Dynamic Parallax Occlusion Mapping with Approximate Soft Shadows

Natalya Tatarchuk
ATI Research, Inc.

Figure 1. Realistic city scene rendered using parallax occlusion mapping applied to the cobblestone sidewalk in (a) and using the normal mapping technique in (b).

Abstract

This paper presents a per-pixel ray tracing algorithm with dynamic lighting of surfaces in real-time on the GPU. First, we propose a method for increased precision of the critical ray-height field intersection and adaptive height field sampling. We achieve higher quality results than the existing inverse displacement mapping algorithms. Second, soft shadows are computed by estimating light visibility for the displaced surfaces. Third, we describe an adaptive level-of-detail system which uses the information supplied by the graphics hardware during rendering to automatically manage shader complexity. This LOD scheme maintains smooth transitions between the full displacement computation and a simplified representation at a lower level of detail without visual artifacts. The algorithm performs well for animated objects and supports dynamic rendering of height fields for a variety of interesting displacement effects. The presented method is scalable for a range of consumer grade GPU products. It exhibits a low memory footprint and can be easily integrated into existing art pipelines for games and effects rendering.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism: Color, shading, shadowing, and texture; Visible line/surface algorithms

Keywords: image-based rendering, motion-parallax, real-time rendering, displacement mapping, soft shadows, surface details, adaptive level-of-detail system.

I3D 2006, Redwood City, California, 14–17 March 2006.
© 2006 ACM 1-59593-295-X/06/0003 $5.00

1 Introduction

Texture mapping is essential for creating a compelling impression of a realistic scene without paying the full cost of rendering complex geometry. Bump mapping was introduced in the early days of computer graphics in [Blinn 1978] to avoid rendering high polygonal count models. Despite its low computational cost and ease of use, bump mapping fails to account for important visual cues such as shading due to interpenetrations and self-occlusion, nor does it display perspective-correct depth at all angles.

Displacement mapping, introduced by [Cook 1984], addressed the issues above by actually modifying the underlying surface geometry. Ray-tracing based approaches dominated in the offline domain [Pharr and Hanrahan 1996; Heidrich and Seidel 1998]. These methods adapt poorly to current programmable GPUs and are not applicable to the interactive application domain due to high computational costs. Other approaches included software-based image-warping techniques for rendering perspective-correct geometry [Oliveira et al. 2000] and precomputed visibility information [Wang et al. 2003; Wang et al. 2004; Donnelly 2005]. Despite being interactive, these methods suffer from a large memory footprint. The majority of these techniques require high amounts of specialized precomputed data, thus making their integration into existing art pipelines for game development unnecessarily complex. Our proposed method requires a low memory footprint comparable to bump mapping and can be used for dynamically rendered height fields.

Recent inverse displacement mapping approaches take advantage of the parallel nature of novel GPUs' pixel pipelines to render displacement directly on the GPU ([Doggett and Hirche 2000; Kautz and Seidel 2001; Hirche et al. 2004; Brawley and Tatarchuk 2004; Policarpo et al. 2005]. One of the significant disadvantages of these approaches is the lack of correct object silhouettes since these techniques do not modify the actual geometry. Accurate silhouettes can be generated by using view-

dependent displacement data as in [Wang et al. 2003; Wang et al. 2004] or by encoding the surface curvature information with quadric surfaces as in [Oliveira and Policarpo 2005].

We perform per-pixel ray tracing for inverse displacement mapping with an easy-to-implement, efficient algorithm. Our method allows interactive rendering of displaced surfaces with dynamic lighting, soft shadows, self-occlusions, and motion parallax. Previous methods displayed strong aliasing at grazing angles, thus limiting potential applications' view angles, making these approaches impractical in realistic game scenarios. We present a significant improvement in the rendering quality necessary for production level results.

Our contributions include:

- A high precision computation algorithm for critical height field-ray intersection and an adaptive height field sampling scheme, well-designed for a range of consumer GPUs (Section 3.1). This method significantly reduces visual artifacts at oblique angles.
- Estimation of light visibility for displaced surfaces allowing real-time computation of soft shadows due to object self-occlusions (Section 3.2).
- Adaptive level-of-detail control system with smooth transitions (Section 3.3) for controlling shader complexity using per-pixel level-of-detail information.

The contributions presented in our paper are useful for easy integration of inverse displacement mapping into interactive applications such as games. They improve resulting visual quality while taking full advantage of programmable GPU pixel and texture pipelines' efficiency.

2 Related Work

The horizon mapping technique ([Max 1988], [Sloan and Cohen 2000]) allows shadowing bump mapped surfaces using precomputed visibility maps. A simple scheme for simulating motion parallax was presented by [Kaneko et al. 2001]. Although inexpensive, this technique exhibits strong pixel swimming and excessive surface flattening at grazing angles.

A precomputed three-dimensional distance map for a rendered object can be used for surface extrusion along a given view direction ([Donnelly 2005]). The cost of a 3D texture and dependent texture fetches' latency make this algorithm not applicable to most real-time applications.

Mapping relief data in tangent space for per-pixel displacement mapping in real-time was proposed in [Brawley and Tatarchuk 2004; Policarpo et al. 2005; McGuire and McGuire 2005] and further extended in [Oliveira and Policarpo et al. 2005] to support silhouette generation. These methods take excellent advantage of the programmable pixel pipeline efficiency by performing height field-ray intersection in the pixel shader to compute the displacement information. These approaches generate dynamic lighting with self-occlusion, shadows and motion parallax. Using the visibility horizon to compute hard shadows as in [Policarpo et al. 2005; McGuire and McGuire 2005; Oliveira and Policarpo 2005] can result in shadow aliasing artifacts. All of the above approaches exhibit strong aliasing and excessive flattening at steep viewing angles. No explicit level of detail schemes were provided with these approaches, relying on texture filtering capabilities of the GPUs.

Adaptive level of detail control systems are crucial for all computation-intensive algorithms and there have been many contributors in the field of rendering. A level of detail system for bump mapped surfaces using pre-filtered maps was presented in [Fournier 92]. RenderMan® displacement maps were automatically converted to bump maps and BRDF representations in [Becker and Max 1993]. An automatic shader simplification system presented in [Olano et al. 2003] uses controllable parameters to manage system complexity. The resulting level-of-detail shader appearance is adjustable based on distance, size, importance and given hardware limits.

3 Parallax Occlusion Mapping

This section will provide a brief overview of concepts of the parallax occlusion mapping method.

We render highly detailed surfaces using the programmable pixel pipeline of the GPU. The inherent planarity of the tangent space allows us to compute displacement for arbitrary polygonal surfaces. Height field-ray intersections are performed in tangent space. The lighting can be computed in any space using a variety of illumination models. Efficient GPU implementation allows us to compute per-pixel shading, self-occlusion effects, and soft shadows, dynamically scaling the computations.

The effect of motion parallax for a surface can be computed by applying a height map and offsetting each pixel in the height map using the geometric normal and the view vector. We trace a ray through the height field to find the closest visible point on the surface.

The input mesh provides the reference plane for displacing the surface downwards. The height field is normalized for correct ray-height field intersection computation (0 representing the reference polygon surface values and 1 representing the extrusion valleys).

The parallax occlusion mapping algorithm execution can be summarized as follows:

- Compute the tangent-space viewing direction $\hat{V}_{ts}$ and the light direction $\hat{L}_{ts}$ per-vertex, interpolate, and normalize in the pixel shader
- Compute the parallax offset vector P (either per-vertex or per-pixel) to determine the maximum visual offset in texture-space for the current level (as described in [Brawley and Tatarchuk 2004])

In the pixel shader:

- Ray cast the view ray $\hat{V}_{ts}$ along P to compute the height profile–ray intersection point. We sample the height field profile along P to determine the correct displaced point on the extruded surface. This yields the texture coordinate shift offset necessary to arrive at the desired point on the extruded surface (Figure 3). We add this parallax offset amount to the original sample coordinates to yield the shifted texture coordinates t_{off}.
- Estimate the light visibility coefficient v by casting the light direction ray $\hat{L}_{ts}$ and sampling the height profile for occlusions.

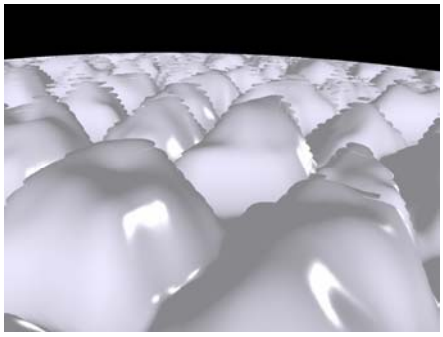

Figure 2a. Relief mapping rendered with both linear and binary search but without depth bias applied. Notice the visual artifacts due to sampling aliasing at grazing angles.

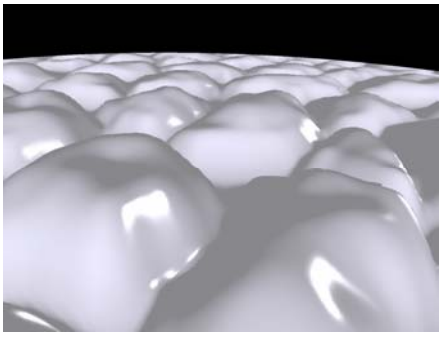

Figure 2b. Relief mapping rendered with both linear and binary search and with depth bias applied. Notice the flattening of surface features towards the horizon.

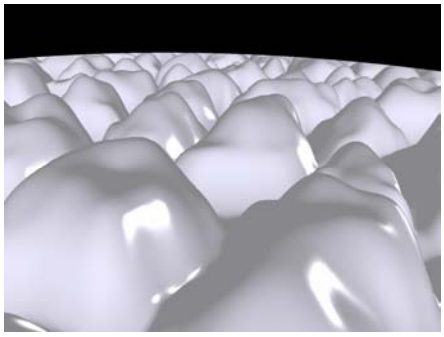

Figure 2c. Parallax occlusion mapping rendered with the high precision height field intersection computation. Notice the lack of aliasing artifacts or feature flattening toward the horizon.

- Shade the pixel using the visibility coefficient v, $\hat{L}_{ts}$ and the pixel's attributes (such as the albedo, color map, normal, etc.), sampled at the texture coordinate offset t_{off}.

Figure 6 illustrates the process above for a given pixel on a polygonal face.

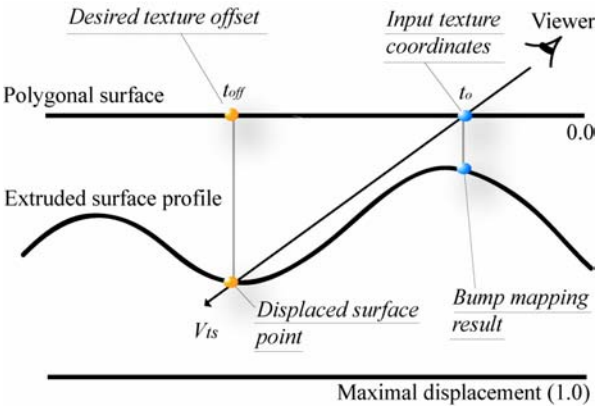

Figure 3. Displacement based on sampled height field and current view direction.

3.1 Height Field – Ray Intersection

Techniques such as [Policarpo et al. 2005; Oliveira and Policarpo 2005] determine the intersection point by a combination of a linear and a binary search routines. These approaches sample the height field as a piecewise constant function. The linear search allows arriving at a point below the extruded surface intersection with the view ray. The following binary search helps finding an approximate height field intersection utilizing bilinear texture filtering to interpolate the intersection point.

The intersection of the surface is approximated with texture filtering, thus only using 8 bit of precision for intersection computation. This results in visible stair-stepping artifacts at steep viewing angles (as seen in Figure 2a). Depth biasing toward the horizon hides these artifacts but introduces excessive feature flattening at oblique angles (Figure 2b).

The binary search from [Policarpo et al. 2005] requires dependent texture fetches for computation. These incur a latency cost which is not offset by any ALU computations in the relief mapping ray-height field intersection routine. Increasing the sampling rate during the binary search increases the latency of each fetch by increasing the dependency depth for each successive fetch.

Using a linear search from [Policarpo et al. 2005] without an associated binary search exacerbates the stair-stepping artifacts even with a high sampling rate (as in Figure 4).

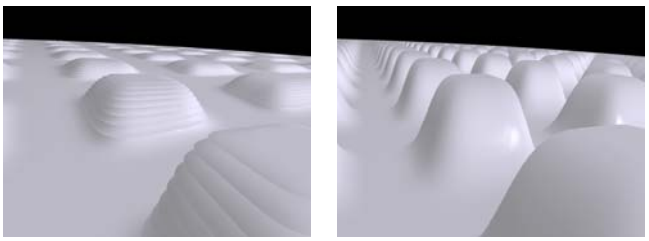

Figure 4. Comparison of height field intersection precision using the linear search only. (a) Relief mapping. (b) Parallax occlusion mapping

The advantage of a linear search for intersection root finding lies in an effective use of texturing hardware with low latency cost as it does not require dependent texture fetches. Simply using a linear search requires higher precision for the root-finding.

We sample the height field using a linear search and approximating the height profile as a piecewise linear curve (as illustrated in Figure 6). This allows us to combine the 8 bit precision due to bilinear texture filtering with the full 32 bit precision for root finding during the line intersection. Figure 2c displays the improved visual results with the lack of aliasing with using our approach.

Since we do not encode feature information into additional look-up tables, the accuracy of our technique corresponds to the sampling interval δ (as well as for [Policarpo et al. 2005]). Both algorithms suffer from some amount of aliasing artifacts if too few samples are used for a relatively high-frequency height field, though the amount will differ between the techniques.

Automatically determining δ by using the texture resolution is impractical. At grazing angles, the parallax amount is quite large

Figure 5a. Hard shadows generated with the relief mapping horizon visibility threshold computation.

Figure 5b. Soft shadows generated with the parallax occlusion mapping penumbra approximation technique.

and thus we must march along a long parallax offset vector in order to arrive at the actual displaced point. In that case, the step size is frequently much larger than a texel, and thus unrelated to the texture resolution. To solve this, we provide both directable and automatic controls.

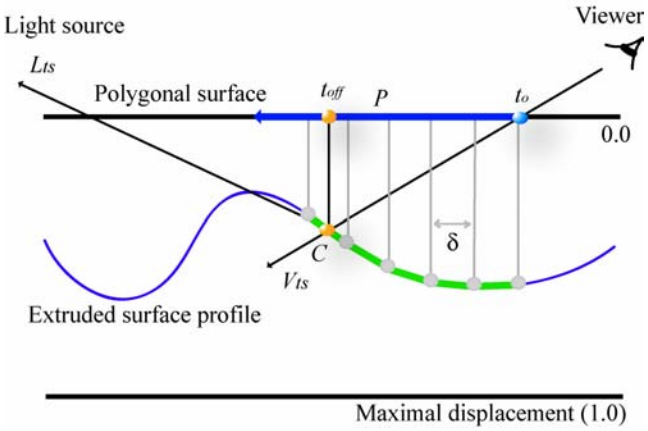

Figure 6. We start at the input texture coordinates t_o and sample the height field profile for each linear segment of the green piecewise linear curve along parallax offset vector *P*. The height field profile-view ray intersection yields parallax-shifted texture coordinate offset t_{off}. δ is the interval step size. Then we perform the visibility tracing. We start at texture offset t_{off} and trace along the light direction vector L_{ts} to determine any occluding features in the height field profile.

The artists can control the sampling size bounds with artist-editable parameters. This is convenient in real game scenarios as it allows control per texture map. If dynamic flow control (DFC) is available, we can automatically adjust the sampling rate during ray tracing. We express the sampling rate as a linear function of the angle between the geometric normal and the view direction ray:

$$n = n_{\min} + \hat{N} \bullet \hat{V}_{ts}(n_{\max} - n_{\min}) \quad (1)$$

where $n_{\min}$ and $n_{\max}$ are the artist-controlled sampling rate bounds, $\hat{N}$ is the interpolated geometric unit normal vector at the current pixel. This assures that we increase the sampling rate along the steep viewing angles. We increase the efficiency of the linear search by using the early out functionality of DFC to stop sampling the height field when a point below the surface is found.

3.2 Soft Shadows

The height map can in fact cast shadows on itself. This is accomplished by substituting the light vector for the view vector when computing the intersection of the height profile to determine the correct displaced texel position during the reverse height mapping step. Once we arrive at the point on the displaced surface (the point *C* in figure 6) we can compute its visibility from the any light source. For that, we cast a ray toward the light source in question and perform horizon visibility queries of the height field profile along the light direction ray $\hat{L}_{ts}$.

If there are intersections of the height field profile with $\hat{L}_{ts}$, then there are occluding features and the point in question will be in shadow. This process allows us to generate shadows due to the object features' self-occlusions and object interpenetration.

If we repeated the process for the height field profile – view direction ray tracing for the visibility query by stopping sampling at the very first intersection, we would arrive at the horizon shadowing value describing whether the displaced pixel is in shadow. Using this value during the lighting computation (as in [Policarpo et al. 2005]) generates hard shadows which can display aliasing artifacts in some scenes (Figure 5a).

With our approach, we sample *n* samples h_1 - h_n along the light direction ray (figure 7). We assume that we are lighting the surface with an area light source, and, similar to [Chan and Durand 2003; and [Wyman and Hansen 2003], we heuristically approximate the size of penumbra for the occluded pixel. Figure 8 demonstrates the penumbra size computation given an area light source, a blocker and a receiver surface.

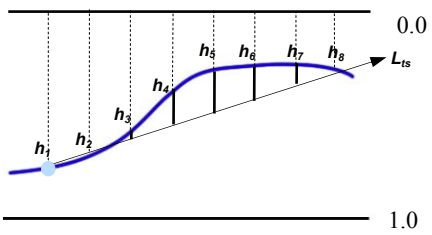

Figure 7. Sampling the height field profile along the light ray direction L_{ts} to obtain height samples $h_1 - h_8$ (n=8)

We can use the height field profile samples h_i along the light direction ray to determine the occlusion coefficient. We sample the height value h_0 at the shifted texture coordinate t_{off}. Starting at this offset ensures that the shadows do not appear floating on top of the surface. The sample h_0 is our reference ("surface") height. We then sample n other samples along the light ray, subtracting h_0 from each of the successive samples h_i. This allows us to compute the blocker-to-receiver ratio as in figure 8. The closer the blocker is to the surface, the smaller the resulting penumbra. We compute the penumbra coefficient (the visibility coefficient v) by scaling the contribution of each sample by the distance of this sample from h_0, and using the maximum value sampled. Additionally we can weigh each visibility sample to simulate the blur kernel for shadow filtering.

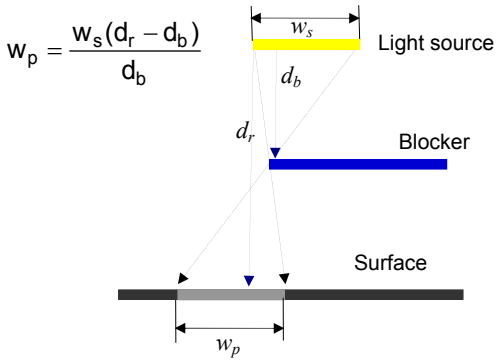

Figure 8. Penumbra size approximation for area light sources, where w_s is the light source width, w_p is the penumbra width, d_r is the receiver depth and d_b is the blocker depth from the light source.

We apply this visibility coefficient v during the lighting computation for generation of smooth soft shadows. This allows us to obtain well-behaved soft shadows without any edge aliasing or filtering artifacts. Figures 5b and 10 demonstrate examples of smooth shadows using our technique. One limitation of our technique is the lack of hard surface contact shadow for extremely high frequency height maps.

Estimating light visibility increases shader complexity. We perform the visibility query only for areas where the dot product between the geometric normal and the light vector is non-negative by utilizing dynamic branching. This allows us to compute soft shadows only for areas which are actually facing the light source.

3.3 Adaptive Level-of-Detail Control System

We compute the current mip level explicitly in the pixel shader and use this information to transition between different levels of detail from the full effect to bump mapping. Simply using mip-mapping for LOD management is ineffective since it does not reduce shader complexity during rendering. Using the full shader for the height field profile intersection with the view ray and the light ray, the visibility and lighting computations is expensive. Although at lower mip levels the fill is reduced, without our level-of-detail system, the full shader will be executed for each pixel regardless of its proximity to the viewer. Instead, with our algorithm, only a simple bump mapping shader is executed for mip levels higher than the specified threshold value.

This in-shader LOD system provides a significant rendering optimization and smooth transitions between the full parallax occlusion mapping and a simplified representation without visual artifacts such as ghosting or popping. Since all calculations are performed per pixel, the method robustly handles extreme close-ups of the object surface thus providing an additional level of detail management.

We compute the mip level directly in the pixel shader (as described in Shreiner et al. [2005]) on SM 3.0 hardware. The lowest level of details is rendered using bump mapping. As we determine that the currently rendered level of detail is close enough to the threshold, we interpolate the parallax occlusion-mapped lit result with the bump-mapped lit result using the fractional part of the current mip level as the interpolation parameter. There is almost no associated visual quality degradation as we move into a lower level of detail and the transition appears quite smooth (figure 12).

We expose the threshold level parameter to the artists in order to provide directability for game level editing. In our examples we used a threshold value of 4. Thus even at steep grazing angles the close-up views of surfaces will maintain perspective correct depth.

4 Results

We have implemented the techniques described in this paper using DirectX 9.0c shader programs on a variety of graphics cards. We use different texture sizes selected based on the desired feature resolution. For figures 1, 12, 13, and 15 we apply 1024x1024 RGBα textures with non-contiguous texture coordinates. For figures 2, 5 and 11 we apply repeated 256x256 RGBα textures, and for figures 4 and 10 repeated 128x128 RGBα textures.

We applied parallax occlusion mapping to a 1,100 polygon soldier character shown in figure 9a. We compared this result to a 1.5 million polygon version of the soldier model used to generate normal maps for the low resolution object (Figure 9b). Note that the two images in figure 9 are from slightly different viewpoints though extracted from a demo sequence with similar viewpoint paths. We apply a 2048x2048 RGBα texture map to the low resolution object. We render the low resolution soldier using DirectX on ATI Radeon X1600 XL at 255 fps. The sampling rate bounds were set to the range of [8, 50] range. The memory requirement for this model was 79K for the vertex buffer, 6K for the index buffer, and 13Mb of texture memory (using 3DC texture compression).

Figure 9a: An 1,100 polygon game soldier character with parallax occlusion mapping

Figure 9b: A 1.5 million polygon soldier character with diffuse lighting

The high resolution soldier model rendered on the same hardware at a rate of 32 fps. The memory requirement for this model was 31Mb for the vertex buffer and 14Mb for the index buffer. Due to memory considerations, vertex transform cost for rendering, animation, and authoring issues, characters matching the high resolution soldier are impractical in current game scenarios. However, using our technique on an extremely low resolution model provided significant frame rate increase with 32Mb memory being saved, at a comparable quality of rendering.

This demonstrates the usefulness of the presented technique for texture-space displacement mapping via parallax occlusion mapping. In order to render the same objects interactively with equal level of detail, the meshes would need an extremely detailed triangle subdivision, which is impractical even with the currently available GPUs.

Our method can be used with a dynamically rendered height field and still produce perspective-correct depth results. In that case, the dynamically updated displacement values can be used to derive the normal vectors at rendering time by convolving the height map with a Sobel operator in the horizontal and vertical direction. The rest of the algorithm does not require any modifications.

5 Conclusion

We have presented a pixel-driven displacement mapping technique for rendering highly detailed surfaces under varying light conditions generating soft shadows due to self-occlusion. We have described an efficient algorithm for computing intersections of the height field profile with rays with high precision. Our method includes estimation of light visibility for generation of soft shadows. An automatic level-of-detail control system manages shader complexity efficiently at run-time, generating smooth LOD transitions without visual artifacts. Our technique takes advantage of the programmable GPU pipeline resulting in highly interactive frame rates coupled with a low memory footprint.

One limitation of our technique is the lack of detailed silhouettes, betraying the low resolution geometry actually rendered with our method. This is an important feature for enhancing the realism of the effect and we are investigating ideas for generating correct silhouettes.

Parallax occlusion mapping can be used effectively to generate an illusion of very detailed geometry exhibiting correct motion parallax as well as producing very convincing self-shadowing effects. We provide a high level of directability to the artists and significantly improved visual quality over the previous approaches. We hope to see more games implementing compelling scenes using this technique.

Acknowledgements

We would like to thank Dan Roeger, Daniel Szecket, Abe Wiley and Eli Turner for their help with the artwork. We also would like to thank Pedro Sander, John Isidoro, Thorsten Scheuermann, and Chris Oat of ATI Research, Inc., Jason L. Mitchell of Valve, Eric Haines, of Autodesk., as well as the anonymous reviews of I3D for their help, suggestions and review of this work.

References

BECKER, B. G., AND MAX, N. L. 1993. Smooth Transitions between Bump Rendering Algorithms. In *ACM Transactions on Graphics (Siggraph 1993 Proceedings)*, ACM Press, pp. 183-190

BLINN, J. F. 1978. "Simulation of Wrinkled Surfaces". In *Proceedings of the 5th annual conference on Computer graphics and interactive techniques,* ACM Press, pp. 286-292.

BRAWLEY, Z., AND TATARCHUK, N. 2004. Parallax Occlusion Mapping: Self-Shadowing, Perspective-Correct Bump Mapping Using Reverse Height Map Tracing. In *ShaderX3: Advanced Rendering with DirectX and OpenGL*, Engel, W., Ed., Charles River Media, pp. 135-154.

CHAN, E., AND DURAND, F. 2003. Rendering fake soft shadows with smoothies, In *Eurographics Symposium on Rendering Proceedings,* ACM Press, pp. 208-218.

COOK, R. L. 1984. Shade Trees, In *Proceedings of the 11th annual conference on Computer graphics and interactive techniques,* ACM Press, pp. 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, pp. 59-66.

DONNELLY, W. 2005. Per-Pixel Displacement Mapping with Distance Functions. In *GPU Gems 2*, M. Pharr, Ed., Addison-Wesley, pp. 123 – 136.

FOURNIER, A. 1992. Filtering Normal Maps and Creating Multiple Surfaces, *Technical Report,* University of British Columbia.

HEIDRICH, W., AND SEIDEL, H.-P. 1998. Ray-tracing Procedural Displacement Shaders, In *Graphics Interface*, pp. 8-16.

HIRCHE, J., EHLERT, A., GUTHE, S., DOGGETT, M. 2004. Hardware Accelerated Per-Pixel Displacement Mapping. In *Graphics Interface*, pp. 153-158.

KANEKO, T., TAKAHEI, T., INAMI, M., KAWAKAMI, N.,

YANAGIDA, Y., MAEDA, T., TACHI, S. 2001. Detailed Shape Representation with Parallax Mapping. In *Proceedings of ICAT 2001*, pp. 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., pp. 61–70.

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

MCGUIRE, M. AND MCGUIRE, M. 2005. Steep Parallax Mapping. *I3D 2005 Poster*.

OLANO, M., KUEHNE, B., SIMMONS, M. 2003. Automatic Shader Level of Detail. In *Siggraph/Eurographics Workshop on Graphics Hardware Proceedings*, ACM Press, pp. 7-14.

OLIVEIRA, M. M, AND POLICARPO, F.. 2005. An Efficient Representation for Surface Details. UFRGS Technical Report RP-351.

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., pp. 359–368.

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., pp. 31–40.

POLICARPO, F., OLIVEIRA, M. M., COMBA, J. 2005. Real-Time Relief Mapping on Arbitrary Polygonal Surfaces. In *ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games Proceedings*, ACM Press, pp. 359-368.

SHREINER, D., WOO, M., NEIDER, J., DAVIS, T.. 2005. *OpenGL® Programming Guide: The Official Guide to Learning OpenGL®,* version 2, Addison-Wesley.

SLOAN, P-P. J., AND COHEN, M. F. 2000. Interactive Horizon Mapping. In *11th Eurographics Workshop on Rendering Proceedings*, ACM Press, pp. 281-286.

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, pp. 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, pp. 227–233.

WYMAN, C., AND HANSEN, C. 2002. Penumbra maps: approximate soft shadows in real-time. In *Eurographics workshop on Rendering 2003*, EUROGRAPHICS, Keller and Jensen, Eds., EUROGRAPHICS, pp. 202-207.

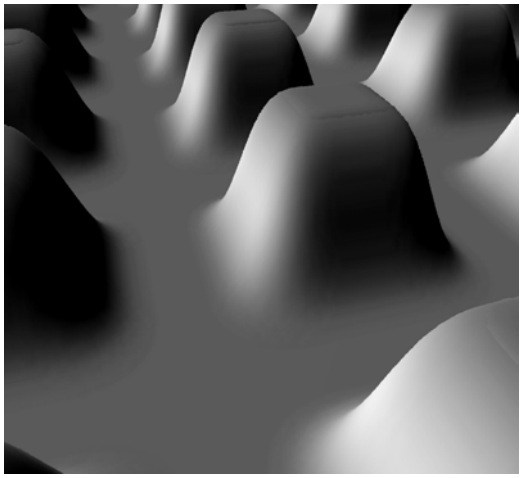

Figure 10. Smooth soft shadows and perspective-correct depth details generated with the parallax occlusion rendering algorithm

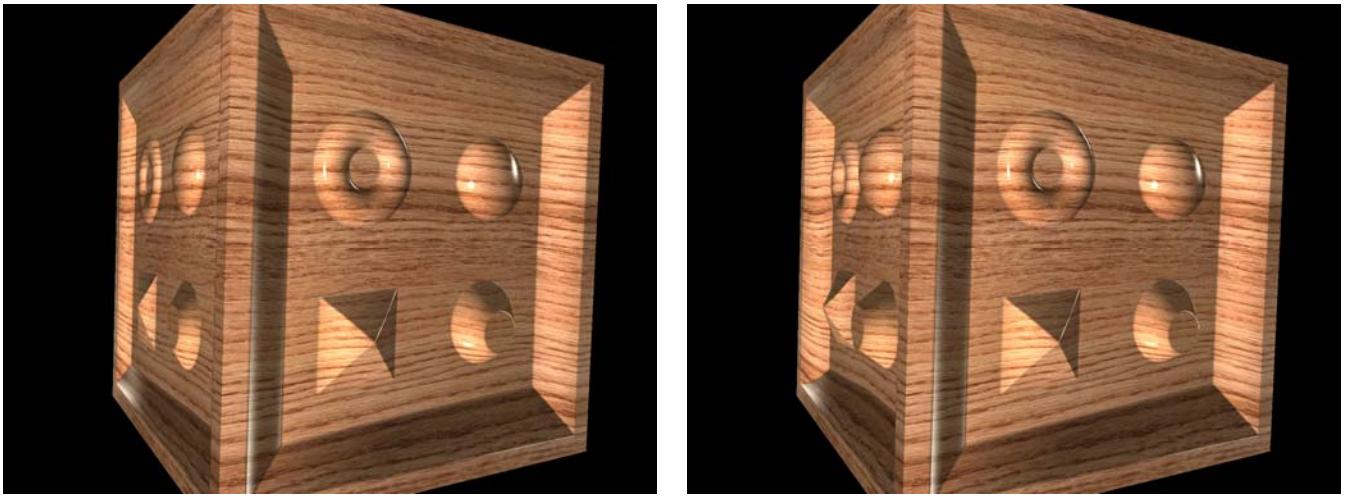

Figure 11. Simple cube model rendered with detailed surface detailed from the same viewpoint. On the left, relief mapping is used to create surface complexity. On the right, parallax occlusion mapping is used to render perspective-correct extruded surfaces. Notice the differences on the left face of the cube as the surface is viewed at a steep angle.

Figure 12. A complex scene with parallax occlusion mapping on the sidewalk and the brick wall. The strength of the blue tint in (b) denotes the decreasing level of detail (deepest blue being bump mapping and no blue displays full computation). Note the lack of visual artifacts and smooth transition due to the level-of-detail transition discernable in (a). The transition region is very small.

Figure 13. A portion of a realistic city environment with the cobblestone sidewalk and the brick wall rendered with parallax occlusion mapping (left) and bump mapping (right). We are able to use shadow mapping on the surfaces and dynamically rendered reflections from objects in the scene.

Figure 14. Displaced text rendering with the sign rendered using parallax occlusion mapping technique