Ray and Path Tracing Today

Eric Haines
NVIDIA
erich@acm.org
erichaines.com

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.

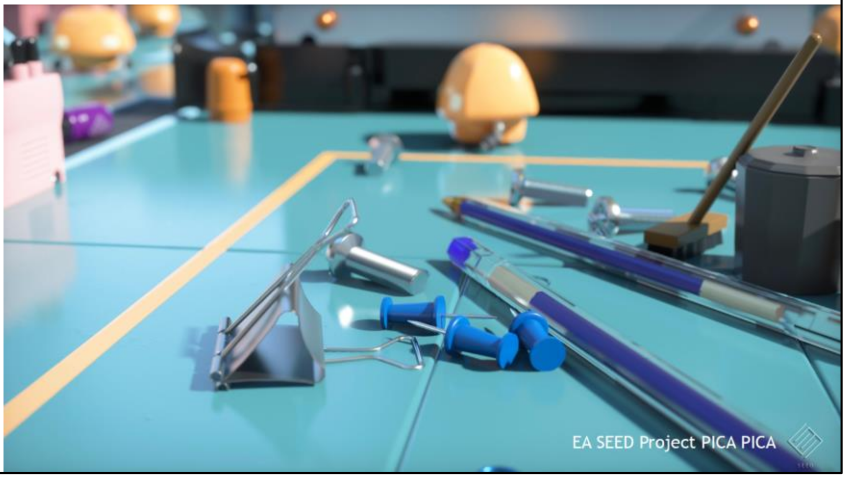
Before starting, or at end:
http://www.infinitelooper.com/?v=AdTxrggo8e8&p=n#/3;96

A Single RTX (a.k.a. Turing) Card

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.
https://vimeo.com/291876490https://vimeo.com/291876490

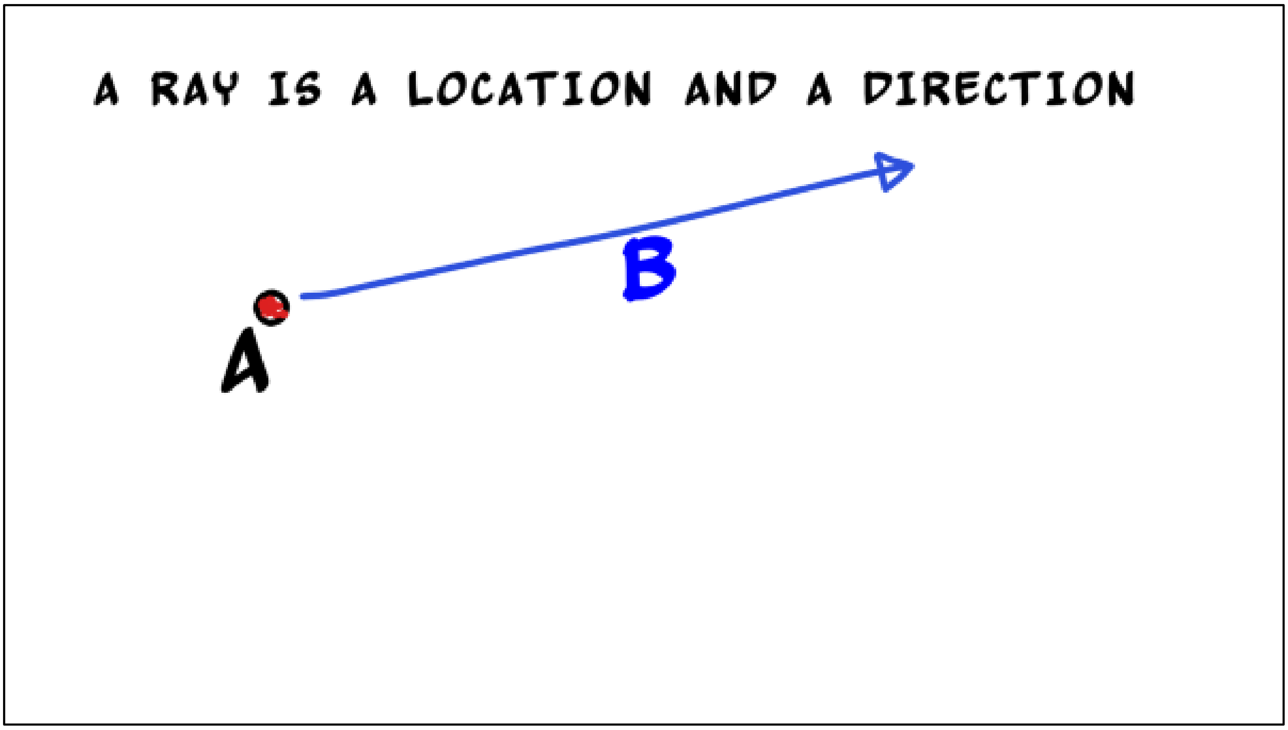
At GDC in March ‘18, we ran this Star Wars demo in real time -- on 4 water-cooled Volta GPUs. Today, we can run this demo on a single Turing, Quadro RTX 800.
https://vimeo.com/291876490https://vimeo.com/291876490

There is an old joke that goes, "Ray tracing is the technology of the future, and it always will be!"
– David Kirk



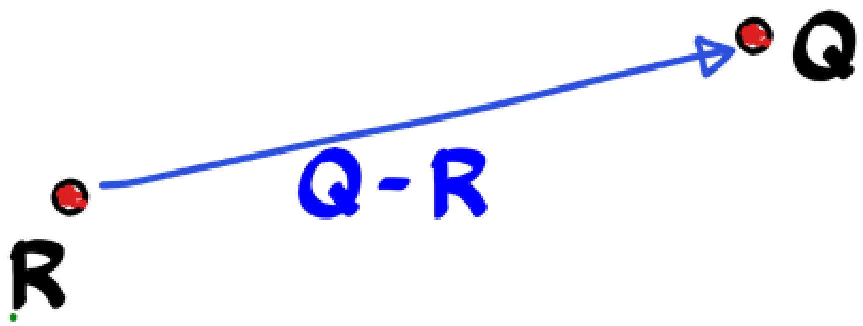
http://www.pcper.com/article.php?aid=530

Note that it's a technology, a tool, not an algorithm.



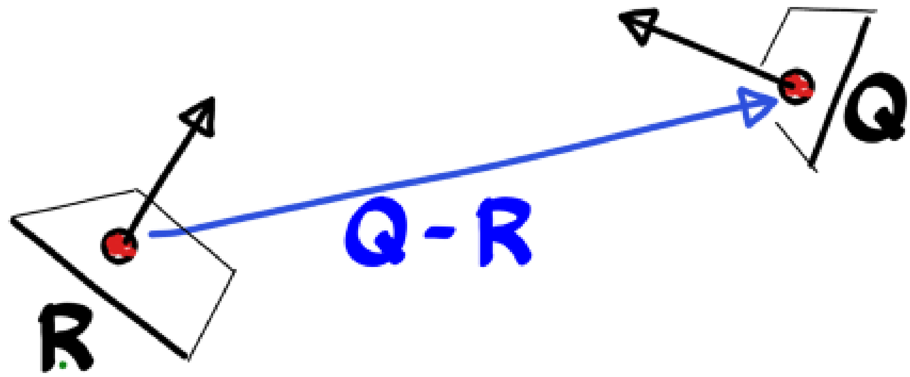
Taken from Pete Shirley's intro to ray tracing notes, SIGGRAPH 2019

A RAY CAN TEST VISIBILITY BETWEEN TWO POINTS



$$P(t) = R + t * (Q-R)$$

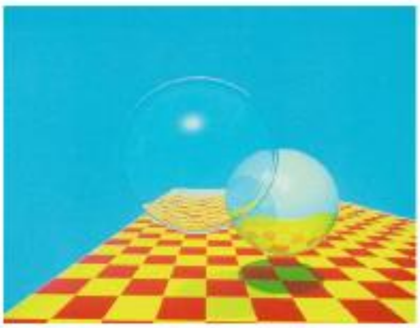
A RAY CAN TEST VISIBILITY
BETWEEN TWO POINTS ON SURFACES



$$P(t) = R + t * (Q-R)$$

Back in 1980…

- Included a bounding volume hierarchy
- Reflection and refraction
- Adaptive antialiasing



Graphics and Image Processing — J.D. Foley, Editor

An Improved Illumination Model for Shaded Display

Turner Whitted
Bell Laboratories
Holmdel, New Jersey

To accurately render a two-dimensional image of a three-dimensional scene, global illumination information that affects the intensity of each pixel of the image must be known at the time the intensity is calculated. In a simplified form, this information is stored in a tree of "rays" extending from the viewer to the first surface encountered and from there to other surfaces and to the light sources. A visible surface algorithm creates this tree for each pixel of the display and passes it to the shader. The shader then traverses the tree to determine the intensity of the light received by the viewer. Consideration of all of these factors allows the shader to accurately simulate true reflection, shadows, and refraction, as well as the effects simulated by conventional shaders. Anti-aliasing is included as an integral part of the visibility calculations. Surfaces displayed include curved as well as polygonal surfaces.

The role of the illumination model is to determine how much light is reflected to the viewer from a visible point on a surface as a function of light source direction and strength, viewer position, surface orientation, and surface properties. The shading calculations can be performed on three scales: microscopic, local, and global. Although the exact nature of reflection from surfaces is best explained in terms of microscopic interactions between light rays and the surface [3], most shaders produce excellent results using aggregate local surface data. Unfortunately, these models are usually limited in scope, i.e., they look only at light source and surface orientations, while ignoring the overall setting in which the surface is placed. The reason that shaders tend to operate on local data is that traditional visible surface algorithms cannot provide the necessary global data.

A shading model is presented here that uses global information to calculate intensities. Then, to support this shader, extensions to a ray tracing visible surface algorithm are presented.

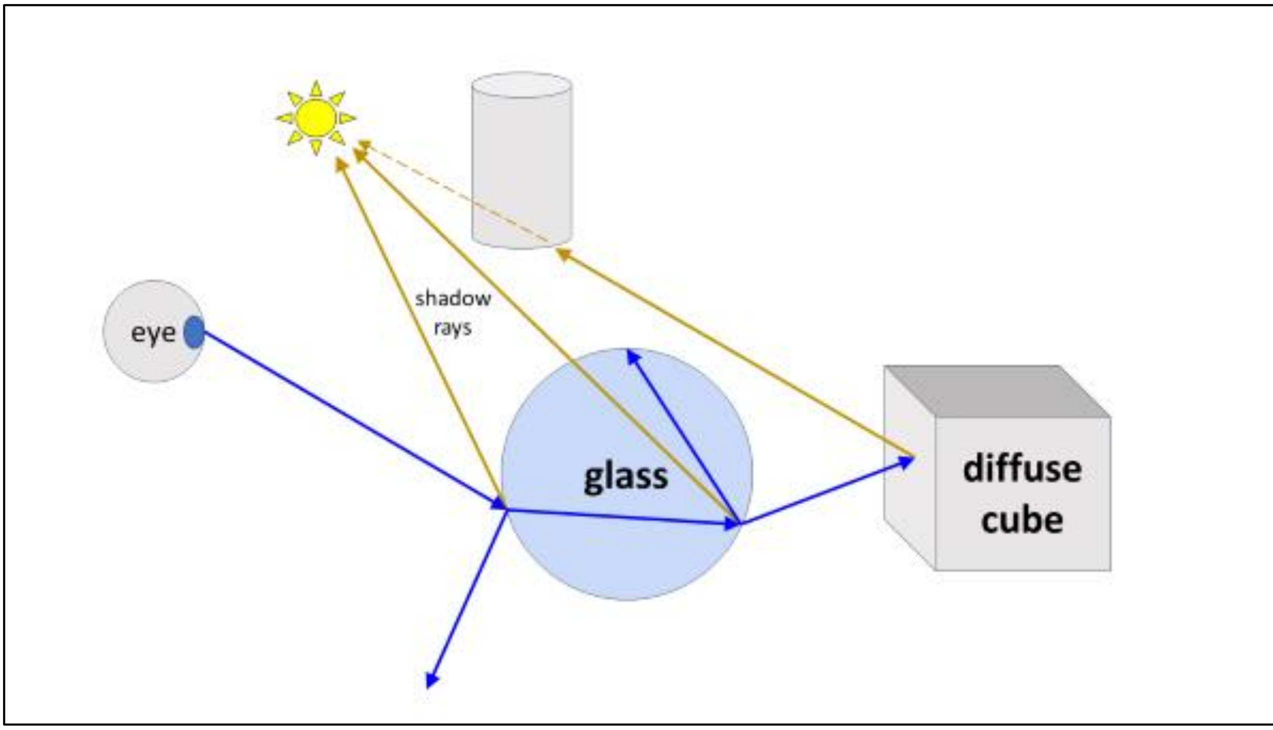
1. Conventional Models

The simplest visible surface algorithms use shaders based on Lambert's cosine law. The intensity of the reflected light is proportional to the dot product of the surface normal and the light source direction, simulating a perfect diffuser and yielding a reasonable looking approximation to a dull, matte surface. A more sophisticated model is the one devised by Bui-Tuong Phong [8]. Intensity from Phong's model is given by

$$I = I_a + k_d \sum_{j=1}^{j=ls} (\bar{N} \cdot \bar{L}_j) + k_s \sum_{j=1}^{j=ls} (\bar{N} \cdot \bar{L}_j')^n, \qquad (1)$$

where

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.


shadow
rays
eye
glass
diffuse
cube

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

1986: Kajiya-Style Diffuse Interreflection

Path tracing: shoot each ray and follow it along a series of interreflections.

"The Rendering Equation"

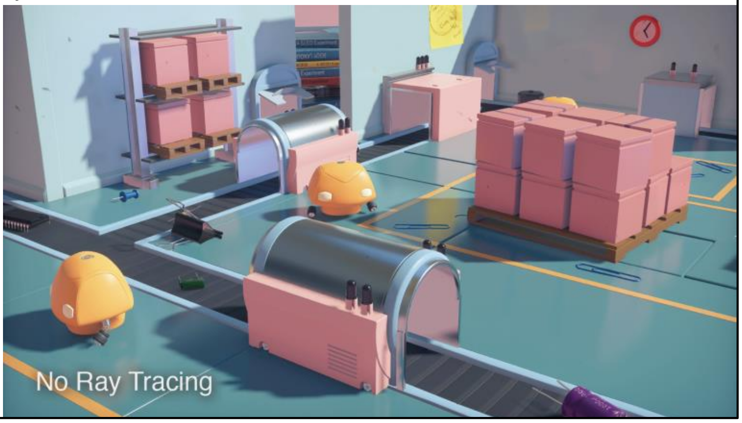
Guaranteed to give the right answer *at the limit*.

diffuse surface reflection

Figure 6. A sample image. All objects are neutral grey. Color on the objects is due to caustics from the green glass balls and color bleeding from the base polygon.

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



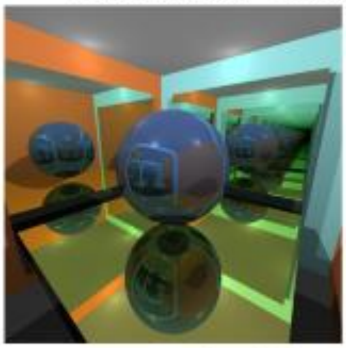
Which algorithm is this? About the z-buffer algorithm. Brute force beats elegance on transistor count. Uniform data structures much cheaper to put into silicon.
128 MB in 1971 would have cost $59,360 in 2018 dollars.

Rasterization vs. Ray Tracing

Rasterization loop:

For each object

For each pixel – closer?



Timothy Cooper

Ray Tracing loop:

For each pixel

For each object – closest?

Kasper Høy Nielsen

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.

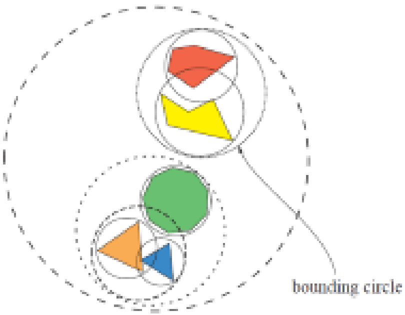
- No limitations such as one Z-buffer value stored.

But, each transparent object may spawn a new ray.

The Bounding Volume Hierarchy (BVH)

This data structure mostly won the efficiency data structure wars

Usual program: build BVH as preprocess
per frame: if needed, update BVH
trace rays



On a single (*non-parallel*) processor:

- Building a BVH is typically **O(N log N)** for **N** objects
- Updating a BVH is typically **O(N)** for **N** objects
- Traversing a BVH (i.e., tracing a ray) is typically **O(log N)**

Nested grids do see use for voxel/volume rendering, and k-d trees for point clouds

Fake News

~~Ray Tracing's O(log N) beats Rasterization's O(N)~~

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

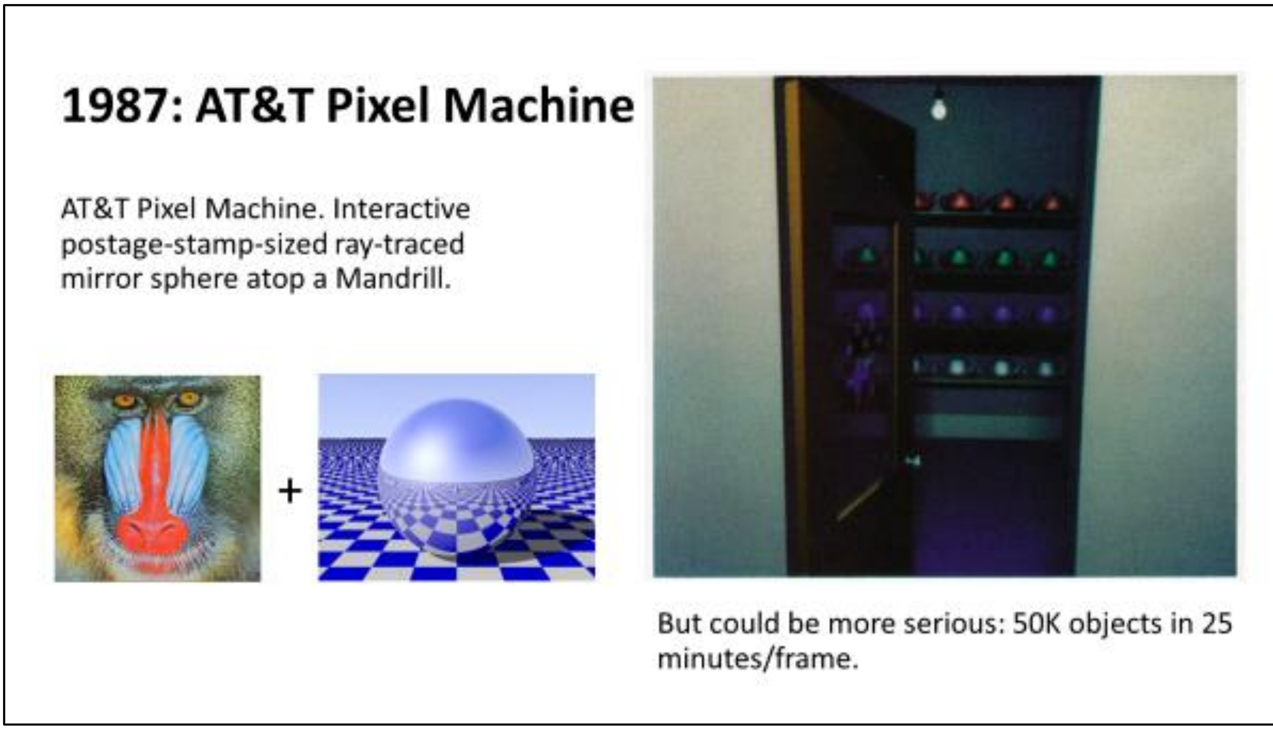
Rasterization and Ray Tracing

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

Pete Shirley's slide

Pretty soon, computers will be fast.
– Billy Zelsnack

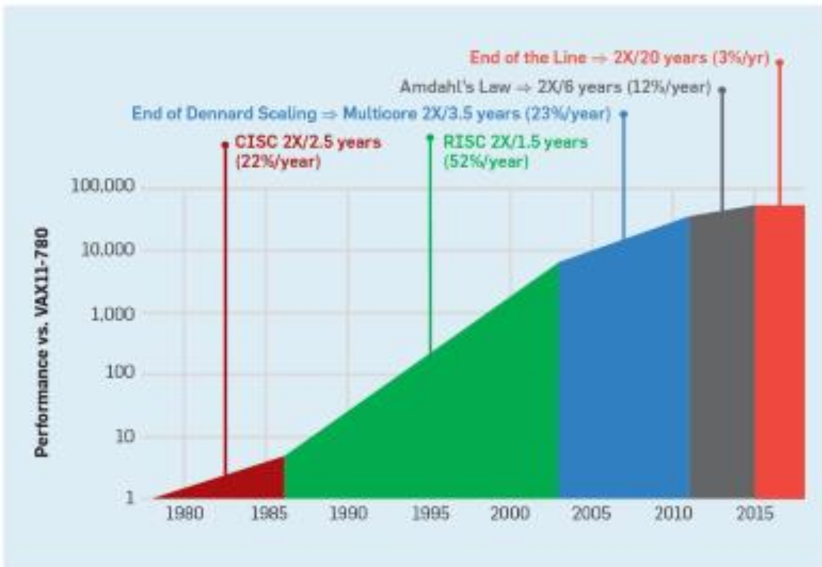
However, it still takes up to twenty seconds for me to find what's in a directory when I double-click it.



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

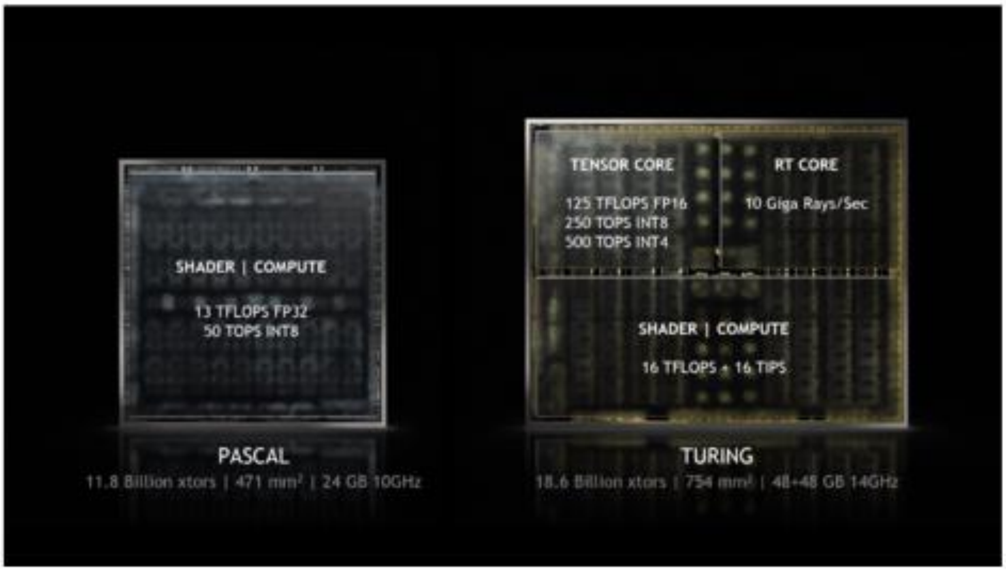
Moore's Law is Ending (Really!)



From Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, "A Domain-Specific Architecture for Deep Neural Networks," Communications of the ACM, September 2018, Vol. 61 No. 9, Pages 50-59.

NVIDIA is based on this idea.

More Special-Purpose Hardware



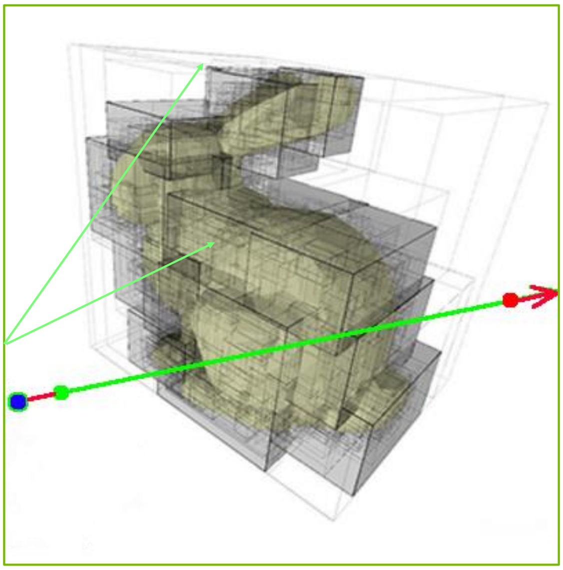
RT Cores

RT Cores perform

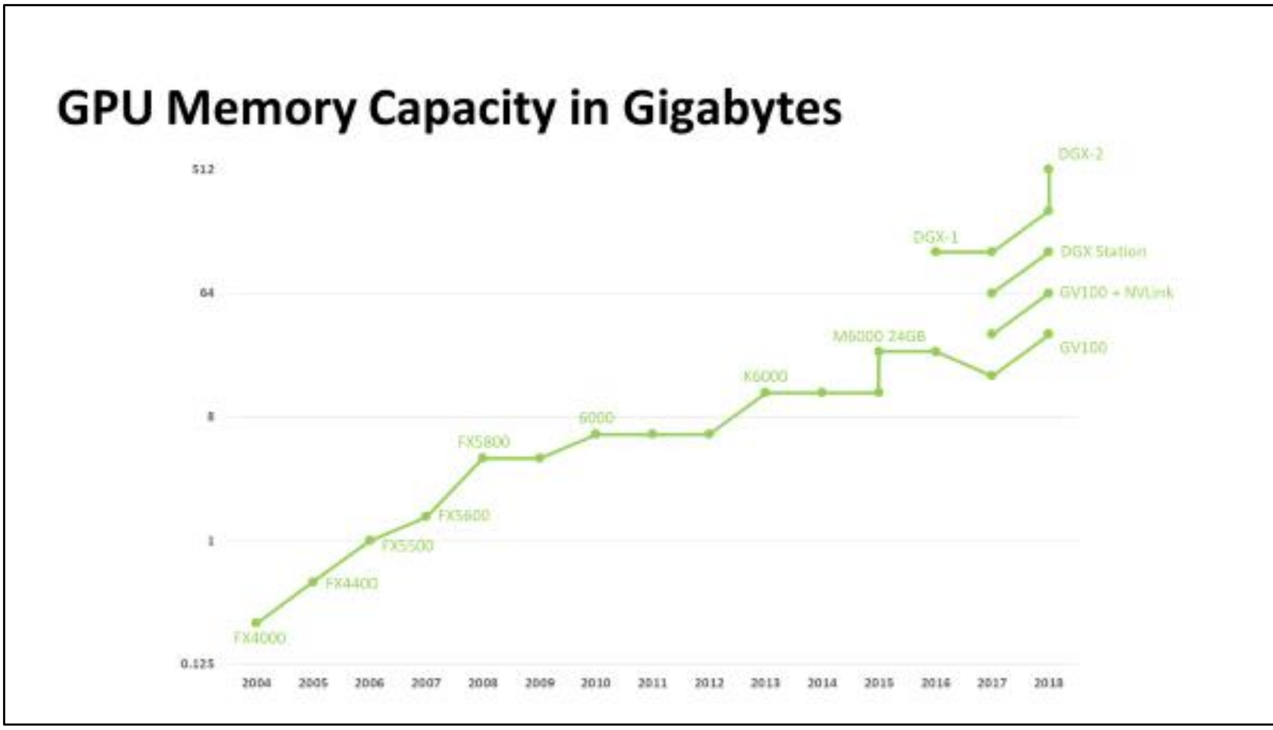
- Ray-BVH Traversal
- Instancing: 1 Level
- Ray-Triangle Intersection

Return to Streaming Multiprocessors for

- Multi-level Instancing
- Custom Intersection
- Shading



SM – streaming multiprocessor

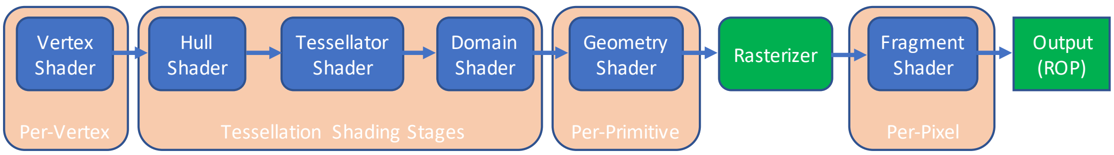


4K: 3840 x 2160 pixels takes 33 MB (including alpha) – means 30 images is 1 GB

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?



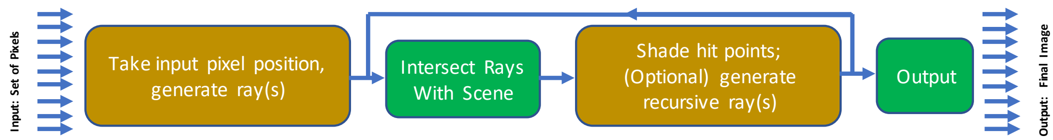
- Merge each pixel into the final output image (e.g., doing blending)
  - *Usually done with special-purpose hardware*
  - *Hides optimizations like memory compression and converting image formats*

27

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

28

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

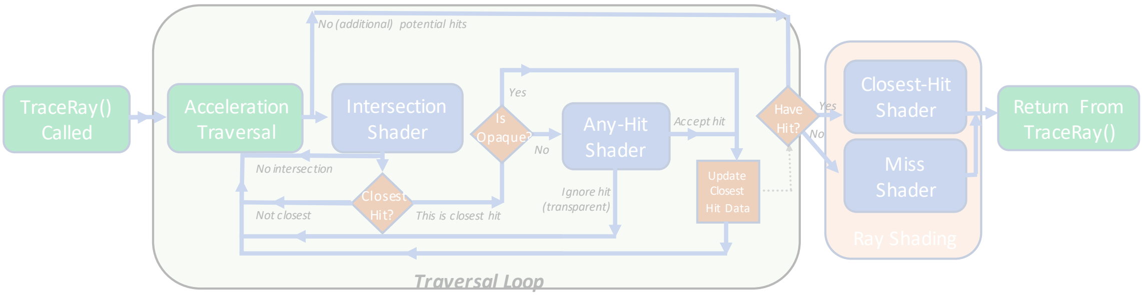
Five Types of Ray Tracing Shaders

- 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)***
  - ***Closest-hit shader(s)*** ← Controls per-ray behavior (often many types)
  - ***Any-hit shader(s)***

30

From Chris Wyman's introduction to ray tracing SIGGRAPH 2019 notes

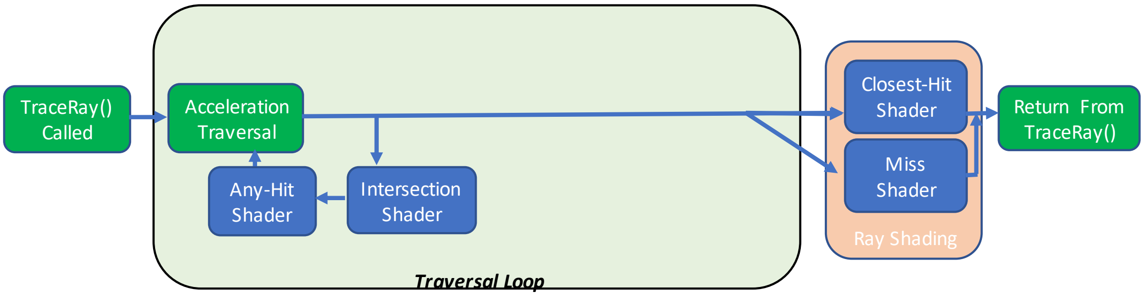
How Do these Fit Together? [Eye Chart Version]



31

How Do these Fit Together? [LOGICAL Version]

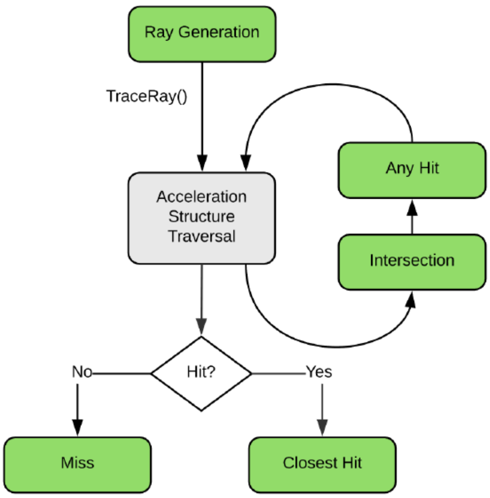
- Loop during ray tracing, testing hits until there's no more; then shade

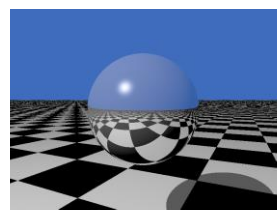


- Closest-Hit Shader can generate new rays: reflection, shadow, etc.

32

Any Hit





33

http://www.realtimerendering.com/Real-Time_Rendering_4th-Real-Time_Ray_Tracing.pdf

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

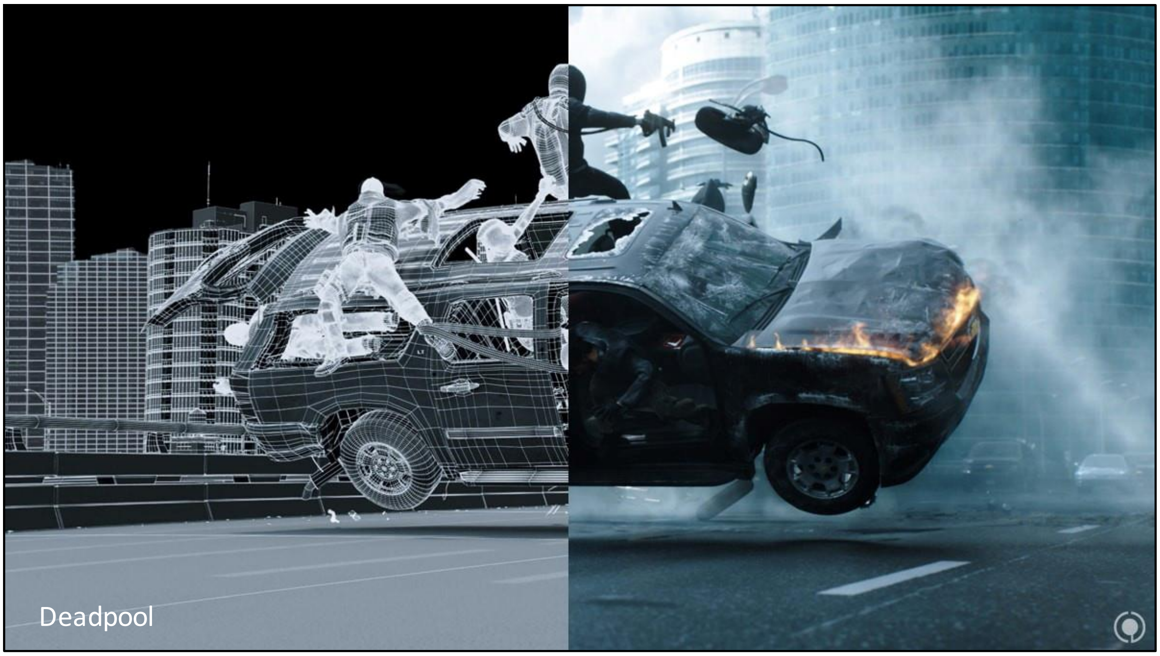
- In film production and computer-aided design:
  - Save artists and engineers time waiting
  - Possibly even final frames

http://erich.realtimerendering.com/rtrt/index.html

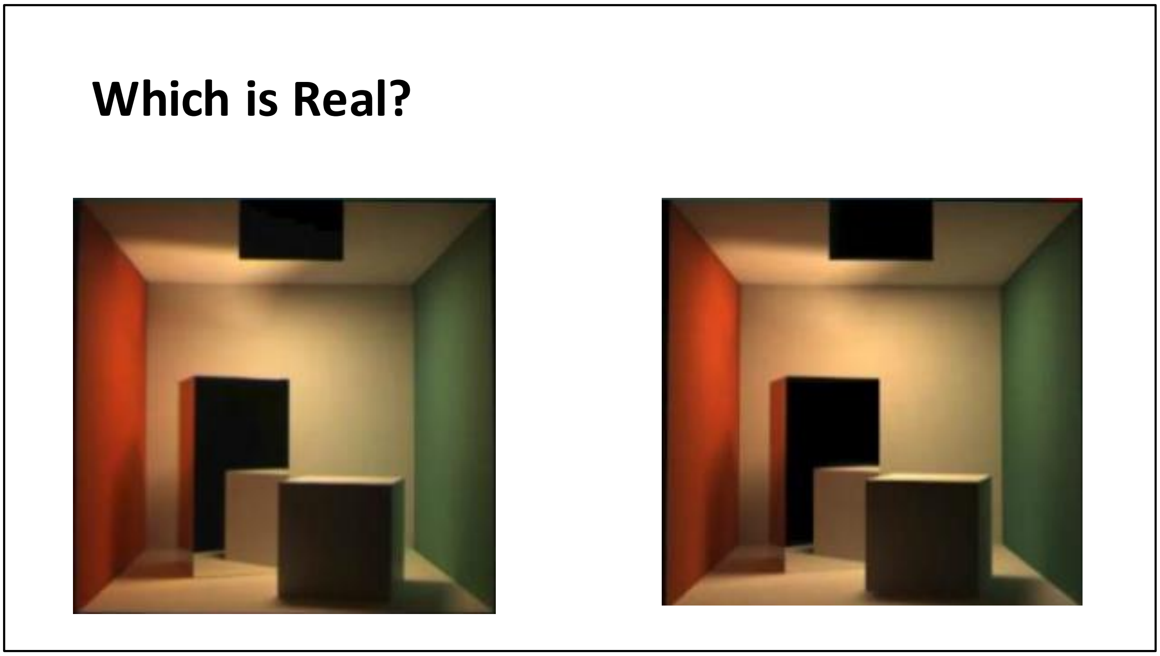
Is this the real life? Is this just fantasy?
– Freddie Mercury

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)

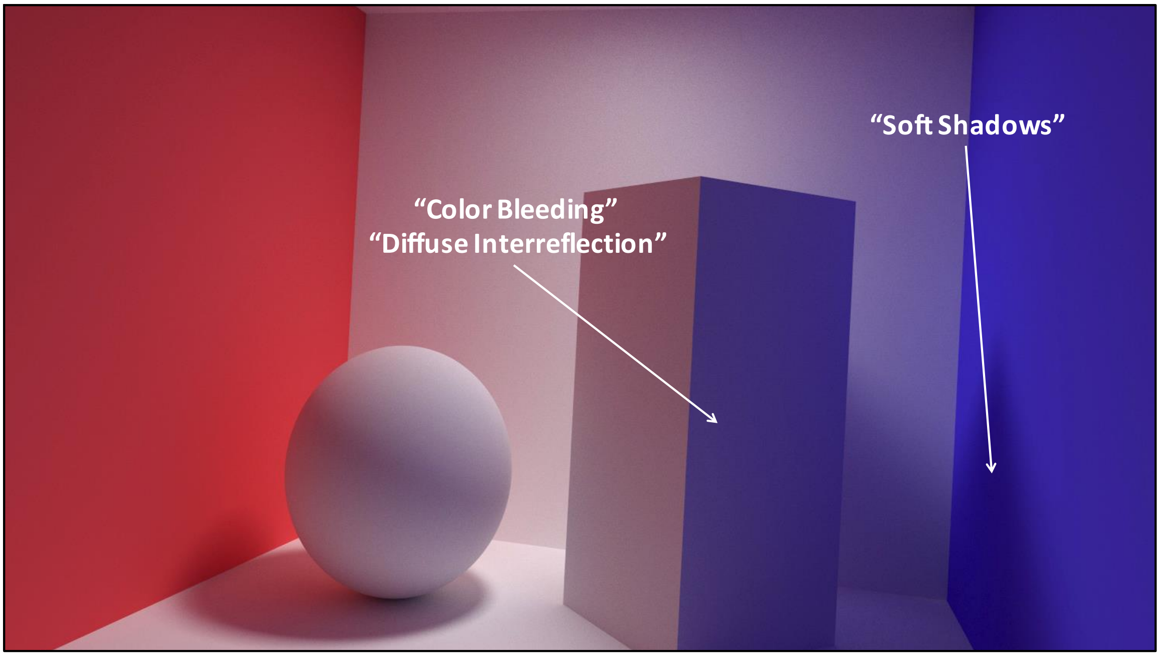


But did you expect *the entire scene* to be CGI?
Action movies are increasingly like this…



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:*

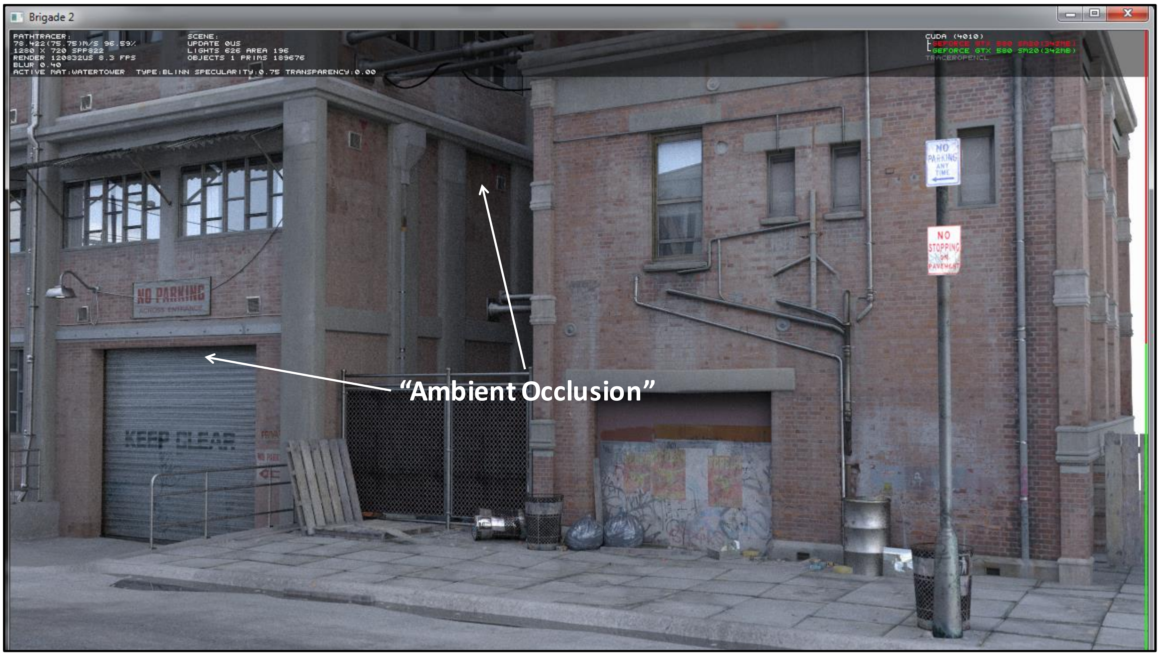

"Soft Shadows"
"Color Bleeding"
"Diffuse Interreflection"


"Indirect Lighting"

Motion Blur

Defocus Blur ("Depth of Field")

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.


"Transmission"
"Glossy Reflection"

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.

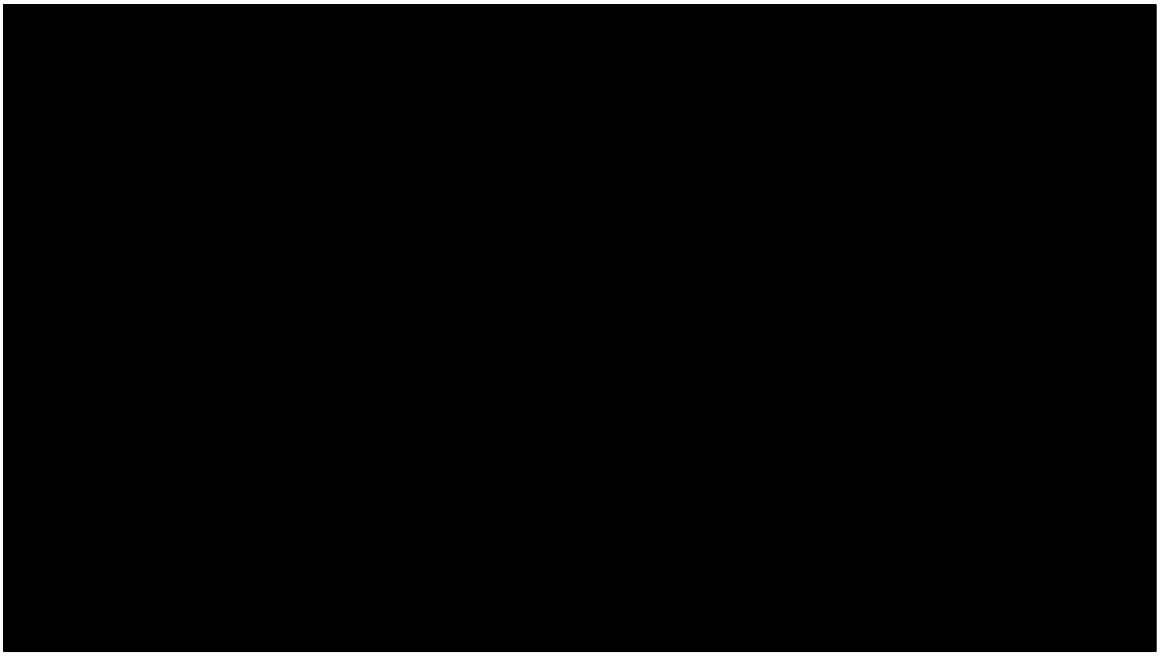
Ironically, real caustics are often suppressed in film production because it is confusing when there's a bright spot somewhere in a scene, like the reflection on the ceiling off someone's watch face. They get painted out of the final frame or the director moves lights and roughens surfaces to hide them. 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

Outgoing direction

Incoming direction

$$L_{\mathrm{o}}(X, \hat{\omega}_{\mathrm{o}}) = L_{\mathrm{e}}(X, \hat{\omega}_{\mathrm{o}}) + \int_{\mathbf{S}^2} L_{\mathrm{i}}(X, \hat{\omega}_{\mathrm{i}})\ f_X(\hat{\omega}_{\mathrm{i}}, \hat{\omega}_{\mathrm{o}})\ |\hat{\omega}_{\mathrm{i}} \cdot \hat{n}|\ d\hat{\omega}_{\mathrm{i}}$$

A point in the scene

<u>All</u> incoming directions
(a sphere)

From Morgan McGuire's "Path Tracing Review"

The Rendering Equation

$$\boxed{L_{\mathrm{o}}(X, \hat{\omega}_{\mathrm{o}})} = \boxed{L_{\mathrm{e}}(X, \hat{\omega}_{\mathrm{o}})} + \int_{\mathbf{S}^2} \boxed{L_{\mathrm{i}}(X, \hat{\omega}_{\mathrm{i}})}\ \boxed{f_X(\hat{\omega}_{\mathrm{i}}, \hat{\omega}_{\mathrm{o}})}\ |\hat{\omega}_{\mathrm{i}} \cdot \hat{n}|\ d\hat{\omega}_{\mathrm{i}}$$

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

Vary sampling direction
uniformly over a sphere

$$L_{\mathrm{o}}(X, \hat{\omega}_{\mathrm{o}}) = L_{\mathrm{e}}(X, \hat{\omega}_{\mathrm{o}}) + \int_{\mathbf{S}^2} L_{\mathrm{i}}(X, \hat{\omega}_{\mathrm{i}})\ f_X(\hat{\omega}_{\mathrm{i}}, \hat{\omega}_{\mathrm{o}})\ |\hat{\omega}_{\mathrm{i}} \cdot \hat{n}|\ d\hat{\omega}_{\mathrm{i}}$$

```
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 Li!

Code: https://jacco.ompf2.com/2019/07/18/wavefront-path-tracing/, slightly modified to nearly match notation.

Importance Sampling

$$L_\mathrm{o}(X,\hat{\omega}_\mathrm{o}) = L_\mathrm{e}(X,\hat{\omega}_\mathrm{o}) + \int_{\mathbf{S}^2} L_\mathrm{i}(X,\hat{\omega}_\mathrm{i}) \underbrace{f_X(\hat{\omega}_\mathrm{i},\hat{\omega}_\mathrm{o}) \; |\hat{\omega}_\mathrm{i} \cdot \hat{n}|}_{} \; d\hat{\omega}_\mathrm{i}$$

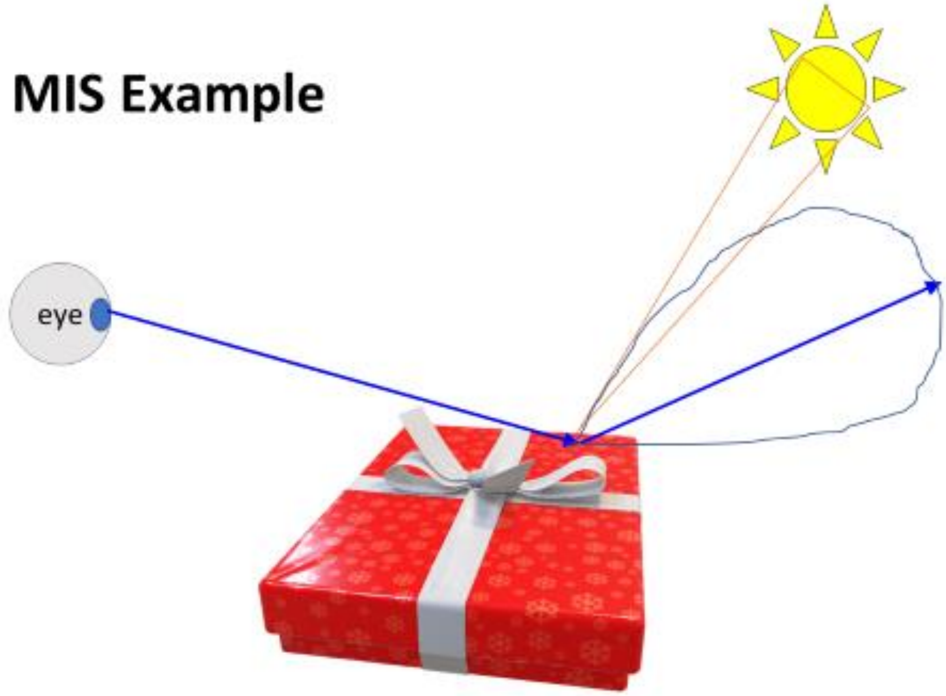
Vary sampling direction
dependent on BSDF and angle

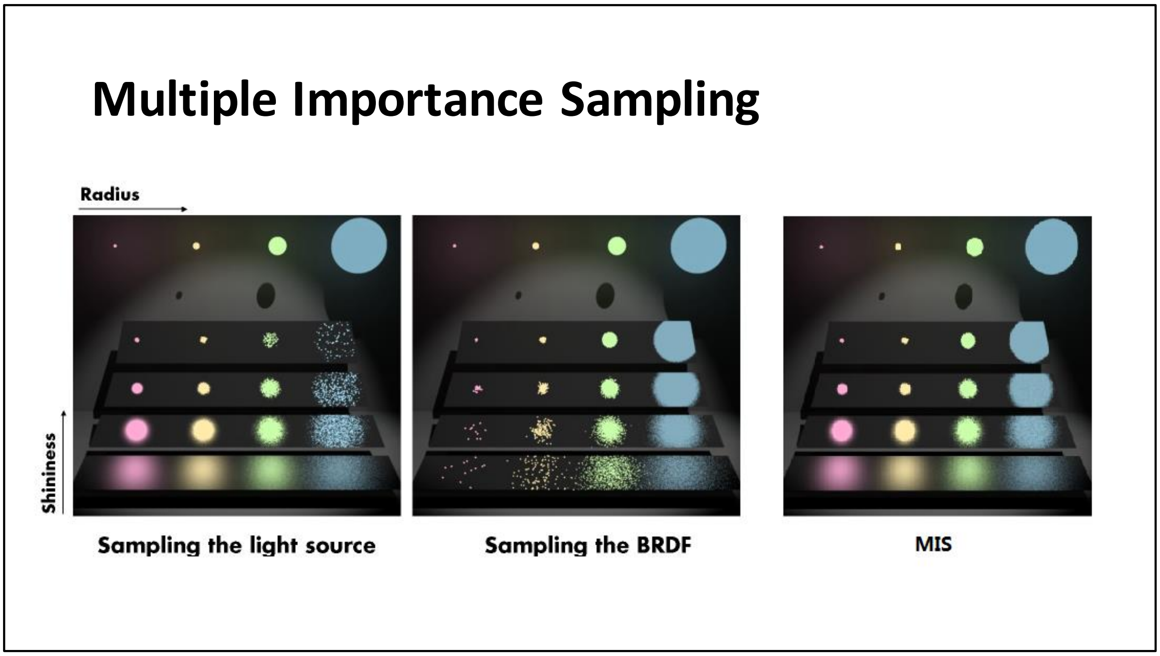
Multiple Importance Sampling

$$L_{\mathrm{o}}(X, \hat{\omega}_{\mathrm{o}}) = L_{\mathrm{e}}(X, \hat{\omega}_{\mathrm{o}}) + \int_{\mathbf{S}^2} \underbrace{L_{\mathrm{i}}(X, \hat{\omega}_{\mathrm{i}}) \; f_X(\hat{\omega}_{\mathrm{i}}, \hat{\omega}_{\mathrm{o}}) \; |\hat{\omega}_{\mathrm{i}} \cdot \hat{n}|}_{\substack{\text{Vary sampling direction} \\ \text{dependent on lighting,} \\ \text{BSDF, and angle}}} \; d\hat{\omega}_{\mathrm{i}}$$

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

MIS Example





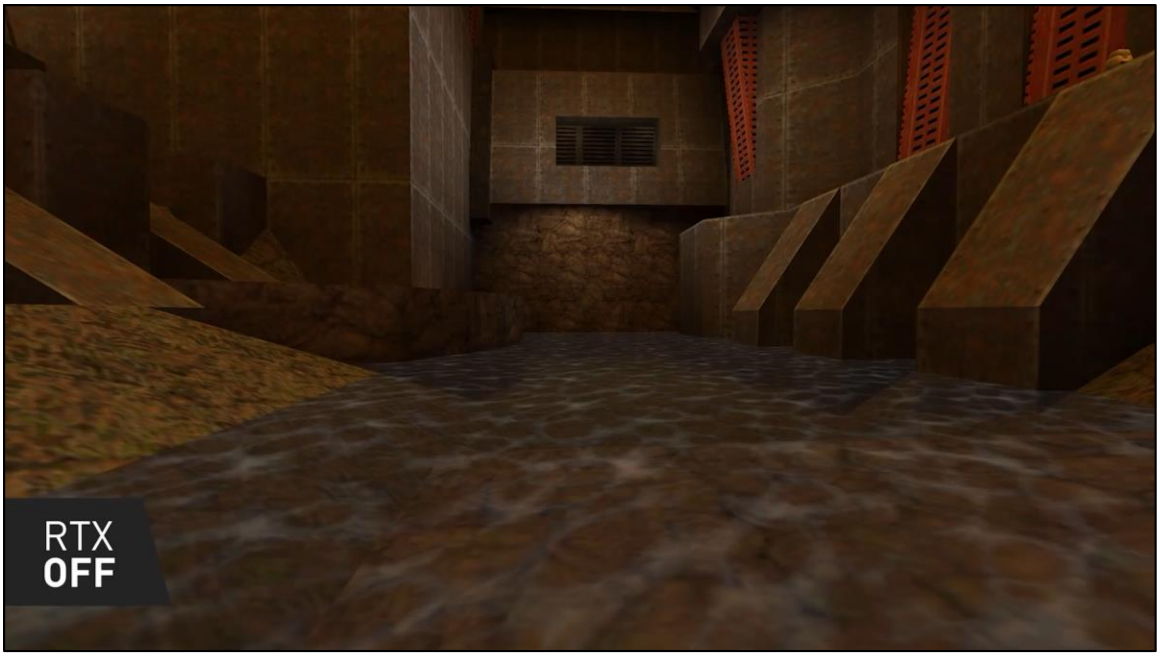
From **Multiple Importance Sampling** (MIS) demonstrated by Veach and Guibas [1] in 1995.

Path-Traced Game: Quake II

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

https://www.nvidia.com/en-us/geforce/campaigns/quake-II-rtx/

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


RTX
OFF

Denoising

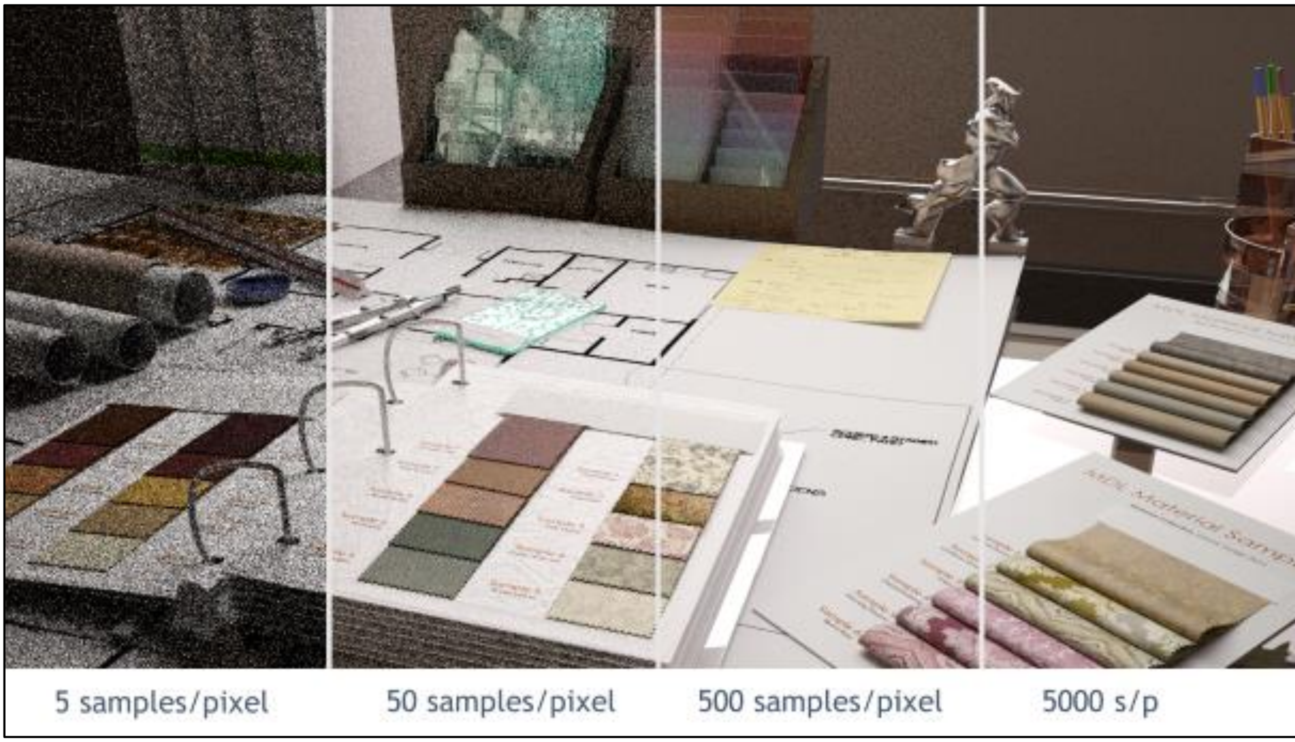
Tensor cores: evidence that fast denoising (enabled by tensor cores) helps a lot for ray tracing

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

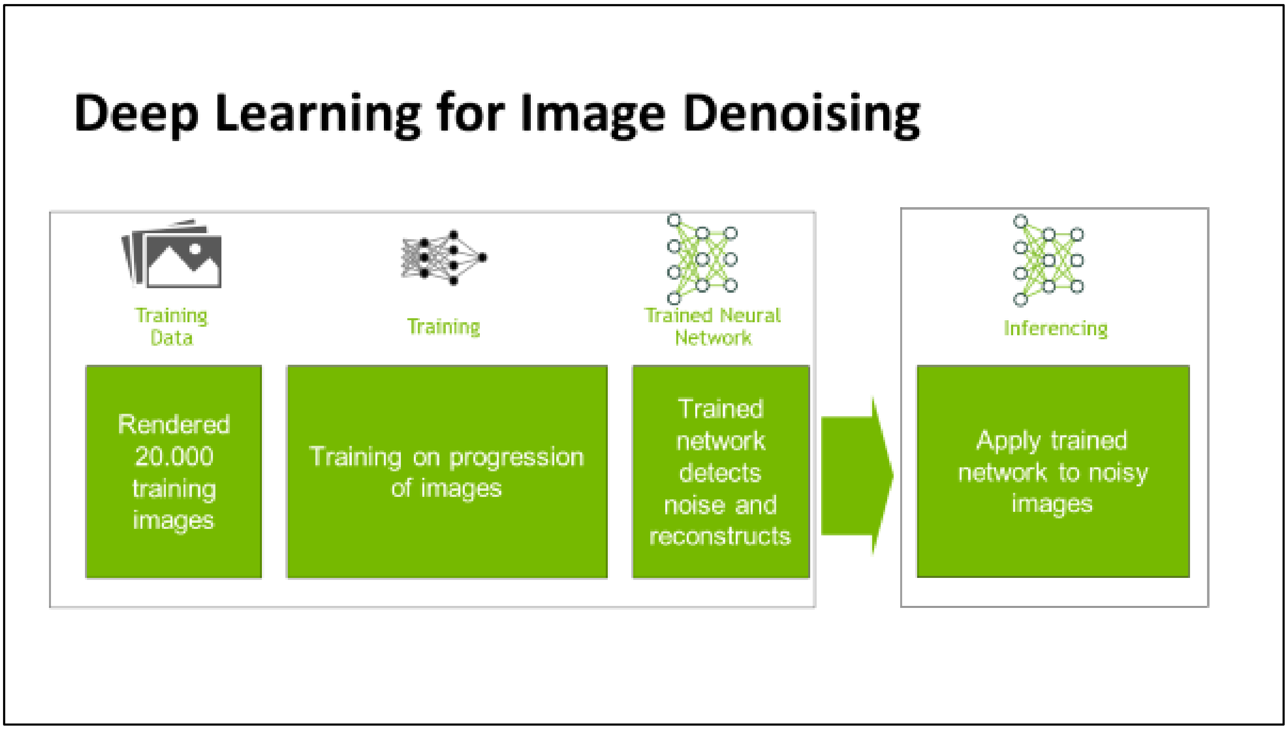
Denoising by Effect

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

Even films use denoising

https://developer.nvidia.com/gameworks-ray-tracing

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.

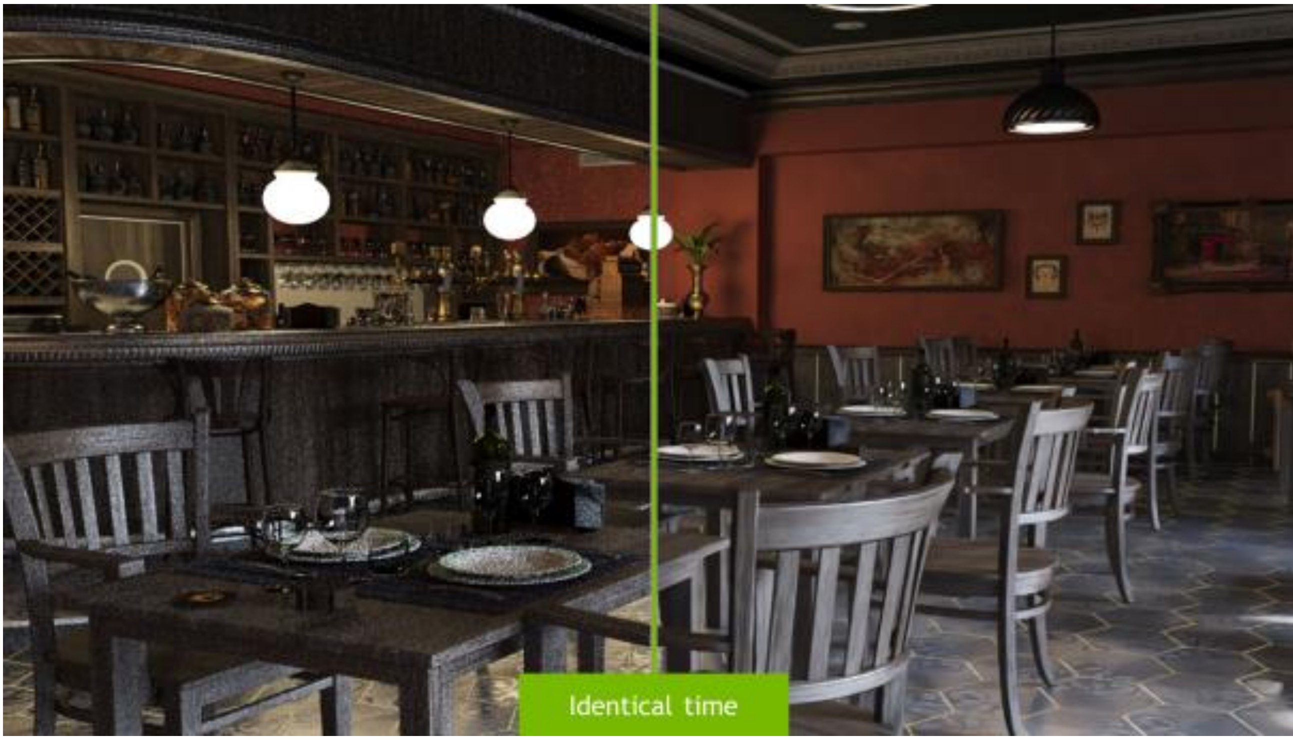
The task could be anything, like identifying types of vehicles in an image, reading the speed limit sign as it goes whizzing past, translating English to Chinese, etc.

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.

From NVIDIA’s “Deep Learning for Rendering” 2018



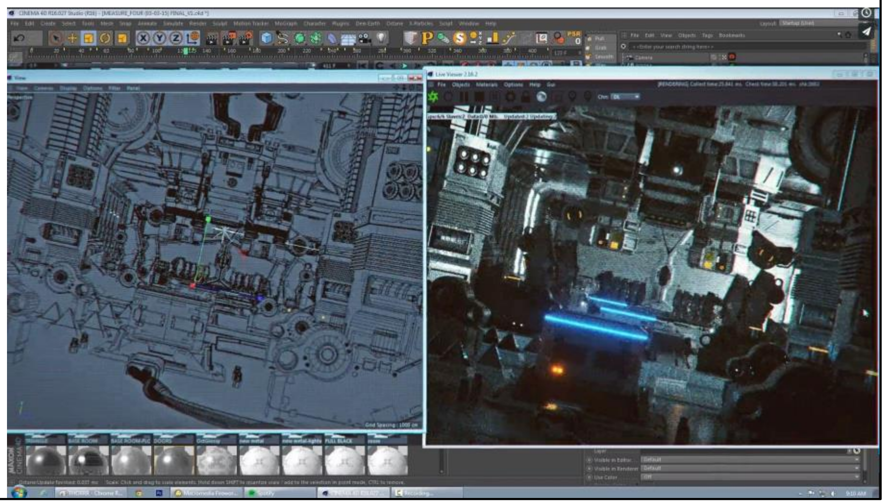
Tensor cores: evidence that fast denoising (enabled by tensor cores) helps a lot for ray tracing

From NVIDIA's "Deep Learning for Rendering" 2018

Movie Time!

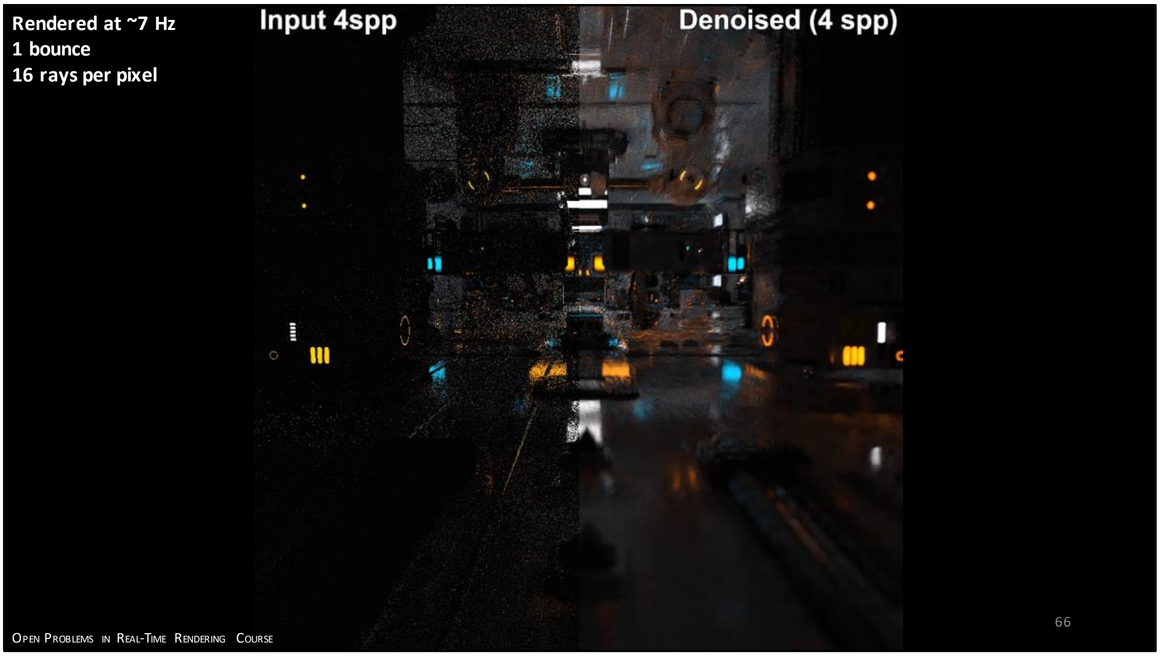
data downloadable!

Zero-Day,
by Beeple



Zero Day, data at https://www.beeple-crap.com/resources, WTFYW license. Movie at https://www.beeple-crap.com/films

10,500 emissive triangles
~350 emissive meshes



Zero Day, data at https://www.beeple-crap.com/resources, WTFYW license. Movie at https://www.beeple-crap.com/films

10,500 emissive triangles

~350 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

Zero Day, data at https://www.beeple-crap.com/resources, WTFYW license. Movie at https://www.beeple-crap.com/films

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 jcgt.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:
http://www.realtimerendering.com/blog/moving-targets-and-why-theyre-bad/

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

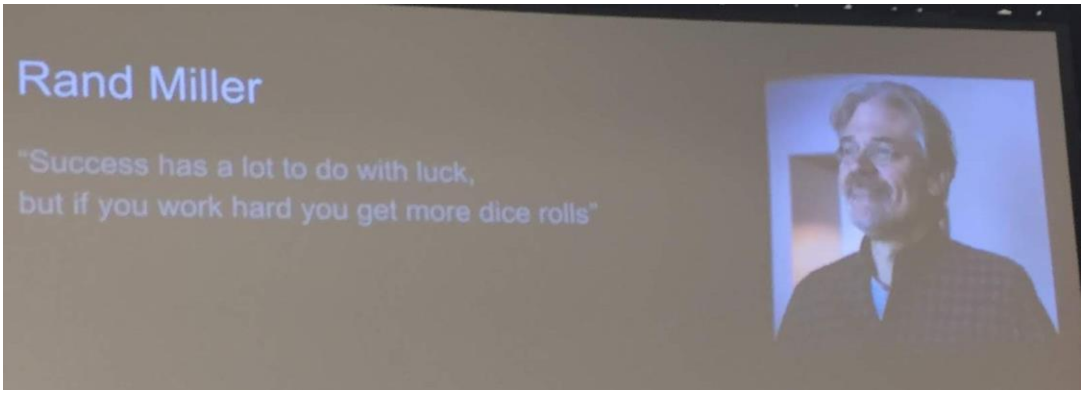
At Work

- 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 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)
- Made a ray tracing benchmark test suite, *Standard Procedural Databases* (1987)
- Created an informal journal: *Ray Tracing News* (1988)
- Coauthored the book *An Introduction to Ray Tracing* (1989) – now free!
- Helped with *Graphics Gems*: author, reviewer, code repo (1990-1995-present)
- Coauthored *Real-Time Rendering* (1999, 2002, 2008, 2018)
- Co-chaired I3D Symposium (general 2006, papers 2007, and again for 2020)
- Helped found a Gems-like journal: *journal of graphics tools* (1996)
- Co-founded open access version, *Journal of Computer Graphics Techniques* (2011)
- Created *Interactive 3D Graphics* MOOC (2013-present)
- Co-edited *Ray Tracing Gems* (2018-2019)
- 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