Parallax Bumpmapping

Whitepaper

Filename	:	Parallax Bumpmapping.Whitepaper
Version	:	PowerVR SDK REL_16.1@3976567a External Issue
Issue Date	:	17 Mar 2016
Author	:	Imagination Technologies Limited

Contents

1. Introduction 3

2. Technique 4

2.1. Overview 4
2.2. Required Information 4
2.3. How It Works 4

3. Optimizations 6

3.1. Initial Prototyping 6
3.2. On Platform 6
3.2.1. Optimizing Texture Accesses 7
3.2.2. Optimizing Precision Settings 7

4. Benefits and Limitations 8

4.1. Advantages 8
4.2. Disadvantages 8

5. Conclusion 9

6. References 10

7. Contact Details 11

List of Figures

Figure 1. Observed texel vs. real texel 4
Figure 2. New UV coordinate 5
Figure 3. Texture crawling with no scaling (left) and with scaling (right) 8

1. Introduction

The parallax effect is exhibited when areas of a surface appear to move relative to one another as the position of the viewer changes. This is visible on any non-flat surface, a stone wall, or a speaker stack for example. It is easy to simulate parallax within a scene using geometry as the mathematics used to display a three dimensional world in a computer, generating parallax as a by-product of the display process. This is, however, computationally expensive to add to every rough or lined surface due to the additional geometry required.

The method described here provides a means to approximate the parallax effect using height data and the view position to adjust the UV coordinates of an object; all without the addition of further polygons. In addition, optimizations will be discussed for a parallax bump mapping shader specifically optimized for PowerVR platforms.

2. Technique

2.1. Overview

On a real surface the variations in height produce a parallax effect; with non-parallax bump mapping there are no height variations on a polygon. As can be seen in Figure 1 the observed texel T_a corresponds to point A when using a flat polygon. If the object were real, the eye would see texel T_b as this corresponds with the point of intersection of the 'real surface' with the view vector. The goal of parallax bump mapping is to adjust the UV coordinates used at point A so they match those of point B. If this is done per pixel over an entire surface then a parallax effect should emerge.

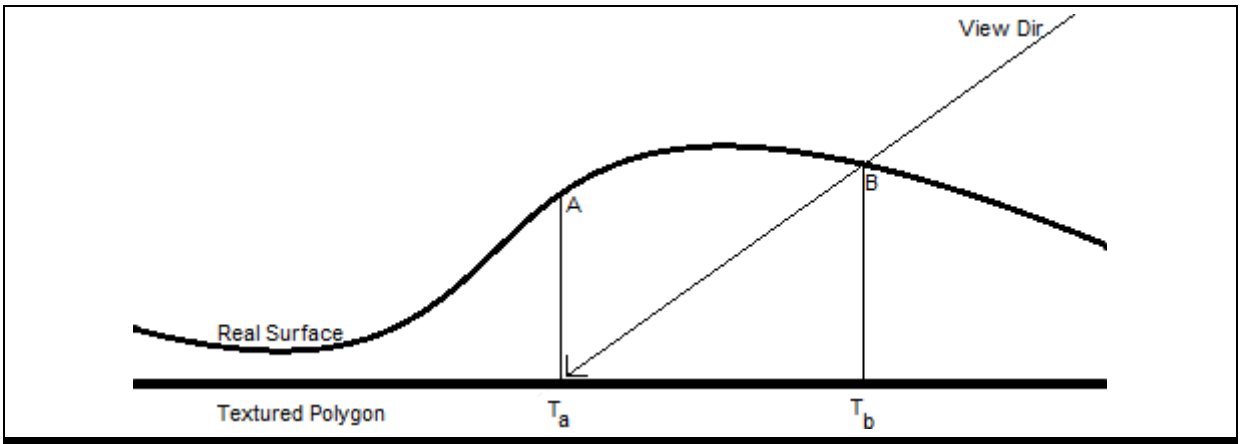

Figure 1. Observed texel vs. real texel

2.2. Required Information

Reasonably advanced graphics systems often describe objects with a diffuse colour, a normal map, and occasionally a specular map. In addition to this information, the parallax effect requires the use of a height map. This height map stores information pertaining to the height of a surface at a given point on it. Finally, the view direction is required to ensure that the texture coordinate is moved in the correct direction.

As the calculations are being used to determine a modified UV coordinate they will be done in tangent space. As such, there will be a requirement for a world to tangent space matrix. This matrix will be used to transform the view direction into tangent space on a per-vertex basis (per-pixel information is determined by interpolating this data across a polygon).

2.3. How It Works

In standard texture mapping each vertex has a UV within the texture. When each polygon is rendered this UV is interpolated across the surface to determine the segment of the texture that is to be sampled. In the case of parallax, instead of sampling the texture the coordinates are used to sample a height map. With this information, a ray is traced from point A until it intersects the view vector. The coordinate of this new point is then used to sample all of the remaining maps (texture, normal, etc.).

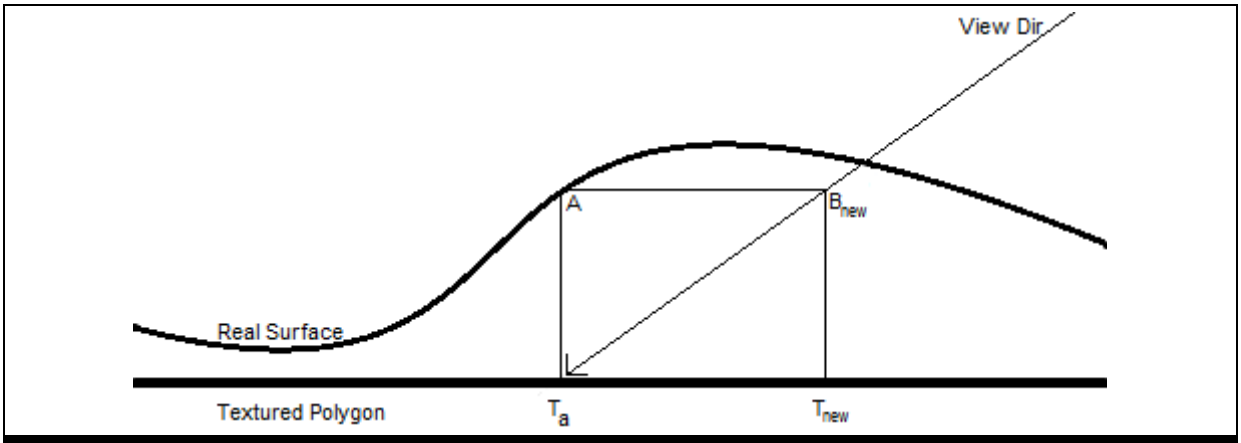

Figure 2. New UV coordinate

As can be seen in Figure 2, the result is only an approximation. Techniques do exist for more accurate simulations, but in all cases they are far more computationally expensive. An example of one of these techniques would be Parallax Occlusion Bump Mapping with Reverse Height Tracing (Brawley & Tatarchuk, 2004).

The offset $\overrightarrow{AB_{new}}$ is calculated by first converting the view direction (V) into tangent space and normalizing it. The normalized view direction contains x and y components that lie on the plane of the surface and a z component which is perpendicular to the surface. Next the height value (in the range 0.0 to 1.0) is scaled and biased and then multiplied by the x and y components of the view direction. The resulting vector is finally added to the original UV coordinate to give the offset UV. The original equation for determining the texture offset is:

$$T_{new} = T_a + \frac{V_{(x,y)} * h_{scaled}}{V_z}$$

Using this equation V_z begins to cause problems at grazing angles where it begins to become close to zero. The offset values become unusably high, breaking the parallax effect. To solve this V_z is dropped from the equation entirely, limiting the offset to no more than the value of h_{scaled}. At grazing angles this improves the image quality with little difference to the overall quality in general. This gives a final equation of:

$$T_{new} = T_a + (V_{(x,y)} * h_{scaled})$$

3. Optimizations

Parallax bump mapping while considerably less expensive, computationally than other available options, is still quite expensive for mobile platforms. As such, much care was taken to optimize the effect.

3.1. Initial Prototyping

Initial development was done in PVRShaman with all precisions set to `highp`. The counts from PVRShaman's editor window were used to suggest areas for improvement. The primary areas identified were matrix multiplication within the vertex shader and normalisation within the fragment shader. As the first identified optimization, a faster means of creating an eye to tangent space conversion matrix was devised. This optimization used the inverse transpose of the world view matrix to create a tangent, binormal and normal. These were then used to manually create the eye to tangent space matrix.

```
highp vec3 n = normalize(worldViewIT * vertNormal);
highp vec3 t = normalize(worldViewIT * vertTangent);
highp vec3 b = cross(n,t);

// Create the matrix from the above
mat3 mEyeToTangent = mat3( t.x, b.x, n.x, t.y, b.y, n.y, t.z, b.z, n.z);
```

This moved some of the grunt work to the CPU (specifically calculating `worldViewIT` which would be passed as a uniform). On investigation little more could be gained from optimizing the vertex shader without changing precisions.

The next identified optimization was in the various normalizes within the fragment shader. The fragment shader for parallax bump mapping usually begins by normalizing the light and view directions. It was possible to move the normalization of the view direction into the vertex shader with a very minor drop in visual quality. While this did increase the cycle count of the vertex shader it reduced the cycle count of the fragment shader by a commensurate amount. This proved to be a small gain as the fragment shader is run far more often than the vertex shader. This was attempted with the light direction, but the drop in visual quality was significant and deemed unacceptable.

3.2. On Platform

Once the initial prototyping had been completed the effect was tested on a device using PVRTune to profile the applications performance. Initial results showed the effect was bandwidth limited. Parallax bump mapping requires at least three texture reads: a height map, a normal map and a texture map. Of these, only the height map is an independent read, the other two being dependent on the initial read of an offset calculation. As such, only the height map read could be prescheduled. To amortise this cost the textures were all compressed using PVRTC 2 bpp (bits per pixel). This provided a considerable gain in performance, approximately 20 fps (frames per second).

With the textures now heavily compressed PVRTune was once again used to profile the application. This second run identified that the effect had become fragment shader limited so this became the area of focus for the next pass of optimizations. The first fragment shader optimization was to move the multiplication by `fParallaxScale` into the vertex shader. As with the normalization of the view direction this provides a gain as there are more fragments being processed than vertices. This saved a single cycle per fragment. While this might seem a small gain as compared to the texture compression, these small gains can add up over many profiling and optimisation passes to a significant improvement. It is important to note that at each stage further profiling was performed, without repeated profiling, time would have been wasted optimising areas of the pipeline that were not performance limiting, or on fixing bottlenecks that had moved.

3.2.1. Optimizing Texture Accesses

Any operation that depends on values from texture accesses cannot execute until that data is returned from System Memory. The original way the shader was written meant that there were two such points within the shader that forced it to wait for texture data during execution.

- Mathematics.
- Texture Read.
- Wait.
- Mathematics.
- Texture Read.
- Wait.

By rearranging the shader code, and moving the two texture reads to be back-to-back, the hardware is able to request both memory fetches together and only wait once. This means the shader now looks as follows:

- Mathematics.
- Texture Read.
- Texture Read.
- Wait.

This parallelises the two lookups so that the wait times overlap. Waiting is inevitable for any memory fetches like texture reads, more so for those which are looked up depending on shader code (also known as dependent reads). Making dependent reads non-dependent by removing any in-shader calculation of their coordinates can allow them to be pre-fetched, removing the in-shader wait, but that was not possible in this case. However, the parallel execution model in PowerVR hardware ensures that whilst individual shader instances are waiting for memory reads, the hardware can run other shader instances that are not waiting (see “PowerVR Hardware Architecture Guide for Developers” for more information).

3.2.2. Optimizing Precision Settings

The last and most difficult area to optimize was the precision settings. Initially, everything used high precision (`highp`). However, there are additional costs to `highp` and, where possible, low precision (`lowp`) should be used. After some testing, it was determined that all calculations done within the fragment shader could be `lowp` with little decrease in visual appeal. Likewise, the view direction could be `lowp` as could the texture coordinate (in both cases, because all values were in the range of 0-1 only precision was lost, not actual integer information). Light direction was also moved to `lowp`; initially this damaged the overall effect, however, moving the `lowp` to `highp` conversion into the vertex shader provided a gain of 4 cycles in the fragment shader at a cost of 2 cycles in the vertex shader.

The key to producing gains from the use of `lowp` is to be aware of the cost of converting from one precision to another. If a calculation includes a `highp` value then there will be no performance gains from using a `lowp` value in the same calculation.

With the above performance enhancements completed, the shader ran at ~80fps on the non-vsync limited test device.

4. Benefits and Limitations

4.1. Advantages

While this parallax bump mapping effect is still computationally expensive on mobile devices, the alternatives, e.g., greater model complexity, displacement mapping, etc., are far more expensive. Parallax bump mapping provides an adequate solution that is artistically pleasing using simple geometry at a reduced cost to all the alternatives.

4.2. Disadvantages

There are several disadvantages to parallax bump mapping: the first is the lack of a complex silhouette. While parallax bump mapping allows for the simulation of false geometry within a polygon, the silhouette of that polygon does not change. This can be a problem on surfaces with steep complex bump maps when viewed at extreme angles. Effectively, the lack of a complex silhouette breaks the visual cues that allow the parallax effect to function.

Another disadvantage is a lack of self-shadowing; areas of the height map that are higher than others do not cast shadows over the rest of the height map. Techniques exist that can simulate this via reverse height tracing. However, at the time of writing it was decided that the computational cost of these techniques was too expensive for widespread use in the current generation of mobile graphics hardware.

Finally, the parallax technique makes an assumption that, as can be seen in Figure 2 does not always hold true, that point A and point B_{new} both lie on the surface of the 'real surface'. The effect of this assumption is that artifacting occurs at points of steep height change. This becomes even more pronounced at points with vertical or near vertical height changes, where the texture can be seen to crawl up the sides of the raised section. In the case of this demo the effect has been minimized through the use of a scaling vector that restricts how much the UV can be perturbed. Figure 3 depicts texture crawling with and without scaling.

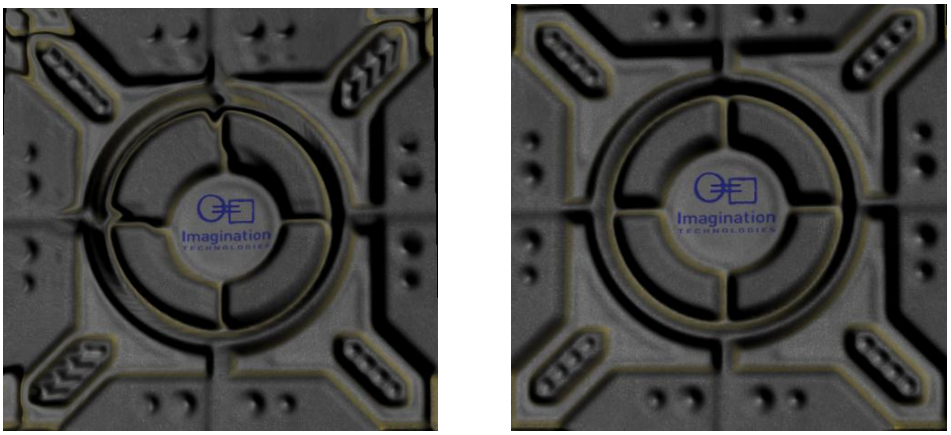

Figure 3. Texture crawling with no scaling (left) and with scaling (right)

5. Conclusion

Relative to other techniques that perform similar functions, the parallax bump mapping technique is an inexpensive means to better simulate complex surfaces than bump mapping alone. The use of bump mapping helps to offset some of the issues with parallax mapping by itself, specifically the lack of lighting and shadowing, though not self-shadowing. Parallax mapping provides additional visual cues that bump mapping alone does not, that help improve the overall effect.

With correct optimizations the effect can be used widely on PowerVR platforms reaching acceptable performance levels even when running full screen on lower end devices. Future PowerVR platforms should be more than capable of running full scene parallax bump mapping, possibly including some more advanced techniques for complex silhouettes and self-shadowing that are impractical presently.

6. References

Brawley, Z., & Tatarchuk, N. (2004). “Parallax Occlusion Mapping: Self-Shadowing, Perspective-Correct Bump Mapping Using Reverse Height Map Tracing”. In: *Shader X3.*

7. Contact Details

For further support, visit our forum:
http://forum.imgtec.com

Or file a ticket in our support system:
https://pvrsupport.imgtec.com

To learn more about our PowerVR Graphics SDK and Insider programme, please visit:
http://www.powervrinsider.com

For general enquiries, please visit our website:
http://imgtec.com/corporate/contactus.asp