CSC317/418 Computer Graphics

© Tingfeng Xia

Winter Term, 2021

Information

- Starting winter term 2021, Computer Graphics will be switching from its previous course code CSC418 to CSC317.
- Course webpage: https://github.com/karansher/computer-graphics-csc317
- This note contains figures from Prof. Karan Singh's lecture slides. It also might contain figures by Marschner and Shirley and David Levin.
- Textbook: Shirley, P., & Marschner, S. (2009). *Fundamentals of Computer Graphics (Third Edition)*.

Contents

1 Raster Image

This section will be added later.

2 Ray Casting

This section will be added later.

3 Ray Tracing

3.1 Light and Surfaces

There are two types of lights, namely directional and point light sources.

- The directional light has its light direction independent of the object, this typically happens when the light is very far away. (e.g. the sun)
- On the other hand, point light are such that the direction of light depends on position of object relative to light. Think of this as a light bulb in a room.

3.2 Shading

The goal of shading is to compute the light reflected toward camera. As an algorithm it expects the following inputs: eye direction, light direction for each of many lights, surface normal, and surface parameters such as color and shininess.

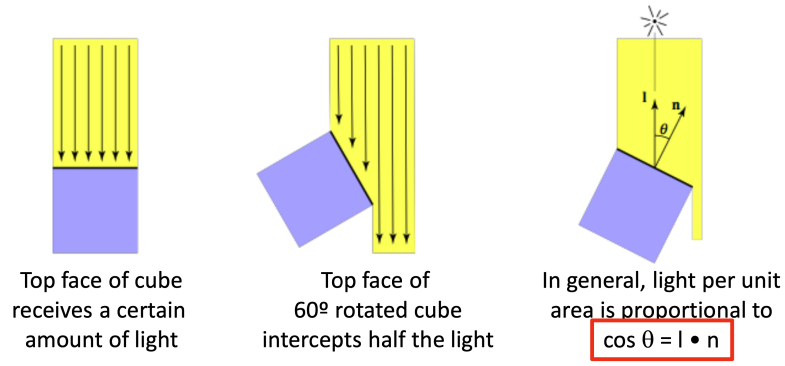


Figure 1: Illustration of Lambert's cosine law.

The Surface Normal at a hit point can be computed, depending on the type of specification

- Polygon normal: cross product of two non-collinear edges,
- Implicit surface normal $f(p) = 0$: $gradient(f)(p)$
- Explicit parametric surface $f(a, b)$:

$$\frac{\partial f(s,b)}{\partial s} \times \frac{\partial f(a,t)}{\partial t} \tag{3.1}$$

3.3 Light Falloff

The light falloff aims to model the concept of diminishing light intensity based on distance. Suppose the light source has intensity of I, then, at a point that is r (Euclidean Distance) away from the light source, the light intensity from that particular light source would be

$$\text{Intensity}(I, r) = I/r^2 \tag{3.2}$$

Clearly, when we are at the light source, $r = 0$ and we attain the max intensity.

3.4 Diffuse Reflection

In the case of diffuse reflection, light are scattered uniformly in all directions, i.e. he surface color is the same for all viewing directions. The amount of light captured by a surface obeys Lamber's cosine law. (We call this Lambertian surface.) Figure 1 illustrates the relationship.

3.5 Lambertian Shading

The Lambertian Shading is independent of view direction. Let's call $L_d :=$ diffusely reflected light, $k_d :=$ diffuse coefficient, and $I :=$ illumination from source, then

k_d could be separate for three color channels.

$$L_d = k_d \circ \left(I/r^2\right) \max(0, \mathbf{n} \cdot \mathbf{l}) \tag{3.3}$$

This produces a matte appearance.

3.6 Shadows

Surface is only illuminated if nothing blocks its view of the light. With ray tracing it's easy to check if a point in the scene is in shadow, all you have to do is just shoot a ray from the point[3.1] to the light and intersect it with the scene.

3.7 Multiple Light Sources

- Important to fill in black shadows,
- and just loop over lights, add contributions
- Ambient shading
  - Black shadows are not really right,
  - easy solution: dim light at the camera, so everything we see must be lit up (might be dim, but not black)
  - alternative: add a constant ambient color to the shading.

3.8 Ambient Shading

Ambient shading are those that does not depend on anything, we add a constant color to account for disregarded illumination and fill in black shadows. Let's call $L_a :=$ reflected ambient light, $k_a :=$ ambient coefficient, and I_a the raw light intensity, then

This k_a could again be separate for different color channels

$$L_a = k_a \circ I_a \tag{3.4}$$

3.9 Algorithm Template

Let's present the algorithm template for images with multiple light sources here

```
shade(ray, lights, point, normal) {
    result = ambient;
    for light in lights {
```

[3.1] This "point" means the actual location in 3d of what a pixel on the image projects to.

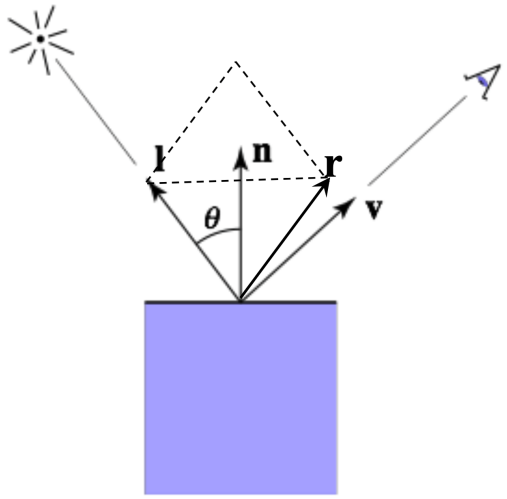


Figure 2: Mirror reflection illustration

```
        I = light.pos - position;
        shadowray = (point + eps * normal, I);
        if !scene.intersection(result, shadowray) {
            it = surface.k * light.intensity * max(0, normal.dot(I));
            result += surface.color * it;
        }
    }
    return result;
}
```

Do notice that we have `shadowray = (point + eps * normal, I);`, and the eps is there to prevent floating point numerical errors. If we don't do that it is possible a ray would intersect immediately with the surface, and produce unusable images.

3.10 Mirror Reflection

Now we've talked about diffuse reflections (matte surfaces), let's see the case of mirror reflection. We discuss the case of imperfect mirror here, where the intensity depends on view direction - reflects incident light from mirror direction. Figure 2 illustrates the scenario. The reflected ray vector is

$$\mathbf{r} = 2(\mathbf{n} \cdot \mathbf{l})\mathbf{n} - \mathbf{l} \tag{3.5}$$

3.11 Phong Specular Shading

With a perfect mirror, only when $\mathbf{r} = \mathbf{v}$ in Figure 2 would the eye see light. In real world, this rarely is the case and we wish to model a common imperfect mirror where there will be some light being reflected to directions around $\mathbf{r}$. Phong specular shading aims to model reflections from this imperfect mirror - the intensity would depend on the view direction, and is bright near mirror configuration:

$$k_s \circ I_s(\mathbf{v} \cdot \mathbf{r})^{shiny} \tag{3.6}$$

where k_s is yet another set of coefficient guarding three channels, I_s is the three channel intensity (distance decay taken into account already), and *shiny* is a constant hyper-parameter raised to exponent dictating how fast the $(\mathbf{v} \cdot \mathbf{r})$ diminishes to 0. A large *shiny* exponent will make the surface more glossy, while a smaller one will make the surface more matte-looking.

3.12 Blinn(-Phong) Specular Shading

Blinn specular shading is just another formulation.[3.2] Figure 3 illustrate what we want to calculate. Let's call the point of reflection $\mathbf{o}$, then in Phong model, we used *angle***rov** to model the angle and here in Blinn we will be using $\angle$**hon** instead. Notice that however, we do need to calculate an extra vector $\mathbf{h}$ that is the normalized sum of $\mathbf{v}$ and $\mathbf{l}$, i.e.

$$\mathbf{h} = \frac{\mathbf{v} + 1}{\|\mathbf{v} + 1\|} \tag{3.7}$$

The specular reflected intensities can be calculated

$$L = k_s \circ I \max\{0, \mathbf{n} \cdot \mathbf{h}\}^p \tag{3.8}$$

where k_s is our guarding coefficients, I is the intensity at the point of reflection (distance decay taken into account already), p is something similar to *shiny* from before. Again, a larger p means a shinier surface and a smaller one means more matte-looking.

3.13 Local Illumination

- Usually include ambient, diffuse, Blinn-Phong in one model

$$\begin{aligned} L &= L_a + L_d + L_s \\ &= k_a I_a + k_d \left(I/r^2\right) \max(0, \mathbf{n} \cdot \mathbf{l}) + k_s \left(I/r^2\right) \max(0, \mathbf{n} \cdot \mathbf{h})^p \end{aligned} \tag{3.9}$$

[3.2] This very simple and widely used model for specular highlights was proposed by Phong (Phong, 1975) and later updated by Blinn (J. F. Blinn, 1976) to the form most commonly used today.

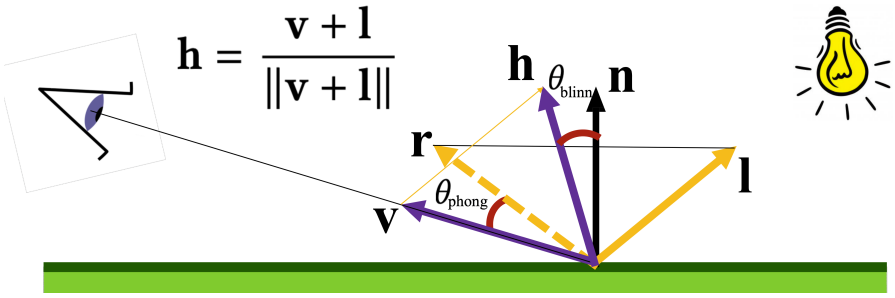


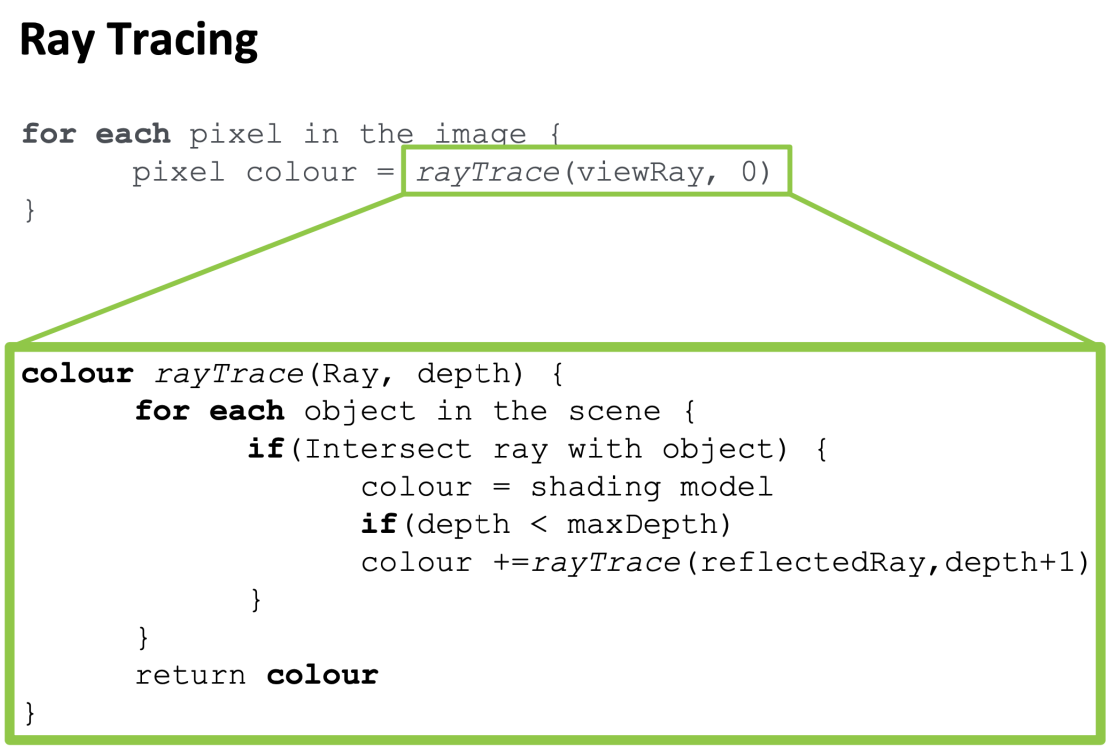
Figure 3: Blinn Specular Shading and Phong Specular Shading.

- The final result is the sum over many lights

$$L = L_a + \sum_{i=1}^{N} \left[(L_d)_i + (L_s)_i \right] \tag{3.10}$$

$$= k_a I_a + \sum_{i=1}^{N} \left[k_d \left(I_i / r_i^2 \right) \max\left(0, \mathbf{n} \cdot \mathbf{l}_i\right) + k_s \left(I_i / r_i^2 \right) \max\left(0, \mathbf{n} \cdot \mathbf{h}_i\right)^p \right] \tag{3.11}$$

3.14 Ray Tracing Template



3.15 Refraction - Snell's Law

Consider a surface with normal $\mathbf{n}$, with incoming light vector $-\mathbf{l}$ (so $\mathbf{l}$ is pointing into the surface), call $\theta_l := \angle\mathbf{nol}$. Suppose the refracted light vector is named $\mathbf{t}$ with $\theta_t := \angle\mathbf{to}(-\mathbf{n})$, then Snell's Law states that

$$c_l \sin\theta_l = c_t \sin\theta_t \tag{3.12}$$

We can derive that our unknown of interest $\mathbf{t}$ is such that

$$\mathbf{t} = -\frac{c_l}{c_t}\mathbf{l} + \frac{c_l}{c_t}\cos(\theta_l)\mathbf{n} - \cos(\theta_t)\mathbf{n} \tag{3.13}$$

To take refractions into account, we just need to slightly modify the ray tracing template presented previously. Namely, we will need to add

```
...
colour += raytrace(reflectedRay, depth+1)
colour += raytrace(refractedRay, depth+1)
```

3.16 Ray Tracing: Pros and Cons

Ray tracing provides a unifying framework for all of the following

- Hidden surface removal,
- shadow computation,
- reflection of light,
- refraction of light,
- globular specular interaction

but at the same time it presents some deficiencies

- intersection computation time can be long, (solution: bounding volumes, next time)
- recursive algorithm can lead to exponential complexity, (solution: stochastic sampling)
- ignores light transport mechanisms involving diffuse surfaces.

3.17 Caustics

3.18 Radiosity

3.19 The Rending Equation

$$L_o(\mathbf{x}, \mathbf{w}) = L_e(\mathbf{x}, \mathbf{w}) + \int_\Omega f_r\left(\mathbf{x}, \mathbf{w}', \mathbf{w}\right) L_i\left(\mathbf{x}, \mathbf{w}'\right) \left(\mathbf{w}' \cdot \mathbf{n}\right) \mathrm{d}\mathbf{w}' \tag{3.14}$$

where

$$L_o(x, \vec{w}) = \text{outgoing light at } \mathbf{x} \text{ direction } \mathbf{w} \tag{3.15}$$

$$L_e(x, \vec{w}) = \text{emitted light at position } \mathbf{x} \text{ and direction } \mathbf{w} \tag{3.16}$$

$$\int_\Omega \ldots \mathrm{d}\mathbf{w} = \text{reflected light at position } \mathbf{x} \text{ and direction } \mathbf{w} \tag{3.17}$$

where

$$f_r\left(\mathbf{x}, \mathbf{w}', \mathbf{w}\right) = \text{BRDF: a function describing how light is reflected at an opaque surface} \tag{3.18}$$

$$L_i\left(\mathbf{x}, \mathbf{w}'\right) = \text{the incoming light from all directions} \tag{3.19}$$

3.20 Soft Light