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

An image is a distribution of light energy on a 2D film. Digital images today are represented as rectangular arrays of pixels.

1.1 Raster Devices

1.1.1 Raster Displays

These are displays that are comprised of a certain number of pixel units. Each of these pixel units contains three subpixels - red, green, and blue. When we say a display is 1920×1080, we mean that the display has that many pixels in the two directions, this also means that we have, at subpixel level, $1920 \times 1080 \times 3$ "elements" on the display.

1.1.2 Raster Input Devices

The raster input filter we will introduce is Bayer Filter. In each pixel of a Bayer Filter, there is only one color - either red, green, or blue. Due to the nature of human eyesight, we are more sensitive to the color green. Hence, the Bayer Filter has twice green elements as compared to red and blue.[1.1]

1.1.3 Data Types for Raster Images

Storage for 1024^2 image (1 megapixel)

- bitmap: 128KB,
- grayscale 8bpp: $1024^2 \times 8/8/1024^2$ =1MB,
- grayscale 16bpp: $1024^2 \times 16/8/1024^2$ =2MB,
- color 24bpp: 3MB, [1.2]
- floating point high dynamic range color: 12MB

1.1.4 Gamma Correction

On a display, the intensity is non-linear with respect to input intensity. The Gamma correction relationship is formularized as

$$\text{display intensity} = \text{max display intensity} a^{\gamma} \tag{1.1}$$

where a is the amplitude from the image in the range in $[0, 1]$. The γ is what we can set, usually in display settings. To set the right gamma correction, we need to find image amplitude that is equal to half of the display's brightness, then

$$\frac{1}{2} = a^{\gamma} \quad \Longrightarrow \quad \gamma = \frac{\ln .5}{\ln a} \tag{1.2}$$

1.1.5 Modelling Transparency

Simple RGB is great, but that only gets us solid colours, meaning that we cannot model transparency information with it. RGBA addresses this by simply appending an alpha channel variable to the end of the original triplet, making it a RGBA quadruple. We will also need this for the Porter/Duff Composition described below.

[1.1] Patterns to the Bayer Filter may vary. The one presented in slides is BGGR.
[1.2] 24bpp means 24 bits per pixel, so 8 bits for each color: 0 - 255

1.2 Image Composition

Compositing is about layering images on top of one another. But there are many ways to do the composition, i.e. which image goes at the top and how you handle places with transparency. The Porter/Duff Composition is a unified approach for this problem.

Suppose we have

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.

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 $f(a, b)$:

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

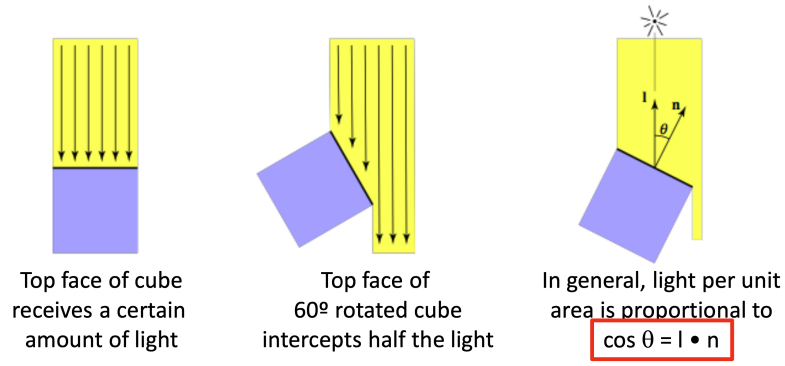

Figure 1: Illustration of Lambert's cosine law.

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

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

This k_a could again be separate for different color channels

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 {
        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;
}
```

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

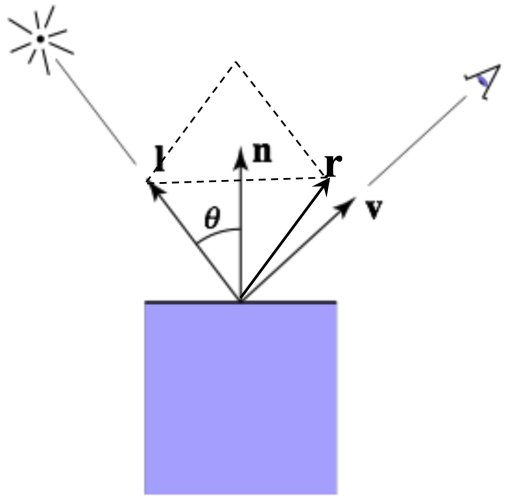

Figure 2: Mirror reflection illustration

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:

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

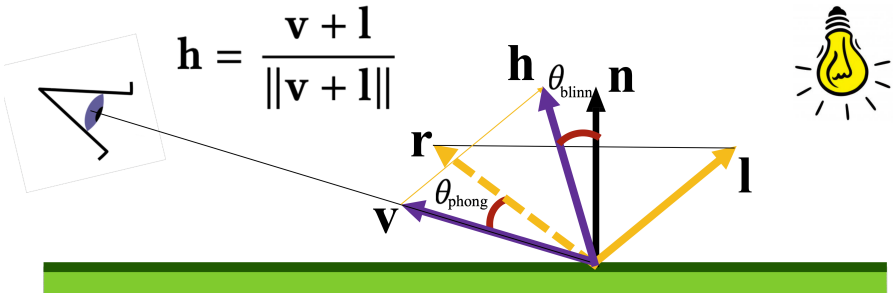

Figure 3: Blinn Specular Shading and Phong Specular Shading.

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 **o**, then in Phong model, we used *angle***rov** to model the angle and here in Blinn we will be using ∠**hon** instead. Notice that however, we do need to calculate an extra vector **h** that is the normalized sum of **v** and **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.

- The final result is the sum over many lights

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

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

3.14 Ray Tracing Template

Ray Tracing

```
for each pixel in the image {
      pixel colour = rayTrace(viewRay, 0)
}
```

```
colour rayTrace(Ray, depth) {
      for each object in the scene {
            if(Intersect ray with object) {
                  colour = shading model
                  if(depth < maxDepth)
                  colour +=rayTrace(reflectedRay,depth+1)
            }
      }
      return colour
}
```

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

4 Spatial Data Structures - Bounding Volume Hierarchy

In many applications of computer graphics, it is crucial that we can quickly perform queries on a set scene. However, the brute force solution may turn out to be computationally too costly. In a remedy to this problem, we introduce the idea of spatial data structures. These structures partition spaces into smaller ones: (following characterizations are from the text book (Shirley, P., & Marschner, S., 2009).)

- Structures that group objects together into a hierarchy are object partitioning schemes: objects are divided into disjoint groups, but the groups may end up overlapping in space.
- Structures that divide space into disjoint regions are space partitioning schemes: space is divided into separate partitions, but one object may have to intersect more than one partition.
- Space partitioning schemes can be regular, in which space is divided into uniformly shaped pieces, or irregular, in which space is divided adaptively into irregular pieces, with smaller pieces where there are more and smaller objects.

4.1 (Axis Aligned) Bounding Boxes

To derive, let's start with the 2D case. Suppose we have a bounding box described by four lines,

$$x = x_{\min}, \qquad \text{the lower horizontal line,} \tag{4.1}$$

$$x = x_{\max}, \qquad \text{the upper horizontal line,} \tag{4.2}$$

$$y = y_{\min}, \qquad \text{the left vertical line,} \tag{4.3}$$

$$y = y_{\max}, \qquad \text{the right vertical line.} \tag{4.4}$$

Then, consider the ray $x = x_e + tx_d, y = y_e + ty_d$, where we see that only when t satisfies the four constraints described in Figure 4 will the ray hit the box. Arithmetically, we

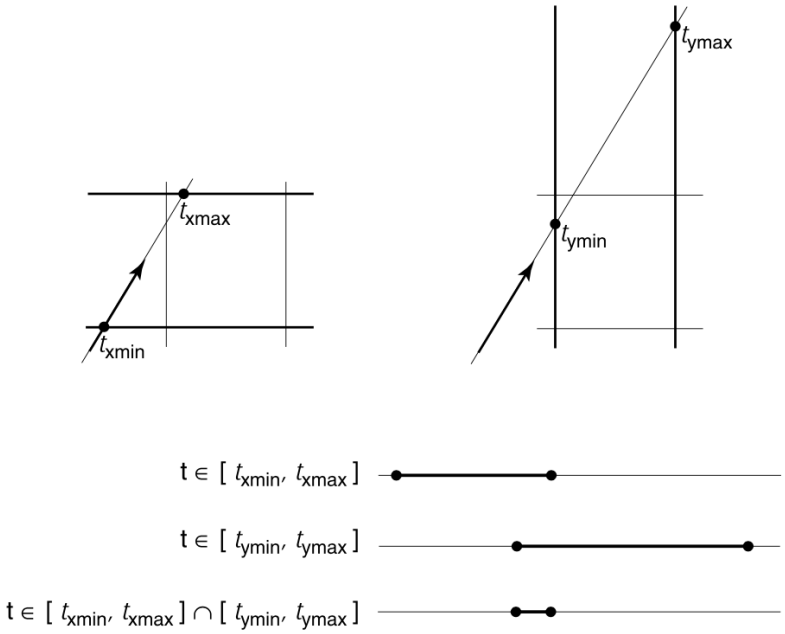

Figure 4: 2D Bounding Box Hit Criterion

calculate[4.1]

$$t_{x\,\min} = x_d \geq 0\,?\,(x_{\min} - x_e)/x_d : (x_{\max} - x_e)/x_d \tag{4.5}$$
$$t_{x\,\max} = x_d \geq 0\,?\,(x_{\max} - x_e)/x_d : (x_{\min} - x_e)/x_d \tag{4.6}$$
$$t_{y\,\min} = y_d \geq 0\,?\,(y_{\min} - y_e)/y_d : (y_{\max} - y_e)/y_d \tag{4.7}$$
$$t_{y\,\max} = y_d \geq 0\,?\,(y_{\max} - y_e)/y_d : (y_{\min} - y_e)/y_d \tag{4.8}$$

Then if $t_{x\,\min} > t_{y\,\max} \vee t_{y\,\min} > t_{x\,\max}$ then the ray missed the bounding box. Otherwise, there is a hit. [4.2]

4.1.1 Handling Divide by Zero

There is yet another concern that needs to be addressed in our formulation: divide by zero errors when $x_d = \pm 0$ or $y_d = \pm 0$. The detailed derivations can be found on pages 281 - 282 of the text book. We note the results here:

$$a = 1/x_d, b = 1/y_d \tag{4.9}$$

[4.1] Note that this solution does not address the divide by zero error discussed in the following section.

[4.2] This if statement condition could be derived from Figure 4: when the two intervals have no intersection, then we have a miss.

If $(a \geq 0)$ **then**

$$t_{x\,\min} = a(x_{\min} - x_e); \quad t_{x\,\max} = a(x_{\max} - x_e) \tag{4.10}$$

else

$$t_{x\,\min} = a(x_{\max} - x_e); \quad t_{x\,\max} = a(x_{\min} - x_e) \tag{4.11}$$

If $(b \geq 0)$ **then**

$$t_{y\,\min} = b(y_{\min} - y_e); \quad t_{y\,\max} = b(y_{\max} - y_e) \tag{4.12}$$

else

$$t_{y\,\min} = b(y_{\max} - y_e); \quad t_{y\,\max} = b(y_{\min} - y_e) \tag{4.13}$$

Then, again, if $t_{x\,\min} > t_{y\,\max} \vee t_{y\,\min} > t_{x\,\max}$ then the ray missed the bounding box. Otherwise, there is a hit. [4.3]

4.1.2 Handling 3D Bounding Boxes

This is almost exactly the same as what we presented in the previous sub-subsection. We calculate a extra c for the z axis: $c = 1/z_d$

If $(c \geq 0)$ **then**

$$t_{z\,\min} = c(z_{\min} - z_e); \quad t_{z\,\max} = c(z_{\max} - z_e) \tag{4.14}$$

else

$$t_{z\,\min} = c(z_{\max} - z_e); \quad t_{z\,\max} = c(z_{\min} - z_e) \tag{4.15}$$

However, this time the return conditions requires a bit more calculations

$$lmin = \max\{t_{*\,\min}\} \quad \text{is the largest of all min} \tag{4.16}$$

and

$$smax = \min\{t_{*\,\max}\} \quad \text{is the smallest of all max} \tag{4.17}$$

Then, if $smax < lmin$, the ray missed the box; otherwise, there is a hit.

[4.3] This if statement condition could be derived from Figure 4: when the two intervals have no intersection, then we have a miss.