Announcements
Visibility Problem

• Rendering: converting a model to an image
• Visibility: deciding which objects (or parts) will appear in the image
  – Object-order
  – Image-order

Raytracing

• Given
  – Scene
  – Viewpoint
  – Viewplane
• Cast ray from viewpoint through pixels into scene
Raytracing Algorithm

Given
List of polygons \{ P_1, P_2, \ldots, P_n \}
An array of intensity \{ x, y \}

\{
  For each pixel \( (x, y) \) {
    form a ray \( R \) in object space through the
    camera position \( C \) and the pixel \( (x, y) \)
    Intensity \( [x, y] = \text{trace}(R) \)
  }
Display array Intensity
\}
Projection

- Perspective
  - Line AB projects to A'B' (perspective projection)

- Parallel
  - Line AB projects to A'B' (parallel projection)
  - Projectors AA' and BB' are parallel

Simple Parallel Tform

View plane is normal to direction of projection

\[ x_s = x_v, y_s = y_v, z_s = 0 \]

\[
T_{ort} = 
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 \\
\end{bmatrix}
\]
Simple Perspective Tform

- Assume line from center of projection to center of view plane parallel to view plane normal.
- Center of projection is at origin.
- Have \( P(x_v, y_v, z_v) \)
- Want \( P(x_s, y_s) \)

By similar triangles:

\[
\frac{x_s}{d} = \frac{x_v}{z_v}, \quad \frac{y_s}{d} = \frac{y_v}{z_v}
\]

\[
\Rightarrow x_s = \frac{x_v}{z_v} d, \quad y_s = \frac{y_v}{z_v} d
\]
Simple Perspective Tform

- Have $P(x_v, y_v, z_v)$, want $P(x_s, y_s)$
- By similar triangles:
  \[
  \frac{x_s}{d} = \frac{x_v}{z_v}, \quad \frac{y_s}{d} = \frac{y_v}{z_v} \Rightarrow x_s = \frac{x_v}{z_v/d}, \quad y_s = \frac{y_v}{z_v/d}
  \]
- In homogeneous coords
  \[
  x = x_v, \quad y = y_v, \quad z = z_v, \quad w = z_v/d
  \]
- Do perspective divide to get screen coords
  \[
  x_s = x/w, \quad y_s = y/w, \quad z_s = z/w = d
  \]

Raytracing Algorithm

Given
List of polygons \{ P_1, P_2, \ldots, P_n \}
An array of intensity \{ x, y \} \{
  For each pixel \( (x, y) \) \{
    form a ray \( R \) in object space through the camera position \( C \) and the pixel \( (x, y) \)
    Intensity \{ x, y \} = \text{trace} ( R )
  \}
  Display array Intensity
\}
Raytracing Algorithm

```plaintext
intensity Function  trace ( Ray )
{
    for each polygon P in the scene
        calculate the intersection of P and the ray R
    if ( The ray R hits no polygon )
        return ( background intensity )
    else {
        find the polygon P with the closest intersection
        calculate intensity I at intersection point
        return ( I ) // more to come here later
    }
}
```
Computing Viewing Rays

• Parametric ray
  \[ p(t) = e + t(s - e) \]
• Camera frame
  – E: eye point
  – u,v,w: basis vectors pointing right, up, backward
• Screen position
  – orthographic
    \[ u_s = l + (r-l)(i+0.5)/n_x \]
    \[ v_s = b + (u-b)(j+0.5)/n_y \]
    \[ s = (e + u_s u + v_s v) - w \]
  – Perspective
    \[ u_s = l + (r-l)(i+0.5)/n_x \]
    \[ v_s = b + (u-b)(j+0.5)/n_y \]
    \[ s = (e) + u_s u + v_s v - d w \]

Calculating Intersections

• Define ray parametrically:
  \[ x = x_0 + t(x_1 - x_0) = x_0 + t\Delta x \]
  \[ y = y_0 + t(y_1 - y_0) = y_0 + t\Delta y \]
  \[ z = z_0 + t(z_1 - z_0) = z_0 + t\Delta z \]
• If \((x_0, y_0, z_0)\) is center of projection and \((x_1, y_1, z_1)\) is center of pixel, then
  \[ 0 \leq t \leq 1 : \text{points between those locations} \]
  \[ t < 0 : \text{points behind viewer} \]
  \[ t > 1 : \text{points beyond view window} \]
Ray-Sphere Intersection

- Sphere in vector form
  - \((\mathbf{p} - \mathbf{c}) \cdot (\mathbf{p} - \mathbf{c}) - R^2 = 0\)
- Ray
  - \(\mathbf{p}(t) = \mathbf{e} + t\mathbf{d}\)
- Intersection with implicit surface \(f(t)\) when
  - \(f(\mathbf{p}(t)) = 0\)
  - \((\mathbf{e} + t(\mathbf{s} - \mathbf{e}) - \mathbf{c}) \cdot (\mathbf{e} + t(\mathbf{s} - \mathbf{e}) - \mathbf{c}) - R^2 = 0\)
  - \((\mathbf{d} \cdot \mathbf{d})t^2 + 2(\mathbf{d} \cdot (\mathbf{e} - \mathbf{c}))t + (\mathbf{e} - \mathbf{c}) \cdot (\mathbf{e} - \mathbf{c}) - R^2 = 0\)
  - \(t = -\frac{(\mathbf{d} \cdot (\mathbf{e} - \mathbf{c})) \pm \sqrt{(\mathbf{d} \cdot (\mathbf{e} - \mathbf{c}))^2 - ((\mathbf{d} \cdot \mathbf{d})((\mathbf{e} - \mathbf{c}) \cdot (\mathbf{e} - \mathbf{c}) - R^2))}{(\mathbf{d} \cdot \mathbf{d})}\)
- Normal at intersection \(\mathbf{p}\)
  - \(\mathbf{n} = (\mathbf{p} - \mathbf{c}) / R\)

Calculating Intersections: Pgons

- Given ray and polygon:
  - \(x = x_0 + t(x_1 - x_0) = x_0 + t\Delta x\)
  - \(y = y_0 + t(y_1 - y_0) = y_0 + t\Delta y\)
  - \(z = z_0 + t(z_1 - z_0) = z_0 + t\Delta z\)
  - Plane: \(Ax + By + Cz + D = 0\)
- 1. What is intersection of ray and plane containing pgon?
  - Substituting for \(x, y, z\):
    - \(A(x_0 + t\Delta x) + B(y_0 + t\Delta y) + C(z_0 + t\Delta z) + D = 0\)
    - \(t(A\Delta x + B\Delta y + C\Delta z) + (Ax_0 + By_0 + Cz_0 + D) = 0\)
    - \(t = -\frac{(Ax_0 + By_0 + Cz_0 + D)}{(A\Delta x + B\Delta y + C\Delta z)}\)
- 2. Does ray/plane intersection point lie in polygon?
  - Substitute into line equations for pgon edges: does point lie inside all edges? (only works for convex)
  - Count edge crossings of ray from point to infinity
Point in Polygon?
• Is P in polygon?
• Cast ray from P to infinity
  – One crossing -> inside
  – Zero, Two crossings -> outside

Point in Polygon?
• Is P in concave polygon?
• Cast ray from P to infinity
  – Odd crossings -> inside
  – Even crossings -> outside
What happens?

Ray-Triangle Intersection

• Intersection of ray with barycentric triangle
  - \( \mathbf{e+td = a+\beta(b-a)+\gamma(c-a)} \)
  - In triangle if \( \beta > 0, \gamma > 0, \beta+\gamma < 1 \)

```java
boolean raytri (ray r, vector a, vector b, vector c, interval [t₀, t₁]) {
  compute t
  if ((( t < t₀ ) or (t > t₁)))
    return ( false )
  compute γ
  if (((γ < 0 ) or (γ > 1))
    return ( false )
  compute β
  if (((β < 0 ) or (β > 1))
    return ( false )
  return true
}
```
Raytracing Characteristics

• Good
  – Simple to implement
  – Minimal memory required
  – Easy to extend

• Bad
  – Aliasing
  – Computationally intensive
    • Intersections expensive (75-90% of rendering time)
    • Lots of rays

Basic Concepts

• Terms
  – Illumination: calculating light intensity at a point (object space; equation) based loosely on physical laws
  – Shading: algorithm for calculating intensities at pixels (image space; algorithm)

• Objects
  – Light sources: light-emitting
  – Other objects: light-reflecting

• Light sources
  – Point (special case: at infinity)
  – distributed
**Ambient light**

$I_a = \text{intensity of ambient light}$

$K_a = \text{reflection coefficient}$

$I = k_a I_a = \text{reflected intensity}$

**Lambert’s Law**

- Intensity of reflected light related to orientation
Lambert’s Law

- Specifically: the radiant energy from any small surface area \( dA \) in any direction \( \theta \) relative to the surface normal is proportional to \( \cos \theta \)

\[ I_{\text{diff}} = k_d I_l \cos \theta = k_d I_l (N \cdot L) \]
Combined Model

\[ I_{\text{total}} = I_{\text{amb}} + I_{\text{diff}} \]
\[ = k_a I_a + k_d I_d (N \cdot L) \]

Adding color:

\[ I_R = k_a I_a R O_{dR} + k_d I_{IR} O_{dR} (N \cdot L) \]
\[ I_G = k_a I_a G O_{dG} + k_d I_{IG} O_{dG} (N \cdot L) \]
\[ I_B = k_a I_a B O_{dB} + k_d I_{IB} O_{dB} (N \cdot L) \]

For any wavelength \( \lambda \):

\[ I_{\lambda} = k_a I_a \lambda O_{d\lambda} + k_d I_d \lambda O_{d\lambda} (N \cdot L) \]

Adding Attenuation

- Attenuation of light source due to distance
  - \( F_{\text{att}} = 1/d_L^2 \) or \( \min(1/(C_1 + C_2 d_L + C_3 d_L^2), 1) \)
  - where \( d_L \) is distance to the light
  - Behavior of \( 1/d_L^2 \)
    - Far from light: little change
    - Near light: much change
    - Accurate, but looks wrong

- Atmospheric attenuation of color
  - \( I_{\lambda}' = S_O I_{\lambda} + (1-S_O) I_{dc\lambda} \)
  - where \( I_{dc\lambda} \) is the depth cue color
  - \( S_O = S_b + (Z_O - Z_b)(S_f - S_b)/(Z_f - Z_b) \)

- Depth cue scale and depth
  - \( S_1 \rightarrow S_2 \) as depth increases
  - \( Z_0 \) to \( Z_f \) depth range
  - \( S_1 \) to \( S_2 \) depth cue range
Specular Reflection

For specific wavelength $\lambda$.

$$I_{\text{spec}, \lambda} = k_s \lambda I_\lambda \cos^a \phi$$

$$= k_s \lambda (R \cdot V)^a$$

Not dependent on surface color $\rightarrow$ white highlights

---

Specular Reflection

For specific wavelength $\lambda$.

$$I_{\text{spec}, \lambda} = k_s \lambda I_\lambda O_{\lambda, \phi} \cos^a \phi$$

$$= k_s \lambda I_\lambda O_{s\lambda} (R \cdot V)^a$$

Dependent on surface color $\rightarrow$ colored highlights
Specular Reflection

- Dull highlights
  - Gradual falloff
  - Approximated by $\cos \phi$

- Glossy highlights
  - Steeper falloff
  - Approximated by $\cos^8 \phi$
Specular Reflection

- Shiny highlights
  - Steep falloff
  - Approximated by $\cos^{128}\phi$

Calculating the Reflection Vector

- Specular: $I_{\text{spec},\lambda} = k_{\lambda} I_{\lambda,\alpha} (R \cdot V)^\mu$
- Have $L$, want $R$
Calculating the Reflection Vector

- Mirror L about N
  \[ R = N \cos \theta + S = 2N \cos \theta - L = 2N(N \cdot L) - L \]

- Alternatively: use halfway vector H
  \[ H = \frac{L+V}{|L+V|} \]

- Maximum highlight when H=N (because then R=V)
  \[ I_{\text{spec}} = k_s I_l O_{\omega} (H \cdot N)^n = k_s I_l O_{\omega} (2N(N \cdot L) - L \cdot V)^n \]

Where N, L are unit length
Projection of L on N is \( N \cos \theta \)
\( S = N \cos \theta - L \)

- Two methods can give different results \( \alpha \neq \phi \)
Combined Model

\[ I_{\text{total}} = I_{\text{amb}} + I_{\text{diff}} + I_{\text{spec}} \]

\[ = k_a I_a + k_d I_l (N \cdot L) + k_s I_l (N \cdot H)^n \]

Multiple lights:

\[ = k_a I_a + \sum (k_d I_{li} (N \cdot L) + k_s I_{li} (N \cdot H)^n) \]

By wavelength (white highlights):

\[ = k_a I_a O_d \lambda + \sum (k_d I_{li} (N \cdot L) O_d \lambda + k_s I_{li} (N \cdot H)^n) \]

By wavelength (colored highlights):

\[ = k_a I_a O_d \lambda + \sum (k_d I_{li} (N \cdot L) O_d \lambda + k_s I_{li} (N \cdot H)^n O_s \lambda) \]

By wavelength (more metallic highlights):

\[ = k_a I_a O_d \lambda + \sum (k_d I_{li} (N \cdot L) O_d \lambda + k_s(\lambda, \theta) I_{li} (N \cdot H)^n O_s \lambda) \]

Basic Raytracing Program

```plaintext
{ 
    for each pixel (x, y) do { 
        compute viewing ray 
        if (ray hit an object with t>0) then { 
            compute n 
            evaluate shading model 
            set pixel to that color 
        } 
    else 
        set pixel color to background color 
    } 
} ```
Effects

• Visibility
• Illumination
• Shadow

Shadow Algorithm

Function raycolor(ray e+td, real t₀, real t₁)
{
    hit-record rec, srec
    if (scene->hit(e+td, t₀, t₁)) then {
        p = e+(rec.t)d
        color c = rec.k_a*I_a
        if (not scene->hit(p+sl, ε, ∞, srec)) then {
            vector h = unit(unit(l)+unit(-d))
            c = c + rec.k_o*I*max(0, rec.n•l) + 
                rec.k_s*I(rec.n•h)rec.p
        }
        return c
    }
    return background color
}
Effects

• Reflection

• Calculate ray direction
  \[ r = d - 2(d \cdot n)n \]
  – \( d \) points from eye to surface

• Trace ray
  \[ m = \text{raycolor}(p+sr, \varepsilon, \infty) \]

• Composite
  \[ c = c + k_m m \]

Effects

• Refraction
How many bounces?