort-first is well suited for IBRW when the current frame is rendered from samples of a fixed number of nearby depth images. We devised such an architecture, the *WarpEngine*, which we presented in [Popescu00a]. In the next section we will analyze the performance of sort-first parallelism for IBRW using the *WarpEngine* simulator. For the reader's convenience we will now briefly describe the *WarpEngine* architecture.

2.4 WarpEngine

The *WarpEngine* is based on a forward reconstruction algorithm that produces high-quality output images at a lower cost than the mesh reconstruction method. The savings come from forward-scan-converting the quads that connect the warped samples by interpolation in continuous image coordinates as opposed to the classic inverse mapping to pixel centers (or to supersampling locations defined with respect to the pixel boundaries). The locations of the sub-samples generated by interpolation are recorded accurately with offsets that are used at the resampling stage. Thus the inversion to the discrete image domain is done efficiently after visibility is solved. See [Popescu00a] for more details.

The *WarpEngine* architecture 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 rasterizing (interpolating) the samples of a tile;
- a Region Accumulator, which includes a double-buffered warpbuffer 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 to each Tile Cache.

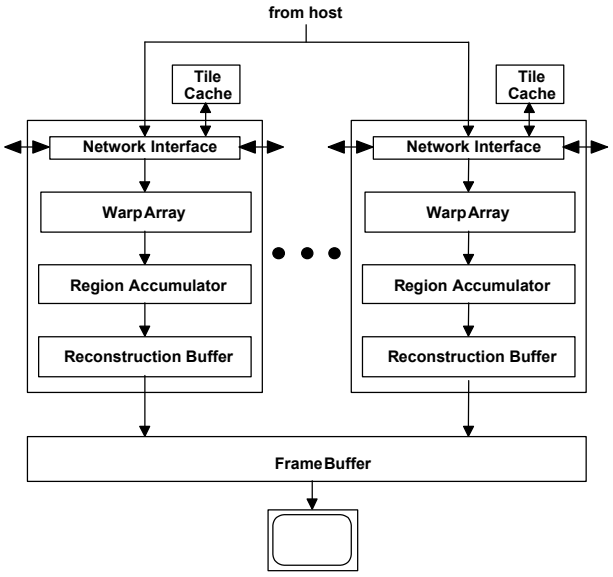


Figure 3. Block diagram of the *WarpEngine*

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 (figure 3) is as follows:

- 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 subsamples to the Region Accumulator.
- The Region Accumulator collects the subsamples 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 next section presents the performance of sort-first parallelism for IBRW using the *WarpEngine* software simulator.

3 SORT-FIRST PERFORMANCE

We ran experiments on three scenes:

- *Eurotown*, a complex model of a city; the depth images are placed on a regular grid and were rendered with a polygon-renderer (see Color Figure 1);
- *Kamov helicopter;* eight depth images also generated from a polygonal model are placed around the helicopter and one samples it from above (see Color Figure 3);
- *Reading Room;* two depth panoramas of the reading room of our department were acquired with the *DeltaSphere* [Nyland99], our in-house laser rangefinder (see Color Figure 2).

We measured the efficiency, communication costs, and load balancing for various host-software modes:

- VFC: tile selection based on view-frustum culling only; all tiles in the view-frustum are sorted and sent to the appropriate node;
- VFCTS: tile selection based on view-frustum culling and tile segmentation: tiles with depth discontinuities were divided into up to eight segments; view-frustum culling and sorting was done at the tile-segment level;
- OC: tile selection based on view-frustum and occlusion culling, with tile segmentation. This is our most elaborate host tile-selection algorithm [Popescu00b]. The tiles are approximated by quads that are rendered into a low-resolution buffer for occlusion culling. The coincident tiles (same surface sampled in more than one depth image) are arbitrated according to the sampling quality: the tiles that have the sampling rate closest to the sampling rate required by the desired image are chosen.

3.1 Efficiency

An important measure of the performance of the parallelization scheme is efficiency, that is, the ratio between the useful and total work. For the *WarpEngine* this translates into the *overlap factor*, which is defined for a frame as the total number of tiles in the region-bins over the number of distinct tiles. Here are the average frame overlap factors for the various scenes and various tile selection methods:

Scene	Tile-selection method	Sel. Tiles	Overlap	Error (%)
Eurotown	VFC	17056	1.330	8.16
	VFCTS	17022	1.283	4.92
	OC	4224	1.341	5.60
Kamov	VFC	9121	2.650	50.50
	VFCTS	9108	1.505	12.81
	OC	2737	1.82	22.88
Reading room	VFC	6246	1.219	5.73
	VFCTS	6245	1.218	5.63
	OC	4756	1.237	5.68

The screen resolution is 720x486, and the screen region size is 128x128 pixels, which is equivalent to 256x256 warpbuffer locations. The tiles are 16x16 reference-image samples in size. The animation paths used can be seen as a QuickTime clip at http://www.cs.unc.edu/~popescu/paths.mov.

The selected-tiles column gives the average number of tiles that are selected for a frame (before bucket sorting). The number of selected tiles decreases slightly when segmentation is introduced since some tiles conservatively ruled as visible by the VFC host-mode are correctly detected as invisible by the VFCTS host-mode. This happens when a whole tile frustum is visible but none of the tile-segment frusta are.

The number of selected tiles decreases substantially when occlusion culling is added. The overlap is reduced by tile segmentation. The slight increase with occlusion culling is due to the fact that the average screen-projection size of the tile increases. (Tiles that sample too densely are eliminated in favor of tiles that better match the desired-image sampling-rate.)

The last column gives the approximation error of the pre-transformation operation used to bucket sort the tiles; it is computed as the percentage of tiles that are unnecessarily rendered at a region (from the total number of tiles rendered). In our simulator such an error is detected when none of the samples of a tile warps to the region to which the tile is allocated. In the case of the helicopter with no tile segmentation the error is quite important, because the tiles of the top image have large screen projections. When tile segmentation is used, the error is not very important and it doesn't justify a more expensive pre-transformation operation.

The overlap does not depend on the number of nodes, it depends only on the size of the regions. How does the overlap vary with the output resolution? If the average screen-projection size of the tile and the screen-region size remain the same, the overlap factor should not change. Out of sampling considerations, the tile screen-projection size *has to* remain the same. Increasing the resolution of the output without increasing the resolution of the input accordingly just increases the amount of interpolation between the samples, which, of course, produces unsatisfactory results. In all our experiments the resolution of the reference images was the same as the resolution of the output image. The following table[3] shows the invariance of the overlap factor with respect to output resolution (*Eurotown* scene).

Res	Tile-selection method	Sel. Tiles	Tiles / pixel	Overlap	Error (%)
VGA 720 x 486	VFC	14031	0.0400	1.318	7.00
	VFCTS	14002	0.0400	1.283	4.94
	OC	4243	0.0121	1.324	5.53
XVGA 1280 x 1024	VFC	49890	0.0380	1.345	7.04
	VFCTS	49872	0.0380	1.316	4.97
	OC	12619	0.0096	1.334	5.96
HDTV 2000 x 1000	VFC	74679	0.0373	1.431	11.56
	VFCTS	74603	0.0373	1.364	7.14
	OC	21696	0.0109	1.377	7.94

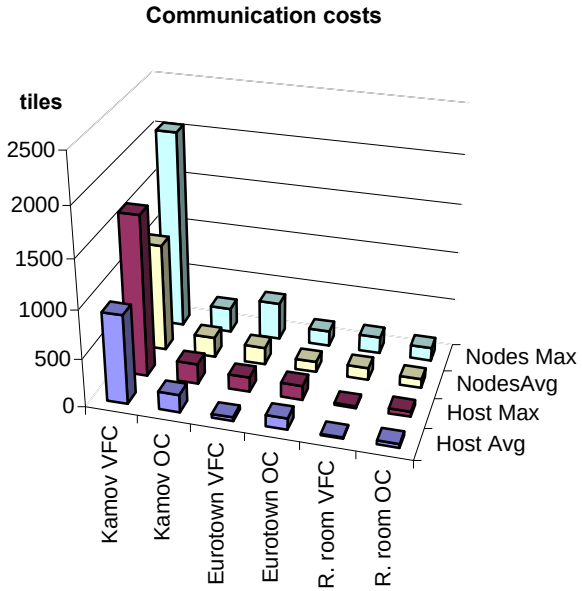
[3] The VGA numbers are slightly different than in the previous table since the path used for this table is different (shorter)

The number of tiles increases close to linearly with the screen area, as can be seen from the resolution-invariant tiles/pixel figure.

We conclude that the overlap factors are small, which makes sort-first an efficient parallelization scheme for IBRW, capable of substantial speed-ups.

3.2 Communication costs

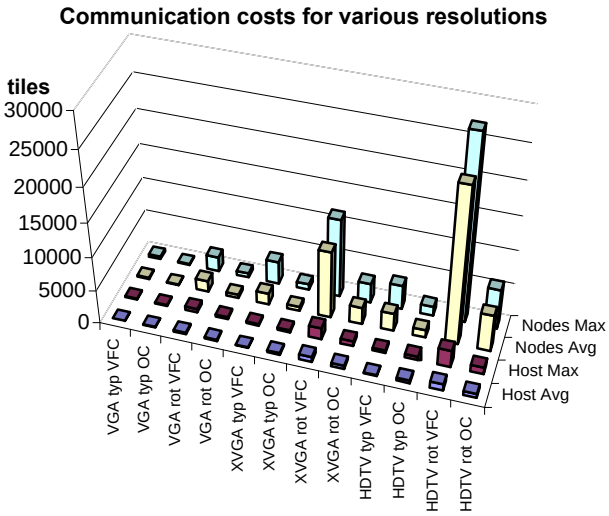
Another crucial factor in determining the performance of the parallelization scheme is the amount of communication required, both between the host and the *WarpEngine,* and between the nodes of the *WarpEngine*. Before a node can render a tile it needs to have it in its local Tile Cache. The node can get a missing tile either from another node, or, if no node has it, from the host. The following graph (also see appendix 1) shows the per-frame average and maximum communication requirements measured in our experiments.



The output resolution is VGA and the *WarpEngine* is configured with 4 nodes to which the 6x4 regions are statically assigned in an interleaved pattern. The size of a tile is 256 samples, which at 8 bytes per sample totals 2KB. The tile-caches will probably be implemented with 256-Mbit SDRAM chips, so each can store up to 16 Ktiles.

In computing the figures above it was assumed that a tile is discarded from the Tile Cache if not used for a frame. This is over-conservative since Tile Caches are large enough to store tiles for several frames, which is useful when the path re-explores parts of the scene. Even so the communication volumes are manageable. A high-end PC can send at peak rate, assuming 30Hz frame rate, at most 8 Ktiles per frame through an AGP2x 533 MB/s interface. The Kamov scene requires the most communication when no occlusion culling is used since the tiles clump in the region to which the helicopter maps.

For *Eurotown*, the numbers given do not take into account the transitions from one cell to the other, when about half of the tiles are replaced with new tiles since 4 out of the 8 sampling locations are new. For such cases we found that about 15 Ktiles need to be sent from the host in VFC mode and 6 Ktiles in OC mode. The projected AGP4x 1GB/s interface might provide enough host bandwidth even for these extreme cases. For higher output resolutions the host will have to predict the new cell and amortize the high host-bandwidth requirement of cell transition over several frames. Of course this limits the translational speed of the camera for a certain cell size. Higher speeds can be supported by increasing the size of the cell accordingly. High rotational speeds of the camera also increase the communication requirements. The following graph (also see appendix 2) shows the per frame communication costs for various resolutions for a typical path (*typ* in the graph) and also for the case of a 90 degrees / second rotation (*rot*).



The host bandwidth required can be supplied by the AGP interface, as all maximum host-to-*WarpEngine* communication volumes are below 4 Ktiles. The Network Interface that interconnects the nodes is not fully designed yet; from experience we are confident that ring bandwidths of 8 to 64 Gb/s are attainable. At 60Hz this translates into 8 to 64 Ktiles / frame which from the above graph appears to be sufficient for the highest node-to-node traffic.

A potential bottleneck is the bandwidth in and out of the Tile Caches. Using 16-pin 133 MHz memory chips for the Tile Caches provides 266 MB/s, which at 30 Hz (60 Hz) means about 2 (1) Ktiles / frame if half of the bandwidth is reserved for the instructions for the *Warp Array* SIMD. For the worst case in the graph (HDTV rot VFC), assuming a full-blown system with 32 nodes, the maximum number of new tiles a node needs at any given frame is 8.6Ktiles[4]. Conservatively increasing the figure by a factor of two, since a node has to provide tiles to other nodes, we obtain about 16 Ktiles/frame maximum Tile Cache accesses, which is considerably more than the 2 Ktiles computed at 30 Hz. However, as it will become apparent from the next subsection, in the VFC host mode only 8 fps can be achieved, reducing the mismatch considerably. Conversely, 30 fps are achieved in the OC host mode when there are considerably

[4] The figure was obtained from our simulations. If fewer nodes are used, the figure goes up since a node is responsible for a larger screen subset. However for HDTV, only a full blown system of 32 nodes can provide enough performance.

fewer tiles, thus fewer Tile Cache accesses. We will investigate the Tile Cache access bottleneck further. An attractive solution is to use 2 or 4 memory chips for each Tile Cache, which, besides providing the required bandwidth, also has the advantage of providing more storage room. The trade-off is a higher number of pins for the rendering-node chip.

The host main-memory accesses are not a bottleneck since the host communication volume is much less important.

3.3 Load balancing

Finally we investigated the load balancing of the *WarpEngine* architecture. The following table (VGA output resolution) shows the ratio between the number of tiles assigned to the busiest node and the least busy node. The other measure of imbalance given is the ratio between the time it takes the busiest node to finish rendering and the time required by the least busy node. The performance of the architecture is given by the performance of the Warp Array, which warps and interpolates between the warped samples. To process a tile it takes approximately 2000 cycles for warping and n * 256 cycles for outputting the n-per-sample interpolated sub-samples.

Scene	Tile Selection Method	Nds	Load Imbalance		Sust. fps
			tiles	time	
Eurotown	VFC	4	1.315	1.358	12
		9	3.09	2.52	20
	OC	4	1.36	1.15	47
		9	2.25	2.34	80
Kamov	VFC	4	1.83	1.89	2
		9	5.36	5.32	2
	OC	4	1.22	1.36	7
		9	1.76	3.86	12
		24	8.16	10.62	19
Reading room	VFC	4	2.02	1.90	26
		9	2.85	3.09	47
	OC	4	1.68	1.62	35
		9	2.34	3.62	70

The sustained frame rate is computed by finding the busiest renderer at any of the frames of the animations. The Warp Array clock-rate is 300 MHz. Load balancing is acceptable in the case of the *Eurotown* and *Reading Room* scenes. The two imbalance figures are close since the tiles are interpolated with similar interpolation factors. The regions do not divide evenly among the nodes, and the extra region some of the nodes have causes imbalance especially in the case of higher number of nodes. The imbalance is moderate and a higher number of nodes provides very high refresh rates.

For the Kamov scene some regions, thus some nodes, are assigned a considerably higher number of tiles, and the interpolation factors are also higher, due to the way the reference images are placed. Even if each of the 24 regions has its own node and even if the most efficient tile selection method is used, the sustained frame rate is below interactivity. Modeling the scene with regularly placed depth images has the advantage we anticipated of better load distribution.

The following table presents the load-balancing measurements for higher resolutions (obtained on the *Eurotown* scene).

Scene	Tile Selection Method	Nds	Load Imbalance		Sust. fps
			tiles	time	
XVGA	VFC	16	2.33	2.75	9
		30	7.34	8.34	14
	OC	16	2.73	3.24	42
		30	6.35	5.85	65
HDTV	VFC	16	1.50	1.83	6
		32	1.76	2.75	8
	OC	16	2.18	1.96	23
		32	2.76	4.68	29

A 32-node *WarpEngine* system is quite potent and can render in OC mode at high resolutions at sustained interactive rates.

For systems with many nodes, load imbalance becomes an issue. For the HDTV case, if each region has assigned a node (a hypothetical 128 node-system), the sustained frame rate is 39 fps for the OC case. This is the upper bound on the refresh rate achievable by improving the regions-to-nodes assignment. The static allocation has the advantage of low node-to-node communication since it takes full advantage of the frame-to-frame coherence. Once regions are assigned dynamically to the nodes, one has to keep the node-to-node communication under control. We experimented with a greedy allocation scheme that assigned iteratively the hottest region to the least busy node. When the greedy allocation scheme was utilized at every frame for the *Eurotown* HDTV OC case, we obtained an almost perfect 1.031 worst rendering-time load-imbalancing with sustained 30 fps refresh rate for a 16-node system. A 32-node system rendered at 38 fps with a time imbalance of 1.689.

Since the allocation tables differed substantially from one frame to the next, the volume of node-to-node communication was quite high (maximum 26,594 tiles per frame), but it did not exceed the projected capacity of the ring network. One could employ quasi-static allocation schemes that exploit the frame to frame coherence and change the node allocation of a region only rarely. Such schemes seem practical, especially since tiles are not used for a very long time as the camera translates to another cell and fresh tiles are downloaded from the host.

To attain even higher frame rates, one has to use smaller regions in order to reduce the maximum rendering time spent at a region. A *WarpEngine* node needs about 4 Kcycles to process a tile (assuming 8 subsamples per sample), thus the 22 Ktiles per frame for the OC HDTV *Eurotown* case could be rendered theoretically (ideal parallelism) 110 times a second by 32 nodes running at 300MHz. Thus there is room for improvement. The overlap factors computed for the 128x128 regions are small, which makes us believe that 128x64 regions will still yield reasonable overlap factors. For the *WarpEngine* architecture, the price of smaller regions is mainly wasting some of the warpbuffer memory of the *Sample Processors*; it does not imply processor underutilization. Moreover it is not necessary to use smaller regions for the entire screen: one could subdivide only the bottleneck regions.

4 CONCLUSIONS and FUTURE WORK

Sort-first is a very attractive parallelization scheme for IBRW. The regular structure of depth images makes it possible to

estimate accurately and inexpensively the screen projections of a set of depth samples.

Some of the difficulties associated with rendering from polygons are pushed upstream to the modeling stage. Partial solutions for occlusion culling and level-of-detail adaptation are computed inherently as the depth images are acquired and they are successfully used for an entire viewing-volume subdivision cell. The *WarpEngine* IBRW sort-first architecture is fairly simple (one ASIC design) but promises high refresh rates.

For high output resolutions, more sophisticated tile-selection host algorithms need to be employed than just choosing all the tiles in the view frustum. The occlusion-culling partial solution provided by the images acquired from the corners of the current cell needs to be further refined to reduce the number of tiles. We previously developed such an algorithm and are currently investigating possible hardware acceleration to assist the CPU of the host. An alternative path would be to enhance the *WarpEngine* architecture such that it can implement the tile selection algorithm, freeing the host of a considerable burden.

The load balancing and thus the performance can be improved by perfecting the regions-to-node allocation scheme and by splitting the regions into smaller regions where and when necessary. The additional node-to-node traffic generated by occasional region-to-node reallocations is small and can be easily managed by a high-capacity ring network.

5 ACKNOWLEDGEMENTS

Many thanks to Lars Nyland and David McAllister for their important contributions in developing the laser rangefinder system used to acquire some of the depth images used in this work. We wish to acknowledge Joshua Steinhurst, Gary Bishop, Nick England, John Poulton for their help.

This work was supported by the UNC CS Alumni Association, the Department of Energy, DARPA order number E278 and NSF grant number MIP-9612643 and ACR-9876914.

REFERENCES

[Akeley93] Akeley K., “RealityEngine Graphics”, *Proc. SIGGRAPH 93*, 109-116 (1993).

[Debevec98] Debevec P., “Rendering Synthetic Objects Into Real Scenes: Bridging Traditional and Image-Based Graphics With Global Illumination and High Dynamic Range Photography”, *Proc. SIGGRAPH 98*, 189-198 (1998).

[Fuchs89] Fuchs H., Poulton J. et al, “Pixel-Planes 5: A Heterogeneous Multiprocessor Graphics System Using Processor-Enhanced Memories”, *Proc. SIGGRAPH 89*, 79-88 (1989).

[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).

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

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

[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).

[Mueller97] Mueller C., “Hierarchical Graphics Databases in Sort-First”, *1997 Parallel Rendering Symposium*, 49-57 (1997).

[Mueller00] Mueller C., “The Sort-First Architecture for Real-Time Image Generation”, *Doctoral Dissertation, CS UNC Chapel Hill*, (2000).

[Nyland99] Nyland L., McAllister D., Popescu V., McCue C., Lastra A. Rademacher P., Oliveira M., Bishop G., Meenakshisundaram G., Cutts M., and Fuchs H., "The Impact of Dense Range Data on Computer Graphics", *Proceedings of Multi-View Modeling and Analysis Workshop* (MVIEW99), (Part of CVPR99), (Fort Collins, CO), June 23-26, 1999.

[Popescu00a] Popescu V., Eyles J., Lastra A., Steinhurst J., England E., Nyland L., “The WarpEngine: An Architecture for the Post-Polygonal Age”, *Proc. SIGGRAPH 00* (2000)

[Popescu00b] Popescu V., Lastra A., “The Vacuum Buffer”, *UNC CS TR00-017*, (2000), also available at http://www.cs.unc.edu/~popescu/vacbuffer.pdf

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

[Yu98] Yu Y. and Malik J., “Recovering Photometric Properties of Architectural Scenes from Photographs”, *Proc. SIGGRAPH 98*, 207-218 (1998).

APPENDIX 1

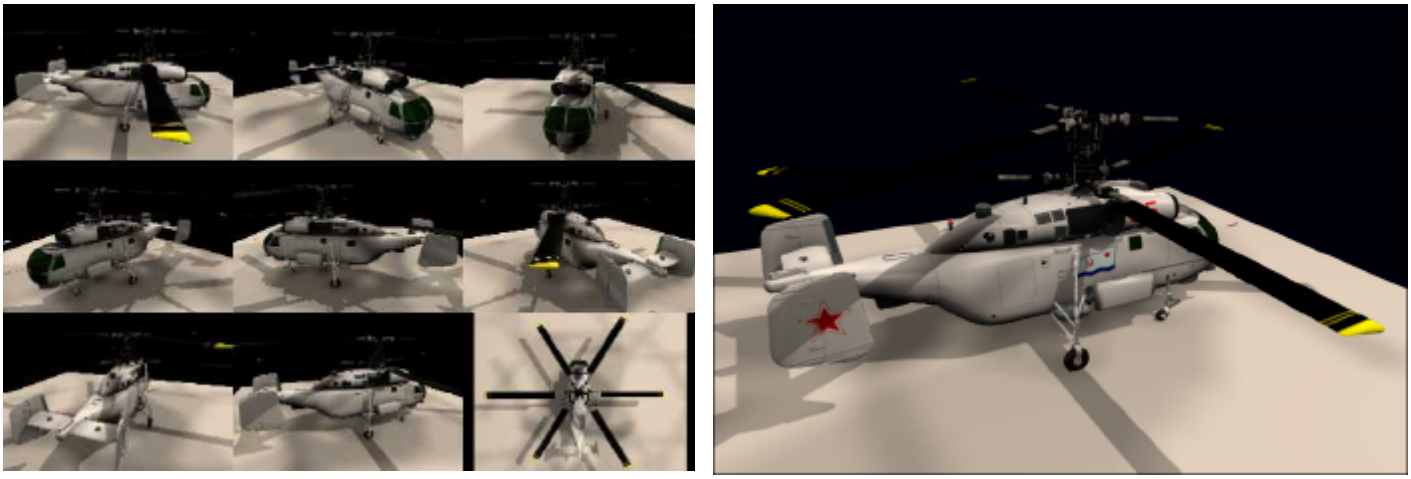
	Host Avg	Host Max	NodesAvg	Nodes Max
Kamov VFC	920	1662	1116	2057
Kamov OC	182	207	209	254
Eurotown VFC	38	152	178	387
Eurotown OC	117	147	109	158
R. room VFC	14	21	124	171
R. room OC	39	53	92	137

APPENDIX 2

	Host Avg	Host Max	Nodes Avg	Nodes Max
VGA typ VFC	38	152	178	387
VGA typ OC	117	147	109	158
VGA rot VFC	235	555	1629	2165
VGA rot OC	206	287	477	607
XVGA typ VFC	95	257	1638	3359
XVGA typ OC	275	342	592	864
XVGA rot VFC	761	1709	9952	11655
XVGA rot OC	500	725	2498	2879
HDTV typ VFC	127	381	2614	3607
HDTV typ OC	441	509	1121	1355
HDTV rot VFC	980	2489	23077	27325
HDTV rot OC	618	996	5458	5975

Color Figure 1 (left). The image shows an overview of Eurotown. The pink cubes indicate the sampling locations from which the depth and color panoramas were acquired. A panorama consists of 6 depth images with 90 degree horizontal and vertical fields of view that correspond to the faces of a cube. The depth images are 1K x 1K in the case of VGA output (70 deg horizontal field of view) and 2.8K x 2.8K in the case of HDTV (70 deg FOV). The total space consumed by all of the depth images is 4.33GB and 26.06GB respectively, with lossless RLA compression. The blue lines define the cells that subdivide the viewing volume.

Color Figure 2. The two images were rendered on the WarpEngine simulator using two spherical depth-panoramas of the reading room acquired with the *DeltaSphere*, our laser rangefinder.



Color Figure 3. The left image shows the nine depth-images used to model the Kamov helicopter. The right image was rendered on the WarpEngine simulator.