Section Notes on Surface Shading and Assignment 3

Niels Joubert

2008-09-16

1 Local Surface Shading

We compute the **local shading values** of a point on a surface by considering the lights, surface properties and viewer in the scene. We can do this because **light is linear**, and we consider only local phenomenon in shading calculations. We approximate real world lighting using one of the following methods:

- **Full Bi-Directional Reflectance Distribution**
  Calculate fraction of incoming light that reaches the viewer for with $\rho = \rho(\theta_{viewer}, \theta_{incidence}, \lambda_{in}, \lambda_{out})$ where $\theta_{incidence}$ is the angle of incoming light w.r.t. the surface normal, and θ_{viewer} is the angle of the viewer w.r.t. the surface normal.

  Good: ____________________

  Bad: ____________________

- **Component-wise Bi-Directional Reflectance Distribution**
  Calculate the fraction of incoming light that reaches the viewer for each color component rather than all λ_{in} and λ_{out}:

$$\begin{aligned}\rho_r &= \rho_r(\theta_{viewer}, \theta_{incidence}) &= \rho(\theta_{viewer}, \theta_{incidence}, K_r)\\ \rho_g &= \rho_g(\theta_{viewer}, \theta_{incidence}) &= \rho(\theta_{viewer}, \theta_{incidence}, K_g)\\ \rho_b &= \rho_b(\theta_{viewer}, \theta_{incidence}) &= \rho(\theta_{viewer}, \theta_{incidence}, K_b)\end{aligned}$$

  Where K_r gives you the **material properties** for the red component, and so forth.

  Good: ____________________

  Bad: ____________________

- **Extended BRDFs**
  Allow us to model complex light behaviour, although this might not technically be local shading.

  Good: ____________________

  Bad: ____________________

All of these models assume that we can look up a ρ value for a given θ_{viewer} and $\theta_{incidence}$, which translates to a table lookup in most cases. Thus we build an approximate model of the BRDF that we use for shading calculations.

1.1 Approximate BRDF

The lectures and the book use Phong shading loosely. What is important to understand is that we approximate the BRDF as the sum of different light terms, which is an approximation of the BRDF:

$$\rho = k_a \ + \ k_d \mathbf{I} \max(\hat{\mathbf{I}} \cdot \hat{\mathbf{n}}, 0) \ + \ k_s \mathbf{I} \max(\hat{\mathbf{r}} \cdot \hat{\mathbf{v}}, 0)^p \tag{1}$$

$\mathbf{I} = (r, g, b)$ is the color of the incoming light.
$\hat{\mathbf{I}}$ is the incidence vector, supplying the angle of incoming light.
$\hat{\mathbf{n}}$ is the surface normal vector.
$\hat{\mathbf{v}}$ is the vector to the viewer.
$\hat{\mathbf{r}}$ is the reflectance vector, supplying the angle of reflected light.
k_a, k_d and k_s, each consisting of (r, g, b), is the ambient, diffuse and specular properties of the material.

1.1.1 Ambient Lighting

Light reflects around a room, illuminating objects uniformly from all sides. We model this effect by simply setting each object to have a default ambient color, as if some uniform illumination is supplying color to the object.

1.1.2 Diffuse Lighting

We assume that surfaces are **Lambertian**
$\Longrightarrow$ they obey *Lambert's Cosine Law*
$\Longrightarrow$ color $c \propto \cos(\theta_{incidence})$ and $c \propto \hat{\mathbf{n}} \cdot \hat{\mathbf{I}}$

Thus, this states that the color of a point on a surface is independent of the viewer, and depends only on the angle between the surface normal and the incidence vector (the direction from which light falls on the point). We want the actual color to depend on both the color of the light source $\mathbf{I} = (r, g, b)$ and the material properties $k_d = (r, g, b)$

$$\rho_{\text{diffuse}} = k_d \mathbf{I} \ \max(\hat{\mathbf{n}} \cdot \hat{\mathbf{I}}, 0) \tag{2}$$

Where ρ_{diffuse} is an (r,g,b) color value. This relation also implies that the most reflection you can get in a single direction on a matte surface is 1 / (area of unit hemisphere). (Why?)
$\hat{\mathbf{n}}$ needs to be calculated for the surface itself.
$\hat{\mathbf{I}}$ needs to be calculated by the light according to what type of light it is.

1.1.3 Specular Lighting

The Phong illumination model states that there is a bright, mirror-like reflection of the light source on the surface. This effect depends on where the viewer is. The effect is the strongest when the viewer vector and reflectance vector is parallel, and it depends on $\mathbf{I} = (\mathbf{r}, \mathbf{g}, \mathbf{b})$ the incidence color and $k_s = (r, g, b)$, the material properties.

$$\rho_{\text{specular}} = k_s \mathbf{I} \max(\hat{\mathbf{r}} \cdot \hat{\mathbf{v}})^p \tag{3}$$

p is the **roughness** of the material - it affects how small the specular highlight is.
$\hat{\mathbf{v}}$ is calculated between the point on the surface and the viewer position.
$\hat{\mathbf{r}}$, the reflectance vector, is calculated using $\hat{\mathbf{I}}$ and $\hat{\mathbf{n}}$:

$$\hat{\mathbf{r}} = -\hat{\mathbf{I}} + 2(\hat{\mathbf{I}} \cdot \hat{\mathbf{n}})\hat{\mathbf{n}} \tag{4}$$

1.2 Applying local shading to an object

So far we have only considered a single point on an object. To shade an object as a whole, we have to take into consideration that our geometry are made of flat, discrete pieces. We calculate shading over a region using one of the following methods:

1.2.1 Flat Shading

What is flat shading? ____________________

____

1.2.2 Gouraud Shading

What is gouraud shading? ____________________

____

1.2.3 Phong Shading

What is phong shading? ____________________

____

1.3 Lights

A basic light is characterized by a position in space $\hat{\mathbf{P}}$ and a color intensity $\mathbf{I}$. This allows us to calculate $\hat{\mathbf{I}}$ for points on a surface. Lights differ primarily in how their $\hat{\mathbf{I}}$ vector and $\mathbf{I}$ color is calculated.

1.3.1 Point Light

What is a point light? ____________________

How do you calculate its $\hat{\mathbf{I}}$? ____________________

1.3.2 Directional Light

What is a directional light? ____________________

How do you calculate its $\hat{\mathbf{I}}$? ____________________

1.3.3 Spot Light

A spot light is a light that drops off in intensity sharply beyond a predefined border.
A spotlight's $\hat{\mathbf{I}}$ is calculated the same as a point light's $\hat{\mathbf{I}}$, but if $\hat{\mathbf{I}} \cdot \hat{\mathbf{X}} > x$ then $\mathbf{I} = 0$ where x is the size of the spot, and $\hat{\mathbf{X}}$ is the direction the spot points in.

1.4 More on lighting and shading

Lighting is a complete department in itself at most studios - we have studied the building blocks of most lighting tools. Some topics that I have not covered in these notes are:

- Falloff (let $\mathbf{I}$ be proportional to the distance from the light to the object).
- Anisotropy (make the roughness term p depend on the parameterization of the object).
- Texturing (eg. Bump Mapping is simply perturbing the surface normals)

2 Shading Assignment

It would be extra-helpful if you write your shader in such a way that you can reuse the code for your ray-tracing assignment. We currently work with *procedural shaders* - they take the world, run some code, and give you color values.

Aside: **How do you draw a shaded 3d sphere pixel-by-pixel on a 2d surface?**

What is the equation for a sphere in 3 dimensional space?

Using this, at any point along x, how do you find the range of y on the sphere?

Using this, how do you calculate the z coordinate?

How do you find the surface normal from these 3 coordinates?

2.1 A possible class structure

We need Lights, a Sphere, a Shader (that can be part of the sphere) and a Viewer. We need vectors to store position and direction, and we need colors that add and multiply component-wise. To complete this class structure for the project, ask yourself:

- What is the interface that Lights provide?
- If Objects are the drivers of the shading routine, what information do they pass to the shader to get a color back for each point on their surfaces?

Example C++ code: (There might be errors in this, it was typed up *very early in the morning*)
Lights.h:

```
class Light {
public:
    Light();                //Default Constructor
    virtual ~Light();       //Default Destructor
    Light(float x, float y, float z, float r,float g,float b);
    const Color getColor(); //Return value may not be modified.
    virtual Vector3d getIncidence(Vector3d & point) = 0; //Abstract function
private:
    Color illumination;
    Vector3d position;
}
class PointLight: public Light { //Child of Light
    PointLight(float x,float y,float z,float r,float g,float b);
    Vector3d getIncidence(Vector3d & point); //Implement parent function.
}
```

Lights.cpp:

```
Light::Light() {};              //Default Constructor
Light::Light(float x, float y, float z, float r,float g,float b) {
    this->x = x; this->y = y; this->z = z; //'this' is a pointer
};
const Color Light::getColor() { return illumination; //data in parent. };
PointLight::PointLight(float x,float y,float z,float r,float g,float b):
    Light(x,y,z,r,g,b) { }; //Parent initialized by : Light(...)
Vector3d PointLight::getIncidence(Vector3d & point) {
        Vector3d incidence;
        Vector3d light_pos = pos.getAbsolutePosition(multiplier);
        incidence.calculateFromPositions(&point,&light_pos);
        return incidence;
};
```