To Shading and Beyond

CS 6965 Fall 2011
Program 1

- Questions on program 1?
- Running simhwrt
simhwrt

- Can set scene and image size
  - --no-scene
  - --width 256
  - --height 256

- Can set number of TMs and threads per tm
  - --num-thread-procs 32
  - --num-cores 10

- Gathers statistics for EVERYTHING

- More on simhwrt later
Ray tracing architecture

- The major components in a ray tracer are:
  - Camera (Pixels to Rays)
  - Objects (Rays to intersection info)
  - Materials (Intersection info and light to color)
  - Lights
  - Background (Rays to Color)

- All together: a Scene
Ray tracing algorithm

- Create scene (objects, materials, lights, camera, background)
- Preprocess scene
- foreach frame
  - foreach pixel
    - foreach sample
      - generate ray
      - intersect ray with objects
      - find normal of closest object
      - Mutually recursive
Create scene
Details

- Create scene
- Preprocess scene
Details

- Create scene
- Preprocess scene
- foreach pixel

Tiled  Progressive  Frameless rendering  Parallel

Row-major order
How rt.s Does It

- We did tiled thread assignments at first
- Currently two candidates:
  - Row-major order
  - Space filling curve
Details

- Create scene
- Preprocess scene
- foreach pixel
  - foreach sample
Details

- Create scene
- Preprocess scene
- foreach pixel
  - foreach sample
  - generate ray
Details

- Create scene
- Preprocess scene
- foreach pixel
  - foreach sample
    - generate ray
    - intersect ray with objects

\( t_{\text{near}} \)
Details

- Create scene
- Preprocess scene
- foreach pixel
  - foreach sample
    - generate ray
    - intersect ray with objects
    - find normal of closest object
Details

- Create scene
- Preprocess scene
- foreach pixel
  - foreach sample
    - generate ray
    - intersect ray with objects
    - find normal of closest object
    - shade intersection point
Background

• What do you do when the ray misses all objects?
  • Constant color
  • Assign color to ray
  • Gradient
  • Map -1 to 1:
    • Map to colors:
    • Star field:
    • Sum up colors:
  • Environment map

\[
s = \frac{\|V_{\text{ray}} \cdot V_{\text{up}}\|}{\|V_{\text{ray}}\| \cdot \|V_{\text{up}}\|}
\]

\[
C_{\text{down}} + \left(\frac{s + 1}{2}\right)C_{\text{up}}
\]

\[
\sum_{\text{stars}} \left(\frac{V_{\text{ray}} \cdot V_{\text{star}}}{V_{\text{star}}}\right)^p C_{\text{star}} \cdot p \approx 10000
\]
Data structures: Light

• Important features:
  • Method to compute light incident to a point
  • Returns Color and Direction

• Point light:
  • Position
  • Color

• Directional light:
  • Vector direction
  • Color
Light

- Stored in global memory
  - A single point light
  - Could treat it as a sphere around the point
- Other lights you should just hard code (for now)
- Program 2 light will probably be hard coded
Shading

- The shading step is the key aspect of ray tracing
Shading

- Path tracing: consider light from all directions
- Ray tracing: consider the dominant directions:
  - direct (unobstructed from light source)
  - reflection
  - refraction
Shadow rays

- Shadows are computed by tracing rays from (to) the light source
- Intersection point: \( \vec{P} = \vec{O} + t\vec{V} \)
- Origin: \( \vec{O} \)
- Direction: \( \vec{V} \)
- \( \text{max } t: 1.0 \)
Shadow ray bugs

- Two bugs might show up when you do this:
  - False shadows (considering rays $<0$ or $>1$)
  - Freckles (considering rays $== 0$)
Numerical precision

- Zoomed in: ideal
Numerical precision

- Zoomed in $x1e7 - ish$: short (numerical roundoff)
Numerical precision

- Zoomed in ($x1e7 - ish$): short (numerical roundoff)
Solutions:

• Only consider intersections where $t > \text{small\_num}$
• Offset ray in normal direction: $P_+ = N \times \text{small\_num}$
• Offset ray in light source direction:
  • $P_+ = (L - P) \times \text{small\_num}$

• $\text{small\_num} = 1.e-6$
Shading models
Lambert’s cosine law

- Light reaching surface is proportional to projected area: $\cos \theta$
Lambertian shading

• Comes from a “rough” surface (at microscopic level)
• Simple: light that reaches the surface is reflected equally in all directions
Lambertian shading

- Color at surface: \((\vec{N} \cdot \vec{L})C_L\)
- (where \(N\) and \(L\) are unit vectors)
Ambient light

- With this mechanism, the light in a shadowed region is 0 (black)
- To avoid this, use “ambient” lighting

\[ C_{\text{ray}} = C_{\text{surface}} \left[ (\vec{N} \cdot \vec{L})C_{\text{Light}}K_d + C_{\text{ambient}}K_a \right] \]
Two-sided lighting

- What if light hits the back of a polygon
- Options:
  - Black on back
  - Different materials for front/back
  - Two sided lighting:
    - okay: $|\mathbf{N} \cdot \mathbf{L}|$
    - better: Negate $\mathbf{N}$ if $\mathbf{N} \cdot \mathbf{V} > 0$
Two-sided lighting

- You might consider checking if the light is on the “right side” of the object

\[
sign(-\vec{V} \cdot \vec{N}) = sign(\vec{L} \cdot \vec{N})
\]

\[
(\vec{V} \cdot \vec{N})(\vec{L} \cdot \vec{N}) < 0: \text{lit}
\]

\[
(\vec{L} \cdot \vec{N}) > 0: \text{lit (when normal flipped)}
\]

- Can avoid casting shadow rays too
- Or use absolute value for \(\vec{L} \cdot \vec{N}\)
Lambertian Shading

Compute hit position \( \mathbf{P} = \mathbf{O} + t \mathbf{V} \)
Call primitive to get normal \( \mathbf{N} \) (normalized)
\[
\text{costheta} = \mathbf{N} \cdot \mathbf{V}
\]
\[
\text{if}(\text{costheta} < 0)
\]
\[
\text{normal} = -\text{normal}
\]
Color light = scene.ambient*Ka
foreach light source
get \( \mathbf{C_L} \) and \( \mathbf{L} \)
\[
\text{dist} = \| \mathbf{L} \|, \mathbf{L}_n = \frac{\mathbf{L}}{\| \mathbf{L} \|}
\]
\[
\text{cosphi} = \mathbf{N} \cdot \mathbf{L}_n
\]
\[
\text{if}(\text{cosphi} > 0)
\]
\[
\text{if}(!\text{intersect with } 0 < t < \text{dist})
\]
\[
\text{light} += \mathbf{C_L} \ast (Kd \ast \text{cosphi})
\]
result=light*surface color
Planes

• The equation for a plane is: \( ax + by + cz + d = 0 \)

• A plane can be defined with a Vector (the normal to the plane) and a point on the plane:

  \[
  a = N_x; b = N_y; c = N_z
  
  d = -\vec{N} \cdot \vec{P}
  \]

• Alternative form of plane equation:

  \( \vec{N} \cdot \vec{P} + d = 0 \)
Ray-plane intersection

• To find the intersection of a ray with a plane, determine where both equations are satisfied at the same time:

\[ \mathbf{N} \cdot \mathbf{P} + d = 0 \text{ and } \mathbf{P} = \mathbf{O} + t\mathbf{V} \]
To find the intersection of a ray with a plane, determine where both equations are satisfied at the same time:

\[ \vec{N} \cdot \vec{P} + d = 0 \quad \text{and} \quad \vec{P} = \vec{O} + t\vec{V} \]

\[ \vec{N} \cdot (\vec{O} + t\vec{V}) + d = 0 \]
Ray-plane intersection

- To find the intersection of a ray with a plane, determine where both equations are satisfied at the same time:

\[ \vec{N} \cdot \vec{P} + d = 0 \quad \text{and} \quad \vec{P} = \vec{O} + t\vec{V} \]

\[ \vec{N} \cdot (\vec{O} + t\vec{V}) + d = 0 \]

\[ \vec{N} \cdot \vec{O} + t\vec{N} \cdot \vec{V} + d = 0 \]
Ray-plane intersection

- To find the intersection of a ray with a plane, determine where both equations are satisfied at the same time:

\[
\vec{N} \cdot \vec{P} + d = 0 \quad \text{and} \quad \vec{P} = \vec{O} + t\vec{V} \\
\vec{N} \cdot (\vec{O} + t\vec{V}) + d = 0 \\
\vec{N} \cdot \vec{O} + t\vec{N} \cdot \vec{V} + d = 0 \\
t\vec{N} \cdot \vec{V} = -(d + \vec{N} \cdot \vec{O})
\]
Ray-plane intersection

- To find the intersection of a ray with a plane, determine where both equations are satisfied at the same time:

\[
\vec{N} \cdot \vec{P} + d = 0 \quad \text{and} \quad \vec{P} = \vec{O} + t\vec{V}
\]

\[
\vec{N} \cdot (\vec{O} + t\vec{V}) + d = 0
\]

\[
\vec{N} \cdot \vec{O} + t\vec{N} \cdot \vec{V} + d = 0
\]

\[
t\vec{N} \cdot \vec{V} = -\left(d + \vec{N} \cdot \vec{O}\right)
\]

\[
t = -\frac{\left(d + \vec{N} \cdot \vec{O}\right)}{\vec{N} \cdot \vec{V}}
\]
Ray-plane intersection

- If \( \vec{N} \cdot \vec{V} = 0 \) (or close to it) then the ray is parallel to the plane.
- The parameter \( t \) defines the point where the ray intersects the plane.
- To determine the point of intersection, just plug it back into the ray equation:

\[
(\vec{O} + t\vec{V})
\]