LIGHTING

Color and Colorspace

In computer graphics, a color space defines a set of reference colors. It also defines, how combination of these reference colors yield actual colors. This combination of reference colors is represented as n-tuple floating point numbers between 0 and 1. For instance in RGB colorspace, a color is represented as 3 floating point numbers. The floating point numbers represent the intensity (brightness) of respective reference color. A linear colorspace are those colorspace, where if we double the intensity of each component of the reference color, the resulting color is twice as bright.

In computer graphics we most commonly deal with RGB linear colorspace. Note that not only the brightness and color could be represented as a RGB triplet, but the surface absorption could be represented in this way too. If we define surface absorption to be RGB(.25, .5, .75) then that would mean that the surface reflects back 25% of red light, 50% of blue light and 75% of green light. This is also often called surface color.

This is not how reality works however. Color of light is defined by the wavelength of the photon. Our eyes are receptive to certain

range of that wavelength, varying from red to blue. Our eyes are however deceived by the varying intensity of red blue and green to produce huge range of colors. Thus, this RGB model is adopted for it is easier to electronically produce varying RGB intensity than the precise wavelengths of various colors.

HDR

lighting that uses values outside of [0,1] range. At the end of lighting computation we map the value to [0,1] range which is also called tone mapping. HDR allows lighting the scene with greater flexibility. For example, when modeling illumination from a very bright source like sun, we can set its intensity to be greater than 1, so that in comparison to other light sources its contribution would be the greatest.

Gamma correction

Display devices apply certain non-linear function to output color values before displaying it. Thus to be able to accurately see the RGB color value, we ought to apply inverse of that non-linear function to final RGB color output. This is gamma corrected RGB color.

$$\text{Gamma corrected RGB} = \text{Intended linear RGB}^{1/\gamma}$$

where γ is gamma symbol, and is commonly set to 2.2

° Intensity of light shone upon surface:

Intensity of light shone upon a surface is a function of original light's intensity and the angle between surface and light source. The angle is called angle of incedence.

° Diffuse Lighting

Light / Surface interaction where light from light source reflects from surface at many angles instead as a perfect mirror.

° Lambertian reflectance model of diffuse lighting

This model of diffuse lighting assumes that surfaces are ideal diffuse reflector i.e. surfaces reflect light equally in all direction. Diffuse lighting equation in this model is:

$R = D \times I \times \cos\Theta$ where,

R is reflected color
I is light intensity
D is diffuse surface color
Θ is angle of incedence

R doesn't depend on the position of the observer and this is what makes it a lambertian reflectance.

Lighting Model

A lighting Model is an algorithm, a mathematical function, that approximates how a surface interacts with light.

Directional light source

Light emitting objects are called light sources. Here the light source is assumed to be sufficiently distant, relative to the size of the surface such that it appears to surface as if a wall of intensity is evenly distrubuted across surface. The direction towards the light is assumed to be the same for every point on every object on the scene.

Global illumination

Lighting model that considers light bouncing off of surrounding objects that endup illuminating objects that aren't directly facing light source. Through this we are able to model objects that are in the shadow that are still visible to the observer yet less bright than objects that are directly facing the light source.

Ambient Lighting

Ambient lighting is a crude, efficient and useful modeling of global illumination. It assumes every point on the scene is illuminated with certain intensity, from every direction. Thus, it doesn't consider angle of incedence in lighting calculation.

Surface Normal

Mesh geometry is made of triangles and each triangle faces a single direction. This direction is the surface normal of the triangle. If we use this surface normal in our lighting equation then the underlying surface that the mesh is supposed to approximate would appear faceted. Instead each vertex in our triangle would be assigned normal that it would have had on the surface it is approximating. This means while mesh is an approximation, the normal of the vertex is the normal of the actual surface at that point.

Gouraud Shading

Lighting computation is performed at each vertex and then the result is interpolated across the surface of triangle. Per vertex computation as opposed to per fragment computation is what makes gouraud shading efficient.

° Lighting model for computing diffuse lighting from directional light source + ambient lighting, using per vertex normal, per vertex diffuse color and Gouraud shading.

For diffuse lighting:
- Transform normal from model space to camera space.
- Compute cosine of angle of incedence between surface normal and direction towards light
- Multiply directional light intensity by cosine of angle of incedence and multiply that by diffuse surface color

for ambient lighting:
- Multiply diffuse color by ambient light intensity.

Finally, add diffuse and ambient lighting terms. This is done for every vertex in the vertex shader and reflected color is thus computed for every vertex. Fragment shader may simply pass the color as output.

Next page demonstrates vertex shader code in GLSL, burrowed from McKesson's "Learning modern 3D graphics programming"

```
#version 330

layout (location = 0)   in vec3   position;
layout (location = 1)   in vec4   diffuseColor;
layout (location = 2)   in vec3   normal;

smooth out vec4 interpColor;

uniform vec3 dirToLight;
uniform vec4 lightIntensity;
uniform vec4 ambientIntensity;

uniform mat4 modelToCameraMatrix;
uniform mat3 normalModelToCameraMatrix;
uniform mat4 camToClipMatrix;

void main() {
    gl_Position = camToClipMatrix * (modelToCameraMatrix
                * vec4(position, 1.0));
vec3 normCamSpace = normalize(normalModelToCameraMatrix
                * normal);
float cosAngIncedence = dot(normCamSpace, dirToLight);
cosAngIncedence = clamp(cosAngIncedence, 0, 1);

interpColor = ((diffuseColor * lightIntensity *
                cosAngIncedence) +
                (diffuseColor * ambientIntensity));
```

Point Light Source

Point light source is a light source that has position in the world and shines with equal intensity in all direction. When computing diffuse lighting, direction to light source must be computed for every point in the scene as opposed to directional light source where the light direction is assumed to be constant for all points in the scene.

Problems with per vertex diffuse lighting and point light source.

Assume a large triangle surface with a point light in the middle of surface, almost touching it. Angle of incedence between 3 vertices and that light source would be close to 90, therefore the light intensity at each vertex would be pretty low. Since these 3 low intensity light are now interpolated across the surface of triangle, the surface overall would appear dim although the light source sits very close to surface. Expected result would instead be very bright spot near the light source dimming radially across the surface.

Another problem with this lighting model is that relatively complex mesh with varying normals

appear faceted. More photorealistic lighting model for point light sources is to compute lighting equation per fragment (pixel).

Per fragment diffuse lighting

In this lighting technique, lighting computation is performed for every fragment generated from vertices of the mesh primitive as opposed to per vertex computation and color interpolation. Idea is to interpolate vertex normal and direction to point light source across the fragments and do the computation in fragment shader stage.

Problem with interpolating light direction is that, interpolating across the boundaries of primitives, say triangle, is the function of only 2 vertices of the primitive.

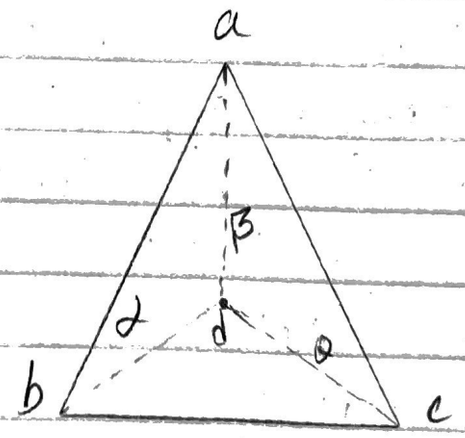

fig1: Interpolating inside triangles' surface

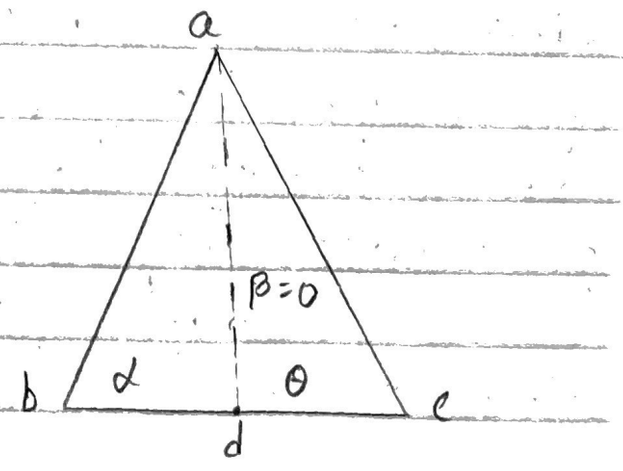

fig2: Interpolating at boundary of triangles' surface

In fig2 above, vertex attribute d is the result of linear interpolation of just b and c and a's contribution is 0. With light direction as vertex attribute, specially in the case of greatly varying light directions, lighting around boundary would appear incorrect.

Thus, vertex position is interpolated instead, which interpolates correctly, and the lighting direction is then computed in the fragment shader.

Similarly if vertex normal varied greatly among the vertices of triangle, then interpolating normal would give similar problems. But unlike the solution to interpolating light direction like described above, there doesn't exist such simple solution. Instead when modeling complex shapes, and when photorealism is the goal, mesh are encoded with normal at many points along the surface not just at vertices.

Light Attenuation

In physical world, as light source travels further away from the surface, the intensity of light upon the surface falls off proportionally. This is light attenuation. When considering light attenuation in our lighting model, we multiply light intensity with attenuation factor. Attenuation factor is given by:

$$\text{Attenuation Factor} = \frac{1}{1 + k * r^2} \quad \text{where}$$

'r' is distance between surface point and light source and K is attenuation constant that we can tweak to our liking.

The lighting equation (diffuse + ambient) with attenuation factor considered, thus becomes:

Attenuated Intensity = light Intensity * (1 / (1 + attenuation constant * r));

R = (diffuse Color * attenuated Intensity * (cosAng Incedence) + (diffuseColor * ambient Intensity);

Specular Reflection

Shiny surfaces reflect light strongly in opposite direction from the angle of incedence. This is specular reflection. A mirror is a perfect specular reflector. Smooth plastic like surface tend to have bright spot, caused by direct illumination from a light source. These are specular highlights. Position of the highlight on the surface changes with position of observer (view direction) as well as with the position of light source (light direction).

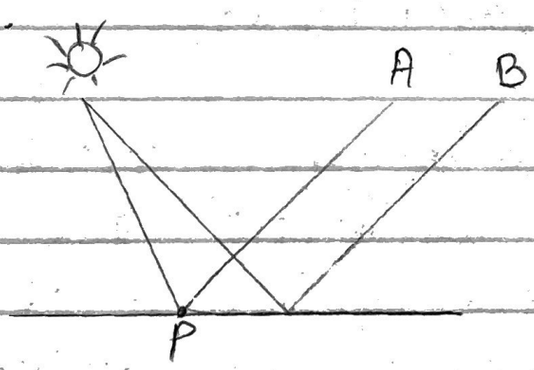

fig: Reflection

Highlight at position 'P' is observed from A but not from B.

Microfacets in the surface

If a surface were perfectly smooth, specular highlight from a point ..... light would be infinetly small. Surfaces that seem smooth to the eyes, can be rough on closer examination. This roughness can be modeled by sharp edges on the microscopic

level. These are microfacets. Thus a more realistic specular highlights on surfaces like would and plastic could be modeled by statistical distrubution of orientation of microfacets on the surface. These highlights are formed because, even though surface normals may not be oriented to directly reflect light to the viewer, some microfacets may still be oriented to reflect a portion of light. Smoother surfaces tend to have smaller, brighter highlights compared to rougher surfaces.

Phong specular lighting Model

It doesn't deal with microfacet distrubutions. Instead it is a hack that looks decent, and is cheap to compute. It states that light reflected in direction of viewer varies based on angle between view direction and direction of perfect reflection

$$\text{Phong term} = (\vec{V} \cdot \vec{R})^s$$

Specular reflection = light Intensity * specular color * Phong term

Phong term is a cosine of angle between view direction and reflection direction and this creates Circular ish area on the surface. Raising 's' term in the power, makes phong term smaller, and thus will model a smaller highlight (smoother surface).

Specular reflection color could be added to diffuse and Ambient terms to result final output color.

Problem with Phong Specular lighting Model

When the exponent in the phong term is a small number, a ring like area is visible on the surface, and any region outside that ring would be sharply dark, i.e. no specular contribution. This dark region is because the reflection direction and view direction are more than 90 degree apart. The phong term $(\vec{V} \cdot \vec{R})$ becomes negative in this case and is clamped to 0, and thus no specular reflection. Under microfacet model however, there is still some chance that some microfacet are oriented toward camera and therefore there be atleast some specular contribution.

Blinn Phong specular lighting Model

This is a modified phong model that solves the above problem with phong Model. It requires computing half way between view direction and light direction also called half angle vector.

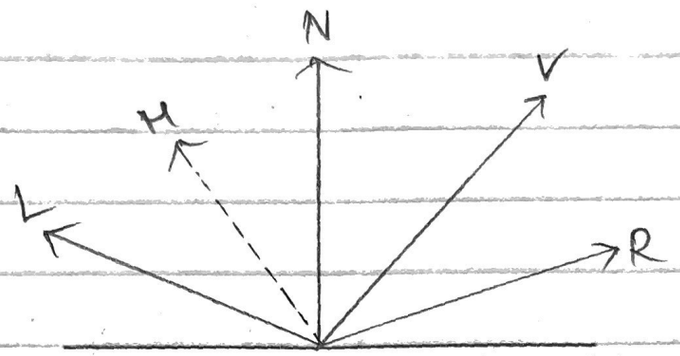

Fig: Half Angle Vector (H)

$$\vec{H} = \frac{\vec{L} + \vec{V}}{\|\vec{L} + \vec{V}\|}$$

When view direction is perfectly aligned with with reflection direction, half angle vector is perfectly aligned with normal. So, instead of comparing reflection vector to viewing direction, Blinn Phong model compares half angle vector to surface normal.

$$\text{Blinn term} = (\vec{H} \cdot \vec{N})^s$$

The angle between $\vec{H}$ and $\vec{N}$ is always less than 90, and thus it doesn't have that problem of Phong Model.