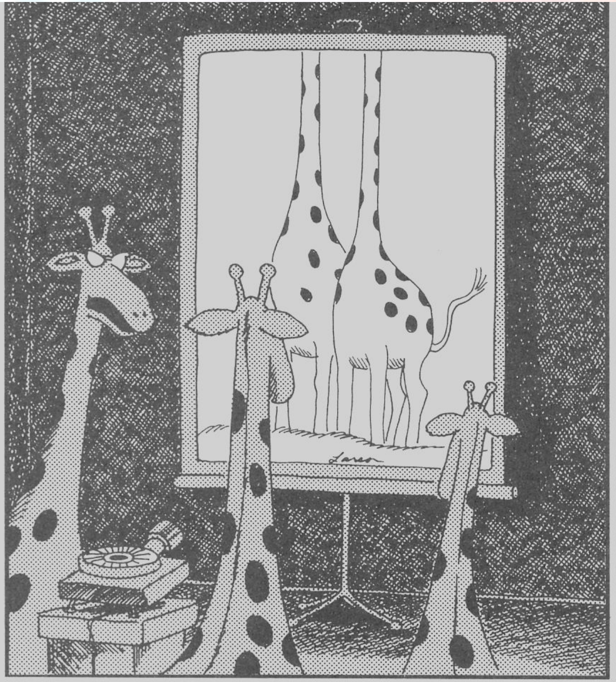
CSCI 4530/6530 Advanced Computer Graphics

https://www.cs.rpi.edu/~cutler/classes/advancedgraphics/S25/

Lecture 17: The Traditional Graphics Pipeline



"Oh, lovely — just the hundredth time you've managed to cut everyone's head off."

Importance of Titles & Common Terminology

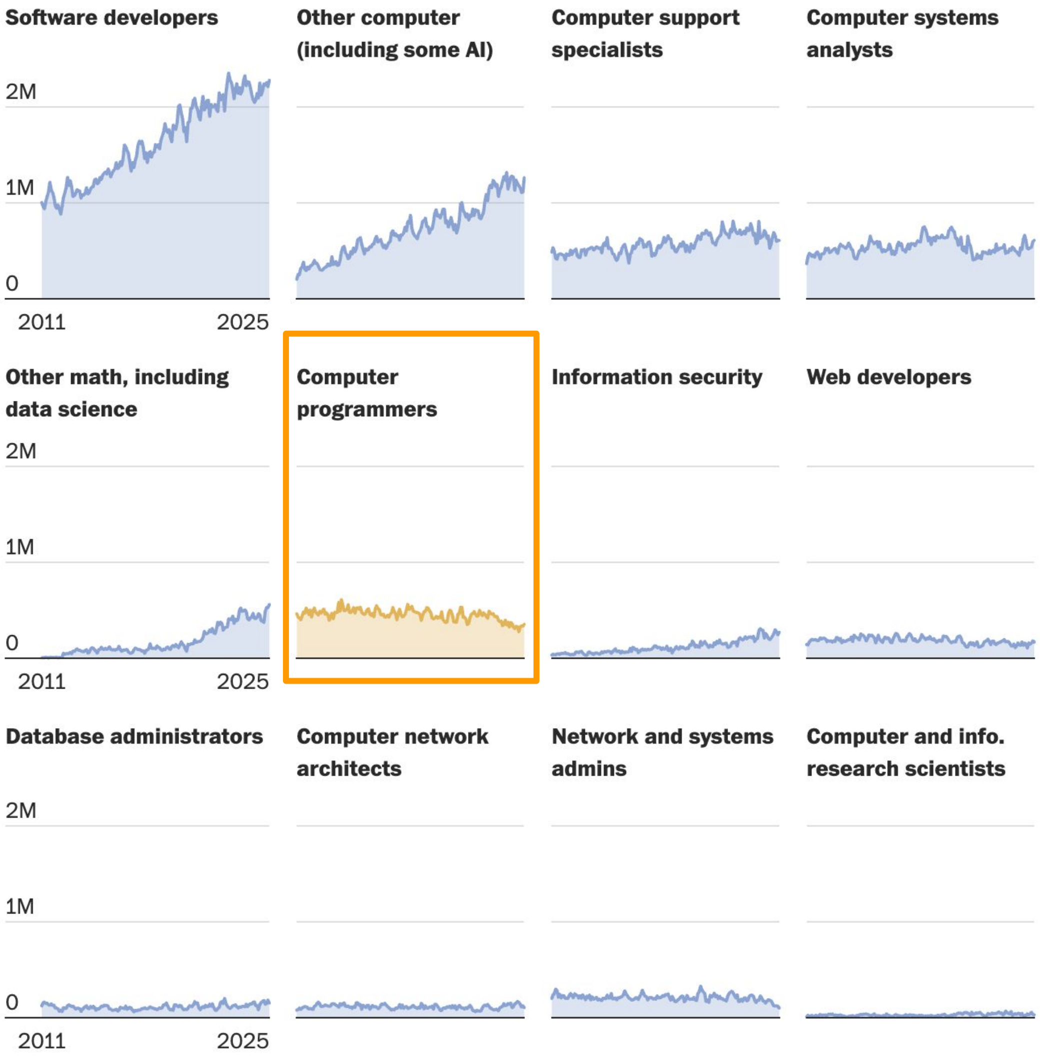
- What's on your resume? Does it matter?
  - "programmer", "software engineer", "developer", "tester", "QA (quality assurance)", "AI/ML", …

Quick Facts: Software Developers, Quality Assurance Analysts, and Testers	
2023 Median Pay	$130,160 per year $62.58 per hour
Typical Entry-Level Education	Bachelor's degree
Work Experience in a Related Occupation	None
On-the-job Training	None
Number of Jobs, 2023	1,897,100
Job Outlook, 2023-33	17% (Much faster than average)
Employment Change, 2023-33	327,900

Quick Facts: Computer Programmers	
2023 Median Pay	$99,700 per year $47.94 per hour
Typical Entry-Level Education	Bachelor's degree
Work Experience in a Related Occupation	None
On-the-job Training	None
Number of Jobs, 2023	139,400
Job Outlook, 2023-33	-10% (Decline)
Employment Change, 2023-33	-13,400

https://www.bls.gov/ooh/computer-and-information-technology/software-developers.htm
https://www.bls.gov/ooh/computer-and-information-technology/computer-programmers.htm

“More than a quarter of computer-programming jobs just vanished. What happened?”

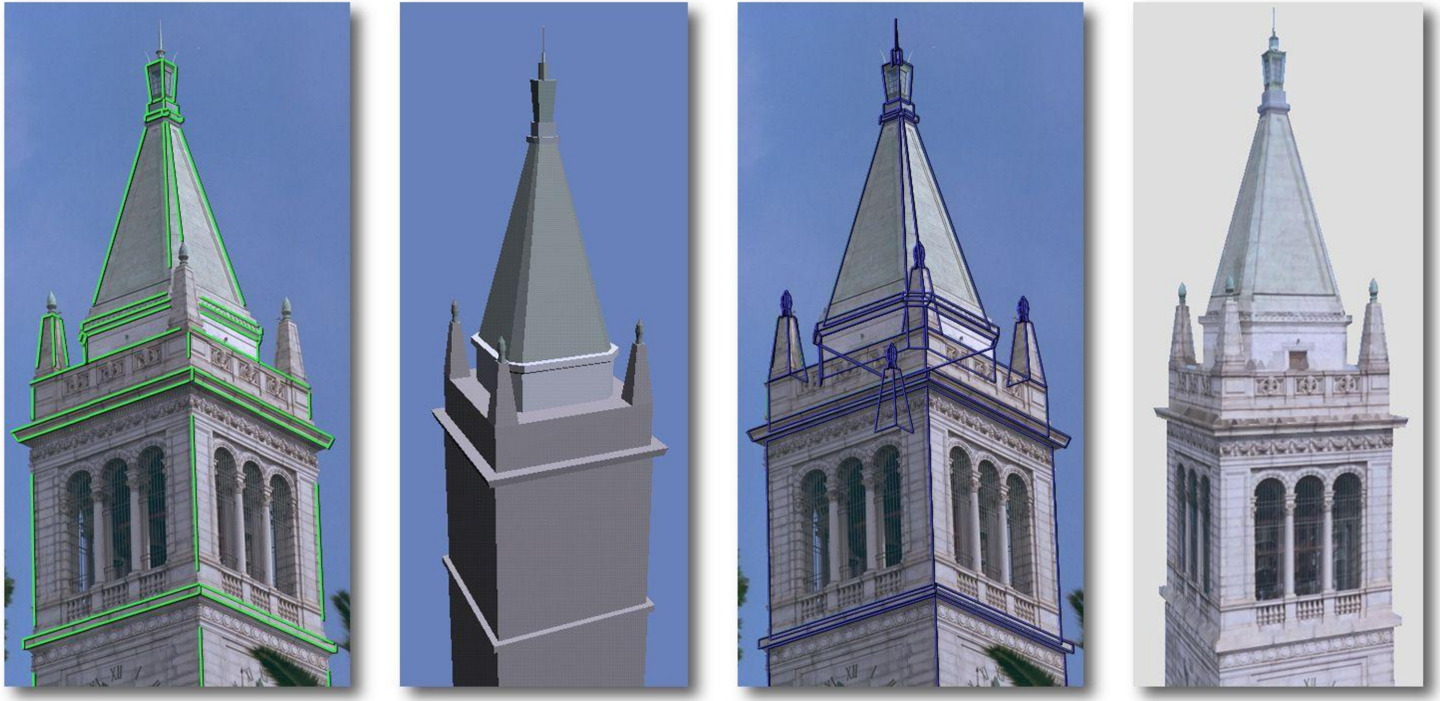


https://www.washingtonpost.com/business/2025/03/14/programming-jobs-lost-artificial-intelligence/

Facade, Debevec et al. 1997

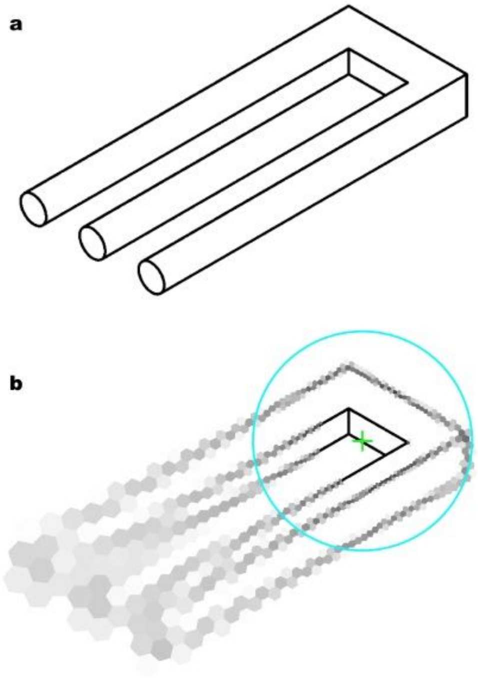
Modeling and Rendering Architecture from Photographs

Debevec, Taylor, and Malik 1996



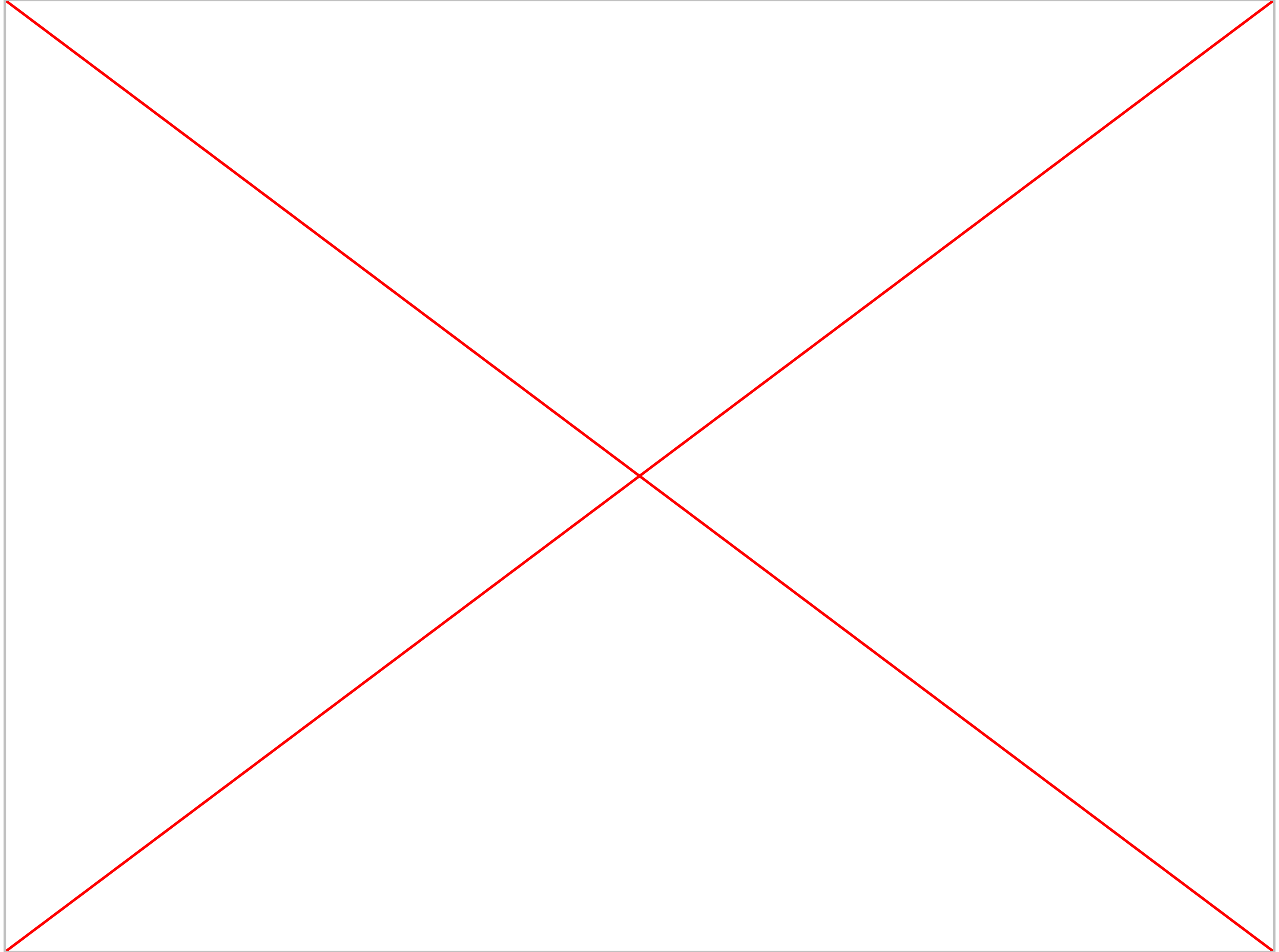
Original photograph with marked edges | Recovered model | Model edges projected onto photograph | Synthetic rendering

Belvedere, M.C. Escher, 1958



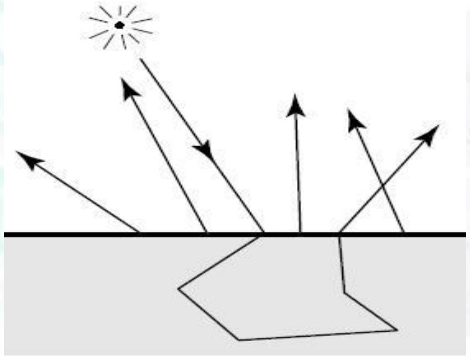
“Combining Deep Learning and Active Contours Opens The Way to Robust, Automated Analysis of Brain Cytoarchitectonics”, Thierbach et al, 2018

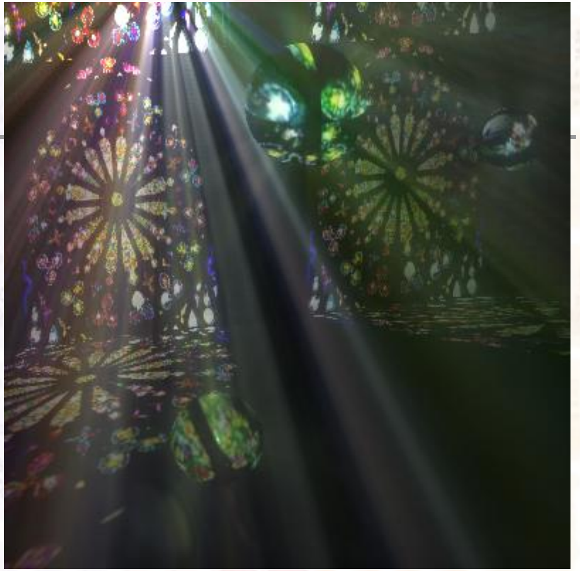
Escher's Belvedere, Sachiko Tsuruno, 1997

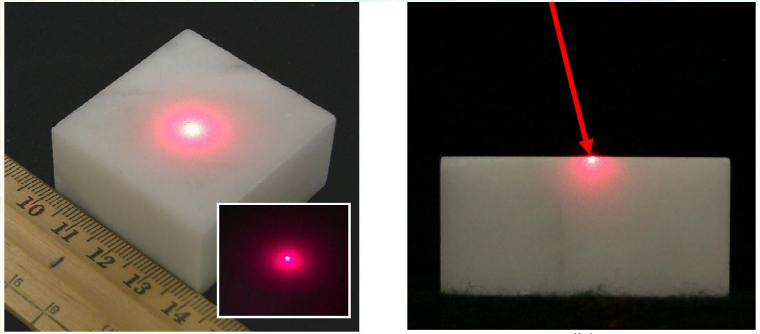


Last Time?

- Participating Media
- Measuring BRDFs
- 3D Digitizing & Scattering
- BSSRDFs
- Monte Carlo Simulation
- Dipole Approximation

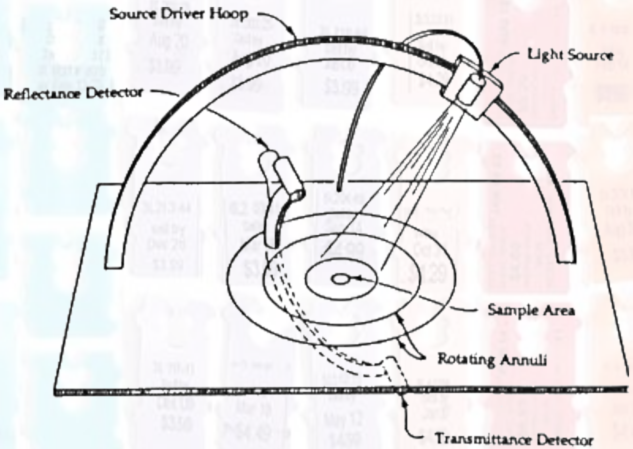






(a) (b)

Figure 1: Diffusion in a sample of Carrara Statuario marble.



Today

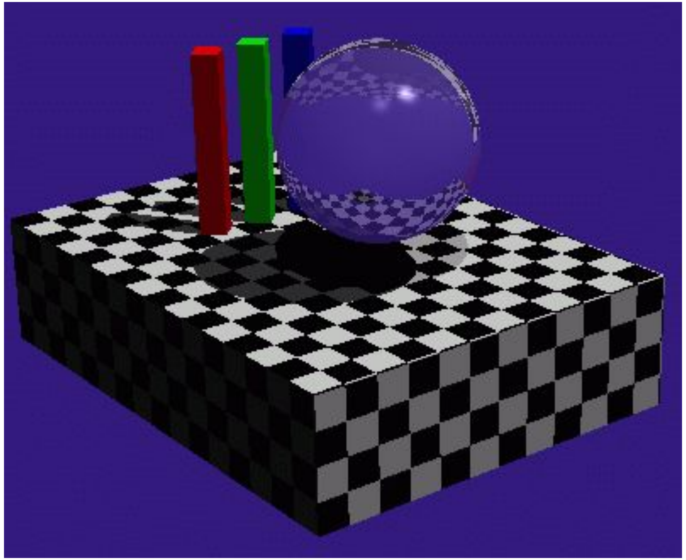
- Ray Casting / Tracing vs. Scan Conversion
- Traditional Graphics Pipeline
- Clipping
- Rasterization / Scan Conversion
- Readings for Today
- Readings for Next Time

Ray Casting / Tracing

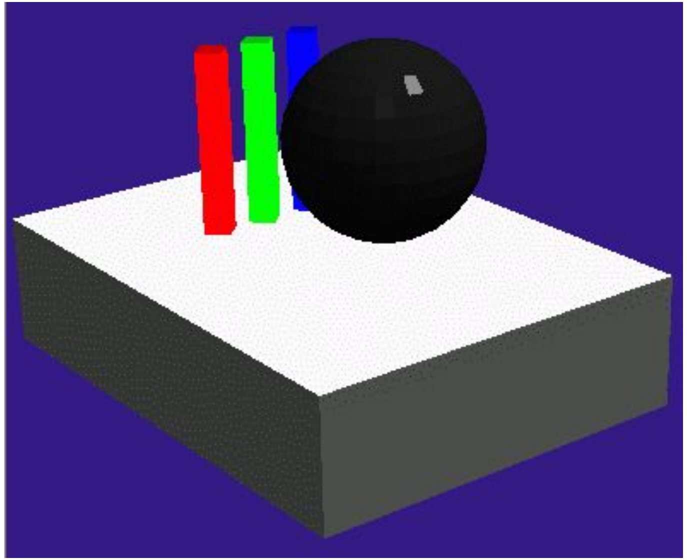
- Advantages?
  - Smooth variation of normal, exact silhouettes
  - Generality: can render anything that can be intersected with a ray
  - Atomic operation, allows recursion
- Disadvantages?
  - Time complexity (*n* objects, *h**w* pixels, *b* bounces)
  - Usually too slow for interactive applications
  - Hard to implement in hardware
    (lacks computation coherence, must fit entire scene in memory)

How Do We Render Interactively?

- Use graphics hardware (the graphics pipeline), via OpenGL, MesaGL, DirectX, or Metal
- Most global effects available in ray tracing will be sacrificed, but some can be approximated



Ray Tracing



Graphics Pipeline (OpenGL)

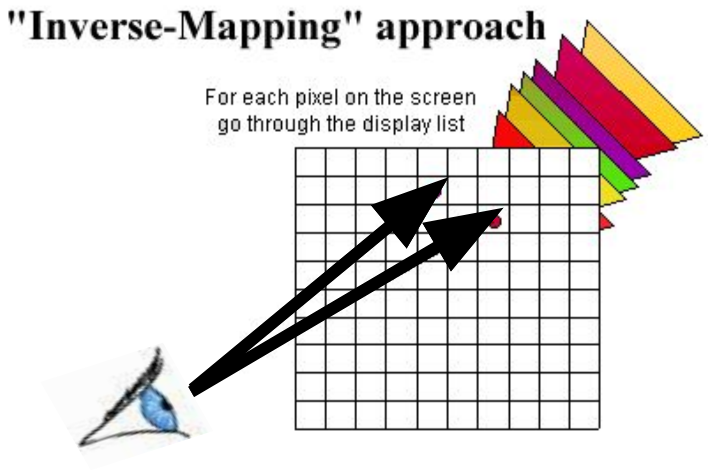
Ray Casting vs. Rendering Pipeline

Ray Casting

```
For each pixel
    For each object
```

Send pixels into the scene

Discretize first

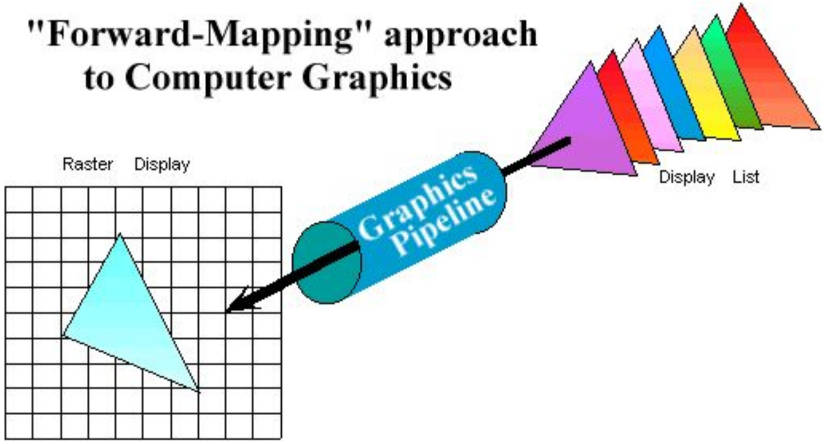


Rendering Pipeline

```
For each triangle
    For each pixel
```

Project scene to the pixels

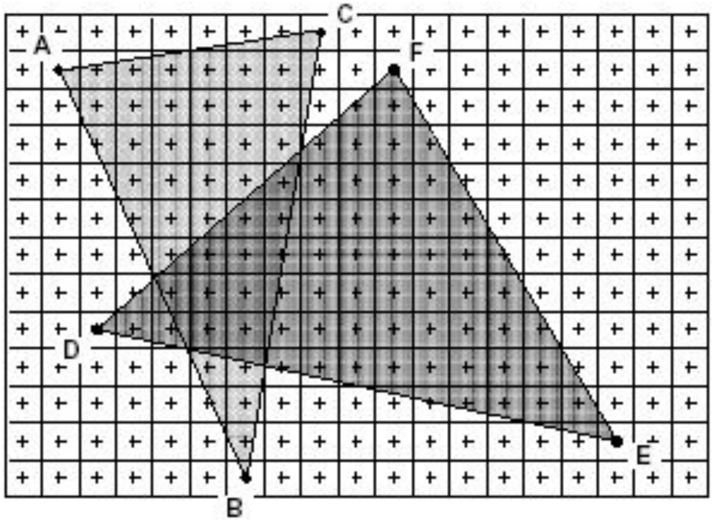
Discretize last



Scan Conversion (Rendering Pipeline)

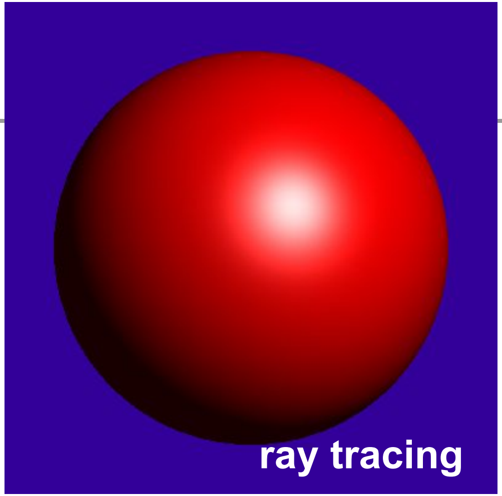
- Given a primitive's vertices & the illumination at each vertex:
- Figure out which pixels to "turn on" to render the primitive
- Interpolate the illumination values to "fill in" the primitive
- At each pixel, keep track of the closest primitive (z-buffer)

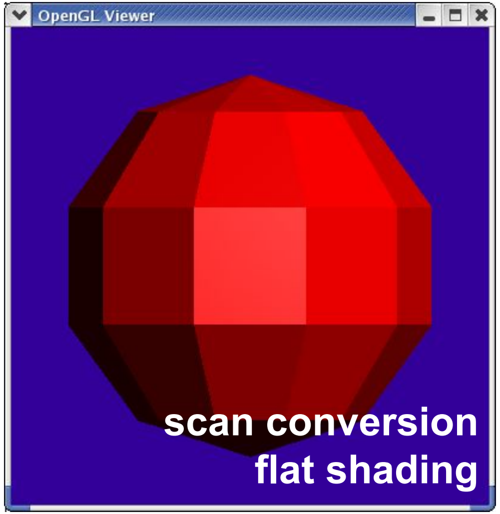
```
glBegin(GL_TRIANGLES)
glNormal3f(...)
glVertex3f(...)
glVertex3f(...)
glVertex3f(...)
glEnd();
```

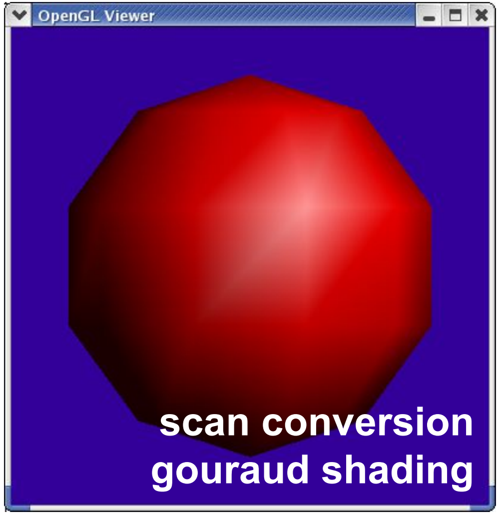


Limitations of Scan Conversion

- Restricted to scan-convertible primitives
  - Must “polygonize” all objects
- Faceting, shading artifacts
- Effective resolution is hardware dependent
- No handling of shadows, reflection, transparency
- Problem of overdraw (high depth complexity)
- What if there are many more triangles than pixels?







Ray Casting vs. Rendering Pipeline

Ray Casting

```
For each pixel
    For each object
```

- Whole scene must be in memory
- Depth complexity: w/ spatial acceleration data structures no computation needed for hidden parts
- Atomic computation
- More general, more flexible
  - Primitives, lighting effects, adaptive antialiasing

Rendering Pipeline

```
For each triangle
    For each pixel
```

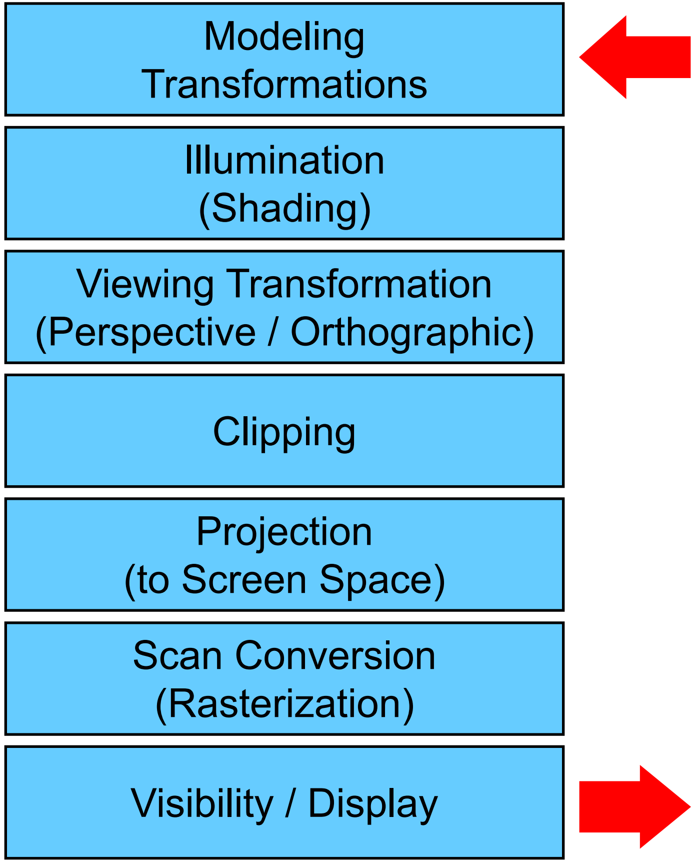
- Primitives processed one at a time
- Coherence: geometric transforms for vertices only
- Early stages involve analytic processing
- Computation increases with depth of the pipeline
  - Good bandwidth/computation ratio
- Sampling occurs late in the pipeline
- Minimal state required

Questions?

Today

- Ray Casting / Tracing vs. Scan Conversion
- Traditional Graphics Pipeline
- Clipping
- Rasterization / Scan Conversion
- Readings for Today
- Readings for Next Time

The Graphics Pipeline



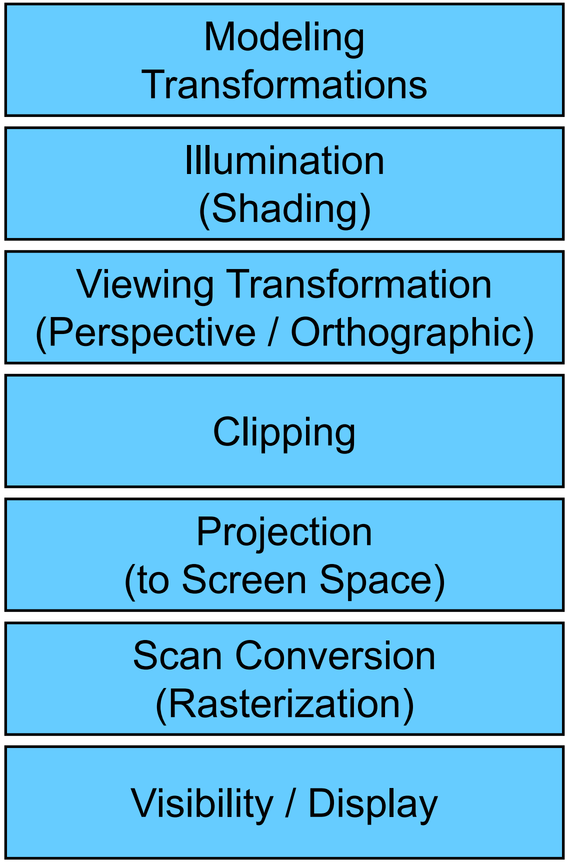
INPUT

- Geometric Model: description of all objects, surfaces, & light sources
- Lighting/Material Model: object & light properties, interactions (reflections)
- Synthetic Viewpoint / Camera: eye position & view frustum
- Raster Viewport: pixel grid onto which image plane is mapped

OUTPUT

- Colors/Intensities for framebuffer display (e.g., 24-bit RGB value at each pixel

The Graphics Pipeline

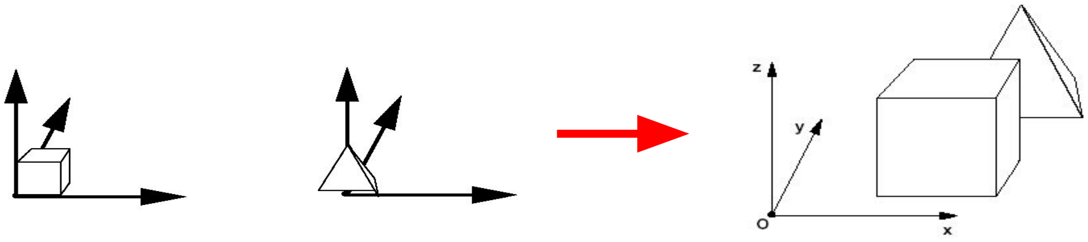


- Primitives are processed in a series of stages
- Each stage forwards its result on to the next stage
- The pipeline can be drawn and implemented in different ways
- Some stages may be in hardware, others in software
- Optimizations & additional programmability are available at some stages

The Graphics Pipeline: Modeling Transformations

- **Modeling Transformations**
- Illumination (Shading)
- Viewing Transformation (Perspective / Orthographic)
- Clipping
- Projection (to Screen Space)
- Scan Conversion (Rasterization)
- Visibility / Display

- 3D models defined in their own coordinate system (object space)
- Modeling transforms orient the models within a common coordinate frame (world space)

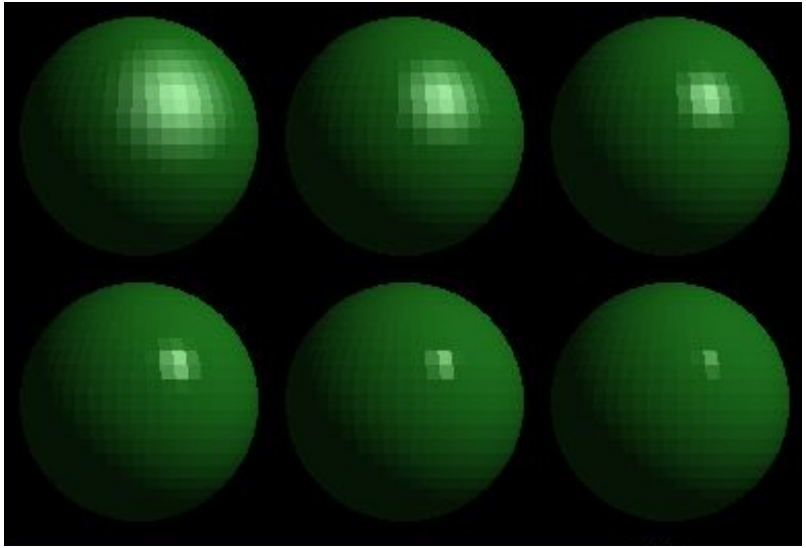


Object space **World space**

The Graphics Pipeline: Illumination (Shading)

- Modeling Transformations
- **Illumination (Shading)**
- Viewing Transformation (Perspective / Orthographic)
- Clipping
- Projection (to Screen Space)
- Scan Conversion (Rasterization)
- Visibility / Display

- Vertices lit (shaded) according to material properties, surface properties (normal) and light sources
- Local lighting model
- Diffuse, Ambient, Phong, etc.

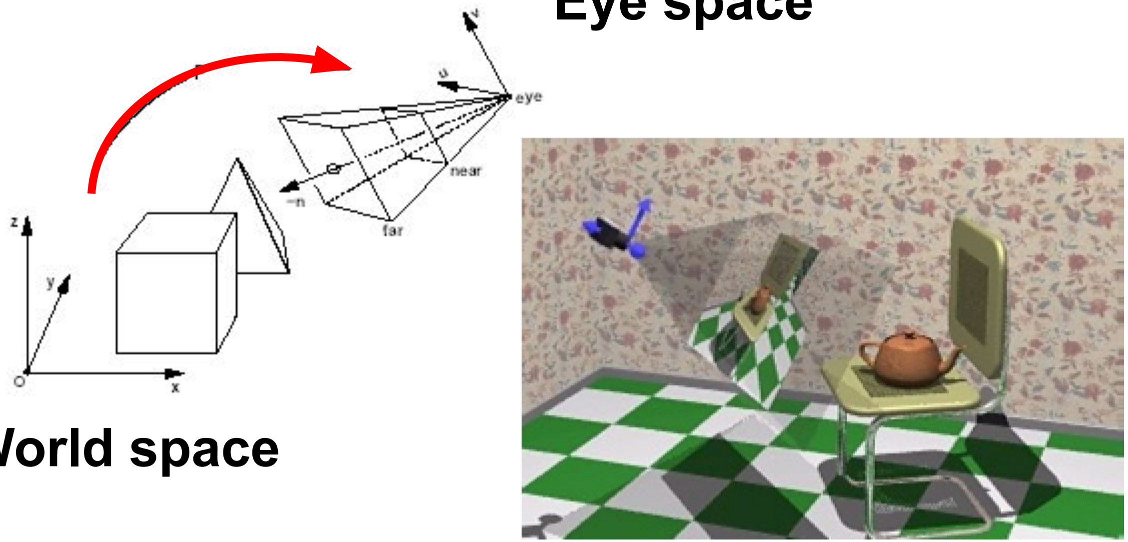


The Graphics Pipeline: Viewing Transformation

- Modeling Transformations
- Illumination (Shading)
- **Viewing Transformation (Perspective / Orthographic)**
- Clipping
- Projection (to Screen Space)
- Scan Conversion (Rasterization)
- Visibility / Display

- Maps world space to eye space
- Viewing position is transformed to origin & direction is oriented along some axis (usually z)

Eye space

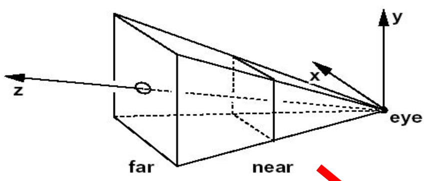


World space

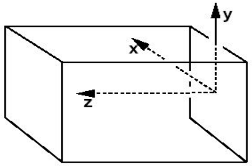
The Graphics Pipeline: Clipping

- Modeling Transformations
- Illumination (Shading)
- Viewing Transformation (Perspective / Orthographic)
- **Clipping**
- Projection (to Screen Space)
- Scan Conversion (Rasterization)
- Visibility / Display

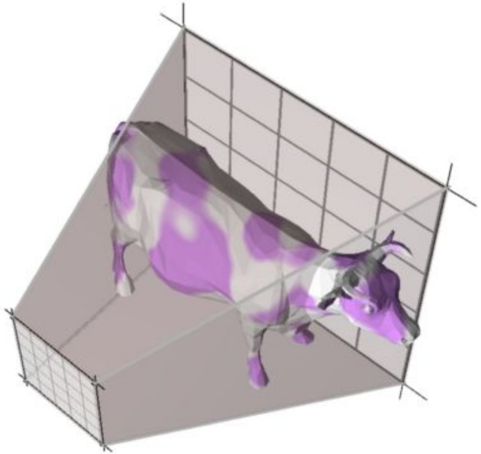
- Transform to Normalized Device Coordinates (NDC)
- Portions of the object outside the view volume (view frustum) are removed



Eye space



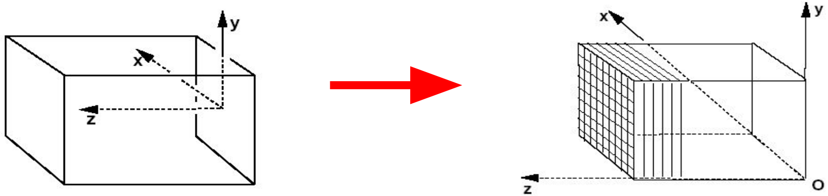
NDC



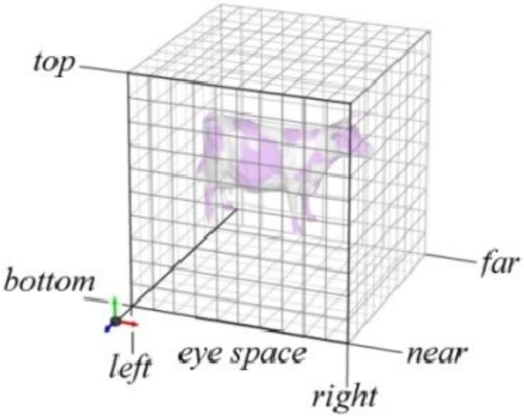
The Graphics Pipeline: Projection to Screen Space

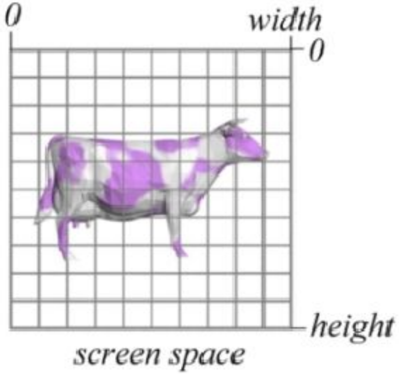
- Modeling Transformations
- Illumination (Shading)
- Viewing Transformation (Perspective / Orthographic)
- Clipping
- **Projection (to Screen Space)**
- Scan Conversion (Rasterization)
- Visibility / Display

- The objects are projected to the 2D image place (screen space)



NDC → **Screen Space**

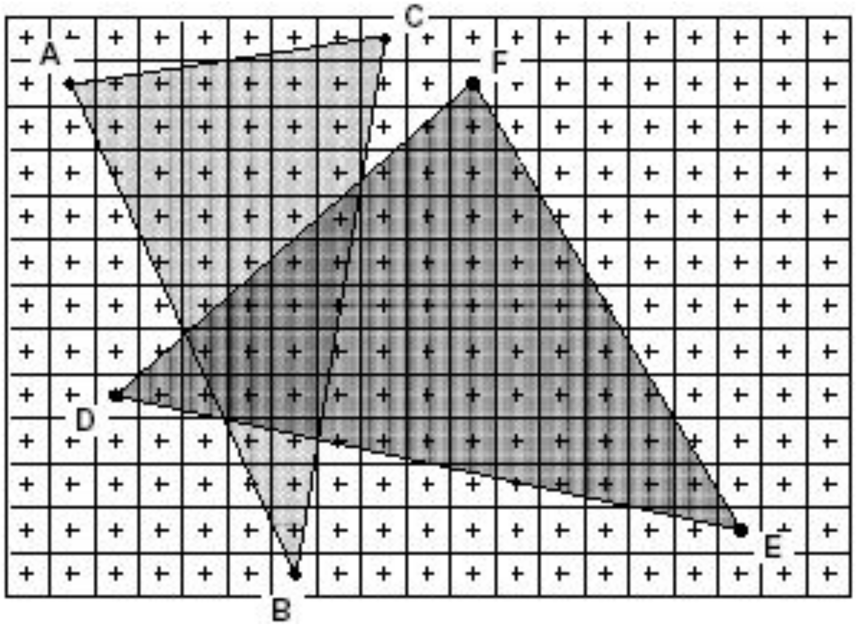




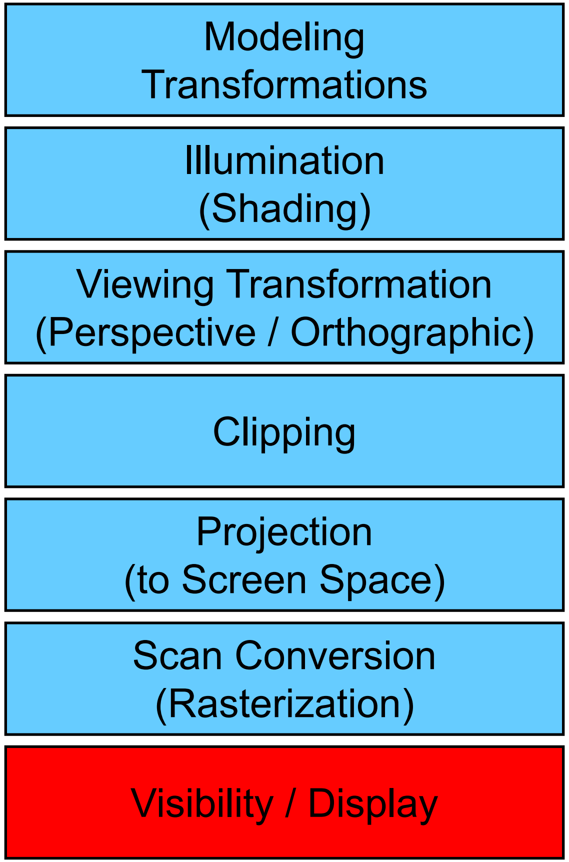
The Graphics Pipeline: Scan Conversion

- Modeling Transformations
- Illumination (Shading)
- Viewing Transformation (Perspective / Orthographic)
- Clipping
- Projection (to Screen Space)
- **Scan Conversion (Rasterization)**
- Visibility / Display

- Rasterizes objects into pixels
- Interpolate values as we go (color, depth, etc.)



The Graphics Pipeline: Visibility / Display

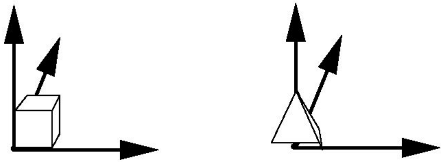


- Each pixel remembers the closest object (depth buffer)

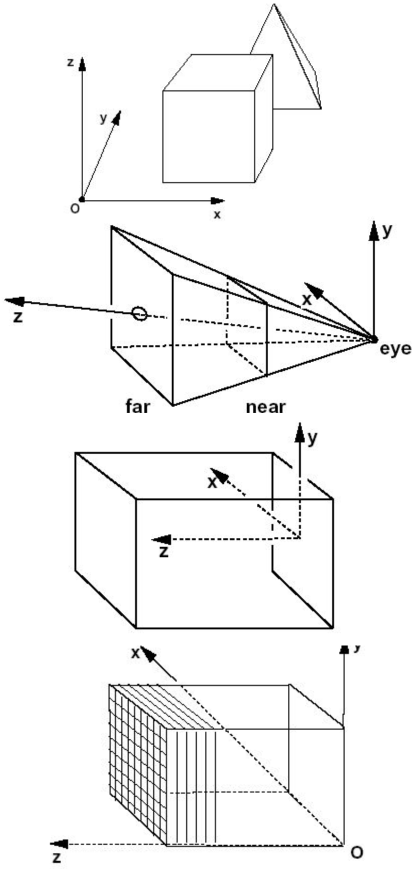
NOTE:

- Almost every step in the graphics pipeline involves a change of coordinate system.
- Transformations are central to understanding 3D computer graphics.

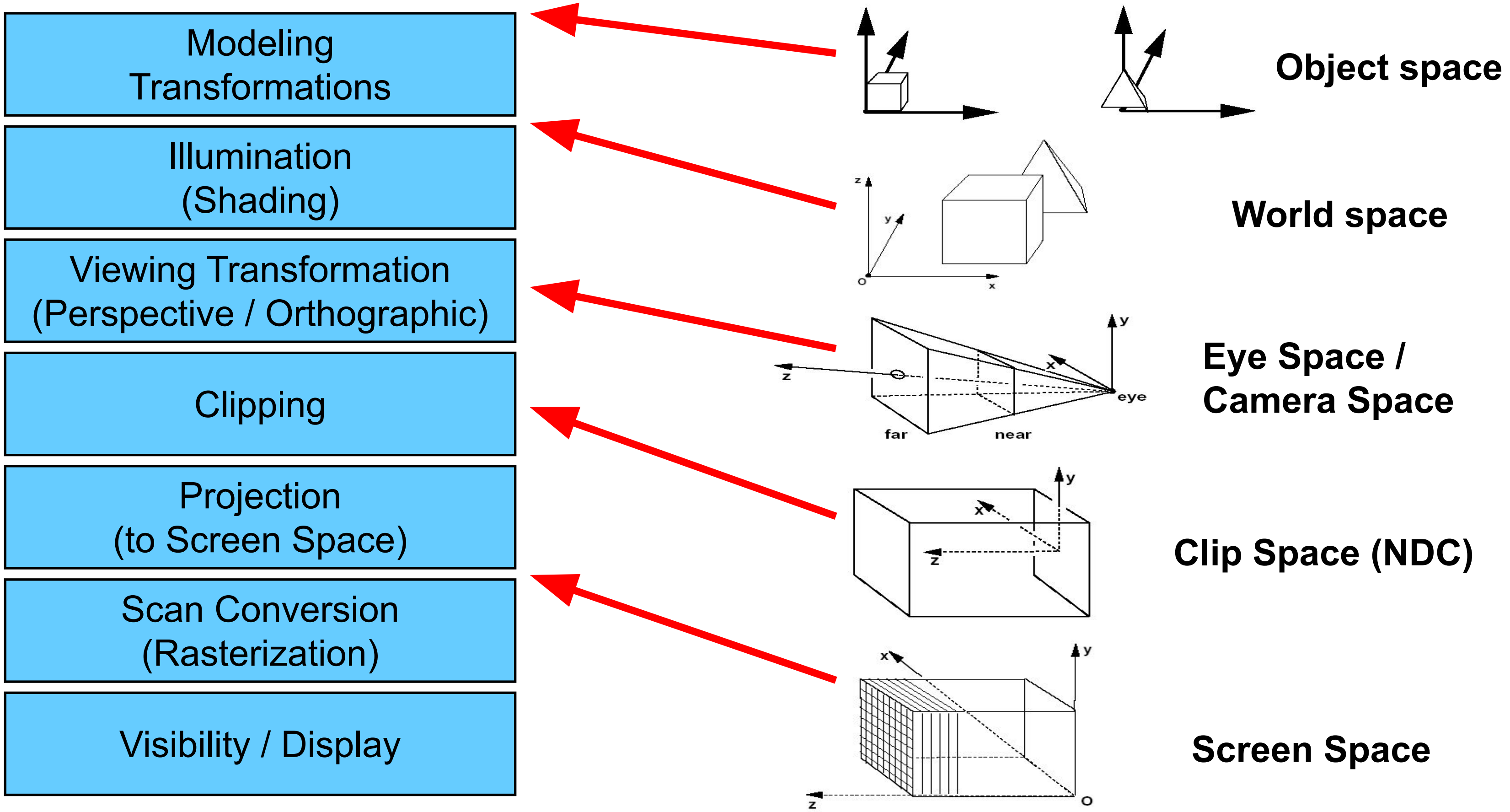
Common Coordinate Systems



- Object space
  - local to each object
- World space
  - common to all objects
- Eye space / Camera space
  - derived from view frustum
- Clip space / Normalized Device Coordinates (NDC)
  - [-1,-1,-1] → [1,1,1]
- Screen space
  - indexed according to hardware attributes



Coordinate Systems in the Graphics Pipeline



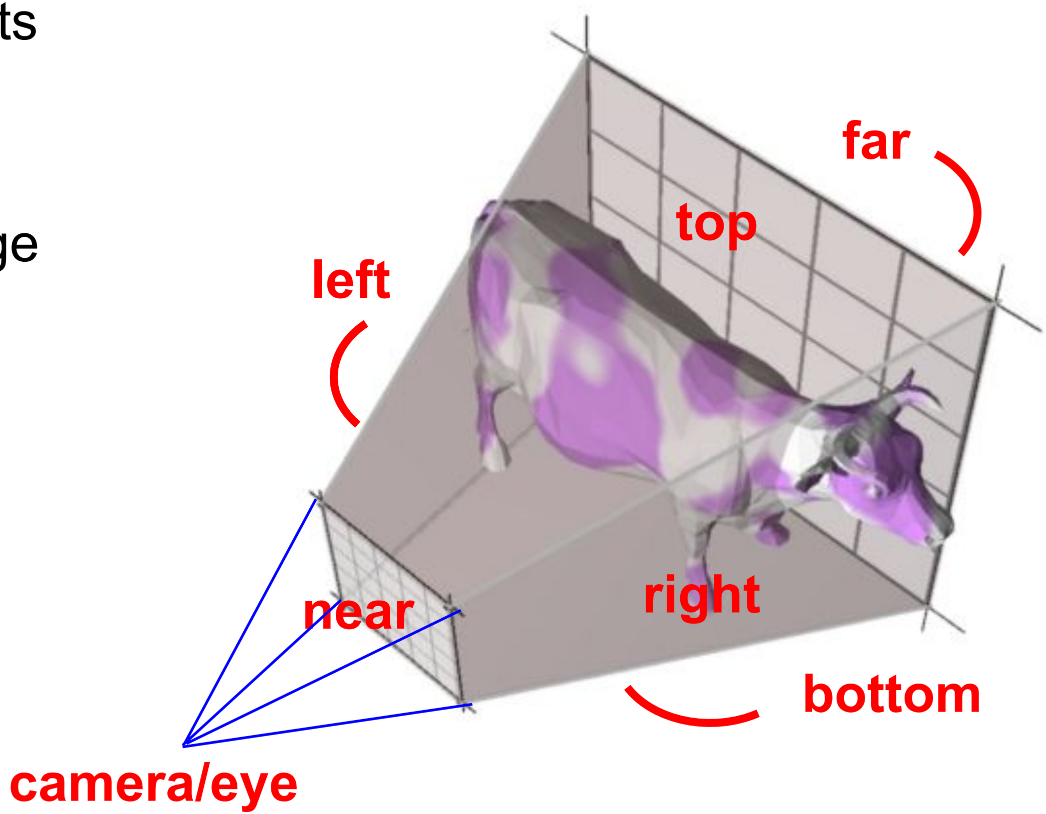
Questions?

Today

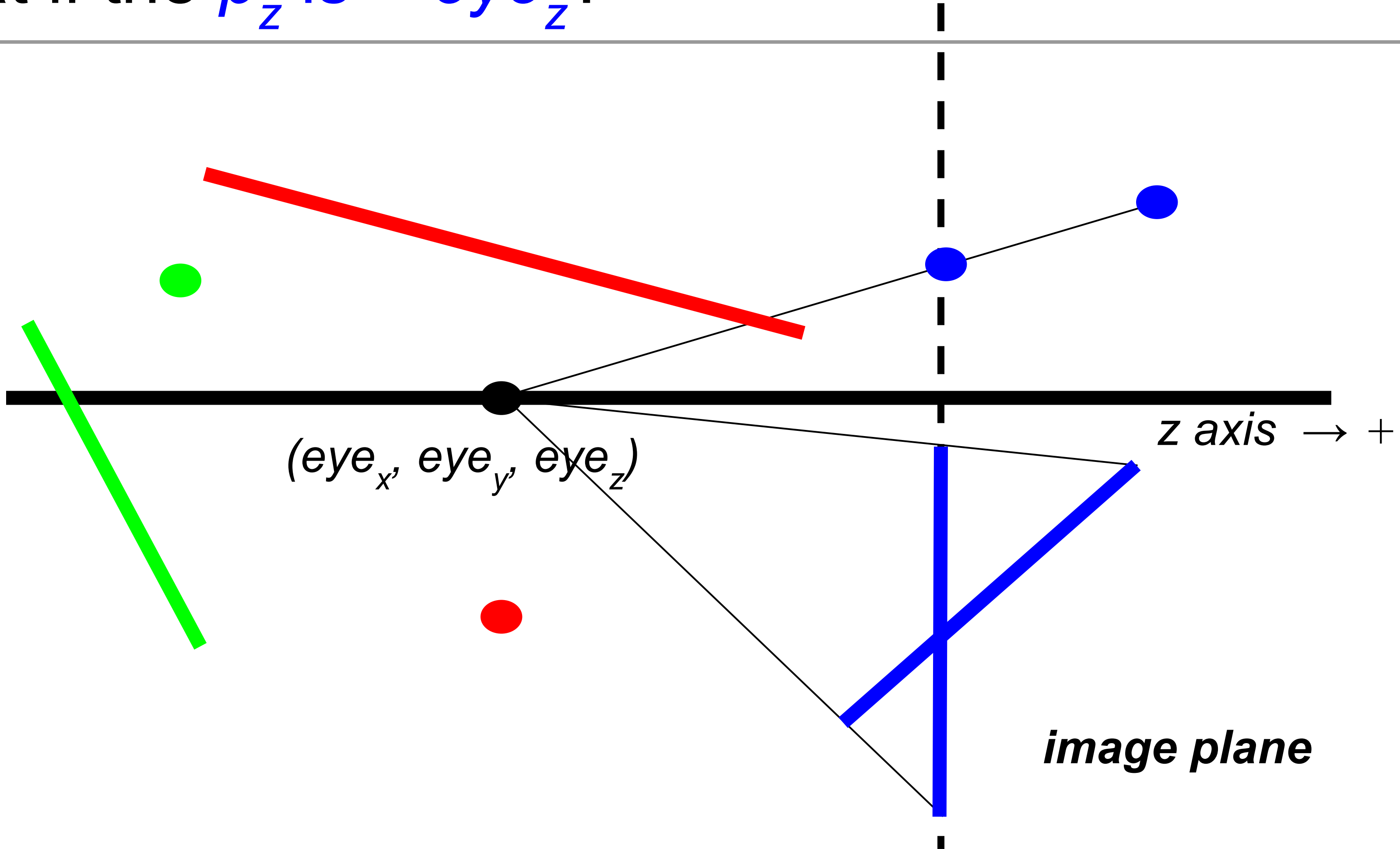
- Ray Casting / Tracing vs. Scan Conversion
- Traditional Graphics Pipeline
- Clipping
  - Coordinate Systems in the Graphics Pipeline
- Rasterization / Scan Conversion
- Readings for Today
- Readings for Next Time

Clipping

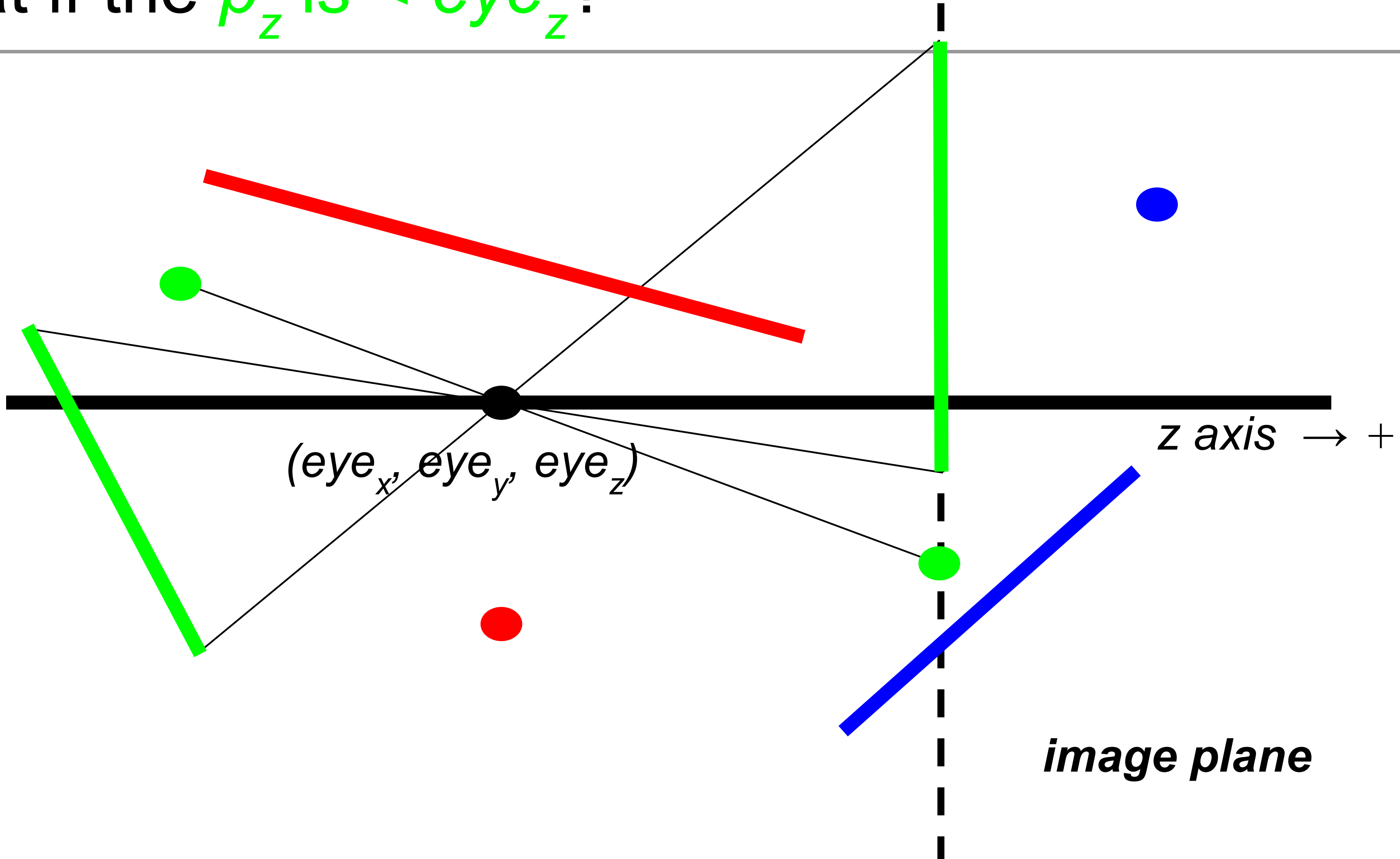
- Eliminate portions of objects outside the view frustum
- View Frustum
  - boundaries of the image plane projected in 3D
  - a near & far clipping plane
- User may define additional clipping planes



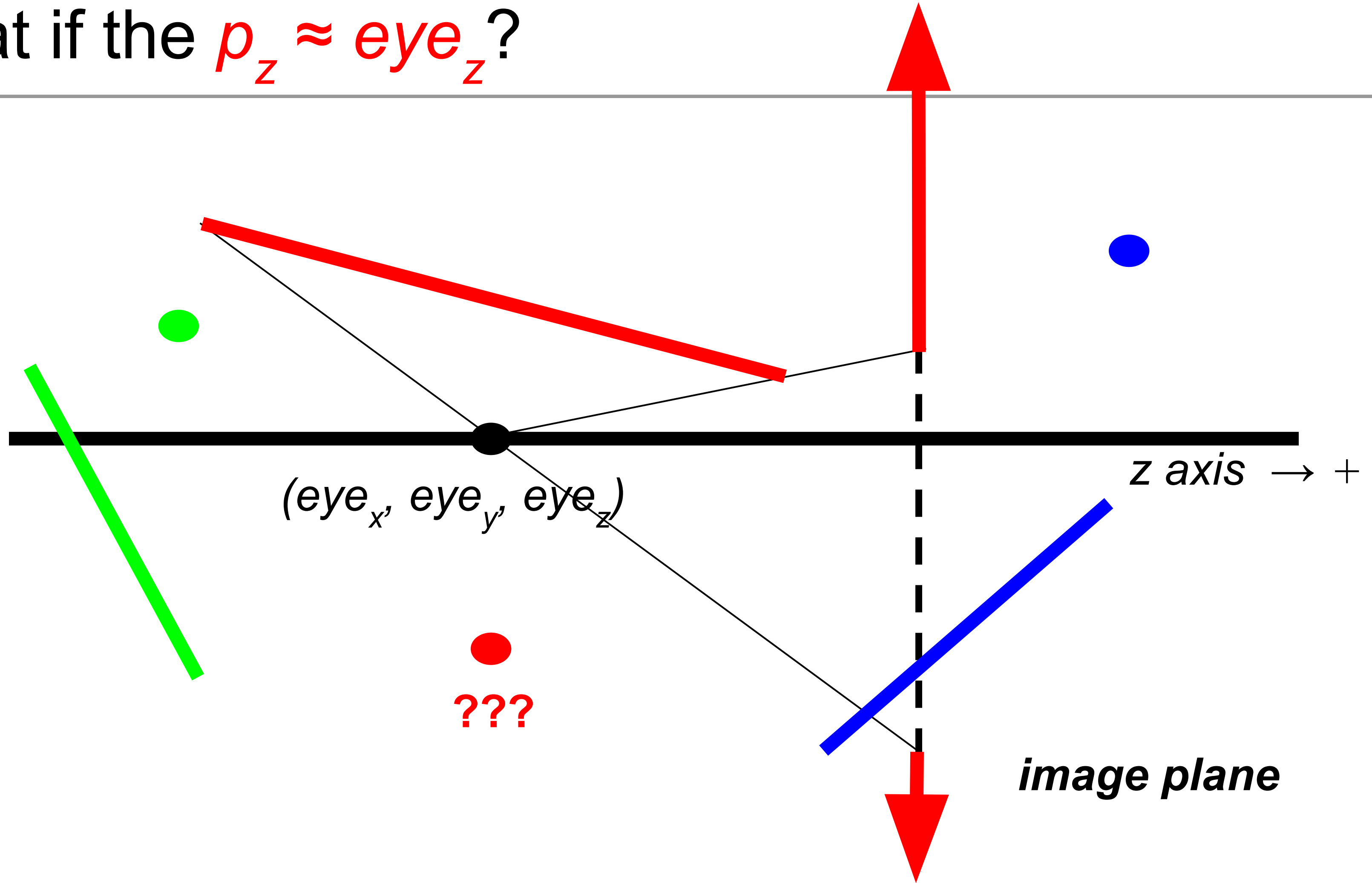
What if the p_z is > eye_z?



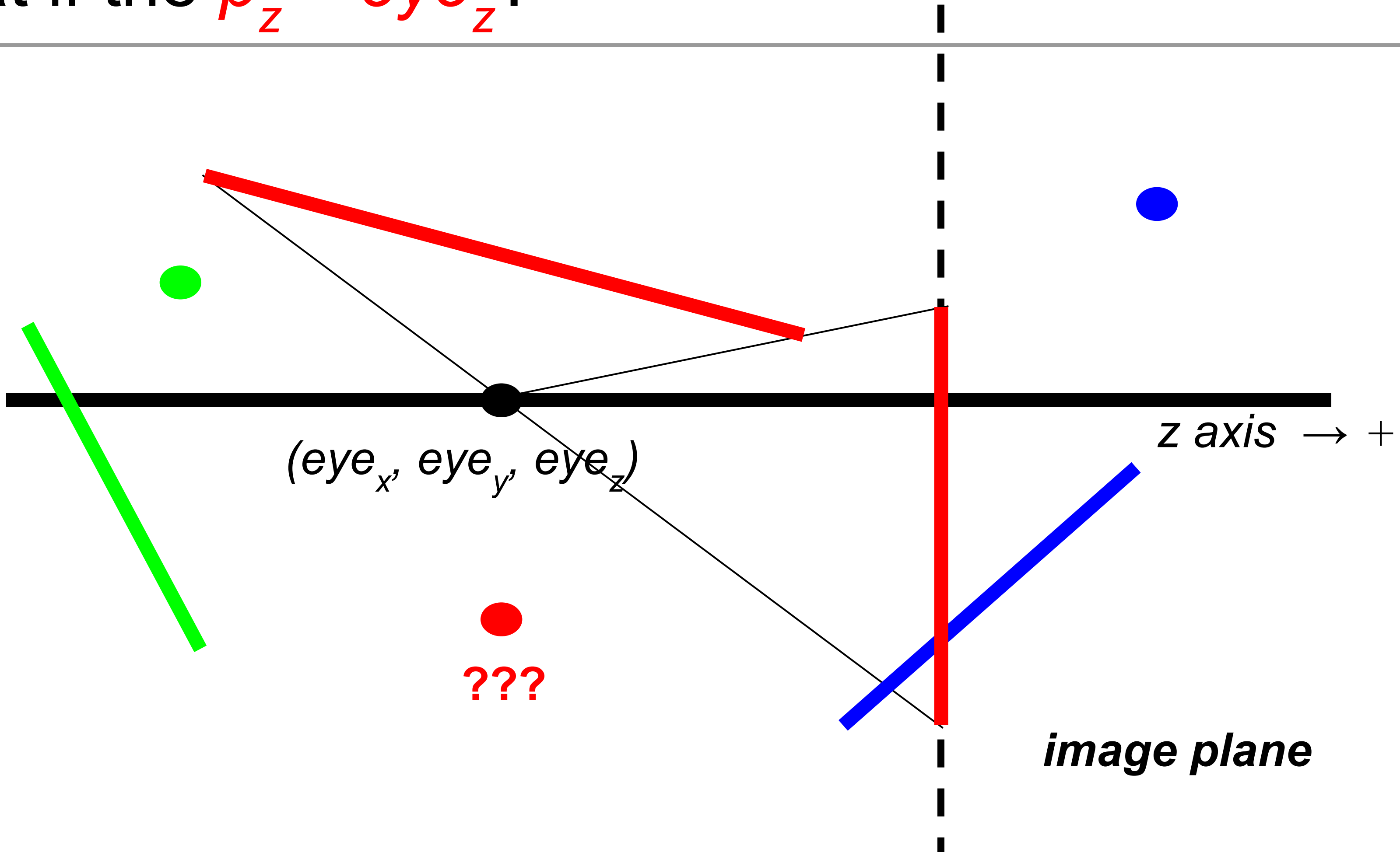
What if the p_z is < eye_z?



What if the $p_z \approx eye_z$?

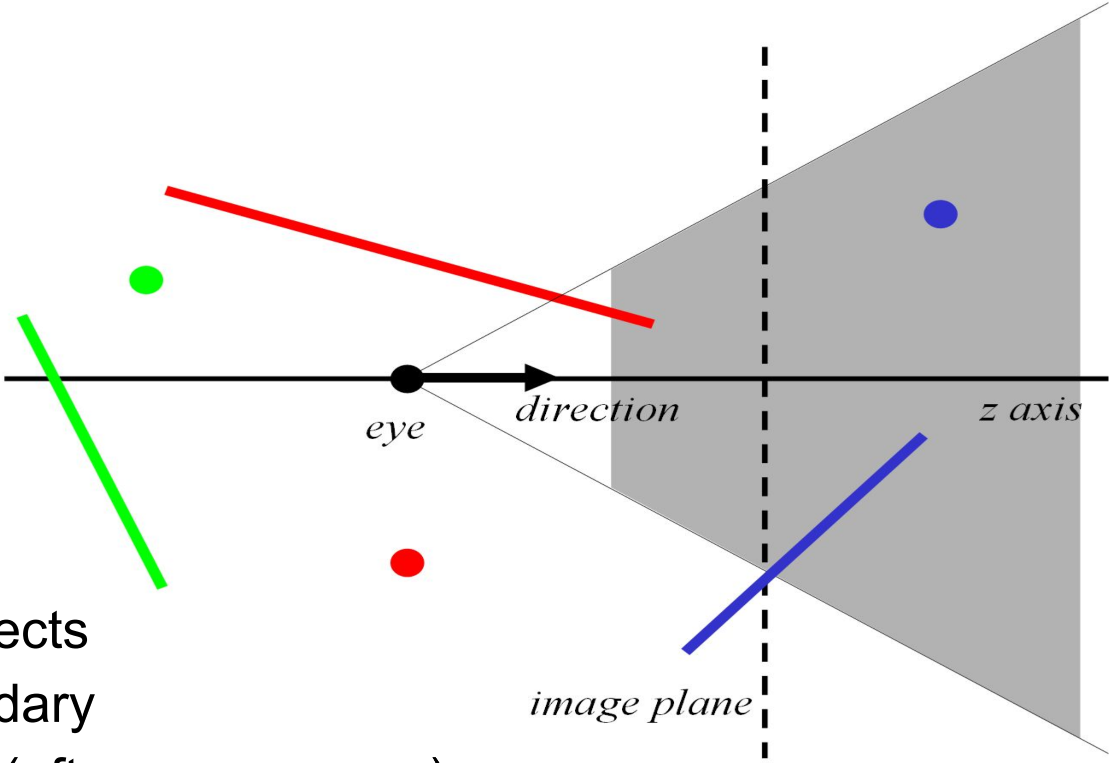


What if the $p_z \approx eye_z$?

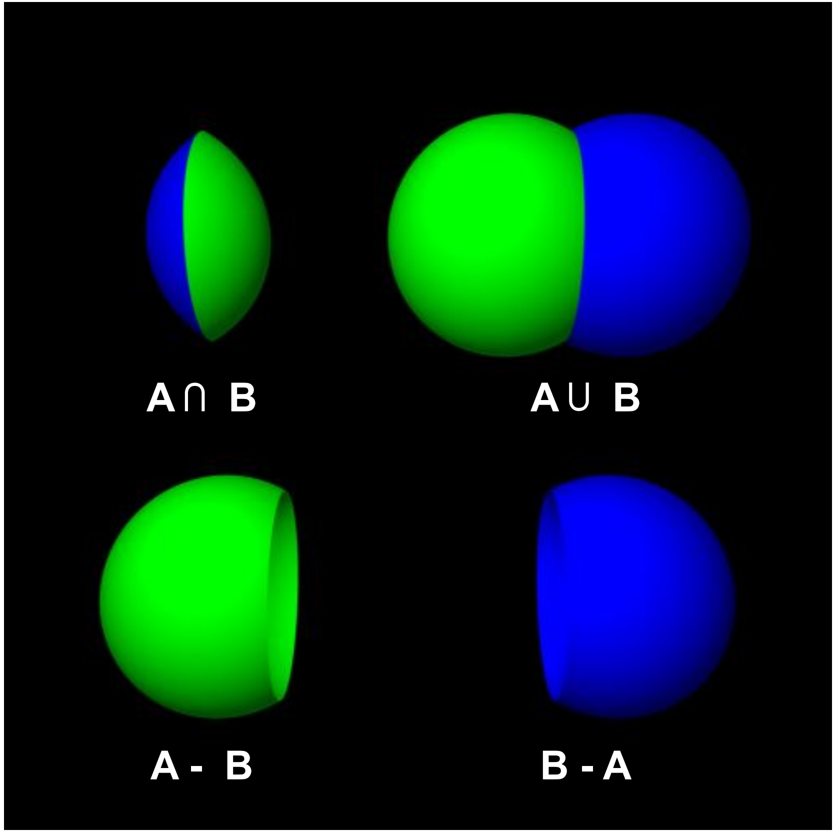


Why Clip?

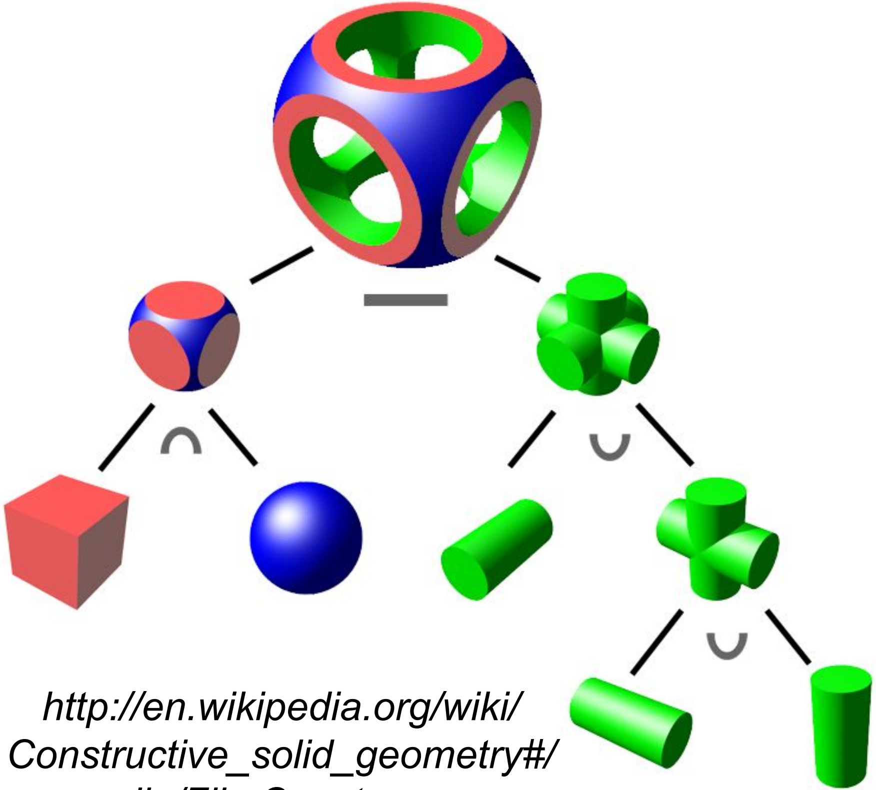
- Avoid degeneracies
  - Don’t draw stuff behind the eye
  - Avoid division by 0 and overflow
- Efficiency
  - Don’t waste time on objects outside the image boundary
- Other graphics applications (often non-convex)
  - Hidden-surface removal, Shadows, Picking, Binning, CSG (Boolean) operations (2D & 3D)



Constructive Solid Geometry *(related to Clipping)*



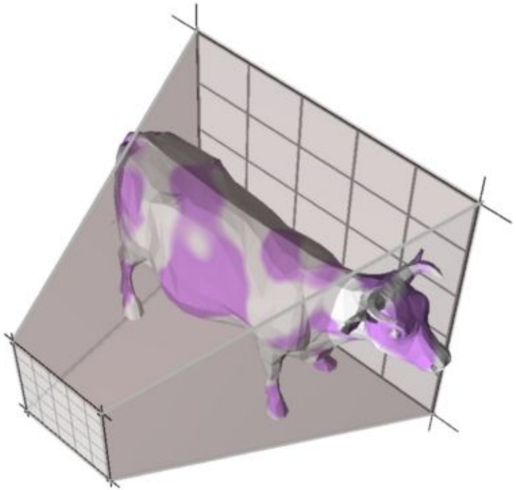
http://matter.sawkmonkey.com/raytracer/csg.html

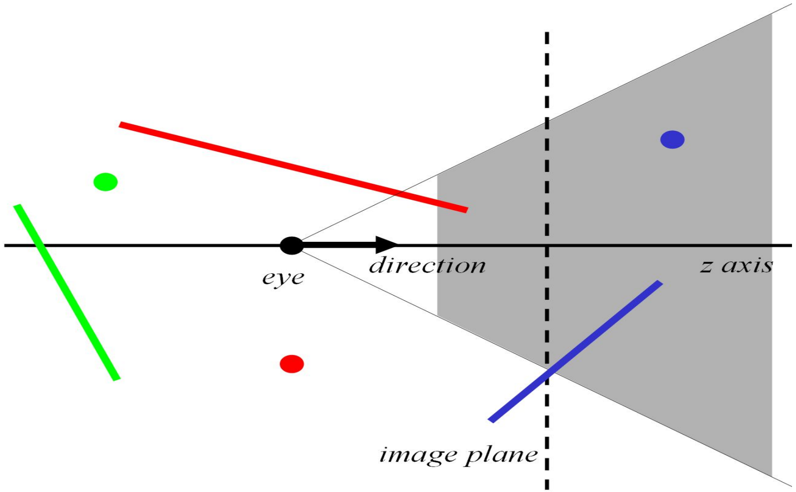


http://en.wikipedia.org/wiki/Constructive_solid_geometry#/media/File:Csg_tree.png

Clipping Strategies

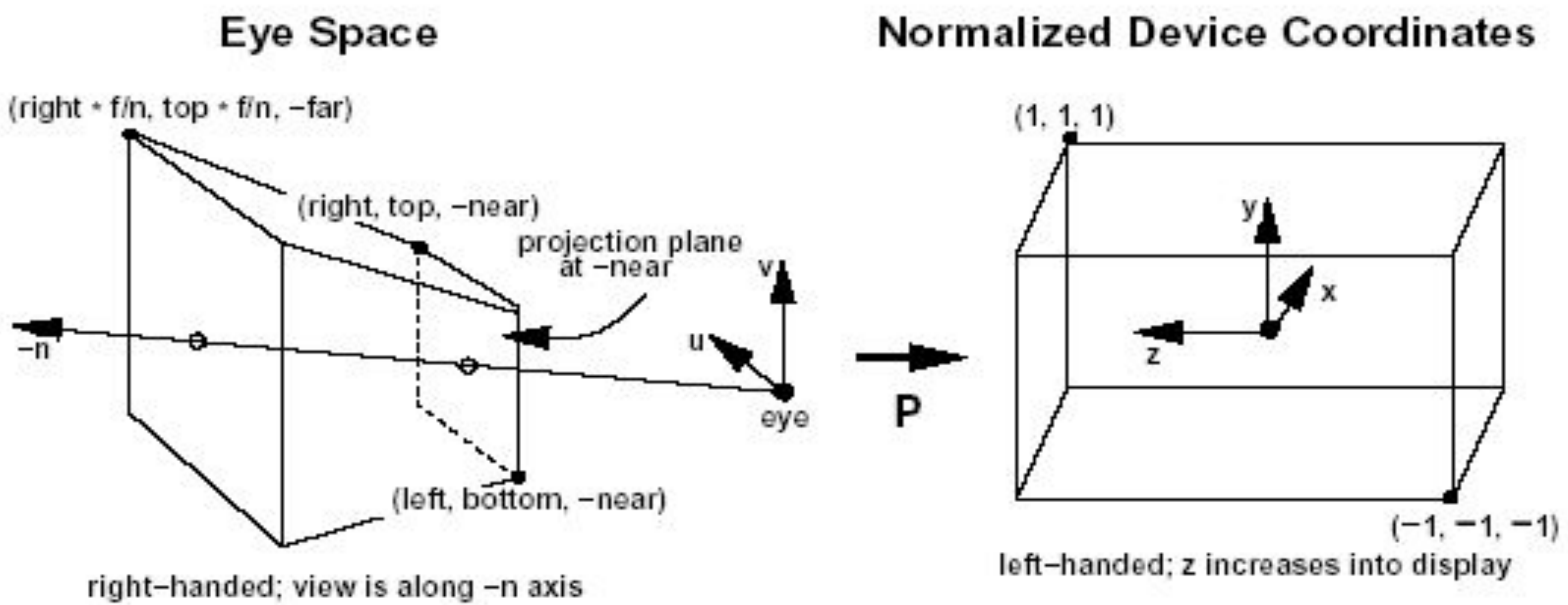
- Don’t clip (and hope for the best)
- Clip on-the-fly during rasterization
- Analytical clipping: alter input geometry



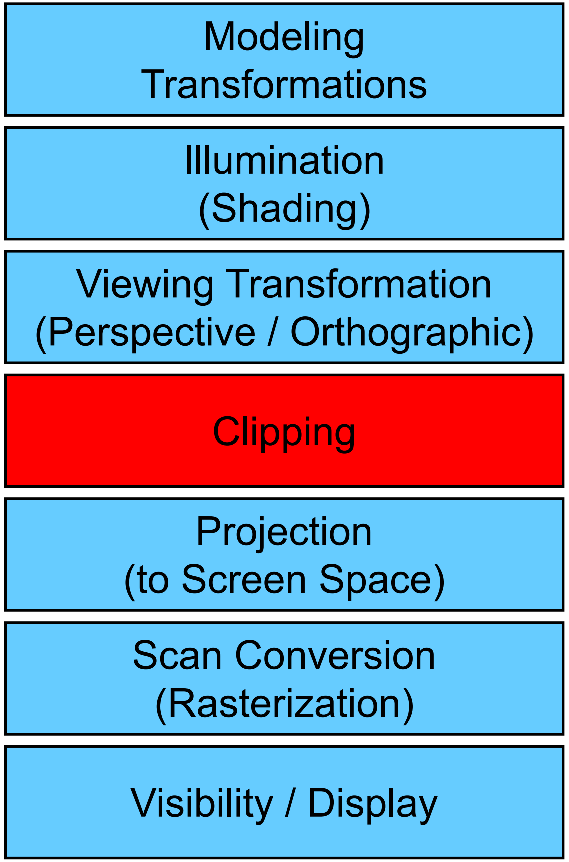


Normalized Device Coordinates

- Clipping is more efficient in a rectangular, axis-aligned volume: (-1,-1,-1) → (1,1,1) *OR* (0,0,0) → (1,1,1)

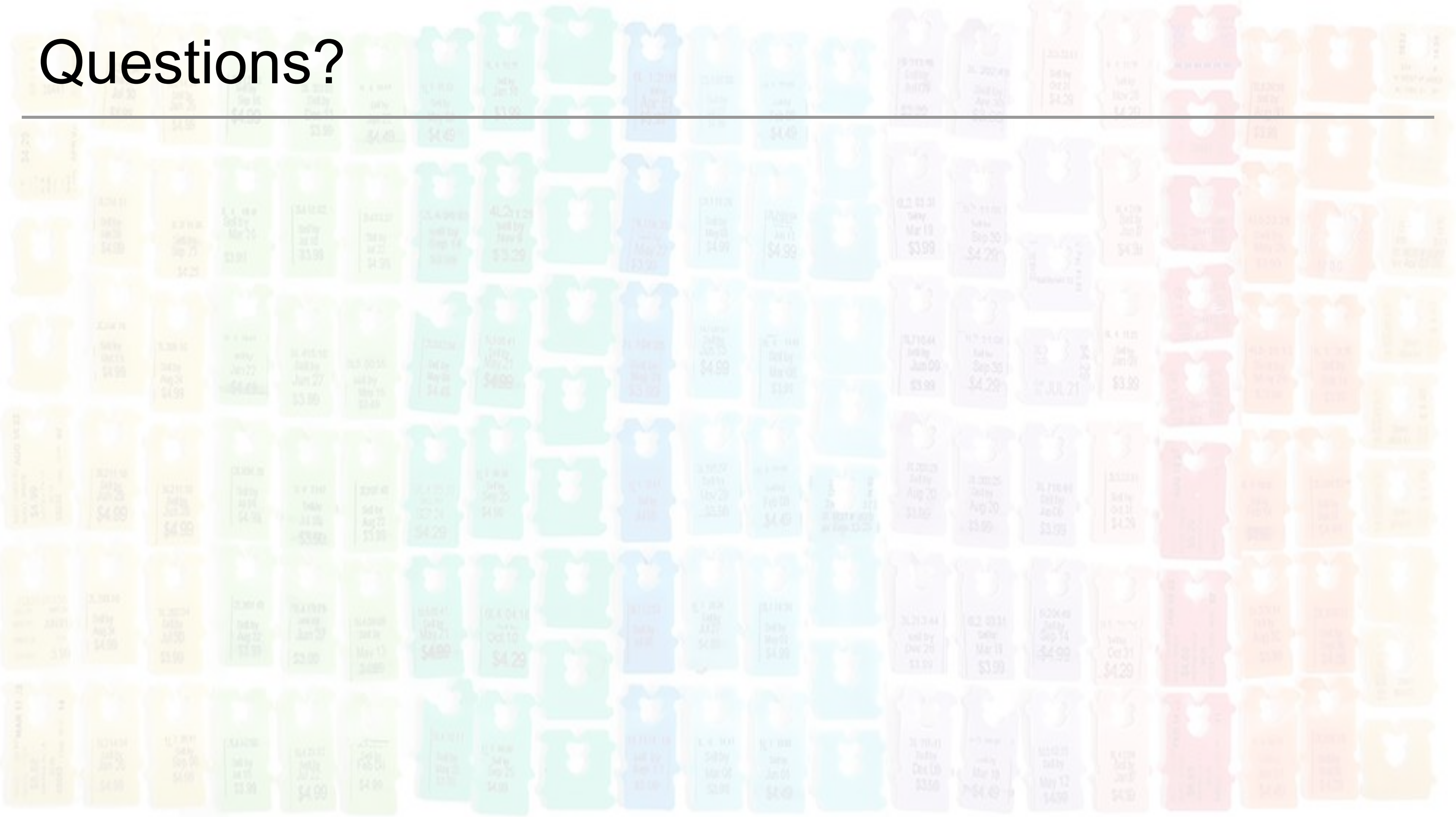


The Graphics Pipeline: Clipping



- Former hardware relied on full clipping
- Modern hardware mostly avoids clipping
  - Only with respect to plane z=0
- In general, it is still useful to learn clipping because it is similar to many other geometric algorithms

Questions?

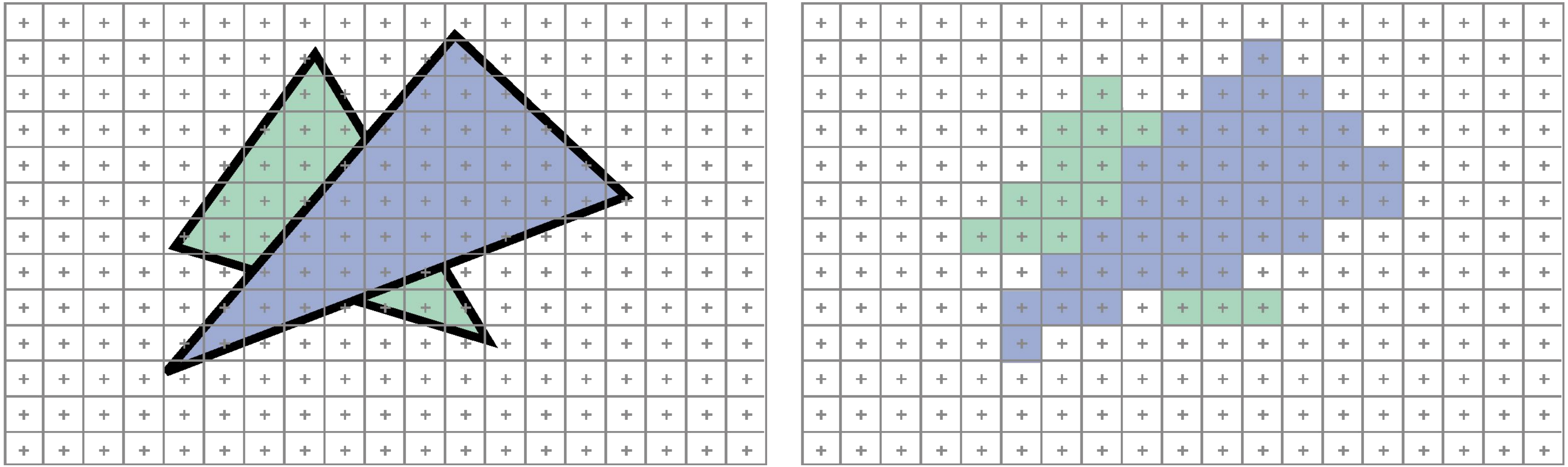


Today

- Ray Casting / Tracing vs. Scan Conversion
- Traditional Graphics Pipeline
- Clipping
- Rasterization / Scan Conversion
  - Line Rasterization
  - Triangle Rasterization
- Readings for Today
- Readings for Next Time

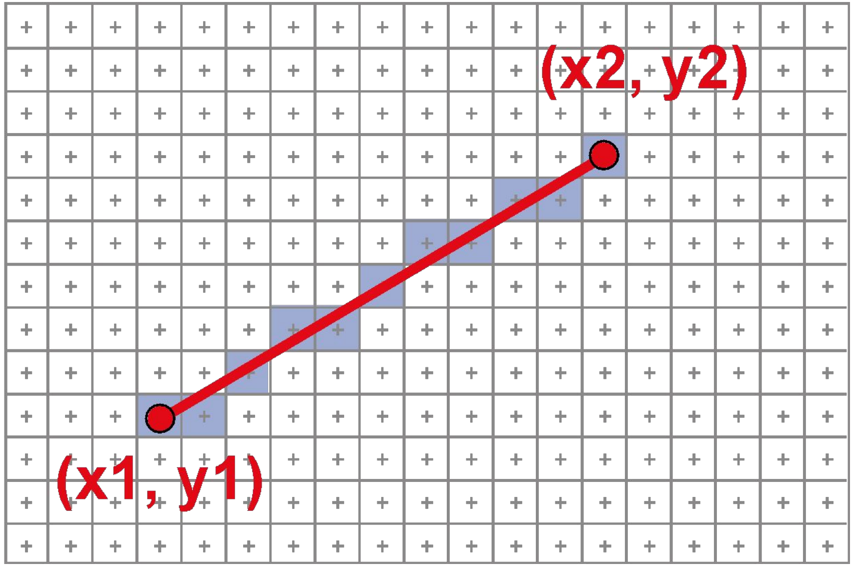
2D Scan Conversion

- Geometric primitives
  - (point, line, polygon, circle, polyhedron, sphere...)
- Primitives are continuous; screen is discrete
- Scan Conversion: algorithms for *efficient* generation of the samples comprising this approximation



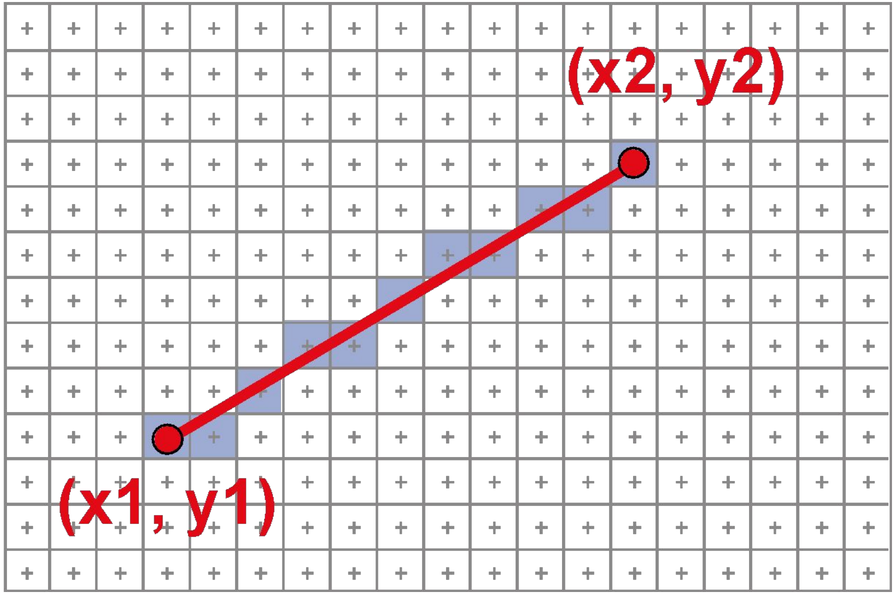
Scan Converting 2D Line Segments

- Given:
  - Segment endpoints (integers x1, y1; x2, y2)
- Identify:
  - Set of pixels (x, y) to display for segment



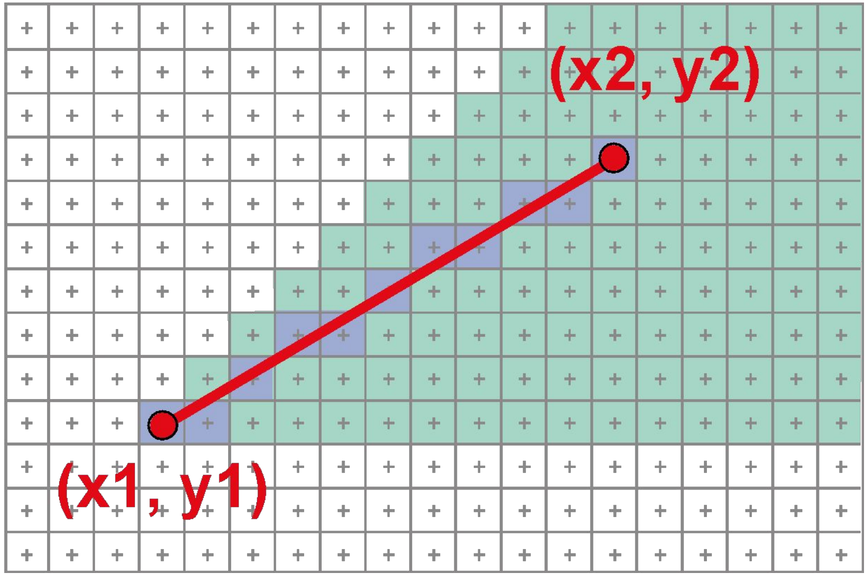
Line Rasterization Requirements

- Transform **continuous** primitive into **discrete** samples
- Uniform thickness & brightness
- Continuous appearance
- No gaps
- Accuracy
- Speed



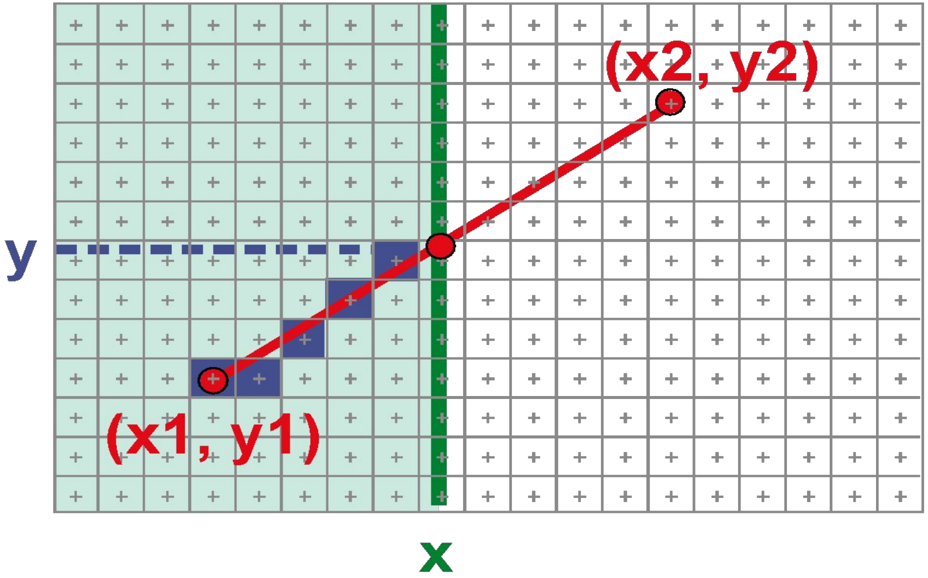
Algorithm Design Choices

- Assume:
  - $m = dy/dx$, $0 < m < 1$
- Exactly one pixel per column
  - fewer → disconnected
  - more → too thick



Naive Line Rasterization Algorithm

- Simply compute y as a function of x
  - Conceptually: move vertical scan line from x1 to x2
  - What is the expression of y as function of x?
  - Set pixel (x, round (y(x)))



$$y = y1 + \frac{x - x1}{x2 - x1}(y2 - y1)$$

$$= y1 + m(x - x1)$$

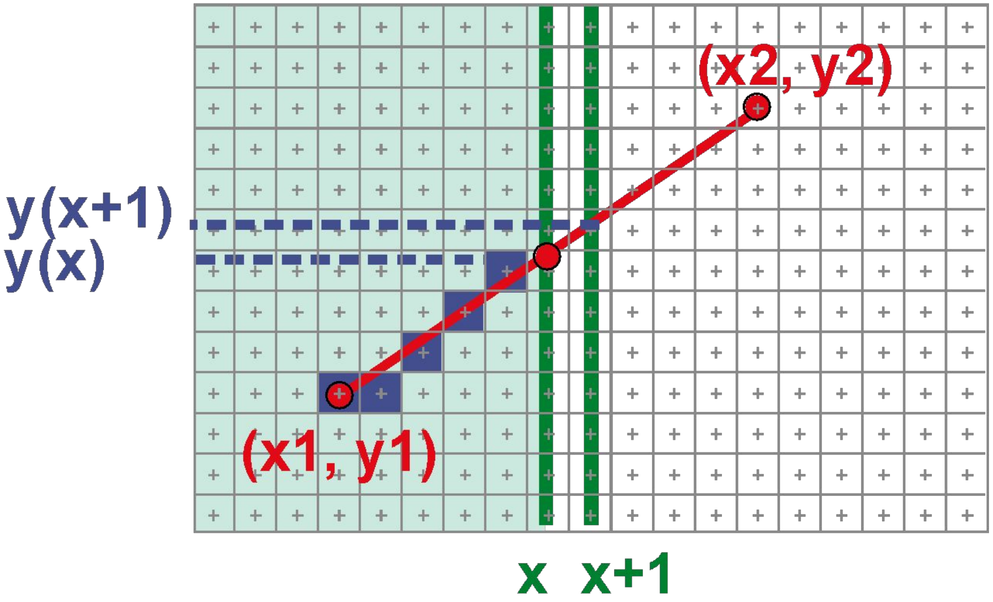
$$m = \frac{dy}{dx}$$

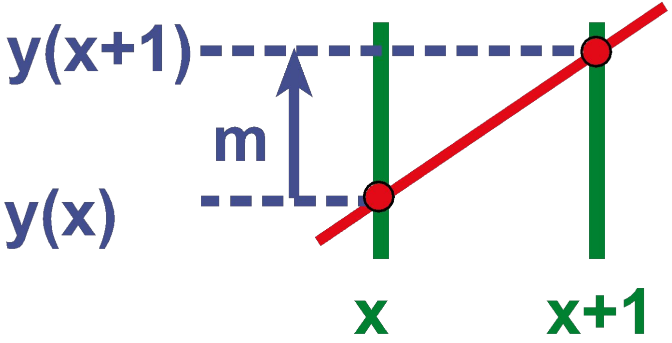
Rasterization Efficiency

- Computing *y* value is multiple mathematical operations

$$y = y_1 + m\,(x - x_1)$$

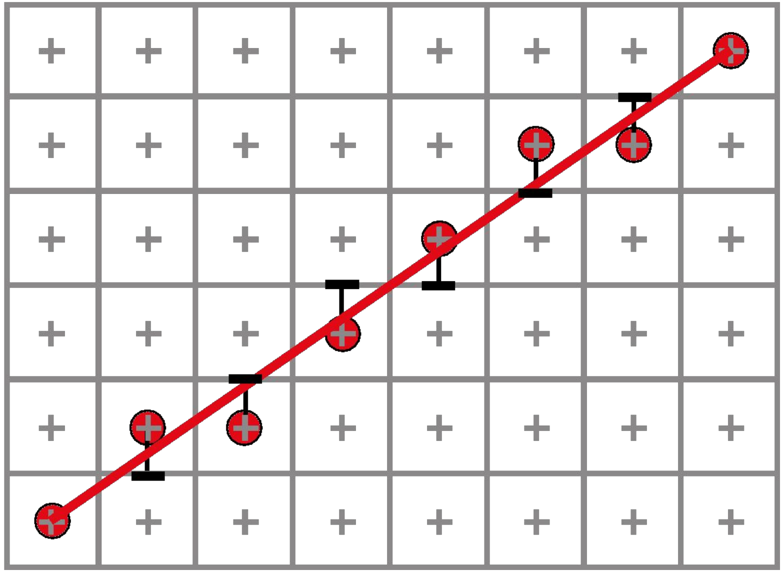
- Observe useful simplification: *y* += *m* at each *x* step (*m* = *dy/dx*)





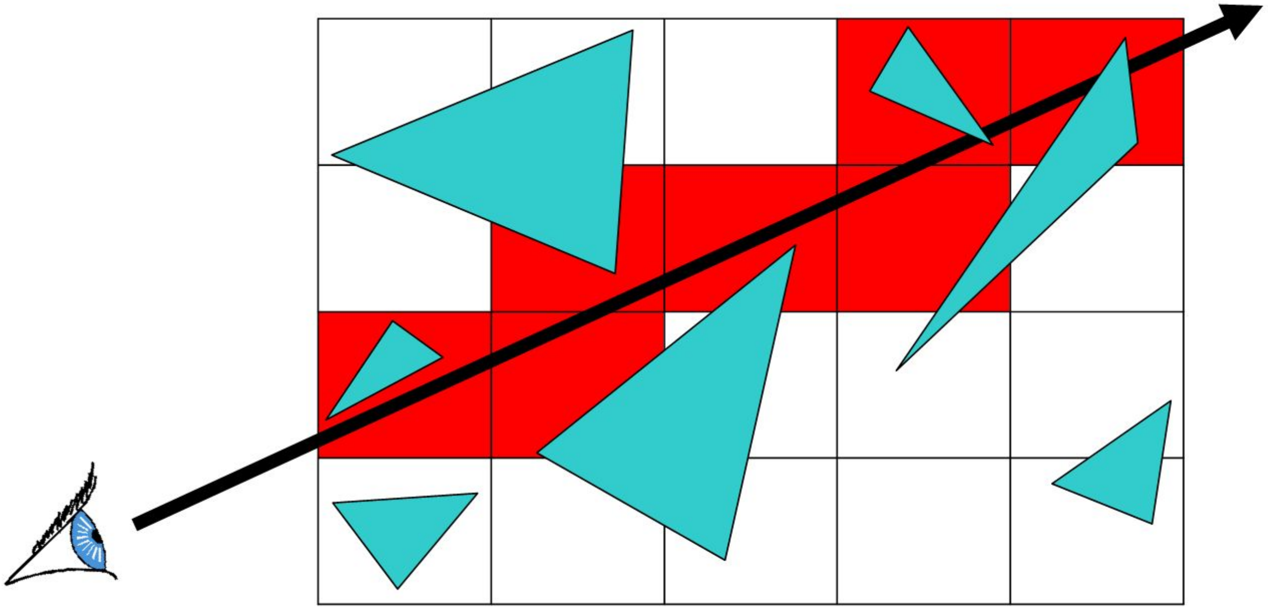
Bresenham's Algorithm (DDA)

- Select pixel vertically closest to line segment
  - intuitive, efficient, pixel center always within 0.5 vertically
- Generalize to handle all eight octants using symmetry
- Can be modified to use only integer arithmetic



Line Rasterization & Grid Marching

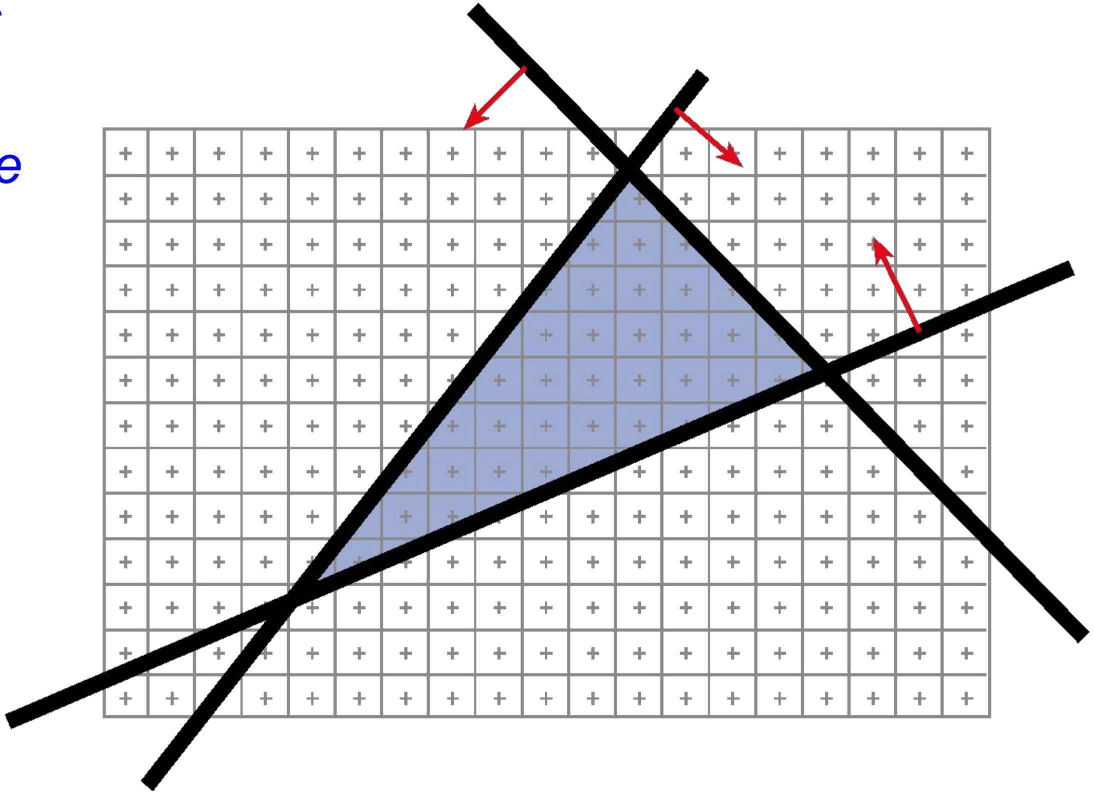
- Can be used for ray-casting acceleration
- March a ray through a grid
- Collect *all* grid cells, not just 1 per column (or row)



Questions?

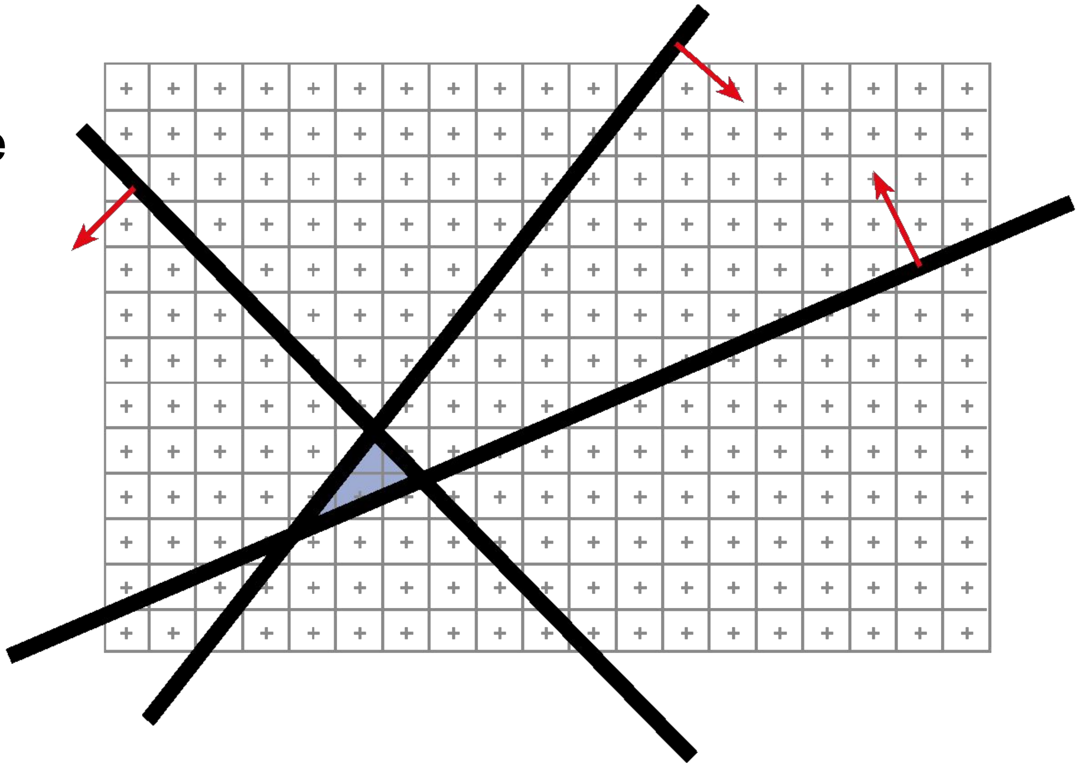
Brute Force Solution for Triangles

- For every pixel in the image
  - *Compute line equations at pixel center*
  - *“clip” against the triangle*
- Problem?



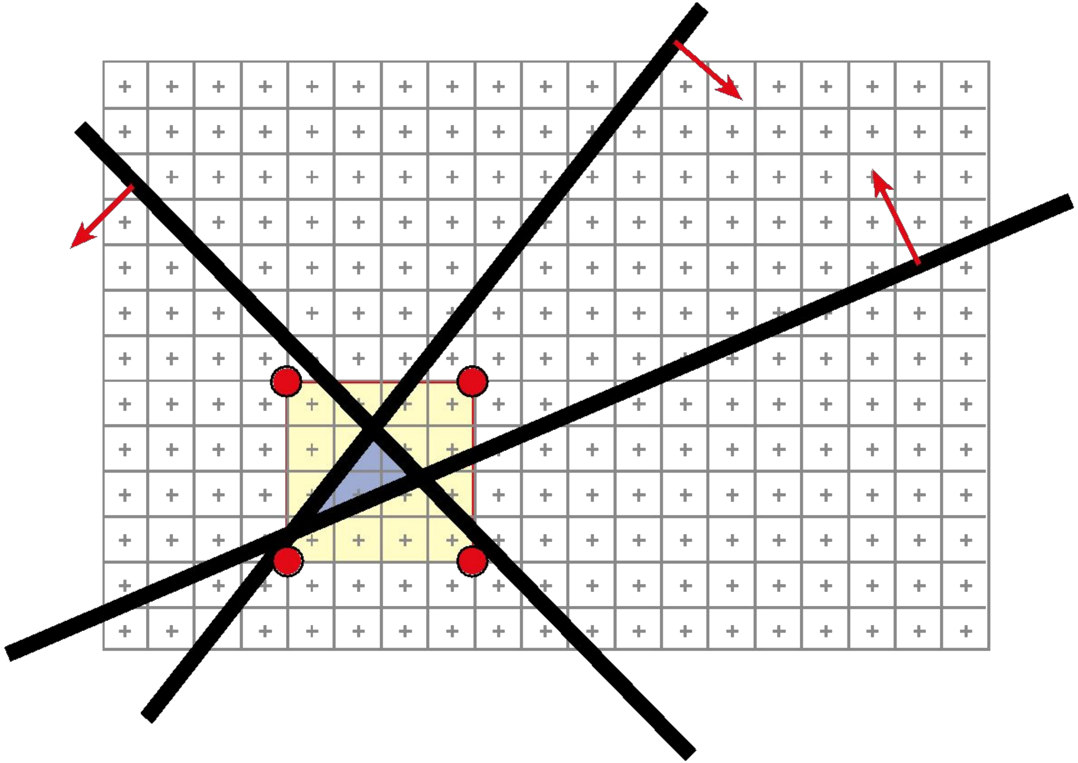
Brute Force Solution for Triangles

- For every pixel in the image
  - Compute line equations at pixel center
  - “clip” against the triangle
- Problem?
  - *If the triangle is small, we waste a lot of useless computation*



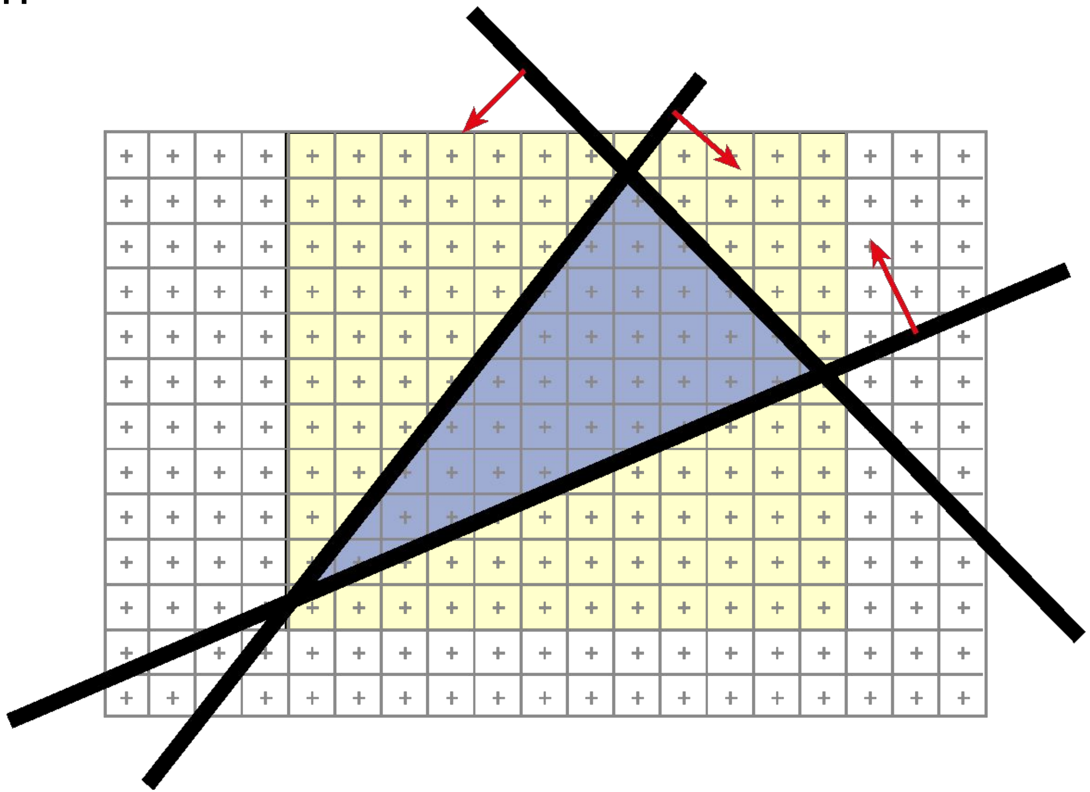
Brute Force Solution for Triangles

- Improvement:
  - *Compute only for the screen bounding box of the triangle*
- How do we get such a bounding box?
  - X_{min}, X_{max}, Y_{min}, Y_{max} of the triangle vertices



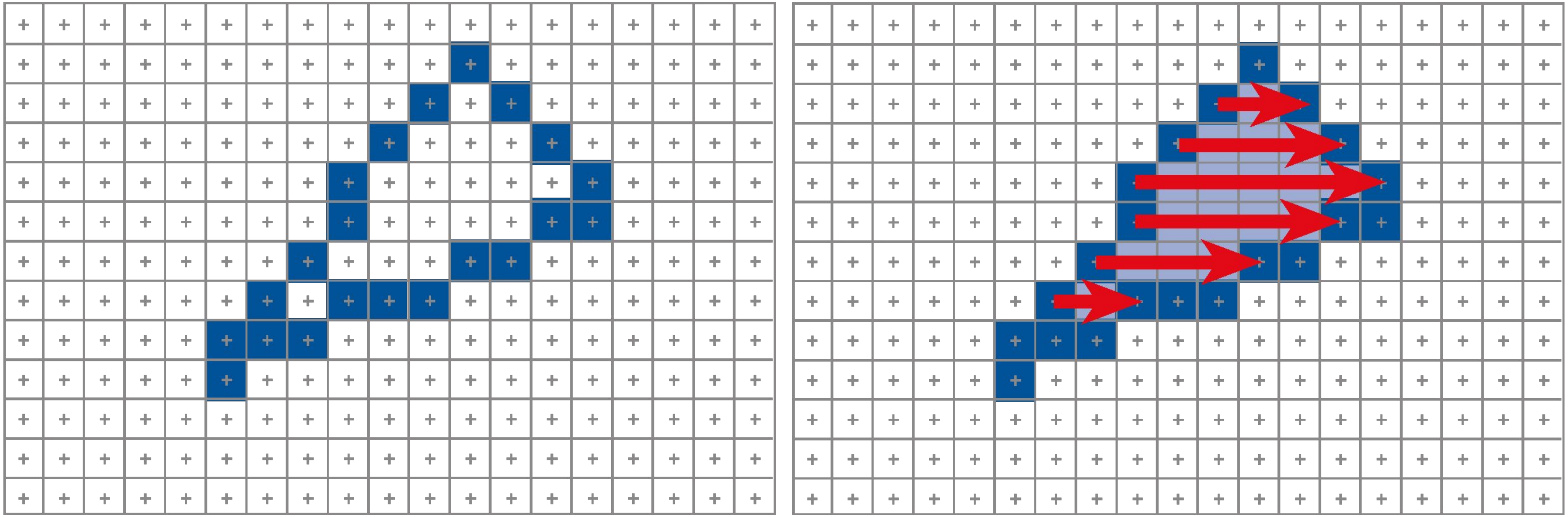
Can we do better? Kind of!

- We compute the line equation for many useless pixels
- What could we do?



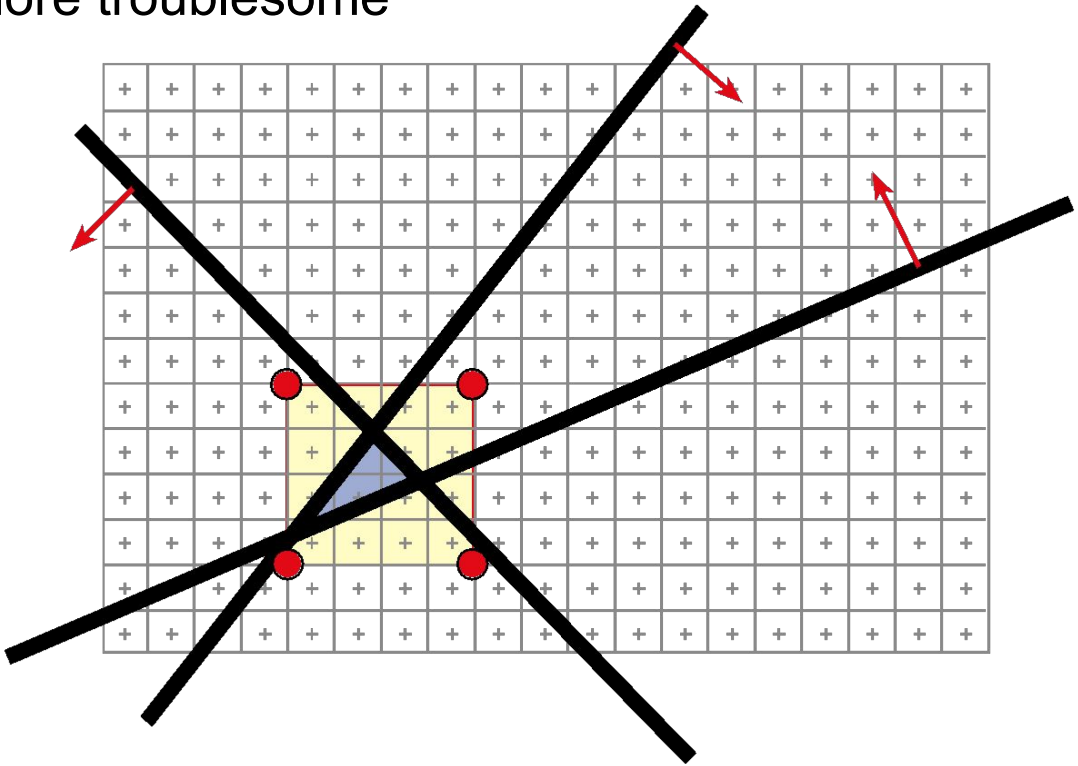
Scan-line Rasterization

- Compute the boundary pixels
- Fill the spans
- Interpolate vertex color along the edges & spans!



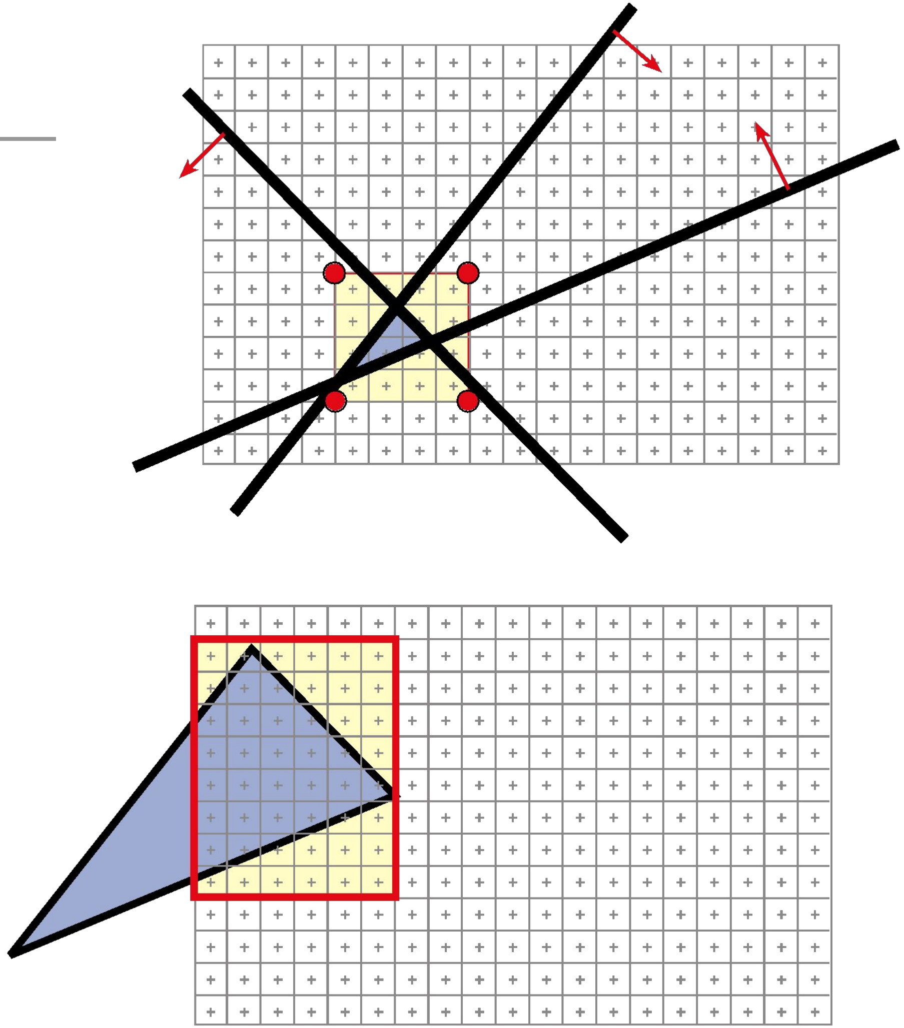
But These Days…

- Triangles are usually very small
- Setup costs are becoming more troublesome
- Clipping is annoying
- Brute force is tractable



Modern Rasterization

```
For every triangle

    Compute projection of vertices

    Compute bbox of vertices &
    clip bbox to screen limits

    For all pixels in bbox

        Compute line equations

        If all line equations > 0
        // pixel [x,y] is in triangle

            Framebuffer[x,y] =
                 triangleColor
```

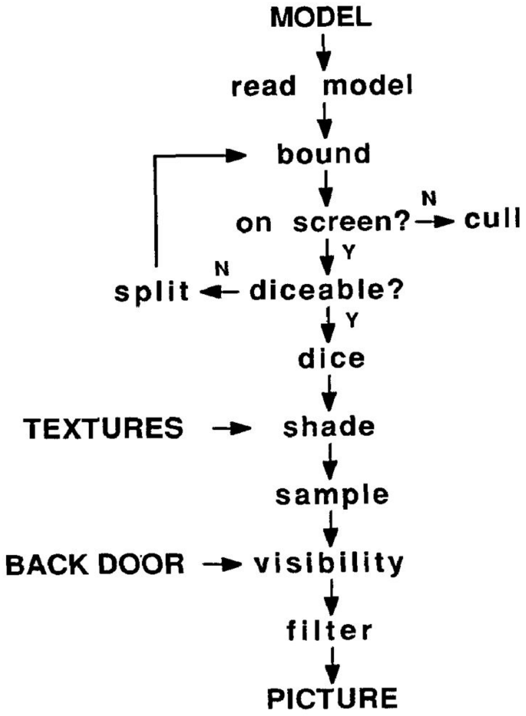


Today

- Ray Casting / Tracing vs. Scan Conversion
- Traditional Graphics Pipeline
- Clipping
- Rasterization / Scan Conversion
- Readings for Today
- Readings for Next Time

Reading for Today

- "The Reyes Image Rendering Architecture", Cook, Carpenter, and Catmull, SIGGRAPH 1987



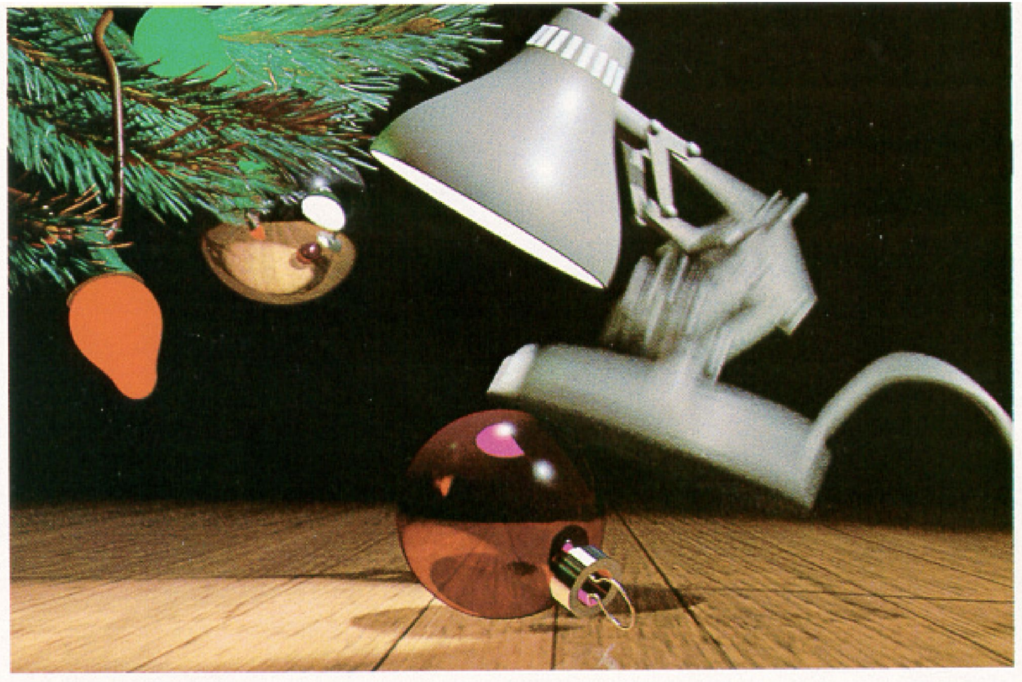
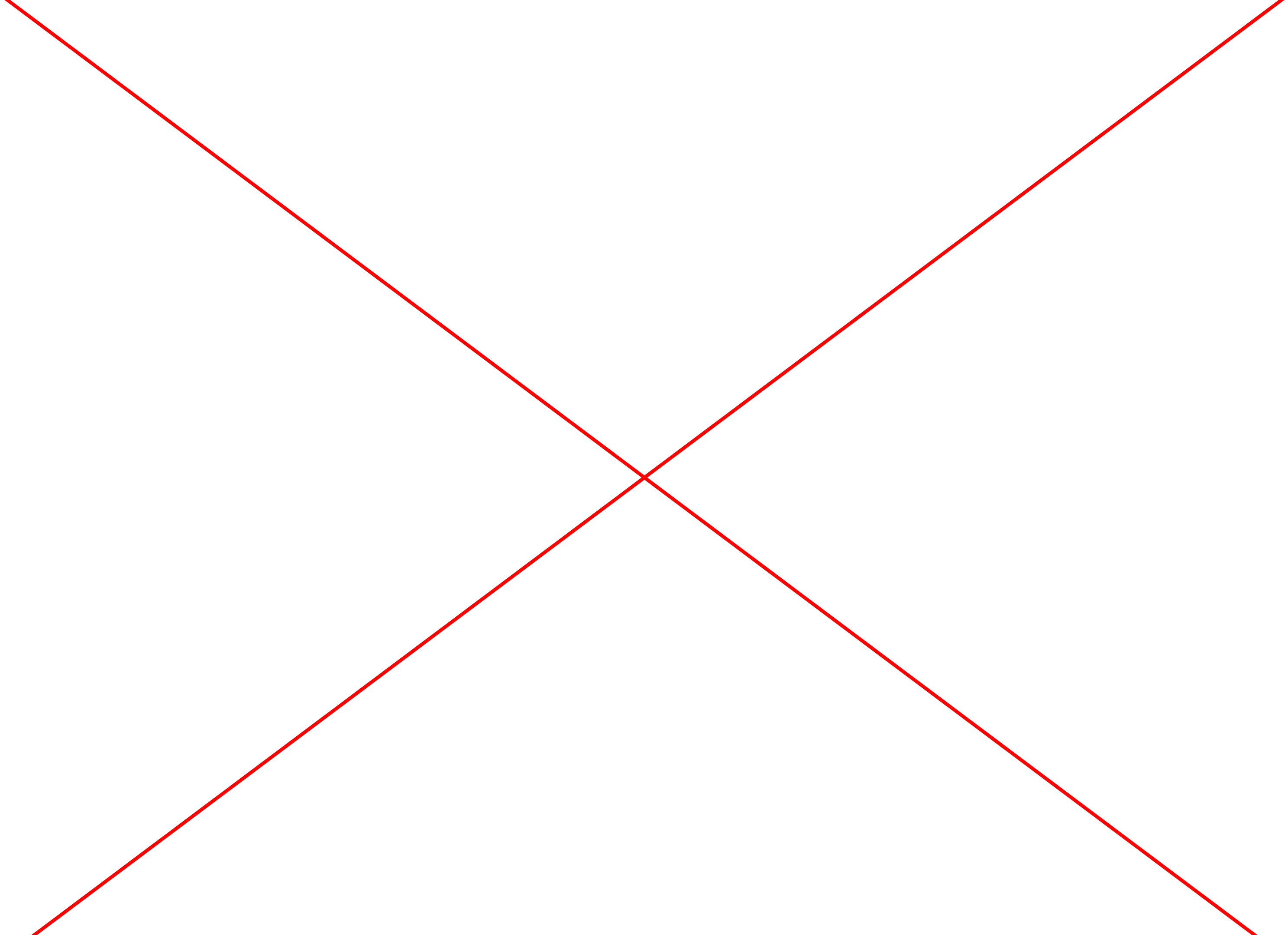


Figure 6. 1986 Pixar Christmas Card by John Lasseter and Eben Ostby.

- Production rendering v. real time rendering - shared production rendering costs
- Upfront with goals - guide development / research
- Determine the current bottlenecks, clever algorithms to minimize
- Focus on memory, minimize accesses to database
- Avoid ray tracing, thus no interactions between geometry
- Micropolygons (subdivision surfaces / displacement mapping) in production rendering
  - Ideal balance? triangles approximately size of a final pixel
- Context of overall timeline of graphics research
- Shading before visibility?
  - Wasting resources on objects that won't be visible
- Texture maps to approximate reflection

Young Sherlock Holmes 1985 (Lucasfilm / ILM)



Reading for Today

- "RenderMan: An Advanced Path Tracing Architecture for Movie Rendering", Christensen et al., TOG 2018

Fig. 8. Complex illumination in *Coco*: 8 million lights (© 2017 Disney•Pixar).

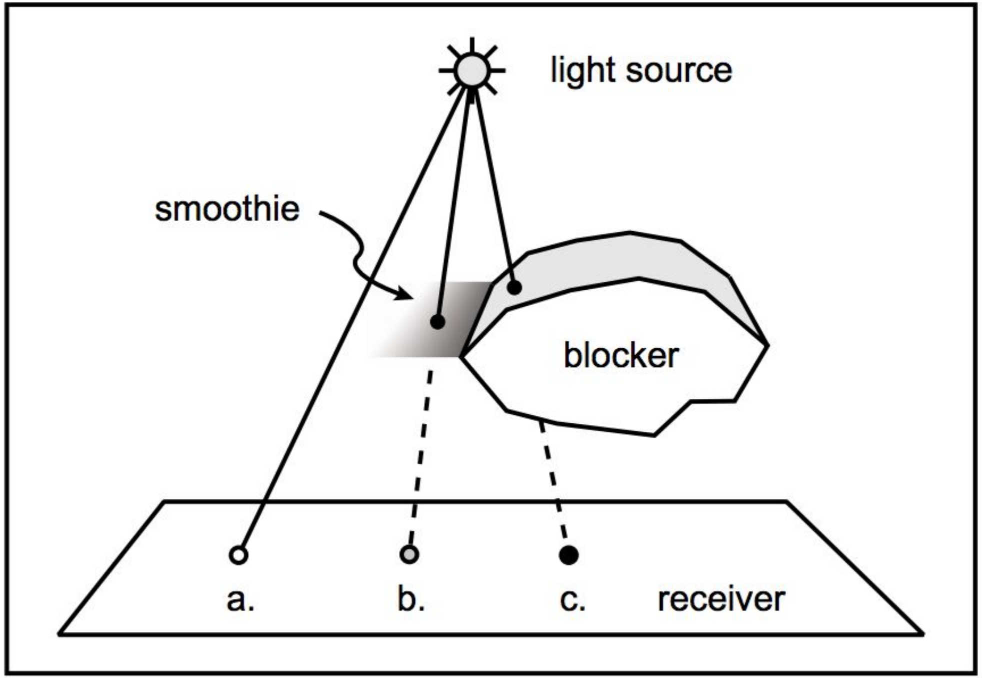
- Surprised it is not GPU based
    - doesn’t have enough memory - 100GB per frame!
- Some clever algorithms of past are now unnecessary thanks to significant hardware improvements
- Different than other papers this term,
  fewer diagrams & equations, more words/discussion

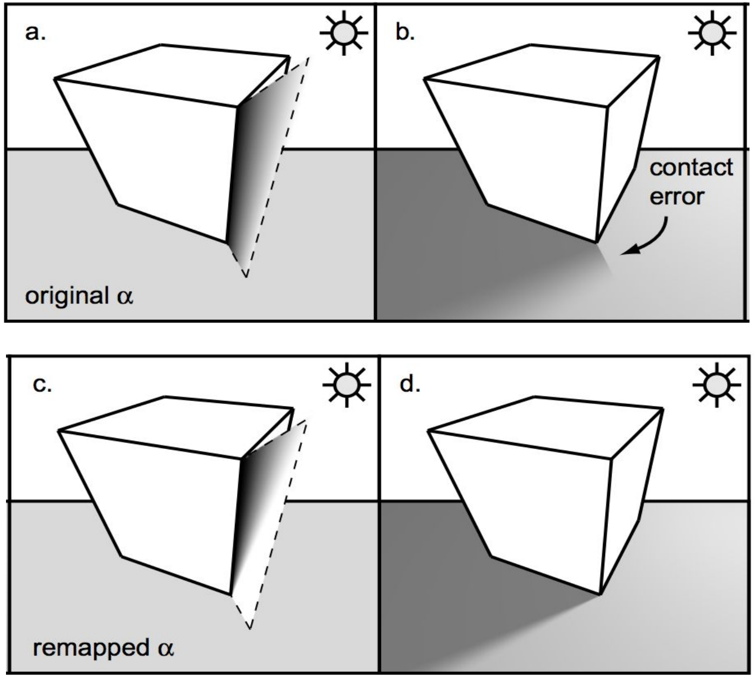
Today

- Ray Casting / Tracing vs. Scan Conversion
- Traditional Graphics Pipeline
- Clipping
- Rasterization / Scan Conversion
- Readings for Today
- Readings for Next Time

Reading for Next Time: *(pick one)*

- "Rendering Fake Soft Shadows with Smoothies", Chan & Durand, EGSR 2003





Reading for Next Time: *(pick one)*

- "Ray Tracing on Programmable Graphics Hardware", Purcell, Buck, Mark, & Hanrahan SIGGRAPH 2002