CSE 167:
Introduction to Computer Graphics
Lecture #5: Rasterization 2

Jürgen P. Schulze, Ph.D.
University of California, San Diego
Fall Quarter 2011

Announcements

- Homework project #2 due this Friday, October 7
  - To be presented starting 1:30pm in lab 260
- Late submissions for project #1 accepted until Friday, October 7

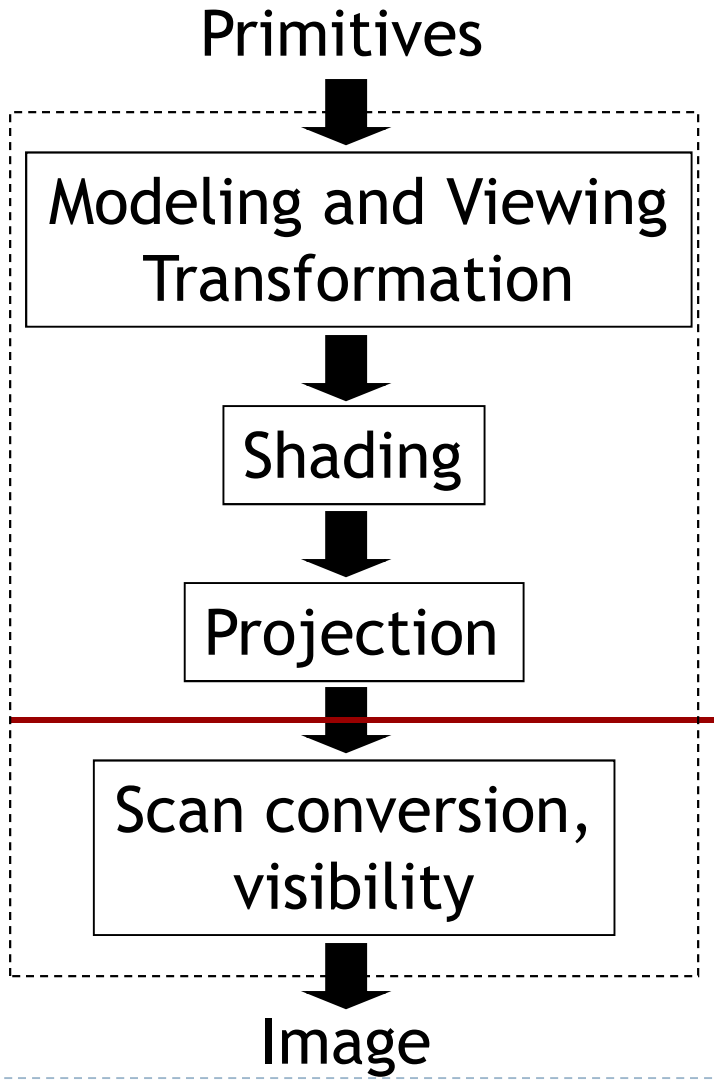
Barycentric Coordinates

- Demo Applets:
  - http://www.ccs.neu.edu/home/suhail/BaryTriangles/applet.htm
  - http://www.cut-the-knot.org/Curriculum/Geometry/Barycentric.shtml

Lecture Overview

- Culling, Clipping
- Rasterization
- Visibility
- Perspectively correct interpolation

Rendering Pipeline



Culling, Clipping

- Discard geometry that should not be drawn

Culling

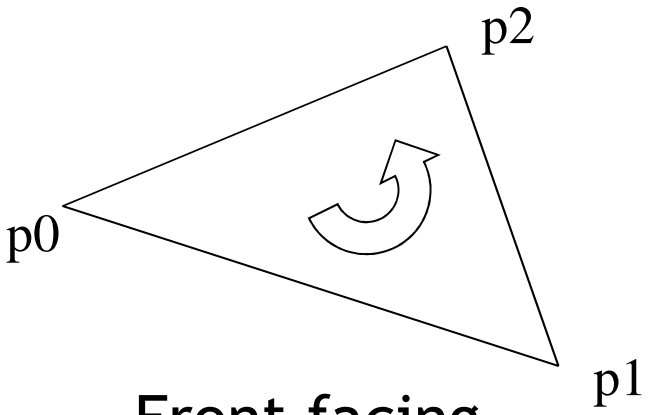
- Discard geometry that does not need to be drawn as early as possible
- Two types of culling:
  - Object-level frustum culling
    - Later in class
  - Triangle culling
    - View frustum culling (clipping): outside view frustum
    - Backface culling: facing "away" from the viewer
    - Degenerate culling: area=0

Backface Culling

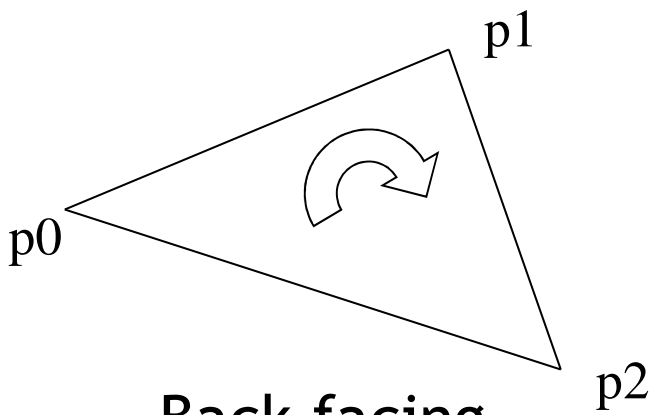
- Consider triangles as "one-sided", i.e., only visible from the "front"
- Closed objects
  - If the "back" of the triangle is facing the camera, it is not visible
  - Gain efficiency by not drawing it (culling)
  - Roughly 50% of triangles in a scene are back facing

Backface Culling

- Convention: front side means vertices are ordered counterclockwise



Front facing



Back facing

- OpenGL allows one- or two-sided triangles
  - One-sided triangles: glEnable(GL_CULL_FACE); glCullFace(GL_BACK)
  - Two-sided triangles (no backface culling): glDisable(GL_CULL_FACE)

Backface Culling

- Compute triangle normal after projection (homogeneous division)

$$\mathbf{n} = (\mathbf{p}_1 - \mathbf{p}_0) \times (\mathbf{p}_2 - \mathbf{p}_0)$$

- Third component of $\mathbf{n}$ negative: front-facing, otherwise back-facing
  - Remember: projection matrix is such that homogeneous division flips sign of third component

Degenerate Culling

- Degenerate triangle has no area
  - Vertices lie in a straight line
  - Vertices at the exact same place
  - Normal $\mathbf{n}=0$

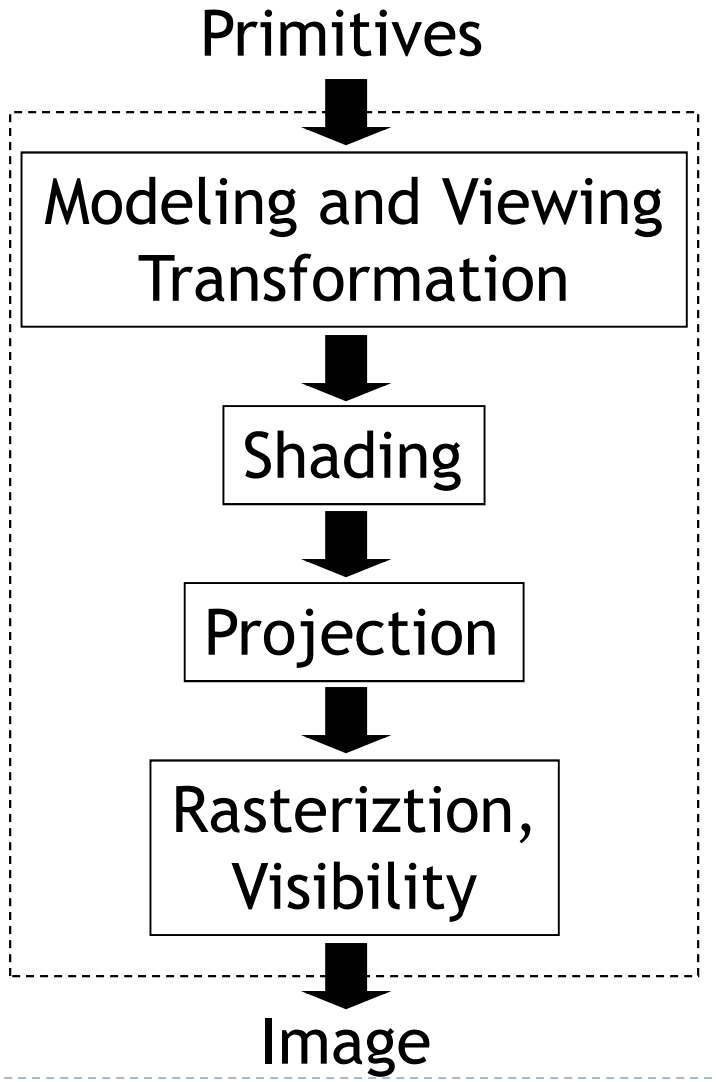
View Frustum Culling, Clipping

- Triangles that intersect the faces of the view volume
  - Partly on screen, partly off screen
  - Do not rasterize the parts that are off-screen
- Traditional clipping
  - Split triangles that lie partly inside/outside viewing volume before homogeneous division
  - Avoid problems with division by zero
- Modern GPU implementations avoid clipping

Lecture Overview

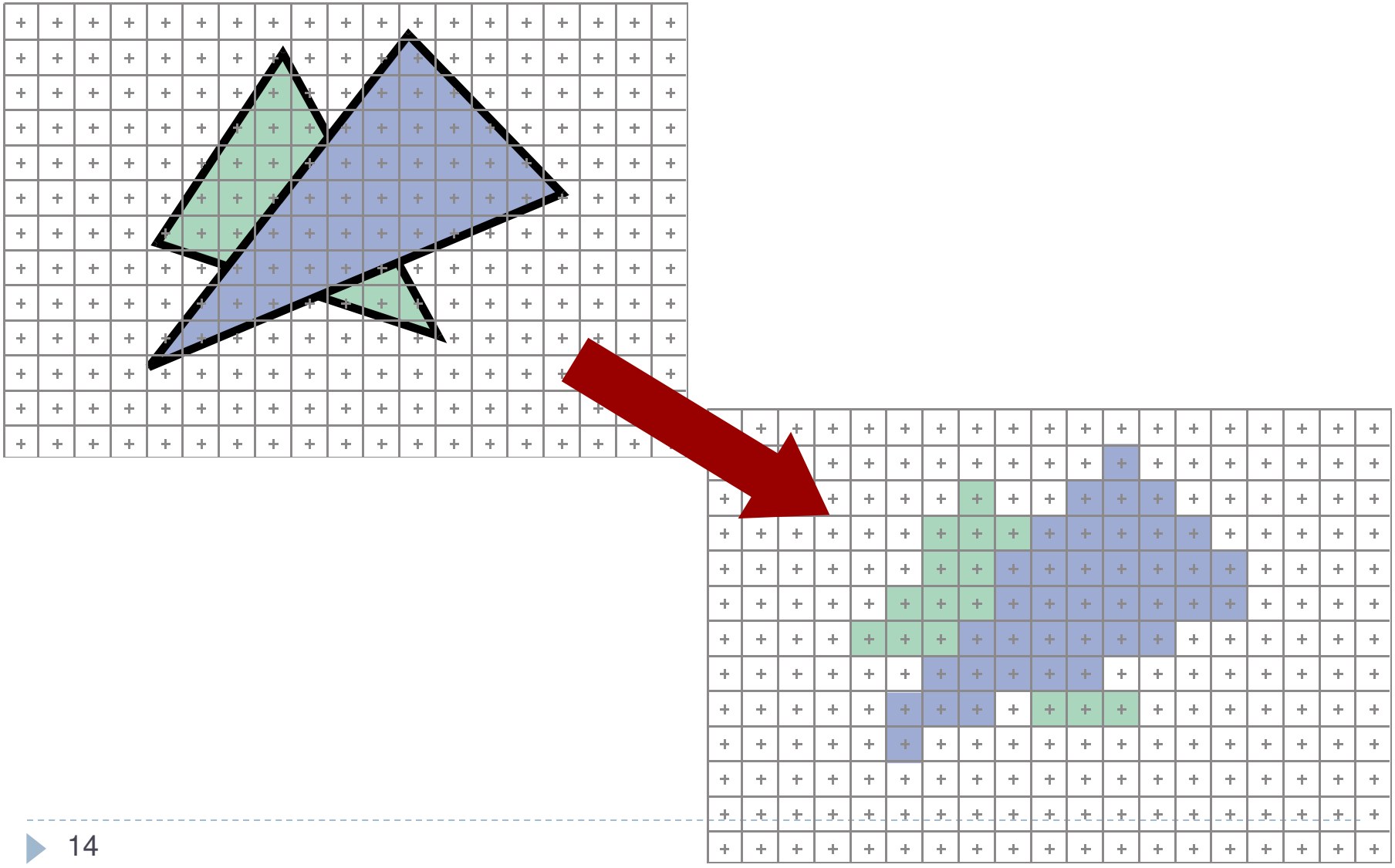
- Culling, Clipping
- Rasterization
- Visibility
- Perspectively correct interpolation

Rendering Pipeline



- Scan conversion and rasterization are synonyms
- One of the main operations performed by GPU
- Draw triangles, lines, points (squares)
- Focus on triangles in this lecture

Rasterization



Rasterization

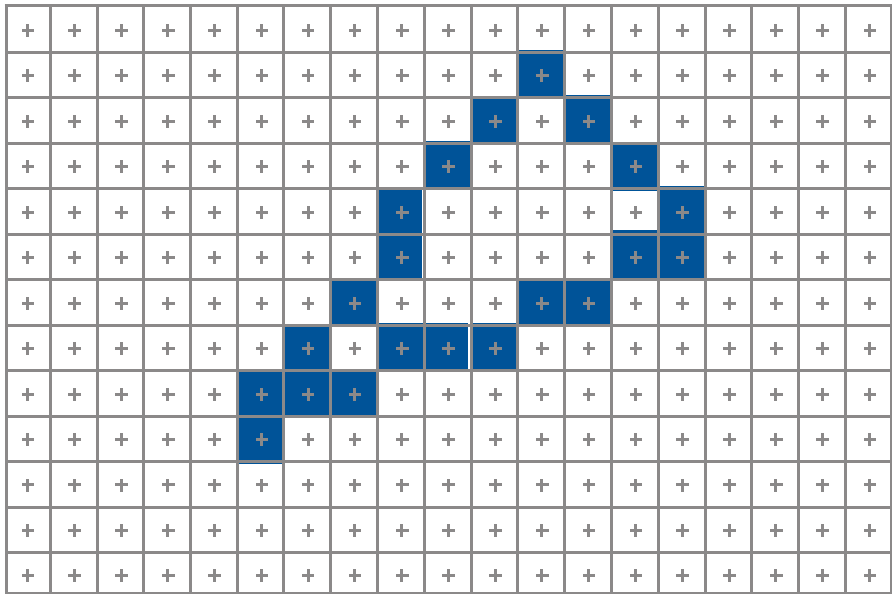
- How many pixels can a modern graphics processor draw per second?

Rasterization

- Rasterization is „hard-coded“ in the graphics card, cannot (currently) be modified by the software
- NVidia GeForce GTX 590
  - 77.7 billion pixels per second (GPix/s)
  - Multiple of what the fastest CPU could do

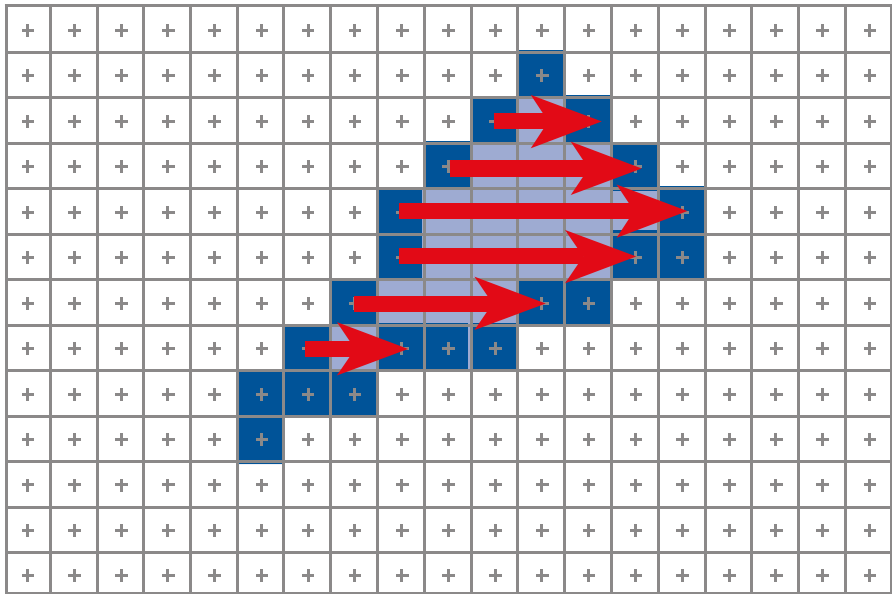
Rasterization

- Many different algorithms
- Old style
  - Rasterize edges first



Rasterization

- Many different algorithms
- Old style
  - Rasterize edges first
  - Fill the spans (scan lines, scan conversion)



Rasterization

- Many different algorithms exist
- Old style
  - Rasterize edges first
  - Fill the spans (scan lines, scan conversion)
  - Requires clipping
  - Straightforward, but not used for hardware implementation today

Rasterization

- GPU rasteriazation today based on "Homogeneous Rasterization"

  http://www.ece.unm.edu/course/ece595/docs/olano.pdf

  Olano, Marc and Trey Greer, "Triangle Scan Conversion Using 2D Homogeneous Coordinates", Proceedings of the 1997 SIGGRAPH/Eurographics Workshop on Graphics Hardware (Los Angeles, CA, August 2-4, 1997), ACM SIGGRAPH, New York, 1995.

- Does not require full clipping, does not perform homogeneous division at vertices
- Today in class
  - Simpler algorithm based on barycentric coordinates
  - More sophisticated than old style algorithm
  - Easy to implement
  - Requires clipping

Rasterization

- Given vertices in pixel coordinates

$$\mathbf{p}' = \mathbf{DPC}^{-1}\mathbf{Mp}$$

World space

Camera space

Clip space

Image space

$$\mathbf{p}' = \begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix}$$

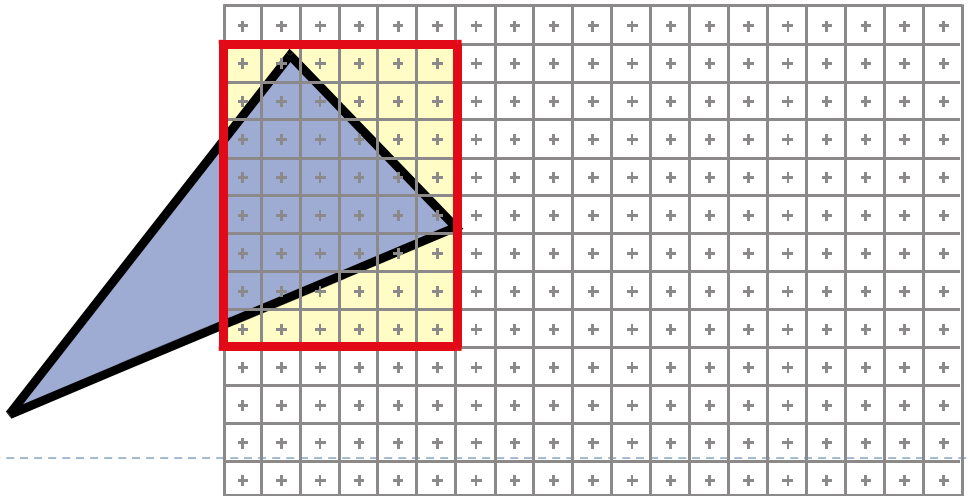
Pixel coordinates $\begin{matrix} x'/w' \\ y'/w' \end{matrix}$

Rasterization

- Simple algorithm

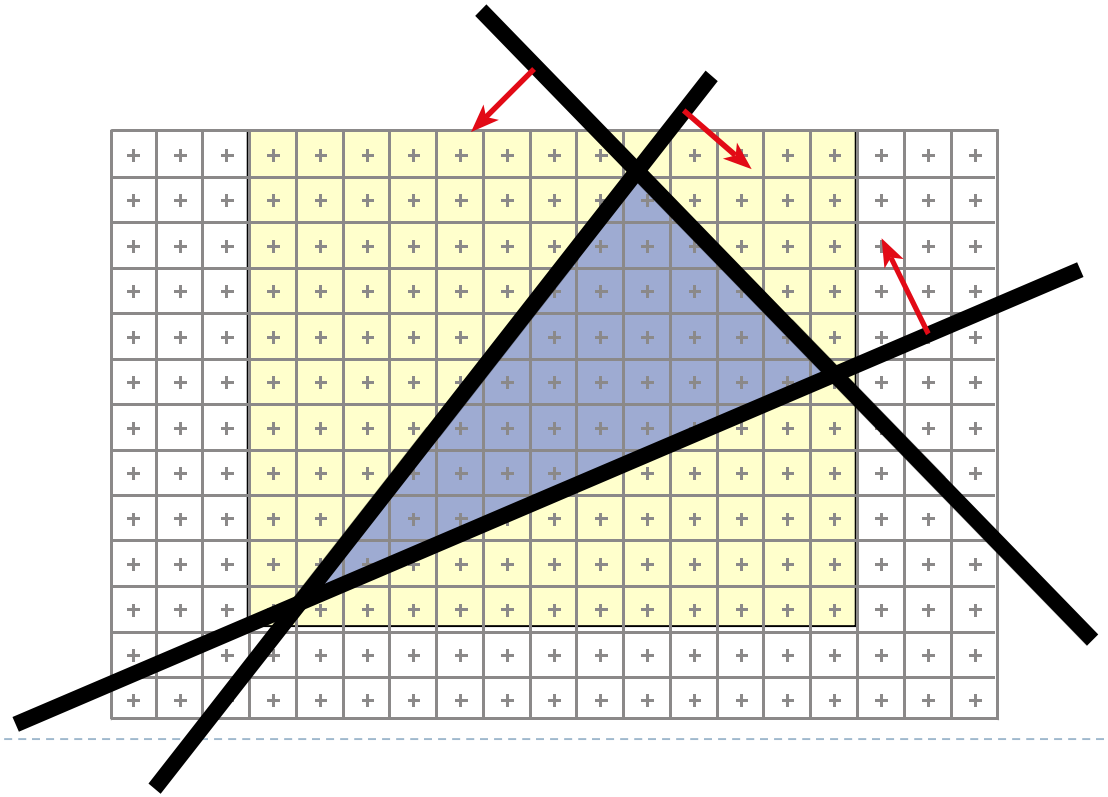
```
compute bbox
clip bbox to screen limits
for all pixels [x,y] in bbox
   compute barycentric coordinates alpha, beta, gamma
   if 0<alpha,beta,gamma<1 //pixel in triangle
            image[x,y]=triangleColor
```

- Bounding box clipping trivial



Rasterization

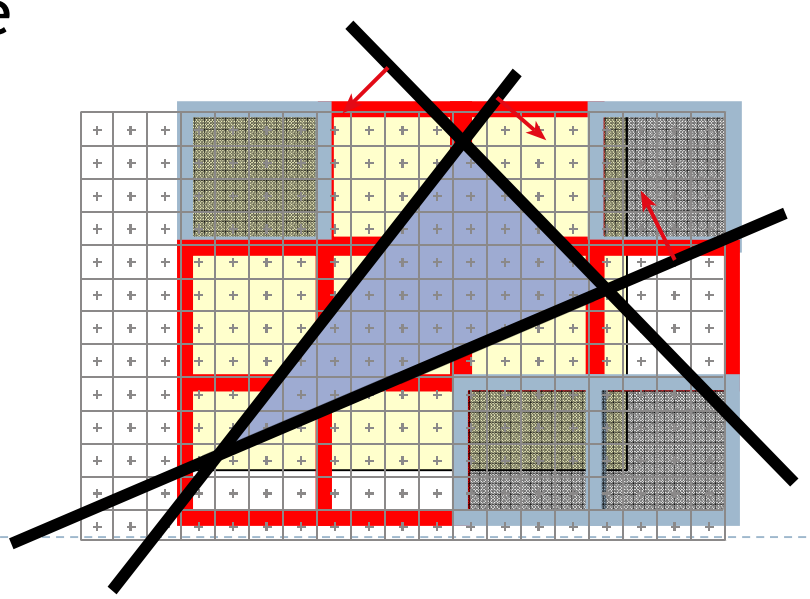
- So far, we compute barycentric coordinates of many useless pixels
- How can this be improved?



Rasterization

Hierarchy

- If block of pixels is outside triangle, no need to test individual pixels
- Can have several levels, usually two-level
- Find right granularity and size of blocks for optimal performance



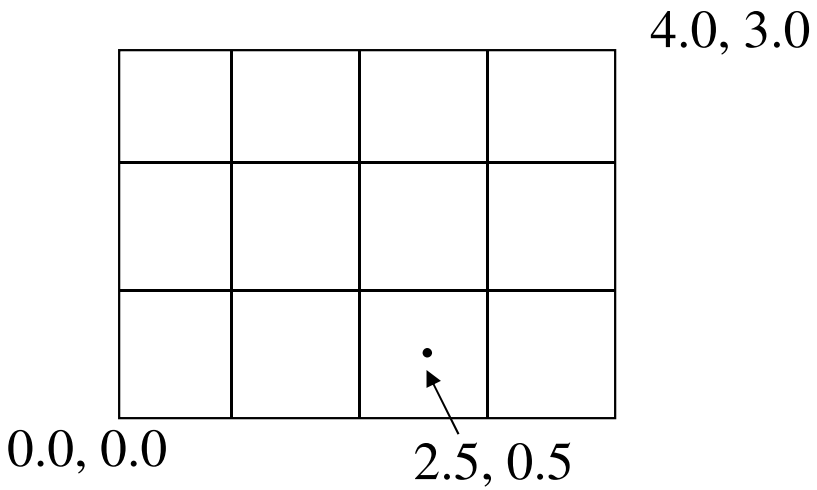
2D Triangle-Rectangle Intersection

- If one of the following tests returns true, the triangle intersects the rectangle:
  - Test if any of the triangle's vertices are inside the rectangle (e.g., by comparing the x/y coordinates to the min/max x/y coordinates of the rectangle)
  - Test if one of the quad's vertices is inside the triangle (e.g., using barycentric coordinates)
  - Intersect all edges of the triangle with all edges of the rectangle

Rasterization

Where is the center of a pixel?

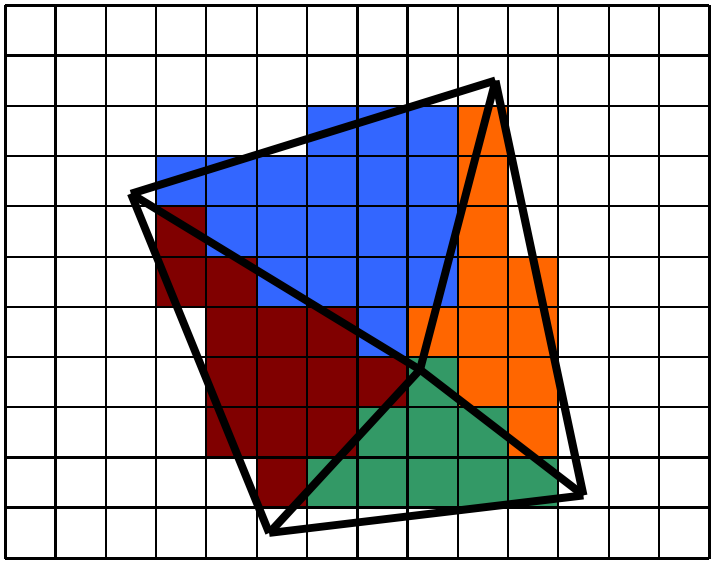
- Depends on conventions
- With our viewport transformation:
  - 800 x 600 pixels ⇔ viewport coordinates are in [0…800]x[0…600]
  - Center of lower left pixel is 0.5, 0.5
  - Center of upper right pixel is 799.5, 599.5



Rasterization

Shared Edges

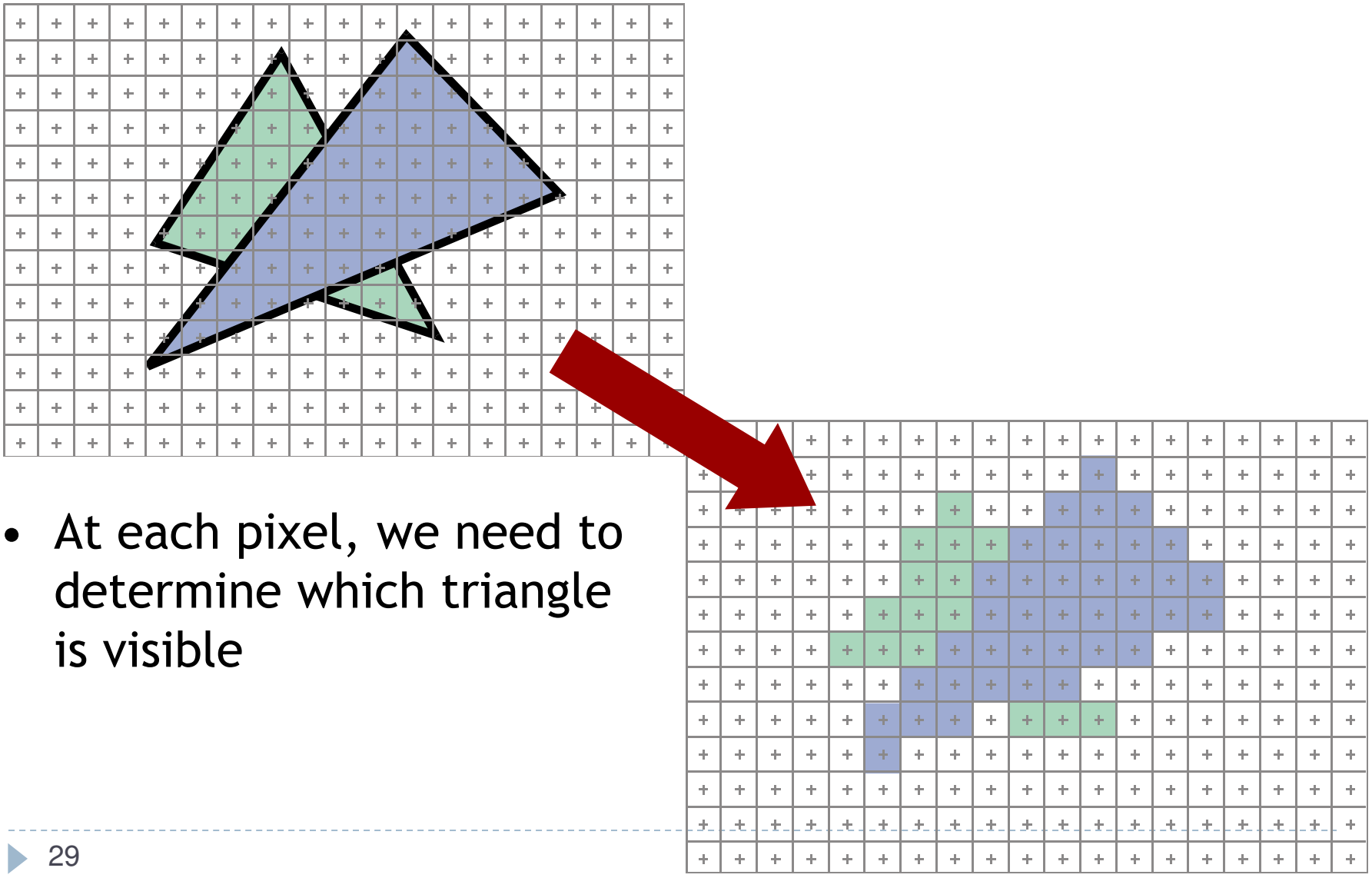
- Each pixel needs to be rasterized exactly once
- Resulting image is independent of drawing order
- Rule: If pixel center exactly touches an edge or vertex
  - Fill pixel only if triangle extends to the right or down



Lecture Overview

- Culling, Clipping
- Rasterization
- Visibility
- Perspectively correct interpolation

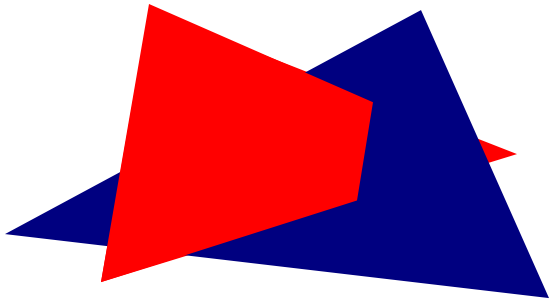
Visibility



- At each pixel, we need to determine which triangle is visible

Painter's Algorithm

- Paint from back to front
- Every new pixel always paints over previous pixel in frame buffer
- Need to sort geometry according to depth
- May need to split triangles if they intersect



- Old style, before memory became cheap

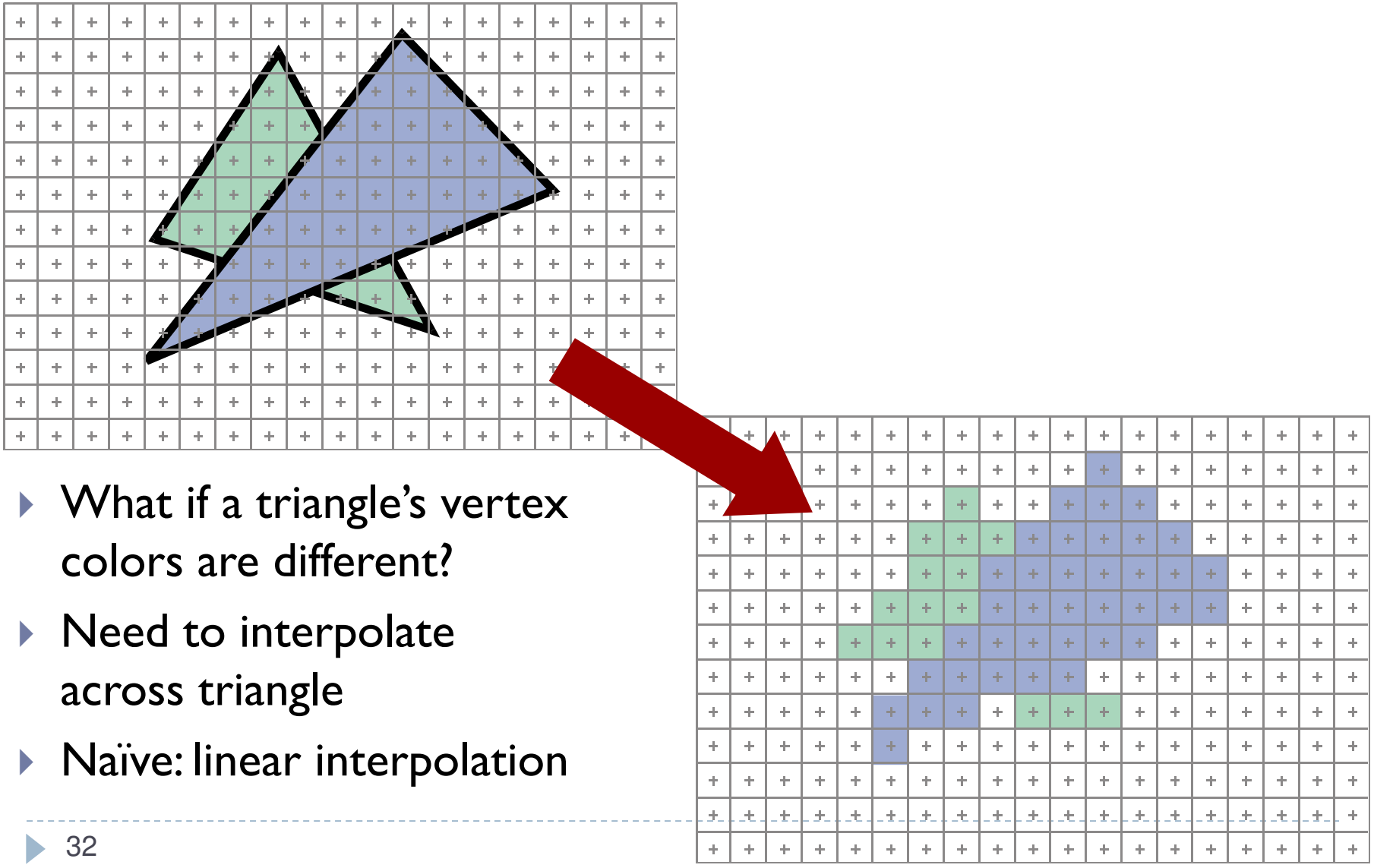
Z-Buffering

- Store z-value for each pixel
- Depth test
  - During rasterization, compare stored value to new value
  - Update pixel only if new value is smaller

```
setpixel(int x, int y, color c, float z)
if(z<zbuffer(x,y)) then
  zbuffer(x,y) = z
  color(x,y) = c
```

- z-buffer is dedicated memory reserved for GPU (graphics memory)
- Depth test is performed by GPU

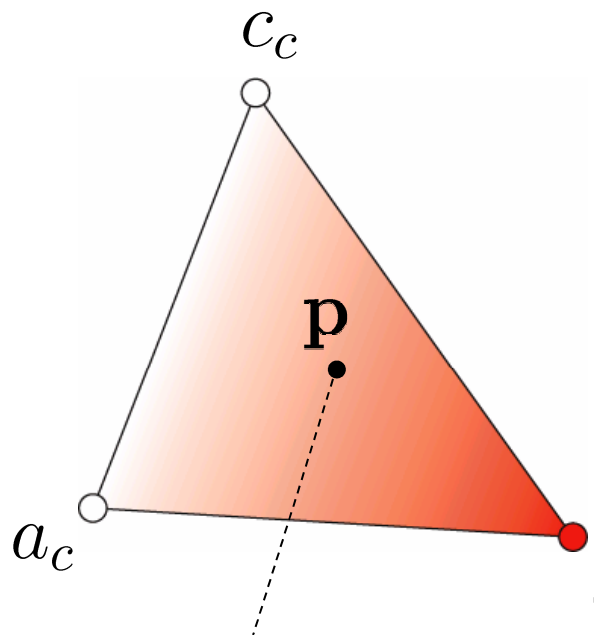
Rasterization



- What if a triangle's vertex colors are different?
- Need to interpolate across triangle
- Naïve: linear interpolation

Barycentric Interpolation

- Interpolate values across triangles, e.g., colors



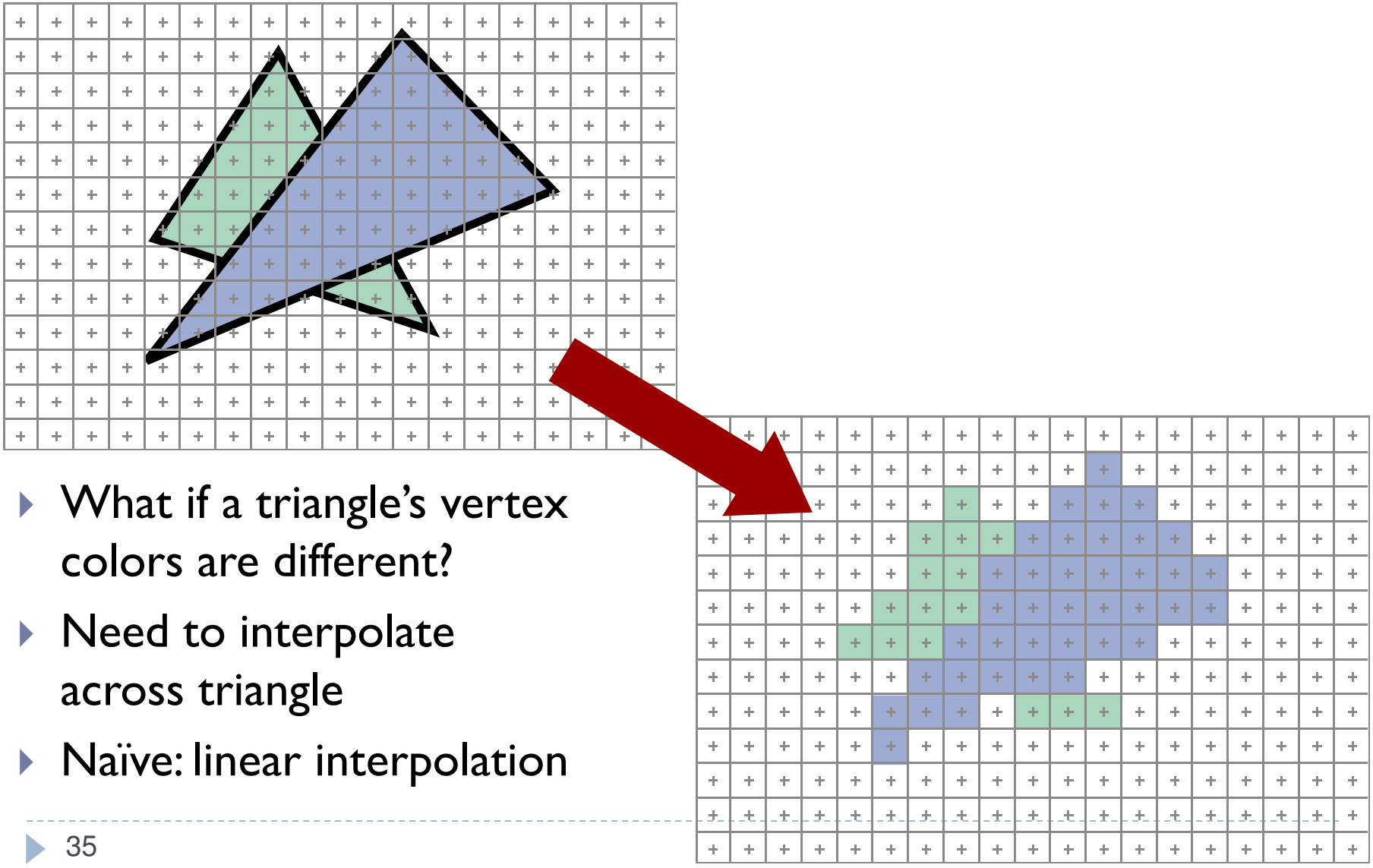
$$c(\mathbf{p}) = \alpha(\mathbf{p})a_c + \beta(\mathbf{p})b_c + \gamma(\mathbf{p})c_c$$

- Linear interpolation on triangles
  - Barycentric coordinates

Lecture Overview

- Culling, Clipping
- Rasterization
- Visibility
- Perspectively correct interpolation

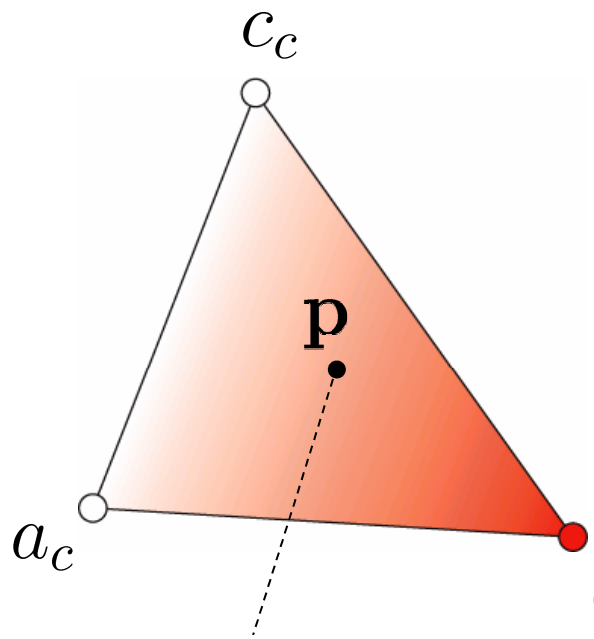
Rasterization



- What if a triangle's vertex colors are different?
- Need to interpolate across triangle
- Naïve: linear interpolation

Barycentric Interpolation

- Interpolate values across triangles, e.g., colors



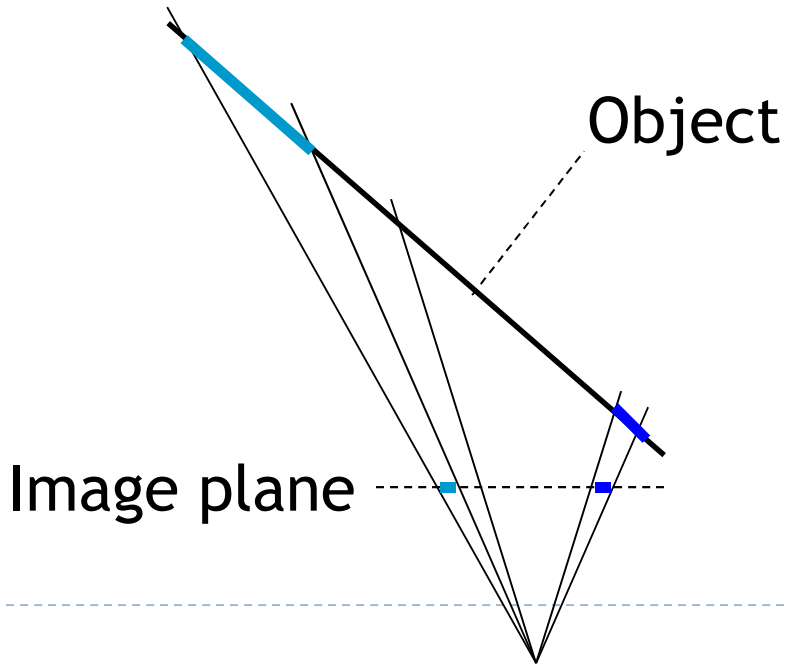
$$c(\mathbf{p}) = \alpha(\mathbf{p})a_c + \beta(\mathbf{p})b_c + \gamma(\mathbf{p})c_c$$

- Linear interpolation on triangles
  - Barycentric coordinates

Perspectively Correct Interpolation

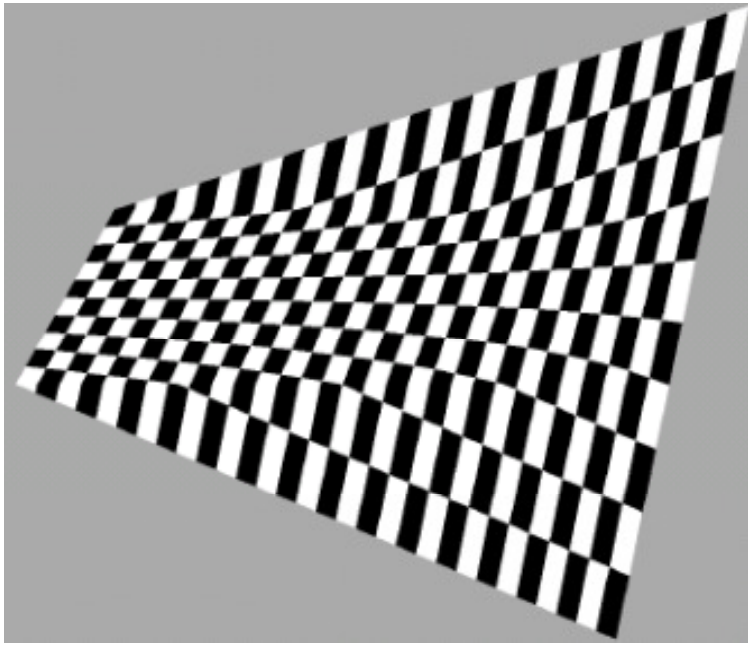
Problem

- Barycentric (linear) interpolation in image coordinates does not correspond to barycentric interpolation in camera space
- Equal step size on image plane does not correspond to equal step size on object

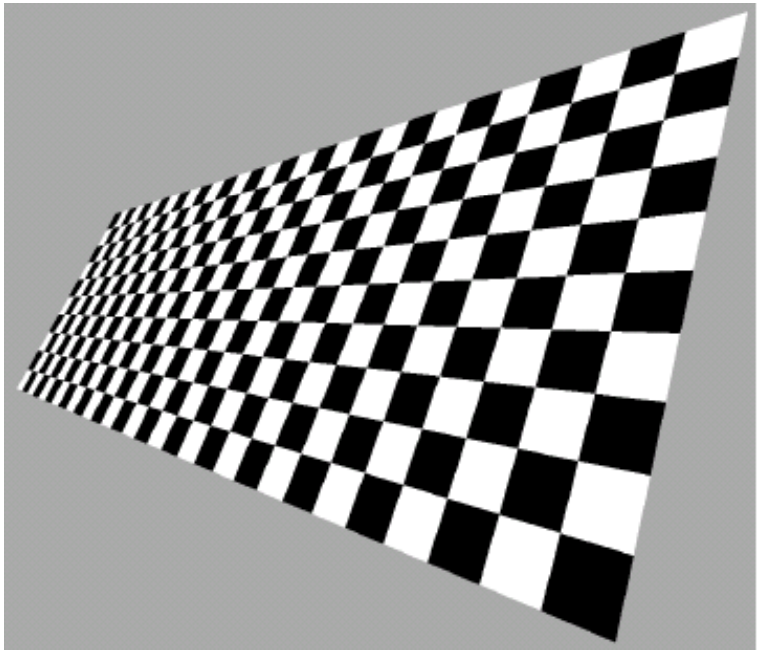


Perspectively Correct Interpolation

Linear interpolation
in image coordinates



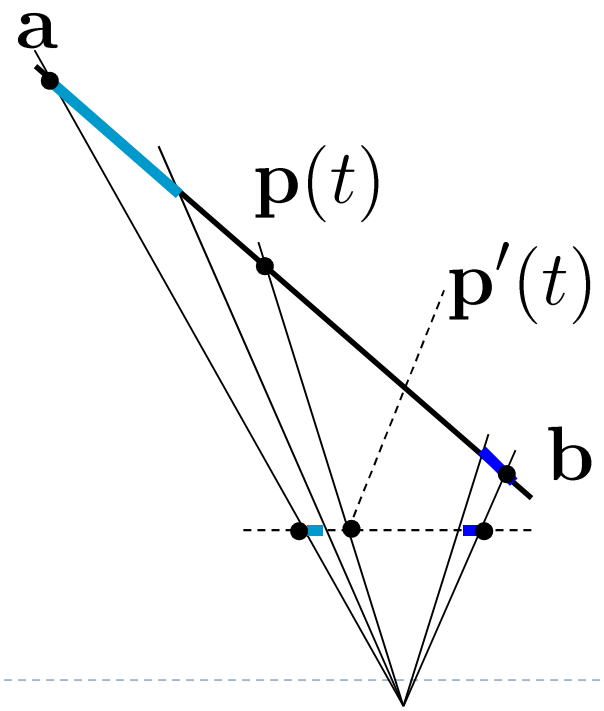
Perspectively correct
interpolation in
object coordinates



Perspective Projection Revisited

- Vertices **a**, **b** before projection
- Linear interpolation: $\mathbf{p}(t) = \mathbf{a} + t(\mathbf{b} - \mathbf{a})$
- Perspective projection, homogeneous division

$$\mathbf{p}'(t) = \frac{\mathbf{a} + t(\mathbf{b} - \mathbf{a})}{a_w + t(b_w - a_w)}$$



Perspective Projection Revisited

- Rewrite

$$\frac{\mathbf{a} + t(\mathbf{b} - \mathbf{a})}{a_w + t(b_w - a_w)} = \frac{\mathbf{a}}{a_w} + s(t)\left(\frac{\mathbf{b}}{b_w} - \frac{\mathbf{a}}{a_w}\right)$$

  with

$$s(t) = \frac{b_w t}{a_w + t(b_w - a_w)}$$

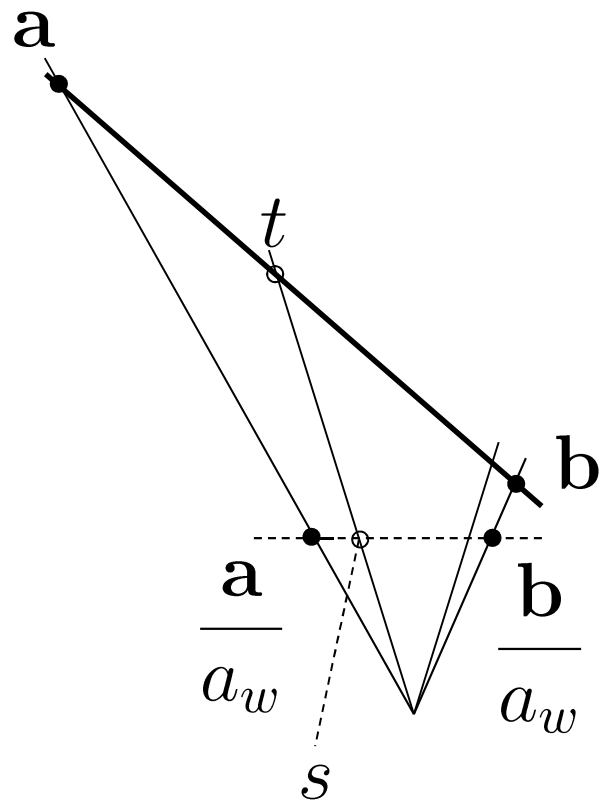
- s is linear interpolation weight in image space
- Straight lines are preserved
- Interpolation speed is different in *s* and *t*

Perspective Projection Revisited

- Relation between parameters

$$s(t) = \frac{b_w t}{a_w + t(b_w - a_w)}$$

$$t(s) = \frac{a_w s}{b_w + s(a_w - b_w)}$$



Perspective Projection Revisited

- Relation between parameters:

$$s(t) = \frac{b_w t}{a_w + t(b_w - a_w)}$$

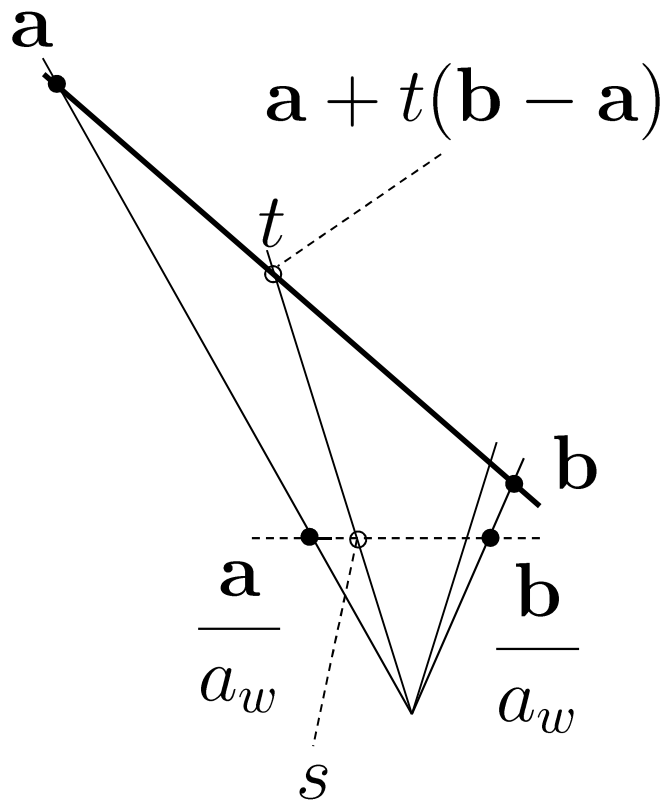
$$t(s) = \frac{a_w s}{b_w + s(a_w - b_w)}$$

- Projection after interpolation:

$$\frac{\mathbf{a} + t(\mathbf{b} - \mathbf{a})}{a_w + t(b_w - a_w)}$$

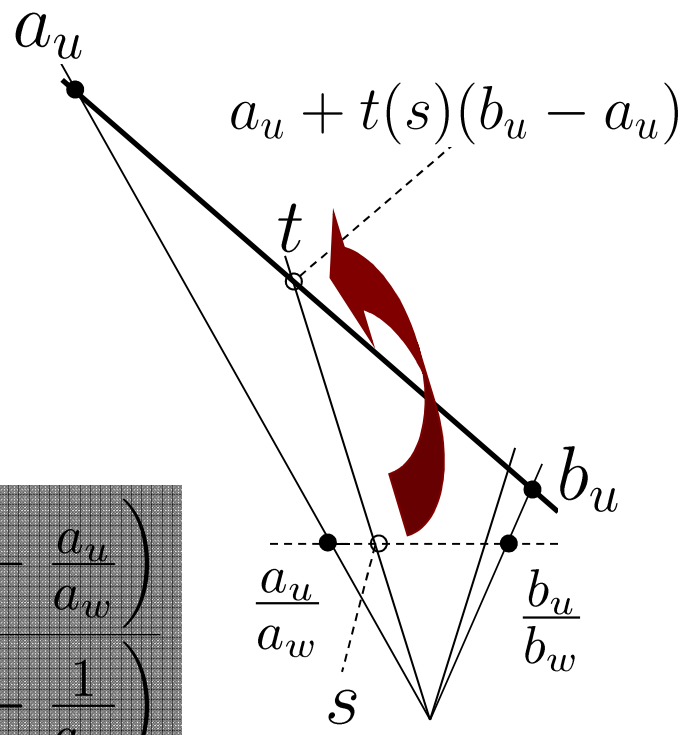
- Interpolation after projection:

$$\frac{\mathbf{a}}{a_w} + s\left(\frac{\mathbf{b}}{b_w} - \frac{\mathbf{a}}{a_w}\right)$$



Perspectively Correct Interpolation

- In order to interpolate (in image space) any vertex attribute we need to compute a_u and b_u



- Hyperbolic interpolation:

$$a_u + t(s)(b_u - a_u)$$

$$a_u + t(s)(b_u - a_u) = \frac{\frac{a_u}{a_w} + s\left(\frac{b_u}{b_w} - \frac{a_u}{a_w}\right)}{\frac{1}{a_w} + s\left(\frac{1}{b_w} - \frac{1}{a_w}\right)}$$

Perspectively Correct Interpolation

Hyperbolic Interpolation

- Note $$\frac{1}{a_w} + s(t)\left(\frac{1}{b_w} - \frac{1}{a_w}\right) = \frac{1}{a_w + t(b_w - a_w)} \equiv \frac{1}{w}$$
- Recipe: given parameter s in image space

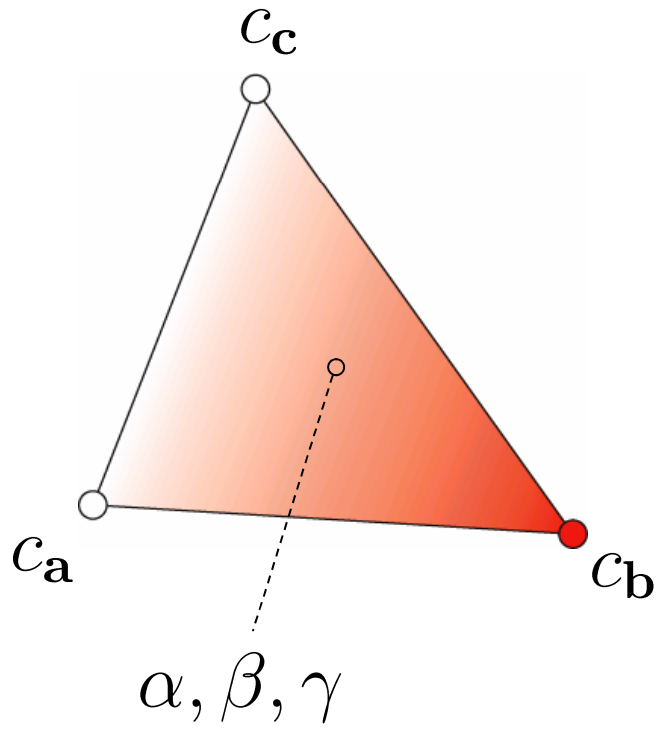
1.
2. $$\frac{1}{w} = \frac{1}{a_w} + s\left(\frac{1}{b_w} - \frac{1}{a_w}\right) = (1-s)\frac{1}{a_w} + s\frac{1}{b_w}$$
3. $$\frac{u}{w} = \frac{a_u}{a_w} + s\left(\frac{b_u}{b_w} - \frac{a_u}{a_w}\right) = (1-s)\frac{a_u}{a_w} + s\frac{b_u}{b_w}$$

$$u = \frac{u}{w} \Big/ \frac{1}{w}$$

Perspectively Correct Interpolation

- Works for triangles with barycentric coordinates
- Given point in image space with barycentric coordinates α, β, γ

1. $$\frac{1}{w} = \alpha\frac{1}{a_w} + \beta\frac{1}{b_w} + \gamma\frac{1}{c_w}$$
2. $$\frac{c}{w} = \alpha\frac{a_c}{a_w} + \beta\frac{b_c}{b_w} + \gamma\frac{c_c}{c_w}$$
3. $$c = \frac{c}{w} \Big/ \frac{1}{w}$$



Next Lecture

- Color