Computer Graphics

Lecture 13

Acceleration Structures
Ray tracing is expensive.

for each pixel:
  for each triangle:
    compute barycentric intersection

How expensive? Let's (informally) count some FLOPs.
float-point operations
Reminder:

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
Reminder:

Barycentric ray-triangle intersection

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

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

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

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

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

\[ \beta = \frac{j(e_i - h f) + k(g f - d i) + l(d h - e g)}{M}, \]

\[ \gamma = \frac{i(a k - j b) + h(j c - a l) + g(b l - k c)}{M}, \]

\[ t = -\frac{f(a k - j b) + e(j c - a l) + d(b l - k c)}{M}, \]

where

Reusing from above:

3 mult \[ M = a(e_i - h f) + b(g f - d i) + c(d h - e g). \]
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

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

Total: 27 FLOPs

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

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

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

5 add/sub
10 mult/div

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

\[= 2,836,684,800\]
\[= 2.8\ \text{GFLOPs}\]
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/comment/2742856/#Comment_2742856
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

so what's the problem?

https://polycount.com/discussion/comment/2742856/

#Comment_2742856
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

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

\[
\begin{align*}
\text{flops} &= 995,328,000,000 \\
\text{GFLOPs} &= 995 \\
\text{TFLOPs} &\approx 1
\end{align*}
\]

Want to render this for an interactive game?
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

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

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

![Diagram of bounding volumes](Glassner 89, Fig 4.5)

- sphere
- axis-aligned box
- oriented box
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 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 (may be BoundedObjects themselves!)

• New `ray_intersect` routine for BoundedObject:
  • 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:
Problems: AABB
Construction and Intersection

How do we intersect with an axis-aligned bounding box (AABB)?

Construction:
- AABB for a sphere
- AABB for a triangle
- AABB for a collection of AABBs

Intersection:
- 1D: intersect a slab
- 3D: intersect the intersection of 3 slabs