On a CPU it took a few minutes at 1920x1080 with 2048 paths per pixel of maximum depth six.
On a GPU with some denoising and a reduced path count it can render in a few milliseconds, for 10 fps interaction at about this quality.
If we drop to one path per pixel, then hits about 5 fps on CPU and 120 Hz on a GPU.

Before starting, or at end:
http://www.infinitelooper.com/?v=AdTxrggo8e8&p=n#/3;96
At GDC in March ‘18, we ran this Star Wars demo in real time -- on 4 water-cooled Volta GPUs. Today, we can run this demo on a single Turing, Quadro RTX 800. 
At GDC in March '18, we ran this Star Wars demo in real time -- on 4 water-cooled Volta GPUs. Today, we can run this demo on a single Turing, Quadro RTX 800.
There is an old joke that goes, “Ray tracing is the technology of the future, and it always will be!” – David Kirk

http://www.pcper.com/article.php?aid=530

Note that it’s a technology, a tool, not an algorithm.
A RAY IS A LOCATION AND A DIRECTION

Taken from Pete Shirley's intro to ray tracing notes, SIGGRAPH 2019
A RAY CAN TEST VISIBILITY BETWEEN TWO POINTS

\[ P(t) = R + t \times (Q-R) \]
A ray can test visibility between two points on surfaces.

\[ P(t) = R + t \times (Q-R) \]
He even talks about previous ray tracing algorithms, such as MAGI and Arthur Appel 1968. Douglas Kay in 1979, "TRANSPARENCY FOR COMPUTER SYNTHESIZED IMAGES", almost did it.
eye → glass → diffuse cube

shadow rays
1980: Whitted-Style Ray Tracing

For each pixel
- Send ray from eye into scene
- Send a ray from the intersection to each light: shadows
- Spawn a new color ray for each reflection & refraction

highly polished surface

74 minutes on a VAX-11/780, 640 x 480
1984: Cook Stochastic ("Distribution") Ray Tracing

Allow shadow rays to go to a random point on area light.
Allow specular rays to be perturbed specularly around the ideal reflection.
Shoot sometime during the frame for motion blur.

https://graphics.pixar.com/library/indexAuthorRobert_L_Cook.html
Note recursion: ray continues along a path until a light is hit (or something entirely black or considered “unchangeable,” such as an environment map).
“... the brute-force approach ... is ridiculously expensive.” - Sutherland, Sproull, and Schumacker, *A Characterization of Ten Hidden-Surface Algorithms*, 1974

Which algorithm is this? About the z-buffer algorithm. Brute force beats elegance on transistor count. Uniform data structures much cheaper to put into silicon. 128 MB in 1971 would have cost $59,360 in 2018 dollars.
Which is ray traced? The image on the left is actually ray traced (note the mirrors on the right going to infinity), the one on the right is actually rasterized.
Rasterization’s Advantages

• Send the triangle or mesh and then forget it.

Triangle doesn’t need to be resident on PC GPU.

• Memory coherence for single viewpoint for triangle and its textures.

Assumes triangle covers a fair number of pixels.
Ray Tracing’s Advantages

• Rays are independent of each other, so can be traced as needed vs. lock-step fashion.

But incoherent rays can cause cache misses – slow.

• No limitations such as one Z-buffer value stored.

But, each transparent object may spawn a new ray.
Nested grids do see use for voxel/volume rendering, and k-d trees for point clouds
Fake News

Rasterization can benefit from:

- Hierarchical frustum culling (also using a BVH)
- Level of detail culling and simplification
- Occlusion culling in various forms
- Hierarchical Z buffer

\[ \text{Ray Tracing’s } O(\log N) \text{ beats Rasterization’s } O(N) \]
## Rasterization and Ray Tracing

<table>
<thead>
<tr>
<th>Key Concept</th>
<th>Rasterization</th>
<th>Ray Tracing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fundamental question</td>
<td>What pixels does geometry cover?</td>
<td>What is visible along this ray?</td>
</tr>
<tr>
<td>Key operation</td>
<td>Test if pixel is inside triangle</td>
<td>Ray-triangle intersection</td>
</tr>
<tr>
<td>How streaming works</td>
<td>Stream triangles (each tests pixels)</td>
<td>Stream rays (each tests intersections)</td>
</tr>
<tr>
<td>Inefficiencies</td>
<td>Shade many tris per pixel (overdraw)</td>
<td>Test many intersections per ray</td>
</tr>
<tr>
<td>Acceleration structure</td>
<td>(Hierarchical) Z-buffering</td>
<td>Bounding volume hierarchies</td>
</tr>
<tr>
<td>Drawbacks</td>
<td>Incoherent queries difficult to make</td>
<td>Traverses memory incoherently</td>
</tr>
</tbody>
</table>

Pete Shirley’s slide
Pretty soon, computers will be fast.
– Billy Zelsnack

However, it still takes up to twenty seconds for me to find what’s in a directory when I double-click it.
Turner Whitted: football field of Cray computers, each with an R G and B light on top.

Sphereflake on pixel machine ran in 30 seconds, 16 seconds a year later due to software tuning.
http://www.realtimerendering.com/resources/RTNews/html/rtnews4a.html#art4
NVIDIA is based on this idea.
More Special-Purpose Hardware

PASCAL
11.8 Billion xtons | 471 mm² | 24 GB 10GHz

TURING
18.6 Billion xtons | 754 mm² | 48-48 GB 14GHz

SHADER | COMPUTE
13 TFLOPS FP32
50 TOPS INT8

TENSOR CORE
125 TFLOPS FP16
250 TOPS INT8
500 TOPS INT4

RT CORE
10 Giga Rays/Sec

SHADER | COMPUTE
16 TFLOPS ± 16 TIPS
RT Cores

RT Cores perform
- Ray-BVH Traversal
- Instancing: 1 Level
- Ray-Triangle Intersection

Return to Streaming Multiprocessors for
- Multi-level Instancing
- Custom Intersection
- Shading

SM – streaming multiprocessor
4K: 3840 x 2160 pixels takes 33 MB (including alpha) – means 30 images is 1 GB
GPUs are the only type of parallel processor that has ever seen widespread success... because developers generally don’t know they are parallel! – Matt Pharr, circa 2008
DirectX Rasterization Pipeline

• What do shaders do in today’s widely-used rasterization pipeline?

- Merge each pixel into the final output image (e.g., doing blending)
  - Usually done with special-purpose hardware
  - Hides optimizations like memory compression and converting image formats

Actually different for most mobile, where the triangles are retained and thrown against multiple tiles.
DirectX Ray Tracing Pipeline

• So what might a simplified ray tracing pipeline look like?

• One advantage of ray tracing:
  • Algorithmically, much easier to add recursion

Please note:
A simplified representation

Actually different for most mobile, where the triangles are retained and thrown against multiple tiles.
Five Types of Ray Tracing Shaders

- Ray tracing pipeline split into **five** shaders:
  - A **ray generation shader** define how to start tracing rays
  - **Intersection shader(s)** define how rays intersect geometry
  - **Miss shader(s)** shading for when rays miss geometry
  - **Closest-hit shader(s)** shading at the intersection point
  - **Any-hit shader(s)** run once per hit (e.g., for transparency)

From Chris Wyman’s introduction to ray tracing SIGGRAPH 2019 notes
Five Types of Ray Tracing Shaders

- Ray tracing pipeline split into **five** shaders:
  - A *ray generation shader* ← Controls other shaders
  - *Intersection shader(s)* ← Defines object shapes (one shader per type)
  - *Miss shader(s)* ← Controls per-ray behavior (often many types)
  - *Closest-hit shader(s)*
  - *Any-hit shader(s)*

From Chris Wyman’s introduction to ray tracing SIGGRAPH 2019 notes
How Do these Fit Together?  [Eye Chart Version]

TraceRay() Called

Acceleration Traversal

Intersection Shader

Is Dope?

Yes

Any-Hit Shader

Is Opaque?

Yes

Accept hit

Closest Hit?

Yes

This is closest hit

Update closest hit data

Not closest

Ignore hit (transparent)

Ignore hit (transparent)

No

Closest Hit Shader

Miss Shader

Ray Shading

Return From TraceRay()
How Do these Fit Together?  [LOGICAL Version]
• Loop during ray tracing, testing hits until there’s no more; then shade

• Closest-Hit Shader can generate new rays: reflection, shadow, etc.
Uses of Fast Ray Tracing

- In video games, of course. But also for development:
  - Fast baking for global illumination
  - Generate ground truth image for comparison
  - And possibly other interesting (ab)uses of the GPU

- In film production and computer-aided design:
  - Save artists and engineers time waiting
  - Possibly even final frames

http://erich.realtimerendering.com/rtrt/index.html
Is this the real life? Is this just fantasy?
– Freddie Mercury

In this scene from Deadpool, you probably expect there to be some computer graphics...

(This and the next N slides are from Morgan McGuire’s path tracing review presentation)
But did you expect *the entire scene* to be CGI? Action movies are increasingly like this...
Which is Real?

Left image is a photograph, right image is rendered by path tracing. This famous ground-truth test of a renderer is the origin of the “cornell box” 3D models—there’s a real box at Cornell.

Of course, modern renderers with good input can simulate non-realistic scenes:
"Color Bleeding"
"Diffuse Interreflection"

"Soft Shadows"
"Indirect Lighting"
There are good algorithms, including Cook et al. 84 for generating motion blur and depth of field phenomena via ray tracing. However, making the ray-triangle intersection efficient for motion blur in particular is tricky—time is the one parameter that affects the *intersection* instead of the ray generation or shading, and it breaks Kajiya’s original steady-state assumption in the rendering equation (notice that he had no “time” parameter). Note that even APIs offer limited support—there’s no “time” parameter for the DirectX or Vulkan GPU ray tracing APIs, and the assumption is that you’ll approximate these effects by post-processing the frame with a blur filter.
Not a physically-based effect, let’s call it physically-adjacent. It’s an (often good) guess as to how light percolates into enclosures and crevices by (usually only) looking at the local geometry.
Here’s the caustics that were one of Kajiya’s motivations for creating path tracing. They really help with the appearance of translucent materials and some curvy reflective ones.

Ironically, real caustics are often suppressed in film production because it is confusing when there’s a bright spot somewhere in a scene, like the reflection on the ceiling off someone’s watch face. They get painted out of the final frame or the director moves lights and roughens surfaces to hide them. So maybe we shouldn’t spend so much time in research trying to make algorithms to generate them. The most visually important case of caustics is probably in the ocean, and that is often faked in games and film using projective textures, which are easier to art direct than real emergent caustics.
The Dangers of Ray Tracing
The Dangers of Ray Tracing
Minecraft RTX - RTX On/Off Gameplay

http://www.infinitelooper.com/?v=AdTxrggo8e8&p=n#/3;96
The Rendering Equation

\[ L_o(X, \hat{\omega}_o) = L_e(X, \hat{\omega}_o) + \int_{S^2} L_i(X, \hat{\omega}_i) \ f_X(\hat{\omega}_i, \hat{\omega}_o) \ |\hat{\omega}_i \cdot \hat{n}| \ d\hat{\omega}_i \]

From Morgan McGuire’s “Path Tracing Review”
The Rendering Equation

\[ L_o(X, \hat{o}) = L_e(X, \hat{o}) + \int_{S^2} L_i(X, \hat{i}) f_X(\hat{i}, \hat{o}) |\hat{i} \cdot \hat{n}| d\hat{i} \]

Outgoing light  Emitted light  Incoming light  Material

From Morgan McGuire’s “Path Tracing Review” – a pure path trace picks \( \omega_i \) randomly in a uniform way.
Pure Path Tracing

\[ L_o(X, \hat{\omega}_o) = L_e(X, \hat{\omega}_o) + \int_{S^2} L_i(X, \hat{\omega}_i) f_X(\hat{\omega}_i, \hat{\omega}_o) \left| \hat{\omega}_i \cdot \hat{n} \right| d\hat{\omega}_i \]

vec3 Trace( vec3 O, vec3 D )
IntersectionData i = Scene::Intersect( O, D )
if (i == NoHit) return vec3( 0 ) // ray left the scene
if (i == Light) return i.material.color // lights do not reflect
vec3 W = RandomDirectionOnHemisphere( i.normal ), pdf = 1 / 2PI
return Trace( i.position, W ) * i.BRDF * dot( W, i.normal ) / pdf

From Morgan McGuire’s “Path Tracing Review”. Note recursion to compute \( L_i \)!

Importance Sampling

\[ L_o(X, \hat{\omega}_o) = L_e(X, \hat{\omega}_o) + \int_{S^2} L_i(X, \hat{\omega}_i) f_X(\hat{\omega}_i, \hat{\omega}_o) |\hat{\omega}_i \cdot \hat{n}| \, d\hat{\omega}_i \]

Vary sampling direction dependent on BSDF and angle
Multiple Importance Sampling

\[ L_o(X, \hat{\omega}_o) = L_e(X, \hat{\omega}_o) + \int_{S_2} L_i(X, \hat{\omega}_i) f_X(\hat{\omega}_i, \hat{\omega}_o) |\hat{\omega}_i \cdot \hat{n}| d\hat{\omega}_i \]

Vary sampling direction dependent on lighting, BSDF, and angle

A more elaborate guess at the PDF, probability density function. May (or may not) ignore shadows.
MIS Example
Multiple Importance Sampling

Simple assets and limit path types
Note: initial implementation is open source, http://brechpunkt.de/q2vkpt/


Original: http://brechpunkt.de/q2vkpt/
Denoising

Tensor cores: evidence that fast denoising (enabled by tensor cores) helps a lot for ray tracing

From the Cinematic article in Ray Tracing Gems, http://raytracinggems.com
Denoising

Tensor cores: evidence that fast denoising (enabled by tensor cores) helps a lot for ray tracing.

Denoisers best when samples uncorrelated or negatively correlated. Rays in adjacent pixels should provide maximal new information.
Different sample counts used per pixel and the perceived noise level
Denoising by Effect

Ray Traced Shadows

Specialized non-graphical data for denoising, like tangents for hairs.

Even films use denoising

From NVIDIA’s “Deep Learning for Rendering” 2018
Developing an application that benefits from DL is different from traditional software development, where software engineers must carefully craft lots of source code to cover every possible input the application may receive.

From NVIDIA’s “Deep Learning for Rendering” 2018

At the core of a DL application, much of that source code is replaced by a neural network.

To build a DL application, first a data scientist designs, trains and validates a neural network to perform a specific task.

The task could be anything, like identifying types of vehicles in an image, reading the speed limit sign as it goes whizzing past, translating English to Chinese, etc.

The trained neural network can then be integrated into a software application that feeds it new inputs to analyze, or “infer” based on its training.

The application may be deployed as a cloud service, on an embedded platforms, in an automobiles, or other platforms.

As you would expect, the amount of time and power it takes to complete inference...
tasks is one of the most important considerations for DL applications, since this determines both the quality/value of the user experience and the cost of deploying the application.
Training set

From NVIDIA’s “Deep Learning for Rendering” 2018
Tensor cores: evidence that fast denoising (enabled by tensor cores) helps a lot for ray tracing

From NVIDIA’s “Deep Learning for Rendering” 2018
Movie Time!

Zero-Day, by Beeple


10,500 emissive triangles
~350 emissive meshes

- 10,500 emissive triangles
- \(\sim350\) emissive meshes
What’s Cooking?

Some hot topics:
• Building or modifying an efficient BVH in parallel
  • The waving tree problem
• Generating ray directions for samples is complex!
  • Some rays are faster than others – can we use this fact?
• How do you deal with a large number of (moving?) lights?
• Adaptive sampling: where to generate more samples
• Denoising, spatial and or temporal
  • especially in a single pass (deep learning?)
• Ray tracing for VR (XR) – somewhat different goals

Resources

The major ray-tracing related APIs:
• Microsoft DirectX 12 DXR
• Vulkan
• Apple’s Metal

Also usable: OptiX 7, Unreal Engine, pbrt, Lighthouse 2, Blender 2.81, Embree, Radeon-Rays, on and on …

Pointers to books and resources: http://bit.ly/rtrtinfo
Pro-ish Tips on Career

Do that extra thing, something you enjoy:
• Make a website for yourself; sites.google.com if nothing else.
• Blog or write articles on things you know or things you’ve tried. (And consider jcg.org.)
• Work on some (usually public, open source) project you like, in a team or on your own. Get a different perspective.
• Volunteer at any conference, for any position – help is always needed, and you meet people.
• Help review papers in an area you know well. Say “yes.”
• Write a book. Make a movie. Create a game. All quite doable!

See my site about why you want a URL:

People think you know something if you write a book. And, dozens of dollars to be made!
At Work

• Ask questions when you don’t know. Get over looking ignorant – everyone is ignorant about 99.99% of everything.
• Solo is fine, failing solo is fine, but failing with others is less likely – get help.
• Don’t toot your own horn – let others sing your praises.
• Don’t “network on purpose.” But, when you’re in a line at a conference, start up a conversation with your neighbors.

“We are all experts in our own little niches.” – Alex Trebek

My gosh, never ever randomly ask for a connection to someone on LinkedIn without an introductory note.
Seen at Pax East this year...

Rand Miller

“Success has a lot to do with luck, but if you work hard you get more dice rolls.”

Rand Miller is the co-creator of the classic videogame “Myst”
Questions?

realtimerendering.com
raytracinggems.com
erichaines.com
How I Got Here, I Think

Over the past 36 years, I’ve:
- Helped teach *An Introduction to Ray Tracing* course at SIGGRAPH (1987)
- Coauthored the book *An Introduction to Ray Tracing* (1989) — now free!
- Co-chaired I3D Symposium (general 2006, papers 2007, and again for 2020)
- Created *Interactive 3D Graphics MOOC* (2013-present)
- All this time, contributed to SIGGRAPH and ACM TOG in various ways: courses, panels, art show, studio session, webmaster, etc.

16/17 things over 35 years = 1 thing every two years