

Displacement Mapping with Ray-casting in Hardware

Keith Yerex
University of Alberta

Martin Jagersand
University of Alberta

1 Introduction

Traditionally, displacement maps have been rendered with micropolygons [Cook et al. 1987]. In both raytracers and real-time systems, these high polygon counts lead to memory/bandwidth inefficiency, and high geometric transformation costs, which limit performance. Recently, displacement maps have also been directly rendered in raytracing, using iterative root-finding methods [Heidrich and Seidel 1998]. Here, we propose a similar ray intersection approach for real-time rendering. However, an iterative solution is infeasible in graphics hardware due to the limitations on *texture indirections*. A texture indirection occurs when the results of one texture access affect the coordinates of subsequent accesses, such as in each step of an iterative solution. These are currently limited to 4 on Radeon 9700/9800 where in contrast, a possible total of 32 texture accesses are allowed per fragment. This has led us to the hybrid sampling/iterative approach described here

2 Algorithm

Our algorithm renders displacement mapped planes, directly from a displacement map given as a texture image. Displacements are assumed to always penetrate into the surface (rather than bumping out from it) and are bounded by some maximum.

The reference surface is rendered as a polygon in OpenGL, processing each pixel as follows. Along the ray from the viewpoint through the current pixel, we sample the displacement map at N discrete points where the ray is between the reference surface and the maximum displacement depth. Displacement values d_i are compared to the ray heights h_i at each point to determine which side of the surface the ray is on at each point.

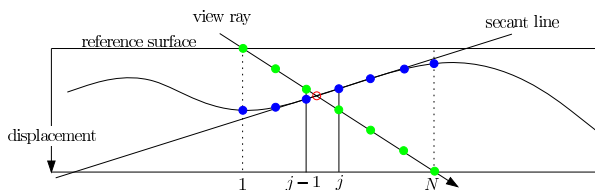


Figure 1: Algorithm: d_i are shown in blue, h_i in green, and the intersection point in red

We take the interval between the sample nearest to the viewer, j , where $h_j > d_j$ and its neighboring sample, $j - 1$, to contain the intersection of the ray with the surface (see figure 1). The surface is then approximated by the secant line from d_j to d_{j-1} . Finally, we calculate texture coordinates, which are used to index texture and normal maps, as the intersection of this secant line with the view ray.

In comparison, using the point j for texture coordinates, rather than performing the final step, is equivalent to rendering displacement maps with volumetric slicing, as in [Deitrich 2000], but with lower depth complexity, and less geometry, since one polygon is rendered rather than N .

3 Implementation and Results

We have implemented this method on a mid-consumer grade Radeon 9700 using an OpenGL fragment program with 83 instructions. $N = 15$ samples are used. Some displacement maps with large or high frequency displacements cause under-sampling artifacts at grazing angles (see video for examples). These artifacts are reduced by multi-pass rendering where each pass processes only a portion of the ray. Single pass performance is approximately 130 fps for an entirely full 640x480 window.

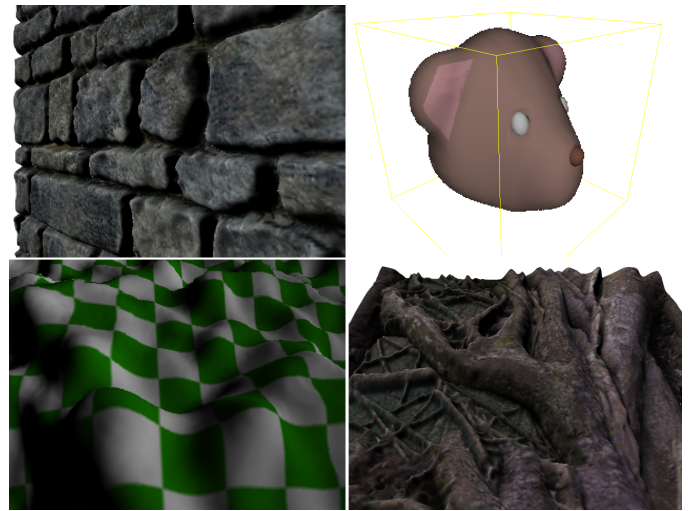


Figure 2: Upper right object is made from 6 displacement mapped planes rendered in 4 passes, others are rendered in a single pass with a single plane. Special thanks to Terry Welsh for sample displacement maps and textures.

4 Conclusions and Future work

We have shown that ray-casting can be used to render planar displacement maps efficiently in hardware. Generalizing this technique to support curved surfaces is work in progress.

References

- COOK, R. L., CARPENTER, L., AND CATMULL, E. 1987. The reyes image rendering architecture. In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, ACM Press, 95–102.
- DEITRICH, S. 2000. Elevation maps. Tech. rep., NVIDIA Corporation.
- HEIDRICH, W., AND SEIDEL, H.-P. 1998. Ray-tracing procedural displacement shaders. In *Graphics Interface*, 8–16.