Ray tracing

- simple ray casting
- recursive ray tracing
- cheap tricks
- optimizations

Cheap tricks

- texture mapping

Texture mapping

"glue" image to surface

Steve Yan, fall 2001

Drew Levin, fall 2001
texture mapping

\[ f(p): \text{coordinates in the texture map corresponding to surface point } p \]

texture mapping triangle

Input specifies texture coordinates of triangle vertices

\[ (0,1) \quad (1,1) \]
\[ (0,0) \quad (1,0) \]

triangle: parametric form

A point \( q \) on the triangle \( T \) can be uniquely represented as
\[ q = t_0 + \beta u + \gamma w \]
where \( \beta \geq 0 \), \( \gamma \geq 0 \), \( \beta + \gamma \leq 1 \)

computing \( f(p) \)

\[ f(t_2) \]
\[ f(t_0) \]
\[ f(u) \]
\[ f(w) \]

\[ f(u) = f(t_1) - f(t_0) \]
\[ f(w) = f(t_2) - f(t_0) \]
computing \( f(p) \)

\[
\begin{align*}
t_0 + \beta u + \gamma w &= t_2 \\
\end{align*}
\]

compute \( f(p) = f(\alpha) + \beta f(u) + \gamma f(w) \)

texture mapping triangle

What is image color at \( f(p) \)?

Need to resample! For your ray tracer use bilinear interpolation.

color

for each channel we'll approximate the color at the intersection point as the sum of five terms
- emission
- ambient reflection
- diffuse reflection
- specular reflection
- specular transmission

scale these by texture color

texture mapping sphere

e.g. latitude/longitude

conceptually: wrap texture

top down view

DO THIS!
conceptually: wrap texture
top down view  
+y is coming towards us

\[ \pi r^2 \]

\[ \pi r \]

\[-z\]

conceptually: wrap texture
top down view  
+y is coming towards us

\[ \pi r^2 \]

\[ \pi r \]

\[-z\]

conceptually: wrap texture
front view  
+z is coming towards us

\[ u=0 & u=1 \]

\[ v=0.75 \]

\[ u=0.25 \]

\[ u=0.5 \]

\[ u=0.75 \]

texture mapping sphere
intersection point \( P \)
has texture coordinate \((u,v)\)

\[ x \]

\[ y \]

\[ -z \]

\[ (0,0) \]  \[ (1,0) \]

\[ (0,1) \]  \[ (1,1) \]

texture mapping sphere
intersection point \( P \)
has texture coordinate \((u,v)\)

\[ x \]

\[ y \]

\[ -z \]

\[ (0,0) \]  \[ (1,0) \]

\[ (0,1) \]  \[ (1,1) \]
texture mapping sphere

Parameterization

Assume $P = C + ax + by + gz$

Let the angle between $P-C$ and $z$ be $\theta$

What is the relationship between $g$ and $\theta$?

Parameterization

Assume $P = C + ax + by + gz$

Let the angle between $P-C$ and $z$ be $\theta$

Implementation note: $\arccos \theta \in [0,\pi]$. 

Parameterization

Assume $P = C + ax + by + gz$

Let the angle between $P-C$ and $z$ be $\theta$

\[-g = r \cos \theta. \quad \text{So...}\]

If $g = 0$ then $\theta = \pi/2$.
If $g \neq 0$ then $\theta = \arccos(-g/r)$.

Parameterization

Assume $P = C + ax + by + gz$

Let the angle between $P-C$ and $z$ be $\theta$

Implementation note: $\arccos \theta \in [0,\pi]$. 

Texture Mapping Sphere

Intersection point $P = C + ax + by + gz$ has texture coordinate $(u,v)$ where $v =$______

Texture Mapping Sphere

Intersection point $P = C + ax + by + gz$ has texture coordinate $(u,v)$ where $v =$______
The intersection point \( P = C + ax + by + az \) has texture coordinate \((u, v)\) where \( v = \theta / \pi \).

Implementation note: \( \arccos \) returns \( \theta \in [0, \pi] \).

Parameterization:
If \( \theta \neq 0 \) and \( \theta \neq \pi \) then let \( P' = (C - az) + ax + by \) be the projection of \( P - C \) on the x-y plane.

Since \( P = C + ax + by + az \)
\( P' = C + ax + by \)

What is length of \( P' \)?
\( r' = r \sin \theta \)
Let $\phi$ be the angle from $y$ to $P'$-C CCW in the $x-z$ plane.

**Parameterization in $x$-$z$ plane**

Intersection point $P = C + ax + by + gz$ has texture coordinate $(u, v)$ where $v = \theta / \pi$ if $\theta = 0$ or $\theta = \pi$ then $u = 1/2$ otherwise $u = \phi/(2\pi)$.

**Texture mapping sphere**

- Use good textures
- Overlap & blend or mix
- Don't look there
- 3D textures
3d textures

use stack of images

how do we generate these images?

procedural textures: we'll come back to this

cheap tricks

- texture mapping
- bump mapping

creating bumpy surfaces

adrian mettler, spring 2003

bump mapping vs texture mapping

- bump mapping is computationally more difficult than texture mapping
- bump mapping creates a better 3d look than texture mapping

smooth vs. bumpy surface models

smooth surface

bumpy surface

\[ q = p + f(p)n \]

bumpy surface

\[ q = p + f(p)n \]

displacement mapping

displacement map=height field
displacement mapping problem

Bump mapping problem:
- where is intersection?

bumpy surfaces

A surface appears to be bumpy because:
- jagged silhouette
- surface normals fluctuate dramatically

Bump mapping: simulate this by perturbing normals in the lighting calculations

bump mapping

Smooth surface

Bump mapped surface

Use smooth surface intersections but normals of a bumpy surface

bump mapping: computing \( n_q \)

1. Find vectors \( v_0 \) and \( v_1 \) in plane tangent to bumpy surface at point \( q \) (i.e. take derivatives)
2. \[ n_q = \frac{v_0 \times v_1}{||v_0 \times v_1||} \]

2D parameterization of smooth surface

In general \( p \rightarrow (u,v) \)
find vectors in tangent plane

take partial derivatives of $Q(u,w)=P(u,w)+f(u,w)n(u,w)$ with respect to $u,w$

\[
\frac{dQ}{du} = \frac{dP}{du} + \frac{df(u,w)}{du}n(u,w) + f(u,w)\frac{dn(u,w)}{du}
\]

\[
\frac{dQ}{dw} = \frac{dP}{dw} + \frac{df(u,w)}{dw}n(u,w) + f(u,w)\frac{dn(u,w)}{dw}
\]

triangle: parametric form

\[
\frac{dP}{du} = \lim_{||u|| \to 0} \left[ \frac{(t_0 + \beta u + \gamma w) - (t_0 + \beta u + \gamma w)}{||u||} \right] = \beta u
\]

where: $u = (t_1-t_0)/||t_1-t_0||$ is a unit vector in the $t_1-t_0$ direction and $w = (t_2-t_0)/||t_2-t_0||$ is a unit vector in the $t_2-t_0$ direction

bump map derivative

convolution kernel

\[
\begin{array}{ccc}
-1 & 1 & 1 \\
-1 & 0 & 1 \\
-1 & 1 & -1 \\
\end{array}
\]

We can compute $\frac{dP}{du}$ and $\frac{dP}{dw}$ for our surfaces.
find vectors in tangent plane

- Take partial derivatives of \( Q(u,w) = P(u,w) + f(u,w)n(u,w) \) with respect to \( u, w \)

\[
\begin{align*}
\frac{dQ}{du} &= \frac{dP}{du} + \frac{df}{du}n(u,w) + f(u,w)\frac{dn}{du} \\
\frac{dQ}{dw} &= \frac{dP}{dw} + \frac{df}{dw}n(u,w) + f(u,w)\frac{dn}{dw}
\end{align*}
\]

- How does normal change with respect to changes in \( u \) and \( w \)

We'll ignore this because

(a) it is small,

(b) it is computationally difficult, and

(c) results look ok if we do.

bump mapping

- Computing \( \mathbf{n}_q \):
  1. Find vectors \( \mathbf{v}_0 \) and \( \mathbf{v}_1 \) in plane tangent to bumpy surface at point \( q \)
  2. \( \mathbf{n}_q = (\mathbf{v}_0 \times \mathbf{v}_1)/|\mathbf{v}_0 \times \mathbf{v}_1| \)

take the cross product

- Take cross product

\[
\frac{dQ}{du} \times \frac{dQ}{dw} = \frac{dP}{du} \times \frac{dP}{dw} + \frac{df}{du}n(u,w) \times \frac{dn}{du} + \frac{df}{dw}n(u,w) \times \frac{dn}{dw}
\]

Computation

1. Compute derivatives of surface
   - \( \frac{dP}{du} \) and \( \frac{dP}{dw} \)

2. Compute derivatives of bump map
   - \( b_u(u,v) \) and \( b_v(u,v) \)

3. Take cross products and add:
   - \( \frac{dP}{du} \times \frac{dP}{dw} = \frac{df}{du}n(u,w) \times \frac{dn}{du} + \frac{df}{dw}n(u,w) \times \frac{dn}{dw} \)
procedural texture mapping

- procedure returns a texture color for any point in 3d space (note this is not an image stack)
- sample to find texture for surface

procedural textures

- advantages
  - don't need to find a mapping from a (complex) 3d surface to a 2d texture image
  - concise representation of texture
- disadvantages
  - ad hoc techniques cannot duplicate photographs

perlin noise - 1D example

step 1: generate discrete noise function with specified length, amplitude, sampling frequency

example: length=8, amplitude = 3, sampling frequency is 7 Hz.

seed → random number generator → r₀, r₁, ..., r₇ → rᵢ ∈ [0,1]

step 2: interpolate with smoothing
**Perlin Noise - 1D Example**

- Step 3: Repeat with various amplitudes/frequencies
- Step 4: Add together

**Perlin Noise - 2D Example**

For more info see Perlin Noise link on proj2 web site.

**Cheap Tricks**

- Texture mapping
- Procedural texture mapping
- Bump mapping
- Transparency mapping
- Depth of field
- Lens effects
- Jittering
- Soft shadows

**Cheap Tricks**

- Texture mapping
- Procedural texture mapping
- Bump mapping
- Transparency mapping
- Depth of field
- Lens effects
- Jittering
- Soft shadows

**Cheap Tricks**

- Texture mapping
- Procedural texture mapping
- Bump mapping
- Transparency mapping
- Depth of field
- Lens effects
- Jittering
- Soft shadows

*Blur based on distance from viewpoint*

**Cheap Tricks**

- Texture mapping
- Procedural texture mapping
- Bump mapping
- Transparency mapping
- Depth of field
- Lens effects
- Jittering
- Soft shadows

*See paper mentioned in assignment*
cheap tricks

- texture mapping
- procedural texture mapping
- bump mapping
- transparency mapping
- depth of field
- lens effects
- jittering
- soft shadows

jittering: anti-aliasing technique

Run rt for example

random ray

Shoot ray \( (i,j) \) through \( P_{ij} + (dx, dy) \) where:
- \( dx \) is chosen uniformly at random over \([-W/(2w), W/(2w)]\)
- \( dy \) is chosen uniformly at random over \([-H/(2h), H/(2h)]\)

soft shadows

Run rt for examples
soft shadows

In reality, we don't have point lights!!

compute random point on "light" and cast shadow ray in that direction

---

ray tracing

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

---

ray tracing complexity

\[ O(\text{# of intersection tests}) = O(\text{#pixels} \times \text{# objects}) \]

Can we reduce the number of intersection tests?

---

optimization

• bounding boxes
• oct-trees
• BSP-trees

---

bounding boxes: intuition

10,000,000 triangles

most rays miss the object

---

bounding boxes

1. rule out rays by simple intersection test with bounding box
2. perform exhaustive test on remaining rays
bounding boxes & 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)

transformed box defined by transformed corners $(x_{min}, y_{min}, z_{min}), (x_{max}, y_{max}, z_{max})$

box defined by extrema of primitive:
- $x_{min}$
- $x_{max}$
- $y_{min}$
- $y_{max}$
- $z_{min}$
- $z_{max}$

compute extrema based on local primitive and the extrema of children's transformed bounding boxes
bounding boxes & 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)

bounding box based on head and bounding boxes of eyes

bounding boxes & 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)

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)

Apply inverse transform to ray
R'

Bounding box is axis-aligned!!!
bounding box intersection

1. find intersection of ray with each bounding plane of box

front plane intersection:
\[ q = p + \alpha v = (?, ?, \text{front}) \]
where \( \alpha = (\text{front} - p_z)/v_z \)

bounding box intersection test

if ray intersects with node’s bounding check for intersection with local object and box pass ray to children otherwise return no intersection

REMEMBER: bounding box encloses objects at node and all bounding boxes of children

optimization

- bounding boxes
- oct-trees
- BSP-trees

We won’t cover these. Look on the web for details.

ray casting

1. cast ray through pixel into scene

1. cast ray through pixel into scene
ray tracing
- ray casting
  - rays
  - intersection tests
  - intersection with scene graph
  - lighting and material properties
- recursive ray tracing
- cheap tricks
- optimizations

red transmission term
the red transmission term is $k_{\text{trans}} \sum R_{L,T}$ where
- the summation is taken over all lights $L$
- $R_{L,T}$ is the intensity of the red transmission of light $L$ at the intersection point

transmission
\[ R_{L,T} = k_{\text{trans}} R_{L,b} \]
where
- $R_{L,b}$ is the luminance of diffuse reflections on the back of the surface

transmission computation
- compute $R_{L,b}$ with the reverse normal.
- scale by $k_{\text{trans}}$

note: thick surfaces
all light falling on the back of the surface is occluded

find vectors in tangent plane
take partial derivatives of $Q$ in two directions
find vectors in tangent plane

- Take partial derivatives of $Q = f(P)$ in two directions:
  - $dQ / du = dP / du = d(f(P)) / du$

what directions?

snell’s law

- Refractive index $n_{in}$
- Refractive index $n_{out}$
- $\theta_{in}$ satisfies: $n_{out} \sin \theta_{out} = n_{in} \sin \theta_{in}$
- If $1 - \beta^2 \sin^2 \theta_{in} > 0$ where $\beta = n_{in} / n_{out}$ then
  - $\theta_{out} = \arcsin \left( \frac{1 - \beta^2 \sin^2 \theta_{in}}{n_{out}} \right) n + \theta_{in}$

parameterization

- $p - c = \alpha x + \beta y + \gamma z$
- Assume $p - c = \alpha x + \beta y + \gamma z$
- Now represent $p - c$ in spherical coordinates

- $\theta = \arcsin (\beta / r)$
- Note range of $\arcsin$: $\theta \in [-\pi/2, \pi/2]$
- e.g. if $\theta = 0$ then $\alpha = r$
- $\phi = \frac{\arccos (\alpha / r)}{\arccos}$
  - else (undefined)
  - $\phi = 0$
- Note range of $\arccos$: $\phi \in [0, \pi]$
  - e.g. if $\phi = 0$ and ...
  - $\phi = \arccos (\frac{\alpha}{r})$
  - $\phi = 0$ then $\alpha = r$
2d parameterization of smooth surface

define unit vectors \( u, w \)

\( u \): direction of constant \( \phi \)

\( w \): direction of constant \( \theta \)

\( P \rightarrow (\phi/2\pi, \theta/\pi) \)

for example

\[
\begin{align*}
P &= t_0 + (t_1 - t_0)u + (t_2 - t_0)w \\
&= t_0 + \beta u + \gamma w
\end{align*}
\]

WARNING: renaming

2d parameterization of smooth surface

\( P(u,w) \)

\( n(u,w) = n(p(u,w)) \)

2d parameterization of bumpy surface

\( Q(u,w) = P(u,w) + f(u,w)n(u,w) \)

bump mapping: computing \( n_q \)

bumpy surface

\( n_q = (v_0 \times v_1)/|v_0 \times v_1| \)

We need to do this!!

1. find vectors \( v_0 \) and \( v_1 \) in plane tangent to bumpy surface at point \( q \)

2. \( n_q = (v_0 \times v_1)/|v_0 \times v_1| \)

find vectors in tangent plane

take partial derivatives of \( Q(u,w) = P(u,w) + f(u,w)n(u,w) \) with respect to \( u, w \)

\( f(u,w) \)

\( Q(u,w) \)

(0,0,0)
\[
dQ(u,v)/du = \lim_{\delta u \to 0} Q(u,w) + \delta u Q(u,w)
\]

find vectors in tangent plane

\[
dQ/du = dP/du + \left[ df(u,w)/du \right] n(u,w) + f(u,w) d/du n(u,w)
\]

dQ/dw = dP/dw + \left[ df(u,w)/dw \right] n(u,w) + f(u,w) d/dw n(u,w)

find vectors in tangent plane