# The Vacuum Buffer

Voicu Popescu, Anselmo Lastra

University of North Carolina at Chapel Hill

## ABSTRACT

When rendering from depth images, an important and difficult task is selecting the subset of depth-image samples that need to be warped to generate the new view.

We present the *vacuum-buffer algo*rithm, 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. Additionally, our method offers an estimate of whether visible surfaces were potentially missed and points to the scene locations where such surfaces might be. The vacuum buffer is essentially a generalized z-buffer and it measures what subvolumes of the current view-frustum have not been seen by any reference image considered so far.

**KEYWORDS:** image-based modeling & rendering, algorithm.

## 1 INTRODUCTION

In image-based rendering by warping (IBRW), the depth-and-color samples of the reference depth images are reprojected (warped) to create new images [McMillan95]. IBRW has the potential of being very efficient if only one can determine a small set of samples that suffice to reconstruct the current view.

Finding such a set however is a challenging problem. When a 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 rate from reference to 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*.

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

Layered Depth Images (LDIs) [Shade98] 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 to the same location and depth.

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. This problem is addressed in [Chang99] by using a tree of LDIs of increasing resolutions.

An important question is what reference images need to be combined in an LDI in order to completely eliminate disocclusion errors? The approach used is to combine many regularly spaced reference images, hoping that all potentially visible surfaces are sampled in at least one of the reference images. Such an approach can evidently miss surfaces.

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 [Popescu00]. LDIs are complicated structures that cannot be easily warped in hardware.

Our sample-selection technique is based on the vacuum-buffer algorithm that 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 section 3 describes its use for choosing reference-image samples to adequately reconstruct the desired view.

## 2 THE VACUUM-BUFFER ALGORITHM

The basic idea of the vacuum-buffer algorithm is to determine the subvolumes of the desired view frustum that could contain visible surfaces not sampled by the current set of reference images. We call these undetermined subvolumes *vacuum*.

In IBRW, the depth of the sample is used to compute its output-image location. The depth 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.

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 area (volume in 3D) shown in light gray 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.

The empty-space information of depth images was used in building polygonal-models from range data. In [Curless96], the bounding box of a scanned object is subdivided into voxels and then the range images are used to carve out the empty voxels. In the IBRW context, depth images can be exploited even more.

In figure 1, the segment $A_1A_2$ 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

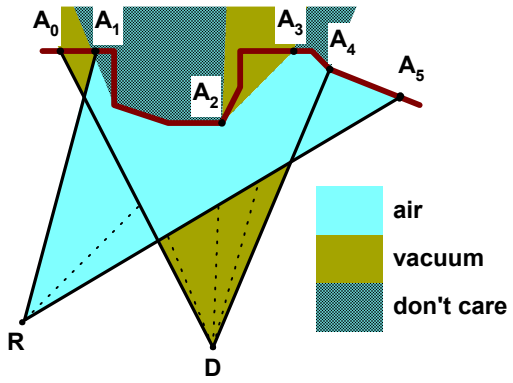



**Figure 1.** The reference image R is used to determine the volume of the view frustum of the desired image D.

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.

Looking at figure 1, one could imagine that we could detect that we are missing samples by searching for uninstantiated pixels in the framebuffer. 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 fully instantiated as illustrated in figure 2.

## 2.1 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 reference images are processed in turn by first intersecting the air and then the occluder shadows with the vacuum.

These volume intersections are computed efficiently using a generalized z-buffer, which we call the *vacuum buffer*. The vacuum buffer stores z intervals, or *spans*, corresponding 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 [vanHook86].

As a preprocess, each reference image is recursively subdivided in quadtree fashion and the closest-z values are precomputed for each subregion. This subdivides the reference-image frustum into air subfrusta (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 and each subregion's closest z.

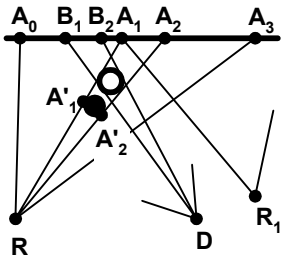



**Figure 2.** R samples the surfaces $A_0A_1A'_1A'_2A_2A_3$. It does not sample the hollow sphere. When used to reconstruct the desired image D, the hollow sphere projects to $B_1B_2$ and the corresponding pixels are already instantiated with a fragment of $A_0A_1$. Just filling the gap between $A_1$ and $A_2$, like from reference image $R_1$, might not suffice to reveal the missing object.

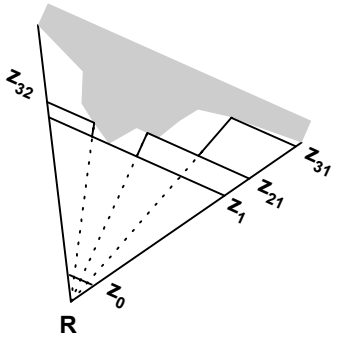



**Figure 3.** 2D view of the quadtree subdivision of the reference image frustum. Only three levels are shown. At each subdivision one of the children will have the same closest z as its parent so three new frusta are created with each subdivision (one in the 2D view shown).

Initially, each vacuum buffer location contains the span (*hither, yon*) since nothing has been determined. We process the reference-image frusta recursively, starting from the root of the quadtree. The six faces of each frustum are scan-converted into the vacuum buffer. Since the frustum is a convex polyhedron, if a vacuum buffer location is hit, it will be hit twice. The two z's define an air span that is subtracted from the list of vacuum spans stored at that location (see figure 4).

The second step of the algorithm is to ignore all vacuum behind occluders. If the reference-image regions at the leaf of the quadtree -- which we call *tiles* -- are small enough, they are, in general, part of the same occluder. Computing the exact desired-image projection of a tile is expensive and is equivalent to warping it. This 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. For surface continuity we use the four corner samples of the tile, which are connected by two triangles. All vacuum behind the projection of the two triangles is eliminated (see figure 5).

If the tiles are too large, the approximation is too coarse and one can incorrectly eliminate vacuum, potentially missing surfaces. We found that 16x16 sample tiles are adequate for a variety of test scenes.

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 [Popescu00]. The tiles are segmented according to the surfaces sampled. The resulting tile segments will each model one object, and they can be treated as

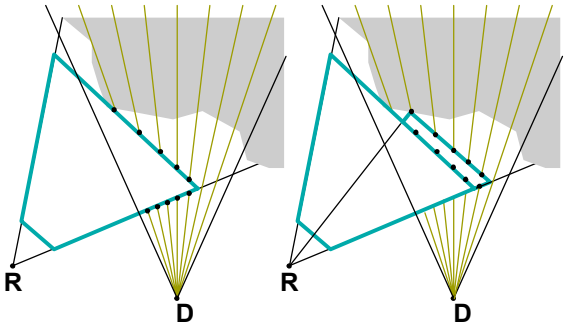



**Figure 4.** The figure on the left shows the vacuum buffer after the first level frustum of the reference image R has been processed. Several locations contain two spans since part of the vacuum has been determined. The right figure shows the vacuum buffer after the second level frustum has been processed.

regular tiles. Note that tile segmentation does not depend on the desired view so it can be done as a preprocess.

### 2.2 Algorithm Implementation

```
1. For each reference image (Preprocess)
     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 remaining vacuum
```

The 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. for (i = 0; i < 4; i++)
     6.1. 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. We used a 320x240 vacuum buffer for VGA output resolution. However, the resolution cannot be lowered indefinitely since the projections of the quads that approximate the tile height-field must have relatively accurate sizes and shapes.

The VAT (vacuum accounting tree) is a data structure for fast lookup of how much vacuum is left in the desired view frustum and where it is located. It is a quadtree subdivision of a buffer at vacuum-buffer resolution. A leaf node stores, for every vacuum-buffer location, the sum of the lengths of the vacuum spans in that linked list. A higher level node stores the sum of the vacuum stored at its four children. The VAT is initialized and updated together with the vacuum buffer.

The item buffer is also at the vacuum-buffer resolution and 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. The amount of vacuum determined is used to update the VAT. The z-buffer does not need to be cleared since it is used in conjunction with the item buffer.

The frusta are processed recursively by ProcessFrustum. 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 (step 2). The segmented tiles generate two or more frusta at the leaf level of the reference-image quadtree (step 5.1).

Although the frusta are convex and should hit the vacuum buffer twice, there are cases of *one* or *more than two* hits because of finite resolution. If only one hit occurs, (at 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.

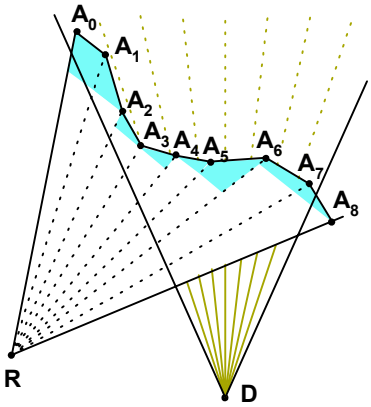



**Figure 5.** The reference image is split into 8 tiles. The leaves of the quadtree are frusta defined by the closest-z plane of the previous level and the quads ($A_kA_{k+1}$ segments) that approximate the height field of the tile. The occluding faces are used to eliminate all vacuum behind them (the dotted rays in D's frustum indicate the eliminated vacuum).

From figure 4 one can see that the typical vacuum buffer update is done with adjacent air spans. 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.

## 3 TILE CHOOSING

An ideal set of samples satisfies the following conditions:

- *completeness*: all visible surfaces must be sampled;
- *good quality*: the sampling rate should be as close as possible to the sampling rate required by the desired image;
- *non-redundancy*: a particular surface should be sampled by only one reference image;
- *low depth complexity*: no surfaces should be sampled that are hidden in the desired view.

For the reasons discussed previously, we split the reference images into tiles, and the sample-selection is done at tile level.

To satisfy the quality condition we define a quality metric for tiles that relies on approximating the sampling-density change from reference to desired image. A similar quality metric was used in [Mark97]. In our implementation, the quality is efficiently derived from the size of the bounding box of the projection of the quad corresponding to the tile.

Together with the vacuum buffer, we use an additional z-buffer and an additional item buffer. The two buffers store the z of the closest occluder sample and a pointer to the tile it came from. When an occluding face is rasterized, the occluder z-buffer is consulted. Close z values indicate tiles that sample the same surface, a case in which we give preference to the occluder sample that belongs to the higher quality tile. If the current occluder sample is clearly behind the closest occluder sample, the hidden sample is discarded, reducing 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 (see color plate). 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 neither completely occluded nor completely replaced by better tiles. In order to avoid scanning through the occluder item buffer, we maintain presence counters for each tile.

### 3.1 Results

We tested the tile-choosing technique on a complex model of a town - *eurotown*. The reference images used as input were rendered from locations that form a regular 3D grid.

Our first version of the tile-choosing algorithm [Popescu00] 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. No tile segmentation was attempted.

With the vacuum buffer we were able to double the lengths of the sides of the cell, reducing the number of reference images by a factor of 8. As in the previous case, tile choosing starts by considering the images of the current cell. It 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). 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).

Tiles were segmented, allowing a maximum of 6 segments per tile. If a tile could not be successfully segmented it was conservatively chosen.

For VGA output resolution, the number of tiles chosen per frame was, on average, 3,300. The ratio between the number of reference-image samples selected and the number of output-image samples is 2.75. This is due to differences in sampling rates from input to output and to tile-level sample selection, which does not allow the elimination of all redundant samples.

## 4 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 estimate the subvolumes of the view frustum that have not been determined by the reference images considered so far. To our knowledge this is the first IBRW sample-selection method that is conservative at the conceptual level; the implementation is not absolutely conservative due to the various optimizations introduced (i.e. approximation of tiles with quads), but good results were obtained.

A software-only implementation is too slow to be practical. We found that the algorithm requires rendering about 7 million triangles per second, which is within the capabilities of today's hardware. However the algorithm cannot be run on existing graphics hardware. The additional capability required is an extended z-buffer that can store several z-spans for each location.

Due to the incremental nature of the updates of the vacuum buffer, the span-lists are typically short. We found that for eurotown, no list grew longer than 11 spans, which suggests that a simple hardware implementation that supports lists of fixed maximum length should be possible.

## 5 FUTURE WORK

The current version considers the reference images of the current cell in the order defined by 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 a view direction closest to the desired image view; this eliminates all the vacuum close to the camera;
- use the VAT to localize the remaining vacuum and then select the reference images that see that vacuum.

This improved heuristic would probably reduce the number of reference images that need to be considered. Note that reducing the vacuum as quickly as possible is orthogonal to finding the samples that best sample a surface for the current view. The current implementation relies on the reference images' proximity to the camera position for finding good samples. More aggressive implementations probably need to consider sample quality as part of their termination condition.

Another possible application of the vacuum buffer is in scene acquisition, as a mean of determining the sampling locations. One wants to acquire as few images as possible while minimizing the number of disocclusion errors. This is the well-known *next best view* problem ([Connoly85] and others).

Given a set of initial reference images, the vacuum-buffer algorithm can be used iteratively to determine the locations of subsequent reference images. For each iteration, first the algorithm is run for a representative subset of all possible viewing locations. Then a new reference-image location is chosen in order to eliminate as much vacuum as possible. The iterative process stops when the maximum number of sampling locations is met or when the scene is adequately sampled.

## 6 ACKNOWLEDGEMENTS

We would like to thank Leonard McMillan and the UNC IBR group for useful suggestions. This work was supported by the Link Foundation, the Department of Energy, NSF ACR-9876914, Intel and Microsoft.

## REFERENCES

[Chang99] Chang C., Bishop G. and Lastra A., "LDI Tree: A Hierarchical Representation for Image-Based Rendering", *In Proceedings of SIGGRAPH 99*, 291-298 (1999).

[Connolly85] Connolly, "The determination of next best views", *In Proceedings of IEEE International Conference on Robotics and Automation*, pages 432-435, (1985).

[Curless96] Curless B., and Levoy M., "A Volumetric Method for Building Complex Models from Range Images", *In Proceedings of SIGGRAPH 96*, 303-312 (1996).

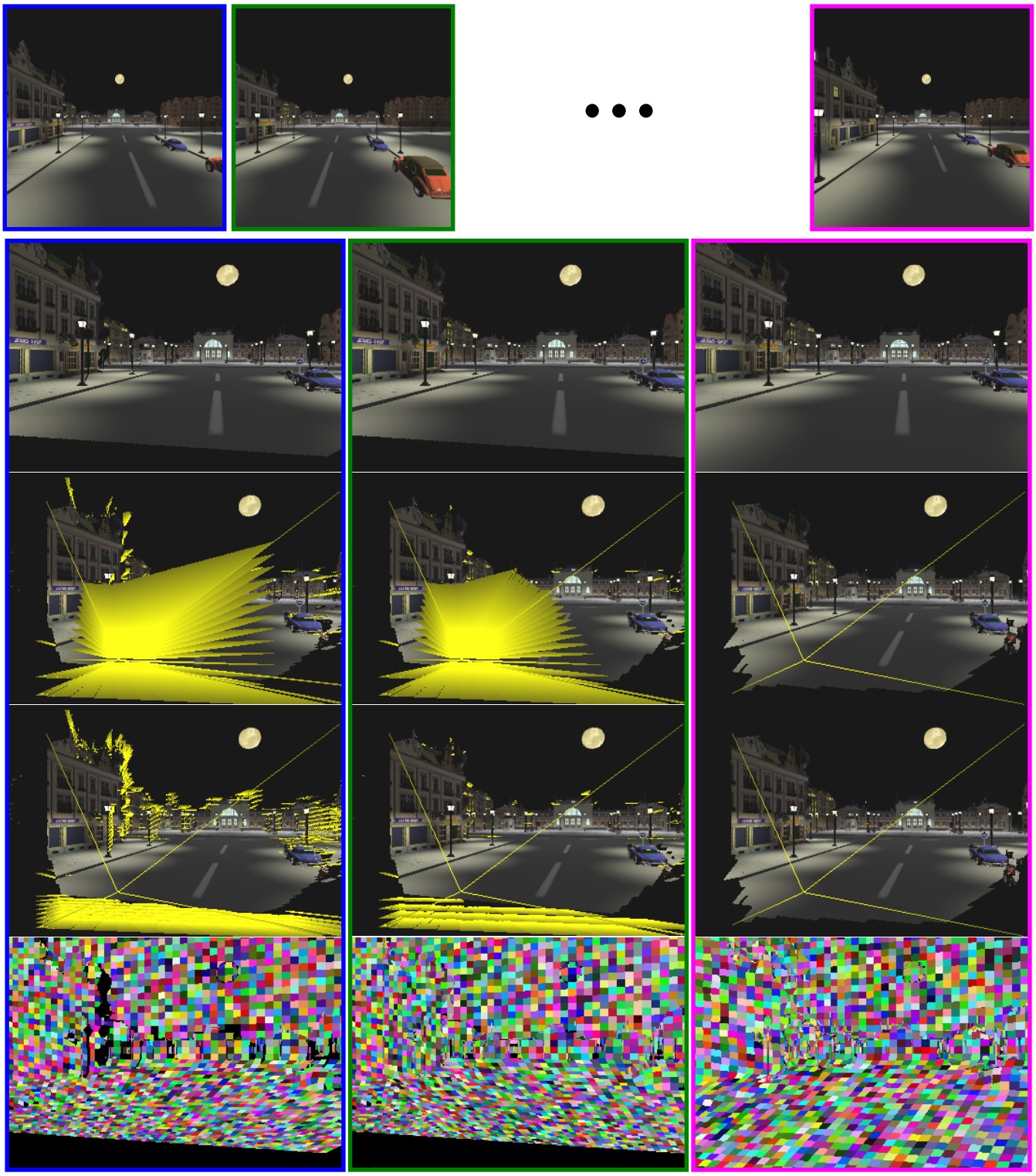
[Mark97] Mark W., McMillan L., and Bishop G., "Post-Rendering 3D Warping", *In Proceedings of Symposium on Interactive 3D Graphics 97*, 7-16, (1997).

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

[Popescu00] Popescu V., Eyles J., Lastra A., Steinhurst J., England N., and Nyland L., "The WarpEngine: An Architecture for the Post-Polygonal Age", *In Proceedings of SIGGRAPH 00*, 433-442 (2000).

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

[vanHook] van Hook T., "Real-Time Shaded NC Milling Display", *In Proceedings of SIGGRAPH 86*, 15-20 (1986).



**The Vacuum Buffer:** ***tile choosing with the vacuum-buffer algorithm***
Some reference images are shown at the top. The columns show results as the reference images are processed. Row 1 shows the result of warping the tiles chosen so far. Rows 2 and 3 show (only every $k^{th}$ row of) the vacuum buffer from an offset view. The vacuum spans are shown in yellow. 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 color. After the first reference image is added (first column), vacuum persists close to the camera and in the volumes that were occluded in the reference image and are now exposed (e.g. behind light poles). As one more reference image is processed (second column), the vacuum is reduced and 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 projected size close to the original size (see the sky tiles which are not affected by the projection).