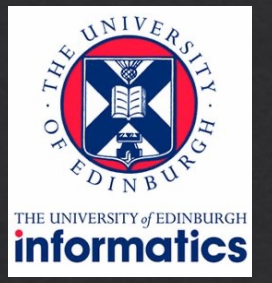
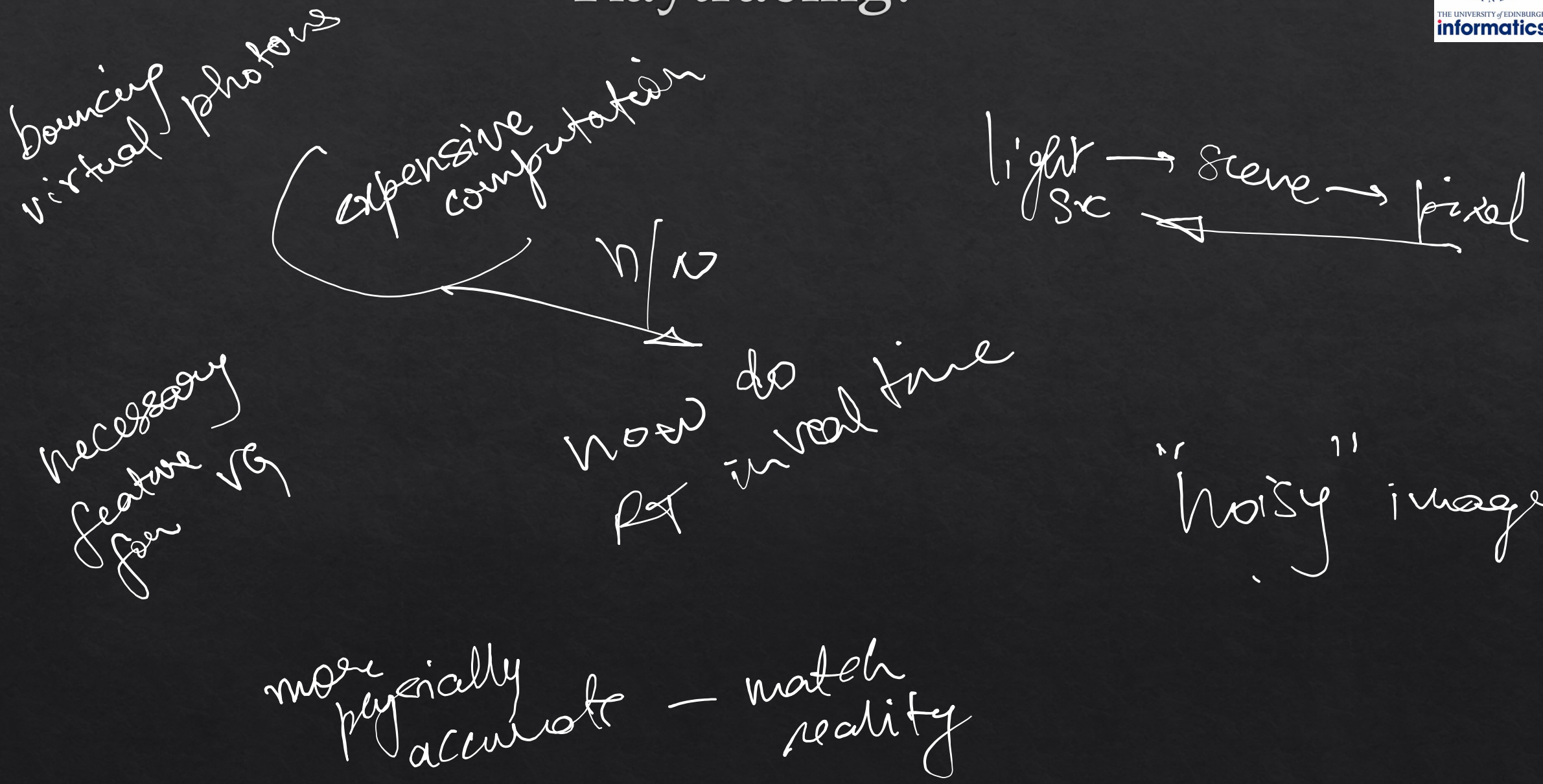


Computer Graphics

Lecture 6: Raytracing

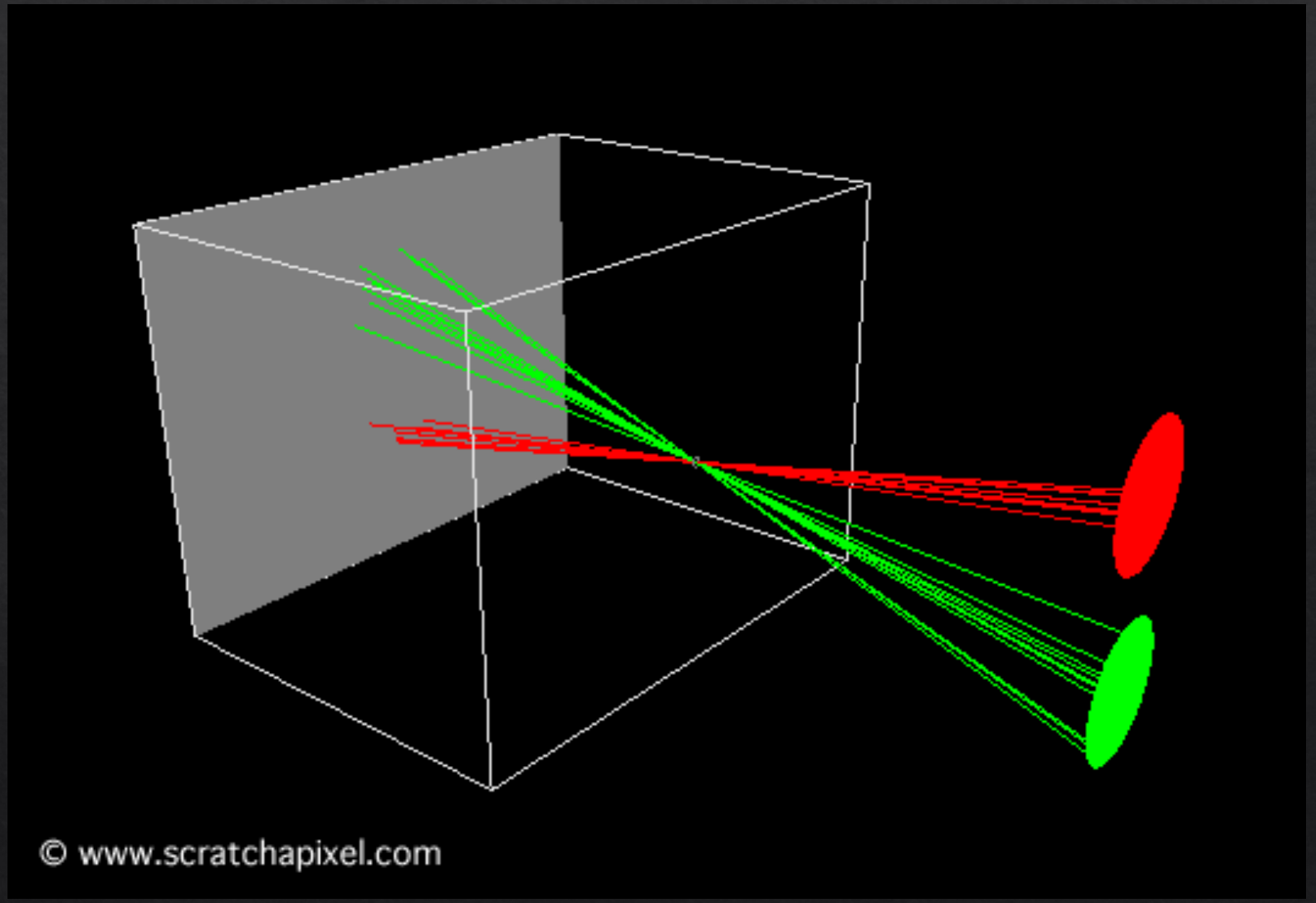
Kartic Subr

Raytracing?

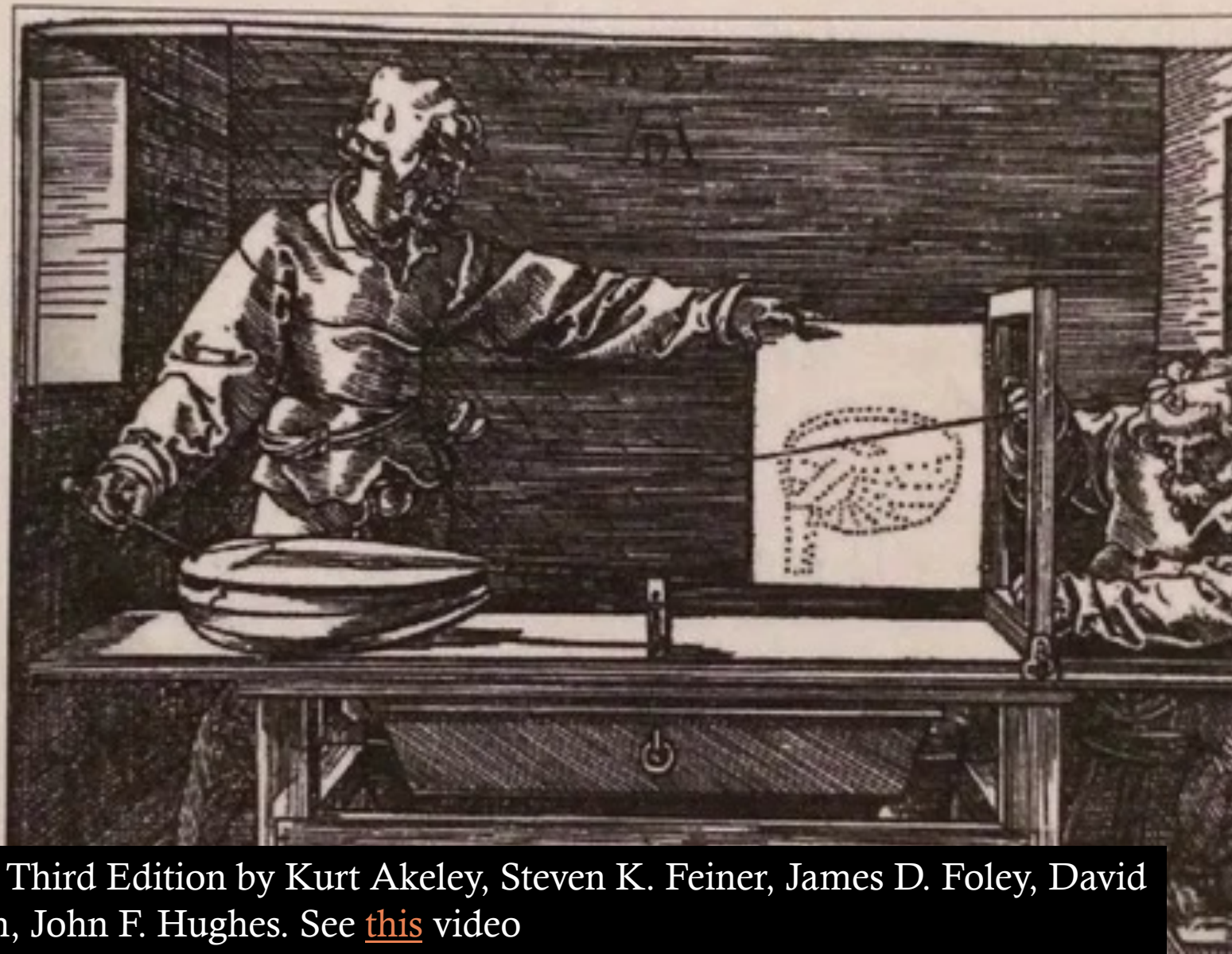

bouncing virtual photons
expensive computation
n/2
light src
scene
pixel
necessary feature for VR
how do RT in real time
"noisy" image
more physically accurate — match reality

Pinhole camera

Ibn al-Haytham (965-1040 AD)

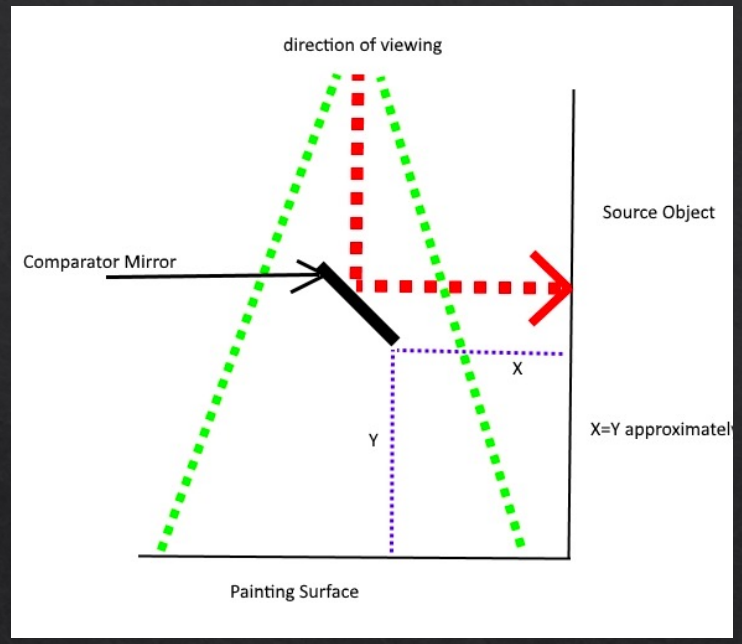


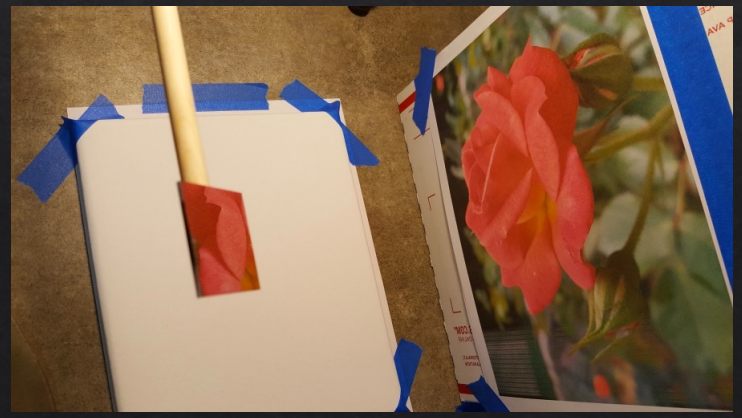
Albrecht Dürer, 1525



Computer Graphics: Principles and Practice, Third Edition by Kurt Akeley, Steven K. Feiner, James D. Foley, David F. Sklar, Morgan McGuire, Andries van Dam, John F. Hughes. See this video


Johannes Vermeer
17th Century





Vermeer's comparator mirror

https://youtu.be/deAK0oV6Vt8

Some techniques for shading machine renderings of solids

by ARTHUR APPEL
IBM Research Center
Yorktown Heights, N. Y.

INTRODUCTION

Some applications of computer graphics require a vivid illusion of reality. These include the spatial organization of machine parts, conceptual architectural design, simulation of mechanisms, and industrial design. There has been moderate success in the automatic generation of wire frame,[1] cardboard model,[2] polyhedra,[3,4] and quadric surface[5] line drawings. The capability of the machine to generate vivid sterographic pictures has been demonstrated.[6] reflection, the effect of surface texture, tonal specification, and the transparency of surfaces. At present, there is the additional problem of hardware for display of the calculated picture. Devices presently available use lines or points as the principal pictorial element and are not comparable to oil paint, or wash, or crayon in the ability to render the subtle changes in tone or color across an area. The best we can hope to do is to simulate the halftone process of printing.

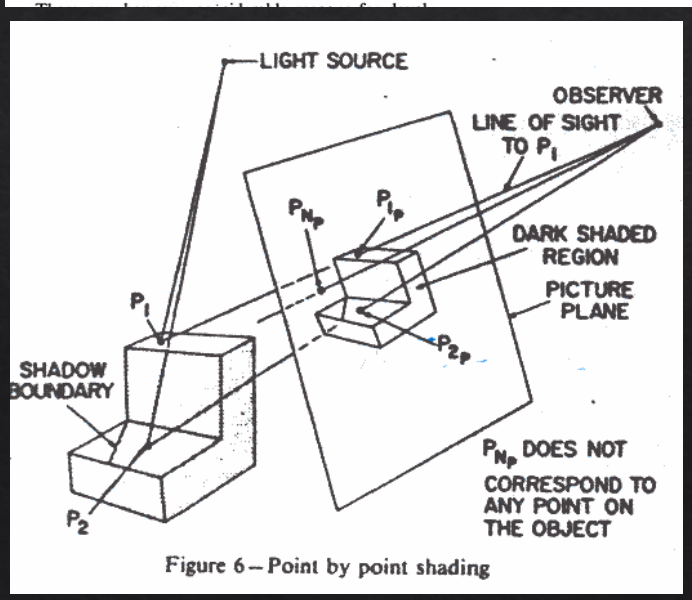


Figure 6—Point by point shading

1968

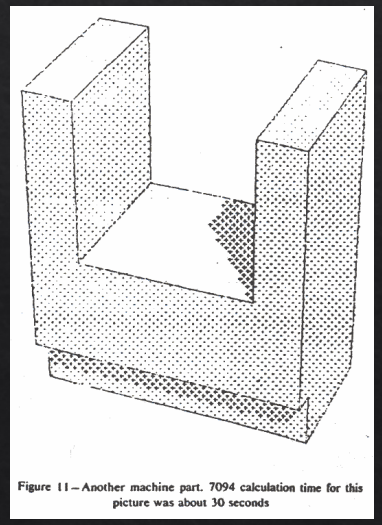


Figure 11—Another machine part. 7094 calculation time for this picture was about 30 seconds

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.

Key Words and Phrases: computer graphics, computer animation, visible surface algorithms, shading, raster displays
CR Category: 8.2

Introduction

Since its beginnings, shaded computer graphics has progressed toward greater realism. Even the earliest visible surface algorithms included shaders that simulated

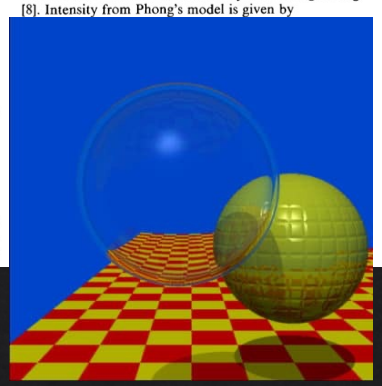
The role of the illumination model is to determine how much light is reflected to the viewer from a visible point on a surface as a function of light source direction and strength, viewer position, surface orientation, and surface properties. The shading calculations can be performed on three scales: 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

1979



Ray casting

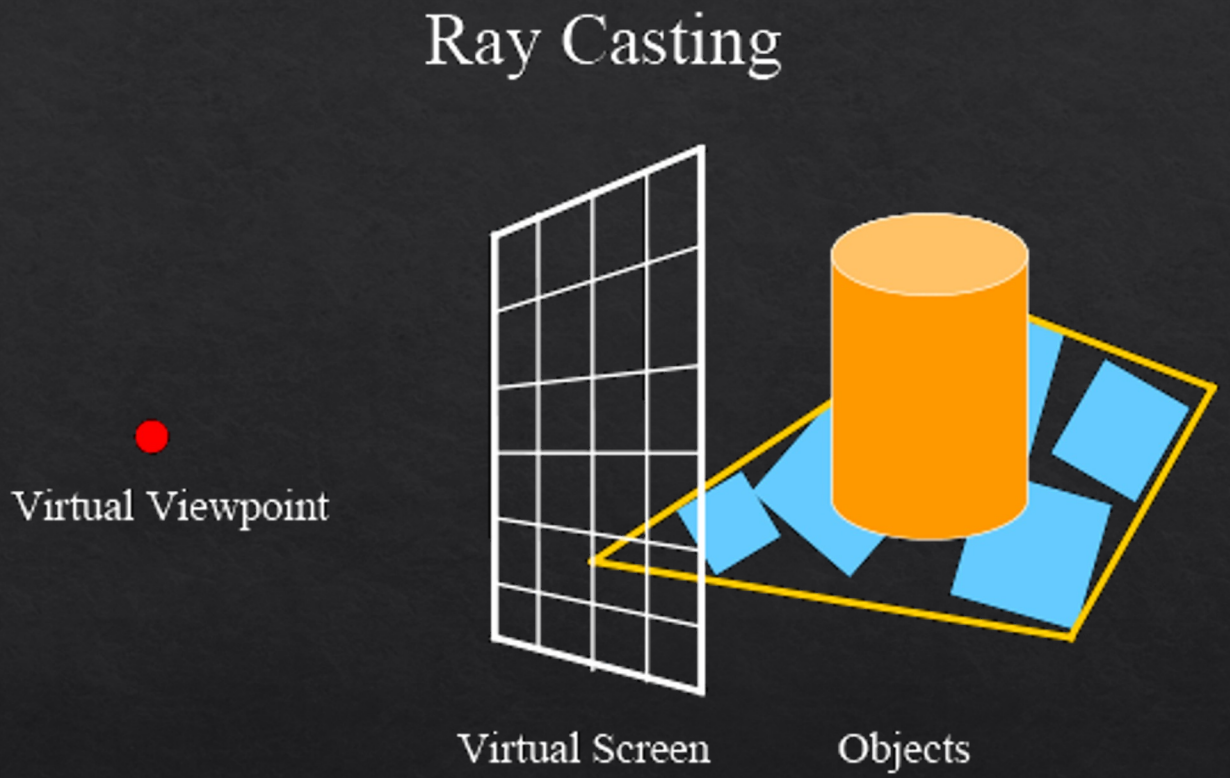
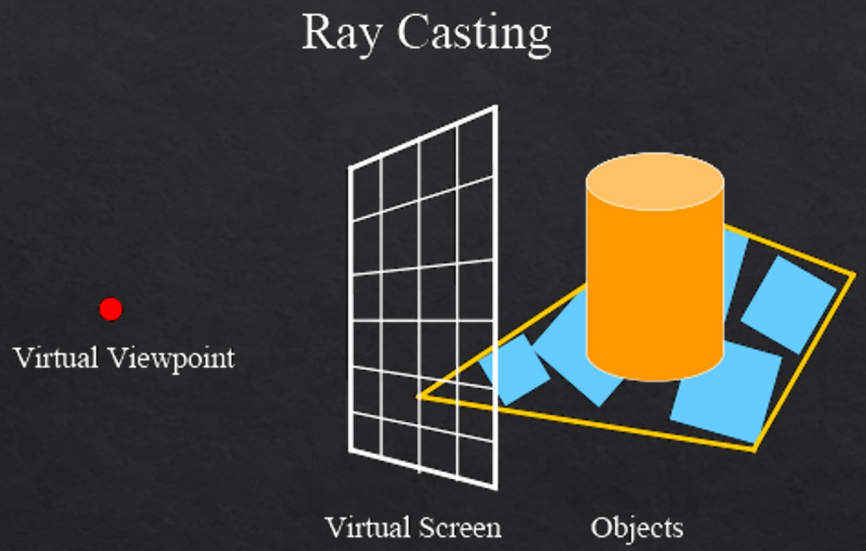


image credit http://studiomaven.org/Context__Overview_of_Ray-Traced_Rendering.html

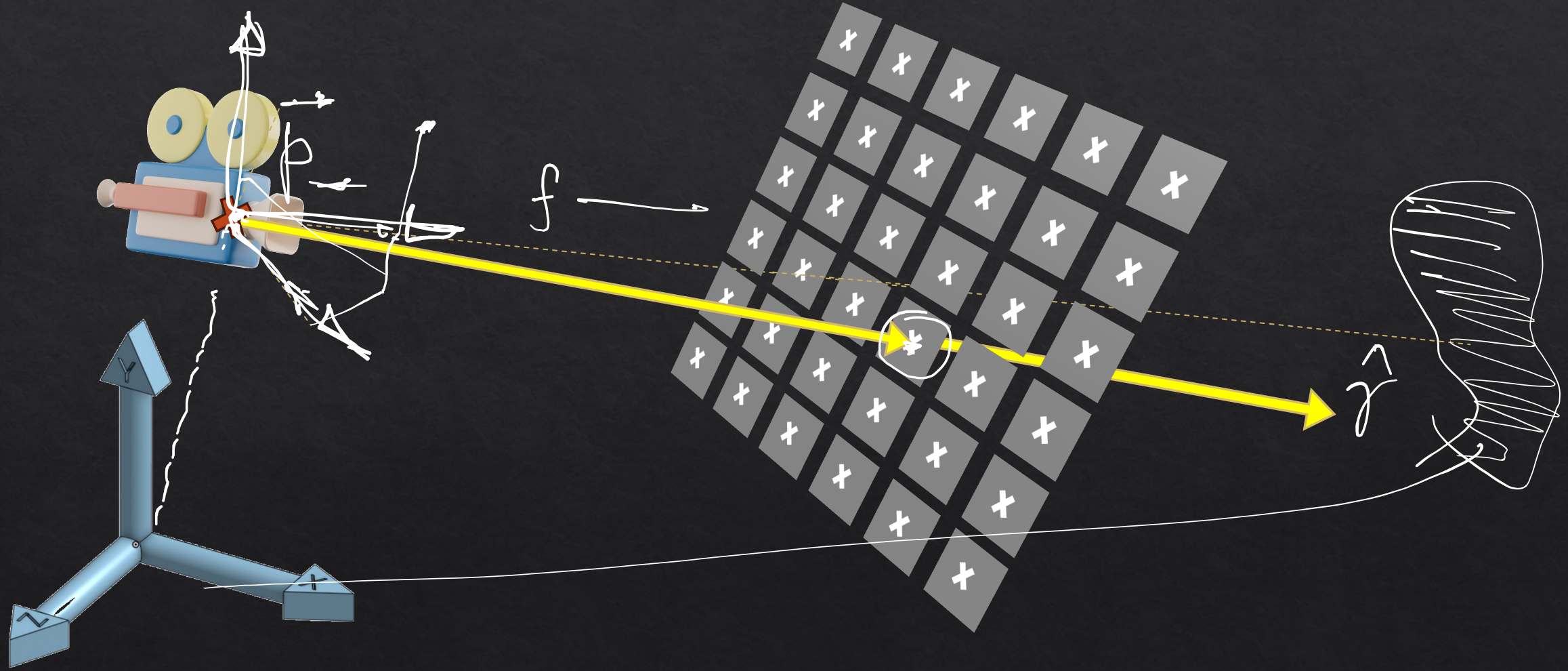
Ray casting



for each pixel p
1 cast ray r through p into the scene
2 test if r intersects objects in the scene
if yes
find closest intersection point h
3 shade h
else
return background colour (black)
endif

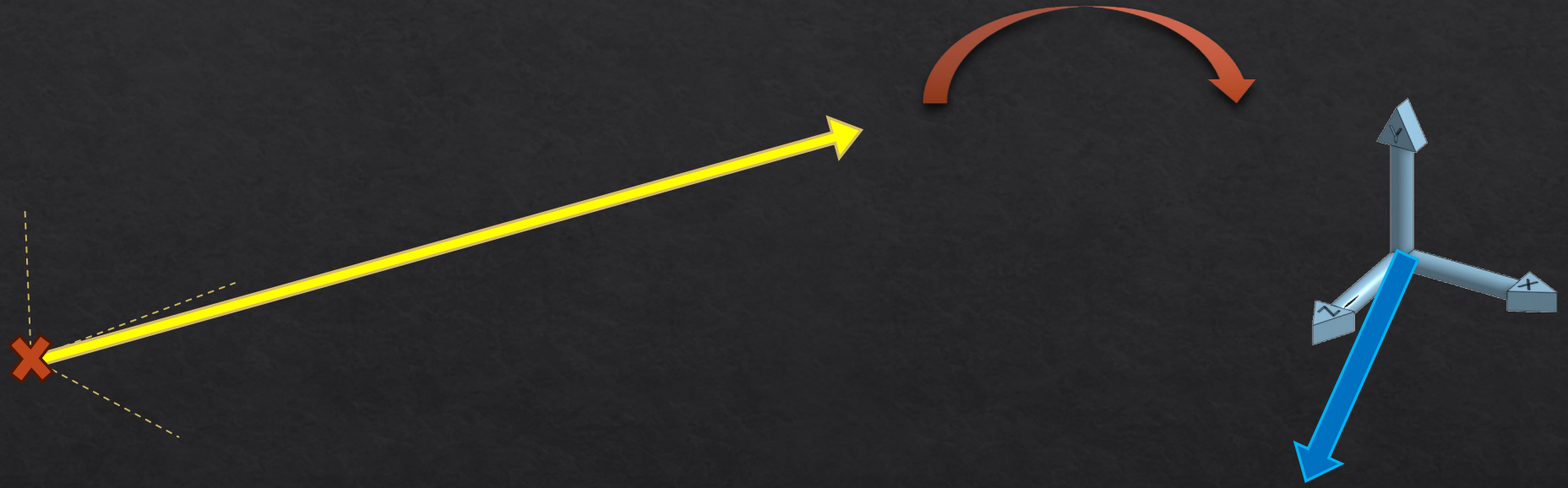
image credit http://studiomaven.org/Context_Overview_of_Ray-Traced_Rendering.html

Ray casting



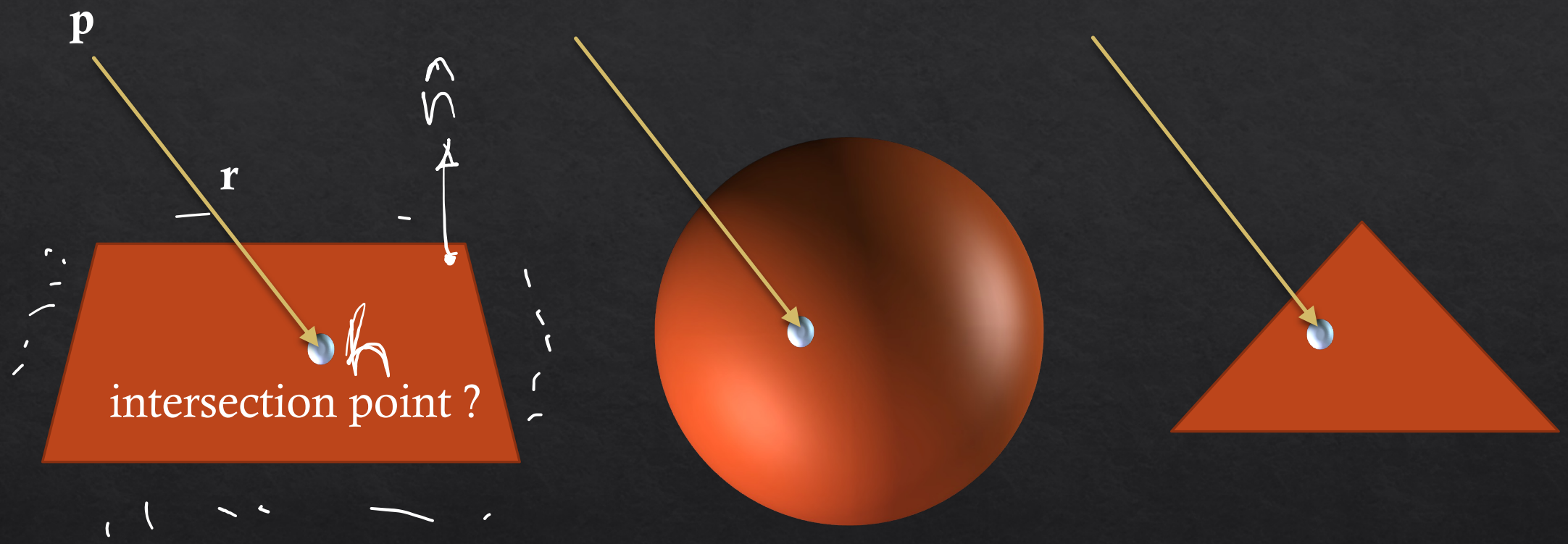
https://www.americanscientist.org/article/a-pixel-is-not-a-little-square

Ray casting



https://graphicscompendium.com/raytracing/02-camera

Ray intersection



Ray – plane intersection

$$h = \vec{p} + t\hat{r}$$

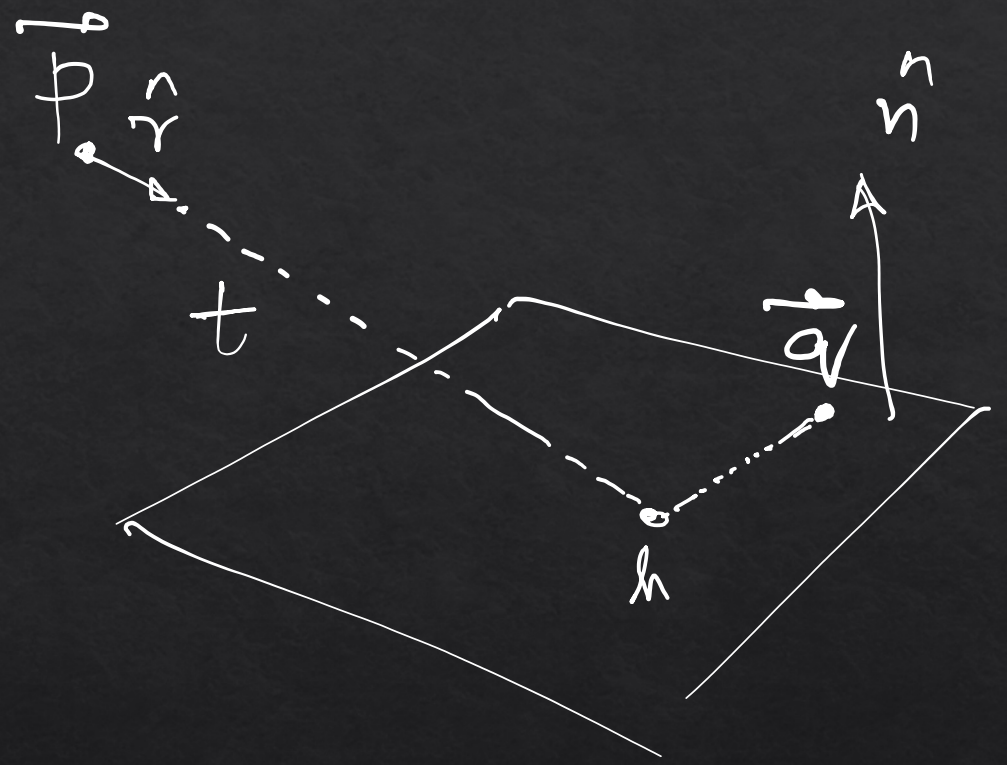
$$(q - h) \cdot \hat{n} = 0$$

$$(\vec{q} - \vec{p} - t\hat{r}) \cdot \hat{n} = 0$$

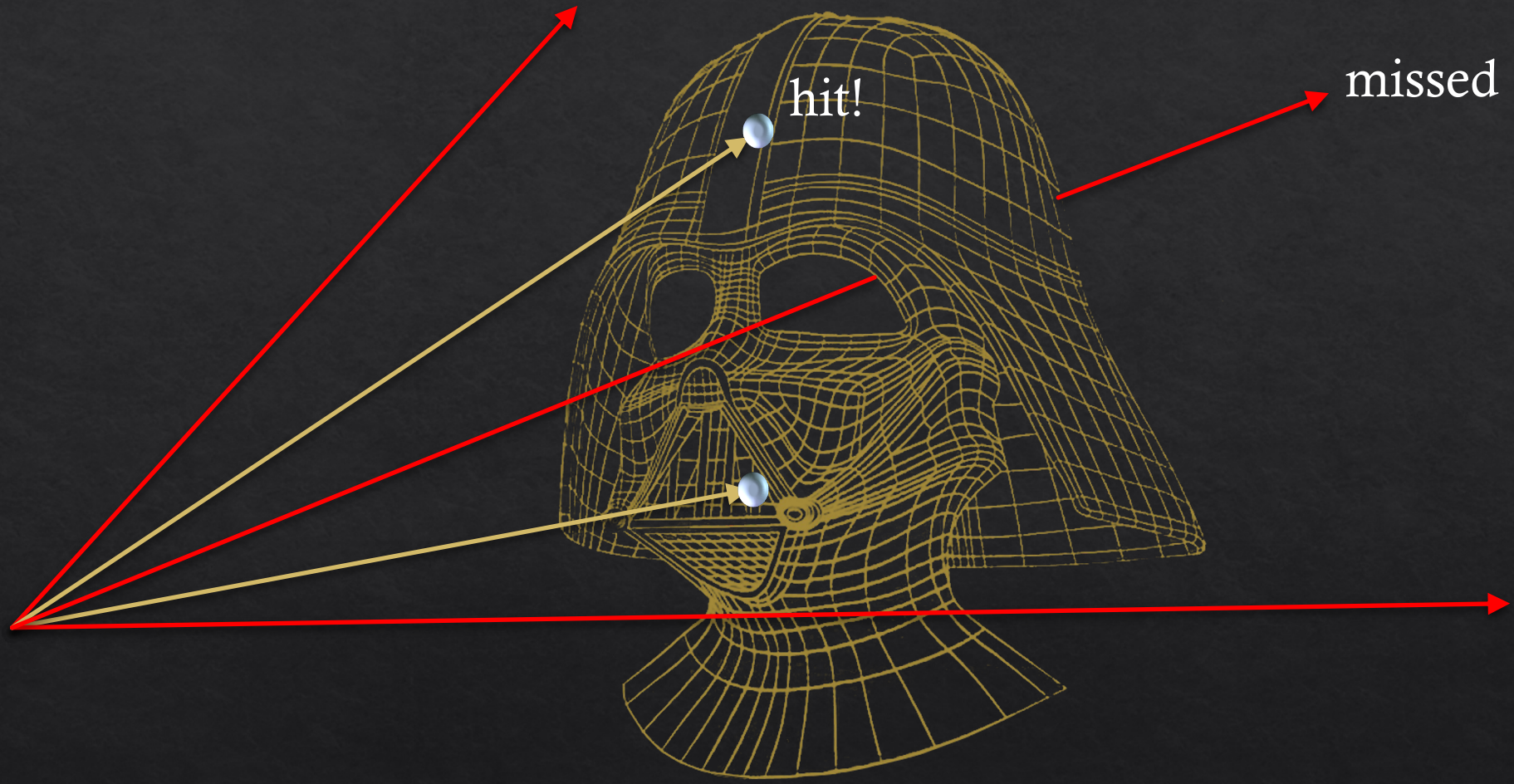
$$(\vec{q} - \vec{p}) \cdot \hat{n} - t\hat{r} \cdot \hat{n} = 0$$

$$t\hat{r} \cdot \hat{n} = (\vec{q} - \vec{p}) \cdot \hat{n}$$

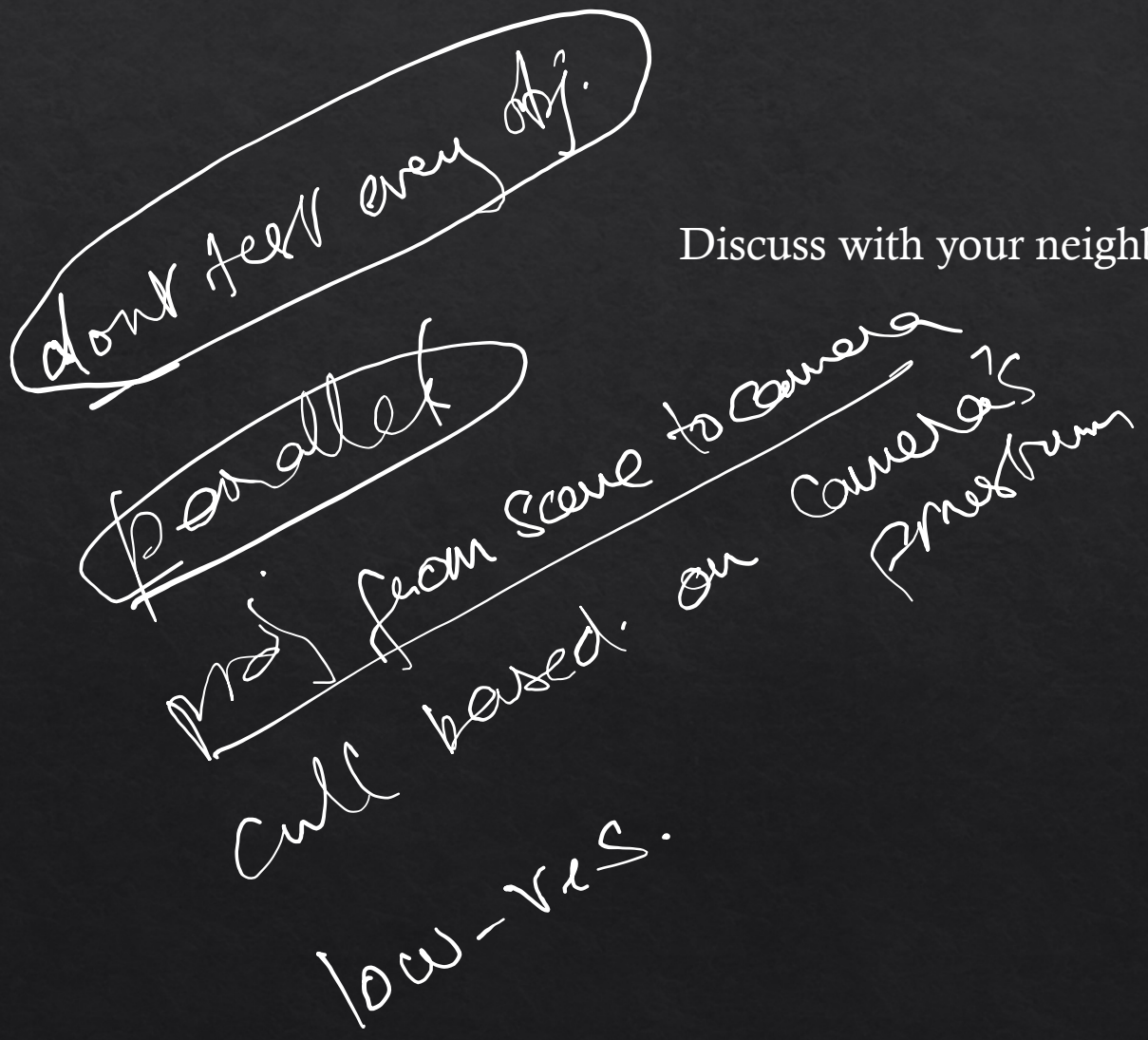
$$t = \frac{(\vec{q} - \vec{p}) \cdot \hat{n}}{\hat{r} \cdot \hat{n}}$$



Computation: test many objects



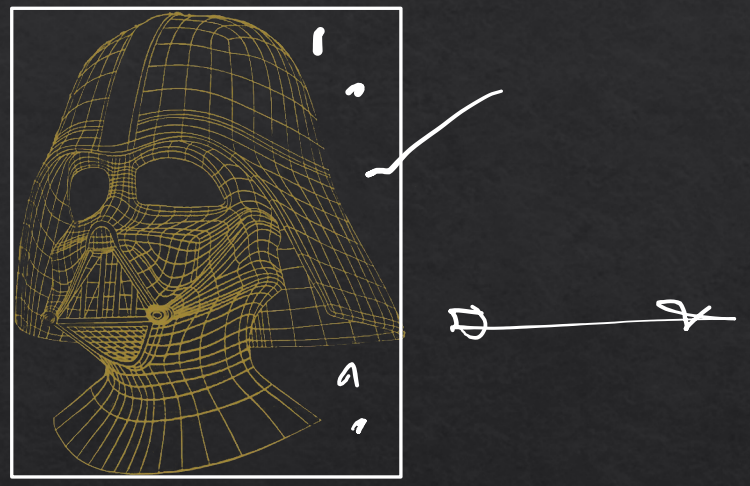
Speed up testing many objects?



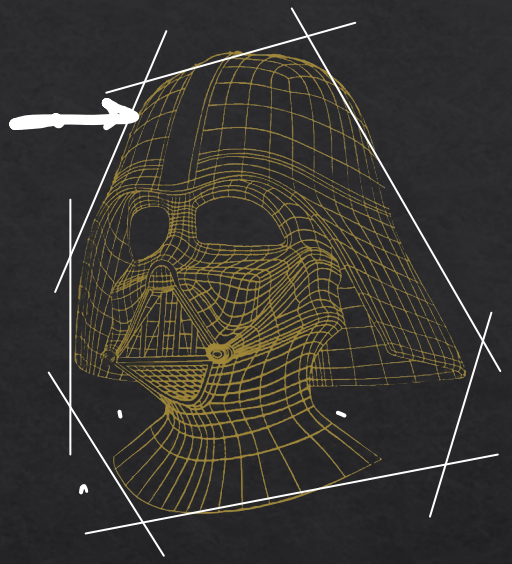
Discuss with your neighbour(s) and propose strategies

Acceleration data structures

grids
- uniform
- adaptive e.g. octree

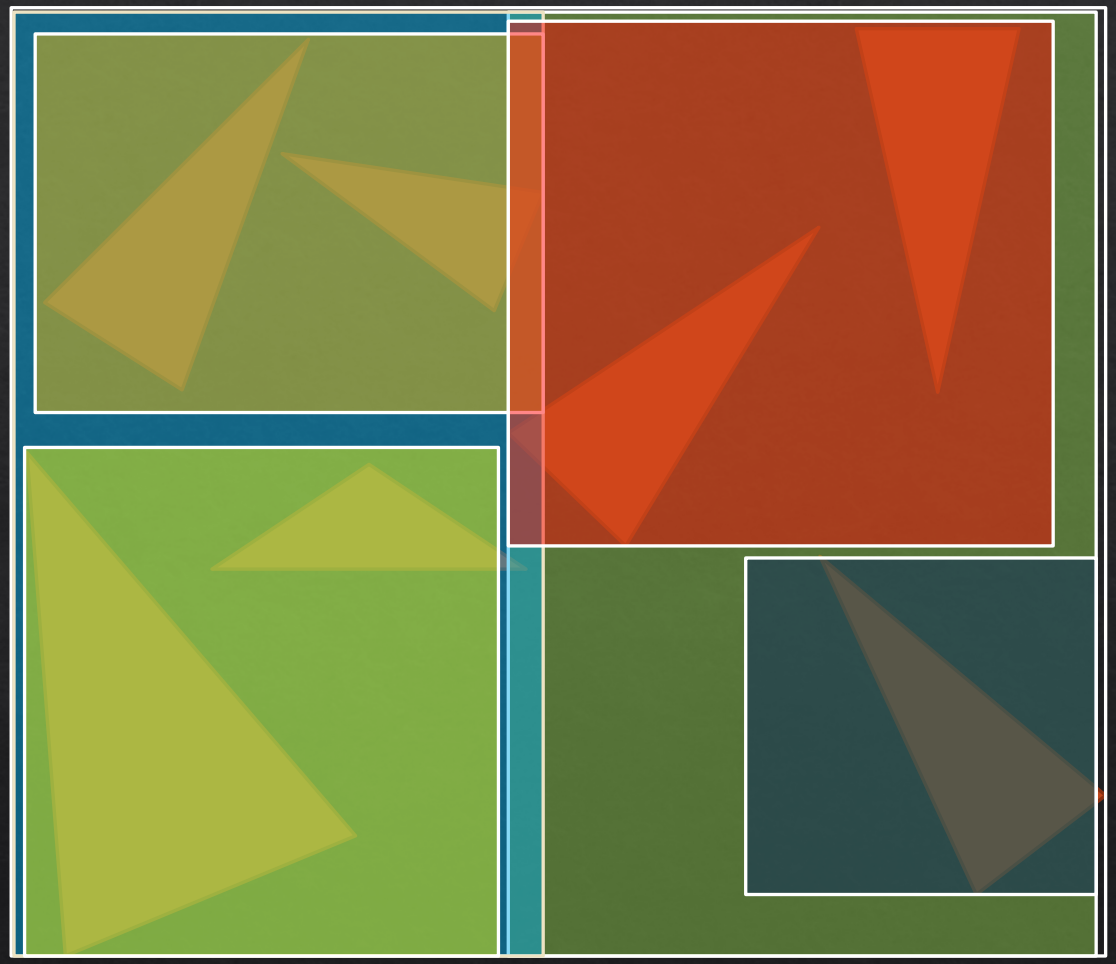


bounding box
- simple
- concave objects?

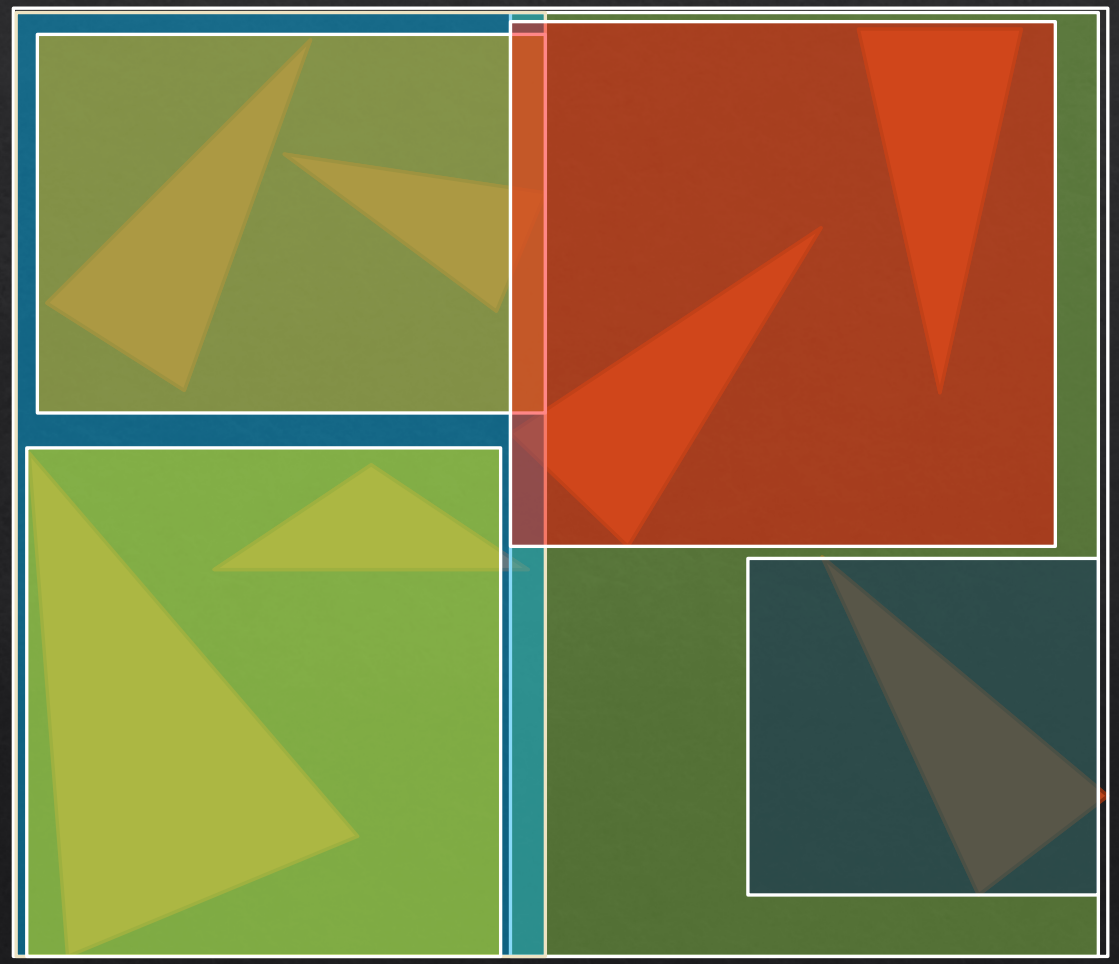


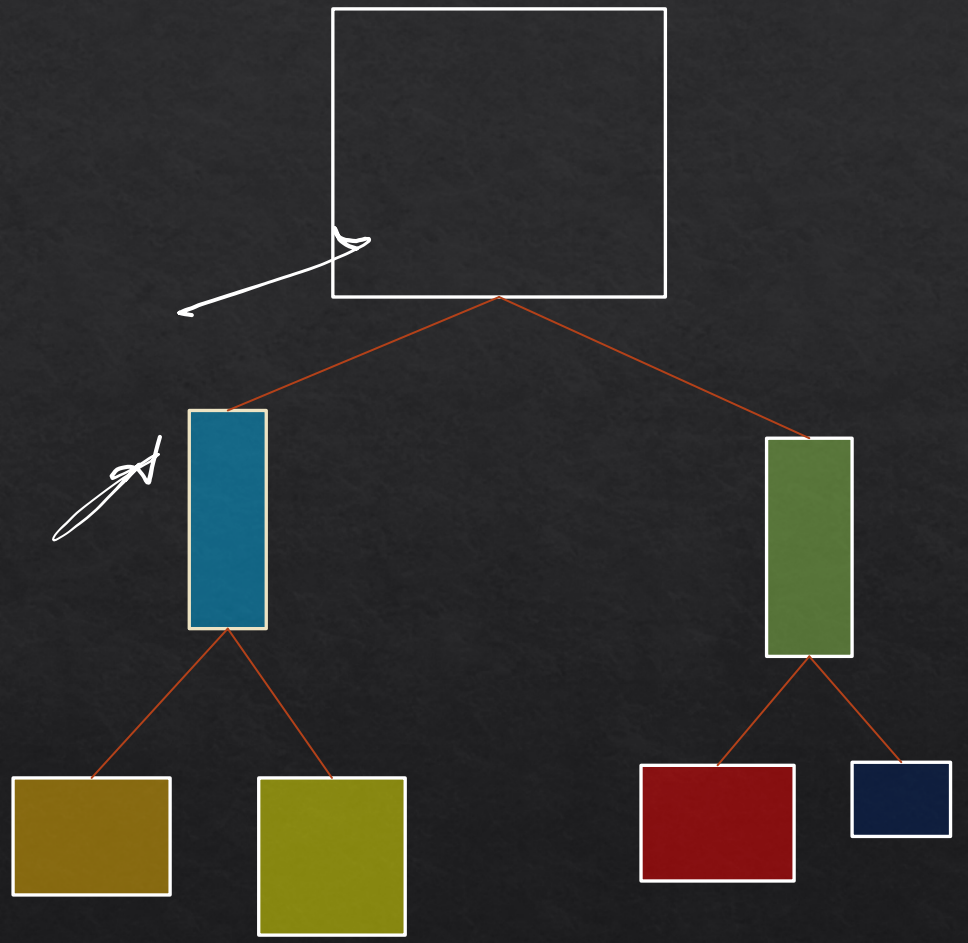
arbitrary bounding volume
- ‘tight’ but more tests
- concave objects?

Acceleration technique

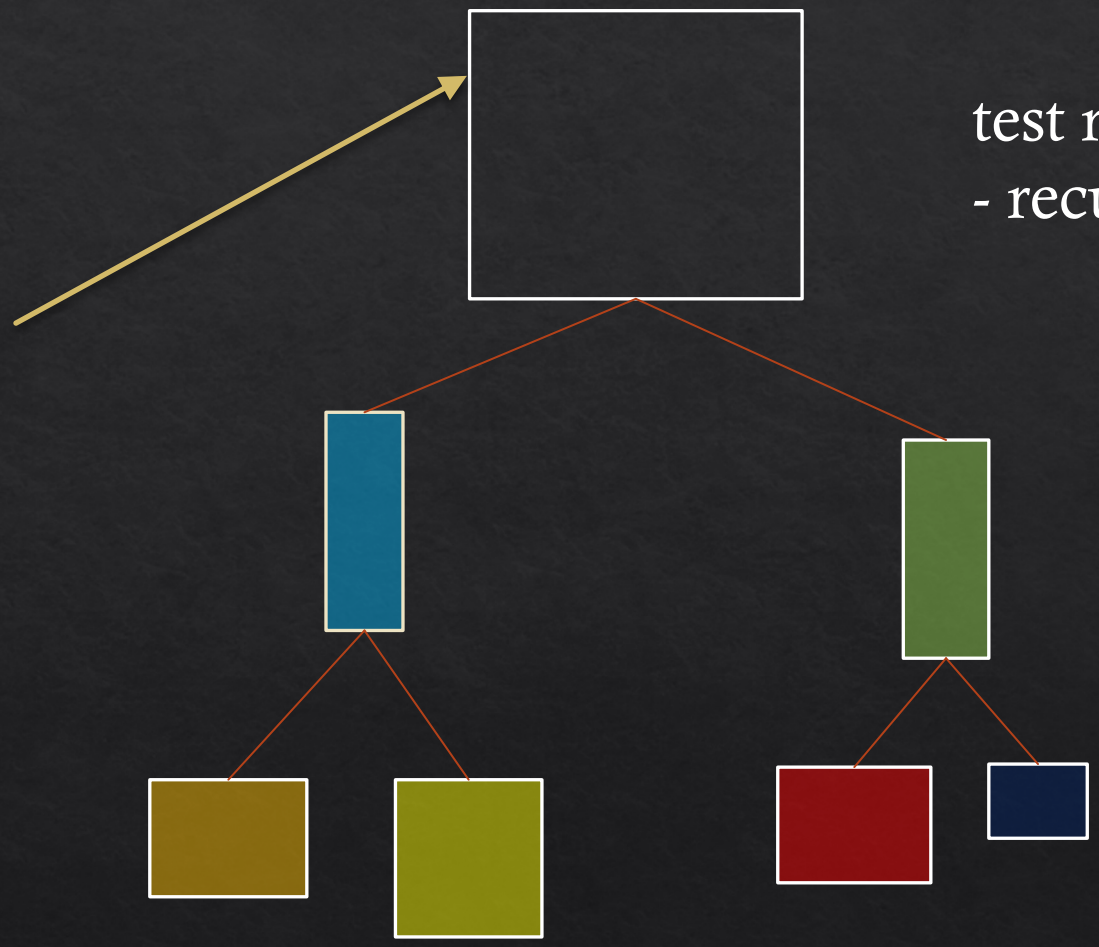


Acceleration: bounding volume hierarchy





Acceleration: bounding volume hierarchy

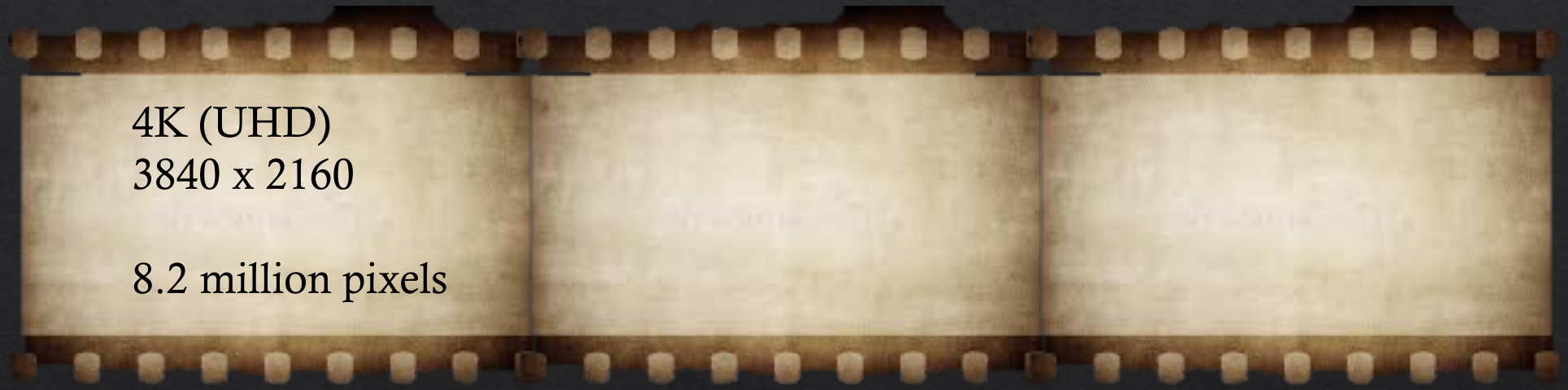


test ray at highest level
- recurse based on intersection result

Interested readers: slides

Video games: 0.5 billion rays per second!

60 frames per second



Ray intersection - GPU

Rendering / Ray Tracing

English

Flexible and Powerful Ray Tracing with NVIDIA OptiX 8

Aug 07, 2023

+8 Like Discuss (1)

By Zach Lo

In the realm of computer graphics, achieving photorealistic visuals has been a long-sought goal. NVIDIA OptiX is a powerful and flexible ray-tracing framework, enabling you to harness the potential of ray tracing. NVIDIA OptiX is a GPU-accelerated, ray-casting API based on the CUDA parallel programming model. It gives you all the tools required to implement ray tracing, enabling you to define and execute complex ray tracing algorithms efficiently on NVIDIA GPUs. Used with a graphics API like OpenGL or DirectX, NVIDIA OptiX permits you to create a renderer that enables faster and more cost-effective product development cycles.

https://developer.nvidia.com/blog/flexible-and-powerful-ray-tracing-with-optix-8/

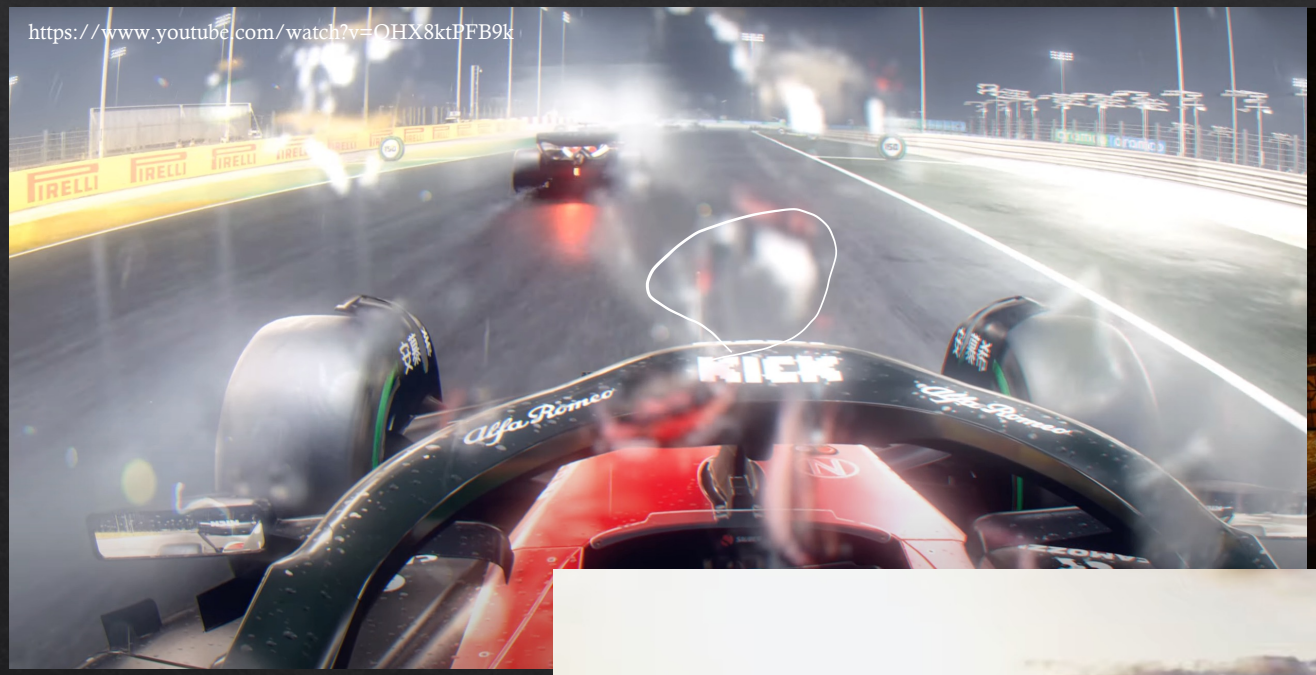
Ray intersection - GPU

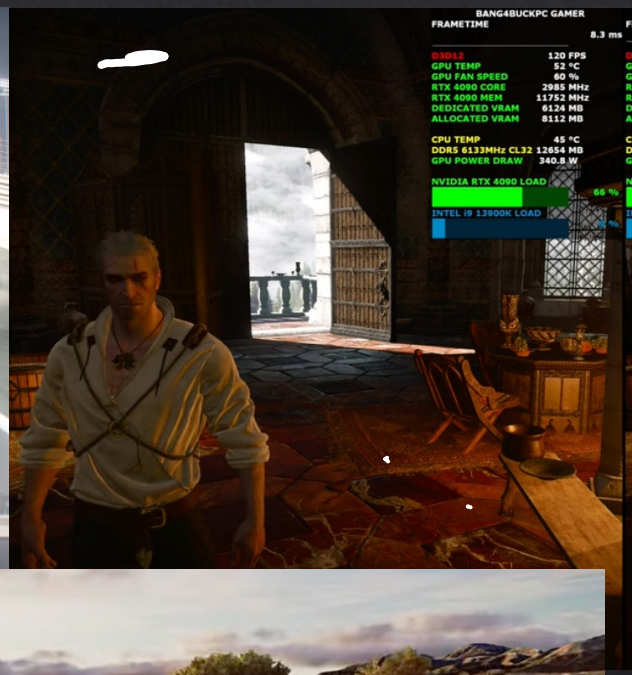
- Reduce computation time by executing parallel tests

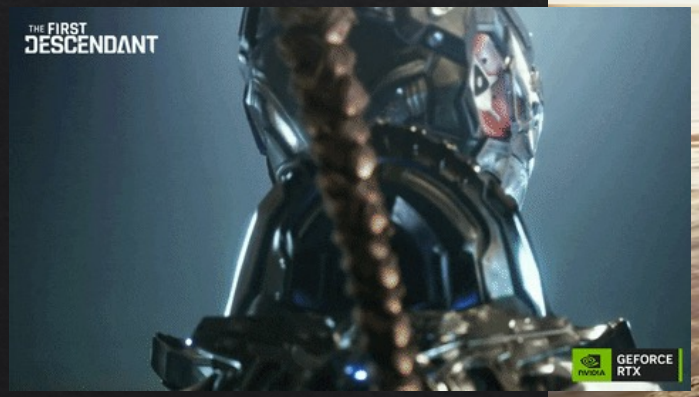
- Reduce number of tests via acceleration structures
  - trade off: building data structures and book-keeping
  - exploring data structures may not be easy or efficient

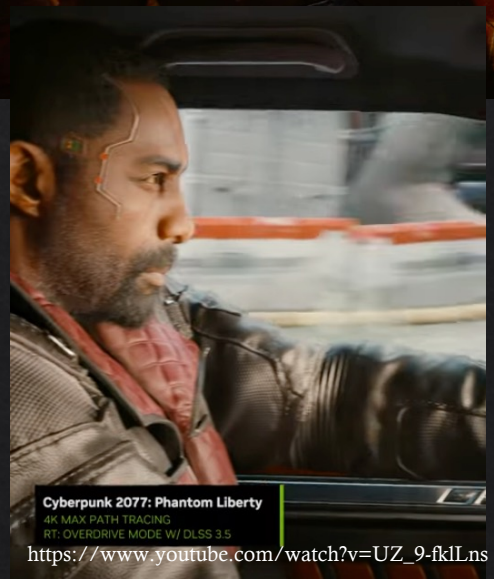
NVIDIA 4090

~£1800, > 10b rays/s

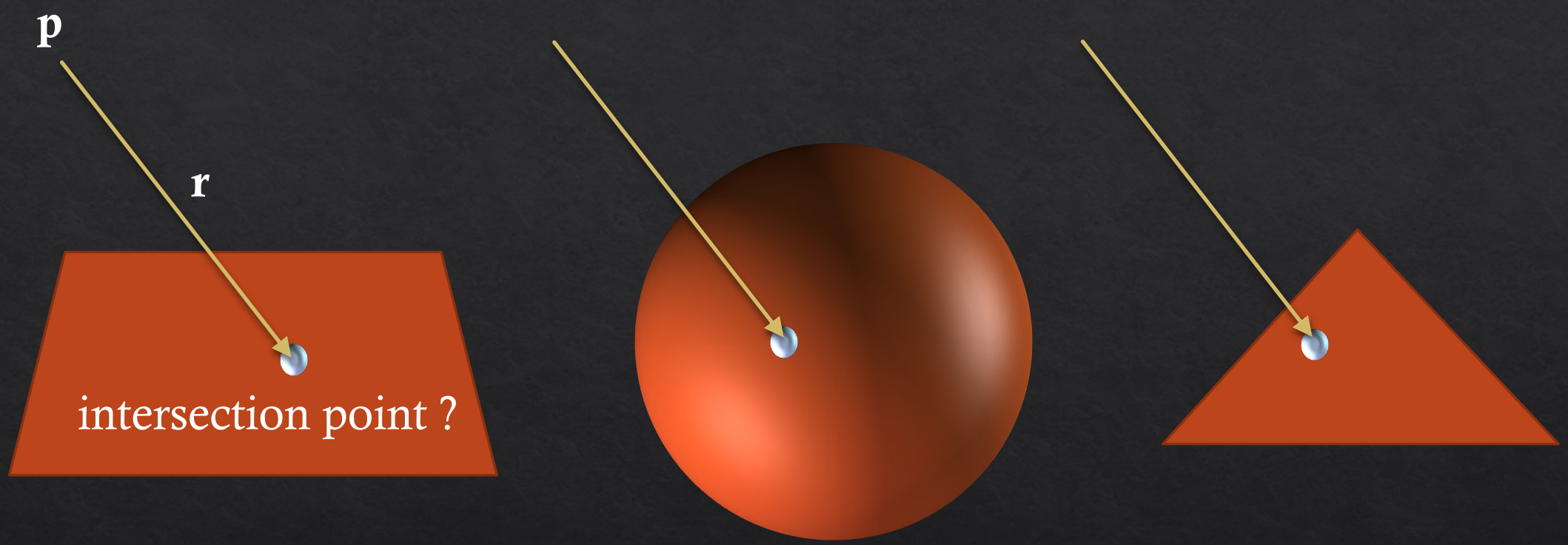




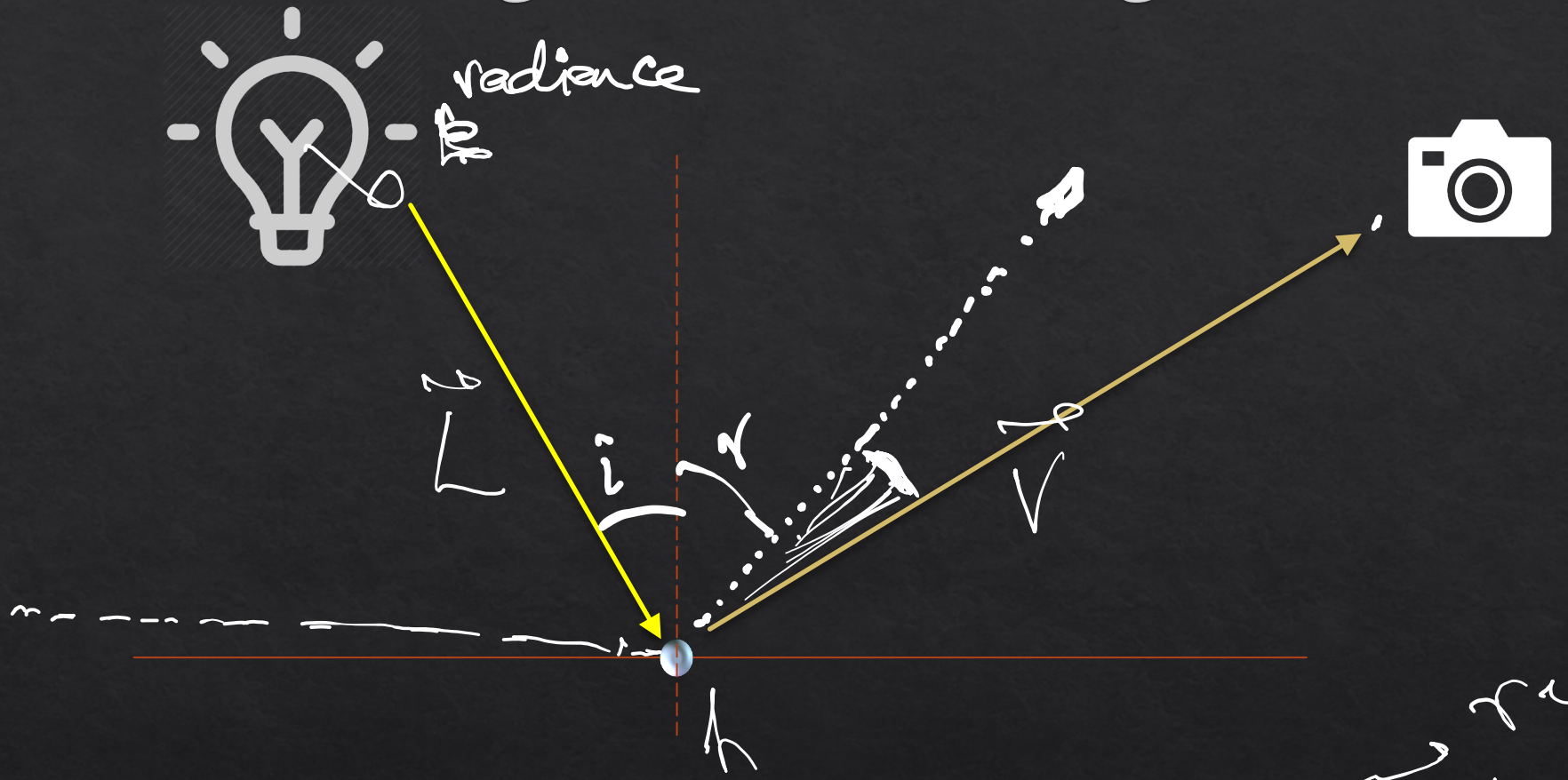




Ray intersection

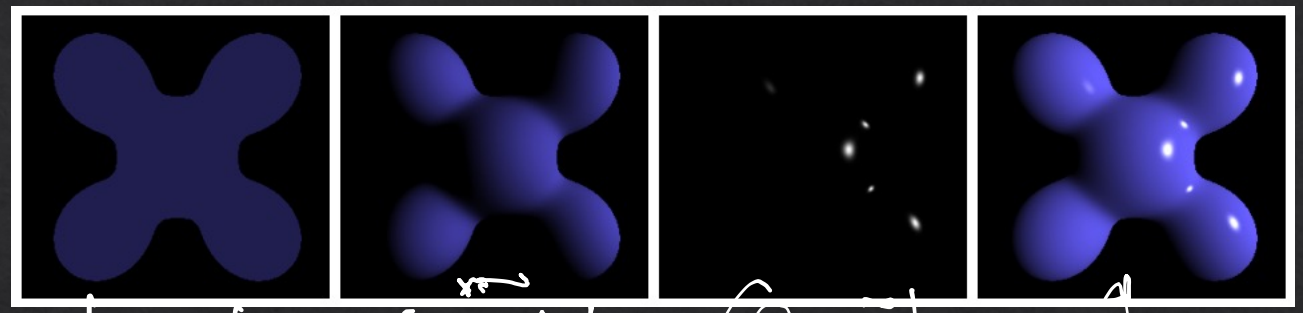


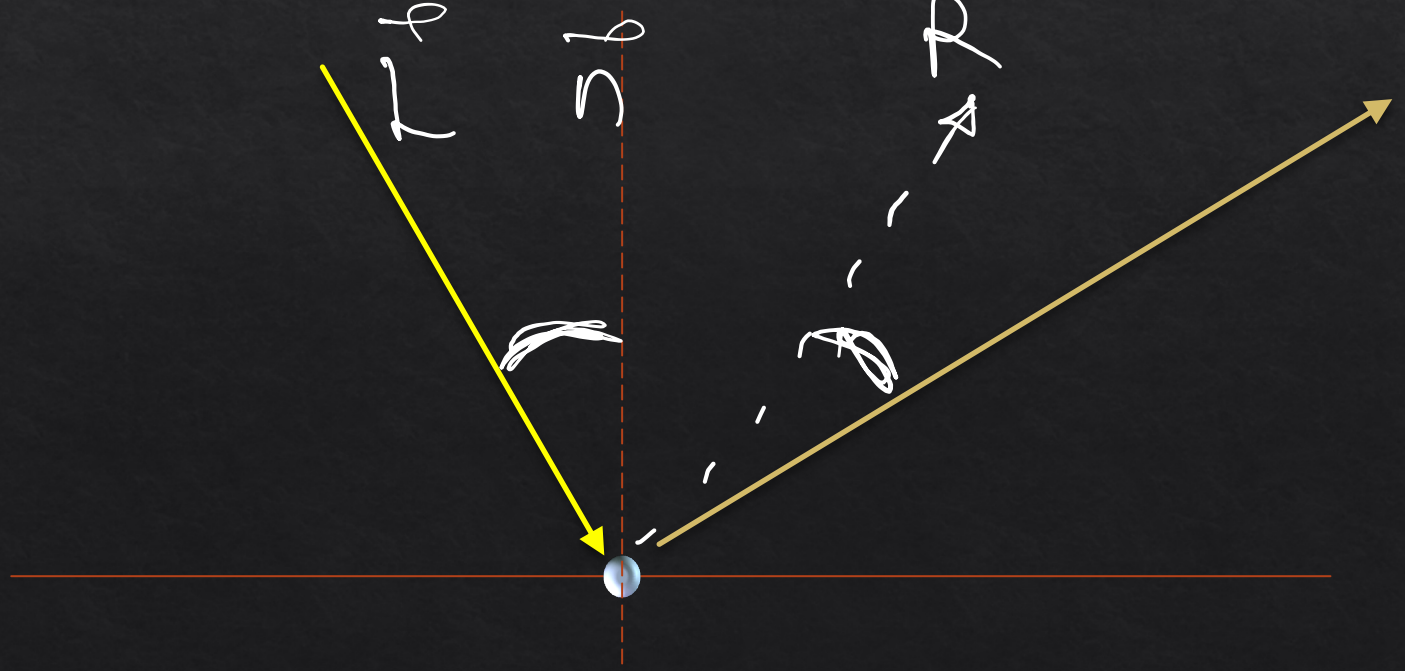
Shading: How much light?



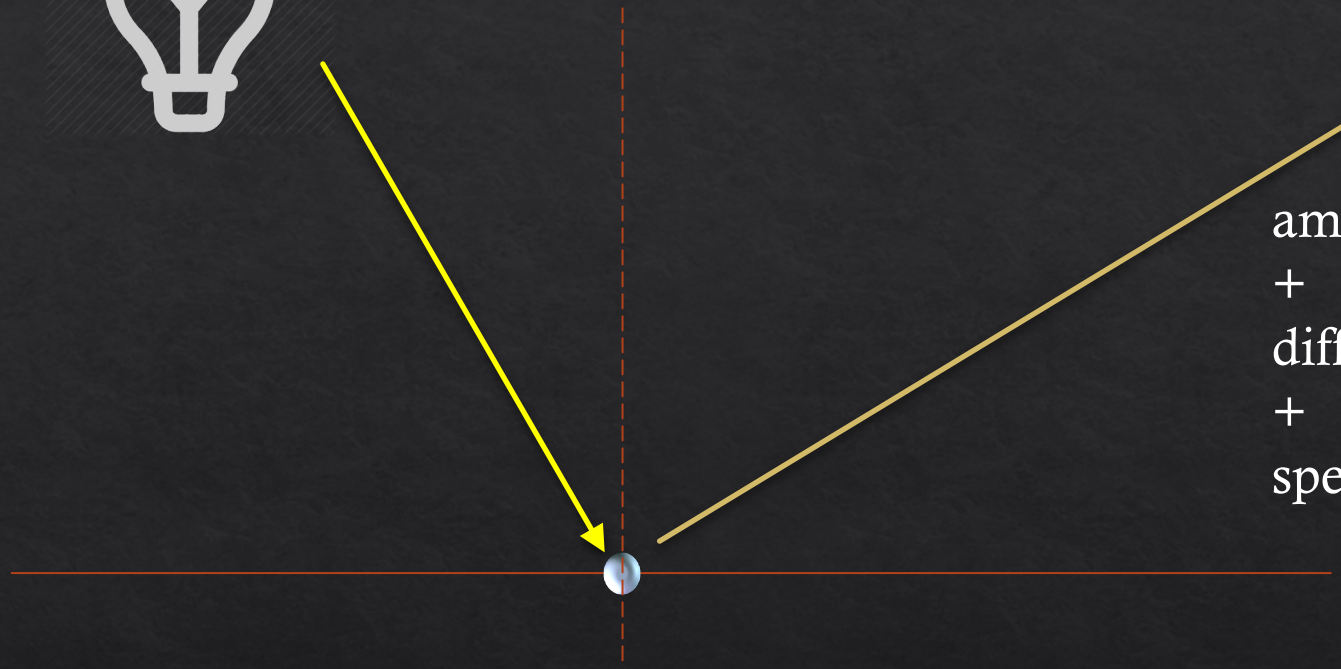
refl. prop
material
colour
texture

Approximation!





Shading: approximation



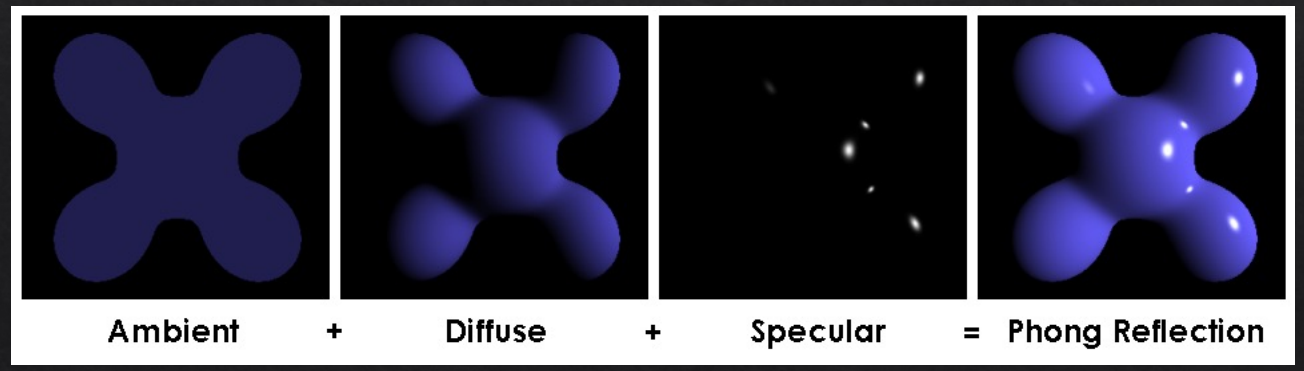
ambient light (constant)
+
diffuse
+
specular reflection of light

Phong shading

material property

$$I_\mathrm{p} = k_\mathrm{a} i_\mathrm{a} + \sum_{m \in \text{lights}} \left(k_\mathrm{d} (\hat{L}_m \cdot \hat{N}) i_{m,\mathrm{d}} + k_\mathrm{s} (\hat{R}_m \cdot \hat{V})^\alpha i_{m,\mathrm{s}} \right)$$

ambient | diffuse | specular

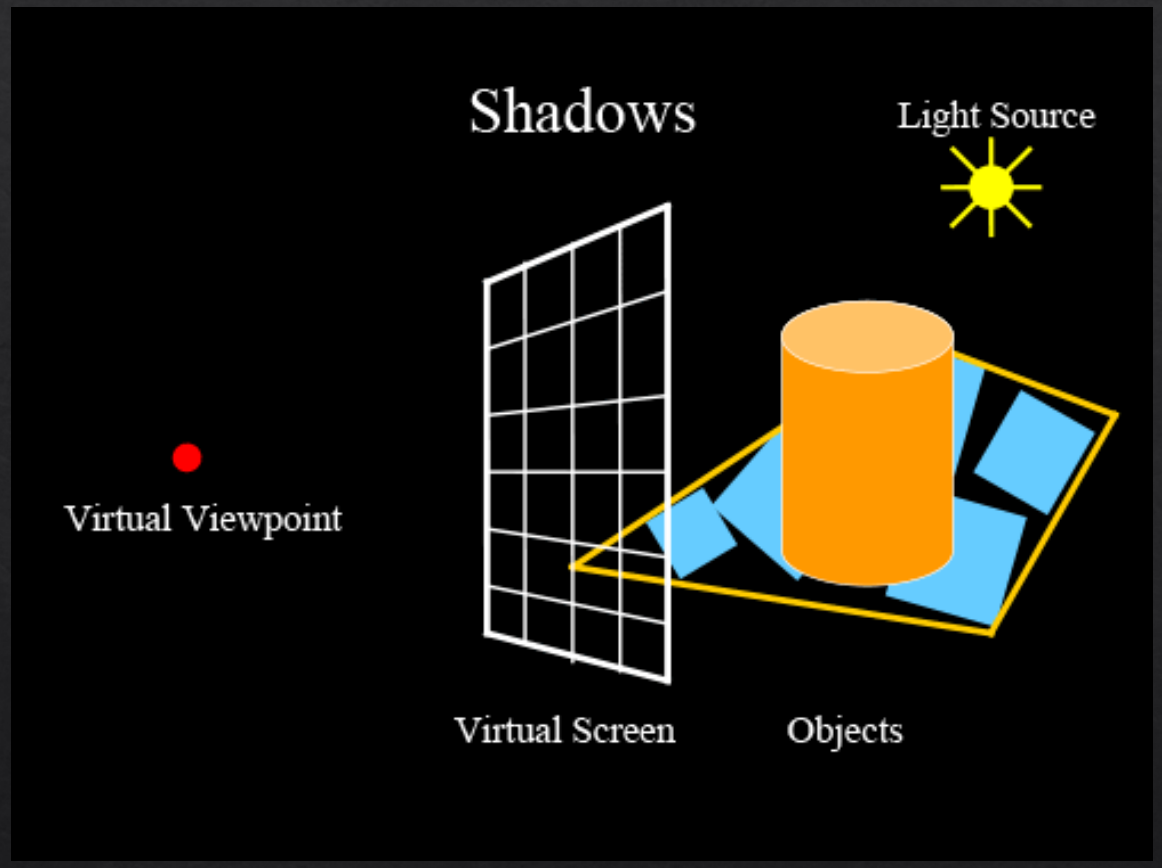


source:wikipedia

light property

R is the mirror reflection direction

Shadows



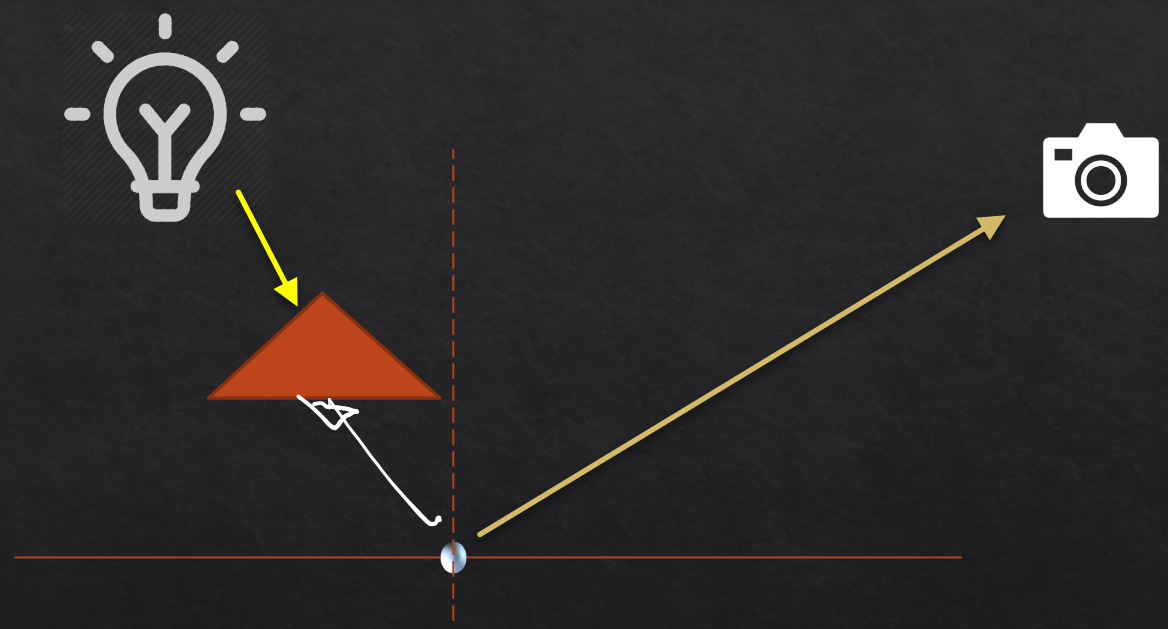
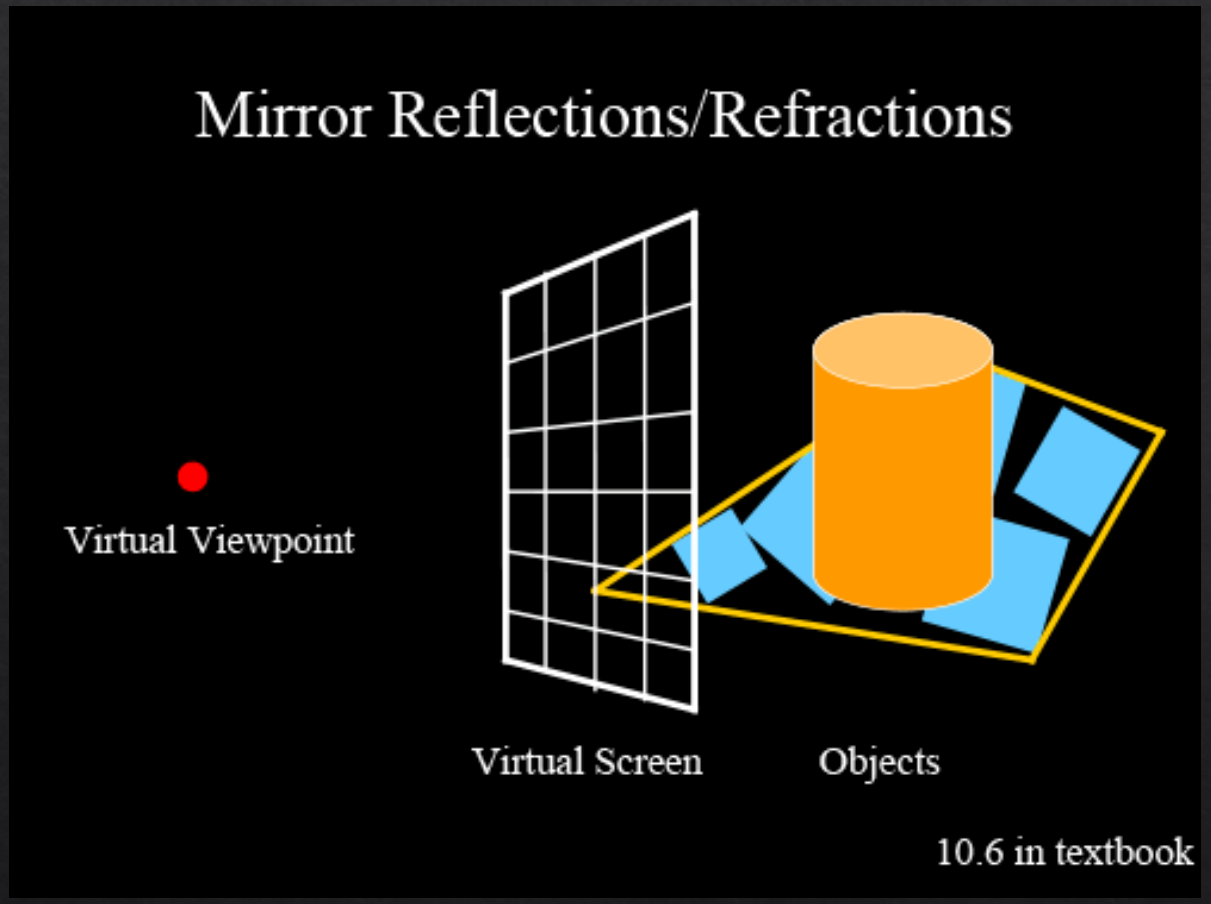


image credit http://studiomaven.org/Context_Overview_of_Ray-Traced_Rendering.html

Mirror (specular) reflection/refraction



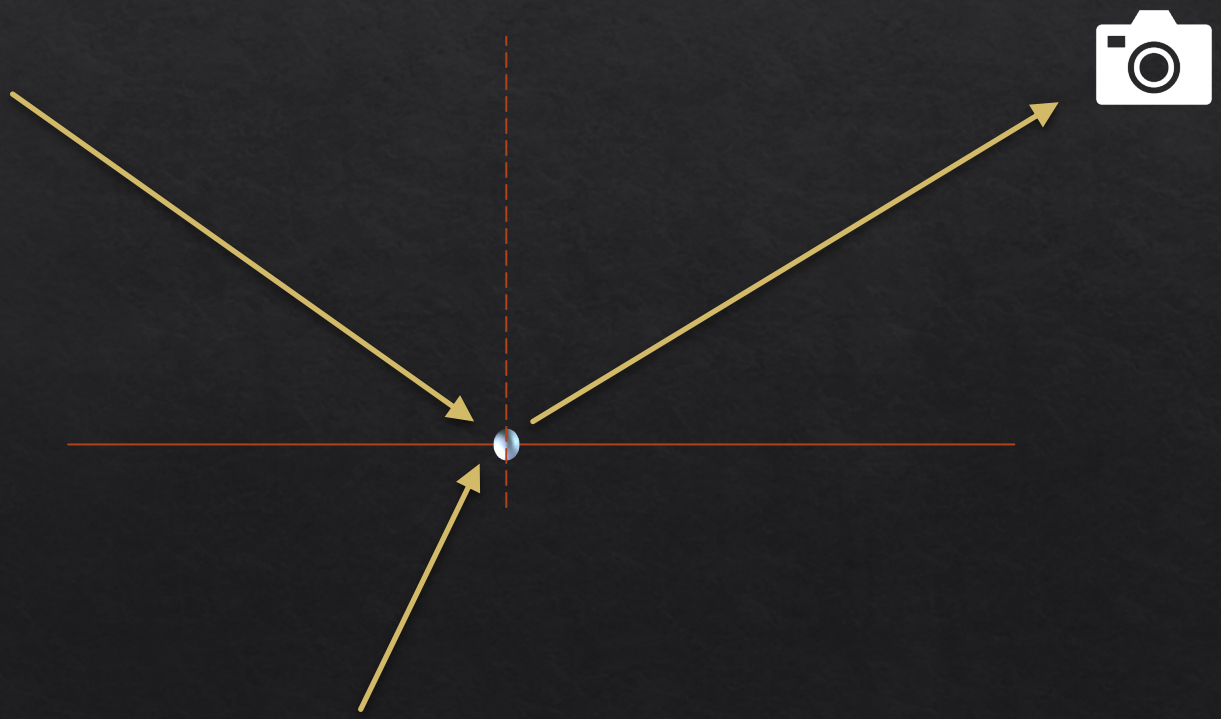
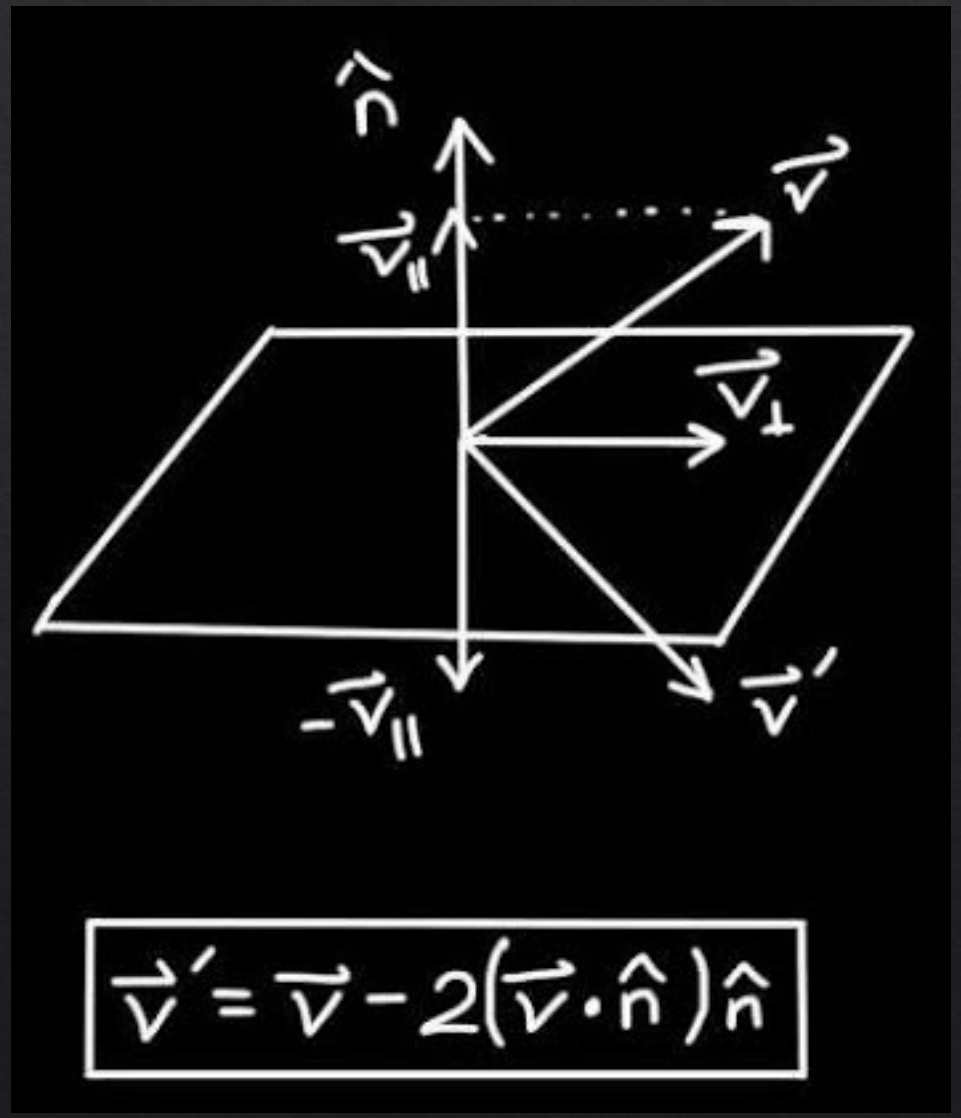


image credit http://studiomaven.org/Context__Overview_of_Ray-Traced_Rendering.html

Reflection


$\hat{n}$
$\vec{v}$
$\vec{v}_{\parallel}$
$\vec{v}_{\perp}$
$-\vec{v}_{\parallel}$
$\vec{v}'$
$\vec{v}' = \vec{v} - 2(\vec{v} \cdot \hat{n})\hat{n}$

Resources

https://www.cs.utah.edu/~shirley/books/fcg2/rt.pdf

http://www.cs.utah.edu/~shirley/papers/rayedu.pdf

http://www.realtimerendering.com/raytracing/roundup.html

http://rtintro.realtimerendering.com/

https://benedikt-bitterli.me/tantalum/tantalum.html