Virtual Displacement Mapping Techniques: Fall'24 CSCI-580 Term Project Report

1st Justin Shi
Thomas Lord Department of Computer Science
University of Southern California
Los Angeles, California, USA
jmshi@usc.edu

2nd Matthew Tran
Thomas Lord Department of Computer Science
University of Southern California
Los Angeles, California, USA
mgtran@usc.edu

3rd Qihua "Josh" Sun
Thomas Lord Department of Computer Science
University of Southern California
Los Angeles, California, USA
qihuasun@usc.edu

4th Chaeho Shin
Thomas Lord Department of Computer Science
University of Southern California
Los Angeles, California, USA
chaehosh@usc.edu

***Abstract*—Virtual Displacement Mapping, more commonly referred to as Parallax Mapping, is a set of techniques where parallax effects and the illusion of depth are achieved through texture coordinate displacement based on view angle and surface elevation without requiring geometry of higher detail. First introduced in 2000, similar techniques were rapidly introduced in the first half of 2000-2010, but generally fell out of favor by the mid-2010s, being mostly replaced by true Displacement Mapping through Tessellation. In this report, we will implement, examine, and compare the results of Parallax Mapping and its variants in HLSL using the Unity Engine. We will go over the algorithms and principles behind each variation, and discuss their strengths and failures cases. Finally, we will briefly examine why Parallax Mapping and its derivatives fell out of favor.**

***Index Terms*—virtual displacement mapping, image-based rendering, texture mapping, relief mapping, surface details, real-time rendering, 3D graphics**

I. Introduction

The original basic Texture Mapping technique is a powerful method for improving detail without requiring additional polygons when rendering 3D scenes. The model vertices contain texture coordinates (u, v) that are interpolated across polygons alongside other vertex parameters and used to sample textures to determine colors. These colors are then used in lighting calculations to determine the final color of visible points. Thus, we can create small-scale polygon interior detail complexity instead of increasing geometry complexity.

However, because vertex parameters are linearly interpolated, despite the arbitrary variation of texture color across a polygon, lighting calculations still result in smoothly varying intensities. This, combined with the view-independent nature of texture coordinate interpolation, leads to textured polygons appearing flat as we move around them (Fig. 1(a)).

In this report, we detail the implementation of five different Virtual Displacement Mapping methods, as well as an implementation of true Vertex Displacement Mapping, through HLSL in the Unity Engine. We compare and contrast the six methods, and discuss their strengths and failure cases as well as performance.

TABLE I: Performance Measurement Environment Specifications

CPU	Intel i9-13900K
GPU	NVIDIA GeForce RTX 4090
RAM	DDR5 128GB
Screen Resolution	3840 x 2160
OS	Windows 11 Pro 23H2

II. Implementation Details

A. Implementation Environment

All different displacement mapping techniques were implemented in Unity 6 (6000.0.24f1). Unity allowed us to streamline the process for setting up scenes, materials/textures, and controls for manipulating the camera, so that we could focus on shader implementation. Unity also provided sophisticated profiling tools for assessing performance, as well as lighting/shadowing frameworks that allowed us to demonstrate the benefits of vertex displacement (as self-occlusion/self-shadowing with virtual displacement techniques requires additional manual shader implementation and does not affect global illumination).

Performance was measured using the default Unity Profiler, more specifically using the length of the `DrawOpaqueObjects` call, which draws all opaque objects in the scene excluding the skybox. Empty scenes with just a sphere and a plane were set up with each shader type applied to both objects, and then the range of draw call execution lengths per frame across a time frame of a few seconds was recorded.

The specifications of the local machine in which the performance was measured is detailed in TABLE I.

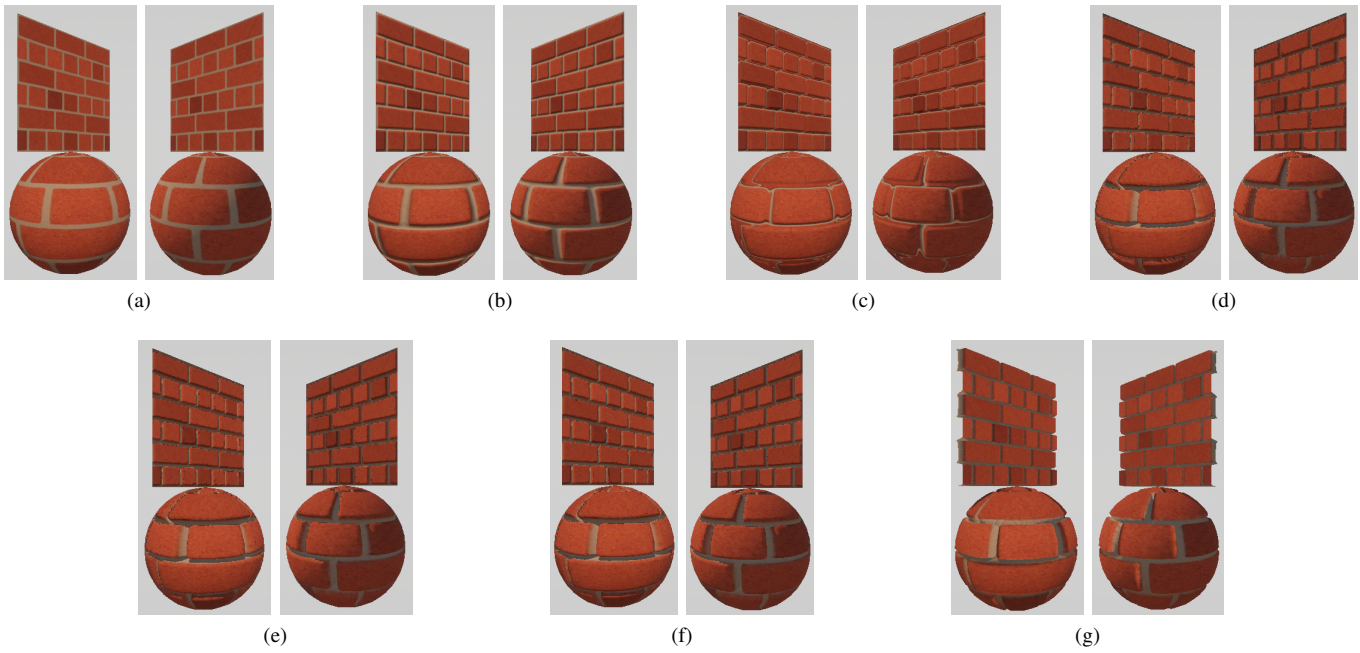

Fig. 1: Mapping Techniques Comparison.
(a) Texture Mapping (b) Normal Mapping (c) Parallax Mapping (d) Steep Parallax Mapping
(e) Parallax Occlusion Mapping (f) Relief Mapping (g) Vertex Displacement

B. Normal Mapping

1) Methods: Normal Mapping is a common improvement that aims to address the issue of smoothly varying intensities. It is itself an implementation of bump mapping, a general technique in which the interpolated normal for a visible point on a polygon is perturbed in some manner before being used in lighting calculations. This results in the polygon appearing to have small displacements across itself, becoming bumpy (as the name suggests) instead of smooth and flat, despite still introducing no additional geometry. In the case of Normal Mapping, this is done by sampling a second texture that encodes normals in tangent space as colors; these normals are then transformed to the space in which lighting calculations are performed and used in place of interpolated normals from vertices.

In our implementation, we calculate the bitangent from the cross product of the provided normal and tangent for vertices, then interpolate these three vectors to approximate the axes (and thereby construct the needed transformation matrix) of tangent space for visible points.

2) Results Discussion: In comparison to basic Texture Mapping, note that, despite the apparent increased detail, the silhouette of geometry remains the same, and the surface details are independent from the view direction (Fig. 1(b)).

C. Parallax Mapping

1) Methods: Parallax Mapping is a technique similar to Normal Mapping, but based on different principles. It significantly improves the detail of a textured surface and gives it a sense of depth. In Parallax Mapping, height or depth values are introduced with height or depth map textures, and the texture coordinates are altered higher or lower to match the view direction and height map. The result will reach a similar approach to Normal Mapping, where the surface may have displaced geometry to reflect the details.

In our implementation, we scale the fragment-to-view direction vector by the depth of the fragment on the depth map to interpolate the displacements. We also applied Parallax Mapping in conjunction with Normal Mapping to keep the displacement consistent with the lighting.

2) Results Discussion: Compared to Normal Mapping, we can see that the bricks have much clearer depth details. Note the changing surface texture as the perspective changes, resulting in a parallax effect (Fig. 1(c)). However, there are two issues that exist, as can be seen in Fig. 2(a):

1. The gap between the bricks is sometimes overshadowed by one of the bricks based on the view direction.

2. The depth detail is not obvious when looking from a sharp angle (ex. looking at the left side of the plane in Fig. 2(a)).

Both issues are caused by inaccurate approximation when depth changes rapidly.

D. Steep Parallax Mapping

1) Methods: Steep Parallax Mapping is an extension on top of Parallax Mapping which focuses on providing more accurate results, even with steep height changes, by taking a number of samples. This is done by traversing depth layers through the view ray until the layer's depth value is less than

(a) (b) (c) (d) (e)

Fig. 2: Mapping Techniques at Oblique Angles.
(a) Parallax Mapping (b) Steep Parallax Mapping (c) Parallax Occlusion Mapping
(d) Relief Mapping (e) Vertex Displacement

the height map's value. In this way, we can greatly improve the accuracy of the displacement interpolation through a more accurate estimation of the intersection between the view ray and the virtual surface.

In our implementation, we followed McGuire's approach [3] to set up layers. We also dynamically change the number of layers based on the view direction from between 8-32. For example, when looking straight onto the surface, we will reduce the number of layers as there isn't as much texture displacement at this angle.

We have also implemented the self-shadowing logic as a hard shadow in the fragment shader by marching along each of the light rays in the scene, starting from the estimated intersection point on the virtual surface.

2) Results Discussion: Compared to Parallax Mapping, we can see more accurate details at gaps and faces when viewing from sharp angles (Fig. 1(d)), as well as self-occlusion and self-shadowing. The texture now takes more time to render due to the introduction of multiple depth layers (TABLE II), but this is still within a reasonable amount of time as we change the number of depth layers dynamically. The self-shadowing effect is also clear and correct.

However, when looking from a close distance, we can see jagged artifacts of multiple layers as a result of this implementation (Fig. 2(b)).

E. Parallax Occlusion Mapping

1) Methods: Parallax Occlusion Mapping is based on the same principles as Steep Parallax Mapping, but instead of taking the texture coordinates of the first depth layer after a collision, it will be linearly interpolated based on the depth layer after and before the collision of the height map. This improves the accuracy of the interpolation of vertex displacement.

In our implementation, based on Tatarchuk and Buhler's studies [4], we kept most of the implementations from Steep Parallax Mapping. On top of that, we assume the surfaces are planar at the intersection of layers and thus perform the 'secant' method, which is a linear search based on the two depth layers before and after the intersection.

2) Results Discussion: The result produces an image similar to the one with Steep Parallax Mapping, but successfully removes the jagged layers thanks to the linear interpolation (Fig. 1(e)). Furthermore, the added step for Parallax Occlusion Mapping adds minimum complexity to the implementation, and therefore the running time remains almost the same as Steep Parallax Mapping (TABLE II).

However, due to depth being calculated based on the assumption of planar surfaces between intervals, the mapping between the layers is not exactly accurate and still has minor aliasing issues (Fig. 2(c)).

F. Relief Mapping

1) Methods: Relief Mapping is based on the same principles as Steep Parallax Mapping, but the texture and normal coordinates are refined after the first depth layer collision by a binary search of some preset depth. This prevents aliasing issues that are found in Steep Parallax Mapping at steep angles when the number of depth layers is not sufficient.

This was implemented in the HLSL shader by adding the binary search component after the layer search component in the fragment shader. We followed the algorithm outlined by Policarpo et al. [5]. We used a fixed binary depth of 6, which ensures that the search resolution is at least equivalent to a pure linear depth search of 512 layers. This ensures that the correct texture and normal coordinates are found with a very high fidelity.

2) Results Discussion: As can be seen in Fig. 1(f) and Fig. 2(d), Relief mapping produces good quality texturing with almost little to no aliasing artifacts. However this comes at a cost, as the performance requires approximately 15% more render time per frame on the GPU compared to more efficient methods like Steep Parallax Mapping or Parallax Occlusion Mapping (TABLE II).

G. Vertex Displacement with Tessellation

1) Methods: Vertex Displacement is a technique that uses a height map to displace mesh vertices, creating more complex geometry. Each vertex is displaced using the depth map texel, a scalar, and the vertex normal.

Tessellation is a technique that subdivides the triangles in a mesh into smaller triangles. This is often used in conjunction with vertex displacement, as it provides more vertices to achieve greater geometric complexity.

Our naive and straightforward implementation was based off a description by Szirmay-Kalos and Umenhoffer [6] and utilizes the Tesselation functions provided by HLSL. The first stage we implemented was the Hull Stage. This consists of the Hull Function and the Patch Constant Function, which generate tessellation factors and use them to subdivide an input patch into smaller patches. Next was the Domain Stage, which contains the logic for the newly created vertices and passes them to the fragment shader. Since the Domain Stage handles the logic for the newly created vertices, this is where the vertex displacement was implemented. We read the texel value from the depth map and multiplied it by an adjustable scalar and the vertex normal to calculate the displacement vector. This displacement vector was then used to adjust the position of each vertex. The modified geometry can be seen where the edge of the object is no longer flat (Fig. 1(g)) (Fig. 2(e)).

2) Results Discussion: The result produces more complex geometry based on a depth map. This creates an image without the view angle or aliasing issues of the virtual displacement techniques. By changing the actual object geometry, we automatically get correct true silhouettes on the object itself. Furthermore there is no need to implement custom shadows in the fragment stage, as we can simply utilize the framework's implementation of shadows and other global lighting effects to achieve details such as receiving or casting shadows, or different shadow types such as soft-shadowing.

Performance is also excellent as seen in TABLE II, primarily because in modern GPUs the performance bottleneck comes from increased number of shader instructions rather than the size of the geometry. Even at Tessellation Factors of 20, which should produce on average 400 more triangles per triangle, performance remains better than Steep Parallax Mapping.

TABLE II: Displacement Mapping Methods Performance

Displacement Method	GPU Render Loop time range (ms)
Texture Mapping	0.005-0.006
Normal Mapping	0.006-0.007
Parallax Mapping	0.009-0.011
Steep Parallax Mapping	0.059-0.066
Parallax Occlusion Mapping	0.062-0.067
Relief Mapping	0.067-0.075
Vertex Displacement (20)	0.049-0.051

III. Summary and Discussion

Overall, we can see that the various Virtual Displacement Mapping techniques all have their advantages and disadvantages. As you add more and more complexity and shader instructions to refine the search for the intersection between the view ray and the virtual surface and obtain a more accurate texture map, you get a steadily increasing performance cost. In fact, on modern GPU systems, adding more triangles via tessellation and doing naive, true vertex displacement is faster than typical virtual displacement mapping techniques for reasonable geometries.

As such, while Virtual Displacement Mapping techniques were popular in the 2000s when the main performance bottleneck was geometry (as referenced in [6]), it has fallen out of vogue in favor of true displacement mapping in contemporary use. True displacement mapping simply offers higher quality results with more features, such as complex shadows and global effects, correct silhouettes, etc. at better performance. While we knew this in the abstract, actually being able to confirm this for ourselves was a surprise for us while we were carrying out the project assignment.

Acknowledgments and Deliverables

We would like to thank Professor Ulrich Neumann for his series of great lectures on the basics of 3D Rendering and providing us with the inspiration for this project.

We would also like to thank our lab TA John Manard for his helpful guidance and support throughout the semester on both the class slack channel and during the lab sessions.

Our implementation source can be found at: https://github.com/usc-jmshi/csci_580_project.

A link to a downloadable demo of our implementation can be found at: https://drive.google.com/file/d/1b7FI0V4S2heT-GA76rLjNI3OavCxQJKH/view?usp=sharing.

References

[1] M. M. Oliveira, G. Bishop, and D. McAllister, "Relief texture mapping," in *Proceedings of the 27th annual conference on Computer graphics and interactive techniques (SIGGRAPH '00)*, 2000, pp. 359–368, [Online]. doi:10.1145/344779.344947

[2] T. Kaneko et al., "Detailed Shape Representation with Parallax Mapping," in *Proceedings of ICAT 2001*, 2001, pp. 205–208.

[3] M. McGuire and M. McGuire, "Steep Parallax Mapping," *Brown University*, Apr. 1, 2005. Accessed: Oct. 27, 2024. [Online]. Available: https://casual-effects.com/research/McGuire2005Parallax/index.html

[4] N. Tatarchuk and J. Buhler, "Practical dynamic parallax occlusion mapping," in *ACM SIGGRAPH 2005 Sketches*, 2005, p. 106, [Online]. doi: 10.1145/1187112.1187240.

[5] F. Policarpo, M. M. Oliveira, and J. L. D. Comba, 'Real-time relief mapping on arbitrary polygonal surfaces', in *Proceedings of the 2005 Symposium on Interactive 3D Graphics and Games*, Washington, District of Columbia, 2005, pp. 155–162, [Online]. doi: 10.1145/1053427.1053453

[6] L. Szirmay-Kalos and T. Umenhoffer, 'Displacement Mapping on the GPU - State of the Art', *Computer Graphics Forum*, vol. 27, no. 1, 2008, pp. 1567-1592, [Online]. doi: 10.1111/j.1467-8659.2007.01108.x