Computer Graphics
Lecture 11
Acceleration Structures
Advanced Ray Tracing
Announcements
Announcements

• Feedback survey out this afternoon - please respond by Monday night (10pm)
Announcements

• Feedback survey out this afternoon - please respond by Monday night (10pm)

• A1 grading should be done by Monday.
Announcements

- Feedback survey out this afternoon - please respond by Monday night (10pm)

- A1 grading should be done by Monday.

- Final projects - proposals will be due in ~2 weeks; start thinking about topics now. More on this later.
Today

• A high-level overview of what comes next in ray tracing.

• Useful for A2 extensions and/or final project ideas.

• Not getting into gory detail - see the book references on the slides.
Barycentric ray-triangle intersection

- Every point on the plane can be written in the form:
  \[ a + \beta(b - a) + \gamma(c - a) \]
  for some numbers \( \beta \) and \( \gamma \).
- If the point is also on the ray then it is
  \[ p + td \]
  for some number \( t \).
- Set them equal: 3 linear equations in 3 variables
  \[ p + td = a + \beta(b - a) + \gamma(c - a) \]
  …solve them to get \( t, \beta, \) and \( \gamma \) all at once!
Barycentric ray-triangle intersection

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

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

\[
\begin{bmatrix}
    a - b & a - c & d
\end{bmatrix}
\begin{bmatrix}
    \beta \\
    \gamma \\
    t
\end{bmatrix}
= \begin{bmatrix}
    a - p
\end{bmatrix}
\]

\[
\begin{bmatrix}
x_a - x_b & x_a - x_c & x_d \\
y_a - y_b & y_a - y_c & y_d \\
z_a - z_b & z_a - z_c & z_d
\end{bmatrix}
\begin{bmatrix}
    \beta \\
    \gamma \\
    t
\end{bmatrix}
= \begin{bmatrix}
x_a - x_p \\
y_a - y_p \\
z_a - z_p
\end{bmatrix}
\]

• This is a linear system: \( Ax = b \)
• Various ways to solve, but a fast one uses Cramer's rule.
• See 4.4.2 for the TL;DR formula
• See 5.3.2 for an explanation of Cramer's rule
Ray tracing is expensive.

for each pixel:
  for each triangle:
    compute barycentric intersection

How expensive? Let's (informally) count some FLOPs.

floating-point operations
Last time: barycentric ray-triangle intersection

\[ \mathbf{p} + t \mathbf{d} = \mathbf{a} + \beta (\mathbf{b} - \mathbf{a}) + \gamma (\mathbf{c} - \mathbf{a}) \]

\[ \beta (\mathbf{a} - \mathbf{b}) + \gamma (\mathbf{a} - \mathbf{c}) + t \mathbf{d} = \mathbf{a} - \mathbf{p} \]

\[
\begin{bmatrix}
\mathbf{a} - \mathbf{b} & \mathbf{a} - \mathbf{c} & \mathbf{d}
\end{bmatrix}
\begin{bmatrix}
\beta \\
\gamma \\
t
\end{bmatrix}
= 
\begin{bmatrix}
\mathbf{a} - \mathbf{p}
\end{bmatrix}
\]

\[
\begin{bmatrix}
x_a - x_b & x_a - x_c & x_d \\
y_a - y_b & y_a - y_c & y_d \\
z_a - z_b & z_a - z_c & z_d
\end{bmatrix}
\begin{bmatrix}
\beta \\
\gamma \\
t
\end{bmatrix}
= 
\begin{bmatrix}
x_a - x_p \\
y_a - y_p \\
z_a - z_p
\end{bmatrix}
\]

9 subtractions

Pre-calculate entries and rename:

\[
\begin{bmatrix}
a & d & g \\
b & e & h \\
c & f & i
\end{bmatrix}
\begin{bmatrix}
\beta \\
\gamma \\
t
\end{bmatrix}
= 
\begin{bmatrix}
j \\
k \\
l
\end{bmatrix}
\]
Barycentric Ray-Triangle Intersection

Cramer’s rule gives us

\[ \beta = \frac{j(ei-hf)+k(gf-di)+l(dh-eg)}{M}, \]

\[ \gamma = \frac{i(ak-jb)+h(jc-al)+g(bl-kc)}{M}, \]

\[ t = -\frac{f(ak-jb)+e(jc-al)+d(bl-kc)}{M}, \]

where

Reusing from above:

3 mult \[ M=a(ei-hf)+b(gf-di)+c(dh-eg). \]

5 add/sub

10 mult/div
Barycentric Ray-Triangle Intersection

Cramer’s rule gives us

\[
\beta = \frac{j(ei-hf)+k(gf-di)+l(dh-eg)}{M},
\]

\[
\gamma = \frac{i(ak-jb)+h(jc-al)+g(bl-kc)}{M},
\]

\[
t = -\frac{f(ak-jb)+e(jc-al)+d(bl-kc)}{M},
\]

where

Reusing from above:

3 mult \[ M = a(ei - hf) + b(gf - di) + c(dh - eg). \]
Barycentric Ray-Triangle Intersection

Cramer’s rule gives us

\[ \beta = \frac{j(ei-hf)+k(gf-di)+l(dh-eg)}{M} , \]

\[ \gamma = \frac{i(ak-jb)+h(jc-al)+g(bl-kc)}{M} , \]

\[ t = -\frac{f(ak-jb)+e(jc-al)+d(bl-kc)}{M} , \]

where

Reusing from above:

3 mult

\[ M = a(ei-hf)+b(gf-di)+c(dh-eg) . \]

Assume, conservatively that on average, we calculate \( \beta \) and determine that it doesn't intersect (because \( \beta < 0 \) or \( \beta > 1 \))

Total: 27 FLOPs
Ray tracing is expensive.

for each pixel: $720p = 1280 \times 720 = 921600$ pixels

for each triangle: bunny: 114 triangles
compute barycentric intersection 27 flops

$= 2,836,684,800$
$= 2.8$ GFLOPs

A typical laptop can currently can do about 100-200 GFLOPS gigaflops per second
Ray tracing is expensive.

For each pixel: \(720p = 1280 \times 720 = 921600\) pixels

For each triangle:
- bunny: 114 triangles
- compute barycentric intersection: 27 flops

\[= 2,836,684,800\]
\[= 2.8\] GFLOPs

A typical laptop can currently can do about 100-200 GFLOPS gigaflops per second

https://polycount.com/discussion/141061/polycounts-in-next-gen-games-thread
Ray tracing is expensive.

for each pixel: 720p = 1280×720 = \textbf{921600} pixels

for each triangle: bunny: 114 triangles
compute barycentric intersection 27 flops

= 2,836,684,800
= 2.8 GFLOPs

A typical laptop can currently can do about 100-200 GFLOPS gigaflops per second

so what's the problem?

https://polycount.com/discussion/141061/polycounts-in-next-gen-games-thread
Ray tracing is expensive.

for each pixel: \(720p = 1280 \times 720 = 921600\) pixels

for each triangle: computer game model: \(40k\) triangles

compute barycentric intersection \(27\) flops

\[
\begin{align*}
&= 995,328,000,000 \\
&= 995 \text{ GFLOPs} \\
&\approx 1 \text{ TFLOP}
\end{align*}
\]
Ray tracing is expensive.

for each pixel: 720p = 1280×720 = 921600 pixels
for each triangle: computer game model: 40k triangles
compute barycentric intersection 27 flops

= 995,328,000,000
= 995 GFLOPs
~= 1 TFLOP

Want to render this for an interactive game?
Ray tracing is expensive.

for each pixel: \( 720p = 1280 \times 720 = 921600 \) pixels

for each triangle: computer game model: \( 40k \) triangles

compute barycentric intersection \( 27 \) flops

\[ = 995,328,000,000 \]
\[ = 995 \text{ GFLOPs} \]
\[ \approx 1 \text{ TFLOP} \]

Want to render this for an interactive game? Simply do this \( 30+ \) times per second.
What can we do?
What can we do?

- Optimize the inner-inner loop: more efficient intersection routines
What can we do?

• Optimize the inner-inner loop: more efficient intersection routines

• Carefully reduce triangle count
What can we do?

- Optimize the inner-inner loop: more efficient intersection routines
- Carefully reduce triangle count
  these only go so far...
What can we do?

- Optimize the inner-inner loop: more efficient intersection routines
- Carefully reduce triangle count
  
  these only go so far...

- Intersect fewer things
  
  - Most ray intersections don't hit the object!
  
  - Basic strategy: efficiently find big chunks of the scene that definitely don't intersect your ray
Bounding Volumes

- Quick way to avoid intersections: bound object with a simple volume
  - Object is fully contained in the volume
  - If it doesn’t hit the volume, it doesn’t hit the object
  - So test bvol first, then test object if it hits

![Images of a sphere, axis-aligned box, and oriented box]

sphere  axis-aligned box  oriented box

[Glassner 89, Fig. 4.5]
Bounding Volumes

Algorithm:

if ray intersects bounding volume:
  if ray intersects object:
    do stuff

sphere  axis-aligned box  oriented box
Bounding Volumes

Algorithm: if ray intersects bounding volume:
            if ray intersects object:
                do stuff

Cost: more for hits and near misses, but less for far misses

Is this worth it?

• bvol intersection should be much cheaper than object intersection
  • works best for simple bvols, complicated objects
  • bvol should bound object as tightly as possible

Tradeoff: efficient intersection vs tightness
Bounding Volume Intersection

Exercise: In 2D, devise an algorithm to intersect a ray with an **axis-aligned bounding box**.

Inputs:
- ray (p and d)
- left_x
- right_x
- left_y
- right_y

Output: boolean, whether ray hits box
Bounding Volumes

Algorithm: if ray intersects bounding volume:
if ray intersects object:
do stuff

Cost: more for hits and near misses, but less for far misses

Is this worth it?

- bvol intersection should be much cheaper than object intersection
  - works best for simple bvols, complicated objects
- bvol should bound object as tightly as possible

Tradeoff: efficient intersection vs tightness
Bounding Volume Hierarchy

• Bvols around objects *might* help
• Bvols around groups of objects *will* help
• Bvols around parts of complex objects will help
• Idea: use bounding volumes all the way from the whole scene down to groups of a few objects
Building the Hierarchy

- Ideally: bound nearby clusters of objects
- Practical solution: partition along axis
BVH construction example
BVH construction example
BVH construction example
BVH construction example
BVH ray-tracing example
BVH ray-tracing example
BVH ray-tracing example
BVH ray-tracing example
BVH ray-tracing example
BVH ray-tracing example
BVH ray-tracing example
BVH ray-tracing example
BVH ray-tracing example
BVH ray-tracing example
Implementation

• New kind of object: BoundedObject
  • stores references to contained objects (can be BoundedObjects themselves!)

• New `ray_intersect` routine:
  • Intersect with each child; if any, return closest.
Other Approaches:

- Uniform Space Subdivision
Uniform Space Subdivision

• Grid cells store references to overlapping objects
Compute the grid cells intersected by a ray

Constant offset between cell edge intersections in each dimension:
Ok, what else can't we do?

• Rotate, scale, shear objects - *transformations* (more on this next week, and in 13.2)

• Render transparent things - *transmission and refraction* (Ch 13.1)

• Intersect more kinds of objects - *Constructive Solid Geometry* (Ch

• Area light sources, soft shadows, depth of field - *distribution ray tracing* (Ch 13.4)
Transformations and Instancing

• Next week we'll talk about how to transform objects:
Transformations and Instancing

Next week we'll talk about how to transform objects:

When ray tracing, we can alternatively transform the rays:
Transformations and Instancing

Next week we'll talk about how to transform objects:

When ray tracing, we can alternatively transform the rays:

Same idea allows us to include multiple instances of the same object in a scene.
Transparency and Refraction

Our framework assumes surfaces reflect light.

What if they don't?
Basically, physics

- Laws of physics govern how light transmits through *dielectric* surfaces. Snell's law:

\[ n \sin \theta = n_t \sin \phi \]
Basically, physics

- Laws of physics govern how light transmits through *dielectric* surfaces. Snell's law:

\[ n \sin \theta = n_t \sin \phi. \]

**Similar to mirror reflection:** When light hits a special kind of surface, shoot a new ray in new direction.
Constructive Solid Geometry

- Compose objects from other objects using set operations:
Constructive Solid Geometry

- Intersections yield intervals of $t$
- Perform the set operations on those intervals to determine intersection point.
Distribution Ray Tracing

- Problem: jagged object and shadow edges
Distribution Ray Tracing

- Problem: jagged object and shadow edges
we have this

we want this

we have this
we have this

we want this

we have this

Idea: *supersample* rays within each pixel.
we have this

we want this

we have this

Idea: **supersample** rays within each pixel.
Distribution Ray Tracing

- Problem: area light sources

[Diagram: A red and green room with a light source above and global illumination and soft shadows indicated]
Distribution Ray Tracing

- Problem: area light sources

(global illumination) → (soft shadows)
Distribution Ray Tracing

- Problem: area light sources
Next week:

• Transformations - positioning, scaling, rotating, shearing, etc. of objects and cameras in the scene.

• Intro to object-order rendering.