Ray tracing vs rasterization

UCSD CSE 167
Tzu-Mao Li
Rasterization: project shapes to image and resolve visibility using Z-buffer

```
z_buffer = inf(w, h)
img = zeros(w, h)
for each clipped 3D triangle:
    get the 2D triangle by dividing -z
    compute image space bounding box
    for each pixel (x, y) in the bounding box:
        if pixel center hits triangle
            interpolate Z
        if (Z < z_buffer[x, y]):
            z_buffer[x, y] = Z
            img[x, y] = ...
```
Ray tracing:

trace rays from the camera and find the first hit
Ray tracing:
trace rays from the camera and find the first hit

```python
img = zeros(w, h)
for each pixel (x, y):
    ray = ray_from_camera(x, y)
    min_Z = inf
    for each triangle:
        if ray hits the triangle:
            compute Z
            if (Z < min_Z):
                min_Z = Z
                img[x, y] = ...
```
Ray tracing: trace rays from the camera and find the first hit

```python
img = zeros(w, h)
for each pixel (x, y):
    ray = ray_from_camera(x, y)
    min_Z = inf
    for each triangle:
        if ray hits the triangle:
            compute Z
            if (Z < min_Z):
                min_Z = Z
                img[x, y] = ...
```
Generating a ray from camera

camera position
(0, 0, 0)
camera right
(1, 0, 0)
camera up
(0, 1, 0)
camera direction
(0, 0, -1)
image plane (z = -1)
Generating a ray from camera

camera position
(0, 0, 0)
camera right
(1, 0, 0)
camera up
(0, 1, 0)
camera direction
(0, 0, -1)
pixel location
image plane (z = -1)
Ray tracing:
trace rays from the camera and find the first hit

```python
img = zeros(w, h)
for each pixel (x, y):
    ray = ray_from_camera(x, y)
    min_Z = inf
    for each triangle:
        if ray hits the triangle:
            compute Z
            if (Z < min_Z):
                min_Z = Z
                img[x, y] = ...
```
Ray-triangle intersection

points on a ray can be parameterized using origin $o$, direction $d$, and time $t$

\[ x = o + t \cdot d \]
Ray-triangle intersection

Points on a ray can be parameterized using origin \( \mathbf{o} \), direction \( \mathbf{d} \), and time \( t \):

\[
\mathbf{x} = \mathbf{o} + t \cdot \mathbf{d}
\]

Points on a triangle can be parameterized using barycentric coordinates:

\[
\mathbf{x} = b_0 \mathbf{P}_0 + b_1 \mathbf{P}_1 + b_2 \mathbf{P}_2 \quad b_0, b_1, b_2 \geq 0
\]

\[
b_0 + b_1 + b_2 = 1
\]
Ray-triangle intersection

\[
o + t \cdot d = b_0 P_0 + b_1 P_1 + b_2 P_2
\]

\[
o_x + t \cdot d_x = b_0 P_{0x} + b_1 P_{1x} + b_2 P_{2x}
\]

\[
o_y + t \cdot d_y = b_0 P_{0y} + b_1 P_{1y} + b_2 P_{2y}
\]

\[
o_z + t \cdot d_z = b_0 P_{0z} + b_1 P_{1z} + b_2 P_{2z}
\]

\[
b_0 + b_1 + b_2 = 1
\]

4 unknowns (t, b0, b1, b2), 4 linear equations

\[
b_0, b_1, b_2 \geq 0
\]
Ray-triangle intersection

\[ o_x + t \cdot d_x = (1 - b_1 - b_2)P_{0x} + b_1P_{1x} + b_2P_{2x} \]
\[ o_y + t \cdot d_y = (1 - b_1 - b_2)P_{0y} + b_1P_{1y} + b_2P_{2y} \quad b_1, b_2 \geq 0 \]
\[ o_z + t \cdot d_z = (1 - b_1 - b_2)P_{0z} + b_1P_{1z} + b_2P_{2z} \quad 1 - b_1 - b_2 \geq 0 \]

3 unknowns \((t, b_1, b_2)\), 3 linear equations
Ray-triangle intersection

check out the Moller-Trumbore intersection algorithm
https://en.wikipedia.org/wiki/M%C3%B6ller%E2%80%93Trumbore_intersection_algorithm

\[ o_x + t \cdot d_x = (1 - b_1 - b_2)P_{0x} + b_1 P_{1x} + b_2 P_{2x} \]
\[ o_y + t \cdot d_y = (1 - b_1 - b_2)P_{0y} + b_1 P_{1y} + b_2 P_{2y} \quad b_1, b_2 \geq 0 \]
\[ o_z + t \cdot d_z = (1 - b_1 - b_2)P_{0z} + b_1 P_{1z} + b_2 P_{2z} \quad 1 - b_1 - b_2 \geq 0 \]

3 unknowns (t, b1, b2), 3 linear equations
Ray tracing can easily handle more complex shapes

Q: how do we know if a ray intersects with a sphere? if it hits, how do we know where?

\[ \mathbf{x} = \mathbf{o} + t \cdot \mathbf{d} \]

\[ \| \mathbf{x} - \mathbf{x}_c \|^2 = r^2 \]
Ray tracing can handle more complex camera models

Ray tracing can handle more complex camera models

A Realistic Camera Model for Computer Graphics

Craig Kolb  
Computer Science Department  
Princeton University

Don Mitchell  
Advanced Technology Division  
Microsoft

Pat Hanrahan  
Computer Science Department  
Stanford University
Accelerating ray tracing

```python
img = zeros(w, h)
for each pixel (x, y):
    ray = ray_from_camera(x, y)
    min_z = inf
    for each triangle:
        if ray hits the triangle:
            compute Z
            if (Z < min_z):
                img[x, y] = …
```

avoid looping over all triangles!
Bounding Volumes Hierarchy (BVH)

- idea: test a group of triangles at a time by looking at their 3D bounding box
Bounding Volumes Hierarchy (BVH)

- idea: test a group of triangles at a time by looking at their 3D bounding box

skip testing the whole branch if ray miss
Bounding Volumes Hierarchy (BVH)

- idea: test a group of triangles at a time by looking at their 3D bounding box

skip testing the whole branch if ray miss
BVH construction

1. compute bounding box of all objects
BVH construction

1. compute bounding box of all objects

2. pick an axis, split the objects into half
BVH construction

1. compute bounding box of all objects
2. pick an axis, split the objects into half
3. recursively split the two children
BVH construction

1. compute bounding box of all objects

2. pick an axis, split the objects into half

3. recursively split the two children

very tricky to parallelize on GPUs!
many advanced algorithms for faster build & higher-quality build
Rasterization vs ray tracing

for each triangle:  
  project triangle to screen
for each pixel:  
  if pixel center hits triangle:  ...  

for each pixel:  
  generate camera ray
for each triangle:  
  if ray hits triangle:  ...  

rasterization  

ray tracing
If there is only one triangle, rasterization and ray tracing aren’t that different.

project triangle to screen for each pixel:
  for each pixel:
    if pixel center hits triangle:  if ray hits triangle:
      ...

rasterization ray tracing

check out this paper!

3D Rasterization – Unifying Rasterization and Ray Casting

Carsten Dachsbacher\textsuperscript{1}\textsuperscript{1} Philipp Slusallek\textsuperscript{1}\textsuperscript{2} Tomas Davidovic\textsuperscript{1} Thomas Engelhardt\textsuperscript{1} Mike Phillips\textsuperscript{1} Iliyan Georgiev\textsuperscript{1}

\textsuperscript{1}VISUS, University of Stuttgart \textsuperscript{2}DFKI, Saarland University \textsuperscript{3}Excellence Cluster M 2 CI, Saarland University
Rasterization and ray tracing differ in the acceleration structures.

**Rasterization:**
- screen-space bounding box
- occlusion culling (hierarchical Z-buffer)

**Ray Tracing:**
- bounding volume hierarchy
Rasterization is usually faster

- memory coherent access
- simpler acceleration data structures
- much faster to build hierarchical Z than the BVH

but ray tracing is also getting faster

from Gruen 2020
1080p, ~19M triangles
raster: 2.7 ms
raytrace: 8.6 ms
(2.5 ms for animation update)
Ray tracing is more versatile and can easily handle more primitives.
Ray tracing is more versatile

- can easily handle more primitives
- can easily handle more camera models
Ray tracing is more versatile

can easily handle more primitives

can easily handle more camera models

can "easily" handle more rendering effects

mirrors and glasses

soft shadow

global illumination

caustics

https://renderman.pixar.com/resources/RenderMan_20/
Ray tracing for mirror reflections

- continue tracing the ray in mirror reflection direction if we hit a mirror surface

\[ r = v - (2v \cdot n) n \]
Reflection vector

\[ r = v - (2v \cdot n)n \]
Ray tracing for shadow

color = light * reflectance if visible else 0
More about ray tracing
(or you should just take CSE 168!)

Ray Tracing in One Weekend
—
The Book Series

Ray Tracing in One Weekend
Ray Tracing The Next Week
Ray Tracing The Rest of Your Life

Peter Shirley
Trevor D Black
Steve Hollysch

free online!
https://raytracing.github.io/
It’s not impossible to do these with rasterization

but it can be slower and/or less numerically stable

“just” need to solve for the projected ellipse
can solve for screen-space projection or warp the image afterwards

can collect secondary rays and rasterize primitives to them

mirrors and glasses
soft shadow
global illumination
caustics

https://renderman.pixar.com/resources/RenderMan_20/
For example, it is possible to approximate refraction using rasterization.

Rasterization

Ray tracing
For example, it is possible to simulate soft shadow/global illumination using rasterization.

Chapter 38. High-Quality Global Illumination Rendering Using Rasterization

While the visual richness of images that current GPUs can render interactively continues to increase quickly, there are many important lighting effects that are not easily handled with current techniques. One important lighting effect that is difficult for GPUs is high-quality global illumination, where light that reflects off of objects in the scene that are not themselves light emitters is included in the final image. Incorporating this indirect illumination in rendered images greatly improves their visual quality. Figure 38-1 shows an indoor scene with indirect lighting and large area-light sources. Like all images in this chapter, it was rendered with a GPU-based global illumination algorithm that uses rasterization hardware for efficient ray casting.

rendered using rasterization!
Rasterization-based global illumination was mainstream in movie industry for a while. They all move to ray tracing now though.

@Pixar

@Disney and @Industrial Light & Magic

https://graphics.pixar.com/library/PointBasedGlobalIlluminationForMovieProduction/paper.pdf
Rasterization and ray tracing are just **visibility algorithms** with different data structures.

Rasterization:
- Screen-space bounding box
- Occlusion culling (hierarchical Z-buffer)

Ray tracing:
- Bounding volume hierarchy
In practice

• most real-time rendering: rasterization

• because not all people have a ray tracing GPU

• high-end real-time rendering: hybrid

• rasterization for projecting geometry to image
  ray tracing for shadows and reflections

• real-time rendering with massive geometry: rasterization

• see Unreal’s Nanite

• offline rendering (e.g., movies): ray tracing

• mobile rendering: rasterization (hot area to add ray tracing)

Terminology

- “ray tracing”: graphics algorithms that involve shooting rays
- “raycasting”: shooting rays from camera and find the first hit
- “path tracing”: a ray tracing algorithm for rendering global illumination

also see Peter Shirley’s thread
https://twitter.com/Peter_shirley/status/1668975090674057217
On to the next thing

May 27, 2018

I’ve been a long-time skeptic about ray tracing for interactive rendering. Whenever the topic’s come up, I’ve always been on the side of “I wish, but it just doesn’t make sense for real-time rendering”. It’s certainly not that I don’t like ray tracing, but for a long time, I felt that most of the arguments that were made in favor of it never really stood up to close scrutiny.

I don’t want to go through all of them (in part because smoke will start to come out of my ears), but just to mention a few:

- **Rasterization is a hack.** This is the worst one. Rasterization is a wonderfully efficient algorithm for computing visibility from a single viewpoint. That’s an awfully useful thing. And by the way, for the past 30 years, it’s done that robustly, without cracks along shared triangle edges—something that has just recently been solved for ray tracing.

- **Rasterization is only fast because it has dedicated hardware.** The implication being that ray tracing could be just as fast, just given a few transistors. This one misses the substantial advantages of coherent and limited memory access with rasterization (both texture and framebuffer) and ignores the substantial computational advantages of computing visibility from a single viewpoint rather than from arbitrary viewpoints.

- **Ray tracing has \(O(\log n)\) time complexity while rasterization is \(O(n)\) so it wins for sufficiently complex scenes.** Apparently, the fact that culling and LOD can be and is applied with rasterization renderers is not universally understood.
History

- Ray casting [Appel 1968]
- A Characterization of Ten Hidden-Surface Algorithms [Ivan Sutherland et al. 1974]
- Ray tracing (add reflection/refraction/antialiasing to ray casting) [Turner Whitted 1980]
- Acceleration structures for ray tracing [Steven Rubin and Turner Whitted 1980]
- Hierarchical Z [Ned Greene et al. 1993]
Next: 3D linear transformation

Diagram showing the transformation process from object space to screen space via model matrix, world space, view matrix, camera space, and projection matrix.