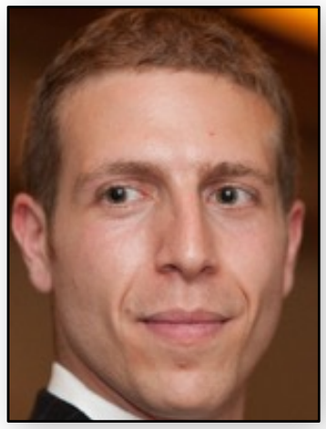
ECSE 446/546

Realistic & Advanced Image Synthesis

Prof. Derek Nowrouzezahrai

derek@cim.mcgill.ca

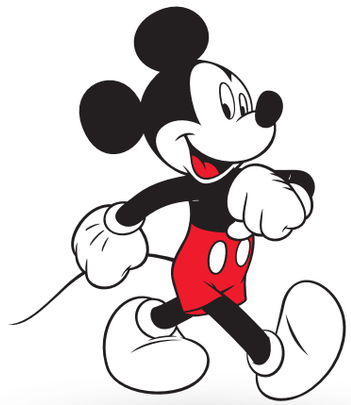
About me



Derek Nowrouzezahrai

Associate Professor

derek@cim.mcgill.ca



- "Those who can't do, teach."
- Previously worked films, games, parks, and products at:
  - Disney Research, Electronic Arts, Microsoft & Microsoft Research
- McGill's Graphics & Imaging Lab
- NSERC/Ubisoft Industrial Research Chair

Course Details

Lecture Time and Location

- Wednesdays & Fridays
    - 2:35pm – 3:55pm in Trottier 1100

Tutorials:

- Mondays, Wednesdays, Fridays
    - 9:35pm – 11:25pm in Trottier 4160

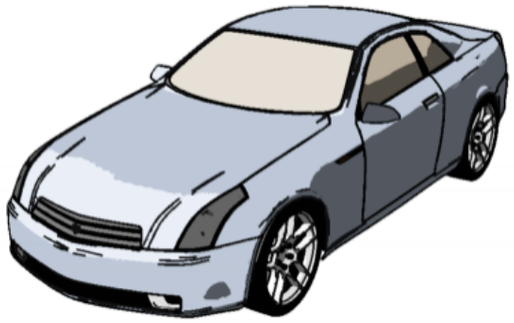
Website

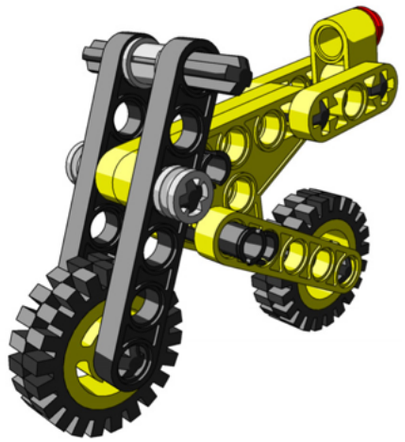
- http://www.cim.mcgill.ca/~derek/ecse446.html
- Will also use **myCourses** for discussion, slides and assignment hand-in

Rendering

The process of converting a description of a 3D scene into an image

- **Non-Photorealistic Rendering (NPR)**





Rendering

The process of converting a description of a 3D scene into an image

- **Photorealistic Rendering**: create an image of a 3D scene that is indistinguishable from a photo
- **Physically-based Rendering**: *simulate* the physical behaviour of light as closely as possible to *predict* what would be observed

www.pbrt.org/gallery

Curriculum

Graphics & Imaging

- ECSE532: Computer Graphics
- COMP557: Computer Graphics
- ECSE446/546: Image Synthesis
- COMP559: Computer Animation
- ECSE544: Computational Photography
- COMP521: Modern Computer Games

Vision/Imaging

- ECSE529: Computer & Biological Vision
- ECSE626: Statistical Computer Vision
- COMP558: Computer Vision

- Probability & Statistical Methods
- Applied Numerical Methods

Relation to curriculum

ECSE532: Computer Graphics

COMP557: Computer Graphics

Modeling

How to build a digital representation of a 3D scene?

Animation

How to bring a 3D scene to life?

Rendering

How can we generate an image from a digital representation of a 3D scene?

Relation to curriculum

ECSE532: Computer Graphics

COMP557: Computer Graphics

Modeling

How to build a digital representation of a 3D scene?

Animation

How to bring a 3D scene to life?

- Physics-based Motion
- Complex Deformation
- Rigid & Fluid Dynamics

COMP 559

Rendering

How can we generate an image from a digital representation of a 3D scene?

- Physically-based rendering
- Appearance modelling
- Light transport
- Monte Carlo simulation
- Volumetric scattering

ECSE 446/546

Motivating questions

Why do things look the way they do?

How can we generate realistic images?

Motivation

Motivation

Motivation

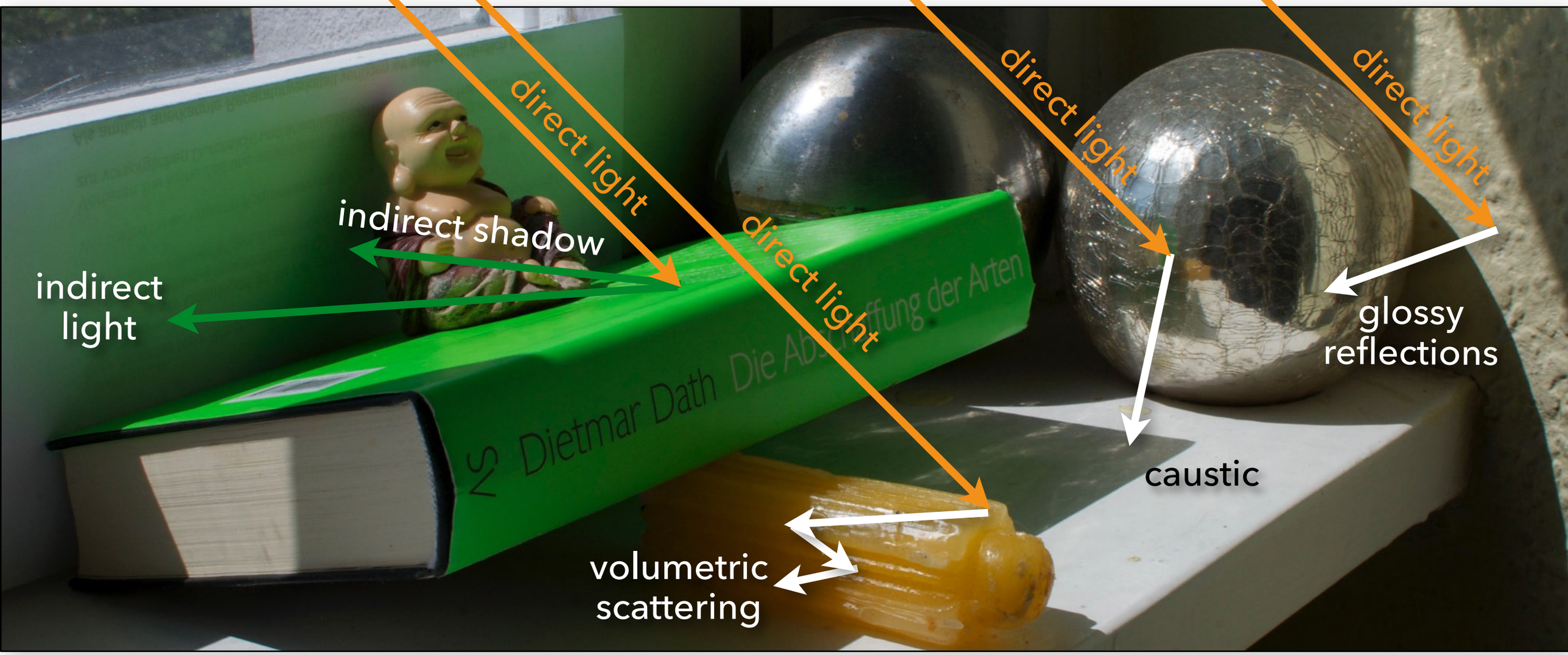
Motivation

Motivation

T.A.Rector (NOAO/AURA/NSF) and the Hubble Heritage Team (STScI/AURA/NASA)

Motivation

Light transport in the real world



After [Ritschel et al. 2011]

Visual effects

Animated films

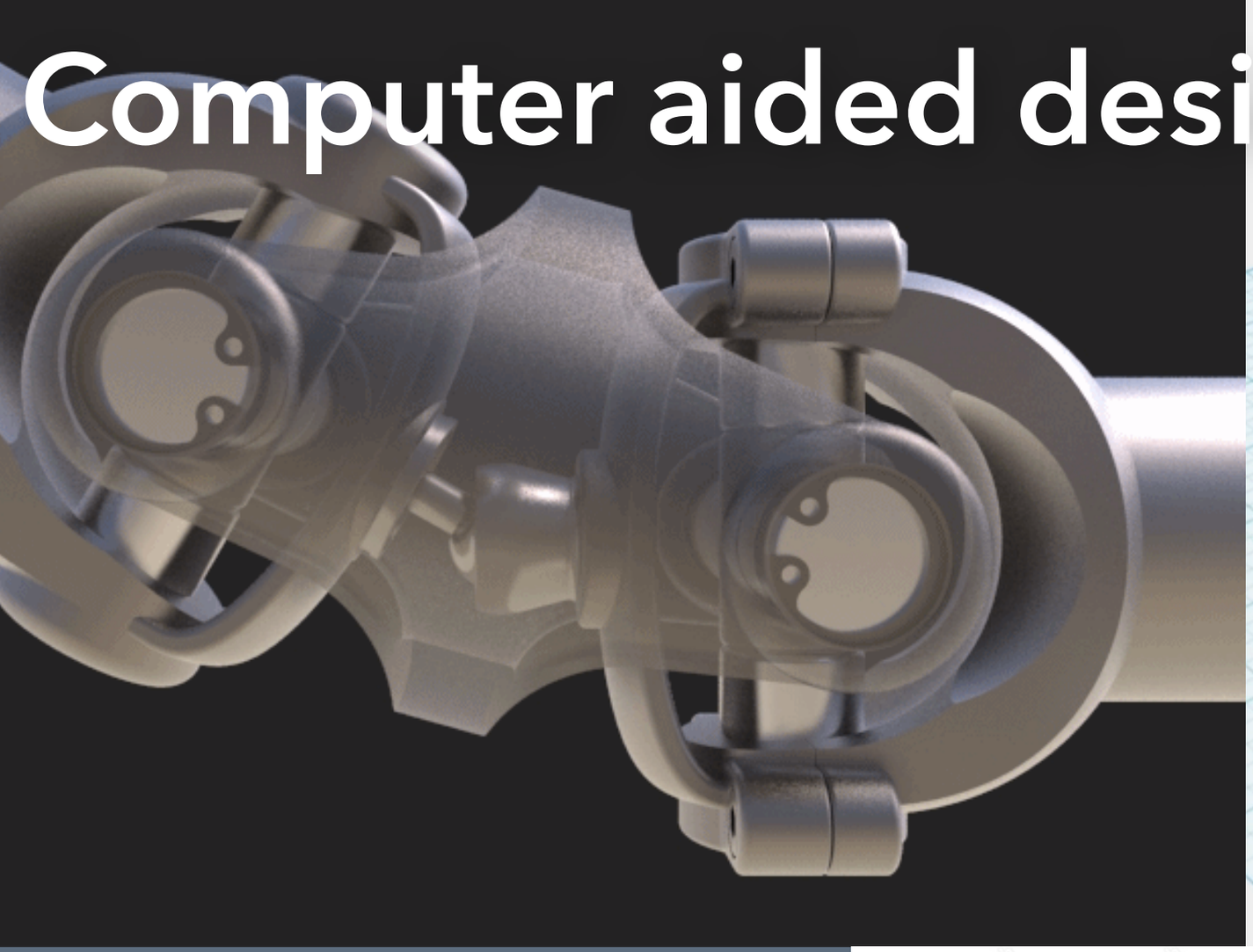
Video games

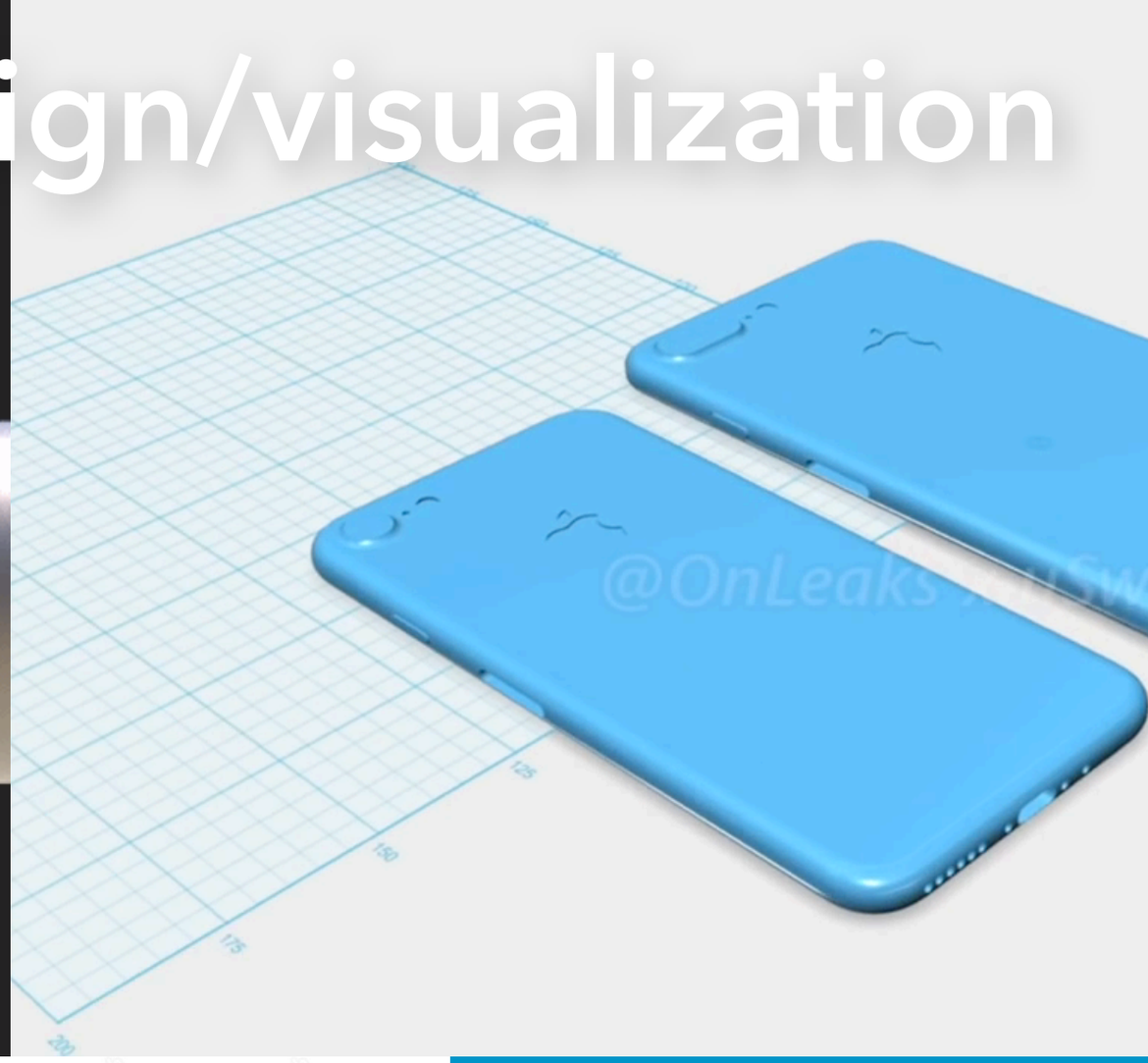
Architectural visualization

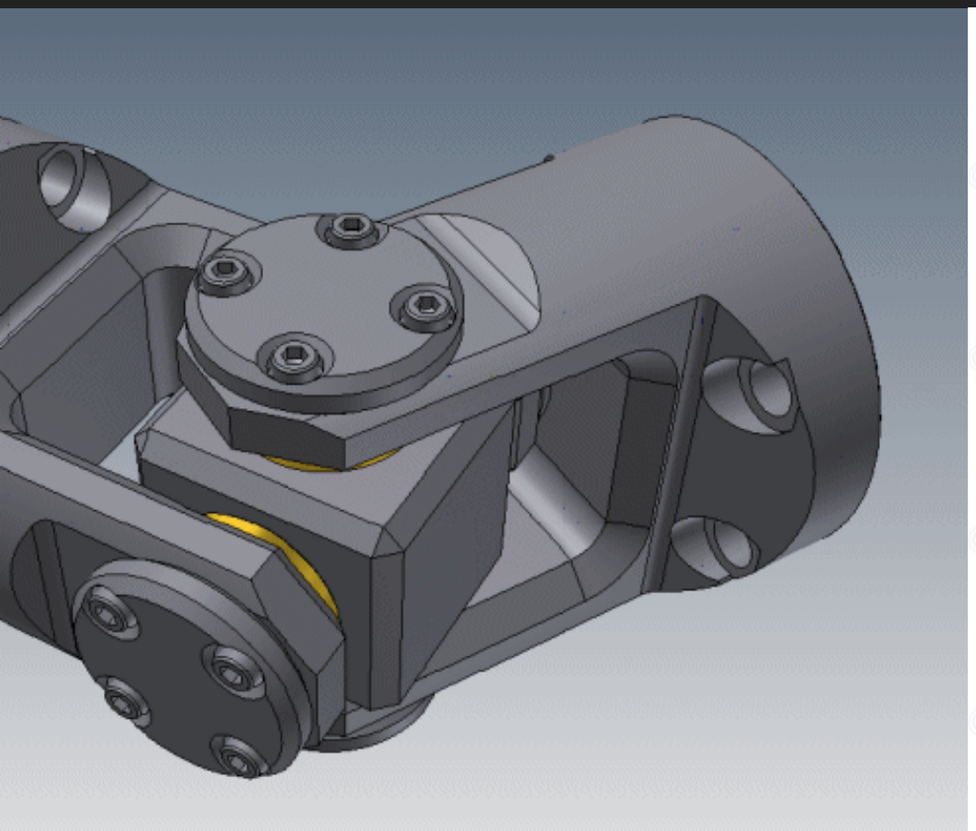
Architectural visualization

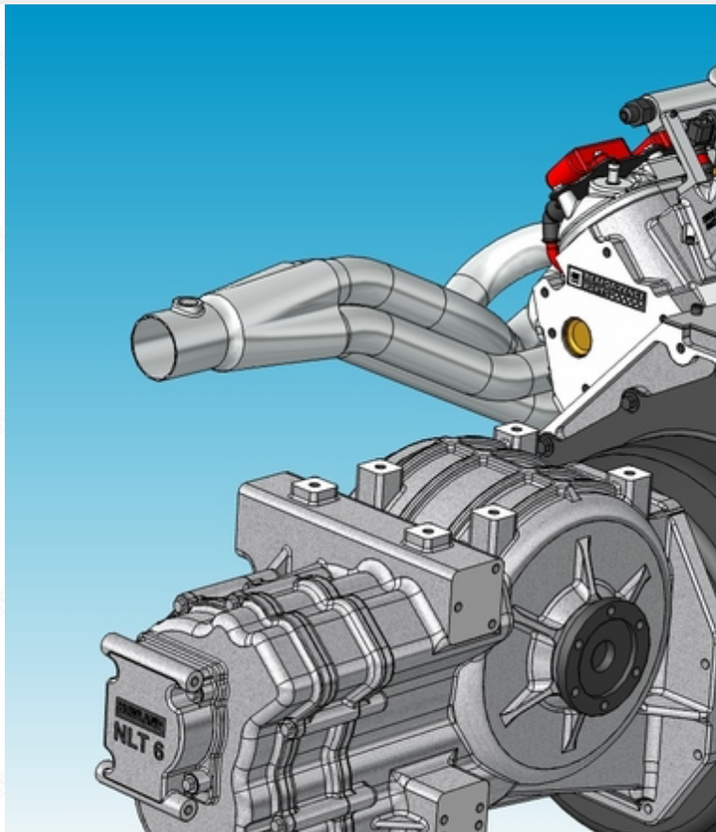
Architectural visualization

Computer aided design/visualization









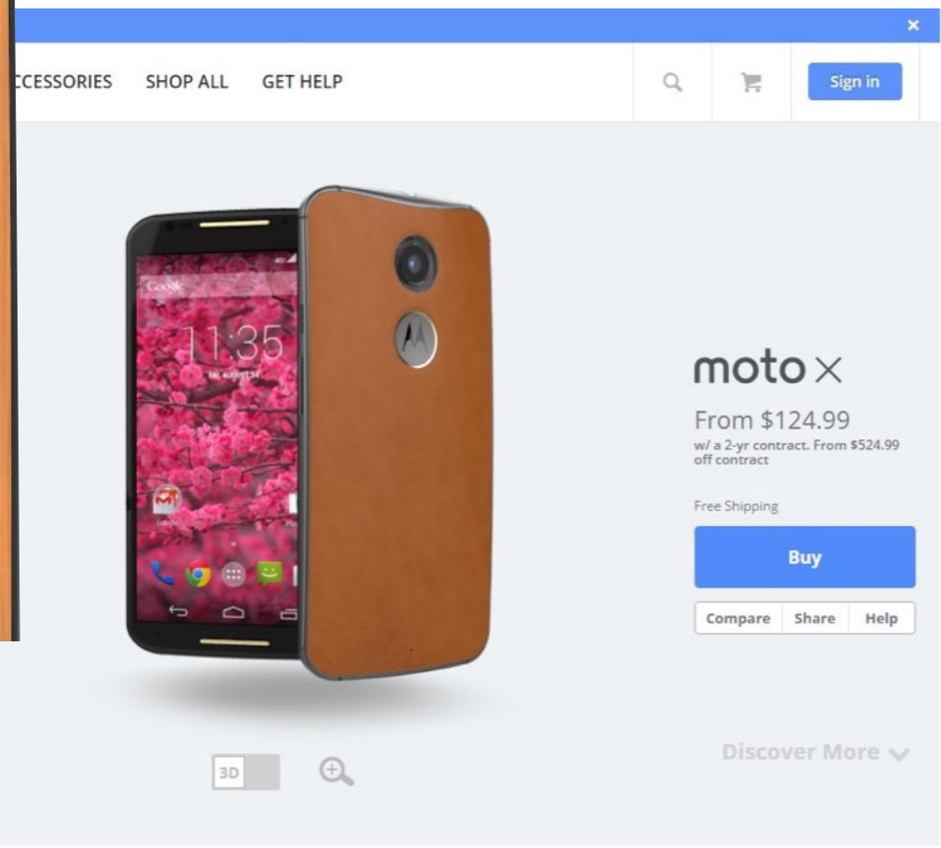
Product visualization

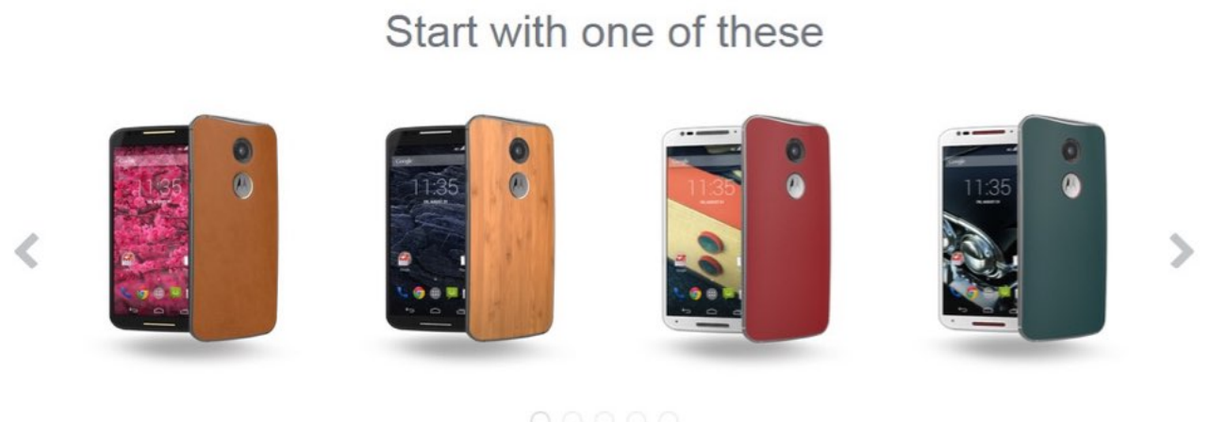
Product visualization

Product visualization

Product visualization

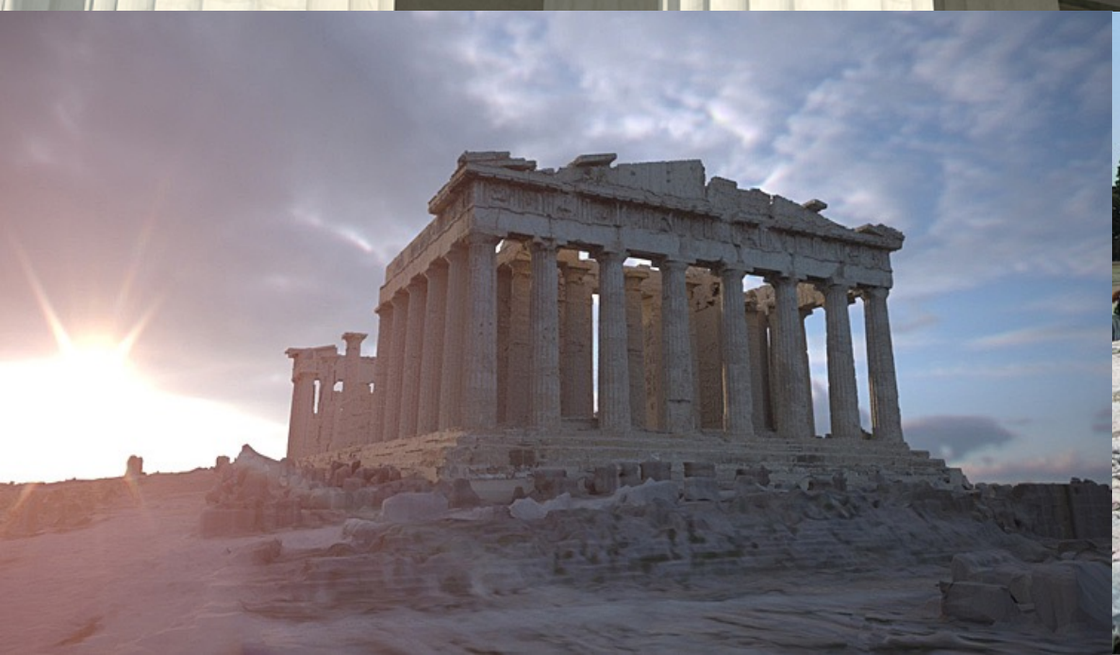
Advertising & E-commerce



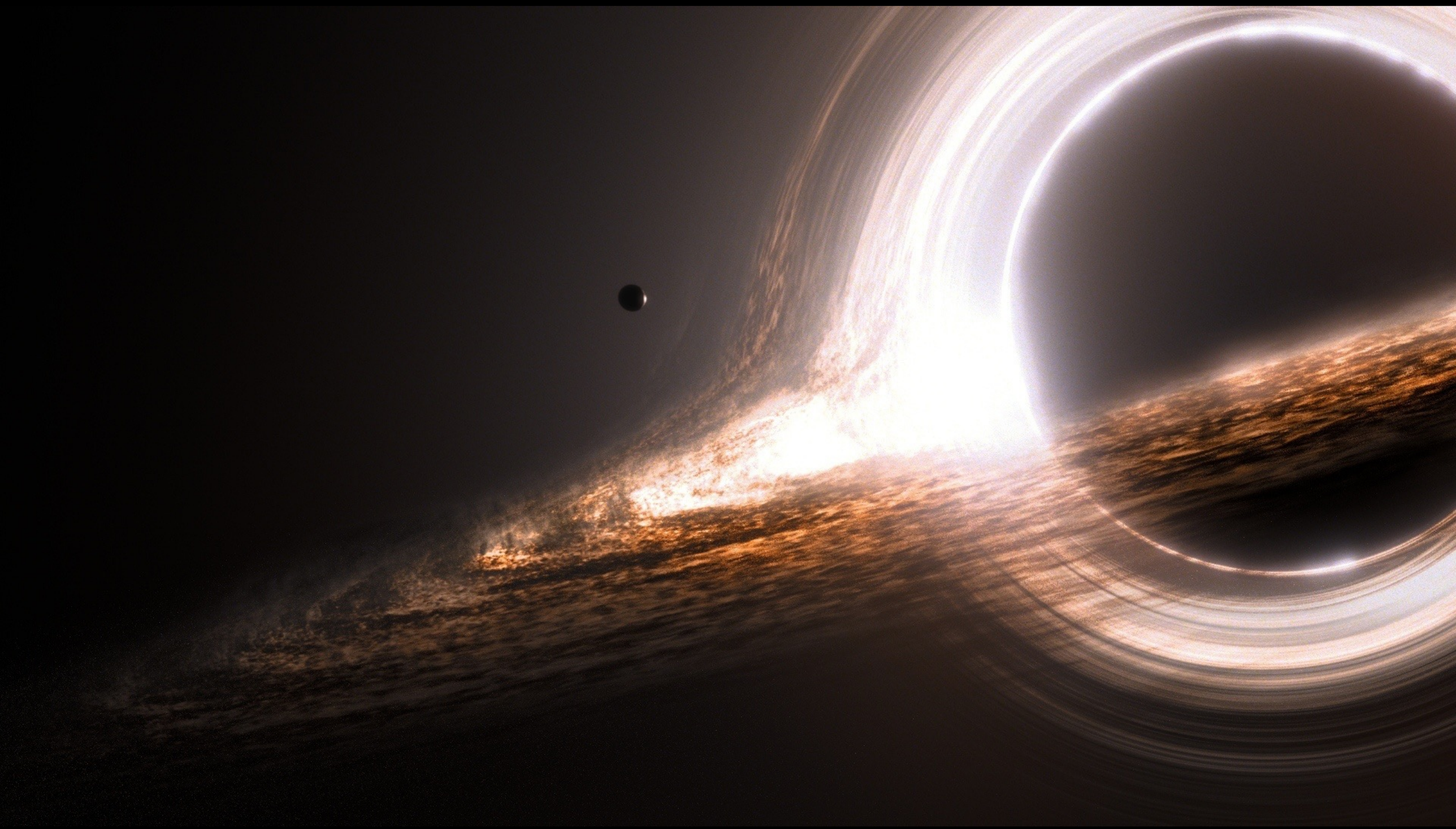


Advertising & E-commerce

Cultural heritage



Scientific visualization (& Visual Effects)



Digital Fabrication

Graphic arts

What is this class about?

Algorithms for realistic image synthesis

Simulating light transport (global illumination)

Modelling appearance (simulating materials)

Understanding why things look the way they do:

- Why is the sky blue?
- Why is the grass green?
- Why does metal look different than marble?

What you should already know

Basic concepts of algorithms and data structures

Calculus

- you will need to compute some derivatives & integrals

Linear algebra

- should know how to perform basic vector/matrix ops.

Programming

Basic Concepts of Computer Graphics*

- basics of 3D spaces + objects*
- basics of ray tracing + shading/lighting concepts*

What you will learn

By the end of course you should:

- understand advanced concepts in rendering, light transport, and appearance modelling
- be able to program both **interactive** and **offline** graphics applications
- have a better sense of:
  - good software engineering practices
  - applied numerical methods and analysis
  - hardware-accelerated graphics

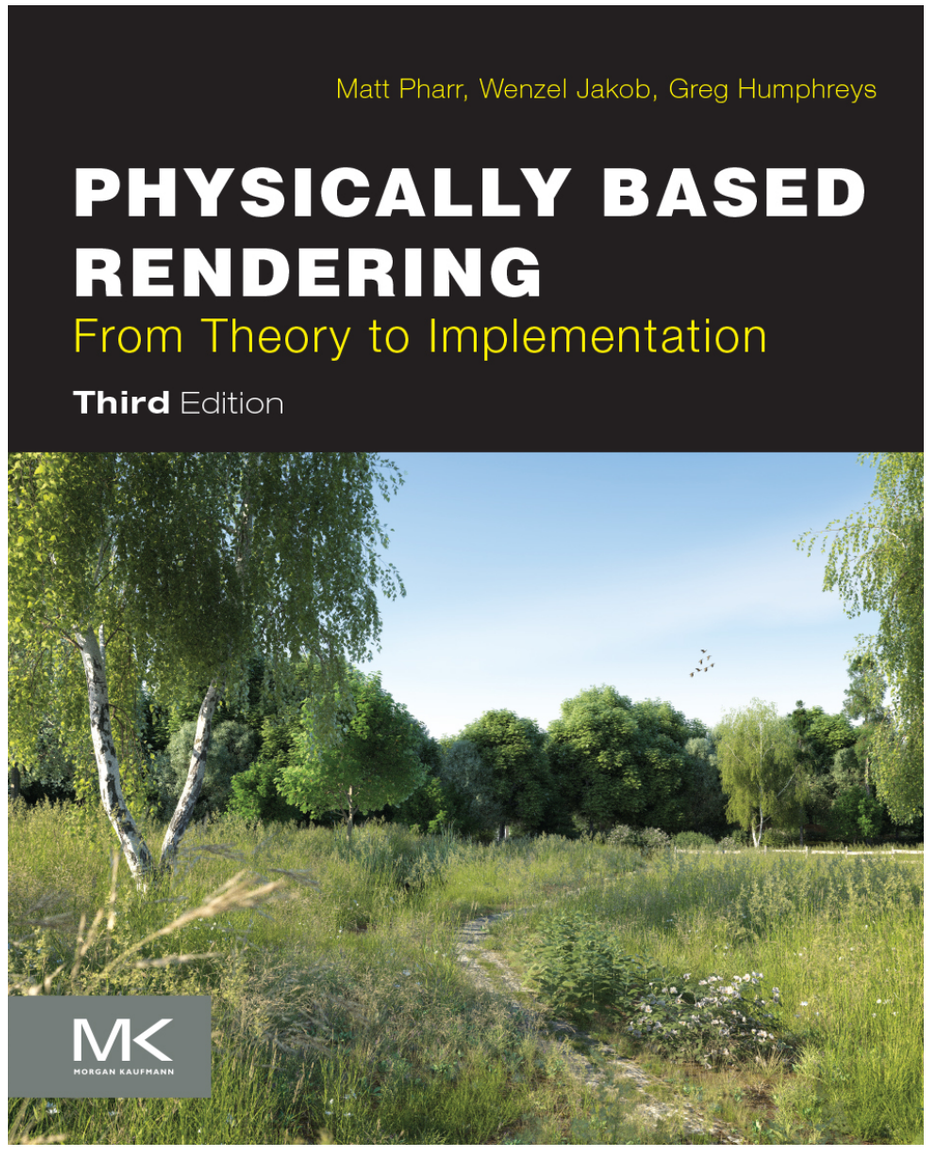
Literature

Reference Textbook:

- Pharr & Humphreys. ***Physically Based Rendering: From Theory to Implementation***, 3rd edition (2016).

- digital e-book version available from McGill IPs

  - see link on course website



Literature (optional)

Get up to speed on ray tracing basics quickly:

- 3 kindle mini-book series by Pete Shirley

Ray Tracing: in one weekend

Ray Tracing: the next week

Ray Tracing the rest of your life

Tentative Schedule

Week	Lecture Topics	Checkpoints
1	Rendering Systems and Radiometry	
2	Appearance Modelling and the Reflection Equation	
3	Monte Carlo Methods	**Assignment 1**
4	Direct Illumination	**Assignment 2**
5	Advanced Monte Carlo for Variance Reduction	
6	**[THANKSGIVING]** Path-space Light Transport Formulation	**Assignment 3**
7	Density Estimation	

Tentative Schedule

Week	Lecture Topics	Checkpoints
8	Many Light Methods and Hybrid Rendering Systems	
9	Basis-space Representations for Interactive Shading	**Assignment 4**
10	Volumetric Participating Media	
11	Advanced Monte Carlo and Density Estimation in Volumes	**Assignment 5**
12	Markov Chain Monte Carlo for Path-space Exploration	
13	Subsurface Scattering Approximations	**Project***

Schedule – Coding Viewpoint

Week	Lecture Topics	Checkpoints
1	Rendering Systems and Radiometry	
2	Appearance Modelling and the Reflection Equation	
3	Monte Carlo Methods	**Assignment 1**
4	Direct Illumination	**Assignment 2**
5	Advanced Monte Carlo for Variance Reduction	
6	**[THANKSGIVING]** Path-space Light Transport Formulation	**Assignment 3**
7	Density Estimationt*	

Schedule – Coding Viewpoint

Week	Lecture Topics	Checkpoints
8	Many Light Methods and Hybrid Rendering Systems	
9	Basis-space Representations for Interactive Shading	**Assignment 4**
10	Volumetric Participating Media	
11	Advanced Monte Carlo and Density Estimation in Volumes	**Assignment 5**
12	Markov Chain Monte Carlo for Path-space Exploration	
13	Subsurface Scattering Approximations	**Project***

Grading Scheme

0%: Participation

- I want an *interactive course experience*
  - Don't be shy! Ask questions. Start discussions.
  - Technical questions are fine, too. But not "how do I compile the code"-type questions.

Without organic participation, I will move to a "volunteered" participation model…

Grading Scheme – ECSE 446

75%: Programming Assignments

- A1 = 10%
- A2 = 10%
- A3 = 17%
- A4 = 18%
- A5 = 20%

(up to) 15%: Bonus/Hacker Points

Grading Scheme – ECSE 546

50%: Programming Assignments

- A1 = 5%
- A2 = 5%
- A3 = 12%
- A4 = 13%
- A5 = 15%

25%: Project*

(up to) 15%: Bonus/Hacker Points

Programming assignments

Late policy:

- Submit on time
  - **–100%** for late submissions
- Follow submission instructions exactly
  - **–100%** for invalid submissions

Zero tolerance for cheating!

All tasks will be submitted using **myCourses**

Exam

25%: one (1) final exam

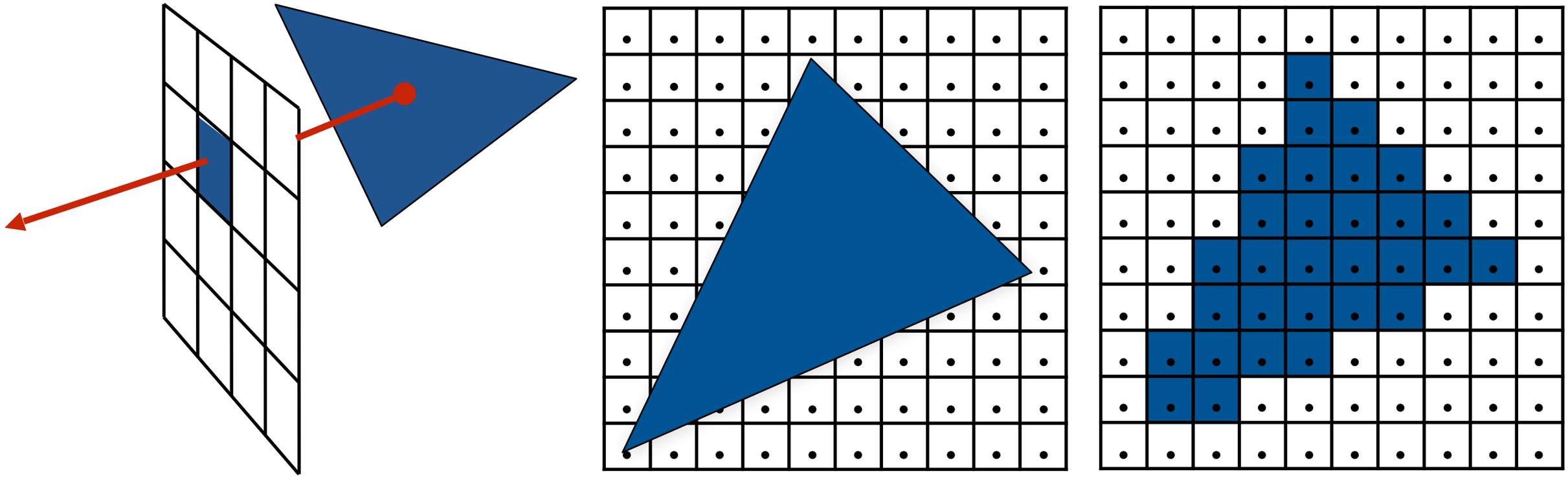
- multiple choice
- technical questions based on assignments
- conceptual questions based on remaining topics (inclusive)

Takeaway: do the assignments yourself, attend class, and the exam will be easy

Questions?

ECSE 446/546

IMAGE SYNTHESIS



RENDERING SYSTEMS

Prof. Derek Nowrouzezahrai

derek@cim.mcgill.ca

Rendering Systems

We provide you with a medium-sized codebase that includes basic plumbing:

- loading and representing 3D objects
- generating and saving 2D images
- rendering systems skeleton & support API

The code allows you to abstract **two** fundamental rendering system paradigms

- rasterization-based rendering, and
- ray tracing-based rendering

Rendering Systems

Rasterization and ray-tracing are two ways of answering the following two questions:

1. what objects do I see from my point of view?
    - the *eye-* or *primary-* or *camera-visibility problem*
2. how do I compute their colors?
    - the *shading* or *lighting problem*

You'll build atop scaffolding we provide for both these paradigms, in the base code

- it's still a good idea to understand how they operate and their differences

Rendering Systems

Rasterization and ray-tracing are two ways of answering the following two questions:

1. what objects do I see from my point of view?
    - the *eye-* or *primary-* or *camera-visibility problem*
2. how do I compute their colors?
    - the *shading* or *lighting problem*

You'll build atop scaffolding we provide for both these paradigms, in the base code

- it's still a good idea to understand how they operate and their differences

Rendering Systems

Rasterization and ray-tracing are two ways of answering the following two questions:

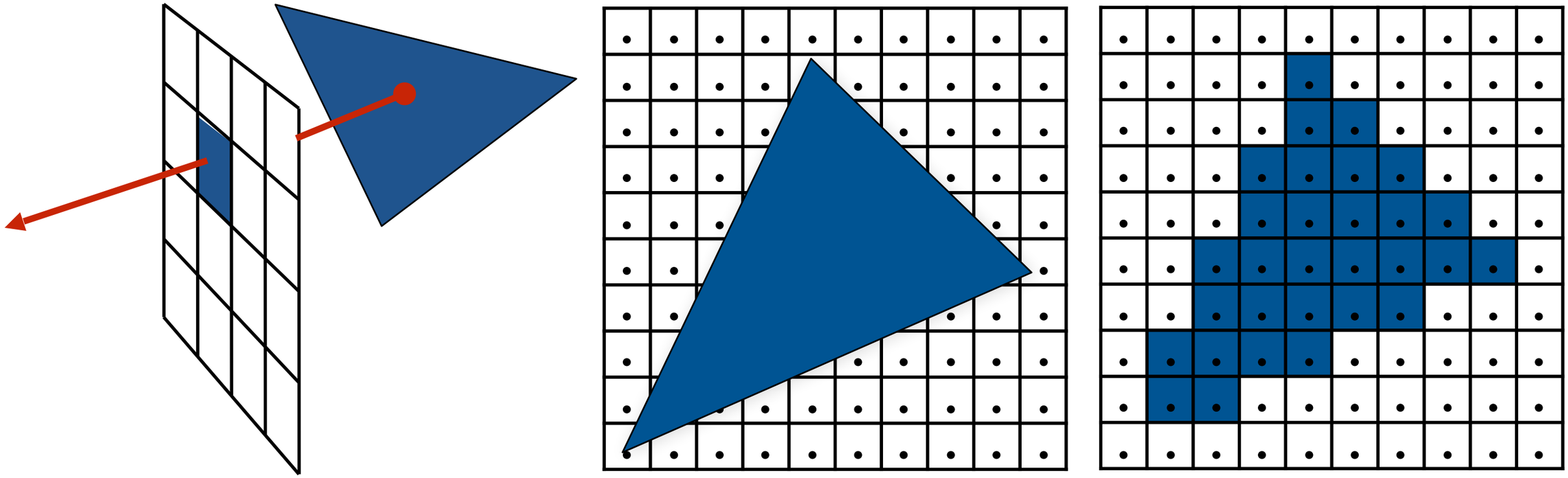
1. what objects do I see from my point of view?
    - the *eye-* or *primary-* or *camera-visibility problem*
2. how do I compute their colors?
    - the *shading* or *lighting problem*

You'll build atop scaffolding we provide for both these paradigms, in the base code

- it's still a good idea to understand how they operate and their differences

ECSE 446/546

IMAGE SYNTHESIS



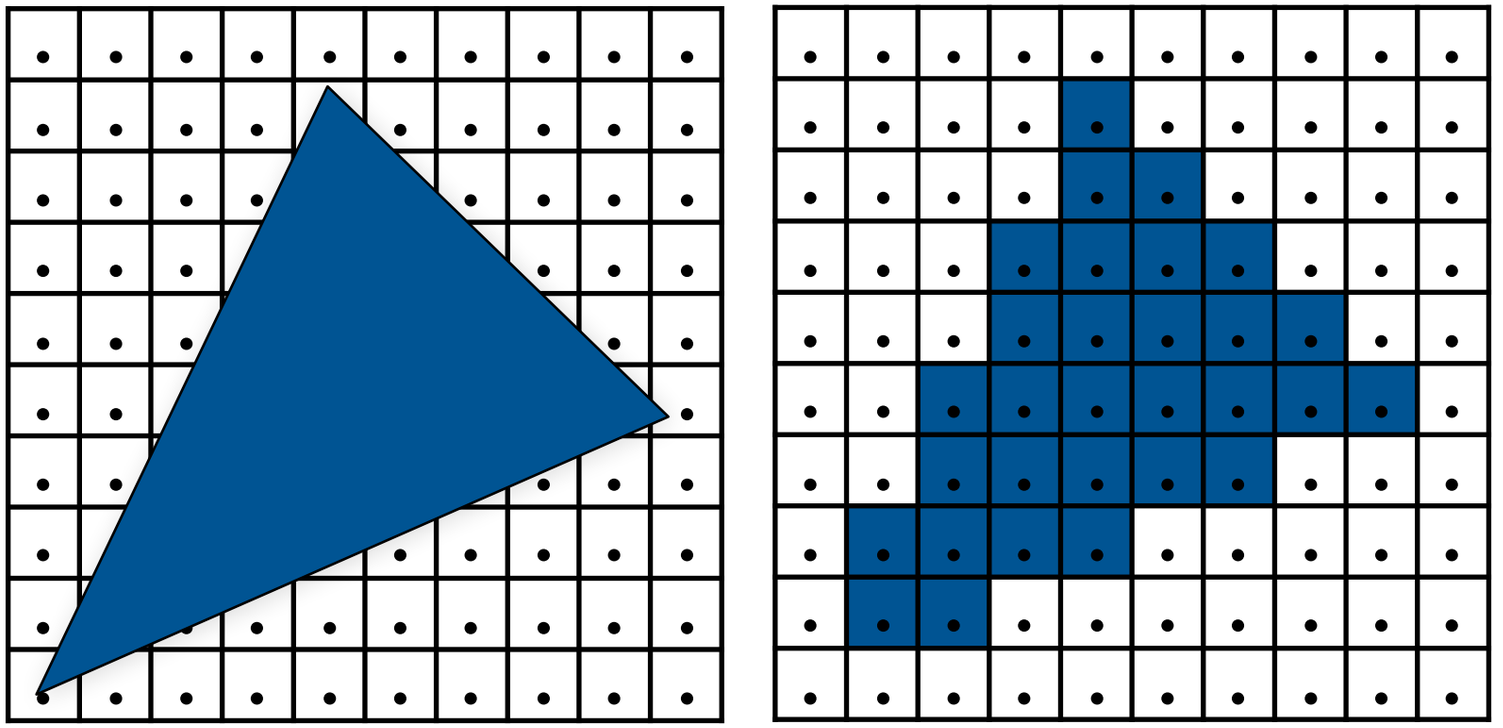
RENDERING SYSTEMS

Prof. Derek Nowrouzezahrai

derek@cim.mcgill.ca

ECSE 446/546

IMAGE SYNTHESIS



SYSTEMS 1 – RASTERIZATION

Prof. Derek Nowrouzezahrai

derek@cim.mcgill.ca

Rasterization Pipeline – GPUs

GPUs have dedicated hardware* that implements a programmable rasterization pipeline

This HW is exposed through various APIs:

- OpenGL
- DirectX
- Vulcan
- Metal

Rasterization Pipeline

We'll outline the different steps that the GPU can be directed to perform in this pipeline…

… but first, a simple example…

Activity: Modelling & Drawing a Cube

Goal: realistically draw a cube

Key questions:

- Modelling: how do we describe the cube?
- Rendering: how do we visualize this model?

After a slide by Keenan Crane

Modelling the Cube

Suppose our cube is…

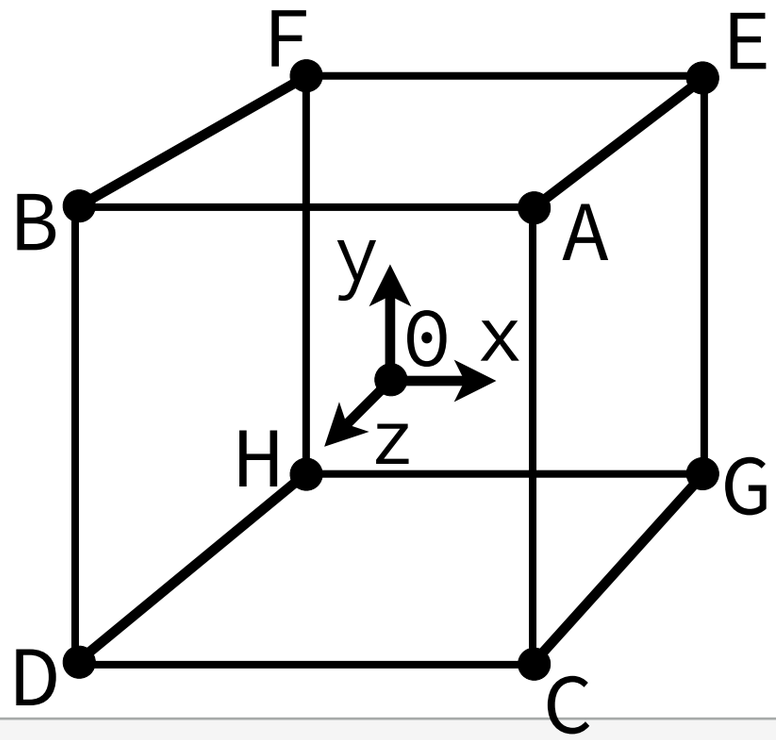
- centered at the origin (0,0,0)
- has dimensions 2x2x2
- edges are aligned with the x/y/z-axes

After a slide by Keenan Crane

Q: What are the cube vertex coordinates?

VERTICES

```
A: ( 1, 1, 1)    E: ( 1, 1,-1)
B: (-1, 1, 1)    F: (-1, 1,-1)
C: ( 1,-1, 1)    G: ( 1,-1,-1)
D: (-1,-1, 1)    H: (-1,-1,-1)
```



From Modelling to Drawing

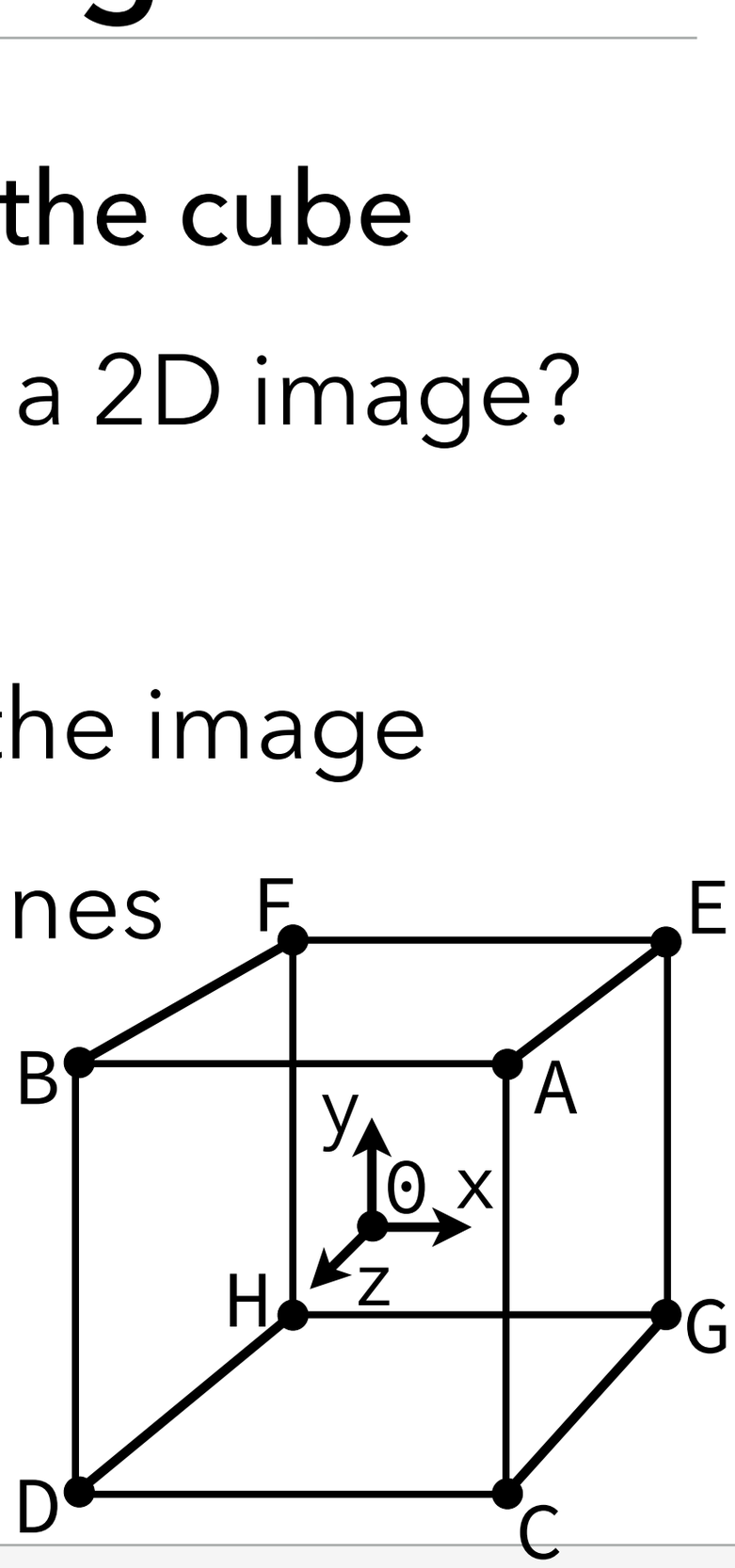
Now with a digital description of the cube

- How do we draw the 3D cube as a 2D image?

Basic strategy:

- map 3D vertices to 2D points in the image
- connect 2D points with straight lines

OK… but how?



Perspective

After a slide by Alyosha Efros

MENU the ONION Study: People Far Away From You Not Actually Smaller SEARCH TOP HEADLINES

Study: People Far Away From You Not Actually Smaller

NEWS
August 22, 2013

VOL 49 ISSUE 34
Science & Technology

Researchers say that, contrary to prior assertions, the subject above stands at equal height at left and at right, and does not grow smaller as he walks away from the camera.

Perspective

MENU | the ONION | Study: People Far Away From You Not Actually Smaller | SEARCH | TOP HEADLINES

Study: People Far Away From You Not Actually Smaller

NEWS
August 22, 2013

VOL 49 ISSUE 34
Science & Technology

Researchers say that, contrary to prior assertions, the subject above stands at equal height at left and at right, and does not grow smaller as he walks away from the camera.

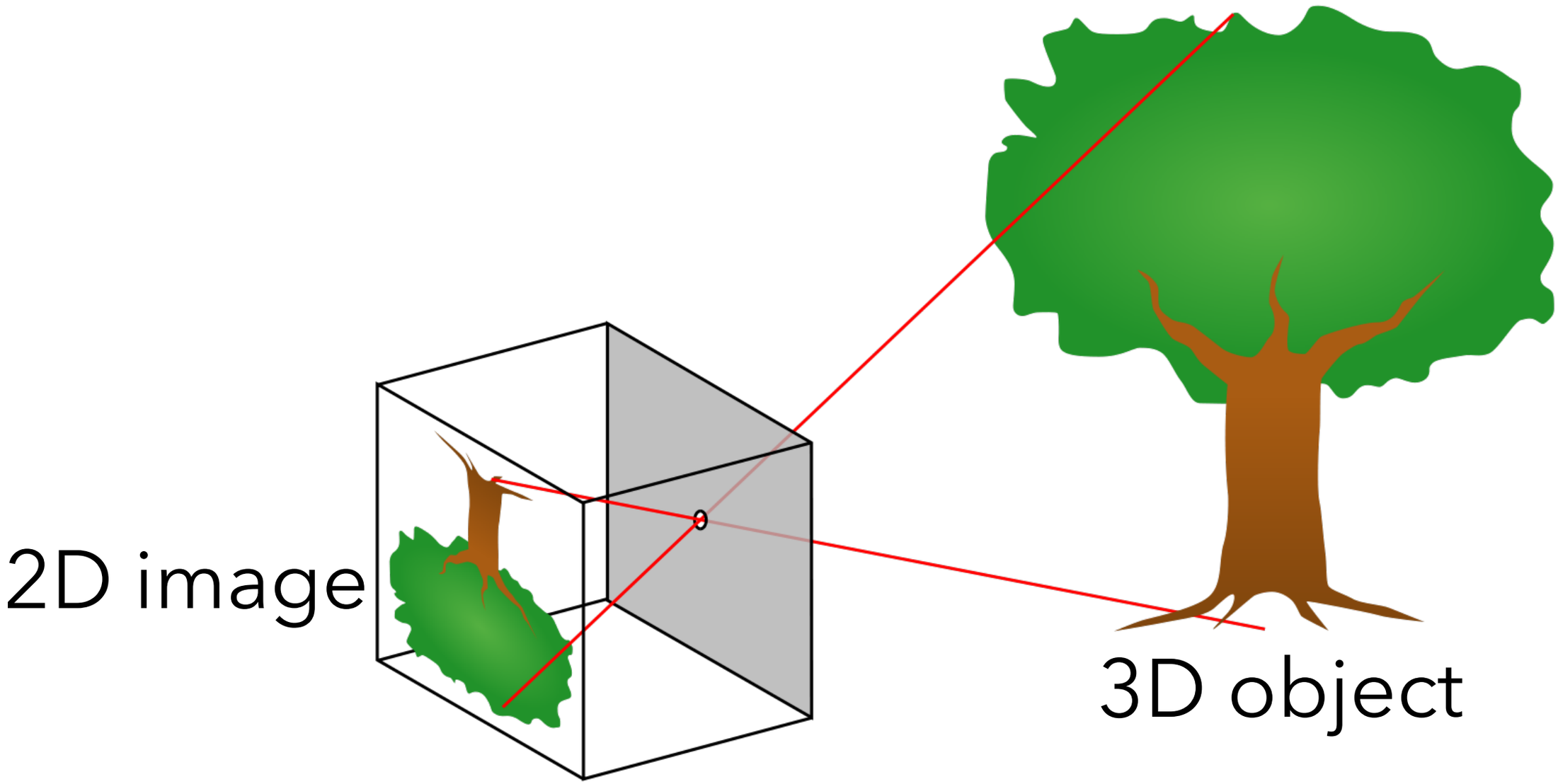
After a slide by Alyosha Efros

Objects *appear* smaller as they get further away ("perspective")

Why does this happen?

Perspective Projection

Consider simple (“pinhole”) camera:

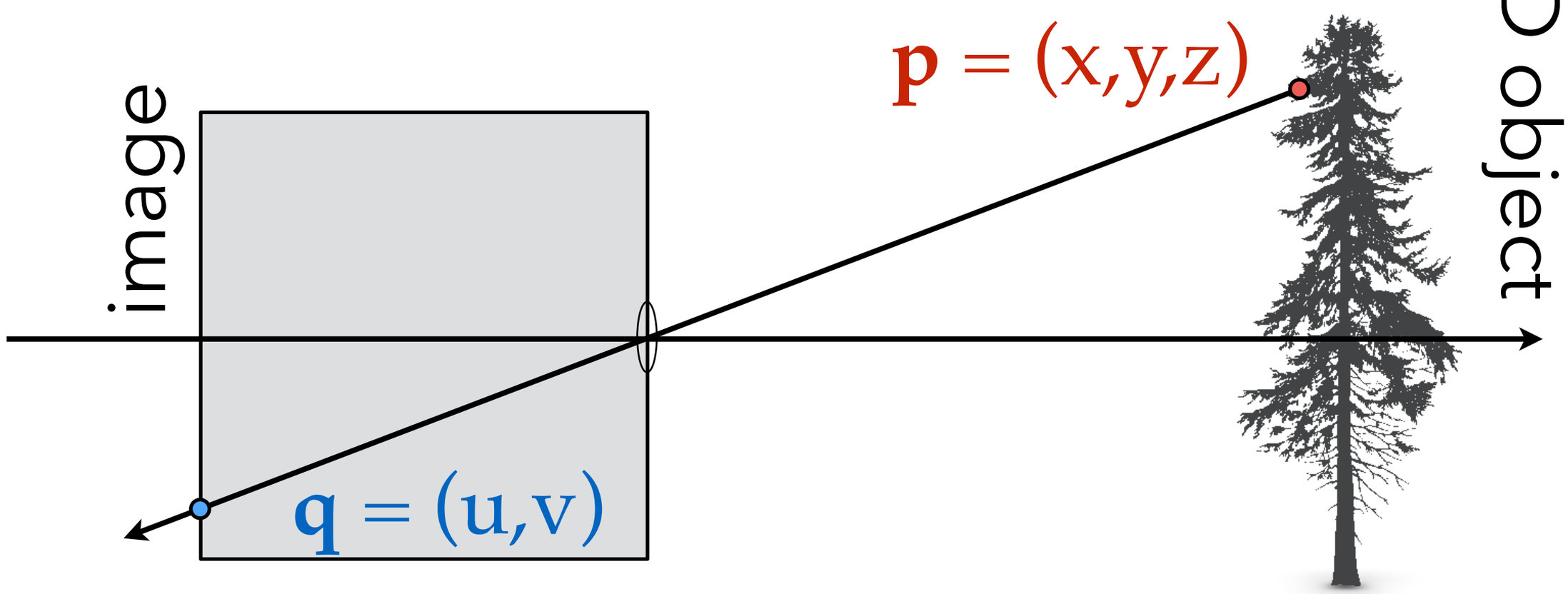


After a slide by Keenan Crane

Perspective Projection

Where does a point $\mathbf{p} = (x,y,z)$ end up on the image?

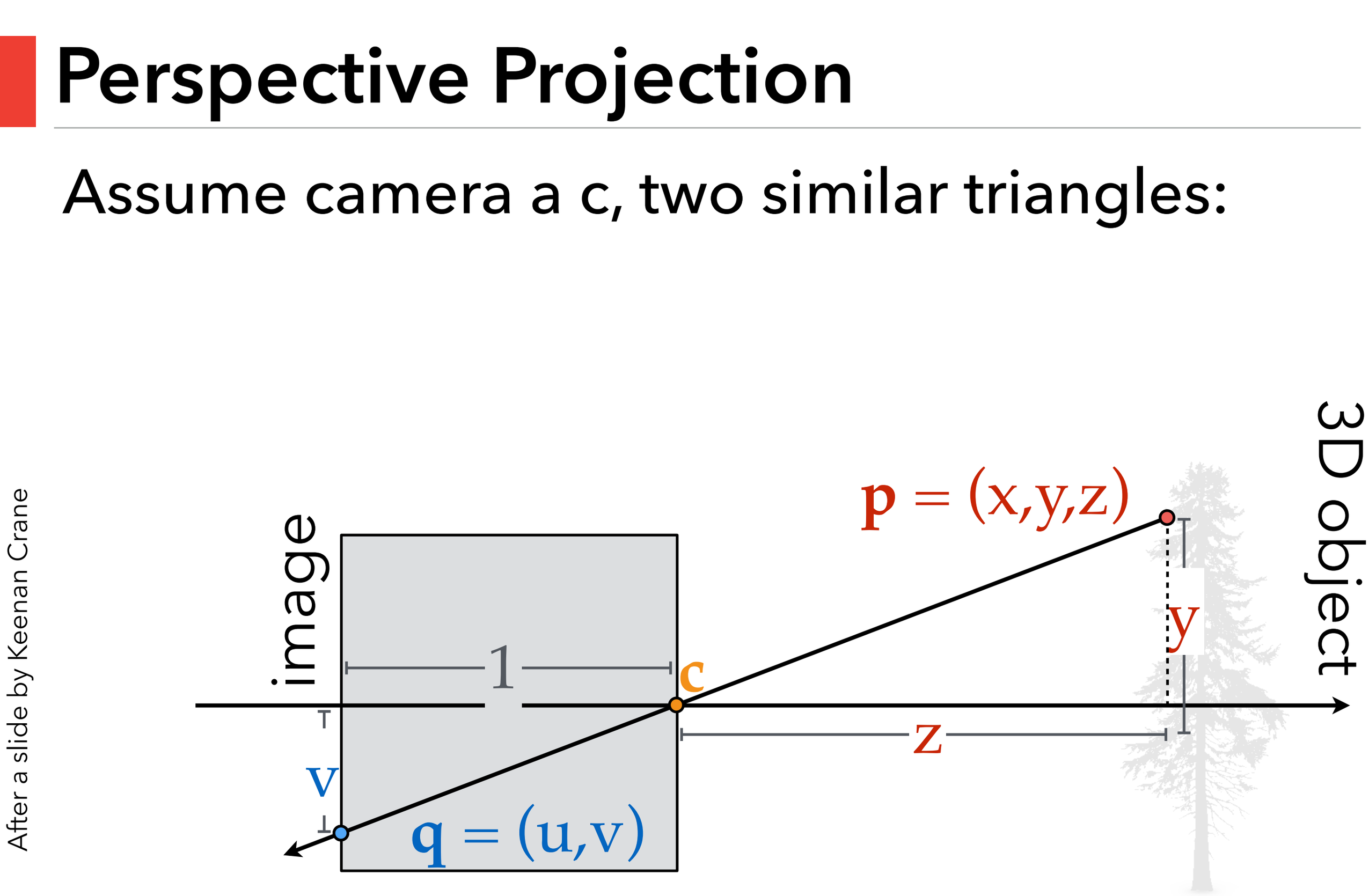
Let's call the image point $\mathbf{q} = (u,v)$



After a slide by Keenan Crane

Perspective Projection

Assume camera a c, two similar triangles:



After a slide by Keenan Crane

Draw the Projected Cube

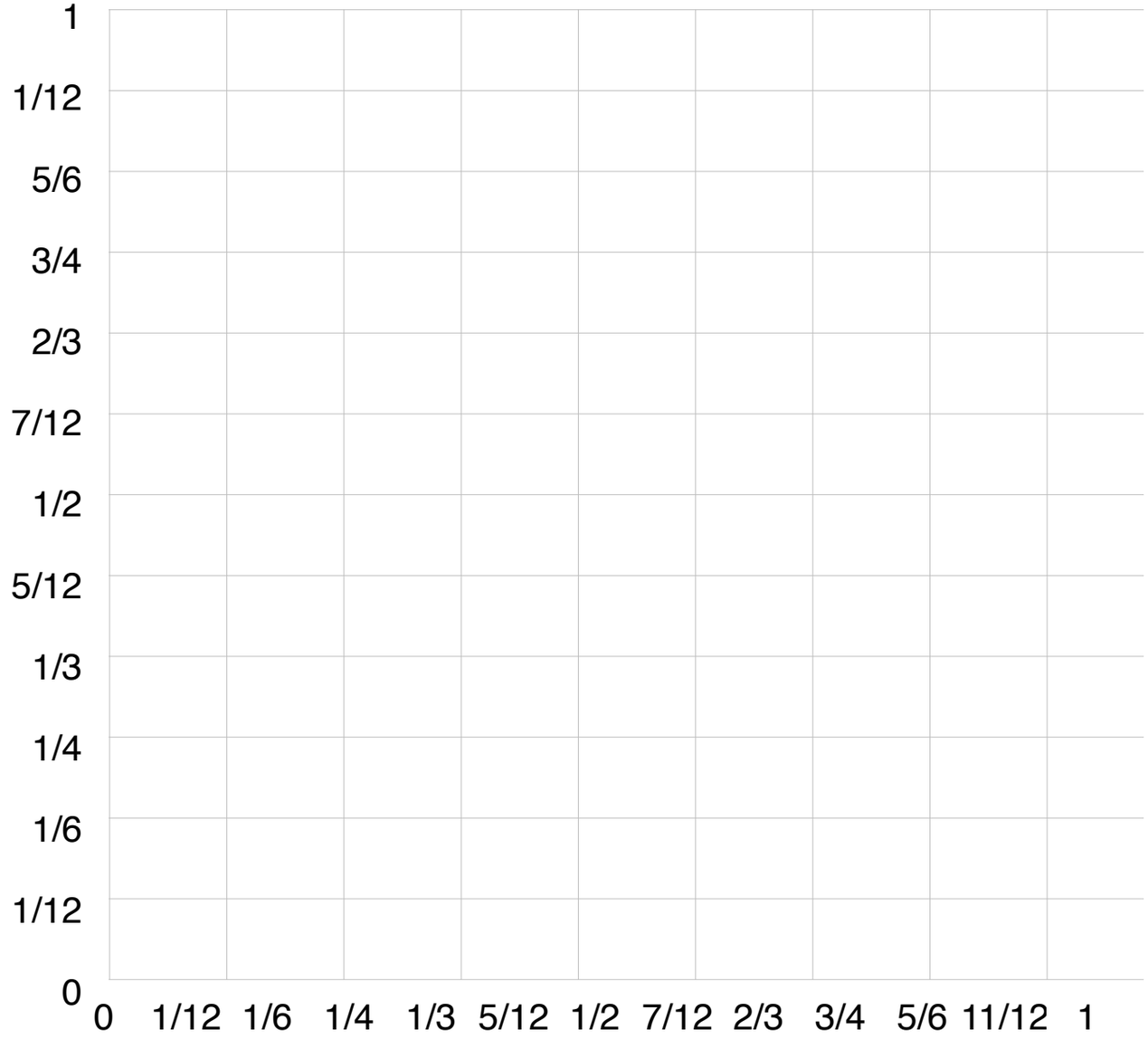
Specification & Algorithm:

- assume camera is at $\mathbf{c} = (2,3,5)$
- for each edge
  - convert (x,y,z) of both endpoints to (u,v):
    1. subtract camera $\mathbf{c}$ from vertex (x,y,z) to get (x',y',z')
    2. divide (x',y') by z' to get (u,v) – write as a fraction
  - draw line between (u_1,v_1) and (u_2,v_2)

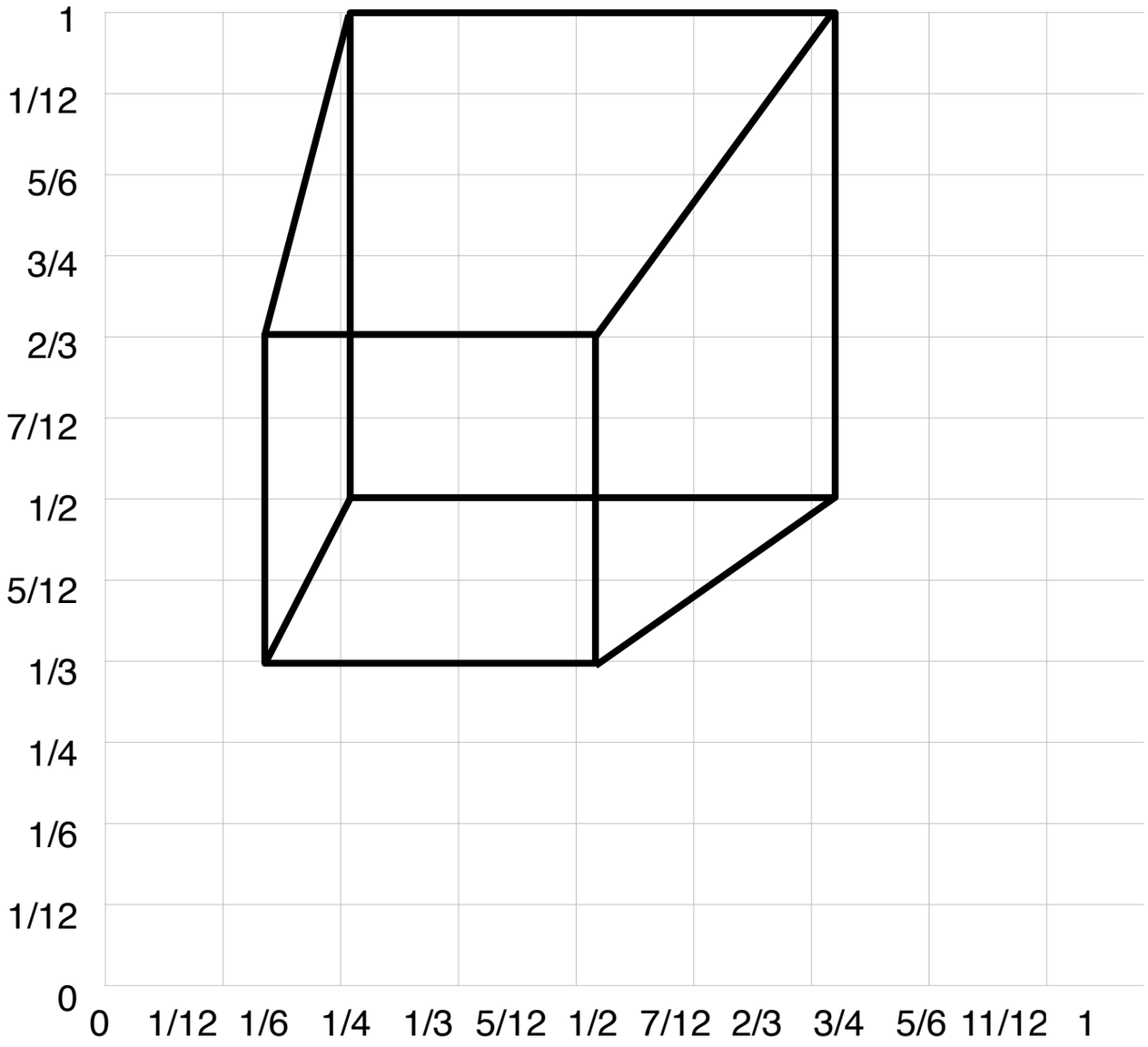
After a slide by Keenan Crane

Draw the Projected Cube

After a slide by Keenan Crane



Draw the Projected Cube



2D coordinates:

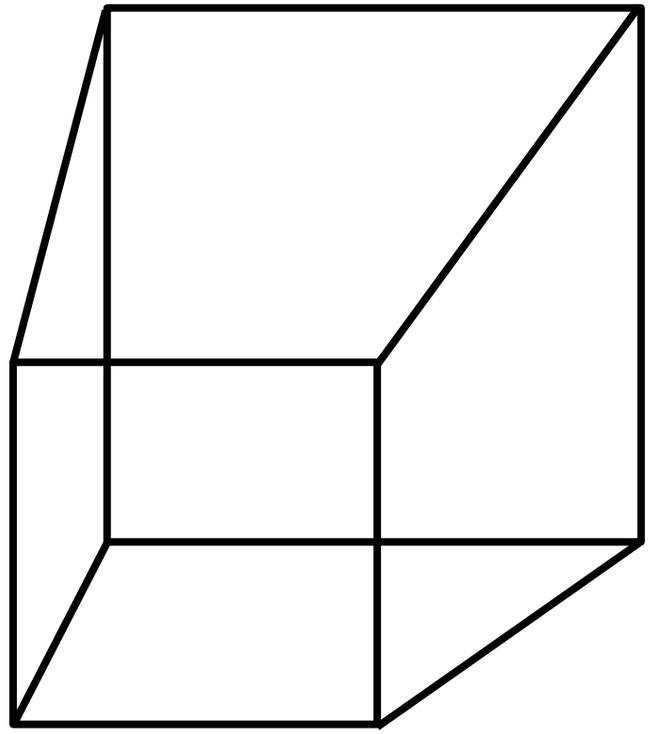
A: (?, ?)
B: (?, ?)
C: (?, ?)
D: (?, ?)

E: (?, ?)
F: (?, ?)
G: (?, ?)
H: (?, ?)

After a slide by Keenan Crane

But wait…

How do we actually draw lines on a computer?



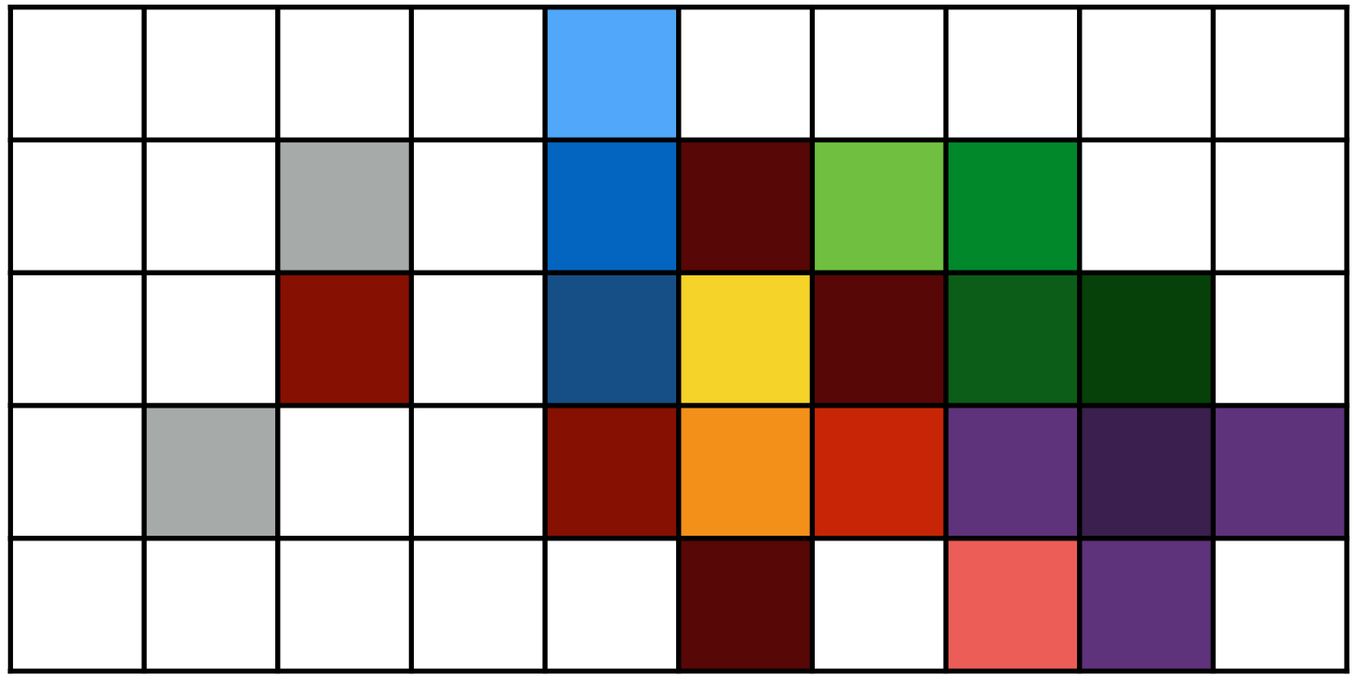
Raster Graphics

After a slide by Keenan Crane

Raster Graphics

Common abstraction of a raster display:

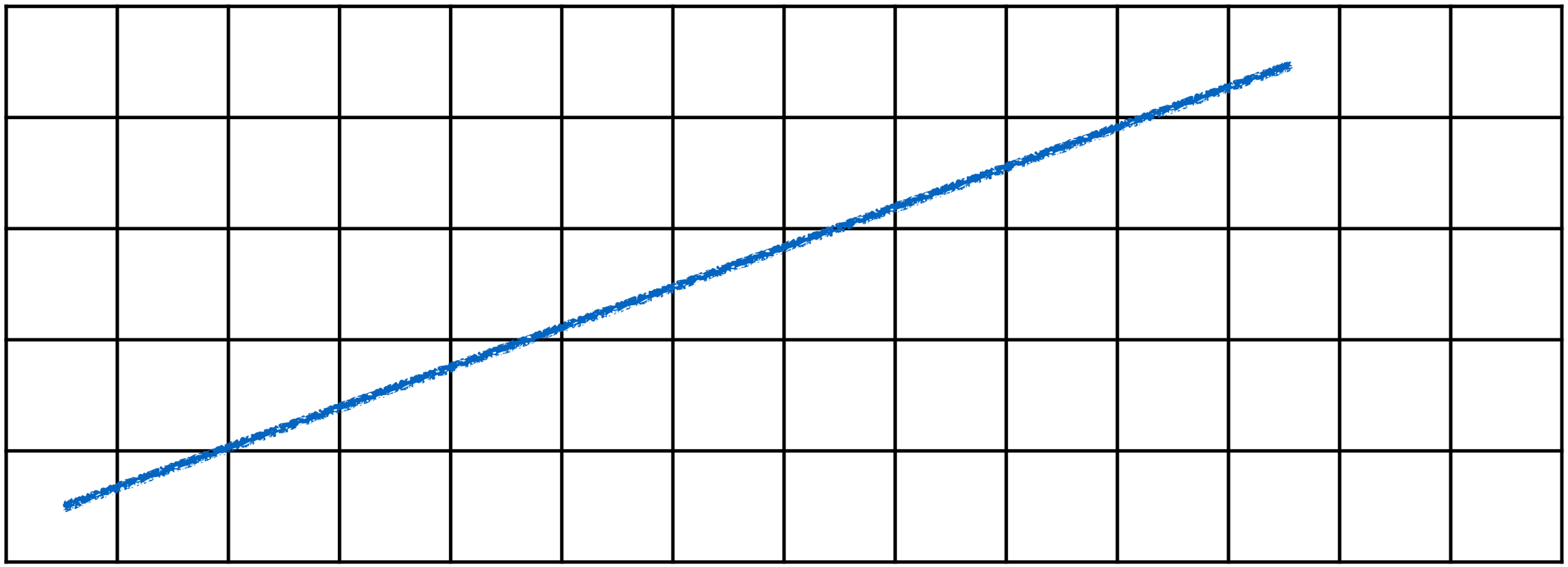
- Image represented as a 2D grid of "pixels" (picture elements)
- Each pixel can take on a unique color value



After a slide by Keenan Crane

Which Pixels to Fill to Draw the Line?

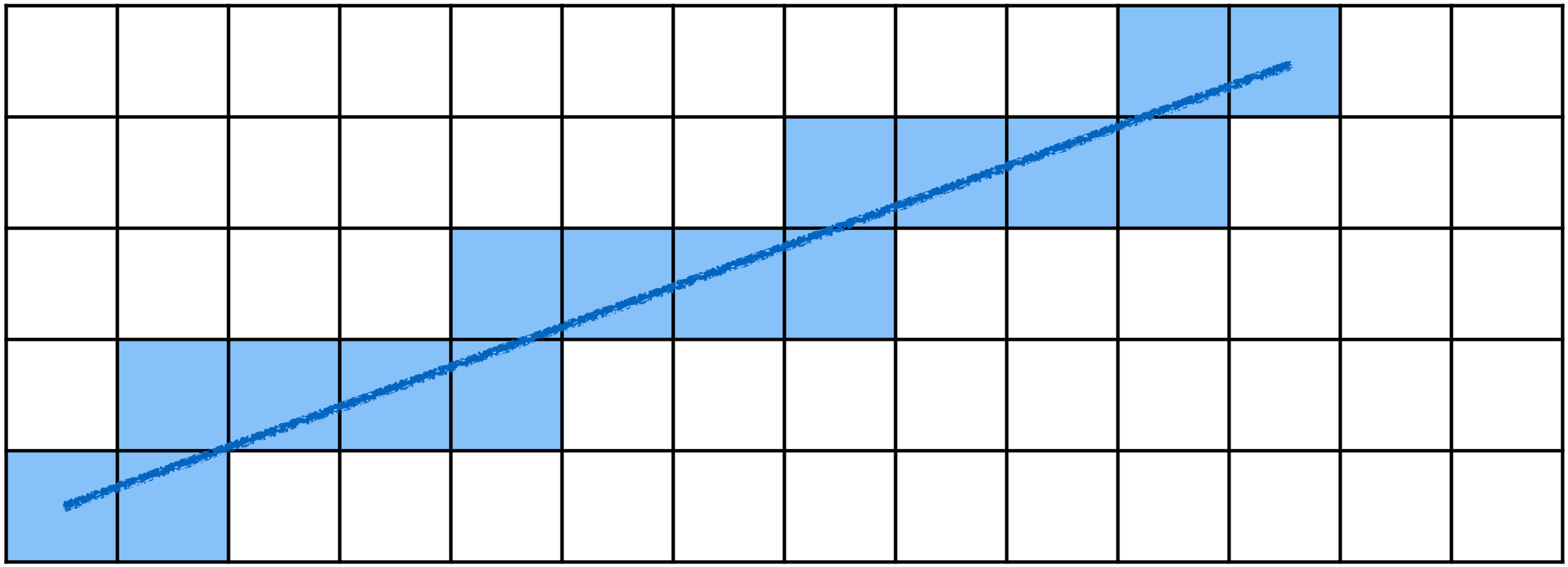
Rasterization is the process of converting a continuous object to a discrete representation on a raster (pixel) grid



After a slide by Keenan Crane

Rasterization Rules

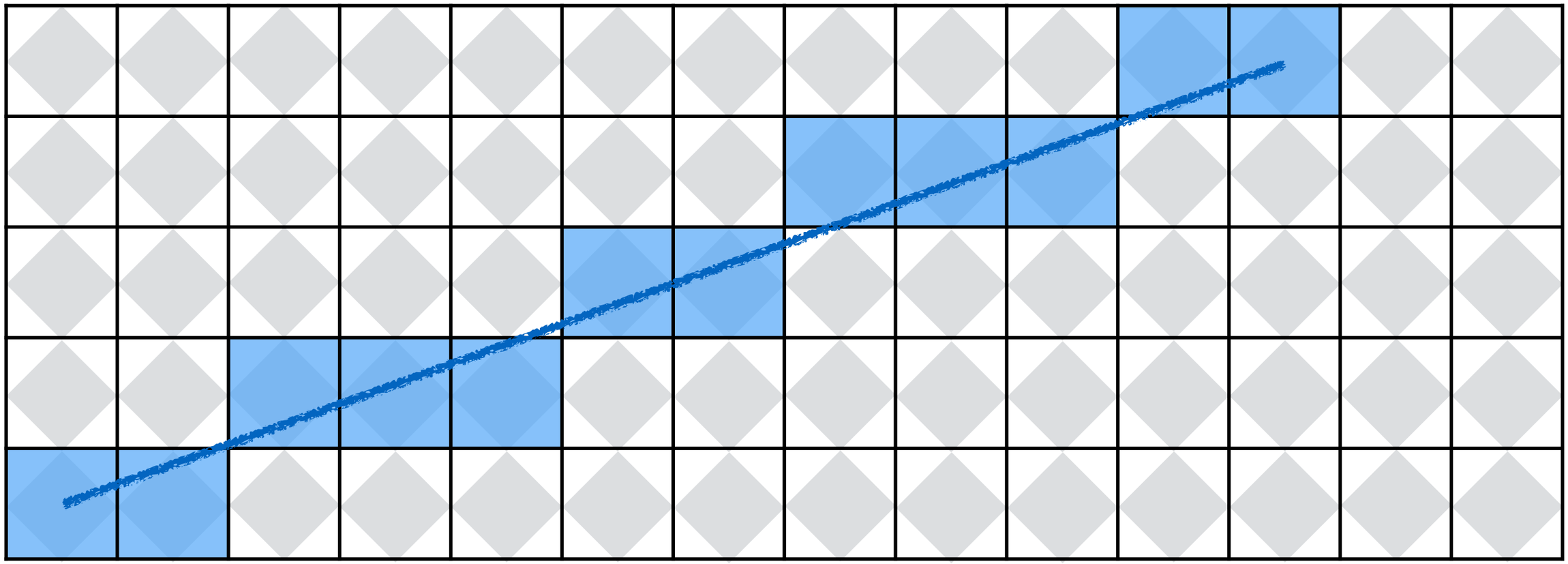
Light up all pixels intersected by the line?



After a slide by Keenan Crane

Rasterization Rules

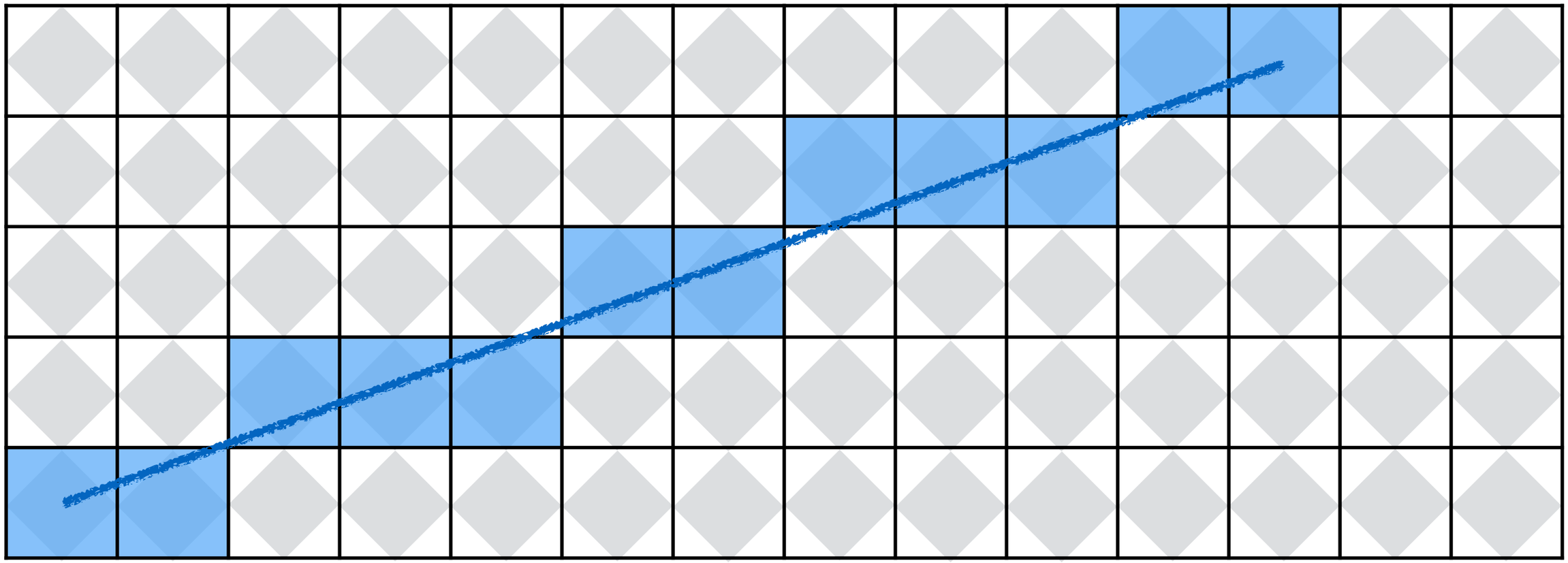
Diamond rule (used by modern GPUs): light up pixel if line passes through associated diamond



After a slide by Keenan Crane

Rasterization Rules

Is there a right answer?



After a slide by Keenan Crane

Line Rasterization

How do we find the pixels satisfying a chosen rasterization rule?

- Could check every single pixel in image to see if it meets the condition…
  - $O(n^2)$ image pixels vs. at most $O(n)$ "lit up"
  - *must* be able to do better!
    - e.g., work proportional to number of pixels in the drawing of the line

After a slide by Keenan Crane

Incremental Line Rasterization

Let's say a line has integer endpoints: (u_1,v_1), (u_2,v_2)

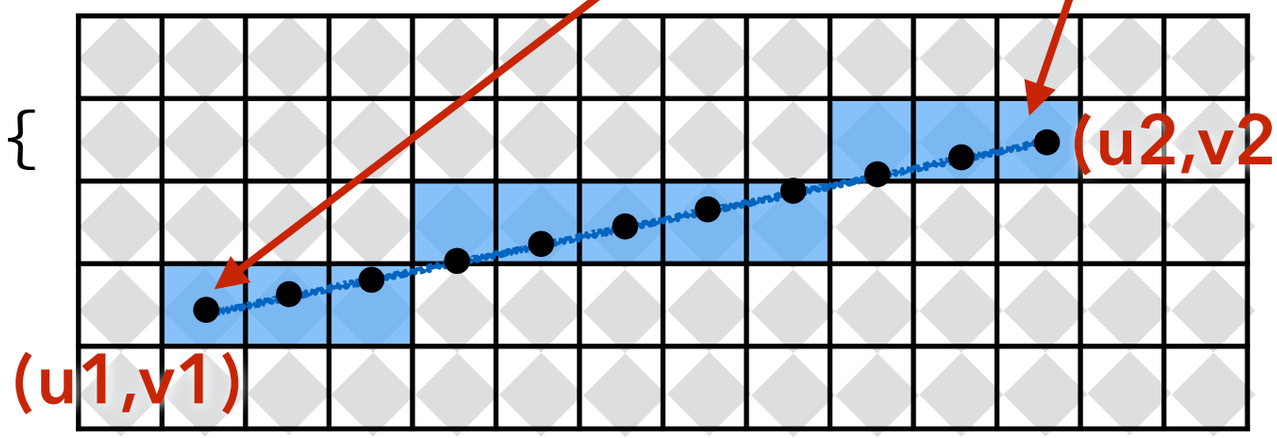
The line's slope is $s = (v_2 - v_1)/(u_2 - u_1)$

Consider the case where:

- $u_1 < u_2$, $v_1 < v_2$ (line points toward upper-right)
- $0 < s < 1$ (more change in x than y)

```
v = v1;
for (u = u1; u <= u2; ++u) {
        v += s;
        draw(u, round(v))
}
```

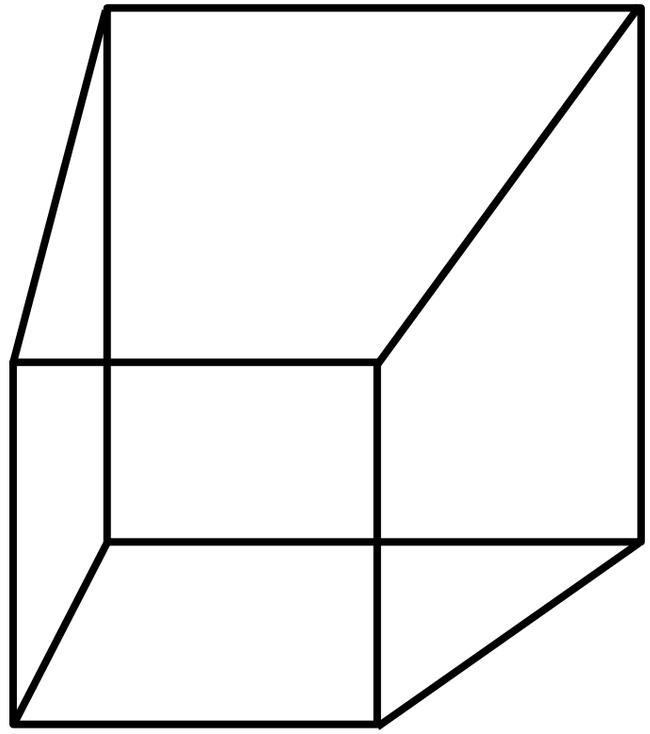
Assume integer coordinates are at pixel centers



After a slide by Keenan Crane

But wait…

How do we actually draw lines on a computer?

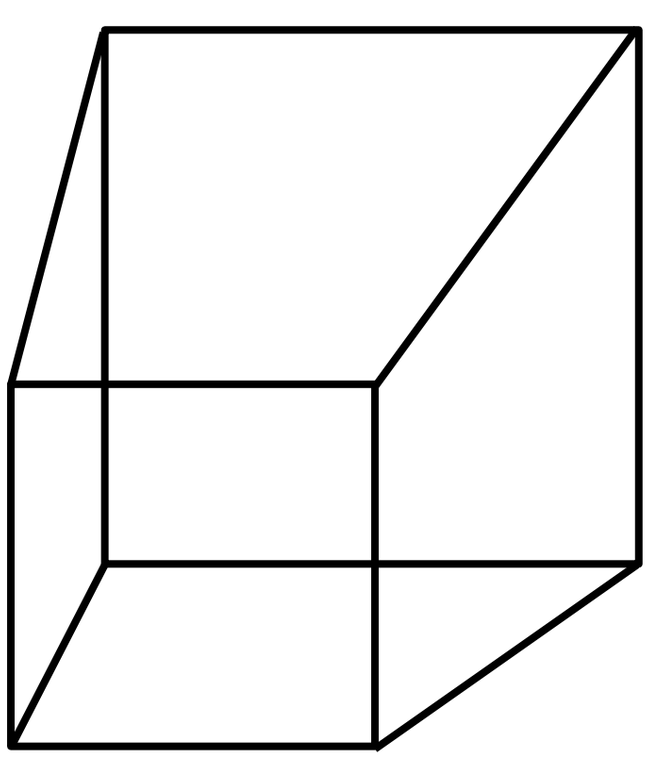


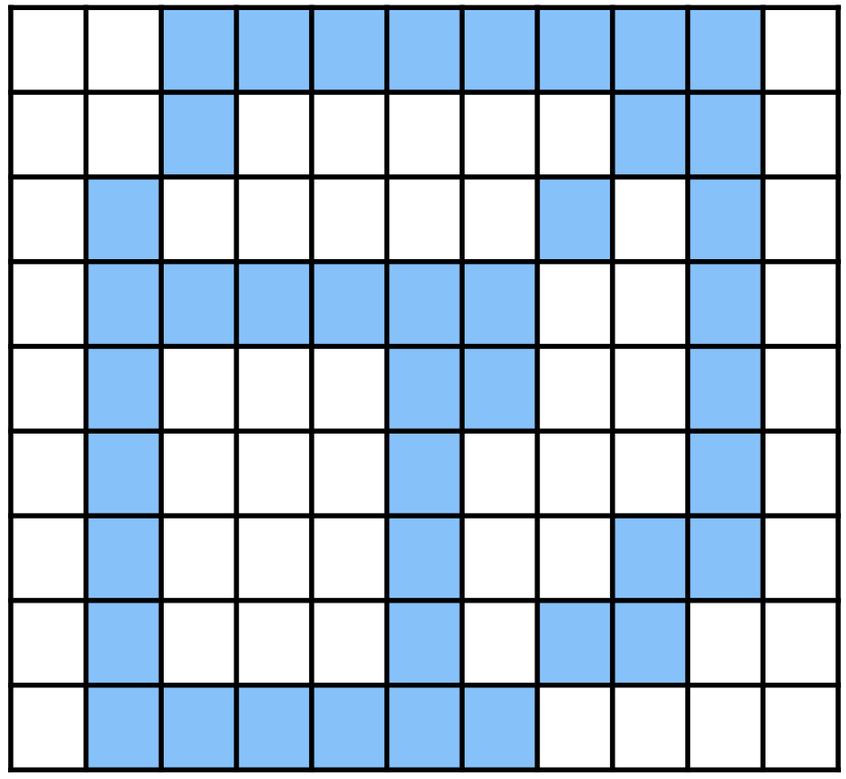
But wait…

~~How do we actually draw lines on a computer?~~

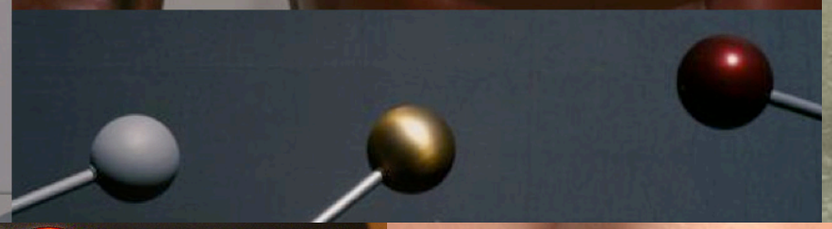
What is this object/picture missing?

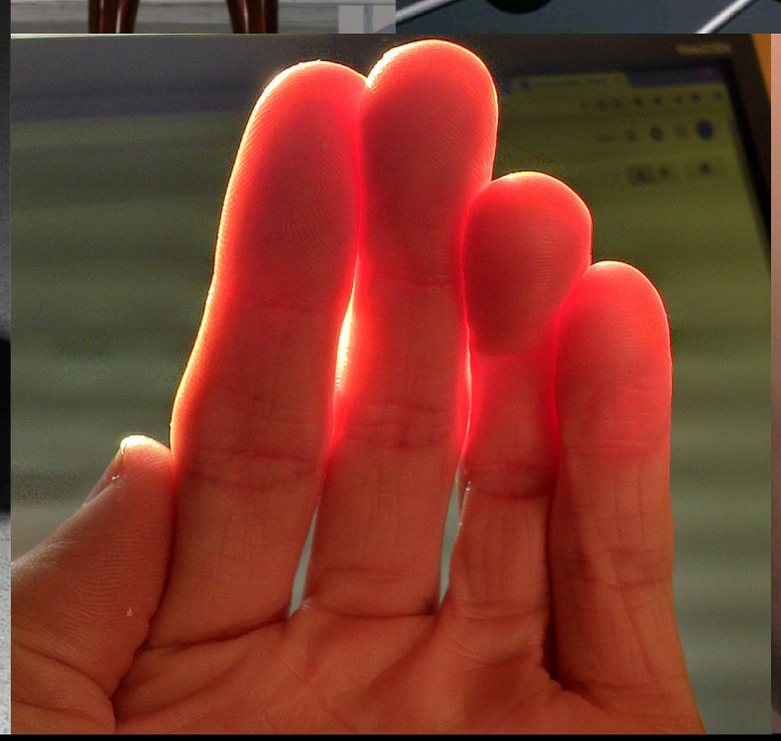
- surface, material, lighting/shading, motion…



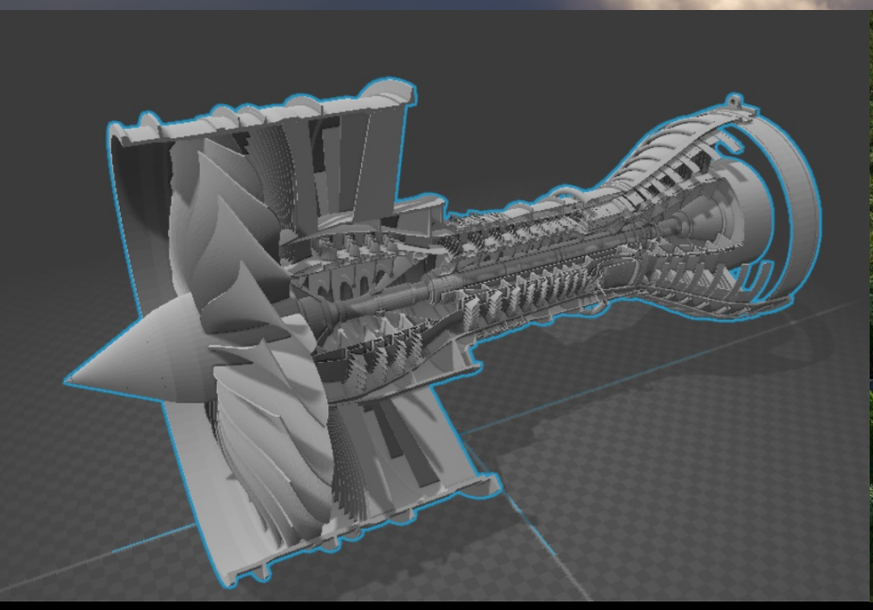


Modelling Material Properties





Complex 3D Surfaces & Volumes



Realistic Lighting Environments

WALL·E (Pixar, 2008)

Realistic Lighting Environments

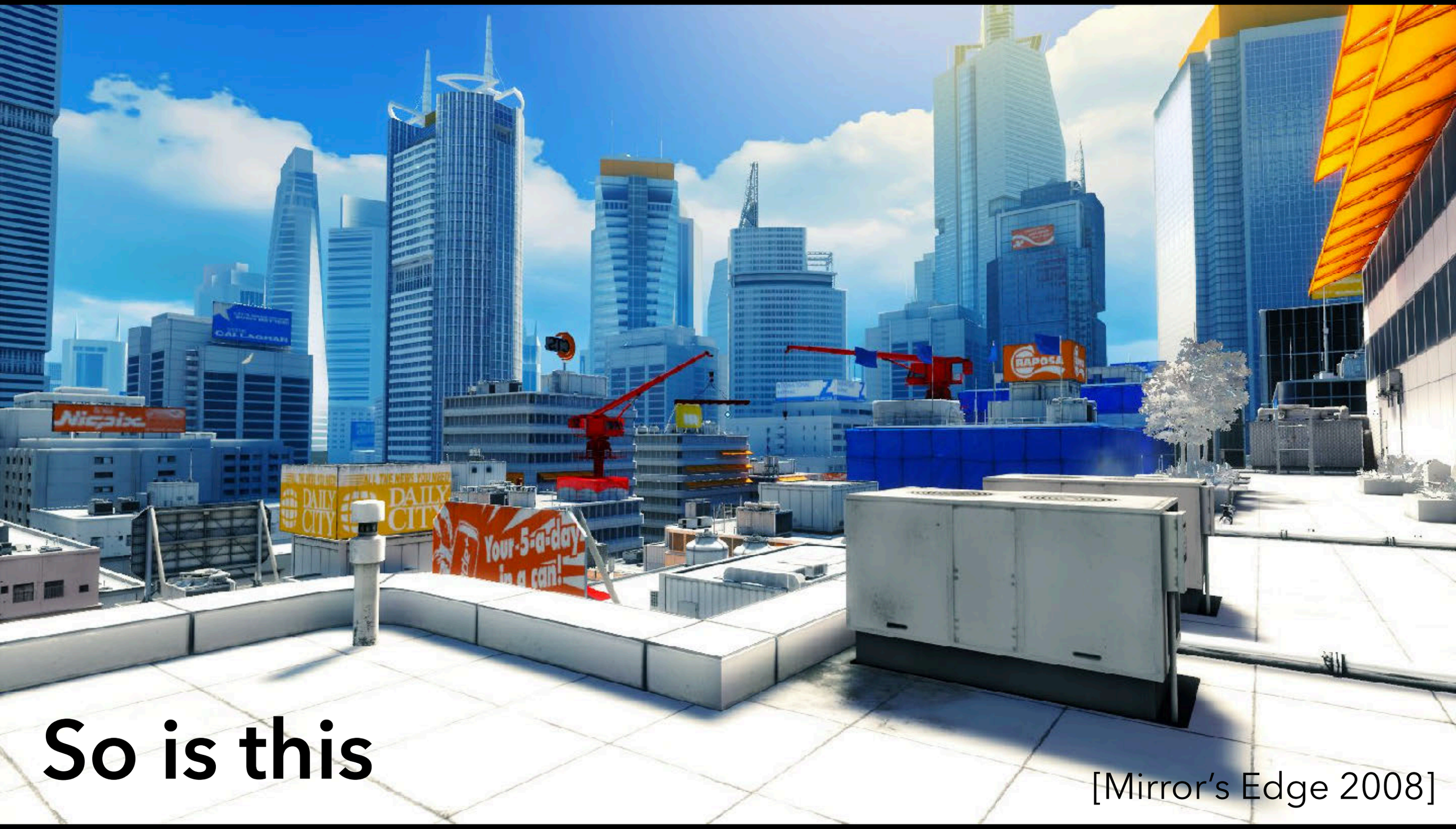
Industrial Light + Magic

Realistic Lighting Environments

Big Hero 6 (Disney 2014)

This image is rendered in realtime on a modern GPU

Unreal Engine Kite Demo (Epic Games 2015)



So is this

[Mirror's Edge 2008]

Rasterization Pipeline

The modern (programmable) rasterization **pipeline** allows us to **scale** simple line drawings to these complex examples

To do so, it works in "stages"

- turns out that **triangles** subsume lines as the baseline primitive in a rasterizer
- moving forward, you can assume that every rasterizer acts on a set of triangles

Rasterization Pipeline – GPUs

You'll be exposed to some of these gritty API details in the tutorial sessions

- we abstract a good deal of this from you*

At a high-level, the GPU rasterizer and these APIs allow you to:

- take triangles,
- figure out which pixel**s** they cover, and
- perform some computation on these pixels

```
for(each triangle)
```

these APIs allow you to:

- take triangles,
- figure out which pixel**s** they cover, and
- perform some computation on these pixels

```
for(each triangle)
    for(each pixel)
        if(triangle covers pixel)
            perform some computation
```

We're not actually going to test all pixels for each triangle; that would be impractical!

Programmable Rasterization Pipeline™

After a slide by Steve Marschner

transformations & shading → vertex data in **[vertex processing]** transformed vertex data

convert to pixels → **[clipping and rasterization]** fragments w/ interpolated data

visibility & shading → **[fragment processing]** frame buffer

user output → **[display]**