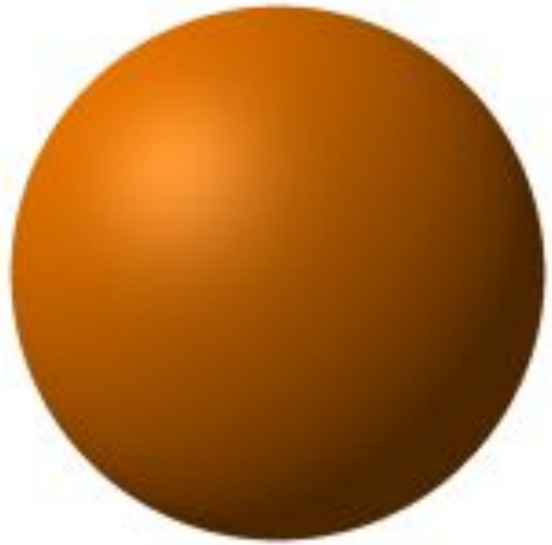
Tangent Space & Bump/Parallax/Horizon Mapping

Tino Abate & Gigi Bachtel

Overview

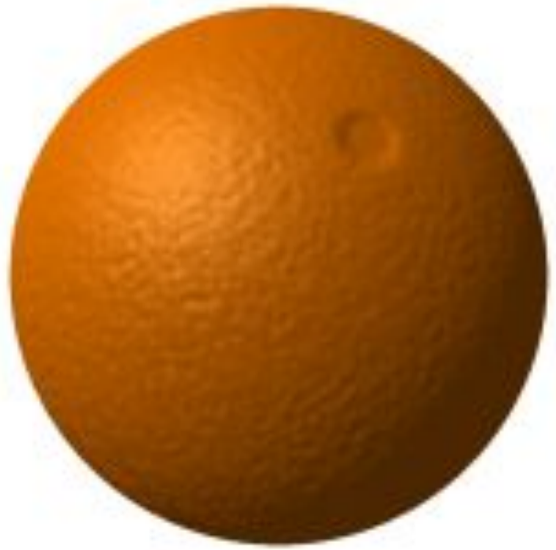
Overview

In class, we've already covered how to make a nicely lit sphere with the Blinn-Phong lighting model. But this sphere is round and smooth.



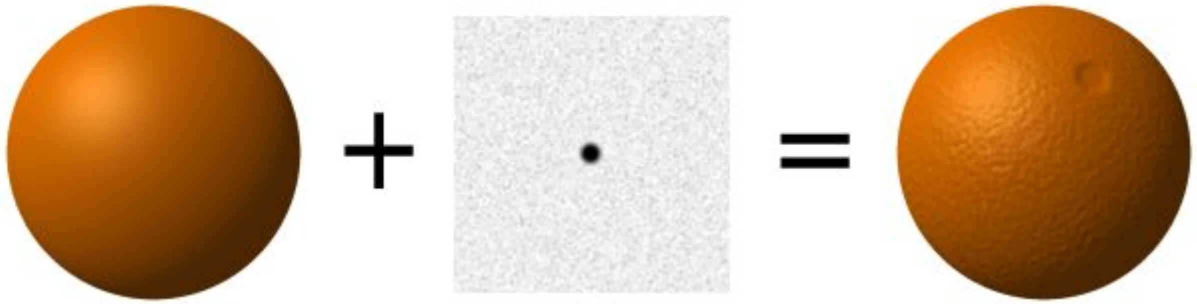
Overview

What if we could make it look like this?



Overview

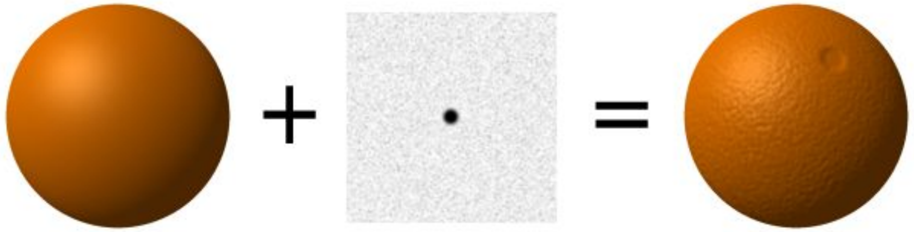
Let’s use a bump map!



Overview

Bump mapping (also called normal mapping) is the process of sampling a 2D texture to modify the normal vectors of surfaces for lighting calculations on a per-fragment basis.

Bump mapping is, as its name implies, used to create the appearance of bumpy or uneven surfaces without additional geometry.



Other Techniques

Parallax mapping and Horizon mapping also contribute to the initial goal of realistically simulating rough surfaces without additional geometry by accounting for weakness in the Bump mapping method and adding additional calculations.

Parallax mapping fixes some of the issues with regular bump mapping by distorting texture sampling based on the view angle.

Horizon mapping is the process of simulating an object casting shadows on itself using Bump mapped and Parallax mapped normal values.

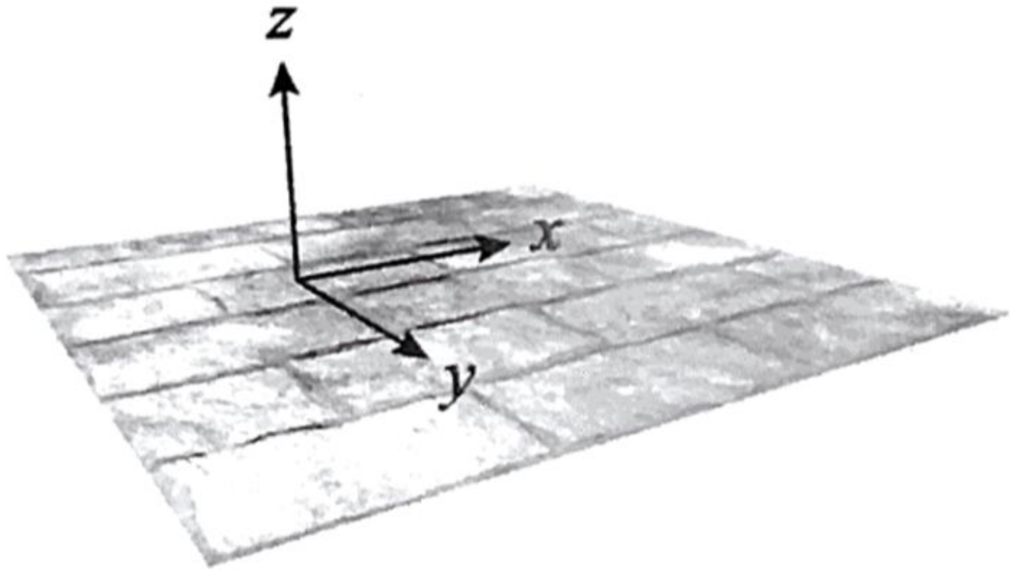
But wait!

How, Gigi and Tino, would I apply a 2D bump texture to a complex 3D model?

Tangent Space

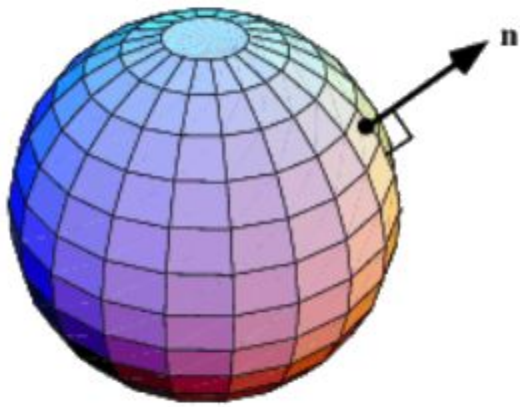
The Problem

Problem: we have a 2D texture containing vector data in texture space



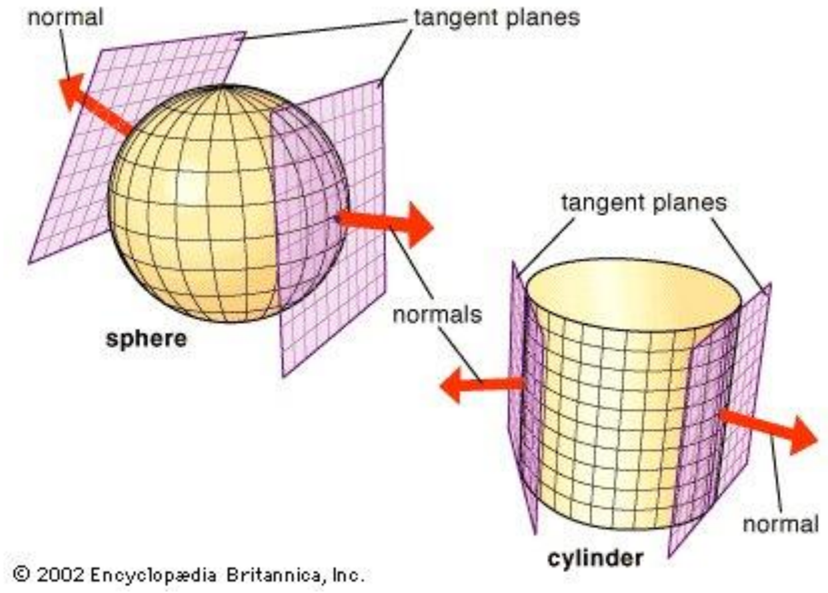
The Goal

Goal: we want a distortion we can apply to the surface normal (which is in object space)



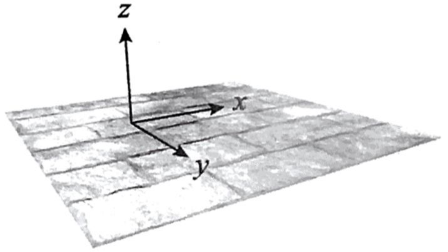
The Solution

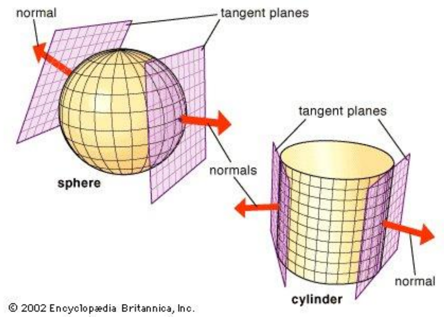
Solution: transform the data from texture space to the object’s tangent space for a given surface

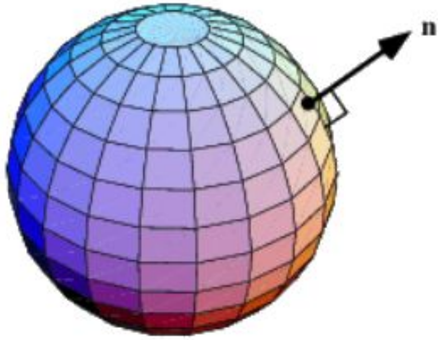


Tangent Space Transformation

This transformation is done by aligning the x and y axes of the texture space with the u and v axes of the surface in object space and the z axis to the surface normal







Texture space transformed to Tangent Space can be converted to Object Space!

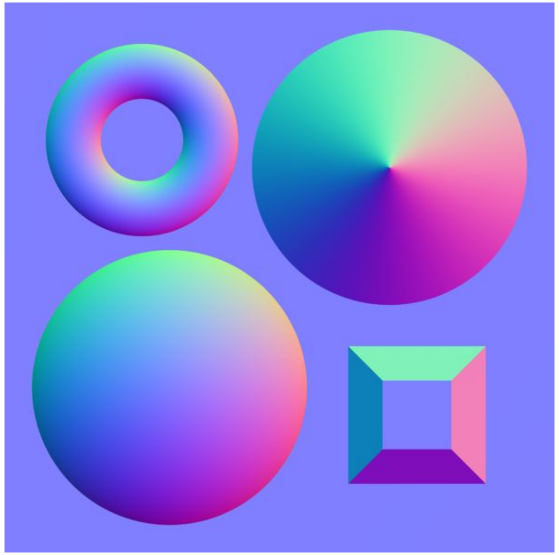
Tangent Space Transformation: Details

To do these transformations, we will have to use BIG SCARY MATH, which we will talk about later in the presentation (when we are making bump maps).

Bump Mapping

Normal Maps

Normal maps are bump maps that store normal vector distortions as RGB values. They store this information in tangent space.



Red distorts the **x** direction

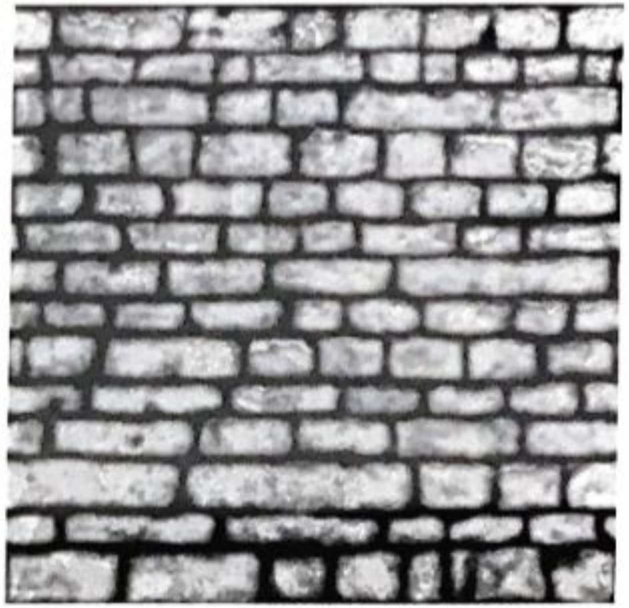
Green distorts the **y** direction

Blue distorts the **z** direction

Default value (no distortion): **(**0.5, 0.5, 1.0**)**

Making Normal Maps

To make a normal map, you'll first need a type of bump map called a "height map," where the relative heights of a texture are coded as values from black to white.



Making Normal Maps

In a height map, the height of each texel can be represented as

h(i, j)

where **i** and **j** are the coordinates of the texel, and **h** is a value between 0 and 1.

We also define a scale factor **s**.

We can then calculate x and y slopes for each texel like so:

$$d_x = \Delta \mathbf{z} / \Delta \mathbf{x} = (\mathbf{s} / 2) * [\ \mathbf{h}(\mathbf{i}+1, \mathbf{j}) - \mathbf{h}(i-1, \mathbf{j})\ ]$$

$$d_y = \Delta \mathbf{z} / \Delta \mathbf{x} = (\mathbf{s} / 2) * [\ \mathbf{h}(\mathbf{i}, \mathbf{j}+1) - \mathbf{h}(i, \mathbf{j}-1)\ ]$$

We’ll need to either clamp or wrap values at the edges of the texture.

Making Normal Maps

Now that we have d_x and d_y slopes, we can find some tangent vectors $\mathbf{u}_x$ and $\mathbf{u}_y$

$$\mathbf{u}_x = (1, 0, d_x)$$

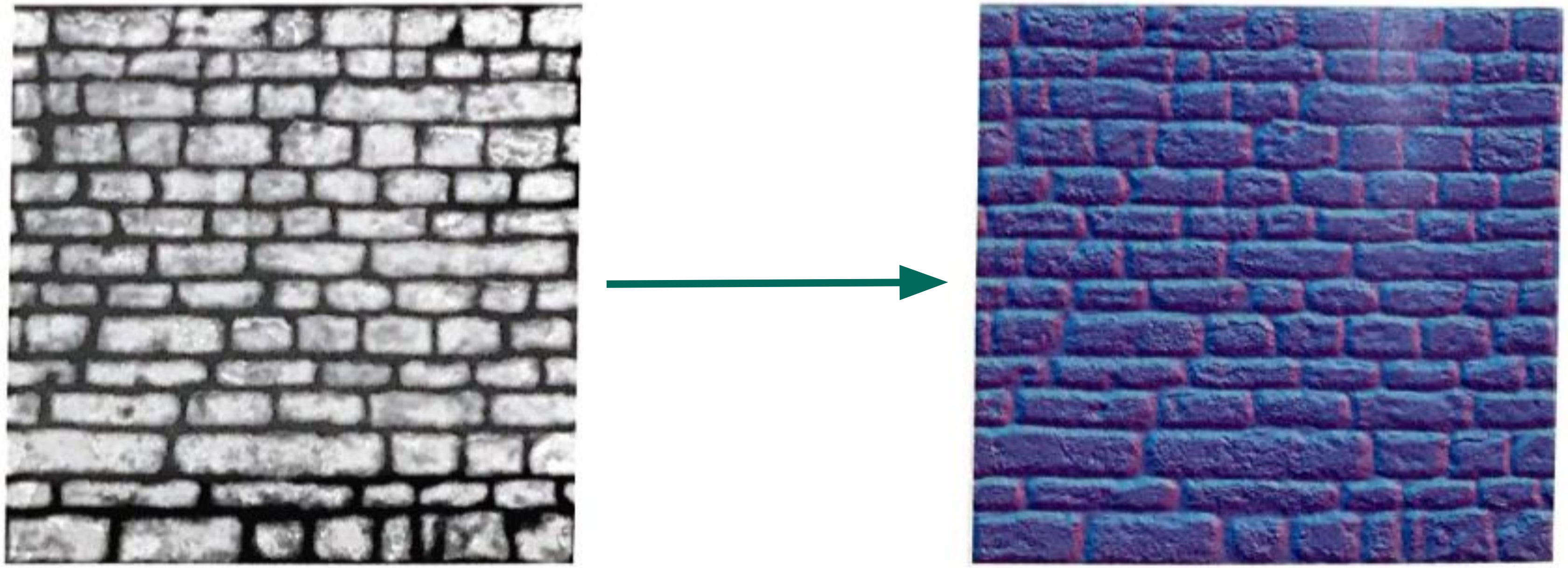
$$\mathbf{u}_y = (1, 0, d_y)$$

And the vector **m** we'll store as an RGB value in our Normal map will be the cross product of these two vectors, since we want to store our Normal map in **tangent space**.

$$\mathbf{m} = \text{normal}(\ \text{cross}(\mathbf{u}_x, \mathbf{u}_y)\ )$$

Making Normal Maps

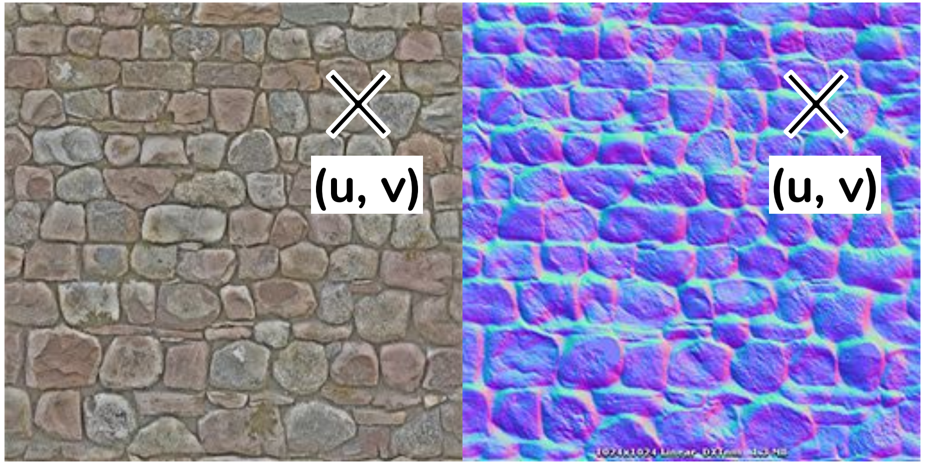
If we do these calculations for every pixel, we can turn our Height map into a Normal map!



Normal Maps

Because we can turn a texture into a height map, then convert that height map to a Normal map, they share UV coordinates!

When sampling a texture to get the color of a certain fragment, we use the same UV values to sample the Normal map, then use our distorted normal to calculate lighting.



Pretty cool right!

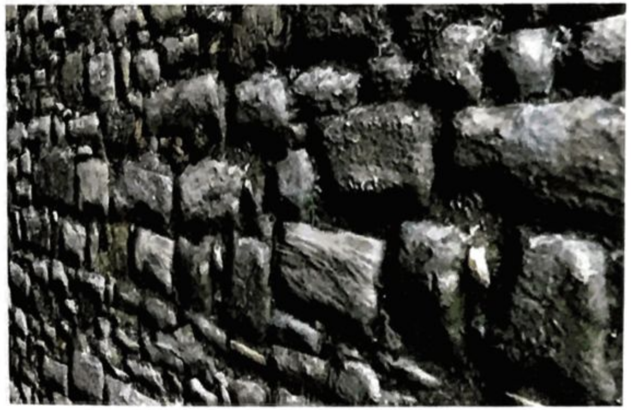
But what if there were problems?

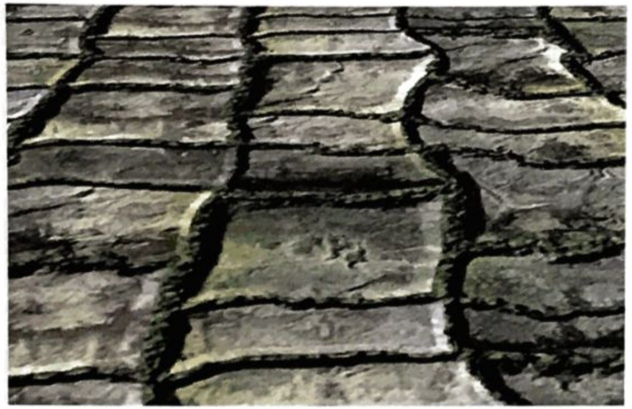
Parallax Mapping

The Problem

A big problem with basic normal mapping is that the bumpy effect it produces breaks down progressively as the angle between the view vector and the surface gets smaller

Surfaces with bump mapping viewed from small surface-view angles

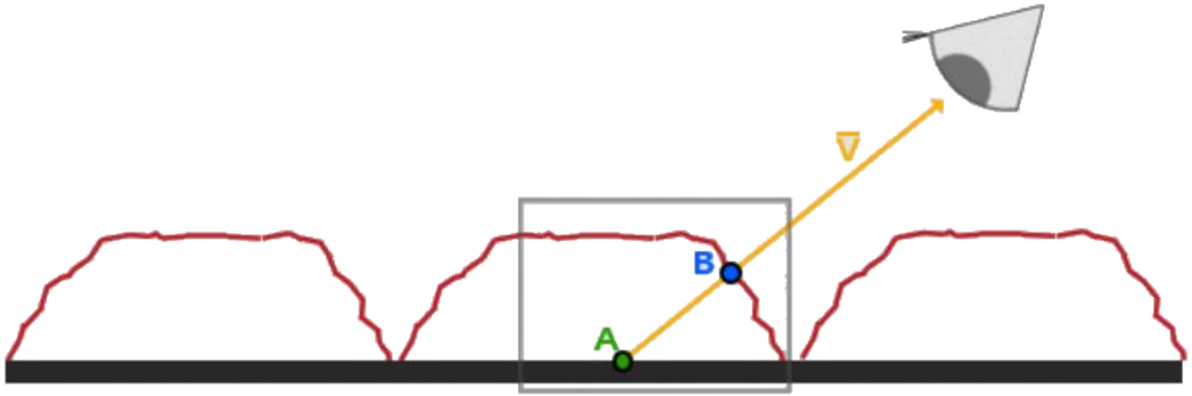




The Problem: Explained

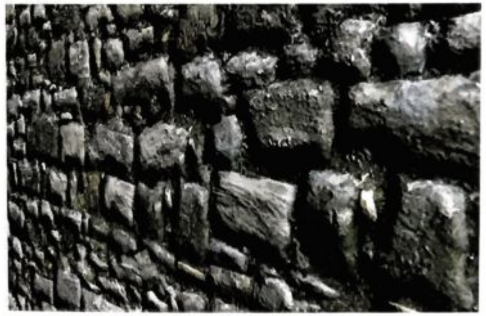
This problem occurs because the color texture's texels don't change locations as the viewing angle changes.

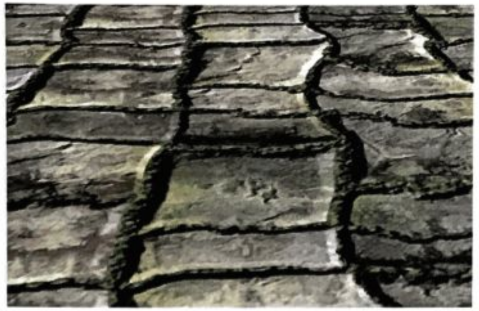
If the surface were actually bumpy, **we would expect to see parts of the texture be hidden** because they would be blocked by the bumps at certain view angles.



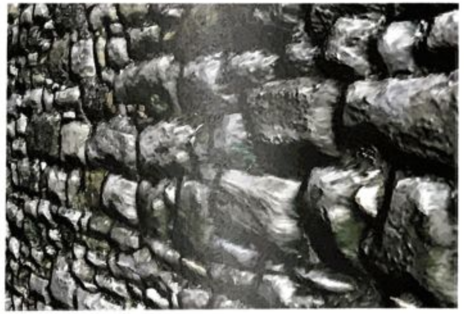
The Solution: Parallax Mapping!

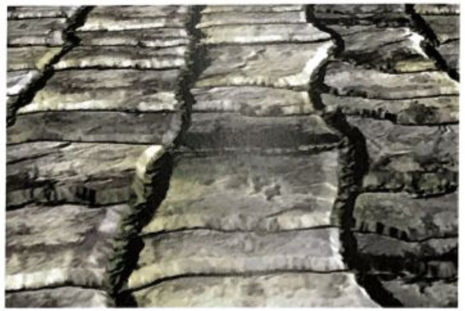
No Parallax





Parallax Mapping!





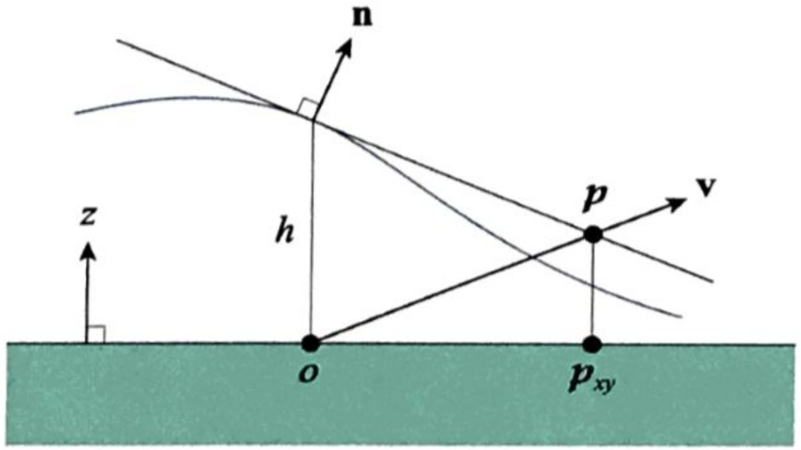
Parallax Mapping

Parallax mapping is a technique fixes the viewing angle problem by shifting texture sampling by a precomputed “parallax map” in relation to the current view.

This shift is applied to all textures that are sampled, including the color texture and our bump map!

Parallax Mapping: Concept

Conceptually, Parallax mapping is achieved by using a height value (sampled from a heightmap) and the surface normal to create a plane that approximates the virtual "bump" on the surface. **The intersection between this plane and the view vector is approximately what would be visible if there was actually a bump.**



Parallax Mapping: The Math

The concept from the previous slide can be applied with the following equation

$$p = o + \frac{n_z h}{\mathbf{n} \cdot \mathbf{v}} \mathbf{v}.$$

Where p is the point of intersection with the approximation plane, n is the surface normal, v is the view vector, h is the height value, and o is the initial point

A parallax map is then precomputed by applying this equation at every texel

Parallax Mapping: The Math (In practice)

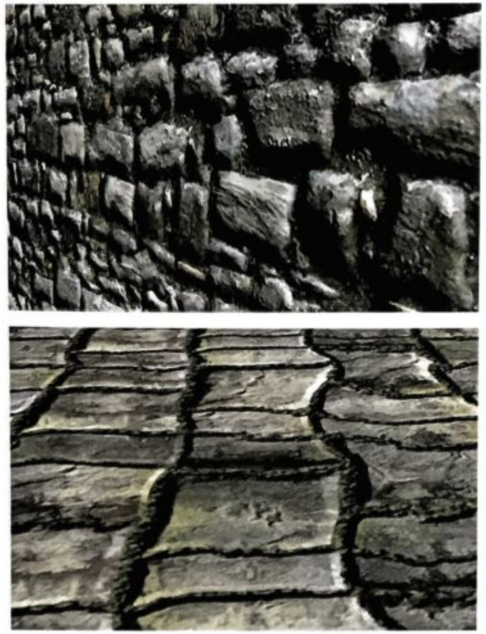
However, since the previous equation has issues when n dot v approaches zero or is negative, an alternate equation is used in practice.

$$p = o + \frac{n_z h}{\mathbf{n} \cdot \mathbf{v}} \mathbf{v}. \quad \Longrightarrow \quad p_{xy} = n_z h \mathbf{v}_{xy}.$$

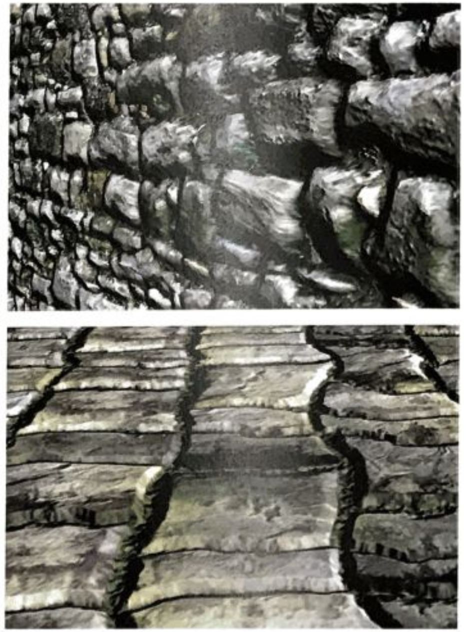
This alternate equation simplifies out the division of n dot v, removing the issues the original equation has as n dot v approaches zero or is negative.

Let’s Check Out our Cool Results Again!

No Parallax



Parallax Mapping!



Awesome!

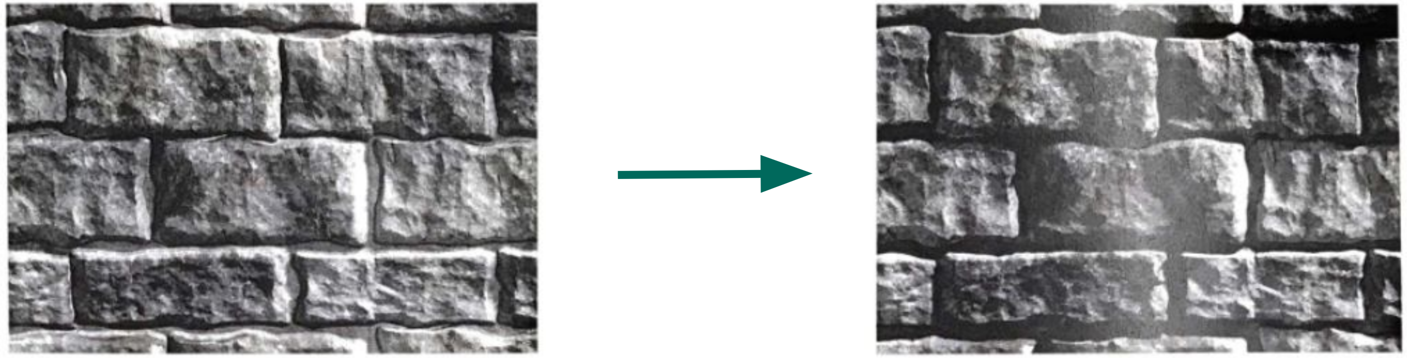
Ready for more problems?

Horizon Mapping

Horizon Mapping

Bump mapping and Parallax mapping do a lot to make surfaces look more realistic, but something they don't model is how a bumpy surface can cast shadows on itself.

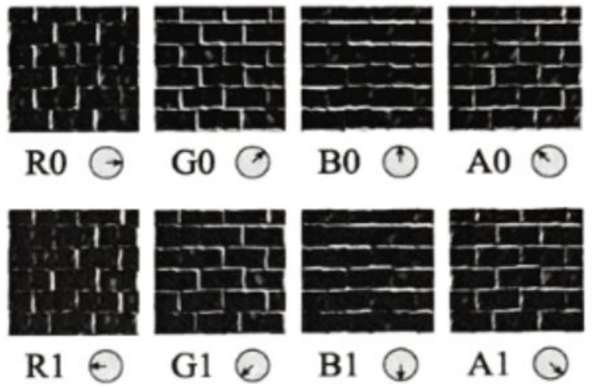
Horizon mapping is the process of creating and using textures to show how a surface would cast shadows on itself when light is cast at different angles.



Making Horizon Maps

Horizon maps are usually generated **4** or **8** at a time. This is because each horizon map contains data about self-shadowing when a light is at a particular angle. The more angles of horizon maps are generated, the more accurate the self-shadowing looks.

These horizon maps can be compacted into 1-2 actual textures, though.



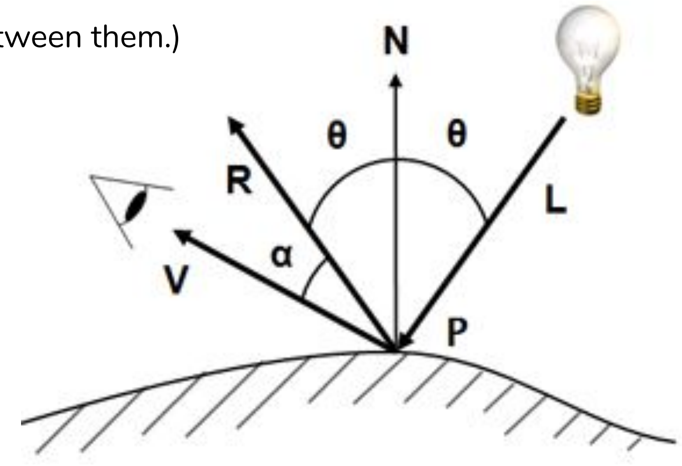
A Horizon map with two “layers,” where the R, G, B, and A values of each “layer” hold shadow data for light at a certain angle

Using Horizon Maps

When using horizon maps, the actual computation is quite simple because all the shadow data is already in our maps.

For each fragment, we find the angle between our normal vector and the light vector by taking the **arccosine** of their dot product.
(The dot product between two unit vectors is also the cosine of the angle between them.)

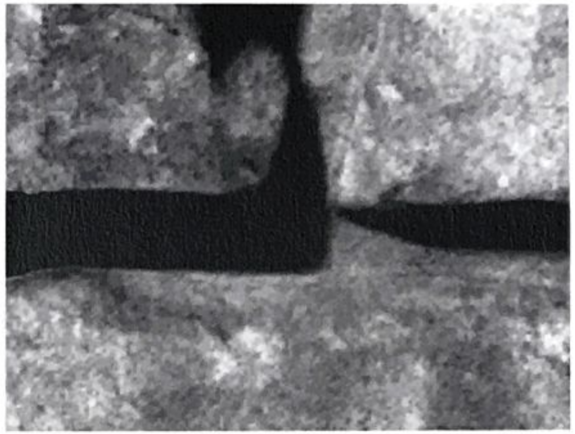
This angle will be **between** two of the light angles we have horizon maps for.



Using Horizon Maps

Now that we have the two angles that our current light is between, we can combine our horizon maps to get the right shadow.

We sample from the horizon maps of the two closest angles using our UV values, then **lerp** between them using our angle. This gives us a value between 0 and 1 that we can then multiply by our fragment color to create a soft shadow effect.



Any Questions?

References

Real Time Rendering, Fourth Edition
https://www.realtimerendering.com/blog/

Wikipedia - Tangent Space
https://en.wikipedia.org/wiki/Tangent_space

Unity Manual - Normal Map (Bump mapping)
https://docs.unity3d.com/Manual/StandardShaderMaterialParameterNormalMap.html

Learn OpenGL - Parallax Mapping
https://learnopengl.com/Advanced-Lighting/Parallax-Mapping