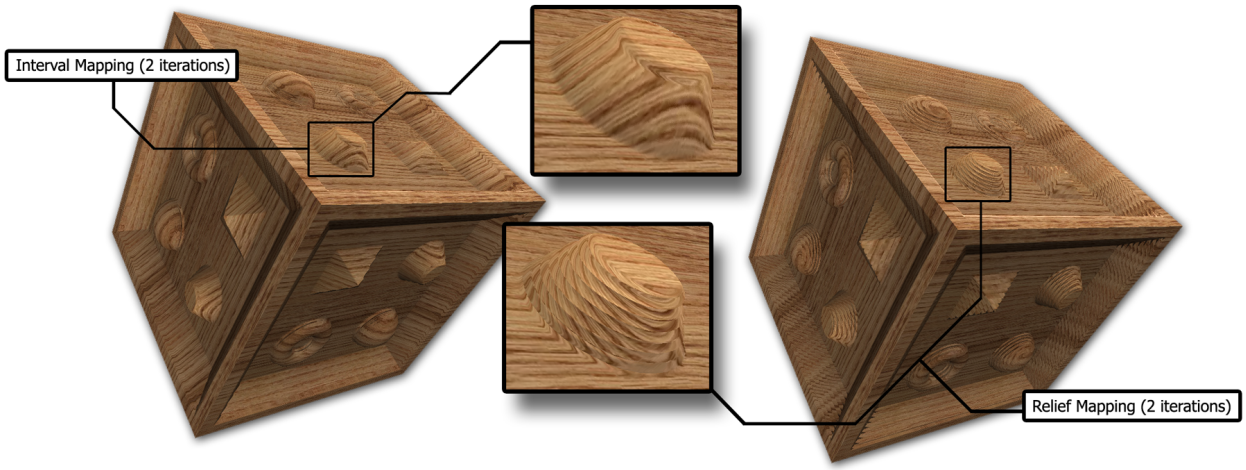
Faster Relief Mapping Using the Secant Method

Eric Risser*
University of Central Florida

Musawir Shah†
University of Central Florida

Sumanta Pattanaik‡
University of Central Florida



Abstract

Relief mapping using the secant method offers an efficient method for adding per pixel height field based displacement to an arbitrary polygonal mesh in real time. The technique utilizes an interval based method in which bounds of the interval are computed in the beginning and are refined at every iteration until the intersection point is reached. The search space defined by the interval reduces and converges to the intersection point rapidly and outperforms currently popular binary search based method (relief mapping) used for performing this task. We compute the bounds using simple ray segment intersection method. We demonstrate the algorithm and show empirical and explicit evidence of the speedup.

Keywords: per-pixel displacement mapping, image based rendering

1 Introduction

Most of the interesting objects, natural as well as synthetic, have complex surface structures that require huge sets of polygons for their accurate modeling of the complex structural feature variations such as those on the trunk of a tree. This limitation immensely hinders the process of rendering such objects, and is virtually impossible for applications requiring rendering at interactive frame-rates. To remedy the problem of dealing with unmanageable amounts of geometry image-based techniques have been explored. The main advantage of employing an image-based solution is that it decouples the rendering complexity from the geometric complexity of the surface being rendered. One of the earliest and most commonly used image-based techniques is bump mapping, introduced by Blinn [Blinn 1978]. Bump mapping lights each pixel of a polygon as though it had a 3D structure represented by its texture. Bump mapping does not, however, capture all of the visual effects such as shadowing and occlusion. A more accurate technique called displacement mapping was introduced by Cook[Cook 1984] .This method utilizes a height field texture which is used to displace the actual geometry of the surface. Modern graphics hardware allows texture lookups in the vertex processing program and therefore can be used to access the height field to perform per-vertex displacement. However, the current height field rendering research focus is on per-pixel displacement. This enables strikingly accurate rendering of complex surfaces with fine-scale details.

We propose a method for achieving better performance over current per-pixel displacement mapping methods by performing multiple iterations of the secant method in order to find the roots of an arbitrary function.

In the next section, we present a survey of previous work in height field rendering.

2 Background

Parallax mapping [Kaneko et al. 2001] is a method for approximating the parallax seen on uneven surfaces. Using the view ray transformed to tangent space, parallax mapping samples a height texture to find the approximate texture coordinate offset that will give the illusion that a three dimensional structure is being rendered. Parallax mapping is a crude approximation of displacement mapping. It can not simulate occlusion, self shadowing or silhouettes. Since it requires only one additional texture read, it is a fast and therefore a relevant approximation for use in video games.

McGuire et al. improved on the algorithm by proposing steep paral-

*e-mail: erisser@cs.ucf.edu
†e-mail: mali@cs.ucf.edu
‡e-mail: sumant@cs.ucf.edu

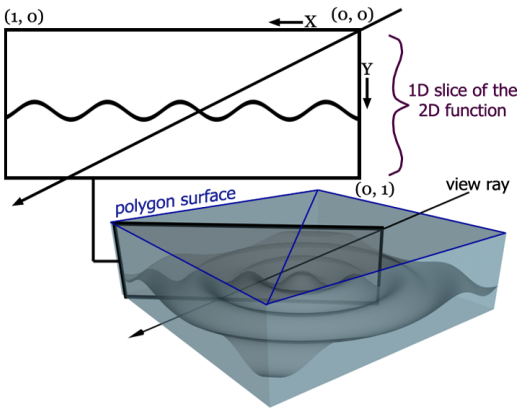


Figure 1: 3D height-field and view ray geometry used in Section 3 to illustrate the algorithm.

lax mapping [McGuire 2005]. Steep parralax mapping uses a linear search that marches along the view ray checking for intersection with the surface at regular intervals until it finds a point where it is found to have pierced the surface. The texture coordinates at this point are then used to get an approximation to the actual point of intersection. The need for many texture reads makes steep parallax mapping far slower than parallax mapping. However, the extra testing results in a better approximation that supports self shadowing, occlusion and a visual quality rivaling that of the shells technique without any extra preprocessing or memory requirements other than those of parallax mapping.

Per-pixel displacement mapping with distance functions [Donnelly 2005] uses a three dimensional source texture which stores for each voxel the distance to the closest point on the surface to be viewed. The technique is based on sphere tracing [Hart 1996], a technique developed to accelerate ray tracing of implicit surfaces.

Relief mapping [Policarpo et al. 2005] begins with a linear search in the same fashion as steep parallax mapping. However, once it finds the point at which the ray has pierced the surface, it then performs a binary search along the view ray to home in on the exact point of intersection.

Dynamic Parallax Occlusion Mapping [Tatarchuk. 2006] relies on an initial linear search followed by a single iteration of the secant method to fit a series of discrete linear functions to the curve. This is a highly efficient method for rendering a good approximation of the surface. Whereas it doesn't offer the same level of accuracy as a technique using a true root finding method, it also doesn't suffer from the performance degradation caused by branch dependent texture lookups.

We make the claim that by extending the use of the secant method as used in Dynamic Parallax Occlusion Mapping to multiple iterations, we can achieve the same visual quality as Relief Mapping with fewer dependent texture lookups than is required with a binary search.

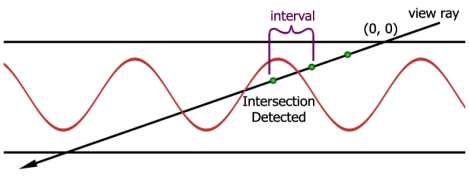


Figure 2: Linear search along the view ray.

3 The Algorithm

Like similar methods, our algorithm conceptually takes a 2D slice out of a 3D height-field as can be seen in figure 1. The algorithm comprises of two main steps:

1. Find initial intersection interval by performing a uniform linear search along the view ray. This involves sampling the view ray at equally spaced steps and looking up the height field until the point on the ray is below the corresponding height value. This step is identical to the first step of the relief mapping technique and is illustrated in Figure 2.
2. Iteratively reduce the intersection interval (shown in Figure 2) to a specified threshold by adjusting the upper and lower bound of the interval with a ray-line segment intersection test. This step is explained in detail in the remaining paragraphs of this section.

We refer to the two bounding points of the interval as upper bound and lower bound, the upper being the point on the view ray before the intersection and the lower being the point after intersection. Since we are dealing with 2D slices of 3D space, the bounding points are represented by 2D co-ordinates (x,y).

We sample the height field surface at our upper and lower points of the interval to find the two surface points and join the resulting two points to define a line (see Figure 3 in which the blue line joins the two points). The point at which this line intersects the view ray is defined as either the new upper or new lower bound depending on whether the height of the surface at that point is greater or less than our intersection point on the view ray (Figure 3 shows the updated upper bound). Our view ray is a line passing through the origin and hence may be represented as

$$A_{view}x + B_{view}y = 0 \tag{1}$$

and the line passing through the two surface points may be represented as

$$A_{line}x + B_{line}y + C_{line} = 0 \tag{2}$$

where

$$A_{line} = y_{UpperBound} - y_{LowerBound} \tag{3}$$
$$B_{line} = x_{LowerBound} - x_{UpperBound} \tag{4}$$
$$C_{line} = -B_{line}y_{UpperBound} - A_{line}x_{UpperBound} \tag{5}$$

Solving for the point of intersection we get

$$x = C_{line}\frac{B_{view}}{A_{view}B_{line} - A_{line}B_{view}} \tag{6}$$
$$y = -C_{line}\frac{A_{view}}{A_{view}B_{line} - A_{line}B_{view}} \tag{7}$$

The intersection point updates one of the boundary points of the interval. By iterating through this method, we quickly converge to the point of intersection.

In the next sections we show the results from our methods and compare its performance with that of the relief method.

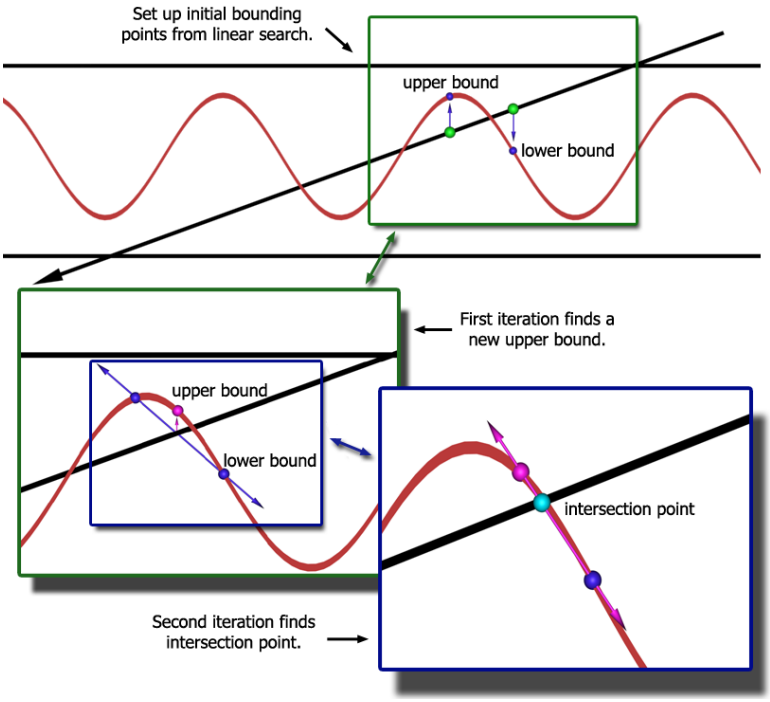


Figure 3: Search along the view ray; for this specific example an intersection is found in two iterations. The number of iterations for any given case depends on the angle of the incoming view ray and the function it intersects with.

4 Results and Analysis

We implemented our technique and relief mapping on the GPU using DirectX and HLSL. Both use the same linear search function to find the initial interval, and have the same basic structure for the secant/binary search. The only difference between the two is how they refine the bound.

For relief mapping computation we assigned the midpoint between upper and lower bound i.e (upper + lower)/2 as one of the bounds. For the secant method we set the intersection point as one of the bounds. So the additional cost of the secant method is an extra division, multiplication and three subtractions per iteration. The secant method recovers this loss by converging faster to the intersection and thus requiring fewer iterations and fewer slow dependent texture reads. The illustration on the first page demonstrates the faster convergence of our method using a side-by-side comparison of the images generated after two iterations of our method and relief mapping method. Both the methods used five linear searches before carrying out the iterations. In addition to this, we demonstrate the performance increase by counting the number of iterations required by each pixel to reach convergence in each of the methods. We show the comparison results in Figure 4.

5 Discussion

We have presented a fast converging algorithm based on the secant method for per-pixel height field rendering. By utilizing the available height field and viewing information, we have shown an efficient way of adaptively reducing the search space for finding the intersection point. Though, empirical evidence shows that both methods require about the same iterations in the worst case, on average the secant based algorithm converges in two to three iterations whereas it takes five to eight binary search steps.

References

BLINN, J. F. 1978. Simulation of wrinkled surfaces. In *SIGGRAPH '78: Proceedings of the 5th annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 286–292.

COOK, R. L. 1984. Shade trees. In *SIGGRAPH '84: Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 223–231.

HART, J. C. 1996. Sphere tracing: A geometric method for the antialiased ray tracing of implicit surfaces. *The Visual Computer 12*, 10, 527–545.

HIRCHE, J., EHLERT, A., GUTHE, S., AND DOGGETT, M. 2004. Hardware accelerated per-pixel displacement mapping. In *GI '04: Proceedings of the 2004 conference on Graphics interface*, Canadian Human-Computer Communications Society, School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, 153–158.

KANEKO, T., TAKAHEI, T., INAMI, M., KAWAKAMI, N., YANAGIDA, Y., AND MAEDA, T. 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 *GRIN'01: No description on Graphics interface 2001*, Canadian Information Processing Society, Toronto, Ont., Canada, Canada, 61–70.

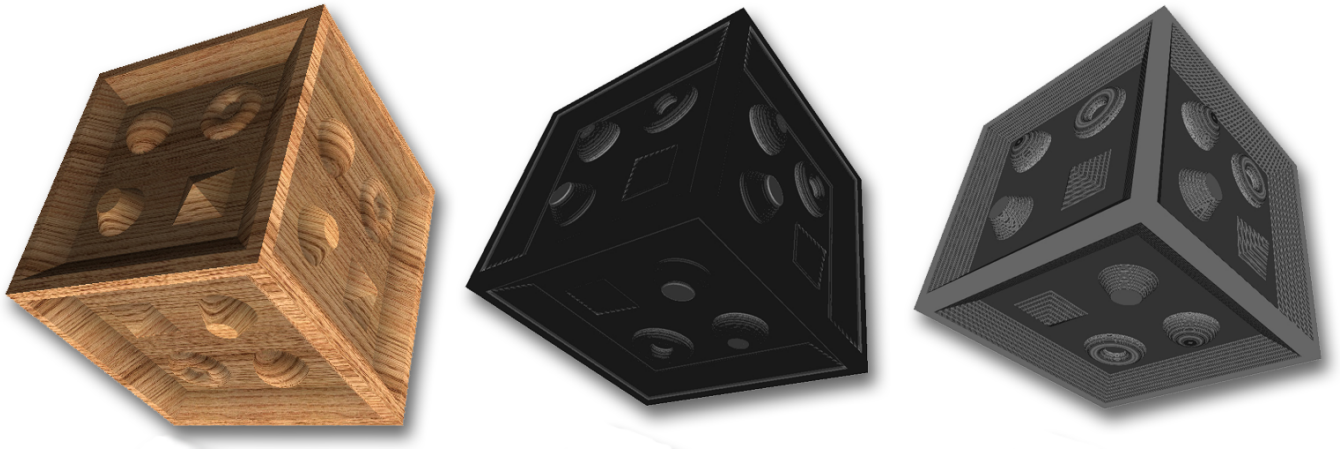


Figure 4: Empirical results. Images in the first column are rendered normally using our method. Images in the second and third column show the number of pixel iterations required for convergence. The grey levels in these two columns are coded black for 1 iteration, up to white for a maximum of 10 iterations. The second column shows the results from the secant method while the third column represents binary search. By comparing the average color value of the second column with the third, it is clear to see that the secant method converges quicker.

KOLB, A., AND REZK-SALAMA, C. 2005. Efficient empty space skipping for per-pixel displacement mapping. In *Proc. Vision, Modeling and Visualization*.

MAX, N. 1998. Horizon mapping: shadows for bump-mapped surfaces. In *The Visual Computer 4, 2*, 109–117.

MCGUIRE, M. 2005. Steep parallax mapping. In *I3D 2005 Poster*.

OLIVEIRA, M. M., BISHOP, G., AND MCALLISTER, D. 2000. Relief texture mapping. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 359–368.

PATTERSON, J. W., HOGGAR, S. G., AND LOGIE, J. R. 1991. Inverse displacement mapping. *Comput. Graph. Forum 10*, 2, 129–139.

POLICARPO, F., OLIVEIRA, M. M., AND COMBA, J. L. D. 2005. Real-time relief mapping on arbitrary polygonal surfaces. In *SI3D '05: Proceedings of the 2005 symposium on Interactive 3D graphics and games*, ACM Press, New York, NY, USA, 155–162.

PRESS, W., FLANNERY, B., TEUKOLSKY, S., AND VETTERLING, W. 2002. Root finding and non-linear sets of equations. In *Numerical Recipes in C*, 354–360.

SCHAUFLER, G., AND PRIGLINGER, M. 1999. Horizon mapping: shadows for bump-mapped surfaces. In *Efficient displacement mapping by image warping*, 175–186.

SLOAN, P., AND COHEN, M., 2000. Interactive horizon mapping.

TATARCHUK. 2006. Dynamic parallax occlusion mapping with approximate soft shadows. In *SI3D '06: Proceedings of the 2006 symposium on Interactive 3D graphics and games*, Redwood City, California, 63–69.

WALSH. 2003. Parallax mapping with offset limiting. In *Infiniscape Tech Report*.

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. 2003. Generalized displacement maps. In *Eurographics Symposium on Rendering*, 227–233.


```
////////////////////////////////////////////////////////////////////////
// secant method                                                      //
////////////////////////////////////////////////////////////////////////
// this portion of code requires a linear search to first be          //
// performed, with the two points right before and right after        //
// collision stored as the upper and lower variables                  //
////////////////////////////////////////////////////////////////////////

pixel_color.a = 1;
float int_depth = 0;

for(int i = 0; (i < 10) && (abs(pixel_color.a - int_depth) > .01); i++)
{
       float line_slope = (upper_h - lower_h)/(upper_d - lower_d);
       float line_inter = upper_h - line_slope*upper_d;

       float dem = view_slope - line_slope;
       float inter_pt = line_inter / dem;

       tex_coords_offset2D = inter_pt * float2(view_vec.y, -view_vec.x);
       int_depth = view_slope*inter_pt;

       pixel_color=tex2D(heightSampler,(tex_coords_offset2D)+input.tex_coords);

       if(pixel_color.a < int_depth) //new upper bound
       {
              upper_h = pixel_color.a;
              upper_d = inter_pt;
              best_depth = upper_h;
       }
       else //new lower bound
       {
              lower_h = pixel_color.a;
              lower_d = inter_pt;
              best_depth = lower_h;

        }

}
// compute our final texture offset
tex_coords_offset2D = ((1.0f/view_slope)*best_depth)*float2(view_vec.y,-view_vec.x);

// store pixel color
pixel_color = tex2D(textureSampler, tex_coords_offset2D+input.tex_coords);
```

Secant method shader code