UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Technical Report RP-351 /
Relatório de Pesquisa RP-351

January 26, 2005

An Efficient Representation for Surface Details

by

Manuel M. Oliveira and Fabio Policarpo

Instituto de Informática
UFRGS

Paralelo Computação

UFRGS-II-PPGCC
Caixa Postal 15064 - CEP 91501-970
Porto Alegre RS BRASIL
Telefone: +55 (51)3316-6155
Fax: +55 (51) 3336-5576
Email: pgcc@inf.ufrgs

An Efficient Representation for Surface Details

Manuel M. Oliveira*
UFRGS

Fabio Policarpo†
Paralelo Computação

Abstract

This paper describes an efficient representation for real-time mapping and rendering of surface details onto arbitrary polygonal models. The surface details are informed as depth maps, leading to a technique with very low memory requirements and not involving any changes of the model's original geometry (*i.e.*, no vertices are created or displaced). The algorithm is performed in image space and can be efficiently implemented on current GPUs, allowing extreme close-ups of both the surfaces and their silhouettes. The mapped details exhibit correct self-occlusions, shadows and interpenetrations. In the proposed approach, each vertex of the polygonal model is enhanced with two coefficients representing a quadric surface that locally approximates the object's geometry at the vertex. Such coefficients are computed during a pre-processing stage using a least-squares fitting algorithm and are interpolated during rasterization. Thus, each fragment contributes a quadric surface for a piecewise-quadric object-representation that is used to produce correct renderings of geometrically-detailed surfaces and silhouettes. The proposed technique contributes an effective solution for using graphics hardware for image-based rendering.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism

Keywords: surface details, silhouette rendering, image-based rendering, relief mapping, quadric surfaces, real-time rendering.

Figure 1: Relief Room. The columns, the stone object and the walls were rendered using the technique described in the paper.

1 Introduction

The ability to add details to object surfaces can significantly improve the realism of rendered images, and efforts along those lines have a long history in computer graphics. The pioneer works of Blinn on bump mapping [Blinn 1978], and Cook, on displacement mapping [Cook 1984], have inspired the appearance of several techniques during the past two decades [Max 1988; Heidrich et al. 2000; Oliveira et al. 2000]. While bump mapping can produce very impressive results at relatively low computational cost, it relies on shading effects only and, therefore, cannot handle self-occlusions, shadows, interpenetrations, and simulated details at the object's silhouette. All these features are naturally supported by displacement mapping, which actually changes the underlying object geometry. However, since it usually involves rendering a large number of micro-polygons, it is not appropriate for real-time applications.

Several techniques have been proposed to accelerate the rendering of displacement maps and to avoid the explicit rendering of micro-polygons. These techniques are based on ray tracing [Patterson et al. 1991; Pharr and Hanrahan 1996; Heidrich and Seidel 1998; Smits et al. 2000], inverse 3D image warping [Schaufler and Priglinger 1999], 3D texture mapping [Meyer and Neyret 1998; Kautz and Seidel 2001] and precomputed visibility information [Wang et al. 2003; Wang et al. 2004]. The demonstrated ray-tracing and inverse-warping-based approaches are computationally intensive and not suitable for real-time applications. The 3D-texture approaches render displacement maps as stacks of 2D texture-mapped polygons and may introduce objectionable artifacts in some situations. The precomputed visibility approaches are very fast but require considerable amount of memory in order to store a sampled version of a five-dimensional function.

Recently, a new technique called *relief mapping* has been proposed for rendering surface details onto arbitrary polygonal models in real time [Policarpo et al. 2005]. This technique has very low memory requirements, correctly handles self-occlusions, shadows and interpenetrations. Moreover, it supports extreme close-up views of the object surface. However, like in bump mapping [Blinn 1978], the object's silhouette remains unchanged, reducing the amount of realism.

This paper introduces a completely new formulation for relief mapping that preserves all the desirable features of the original technique, while producing correct renderings of objects' silhouettes. In this new approach, the object's surface is locally approximated by a piecewise-quadric representation at fragment level. These quadrics are used as reference surfaces for mapping the relief data. Figure 1 shows a scene where the columns, the stone object and the walls

*e-mail:oliveira@inf.ufrgs.br
†e-mail:fabio@paralelo.com.br

were rendered using the technique described in the paper. Notice the details on the object's silhouette. Our approach presents several desirable features when compared to other recently proposed techniques to represent surface details [Wang et al. 2003; Wang et al. 2004]: it uses a very compact representation, is easy to implement, supports arbitrary close-up views without introducing noticeable texture distortions, and supports mip mapping and anisotropic texture filtering.

The main contributions of this paper include:

- An improved technique for mapping and rendering surface details onto polygonal models in real time (Section 3). The approach preserves the benefits of displacement mapping (*i.e.*, correct self-occlusions, silhouettes, interpenetrations and shadows), while avoiding the cost of creating and rendering extra geometry. The technique works in image-space and, although requiring a very limited amount of memory, it supports extreme close-up views of the mapped surface details;

- New texture-space algorithms for computing the intersection of a viewing ray with a height field mapped onto a curved surface (Section 3);

- An effective solution for using graphics hardware for image-based rendering.

2 Related Work

Looking for ways to accelerate the rendering of surface details, several researchers have devised techniques that exploit the programmability of current GPUs. Hirche et al. [Hirche et al. 2004] extrude a tetrahedral mesh from the object's polygonal surface and use a ray casting strategy to intersect the displaced surfaces inside the tetrahedra. Comparing to the original model, this approach significantly increases the number of primitives that need to be transformed and rendered. The authors report achieving some interactive, but not real-time frame rates with the implementation of their technique. Moule and McCool [Moule and McCool 2002] and Doggett and Hirche [Doggett and Hirche 2000] proposed approaches for rendering displacement maps based on adaptive tessellation of the original mesh.

Wang et al. [Wang et al. 2003] store pre-computed distances from each displaced point along many sampling viewing directions, resulting in a five-dimensional function that can be queried during rendering time. Due to their large sizes, these datasets need to be compressed before they can be stored in the graphics card memory. The approach is suitable for real-time rendering using a GPU and can produce nice results. However, this technique introduces significant texture distortions and can only be applied to closed surfaces. Due to the pre-computed resolution, they should not be used for close-up renderings. The large sizes of these representations tend to restrict the number of these datasets used for the rendering of a given object.

In order to reduce texture distortions and handle surfaces with boundaries, Wang et al. [Wang et al. 2004] introduced another five-dimensional representation capable of rendering non-height-field structures. Since these representations also result in large databases, they too are more appropriate for tiling and renderings from a certain distance.

2.1 Relief Mapping

The technique presented in this paper builds upon some previous work on a technique called relief mapping [Policarpo et al. 2005] and this section provides a quick review of its main concepts. Relief mapping exploits the programmability of modern GPUs to implement an inverse (*i.e.,* pixel-driven) and more general solution to relief texture mapping [Oliveira et al. 2000].

All the necessary information for adding surface details to polygonal surfaces is encoded in regular RGBα textures. Since it uses per-pixel shading, there is no need to store pre-shaded diffuse textures as in [Oliveira et al. 2000]. Instead, the RGB channels of a texture are used to encode a normal map, while its alpha channel stores quantized depth information. The resulting representation can be used with any color texture. Figure 2 shows an example of a relief texture mapped onto the teapot shown in Figure 4 (right).

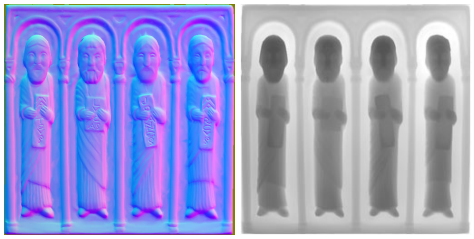

Figure 2: Relief texture represented as a 32-bit-per-texel RGBα texture. The normal data (left) are encoded in the RGB channels, while the depth (right) is stored in the alpha channel.

A relief texture is mapped to a polygonal model using texture coordinates in the conventional way. Thus, the same mapping can be used both for relief and color textures. The depth data is normalized to the $[0,1]$ range, and the implied height-field surface can be defined as the function $h : [0,1]x[0,1] \rightarrow [0,1]$. Thus, let f be a fragment with texture coordinates (s,t). A ray-intersection procedure is performed against the depth map in texture space (Figure 3) and can be described as follows:

- First, the viewing direction is obtained as the vector from the viewpoint to f's position in 3D (on the polygonal surface). The viewing direction is then transformed to f's tangent space and referred to as the *viewing ray*. The viewing ray enters the height field's bounding box *BB* at f's texture coordinates (s,t) (Figure 3 (left));

- Let (u,v) be the coordinates where the viewing ray leaves *BB*. Such coordinates are obtained from (s,t) and from the ray direction. The actual search for the intersection is performed in 2D (Figure 3 (right)). Starting at coordinates (s,t), the texture is sampled along the line towards (u,v) until one finds a depth value smaller than the current depth along the viewing ray, or we reach (u,v);

- The coordinates of the intersection point are refined using a binary search, and then used to sample the normal map and the color texture.

Since shadow computation is just a visibility problem [Williams 1978], they can be computed in a straightforward way. Given the position of the first intersection between the viewing ray and the height-field surface, a shadow ray can be defined (Figure 4 (left)). Thus, the question of whether a fragment should be lit or in shade is reduced to checking whether the intersections of both rays with the height-field surface have the same texture coordinates. Figure 4

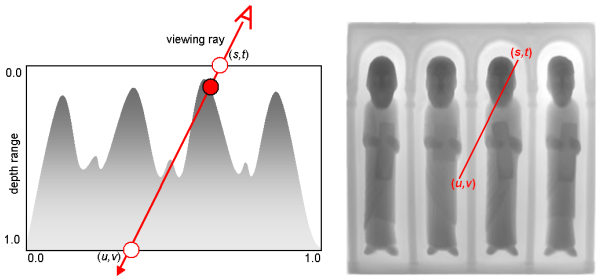

Figure 3: Ray intersection with a height-field surface (left). The search is actually performed in 2D, starting at texture coordinates (s,t) and proceeding until one reaches a depth value smaller than the current depth along the viewing ray, or until (u,v) is reached (right).

(right) shows a teapot rendered using relief mapping and exhibiting per-pixel lighting, shadows and self-occlusions.

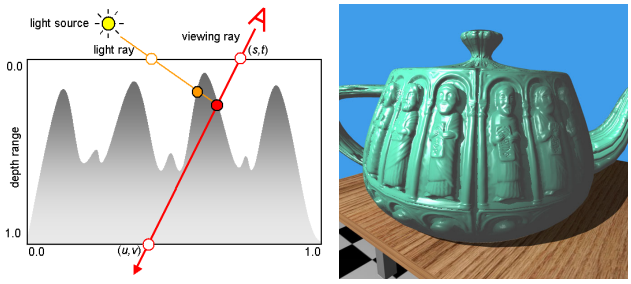

Figure 4: Shadow computation. One needs to decide if the light ray intersects the height-field surface before the point where the viewing ray first hits the surface (left). Rendering of a teapot exhibiting self-shadows and self-occlusions (right).

Despite the quality of the results obtained when rendering the interior of object surfaces, silhouettes are rendered as regular polygonal silhouettes, with no details added to them (Figure 9 (left)). Figure 5 illustrates the two possible paths for a viewing ray entering BB. In this example, ray A hits the height-field surface, while ray B misses it. Since the ray-intersection procedure has no information about whether ray B corresponds to a silhouette fragment (and should be discarded) or not, it always returns the coordinates of the intersection of B with a tiled version of the texture. Thus, all fragments resulting from the scan conversion of the object will lend to an intersection, causing the object's silhouette to match the polygonal one.

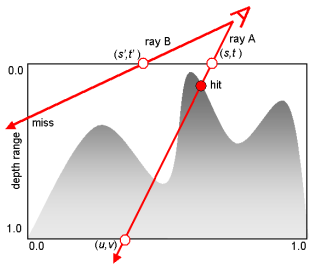

Figure 5: Possible paths for a viewing ray. Ray A intersects the height field surface, while ray B misses it.

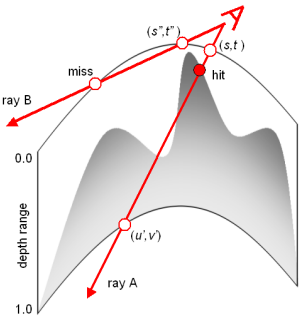

Figure 6: The height field surface is deformed to locally fit the object's surface. In this case, any ray missing the surface can be safely discarded.

3 Relief Mapping with Correct Silhouettes

One way to eliminate the ambiguity regarding to whether ray B belongs to a silhouette or not is to locally deform the the height-field surface forcing it to fit the object's geometry, as shown in Figure 6. In this case, any ray missing the surface belongs to the object's silhouette and can be safely discarded. It turns out that the abstraction depicted in Figure 6 can be directly implemented in texture space using a GPU. Recall that the height-field surface is providing a measure of how deep the relief surface is with respect to the object's surface. Thus, let (s,t) be the texture coordinates of a fragment f, the entry point of the viewing ray into the deformed bounding box (Figure 6). By having a geometric representation of the local surface defined in f's tangent space, finding the intersection of the viewing ray with the deformed height-field surface is equivalent to keep track of how deep the ray is inside the deformed bounding box. In other words, the intersection point can be defined as the point along the ray inside the deformed bounding box where the ray's depth first matches the depth of the height-field surface.

In order to have a local representation of the surface to be used as the deformed bounding box, we fit a quadric surface at each vertex of the polygonal model during a pre-processing stage. Let T be the set of triangles sharing a vertex v_k with coordinates (x_k,y_k,z_k) and let $V=\{v_1,v_2,..,v_n\}$ be the set of vertices in T. The coordinates of all vertices in V are expressed in the tangent space of v_k. Thus, given $V'=\{v'_1,v'_2,..,v'_n\}$, where $v'_i=(x'_i,y'_i,z'_i)=(x_i-x_k,y_i-y_k,z_i-z_k)$, we compute the quadric coefficients for v_k using the 3D coordinates of all vertices in V'_i. For a detailed description of methods for recovering quadrics from triangles meshes, we refer the reader to [Sylvain 2002].

In order to reduce the amount of data that needs to be stored at each vertex, we fit the following quadric to the vertices:

$$z=ax^2+by^2 \tag{1}$$

The a and b coefficients are obtained solving the system $\mathbf{Ax}=\mathbf{b}$ shown in Equation 2:

$$\begin{pmatrix} x'^2_1 & y'^2_1 \\ x'^2_2 & y'^2_2 \\ .. & .. \\ x'^2_n & y'^2_n \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} z'_1 \\ z'_2 \\ .. \\ z'_n \end{pmatrix} \tag{2}$$

These per-vertex coefficients are then interpolated during rasterization and used for computing the distance between the viewing ray and the quadric surface on a per-fragment basis. According to our

experience, the least-squares solution (*i.e.*, $\mathbf{x} = (\mathbf{A^T A})^{-1}\mathbf{A^T b}$) is sufficiently stable for this application, requiring the inversion of a matrix that is only 2 by 2. Despite the small number of coefficients, Equation 1 can represent a large family of shapes, some of which are depicted in Figure 7.

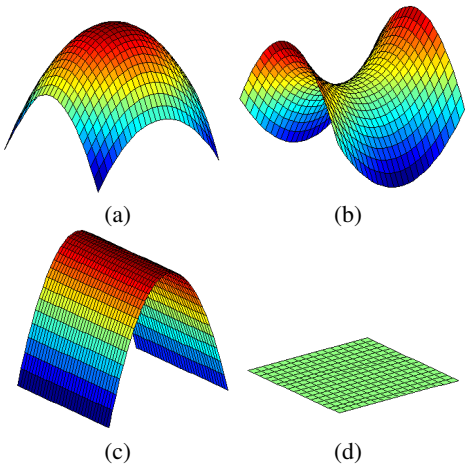

Figure 7: Examples of quadrics used to locally approximate the object's surface at each vertex. (a) Paraboloid. (b) Hyperbolic paraboloid. (c) Parabolic cylinder. (d) Plane.

3.1 Computing the Ray-Quadric Distance

In order to compute the distance to the quadric as the viewing ray progresses, consider the situation depicted in Figure 8, which shows the cross sections of two quadric surfaces. In both cases, the viewer is outside looking at the surfaces, $\hat{V}$ is the unit vector along the viewing direction $\vec{V}$, $\vec{N}$ the normal to the quadric Q at the origin of fragment f's tangent space (the point where the viewing vector first intersects Q), and P is a point along $\vec{V}$ for which we want to compute the distance to Q. P can be defined as

$$P = \hat{V}t \tag{3}$$

where t is a parameter. First, consider the case shown in Figure 8 (left). Let $\hat{U}$ be a unit vector perpendicular to $\hat{V}$ and coplanar to $\hat{V}$ and $\vec{N}$. Let R be a point on Q obtained from P by moving s units along the $\hat{U}$ direction:

$$R = (R_x, R_y, R_z) = P + \hat{U}s. \tag{4}$$

The distance from P to the quadric Q is simply s, which can be obtained by substituting the coordinates of R into the quadric equation (Equation 1):

$$a(R_x)^2 + b(R_y)^2 - Rz = 0$$

$$a(P_x + \hat{U}_x s)^2 + b(P_y + \hat{U}_y s)^2 - (P_z + \hat{U}_z s) = 0 \tag{5}$$

After grouping the terms, the positive value of the parameter s is given by:

$$s = \frac{-B + \sqrt{B^2 - 4AC}}{2A} \tag{6}$$

where $A = (a\hat{U}_x{}^2 + b\hat{U}_y{}^2)$, $B = (2aP_xU_x + 2bP_yU_y - U_z)$, and $C = (aP_x{}^2 + bP_y{}^2 - P_z)$. Note that for the range of values of the variable t for which the viewing ray is inside the quadric, the discriminant of Equation 6 is non-negative. A simple inspection of the geometry of Figure 8 (left) confirms this.

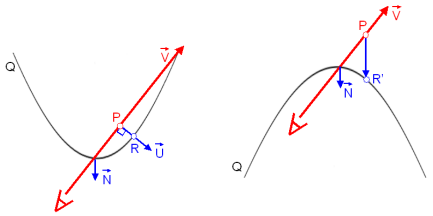

Figure 8: Cross sections of two quadric surfaces. P is a point along the viewing ray. (Left) R is a point on the quadric Q, obtained from P along the direction $\vec{U}$, which is perpendicular to $\vec{V}$. The distance between P and Q is given by the segment $\overline{PR}$. (Right) The distance between P and Q is given by the segment $\overline{PR'}$

If the value of the discriminant of Equation 6 is negative, this indicates that either: (i) both principal curvatures of the quadric are negative (*i.e.*, κ_1 and $\kappa_2 < 0$), or (ii) the Gaussian curvature of the quadric is negative or zero (*i.e.*, $\kappa_1 \kappa_2 \leq 0$). In both cases, the distance between the viewing ray at point P and the quadric Q should be computed as depicted in Figure 8 (right). Thus, given (P_x, P_y, P_z), the coordinates of point P, the expression for computing the ray-quadric distance is given by

$$PQ_{distance} = P_z - (a(P_x)^2 + b(P_y)^2) \tag{7}$$

which is the difference between the Z coordinate of P and the Z coordinate of the quadric evaluated at (P_x, P_y).

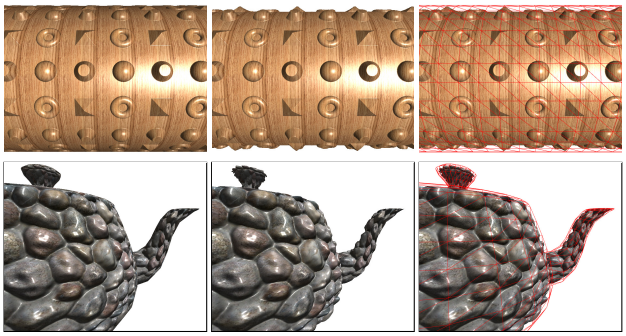

Figure 9: Renderings of two relief-mapped objects: a cylinder (top) and a teapot (bottom). (Left) Image created using the original relief mapping technique. Note the lack of details at the silhouette. (Center) Correct silhouette produced by the proposed technique. (Right) Same as the center image with the superimposed triangle mesh, highlighting the differences.

3.2 A Faster, Approximate Solution

Computing the discriminant of Equation 6 and then deciding whether to evaluate Equation 6 or Equation 7 requires a considerable amount of effort from a GPU, as this procedure has to be repeated several times for each fragment. A much simpler approximate solution can be obtained by using Equation 7 to handle all cases. When the discriminant of Equation 6 is non-negative, the approximation error increases as the viewing direction approaches the direction of vector $\vec{N}$. Figure 10 illustrates this situation.

Compared to the solution represented by Equation 6, this simplification lends to a GPU code with about only half of the number of instructions. In our experiments, we have experienced a two-fold speedup with the implementation of the approximate solution compared to the correct one. It should be noted that approximation errors are bigger for viewing directions closer to the surface normal (see Figure 10). According to our experience, although switching between the two solutions tends to reveal differences between images, consistent use of the approximate solution produces plausible renderings and does not introduce distracting artifacts. Figure 11 compares the renderings produced by the two methods. By examining one image at a time, it is virtually impossible to say which method was used to render it.

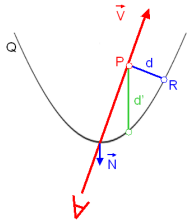

Figure 10: Error in the ray-quadric distance resulting from the approximate solution. The actual distance computed with Equation 6 is represented by the blue segment, while the green segment indicates the approximate distance as computed using Equation 7.

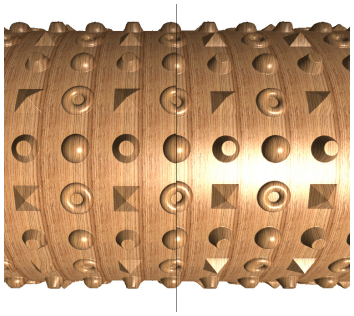

Figure 11: Comparison between the two approaches for computing the distance between the viewing ray and a quadric. (Left) More accurate solution as described in Section 3.1. (Right) The approximate solution based only on Equation 7.

3.3 Reducing the Search Space

In order to optimize the sampling during the linear search and improve rendering speed, the distance D_{rq} from the viewing ray to the quadric should only be computed for the smallest possible range of the parameter t (in Equation 3). Since the maximum depth in the normalized height-field representation is limited to 1.0 (Figure 6), the search can be restricted to the values of t in the interval $[0, t_{max}]$. As the depth map includes no holes, a ray that hits depth 1.0 in texture space has reached the bottom of the height field, characterizing an intersection. On the other hand, a ray that returns to depth 0.0 (*e.g.*, ray B in Figure 6) can be safely discarded as belonging to the silhouette. Thus, t_{max} is the smallest $t > 0$ for which $D_{rq} = 0$ or $D_{rq} = 1$.

For the more accurate solution, t_{max} should be computed by setting $s = 0$ and $s = 1$, substituting $P = (\hat{V}_x t, \hat{V}_y t, \hat{V}_z t)$ (Equation 3) into Equation 5 and then solving for t. For the approximate solution, the value of t_{max} is obtained by setting $PQ_{distance} = 0$ and $PQ_{distance} = 1$ into Equation 7 and then solving for t.

3.4 Computing Intersections in Texture Space

So far, the ray-intersection procedure has been described in the fragment's tangent space. In order to perform the intersection in texture space, we first need to transform both the quadric and the viewing ray to texture space. Thus, let s_x and s_y be the actual dimensions of a texture tile in the tangent space of a given vertex v_k (*i.e.*, the dimensions of a texture tile used to map the triangles sharing v_k). These values are defined during the modeling of the object and stored on a per-vertex basis. Also, let s_z be the scaling factor to be applied to the normalized height-field surface (*i.e.*, the maximum height for the surface details in 3D). The mapping from the relief-texture space to tangent (or object space) can be described as:

$$(x_o, y_o, z_o) = (x_t s_x, y_t s_y, z_t s_z)$$

where the subscripts o and t indicate object and texture space, respectively. Therefore, the quadric defined in tangent space

$$z_o = a x_o^2 + b y_o^2$$

can be rewritten as

$$z_t s_z = a(x_t s_x)^2 + (y_t s_y)^2$$

By rearranging the terms, the same quadric can be expressed in texture space as

$$z_t = \alpha x_t^2 + \beta y_t^2 \tag{8}$$

where $\alpha = a(s_x^2/s_z)$ and $\beta = b(s_y^2/s_z)$. x_t, y_t and z_t are all in the range $[0, 1]$. Likewise, the viewing direction in texture space, V_t, is obtained from V_o, the viewing direction in object space, as:

$$(V_{tx}, V_{ty}, V_{tz}) = (V_{ox}/s_x, V_{oy}/s_y, V_{oz}/s_z) \tag{9}$$

Using Equations 8 and 9, the entire computation can be performed in texture space. The values of s_x and s_y will be interpolated during rasterization, while s_z is parameter controlled by the user.

3.5 Depth Correction and Shadows

In order to be able to combine the results of relief-mapped renderings with arbitrary polygonal scenes, one needs to update the Z-buffer appropriately to compensate for the simulated geometric details. This is required for achieving correct surface interpenetrations as well as to support the shadow mapping algorithm [Williams 1978]. Thus, let *near* and *far* be the distances associated with the near and far clipping planes, respectively. The Z value that needs to be stored in the Z-buffer for a given relief-mapped fragment is given by:

$$Z = \frac{z_e(far+near)+2(far)(near)}{z_e(far-near)}$$

where z_e is the z coordinate of the fragment expressed in eye space. Figure 12 shows two coincident cylinders mapped with relief of different materials. Note the correct interpenetration involving the surface details. Depth correction at fragment level using Z-buffer modulation has also been used in [Oliveira et al. 2000; Gumhold 2003].

As we distort a height field according to the local surface, one has no direct way of computing the texture coordinates associated with the entry point of a shadow ray, as done in the original

relief-mapping formulation (Figure 4 (left)). As a result, we render shadows using the hardware support for shadow mapping [Williams 1978]. Figure 13 shows a scene depicting shadows cast by the simulated relief onto other scene objects and vice-versa.

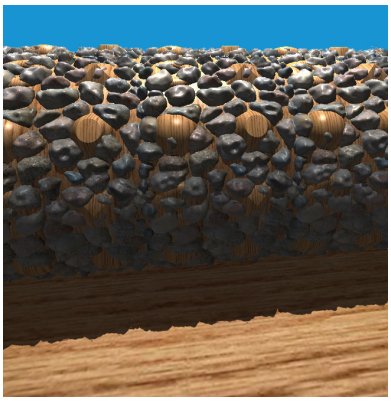

Figure 12: Two coincident cylinders mapped with relief of different materials. Note the correct interpenetrations of the surface details.

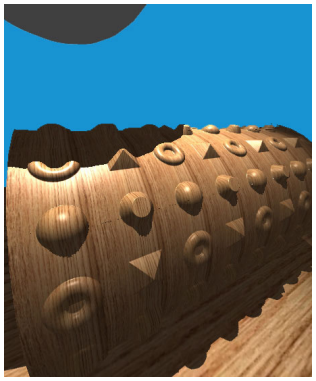

Figure 13: Shadows cast by the simulated relief onto other scene objects and vice-versa.

4 Results

We have implemented the techniques described in the paper as fragment programs written in Cg and used them to map details to the surfaces of several polygonal objects. The computation of the per-vertex quadric coefficients was performed off-line using a separate program. Relief mappings are defined in the conventional way, assigning texture coordinates to the vertices of the model. All textures used to create the illustrations shown in the paper and the accompanying videos are 256x256 RGBα textures. The depth maps were quantized using 8 bits per texel. The quantized values represent evenly spaced depths, and can be arbitrarily scaled during rendering time using the s_z parameter mentioned in Section 3.4. Shadows were implemented using shadow maps with 1024x1024 texels. Except for the examples shown in Figures 11 (left) and 14, all images were rendered using the approximate algorithm described in Section 3.2. The scenes were rendered at a resolution of 800x600 pixels at 85 frames per second, which is the refresh rate of our monitor. These measurements were made on a 3 GHz PC with 512 MB of memory using a GeForce FX6800 GT with 256 MB of memory. Accompanying videos were generated in real time.

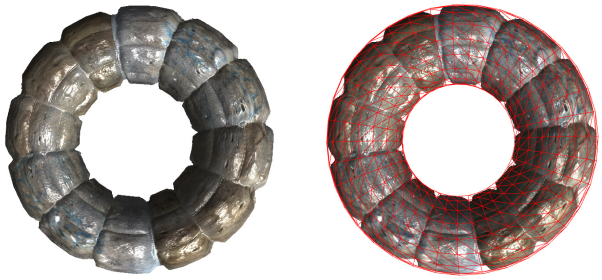

Figure 15: Torus with stone relief mapping. (Left) Image produced with the new algorithm. Note the silhouette. (Right) The superimposed triangle mesh reveals silhouette fragments.

Figures 1 and 16 show two scenes containing several relief texture-mapped objects. In Figure 1, the columns, walls and a stone object at the center of the room were rendered using relief mapping. Notice the correct silhouettes and the shadows cast on the walls and floor. Figure 16 shows another scene where the ceiling and a pathway are also relief-mapped objects.

Figures 9 compares the renderings produced by the original technique and the new one. The images on the left were rendered using the technique described in [Policarpo et al. 2005]. Notice the objects' polygonal silhouettes. At the center, we have the same models rendered using the proposed technique. In this case, the silhouettes exhibit the correct profile for the surface details. The rightmost columns of Figures 9 and 17 also show the renderings produced with the new technique, but superimposed with the corresponding triangle meshes. The meshes highlight the areas where differences between the results produced by the two techniques are mostly noticeable. The texture filtering associated with the sampling of the relief textures guarantees that the sampling is performed against a reconstructed version of the height-field surfaces. As a result, the technique allows for extreme close-up views of objects' surfaces and silhouettes. The results are good even for low-resolution textures.

Figure 11 shows a comparison between some results produced by the more accurate and the approximate solutions for computing the intersection of a viewing ray with a height-field surface. Although a side-by-side comparison reveals some differences, according to our experience, the use of the approximate solution does not introduce any distracting artifacts. Moreover, considering one image at a time, it is virtually impossible to distinguish the results produced by both algorithms.

Figure 12 shows two interpenetrating surfaces whose visibility is solved by Z-buffer modulation, as discussed in Section 3.5. These surfaces correspond to two coincident cylinders that were relief-mapped using different textures.

Examples of shadows involving mapped relief and other scene objects are depicted in Figure 13. Note that the relief details cast correct self-shadows as well as shadows on other polygonal objects in the scene. Likewise, shadows cast by other elements of the scene are correctly represented in the simulated geometry.

Figure 15 show a top view of a torus revealing some of the details of its silhouette. Figure 14, on the other hand, shows a close-up view of a saddle region corresponding to a portion of another torus. This illustrates the ability of our algorithm to successfully handle

Figure 14: Close-up of a portion of a torus mapped with a stone relief texture, illustrating the rendering of a surface with negative Gaussian curvature. (Left) Original technique. (Center) Proposed approach; (Right) Polygonal mesh highlighting the differences.

Figure 16: Scene containing several relief-mapped objects: a column, the ceiling, the walls, and a pathway.

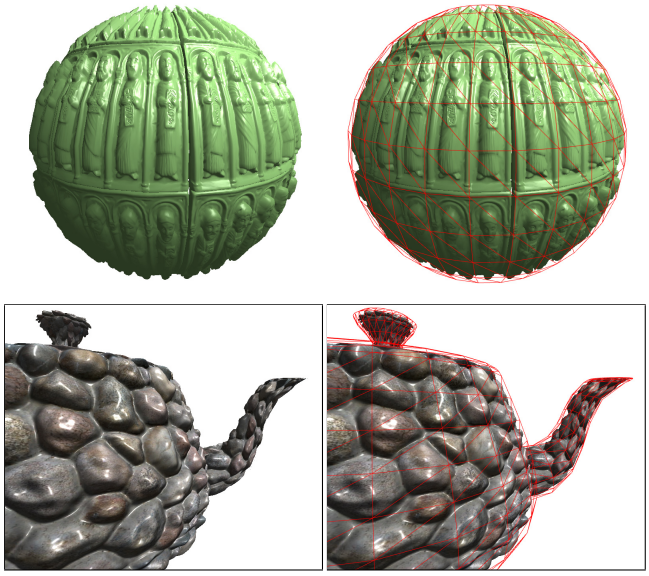

Figure 17: Relief-mapped objects rendered using the proposed approach. Notice the correct silhouettes (left).The superimposed triangle meshes highlight the differences between the obtained silhouettes and the polygonal ones.

surfaces with negative Gaussian curvature. Figure 18 shows the same torus of Figure 15 textured using different numbers of tiles.

5 Conclusion

We have introduced an efficient technique for rendering surface details onto arbitrary polygonal models. Like conventional displacement mapping, this new approach produces correct silhouettes, self-occlusions, interpenetrations and shadows. Unlike displacement maps, however, it does not require any changes to the object's original geometry nor involves rendering micro-polygons. The technique works in image space and has very low memory requirements. The on-the-fly filtering performed during the sampling of the textures used to store depth maps guarantees that we always sample a reconstructed version of the height-field surfaces. As a result, the technique supports extreme close-up views even with low-resolution relief textures.

In this new approach, object surfaces are locally approximated using a piecewise-quadric representation at fragment level. The algorithm is based on a ray-intersection procedure performed in texture space that can be efficiently implemented on current GPUs. As such, this paper demonstrates an effective way of using graphics hardware for image-based rendering.

As in the original relief mapping algorithm [Policarpo et al. 2005], a linear search is used to obtain an approximate location of the first intersection between the viewing ray and the height-field surface. This approximate location is further improved using a binary search. However, depending on the step size used, it might be possible for the linear search to miss very fine structures in the height field. Although in practice we have not noticed such aliasing artifacts, an improved sampling strategy will probably be necessary for rendering very fine details. We are also investigating ways of accelerating the search for the first intersection point. The use of space leaping [Wan et al. 2002] seems to be a promising approach, with the potential to significantly reduce the number of instructions that need to be executed by a pixel shader.

Figure 18: A torus mapped with a relief texture using different numbers of tiles.

References

BLINN, J. F. 1978. Simulation of wrinkled surfaces. In *Proceedings of the 5th annual conference on Computer graphics and interactive techniques*, ACM Press, 286–292.

COOK, R. L. 1984. Shade trees. In *Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, ACM Press, 223–231.

DOGGETT, M., AND HIRCHE, J. 2000. Adaptive view dependent tessellation of displacement maps. In *HWWS '00: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*, ACM Press, 59–66.

GUMHOLD, S. 2003. Splatting illuminated ellipsoids with depth correction. In *8th International Fall Workshop on Vision, Modelling and Visualization*, 245–252.

HEIDRICH, W., AND SEIDEL, H.-P. 1998. Ray-tracing procedural displacement shaders. In *Graphics Interface*, 8–16.

HEIDRICH, W., DAUBERT, K., KAUTZ, J., AND SEIDEL, H.-P. 2000. Illuminating micro geometry based on precomputed visibility. In *Siggraph 2000, Computer Graphics Proceedings*, ACM Press / ACM SIGGRAPH / Addison Wesley Longman, K. Akeley, Ed., 455–464.

HIRCHE, J., EHLERT, A., GUTHE, S., AND DOGGETT, M. 2004. Hardware accelerated per-pixel displacement mapping. In *Graphics Interface*, 153 – 158.

KANEKO, T., TAKAHEI, T., INAMI, M., KAWAKAMI, N., YANAGIDA, Y., MAEDA, T., AND TACHI:, S. 2001. Detailed shape representation with parallax mapping. In *Proceedings of the ICAT 2001*, 205–208.

KAUTZ, J., AND SEIDEL, H.-P. 2001. Hardware accelerated displacement mapping for image based rendering. In *Proceedings of Graphics Interface 2001*, B. Watson and J. W. Buchanan, Eds., 61–70.

MAX, N. 1988. Horizon mapping: shadows for bump-mapped surfaces. *The Visual Computer 4*, 2, 109–117.

MEYER, A., AND NEYRET, F. 1998. Interactive volumetric textures. In *Eurographics Rendering Workshop 1998*, Springer Wein, New York City, NY, G. Drettakis and N. Max, Eds., Eurographics, 157–168. ISBN 3-211-83213-0.

MOULE, K., AND MCCOOL, M. 2002. Efficient bounded adaptive tesselation of displacement maps. In *Graphics Interface*, 171–180.

OLIVEIRA, M. M., BISHOP, G., AND MCALLISTER, D. 2000. Relief texture mapping. In *Siggraph 2000, Computer Graphics Proceedings*, ACM Press / ACM SIGGRAPH / Addison Wesley Longman, K. Akeley, Ed., 359–368.

PATTERSON, J., HOGGAR, S., AND LOGIE, J. 1991. Inverse displacement mapping. *Computer Graphics Forum 10*, 2, 129–139.

PHARR, M., AND HANRAHAN, P. 1996. Geometry caching for ray-tracing displacement maps. In *Eurographics Rendering Workshop 1996*, Springer Wien, New York City, NY, X. Pueyo and P. Schröder, Eds., 31–40.

POLICARPO, F., OLIVEIRA, M. M., AND COMBA, J. 2005. Real-time relief mapping on arbitrary polygonal surfaces. In *Proceedings of ACM Symposium on Interactive 3D Graphics and Games 2005*, To Appear.

SCHAUFLER, G., AND PRIGLINGER, M. 1999. Efficient displacement mapping by image warping. In *Eurographics Rendering Workshop 1998*, Springer Wein, New York City, NY, D. Lischinski and G. Larson, Eds., Eurographics, 175–186. ISBN 3-211-83382-X.

SMITS, B. E., SHIRLEY, P., AND STARK, M. M. 2000. Direct ray tracing of displacement mapped triangles. In *Proceedings of the Eurographics Workshop on Rendering Techniques 2000*, Springer-Verlag, 307–318.

SYLVAIN, P. 2002. A survey of methods for recovering quadrics in triangle meshes. *ACM Computing Surveys 2*, 34 (July), 1–61.

WAN, M., SADIQ, A., AND KAUFMAN, A. 2002. Fast and reliable space leaping for interactive volume rendering. In *Proceedings of the conference on Visualization '02*, IEEE Computer Society, 195–202.

WANG, L., WANG, X., TONG, X., LIN, S., HU, S., GUO, B., AND SHUM, H.-Y. 2003. View-dependent displacement mapping. *ACM Trans. Graph. 22*, 3, 334–339.

WANG, X., TONG, X., LIN, S., HU, S., GUO, B., AND SHUM, H.-Y. 2004. Generalized displacement maps. In *Eurographics Symposium on Rendering 2004*, EUROGRAPHICS, Keller and Jensen, Eds., EUROGRAPHICS, 227–233.

WILLIAMS, L. 1978. Casting curved shadows on curved surfaces. In *Siggraph 1978, Computer Graphics Proceedings*, 270–274.