[11 Aug 2011]

Blinn-Phong reflection model in GLSL

This tutorial describes Blinn-Phong reflection model that is one of the simplest reflection models in computer graphics. This model was used until recent time in fixed graphical pipelines of OpenGL and DirectX graphics APIs. Now developers, in newest versions of OpenGL and DirectX, have to create shaders that implement reflection models.

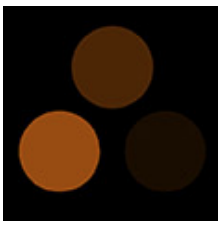
Also this tutorial introduces different shading models for triangles: flat shading, Gouraud shading and Phong shading.

Blinn-Phong reflection model

Phong reflection model uses combination of diffuse reflection (like rough dim materials), specular reflection (like smooth shiny materials) and approximation of ambient lighting (lighting in places which aren't lightened by direct light rays). This is model of local lighting of points on a surface, where result of lighting doesn't depend on other objects in the scene or on repeatedly reflected light rays.

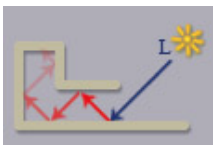
Ambient lighting

Ambient lighting represents lighting that isn't created by direct light rays. This is indirect lighting. In real world ambient lighting occurs due to multiple reflections of light rays from different objects. The most simple way to implement ambient lighting is to use constant value of ambient light intensity for whole scene. And Phong refelection model uses such approximation of ambient lighting. Constant value for ambient lighting gives very good performance to the shader, but it's very unrealistic.



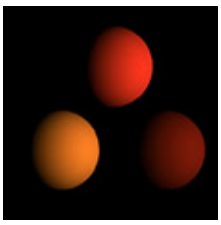
The main disadvantage of such ambient lighting is that all objects will be lightened with same ambient lighting intensity even if objects aren't accessible for direct or reflected light rays. Another disadvantage is that ambient lighting in Phong reflection model doesn't take into accout change of color of light rays after multiple reflections. For example, environment around a red object is partly lightened with red color, even if color of the light doesn't have red component.

$$I_{amb} = const$$

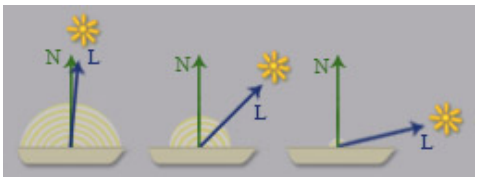


Diffuse reflection

Diffuse reflection emerges when direct light rays hit the surface. After that light rays are scattered in all directions with same intensity. Intensity of diffuse reflection doesn't depend on position of the camera. This definition of reflection corresponds to Lambertial reflectance. Lambertian reflectance defines behavior of light rays after hit with matte surface. Visible brightness of the surface doesn't depend on angle under which the camera looks at it. For Lambertian reflectance, intensity of reflected light is calculated with Lambert's cosine law: intensity of reflected light is directly proportional to cosine of angle between the normal (N) and vector to the light (L):

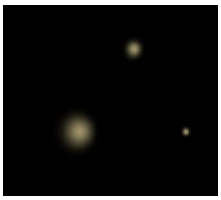


$$I_{dif} = clamp(N \cdot L, 0, 1)$$

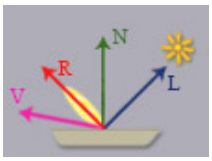


Specular reflection

Specular reflection emerges when direct light rays reflect from the surface



in direction to the camera. Specular reflection creates highlights on the surface. Highlight is a bright spot, which is visible only when light from light source reflects directly to the camera. Light rays go from the light source to the point on the surface, and reflect relative to the normal at the point. Specular reflection is visible only if direction to the camera (V) and direction of reflected rays (R) are similar enough.



To calculate specular reflection you have to use normal at the point (N), direction from the point to the camera (L), and also reflected direction to light (V). Direction to the light is reflected relative to the normal of the surface, and creates reflected direction (R):

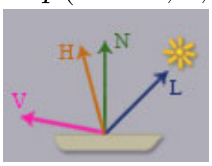
$$R = reflect(L, N)$$

Now you can calculate intensity for specular reflection as for diffuse reflection, but for direction to the camera (V) and reflected direction (R). So you can calculate intensity as cosine of angle between these two vectors. Result is similar to diffuse reflection. Light fades slowly from full intensity to shadow. But specular highlights in real world are small, brighter and have rapid changes in intensity. To decrease size of specular highlight you can take calculated reflected intensity to power of 32, 64 or 128, etc. The larger the power, the smaller and sharper the highlights will be. To increase intensity of specular highlights you can multiply intensity by some constant value.

$$I_{spec} = clamp(V \cdot R, 0, 1)^{shininess}$$

Instead of calculation of reflected direction (R) you can calculate half vector (H), which is normalized average between direction to the camera (V) and direction to the light (L). With vector H you can calculate intensity of specular reflection as cosine of angle between normal (N) and half vector (H). Reflection model that uses R and V vectors for specular reflections is called Phong reflection model, and another model that uses N and H vectors is Blinn-Phong reflection model. Blinn-Phong reflection model was used in fixed function pipelines of OpenGL and DirectX.

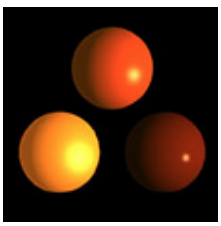
$$I_{spec} = clamp(N \cdot H, 0, 1)^{shininess}$$



Specular reflectance should be calculated only for surfaces oriented to the light source:

$$N \cdot L > 0$$

Combining lighting components



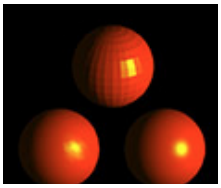
Source of the light has intensity and color. In Blinn-Phong reflection model these parameters are separate for diffuse reflection, specular reflection and for ambient lighting. Color and intensity of the light source are treated as single RGB value - each channel of the color represents intensity. Diffuse color and intensity of the light sources is denoted as K_{dif}, specular color - K_{spec}, ambient color - K_{amb}. Color of each lighting component is multiplied by corresponding intensity of diffuse reflectance, specular reflectanse or amount of ambient lighting.

Blinn-Phong reflactance model also takes into account a material of the surface. The material has following parameters: level of diffuse reflectance M_{dif}, level of specular reflectance M_{spec} and level of reflectance for ambient lighting M_{amb}. If reflectance level is 0, then surface doesn't reflect light, and if reflectance level is 1, then surface reflects all incoming light. Similarly to color and intensity of the light source, level of reflectance for each component is represented with RGB value.

To calculate final color of each point on the surface, components of Blinn-Phong reflectance model are multiplied with corresponding parameters of the light source and the material, and then added together:

$$C_{final} = K_{amb}M_{amb}I_{amb} + K_{dif}M_{dif}I_{dif} + K_{spec}M_{spec}I_{spec}$$

Shading models



Shading model determines on which shader stage and with which quality lighting for triangles is calculated. You can calculate lighting once for each triangle, once for each vertex of the triangle or for each fragment of the triangle. Each shading model has its own quality of lighting, computation speed, style, implementation method, etc. Following sections shows differences between different shading models.

Flat shading

Flat shading model calculates lighting in such a way that each point of the triangle has same lighting. This is done through usage of same normal for each vertex of the triangle. As the normal is same in all points of the triangle, so lighting that depends on the normal will have same intensity in all points of the triangle. You can pass normals to shaders as vertex attribute, but in such case you have to duplicate all vertices that are shared among triangles with different normals (to preserve same normal for each triangle). When "flat" normal is available in vertex shader you can calculate lighting there. Otherwise you can use geometry shader to calculate normal of the triangle on the fly (as perpendicular to two vectors that form triangle). In such case you can calculate lighting in geometry shader, not in vertex shader.

Gouraud shading

In Gouraud shading each vertex of the triangle can have different normals. Lighting is calculated in vertex shader and takes into account normal at the vertex. Normals can be different, and as result each vertex will have different lighting intensity. Calculated lighting intensity (or color of the lighting) is then passed to fragment shader. Each fragment gets interpolated (among three vertices) lighting intensity.

The main disadvantage of this model is that lighting is calculated only for three vertices and then linearly interpolated. If the mesh contains large triangle, then you will notice that lighting is interpolated rather than calculated independently for each fragment. This disadvantage is clearly visible during calculation of specular reflections, when specular highlights are poorly visualized or even became invisible. Lighting is calculated at vertices, and if specular hightlight is at the middle of the large triangle, then it won't affect lighting at the vertices. After that lighting is interpolated linearly, and you won't see any specular highlights. This is because linear interpolation doesn't know about highlights at the middle of the triangle. Rotate mesh and you will notice flickering. Highlight will be visible when it's near the vertex, and then again invisible when it's far from the vertex. To minimize this effect you can increase number of triangles in the mesh, but it won't totally elliminate the problem.

Phong shading

In contrast to Gouraud shading, Phong shading model calculates lighting for each fragment of the triangle. The vertex shader doesn't calculate any lighting, and only passes vectors required for lighing to fragment shader (the normal N, the light vector L, the view vector V). Each fragment gets interpolated normal from three normals at each vertex of the triangle. This normal and other interpolated vectors are then used to calculate lighting. This lighting model is best for visualization of specular reflections because it calculates data for each fragment without any interpolation.

Purpose of different shading models

Each shading model has its own advantages and disadvantages. Flat shading model can be used for stylized rendering when you want to show triangles of the mesh. Phong shading model is best for rendering of highlights and small details, and also it's the most realistic among simple shading models. But Gouraud shading model gives better performance than Phong shading model, as it computes lighting only for vertices, not for fragments. And difference in amount of calculations is considerable.

Shaders that implements Blinn-Phong reflection model

Following code snippets contain vertex and fragment shaders which implements Blinn-Phong reflection model with Phong shading:

Vertex shader:

```
#version 330

// attributes
layout(location = 0) in vec3 i_position; // xyz - position
layout(location = 1) in vec3 i_normal; // xyz - normal
layout(location = 2) in vec2 i_texcoord0; // xy - texture coords

// matrices
uniform mat4 u_modelMat;
uniform mat4 u_viewMat;
uniform mat4 u_projMat;
uniform mat3 u_normalMat;

// position of light and camera
uniform vec3 u_lightPosition;
uniform vec3 u_cameraPosition;

// data for fragment shader
out vec3 o_normal;
out vec3 o_toLight;
out vec3 o_toCamera;
out vec2 o_texcoords;
```

```
out vec2 o_texcoords;

////////////////////////////////////////////////////////////////////

void main(void)
{
   // position in world space
   vec4 worldPosition = u_modelMat * vec4(i_position, 1);

   // normal in world space
   o_normal = normalize(u_normalMat * i_normal);

   // direction to light
   o_toLight = normalize(u_lightPosition - worldPosition.xyz);

   // direction to camera
   o_toCamera = normalize(u_cameraPosition - worldPosition.xyz);

   // texture coordinates to fragment shader
   o_texcoords = i_texcoord0;


   // screen space coordinates of the vertex
   gl_Position = u_projMat * u_viewMat * worldPosition;
}
```

Fragment shader:

```
#version 330

// data from vertex shader
in vec3 o_normal;
in vec3 o_toLight;
in vec3 o_toCamera;
in vec2 o_texcoords;

// texture with diffuese color of the object
layout(location = 0) uniform sampler2D u_diffuseTexture;

// color for framebuffer
out vec4 resultingColor;

////////////////////////////////////////////////////////////

// parameters of the light and possible values
uniform vec3 u_lightAmbientIntensitys; // = vec3(0.6, 0.3, 0);
uniform vec3 u_lightDiffuseIntensitys; // = vec3(1, 0.5, 0);
uniform vec3 u_lightSpecularIntensitys; // = vec3(0, 1, 0);

// parameters of the material and possible values
uniform vec3 u_matAmbientReflectances; // = vec3(1, 1, 1);
uniform vec3 u_matDiffuseReflectances; // = vec3(1, 1, 1);
uniform vec3 u_matSpecularReflectances; // = vec3(1, 1, 1);
uniform float u_matShininess; // = 64;

////////////////////////////////////////////////////////////

// returns intensity of reflected ambient lighting
vec3 ambientLighting()
{
   return u_matAmbientReflectance * u_lightAmbientIntensity;
}

// returns intensity of diffuse reflection
vec3 diffuseLighting(in vec3 N, in vec3 L)
{
   // calculation as for Lambertian reflection
   float diffuseTerm = clamp(dot(N, L), 0, 1) ;
   return u_matDiffuseReflectance * u_lightDiffuseIntensity * diffuseTerm;
}

// returns intensity of specular reflection
vec3 specularLighting(in vec3 N, in vec3 L, in vec3 V)
{
   float specularTerm = 0;

   // calculate specular reflection only if
   // the surface is oriented to the light source
   if(dot(N, L) > 0)
   {
      // half vector
      vec3 H = normalize(L + V);
      specularTerm = pow(dot(N, H), u_matShininess);
```

```
    }
    return u_matSpecularReflectance * u_lightSpecularIntensity * specularTerm;
}

void main(void)
{
    // normalize vectors after interpolation
    vec3 L = normalize(o_toLight);
    vec3 V = normalize(o_toCamera);
    vec3 N = normalize(o_normal);

    // get Blinn-Phong reflectance components
    float Iamb = ambientLighting();
    float Idif = diffuseLighting(N, L);
    float Ispe = specularLighting(N, L, V);

    // diffuse color of the object from texture
    vec3 diffuseColor = texture(u_diffuseTexture, o_texcoords).rgb;


    // combination of all components and diffuse color of the object
    resultingColor.xyz = diffuseColor * (Iamb + Idif + Ispe);
    resultingColor.a = 1;
}
```