Florida Institute of Technology
Scholarship Repository @ Florida Tech

Link Foundation Modeling, Simulation and Training Fellowship Reports | Link Foundation Fellowship Reports

11-5-2001

The Vacuum Buffer

Voicu Popescu

Follow this and additional works at: https://repository.fit.edu/link_modeling

Part of the Computer Sciences Commons

Voicu Popescu
Department of Computer Sciences
1398 Computer Science Building
West Lafayette, Indiana, 47907-1398

11.05.2001

Marybeth Thompson
Link Administrator,
Simulation and Training

Hello Marybeth,

Please find on this CD-ROM the electronic version of my manuscript "The Vacuum Buffer". I would like to thank again the Link Foundation for its generous support. Also thank you for all your help.

The Vacuum Buffer algorithm was published as part of the Proceedings of the Symposium on Interactive 3D Graphics, Chapel Hill, 2001. The current manuscript contains an extra section that was not published before.

Regarding the color illustration. It is currently the last page of the document. I am aware of the fact that the publication guidelines are discouraging color photographs. I am also aware of the fact that the figures have to fit on a 4.5" wide page. However the color illustration is very important for the paper and I am asking you to kindly help me figure out a way of including it. Please do not worry about the caption, I will gladly rewrite it to fit any arrangement. Also I have highlighted the reference to the figure with red in the text.

I hope the document is as close as possible to its final form and I apologize again for the delay in getting it to you (I have started as an assistant professor at Purdue in August and my baby daughter was also born this summer (late July) so these two projects took about 200% of my time.) Should it be needed, I will promptly accommodate any changes you might recommend.

Also included is a BW face-and-shoulders photo.

Thank you,

Voicu

THE VACUUM BUFFER

V. Popescu and A. Lastra
University of North Carolina at Chapel Hill
{popescu, lastra}@cs.unc.edu

ABSTRACT

Image-based rendering (IBR) techniques have the potential of alleviating some of the bottlenecks of traditional geometry-based rendering such as modeling difficulty and prohibitive cost of photorealism. One of the most appealing IBR approaches uses images enhanced with per-pixel depth and creates new views by 3D warping (IBRW). Modeling a scene with depth images lets one automatically capture intricate details, which are hard to model conventionally. Also, rendering from such representations has the potential of being efficient since it seems that the number of samples that need to be warped is independent of the scene complexity and is just a fraction above the number of samples in the final image. However, selecting the subset of reference-image samples that need to be warped to generate the new view is a very difficult task.

We present the *vacuum buffer* algorithm, and its use within a sample-selection method. Like other techniques, our method proceeds by considering samples of reference images that were acquired from locations close to the current camera position. Unlike other techniques however, our method offers a conservative estimate on whether samples of visible surfaces were potentially missed and it also points to the scene locations where such surfaces might be. The vacuum buffer is essentially a generalized z-buffer and it measures what sub-volumes of the current view-frustum are yet to be determined. Another important difference is that our method uses the current view, which allows it to reduce the number of chosen samples more than other methods that offer a sample-selection solution to be used for several desired camera views. The tradeoff for using the current view is having to solve the sample selection at each frame. By exploiting the coherence in the reference images, groups of nearby samples become the actual primitive, which massively reduces the total cost.

INTRODUCTION

The elusive goal of interactive photorealism and the enormous difficulty of modeling complex scenes with polygons and material-and-light descriptions made 3D-computer-graphics researchers consider images as the rendering primitive. Images capture automatically intricate geometric detail and complex light / material interactions and there is hope that the quality of the photographs can be conveyed to the rendered images. Numerous image-based rendering (IBR) techniques have been developed.

Chen [2] proposes panoramas that, even though they offer a correct 3D view only from one location, are a simple way of modeling and rendering complex natural scenes. Other methods approximate the 3D transformations of scenes with simpler 2D transforms ([11], [13]). The Lumigraph and Lightfield ([4], [5]) approaches create a database of rays that is queried at rendering time. Unfortunately the databases grow to impractical sizes for complex scenes.

McMillan suggested enhancing the photographs with per pixel depth [7]. The depth, the pixel coordinates and the camera pose uniquely locate the color sample in 3D space so samples can be reprojected (warped) at will on new images. The IBR by warping method (IBRW) is essentially reusing the original rays, under the lambertian-reflectance-model assumption. The method is general and it has reasonable storage / memory bandwidth requirements.

The method also has the potential of being very efficient if only one can determine a small set of reference-image samples that suffice to reconstruct the current view. Ideally the number of samples needed would not depend on the scene complexity but only on the size of the output image. Finding such a set however is a challenging problem. When an

input (reference) image is warped, surfaces that were not originally visible can become disoccluded due to motion parallax, and gaps form in the warped image (disocclusion errors). The gaps need to be covered with samples from other reference images. Also the sampling of surfaces from reference to output (desired) image changes differently for every surface.

This paper presents the vacuum-buffer algorithm, and then its use within a new sample-selection method for IBRW. The vacuum buffer measures what sub-volumes of the current view-frustum are yet to be determined, by storing *z-z spans*.

Related Work

One simple solution is to just warp all samples of reference images that were taken from locations nearby the desired camera position. Complicated scenes require dense sampling with reference images and this approach generates too many samples.

Layered Depth Images (LDIs) bring substantial improvement [12]; they generalize the concept of a depth image by allowing for more than one sample along a ray. Consequently an LDI can store samples of surfaces that are hidden from the view of the LDI. As the view changes, the originally hidden samples become visible, avoiding disocclusion errors. Since there are only few samples in the deeper layers, the total number of samples in an LDI is only marginally larger than the number of samples in an equivalent depth image. The LDIs are constructed as a preprocess by warping reference images to the view of the LDI and discarding samples that warp at the same location and at the same depth.

An important question is what reference images need to be combined in an LDI in order to completely eliminate disocclusion errors? One approach is to combine many

regularly spaced reference images, hoping that all potentially visible surfaces are sampled in at least one of the reference images ([12], [9]). Such an approach can evidently miss surfaces. Another concern is that an LDI offers only one sampling rate for a particular surface, which has to be adapted to the desired-image sampling rate at rendering time. Chang addresses this problem by using a tree of LDIs [1] and choosing the appropriate resolution level according to the desired-image sampling requirement for each surface. Also since LDIs are used to recreate several views, an LDI will inherently contain samples that are hidden for a particular view, and thus are unnecessarily warped.

Another important motivation in looking for a new sample-selection technique is the interest in designing and building hardware for accelerating IBRW. LDIs are complicated structures that cannot be easily warped in hardware.

Our sample-selection technique is based on the vacuum buffer algorithm that conservatively decides whether a set of reference-image samples is sufficient for a particular desired view. If not, the algorithm will indicate where in the scene surfaces might have been missed. The next section presents the vacuum buffer algorithm in detail and the following section describes its use for choosing reference-image samples to adequately reconstruct the desired view.

THE VACUUM BUFFER ALGORITHM

The main idea behind the vacuum buffer algorithm is to determine the subvolumes of the desired view frustum that could contain visible surfaces that were missed by the current set of reference images. We call these undetermined subvolumes *vacuum.*

The vacuum buffer algorithm makes use of information contained in depth images that is usually ignored. The depth to the sample has been used to warp (reproject) the sample to

the output image but it also tells us the distance to the *first surface* in the reference image. Consequently we know that there are no other occluders between the center of projection of the reference image and the surfaces sampled. This must have been empty space.

Figure 1 shows a reference image with center of projection R used to reconstruct the desired image with center of projection D. The scene is shown by the line $A_0A_1 \ldots A_5$. The air area (volume in 3D) is determined as being free of occluders by the reference image R. For simple referencing we call it *air*, since in most scenes it indeed corresponds to air. The desired image D sees part of the air seen by the reference image R and that subvolume of the desired image is determined as empty.

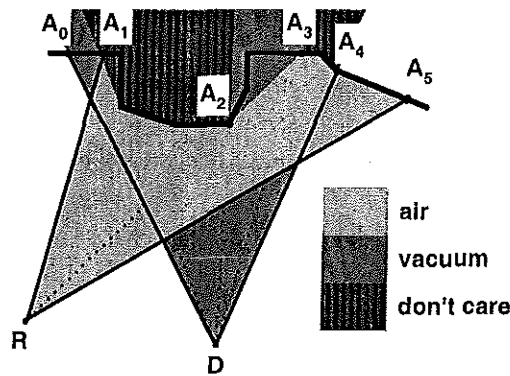


Figure 1. The reference image R is used to determine the volume of the view frustum of the desired image D

The A_1A_2 segment of the scene is a connected opaque surface (occluder) and although other surfaces might be located behind it as seen from D, such surfaces cannot affect the desired image since they are hidden. The "shadow" that is cast in the desired-image view-frustum by occluders define a *don't care* subvolume. For the purpose at hand, don't care subvolumes are equivalent to air volumes.

The *vacuum* subvolumes could contain surfaces that are not sampled in the reference image R and are visible in the desired image D. Vacuum is not a guarantee that visible

surfaces were missed. In figure 1 for example, the vacuum zone close to D might resolve to air when an appropriate reference image that encompasses it is used. However vacuum zones *might* contain visible surfaces and one has to resolve them.

Considering the case presented in figure 1, one could naively consider that the problem of missing samples could be detected by searching the frame buffer for pixels to which no sample mapped. Indeed the framebuffer will contain a gap between the new positions of A_2 and A_3 but missing samples can occur even when the framebuffer is instantiated as seen in the simple case presented in figure 2.

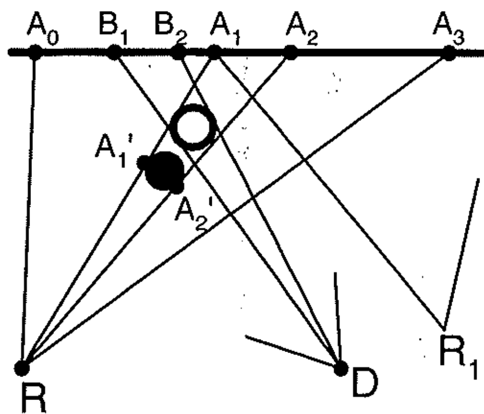


Figure 2. The reference image R samples the surfaces $A_0A_1A_1'A_2'A_2A_3$. It does not sample the hollow sphere. When used to reconstruct the desired image D, the hollow sphere projects to B_1B_2 and the corresponding pixels are already instantiated with a fragment of A_0A_1. Choosing another reference image to fill the gap between A_1 and A_2 might not suffice to reveal the missing object as is the case of reference image R_1.

Algorithm Overview

Given a set of reference images and a desired view, the algorithm computes the amount and location of vacuum that remains after all reference images are used. Initially the entire view frustum of the desired image is undetermined, thus filled with vacuum. The air and the occluders of each reference image are used to resolve vacuum by intersecting

the vacuum with the air and with the shadows of the occluders. The volume intersections are computed efficiently using a generalized z-buffer, which we call the *vacuum buffer*.

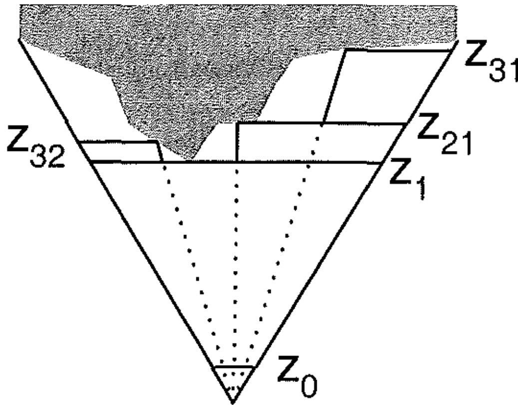


Figure 3. 2D view of the quadtree subdivision of the reference image frustum. Only three levels are shown. At each subdivision one of the children will have the same closest z as its parent so three new frusta are created with each subdivision (one in the 2D view shown).

The vacuum buffer is similar to the z-buffer but it stores z intervals, or *spans*, along each ray, whereas the classic z buffer stores a single value. The list of z spans at one vacuum buffer location corresponds to the vacuum remaining in the view frustum along that particular ray. The method of intersecting volumes using rasterization buffers was first used in constructive solid geometry ([14], [10] and others).

The algorithm first processes the air of the current reference image. As a preprocess, the reference image is recursively subdivided in quadtree fashion and the closest z values are precomputed for each subregion. This subdivides the reference image frustum in subfrusta as seen in figure 3. The first frustum is defined by the hither plane and the plane of closest z. The next level frusta are defined by the parent-region's closest z plane and each subregion's closest z plane.

The vacuum buffer algorithm processes the frusta recursively, starting from the root of the quadtree. The frustum composed of six quadrilateral faces is transformed, projected to

the desired image plane and each of the six faces are scan-converted. Since the frustum is a convex polyhedron, a vacuum buffer location will be either hit twice or not hit at all. When a vacuum buffer location is hit by two samples of the faces of the air frustum, the list of intervals is updated by eliminating the vacuum according to the span of air between the two samples (see figure 4).

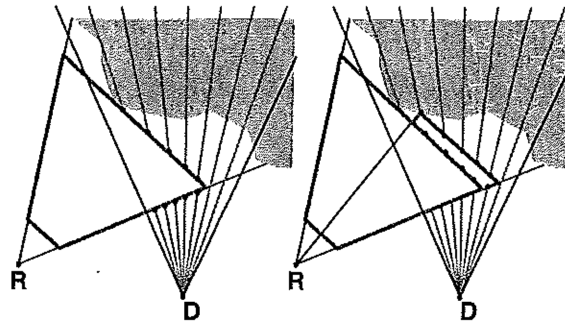


Figure 4. Initially, each vacuum buffer location contains the span (*hither, yon*) since nothing has been determined. The figure on the left shows the vacuum buffer after the first level frustum of the reference image R has been processed. Several locations contain two spans since part of the vacuum has been determined. The right figure shows the vacuum buffer after the second level frustum has been processed.

The next important question is when to stop the recursive subdivision of the reference image frustum. The lower the level of the recursion, the smaller the progress (amount of eliminated vacuum) while the cost increases exponentially. On the other hand, stopping the recursion early doesn't use all the information available in the reference image; some of the air information is wasted.

There is another consideration crucial for deciding what the smallest subregion (tile) of the reference image should be. Remember that the second step of the algorithm is to ignore all vacuum behind occluders. If the tiles are small enough they are, in general, part of the same occluder and one can ignore everything behind the desired image projection of the tile. However computing the exact desired image projection of the tiles is expensive and is equivalent to warping the tile. This obviously defeats the purpose when

the vacuum buffer algorithm is used to choose the tiles needed for the current frame. In order to compute the desired image projection of tiles efficiently we replace the height field corresponding to the tile with a single quad. All vacuum behind the projection of the quad is eliminated (see figure 5).

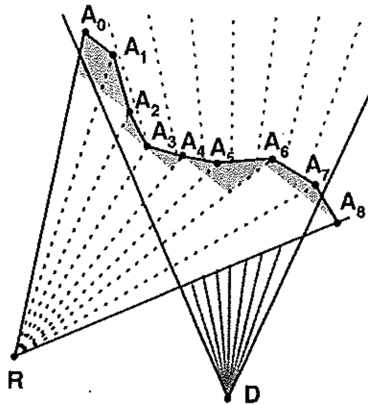


Figure 5. The reference image is split into eight tiles. The leaves of the quadtree are frusta defined by the closest-z plane of the previous level and the quads (here A_kA_{k+1} segments) that approximate the height field of the tile. The leaf frusta (see shaded area), are processed similarly to the other frusta. The occluding faces of the frusta are used to eliminate all vacuum behind them (the dotted rays in D's frustum indicate the eliminated vacuum).

If the tiles are too large then the approximation is too coarse and one can incorrectly eliminate vacuum where unsampled surfaces could potentially hide. We found that 16x16 sample tiles are adequate for a variety of test scenes. Moreover, 16x16 tiles are rendering primitives favorable to efficient IBRW architectures [8].

No matter how small the rectangular tiles, there will be some tiles that stretch from one object to the other. Such silhouette tiles that are not patches of a single occluder cannot be assumed to be a continuous surface and the vacuum behind them cannot be eliminated. We detect these tiles by estimating the depth discontinuities in the reference image, using the method described in [8]. Tiles with depth discontinuities are still useful for the first phase of the algorithm when the air of reference images is used to resolve vacuum from

the vacuum buffer. At the second step, depth discontinuity tiles can be ignored, relying on tiles from other reference images to sample the two occluders independently. However for small features, it is very likely that no reference image will have tiles that map entirely on those small features. Thus one should segment the tile according to the surfaces sampled. The resulting tile segments will each model one object, and they can be treated as regular tiles. Note that tile segmentation does not depend on the desired view so it can be done as a preprocess. The next subsection summarizes the algorithm and presents implementation guidelines.

Algorithm Implementation

```
1. For each reference image
        1.1. Compute depth discontinuities
        1.2. Segment tiles
        1.3. Build quadtree of frusta
2. Initialize vacuum buffer
3. Initialize vacuum accounting tree (VAT)
4. Clear item buffer
5. For each reference image
        5.1. ProcessFrustum(F0)
6. Done: VAT measures and locates the vacuum left in the view frustum
```

The recursive ProcessFrustum(Frustum *F) routine is summarized next:

```
1. if (F == null) return
2. if F not leaf and F->closestZ is F->parent->closestZ
        2.1. go to 6
3. Transform, clip, and project frustum
4. Scan-convert faces in item buffer
        4.1. if item buffer location hit first time
                4.1.1. UpdateZBuffer(z0)
                4.1.2. UpdateItemBuffer(F)
        4.2. if item buffer location hit second time
                4.2.1. dv = UpdateVacuumBuffer(z0, z1)
                4.2.2. UpdateVAT(dv)
```

```
        4.3. if current face is occluder
                4.3.1. dv =UpdateVacuumBuffer(z0, yon)
                4.3.2. UpdateVAT(dv)
5. if F is leaf
        5. 1. ProcessFrustum(F->next)
6. else
        6.1. for (i = 0; i < 4; i++)
                6.1.1. ProcessFrustum(F->child[i])
```

The reader will recognize the algorithm described previously. The resolution at which vacuum is determined, in other words, the number of rays in the vacuum buffer, does not have to be the resolution at which desired images will ultimately be produced. A lower vacuum-buffer resolution can be used (we used 320x240 for VGA resolution output). However, the resolution cannot be lowered indefinitely since the projection of the quads that approximate the tile height-field need to have meaningful sizes and shapes.

The VAT (vacuum accounting tree) is a fast way of knowing how much vacuum is left in the desired-view frustum and where it is located. It is a quadtree subdivision of a buffer of vacuum buffer resolution. A leaf node stores, for every vacuum buffer location, the sum of the vacuum spans in that linked list. In other words a leaf stores the amount of vacuum along a certain ray of the vacuum buffer. A higher level node stores the sum of the vacuum stored at its four children. The root stores the amount of vacuum left in the entire vacuum buffer. The VAT is initialized to *full* once per desired view: vacuum from hither to yon along all rays in the vacuum buffer. It is then updated as frusta are processed, using the amount of vacuum determined by the current air-faces sample pair or current occluder sample.

The item buffer is also at the vacuum buffer resolution and it stores unique frusta identifiers. It is used to detect second hits of samples from the faces of the same frustum.

Since the frusta identifiers are unique, the item buffer does not need to be cleared from one frustum to the next. The z-buffer, also at the vacuum buffer resolution, is used to record the z of the first hit. When the second hit occurs, the value in the z-buffer and the new z value are used to update the vacuum buffer. Updating the vacuum buffer is equivalent to subtracting the newly determined interval from the vacuum intervals stored in the list at that location. The amount of vacuum determined is returned and used to update the VAT. The z-buffer does not need to be cleared since it is always used in conjunction with the item buffer.

The tiles that are segmented generate two or more frusta at the leaf level of the reference image quadtree. In this case the the additional frusta also need to be processed (ProcessFrustum(Frustum *F)). The frusta are processed recursively. If the current frustum has the same closest z as its parent, it doesn't need to be processed since no progress can be made (ProcessFrustum(Frustum *F), step 2).

A case that needs to be treated carefully is the case of frusta clipped by the hither plane. New faces need to be introduced to make sure that the faces that are scan-converted indeed form a convex polyhedron; scan-converting a polyhedron without a face incorrectly misses second hits since one can "see inside" the polyhedron.

One might wonder what happens when only one hit occurs, or several hits occur, due to potential scan-conversion precision limitations. If only one hit occurs, typical for the desired-view silhouette of the frustum, there will be no air span generated to update the vacuum. This is the correct degenerate-case behavior. If more than two hits occur, typical for edges of the frustum that project over another face, the vacuum buffer is updated

using tiny air spans. The results are correct and the sole penalty is in efficiency. We avoid such cases by setting a threshold below which air spans are discarded.

From figure 4 one can see that the typical vacuum buffer update is done with adjacent air spans. To insure the adjacency in the presence of numerical error, the vacuum-buffer update routine starts by increasing the air span with epsilon at each end. Adjacency is of course important since it keeps the vacuum-span lists short, which translates into efficient update times.

The next section presents the application of the vacuum buffer algorithm to reference-image sample choosing, called *tile choosing* since tiles are the level at which samples are selected.

TILE CHOOSING

In the introduction, we discussed the requirements for a sample-selection method. An ideal set of samples satisfies the following conditions:

- *completeness*: all surfaces visible in the desired view should be represented
- *good quality*: the resolution at which the surfaces are sampled in the reference images should be as close as possible to the resolution at which the surfaces are sampled in the desired view; *oversampling* leads to aliasing artifacts and high cost, and *undersampling* leads to poor quality (blurriness)
- *non-redundancy*: invariably some surfaces are sampled in more than one reference image; assuming that the surfaces are close to diffuse, the multiple samples are equivalent and one can / should discard the redundant copies.
- *low depth complexity*: for efficiency, the set should contain very few or no samples that are hidden in the desired image.

For the reasons discussed previously, we split the reference images into tiles, and the set of samples that has to be determined is actually a set of tiles. It is obvious that the vacuum buffer measures the completeness of a set of tiles but more needs to be said in order to explain how it can help satisfy the other three conditions.

We measure the quality of the samples in a tile by analyzing how much it stretches or shrinks when projected in the desired image. Small changes indicate similarity between the sampling rates in the reference and output images, thus the samples of such a tile are preferred. From an implementation point of view, the quality is efficiently derived from the size of the bounding box of the projection of the quad corresponding to the tile. Having defined a quality metric we need to explain how to determine whether two samples originating from different reference images sample the same surface.

In order to do this we utilize an additional z-buffer and an additional item buffer at the resolution of the vacuum buffer that simply store at each location the z of the closest occluder and a pointer to the tile from which it originates. When an occluding face is processed, the occluder z-buffer is consulted. If the current occluder sample is within epsilon of the closest occluder z, it is assumed to sample the same surface. If the quality of the current tile is better, the occluder item buffer location is overwritten, otherwise the current sample is simply discarded. This eliminates redundant samples of lesser quality.

If the current occluder sample is clearly behind the closest occluder sample, the current sample is again discarded. This reduces the depth complexity.

The tiles that could not be segmented in order to avoid internal depth discontinuities cannot be processed by the algorithm, and, conservatively, have to be chosen.

The tile-choosing algorithm starts with the reference images that were acquired from a location closest to the desired camera location. More and more distant reference images are processed until the amount of vacuum remaining is below a certain threshold. The chosen tiles are the tiles that have at least one sample present in the occluder item buffer: they were not completely occluded nor were they completely replaced by better tiles. In order to avoid scanning through the occluder item buffer, we maintain presence counters for each tile. The color figure illustrates tile choosing using the vacuum buffer algorithm

Results

We tested the tile choosing technique on a complex model of a town - *eurotown* - (seen in the accompanying videotape). The reference images used as input were rendered using the 3D Studio Max modeling package. The reference images were rendered from locations that form a regular 3D grid; the grid subdivides the viewing volume in equal cells (boxes).

Our first version of the tile-choosing algorithm did not use the vacuum buffer. It just considered the 8 sampling locations defining the cell of the current camera position and rendered the tiles in the occluder item buffer and the occluder z-buffer. In order to minimize the chances of missing a visible surface the cell size was small (3mx3mx3m for *eurotown*). Also no tile segmentation was attempted.

With the vacuum buffer we were able to double the sides of the cell. This equates to 8 times fewer reference-images, which is obviously of great importance when real world data is acquired. As in the previous case, the tile choosing starts by considering the images of the current cell. Tile choosing stops when the total amount of vacuum decreases below a threshold. We used 1/z for the vacuum buffer and the initial (hither,

yon) vacuum span was (100.0, 0). Using 1/z is convenient since it gives more importance to vacuum spans that are close to the camera. Objects that are close have a large screen area and the artifact resulting from missing them is more noticeable. In our particular case, the threshold below which no more disocclusion errors were noticeable was 1500 (the average amount of vacuum per location is about 0.02). Increasing the size of the cell means that occasionally one had to consider images from neighboring cells since the vacuum would not decrease below the set threshold only by using the reference images of the current cell.

Tile segmentation was done, allowing a maximum of 6 segments per tile. All tiles that cannot be segmented are conservatively chosen. The following table presents the results of tile choosing for both the old and the current method with output at VGA resolution.

	old	**current**
Visible tiles	15K	13K
Selected tiles & tile segments	7K	3.3K
Selected unsegmented tiles	2.5K	1.8K
Selected tiles that are segmented	0	1.1K
Selected tile segments	0	1.5K
Unsegmentable tiles	4.5K	0.4K

First, there are fewer visible tiles since the reference images are further apart. The number of selected tiles (including segments) is much lower mainly because the number of unsegmentable tiles decreased substantially. The number of selected tiles that did not have to be segmented (no depth discontinuities) also decreased since the tile segments eliminated some of the full tiles.

In a VGA-resolution image there are 1.2K 16x16-sample tiles. So the ratio between the number of reference-image samples selected and the number of output image samples is 2.75. We counted 256 samples per tile-segment having in mind a hardware architecture like the WarpEngine, that processes a tile in SIMD fashion and one cannot save work if some samples are "off". Other factors that make this figure deviate from the ideal value of 1.0 are:

- The closest reference-image sampling rate for a surface is not *exactly* the desired image sampling rate
- There are still a few tiles that are not segmented, which cause undetected redundancy.

The next section analyzes the complexity of the algorithm and investigates possible hardware acceleration.

HARDWARE ACCELERATION FOR VACUUM BUFFER ALGORITHM

As stated earlier, the resolution of the buffers used in the algorithm can be lower than the resolution of the final image. Thus clearing the buffers (once per frame) is not a substantial burden. The bulk of the work is done at step 4 of the ProcessFrustum routine. If at most eight sampling locations are considered at each frame, and if cube depth-panoramas centered at the sampling location are used, there could be as many as 6 x 8 = 48 reference images that need to be considered.

Of course not all reference images have samples that project into the view frustum. At step 3, if the current sub-region of the reference image is completely outside the view frustum, it is culled and the recursion is stopped early. In order to determine whether the sub-region is needed, one transforms, clips and projects a second bottom face of the

frustum, defined by the *farthest* z. If the larger frustum defined by the parent's closest z plane and the farthest z is outside of the view frustum, the subregion is not useful for the current view and can be safely ignored (see figure 6). In our test scenes, using 1K x 1K reference images, on average 13K 16x16-sample tiles passed the view-frustum-culling test. This is equivalent to 3-4 full reference images.

The number of frusta that have to be processed for each full reference image is:

$$N = F_{1024} + 3F_{512} + 4(3F_{256} + 4(\ldots + 4(3F_{32} + 4F_{16}))) = 5120,$$

where F_w is a frustum that corresponds to a w x w subregion. Analytically, the number of frusta can be computed with the formula

$$N = 4^k + 4^{k-1} = 5120$$

where k is defined as

$$k = \log_2 w_{image} - \log_2 w_{tile} = \log_2 1024 - \log_2 16 = 6$$

Each frustum has 6 quadrilateral faces, thus, the number of triangles that need to be rendered per second, assuming 4 reference images worth of tiles and 30Hz update rate is:

$$T = 30 * 4 * 6 * 2 * 5120 \approx 7Mtris/s$$

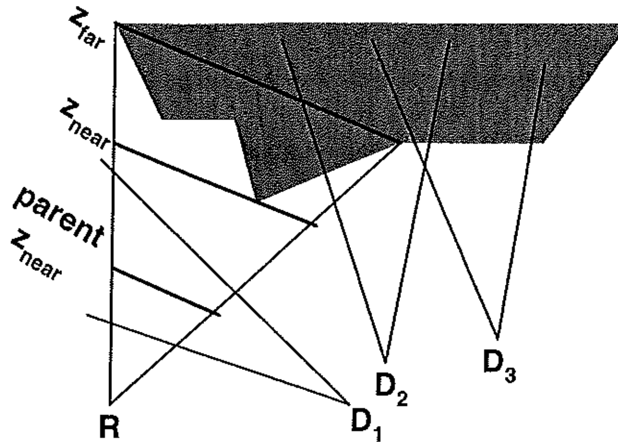


Figure 6. The figure shows the current subregion of the reference image R and three possible desired views. In order to decide whether the subregion is relevant to the desired view, one needs to consider the frustum defined by the planes *parent* z_{near} and z_{far}. Using the planes z_{near} and z_{far} can incorrectly rule the region as irrelevant. For example the air information of the subregion is relevant for D_1 although the frustum z_{near} - z_{far} does not intersect the D_1 frustum. Using the planes *parent* z_{near} and z_{near} is also incorrect as shown by D_2. If the *parent* z_{near}- z_{far} frustum does not intersect the view frustum, the subregion can be safely ignored, as shown by D_3.

Our software implementation takes about 20 seconds per frame, which is orders of magnitude too slow for interactive rendering. The average number of triangles rendered per frame for the eurotown scene was *263Ktris,* which represents *7.8Mtris/s*, confirming the estimate above.

The obvious way to accelerate the vacuum buffer is to use polygon-rendering hardware. Besides the fact that *7Mtris/s* is a sizeable task even for the most recent polygon-rendering hardware, there is the problem of updating the vacuum buffer.

Graphics hardware doesn't offer the possibility of storing several z values at each location. Having the CPU read back the z-buffer and item buffer after each face of a frusta is rendered is not a practical solution. First, the operation is very costly on most graphics architectures. Second, the CPU would have to traverse the bounding box of the

projection of the face in order to use the z's of the current face, which most likely offsets the advantage of hardware rendering.

We believe that the most promising solution is to design hardware for accelerating the vacuum buffer. The architecture would be similar to current triangle rendering architectures. However, the rendering task consists only of flat shaded triangles, so the architecture will be simpler in many ways.

The only additional complexity is a z-buffer that can store several samples at each location. Because of the incremental updates to the vacuum buffer, the number of z's that need to be stored is small, and it seems practical to provide storage for a fixed number of z values. The next table shows (for our scene), how many times per frame, on average, a list grew longer than a certain value. For example on average, at each frame, there were 118 lists that grew from length 7 to length 8.

Length	4	5	6	7	8	9	10	11
Exceeded	2785	925	399	118	28	5	2	0

If the limit is exceeded, exceptions can be raised that discard the shortest interval or eliminate the smallest gap by merging nearly adjacent intervals (whichever is smaller). From our simulations, if the maximum hardware list length is set to 8, the amount of vacuum incorrectly eliminated to serve the exception was negligible and the same set of tiles was chosen as in the case of unlimited list lengths. Since the resolution of the vacuum buffer is low, it is feasible to store a maximum of 8 spans at each location.

CONCLUSIONS

We presented a method of selecting reference samples to be warped to create the desired view. The method uses the vacuum buffer algorithm to conservatively estimate the

subvolumes of the view frustum that have not been determined by the reference images considered so far. A software only implementation is too slow to be practical. The bulk of the work consists of rendering polygons and the number of polygons that need to be rendered per second is within the capabilities of today's hardware. The modification required is to extend the z-buffer to contain several spans of z. The number of spans at a vacuum buffer location is small. This enables a simple hardware implementation that supports vacuum-span lists of fixed maximum length.

FUTURE WORK

The current version considers the reference images of the current cell in the order defined by the increasing distance to the camera position. The residual amount of vacuum should decrease more rapidly if we:

- First process the reference image that sees the desired view location and has the view direction the closest to the desired image view; this eliminates all the vacuum close to the desired view camera
- Use the VAT to localize the remaining vacuum and then select the reference images that see that vacuum

This reference image ordering might allow us to space the reference images even farther apart. This brings us to what we believe is another important application of the vacuum buffer algorithm.

Determining Sampling Locations

An obvious goal in computing the locations where the reference images should be placed is to acquire as few images as possible while minimizing the number of disocclusion errors. This is the well known *next best view* problem ([3], [6] and many others). We

speculate on an iterative process based on the vacuum buffer algorithm that automatically produces a set of candidate sampling locations.

First, the scene is subdivided according to a regular 3D grid. A cell of this grid would be a cube of side 0.5m for a room or 20m for a town. Complete panoramas (6 faces of a cube) should be acquired at each grid node. Then, for each cell, the vacuum buffer algorithm is executed at locations that form a more refined grid, with nodes that are let's say .1 m apart. The reference images used are the reference images acquired at the corners of the current cell and the cells adjacent to the current cell. The node of the refined grid at which the most vacuum remains undetermined is a new sampling location. The residual vacuum at the refined grid locations is computed again. Again the location with the most undetermined vacuum becomes a new sampling location. The process stops when the budget of sampling locations / cell is reached or when the maximum residual vacuum value dips below a tolerable threshold. The greedy approach described does not provide the optimal solution but it is fully automatic and the solution might be good enough.

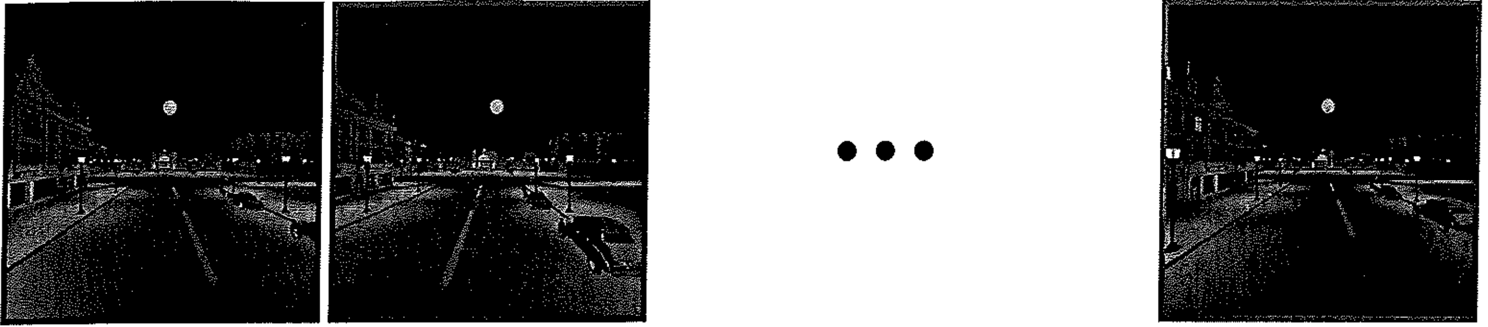
ACKNOWLEDGEMENTS

We would like to thank Leonard McMillan and the UNC IBR group for useful suggestions. This work was supported by the Link Foundation, and Intel Corporation.

REFERENCES

[1] C. Chang, G. Bishop. and A. Lastra, "LDI Tree: A Hierarchical Representation for Image-Based Rendering", *Proceedings of SIGGRAPH 99*, 291-298 (1999).
[2] S. Chen, "Quicktime VR - An Image-Based Approach to Virtual Environment Navigation", *Proceedings of SIGGRAPH '95*, 29-38 (1995).
[3] J. Connolly, "The determination of next best views", *IEEE International Conference on Robotics and Automation*, 432-435, (1985).
[4] S. Gortler, R. Grzeszczuk, R. Szeliski, and M. Cohen, "The Lumigraph", *Proceedings of SIGGRAPH '96*, 43-54 (1996).

[5] M. Levoy, and P. Hanrahan, "Light Field Rendering", *Proceedings of SIGGRAPH '96*, 31-42 (1996).
[6] J. Maver and R. Bajcsy, "Occlusions as a guide for planning the next view", *IEEE Transactions on Pattern Analysis and Machine Intelligence,* 15(5):417-433 (1993).
[7] L. McMillan and G. Bishop, "Plenoptic Modeling: An Image-Based Rendering System", *Proceedings of SIGGRAPH '95*, 39-46 (1995).
[8] V. Popescu, J. Eyles, A. Lastra, J. Steinhurst, N. England, and L. Nyland, "The WarpEngine: An Architecture for the Post-Polygonal Age", *Proceedings of SIGGRAPH '00*, 433-442 (2000).
[9] V. Popescu, A. Lastra, D. Aliaga, and M. Oliveira, "Efficient Warping for Architectural Walkthroughs using Layered Depth Images", *In Proceedings of VIS'98*, 211-216 (1998).
[10] D. Salesin, "Rendering CSG Models with a ZZ-Buffer", *In Proceedings of SIGGRAPH '90*, 67-76 (1990).
[11] S. Seitz and C. Dyer, "View Morphing: Synthesizing 3D Metamorphoses Using Image Transforms", *In Proceedings of SIGGRAPH '96*, 21-30 (1996).
[12] J. Shade, S. Gortler, L. He, and R. Szeliski, "Layered Depth Images", *In Proceedings of SIGGRAPH '98*, 231-242 (1998).
[13] J. Torborg. and J. Kajiya, "Talisman: Commodity Real-time 3D Graphics for the PC", *In Proceedings of SIGGRAPH '96*, 353-364 (1996).
[14] T. van Hook, "Real-Time Shaded NC Milling Display", *In Proceedings of SIGGRAPH '86*, 15-20 (1986).



Color Figure: tile choosing with the vacuum buffer algorithm. The reference images processed by the algorithm are shown at the top. The columns show the content of various buffers as the reference images are processed. Row 1 shows the warpbuffer, which is not computed by the algorithm and is shown here just for illustration purposes. It is obtained by warping the tiles chosen so far. Rows 2 and 3 show the vacuum buffer; segments shaded from bright yellow (close end) to dark yellow (far end) represent the vacuum spans. The vacuum buffer is shown from an offset view to expose the rays. The desired view frustum is shown with the four concurrent yellow segments. Every column and every *k* row of the vacuum buffer are shown. In row 3, nearby vacuum is not shown and the number of vacuum-buffer rows shown is increased. The vacuum buffer is shown composited with the warpbuffer. Row 4 shows the occluder item buffer, which stores the tiles chosen so far. Each tile (or tile segment) is shown with a different (random) color.

After the first reference image is added, vacuum persists close to the camera (since outside of the reference view frustum) and in the volumes that were occluded in the reference image and are now disoccluded (e.g. behind light poles). As one more reference image is processed, the vacuum is further reduced and it drops below a pre-established threshold after the last image. The item buffer contains the chosen tiles. In order to match the sampling rate of the desired image, the algorithm gives preference to tiles that have the projected size close to the original size (see the sky tiles which are not affected by the projection).