# 3D developemt research

## Spot light in directx11

Research done on how to implement spotlight had been great as there are multiple example and tutorial on the internet show how and the theory of spotlight. As follow and in the appendix will have the figure and picture showing the cone and the formula to calculate the light to be limited in a cone shape hence turning directional light to a spot light. In fig 1.1 in a representation of the cone of light and in fig1.2 will have the explanation of the light and math of the cone and light calculation.

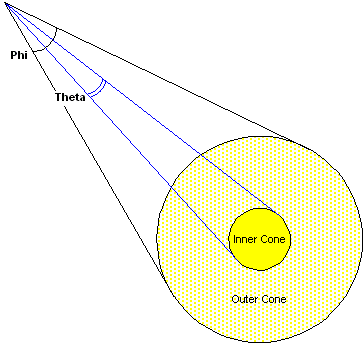


Fig1.1 <https://docs.microsoft.com/en-us/windows/desktop/direct3d9/light-types>

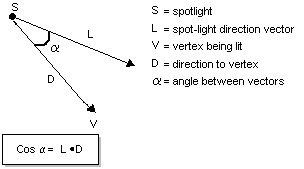


Fig1.2 <https://docs.microsoft.com/en-us/windows/desktop/direct3d9/light-types>

finalColor \*= pow(max(dot(-lightToPixelVec, light.dir), 0.0f), light.cone);

the formula highlighted above is the formula to calculate the to define in cone light turning a point light in existing code and turning the point light to a cone and forming a spotlight. Further explanation is in appendix (1.1)

## anisotropic filtering

anisotropic filtering is a technique use to output a better solution for texture that are going to be view at angle on object in the game or scene. Anti-aliasing is an isotropic filtering so anisotropic filtering from what the research be said is that anisotropic filtering is the opposite of anti-aliasing.

# appendix 1.1

**SpotLight Factors**

Spotlights are basically just point lights but with a direction. Because of this, we will build directly off our implimentation of the point light, adding only a direction, and a value representing the size of the cone of our spotlight.

## Defining the Cone of Light

Here is the only new equation we will put into our effect file. This will create a cone that our light will shine through.

The first thing is find the angle between our lights direction, and the direction from the light to the pixel. We use the dot product to find this.

Then we use the max function to make sure we do not get dot product less than 0.0f. A dot product less than 0.0f would be behind the position of the spotlight, therefore shining light not only in the front of the spotlight, but also behind the spotlight.

Finally, we use the pow function with the light.cone value as the exponent. This will complete the equation which defines our cone of light.

finalColor \*= pow(max(dot(-lightToPixelVec, light.dir), 0.0f), light.cone);

|| Website-from <https://www.braynzarsoft.net/viewtutorial/q16390-21-spotlights>

“Setting up a spot light is exactly like setting up a point light, but with the new properties included. The four properties added here from the code above are phi, theta, falloff and direction. Also, the type was changed to D3DLIGHT\_SPOT.”

||Website- from <http://www.directxtutorial.com/Lesson.aspx?lessonid=9-4-9>

## Adding a Spotlight Effect

Another commonly used extension for the Basic lighting model is making the light a spotlight instead of an omnidirectional light. A spotlight cut-off angle controls the spread of the spotlight cone, as shown in Figure 5-18. Only objects within the spotlight cone receive light.

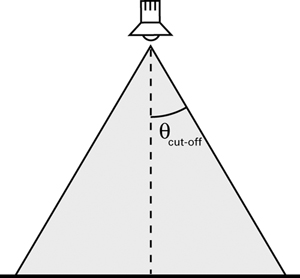


Figure 5-18 Specifying a Spotlight Cut-Off Angle

To create the spotlight cone, you need to know the spotlight position, spotlight direction, and position of the point that you are trying to shade. With this information, you can compute the vectors V (the vector from the spotlight to the vertex) and D (the direction of the spotlight), as shown in Figure 5-19.

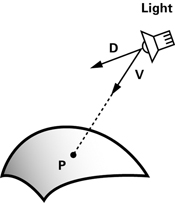


Figure 5-19 Vectors for Calculating the Spotlight Effect

By taking the dot product of the two normalized vectors, you can find the cosine of the angle between them, and use that to find out if P lies within the spotlight cone. P is affected by the spotlight only if **dot(V, D)**is greater than the cosine of the spotlight's cut-off angle.

Based on this math, we can create a function for the spotlight calculation, as shown in Example 5-8. The function **C5E8\_spotlight**returns 1 if P is within the spotlight cone, and 0 otherwise. Note that we have added **direction**(the spotlight direction—assumed to be normalized already) and **cosLightAngle**(the cosine of the spotlight's cut-off angle) to the **Light**structure from Example 5-6.

|| Website from <http://developer.download.nvidia.com/CgTutorial/cg_tutorial_chapter05.html>

# Appendix 2.1

## Anisotropic Filtering

**Anisotropic Texture Filtering**  code like this: C++Copy

D3D11\_SAMPLER\_DESC sampler\_description;

// ... other sampler description setup ...

sampler\_description.Filter = D3D11\_FILTER\_ANISOTROPIC;

sampler\_description.MaxAnisotropy = 16;

d3d\_device->CreateSamplerState(&sampler\_desc, &sampler);

d3d\_context->PSSetSamplers(0, 1, &sampler

|| Website from

<https://docs.microsoft.com/en-us/visualstudio/debugger/graphics/point-bilinear-trilinear-and-anisotropic-texture-filtering-variants?view=vs-2017>

“MaxAnisotropy - Clamping value used if D3D11\_FILTER\_ANISOTROPIC or D3D11\_FILTER\_COMPARISON\_ANISOTROPIC is specified in Filter. Valid values are between 1 and 16.” --||website from <https://www.braynzarsoft.net/viewtutorial/q16390-11-textures>

“Anisotropic filtering exists to provide superior image quality in virtually all cases at the slight expense of performance. By the computer science definition, anisotropy is the quality of possessing dissimilar coordinate values in a space, which applies to any texture not displayed as absolutely perpendicular to the camera. As previously mentioned, bilinear and trilinear filtering suffer from resultant quality loss when the sampled textures are oblique with the camera due to both methods obtaining texel samples from mipmaps assuming that the mapped texel is perfectly square in the rendered space, which is rarely true. This quality loss is also related to the fact that mipmaps are isotropic, or possessing identical dimensions, so when a texel is trapezoidal there is insufficient sampling in both directions. To solve this, anisotropic filtering scales either the height or width of a mipmap by a ratio relative to the perspective distortion of the texture; the ratio is dependent on the maximum sampling value specified, followed by taking the appropriate samples. AF can function with anisotropy levels between 1 (no scaling) and 16, defining the maximum degree which a mipmap can be scaled by, but AF is commonly offered to the user in powers of two: 2x, 4x, 8x, and 16x. The difference between these settings is the maximum angle that AF will filter the texture by. For example: 4x will filter textures at angles twice as steep as 2x, but will still apply standard 2x filtering to textures within the 2x range to optimize performance. There are subjective diminishing returns with the use of higher AF settings because the angles at which they are applied become exponentially rarer.”--|| Website from <https://www.geforce.com/whats-new/guides/aa-af-guide#1>