CSE168
Computer Graphics II, Rendering

Spring 2006
Matthias Zwicker
Last time

- Course overview
- Ray tracing algorithm
- Computing primary rays
Today

• Ray-surface intersection
• Implicit, parametric surfaces
• Spheres, triangles, polygons
• Shading
Ray tracing pseudocode

for all pixels {
    computeprimary( &ray )
    for all objects {
        intersect( ray, &hit )
        if hit is closer than firsthit {
            firsthit = hit
        }
    }
}

shade( firsthit )
Implicit surfaces

- Implicit surfaces are defined by
  \[ f(p) = 0, \quad \text{where} \quad p = (x, y, z) \]

- Surface normal
  \[ n = \nabla f(p) = \left( \frac{\partial f(p)}{\partial x}, \frac{\partial f(p)}{\partial y}, \frac{\partial f(p)}{\partial z} \right) \]

- Given a ray
  \[ p(t) = e + td \]

- Ray-surface intersection
  \[ f(p(t)) = 0 \]
Example: infinite plane

- Normal \( \mathbf{n} \), point on plane \( \mathbf{a} \)
  \[
  f(p) = (p - a) \cdot n = 0
  \]

- Intersection with \( p(t) \)
  \[
  (e + td - a) \cdot n = 0
  \]
  
  \[
  t = \frac{(a-e) \cdot n}{d \cdot n}
  \]

- Surface normal
  \[
  \nabla ( (p - a) \cdot n ) = n
  \]
Parametric surfaces

- Parameters $u, v$

$$x = f(u, v)$$
$$y = g(u, v)$$
$$z = h(u, v)$$

- Surface normal

$$\mathbf{n}(u, v) = \begin{pmatrix} \frac{\partial f}{\partial u}, & \frac{\partial g}{\partial u}, & \frac{\partial h}{\partial u} \end{pmatrix} \times \begin{pmatrix} \frac{\partial f}{\partial v}, & \frac{\partial g}{\partial v}, & \frac{\partial h}{\partial v} \end{pmatrix}$$

- Cross product of tangent vectors
Parametric surfaces

• Ray-surface intersection

\[ e_x + t d_x = f(u, v) \]
\[ e_y + t d_y = g(u, v) \]
\[ e_z + t d_z = f(u, v) \]

• Easy to solve for parametric planes (see ray-triangle intersection)

• Requires iterative solution for most non-linear surfaces (e.g., NURBS)
Ray-sphere intersection

• Sphere with center \( c = (c_x, c_y, c_z) \), radius \( R \)

\[
(x - c_x)^2 + (y - c_y)^2 + (z - c_z)^2 - R^2 = 0
\]

\[
(p - c) \cdot (p - c) - R^2 = 0
\]

• Use \( p(t) = e + td \)

\[
(e + td - c) \cdot (e + td - c) - R^2 = 0
\]

• Rearrange

\[
(d \cdot d)t^2 + 2d \cdot (e - c)t + (e - c) \cdot (e - c) - R^2 = 0
\]
Ray-sphere intersection

- Surface normal
  \[ n = \nabla \left( (p - c) \cdot (p - c) - R^2 \right) = 2(p - c) \]

- Unit normal
  \[ (p - c)/R \]
Ray-triangle intersection

- Parametric description of a triangle with vertices \(a, b, c\)

\[
p = a + \beta(b - a) + \gamma(c - a)
\]

where \(\beta > 0, \gamma > 0, \beta + \gamma < 1\)

- Convention: vertices are ordered counter-clockwise as seen from the outside
Ray-triangle intersection

• Outward-pointing normal
  \[ n = (b - a) \times (c - a) \]

• Triangle in barycentric coordinates
  \[ p = \alpha a + \beta b + \gamma c \]
  where \( \alpha + \beta + \gamma = 1 \),
  \[ 0 < \alpha < 1, 0 < \beta < 1, 0 < \gamma < 1 \]
Ray-triangle intersection

- Intersection condition
  \[ e + td = a + \beta(b - a) + \gamma(c - a) \]

- One equation for each coordinate
  \[
  \begin{align*}
  e_x + td_x &= a_x + \beta(b_x - a_x) + \gamma(c_x - a_x) \\
  e_y + td_y &= a_y + \beta(b_y - a_y) + \gamma(c_y - a_y) \\
  e_z + td_z &= a_z + \beta(b_z - a_z) + \gamma(c_z - a_z)
  \end{align*}
  \]
Ray-triangle intersection

• In matrix form

\[
\begin{bmatrix}
  a_x - b_x & a_x - c_x & d_x \\
  a_y - b_y & a_y - c_y & d_y \\
  a_z - b_z & a_z - c_z & d_z 
\end{bmatrix}
\begin{bmatrix}
  \beta \\
  \gamma \\
  t 
\end{bmatrix}
=
\begin{bmatrix}
  a_x - e_x \\
  a_y - e_y \\
  a_z - e_z 
\end{bmatrix}
\]

• Solve for $\beta, \gamma, t$ using Cramer’s rule (Shirley page 157)
Interpolating normals

- Sometimes, we store a normal with each vertex $n_a, n_b, n_c$
- Interpolating normals at barycentric coordinates $\alpha, \beta, \gamma$
  \[ n(\alpha, \beta, \gamma) = \alpha n_a + \beta n_a + \gamma n_a \]
- The same for other vertex attributes
Ray-polygon intersection

- Vertices $p_1 \ldots p_m$
- Polygon lies on plane
  
  $$(p - p_1) \cdot n = 0$$

- Intersection with ray $p(t) = e + td$
  
  $$t = \frac{(p_1 - e) \cdot n}{d \cdot n}$$

- Determine whether $p(t)$ lies within the polygon
Ray-polygon intersection

$p_1$, $p_2$, $p_3$, $p_4$, $p_5$

$p(t)$

$xy, yz$ or $zx$ plane
Ray-polygon intersection
Ray-polygon intersection

$\mathbf{p}_1 \mathbf{p}_2 \mathbf{p}_3 \mathbf{p}_4 \mathbf{p}_5$

$p(t)$

$xy, yz$ or $zx$ plane
Ray-polygon intersection

$p(t)$

$p_1$, $p_2$, $p_3$, $p_4$, $p_5$

$xy, yz$ or $zx$ plane
The hit record

class hit_record {
    t       // intersection parameter
    p       // intersection point
    n       // normal at intersection point
    *object // pointer to hit object
}
Intersecting transformed objects

- Primary rays are in world coordinates
- Objects may be expressed in local object coordinates
- Transformation

\[ p_{\text{world}} = M_{\text{obj} \rightarrow \text{world}} p_{\text{object}} \]
Intersecting transformed objects

• Transform rays from world to object coordinates

\[ p_{object}(t) = M_{obj \rightarrow world}^{-1} p_{world}(t) \]
\[ = M_{obj \rightarrow world}^{-1} e + t M_{obj \rightarrow world}^{-1} d \]
Instancing

- Re-use objects without replicating them in memory
- Allow additional transformation

```cpp
class instance : object {
    *object // pointer to an object
    M      // object to world transformation
    M_inv  // world to object transformation
}
```
Instancing

• For intersection, transform ray instead of geometry
• Hit record needs to be transformed back to world coordinates

```cpp
instance::intersect( ray, &hit ) {
    ray_object = transform_ray( ray, M_inv )
    object->intersect( ray_object, &hit_object )
    hit = transform_hit( hit_object, M )
}
```
Shading

for all pixels {
    computeprimary( &ray )
    for all objects {
        intersect( ray, &hit )
        if hit is closer than firsthit {
            firsthit = hit
        }
    }
}

shade( firsthit )
Shading

- Disclaimer
- The BRDF
- A simple model with diffuse, specular, and ambient components
Disclaimer

- Ultimate goal is to model physical process of light transport
- Light transport is complicated in reality
- For now, a hacker’s approach
  - Simple mathematical models, easy to implement
  - Loosely connected to physics
  - Reproduce perceptually prominent effects
Disclaimer

- More physics background in the second part of the course
- Colors, reflectance coefficients
  - RGB triplets
  - Each color value between 0 and 1
The BRDF

- *Bi-directional reflectance distribution function*
- Describes how a surface reflects light
  - Locally, at a specific point
- Contains *almost* all information about the appearance of surfaces
  - “Color”, shiny, glossy, mirror, matte, ...
The BRDF

• Given
  - Light direction \( l \)
  - Viewing direction \( v \)

• Return
  \[ \rho(l, v) \]
  - Fraction of light arriving from source that is transported towards viewer
  - Function of two directions, i.e., 4 angles (variables/dimensions)
The BRDF

• Given
  - Light direction \( l \)
  - Viewing direction \( v \)

• Return
  
  \[ \rho(l, v) \]
  
  - Fraction of light arriving from source that is transported towards viewer
  - Function of two directions, i.e., 4 angles (variables/dimensions)
The BRDF

• Given
  - Light direction \( l \)
  - Viewing direction \( v \)

• Return
  \[ \rho(l, v) \]
  - Fraction of light arriving from source that is transported towards viewer
  - Function of two directions, i.e., 4 angles (variables/dimensions)
The BRDF

• More about physics of BRDFs (units) later
• Frequency dependent, use RGB components
• Real materials have spatially varying BRDFs
  - Different BRDF at each point
  - Function with 6 degrees of freedom/dimensions (two for location on the surface, 4 for directions)
A simple model

- Sum of 3 components

\[ \text{diffuse} + \text{specular} + \text{ambient} = \]
Diffuse term

- Matte, not shiny materials
- Highly diffuse materials
  - Paper
  - Unfinished wood
  - Unpolished stone
- View-independent
Diffuse term

\[ \rho(l, v), \text{ fixed } l \]

“slice” through BRDF
Diffuse term

• Ideal diffuse surfaces follow Lambert’s law

\[ L \propto \cos(\theta) \]

\[ L \propto \mathbf{n} \cdot \mathbf{l} \]

where \( L \) is apparent surface color

• Surface normal \( \mathbf{n} \)

• Light direction \( \mathbf{l} \)

• Unit vectors!
Diffuse term

- **Practical model**
  - Light color $L_i$
  - Surface color $R_d$
  - Avoid negative dot products

  $$L = R_d L_i \max(0, \mathbf{n} \cdot \mathbf{l})$$

- **Two-sided lighting**

  $$L = R_d L_i |\mathbf{n} \cdot \mathbf{l}|$$
Specular term

- Highlights
- View dependent
  - Highlights move across surfaces as the viewpoint moves
- Blurry reflections of light sources
- Are often the color of the light source (e.g. plastics, but not metals)
Specular term

Fixed 1
“Specular lobe”
Specular highlights

- Phong shading
- Reflected light direction $r$
- Viewing direction $e$
- Phong exponent $p$

$$L = L_i R_s \max(0, e \cdot r)^p$$
Specular highlights

- Unit reflection vector

\[ r = -l + 2(l \cdot n)n \]
Specular highlights

- Alternative formulation
- Halfway vector $h$

\[ h = \frac{e + l}{\|e + l\|} \]

\[ L = L_i R_s (h \cdot n)^p \]
Specular highlights

- Varying specular exponents
Ambient term

- “Ambient, omni-directional” light
  \[ L = R_d L_a \]
- Account for light that is not directly emitted from light sources
- Cheapest of all hacks
Combined lighting model

\[ L = L_i R_d \max(0, n \cdot l) + L_i R_s (h \cdot n)^p + R_d L_a \]

- Several light sources \( L_{i,k} \)

\[ L = R_d L_a + \sum_k L_{i,k} (R_d \max(0, n \cdot l)) + R_s (h \cdot n)^p \]
Point and directional lights

- **Directional light**
  - Approximates light sources far away from objects
  - Constant direction

- **Point light**
  - Direction changes over surfaces
Falloff, spot lights

- Physically, intensity falloff from point light $1/r^2$
- Looks bad, in practice $1/r$
- Spot light
Discussion section

- Tomorrow Thursday
- 11:00-12:30, EBU3B b250
- Introduction to base code