Pacific Graphics 2008
T. Igarashi, N. Max, and F. Sillion
(Guest Editors)

Volume 27 (2008), Number 7

A Prism-Free Method for Silhouette Rendering in Inverse Displacement Mapping

Ying-Chieh Chen[1] and Chun-Fa Chang[2]

[1]National Tsing Hua University, Taiwan
[2]National Taiwan Normal University, Taiwan

Abstract

Silhouette is a key feature that distinguishes displacement mapping from normal mapping. However the silhouette rendering in the GPU implementation of displacement mapping (which is often called inversed displacement mapping) is tricky. Previous approaches rely mostly on construction of additional extruding prism-like geometry, which slows down the rendering significantly. In this paper, we proposed a method for solving the silhouette rendering problem in inverse displace mapping without using any extruding prism-like geometry. At each step of intersection finding, we continuously bends the viewing ray according to the current local tangent space associated with the surface. Thus, it allows mapping a displacement map onto an arbitrary curved surface with more accurate silhouette. While our method is simple, it offers surprisingly good results over Curved Relief Map (CRM) [OP05] in many difficult or degenerated cases.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Picutre/image Generation]: Display algorithms I.3.7 [Three-Dimensional Graphics an Realism]: Color, shading, shadowing, and texture

1. Introduction

Over the past years, we have seen an impressive improvement in graphics hardware, especially on the capabilities of graphics processing units (GPUs). Among many hardware features, the most prominent improvement is the realization of programmable shaders for real-time rendering. Instead of relying on the raw power of geometry processing, we now have an alternative of improving the visual realism of surface details through advanced texturing techniques.

Bump mapping [Bli78] provides more realistic look by adjusting per-pixel normal. However it captures only the shading effect caused by surface normal of the rasterization point. The other important visual effects such as self-occlusion, self-shadow and silhouette are not performed. Displacement mapping [CCC87] provides a more effective representation for surface details with a displacement map. Points on a surface are moved along their normals by the amount specified by the displacement map. All the features that bump mapping fails to capture are naturally supported by displacement mapping. However, it requires altering the geometry during shading. Even with today's graphics hardware, this is still an expensive operation.

Inverse displacement mapping [PHL91] is a class of mapping methods which attempt to render those effects offered by displacement mapping, such as motion parallax, self-occlusion, self-shadowing and silhouette, without actually perturbing the geometry of the surface. These methods have several advantages: they require lower amount of memory; they do not need to change the original geometry; and most importantly, they are performed in image space and can be efficiently implemented using fragment shaders on current GPUs. Recent progresses of inverse displacement mapping take advantage of the programmability and parallel nature of modern graphics hardware to render surface details in real time. The key question these methods attempt to answer is how to effectively search for the closest intersection to the surface defined by the displacement map along a given viewing ray. A compromise between convergence speed and rendering quality is often made by these methods. Another important problem that inverse displacement mapping has to deal with is how to render object's silhouette correctly.

Most inverse displacement mapping methods focus on flat surfaces, thus ignoring the subtle problem of silhouette rendering. Although solutions do exist to tackle the silhouette problems, they mostly rely on construction of additional extruding prism-like geometry, which slows down the rendering significantly. A solution without the use of prism remains elusive.

This paper proposes an extensions to existing inverse displacement mapping methods, called normal-based curved silhouette (NCS). It renders more accurate silhouette for inverse displacement mapping, especially when mapping to a curved surface. It differs from curved relief mapping [OP05] by offering a more accurate and robust representation of the surface curvature.

The main contribution of this paper is a novel method to render object's silhouette on arbitrary surface, which does not require the construction of additional prism-like geometry [HEGD04] [JMW07] and is more robust and accurate than previous methods. In the next section, we do a thorough survey on available real-time inverse displacement mapping methods.

2. Background and Related Work

The basic idea of real-time inverse displacement mapping is shown in Figure 1(a). Given a height field $h(s)$ and a viewing vector V, assume that V intersects the surface to be mapped at the point P with the texture coordinate s. (Note that, as most inverse displacement mapping methods, we assume that the surface to be mapped is at the top.) Bump mapping assumes that the real intersection is exactly at the texture coordinate and performs shading at P. However, the real intersection should be I with texture coordinate t. Hence, the color of P should be calculated according to the shading attributes associated with t instead of s.

The basic procedure of inverse displacement mapping is [POC05] : (1) transform the viewing direction V to the tangent space associated with the current fragment with texture coordinate s (defined by tangent, normal, and bi-normal vectors), (2) find the intersection I of V and the surface defined by the displacement map h, and (3) compute shading of the fragment using the attributes associated with I.

Inverse displacement mapping methods mainly differ in the ways to find the intersection I. There are basically two types of methods, root finding and space leaping. The former requires no preprocessing but is more prone to aliasing artifacts, especially from grazing angles. The later often needs preprocessing but tends to find more accurate intersections.

Parallax mapping (PM) adds an offset to s which is linearly proportional to the height specified by the map [KTI*01] as shown in Figure 1(b). This, of course, is an very rough approximation. It only works well for low-frequency bump maps. It often breaks for steep bump maps or when viewed from a shallow viewing angle. Walsh added a heuristics to alleviate artifacts for shallow views to some degree [Wal03].

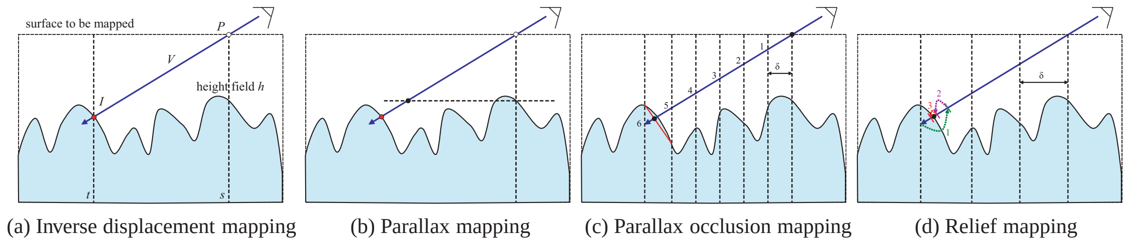
Instead of using heuristics to find intersection, both parallax occlusion mapping (POM) [BT04] [Tat06] and steep parallax mapping [MM05] use linear search to find a more accurate intersection. Though effective, the step size δ has to be selected carefully. A smaller step size leads to more accurate result but at the expense of longer convergence time. On contrary, larger step size is faster but might lead to errors. The final intersection is found by intersecting the viewing ray and a linear approximation of the displacement map(Figure 1(c)).

Relief mapping (RM) proposed to employ binary search to speed up intersection finding [POC05]. However, pure binary search may miss the closest intersection and lead to errors when the viewing ray intersects the surface more than once. Hence, RM used a combination of linear and binary search to make a good tradeoff between guaranteed accuracy and convergence speed (Figure 1(d)). As pointed by Tatarchuk [Tat06], this approach could lead to visual artifacts due to sampling aliasing at grazing angles. Such artifacts could be relieved by applying some depth bias. This bias however flattens the surface features towards the horizon.

Interval mapping (IM) replaces the binary search of RM with a root finding algorithm to speed up the convergence [RSP06] (Figure 1(e)). These root finding methods, however, all face two fundamental problems. First, it is necessary to carefully determine a step size for different displacement maps. Second, there is an accuracy problem due to the discontinuity in the first derivatives.

The second category of approaches, space leaping, tries to find more accurate intersections by building a conservative bounding volume for empty space. Instead of marching slowly, the bounding volume allows quickly skipping through empty space while maintaining sufficient accuracy. These methods differs mainly on the types of bounding volumes. This set of techniques almost guarantee artifact free even when zoomed in or viewed at grazing angles. For the same visual quality, they are often faster than techniques of the first category. They, however, often require preprocessing to calculate the bounding volumes. Baboud et al. used a precomputed safety radius texture such that a viewing ray could walk along until a binary search can be safely run [BD06] (Figure 1(f)). One disadvantage for this is that it requires a long preprocessing by sampling lots of rays. In addition, to find the intersection, a binary search is still required.

Distance mapping (DM) uses a 3D texture to store the shortest distance from any 3D-point of the empty space to the surface specified by the displacement map [Don05]. It is equivalent to use a sphere as the bounding volume for the empty space around any point within the volume (Figure 1(g)). The results are very accurate. However, it uses



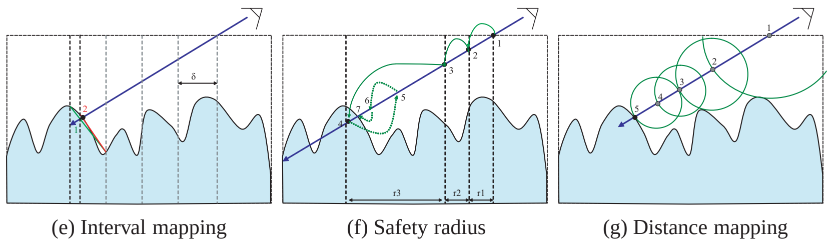


Figure 1: *Summary of existing inverse displacement mapping methods.*

an order of magnitude more memory than standard 2D textures. Other more general precomputation-based approaches such as view-dependent displacement mapping [WWT*03] and generalized displacement mapping [WTL*04] can be used for inverse displacement mapping as well. However, they face the same problem of large memory consumption.

Though effective for self-occlusion, shadows and interpenetrations, most methods do not handle object's silhouette correctly, especially when applying to non-planar surfaces. Oliveira et al. [OP05] proposed Curved Relief Mapping (CRM) to address this issue. This method, however, requires a preprocessing stage to find a piecewise-quadric approximation for the surface to be mapped. Hirche et al. [HEGD04] and Dachsbacher et al. [DT07] tackled the silhouette problem by extruding each triangle of the base mesh along the normal directions to form a prism. The use of prisms however inevitably produces larger number of triangles and complicates the intersection process. Jeschke et al. [JMW07] proposed more accurate prism representations for smooth and curved shell maps. However, the increased accuracy comes at the price of slower performance. While the silhouette problem can be solved by using additional prism geometry, a solution with no prism remains elusive.

3. Prism-Free Silhouette Rendering Methods

Curved Relief Mapping(CRM) [OP05] approximates the surface curvature using a piecewise-quadratic approximation. While this approach works well in most cases, it could break in some special situations such as the case shown in Figure 2. In this section, we present *Normal-Based Curved Silhouette* (NCS) to render object's silhouette more accurately than CRM though imposing some constraints on surface parameterization.

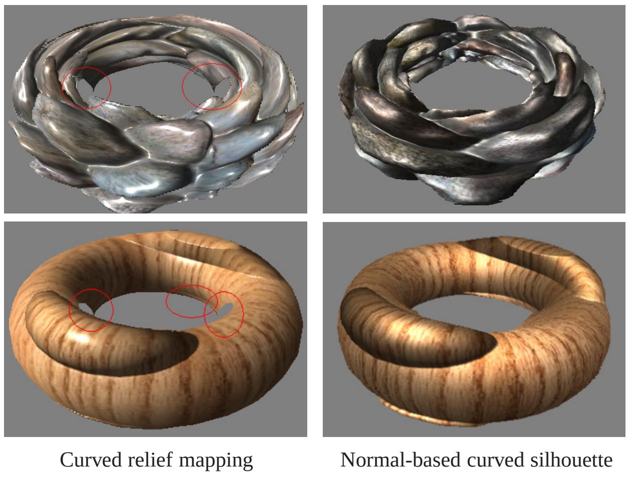


Figure 2: *Comparisons between curved relief mapping and normal-based curved silhouette. Circled areas show rendering artifacts using CRM.*

CRM pre-computes curvatures of the base mesh. Instead of using pre-computed curvatures, we use the tangent space associated with each vertex to describe the local shape of the base mesh. Such a tangent space is defined with vertex's normal, tangent and bi-normal vectors. The advantage of using these vectors is that they have often been included in the base mesh.

The basic idea of our algorithm is as follows. As shown in Figure 3, the viewing ray is straight in the object space, but curved in the texture space. If we bend the viewing ray according to the tangent space of the current fragment when marching along the viewing ray, we essentially obtain

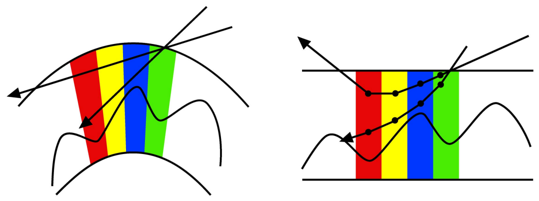


Figure 3: *The basic idea of our rendering method. The colors show the mapping between object space and tangent space. We update the surface local information (green, blue, yellow and red parts) for each step of intersection finding process. A straight viewing ray in the object space becomes curved in the texture space.*

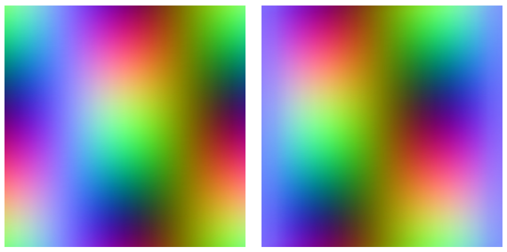


Figure 4: *Normal and tangent maps for a torus model.*

a piecewise linear approximation of the curved viewing ray in the texture space. Thus, if the bent ray hits the displacement map, it is shaded. Otherwise, it should be discarded. However, to achieve this, we need to obtain the local tangent space for any fragment, not just for vertices. In the next section, we explain how to generate tangent space maps and how to use those maps.

3.1. Tangent Space Map Generation

The tangent space attributes of a surface point include the tangent $\vec{T}$, normal $\vec{N}$ and bi-normal vectors $\vec{B}$,and two scaling factors $S_{tangent}$ and $S_{binormal}$ to handle the variation in triangle sizes. (Without those scaling factors, texture swimming artifacts may appear.) To answer the tangent space query, , we render the normal map ($\mathbf{N}_{map}$) and the tangent map ($\mathbf{T}_{map}$), including their scaling factors, for the base mesh. With these maps, for a given texture coordinate (u,v) of the current fragment, we can look up the textures for the associated tangent space attributes by $\mathbf{N}_{map}(u,v)$ and $\mathbf{T}_{map}(u,v)$. Algorithm 1 presents the method for rendering normal and tangent maps. Figure 4 shows the normal and tangent maps for a torus. Note that this method assumes that the texture coordinate is parameterized continuously over the base mesh. In addition, tiling of the displacement maps needs to be handled with extra care.

Algorithm 1 Normal/tangent map rendering. *Given a base mesh M, render normal map* **N** *and tangent map* **T**.

```
1: procedure RenderingNTMaps(TriangleMesh $M$)
2:     set up the rendering targets to normal map N and tangent map T;
3:     for each triangle $t$ of $M$ with vertex normals $N_1$, $N_2$, $N_3$, vertex tangents $T_1$, $T_2$, $T_3$, and texture coordinates $(u_1,v_1)$, $(u_2,v_2)$, $(u_3,v_3)$ do
4:         form a triangle $t'$ with vertices $(u_1,v_1,0)$, $(u_2,v_2,0)$, $(u_3,v_3,0)$;
5:         render $t'$ with vertex colors $N_1$,$N_2$,$N_3$, $S_{binormal}$ to texture N;
6:         render $t'$ with vertex colors $T_1$,$T_2$,$T_3$, $S_{tangent}$to texture T;
7:     end for
8: end procedure
```

Algorithm 2 Rendering pass. *Given a displacement map* **H**, *normal and tangent maps,* **N** *and* **T**, *of the base mesh, render a fragment.* $^tP_0$ *is the texture coordinate from resterizer.* $^o\vec{V}$ *is the viewing direction vector in object space.* $[S]_i$ *and* $[TBN]_i$*mean the scaling factors and the tangent space at ith iteration.* δ *is the default step size, which could be enlarged by a constant c (See Section 5).*

```
1: procedure RenderingPass($^tP_0$, $^o\vec{V}$, H, N, T )
2:     for $i$ = 0 to num_step do
3:         $([TBN]_i, [S]_i) \leftarrow$ SetData(T, N, $^tP_i$);
4:         $^tP_{i+1} \leftarrow {}^tP_i + c\cdot\delta\cdot[S]_i\cdot[TBN]_i\cdot{}^o\vec{V}$;
5:         if $(^tP_{i+1}).z < 0$ then
6:             DiscardFragment(); return ;
7:         else if $(^tP_{i+1}).z > \mathbf{H}((^tP_{i+1}).xy)$ then
8:             FindIntersection(); return ;
9:         end if
10:    end for
11: end procedure
```

One may argue that it is not necessary to render the normal and tangent maps into textures, since they may be interpolated from the vertices during the rasterization step. The main problem with such a strategy is that the viewing ray may go beyond the texture volume of the currently shaded triangle, where the interpolation no longer works.

3.2. Normal-Based Curved Silhouette Rendering

During rendering, for each step, we obtain the tangent space of the current fragment by looking up the normal and tangent maps and bend the viewing ray accordingly. If the ray runs away from the texture volume, the fragment is discarded. If there is an intersection, the exact intersection is found. Otherwise, the process continues. When the loop exceeds a pre-defined number of steps, the fragment is discarded since it does not have any intersection. Algorithm 2 presents the process.

Although the idea of representing the tangent space in

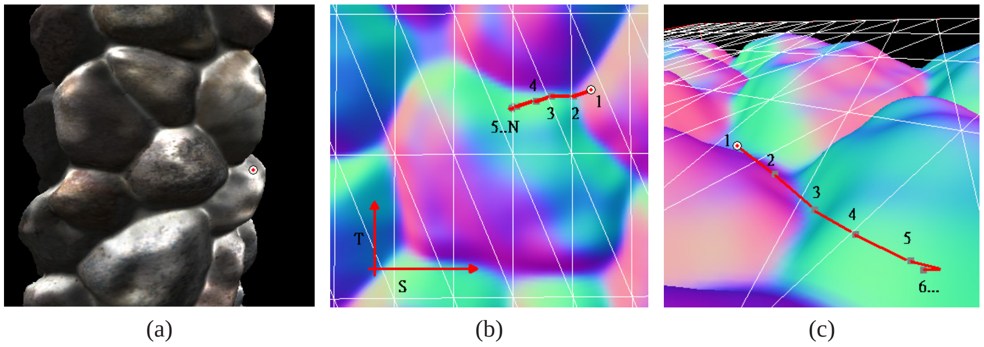


Figure 5: *Trace of the viewing ray for the traced pixel (a). (b) shows the trace in the texture volume and (c) shows a perspective view of the texture volume.*

textures is simple, it offers surprisingly good results over CRM in many difficult or degenerated cases and better performance than prism-based methods. Figure 2 shows such examples. (Note that the rendering views are slightly different because the renderings for CRM are obtained using the CRM program available from the author's webpage.) In the circled areas of Figure 2, CRM causes some viewing rays (that hit the base mesh at grazing angles) to follow the surface curvature toward the back-facing side of the torus and miss the actual intersection with the torus at a farther front-facing triangle. Our proposed method avoids such a problem because it follows the surface curvature more accurately by looking up the normal and tangent maps (Algorithm 2 line 3) at each traversal step. Figure 5 shows an example of the trace of the viewing ray in the texture volume. Figure 5(a) shows the location of the traced pixel in the rendered image. Figure 5(b) shows the trace of the viewing ray in the texture volume. Figure 5(c) shows a perspective view of the texture volume.

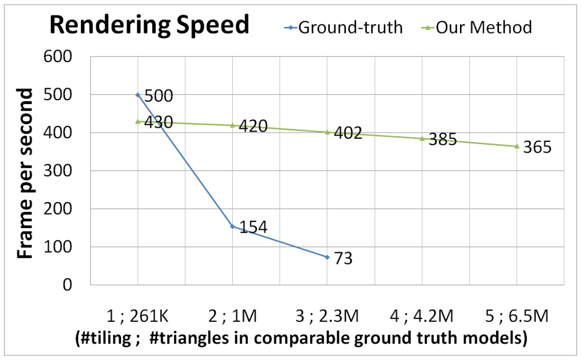


Figure 6: *This figure shows the frame rate for our method, ground-truth at different times of texture tiling. The resolution of both the screen and the height map are 512 by 512, and the base model has 528 triangles. The triangles of ground truth models are generated at the texture resolution and show on the horizontal axis.*

4. Results and Performance

Methods	Frame-rate	#tri.	height map res.
CRM [OP05]	> 999	538	512 × 512
Our method	350	538	512 × 512
Tetra shell [HEGD04]	*51	602	128 × 128
Smooth shell [JMW07]	18	602	128 × 128

Table 1: *Comparison of rendering speed. All renderings are in the same 640 by 480 resolution. * The frame rate in [HEGD04] is obtained from the comparison reported in [JMW07]. The first two are performed on NVIDIA Geforce 8800GTS/320MB. The last two are on two NVIDIA Geforce 8800GTX running in SLI mode.*

We implemented our algorithms as Cg shaders with fp40 profile. Our experiments were performed on an Intel Pentium4 3.0GHz machine with NVIDIA Geforce 8800GTS with 320MB video memory. Figure 2 shows our methods produce correct silhouettes in cases that are known to be problematic for Curved Relief Mapping (which is, to our knowledge, the only other prism-free inverse displacement mapping method). Figure 8 shows more rendering results using our methods. Although the base meshes in the typical applications of displacement mapping are relatively simple, we performed a test using the Venus model as our base mesh. Its purpose is to show our methods can handle complex base mesh as well. The Venus model contains 524K triangles and is parameterized using the geometry images [GGH03]. Figure 9 shows its rendering results. Figure 10 demonstrates the results for a complex scene consisting of multiple displacement-mapped objects.

The rendering speed of our methods is about 1/3 of the Curved Relief Mapping. The slow-down is caused by more texture memory accesses. However our rendering speed is significantly faster than prism-based approaches [HEGD04] [JMW07]. Table 1 summarizes the frame rates of various methods, all rendered at the same 640 by 480 resolution. Both the Curved Relief Mapping (CRM) and our method are measured on our machine with NVIDIA Geforce 8800GTS/320MB. The other frame rates were reported in [JMW07], which used two NVIDIA Geforce 8800GTX graphics boards in SLI mode.

Since the modern GPUs possess huge raw geometry processing power, it is interesting to compare the performance of our method to the alternative of using more geometry to replace the displacement maps. We could form the "ground truth" models by generating a new mesh vertex for each texel on the displacement mapped surface. Figure 6 shows the speed comparison of our methods versus the rendering using the ground truth models. We used a base mesh that contains 528 triangles and a 512 by 512 displacement map. The horizontal axis represents the number of displacement map tiling. With multiple tiling of the displacement map, the number of triangles in the corresponding ground truth model increases as well. Figure 6 clearly shows that the ground truth models quickly used up the GPU's geometry processing power, while the inverse displacement mapping offers much better scalability in surface detail.

Recently, real-time tessellation has been demonstrated on some newest generations of GPUs [Tat08]. When the GPU-based tessellation is applied to the displacement mapping, it could produce up to six times of speedup over the ground truth models. However the tessellated data still have to go through the vertex shaders and the remaining GPU pipeline. So our methods can still offer a better scalability.

5. Discussion

A potential problem in our methods is that the ray surface intersection test might terminate prematurely in some extreme cases if the step length (i.e. $c \cdot \delta$ in Algorithm 2 line 4) is not selected properly. This problem does not exist on a flat surface, because it is easy to determine the length of the viewing ray within the texture volume. Thus it is straightforward to divide that length l by the number of steps n and set the step length to be l/n. However, on arbitrary surfaces, the viewing ray becomes an unpredictable curve in texture space. Therefore, it is hard to guarantee that the viewing path is completely traced after all iterations. We call this the early termination problem if the trace of the viewing ray still stays within the texture volume after all iterations. We handle the early termination problem by adding a constant c to enlarge the step length in Algorithm 2 (line 4). Figure 7 shows the effects of using different choices of constant c. Note that the tangent approximation error becomes visible when the step length is set too large. We choose the constant $c = 2.5$ and use it throughout the results in this paper.

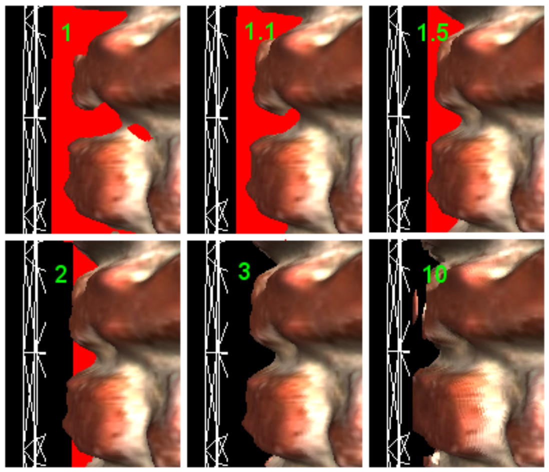


Figure 7: *This figure shows the early termination problems under different setting of constant c in Algorithm 2. The constant c is set to 1.0, 1.1, 1.5, 2.0, 3.0, and 10.0 respectively. Early termination occurs in red pixels.*

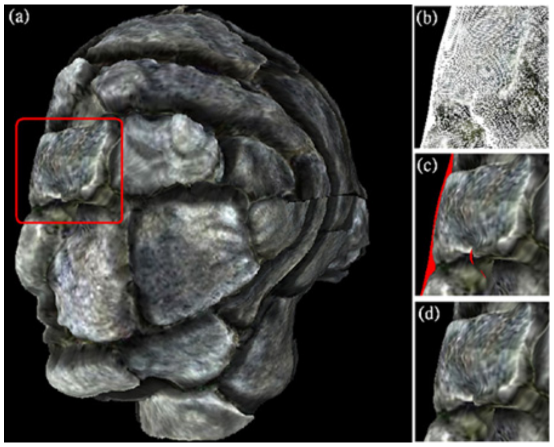


Figure 9: *Rendering of a complex model, Venus, with roughly 524K triangles. We obtained a continuous parameterization using geometry image and applied the stone displacement map over the base mesh. (a) The rendering. (b) The base mesh for the circled region. (c) The red pixels are carved by our algorithm. (d) Rendering with correct silhouette.*

6. Conclusions and Future Work

We have presented an extension towards more accurate silhouette for real-time inverse displacement mapping on arbitrary surface. Our methods do not require the construction of additional prism-like geometry and is more robust than Curved Relief Mapping (CRM). In the comparison with the ground truth models, our methods also demonstrate better scalability in the increase of surface detail. A possible im-

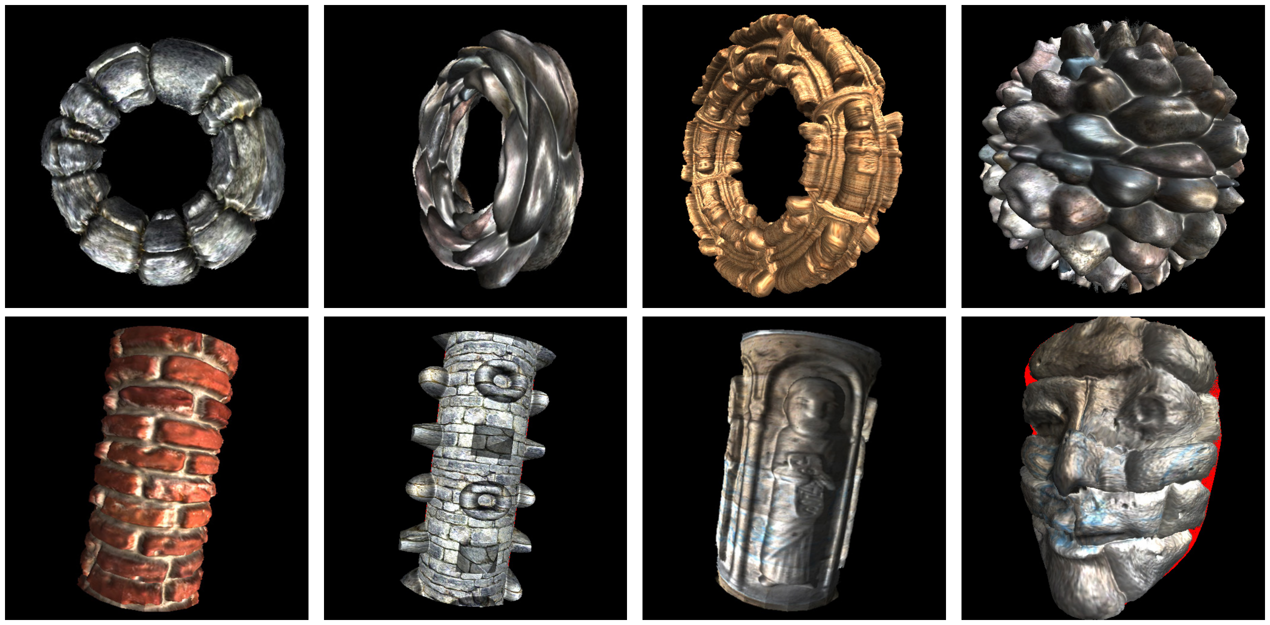


Figure 8: *Renderings of our method for various meshes on several displacement maps.*

provement for the future work is to use a better strategy for selecting the step length. It would also be interesting to explore the possibility of applying inverse displacement mapping to more general texture parameterizations.

7. Acknowledgement

Special thanks to Yung-Yu Chuang and Yu-Jen Chen for their help during the early stage of this work. This work is supported in part by National Science Council (Taiwan) under grant NSC 96-2220-E-003-001.

References

[BD06] BABOUD L., DÉCORET X.: Rendering geometry with relief textures. In *Proceedings of Graphics Interface 2006* (2006).

[Bli78] BLINN J. F.: Simulation of wrinkled surfaces. In *Proceedings of ACM SIGGRAPH* (1978), pp. 286–292.

[BT04] BRAWLEY Z., TATARCHUK N.: Parallax occlusion mapping: Self-shadowing, perspective-correct bump mapping using reverse height map tracing. In *ShaderX^3: Advanced Rendering with DirectX and OpenGL, W. Engel Ed.* (2004), Charles River Media, pp. 135–154.

[CCC87] COOK R. L., CARPENTER L., CATMULL E.: The Reyes image rendering architecture. In *Proceedings of ACM SIGGRAPH 1987* (1987), pp. 95–102.

[Don05] DONNELLY W.: Per-pixel displacement mapping with distance functions. In *GPU Gems 2, Matt Pharr Ed.* (2005), Addison-Wesley, pp. 123–136.

[DT07] DACHSBACHER C., TATARCHUK N.: Prism parallax occlusion mapping with accurate silhouette generation. In *proceedings of ACM SIGGRAPH Symposium on Interactive 3D graphics and games (SI3D '07) as poster* (2007).

[GGH03] GU X., GORTLER S. J., HOPPE H.: Geometry images. *ACM Transactions on Graphics 22*, 3 (2003), 355–361.

[HEGD04] HIRCHE J., EHLERT A., GUTHE S., DOGGETT M.: Hardware accelerated per-pixel displacement mapping. In *Proceedings of Graphics Interface 2004* (2004), pp. 153–158.

[JMW07] JESCHKE S., MANTLER S., WIMMER M.: Interactive smooth and curved shell mapping. In *Rendering Techniques 2007 (Proceedings Eurographics Symposium on Rendering)* (2007), Kautz J., Pattanaik S., (Eds.), pp. 351–360.

[KTI*01] KANEKO T., TAKAHEI T., INAMI M., KAWAKAMI N., YANAGIDA Y., MAEDA T., TACHI S.: Detailed shape representation with parallax mapping. In *Proceedings of the ICAT 2001* (2001), pp. 205–208.

[MM05] MCGUIRE M., MCGUIRE M.: Steep parallax mapping. In *Symposium on Interactive 3D Graphics and Games 2005 Poster* (2005).

[OP05] OLIVEIRA M. M., POLICARPO F.: An efficient

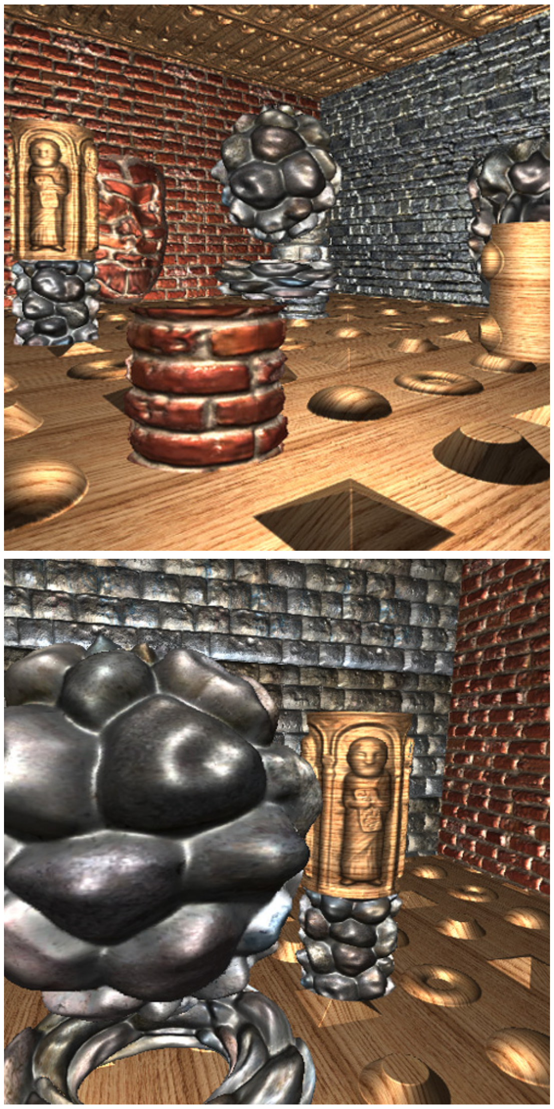


Figure 10: *Renderings of a more complex scene consisting of multiple objects.*

representation for surface details. UFRGS Technical Report RP-351, 2005.

[PHL91] PATTERSON J. W., HIGGAR S. G., LOGIE J. R.: Inverse displacement mapping. *Computer Graphics Forum 10* (1991), 129–139.

[POC05] POLICARPO F., OLIVEIRA M. M., COMBA J. L. D.: Real-time relief mapping on arbitrary polygonal surfaces. In *Proceedings of Symposium on Interactive 3D Graphics and Games 2005* (2005), pp. 359–368.

[RSP06] RISSER E. A., SHAH M. A., PATTANAIK S.: Interval mapping. Technical Report, School of Engineering and Computer Science, University of Central Florida, 2006.

[Tat06] TATARCHUK N.: Dynamic parallax occlusion mapping with approximate soft shadows. In *Proceedings of Symposium on Interactive 3D Graphics and Games 2006* (2006), pp. 63–69.

[Tat08] TATARCHUK N.: Future-proof games with real-time tessellation. In *Conference Session. Advanced Direct3D Tutorial Day, Game Developer's Conference* (2008).

[Wal03] WALSH T.: *Parallax mapping with offset limiting*. Tech. rep., Infiniscape Tech Report, 2003.

[WTL*04] WANG X., TONG X., LIN S., HU S., GUO B., SHUM H.-Y.: Generalized displacement maps. In *proceedings of Eurographics Symposium on Rendering* (2004).

[WWT*03] WANG L., WANG X., TONG X., LIN S., HU S., GUO B., SHUM H.-Y.: View-dependent displacement mapping. *ACM Transactions on Graphics 22*, 3 (2003), 334–339. (Proceedings of SIGGRAPH 2003).