Ray tracing

- Simple ray casting
- Recursive ray tracing
- Modeling transforms
- Cheap tricks
- Optimizations

Ray tracing

- Cast ray through pixel into scene
- Find closest intersection (if any)
- Compute luminance at intersection
  - Direct illumination
  - Reflections
  - Refraction

Specular reflections

- Cast ray reflected at P into scene
- Find closest intersection point P' (if any)
- Compute luminance at P'
- Scale by m_{r,g,b} and add to luminance at P

Transmission

- Cast ray transmitted at P into scene
- Find closest intersection point P' (if any)
- Compute luminance at P'
- Scale by k_{trans} and add to luminance at P

Transmission

- Cast ray transmitted at P into scene
- Find closest intersection point P' (if any)
- Compute luminance at P'
- Scale by k_{trans} and add to luminance at P

Refraction - Snell's Law

- Cast ray transmitted at P into scene
- Find closest intersection point P' (if any)
- Compute luminance at P'
- Scale by k_{trans} and add to luminance at P

- Cast ray reflected at P into scene
- Find closest intersection point P' (if any)
- Compute luminance at P'
- Scale by m_{r,g,b} and add to luminance at P
thin surface refraction

ignore Snell's law

incoming ray \((P_0, v)\)

transmitted ray \((P, v)\)

\(P\)

thick surface refraction

\( \theta_{\text{in}} \) satisfies: \( \eta_{\text{out}} \sin \theta_{\text{out}} = \eta_{\text{in}} \sin \theta_{\text{in}} \)

\( \nu_{\text{out}} = (\nu_{\text{in}} (1 - \beta^2 \sin^2 \theta_{\text{in}}))^\frac{1}{2} n + \beta \nu_{\text{in}} \) where \( \beta = \frac{\eta_{\text{in}}}{\eta_{\text{out}}} \)

thick surface recursion

incoming ray \((P_0, v)\)

transmitted ray \((P, v)\)

\(P\)

What is direct illumination at \(P'\)?

implementation issues

offset new ray slightly to make sure you don't find \(P\) again!!!

stopping conditions

recurse until:

a) maximum recursive depth specified by user is reached

b) contribution to luminance is less than user specified bound

stoping conditions

recurse until:

a) maximum recursive depth specified by user is reached

b) contribution to luminance is less than user specified bound

• cast new ray from \(P\) into scene

• find closest intersection point \(P'\) (if any)

• compute luminance at \(P'\)

• scale and add to luminance at \(P\)
ray tracing
- simple ray casting
- recursive ray tracing
- modeling transforms
- cheap tricks
- optimizations

modeling transforms
\[ M \mathbf{p} = \mathbf{p}' \]

object description: \( M, \mathbf{p} \) or \( \mathbf{p}' \)?

ray tracing
- cast ray into scene
- find intersection point (if any) that is closest to eye
- compute luminance at intersection

intersection
1. find intersection with transformed primitive \( \text{Hard!} \)
   OR
2. convert ray to local coordinate system & test for intersection with (non-transformed) primitive

transform
- Points
- Vectors
- Rays

Done!!

transforms: vector
\( \mathbf{v} = \mathbf{p} - \mathbf{q} \) and \( M' \mathbf{v} = M' (\mathbf{p} - \mathbf{q}) = M' \mathbf{p} - M' \mathbf{q} \)

Warnings:
- because of translation we can’t ignore \( \mathbf{q} = (0,0,0) \)
- re-unitize unit vectors
transforms

- Points
- Vectors
- Rays

transforms: ray

- Points
- Vectors
- Rays:
  \[ r = (p, v) \text{ and } M'r = (M'p, M'v) \]

modeling transforms

\[ R = M'R' \]

\[ M p = p' \]

\[ R = M p \]

\[ R' \]

model coordinates

world coordinates

\[ R' = M^{-1}R \]

\[ M = M^{-1} \]

\[ M^{-1}_1 \quad M^{-1}_2 \quad M^{-1}_3 \]

single transform

- scale by \( s \)
- rotate by \( \theta \)
- translate by \( \Delta \)

composite transform

\[ (M_1M_2...M_n)^{-1} \]

\[ M_n^{-1} \quad M_2^{-1} \quad M_1^{-1} \]

ray tracing

- cast ray into scene
- find intersection point (if any) that is closest to eye
- compute luminance at intersection
- compute distance to eye in world coordinates

ray tracing

- cast ray into scene
- find intersection point (if any) that is closest to eye
- compute luminance at intersection

Luminance depends on geometry so compute in world coordinates. Alas, need intersection point and surface normal!
transforms
- Points
- Vectors
- Rays
- Surface normal

Normal of a transformed surface: not the transformed surface normal!

transforms: surface normal

WHY? Because it works!

modeling transforms

Object specification
- World coordinates
- Object coordinates and modeling transform
- Hierarchical coordinates

transform composition

hierarchical coordinates
hierarchical coordinates

body xfm
body description
  head translate wrt body
  head rotate
  head description
    eye1 translate wrt head
    eye1 scale
    eye1 description

order of operations

body xfm
body description
  head translate wrt body
  head rotate
  head description
    (eye1 translate wrt head
     eye1 scale
     eye1 description)
    (eye2 translate wrt head
     eye2 scale
     eye2 description)
hierarchical coordinates

body xfm
body description
  head translate wrt body
  head rotate
  head description
    (eye1 translate wrt head
eye1 scale
eye1 description)
    (eye2 translate wrt head
eye2 scale
eye2 description)

hierarchical coordinates

body xfm
body description
  head translate wrt body
  head rotate
  head description
    (eye1 translate wrt head
eye1 scale
eye1 description)
    (eye2 translate wrt head
eye2 scale
eye2 description)

Ray tracing

- cast ray into scene
- find intersection point (if any) that is closest to eye
- compute luminance at intersection

Intersection

1. find intersection with transformed primitive
   
2. convert ray to local coordinate system & test for intersection with (non-transformed) primitive
intersection

body xfm ← body description
head translate wrt body
head rotate
head description
  (eye1 translate wrt head
eye1 scale
eye1 description)
  (eye2 translate wrt head
eye2 scale
eye2 description)

body xfm ← Apply inverse transform to ray
test for body intersection

head translate wrt body
head rotate
head description
  (eye1 translate wrt head
eye1 scale
eye1 description)
  (eye2 translate wrt head
eye2 scale
eye2 description)

Apply inverse transforms to R' → R''

Test head

Ray tracing

• simple ray casting
• recursive ray tracing
• modeling transforms
• cheap tricks
• optimizations