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] 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

1.1.5 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.

1.2 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[1.3] is a unified approach for this problem.

Suppose we have two images source and destination. We we try to combine these two images, each pixel of the result can be divided into four regions: Both, Source, Dest, and Neither. The formula for this operation is a linear combination of contents of the four regions, where the weights are the ares of each region:

$$A_{src}[s] + A_{dest}[d] + A_{both}[b] \tag{1.3}$$

where

- $[s]$ is either 0 or the color of the source pixel,
- $[d]$ is either 0 or the color of the destination pixel,
- $[b]$ is either 0, the color of the source pixel or the color of the destination pixel.

With the alpha channel being interpreted as coverage, the areas are given by these formulas:

$$A_{src} = \alpha_s(1 - \alpha_d) \tag{1.4}$$
$$A_{dst} = \alpha_d(1 - \alpha_s) \tag{1.5}$$
$$A_{both} = \alpha_s \alpha_d \tag{1.6}$$

The alpha channel of the result is computed in a similar way:

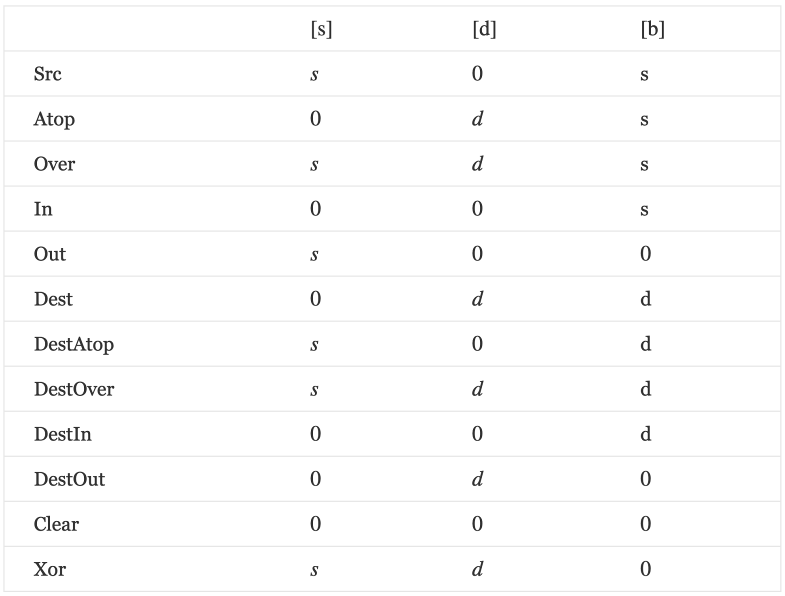
$$A_{src}[as] + A_{dest}[ad] + A_{both}[ab] \tag{1.7}$$

where

- $[as]$ and $[ad]$ are either 0 or 1 depending on whether the source and destination regions are present, and
- $[ab]$ is 0 when both regions is blank (transparent), and 1 otherwise.

The Porter/Duff operations table in Figure 1 shows all 12 possible scenarios.

[1.3] http://ssp.impulsetrain.com/porterduff.html



	[s]	[d]	[b]
Src	*s*	0	s
Atop	0	*d*	s
Over	*s*	*d*	s
In	0	0	s
Out	*s*	0	0
Dest	0	*d*	d
DestAtop	*s*	0	d
DestOver	*s*	*d*	d
DestIn	0	0	d
DestOut	0	*d*	0
Clear	0	0	0
Xor	*s*	*d*	0

Figure 1: Porter/Duff operators table.

2 Ray Casting

In introductory Ray Casting, we consider scenes with no lighting whatsoever, and we consider rays originating from the eye point (camera), through the virtual image plane, and finally hit the objects in the scene. Our goal could be summarized with the following pseudocode,

```
for each image pixel {
  generate a ray
  for each scene object {
    if ray intersects object
      then do something. e.g. set pixel color
  }
}
```

2.1 Ray

We introduce the following parametrization of a ray

$$p(t) = \mathbf{e} + t(\mathbf{s} - \mathbf{e}) \tag{2.1}$$

where $\mathbf{e}$ is the eye point, $\mathbf{s}$ is a known point along the path and $p(t)$ is a parametrization of "point along the ray". Notice that under our parametrization, we have $p(0) = \mathbf{e}$ and

$p(1) = \mathbf{s}$.

2.2 Camera Device

We consider the simplest ideal pin hole camera, where the pinhole has an infinitesimal size. In real world, this would not be the case. In a real pinhole camera, an aperture is present instead of pinhole. An aperture has a certain (small) size, but it is big enough to cause the image formed become blurry, due to the nature of light. A more complex camera system introduces lens to correct the aperture, i.e. to focus.

The camera basis could be calculated as follows

$$\mathbf{w} = -view, \quad \mathbf{u} = view \times up, \quad \mathbf{v} = \mathbf{w} \times \mathbf{u} \tag{2.2}$$

2.3 Projection - Orthographic and Perspective

The most naive ray generation is probably orthographic, where all the rays generated are parallel to each other and they are orthonormal to the image plane. This may sound abstract, but when you draw a cube on paper with parallel edges, you are essentially considering an orthographic projection.

A more natural looking approach is perspective projection, where rays all originate from the eye point. Each ray will pass through one pixel in the virtual image plane, and finally may or may not intersect an object in the scene. Depending on whether the ray hits an object, different operations are done to the manipulate the resulting image.

2.3.1 Transforming Between Coordinate Systems

Figure 2 shows the standard pixel coordinate system, where the centre of the bottom left corner pixel is marked as the origin, $(0, 0)$. We will need to handle this subtlety when casting rays.

Consider n_x, n_y to be the number of pixels on X and Y directions respectively, and $width, height$ to be the physical width and height of the image (size of image plane). Then,

- Bottom left corner (pixel space) is $(-1/2, -1/2)$,
- Top right corner (pixel space) is $(n_x - 1/2, n_y - 1/2)$,
- Bottom left corner (camera space) is $(-width/2, -height/2)$,
- Top right corner (camera space) is $(width/2, height/2)$

Do notice in the camera space, if we start at the position at the camera, and move in the direction of $view$ $(-\mathbf{w})$ then we shall arrive at the centre of the image plane, rather than

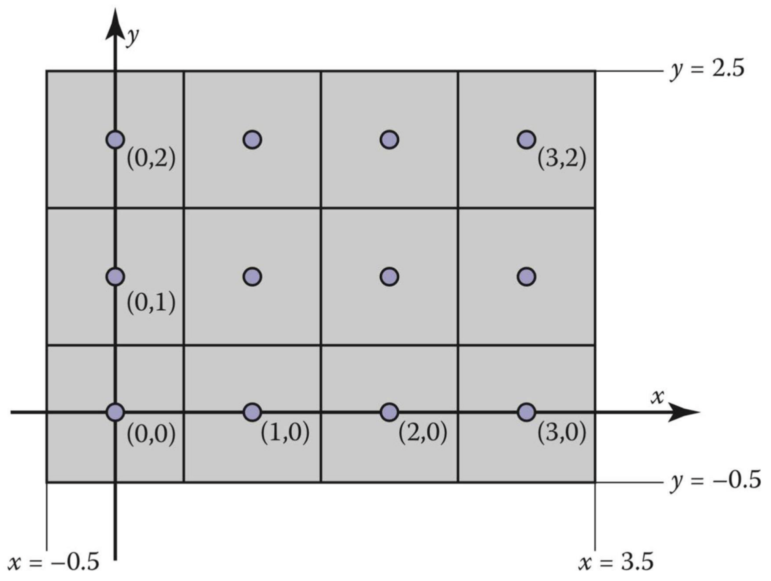


Figure 2: The Standard Pixel Coordinate System.

any corner. Solving the above, we know that pixel at position (i, j) in the raster image has the position (u, v) where

$$u = \frac{width}{n_x}\left(i + \frac{1}{2}\right) - \frac{width}{2} \tag{2.3}$$

and

$$v = \frac{height}{n_y}\left(j + \frac{1}{2}\right) - \frac{height}{2} \tag{2.4}$$

2.4 Ray Equations

2.4.1 ... in Camera Space

We previously stated that the ray equation should have the form

$$\mathbf{p}(t) = \mathbf{e} + t(\mathbf{s} - \mathbf{e}) \tag{2.5}$$

In camera space, we will be at the starting point of the ray, which means that the eye point is exactly origin. Substituting this information into the equation, we have

$$\mathbf{p}(t) = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} + t\left(\begin{bmatrix} u(i) \\ v(j) \\ -d \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}\right) = t\begin{bmatrix} u(i) \\ v(j) \\ -d \end{bmatrix} \tag{2.6}$$

where d is the length between the eye point and the image plane, commonly known as focal length. And u, v calculated using Equations 2.3 and 2.4.

2.4.2 ... in World Space

The world space representation could be attained using a change of coordinates,

$$\mathbf{p}(t) = \mathbf{e} + t(u(i)\mathbf{u} + v(j)\mathbf{v} + -d\mathbf{w}) \tag{2.7}$$

which could be written in matrix form

$$\mathbf{p}(t) = \mathbf{e} + t\begin{bmatrix}\mathbf{u} & \mathbf{v} & \mathbf{w}\end{bmatrix}_{M_{3\times 3}(\mathbb{R})}\begin{bmatrix}u(i) \\ v(j) \\ -d\end{bmatrix} \tag{2.8}$$

where the 3-by-3 matrix in the middle is called the Camera Transformation Matrix, and rest elements calculated as previous.

2.5 Intersection Tests

Now that we have discussed how to generate a ray and how to get the ray in different representations, it is time to move to the `if` statement in our pseudo code, which was to check if an intersection exists between a ray and a scene object.

2.5.1 Ray-Plane Intersection

A plane is defined by the equation ($\mathbf{q}$ such that satisfies the following)

$$\mathbf{n} \cdot (\mathbf{q} - \mathbf{p}_0) = 0 \tag{2.9}$$

where $\mathbf{n}$ is the surface normal and $\mathbf{p}_0$ is a point on the surface. To see if a ray intersects with a plane, it suffices to substitute $\mathbf{p}$ with the parametrized ray equation, and try to solve for t, i.e. we want to solve for t in

$$\mathbf{n} \cdot ((\mathbf{e} + t\mathbf{d}) - \mathbf{p}_0) = 0 \tag{2.10}$$

where $\mathbf{d}$ is the ray direction, equal to $\mathbf{s} - \mathbf{e}$. Solving the above yields

$$t = \frac{-\mathbf{n} \cdot (\mathbf{e} - \mathbf{p}_0)}{\mathbf{n} \cdot \mathbf{d}} \tag{2.11}$$

Notice that the intersection exists iff $t \in \mathbb{R}$, in particular if $\mathbf{n}$ and $\mathbf{d}$ are collinear, then t would blow up. In practice, it is usually useful to do the checking first, to avoid numeric issues.

2.5.2 Ray-Sphere Intersection

We use an implicit equation to define a sphere that is centred at $\mathbf{c}$ with radius r ($\mathbf{q}$ such that satisfies the following)

$$(\mathbf{q}-\mathbf{c})\cdot(\mathbf{q}-\mathbf{c})-r^2=0 \tag{2.12}$$

Again, we substitute $\mathbf{q}$ with the ray parametrization,

$$(\mathbf{e}+t\mathbf{d}-\mathbf{c})\cdot(\mathbf{e}+t\mathbf{d}-\mathbf{c})-r^2=0 \tag{2.13}$$

expanding and rearranging yields

$$(\mathbf{d}\cdot\mathbf{d})t^2+2\mathbf{d}\cdot(\mathbf{e}-\mathbf{c})t+(\mathbf{e}-\mathbf{c})\cdot(\mathbf{e}-\mathbf{c})-r^2=0 \tag{2.14}$$

To solve this, we can resort to the quadratic equation. First we calculate the discriminant,

$$\Delta=(2\mathbf{d}\cdot(\mathbf{e}-\mathbf{c}))^2-4(\mathbf{d}\cdot\mathbf{d})((\mathbf{e}-\mathbf{c})\cdot(\mathbf{e}-\mathbf{c})-r^2) \tag{2.15}$$

Clearly, if $\Delta<0$ then there is no intersection. If $\Delta=0$ then the ray is tangential to the sphere, and the ray has two intersection points with the sphere otherwise. The exact solution(s) could be calculated as

$$t=\frac{-2\mathbf{d}\cdot(\mathbf{e}-\mathbf{c})\pm\sqrt{\Delta}}{2\mathbf{d}\cdot\mathbf{d}} \tag{2.16}$$

2.5.3 Ray-Triangle Intersection

In this case, we parametrized a triangle as

$$\mathbf{q}=\mathbf{p}_0+\alpha\mathbf{t}_1+\beta\mathbf{t}_2 \tag{2.17}$$

where $\mathbf{p}_0$ is a vertex of the triangle and $\mathbf{t}_1,\mathbf{t}_2$ are two edges shooting outwards from $\mathbf{p}_0$. In order to make this a triangle, there are several constraints, namely

$$\alpha\geq 0,\quad \beta\geq 0,\quad \alpha+\beta\leq 1. \tag{2.18}$$

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 $f(a, b)$:

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

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 3 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.

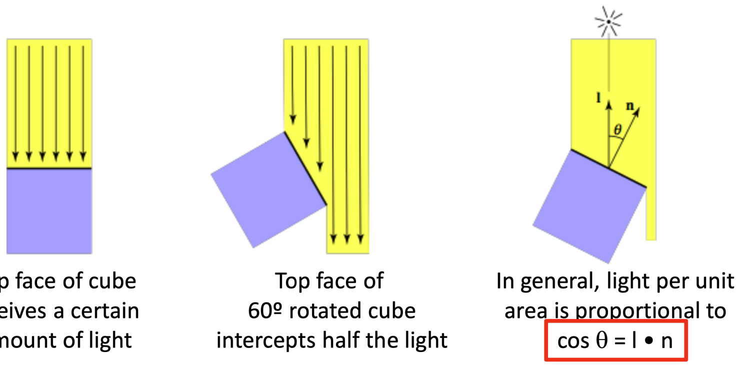

Top face of cube receives a certain amount of light
Top face of 60º rotated cube intercepts half the light
In general, light per unit area is proportional to
cos θ = l • n
l
n
θ

Figure 3: Illustration of Lambert's cosine law.

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

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

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

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;
}
```

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 4 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 4 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*

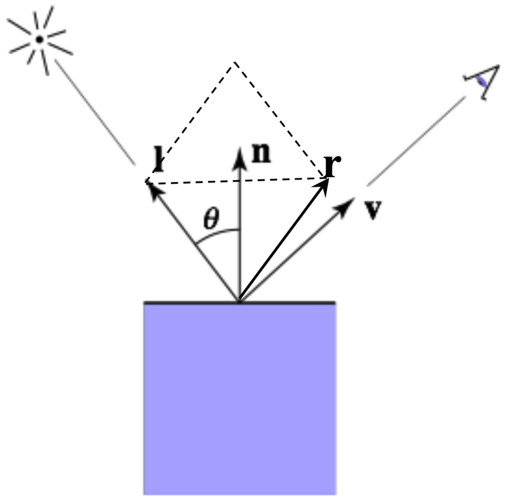


Figure 4: Mirror reflection illustration

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 5 illustrate what we want to calculate. Let's call the point of reflection $\mathbf{o}$, then in Phong model, we used $angle\mathbf{rov}$ to model the angle and here in Blinn we will be using $\angle\mathbf{hon}$ 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.

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

The specular reflected intensities can be calculated

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

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.

[3.2]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.

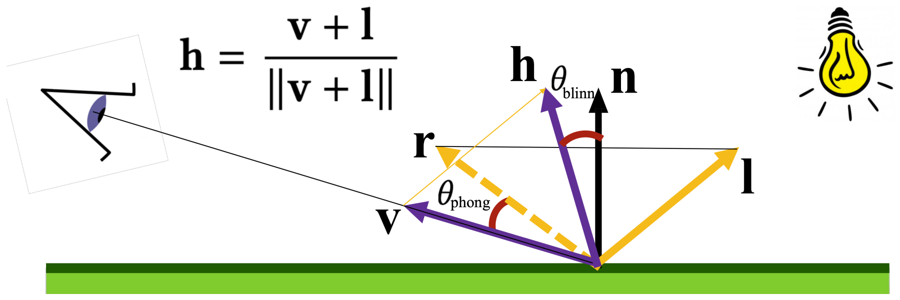

$\mathbf{h} = \dfrac{\mathbf{v}+\mathbf{l}}{\|\mathbf{v}+\mathbf{l}\|}$
h
θ_{blinn}
n
r
l
θ_{phong}
v

Figure 5: Blinn Specular Shading and Phong Specular Shading.

3.13 Local Illumination

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

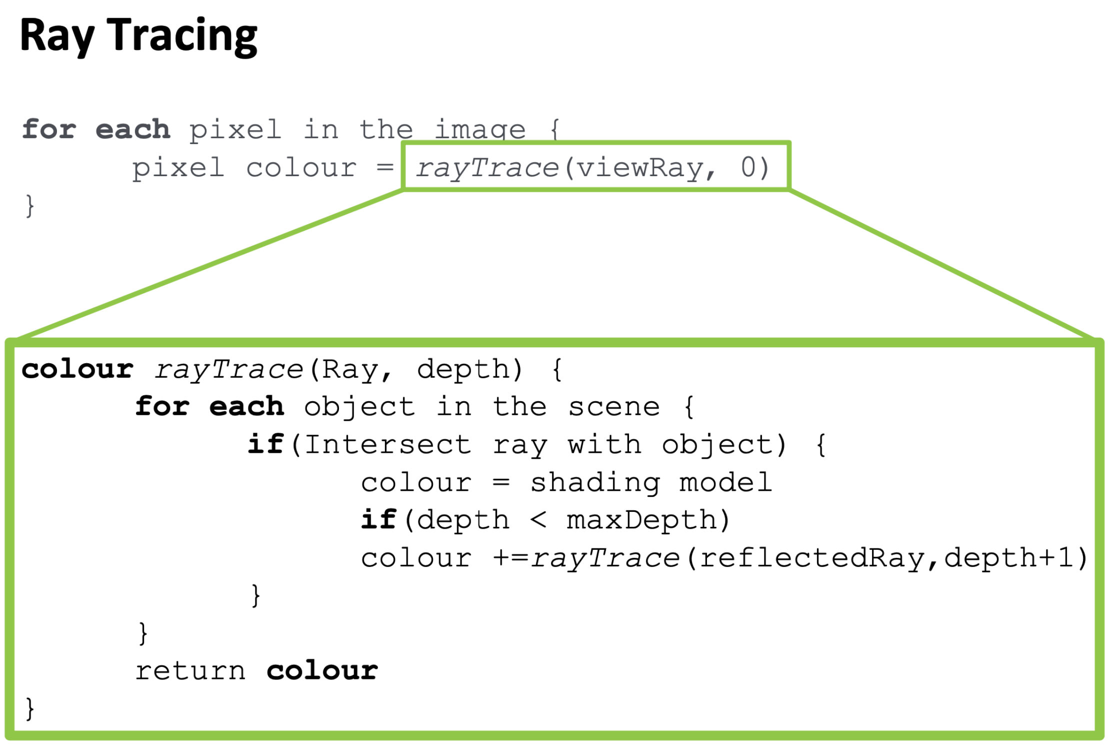
$$\begin{aligned} L &= L_a + L_d + L_s \\ &= k_a I_a + k_d \left(I/r^2\right) \max(0, \mathbf{n}\cdot\mathbf{l}) + k_s \left(I/r^2\right) \max(0, \mathbf{n}\cdot\mathbf{h})^p \end{aligned} \tag{3.9}$$

- 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



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

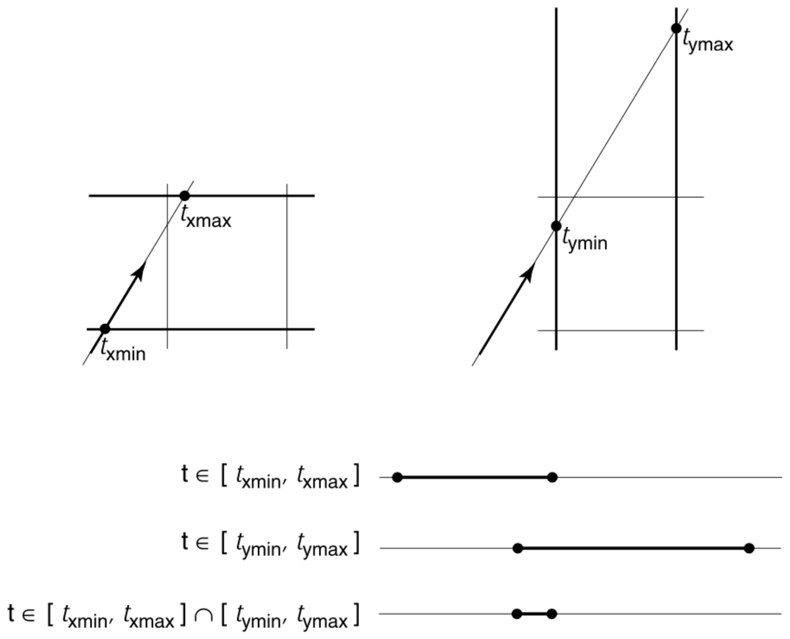
To derive, let's start with the 2D case. Suppose we have a bounding box described by four lines,

$$x = x_{\text{min}}, \qquad \text{the lower horizontal line,} \tag{4.1}$$
$$x = x_{\text{max}}, \qquad \text{the upper horizontal line,} \tag{4.2}$$
$$y = y_{\text{min}}, \qquad \text{the left vertical line,} \tag{4.3}$$
$$y = y_{\text{max}}, \qquad \text{the right vertical line.} \tag{4.4}$$

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 6 will the ray hit the box. Arithmetically, we calculate[4.1]

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

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



t_{xmin}
t_{xmax}
t_{ymin}
t_{ymax}
$t \in [\, t_{\text{xmin}}, t_{\text{xmax}}]$
$t \in [\, t_{\text{ymin}}, t_{\text{ymax}}]$
$t \in [\, t_{\text{xmin}}, t_{\text{xmax}}] \cap [\, t_{\text{ymin}}, t_{\text{ymax}}]$

Figure 6: 2D Bounding Box Hit Criterion

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:

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

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

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

[4.2] This if statement condition could be derived from Figure 6: when the two intervals have no intersection, then we have a miss.

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 6: when the two intervals have no intersection, then we have a miss.