Lecture 1:

Course Introduction:
Welcome to Computer Graphics!

Computer Graphics: Rendering, Geometry, and Image Manipulation
Stanford CS248A, Winter 2024

Hi!

Josephine the (Graphics) Cat

Kayvon Fatahalian

Purvi Goel

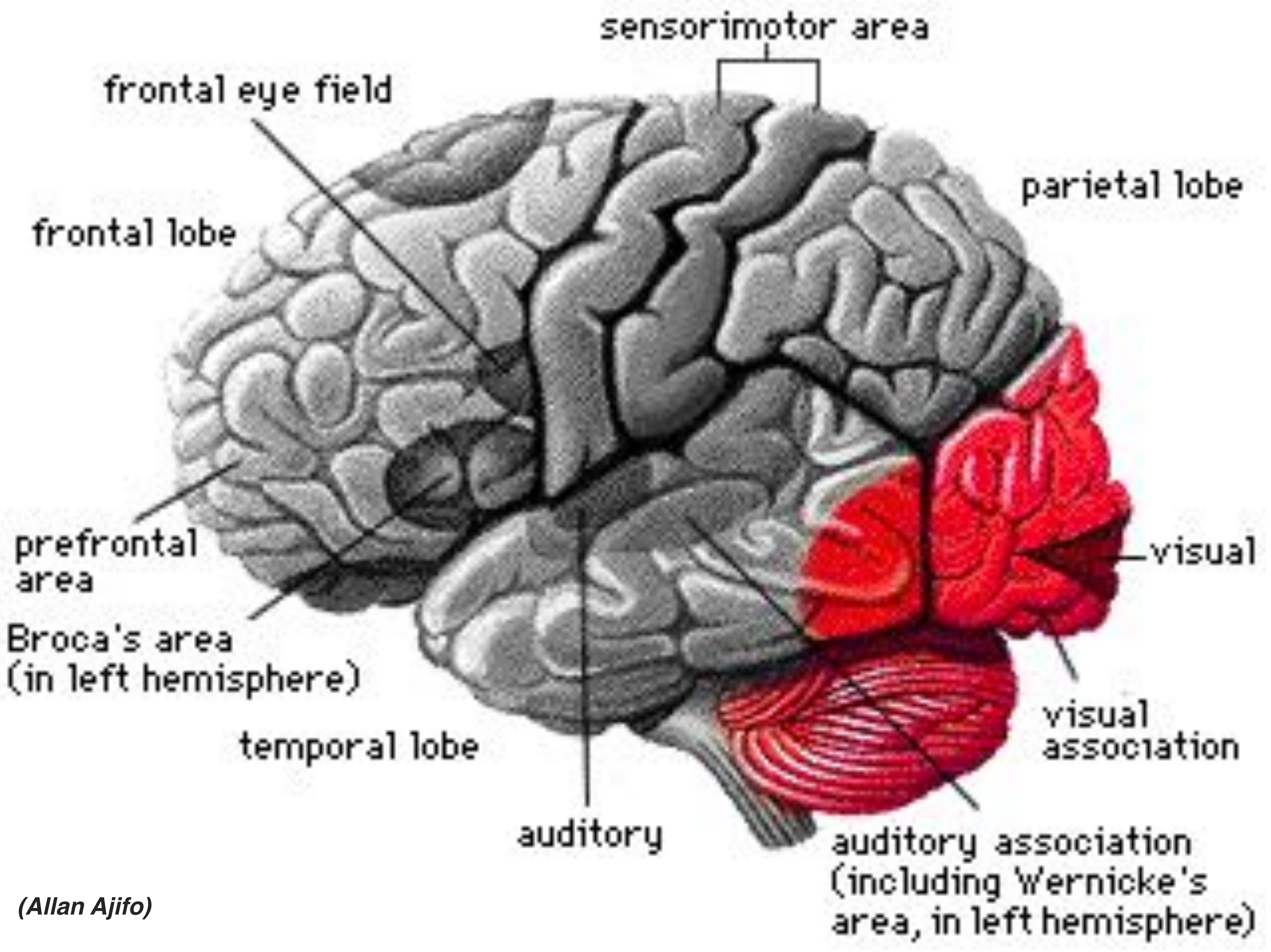
Sofia Wyetzner

Mia Tang

Discussion:
Why study computer graphics?

Why generate *visual* information?

About 30% of brain dedicated to visual processing...



(Allan Ajifo)

(Petar Milošević)

...eyes are highest-bandwidth port into the head!

Movies

Avatar (2009)

Computer games

This image is rendered in real-time on a modern GPU

Unreal 5 Demo (Nanite renderer)

Supercomputing for games

NVIDIA Titan RTX 3090 GPU

~ 40 TFLOPs fp32 *

* doesn't include additional 70 TFLOPS of ray tracing compute + 320 TFLOPS of DNN compute

Specialized processors for performing graphics computations.

Virtual reality experiences

Augmented reality

Apple Vision Pro (to be released this quarter!)

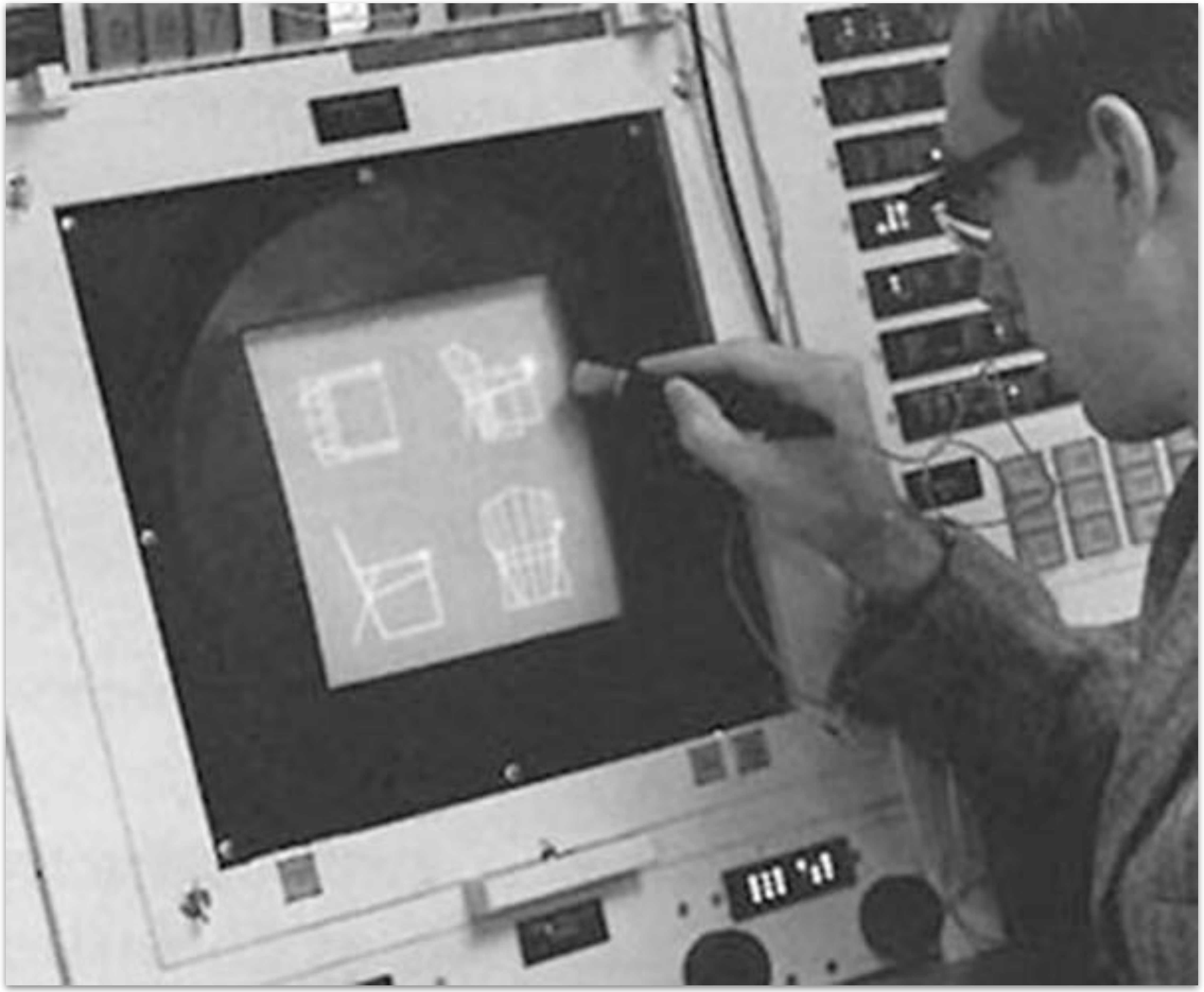
Illustration


Indonesian cave painting (~38,000 BCE)

Digital illustration

Meike Hakkart
http://maquenda.deviantart.com/art/Lion-done-in-illustrator-327715059

Graphical user interfaces



Ivan Sutherland, “Sketchpad” (1963)

Doug Engelbart
Mouse

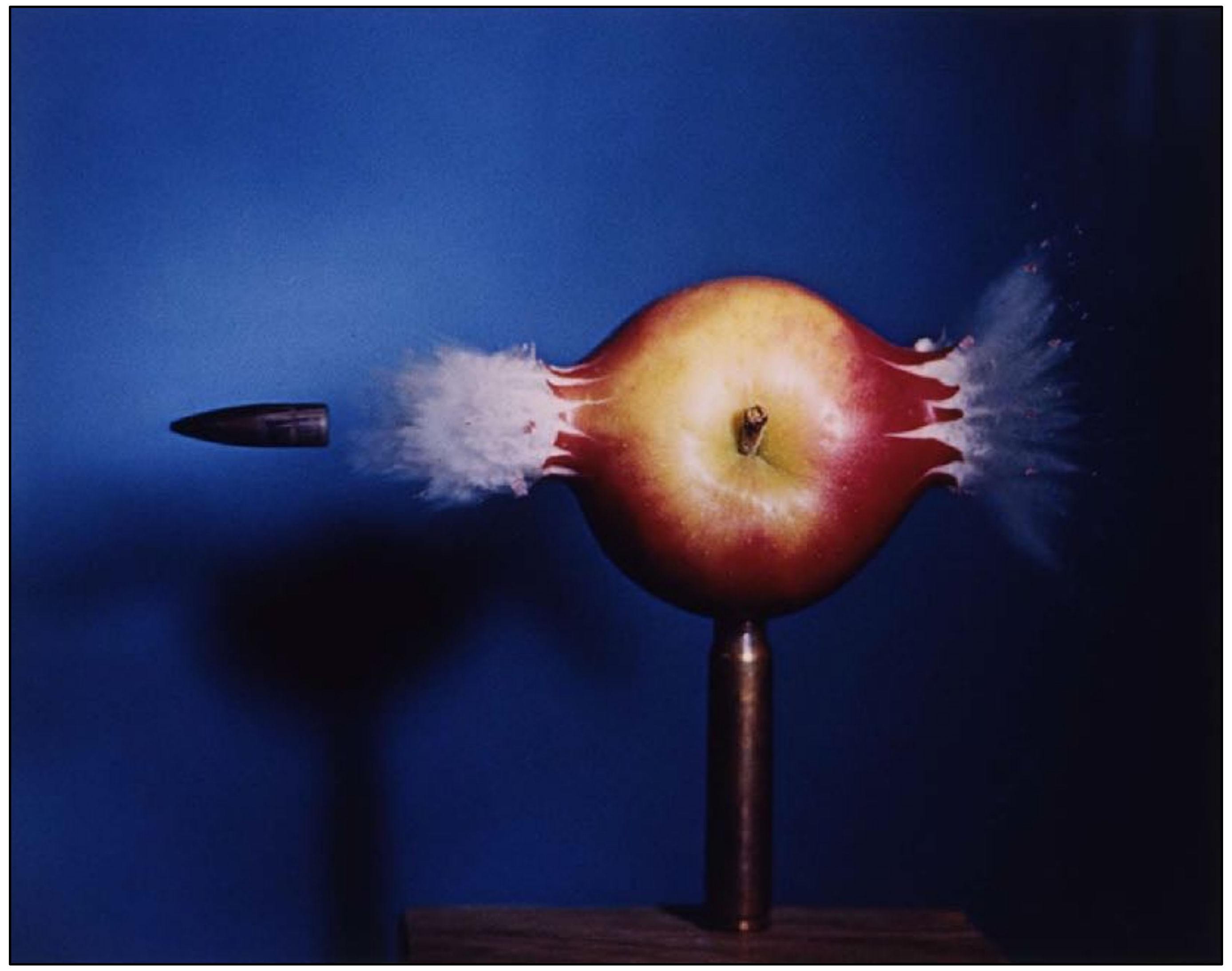
Modern graphical user interfaces

2D drawing and animation are ubiquitous in computing.

Typography, icons, images, transitions, transparency, . . .

(all rendered at high frame rate for rich experience)

Digital photography



NASA | Walter Iooss | Steve McCurry
Harold Edgerton | NASA | National Geographic

Ubiquitous imaging

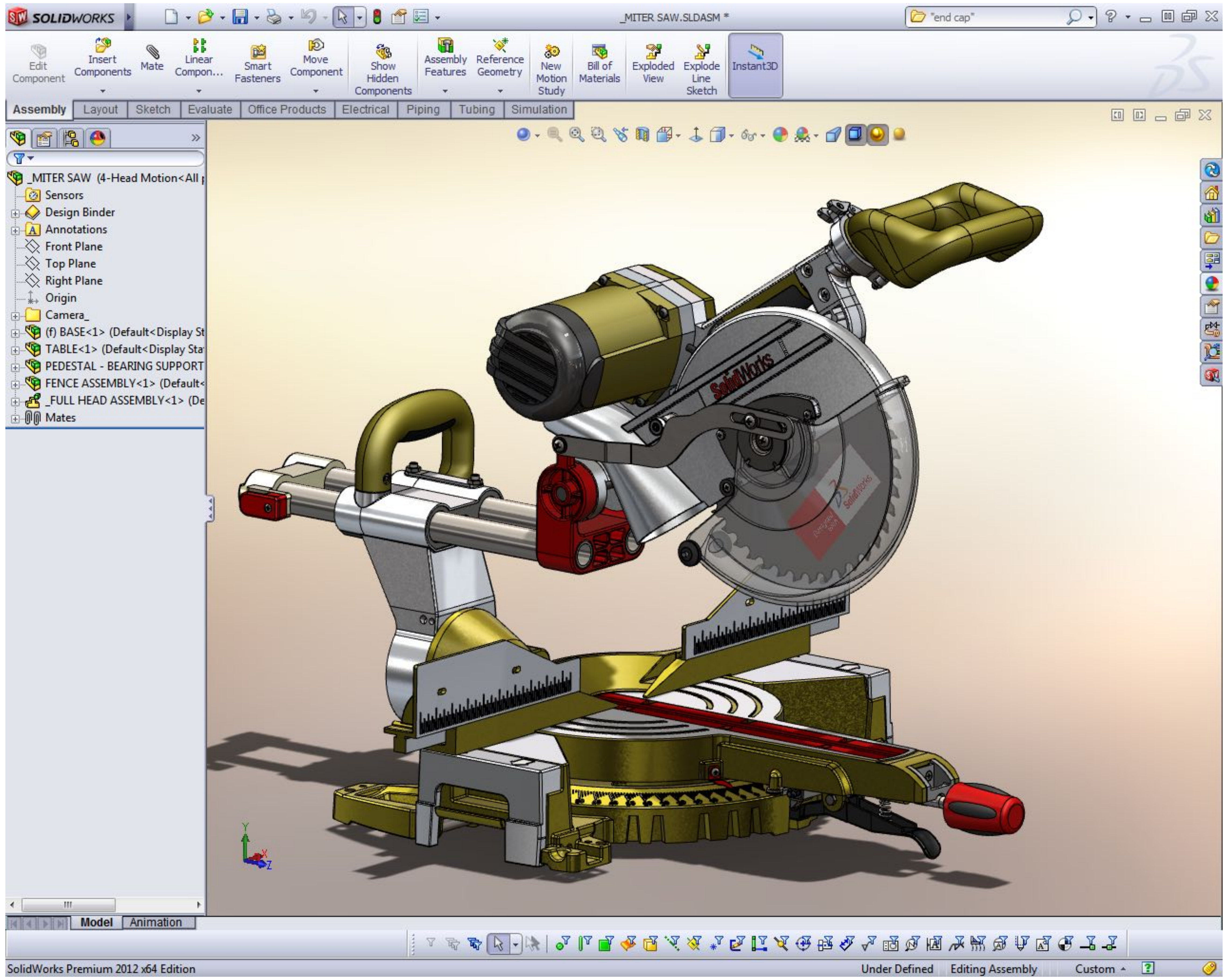
Cameras everywhere

Computational cameras

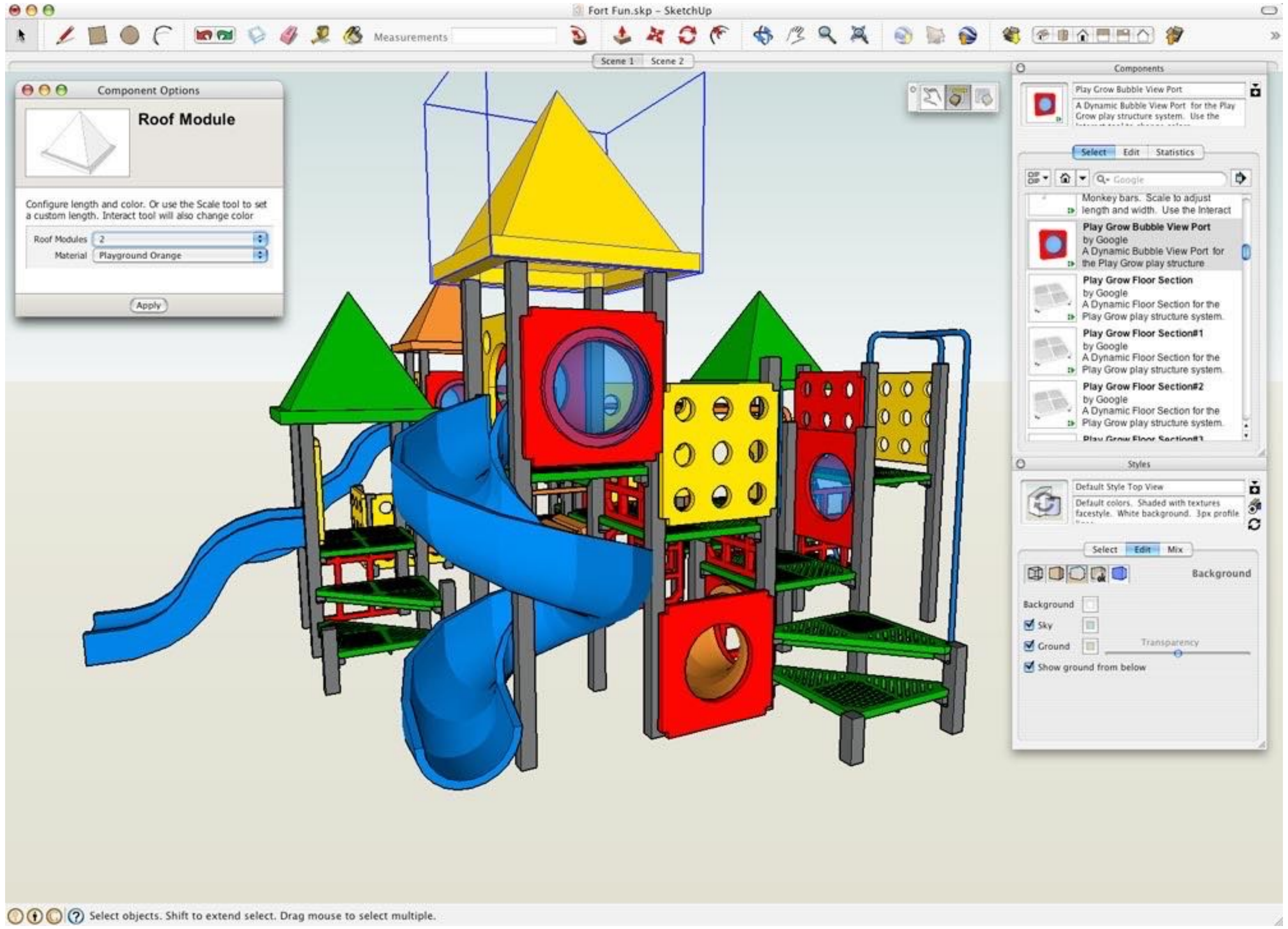
Turning collections of images into 3D worlds

Kerbl et al. 2023

Computer aided design



SolidWorks



SketchUp

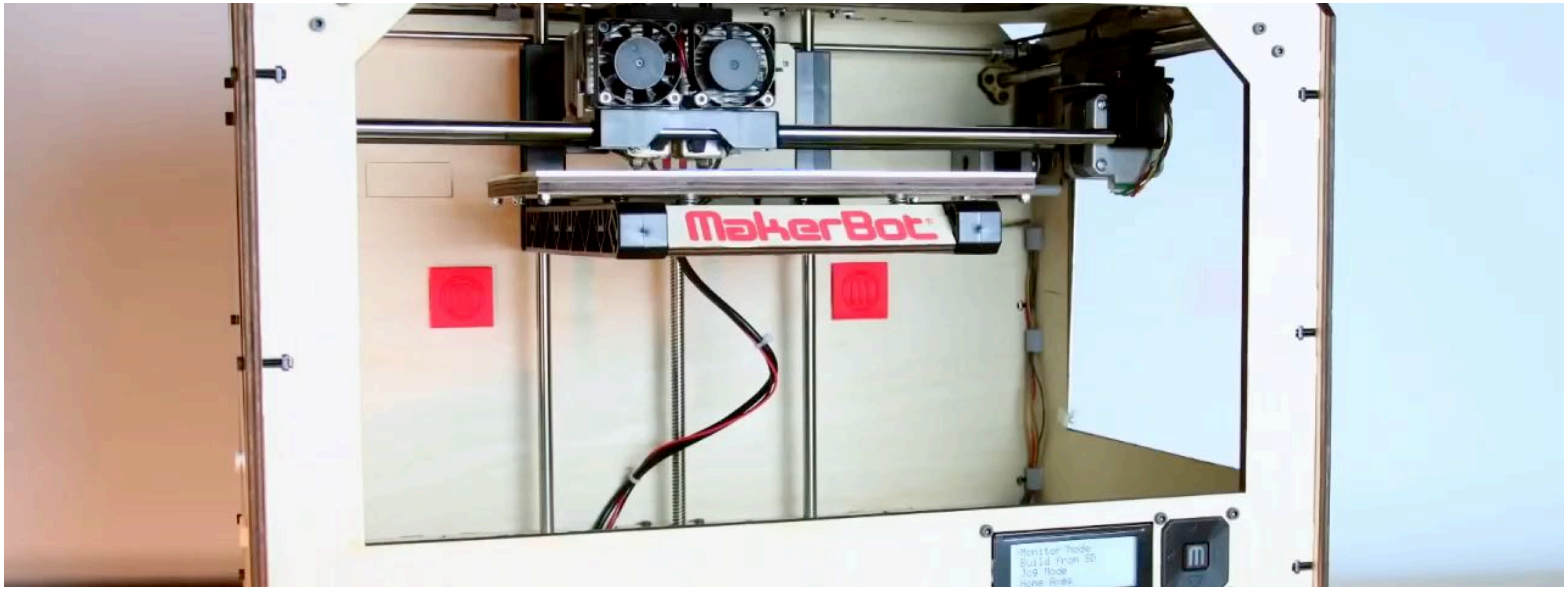
For mechanical, architectural, electronic, optical, …

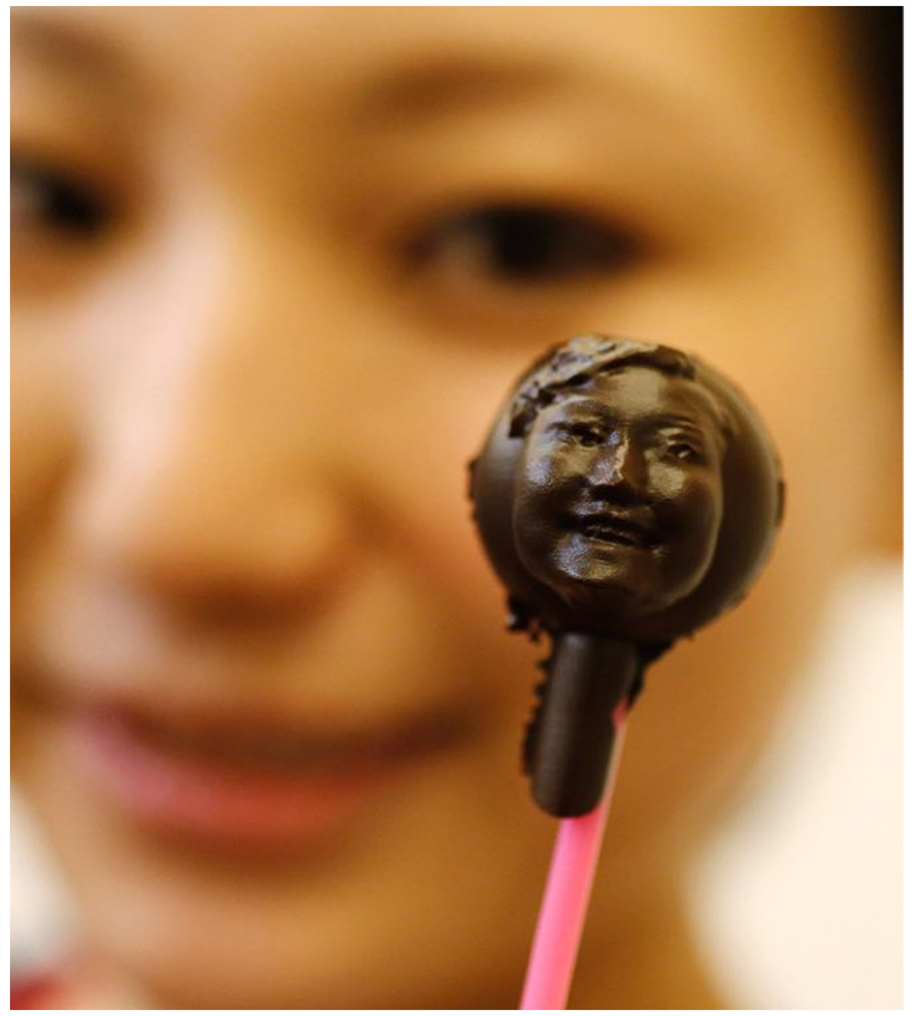
Product design and visualization

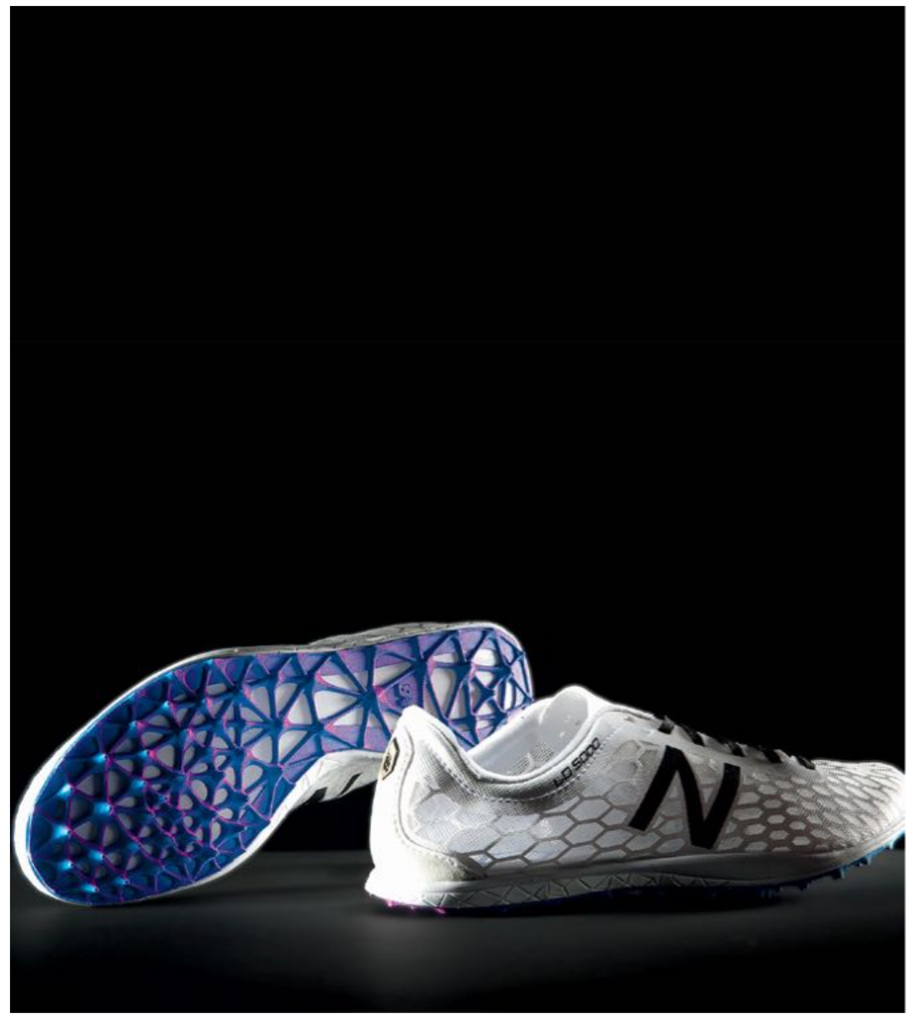
Ikea - 75% of catalog is rendered imagery (several years ago… likely a lot more now)

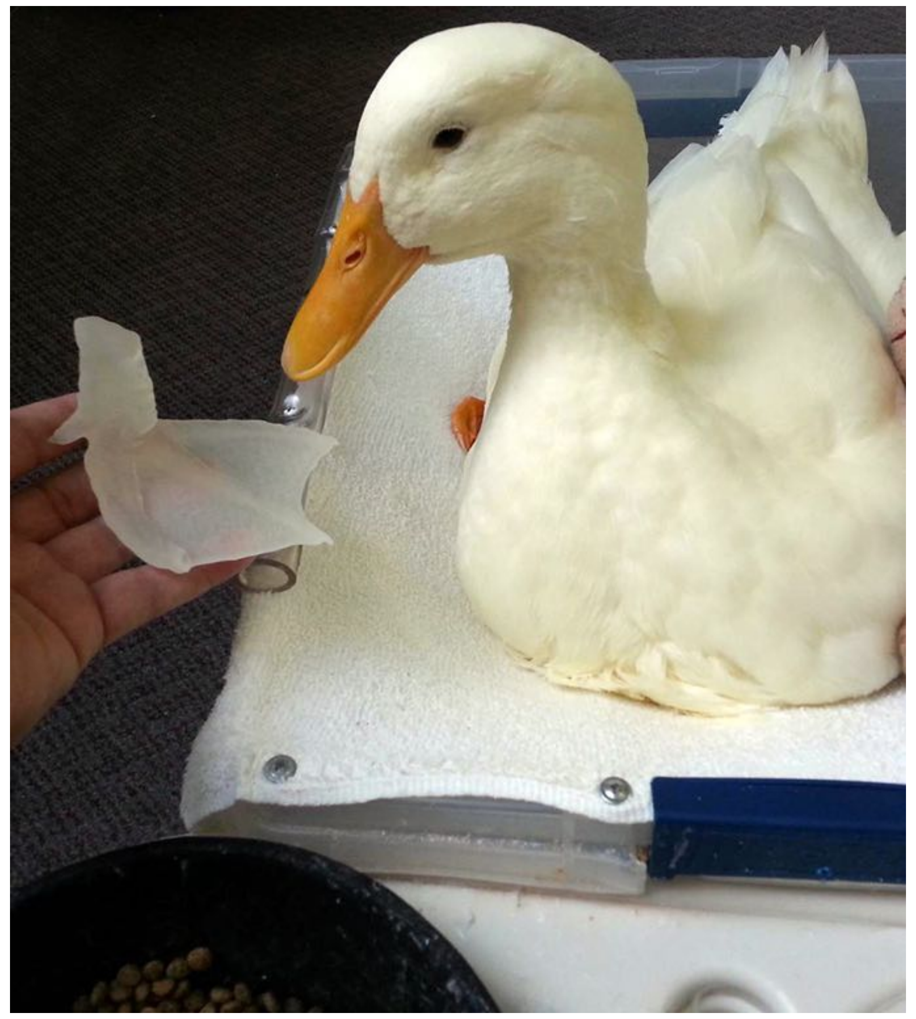
Architectural design

3D fabrication

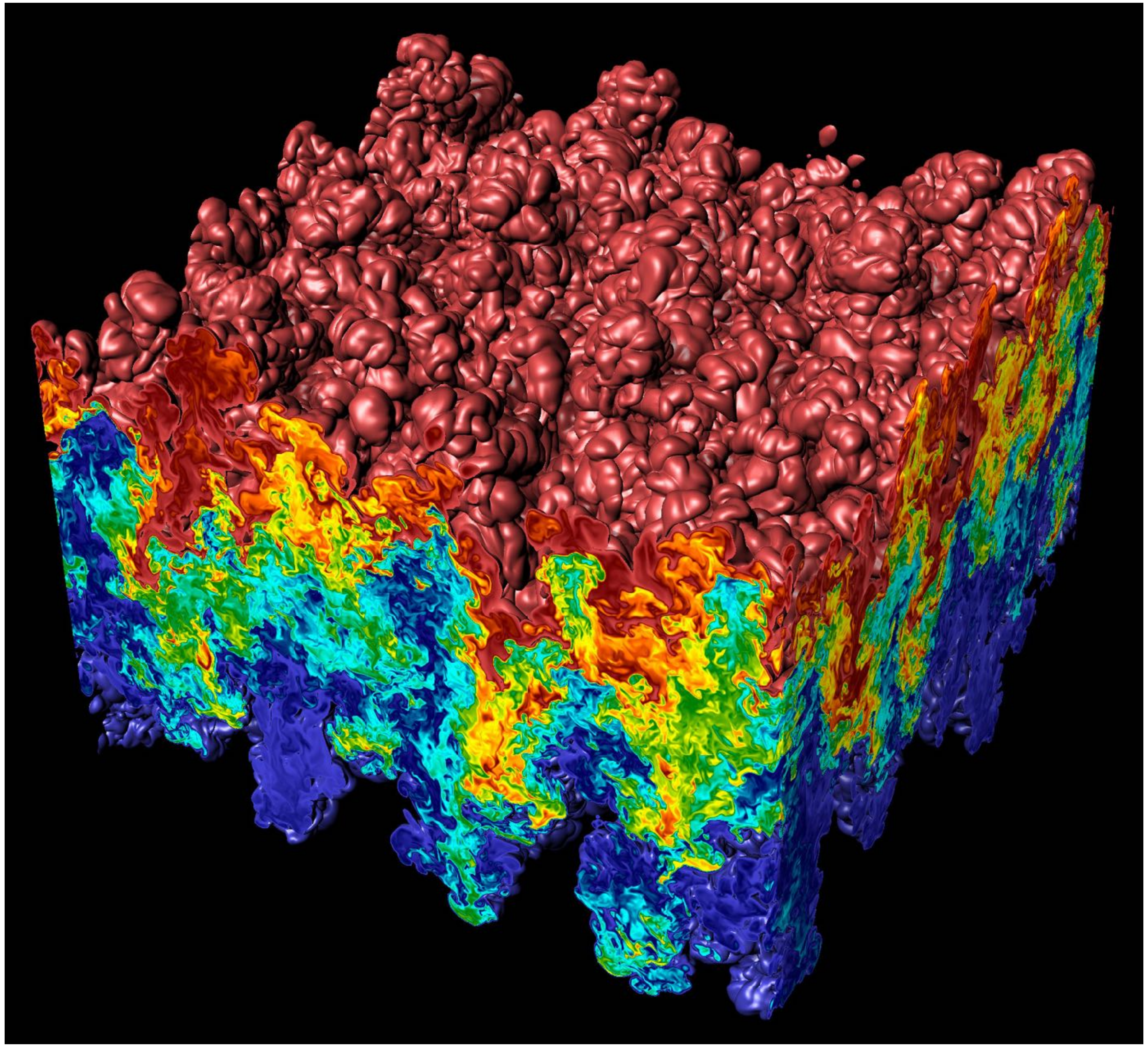








Data visualization



Science, engineering, medicine, journalism, ...

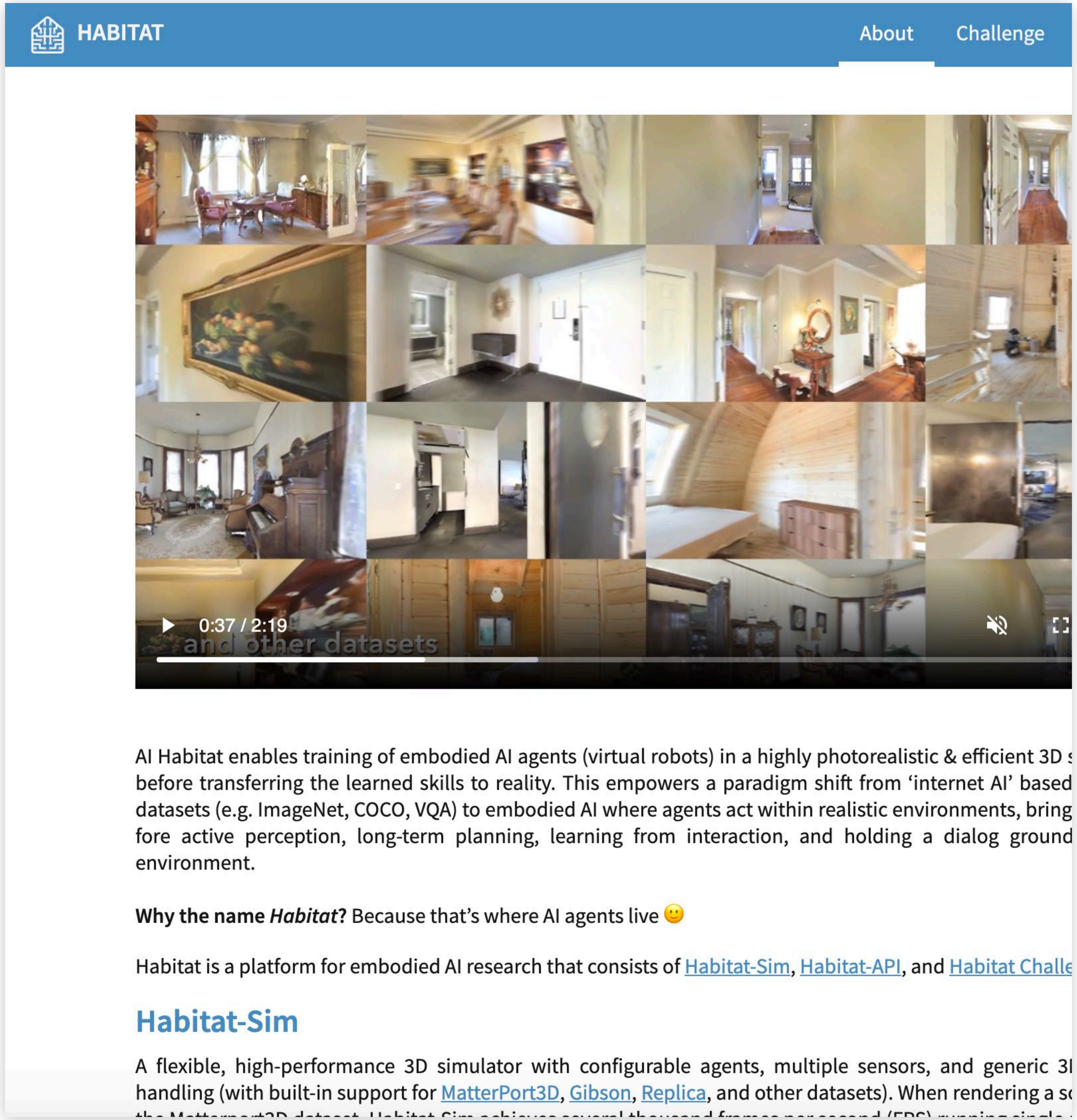
Simulation

Driving simulator
Toyota Higashifuji Technical Center

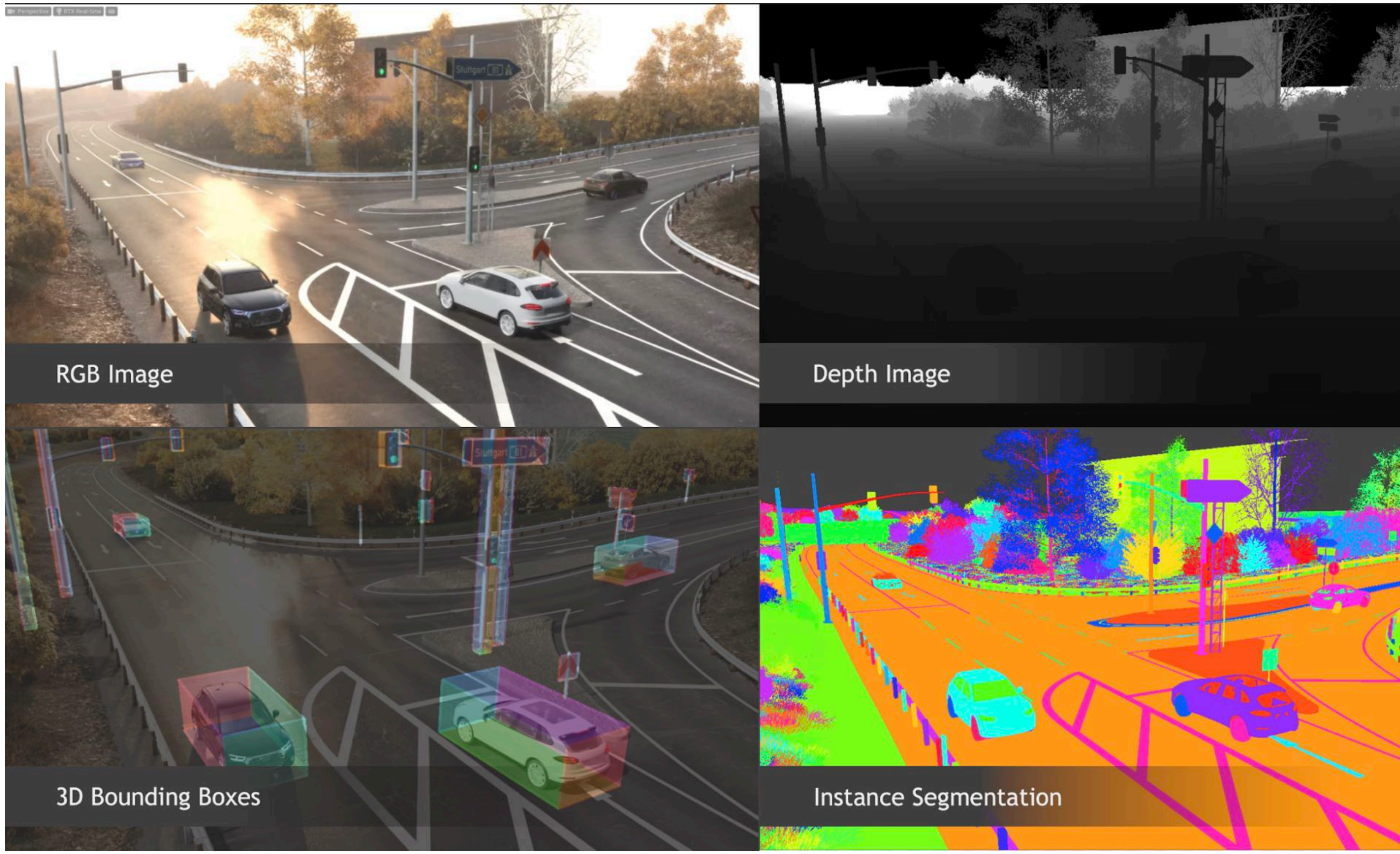
da Vinci surgical robot
Intuitive Surgical

Flight simulator, driving simulator, surgical simulator, . . .

Graphics/simulation used for training ML models



AI Habitat:
simulator for training AI agents



NV Drive Sim:
autonomous driving simulator

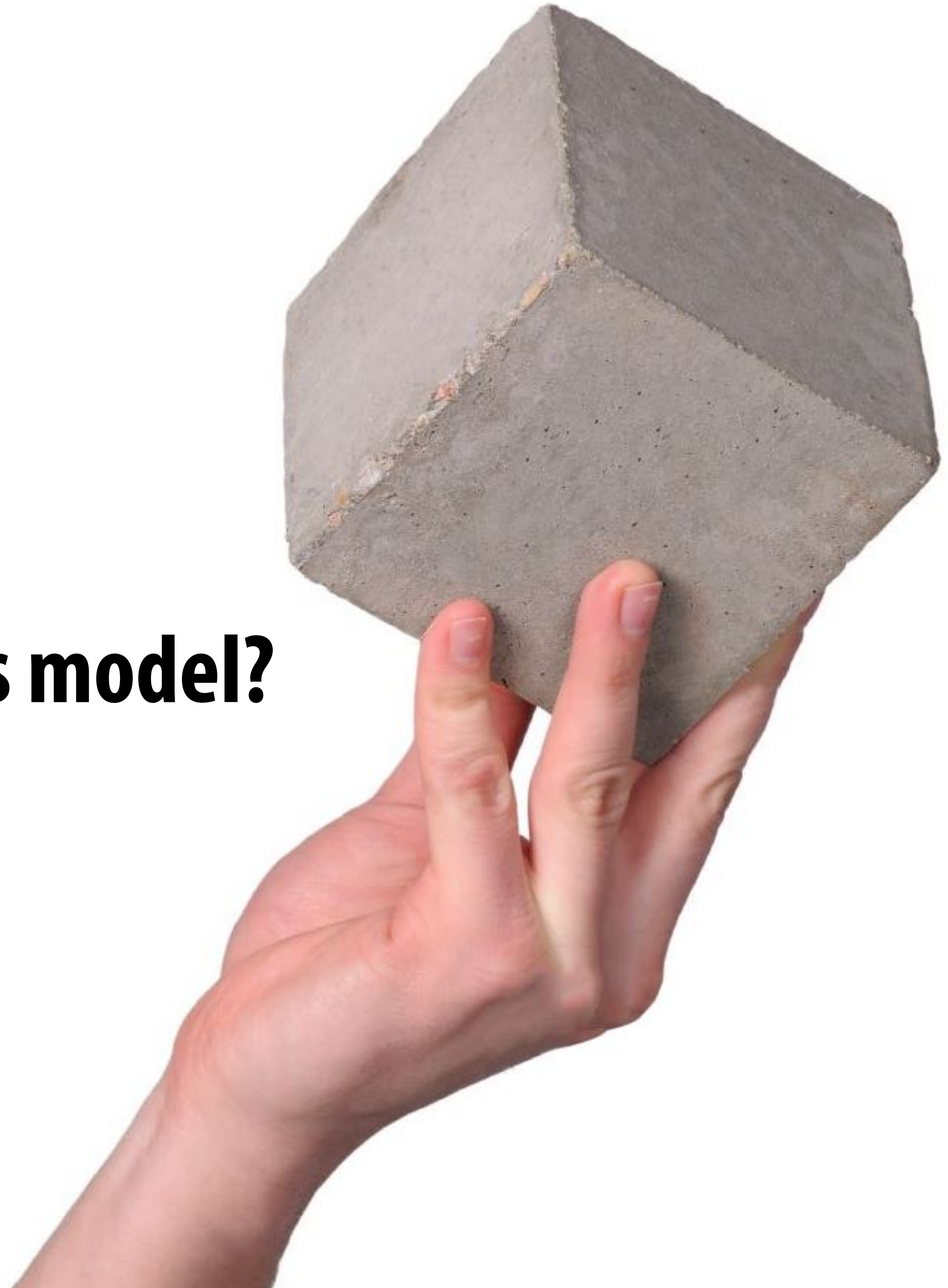
Transformative generative AI capabilities

"A bento box with rice,
edamame, ginger,
and sushi.
Top down view,
white background.
Sushi in right bin of bento box.
Edamame in top left."

Foundations of computer graphics

- All these applications demand *sophisticated* theory and systems
- Science and mathematics
  - Physics of light, color, optics
  - Math of curves, surfaces, geometry, perspective, …
  - Sampling
- Systems
  - Parallel, heterogeneous processing
  - Graphics-specific programming systems
  - Input/output devices
- Art and psychology
  - Perception: color, stereo, motion, image quality, …
  - Art and design: composition, form, lighting, ...

ACTIVITY: modeling and drawing a cube



- Goal: generate a realistic drawing of a cube
- Key questions:
  - *Modeling:* how do we describe the cube?
  - *Rendering:* how do we then visualize this model?

ACTIVITY: modeling the cube

- **Suppose our cube is...**
  - **centered at the origin (0,0,0)**
  - **has dimensions 2 x 2 x 2**
- **QUESTION: What are the coordinates of the cube 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 )
```

- **QUESTION: What about the edges?**

```
AB, CD, EF, GH,
AC, BD, EG, FH,
AE, CG, BF, DH
```

ACTIVITY: drawing the cube

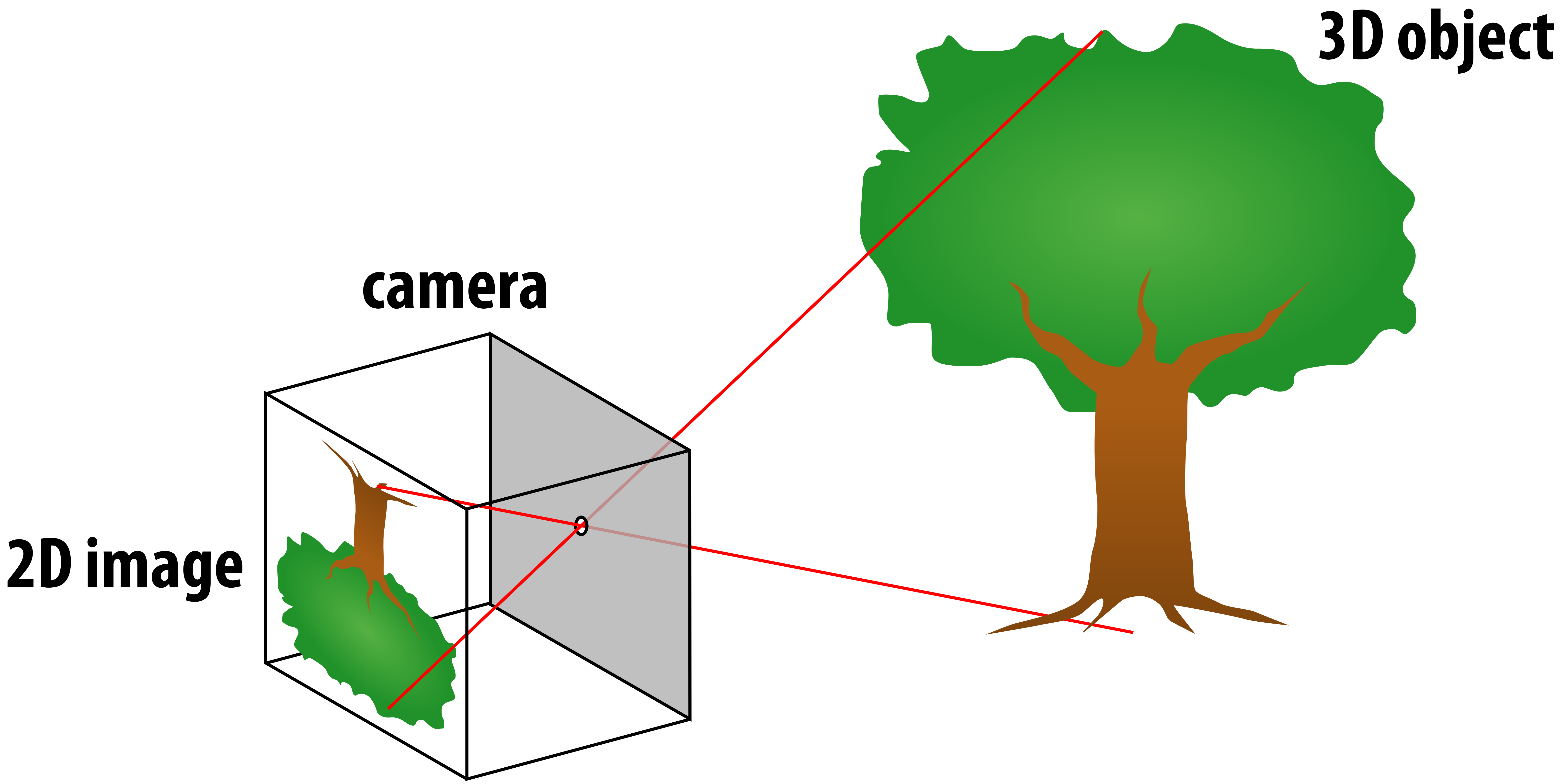
- Now have a digital description of the geometry of the cube:

```
VERTICES                                      EDGES
A: ( 1, 1, 1 )   E: ( 1, 1,-1 )
B: (-1, 1, 1 )   F: (-1, 1,-1 )        AB, CD, EF, GH,
C: ( 1,-1, 1 )   G: ( 1,-1,-1 )        AC, BD, EG, FH,
D: (-1,-1, 1 )   H: (-1,-1,-1 )        AE, CG, BF, DH
```

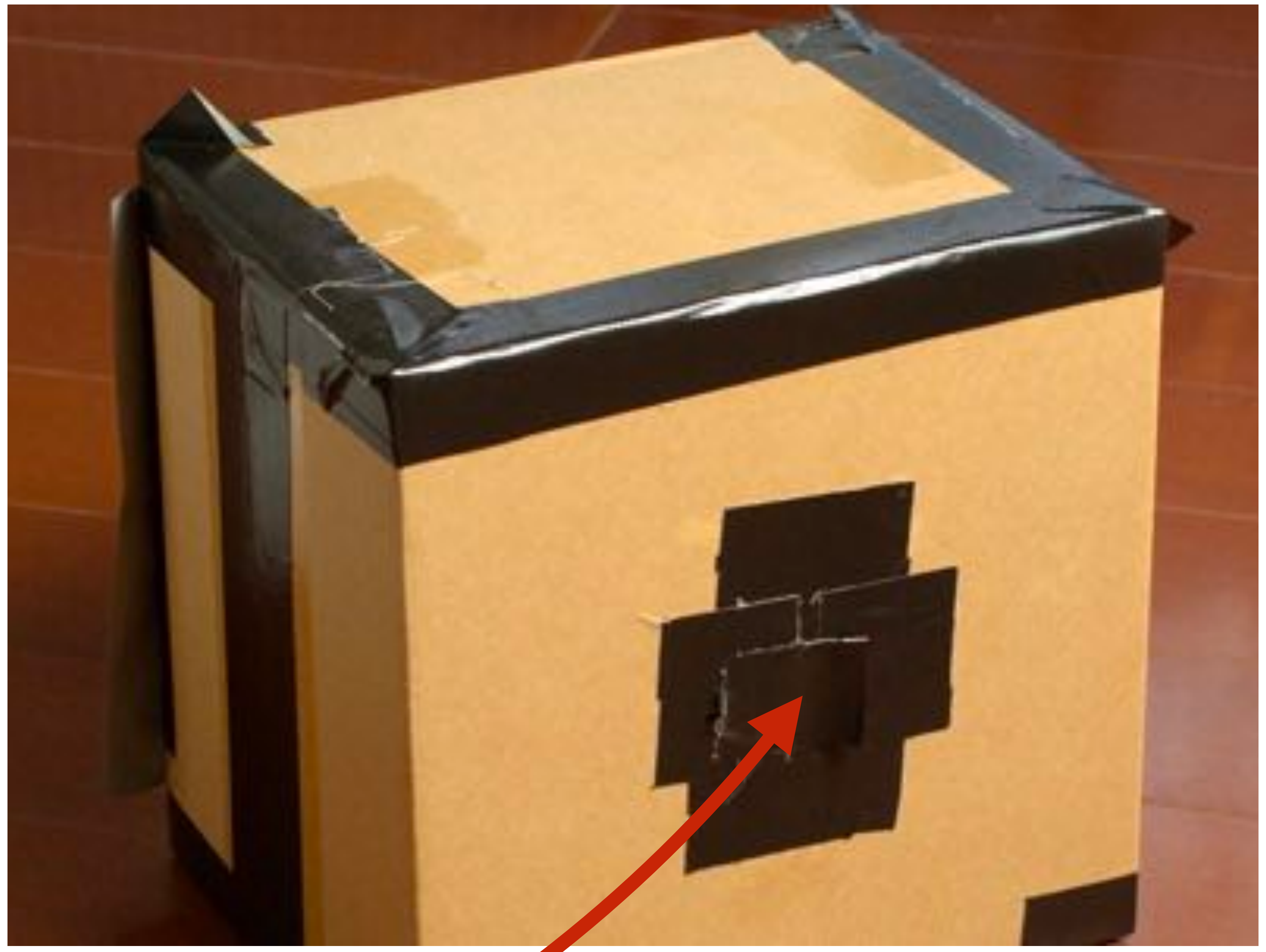
- How do we draw this 3D cube as a 2D (flat) image?

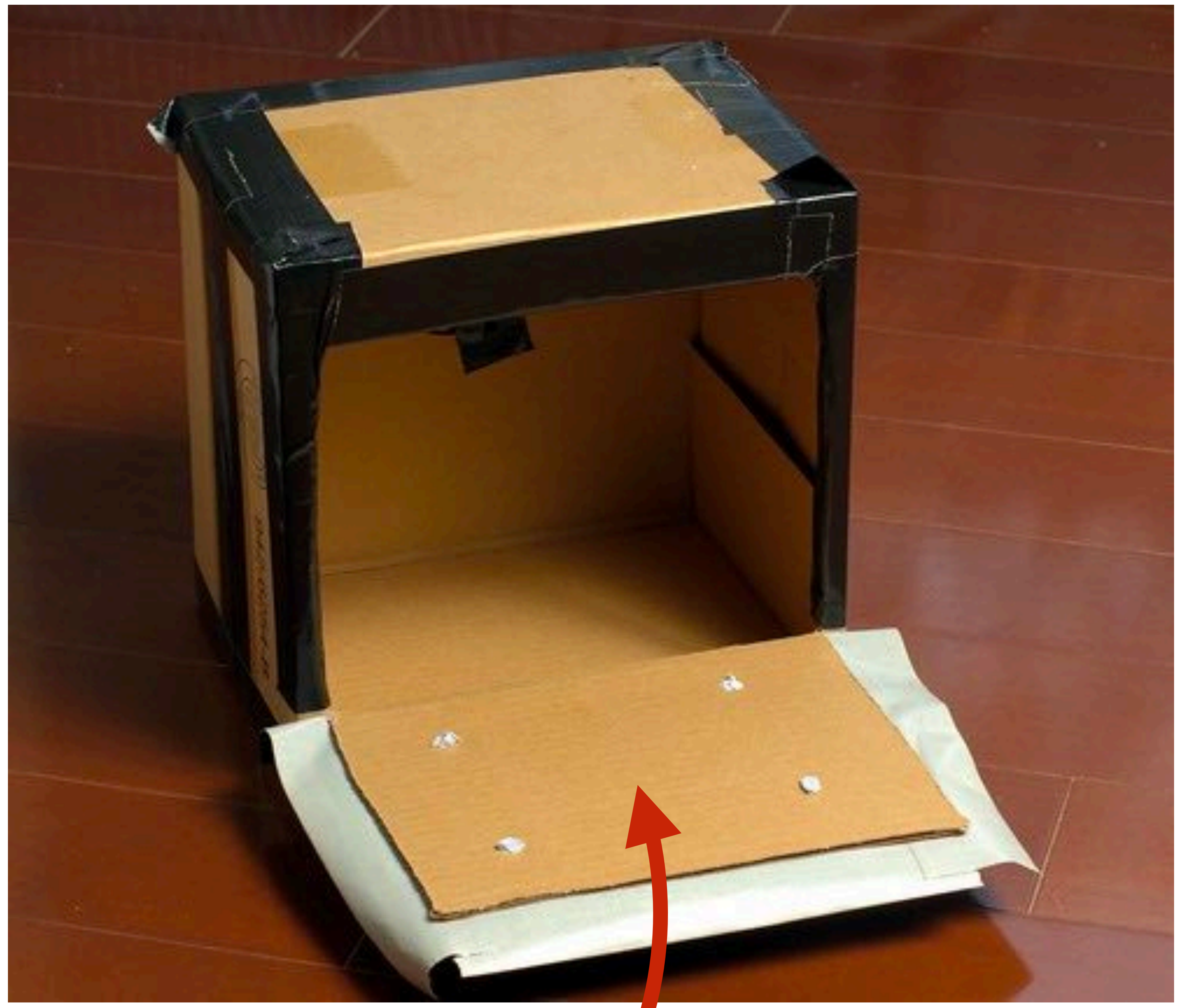
Perspective projection

- Objects look smaller as they get further away ("perspective")
- Why does this happen?
- Consider simple ("pinhole") model of a camera:



For those that didn’t do this in grade school





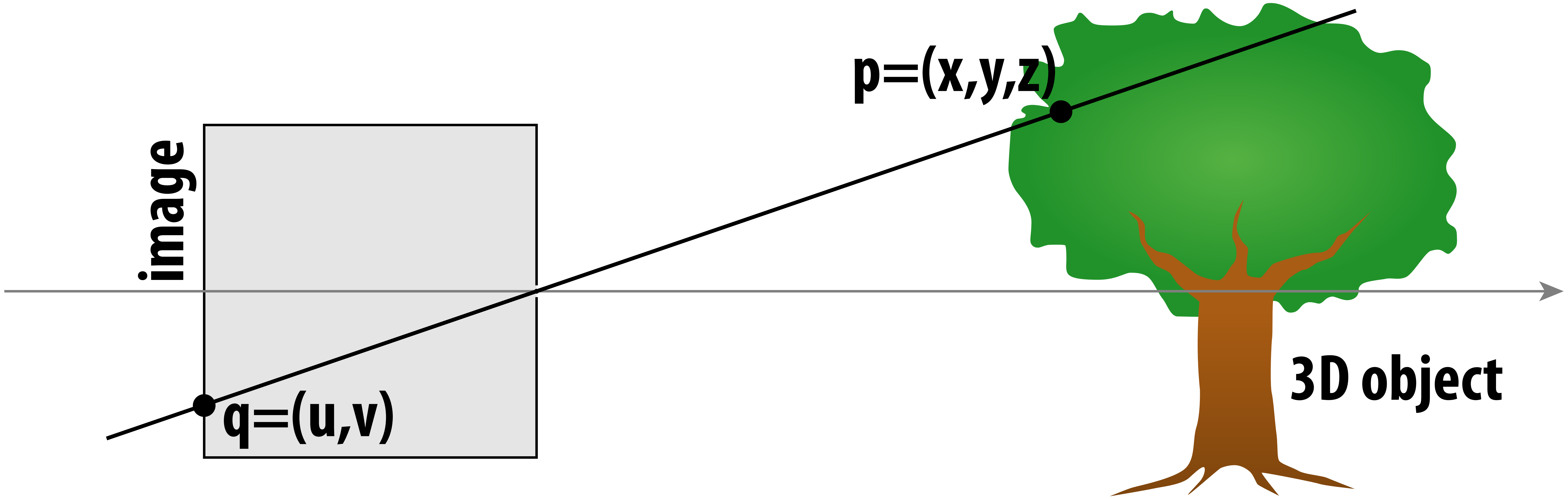
Pin hole

Place photosensitive paper here

http://janneinosaka.blogspot.com/2010/03/pinhole-time.html

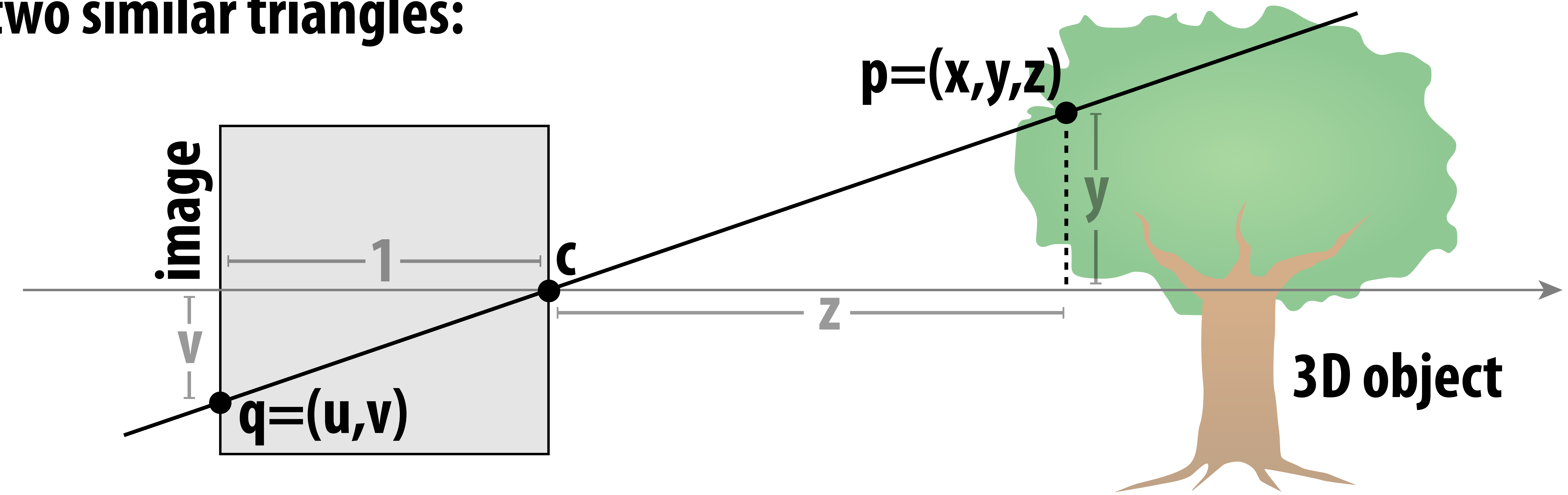
Perspective projection: side view

- Where exactly does a point p = (x,y,z) end up on the image?
- Let's call the image point q=(u,v)



Perspective projection: side view

- Where exactly does a point p = (x,y,z) end up on the image?
- Let's call the image point q=(u,v)
- Notice two similar triangles:

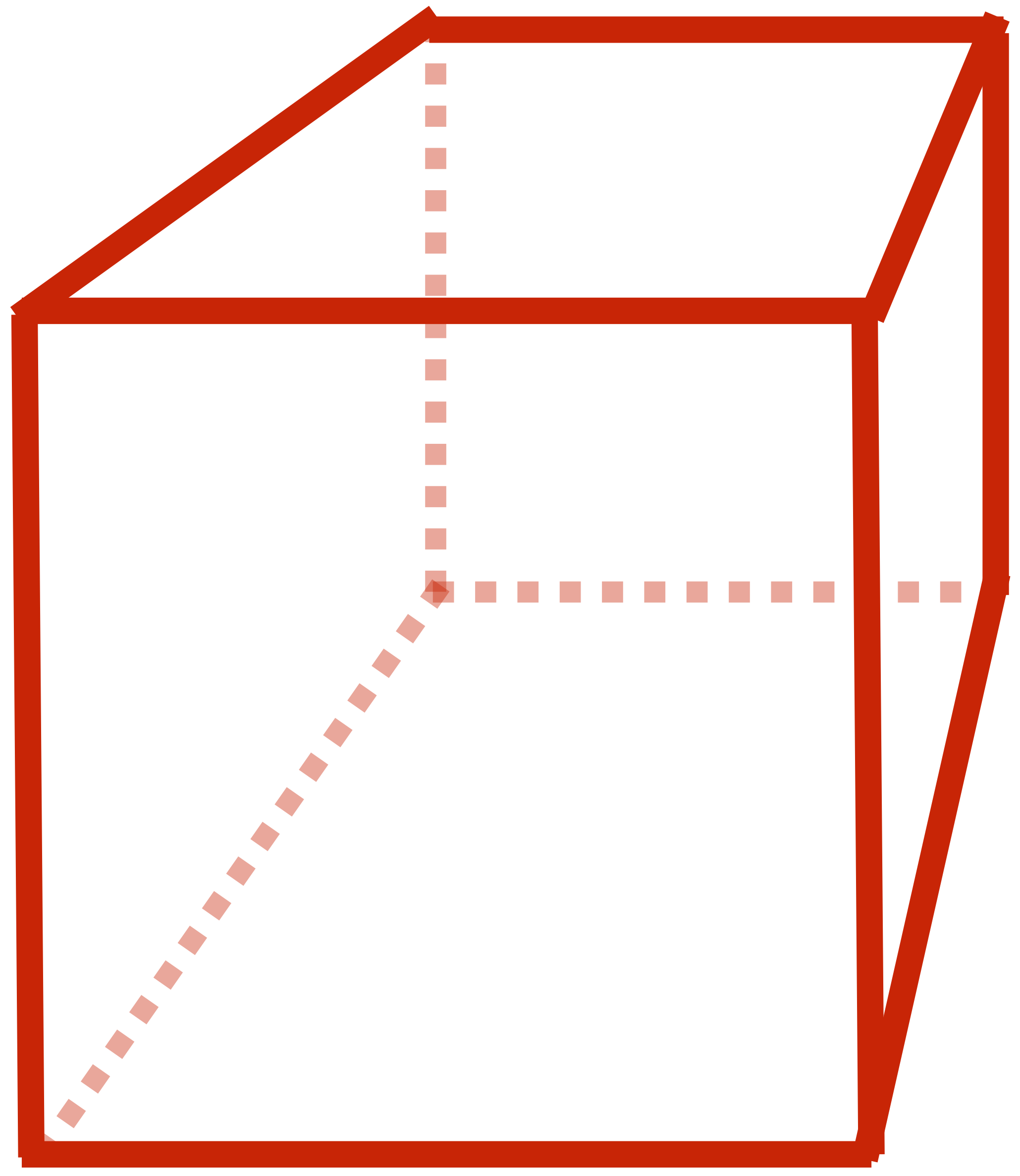


- Assume camera has unit size, coordinates relative to pinhole c
- Then $v/1 = y/z \ldots v = y/z$
- Likewise, horizontal offset $u = x/z$

Can you visualize what it should look like?

- **Consider a cube with these vertices:**

```
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 )

EDGES

AB, CD, EF, GH,
AC, BD, EG, FH,
AE, CG, BF, DH
```



Self-check

- **Now imagine a camera positioned at (2,3,5) looking at the cube… can you picture what it should look like?**

ACTIVITY: draw image made by pinhole camera

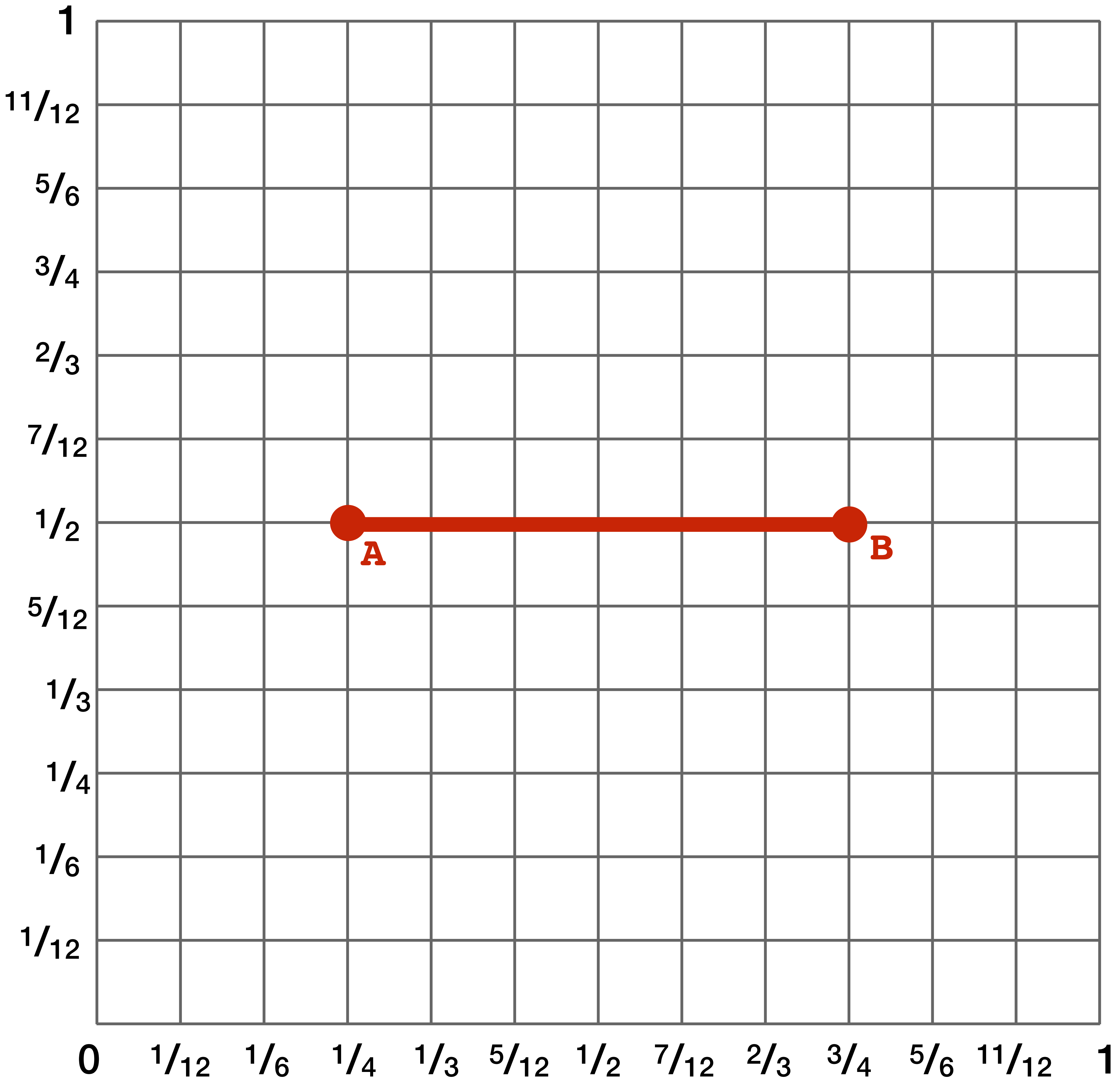
- Pick two vertices that share an edge and do it yourself!
  - Let's assume camera is at point c=(2,3,5)
  - Convert (X,Y,Z) of both endpoints of cube edge to screen point (u,v):
    1. Subtract camera point c from vertex (X,Y,Z) to get (x,y,z)
    2. Divide x and y by z to get (u,v)—*write as a fraction*
  - Then draw a line between (u1,v1) and (u2,v2) for all edges

Vertex position in absolute world coordinates

Vertex position relative to camera

```
VERTICES                                  EDGES
A: ( 1, 1, 1 )    E: ( 1, 1,-1 )
B: (-1, 1, 1 )    F: (-1, 1,-1 )          AB, CD, EF, GH,
C: ( 1,-1, 1 )    G: ( 1,-1,-1 )          AC, BD, EG, FH,
D: (-1,-1, 1 )    H: (-1,-1,-1 )          AE, CG, BF, DH
```

Render a cube!



- Assume camera is at point c=(2,3,5)
- Convert (X,Y,Z) of both endpoints of edge to (u,v):
  1. Subtract camera c from vertex (X,Y,Z) to get (x,y,z)
  2. Divide x and y by z to get (u,v)
- Draw line between (u1,v1) and (u2,v2)

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 )
```

EDGES

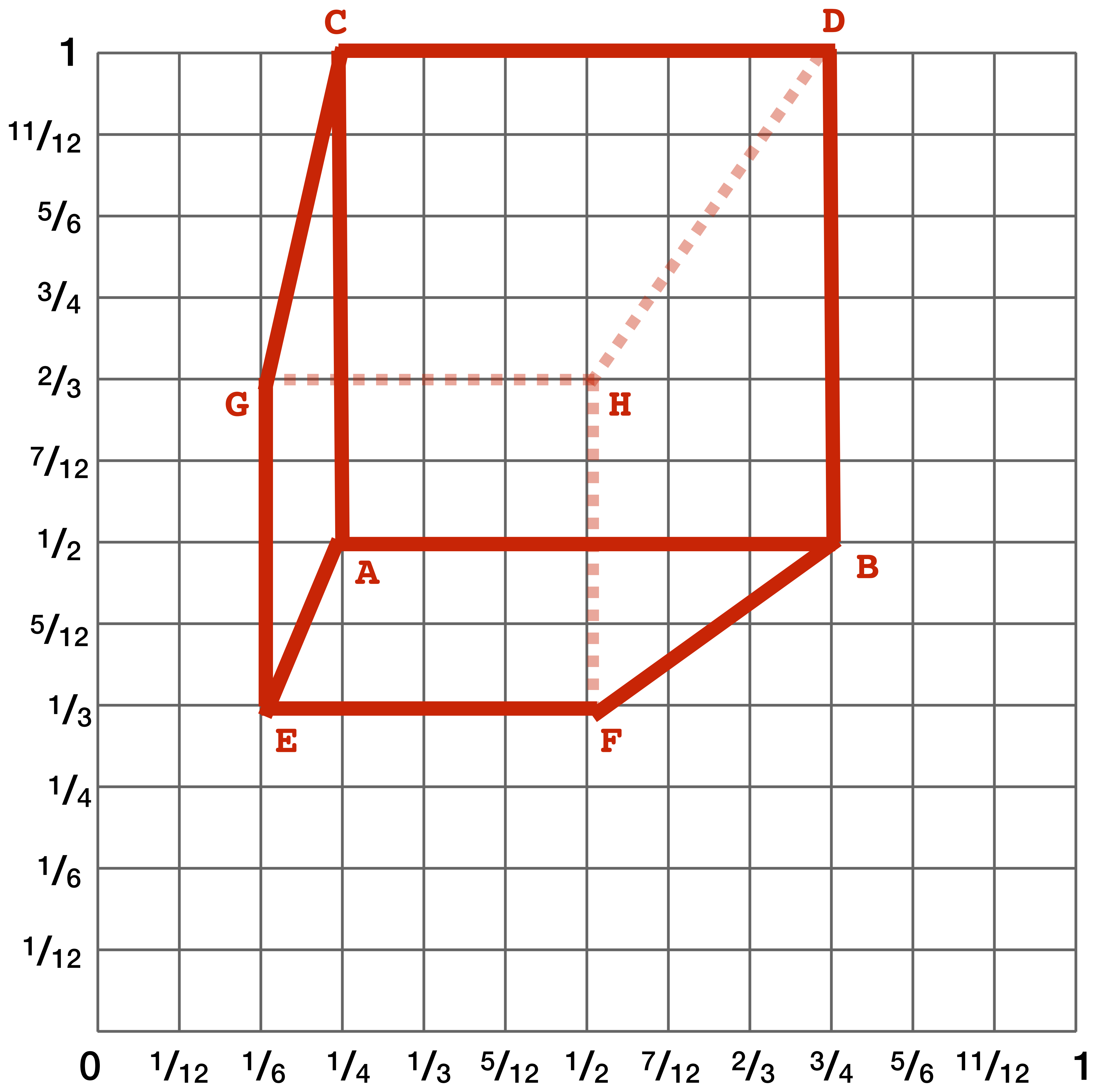
```
AB, CD, EF, GH,
AC, BD, EG, FH,
AE, CG, BF, DH
```

Projected coordinates:

A: (1/4, 1/2)
B: (3/4, 1/2)

How did we do?

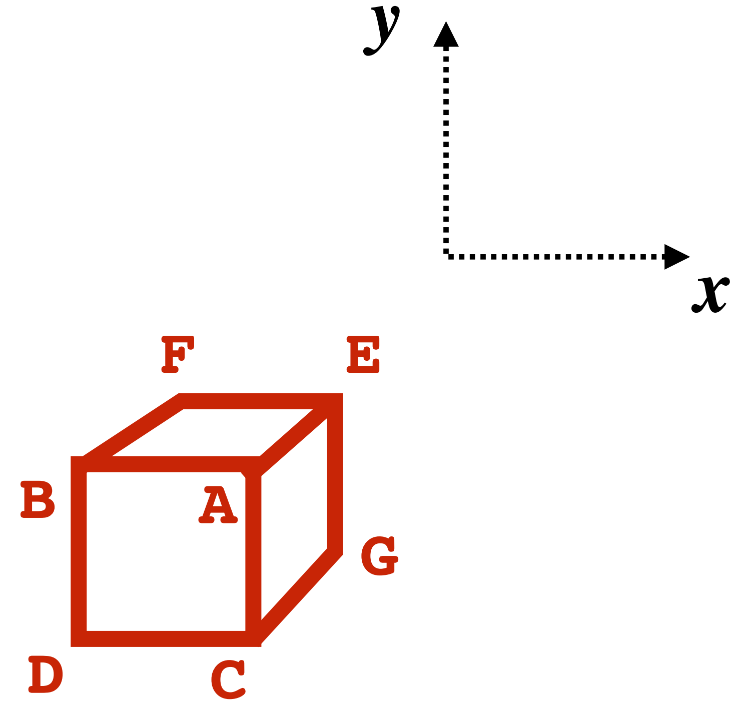
Recall: camera at (2,3,5), looking in -Z direction, cube centered at origin



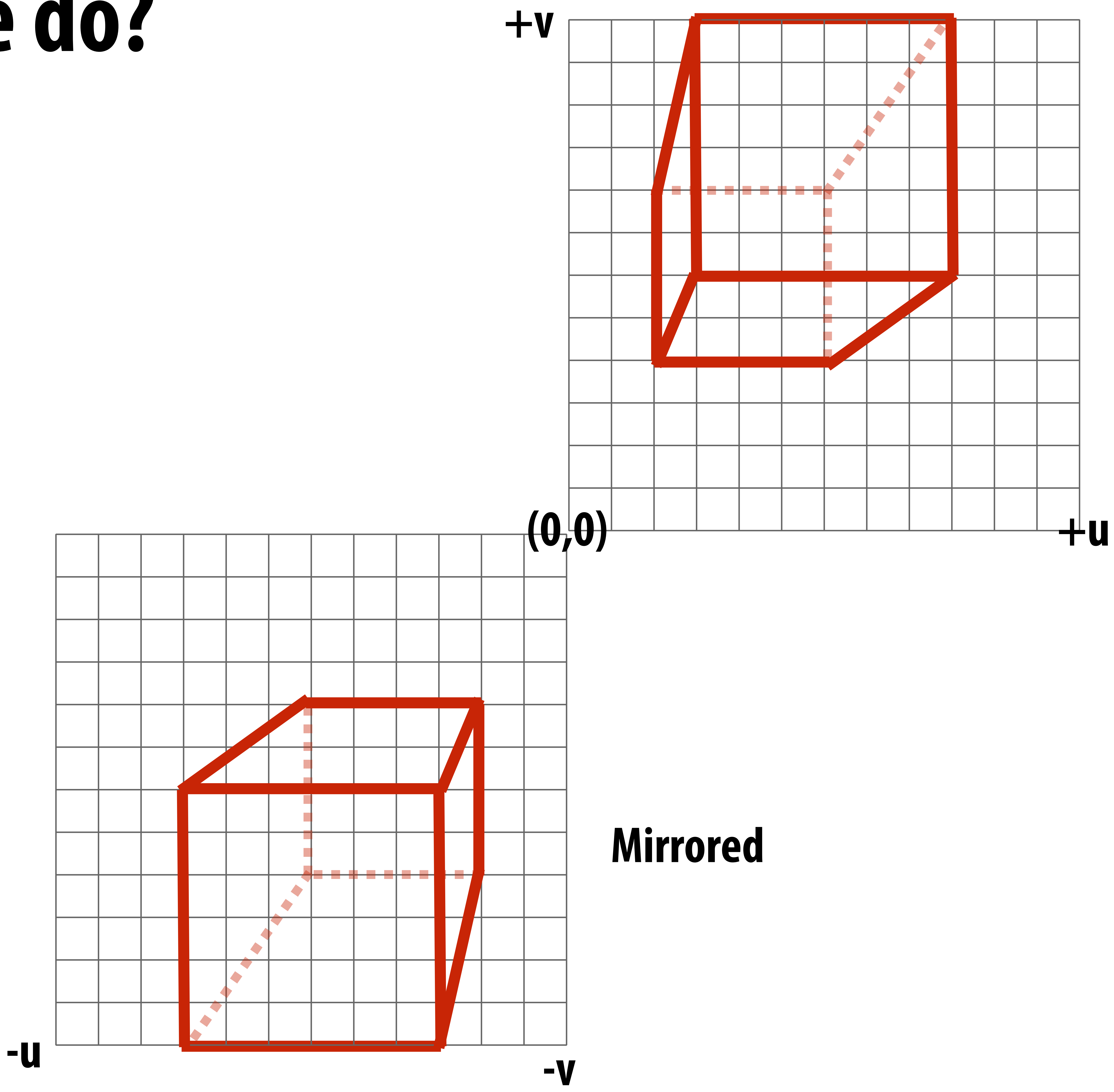
2D coordinates (after projection):

```
A: (1/4, 1/2)
B: (3/4, 1/2)

C: (1/4, 1)
D: (3/4, 1)

E: (1/6, 1/3)
F: (1/2, 1/3)

G: (1/6, 2/3)
H: (1/2, 2/3)
```

Keep in mind, this image is mirrored since it is a pinhole projection. Mirror the result about the origin (0,0) and you get…

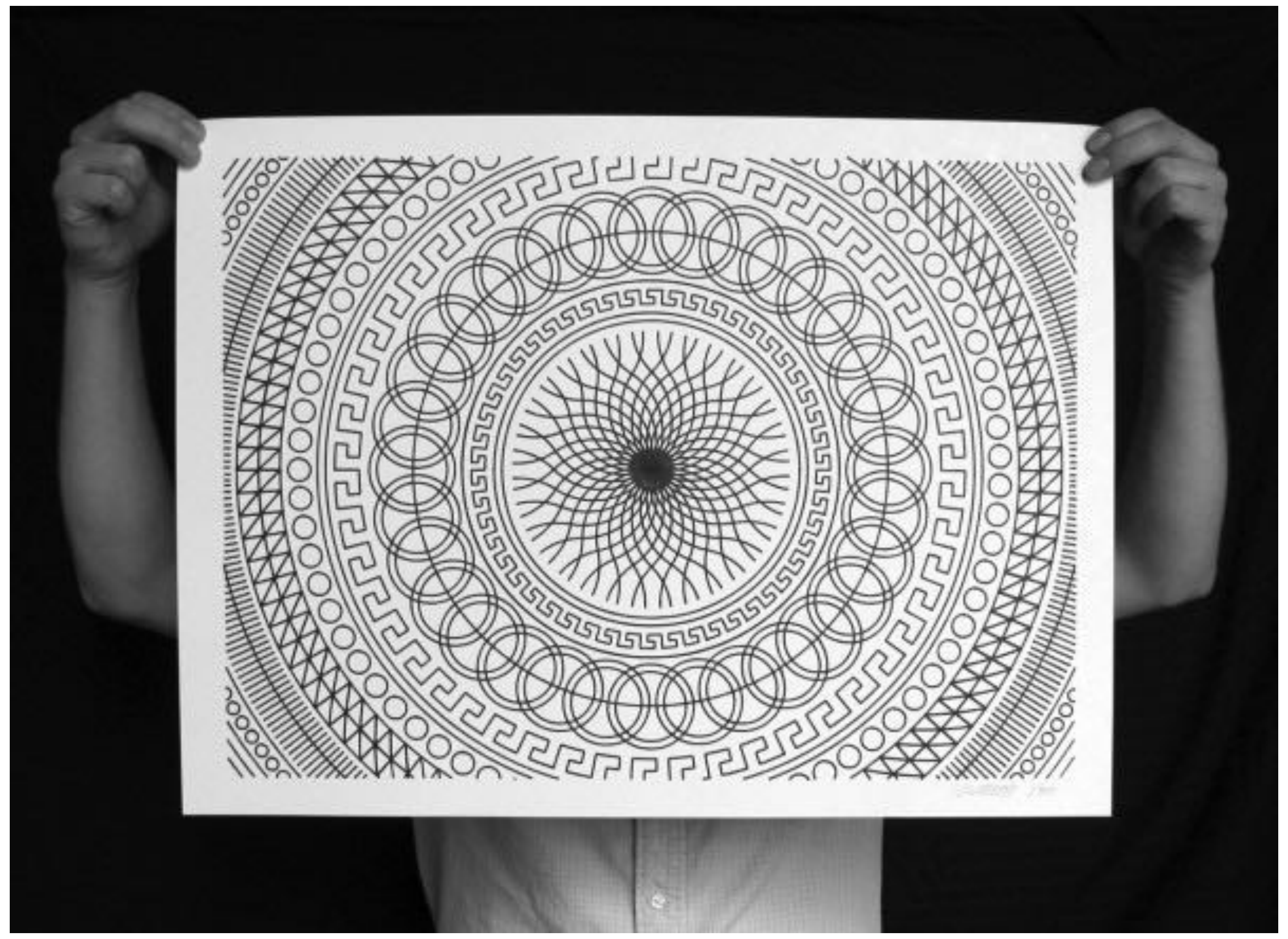


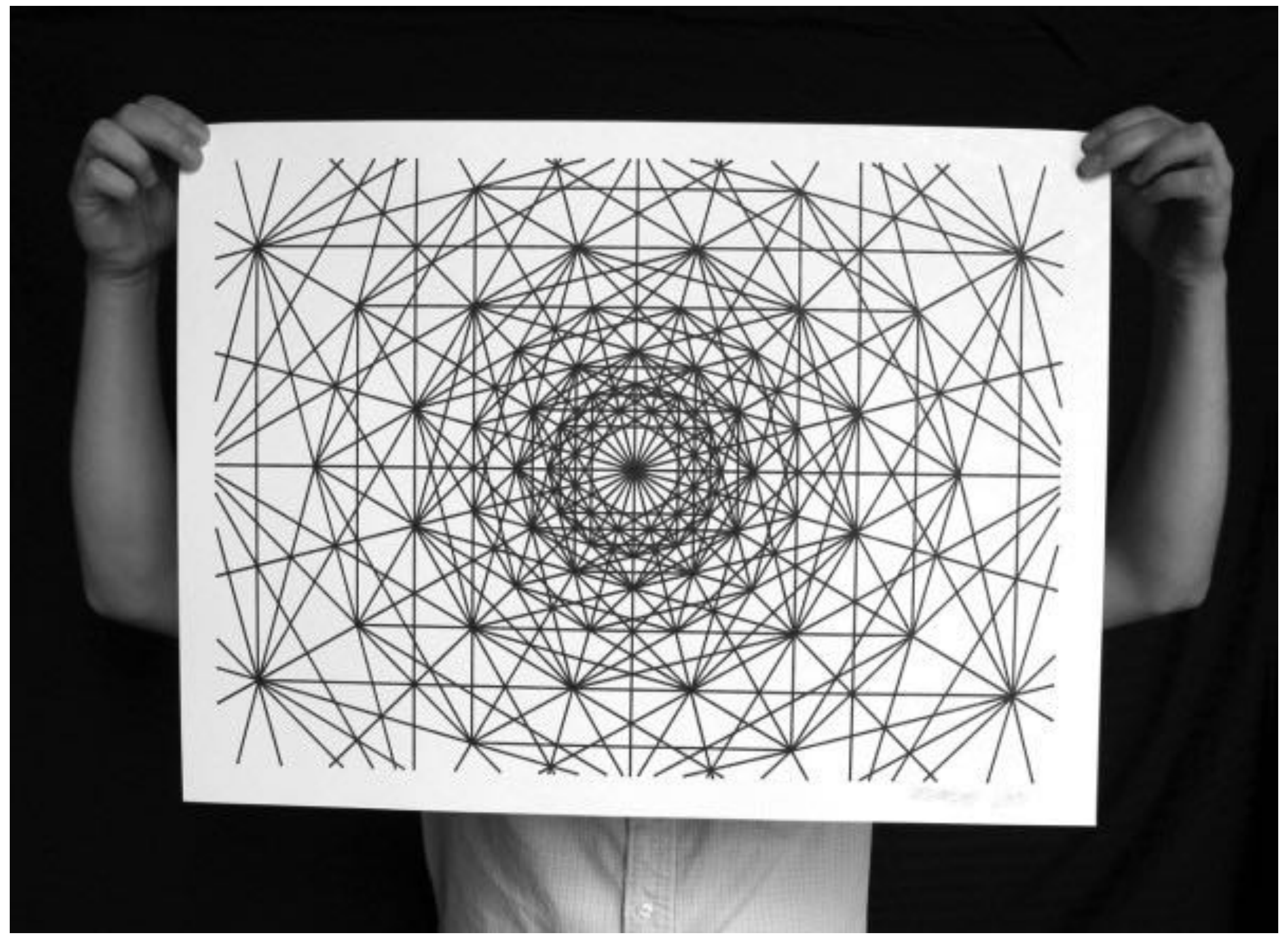
How did we do?



But wait…
How do we draw lines on a computer?

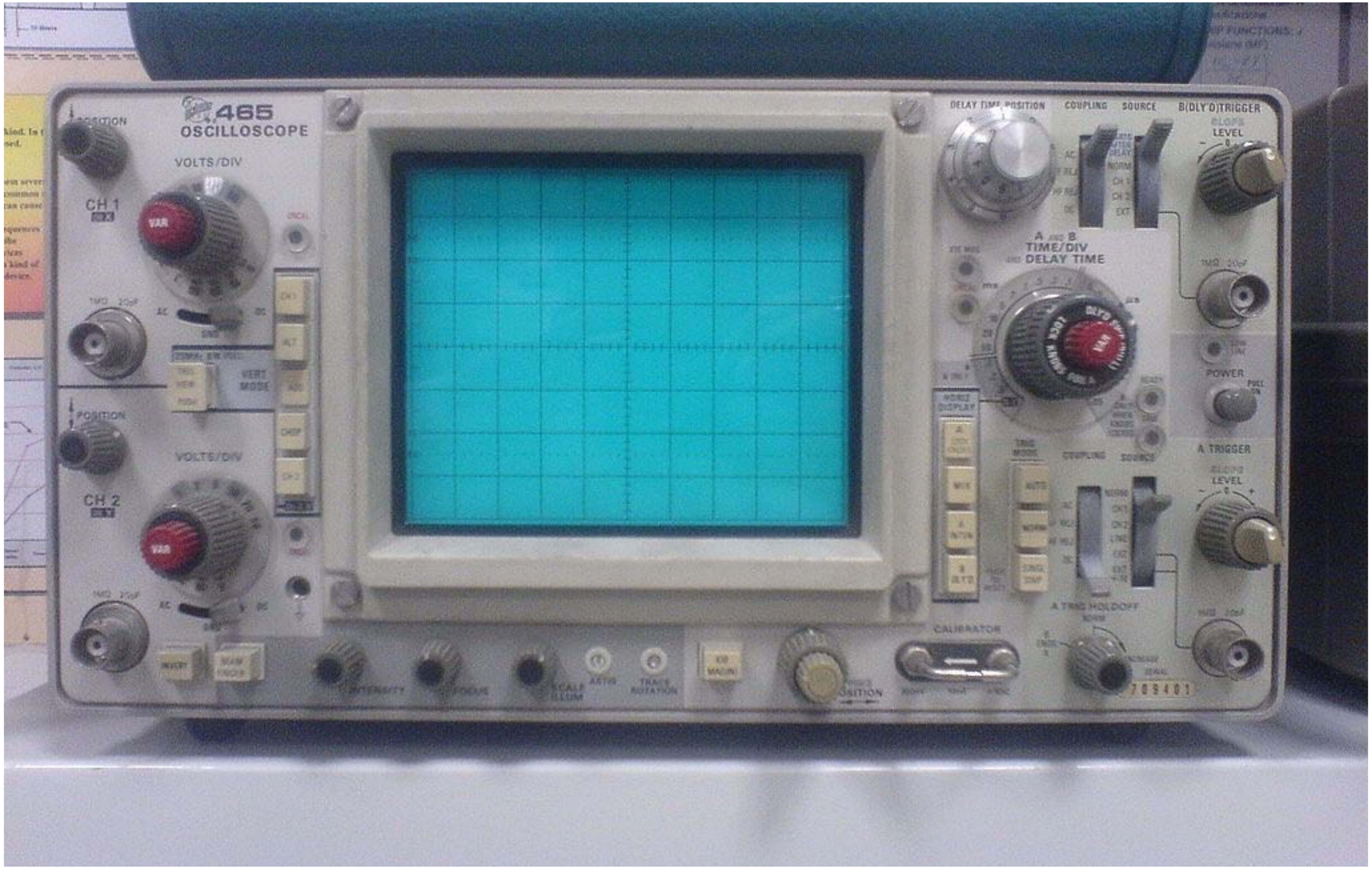
CNC sharpie drawing machine ;-)



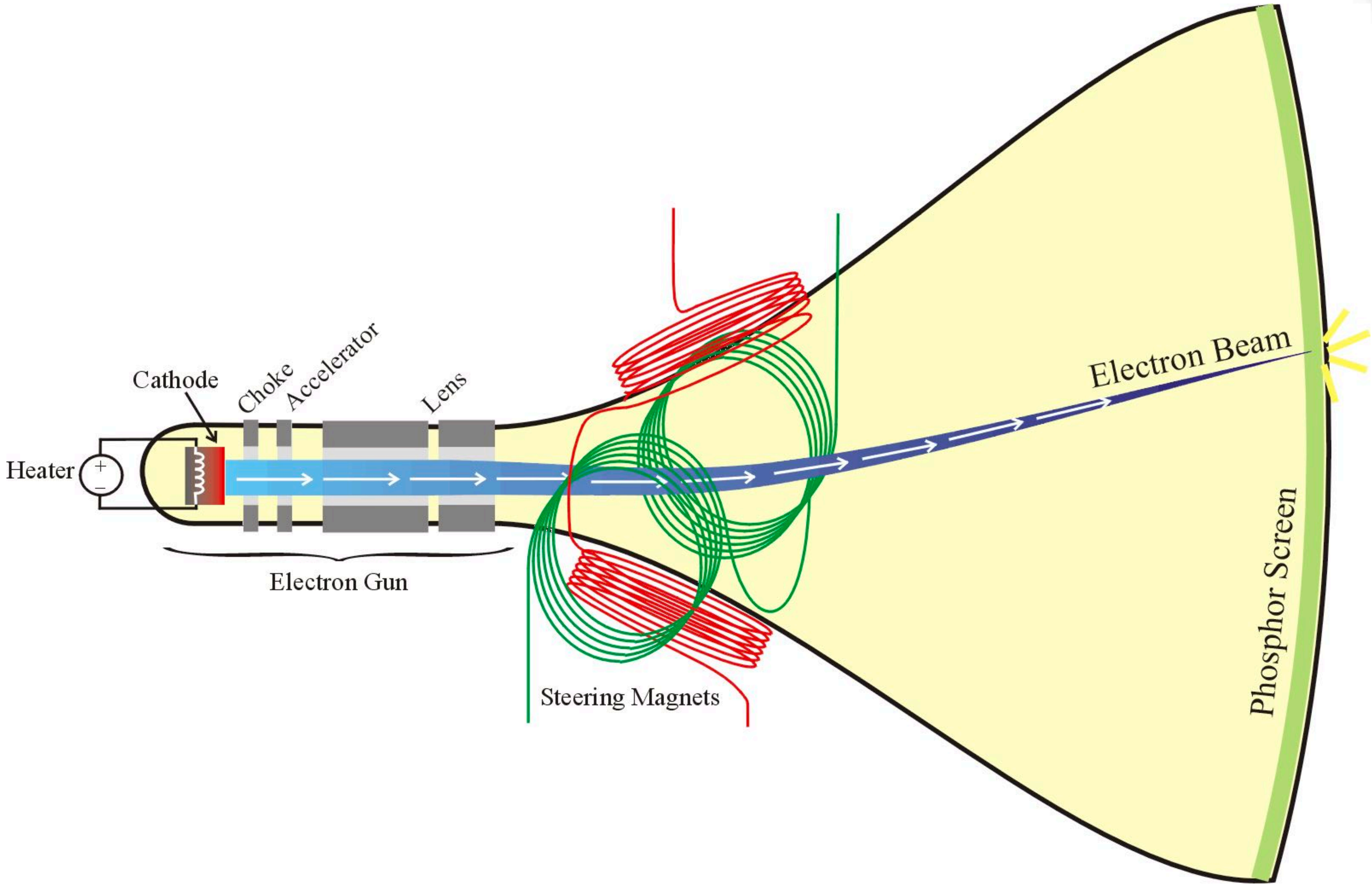


http://44rn.com/projects/numerically-controlled-poster-series-with-matt-w-moore/

Oscilloscope

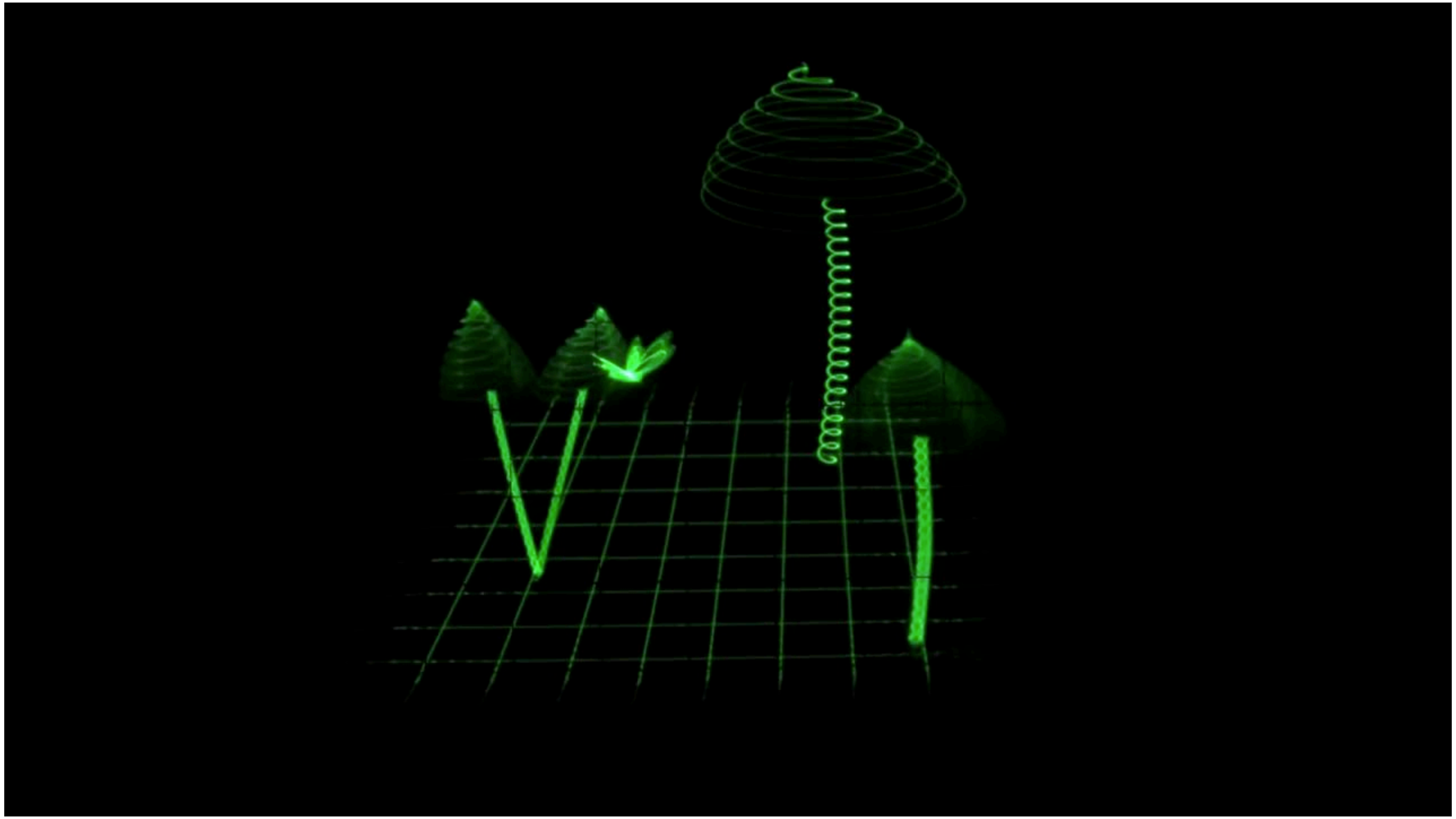


Cathode ray tube



[Credit: http://propagation.ece.gatech.edu/ECE3025/tutorials/CathodeRayTube/CRToverview.htm]

Oscilloscope art :-)



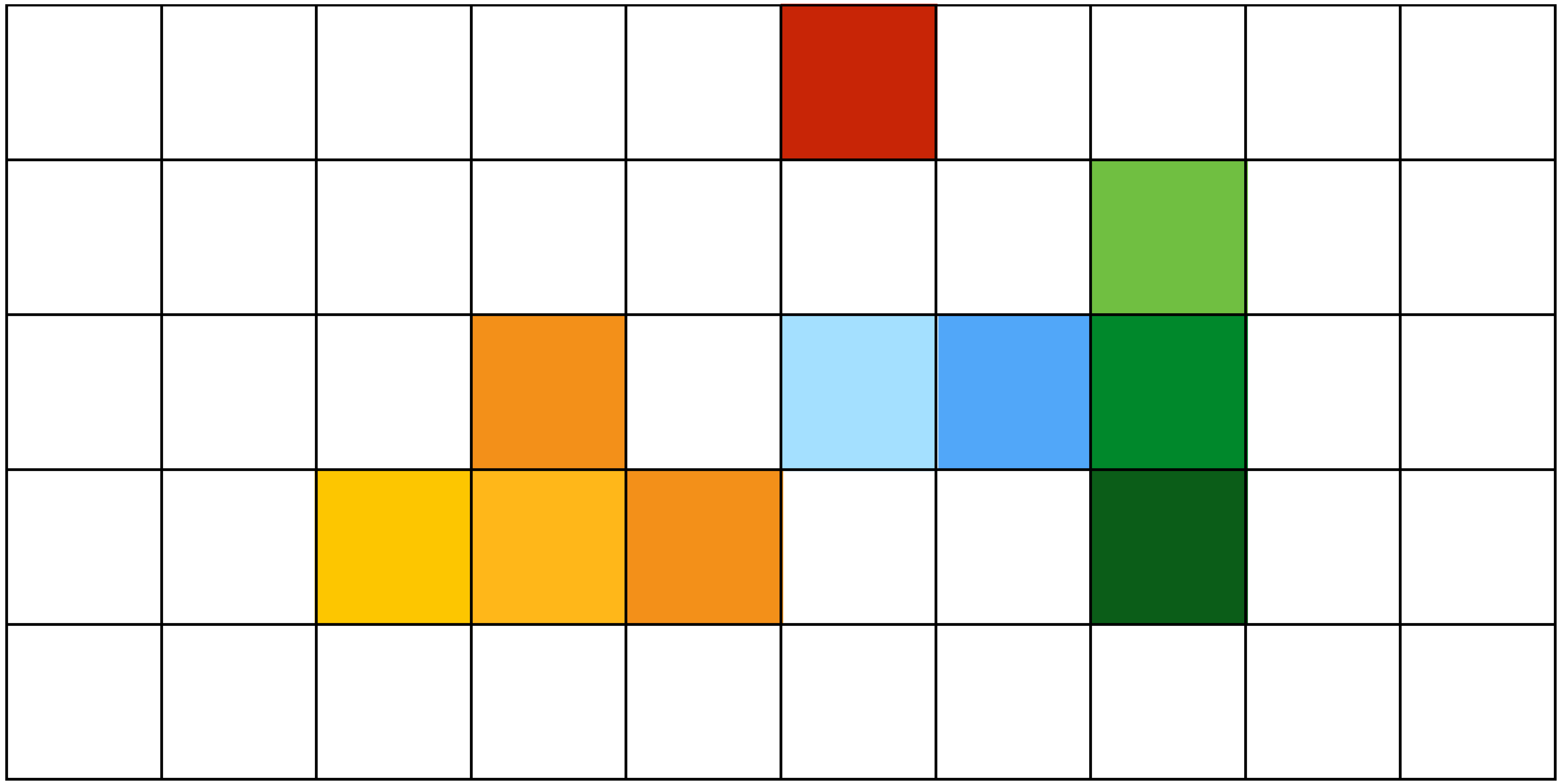
https://www.youtube.com/watch?v=rtR63-ecUNo

Frame buffer: memory for a raster display

image = "2D array of colors"

Output for a raster display

- Common abstraction of a raster display:
  - Image represented as a 2D grid of "pixels" (picture elements) **
  - Each pixel can can take on a unique color value



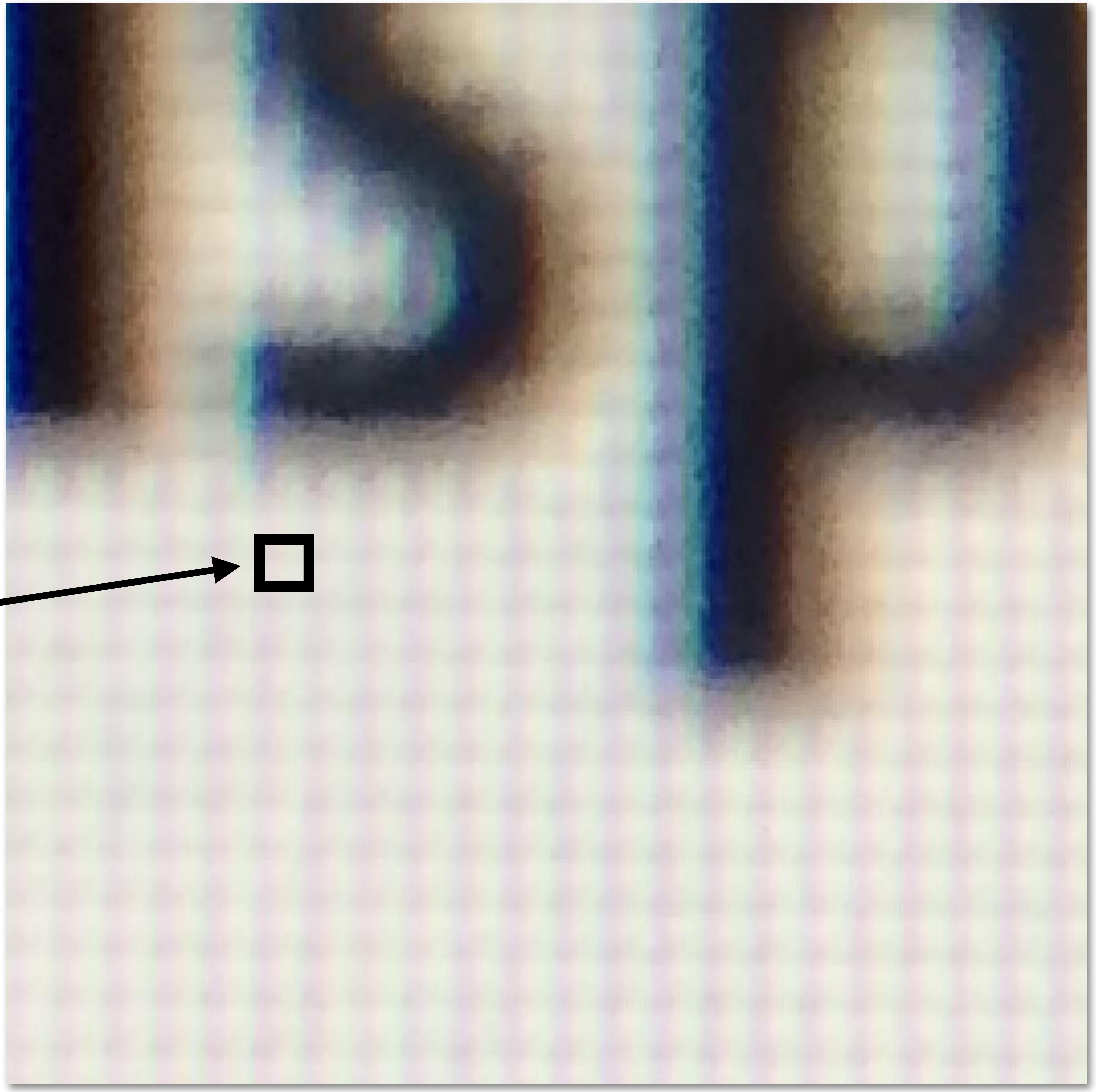
** We will strongly challenge this notion of a pixel "as a little square" next class. But let's go with it for now. ;-)

Flat panel displays

Low-Res LCD Display

High resolution color LCD, OLED, …

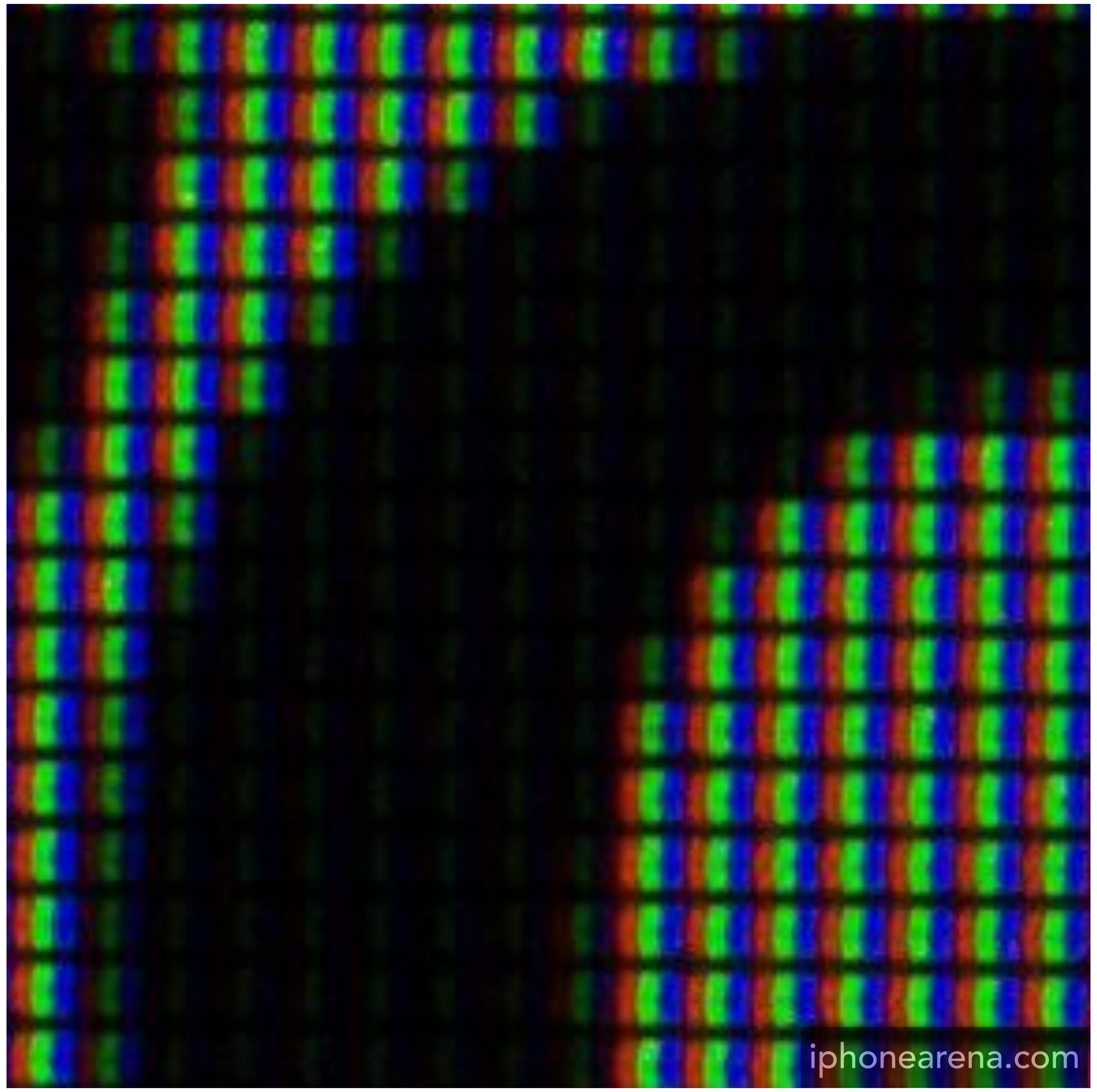
A raster display converts a color value at each pixel in an image to emitted light



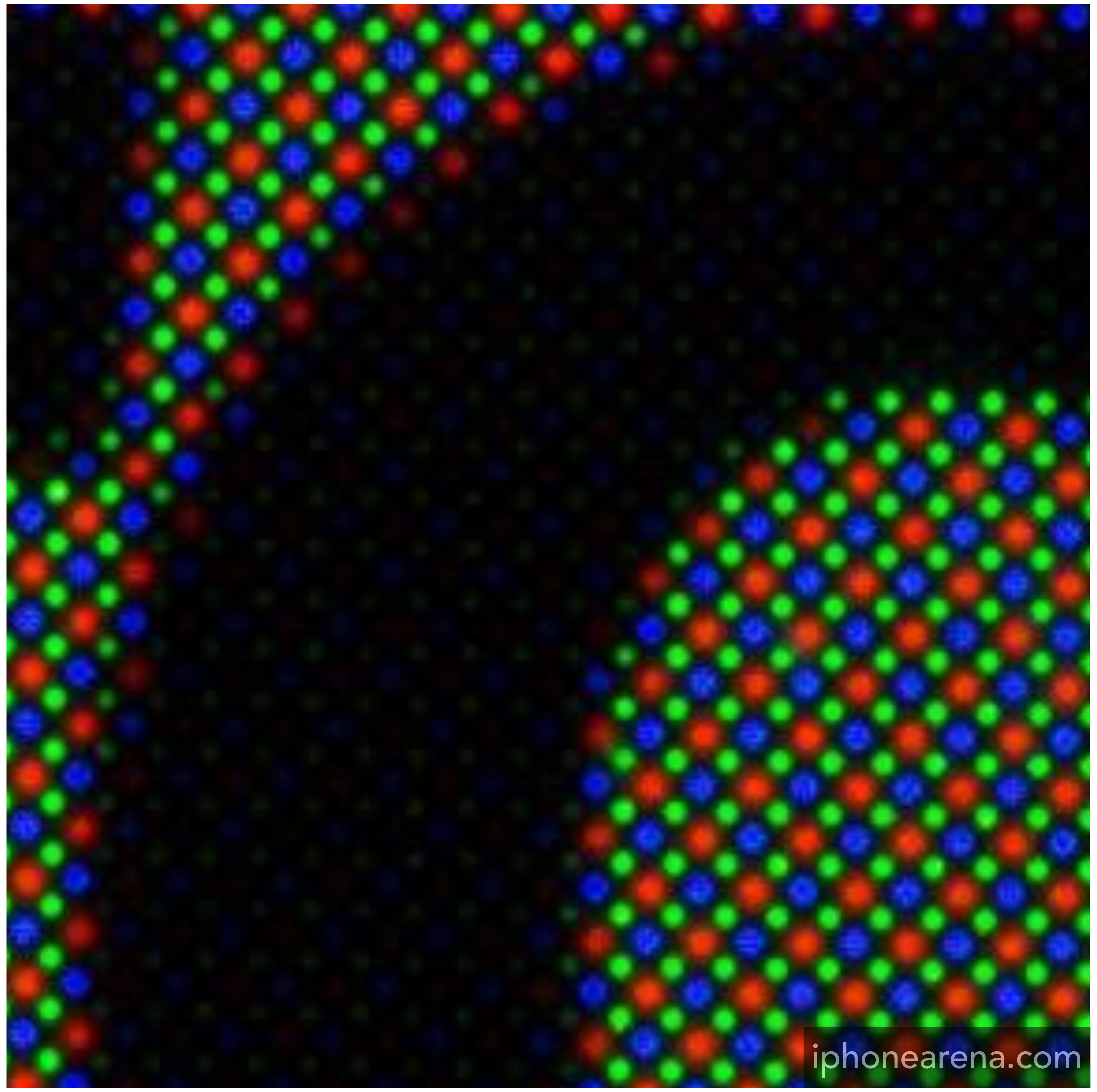
Display pixel on my laptop
(close up photo)

Close up photo of pixels on a modern display

LCD screen pixels (closeup)

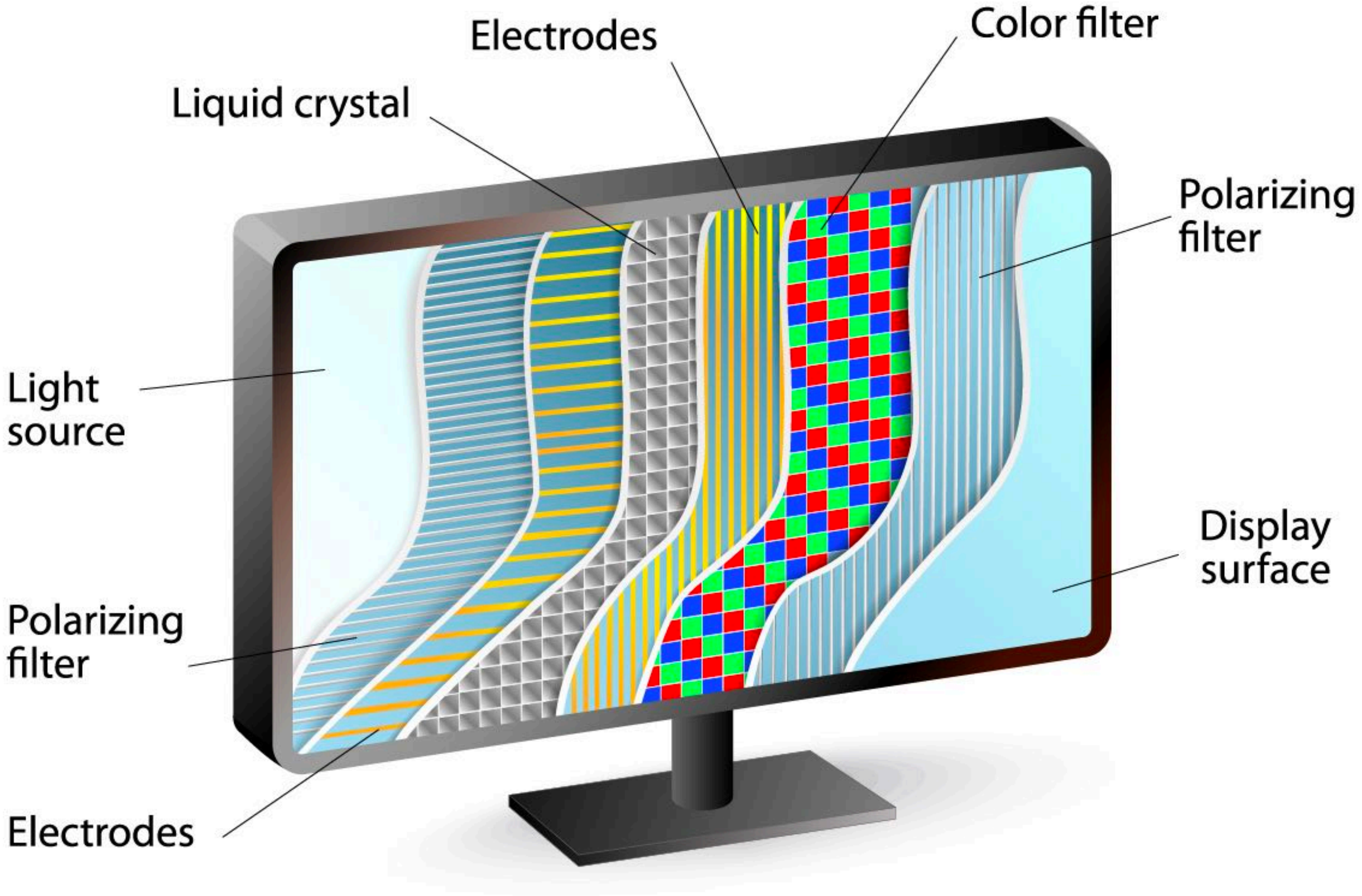


iPhone 6S



Galaxy S5

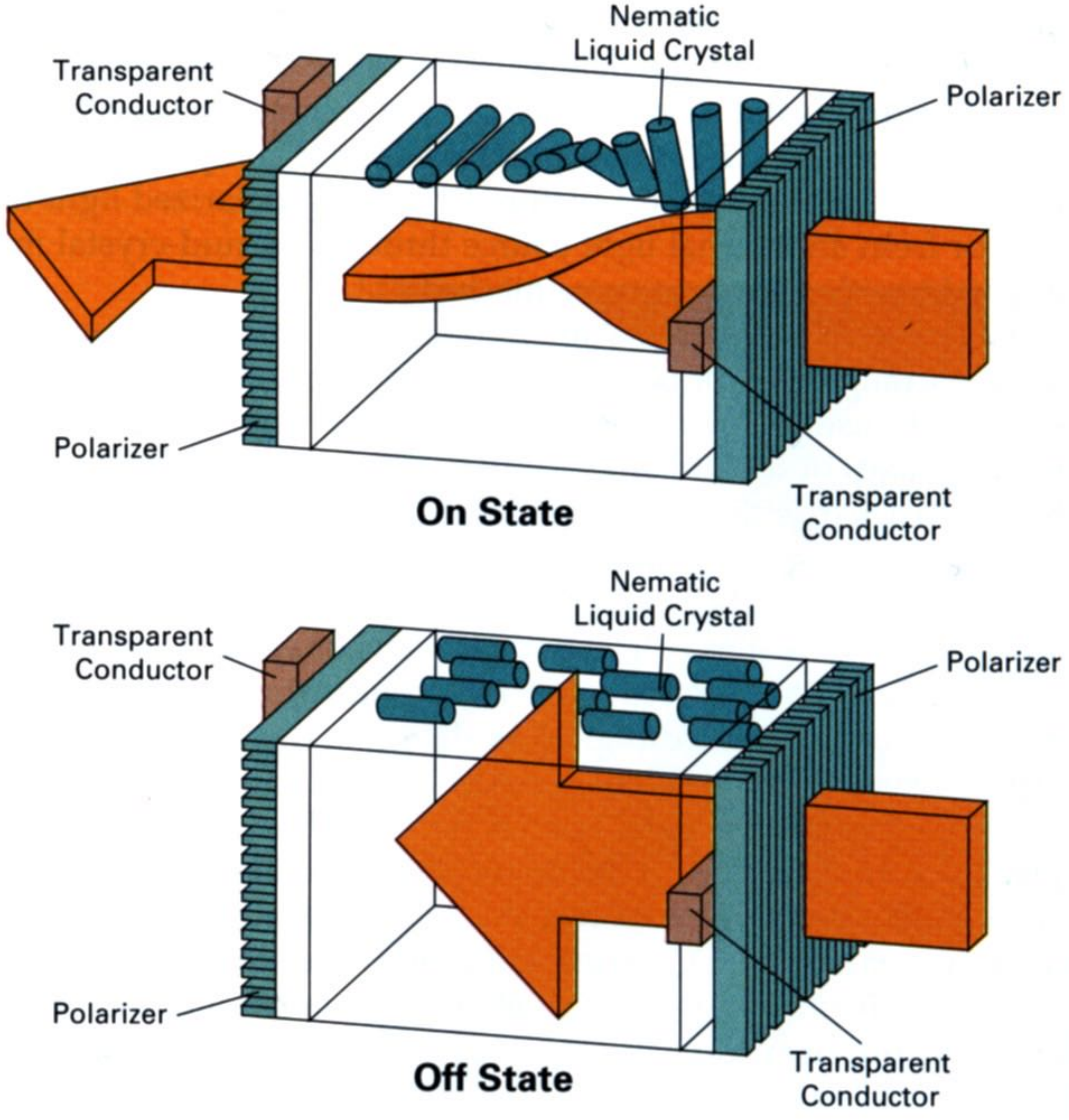
LCD screen



https://www.flexenable.com/blog/how-lcds-work/

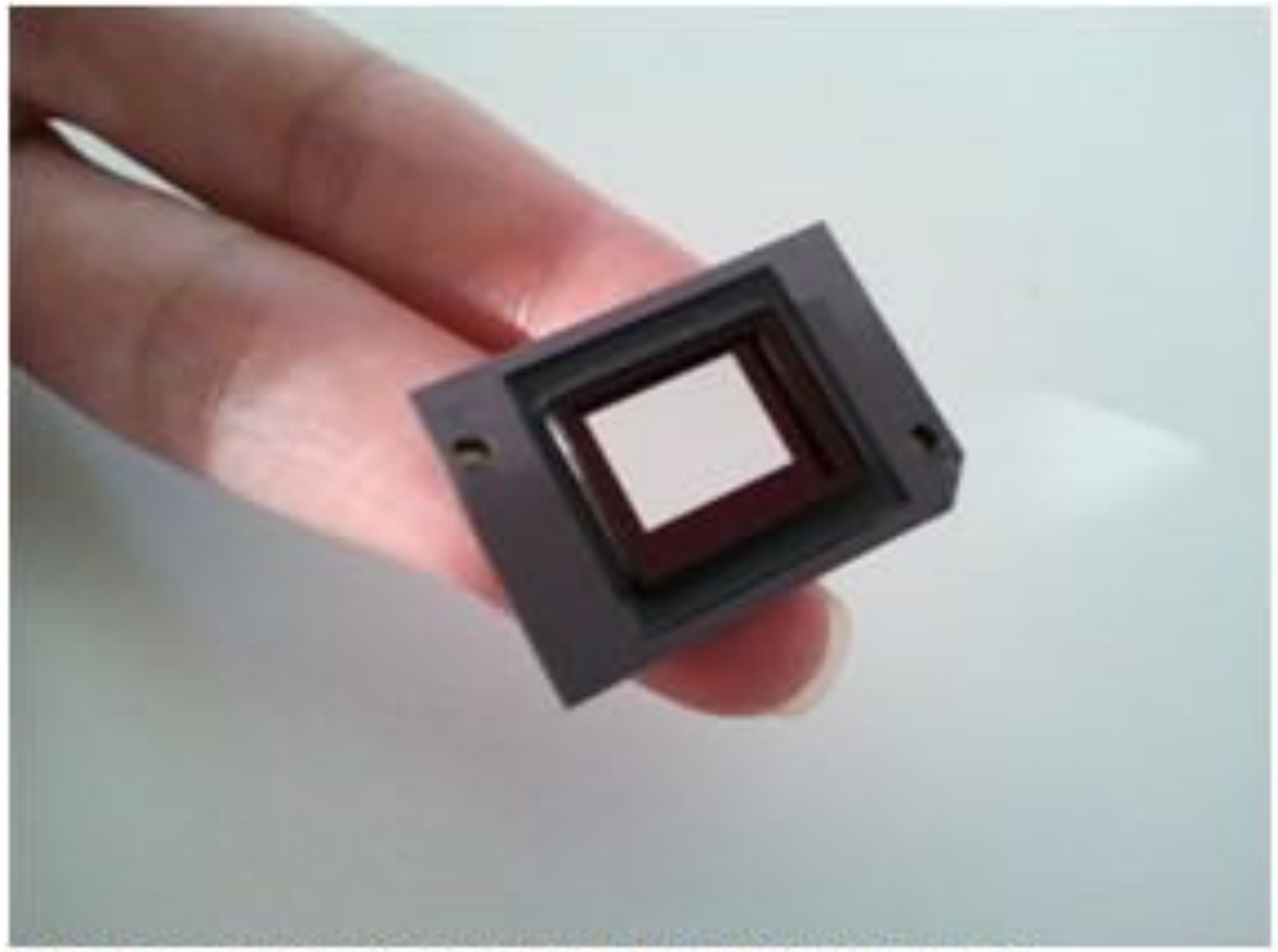
LCD (liquid crystal display) pixel

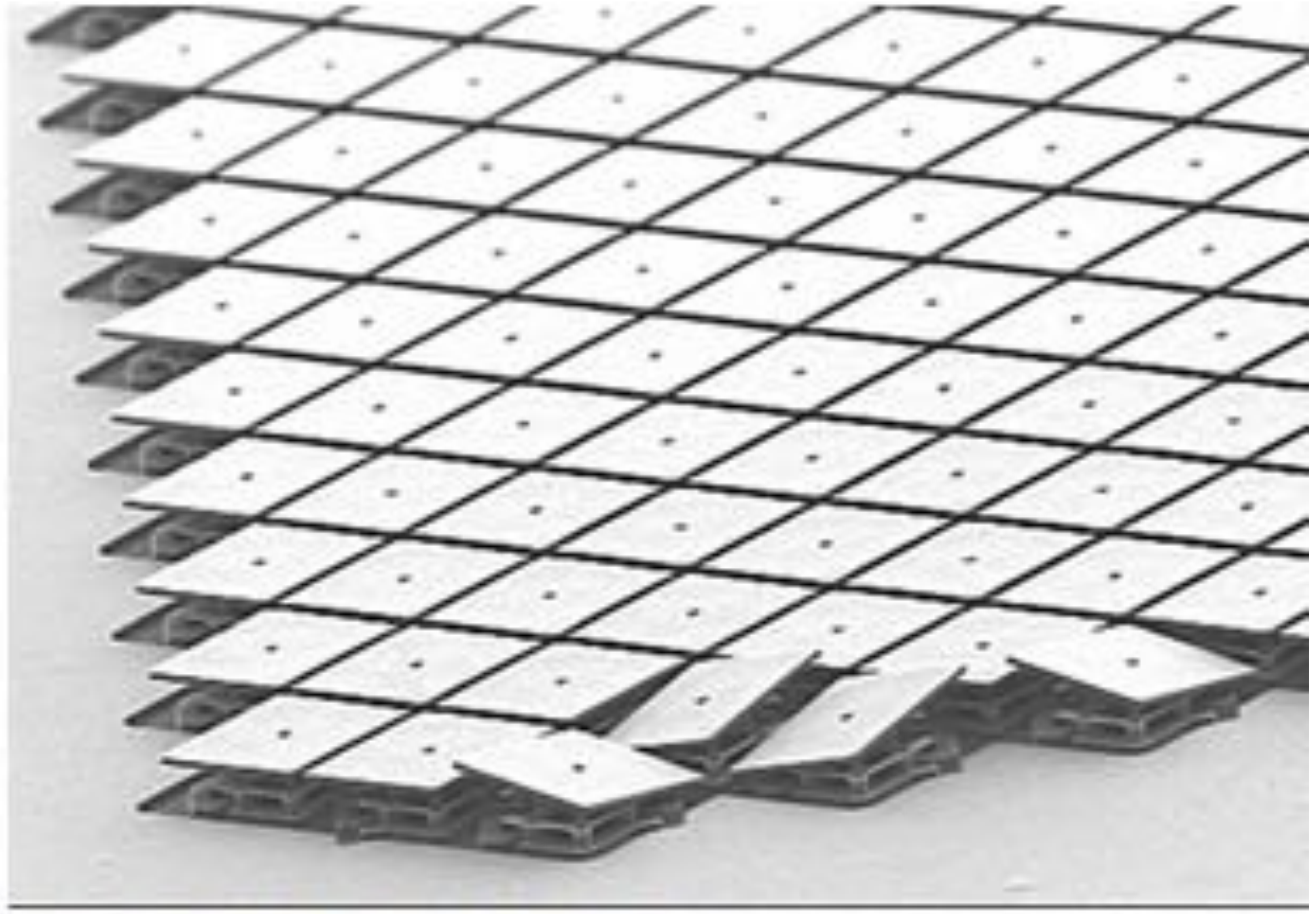
- **Principle: block or transmit light by twisting polarization**
- **Illumination from backlight (e.g. fluorescent or LED)**
- **Intermediate intensity levels by partial twist**



[Image credit: H&B fig. 2-16]

DMD projection display





[Y.K. Rabinowitz; EKB Technologies

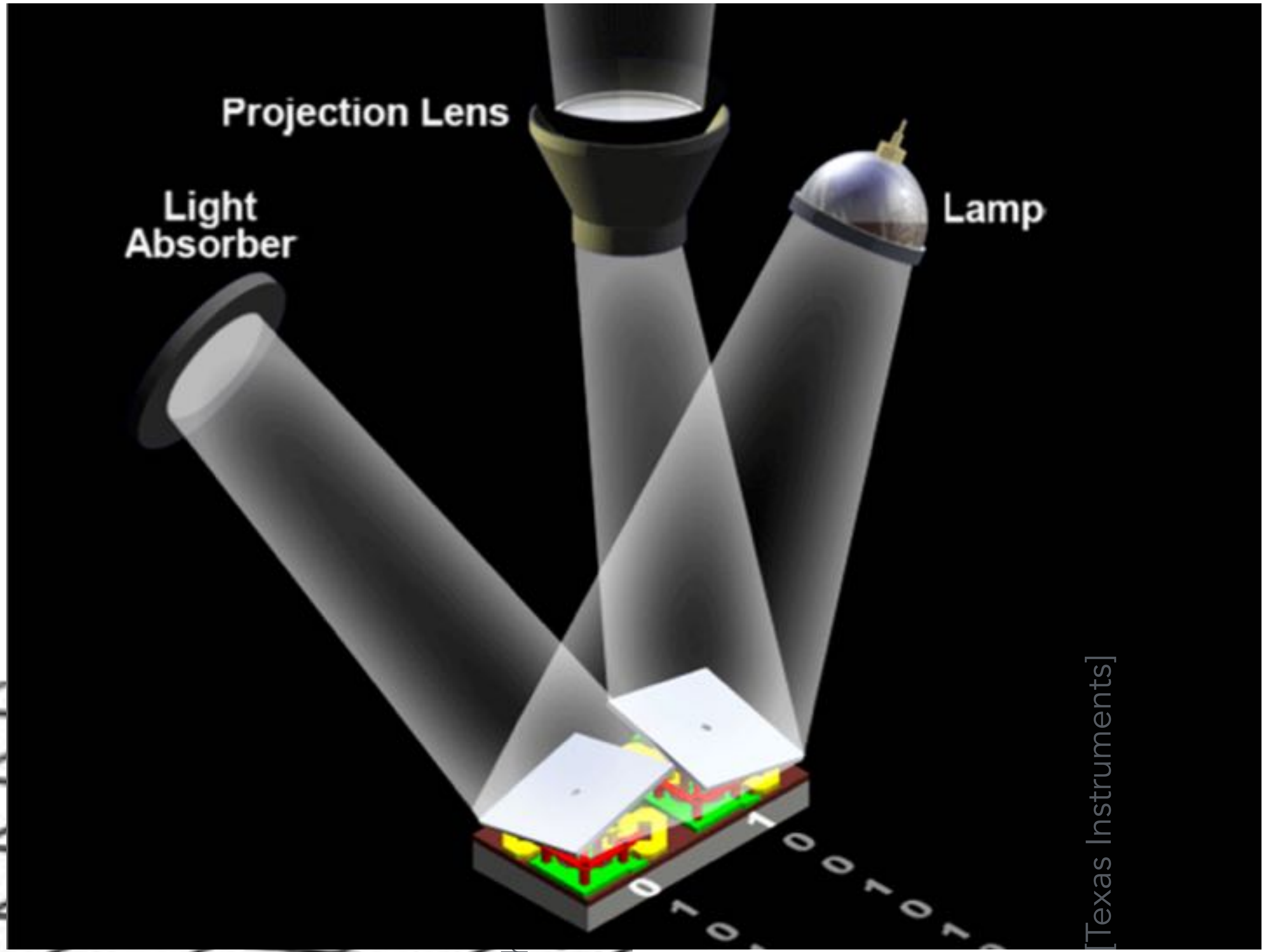
Array of micro-mirror pixels

DMD = Digital micro-mirror device

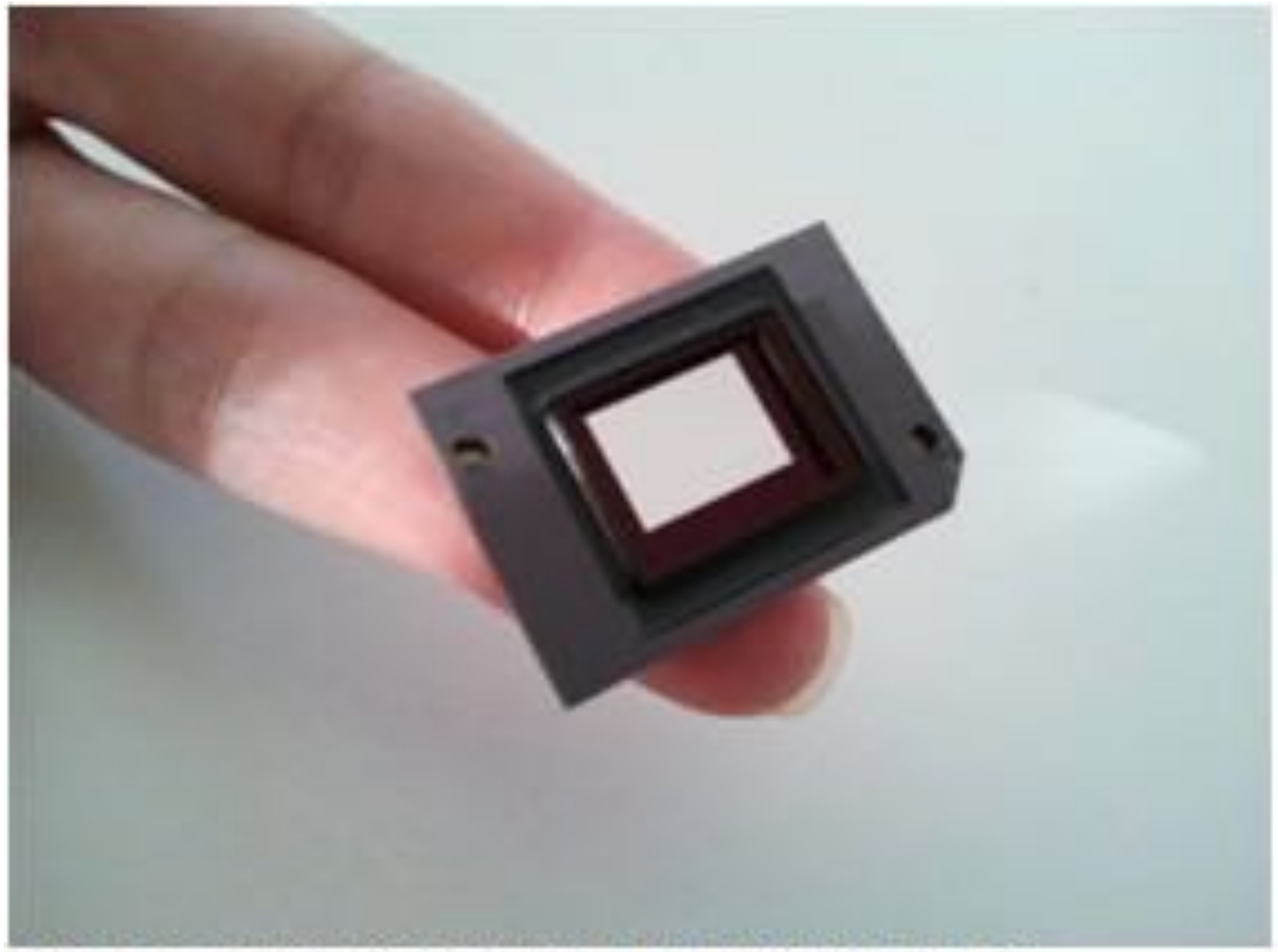
DMD projection display

Array of micro-mirror pixels

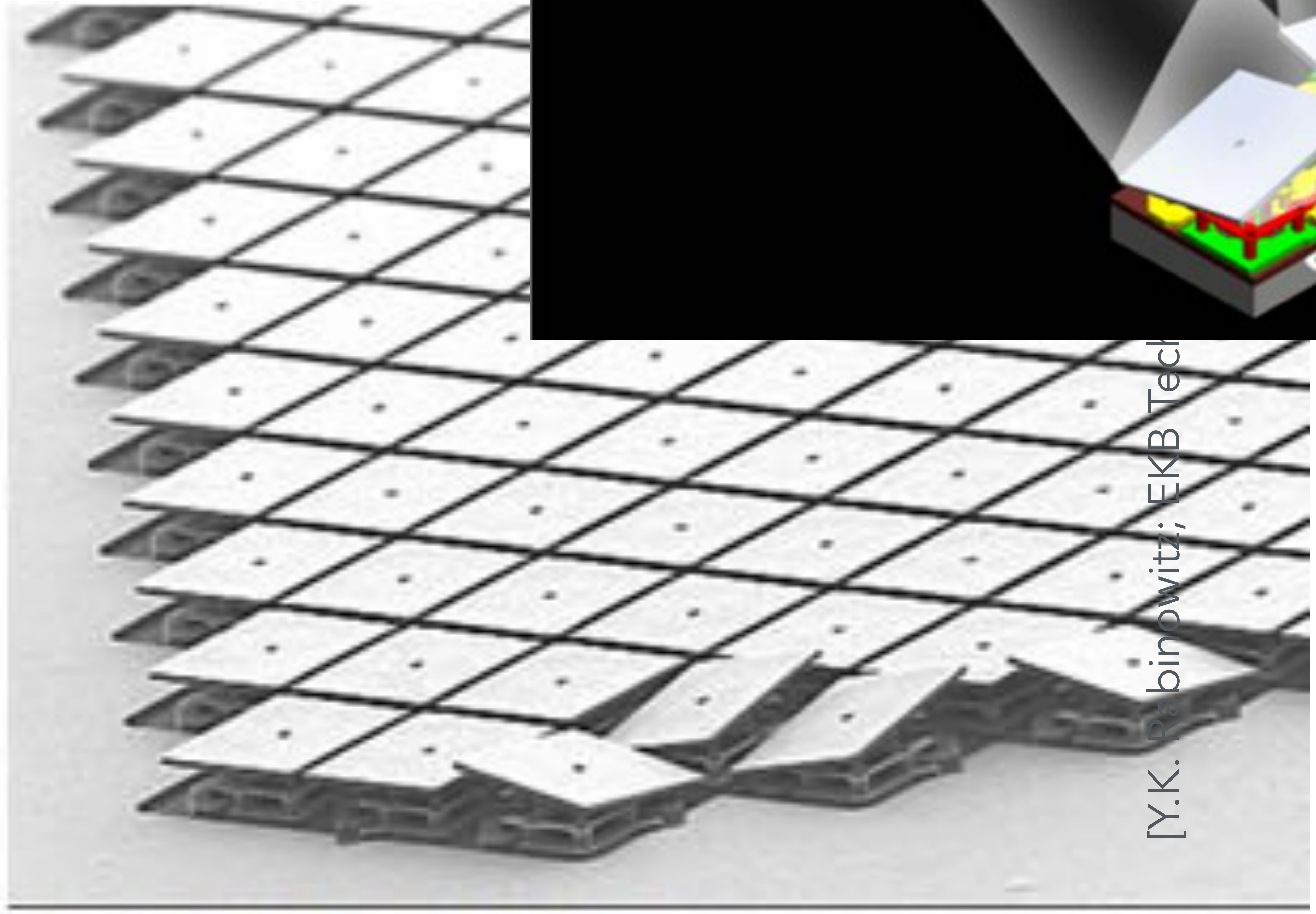
DMD = Digital micro-mirror device



[Texas Instruments]



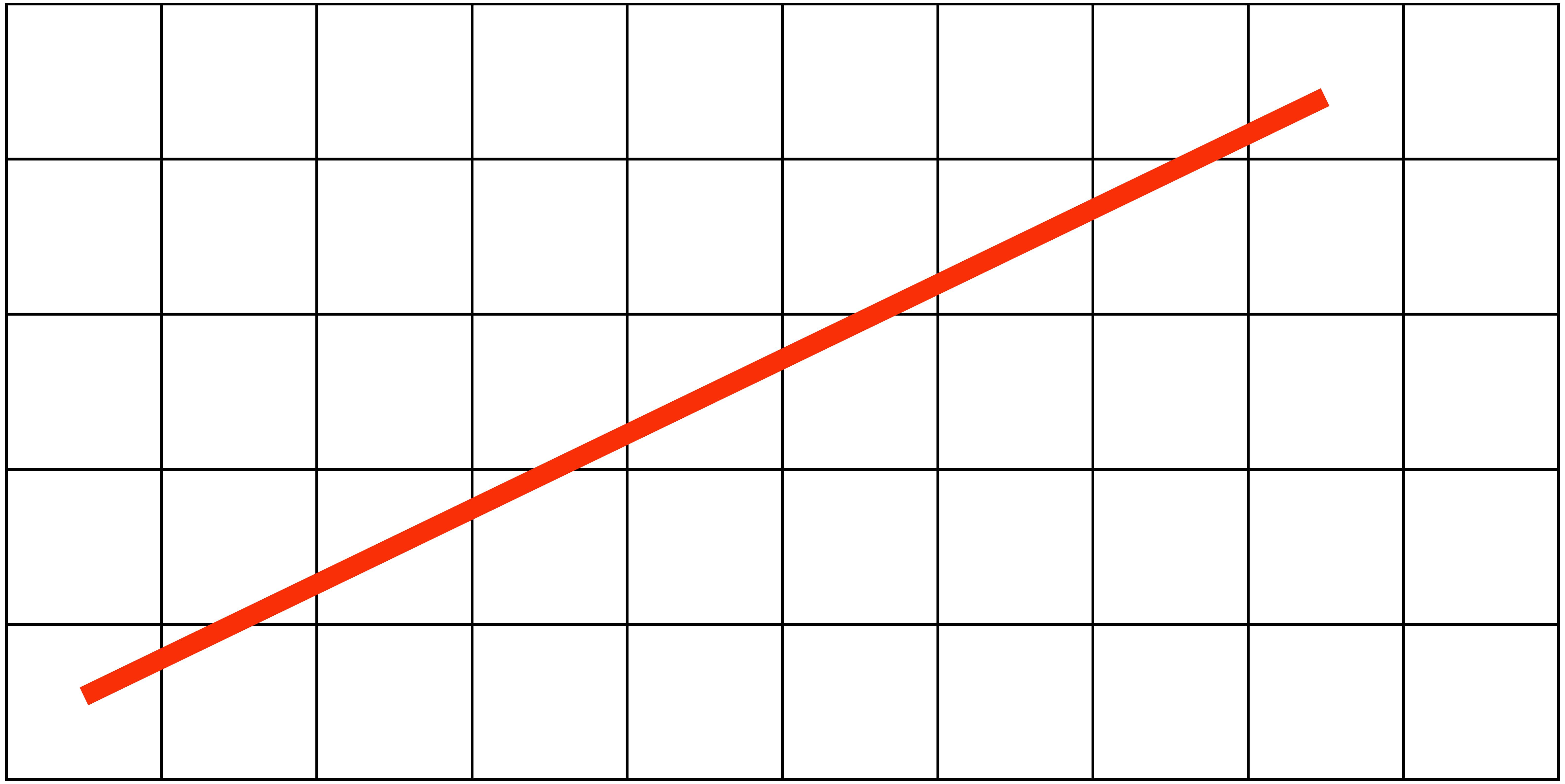
DIGITAL MICRO MIRROR DEVICE **(DMD)**
(**SLM** - Spatial Light Modulator)



MICRO MIRRORS CLOSE UP

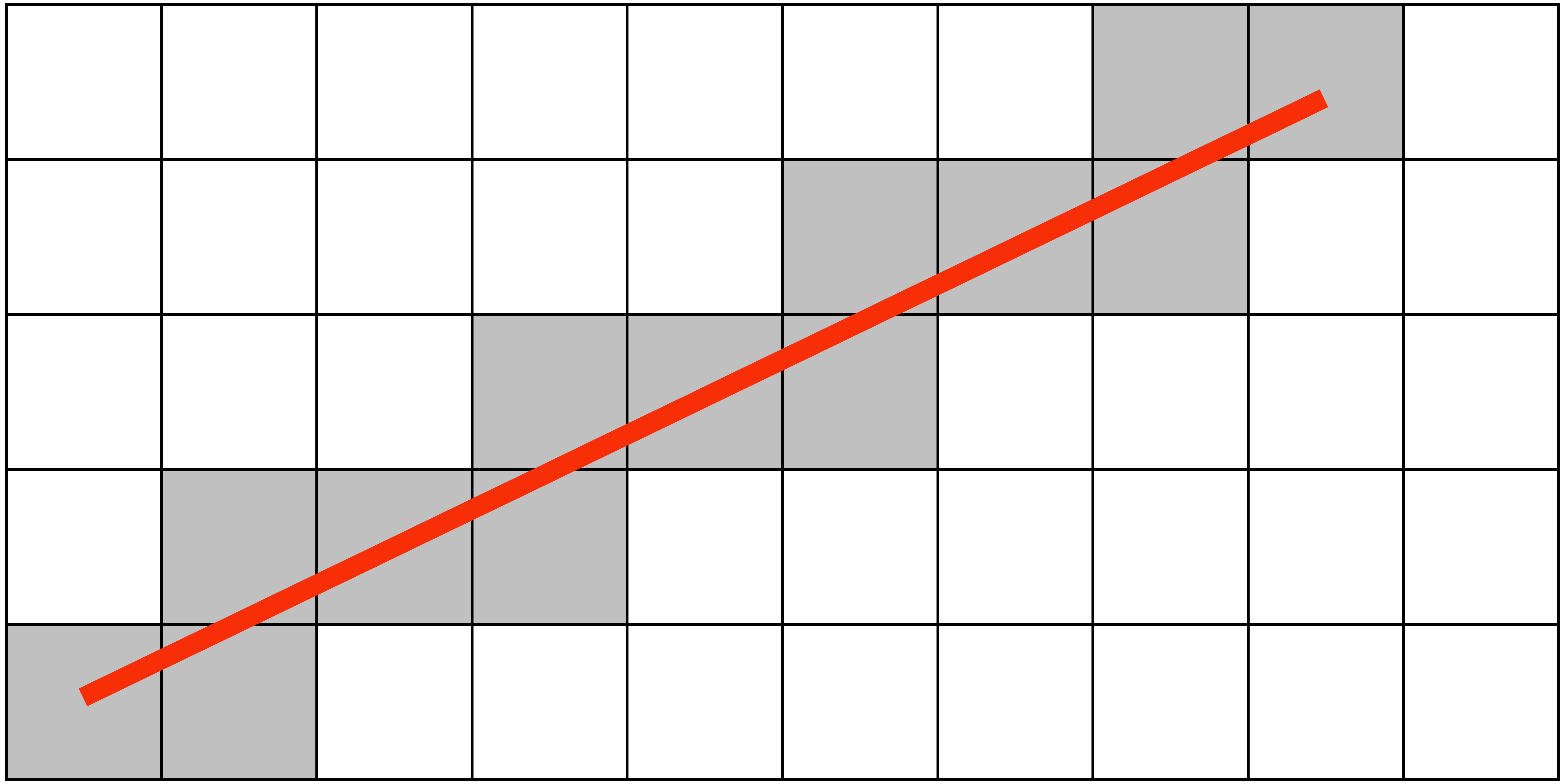
What pixels should we color in to depict a line?

"Rasterization": process of converting a continuous object (a line, a polygon, etc.) to a discrete representation on a "raster" grid (pixel grid)



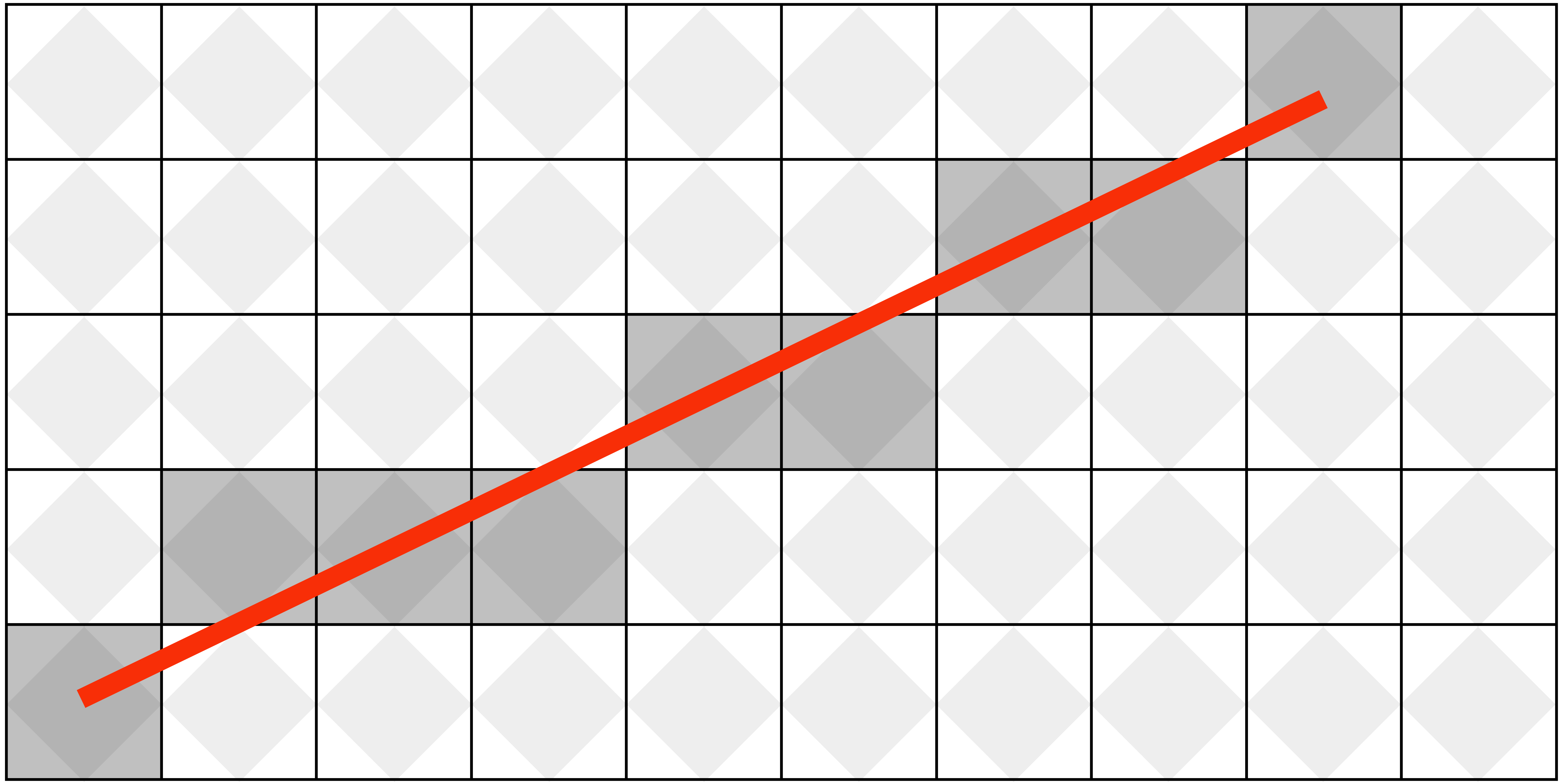
What pixels should we color in to depict a line?

Light up all pixels intersected by the line?



What pixels should we color in to depict a line?

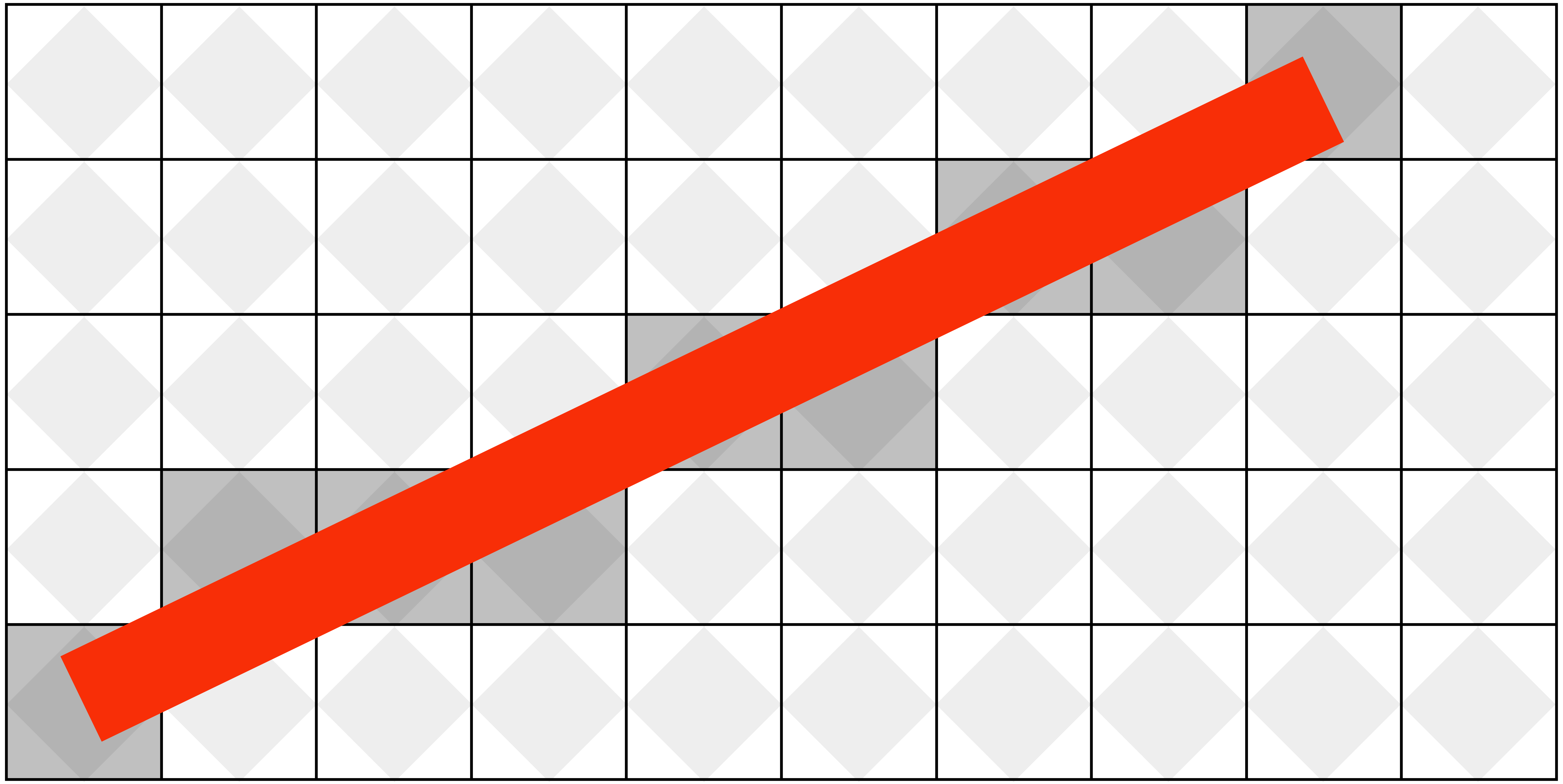
Diamond rule (used by modern GPUs):
light up pixel if line passes through associated diamond



What pixels should we color in to depict a line?

Is there a right answer?

(consider a drawing a "line" with thickness)



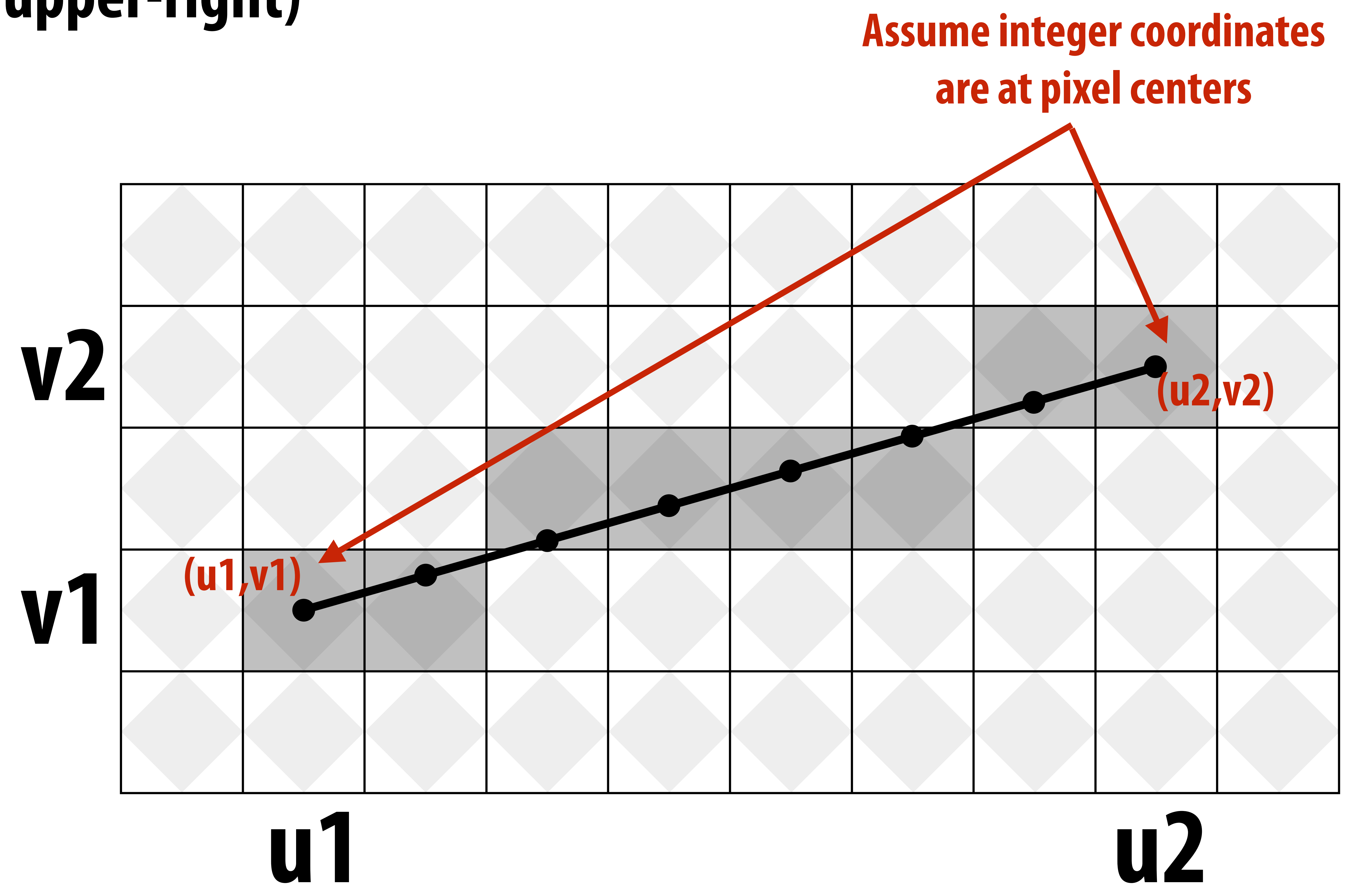
How do we find the pixels satisfying a chosen rasterization rule?

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

Incremental line rasterization

- Let's say a line is represented with integer endpoints: (u1,v1), (u2,v2)
- Slope of line: s = (v2-v1) / (u2-u1)
- Consider an easy special case:
  - u1 < u2, v1 < v2 (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) )
}
```

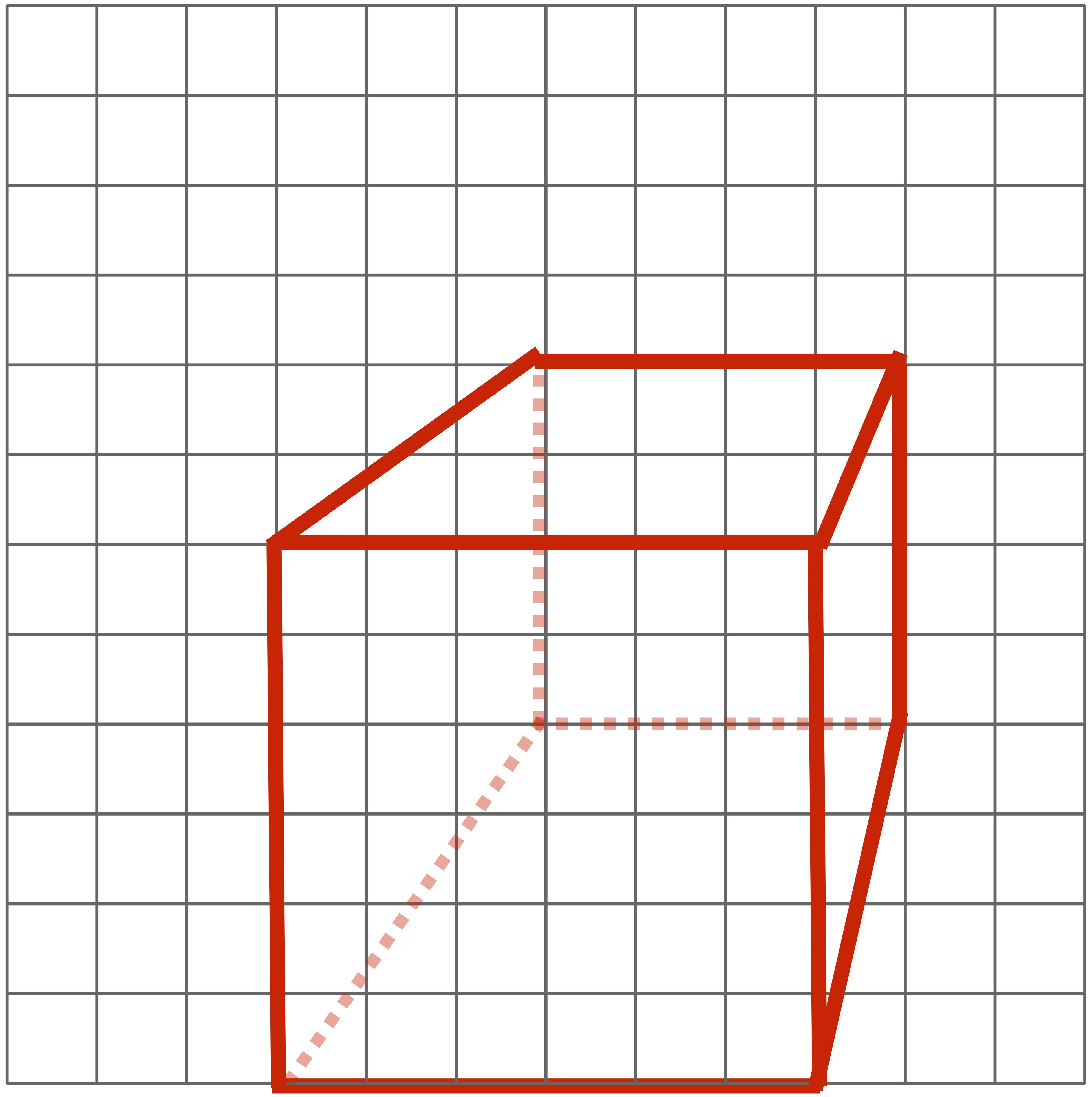


Common optimization: rewrite algorithm to use only integer arithmetic (Bresenham algorithm)

Line drawing of cube

We know how to compute to location of points in 3D on a 2D screen

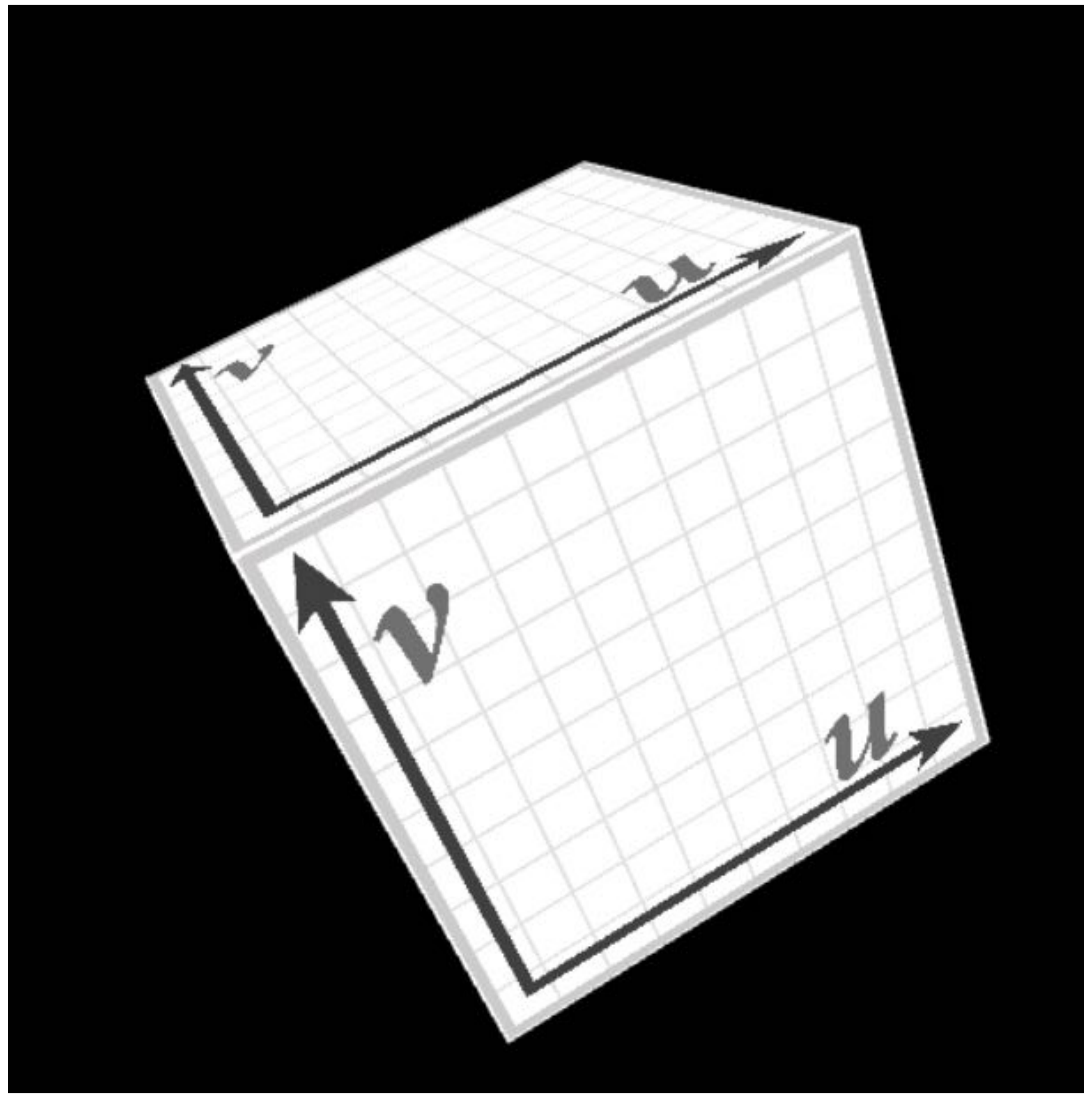
We know how to draw lines between those points.



We just rendered a simple line drawing of a cube.

But to render more realistic pictures
(or animations) we need a much richer model of the world.

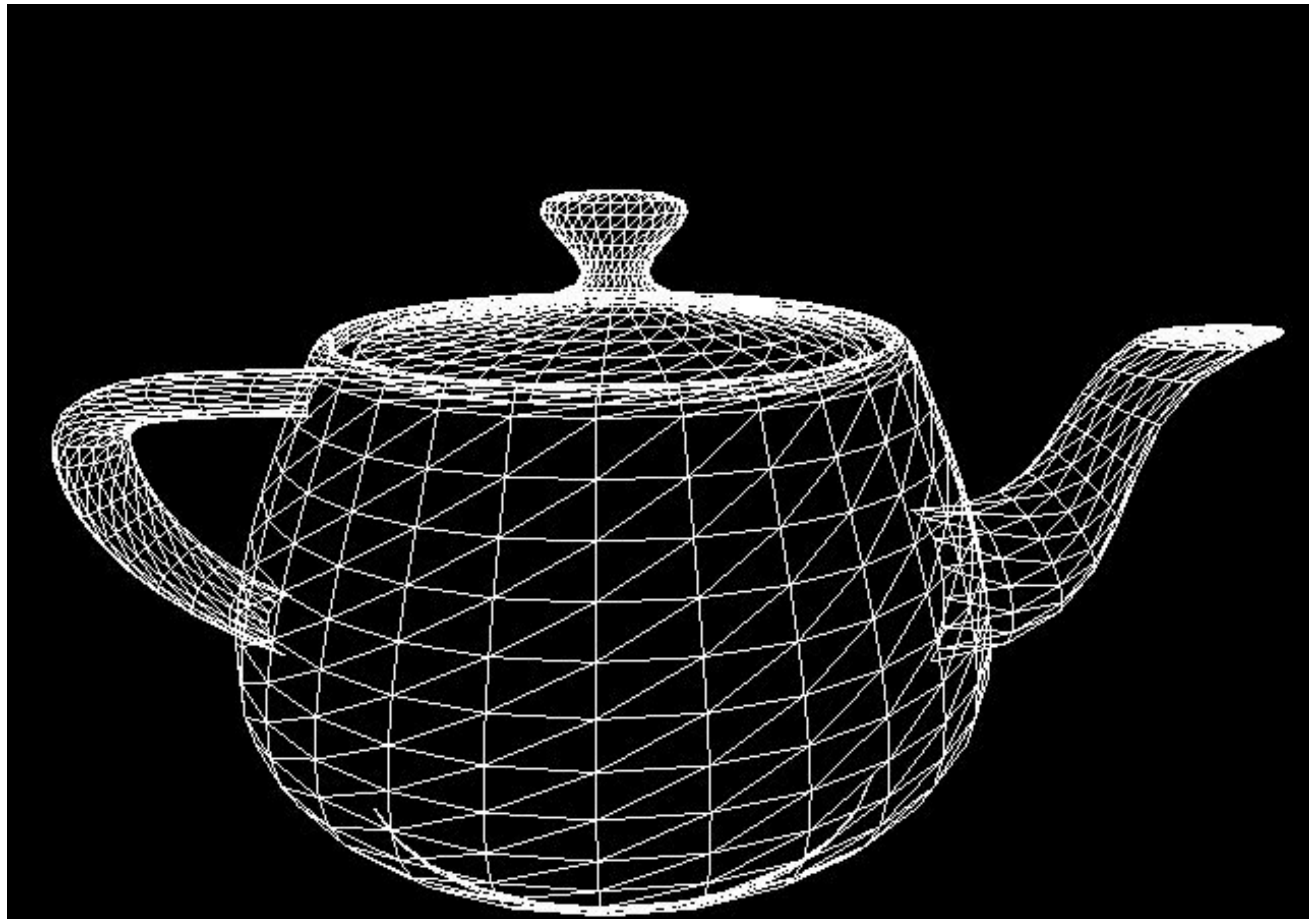
surfaces
materials
lights
cameras



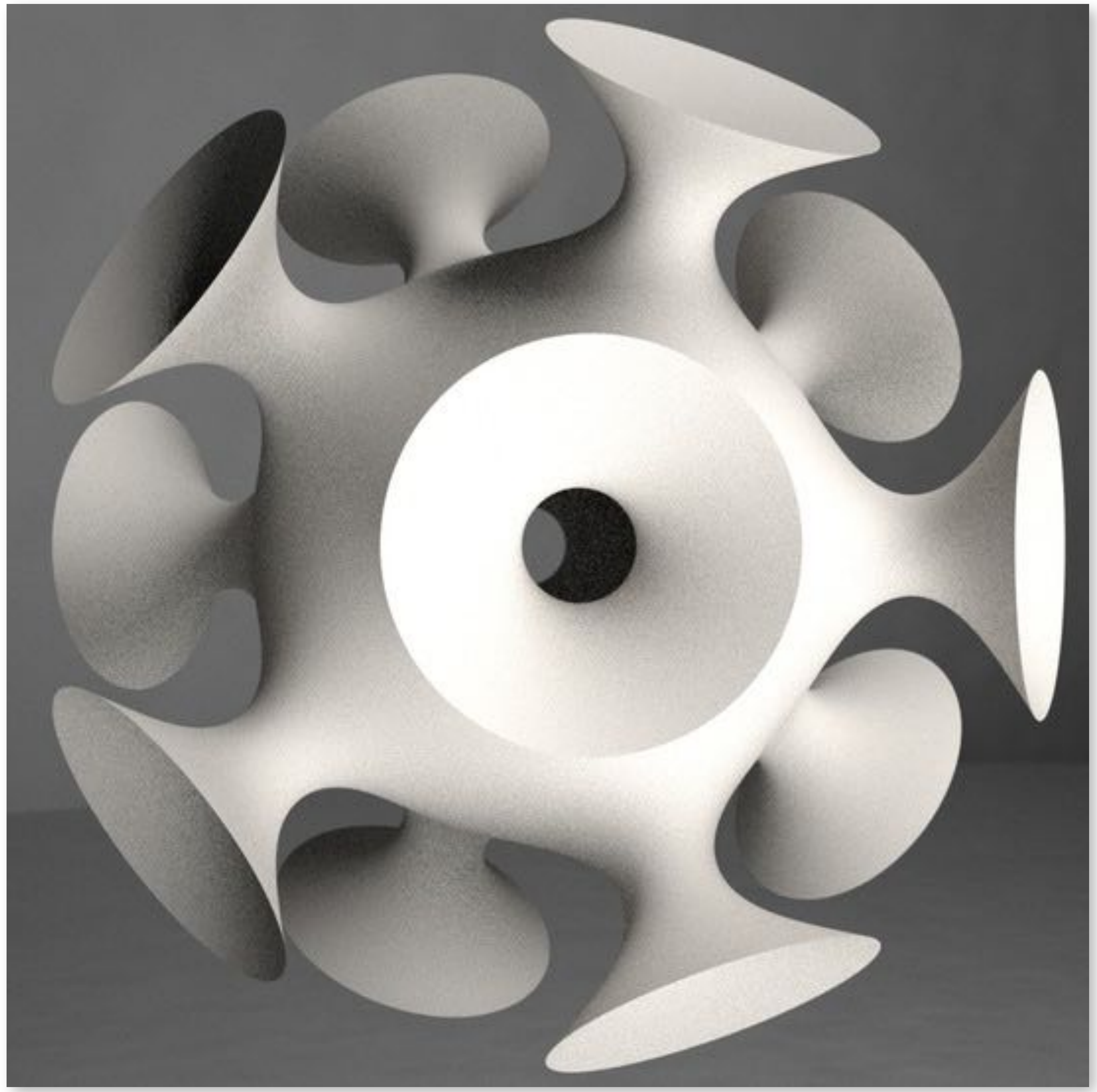
2D shapes

[Source: Batra 2015]

Complex 3D surfaces



[Utah Teapot]



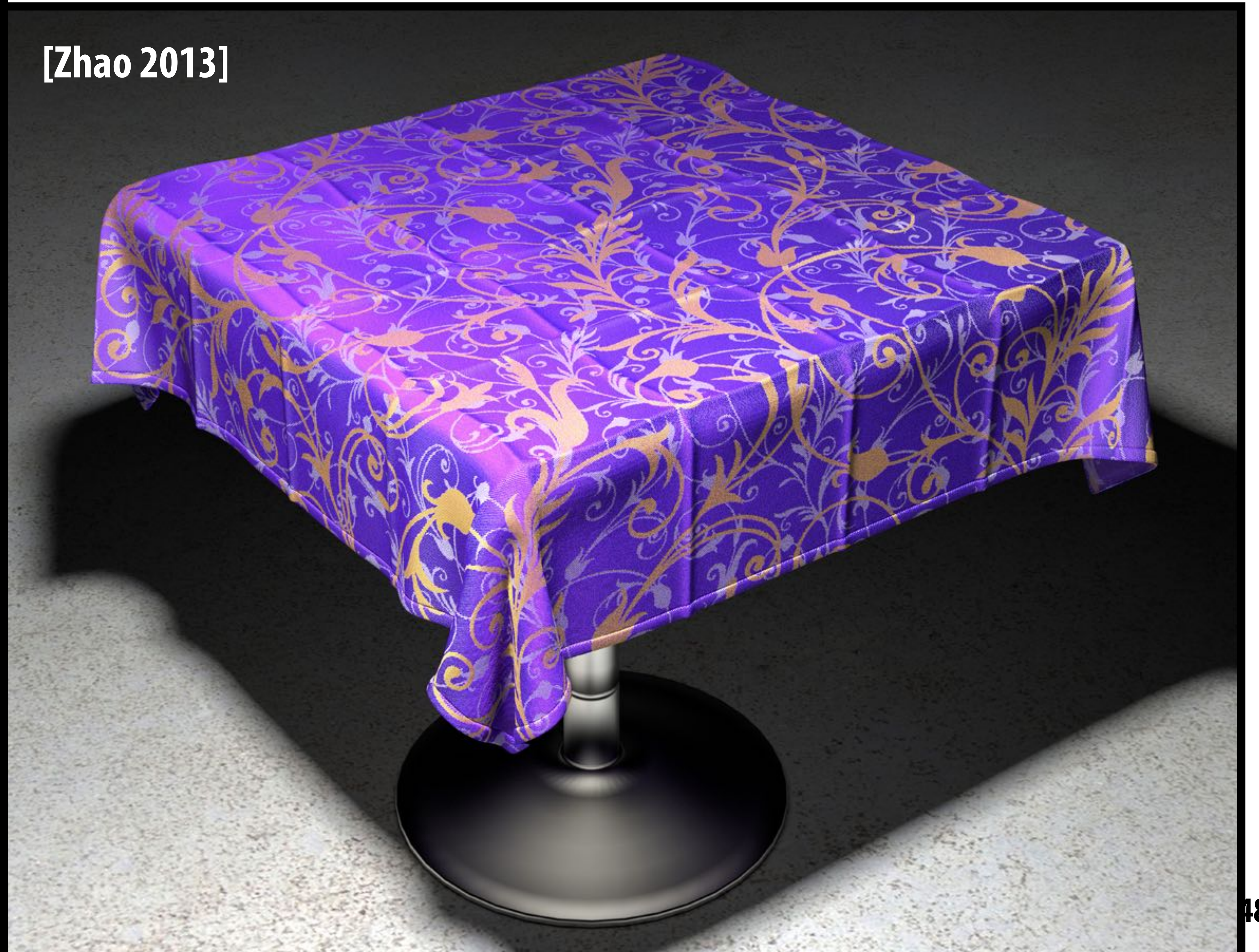
Platonic noid

[Kaldor 2008]

Modeling material properties


[Wann Jensen 2001]


[Jakob 2014]


[Zhao 2013]

Realistic lighting environments

Wall-E, (Pixar 2008)

Animation: modeling motion

https://www.youtube.com/watch?v=6G3O60o5U7w

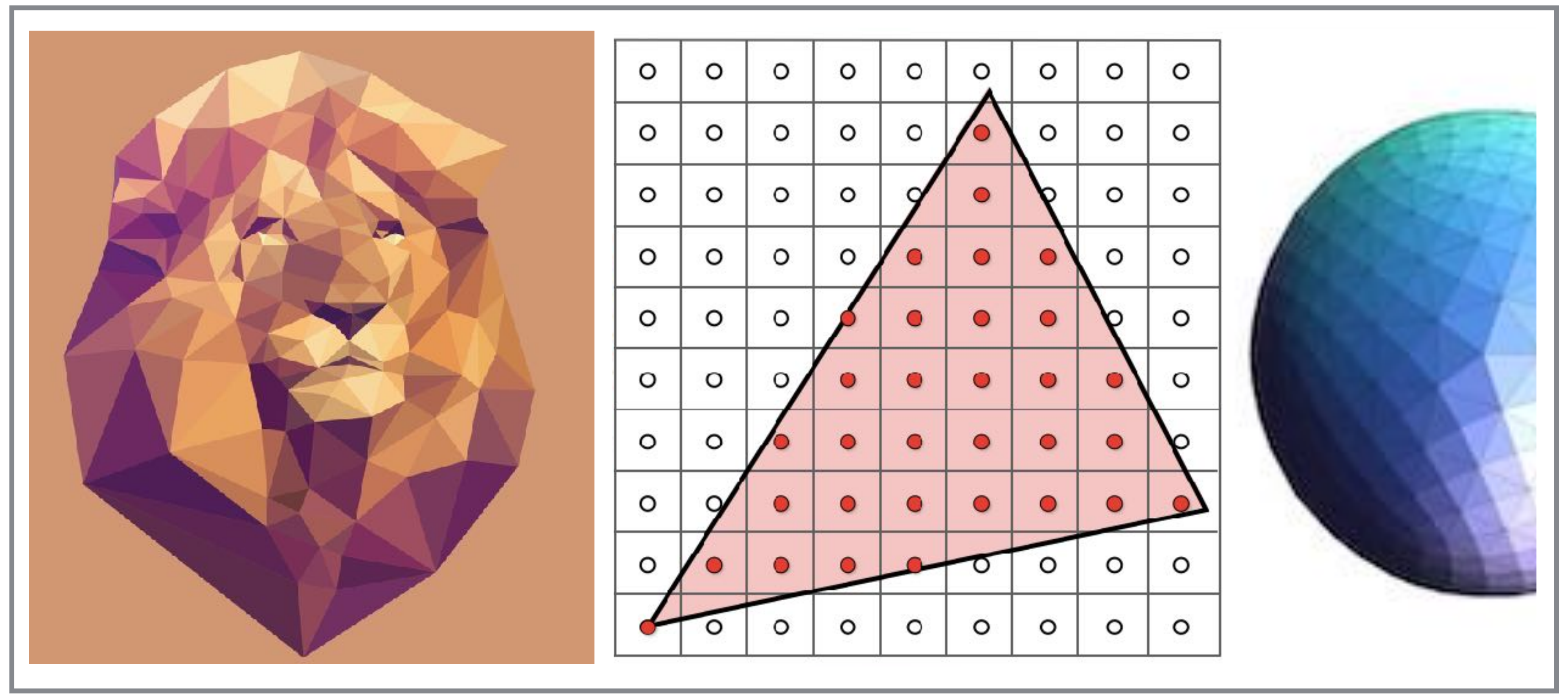
Course Logistics

About this course

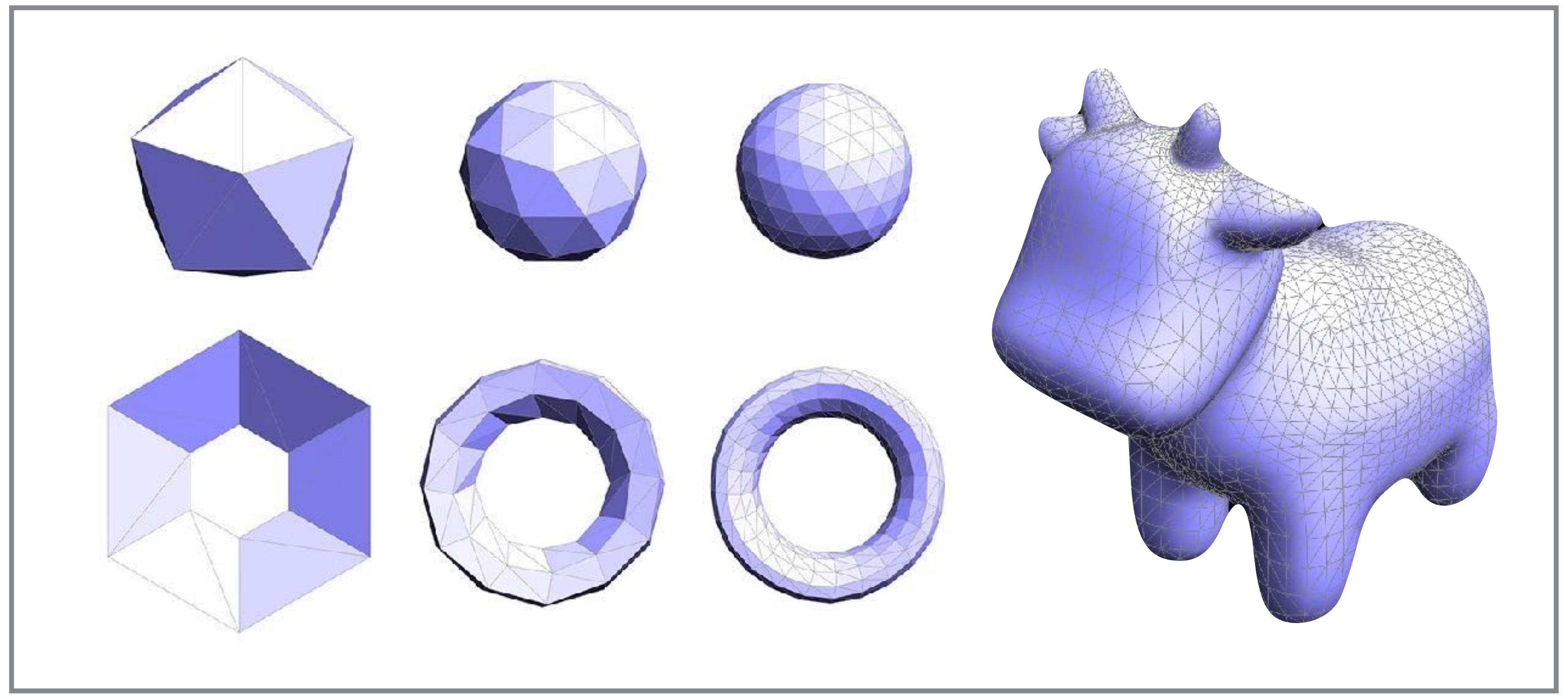
- A broad overview of major topics and techniques in interactive computer graphics: geometry, rendering, imaging

- Learn by implementing:
  - Focus on implementing fundamental data structures and algorithms that are reused across all areas of graphics
  - We expect that you can understand/write/debug C/C++ code

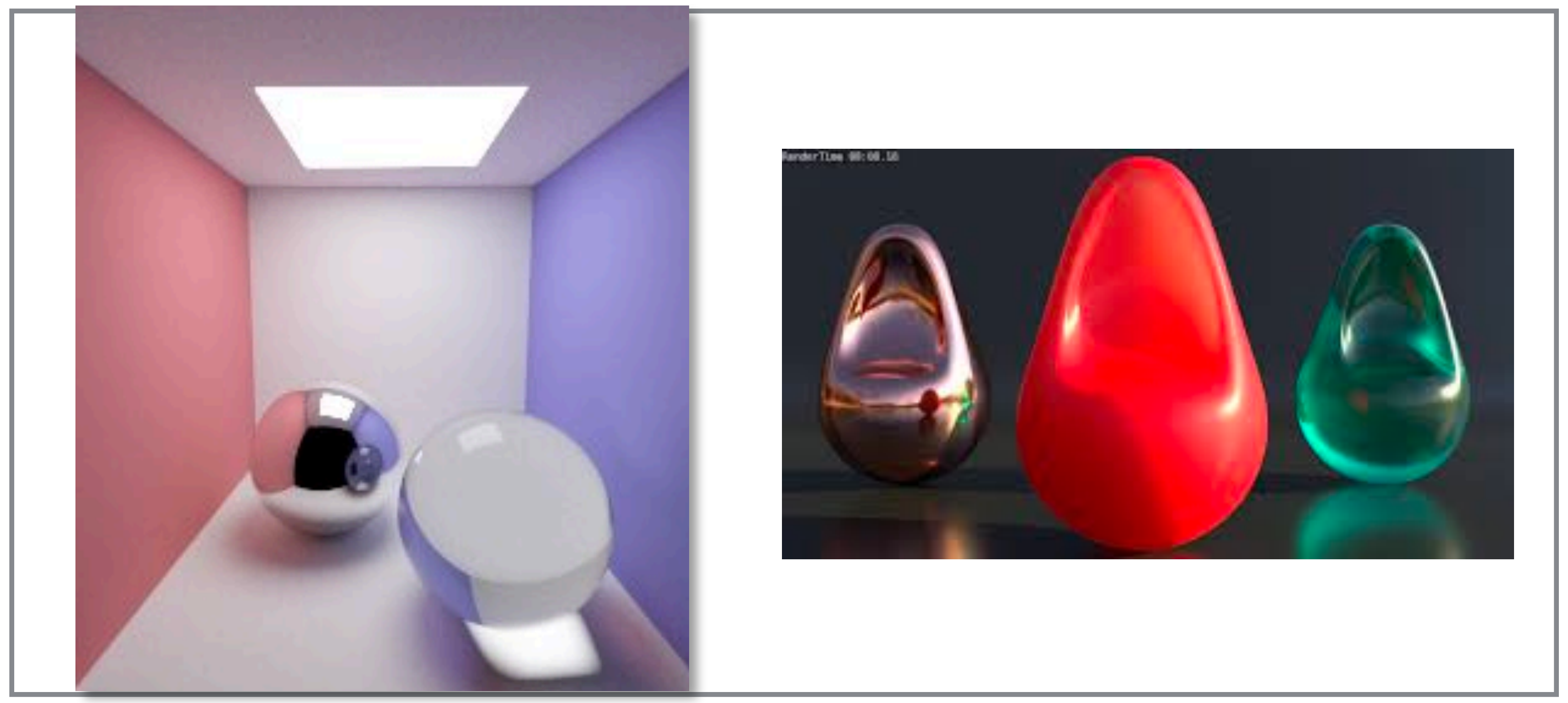
Course programming assignments



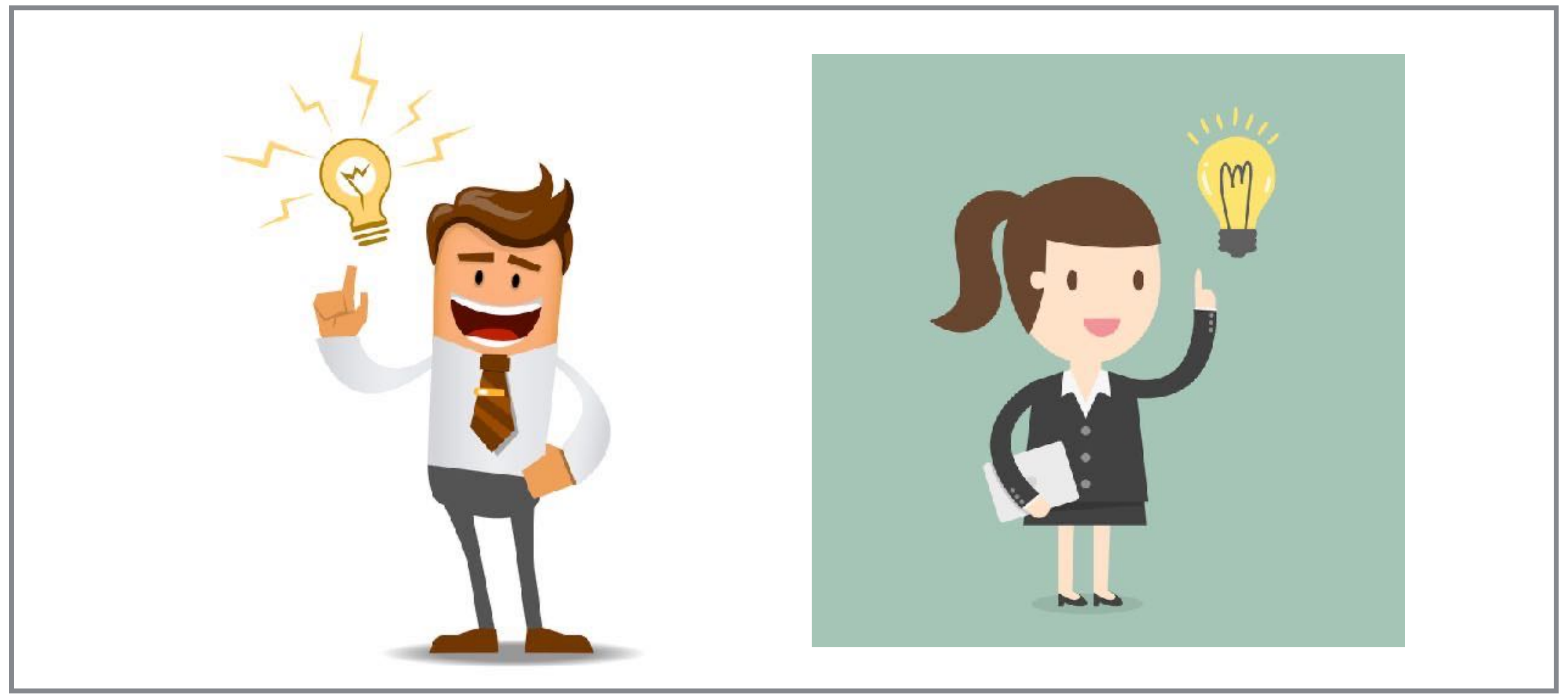
1. 2D drawing (2 weeks)



2. Geometry editing (2 weeks)



3. Path tracer (2 weeks)



4. Self-selected project
extend existing project, or choose your own
(~3 weeks)

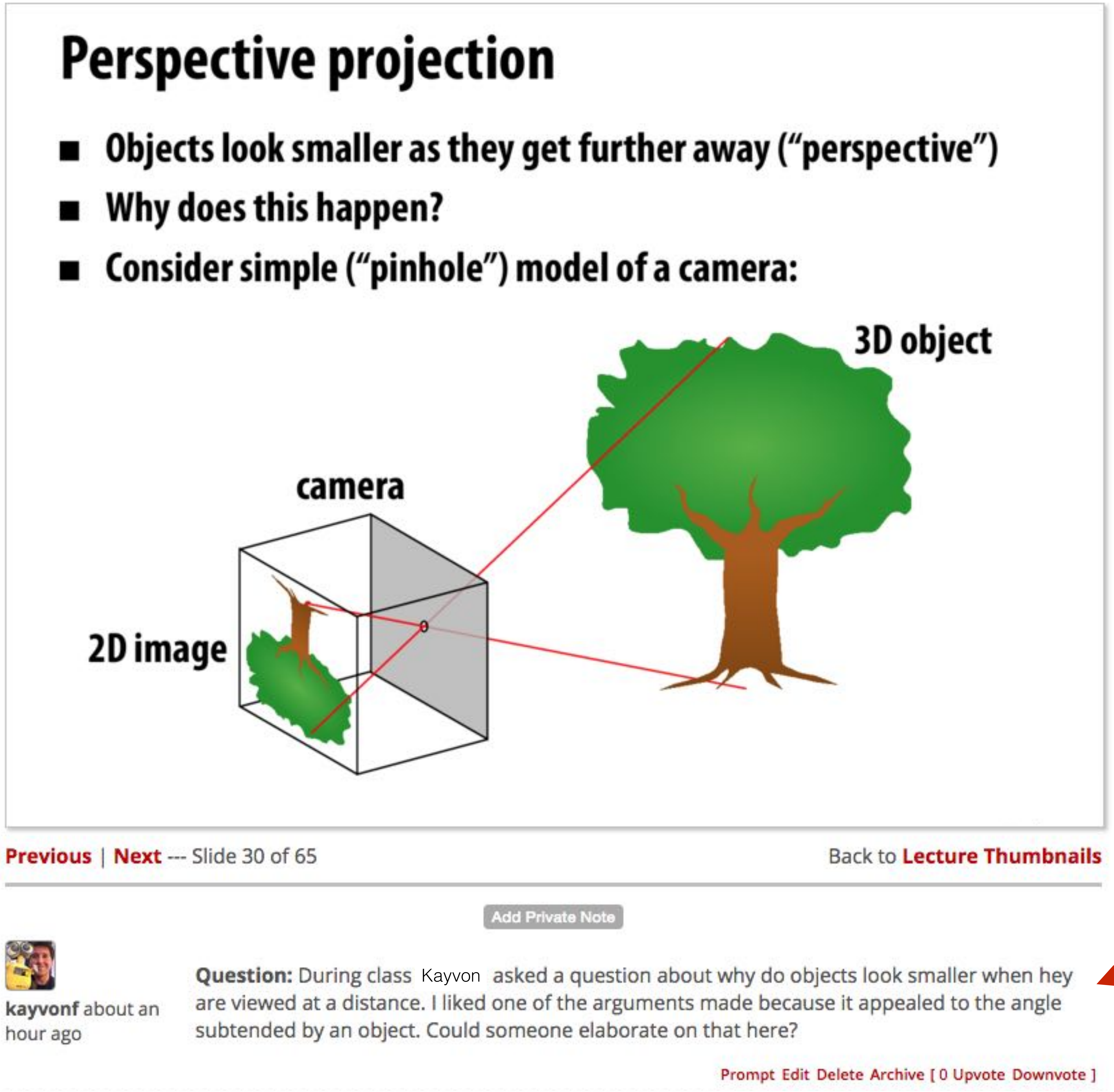
Man Designed by Alekksall / Freepik. Woman designed by Dooder / Freepik

Assignments / grading

- (48%) Three programming assignments
  - In teams of up to two students (yes, you can work alone if you wish)
- (12%) Written Exercises
  - Weekly written exercises (think of these as possible exam problems)
    - Graded (mostly) on participation
- (24%) Self-selected final project
  - Extend an earlier assignment, or do your own thing!
- (16%) Exam
  - Given in week 8 or week 9 of the course, evening exam (not in class)

The course web site

We have no textbook for this class and so the lecture slides and instructor/TA/student discussions on the web are the primary course reference



Slide comments and discussion

FAQ

- **How are CS248A and CS248B related?**
  - They are explicitly designed to be independent starter courses for the visual computing track. There is no assumption you've taken CS248A before CS248B or vice versa.
  - The biggest point of content overlap is the lecture on transforms (lecture 3)

- **Are lectures recorded?**
  - Yes, since this is an SCPD class.
  - My expectation is that all local students come to class. I may or may not find ways to encourage it!

FAQ

- **Is there a final?**
  - **No… the final exam slot is used for our project showcase**
  - **There will be one exam that will be an evening exam in week 8 or 9 of the quarter**

- **Do I need a partner for HW assignments?**
  - **No, each year there are students that choose to do al the assignment alone**
  - **Need a partner: we will find one for you, via our partner search form**

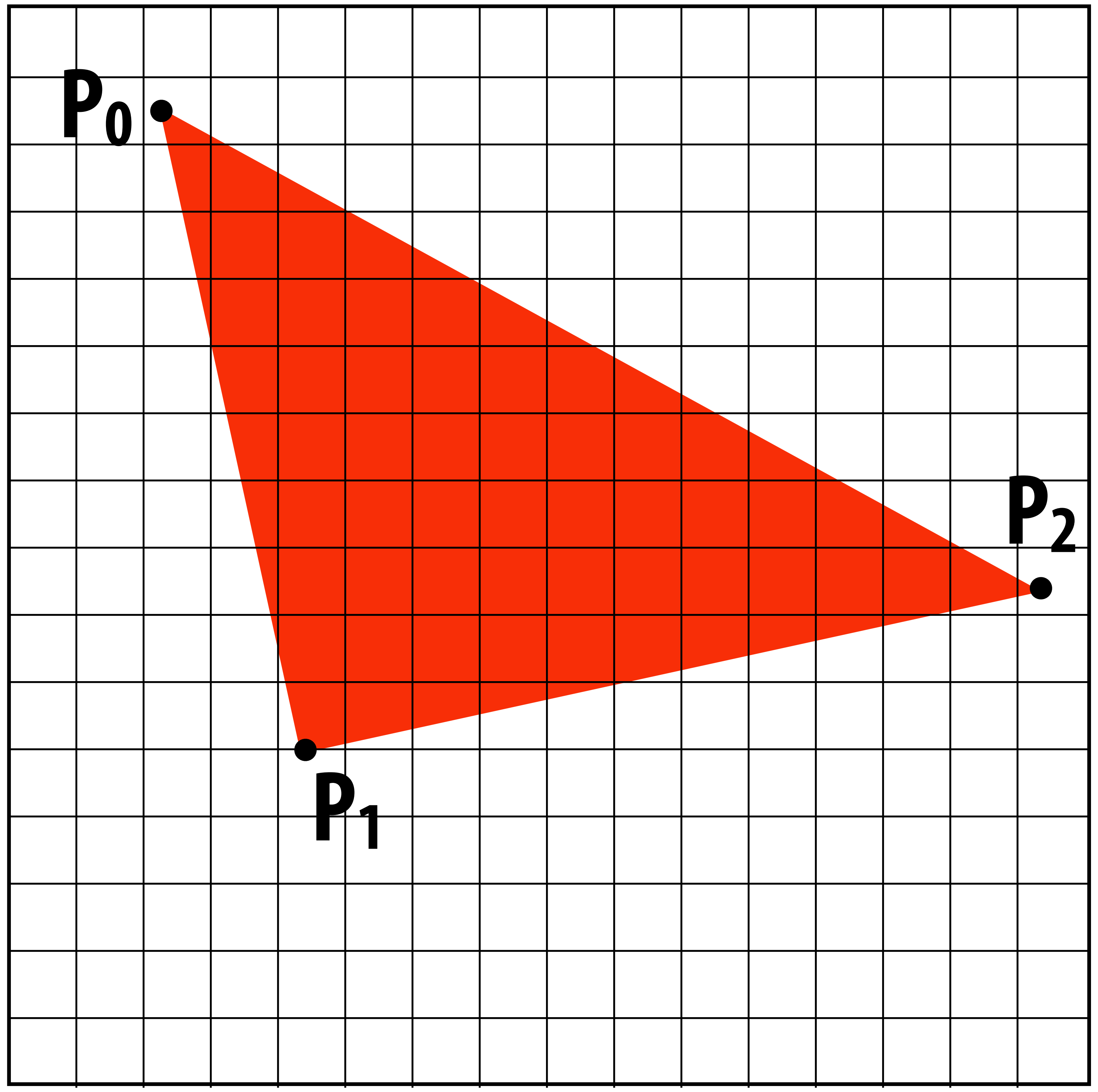
- **What are the prereqs for CS248A?**
  - **You should have the math background: linear algebra (at least MATH 51) and 3D calculus**
  - **You should have the C/C++ coding background (at least CS107)**
  - **CS148 is not a pre-req**

Back to drawing…
We talked about drawing lines, what about triangles?

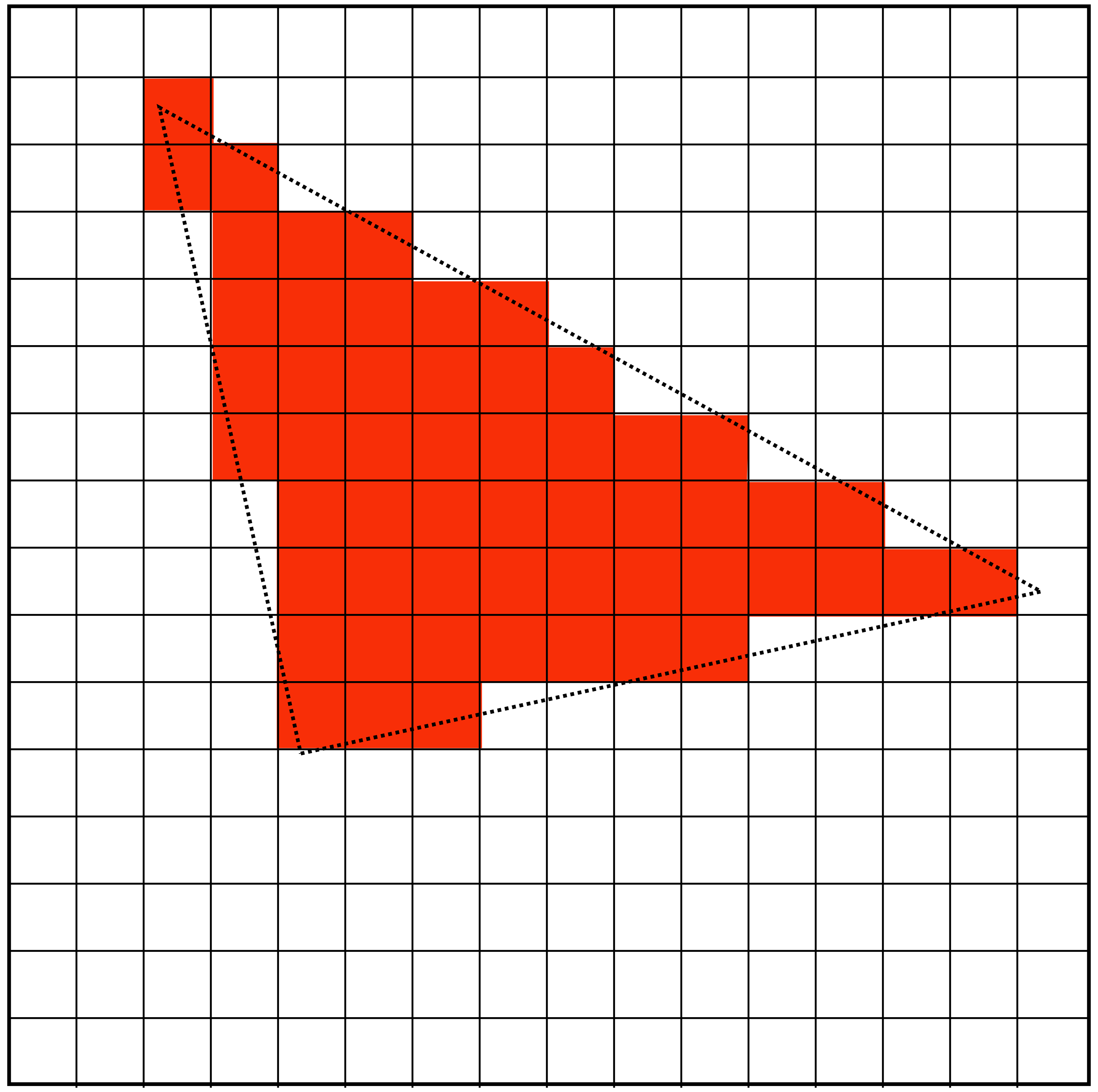
Drawing a triangle ("triangle rasterization”)

(Converting a representation of a triangle into an image)

Input:
2D position of triangle vertices: P_0, P_1, P_2

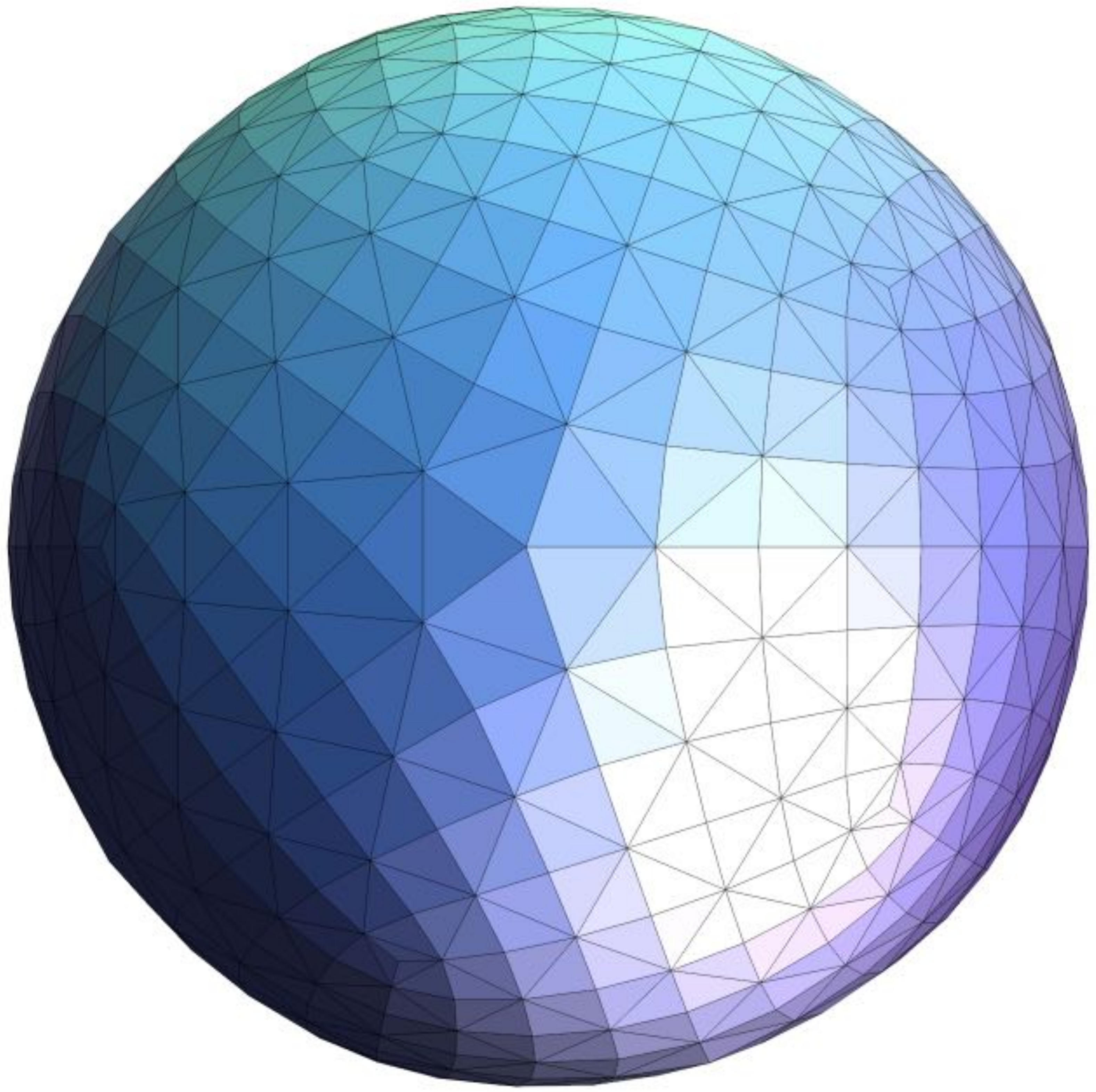


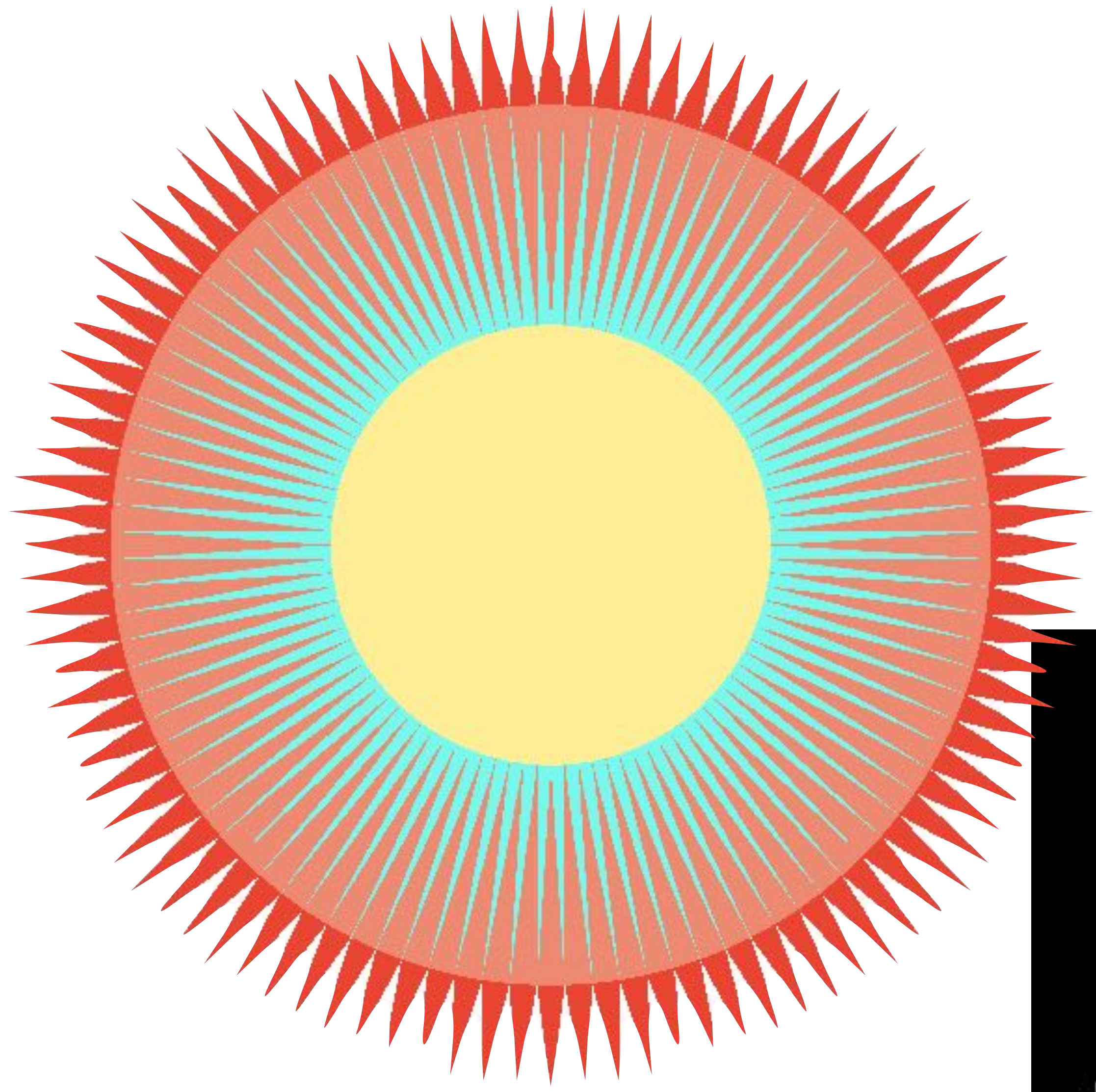
Output:
Set of pixels “covered” by the triangle

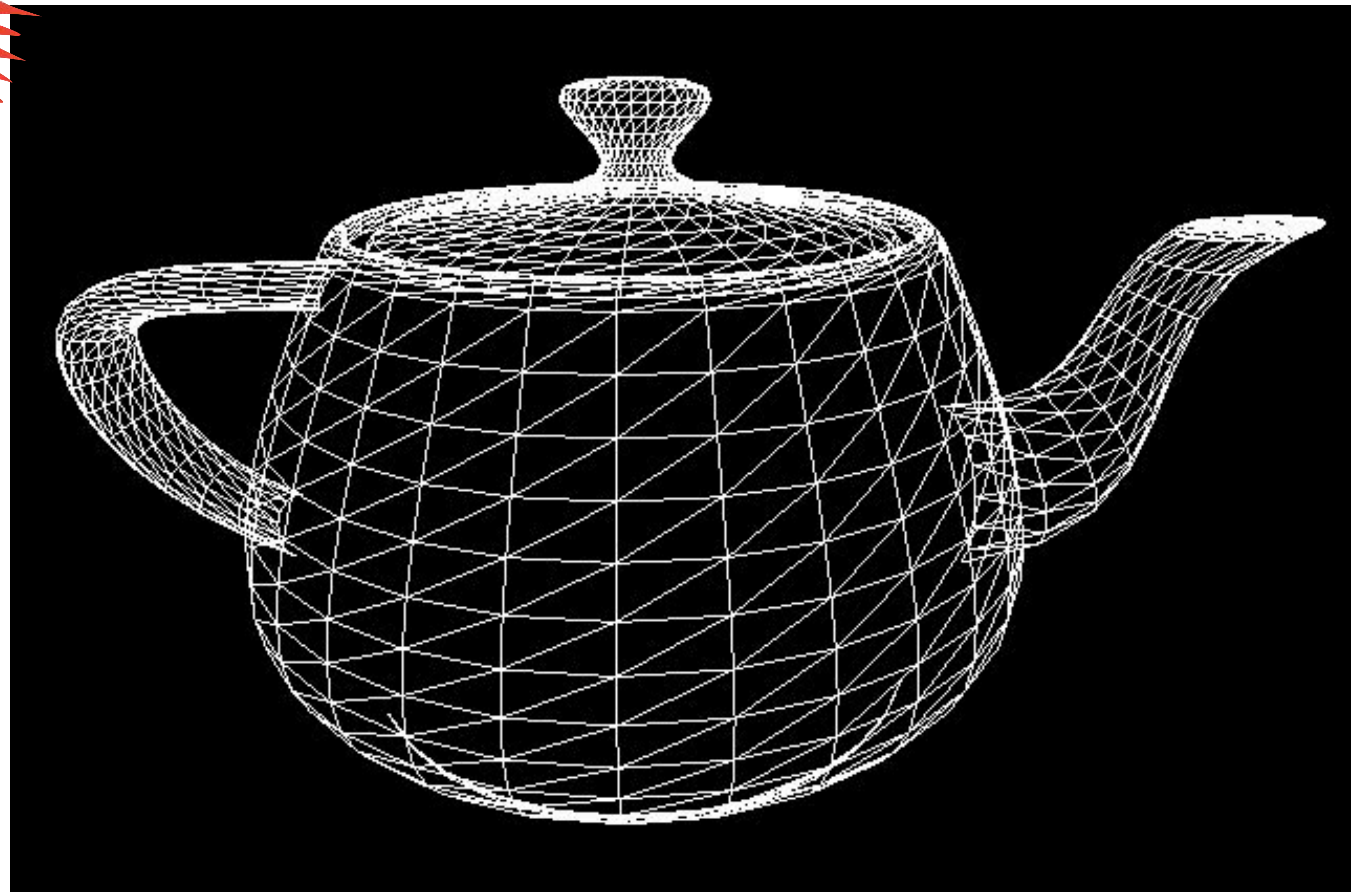


Why triangles?

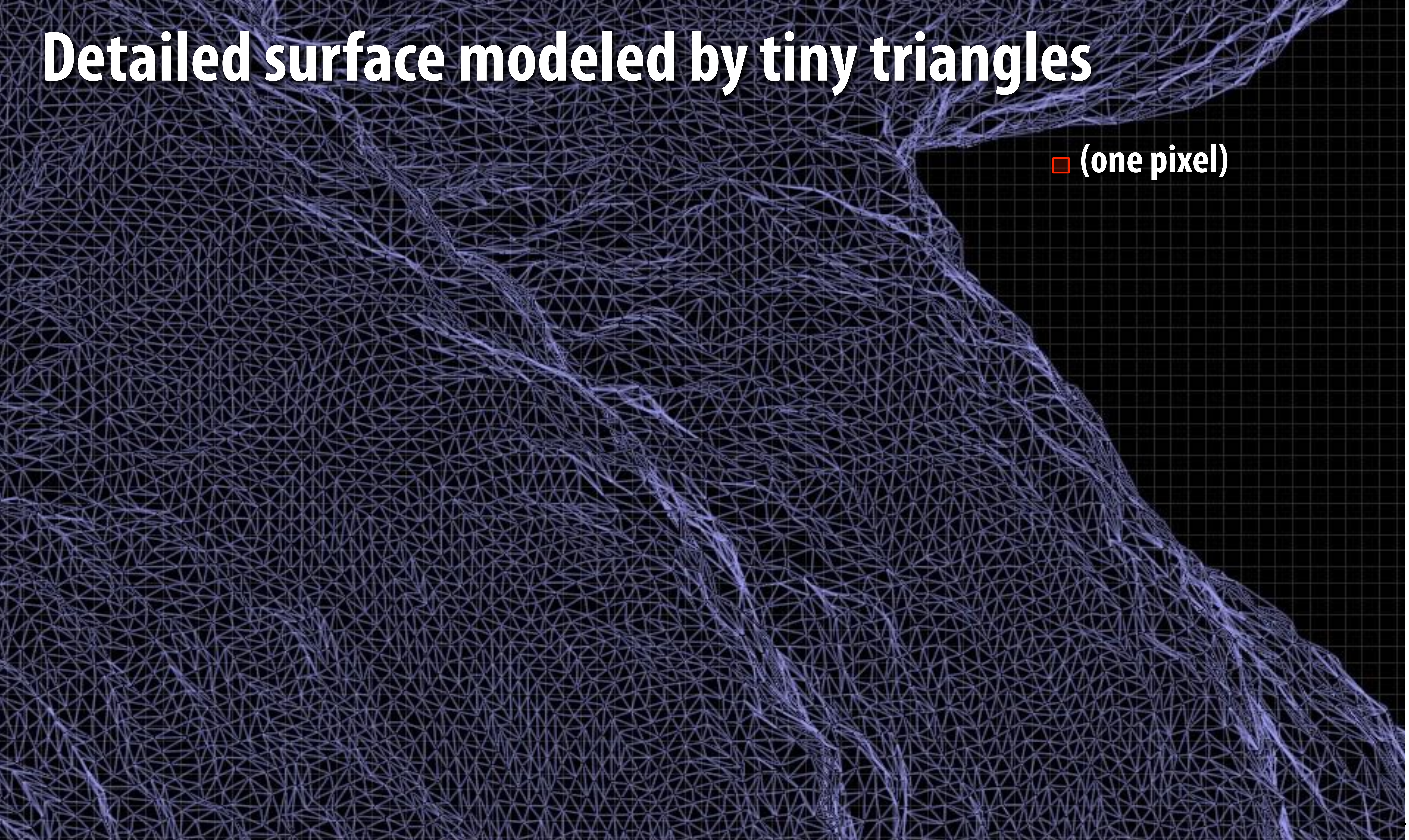
Triangles are a basic block for creating more complex shapes and surfaces



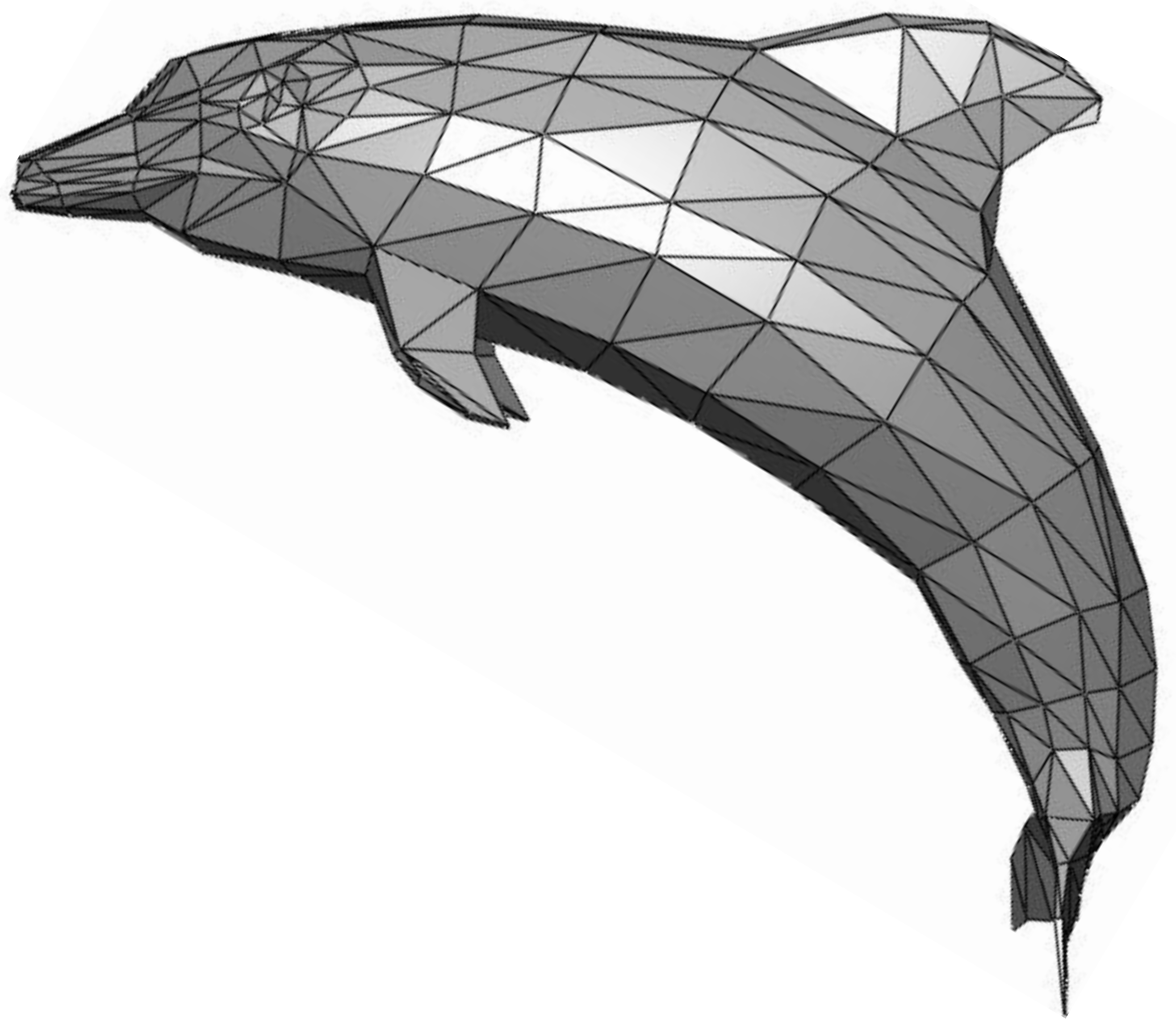




Detailed surface modeled by tiny triangles



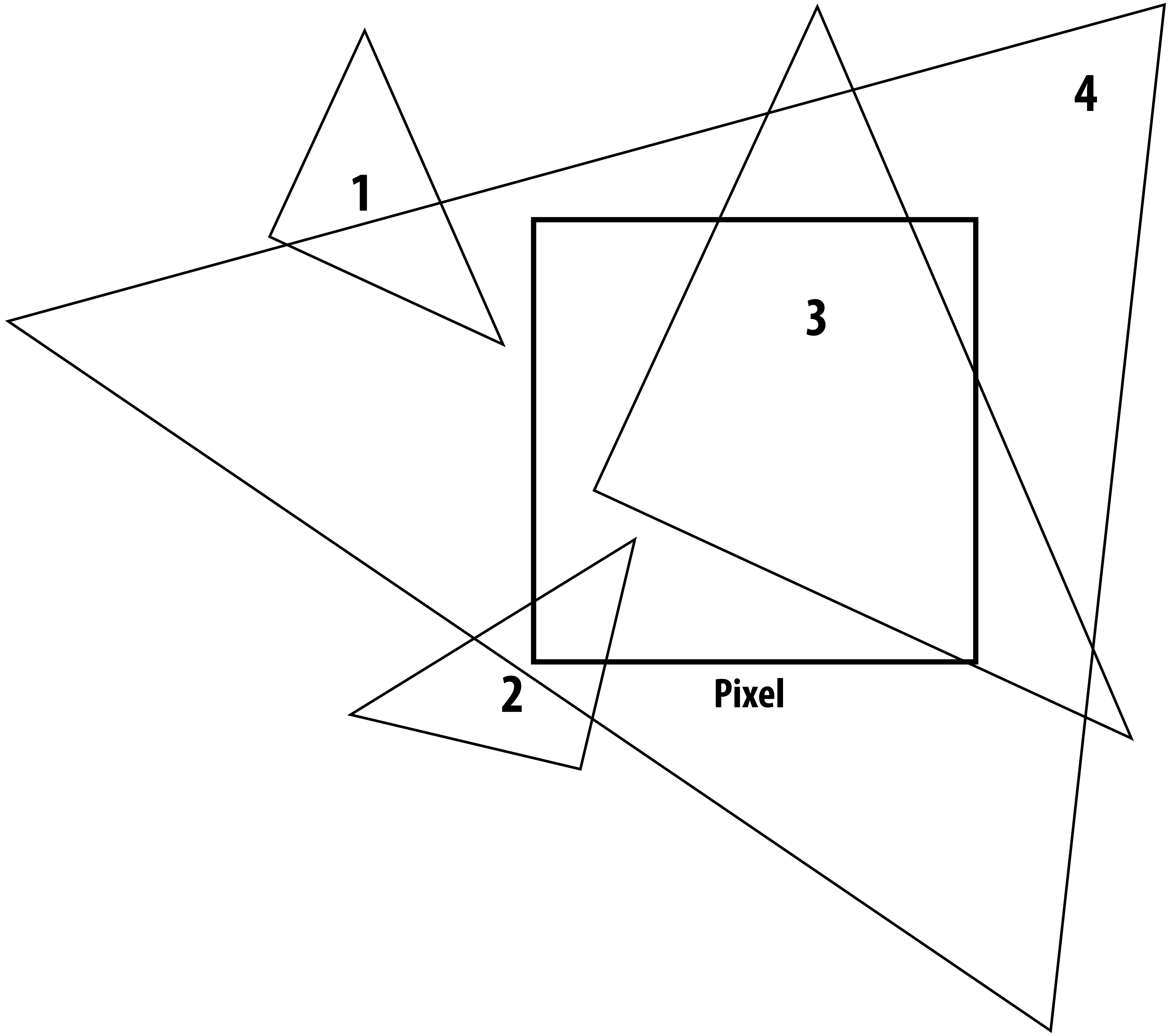
Triangles - a fundamental primitive



- Why triangles?
  - Most basic polygon
    - Can break up other polygons into triangles
    - Allows programs to optimize one implementation
  - Triangles have unique properties
    - Guaranteed to be planar
    - Well-defined interior
    - Well-defined method for interpolating values at vertices over triangle (a topic of a future lecture)

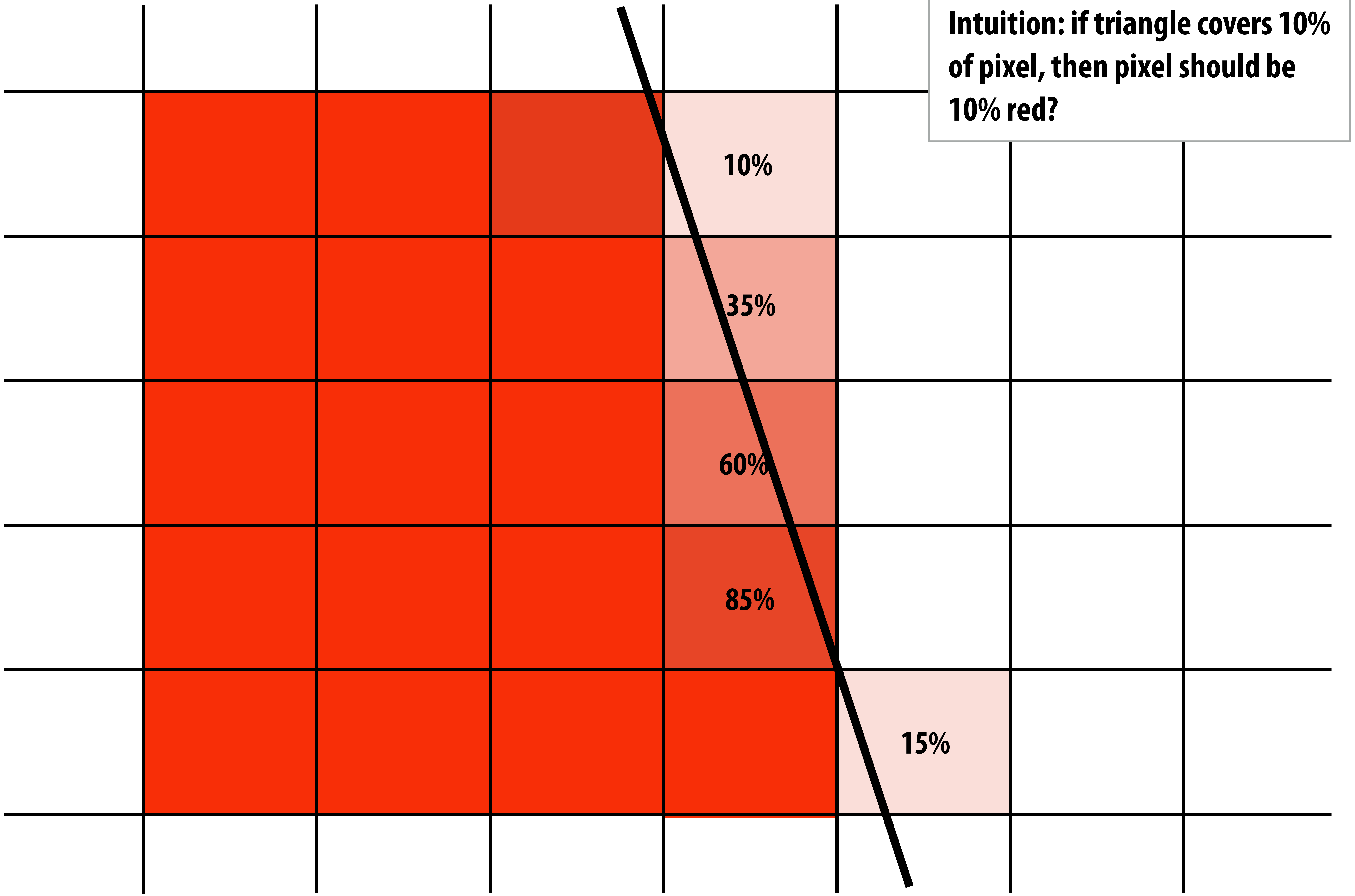
What does it mean for a pixel to be covered by a triangle?

Question: which triangles "cover" this pixel?

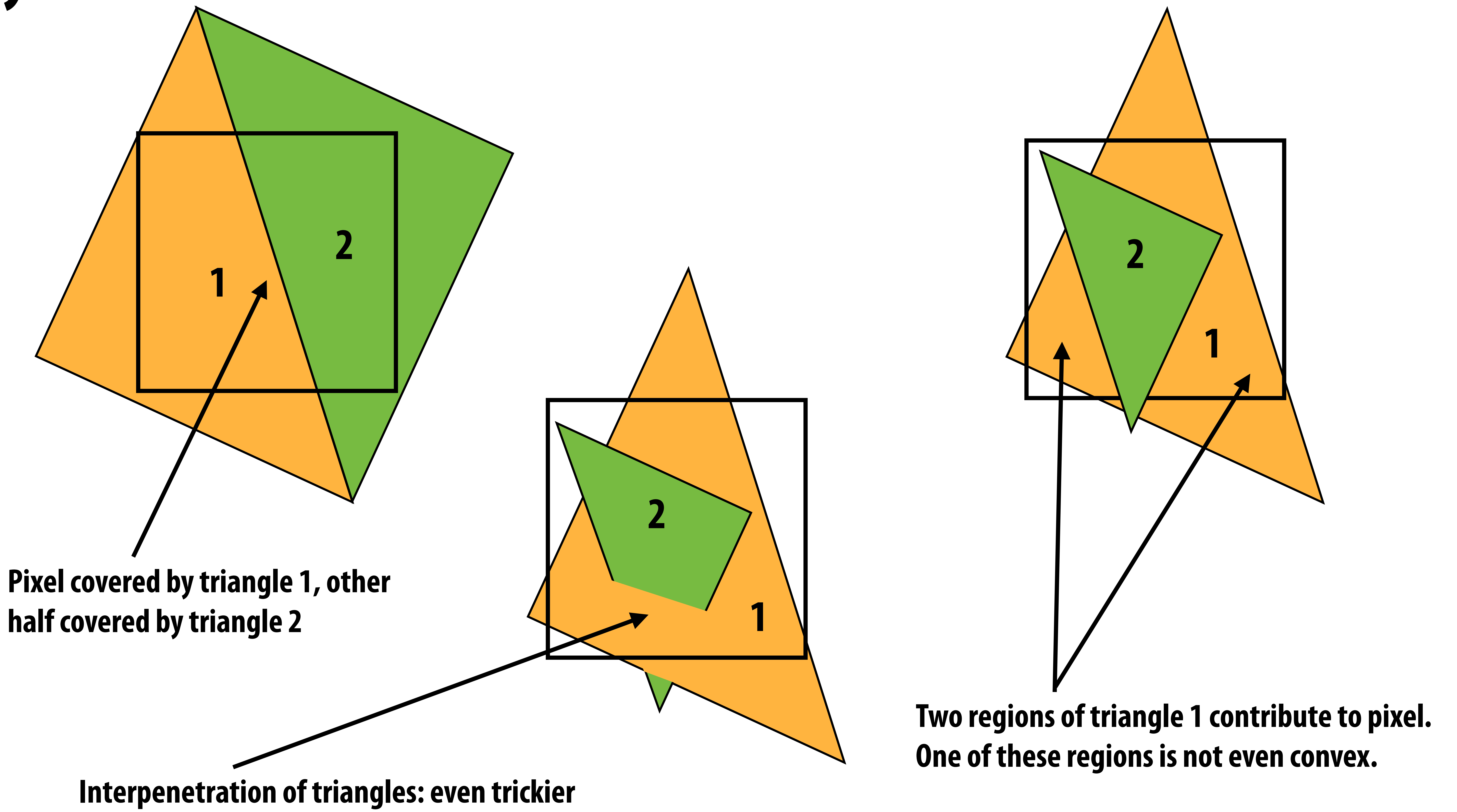


One option: compute fraction of pixel area covered by triangle, then color pixel according to this fraction.

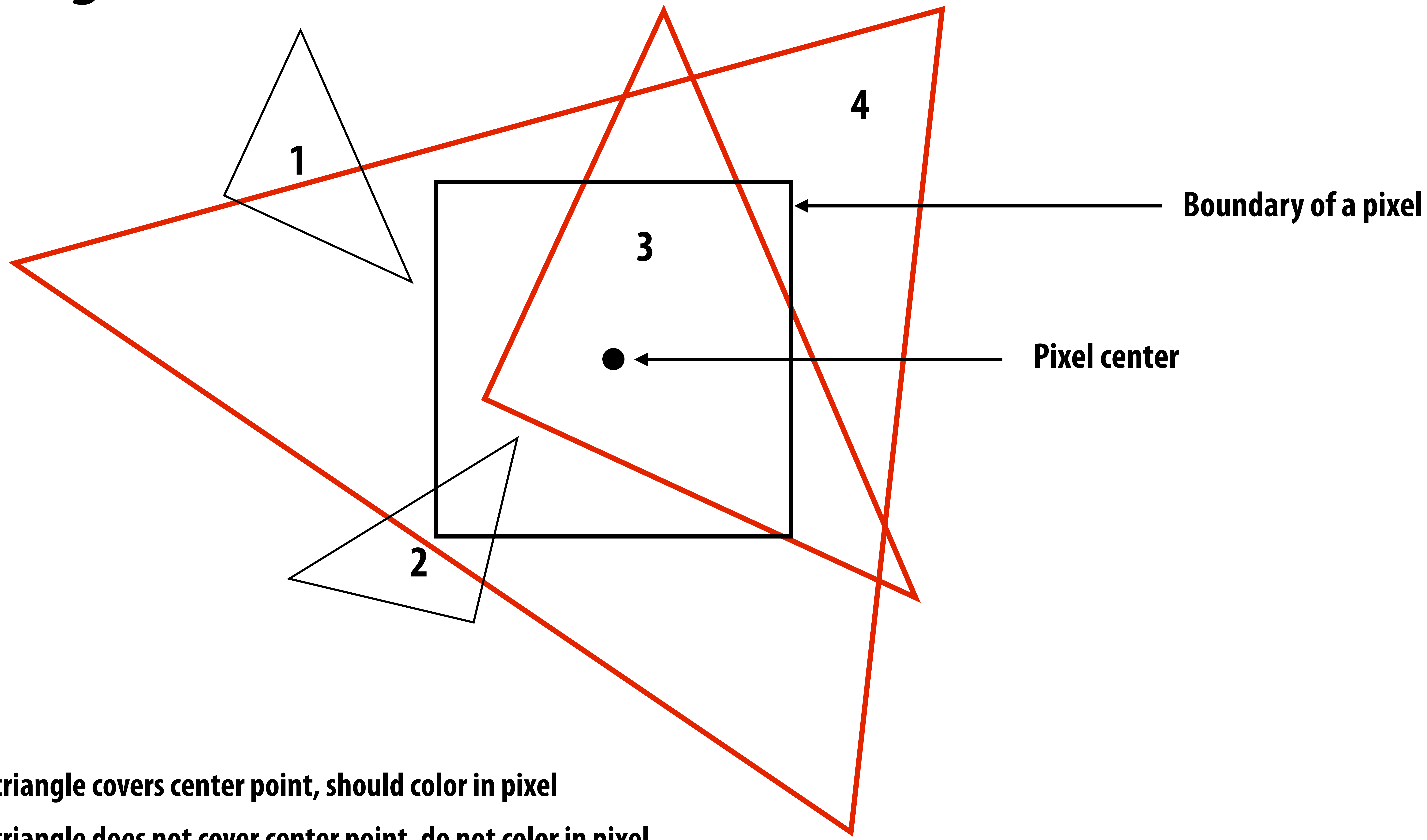
Intuition: if triangle covers 10% of pixel, then pixel should be 10% red?

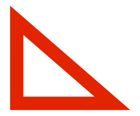


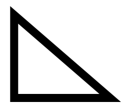
Analytical coverage schemes get tricky when considering occlusion of one triangle by another



Idea: let's call a pixel "inside" the triangle if the pixel center is inside the triangle

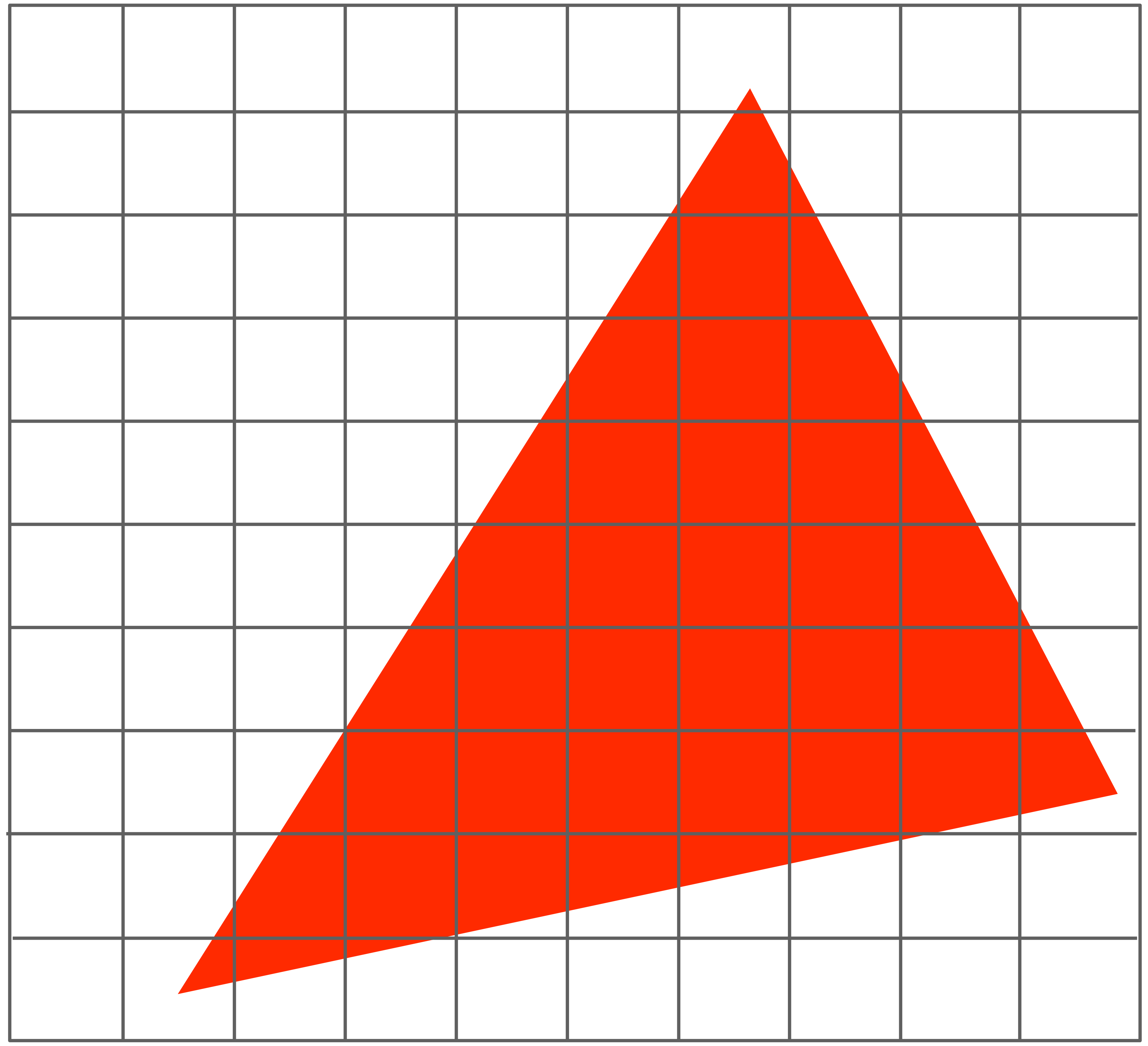



= triangle covers center point, should color in pixel


= triangle does not cover center point, do not color in pixel

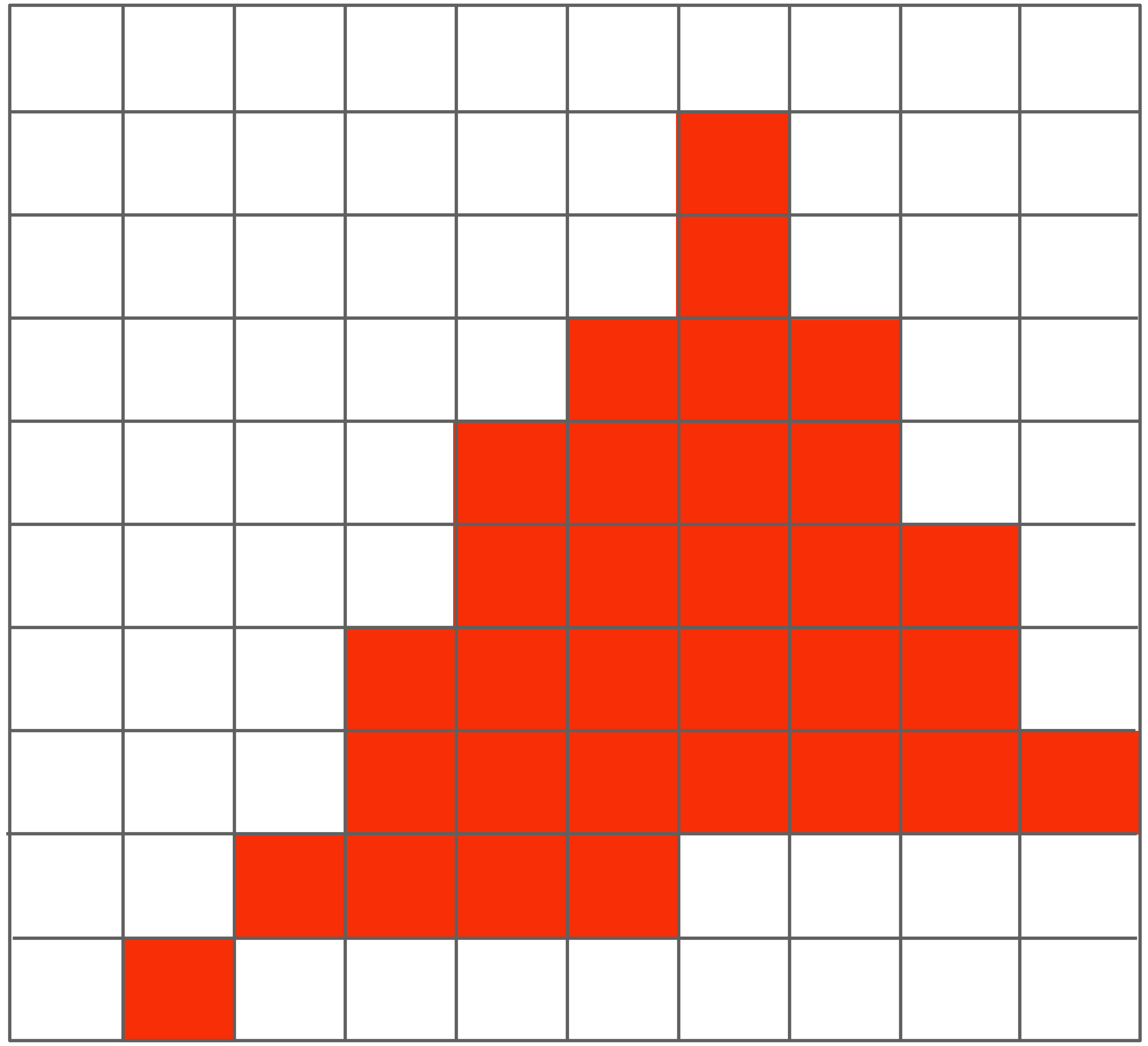
So here's our triangle…

(Overlaid over a pixel grid)



What's wrong with this picture?

(This is the result of rasterizing the triangle using our method)

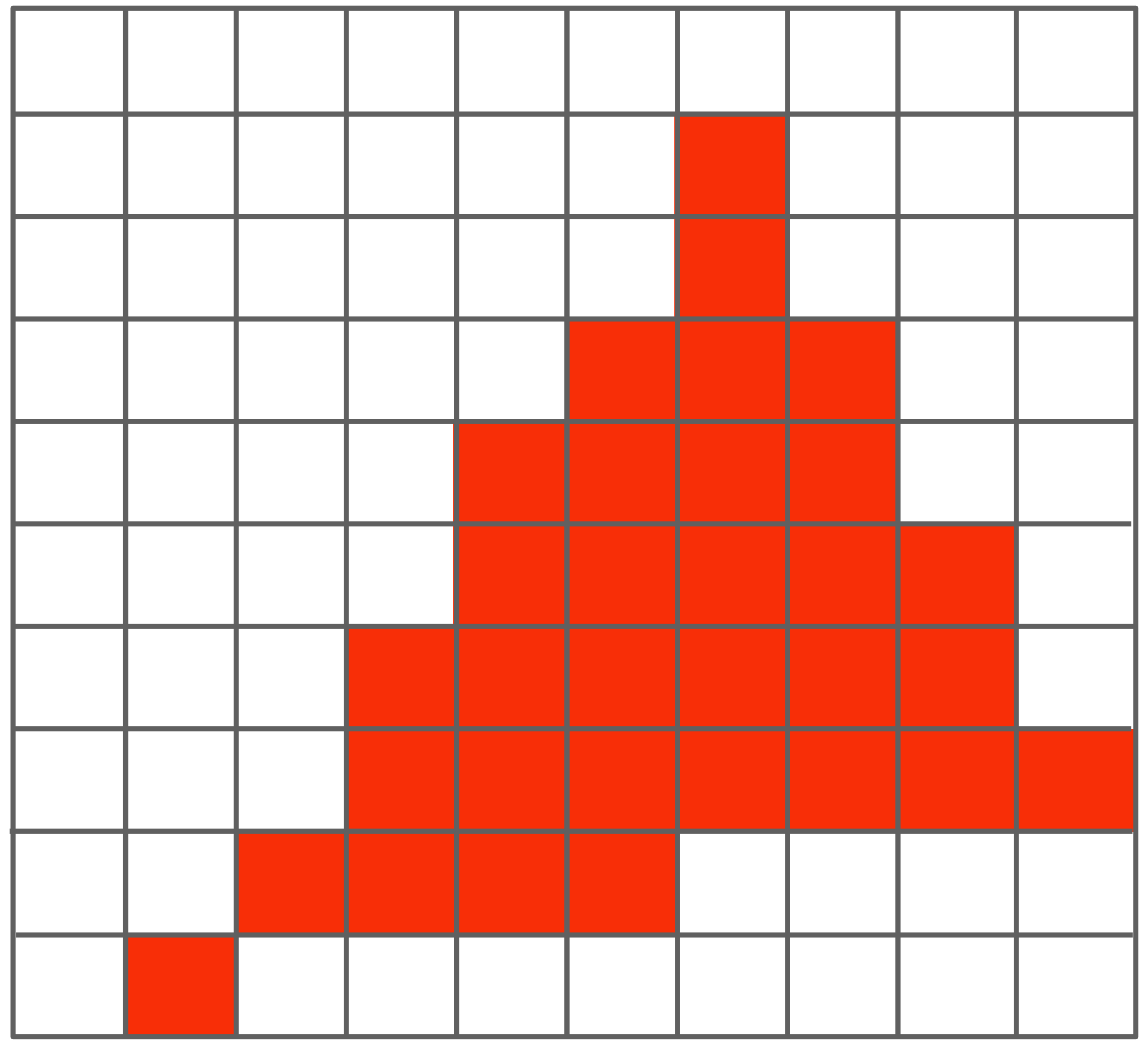


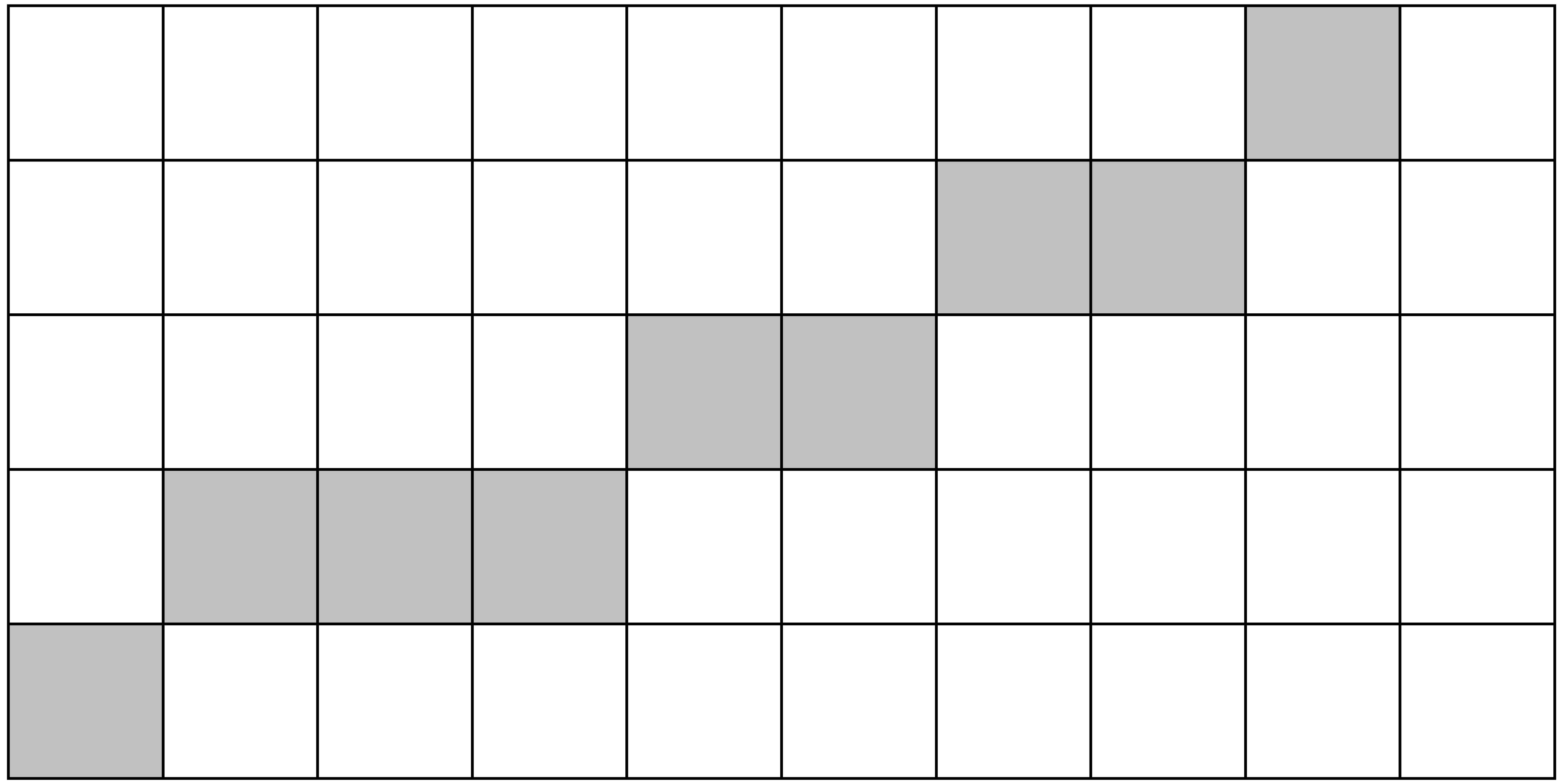
Jaggies!

See you next time!

Next time, we'll talk about drawing a triangle in more rigor

- **How do we compute if a point is inside a triangle?**
- **What's up with these "jagged" lines and triangle edges?**
- **What can we do about it to improve image quality?**





Slide acknowledgements:
Thanks to Keenan Crane and Ren Ng