Raytracing

TraceX

See the script here :
https://docs.google.com/document/d/1K5NvSbPQWx8yGAFF6gtqmNwwStIQUjNpDVjL4rb4Viw/edit

We are the ray tracing group, TraceX.
[Have each group member introduce themselve.]

Performed by All

"One algo, we all knew of it, but no one had the guts to publish it. Because it takes so long to do" [4]

Restate the quote once more.

This is an image generated using raytracing, that highlights the photorealistic way it can render shadows and complex reflections/refractions.

Even though ray tracing was theorized by many, a real world implementation seemed far away due to computational stress.

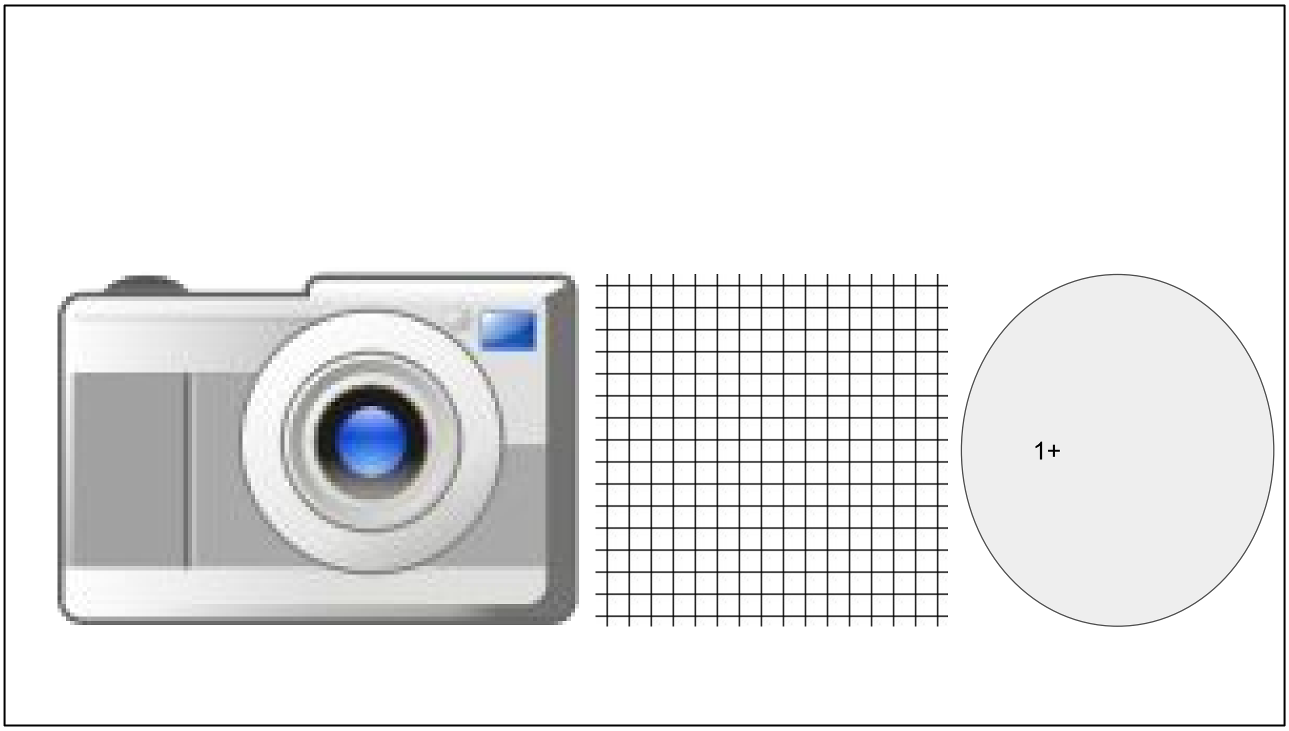
Then go to the general idea slide.

[Performed by Michael Jarvis and Nick Lang]

General Idea

- A very simple way to render a 3D scene
- Conceptually easy to add advanced concepts such as shadowing and reflection
- Very computationally expensive, initially was laughed at for the absurd resource requirements.

[Performed by mbregg]



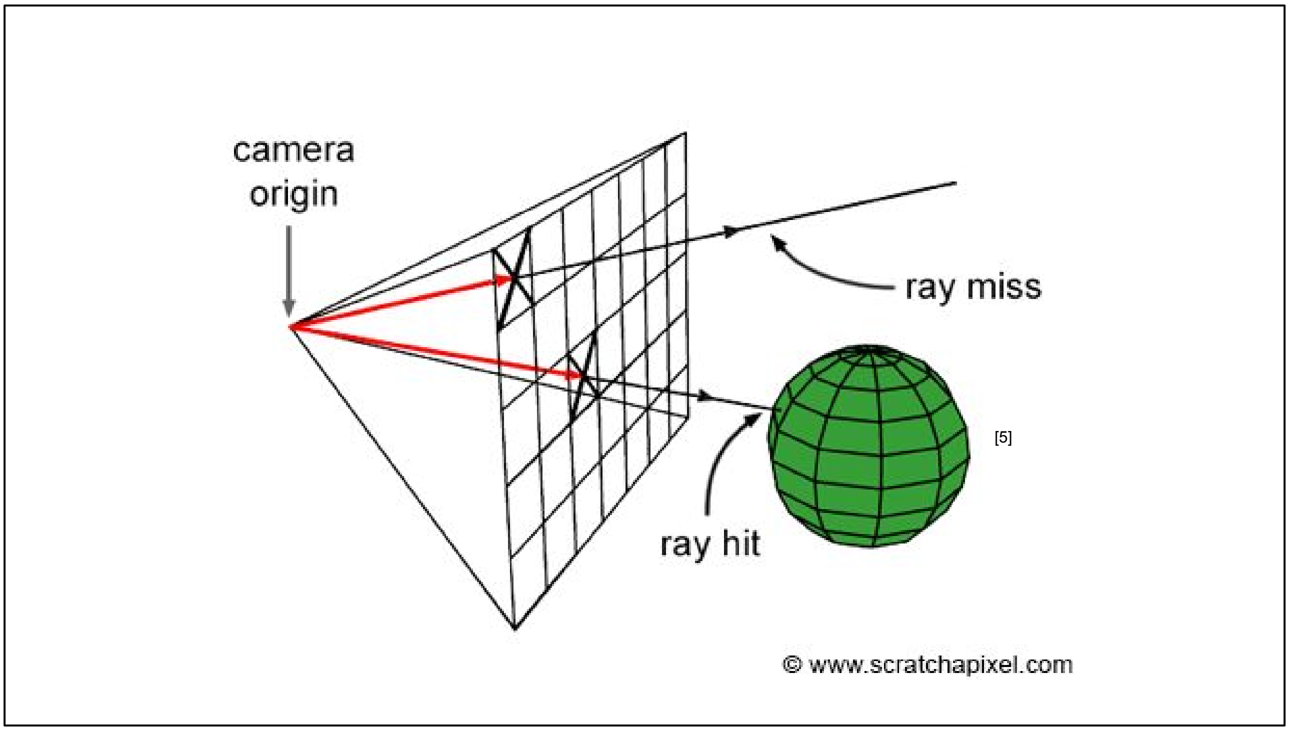
We have three types of things

A camera
A screen door
Some objects

Notice how the camera is in between the screen.
Consider each hole in the screen to be a pixel.

Objects can be spheres, cubes, any 3 dimensional shape.
A sphere, well a 2D cut out of a sphere, will be used here.

[Performed by MBregg]



Here we have an example of it all put together.

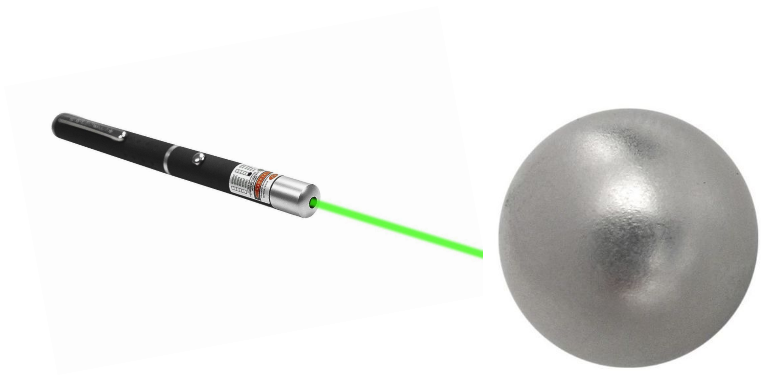
We place the camera behind the screen, and the scene to be displayed in front. (Not much point on an empty scene, if we just want a canvas of a certain color, that's quite easy to do!)
We shoot out rays from the camera, through each pixel in the scene.
If the ray collides with an object, then we set that pixel to be the color of the object we collided with.

Calculation wise, spheres are convenient, as it is easy to calculate if a ray intersects with a sphere.

The actual details of of calculating this intersection boils down easily researchable basic geometry, and is outside the scope of this presentation.

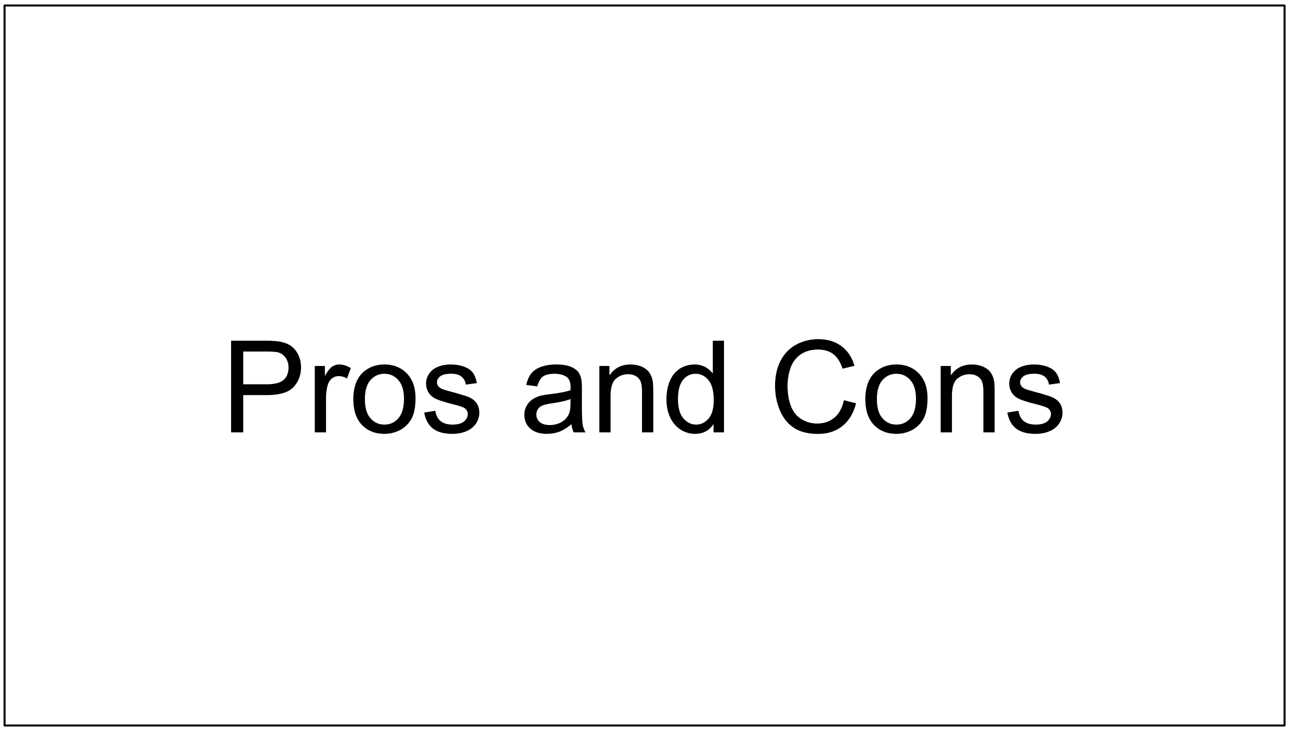
[Performed by MBregg]

Demo



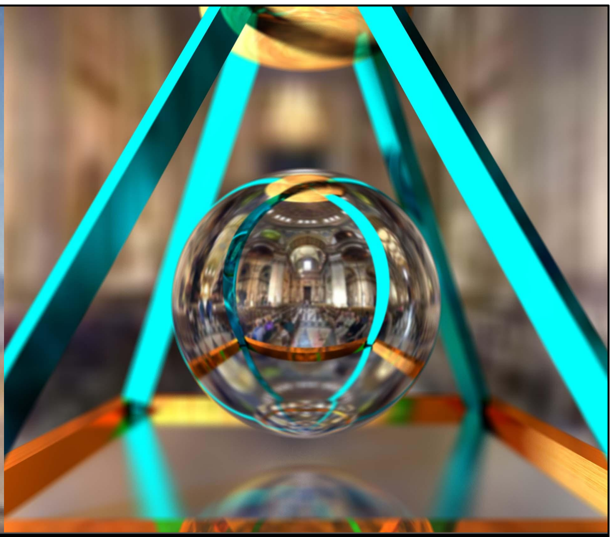
Show off demo with magnetic ball, and a actual laser pointer, and even a screen door!

[Performed by all]



Pros and Cons

Pros and Cons!
[performed by Michael Wulber]



Photorealistic Images

There are three main ways to render images:

1. Rasterization
2. Classical Ray Casting
3. Ray Tracing

The biggest benefit of ray tracing is that it accurately simulates the behavior of light—calculating realistic shadows, reflections, and refractions. This is done by recursively generating shadow, reflection and refraction rays each time a ray intersects an object in the scene. In this way, rasterization and classical ray casting fail to provide the level of detail that ray tracing does.

Simple scanline rendering, or rasterization is an object centric method. This means that for each object in the scene, the graphics engine first identifies the vertices; then calculates the edges; and lastly fills the pixels defined with those bounds with the color of material. Unlike ray tracing, rasterization is not recursive, and does not factor reflection or refraction into the final image.

Classical ray casting is an image centric rendering method much like ray tracing; however, unlike ray tracing it is not recursive. This results in some acceptable lighting, but does not handle reflections or refractions.
Rendering details such as the shadows of Walter White's individual beard hairs being shown on his jacket, the specular highlights on reflective materials, and the world bending effects of transparent objects such as curved glass are possible due to ray tracing.

[performed by Michael W.]

Notice in right image, we have a full reflection of the room rendered in the glass sphere.
The room itself is also full of shadows.
This is a perfect example of ray tracings strengths.

The left most image shows a nearly photorealistic image of an actor from a popular TV series, showing how close to photo realism ray tracing can get.
Notice how even his beard is shadowed on the shirt.

Walter White image from :
http://hplusmagazine.com/2013/05/21/here-comes-photorealistic-virtual-reality/

Glass sphere image from:
https://courses.cs.washington.edu/courses/cse557/08wi/projects/trace/

RAYS!RAYS!RAYS!RAYS!RAYS!RAY
RAYS!RAYS!RAYS!RAYS!RAYS!RAY
RAYS!RAYS!RAYS!RAYS!RAYS!RAY
RAYS!RAYS!RAYS!RAYS!RAYS!RAY
RAYS!RAYS!RAYS!RAYS!RAYS!RAY
RAYS!RAYS!RAYS!RAYS!RAYS!RAY
RAYS!RAYS!RAYS!RAYS!RAYS!RAY
RAYS!RAYS!RAYS!RAYS!RAYS!RAY
RAYS!RAYS!RAYS!RAYS!RAYS!RAY
RAYS!RAYS!RAYS!RAYS!RAYS!RAY
RAYS!RAYS!RAYS!RAYS!RAYS!RAY

Now to give some hard numbers as for why Ray tracing was for so long, considered to be intractable!
Imagine we want to get a 1000x1000 image of a certain scene, with only one object in it.
Then, 1000 by 1000 is 1 million rays per frame!
That's already a lot of rays.
But now, imagine if we want 1000 objects.
1000 objects is not a huge amount in modern video games and animations.
Then each ray must be checked to see if it intersects with each of the thousand objects, and 1000 * 1 million is a massive 1 billion.
1 Billion operations per second is massive, especially for the 80s.
This should give some insight as to why Ray tracing is usually not done in real time, and it's history of being an "intractable" process.

[Performed by MBregg]

Unfortunately, photo realism is not free. Scenes with a high level of detail, such as the one shown at our website's homepage or frames in a modern Pixar movie take days to render. Obviously, this frame rate would not fly in a real-time application, such as a video game. One way to handle this would be to use rasterization or ray casting instead. Both are computationally faster than ray tracing, and still used in many modern games. Rasterization is popular more abstract games that have simple materials and more minimalistic styles. Ray casting is used often in pseudo-realistic games--its hallmark feature is a sort of matte finish on the models. However, if you want your game to have specular highlights, light bending water, beard hair shadows, or other features of the like, you must use ray tracing. This can be done by lowering the recursive depth value of the ray tracing algorithm, and employing many rendering 'tricks' to simplify the models, textures, or other things that reside in the scene. Tricks aside, being able to obtain photo realistic results in a real time application often times running at around 60 frames per second nowadays is amazing. This demonstrates the real versatility of the ray tracing algorithm.

[Performed by Michael W.]

Shadows

Shadows were an unsolved problem.
Raytracing provided a easy solution.

The basic idea is….
When a ray hits an object, emit another ray! Call this a shadow ray.

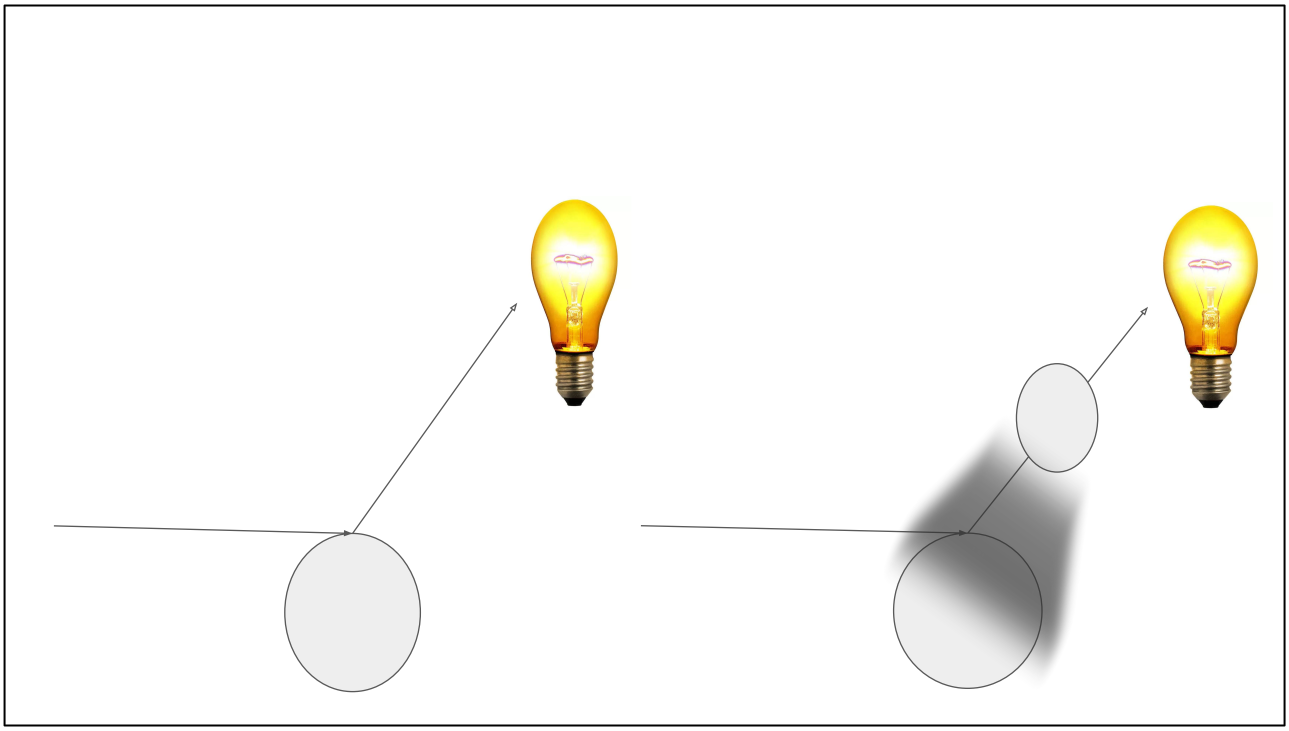
The shadow ray originates from the point the emitted ray hits the object, and goes to the light source.

We trace a ray from the point the emitted ray hits the object, to the light source!

If on the way to the light source, we hit an object, then we have something in the way!

In this case, our pixel is in shadow, so color accordingly.

Alas, this method requires tons more rays!



The basic idea is....

When a ray hits an object, emit another ray! Call this a shadow ray.

The shadow ray originates from the point the emitted ray hits the object, and goes to the light source.

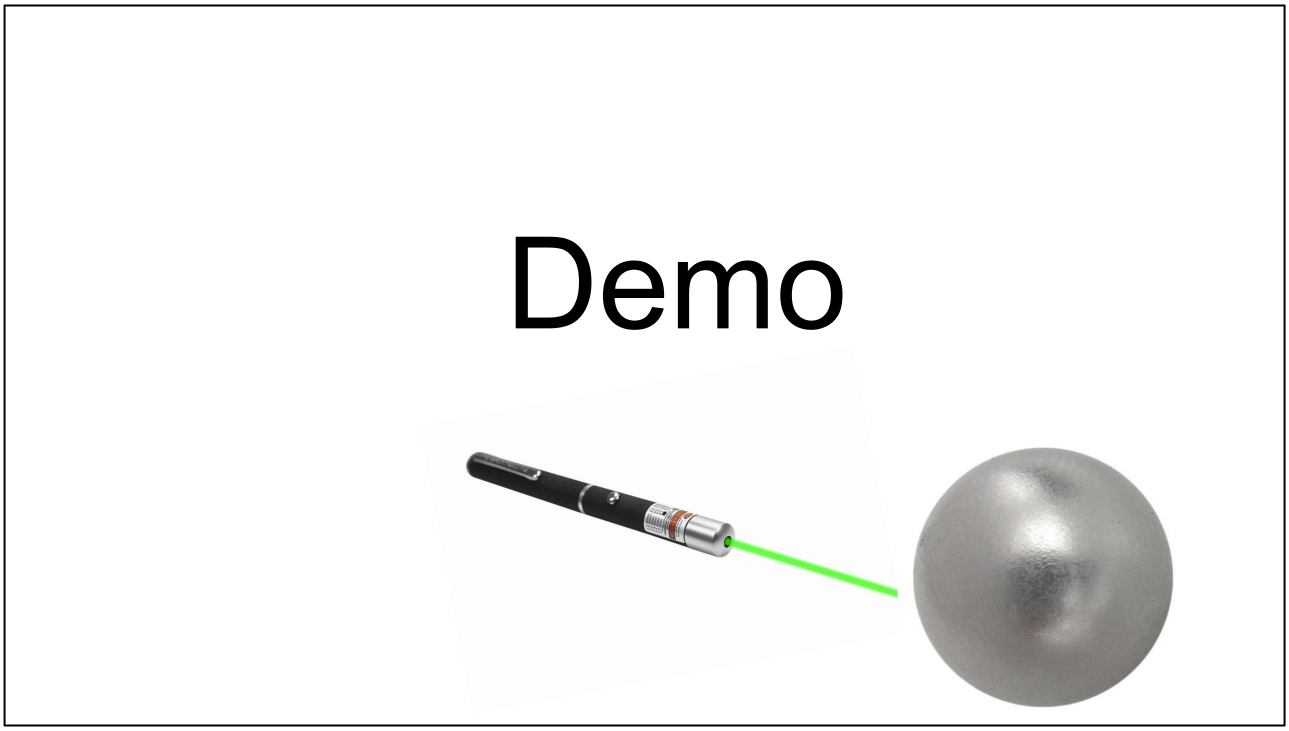
If on the way to the light source, we hit an object, then we have something in the way!

In this case, our pixel is in shadow, so color accordingly.

While this method is simple to do, you might have already noticed a problem

This method requires tons more rays!

Ray tracing was already expensive, and this made it more so!

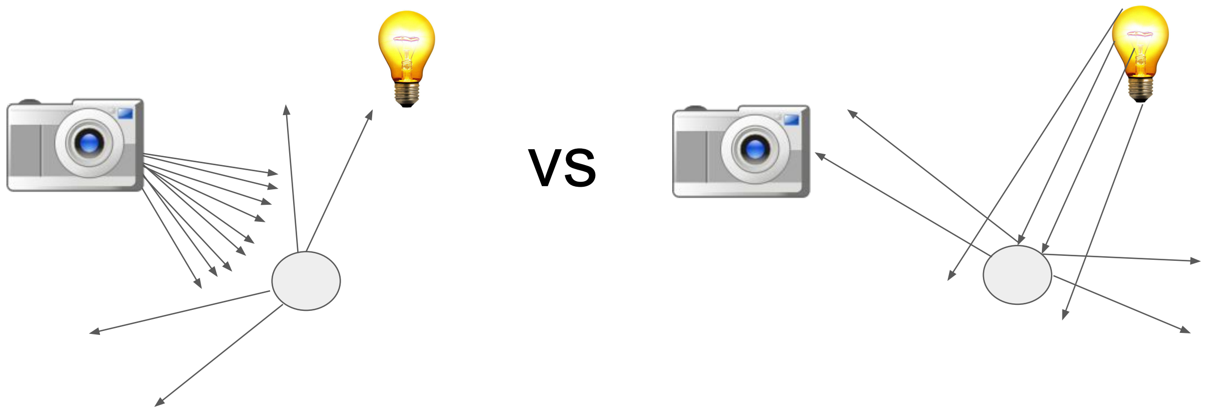


Demo

Show off demo with magnetic ball, and a actual laser pointer, and even a screen door!

[Performed by all]

Backward vs forward ray tracing



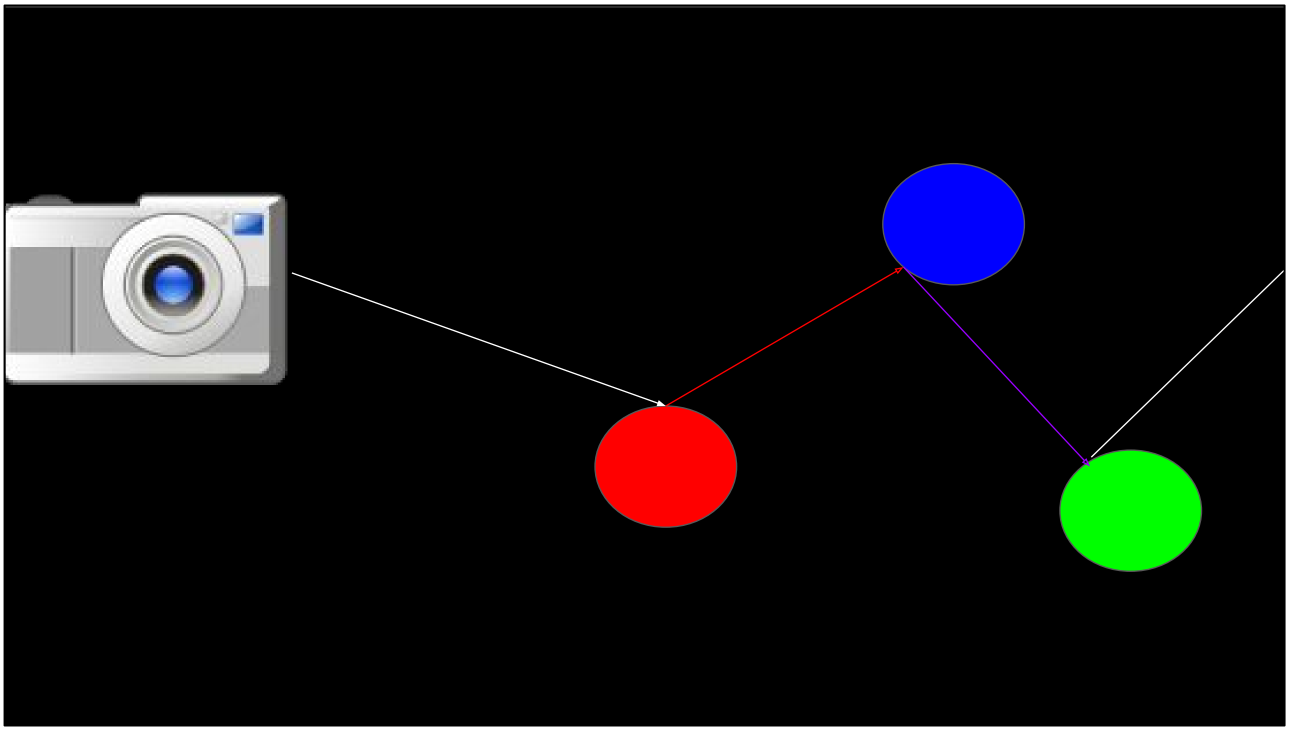
-Backward raytracing entails following rays from a camera towards objects, and then assessing their distances from various light sources and other objects.
-Forward raytracing however, starts at the light source and heads towards the objects. The advantage of forward being that the rays are more efficient; most rays coming off the light source are relevant to rendering, whereas the rays from forward tracing are rather inefficient.
-Forward tracing does however do a better job of determining object colors.
-Backwards tracing makes sometimes false assumptions about relevant light sources
-A healthy combination of both is the optimal way to implement raytracing, and most real world implementations do indeed use both.
[Performed by Michael Jarvis]

Reflection and Shadows were both in a similar position before Ray tracing provided a solution.

Both were considered to be unsolved problems in computer graphics.

Both had a easy solution provided by Ray tracing, at the cost of making ray tracing even more expensive.

Mirror image from : https://i.pinimg.com/736x/c5/1b/b9/c51bb9e8aba8ae616ba8a3503acc8419--green-paint-colors-wall-colors.jpg



Much like shadows, we emit another ray when an existing ray hits an object.

However, instead of heading off towards a light source, we reflect a ray at a certain angle, the angle of incidence!

Then if that ray hits another object within some distance, then multiply the color of the hit object by some reflection factor to weaken the color, and use that color to color the original ray!

But what if we hit another reflective object?

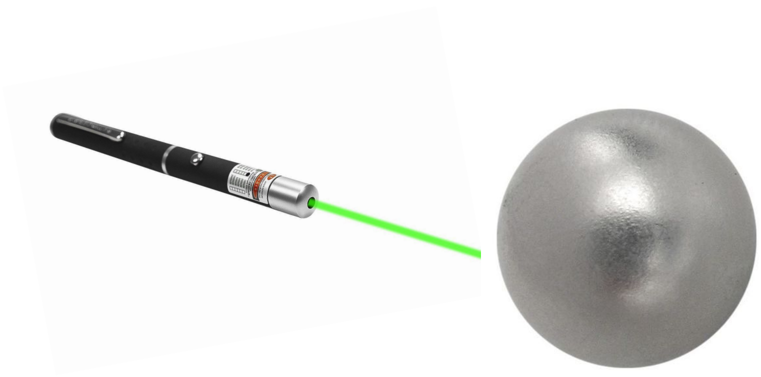
Then do the process again reflective!

Once again, emit a new ray, if that ray hits an object, then the color of the original reflection ray be influence by reflectionFactor*colorOfHitObject.

And if that object is also reflective, then have to recursively continue, up to a certain depth!

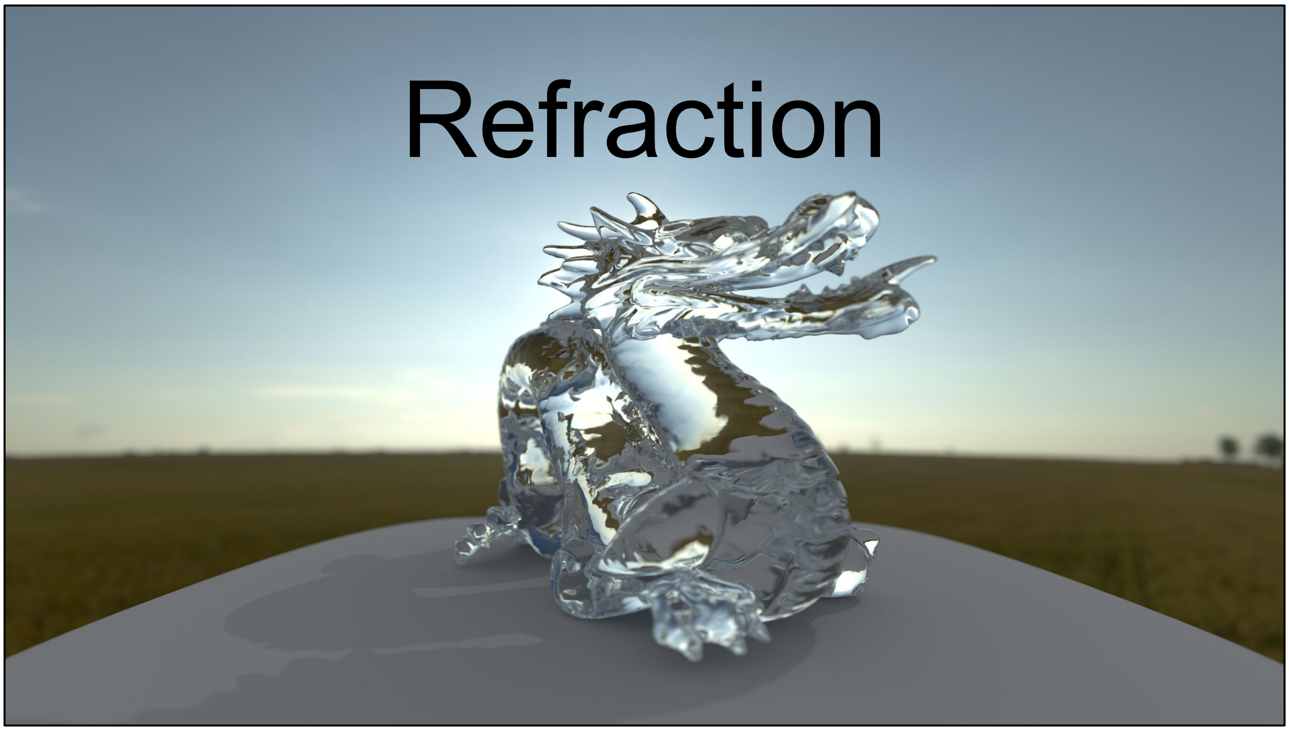
This means a ton more rays are being spit out!

Demo



Show off demo with magnetic ball, and a actual laser pointer, and even a screen door!

[Performed by all]

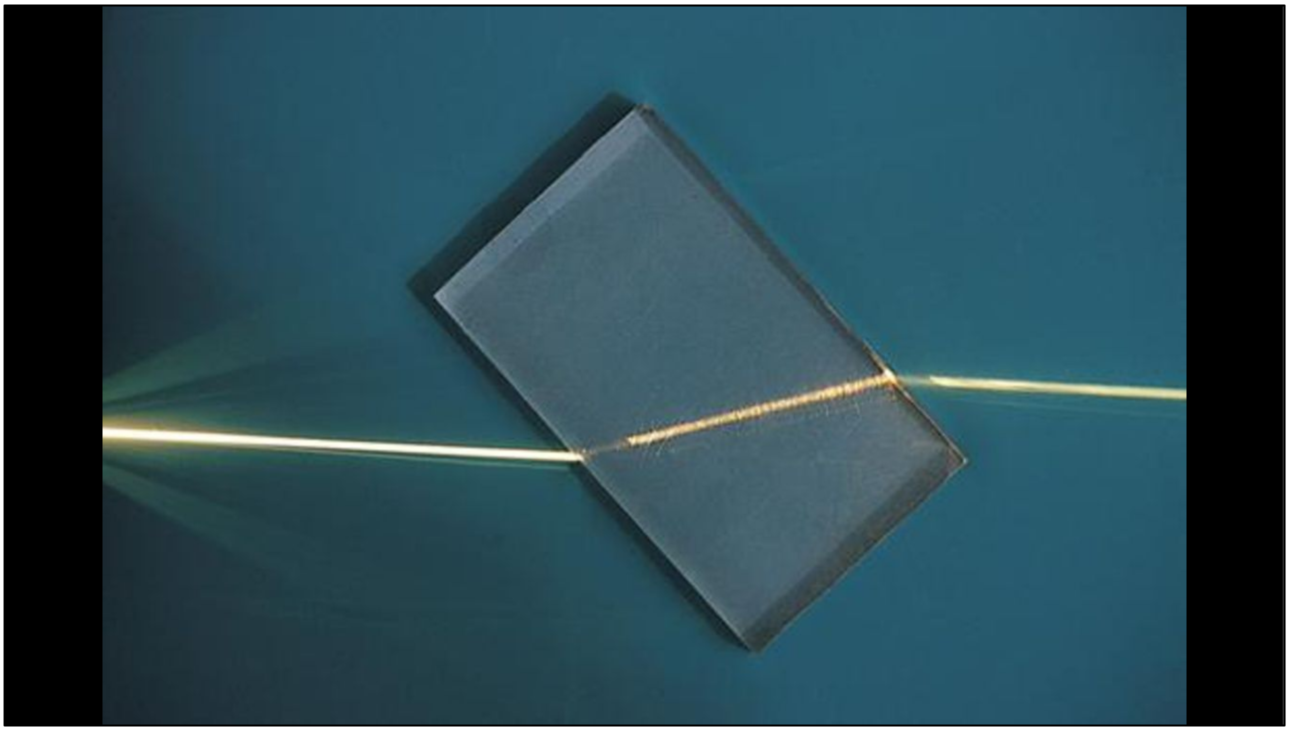


Refraction

Refraction!

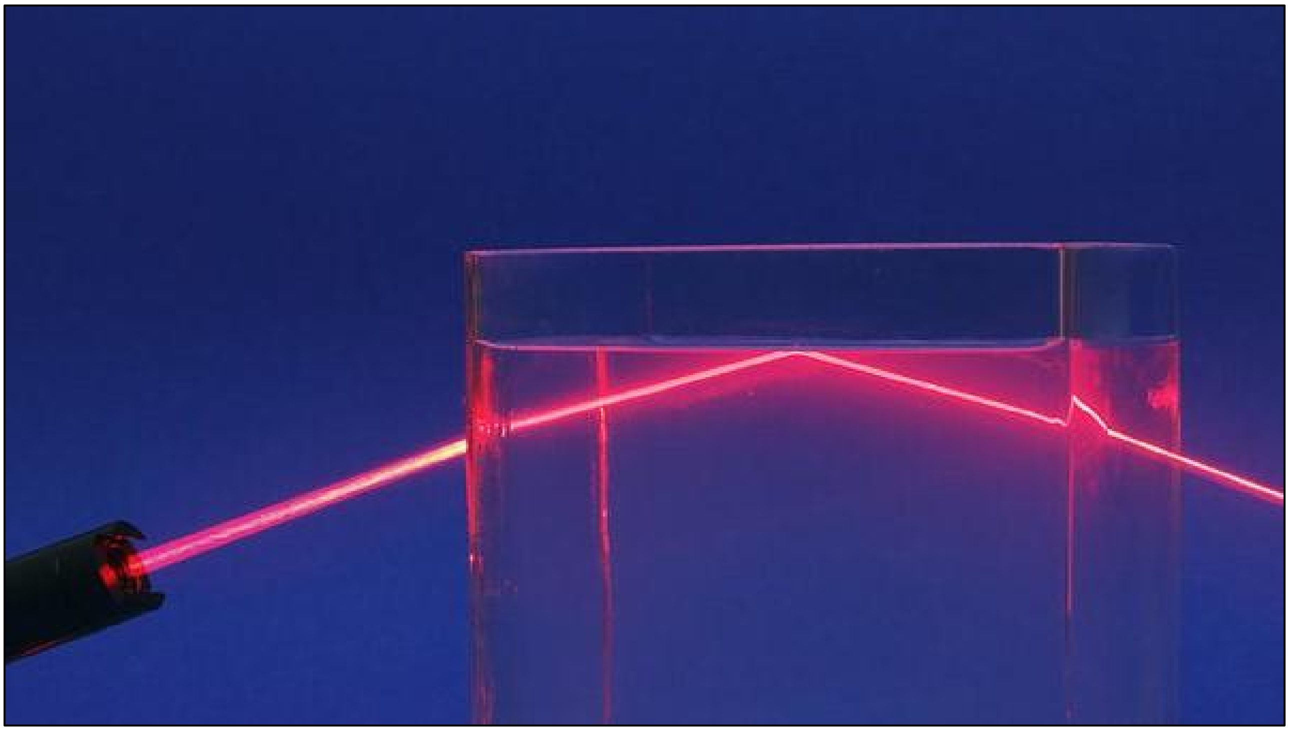
Refraction rays are another type of secondary ray that may be recursively created when a ray intersects an object.

[performed by Michael W.]



Refraction occurs when light rays encounter a new medium. In computer graphics, this occurs when a ray intersects an object that has a transparent material with a different optical density, otherwise known as index of refraction. When this occurs, a secondary refraction ray is generated with a slightly altered direction due to Snell's Law. This new ray eventually intersects another face of the object, and dependent upon the angle of incidence, the slopes of the two intersected faces, and the optical densities of relevant materials, multiple things may happen. First, if the angle of incidence is less than the critical angle (a bound determined by Snell's Law), a new secondary refraction ray will be generated and projected into the scene. Once again, the angle will be slightly altered due to a change in medium.

[performed by Michael W.]

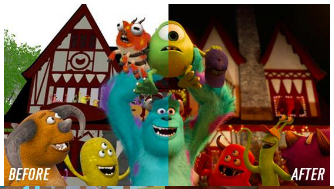


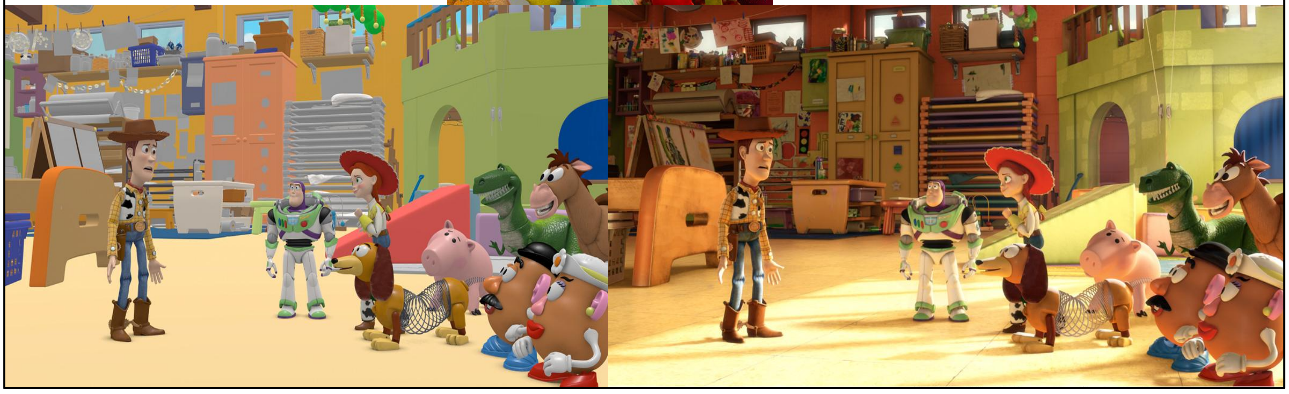
However, if the angle of incidence is greater than the critical angle and the new medium is less optically dense than the former, total internal reflection will occur. For further information on why this happens and how to calculate the effects, research the Fresnel equations and Schlick's approximation. If you are feeling lazy, there is pseudo code for this posted in the resource section of our website. In more realistic and computationally expensive ray tracing engines, multiple refraction rays may be generated instead of a single one to simulate the dispersion of different colors during refraction.

[performed by Michael W.]

Demo

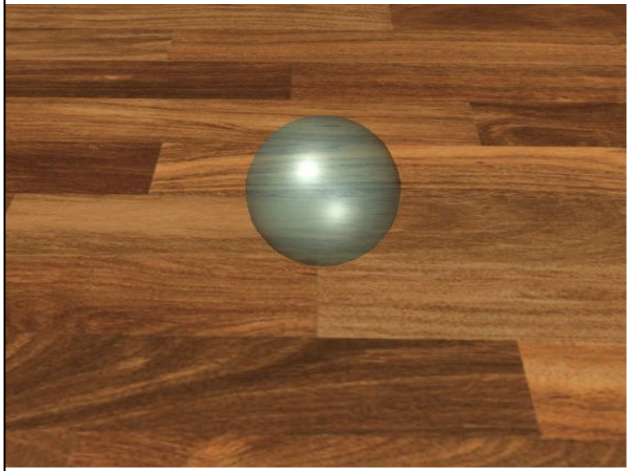
Raytracing in Film

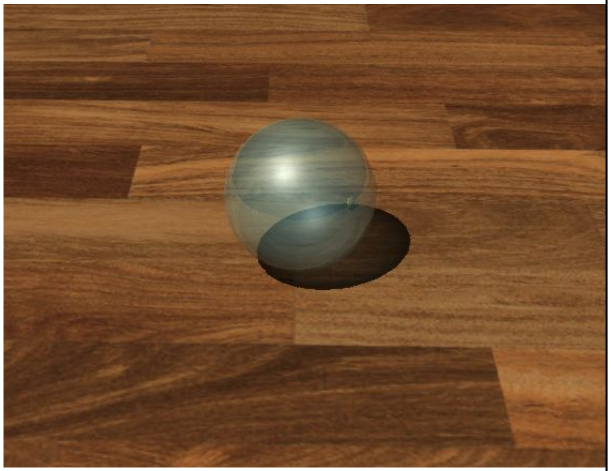




Paula:
-Pixar didn't use raytracing until the movie "Cars" which released in 2006 but even then it wasn't a full implementation, they only used it for specific scenes and main characters.
-Claude Kalache made a controversial decision when he wanted to change how light worked at Pixar because it meant many of the artists had to relearn how to create the animation scenes. Prior to this they had to manually put in the shadows. This transition heavily increased the work that their renderers had to do. The artists would work during the day as the renderers worked at night.
-The change had to do with raytracing. They wanted the effect it produced but it was too costly due to needing to map out millions of beams of light.
-The motive behind adding ray tracing was that it created a more accurate environment in terms of reflection, refraction, colored shadows, etc.
-This big change was first implemented in the movie Monster's University. In these images you can see how the rendering using raytracing made the animations look much more realistic.
-They used a new Global Illumination system "Global illumination = 'super hard core ray tracing'. This system automatically placed reflections and shadows based on where the lights of the scene were located.
-Pixar uses a hybrid rendering approach, which is something you may want to consider when implementing your own.

Raytracing in Autodesk Maya

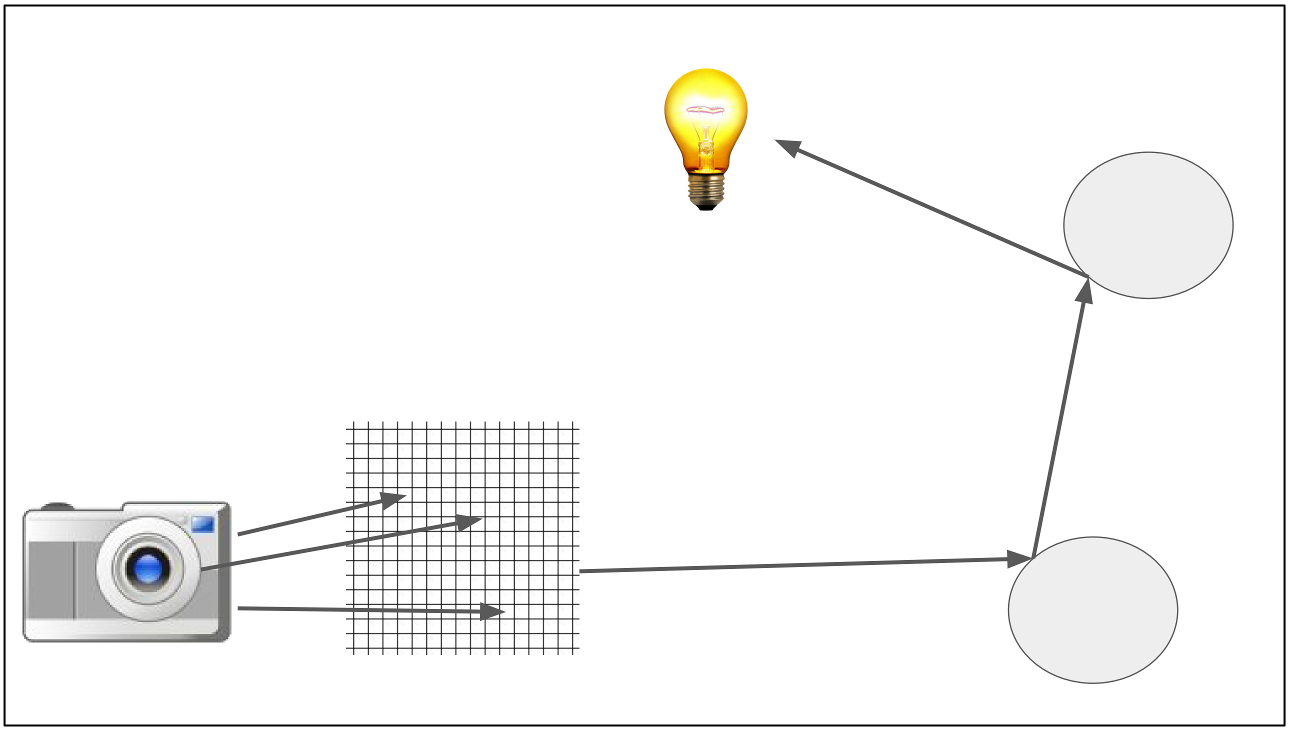




Paula:
-In Autodesk Maya, the algorithm it uses is: for each pixel in the image, set the ray, trace the ray, calculate illumination.
-When generating rays, it traces a ray for each pixel in the image plane
-It simulates light rays from the light source to the eye
-In render settings under the Maya Software raytracing options the user can modify:

-Reflections: max number of times a light ray can be reflected
-Refractions: max number of times a light ray can be refracted
-Shadows: “ “ “ “ “ “ reflected and, or refracted and still cause an object to cast a shadow
-Bias: Corrects issue of dark areas or incorrect shadows on 3D motion blurred objects

-Raytracing is only efficient if the scene fits in memory. A way to lessen the burden of rendering is to separate different aspects into different scenes whenever possible.
-Raytracing is built into Maya Software renderer so it is a simple implementation (despite taking longer to render), where you can modify the aspects of it freely.



In summation, we have a camera, a viewing frame, objects, and a light source. Shadow, reflection, and refraction rays in combination with material properties and illumination allow us to generate an image with accurate and realistic lighting. Previously impassable rendering limitation were easily handled with the development of these techniques.
Walk through the process

[Performed by Michael Jarvis]

References

https://cs.stanford.edu/people/eroberts/courses/soco/projects/1997-98/ray-tracing/types.html
-Charity Lu, Alex Roetter, Amy Schultz

http://www.schorsch.com/en/kbase/glossary/raytracing.html
-Georg Mischler

https://www.scratchapixel.com/lessons/3d-basic-rendering/ray-tracing-overview
-Scratchapixel

https://graphics.stanford.edu/courses/cs148-10-summer/docs/2006--degreve--reflection_refraction.pdf
-Bram de Greve

Reference Images

http://assets.sbnation.com/assets/2820593/pixar-before-after.jpg

https://www.fxguide.com/wp-content/uploads/2012/04/noGI.jpg

https://www.fxguide.com/wp-content/uploads/2012/04/finalGI.jpg

-Disney/Pixar

https://freestocktextures.com/photos-wood/

-FreeStockTextures (used in Maya example)

See the script here :
https://docs.google.com/document/d/1K5NvSbPQWx8yGAFF6gtqmNwwStIQUjNpDVjL4rb4Viw/edit

We are the ray tracing group, TraceX.
[Have each group member introduce themselve.]

Performed by All