Relief Textures

Manuel M. Oliveira and Gary Bishop
{oliveira|gb}@cs.unc.edu

Department of Computer Science
University of North Carolina at Chapel Hill
Sitterson Hall, CB# 3175
Chapel Hill, NC, 27599-3175

Technical Report TR99-015
March 29, 1999

ABSTRACT

We have developed a new method for transforming images with per-pixel displacements into textures that have correct parallax when texture-mapped, in the usual way, onto polygons. Our new method results from factoring the 3-D image-warping equations of McMillan and Bishop into a pre-warp followed by standard texture mapping. The pre-warp handles only the parallax effects resulting from the direction of view and the displacement of texture elements; the subsequent texture-mapping operation handles scaling, rotation, and the remaining perspective transformation.

The pre-warp equations have a very simple 1-D structure that enables the pre-warp to be implemented using only 1-D image operations along scan lines and columns and requires interpolation between only two adjacent pixels at a time. This allows efficient implementation in software and should allow a simple and efficient hardware implementation. The texture-mapping hardware already very common in graphics systems efficiently implements the final texture mapping stage of the warp.

We demonstrate a software implementation of the method and show how our relief textures can be used both to add realistic surface detail and to render complex scenes and objects.

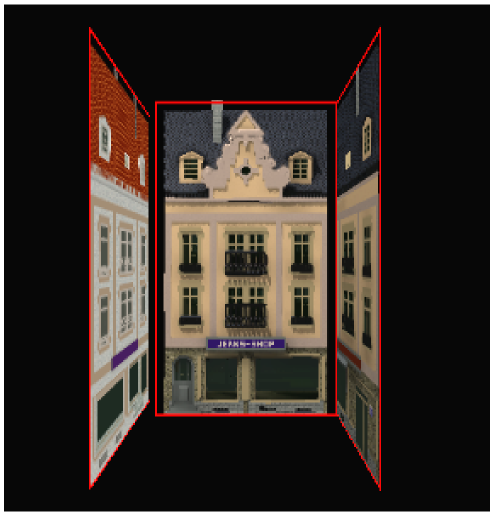
Figure 1. Scene represented with three polygons using conventional texture mapping. The red lines show the borders of the polygons.

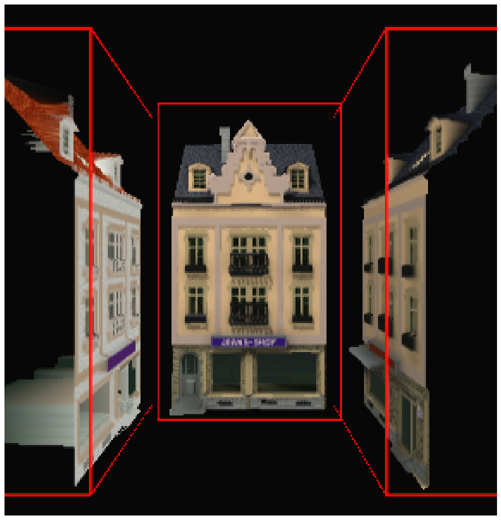
Figure 2. Same scene rendered with three height-field textures and five polygons, from the same viewpoint as in Figure 1. The three original polygons were preserved. Note the dormers.

1 INTRODUCTION

Texture mapping has long been used to enhance the realism of computer-generated images [9] by adding 2-D details to object surfaces. For instance, it can be used to correctly simulate a picture on a wall, or the label on a can. Unfortunately, texture mapping is not as effective for adding 3-D details to a surface. When seen by a moving observer, or by a still observer at an oblique angle, the absence of parallax reveals the flatness of the surface. Consider, for example, the scene shown in Figure 1; it is easy to see that the pictures of the houses have been mapped onto flat polygons.

A much more convincing illusion of 3-D surface detail can be achieved by using a height field in conjunction with a texture map (Figure 2). A height field is a scalar field of distances between surface points and their orthogonal projections onto a plane that forms its algebraic basis.

Unfortunately, rendering height fields is much more difficult than texture mapping. The planar-projective transform of texture mapping has a very convenient inverse formulation. This allows direct computation of texture pixel coordinates from screen coordinates, thus allowing efficient implementation as well as accurate re-sampling and filtering. Height-field rendering allows no such inverse formulation directly. Multiple samples from the height field may be mapped to the same pixel in the final image, depending on the surface shape and the viewpoint. Assuring correct visibility requires either a ray-tracing strategy [12] or a direct forward mapping (*i.e.*, 3-D image warping [12]).

This paper describes a new approach for texture mapping that uses relief textures (textures extended with an orthogonal displacement per texel) to produce correct parallax effects. The results are correct for viewpoints that are static or moving, far away or nearby. Our approach is very simple: the relief texture to be mapped onto a polygon is first converted into an ordinary texture using a surprisingly simple forward transform. The resulting texture is then mapped onto the polygon using standard texture mapping.

First, we re-parameterize the height field such that its basis polygon coincides with the polygon on which it is to be mapped. Then we convert from the relief texture to an ordinary texture map by projecting the height field onto its own basis rectangle from the current viewpoint. The resulting texture has correct parallax for that polygon.

We implement the conversion from relief texture to ordinary texture using the 3-D warping equations of McMillan and Bishop [12]. Their equations reduce to a pair of simple 1-D warping functions for this special case. These 1-D warping functions work in texture coordinates to handle the parallax and visibility changes that result from the 3-D shape of the height-field surface. The subsequent texture-mapping operation handles the transformation from texture coordinates to screen coordinates.

In our relief textures, displacements are the distances from samples to the polygon onto which they are mapped to. Texels with zero displacement are on the surface of the polygon exactly like ordinary textures. We assume that a relief texture represents a continuous surface. In case the surface is not continuous, additional continuity information can be provided, or multiple relief textures can be used to model the surface.

Using a software implementation of our approach, we demonstrate that this technique enhances image quality by adding viewpoint motion parallax to textures, while keeping a low polygon count for the scene description. We also show that our method can be used to produce correct views of complex objects modeled as sets of relief textures.

2 RELATED WORK

Shade et al. [19] enhance the descriptive power of traditional sprites with out-of-plane displacements per pixel. They use a two-step-rendering algorithm to compute the color of pixels in the destination image from pixels in a source image. In the first step, the displacement map associated with the source image is forward mapped using a 2-D transformation to compute an intermediate displacement map $d_3(x_3, y_3)$ [19], which is then stored for later use. In the second pass, each pixel (x_2, y_2) of the desired image is transformed by a homography (planar perspective projection) to compute intermediate coordinates (x_3, y_3). Such coordinates are used to index the displacement map $d_3(x_3, y_3)$ computed in the first pass. The retrieved displacement value is then multiplied by the epipole e_{21} and added to the result of the homography, producing the coordinates (x_1, y_1) in the source image. Such coordinates are used to compute the color of the destination pixel (x_2, y_2) by filtering the color of pixels in the neighborhood of (x_1, y_1) in the source image. It is unclear from the paper how the intermediate displacement map d_3 is computed.

A variant of the same algorithm consists of, in the first step, forward mapping the displacement map associated with the source image I_1 to an intermediate image I_3 and for each pixel the differences $u_3(x_3, y_3) = x_3 – x_1$ and $v_3(x_3, y_3) = y_3 – y_1$ are computed and stored for later use. During the second pass of the algorithm, each pixel (x_2, y_2) of the desired image is transformed by a homography to compute intermediate coordinates (x_3, y_3). Such coordinates are added to $(u_3(x_3, y_3), v_3(x_3, y_3))$ to produce the coordinates (x_1, y_1) in the source image, whose neighborhood is then filtered to produce the color for (x_2, y_2).

Although the approach presented by Shade et al. is expected to produce smoother rendering than traditional forward mapping splat-based techniques, the reconstruction is done using splats and holes may still happen.

In our method, each coordinate of the destination pixel depends only on its counterpart in the original pixel (*i.e.*, u_2 does not depend on v_1, and v_2 does not depend on u_1). This enables our approach to be implemented efficiently as 1-D operations for both reconstruction and filtering. In addition, we use standard texture mapping hardware to perform the final planar perspective warp. Whereas Shade, et al. state that sprites with depth should be used as rendering primitives only when viewed from a distance, the textures produced with our approach can be used even when the viewpoint is very near to the polygon, because all holes are completely filled during the reconstruction process.

A nailboard [17] is a texture-mapped polygon augmented with a displacement value per texel specifying how much its depth deviates from the depth of the represented view of an object. The idea behind nailboards is to take advantage of frame-to-frame coherence in smoth sequences. Thus, instead of rendering all frames from scratch, more complex objects are rendered to separate buffers and re-used as sprites as long as the geometric and photometric errors remain below a certain threshold [17]. An error metric is therefore required. The displacement values associated with each texel are used to modulate the depth buffer of the final composite frame. In conjunction with partially transparent polygons, the associated displacements are used to solve visibility among other nailboards and conventional polygons.

Meyer and Neyret [13], and Schaufler [18] use stacks of 2-D textures with transparency to obtain parallax effects using conventional texture-map hardware.

Debevec, Taylor, and Malik [3] used view-dependent texture mapping to render new views of a scene, by warping and compositing multiple original images of the same scene. Debevec, Yu, and Borshukov [4] use visibility preprocessing, polygon-view maps, and projective texture mapping to produce a three-step view-dependent texture mapping that further reduces the computational cost and produces smoother blending.

3 TWO-PASS 1-D TRANSFORMS

Image operations such as texture mapping and image warping involve transformations among pairs of coordinates. Catmull and Smith [1] showed how affine and perspective transformations applied to planar surfaces and to bilinear and biquadratic patches can be decomposed into a series of 1-D transformations (shears) over rows and columns. Later, Smith [20] showed that texture mapping onto planar quadric and superquadric surfaces, and planar bicubic and biquadratic image warps are two-pass transformable. He also coined the expressions *parallel warp* and *serial warp* to refer to the original 2-D map and to the composition of 1-D transforms that accomplishes a similar result, respectively.

Assuming the row pass takes place first, a two-pass serial warp[1] is accomplished by a *horizontal shear* followed by a *vertical shear* operation applied to the image. The horizontal pass shifts the elements of a row by variable amounts. Likewise, the vertical pass moves the elements along the columns of the resulting image by possibly different amounts.

Let u_s and v_s be the coordinates of a source image, and u_t and v_t be the target coordinates in the desired image. Given two functions H and V, a parallel warp can be defined such that $(u_t, v_t) = (H(u_s,v_s), V(u_s,v_s))$. A serial equivalent of the parallel transformation can be obtained by defining a composite mapping $v \circ h$. Let the horizontal shear h preserve the v_s coordinates of the original pixels: $h(u_s,v_s) = (h_1(u_s,v_s),\ h_2(u_s,v_s)) = (u_t,\ v_s)$. Since $v(h(u_s,v_s))$ should produce the desired result, the vertical warp must preserve u_t computed in the first pass. Similarly, v can be defined as $v(u_s,v_s) = (v_1(u_s,v_s),\ v_2(u_s,v_s)) = (u_s,\ v_t)$. In this case, one needs to compute h_1^{-1}, such that $v(h(u_s,v_s)) = \ (v_1(u_t,v_s),\ v_2(h_1^{-1}(u_t,v_s), v_s)) = (v_1(u_t,v_s),\ v_2(u_s,v_s)) = (u_t, v_t)$.

[1] Some transformations may involve more than two passes [16]. In this work, we are interested in two-pass decompositions.

One difficulty associated with serial warps is to find closed-form solutions for h_1^{-1}. Sometimes, they do not exist at all [1]. In these cases and also when there are multiple such solutions, numerical techniques are preferred [20]. The computation of h_1^{-1} can be avoided if the original coordinates of pixels in the source image are stored in a lookup table. This idea first appeared in [1] and was later explored by Wolberg [22]. Later, we will show that in our approach, the coordinates of pixels in the desired image are computed independently from each other (section 4.1) and there is no need to recover u_s. Such a property makes our approach even more attractive for hardware implementation since no extra storage is required.

The decomposition of a mapping into a series of independent 1-D operations presents several advantages over the original 2-D transform [6][23]: First, the resampling problem becomes simpler. Reconstruction, area sampling and filtering can be efficiently done using only two pixels at a time. Secondly, it lends itself naturally to digital hardware implementation. Thirdly, the mapping is done in scanline order both in scanning the input and output images. This leads to efficient data access and considerable savings in I/O time. Also, the approach is amenable to stream-processing techniques such as pipelining and facilitates the design of hardware that works at real-time video rates [23].

3.1 Difficulties Associated with Serial Warps

Serial warps suffer from a problem commonly referred to as a *bottleneck*, the collapse of the intermediate image into an area much smaller than the final image [1]. Bottleneck can happen if the first pass is not a one-to-one mapping (*i.e.*, not injective). In such cases, entire rows in intermediate image can collapse into points, or the whole image can collapse into a line. When one of these problems happen, the final image is usually disrupted, because even though its final shape could still be recovered in the second pass, some color information has been lost. Two solutions proposed by [1] are to switch the orders between the horizontal and vertical passes, and to transpose the image before applying a complementary transformation. These simple solutions seem work for mappings involving planar surfaces. Given the nature of 3-D image warping, this phenomenon of area contraction followed by area expansion do not happen, although the problem of multiple samples mapping to the same pixel in the intermediate image are not infrequent. For such cases, we have proposed a general solution that consists of interspersing the horizontal and vertical passes, and is explained in section 4.3.2.

Another difficulty associated with serial warps is known as *foldover* [1] and is caused by non-injective 2-D mapping. For example, perspective projections of non-planar patches can cause multiple samples to map to the same pixel on the screen, depending on the viewpoint. Traditionally, this problem has been handled storing color and depth information in multiple layers [1] [22]. In the second pass, the depth information is used to perform the 1-D transformations in back to front order. The algorithm described in section 4.3.2 also handles an arbitrary number of foldovers without requiring extra storage or z comparisons.

4 WARPING RELIEF TEXTURES

Height-field rendering may result in a many-to-one mapping. Thus, no inverse operation can be directly applied. Our approach recasts the 3-D image warping operation as a two-phase process: a first step that handles parallax only by projecting the height field onto its own basis, followed by an inverse mapping operation applied to the resulting flat image.

Our first-stage is based on the methods of McMillan and Bishop [12]. The *source image* for their 3-D warp is the relief texture augmented with an orthographic camera model. Our *destination image* is the 2-D texture we are producing with a perspective camera model. These two cameras share a common image plane (the basis for the relief texture). Thus, the 3-D image warp converts the orthographic relief texture into a 2-D texture with correct perspective.

Note that since the second phase (texture mapping) cannot handle visibility changes, it is required that the position and orientation of the image plane used to produce the texture in the first step are the same as the corresponding position and orientation of the polygon that will be texture-mapped. In such a configuration, the texture is trivially correct, and the final texture mapping stage handles the planar perspective warp and the scaling.

Our method takes advantage of the observation that McMillan and Bishop's warping equations can be greatly simplified if the desired image plane is parallel to the original one, with no relative rotation between them. Pixel coordinates in the destination image are given by [12]:

$$u_2 = \frac{\vec{a}_1 \cdot (\vec{b}_2 \times \vec{c}_2) u_1 + \vec{b}_1 \cdot (b_2 \times c_2) v_1 + \vec{c}_1 \cdot (\vec{b}_2 \times \vec{c}_2) + (\dot{C}_1 - \dot{C}_2) \cdot (\vec{b}_2 \times \vec{c}_2) \delta(u_1, v_1)}{\vec{a}_1 \cdot (\vec{a}_2 \times \vec{b}_2) u_1 + \vec{b}_1 \cdot (\vec{a}_2 \times \vec{b}_2) v_1 + \vec{c}_1 \cdot (\vec{a}_2 \times \vec{b}_2) + (\dot{C}_1 - \dot{C}_2) \cdot (\vec{a}_2 \times \vec{b}_2) \delta(u_1, v_1)}$$

$$v_2 = \frac{\vec{a}_1 \cdot (\vec{c}_2 \times \vec{a}_2) u_1 + \vec{b}_1 \cdot (\vec{c}_2 \times \vec{a}_2) v_1 + \vec{c}_1 \cdot (\vec{c}_2 \times \vec{a}_2) + (\dot{C}_1 - \dot{C}_2) \cdot (\vec{c}_2 \times \vec{a}_2) \delta(u_1, v_1)}{\vec{a}_1 \cdot (\vec{a}_2 \times \vec{b}_2) u_1 + \vec{b}_1 \cdot (\vec{a}_2 \times \vec{b}_2) v_1 + \vec{c}_1 \cdot (\vec{a}_2 \times \vec{b}_2) + (\dot{C}_1 - \dot{C}_2) \cdot (\vec{a}_2 \times \vec{b}_2) \delta(u_1, v_1)}$$

where subscript 1 identifies source image variables; 2, the destination image. Vectors $\vec{a}$ and $\vec{b}$ are orthogonal and form a basis for the plane of the image. The lengths of these vectors are the width and height of a pixel in the Euclidean space, respectively. The generalized disparity associated with pixel (u_1,v_1) is $\delta(u_1, v_1)$. $\dot{C}$ is the center of projection (COP) of the associated projective pinhole camera. $\vec{c}$ is a vector from the COP to the origin of the image plane (Figure 3 (left)).

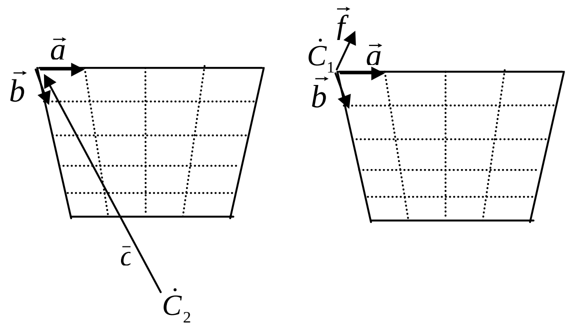

Figure 3 Perspective (left) and parallel (right) projection camera representations.

4.1 Pre-Warping Equations

A detailed derivation of the pre-warping equations is presented in [15], where the authors show that the 3-D image warping equations of McMillan and Bishop [12] can be factored into a pre-warp followed by conventional texture mapping. Part of the material presented in [15] is reproduced here for completeness.

Figure 3 (right) shows the representation we use for a height field. Vectors $\vec{a}$ and $\vec{b}$ have the same definition as in the projective pinhole camera shown in Figure 3 (left). Vector $\vec{f}$ is a unit vector orthogonal to the plane spanned by $\vec{a}$ and $\vec{b}$. The tails of all these vectors are at $\dot{C}_1$, the origin of the height field. This representation is compatible with the projective pinhole camera representation used in [12]. A point $\dot{x}$ in Euclidean space can be described as:

$$\dot{x} = \dot{C}_1 + \begin{bmatrix} a_i & b_i & f_i \\ a_j & b_j & f_j \\ a_k & b_k & f_k \end{bmatrix} \begin{bmatrix} u_1 \\ v_1 \\ displ(u_1, v_1) \end{bmatrix} = \dot{C}_1 + P_1 \vec{X}_1 \qquad (1)$$

where *displ(u₁, v₁)* is the displacement or height associated with the pixel whose coordinates are (u_1, v_1). The original formulation for perspective projection images, as defined in [12] (Figure 3 (left)) is:

$$\dot{x} = \dot{C}_2 + t_2 \begin{bmatrix} a_i & b_i & c_i \\ a_j & b_j & c_j \\ a_k & b_k & c_k \end{bmatrix} \begin{bmatrix} u_2 \\ v_2 \\ 1 \end{bmatrix} = \dot{C}_2 + t_2 P_2 \vec{X}_2 \tag{2}$$

where t_2 is a scalar value defined on a per pixel basis. Solving for $\vec{X}_2$, we get:

$$\begin{aligned} &\dot{C}_2 + t_2 P_2 \vec{X}_2 = \dot{C}_1 + P_1 \vec{X}_1 \\ &t_2 P_2 \vec{X}_2 = P_1 \vec{X}_1 + (\dot{C}_1 - \dot{C}_2) \\ &\vec{X}_2 \doteq P_2^{-1}(P_1 \vec{X}_1 + (\dot{C}_1 - \dot{C}_2)) \end{aligned} \tag{3}$$

where $\doteq$ is projective equivalence, that is, the same except for a scalar multiple. In matrix notation, we have:

$$\begin{bmatrix} u_2 \\ v_2 \\ 1 \end{bmatrix} \doteq \begin{bmatrix} \vec{a} & \vec{b} & \vec{c} \end{bmatrix}^{-1} \left(\begin{bmatrix} \vec{a} & \vec{b} & \vec{f} \end{bmatrix} \begin{bmatrix} u_1 \\ v_1 \\ displ(u_1, v_1) \end{bmatrix} + \left[(\dot{C}_1 - \dot{C}_2) \right] \right) \tag{4}$$

By making both image planes coincide (including their origins - Figure 4), $\vec{c} = (\dot{C}_1 - \dot{C}_2)$ and the equation becomes:

$$\begin{bmatrix} u_2 \\ v_2 \\ 1 \end{bmatrix} \doteq \begin{bmatrix} \vec{a} & \vec{b} & \vec{c} \end{bmatrix}^{-1} \left(\begin{bmatrix} \vec{a} & \vec{b} & \vec{f} \end{bmatrix} \begin{bmatrix} u_1 \\ v_1 \\ displ(u_1, v_1) \end{bmatrix} + \left[\vec{c} \right] \right) \tag{5}$$

Thus

$$u_2 = \frac{\vec{a} \cdot (\vec{b} \times \vec{c}) u_1 + \vec{b} \cdot (\vec{b} \times \vec{c}) v_1 + \vec{c} \cdot (\vec{b} \times \vec{c}) + \vec{f} \cdot (\vec{b} \times \vec{c}) displ(u_1, v_1)}{\vec{a} \cdot (\vec{a} \times \vec{b}) u_1 + \vec{b} \cdot (\vec{a} \times \vec{b}) v_1 + \vec{c} \cdot (\vec{a} \times \vec{b}) + \vec{f} \cdot (\vec{a} \times \vec{b}) displ(u_1, v_1)}$$

$$v_2 = \frac{\vec{a} \cdot (\vec{c} \times \vec{a}) u_1 + \vec{b} \cdot (\vec{c} \times \vec{a}) v_1 + \vec{c} \cdot (\vec{c} \times \vec{a}) + \vec{f} \cdot (\vec{c} \times \vec{a}) displ(u_1, v_1)}{\vec{a} \cdot (\vec{a} \times \vec{b}) u_1 + \vec{b} \cdot (\vec{a} \times \vec{b}) v_1 + \vec{c} \cdot (\vec{a} \times \vec{b}) + \vec{f} \cdot (\vec{a} \times \vec{b}) displ(u_1, v_1)}$$

Note that many of the scalar triple products in the u_2 and v_2 expressions are of the form $\vec{v}\cdot(\vec{v}\times\vec{w})$ or $\vec{w}\cdot(\vec{v}\times\vec{w})$ and therefore reduce to zero. Thus,

$$u_2 = \frac{\vec{a}\cdot(\vec{b}\times\vec{c})u_1 + \vec{f}\cdot(\vec{b}\times\vec{c})displ(u_1,v_1)}{\vec{c}\cdot(\vec{a}\times\vec{b}) + \vec{f}\cdot(\vec{a}\times\vec{b})displ(u_1,v_1)} \tag{6a}$$

$$v_2 = \frac{\vec{b}\cdot(\vec{c}\times\vec{a})v_1 + \vec{f}\cdot(\vec{c}\times\vec{a})displ(u_1,v_1)}{\vec{c}\cdot(\vec{a}\times\vec{b}) + \vec{f}\cdot(\vec{a}\times\vec{b})displ(u_1,v_1)} \tag{6b}$$

But $\vec{a}\cdot(\vec{b}\times\vec{c}) \neq 0$ is the determinant of the 3x3 matrix whose rows are respectively $\vec{a}$, $\vec{b}$, and $\vec{c}$. Also, $\vec{c}\cdot(\vec{a}\times\vec{b})$ is the determinant of the same matrix after two permutations of rows, and therefore has the same value. The same observation holds for $\vec{b}\cdot(\vec{c}\times\vec{a})$. Thus, dividing both numerators and denominators of equations (6*a*) and (6*b*) by $\vec{a}\cdot(\vec{b}\times\vec{c})$, we get

$$r = u_1 + k_1 displ(u_1,v_1) \tag{7}$$
$$s = v_1 + k_2 displ(u_1,v_1) \tag{8}$$
$$t = 1 + k_3 displ(u_1,v_1) \tag{9}$$
$$u_2 = \frac{r}{t} \tag{10a}$$
$$v_2 = \frac{s}{t} \tag{10b}$$

where k_1, k_2, and k_3 are constants across the entire height field and determine the amount of change in the coordinates of corresponding pixels in both images (optical flow [5]). Equations (*10a*) and (*10b*) are called *pre-warping equations*.

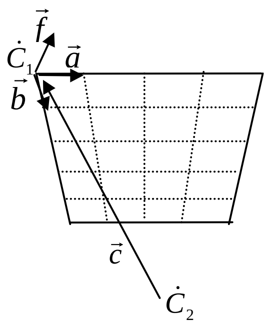

Figure 4. Parallel and perspective projection cameras that share the same image plane (origin, $\vec{a}$ and $\vec{b}$ vectors).

Notice that if the displacement displ(u_1, v_1)= 0, (u_2, v_2) = (u_1, v_1), and no pre-warping is necessary. For cases in which height fields can be constructed with many zero displacements (*e.g.*, the brick relief texture used to render Figures 13 and 14), the computational savings can be considerable.

4.2 Occlusion-Compatible Order for Height Fields

The COP of a parallel projection image is at infinity, and its epipole is the projection of the destination COP onto the plane of the parallel projection image (Figure 5). By similarity of triangles, whenever two samples fall along the same viewing ray, the one whose projection is closer to the epipole is closer to the viewer (Figure 5). Thus, an occlusion-compatible order (essentially a painter's algorithm) for height fields is obtained by warping pixels from the borders towards the epipole.

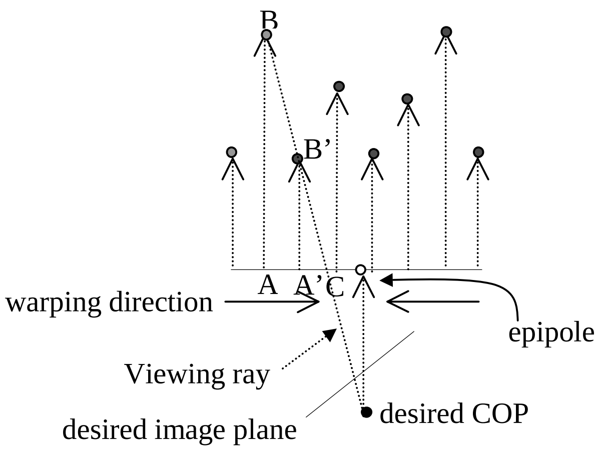

Figure 5. For a parallel projection image, the epipole is the projection of the desired COP onto the plane of the parallel projection image. An occlusion compatible order is obtained by always warping from the borders towards the epipole. Triangles ABC and A'B'C are similar.

4.3 Reconstruction

The previous sections have shown how to determine the coordinates of infinitesimal points in the destination image from points in the source image. Determining these is only the beginning of the image-warping process. The more expensive step is reconstruction and resampling onto the pixel grid of the destination image. The simplest and most common approaches to reconstruction and resampling are splatting and meshing. Splatting requires spreading each input pixel over several output pixels to assure full coverage and proper interpolation. Meshing requires rasterizing a quadrilateral for each pixel in the NxN input texture. The multiple writes of splatting and the setup overhead of rasterizing tiny quadrilaterals makes either approach very expensive. Splats usually involve additional costs associated with splat-shape calculation and antialiasing.

The special structure of our pre-warp equations allows us to implement reconstruction and resampling as a two-pass process using 1-D transforms in scan-line order [1] [20] [23] [6]. The reader should make a clear distinction between the two steps of our method: pre-warping followed by texture mapping, and the two phases used to implement the pre-warping step itself. Such phases consist of a horizontal pass and a vertical pass. Figure 6 shows this hierarchy. For clarity, we first describe a true two-pass approach that may produce occlusion errors, then a pipelined approach that avoids these errors.

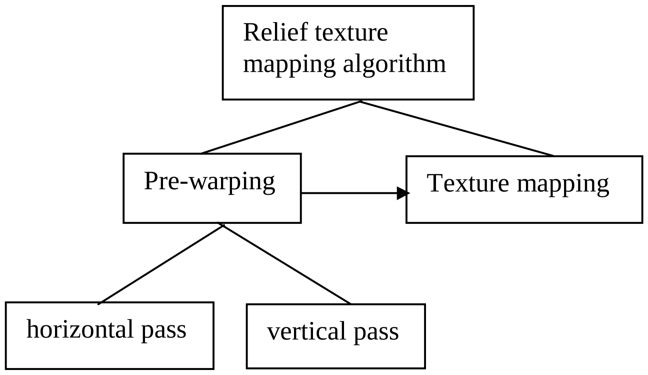

Figure 6. Structure of the two two-pass height-field texture-mapping algorithm.

4.3.1 Two-Pass Pre-Warp Implementation

Assume we do the horizontal pass completely before beginning the vertical pass; either order is acceptable and either may be advantageous [1]. Each scan line is processed independently with all output pixels going to the same scan line in the output texture. Our warping equation (Eqs. 7, 9, and 10*a*), shows that the output u_2 (column coordinate) is a function of only the input u_1 and the input displacement, not the input row number.

The elements of each row are processed in occlusion-compatible order by starting with the element furthest from the epipole and working towards the epipole. We show the case when the epipole is to the right and the warp is proceeding left to right. Figure 7 shows the pseudocode for warping one pixel. For the true two-pass warp, the “get” and “put” operations in the pseudocode are reads and writes of the pixel at the indicated index position. For each successive pixel in the input we compute the output element index based on the input element index and the displacement of the pixel. If the resulting element index is to the right of the previous coordinate, we linearly interpolate the color and displacement between the previous and current pixel values, sampling at each output pixel center. We store both the color and the displacement for the output pixel because the next pass needs both. If the resulting element index is to the left of the previous element index this is an occluded region in the output (a back facing part of the map) and no writes are required. Antialiasing can be implemented using the method described in [6].

After the horizontal warp has processed all the rows, the same algorithm is applied to the columns of the output. This pass handles the vertical shifts of pixels.

We have compared this true two-pass implementation to a pre-warp using mesh-based reconstruction, since the latter is the most common reconstruction method used in computer graphics. The results are almost indistinguishable as can be seen on the accompanying videotape. The differences are limited to areas in the destination image that are not represented in the source image. For example, interpolation across a depth discontinuity can produce different results for mesh reconstruction than for a two-pass 1-D reconstruction. This difference is only noticeable when there is significant change in the texture pattern across the interpolated region. Such artifacts appear to be inherent in serial warps when working across such discontinuities. We have not found these differences to be a significant problem in the models we have used.

```
get $I_{in}$, $C_{in}$, $D_{in}$
$I_{next}$ = Equation_10a($I_{in}$,$D_{in}$)
for ($I_{out}$ = integer($I_{prev}$+1); $I_{out} \le I_{next}$; $I_{out}$++)
    linearly interpolate $C_{out}$ between $C_{prev}$ and $C_{in}$
    linearly interpolate $D_{out}$ between $D_{prev}$ and $D_{in}$
    put $I_{out}$, $C_{out}$, $D_{out}$
$I_{prev}$=$I_{next}$; $C_{prev}$=$C_{in}$; $D_{prev}$=$D_{in}$
```

Figure 7. Pseudocode for left-to-right warp and construction of one pixel with u (or v) index=I, color = C and displacement D.

4.3.2 Pipelined Pre-Warp Implementation for Correct Visibility

The straightforward two-pass implementation may cause information to be lost under certain conditions. For example, in Figure 8, most of the plane has zero displacement, thus those pixels have the same coordinate in both the input and output textures. The pixels in the diagonal slot are below the plane and should move down and to the right because the epipole is in the lower right corner. Unfortunately, when pixels move to the right during the horizontal pass, they are occluded by pixels on that same row. The vertical pass has no information about the occluded pixel and thus cannot move it down to the final location[2]. Notice, however, that these situations are restricted to isolated pixels and cannot be considered as true bottlenecks in the sense of [1], which are characterized by a contraction followed by an expansion of the image area. Foldovers, on the other hand, are due to the three-dimensional nature of the represented surfaces and, therefore, may happen.

Interspersing the horizontal and vertical warps eliminates these visibility problems. Process the rows in occlusion-compatible order by starting with the row furthest from the epipole and working towards the epipole. As before, process the pixels on each row in occlusion-compatible order. As the horizontal warp produces each output pixel and displacement, it hands the pixel to the vertical warp process for that column. The vertical warp immediately interpolates the pixel into the appropriate column of the output texture. Each vertical warp process receives its pixels in occlusion-compatible order so correct visibility is preserved in the output. The pseudocode in Figure 7 also applies to the pipelined warp if the "get" operation in the vertical process for each column waits for the output of the corresponding "put" operation in the horizontal process. The code for the vertical process is identical to the code for the horizontal process except for the last line in Figure 7. In the vertical process, the registers I_{prev}, C_{prev}, and D_{prev} hold the values from the previous row until the current row is complete. This is because the horizontal process may map multiple pixels to the same vertical process.

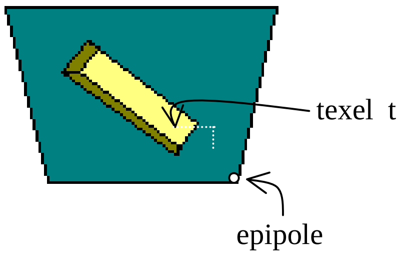

Figure 8. Two-pass 1-D problem. The green region represents zero displacements. The yellow area corresponds to a deep region. Because the epipole is at the lower right corner, texel t should move right (in the horizontal pass) and then down (in vertical pass). But during the horizontal pass, it will be occluded by a texel with zero displacement and no information will be available for the vertical pass.

[2] Notice that the situation depicted in Figure 8 produces no visual artifact in the final image. It was used only to

There are some advantages in computing the two coordinates of the pre-warped pixel in the first step of the algorithm both in the two-pass and in the pipelined pre-warping implementation. In this case, Equation (9) is evaluated only once per pixel instead of twice. Also, minor artifacts due to the non-linear nature of Equations (*10a*) and (*10b*) are avoided. For instance, when Equation (*10b*) is evaluated with linearly interpolated displacement values (computed in the first step) for filling holes caused by expansions of the relief texture in some regions, straight lines may become curved. Notices that in this situation the algorithm is synthesizing (interpolating) some texture to fill holes in regions not visible in the original relief texture. The artifacts may be noticed if the expanded regions are relatively large with respect to the size of the texture itself. Such artifacts are completely eliminated by passing linearly interpolated values obtained from the actual row coordinates of the pixels to the second step. It worth mentioning that even when linearly interpolated values are used, it is still very difficulty to notice artifacts, unless the texture contains stripes and the associate height field presents large variations in displacements in these regions. Examples shown in Figures 13, 14 and 21 were computed using interpolated displacement values.

4.4 Multiple Instantiation of Height Fields

Often a single texture is used to add detail to multiple surfaces. For example, a brick texture might be used for all the exterior walls of a building. This section explains how relief textures can be used in the same way.

Let P be a polygonal representation of a scene S. For simplicity, assume the existence of a set of planar rectangular quadrilaterals (quads, for short) $P_q \subseteq P$ that will be texture-mapped with instances of pre-warped relief texture. Transparency (alpha channel equal to zero) is used if the resulting texture is not rectangular.

Each quad in P_q has an associated relief texture, and each relief texture can be associated with an arbitrary number of quads simultaneously. Also, each vertex of a quad has an associated pair of texture coordinates (s,t). As the scene model is loaded, for each quad $q_i \in P_q$, we compute $\dot{C}_{1i} = v_{1i}$, $\vec{a}_i = (v_{2i} - v_{1i})/(rs_i * c(i))$, $\vec{b}_i = (v_{0i} - v_{i1})/(rt_i * r(i))$, and $\vec{f}_i = normalized(\vec{a}_i \times \vec{b}_i)$, where v_{ji} is the jth

illustrate the nature of a problem that may happen in some configurations.

vertex of quad q$_i$, $0 < rs_i \leq 1$ is the range of the texture parameter *s* ($rs_i = s(v_{2i}) - s(v_{1i})$), $0 < rt_i \leq 1$ is the range of the texture parameter *t* ($rt_i = t(v_{0i}) - t(v_{1i})$), and *c(i)* and *r(i)* are, respectively, the number of columns and rows of the associated relief texture. *rs$_i$* and *rt$_i$* are used as scaling factors for the cases in which the whole parameter space for *s* and *t* are not used. Figure 13 (right) illustrates a situation for which the same relief texture is applied to two quads of different dimensions (one of the quads is half as wide as the other one). The scaling by *rs* = 0.5 produces bricks of the same size in the smaller wall, for which the values of parameter *s* only varies from 0 to 0.5.

At rendering time, the values of $\dot{C}_1$, $\vec{a}$, $\vec{b}$, and $\vec{f}$ of the associated relief texture are replaced with those from quad q$_i$. This has the effect of positioning the relief texture at the same place and orientation as q$_i$ with respect to the desired viewpoint. The result of the pre-warping operation is an image that when texture-mapped on q$_i$ produces the desired result. The texture coordinates associated with the vertices of q$_i$ are used to select the portion of the image to be applied.

Our current implementation uses OpenGL ModelView matrix [24] to transform $\dot{C}_{1i}$, $\vec{a}_i$, $\vec{b}_i$, and $\vec{f}_i$, which are then used to replace the corresponding values of the associated height field parameter vectors. We copy the transformation matrix. The translational component of the transformation is saved and then zeroed. The resulting rotation matrix is used to multiply the 4x4 matrix $[\dot{C}_{1i} \quad \vec{a}_i \quad \vec{b}_i \quad \vec{f}_i]$ (with the fourth coordinate of all vectors set to 1). The translational component is then added to the transformed value of $\dot{C}_{1i}$.

The technique described is applied to all quads associated with relief textures. Notice that by performing the pre-warping operation with vectors associated to the quads, the relief texture is not bound to any particular position or orientation in the scene and can be used with an arbitrary number of quads, producing the desired parallax effect in all cases.

4.5 Representing More Complex Shapes

Several researchers have used image-based approaches to represent complex object shapes [10] [7] [8] [18] [14]. Relief textures may also be used to render complex shapes.

Consider an object represented by six relief textures (the faces of a bounding box) as shown in Figure 9. These relief textures can be used to produce new views of the object when pre-warped to their

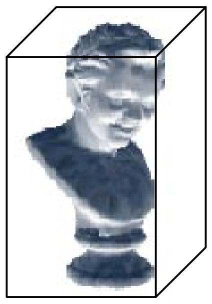
Figure 9. Object represented as six relief textures.

own base polygons. Notice that, although we pre-warp to multiple polygons, since a single viewpoint is used, all images are consistent with each other. However, warping each relief texture to its own basis polygon is not enough to produce the desired result. Figure 10 illustrates the problem using a top view representation of Figure 9. Areas sampled only by one face can project onto others, depending on the viewpoint. If such areas are not appropriately mapped to the correct faces, holes will appear.

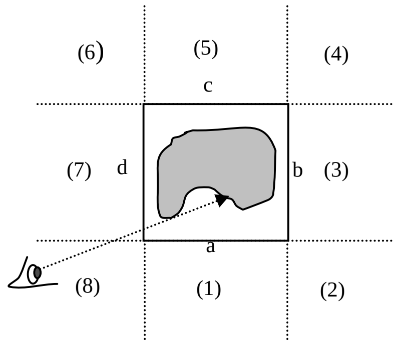

Figure 10. Top view of the object shown in Figure 9. Samples from one face can project onto another. Letters identify the faces, and numbers identify regions used to define the faces that should be pre-warped from each region.

Our solution to this problem is to fill the holes by pre-warping neighbor faces to the destination face. The perpendicular relationship between faces allows the use of the same pre-warping functions described in section 4.1. Figure 10 shows a division of the object space into numbered regions. The concept will be explained in 2-D. Its generalization to 3-D is straightforward. If the viewer is in an odd region, we classify the three closest faces as *front*, *left*, and *right* with respect to the viewpoint. Thus, for instance, if the viewer is in region (1), face *a* is *front*, face *d* is *left*, and face *b* is *right*. Then, faces *left* and *right* are pre-warped to the image plane of *front*. Then *front* is pre-warped to its own image plane. If, however, the viewer is an even region, the two closest faces are classified as *left* and *right*. For instance, if the viewer is in region (8), face *d* is *left* and face *a* is *right*. *left* is pre-warped to the image plane of *right*, then *right* is pre-warped to its own image plane. Similarly, *right* is pre-warped to the image plane of *left*, and then *left* is pre-warped to its own image plane. Figure 11 presents the pseudocode for the hole-filling algorithm. Notice that this algorithm automatically implements a back-face-culling strategy, since it explicitly defines a set of at most three (in the full 3-D version of the algorithm) polygons that need to be displayed.

The right-angle relationship between neighbor faces can be exploited to pre-warp a face to its neighbor image plane as if it were the neighbor face itself. When the viewer is in an odd region, the displacement values associated with *left* and *right* (Figure 12) are converted to column indices for *front*, while their column indices can be used as displacement for *front*. Thus, *left* and *right* can be pre-warped to *front* as if they were *front* itself. The even region is similar.

```
If viewer is in an odd region then
    pre-warp left to front's image plane;
    pre-warp right to front's image plane;
    pre-warp front to its own image plane;
else
    pre-warp left to right's image plane;
    pre-warp right to its own image plane;
    pre-warp right to left's image plane;
    pre-warp left to its own image plane;
endif
```

Figure 11. Pseudocode for filling holes of object.

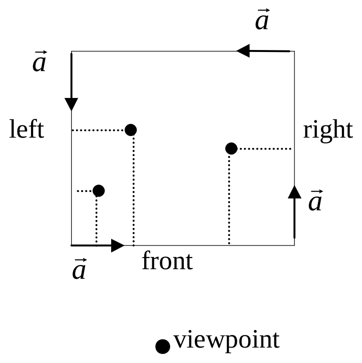

Figure 12. Height values from *left* and *right* are mapped to columns for *front*. Columns from *left* and *right* are mapped to height for *front*.

5 RESULTS

Using the algorithms described in this paper, we implemented in C++ a software test bed for warping relief textures. We tested on a 400MHz Pentium II processor with 128 MB of memory, and an Intergraph Intense 3-D Pro 3410-T graphics card. For this application, it is not clear whether the way the texture memory is managed by the graphics card is of any help. Software implementation of our two-step warping process will work best with the new generation of graphics accelerators that store texture in main memory. For example, the SGI O2 and the new SGI Visual Workstations for Windows NT integrate the memory and graphics system to allow very high bandwidth access to memory from both CPU and graphics accelerator. Pre-warping the textures on one processor of a multi-processor system, then mapping the texture directly from memory onto polygons should allow very efficient implementation.

Figure 13 (left) shows a 256x256-pixel original texture used to produce a relief texture for a brick wall. The displacement values of the brick pixels are zero while the mortar pixels have unit displacement. Figure 13 (right) shows the result of applying the same relief texture to two polygons of different sizes.

Figures 14 (left) shows an oblique view of a wall modeled as a single polygon and texture-mapped with the corresponding pre-warped relief texture shown to the right. Notice the illusion of three-dimensionality especially at the edges of the wall. The correct parallax effect allows the mortar to be visible only at some points of the image. This single wall is rendered at about 10.67 frames per second using a software implementation with a two-pass 1-D reconstruction. Conventional texture mapping with the use of the graphics accelerator achieves an average of 44.5 frames per second. Our current implementation emphasizes code clarity as opposed to speed. Thus, we expect to significantly increase the frame rates as a result of code optimization.

Figure 15 shows the images associated with four height-field textures of a statue originally modeled with 35,280 polygons. Despite of its complex shape, our technique produces very realistic renderings of the object at interactive rates. Figure 16 (left) shows a reconstructed view of the statue obtained by texture mapping two quads (*left* and *right* – section 4.5) whose boundaries are shown in red in Figure 16 (right). The corresponding pre-warped textures are shown in Figure 17. Such textures were obtained by pre-warping the relief textures associated with the two top images shown in Figure 15 to both quads whose boundaries are shown in Figure 16 (right). Figure 17 provides a clear illustration of the factorization of the planar perspective warp from the final images. Figure 18 shows a close up of the same statue from a different viewpoint.

Figures 1, 2, 19, and 20 compare the use of conventional texture mapping and our technique. Figure 1 shows the results of conventionally texture mapping three quads whose borders are shown in red. The result looks unrealistic. Figure 2 shows the same scene rendered using three relief textures and five rectangles. The extra quadrilaterals were used to accommodate the samples that are mapped to beyond the borders of the three original ones. We do this by pre-warping the relief textures to the extras planes taking advantage of their perpendicularity. Figure 19 shows the same scene without the borders of the quadrilaterals. Notice the perception of three-dimensionality. Figure 20 shows the same scene from a different viewpoint. Figure 21 shows the image of a building rendered using one relief texture and a single polygon, and reconstructed with the two-pass 1-D reconstruction algorithm.

6 LIMITATIONS

Several of the tradeoffs and limitations involving the use of texture-mapped flat polygons to represent complex geometric shapes also apply to our technique. The viewer must not pass through the textured polygon. Putting the planes as close to the represented objects as possible minimizes this problem. Alternatively, one can switch to geometry when such a situation is reached.

A limitation inherent to pre-warping a height field to its own basis polygon is the fact that some samples can be mapped to beyond the limits of the image plane. Our solution to this problem is to use an extra plane perpendicular to the original one to accommodate the “overflows”. Figures 2 illustrates its result.

Ideally, all samples at the borders of the relief texture should have zero displacement. This way, height fields can be safely tiled as regular textures with image continuity assured.

7 FUTURE WORK

One important area for investigation is the design of efficient hardware implementations for relief textures using our pre-warping functions. Adding this pre-warping capability to the texture memory of a graphics accelerator may allow relief textures to become as commonly used as conventional textures. Other avenues that we are currently exploring involve the use of a mip-map representation [21] for pre-filtering, use of relief textures for geometry simplification, use of normal maps [1] [2] for view-dependent lighting of relief textures, depth modulation of pre-warped pixels to match the perceived depth value, and scene segmentation into multiple polygons and relief textures. We have also derived similar pre-warping equations for conventional perspective images with depth. One can simplify very complex models by enclosing the viewpoint in a box and projecting pre-warped images representing all of the outside geometry onto the faces of the box.

Acknowledgements

We would like to thank Mary Whitton, Nick England, John Eyles, Anselmo Lastra and the UNC IBR group for their assistence and insights. Special thanks to Fred Brooks for his detailed critique of an earlier draft.

This work was sponsored by CNPq/Brazil – Process # 200054/95, DARPA under order # E278 and NFS under grant # MIP-961.

References

[1] Catmull, E., Smith, A.. *3D Transformations of Images in Scanline Order*. Proc. SIGGRAPH 80 (Seattle, Washington, July 14-18, 1980). In Computer Graphics Proceedings. Annual Conference Series, 1980, ACM SIGGRAPH, pp. 279-285.

[2] Cohen, J., Olano, M., Manocha, D. *Appearance-Preserving Simplification*. Proc. SIGGRAPH 98 (Orlando, FL, July 19-24, 1998). In Computer Graphics Proceedings. Annual Conference Series, 1998, ACM SIGGRAPH, pp. 115-122.

[3] Debevec, P., Taylor, C., Malik, J.. *Modeling and Rendering Architecture from Photographs: A hybrid geometry- and image-based approach*. Proc. SIGGRAPH 96 (New Orleans, LA, August 4-9, 1996). In Computer Graphics Proceedings. Annual Conference Series, 1996, ACM SIGGRAPH, pp. 11-20.

[4] Debevec, P., Yu, Y., Borshukov, G. *Efficient View-Dependent Image-Based Rendering with Projective Texture-Mapping*. Proceedings of the 9th Eurographics Workshop on Rendering. Vienna, Austria, June 1998, pp. 105-116.

[5] Faugeras, Olivier. Three-Dimensional Computer Vision: A Geometric Viewpoint. The MIT Press, 1993.

[6] Fant, Karl. *A Nonaliasing, Real-Time Spatial Tr*ansform Technique. IEEE Computer Graphics and Application, Vol. 6, No 1, January 1986, pp. 71-80.

[7] Gortler, S., et al.. *The Lumigraph*. Proc. SIGGRAPH 96 (New Orleans, LA, August 4-9, 1996). In Computer Graphics Proceedings. Annual Conference Series, 1996, ACM SIGGRAPH, pp. 43-54.

[8] Grossman, J., Dally, W. *Point Sample Rendering*. Proceedings of the 9th Eurographics Workshop on Rendering. Vienna, Austria, June 1998, pp. 181-192.

[9] Heckbert, P. *Fundamentals of Texture Mapping*. Master's thesis. Technical Report No. UCB/CSD 89/516. Computer Science Division, University of California, Berkeley.

[10] Levoy, M., Hanrahan, P. *Light Field Rendering*. Proc. SIGGRAPH 96 (New Orleans, LA, August 4-9, 1996). In Computer Graphics Proceedings. Annual Conference Series, 1996, ACM SIGGRAPH, pp. 31-42.

[11] McMillan, L. *Computing Visibility Without Depth*. UNC Computer Science Technical Report TR95-047, University of North Carolina, October 1995. http://www.cs.unc.edu/~ibr/pubs/mcmillan-visibility/visibility.ps.gz

[12] McMillan, L. *An Image-Based Approach to Three-Dimensional Computer Graphics*. Ph.D. Dissertation. UNC Computer Science Technical Report TR97-013, University of North Carolina, April 1997. http://www.cs.unc.edu/~ibr/pubs/mcmillan-diss/mcmillan-diss.pdf

[13] Meyer, A., Neyret, F. *Interactive Volumetric Textures*. Proceedings of the 9th Eurographics Workshop on Rendering. Vienna, Austria, June 1998, pp. 157-168.

[14] Oliveira, M., Bishop, G. *Image-Based Objects*. To appear in Proceedings of 1999 ACM Symposium on Interactive 3D Graphics. Atlanta, Ga, April 26-28, 1999. http://www.cs.unc.edu/~ibr/pubs/oliveira-i3d99/ibo-paper.pdf

[15] Oliveira, M., Bishop, G. *Factoring 3-D Image Warping Equations into a Pre-Warp Followed by Conventional Texture Mapping*. UNC Computer Science Technical Report TR99-002, University of North Carolina, January 1999. http://www.cs.unc.edu/~oliveira/pubs/TR99-002.pdf

[16] Paeth, A. *A Fast Algorithm for General Raster Rotations*. Proceedings of Graphics Interface'86, May, 1986, pp. 77-81.

[17] Schaufler, G. *Nailboards: A Rendering Primitive for Image Caching in Dynamic Scenes*. Proceedings of the 8th Eurographics Workshop on Rendering. St. Ettiene, France, June 16-18, 1997, pp. 151-162.

[18] Schaufler, G. *Per-Object Image Warping with Layered Impostors*. Proceedings of the 9th Eurographics Workshop on Rendering. Vienna, Austria, June 1998, pp. 145-156.

[19] Shade, J., et al. *Layered Depth Images*. Proc. SIGGRAPH 98 (Orlando, FL, July 19-24, 1998). In Computer Graphics Proceedings. Annual Conference Series, 1998, ACM SIGGRAPH, pp. 231-242.

[20] Smith, Alvy Ray. *Planar 2-Pass Texture Mapping and Warping*. Proc. SIGGRAPH 87 (Anaheim, CA, July 27-31, 1987). In Computer Graphics Proceedings. Annual Conference Series, 1987, ACM SIGGRAPH, pp. 263-272.

[21] Williams, L. *Pyramidal Parametrics*. Proc. SIGGRAPH 83 (Detroit, MI, July 25-29, 1983). In Computer Graphics Proceedings. Annual Conference Series, 1983, ACM SIGGRAPH, pp. 1-11.

[22] Wolberg, George. *Separable Image Warping with Spatial Lookup Tables*. Proc. SIGGRAPH 89 (Boston, MA, July 31-4 August, 1989). In Computer Graphics Proceedings. Annual Conference Series, 1989, ACM SIGGRAPH, pp. 369-378.

[23] Wolberg, George. Digital Image Warping. IEEE Computer Society Press, 1990.

[24] Woo, M., et al. *OpenGL Programming Guide*. 2nd edition. Addison Wesley, 1997.

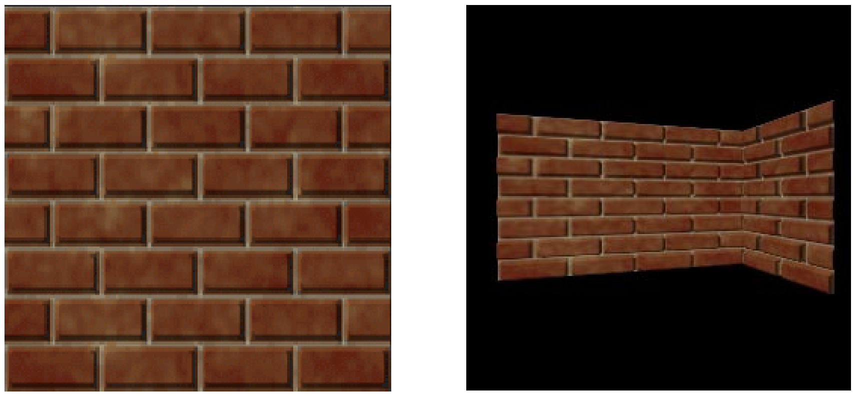

Figure 13. The original texture used to produce a height-field-brick wall (left). The result of mapping it on two polygons of different sizes (right) using our two-pass 1-D reconstruction.

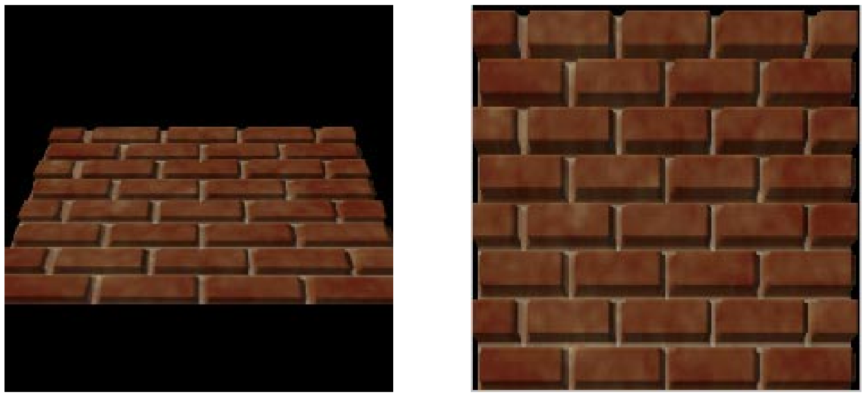

Figure 14. Oblique view of the same wall (left) with the corresponding pre-warped texture (right). Reconstructed with a two-pass 1-D reconstruction algorithm. Note visibility of mortar changes from bottom to top.

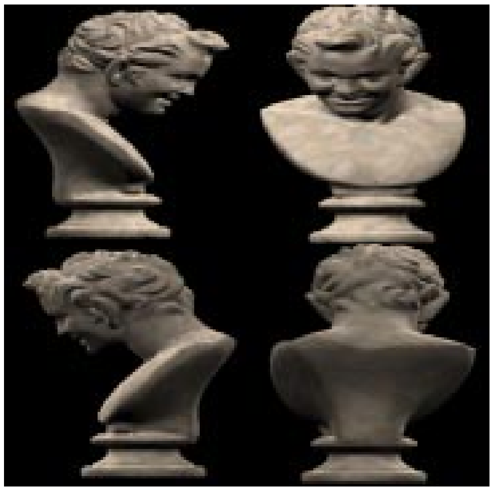

Figure 15. Images associated with four of the six height fields used to reconstruct views of the original statue.

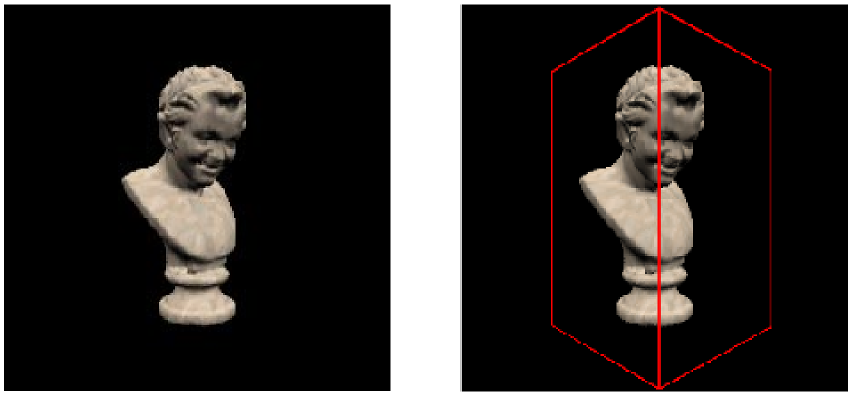

Figure 16. Reconstructed view of the statue obtained by texture mapping two quads (left). The red lines show the boundaries of the polygons (right).

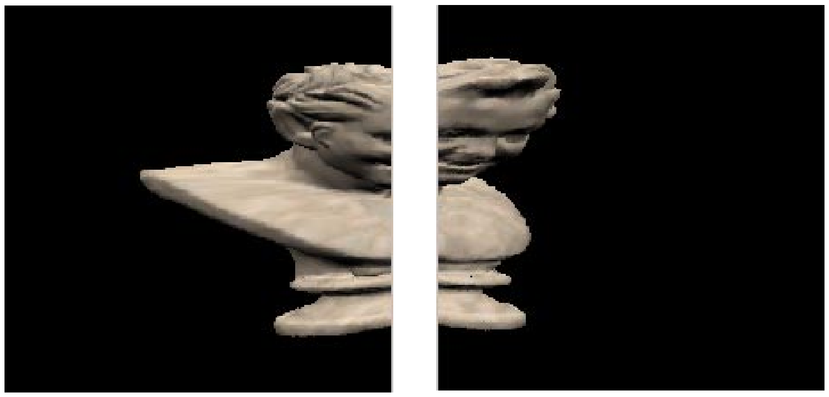

Figure 17. Pre-warped textures used to produce the image shown in figure 14.

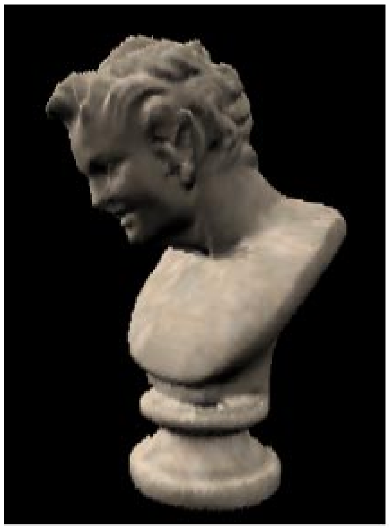

Figure 18. A close-up reconstruction of the object represented in figure 13.

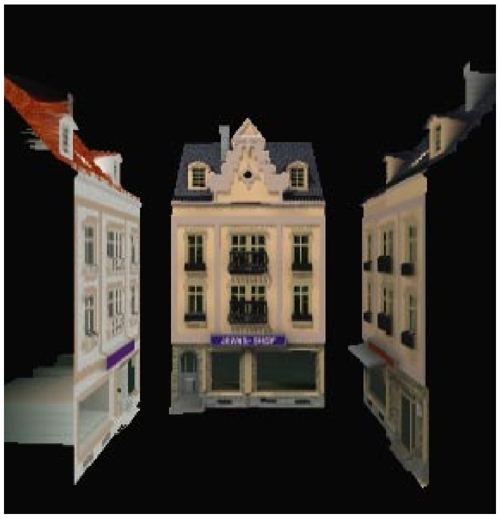
Figure 19. Same as figure 2, with the polygon boundaries hidden.

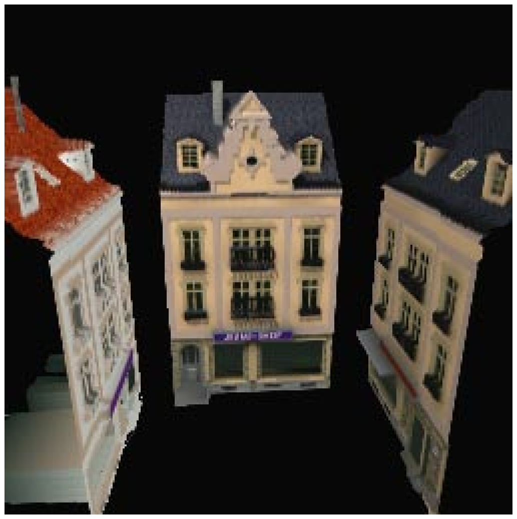
Figure 20. A different view of the same scene shown in Figure 19.

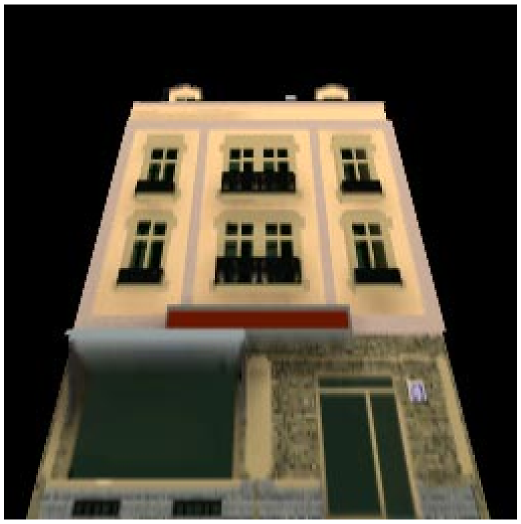
Figure 21. Example of pre-warped height-field texture mapped to a single polygon. Reconstruction performed using the two-pass 1-D algorithm. Notice that the roof becomes occluded from this viewpoint.