

Per-Pixel Strand Based Anisotropic Lighting

John Isidoro
ATI Research

Chris Brennan
ATI Research

Introduction

This technique uses Direct3D pixel shaders to implement a per-pixel version of the per-vertex anisotropic lighting technique presented in [Heidrich99]. An anisotropic lighting model is meant to simulate the effects of lighting on a grooved surface like a record, Christmas ornament, human hair, brushed metal or some fabrics. The per-pixel nature of the shader uses a direction of anisotropy map to determine the direction of the grooves. This allows us to be able to author many different anisotropic surface types by using standard image editing packages.

Strand Based Illumination

The basic idea behind the strand based illumination is the lighting of strands. These strands are modeled as having a circular cross-section. When lighting them, we use the standard Phong illumination model but with a slight modification. The normal used in the Phong model is a normal which is equal to the light vector projected into the normal plane of the strand at the point being lit, and normalized. We call this strand normal vector N' . The modified Phong equation is:

$$I_o = k_a I_a + (k_d (L \cdot N') + k_s (V \cdot R')^n) I_i$$

The terms k_a , k_d , and k_s are material properties for the ambient, diffuse and specular lighting respectively. I_a is the ambient light color, and I_i is the incident light color. L is the light direction vector, V is the view direction vector and n is the specular exponent. The reflection vector R' can be found using

$$R' = 2N'(N' \cdot V) - V$$

As shown in [Banks94] and [Stalling97], using some trigonometric identities, and the knowledge that L , T , and N' are coplanar, $L \cdot N'$ and $V \cdot R'$ can be expressed in terms of L , V , and T , the direction of the strand.

$$L \cdot N' = (1 - (L \cdot T)^2)^{0.5}$$

$$V \cdot R' = (1 - (L \cdot T)^2)^{0.5} (1 - (V \cdot T)^2)^{0.5} - (L \cdot T) (V \cdot T)$$

Luckily, these equations are a function of two dot products, $L \cdot T$ and $V \cdot T$. This allows us to use these dot products to index into a pre-computed 2D texture map to compute the diffuse $L \cdot N'$ and specular $(V \cdot R')^n$ terms. Using the fact that L , T , and V are unit length vectors, it is known that the terms $L \cdot T$ and $V \cdot T$ range between $[-1, 1]$. To use these terms to index into a texture lookup table, first they must be scaled and biased into the range of $[0, 1]$.

Using this knowledge, building the texture lookup table $\text{tex}(u, v)$ is straightforward. In the equations above, just substitute $2u-1$ for $L \cdot T$ and $2v-1$ for $V \cdot T$. The diffuse component can be packed into the rgb component of the texture and the specular component can be packed into the alpha channel of the texture. This way both diffuse and specular lookups can be done with a single texture fetch per light source. Also, the specular exponent n can be pre-baked into the specular lookup table by computing $(V \cdot R')^n$ instead of just computing $(V \cdot R')$.



Diffuse Texture Lookup Table



Specular Texture Lookup Table

The rest of the math in the Phong model can be computed within the pixel shader.

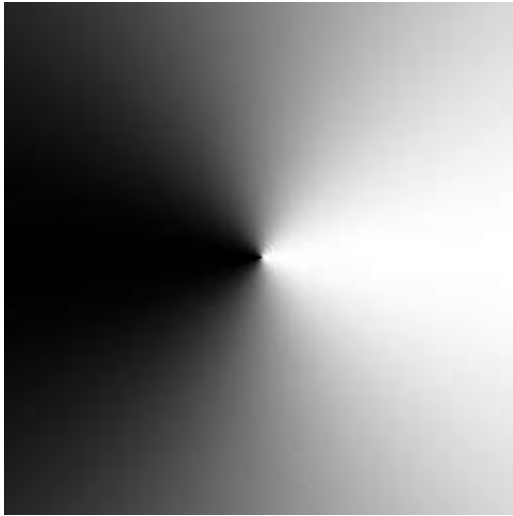
Rendering using the Texture Lookup Table

Depending on whether the technique is applied per-pixel or per-vertex, there are a few places where the dot products, scaling and biasing can be computed. When rendering on a per-vertex basis, all the computation can easily be performed in a vertex shader or even in a traditional texture matrix. The tangent vector T can be stored with the vertex data as a per vertex three-element texture coordinate. The light L and view V vectors can be stored as vertex shader constants. All vectors should be in object space if the texture matrix is used to perform the dot products. This is explained in more detail in [Heidrich99]. The main focus of this chapter is to illustrate the implementation of

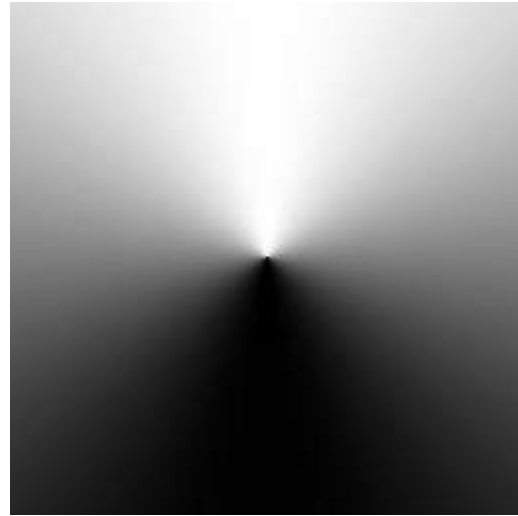
anisotropic on a per-pixel basis and to present a unique orthonormalization technique that allows anisotropic lighting to be combined with bump mapping.

Per-Pixel Strand Based Illumination

Per-pixel strand based anisotropic lighting has a lot in common with standard per-pixel lighting (aka bump mapping). First off, all per-pixel computation is done in tangent space. The direction of anisotropy is stored in a texture map, and being in tangent space has the advantage that the direction map is not coupled to the geometry. The same direction map can be applied to many different objects, and in a way can be thought of as part of the surface description in the same way a bump map is. The direction vectors T_t are encoded in the texture the same way normals are in a bump map, the xyz direction is scaled by 0.5 and biased by 0.5, and encoded as an rgb color. The light and view vectors are computed and transformed into tangent space in the vertex shader. The resulting vectors L_t and V_t are interpolated across the triangle.



Red Channel of Direction of Anisotropy Map

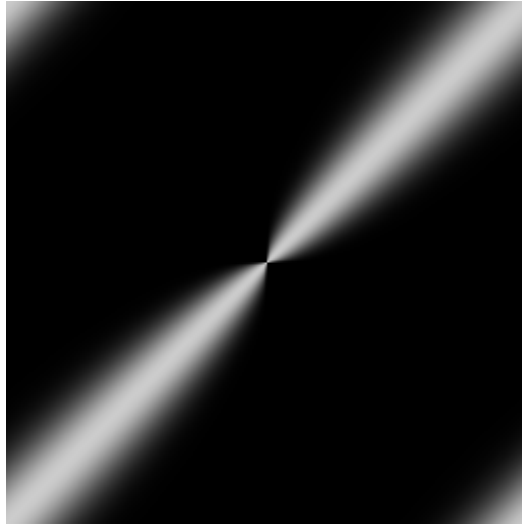


Green Channel of Direction of Anisotropy Map

In the pixel shader, two dot products need to be performed ($T_t \cdot L_t$), and ($T_t \cdot V_t$), and the resulting vector should be scaled and biased by 0.5 and used to fetch a texel in the texture lookup table. However, we can make a few optimizations here.

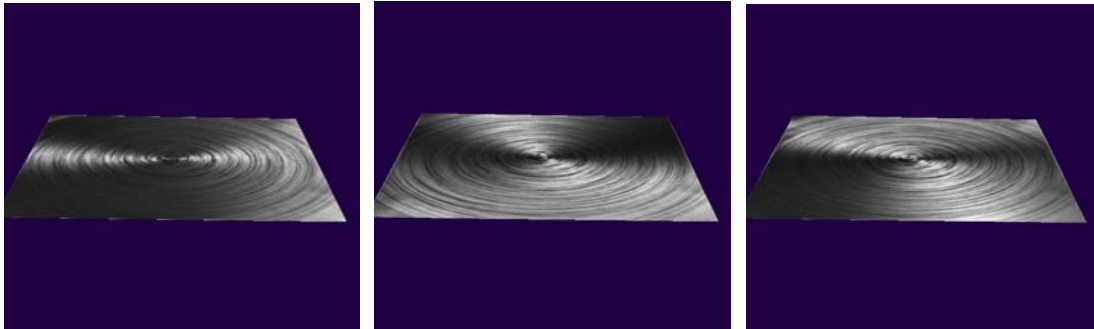
The first optimization is to perform the scale by 0.5 inside the vertex shader by multiplying L_t and V_t by 0.5 before outputting them. The second optimization is to offset the texture lookup table in u and v by half the texture width and height respectively. By doing this and turning on texture wrapping in the texture stage containing the lookup map, this essentially gives us the bias by 0.5 for free. These optimizations save precious pixel shader instructions, and in the case of 1.1 pixel shaders, make it possible to perform the anisotropic lighting at all.

Per-Pixel Strand Based Anisotropic Lighting



Specular component shown offset by half of the texture width and height.

Per-Pixel Strand Based Illumination with Colored Light and Basemap



Result of per pixel anisotropic lighting (only 2 triangles!)

Here is a pixel shader that implements the anisotropic strand illumination.

```
ps.1.1
tex t0          ; Contains direction of anisotropy in tangent space
tex t1          ; base map

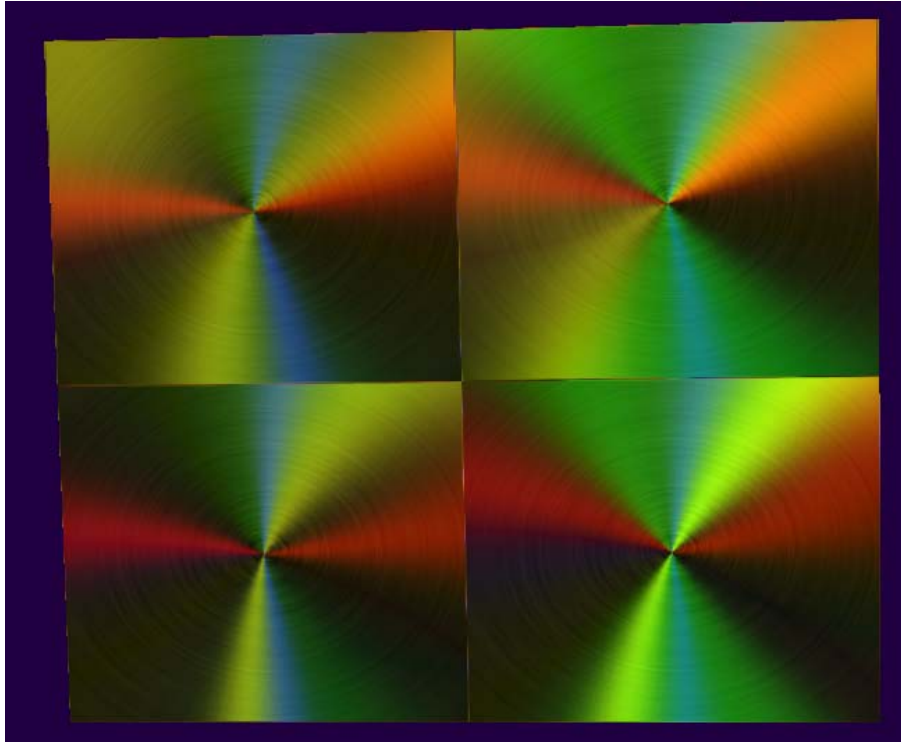
texm3x2pad t2, t0_bx2 ; Perform fist row of matrix multiply.
texm3x2tex t3, t0_bx2 ; Perform second row of matrix multiply to get a
                      ; 3-vector with which to sample texture 3, which is
                      ; a look-up table for aniso lighting

mad r0, t3, t1, t1.a ; basemap * diffuse + specular
mul r0, r0, c0      ; color * basemap *diffuse
```

Excerpted from *ShaderX: Vertex and Pixel Shader Tips and Tricks*

Per-Pixel Strand Based Illumination with 4 Colored Lights and Base Map in one pass

Here is a pixel shader that implements the anisotropic strand shader, with 4 colored lights:



Anisotropic lighting with 4 colored lights

```
ps.1.4
texld r0, t0 //sample bump map (note alpha channel = 1)
texcrd r1, t1
texcrd r1, t2
texcrd r1, t3
texcrd r1, t4
texcrd r1, t5

dp3_d2 r1.rba, r1, r0_bx2 // .5 L0.T
dp3_d2 r2.rba, r2, r0_bx2 // .5 L1.T
dp3_d2 r3.rba, r3, r0_bx2 // .5 L2.T
dp3_d2 r4.rba, r4, r0_bx2 // .5 L3.T
dp3_d2 r1.g, r5, r0_bx2 // .5 V.T
mov r2.g, r1.g
mov r3.g, r1.g
mov r4.g, r1.g

phase
texld r1, r1 //aniso lookup 0
texld r2, r2 //aniso lookup 1
texld r3, r3 //aniso lookup 2
texld r4, r4 //aniso lookup 3
texld r5, t0 //sample base map

mad r1, r1, r5, r1.a // diffuse0 * base + specular
```

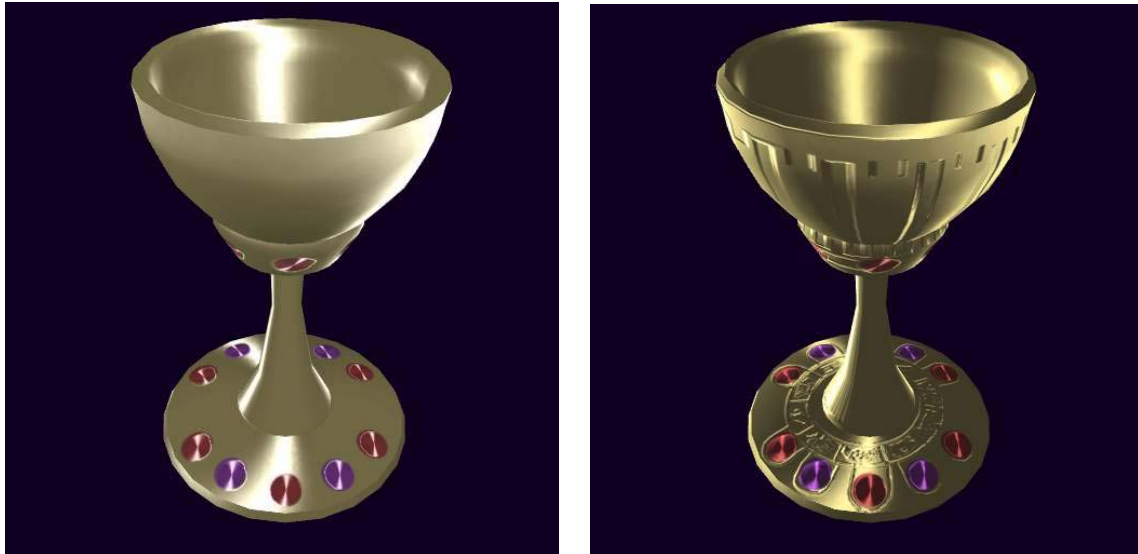
Excerpted from *ShaderX: Vertex and Pixel Shader Tips and Tricks*

Per-Pixel Strand Based Anisotropic Lighting

```
mad r2, r2, r5, r2.a // diffuse1 * base + specular
mad r3, r3, r5, r3.a // diffuse2 * base + specular
mad r4, r4, r5, r4.a // diffuse3 * base + specular
mul r0, r1, c1       // lightcolor0 * light0 effect
mad r0, r2, c2, r0   // lightcolor1 * light1 effect + previous
mad r0, r3, c3, r0   // lightcolor2 * light2 effect + previous
mad r0, r4, c4, r0   // lightcolor3 * light3 effect + previous
```

Per-Pixel Bump-Mapped Strand Based Illumination using Graham-Schmidt Orthonormalization

Another technique for bump mapping combines both anisotropic strand-based lighting and bump mapping at the same time. This way more surface detail can be added to objects using the anisotropic strand shader.

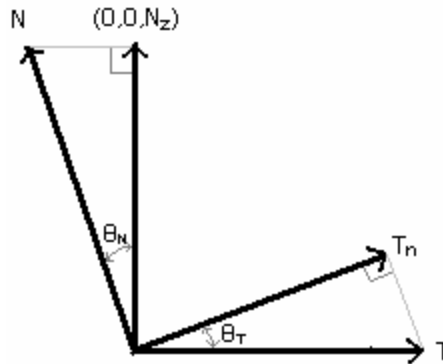


Anisotropically lit chalice without and with bump mapping

This shader uses the bump map to perturb the direction of anisotropy, by using Graham-Schmidt orthonormalization. The basic idea is to make the direction of anisotropy orthogonal to the bump map normal, by subtracting off the component of the tangent vector T projected onto the normal N , and renormalizing the resulting vector:

$$T_{new} = || T - (N \cdot T) N ||$$

Per-Pixel Strand Based Anisotropic Lighting



There are two possible places this math can be performed, either as a preprocessing step on the direction of anisotropy texture, or inside of the pixel shader. Doing the orthonormalization as a preprocessing step has the advantage of not requiring any extra math in the pixel shader. However, it has the negative effect of tying the preprocessed direction map to the corresponding bump map. If the same direction map needs to be used with many different bump maps, additional textures need to be allocated. In addition to this, if the texture coordinates for the bump and direction maps are different, this preprocessing step is usually not possible.

The alternative is to do the orthonormalization in the pixel shader, as is done in the following example. All the math is fairly straightforward to perform in the pixel shader except for the vector normalization step. Luckily for us, this renormalization step has almost no visual impact for due to the way the math works out. In the worst case, the lighting near the severe bumps may become slightly brighter. In addition to this, for most surfaces the direction of anisotropy runs in the direction of the grooves in the bumpmap, thus making the bumpmap vectors perpendicular to the direction of anisotropy.

```
; t1 contains tangent-space L vector          ; Lx, Ly, Lz, 0
; t2 contains tangent-space reflection vector ; Vx, Vy, Vz, 0
; Map bound at stage 0 is a normal map
; Map bound at stage 3 is a T.L x T.V map as the z positive face of a cube map
ps.1.4

texld r0, t0          ; Contains direction of anisotropy in tangent space
texcrd r2.rgb, t1     ; Light vector
texcrd r3.rgb, t2     ; View vector
texld r4, t3         ; Bump map

dp3 r5, r0_bx2, r4_bx2 ; T.N
mad r0.xyz, r4_bx2, -r5, r0_bx2 ; Tn = T - N(T.N)

; Calculate V.T and L.T for looking up into function map.
dp3_d2 r1.x, r2, r0   ; L.Tn
dp3_d2 r1.y, r3, r0   ; V.Tn

phase

texld r2, r1          ; Normalize Tn, and perform anisotropic lighting
; Function lookup
```

Excerpted from *ShaderX: Vertex and Pixel Shader Tips and Tricks*

Per-Pixel Strand Based Anisotropic Lighting

```
texld r3, t0 ; Base map

mul_sat r0.rgb, r3, r2 ; aniso diffuse * basemap
mad_sat r0.rgb, r2.a, r3.a, r0 ; + glossmap * aniso specular
+mov r0.a, c0.b
```

Summary

A technique for per-pixel strand based anisotropy has been described. This technique allows a per-texel direction of anisotropy vector as well as an optional per-texel bump map normal to be specified. Due to the efficiency of the math used, up to 4 lights per rendering pass can be achieved using DirectX 8.1 version 1.4 pixel shaders.

References

[Banks94] D. C. Banks, "Illumination in diverse codimensions," SIGGRAPH 94, pp 327-334, July 1994.

[Heidrich99] Wolfgang Heidrich and Hans-Peter Seidel, "Realistic, Hardware-accelerated Shading and Lighting," SIGGRAPH '99.

[Stalling97] D. Stalling, M. Zöckler and H.-C. Hege, "Fast Display of Illuminated Field Lines," IEEE Transactions on Visualization and Computer Graphics, 3(2):118-129, 1997.