Advanced Shaders Report

Thomas Rhodes – P16203335

Overview

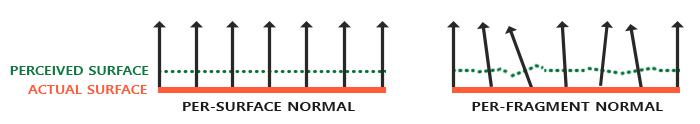
For my advanced shaders project, I utilised the tutorials present on the LearnOpenGL website to create an example of Normal/Parallax mapping. As such, the code itself remains largely the same as that present on the website with only minor alterations. However, I have heavily coded all the functions and such to help demonstrate my understanding of how the program functions. In this report, I will delve deeper into the shader techniques themselves and how they function.

Techniques

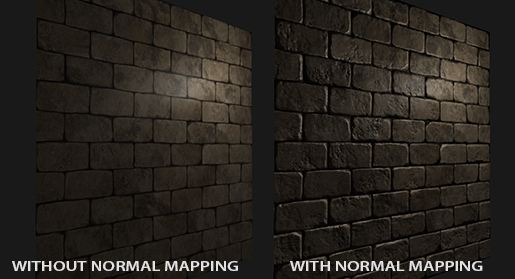
Normal mapping

Normal mapping is a technique used to help develop the realism of a model or texture through the replication of depth and detail. This help to overcome the issue of textures and models being inherently smooth objects, which are largely unfound in reality. This is especially important when used in conjunction with textures as it would be more clear when lighting isn’t reflecting correctly off of the bumps and lumps demonstrated by the texture.

If we use the example of a wall, it’s clearer to see the issue. The issue arises in this case due to the object only having a single normal vector. By only having a single normal vector, the light algorithm sees only one direction to reflect the light and doesn’t have a way to account for the grooves in the wall. This can be overcame by utilising a per-fragment normal rather than a per-surface normal. That way, the normal can deviate to accommodate the changes in the surface more accurately. This is achieved in a “normal map” texture, using RGB colours to indicate which direction in the XYZ that the normal should be facing.



This “tricks” the lighting algorithm into believing the surface is made up of tiny planes which are perpendicular to the surface normal. This process it called normal or bump mapping. The change in detail and realism is clear when compared.

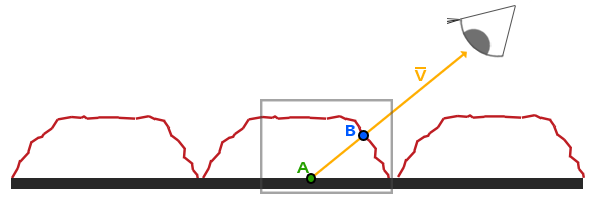


This all achieved with reality low overhead as we only change the normal vectors per fragment, so there is no need to change any lighting equation. Instead, we pass the per-fragment normal instead of the interpolate surface normal to the lighting algorithm which then give the surface detail

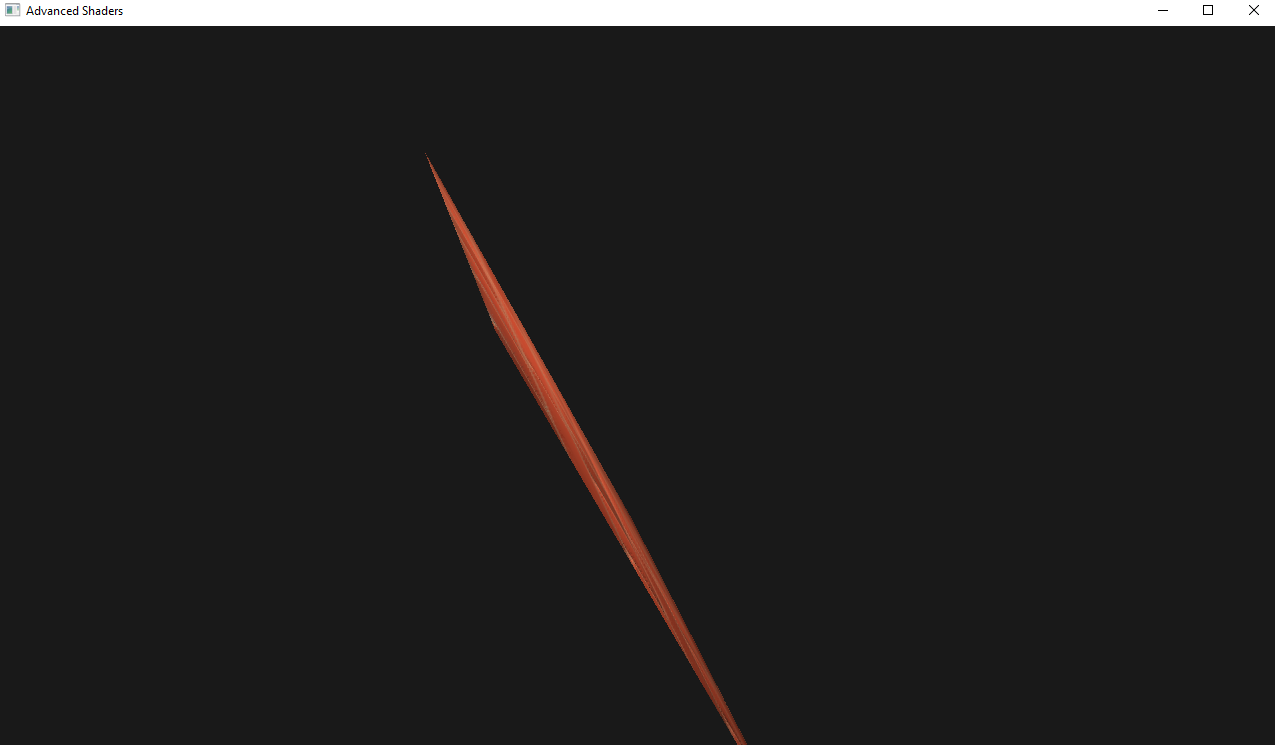
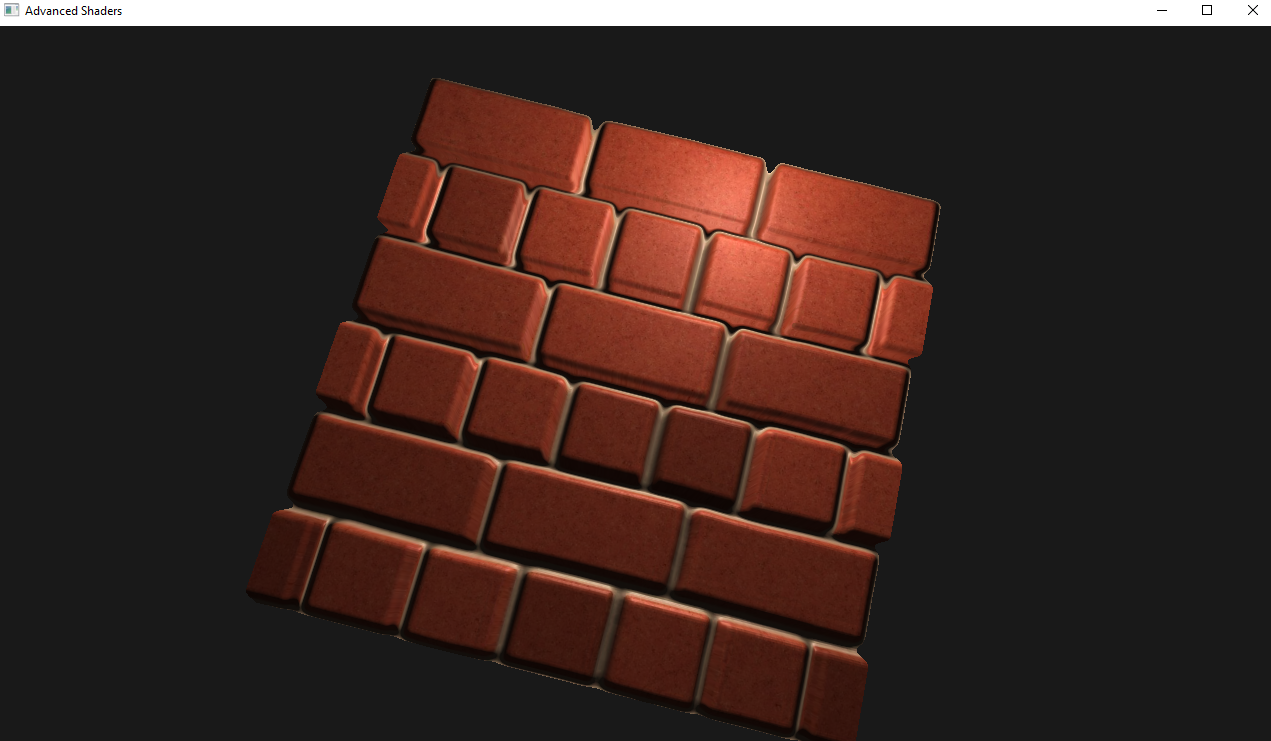
Parallax mapping

Parallax mapping is similar to normal mapping in that is boosts a surface’s detail and provides a sense of depth. While these are both “illusions” of depth, parallax mapping is far better at conveying the depth. Parallax mapping data is stored in a height map which uses black and white to display areas of depth, with black representing areas that are indented in the object.

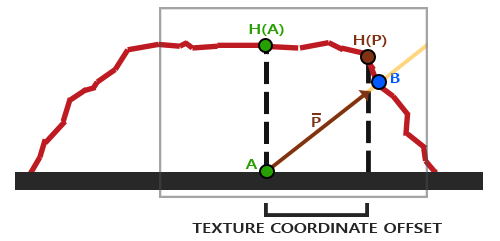
A benefit of parallax mapping compared to other displacement mapping techniques is that it doesn’t require magnitudes of vertices to function. It functions by altering texture coordinates so they are perceived as higher or lower, rather than having a displaced object actually be moved. It’s achieved by offsetting texture coordinates to represent the depth.



This example helps to explain the issue and how it’s overcame by parallax mapping. What we want is for the viewing vector to hit at the B point, which is would if we had actual displacement of the surface. However, it hits at A normally at there is no “real” raised surface on the object. Parallax mapping aims to offset the texture coordinates at A so that we can get the texture coordinates we would expect at point B. From then, we use all texture coordinates from point B for all subsequent texture samples. As such, from the viewing vector, it looks as if we are seeing the raised surface despite no actual displacement. This is especially clear in the project when the object has rotated to its side and you can see the wall is flat.



The complicated aspect of parallax mapping is how the texture coordinates of A are shifted to B. It’s done by scaling the fragment to view direction vector by the height of that fragment at A. This created the scaled vector P.



We then takes this vector P and take its vector coordinates that align with the place as the texture coordinate offset. This works because vector P is using a height value from the height map so the higher the height, the greater the displacement.

However, this can be crude if the surface height changes frequently and can end up looking unrealistic.