University of British Columbia
CPSC 314 Computer Graphics
May-June 2005

Tamara Munzner

Rasterization, Interpolation, Vision/Color

Week 2, Thu May 19

http://www.ugrad.cs.ubc.ca/~cs314/Vmay2005

News

- reminder: extra lab coverage with TAs
  - 12-2 Mondays, Wednesdays
  - for rest of term
  - just for answering questions, no presentations
- signup sheet for P1 demo time
  - Friday 12-5

Reading: Today

- FCG Section 2.11 Triangles (Barycentric Coordinates) p 42-46
- FCG Chap 3 Raster Algorithms, p 49-65
  - except 3.8
- FCG Chap 17 Human Vision, p 293-298
- FCG Chap 18 Color, p 301-311
  - until Section 18.9 Tone Mapping

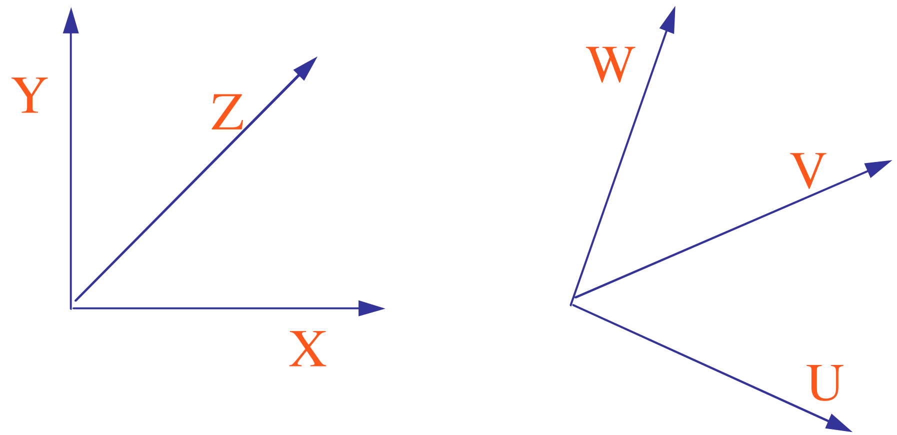
FCG Errata

- p 54
  - triangle at bottom of figure shouldn't have black outline
- p 63
  - The test if numbers a [x] and b [y] have the same sign can be implemented as the test ab [xy] > 0.

Reading: Next Time

- FCG Chap 8, Surface Shading, p 141-150
- RB Chap Lighting

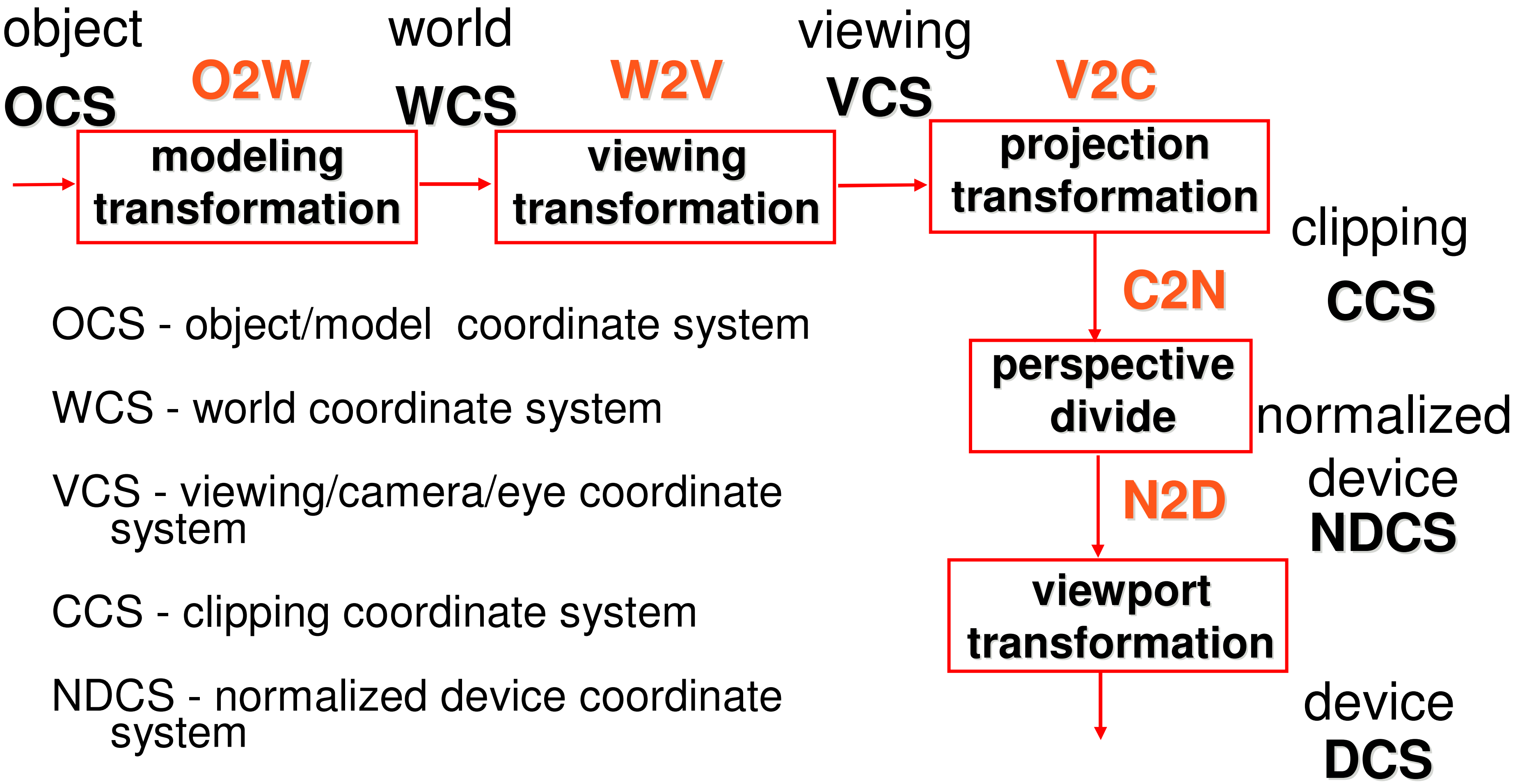
Clarification: Arbitrary Rotation



- problem:
  - given two orthonormal coordinate systems *XYZ* and *UVW*
  - find transformation from *XYZ* to *UVW*
  - answer:
  - transformation matrix R whose columns are *U,V,W:*

$$R = \begin{bmatrix} u_x & v_x & w_x \\ u_y & v_y & w_y \\ u_z & v_z & w_z \end{bmatrix}$$

Review: Projective Rendering Pipeline



OCS - object/model coordinate system

WCS - world coordinate system

VCS - viewing/camera/eye coordinate system

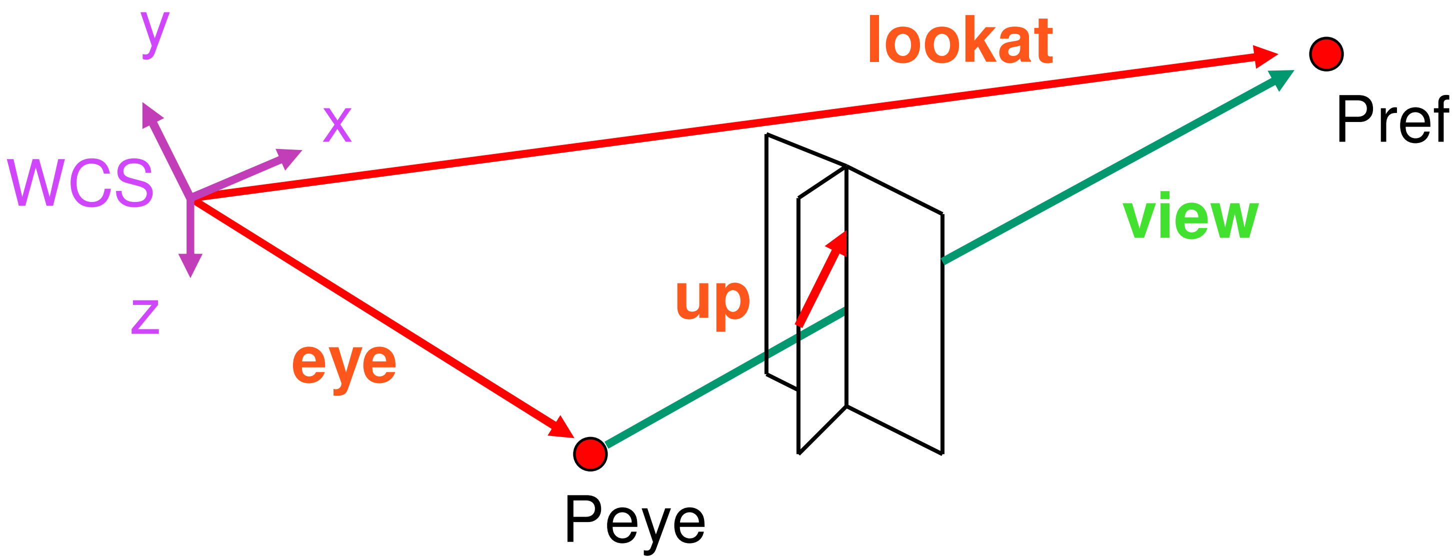
CCS - clipping coordinate system

NDCS - normalized device coordinate system

DCS - device/display/screen coordinate system

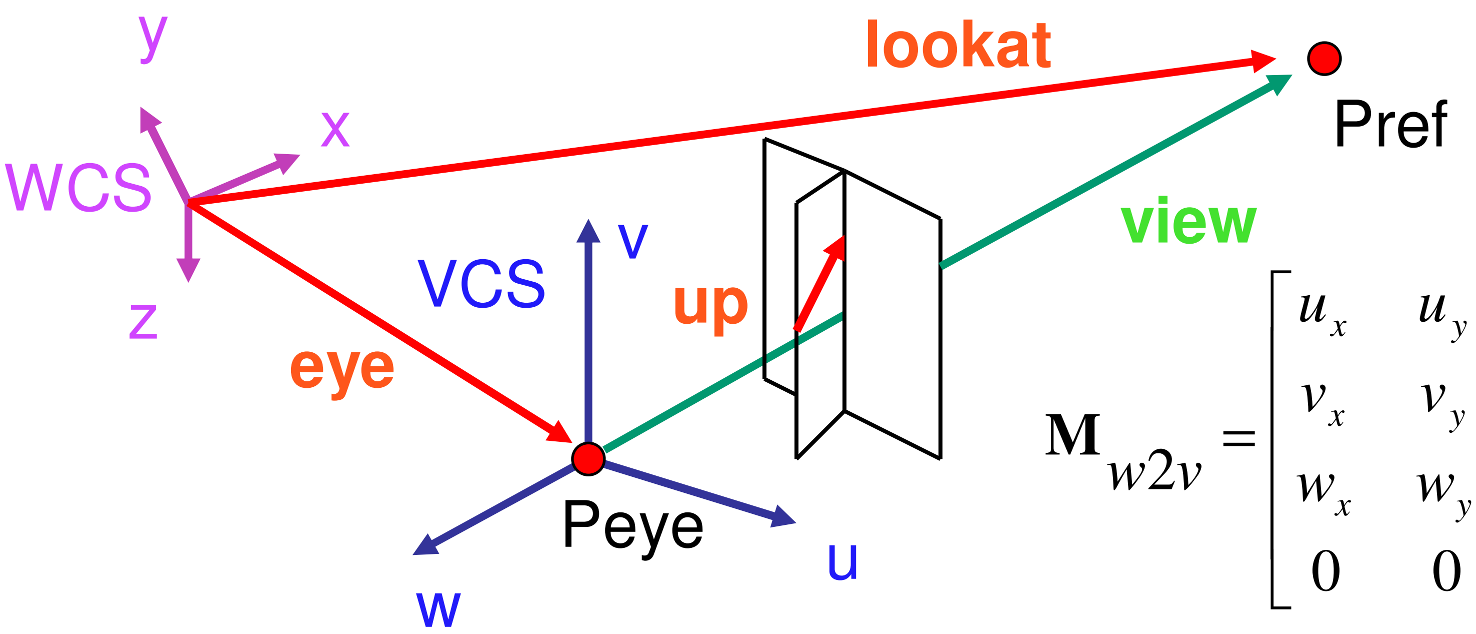
Review: Camera Motion

- rotate/translate/scale difficult to control
- arbitrary viewing position
  - eye point, gaze/lookat direction, up vector



Review: World to View Coordinates

- translate **eye** to origin
- rotate **view** vector (**lookat** – **eye**) to **w** axis
- rotate around **w** to bring **up** into **vw**-plane

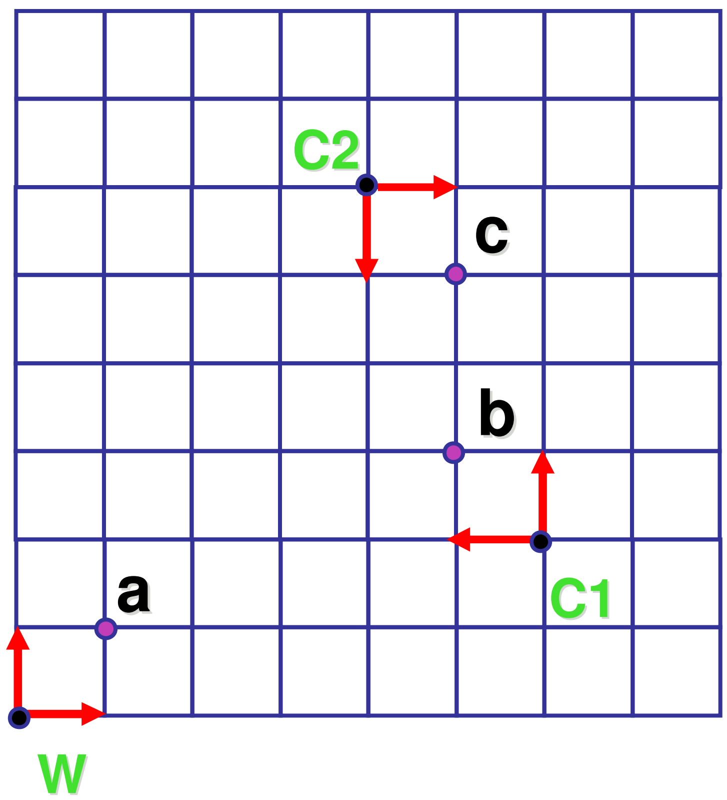


$$\mathbf{M}_{w2v} = \begin{bmatrix} u_x & u_y & u_z & -\mathbf{u} \bullet \mathbf{e} \\ v_x & v_y & v_z & -\mathbf{v} \bullet \mathbf{e} \\ w_x & w_y & w_z & -\mathbf{w} \bullet \mathbf{e} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Correction: Moving Camera or World?

- two equivalent operations
  - move camera one way vs. move world other way
- example
  - initial OpenGL camera: at origin, looking along -z axis
  - create a unit square parallel to camera at z = -10
  - translate in z by 3 possible in two ways
    - camera moves to z = -3
      - Note OpenGL models viewing in left-hand coordinates
    - camera stays put, but world moves to -7
  - resulting image same either way
    - possible difference: are lights specified in world or view coordinates?

Correction: World vs. Camera Coordinates



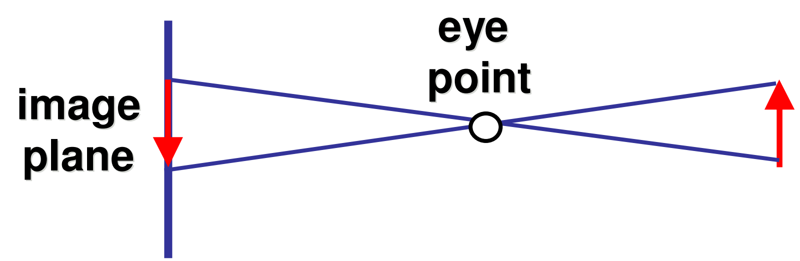
$a = (1,1)_W$

$b = (1,1)_{C1} = (5,3)_W$

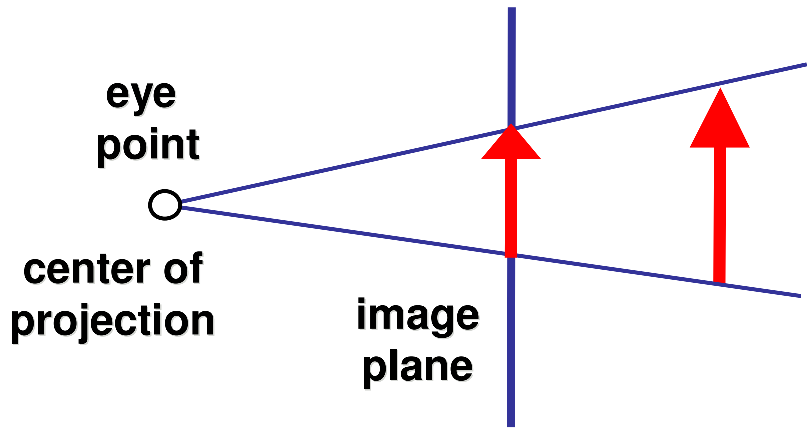
$c = (1,1)_{C2} = (1,3)_{C1} = (5,5)_W$

Review: Graphics Cameras

- real pinhole camera: image inverted

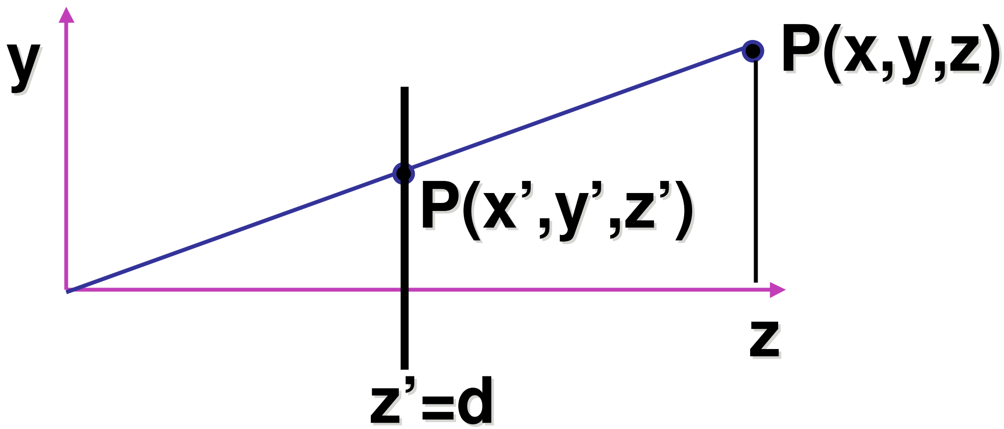


- computer graphics camera: convenient equivalent



Review: Basic Perspective Projection

similar triangles



$$\frac{y'}{d} = \frac{y}{z} \rightarrow y' = \frac{y \cdot d}{z}$$

$$x' = \frac{x \cdot d}{z} \qquad z' = d$$

$$\begin{bmatrix} \frac{x}{z/d} \\ \frac{y}{z/d} \\ d \end{bmatrix}$$

homogeneous coords →

$$\begin{bmatrix} x \\ y \\ z \\ z/d \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix}$$

Correction: Perspective Projection

- desired result for a point $[x, y, z, 1]^T$ projected onto the view plane:

$$\frac{x'}{d} = \frac{x}{z}, \quad \frac{y'}{d} = \frac{y}{z}$$

$$x' = \frac{x \cdot d}{z} = \frac{x}{z/d}, \quad y' = \frac{y \cdot d}{z} = \frac{y}{z/d}, \quad z' = d$$

- what could a matrix look like to do this?

Correction: Simple Perspective Projection Matrix

$$\begin{bmatrix} \dfrac{x}{z/d} \\ \dfrac{y}{z/d} \\ d \\ \boxed{1} \end{bmatrix}$$ is homogenized version of $$\begin{bmatrix} x \\ y \\ z \\ z/d \end{bmatrix}$$

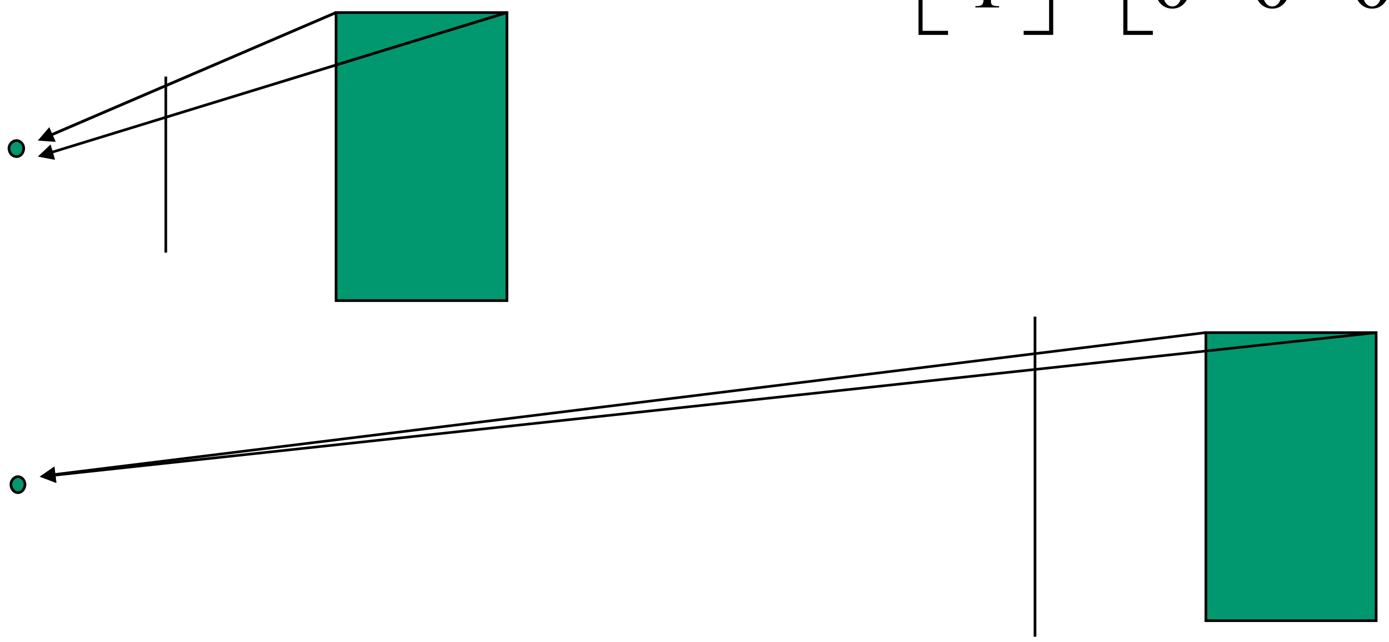
where w = z/d

$$\begin{bmatrix} x \\ y \\ z \\ z/d \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Review: Orthographic Cameras

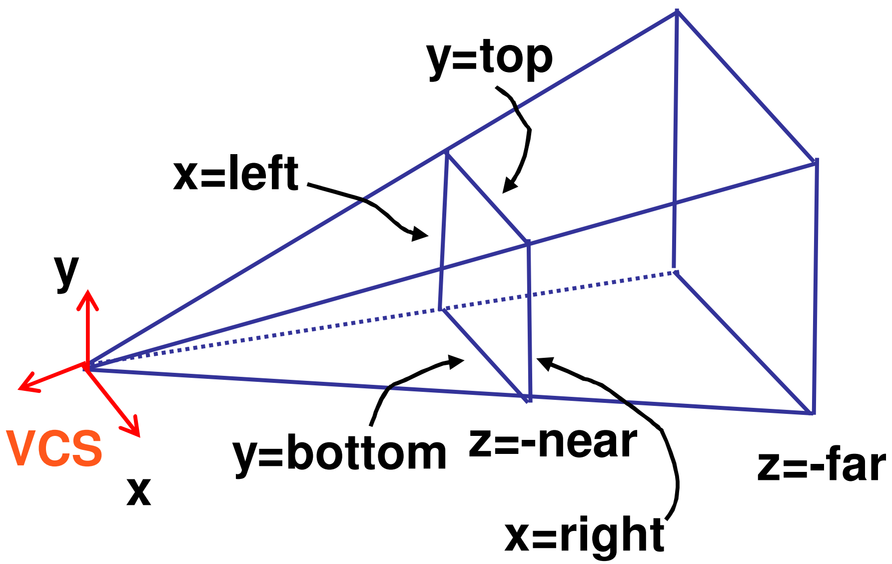
- center of projection at infinity
- no perspective convergence
- just throw away z values

$$\begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

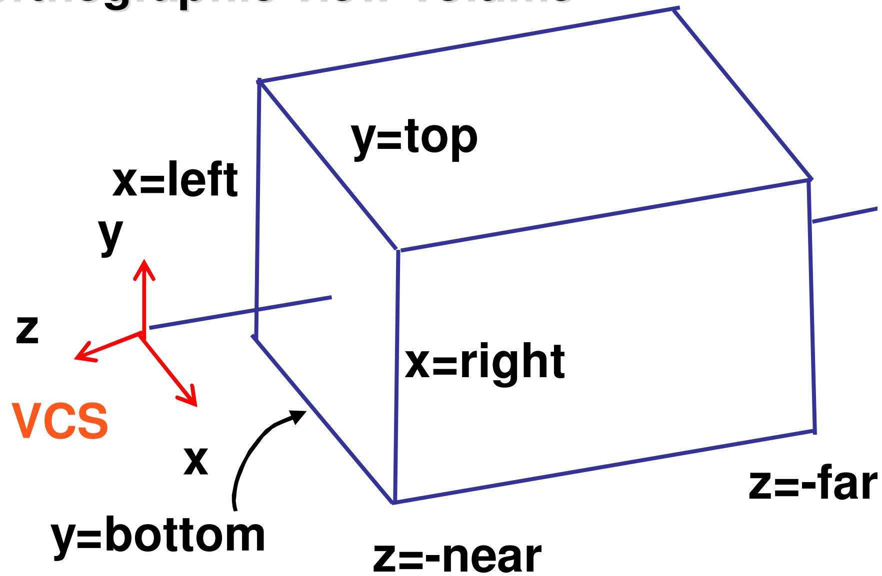


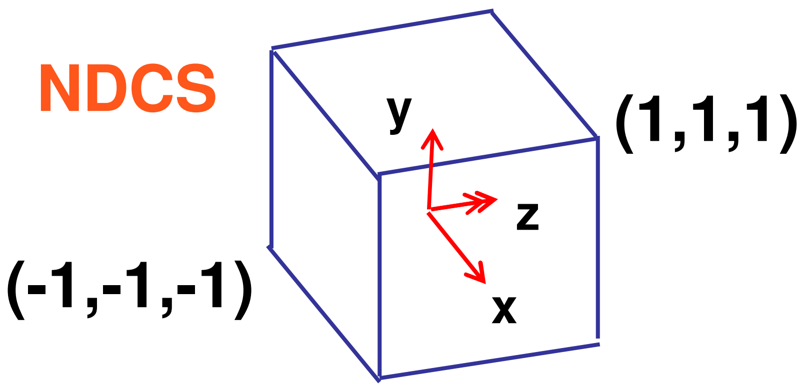
Review: Transforming View Volumes

perspective view volume



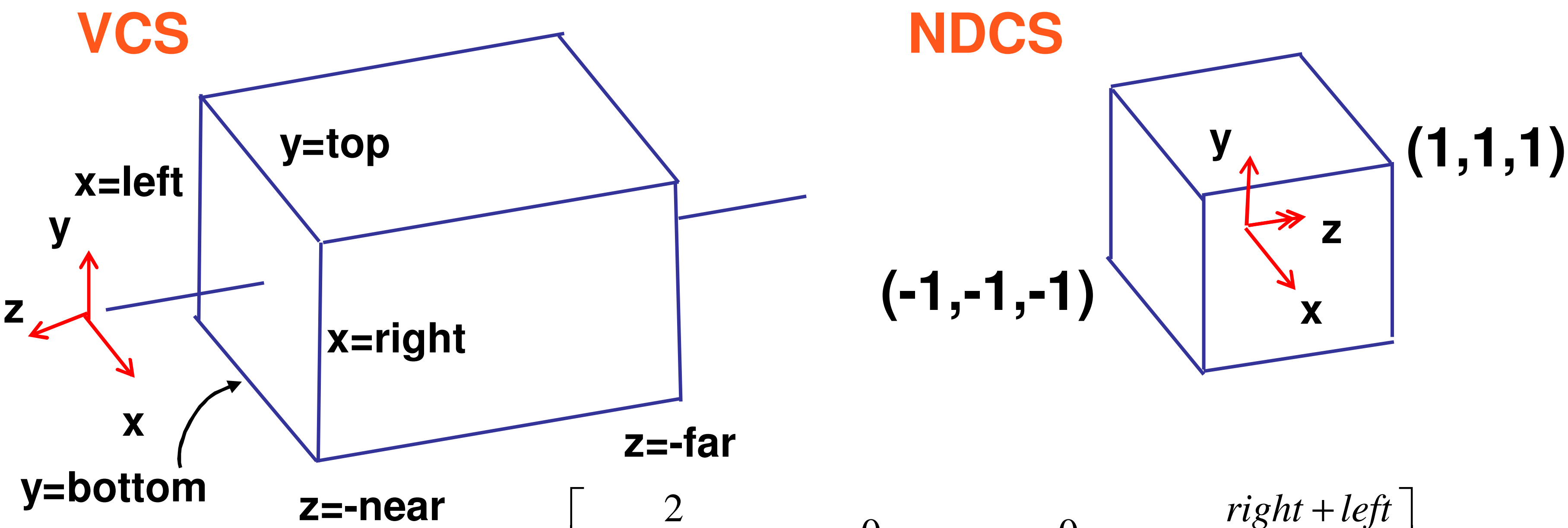
orthographic view volume





Review: Ortho to NDC Derivation

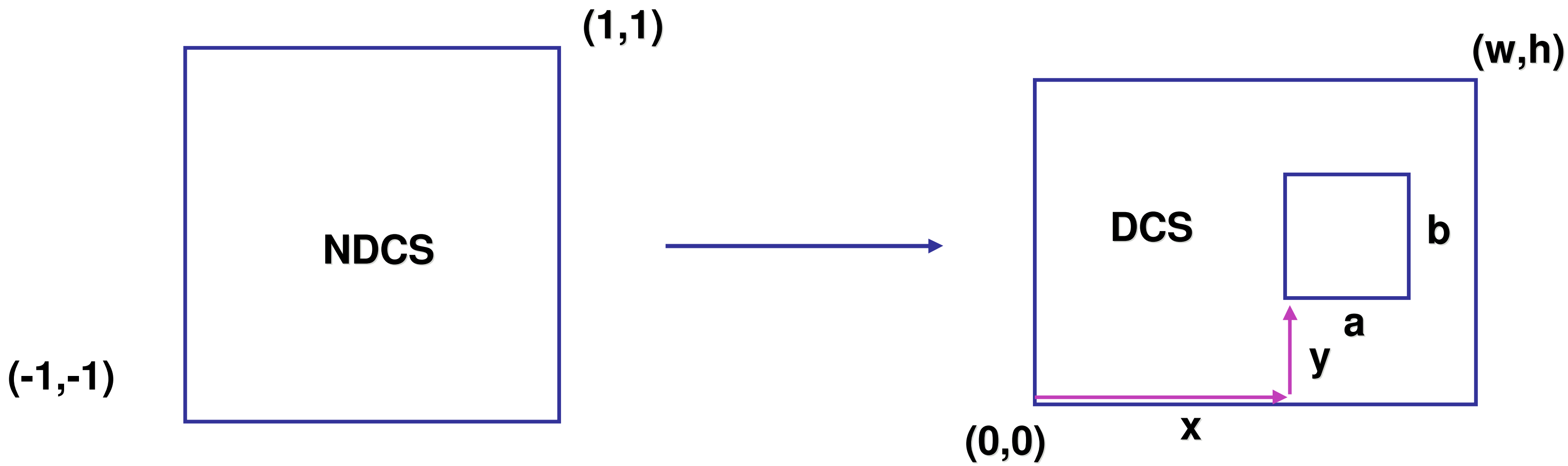
- scale, translate, reflect for new coord sys



$$P' = \begin{bmatrix} \frac{2}{right-left} & 0 & 0 & -\frac{right+left}{right-left} \\ 0 & \frac{2}{top-bot} & 0 & -\frac{top+bot}{top-bot} \\ 0 & 0 & \frac{-2}{far-near} & -\frac{far+near}{far-near} \\ 0 & 0 & 0 & 1 \end{bmatrix} P$$

Review: NDC to Viewport Transformation

- 2D scaling and translation



$$x_{DCS} = w\frac{(x_{NDCS}+1)}{2}$$

$$y_{DCS} = h\frac{(y_{NDCS}+1)}{2}$$

$$z_{DCS} = \frac{(z_{NDCS}+1)}{2}$$

OpenGL

```
glViewport(x,y,a,b);
```

default:

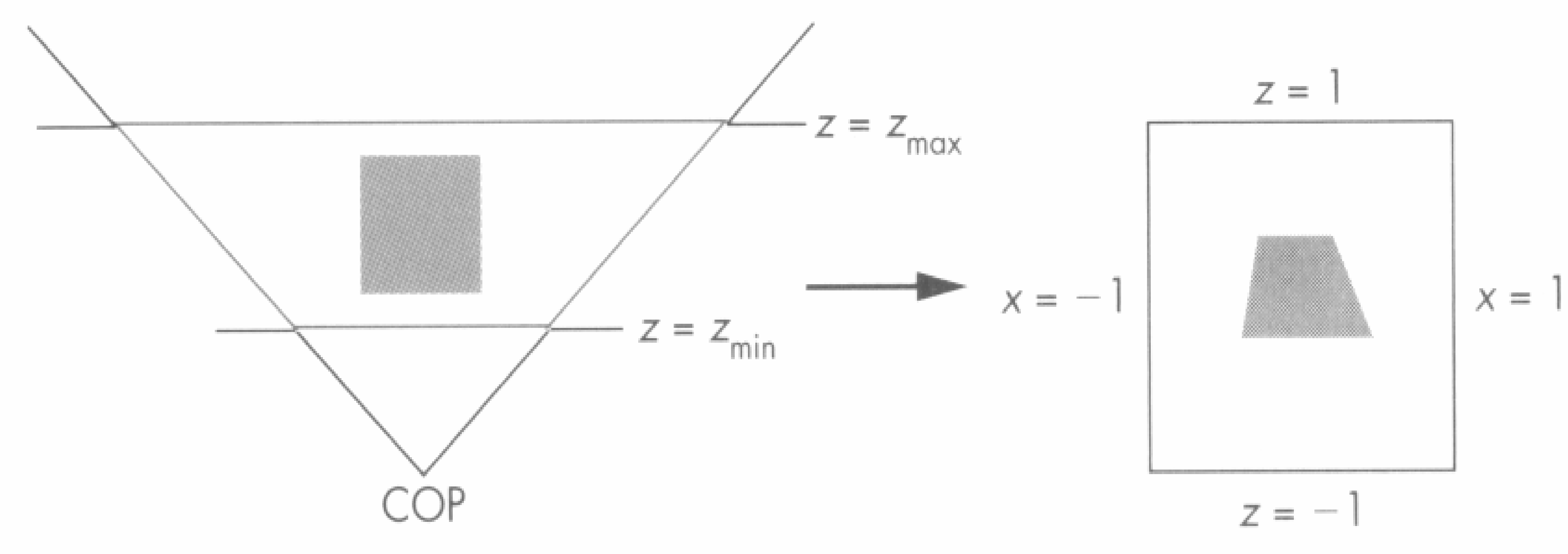
```
glViewport(0,0,w,h);
```

Clarification: N2V Transformation

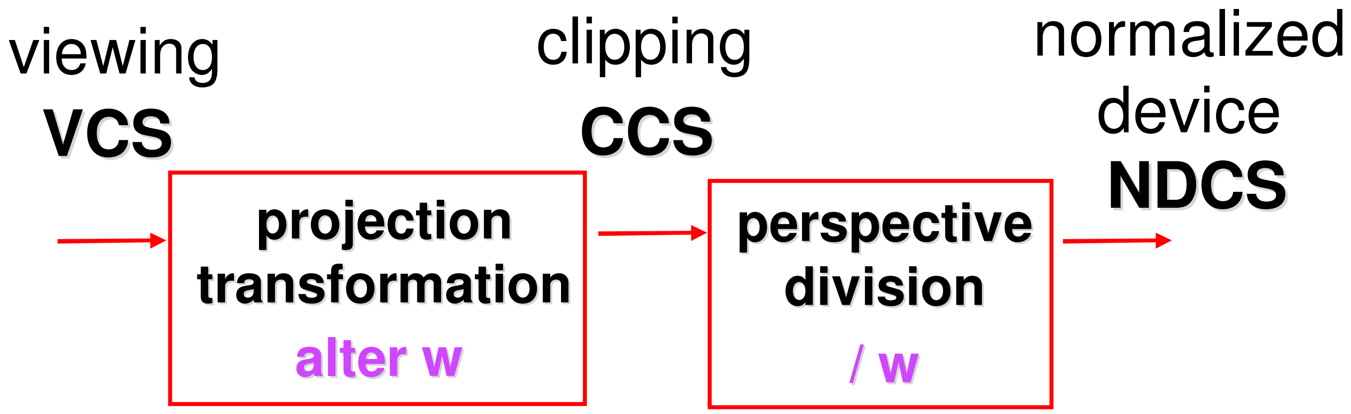
- general formulation
  - translate by
    - x offset, width/2
    - y offset, height/2
  - scale by width/height
  - reflect in y for upper vs. lower left origin
  - FCG includes additional translation for pixel centers at (.5, .5) instead of (0,0)
    - feel free to ignore this

Review: Perspective Normalization

- perspective viewing frustum transformed to cube
- orthographic rendering of cube produces same image as perspective rendering of original

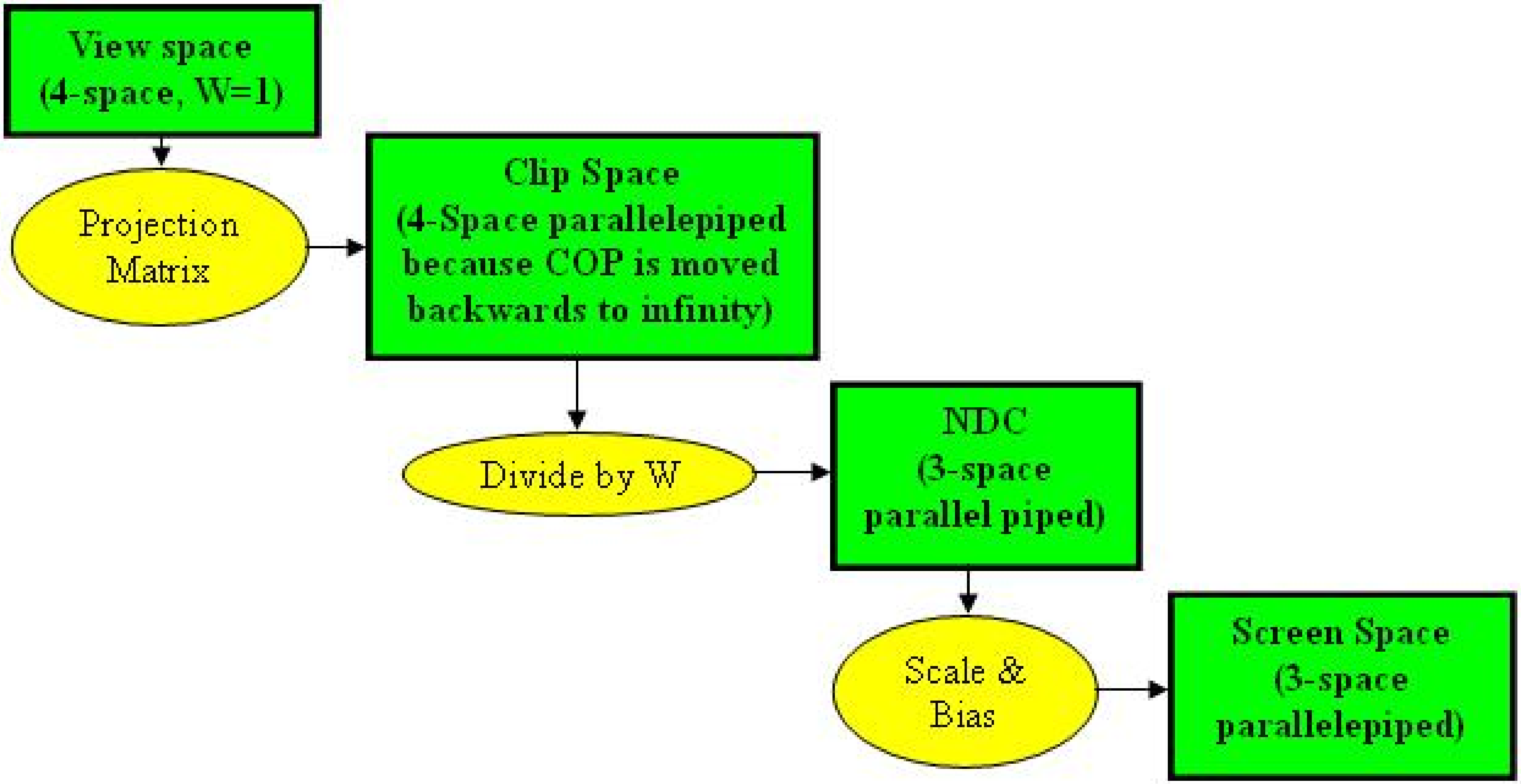


Review: Perspective Normalization



- distort such that orthographic projection of distorted objects is desired persp projection
  - separate division from standard matrix multiplies
  - clip after warp, before divide
  - division: normalization

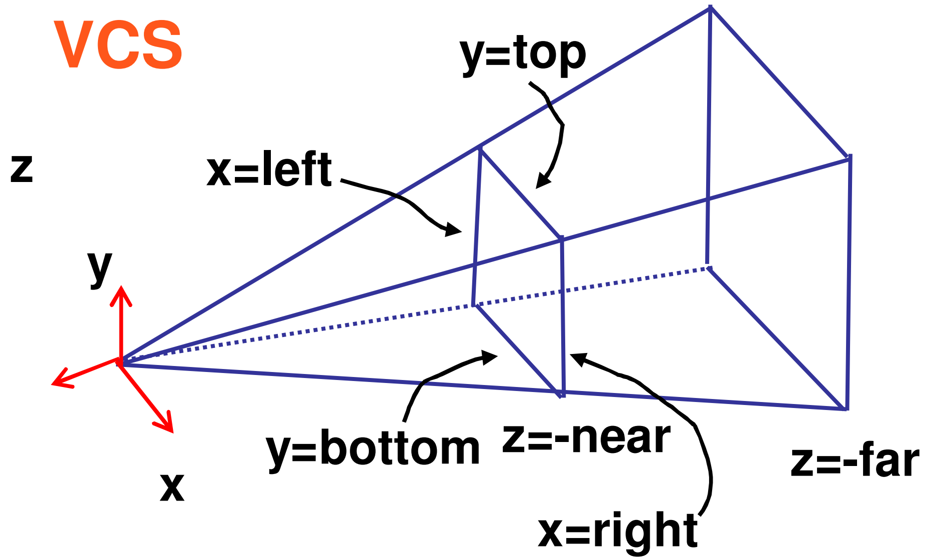
Review: Coordinate Systems

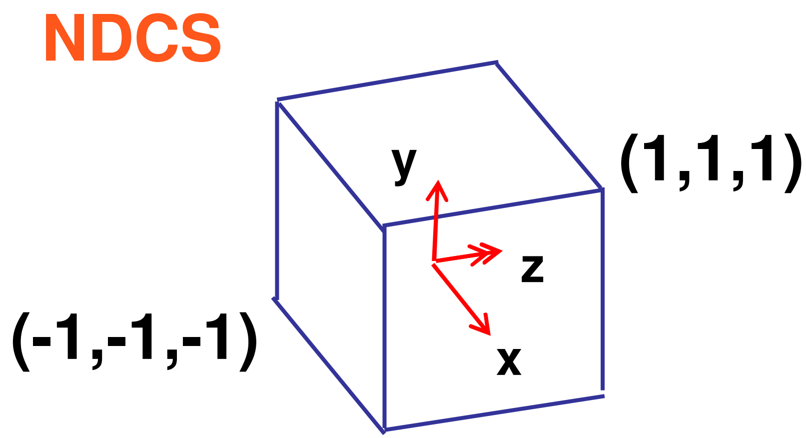


http://www.btinternet.com/~danbgs/perspective/

Review: Perspective Derivation

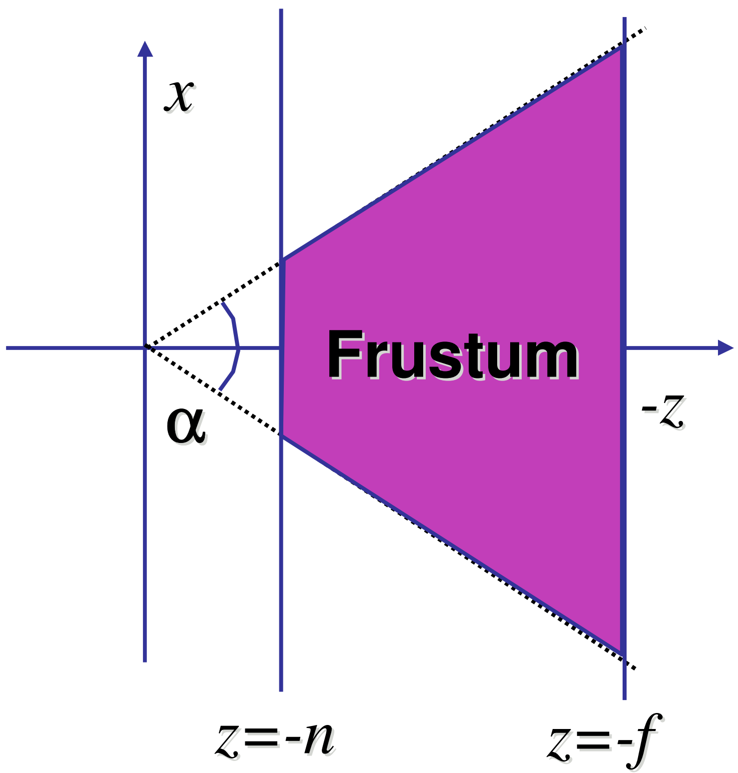
$$\begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$





Review: Field-of-View Formulation

- FOV in one direction + aspect ratio (w/h)
  - also set near, far

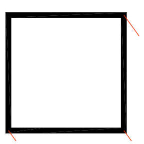


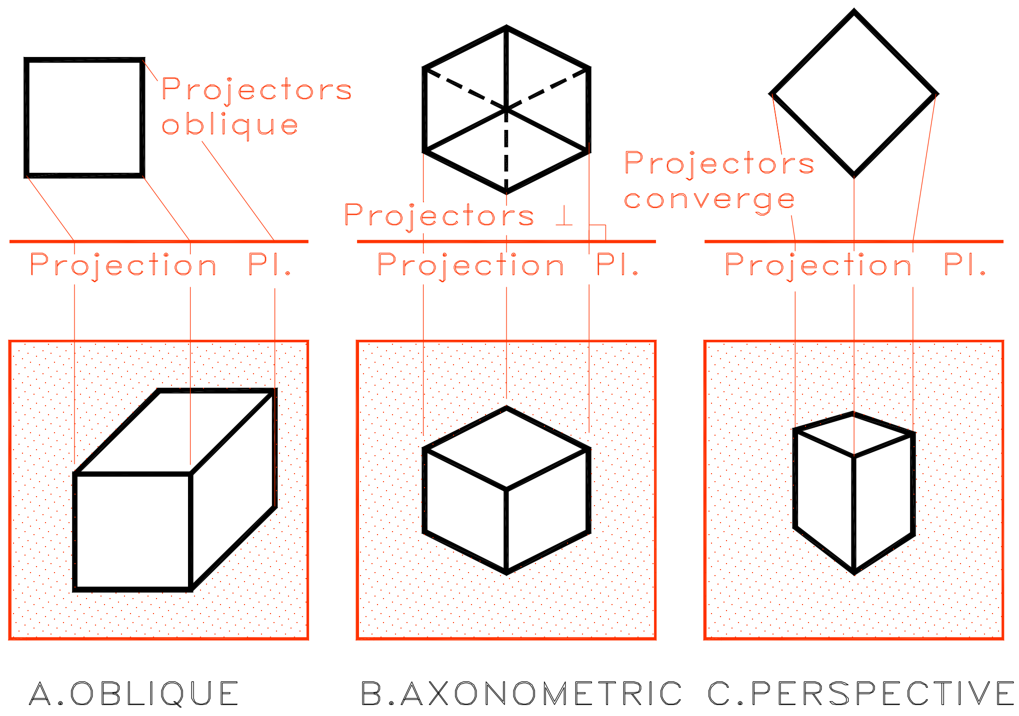
Projection Wrapup

Projection Taxonomy

planar projections

- perspective: 1,2,3-point
- parallel
  - oblique
    - cabinet
    - cavalier
  - orthographic
    - top, front, side
    - axonometric: isometric, dimetric, trimetric

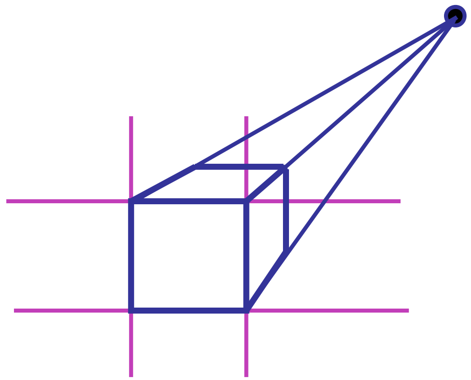




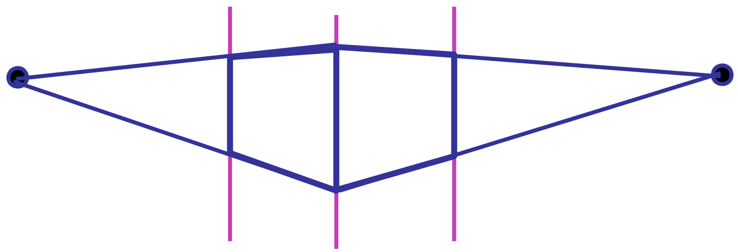
http://ceprofs.tamu.edu/tkramer/ENGR%20111/5.1/20

Perspective Projections

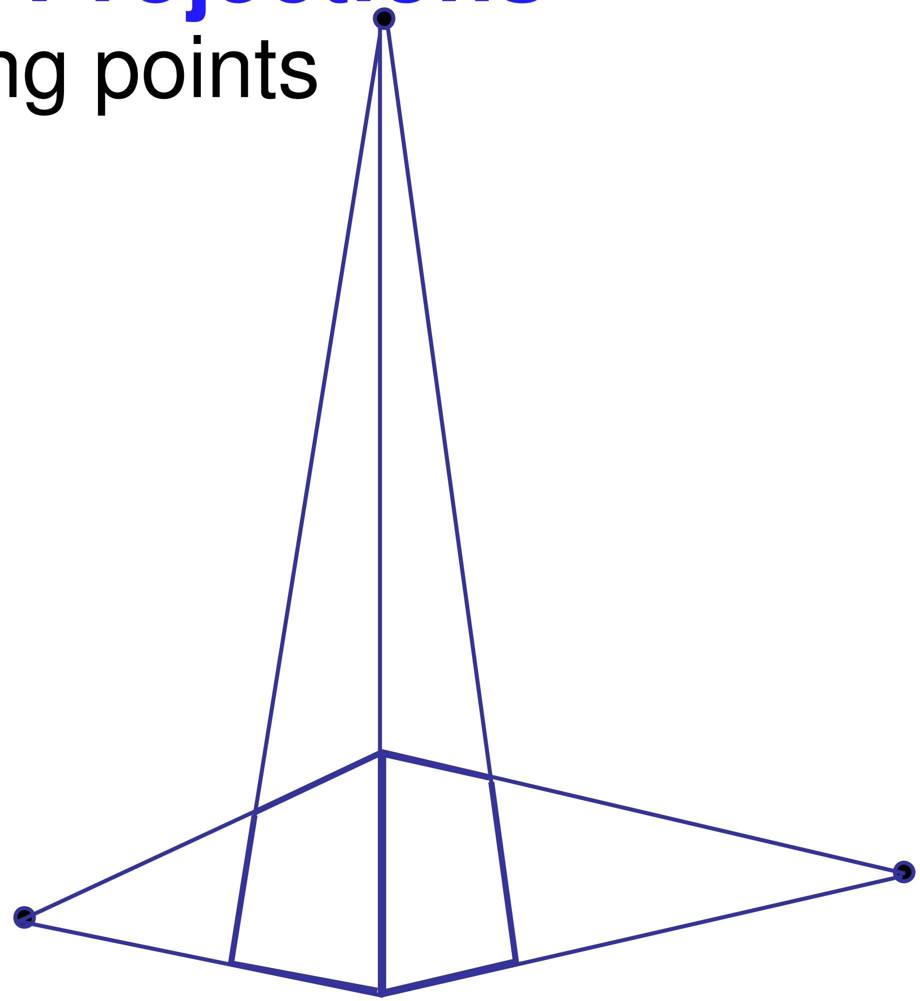
- classified by vanishing points



one-point perspective



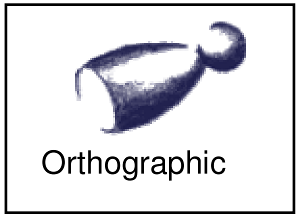
two-point perspective

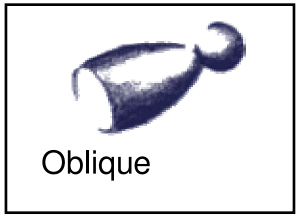


three-point perspective

Parallel Projection

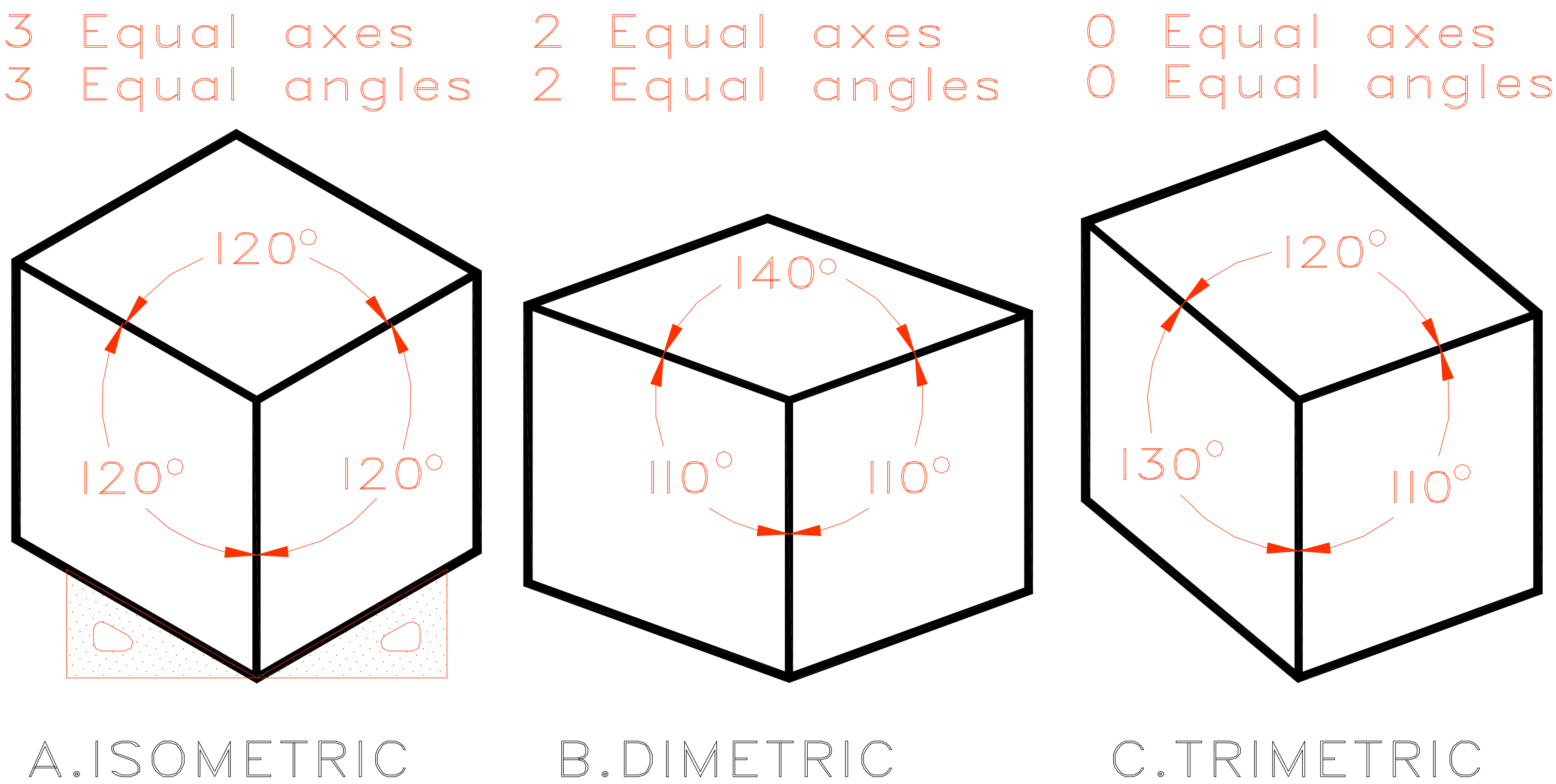
- projectors are all parallel
  - vs. perspective projectors that converge
  - orthographic: projectors perpendicular to projection plane
  - oblique: projectors not necessarily perpendicular to projection plane





Axonometric Projections

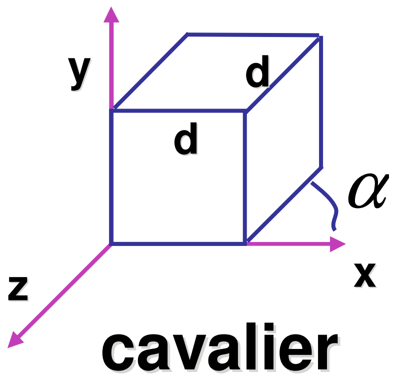
- projectors perpendicular to image plane
- select axis lengths

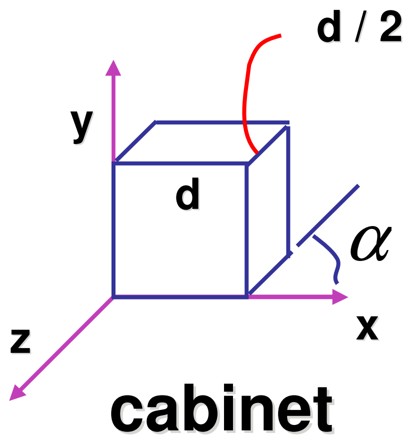


http://ceprofs.tamu.edu/tkramer/ENGR%20111/5.1/20

Oblique Projections

- projectors oblique to image plane
- select angle between front and z axis
  - lengths remain constant
- both have true front view
  - cavalier: distance true
  - cabinet: distance half





Demos

- Tuebingen applets from Frank Hanisch
  - http://www.gris.uni-tuebingen.de/projects/grdev/doc/html/etc/AppletIndex.html#Transformationen

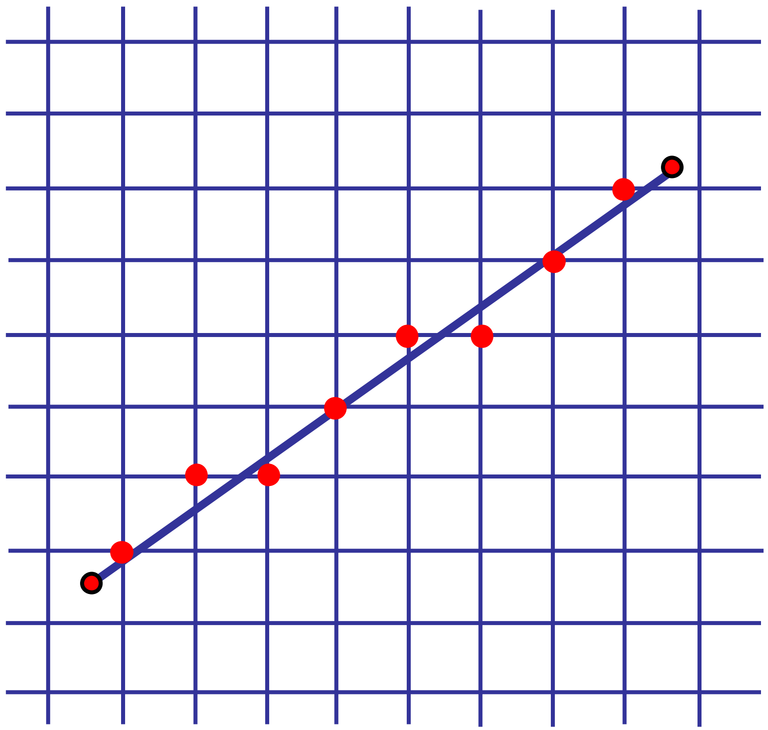
Rasterization

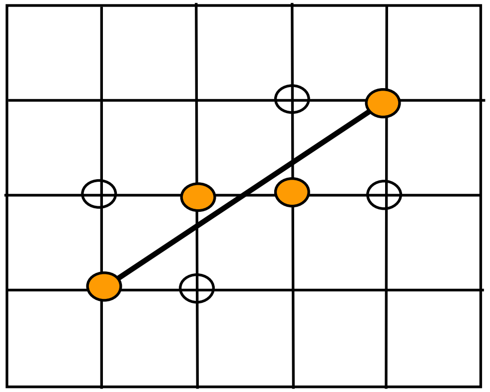
Scan Conversion - Rasterization

- convert continuous rendering primitives into discrete fragments/pixels
  - lines
    - midpoint/Bresenham
  - triangles
    - flood fill
    - scanline
    - implicit formulation
  - interpolation

Scan Conversion

- given vertices in DCS, fill in the pixels
  - start with lines





Basic Line Drawing

$$y = mx + b$$

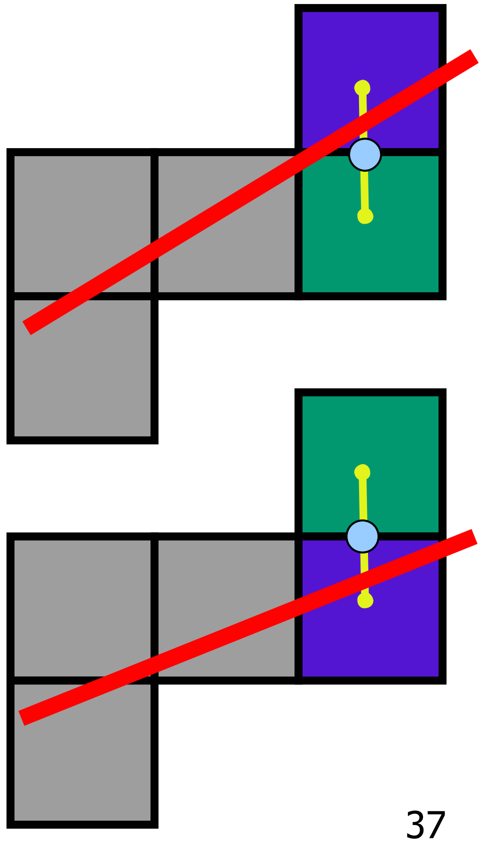
$$y = \frac{(y_1 - y_0)}{(x_1 - x_0)}(x - x_0) + y_0$$

- goals
  - integer coordinates
  - thinnest line with no gaps
- assume
  - $x_0 < x_1$, slope $0 < {dy}/{dx} < 1$
- how can we do this quickly?

Line (x_0, y_0, x_1, y_1)
begin
float $dx, dy, x, y, slope$;
$dx \Leftarrow x_1 - x_0$;
$dy \Leftarrow y_1 - y_0$;
$slope \Leftarrow {dy}/{dx}$;
$y \Leftarrow y_0$
for x from x_0 to x_1 do
begin
PlotPixel (x, **Round** (y));
$y \Leftarrow y + slope$;
end ;
end ;

Midpoint Algorithm

- moving horizontally along x direction
  - draw at current y value, or move up vertically to y+1?
    - check if midpoint between two possible pixel centers above or below line
- candidates
  - top pixel: (x+1,y+1)
  - bottom pixel: (x+1, y)
- midpoint: (x+1, y+.5)
- check if midpoint above or below line
  - below: top pixel
  - above: bottom pixel
- key idea behind Bresenham
  - [demo]



Making It Fast: Reuse Computation

- midpoint: if f(x+1, y+.5) < 0 then y = y+1
- on previous step evaluated f(x-1, y-.5) or f(x-1, y+.05)
- $f(x+1, y) = f(x,y) + (y_0 - y_1)$
- $f(x+1, y+1) = f(x,y) + (y_0 - y_1) + (x_1 - x_0)$

```
y=y0
d = f(x0+1, y0+.5)
for (x=x0; x <= x1; x++) {
   draw(x,y);
   if (d<0) then {
     y = y + 1;
     d = d + (x1 – x0) + (y0 – y1)
   } else {
     d = d + (y0 – y1)
}
```

Making It Fast: Integer Only

- midpoint: if f(x+1, y+.5) < 0 then y = y+1
- on previous step evaluated f(x-1, y-.5) or f(x-1, y+.05)
- $f(x+1, y) = f(x,y) + (y_0 - y_1)$
- $f(x+1, y+1) = f(x,y) + (y_0 - y_1) + (x_1 - x_0)$

```
y=y0
d = f(x0+1, y0+.5)
for (x=x0; x <= x1; x++) {
   draw(x,y);
   if (d<0) then {
     y = y + 1;
     d = d + (x1 – x0) + (y0 – y1)
   } else {
     d = d + (y0 – y1)
}
```

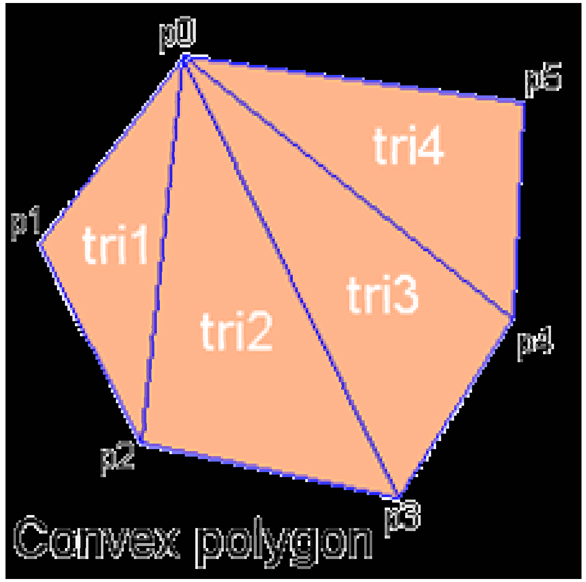
```
y=y0
2d = 2*(y0–y1) (x0+1)  + (x1–
   x0) (2y0+1)  + 2x0y1 – 2x1y0
for (x=x0; x <= x1; x++) {
   draw(x,y);
   if (d<0) then {
     y = y + 1;
     d = d + 2(x1 – x0)  + 2(y0 –y1)
   } else {
     d = d + 2(y0 – y1)
}
```

Rasterizing Polygons/Triangles

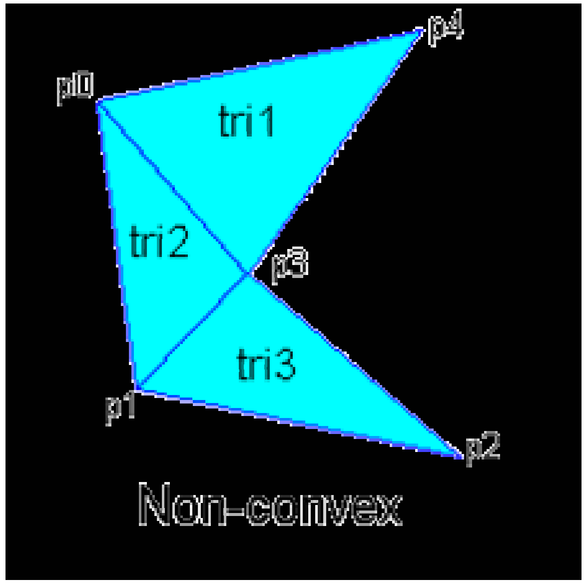
- basic surface representation in rendering
- why?
  - lowest common denominator
    - can approximate any surface with arbitrary accuracy
      - all polygons can be broken up into triangles
  - guaranteed to be:
    - planar
    - triangles - convex
  - simple to render
    - can implement in hardware

Triangulation

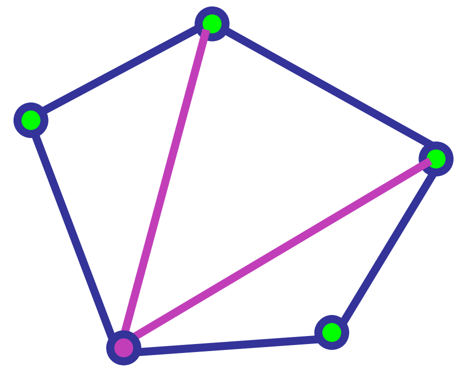
- convex polygons easily triangulated



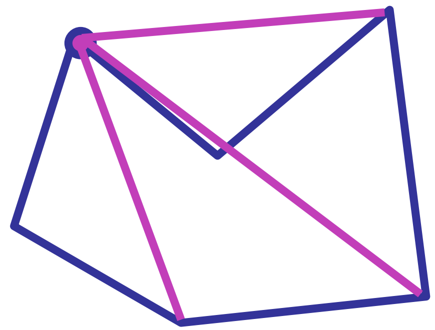
- concave polygons present a challenge



OpenGL Triangulation



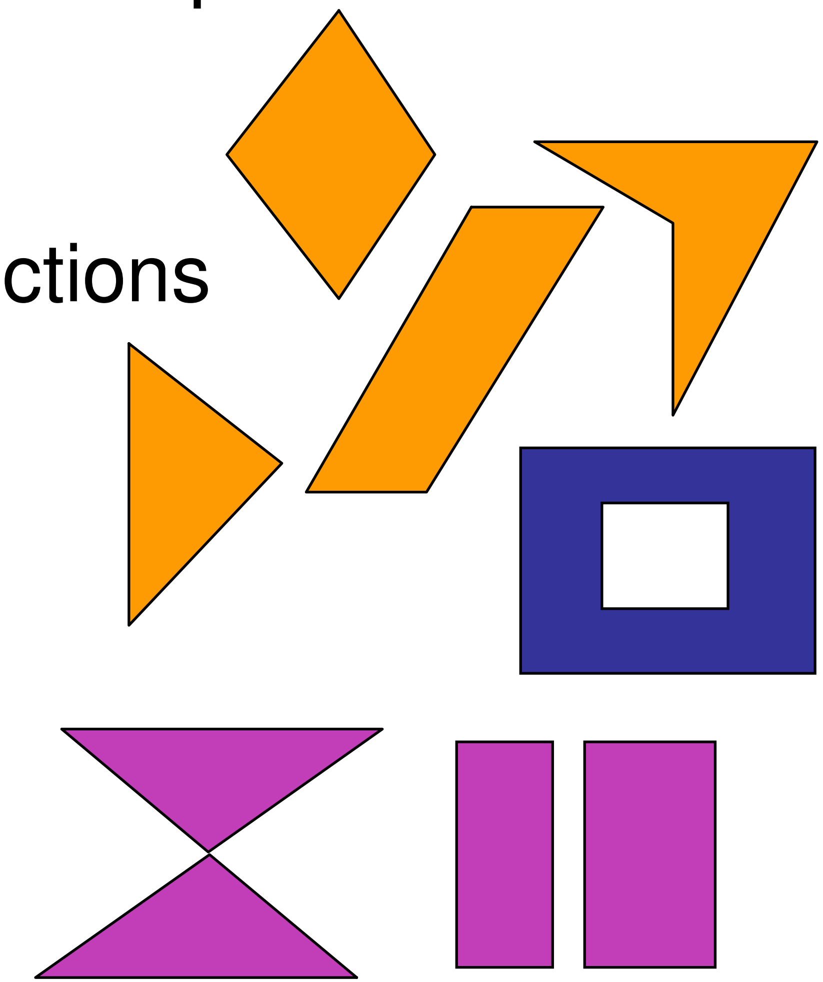
- simple convex polygons
  - break into triangles, trivial
  - glBegin(GL_POLYGON) ... glEnd()



- concave or non-simple polygons
  - break into triangles, more effort
  - gluNewTess(), gluTessCallback(), ...

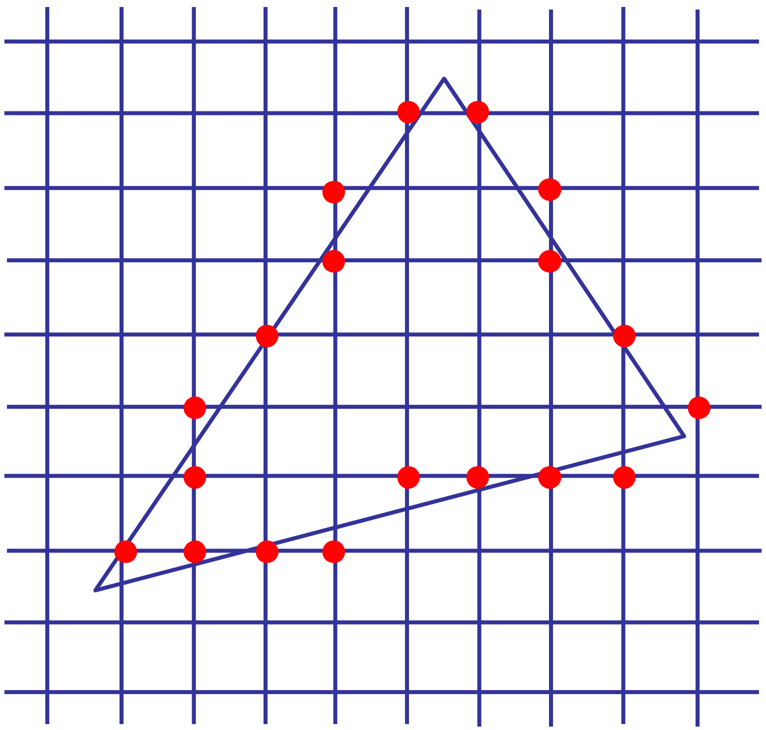
Problem

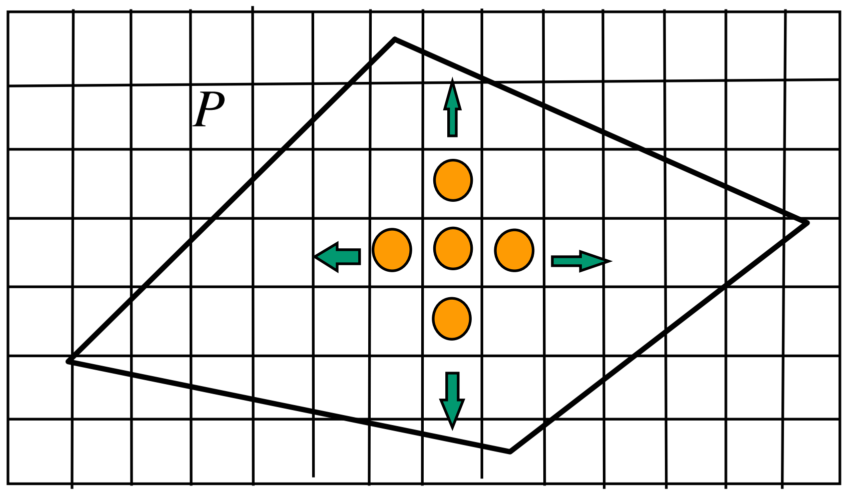
- input: closed 2D polygon
- problem: fill its interior with specified color on graphics display
- assumptions
  - simple - no self intersections
  - simply connected
- solutions
  - flood fill
  - edge walking



Flood Fill

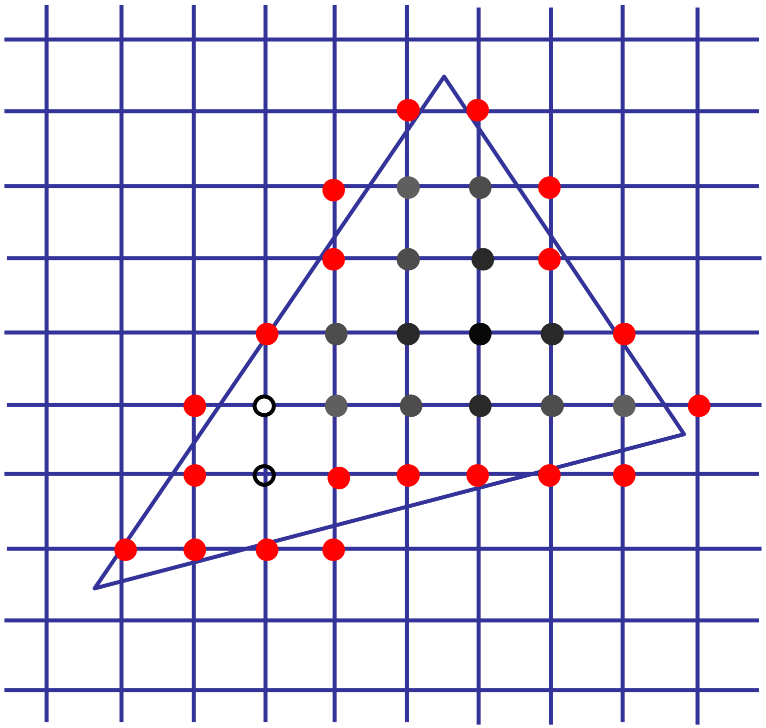
- simple algorithm
  - draw edges of polygon
  - use flood-fill to draw interior





Flood Fill

- start with seed point
  - recursively set all neighbors until boundary is hit



Flood Fill

- draw edges
- run:

FloodFill (Polygon P , int x, int y, Color C)

if not (**OnBoundary** (x, y, P) or **Colored** (x, y, C))

begin

PlotPixel (x, y, C);

FloodFill $(\mathrm{P}, x+1, y, C)$;

FloodFill $(\mathrm{P}, x, y+1, C)$;

FloodFill $(\mathrm{P}, x, y-1, C)$;

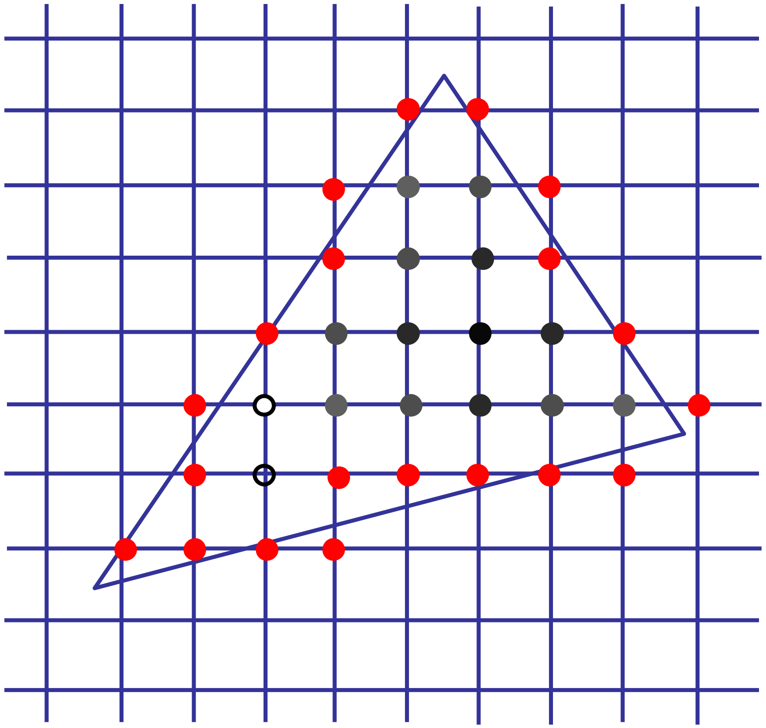
FloodFill $(\mathrm{P}, x-1, y, C)$;

end ;

- drawbacks?

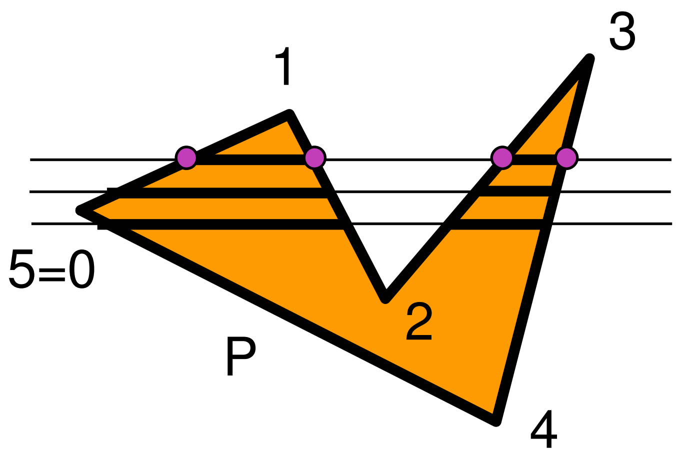
Flood Fill Drawbacks

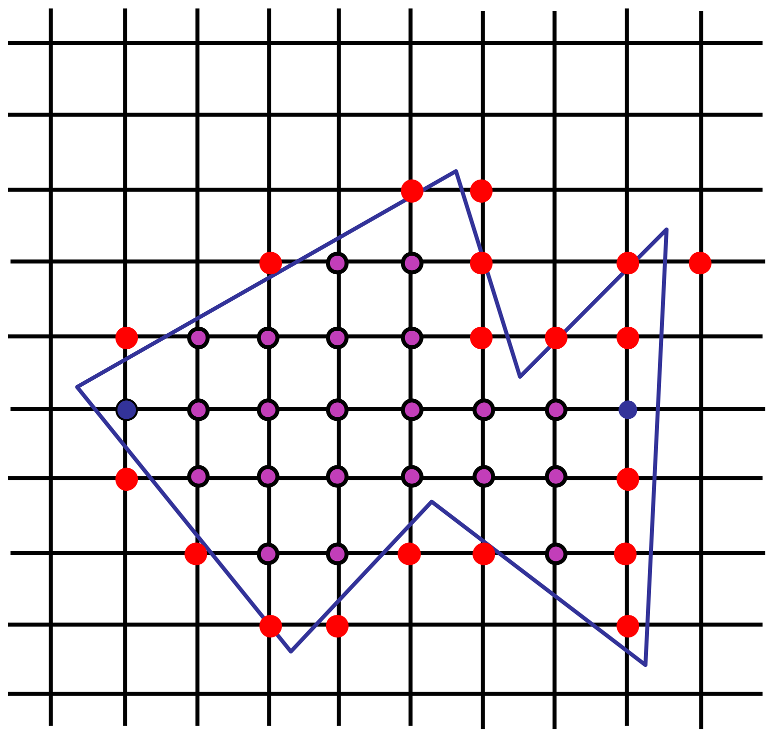
- pixels visited up to 4 times to check if already set
- need per-pixel flag indicating if set already
  - must clear for every polygon!



Scanline Algorithms

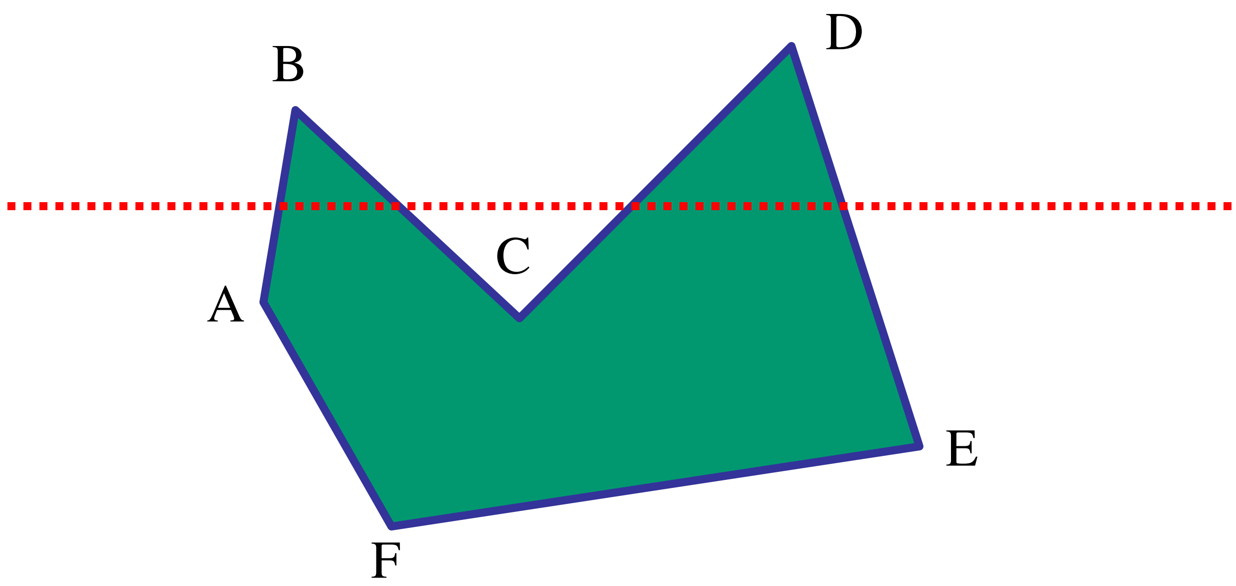
- scanline: a line of pixels in an image
  - set pixels inside polygon boundary along horizontal lines one pixel apart vertically



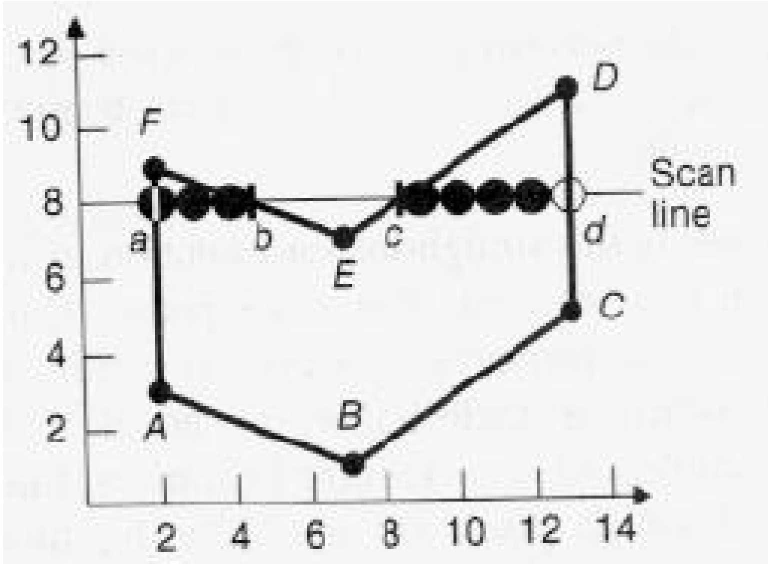


General Polygon Rasterization

- how do we know whether given pixel on scanline is inside or outside polygon?



General Polygon Rasterization

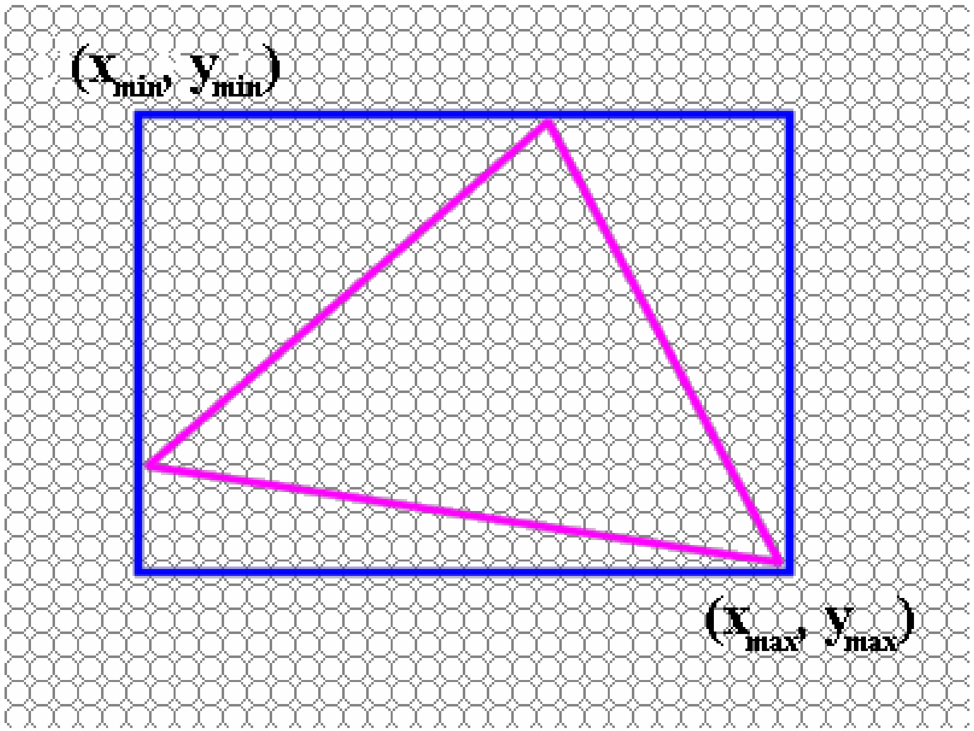


- idea: use a parity test

```
for each scanline
    edgeCnt = 0;
    for each pixel on scanline (l to r)
        if (oldpixel->newpixel crosses edge)
            edgeCnt ++;
        // draw the pixel if edgeCnt odd
        if (edgeCnt % 2)
            setPixel(pixel);
```

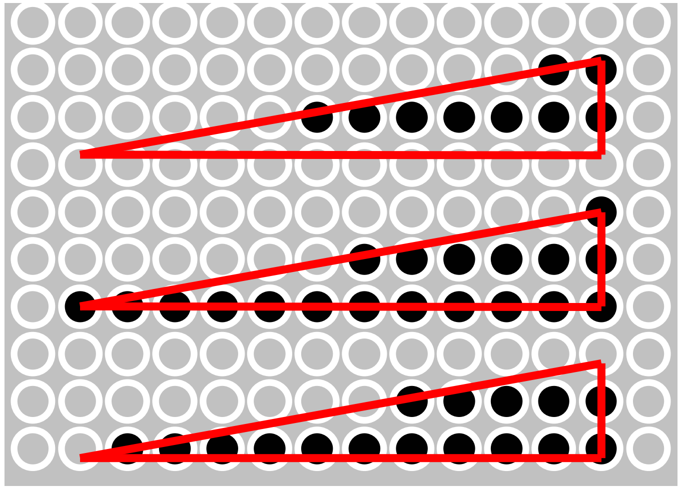
Making It Fast: Bounding Box

- smaller set of candidate pixels
  - loop over xmin, xmax and ymin,ymax instead of all x, all y

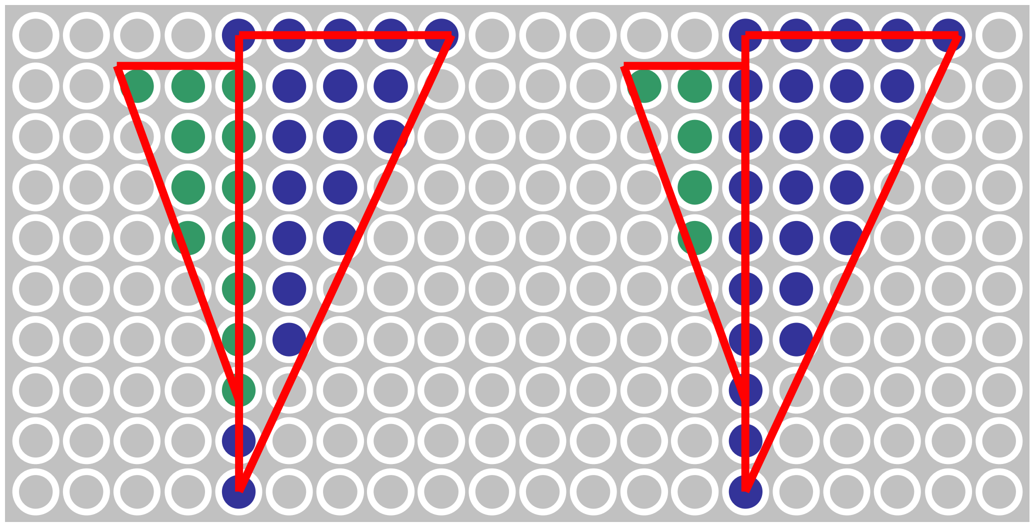


Triangle Rasterization Issues

- moving slivers



- shared edge ordering



Triangle Rasterization Issues

- *exactly which pixels should be lit?*
  - pixels with centers inside triangle edges
- *what about pixels exactly on edge?*
  - draw them: order of triangles matters (it shouldn't)
  - don't draw them: gaps possible between triangles
- need a consistent (if arbitrary) rule
  - example: draw pixels on left or top edge, but not on right or bottom edge
  - example: check if triangle on same side of edge as offscreen point

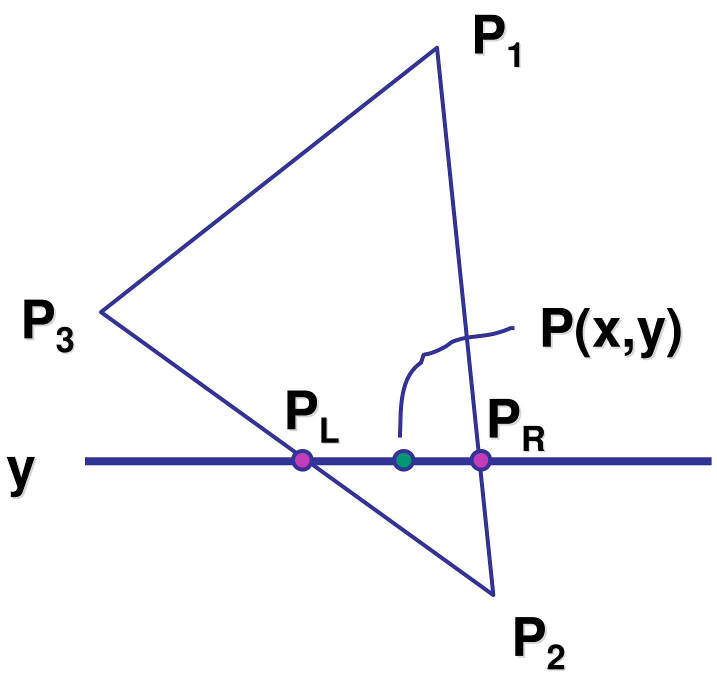
Interpolation

Interpolation During Scan Conversion

- drawing pixels in polygon requires interpolating values between vertices
  - z values
  - r,g,b colour components
    - use for Gouraud shading
  - u,v texture coordinates
  - N_x, N_y, N_z surface normals
- equivalent methods (for triangles)
  - bilinear interpolation
  - barycentric coordinates

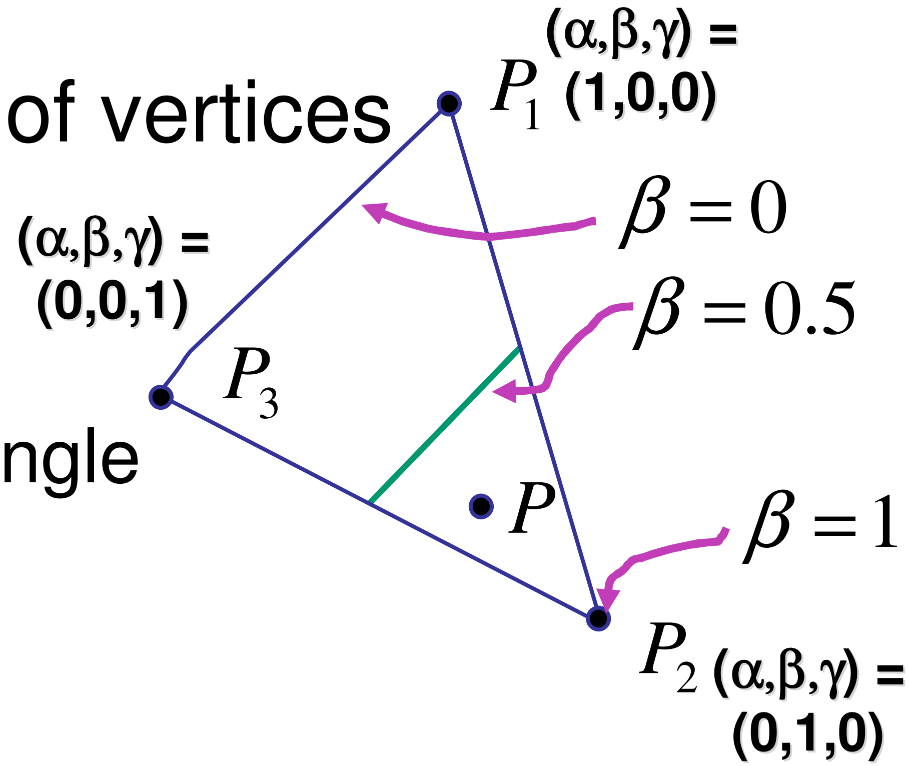
Bilinear Interpolation

- interpolate quantity along L and R edges, as a function of y
  - then interpolate quantity as a function of x



Barycentric Coordinates

- weighted combination of vertices
  - smooth mixing
  - speedup
    - compute once per triangle



$$\begin{cases} P = \alpha \cdot P_1 + \beta \cdot P_2 + \gamma \cdot P_3 \\ \alpha + \beta + \gamma = 1 \\ 0 \le \alpha, \beta, \gamma \le 1 \text{ for points inside triangle} \end{cases}$$

"convex combination of points"

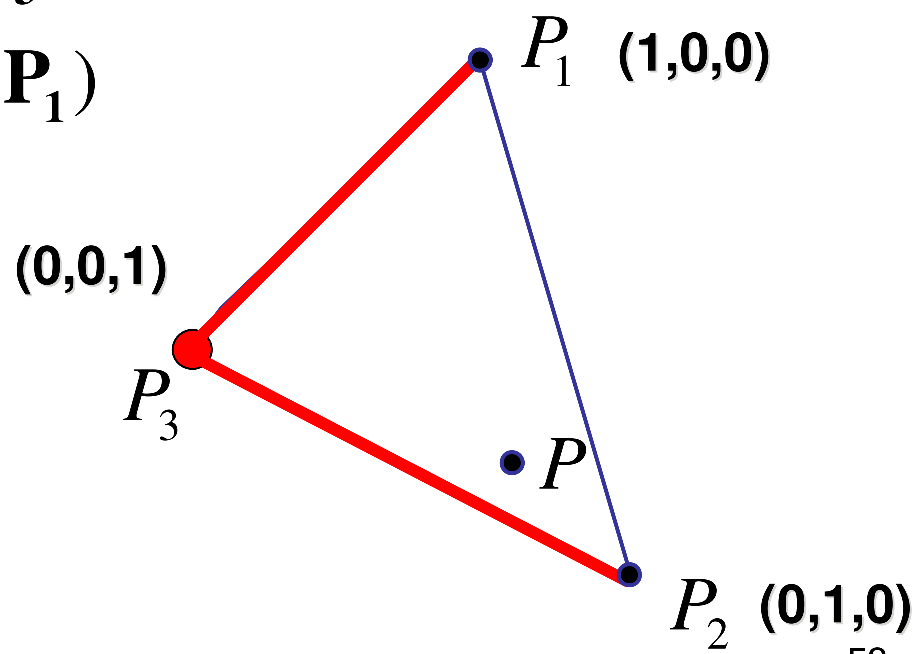
Deriving Barycentric Coordinates I

- non-orthogonal coordinate system
  - P_3 is origin
  - P_2-P_3, P_1-P_3 are basis vectors

$$\mathbf{P} = \mathbf{P_3} + \beta(\mathbf{P_2} - \mathbf{P_3}) + \gamma(\mathbf{P_1} - \mathbf{P_3})$$

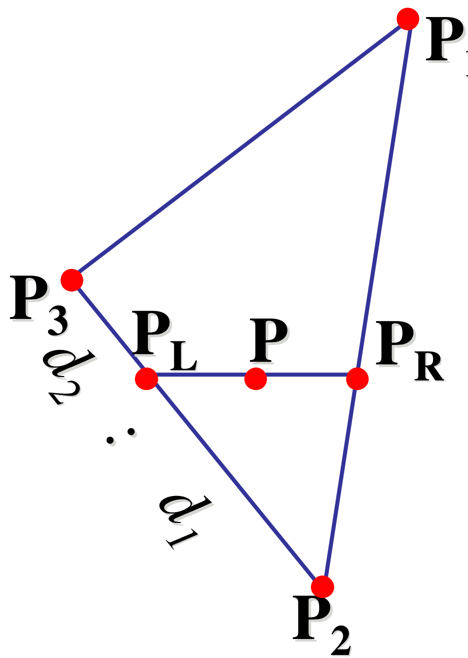
$$\mathbf{P} = (1 - \beta - \gamma)\mathbf{P_3} + \beta(\mathbf{P_2}) + \gamma(\mathbf{P_1})$$

$$\mathbf{P} = \alpha(\mathbf{P_3}) + \beta(\mathbf{P_2}) + \gamma(\mathbf{P_1})$$



Deriving Barycentric Coordinates II

- from bilinear interpolation of point P on scanline



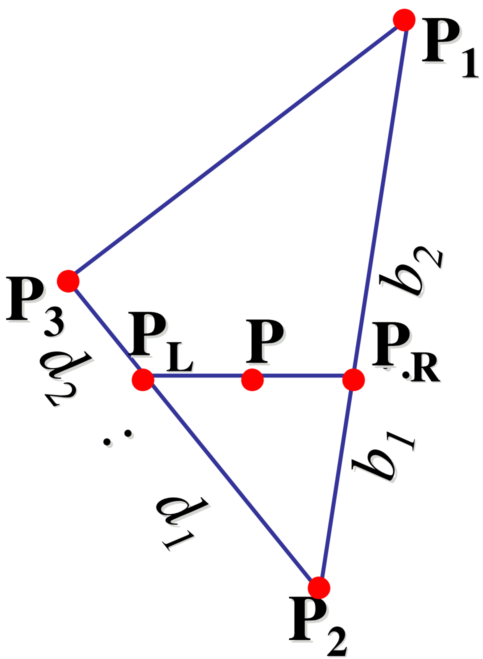
$$P_L = P_2 + \frac{d_1}{d_1 + d_2}(P_3 - P_2)$$

$$= (1 - \frac{d_1}{d_1 + d_2})P_2 + \frac{d_1}{d_1 + d_2}P_3 =$$

$$= \frac{d_2}{d_1 + d_2}P_2 + \frac{d_1}{d_1 + d_2}P_3$$

Deriving Barycentric Coordinates II

- similarly



$$P_R = P_2 + \frac{b_1}{b_1 + b_2}(P_1 - P_2)$$

$$= (1 - \frac{b_1}{b_1 + b_2})P_2 + \frac{b_1}{b_1 + b_2}P_1 =$$

$$= \frac{b_2}{b_1 + b_2}P_2 + \frac{b_1}{b_1 + b_2}P_1$$

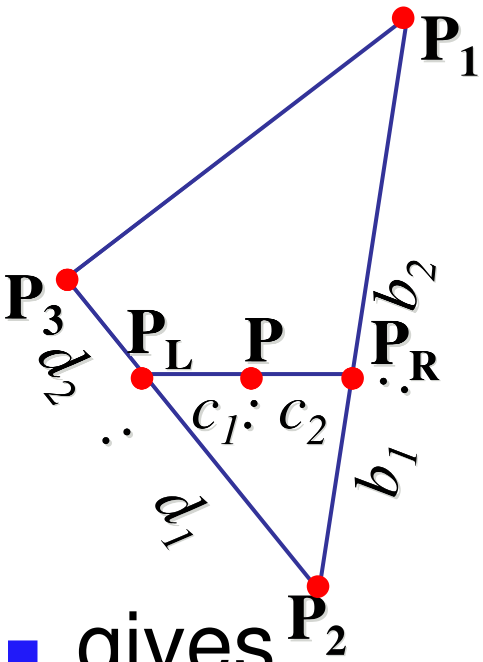
Deriving Barycentric Coordinates II

- combining

$$P = \frac{c_2}{c_1 + c_2} \cdot P_L + \frac{c_1}{c_1 + c_2} \cdot P_R$$

$$P_L = \frac{d_2}{d_1 + d_2} P_2 + \frac{d_1}{d_1 + d_2} P_3$$

$$P_R = \frac{b_2}{b_1 + b_2} P_2 + \frac{b_1}{b_1 + b_2} P_1$$



- gives

$$P = \frac{c_2}{c_1 + c_2}\left(\frac{d_2}{d_1 + d_2} P_2 + \frac{d_1}{d_1 + d_2} P_3\right) + \frac{c_1}{c_1 + c_2}\left(\frac{b_2}{b_1 + b_2} P_2 + \frac{b_1}{b_1 + b_2} P_1\right)$$

Deriving Barycentric Coordinates II

- thus $P = a_1 \cdot P_1 + a_2 \cdot P_2 + a_3 \cdot P_3$ with

$$\alpha = \frac{c_1}{c_1 + c_2} \frac{b_1}{b_1 + b_2}$$

$$\beta = \frac{c_2}{c_1 + c_2} \frac{d_2}{d_1 + d_2} + \frac{c_1}{c_1 + c_2} \frac{b_2}{b_1 + b_2}$$

$$\gamma = \frac{c_2}{c_1 + c_2} \frac{d_1}{d_1 + d_2}$$

- can verify barycentric properties

$$\alpha + \beta + \gamma = 1, \qquad 0 \leq \alpha, \beta, \gamma \leq 1$$

Deriving Barycentric Coordinates III

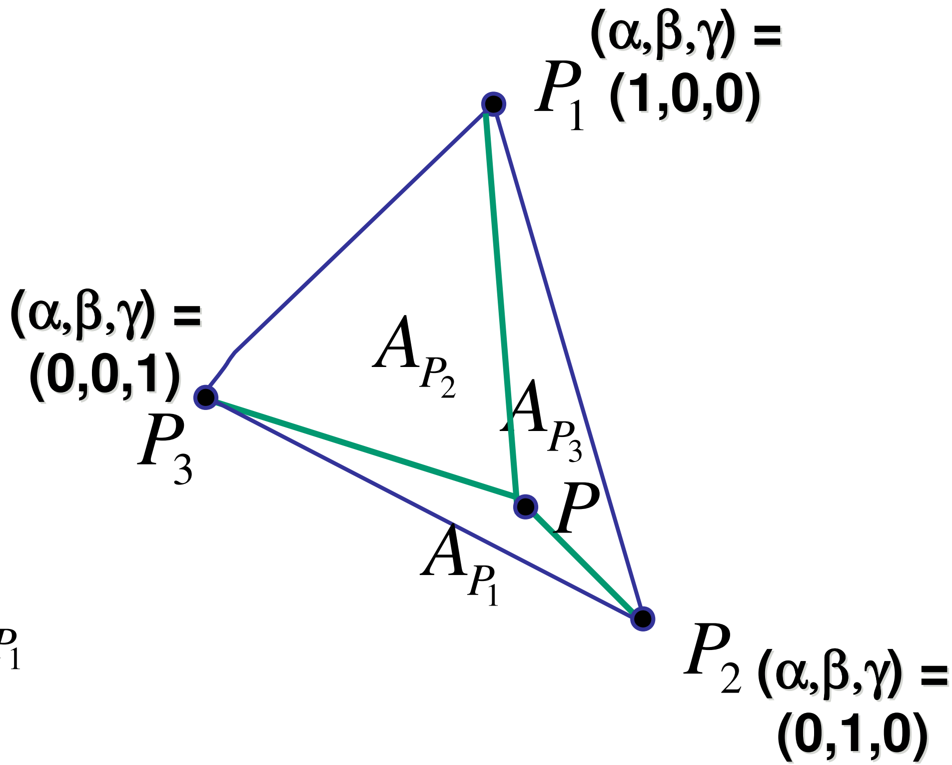
- 2D triangle area

$$\alpha = A_{P_3} / A$$

$$\beta = A_{P_2} / A$$

$$\gamma = A_{P_1} / A$$

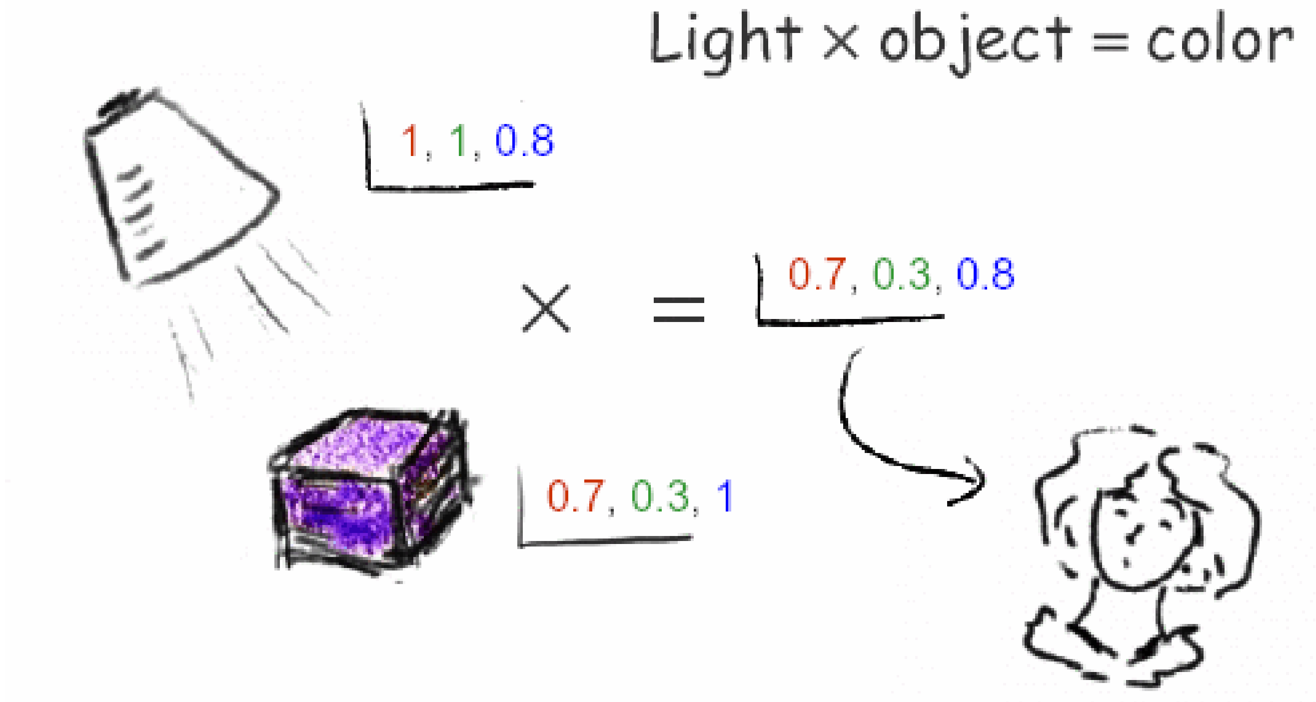
$$A = +A_{P_3} + A_{P_2} + A_{P_1}$$



Vision/Color

Simple Model of Color

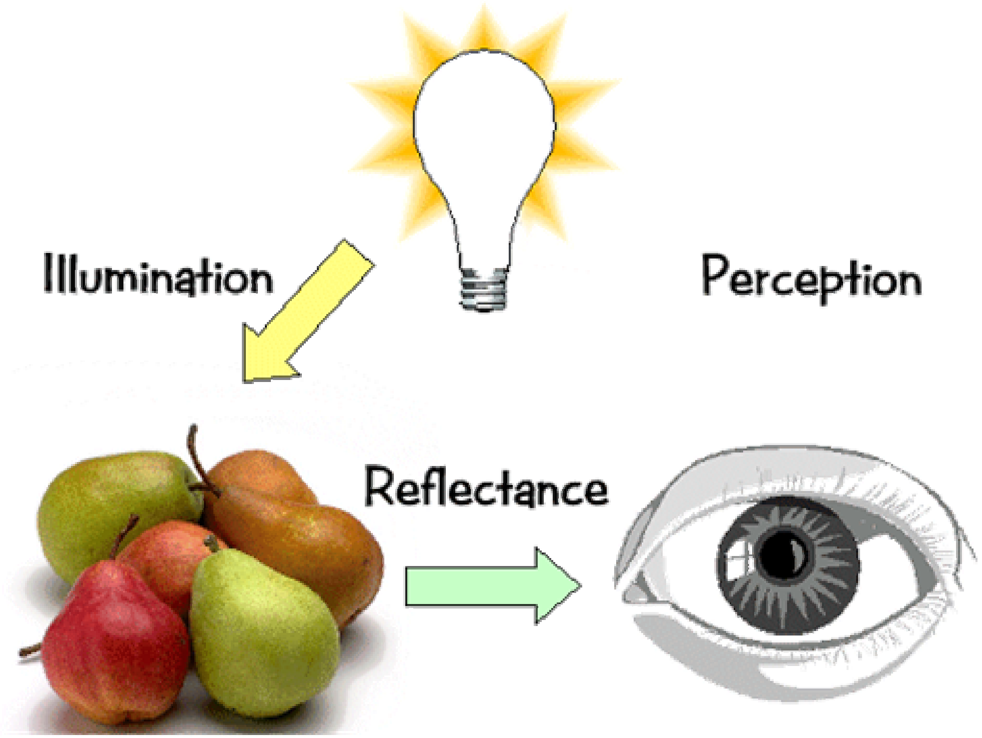
- simple model based on RGB triples
- component-wise multiplication of colors
  - (a0,a1,a2) * (b0,b1,b2) = (a0*b0, a1*b1, a2*b2)



  - why does this work?

Basics Of Color

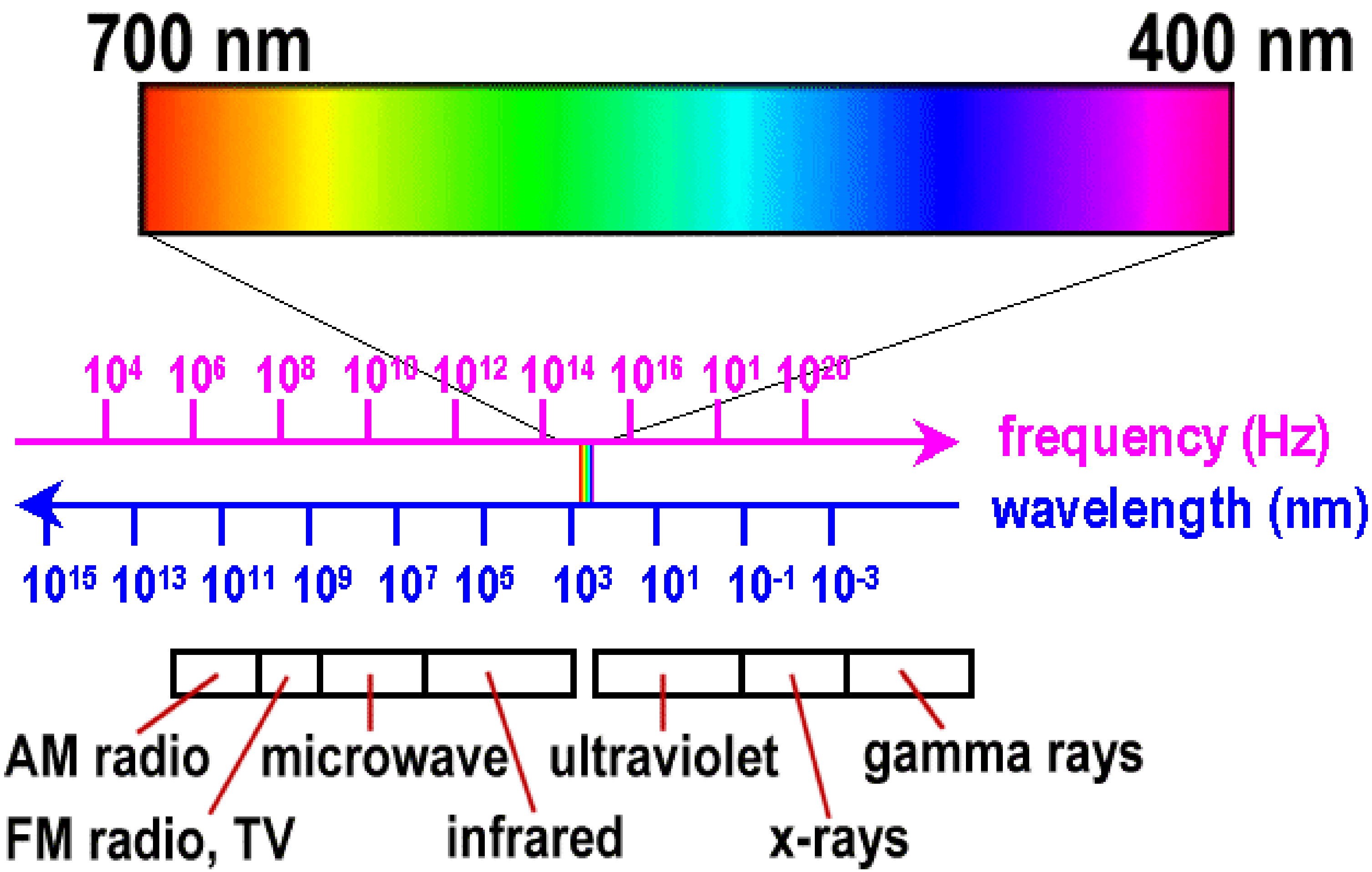
- elements of color:



Basics of Color

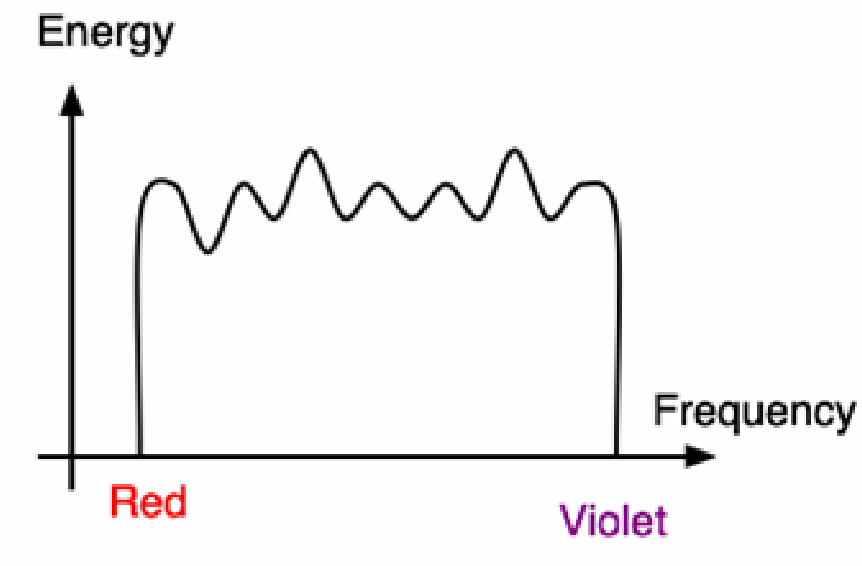
- physics
  - illumination
    - electromagnetic spectra
  - reflection
    - material properties
    - surface geometry and microgeometry (i.e., polished versus matte versus brushed)
- perception
  - physiology and neurophysiology
  - perceptual psychology

Electromagnetic Spectrum

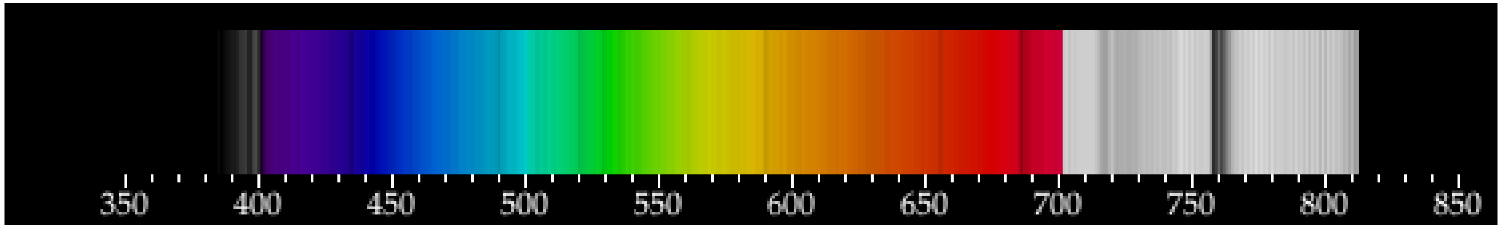


White Light

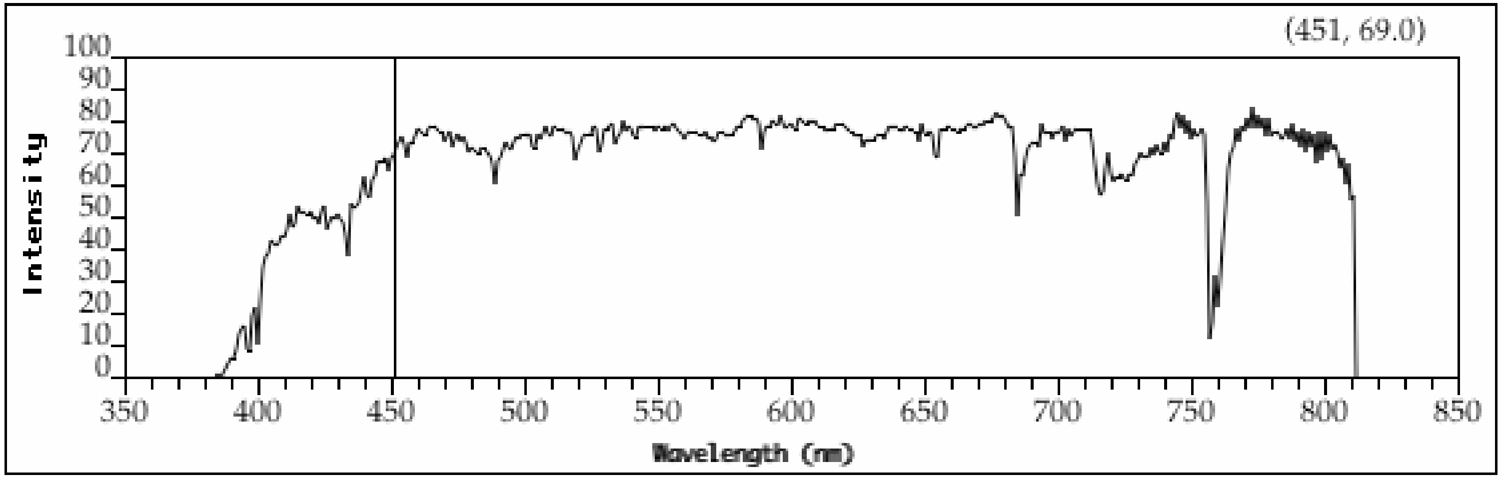
- sun or light bulbs emit all frequencies within the visible range to produce what we perceive as the "white light"



Sunlight Spectrum



Emission Graph



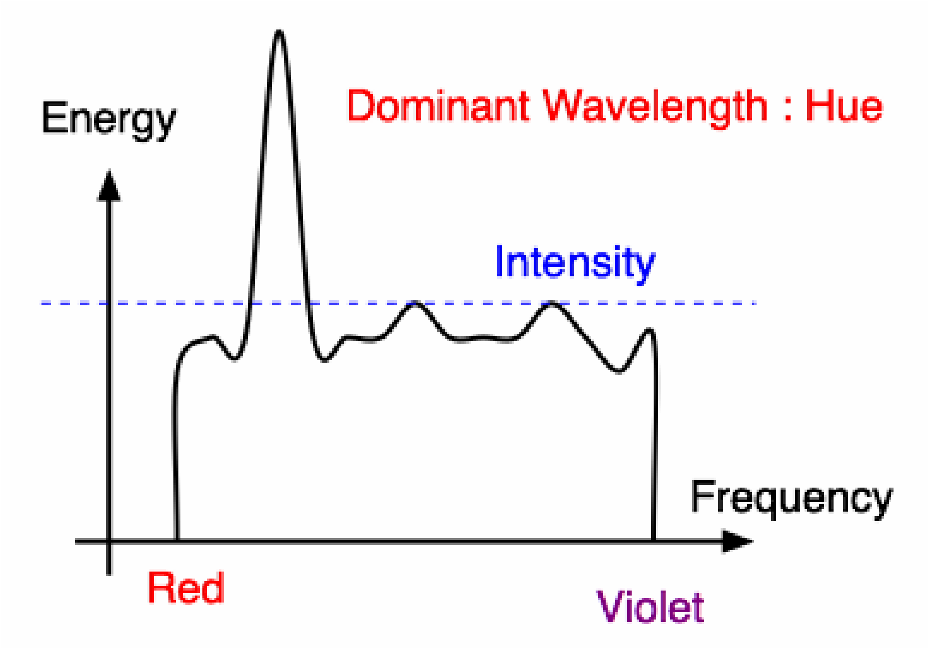
Electromagnetic Spectrum

White Light and Color

- when white light is incident upon an object, some frequencies are reflected and some are absorbed by the object
- combination of frequencies present in the reflected light that determinses what we perceive as the color of the object

Hue

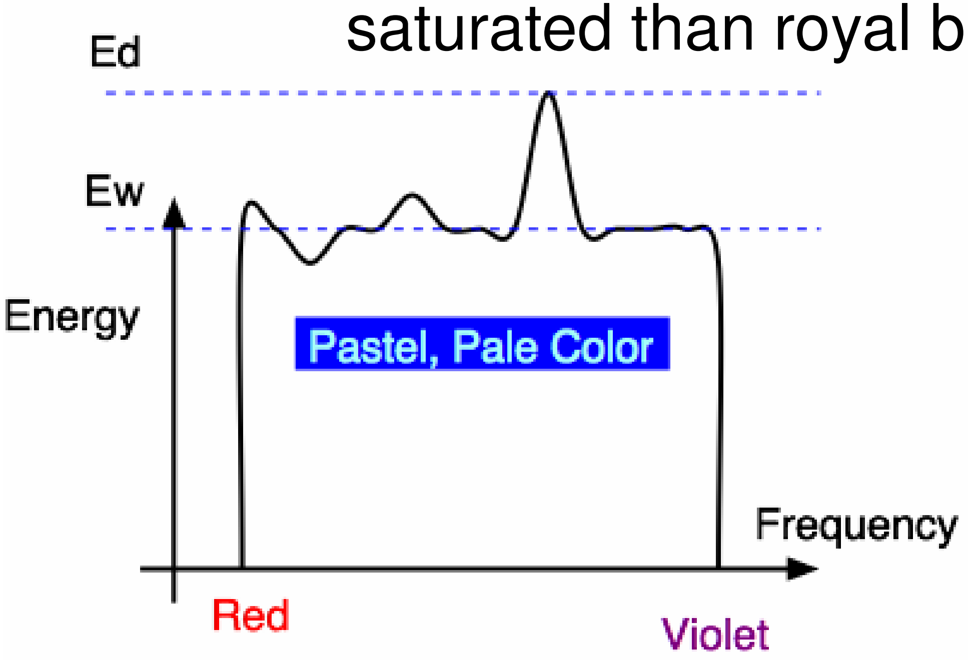
- hue (or simply, "color") is dominant wavelength/frequency

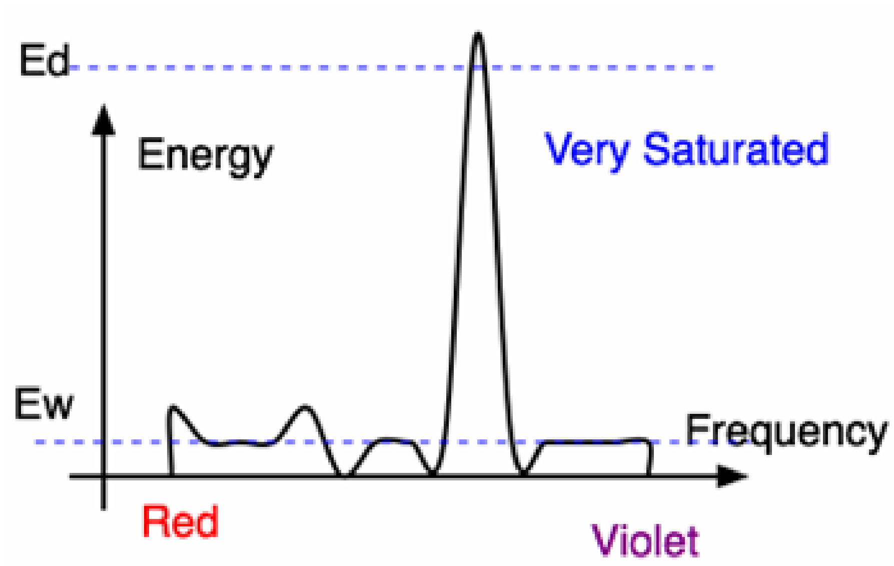


  - integration of energy for all visible wavelengths is proportional to intensity of color

Saturation or Purity of Light

- how washed out or how pure the color of the light appears
  - contribution of dominant light vs. other frequencies producing white light
  - saturation: how far is color from grey
    - pink is less saturated than red, sky blue is less saturated than royal blue





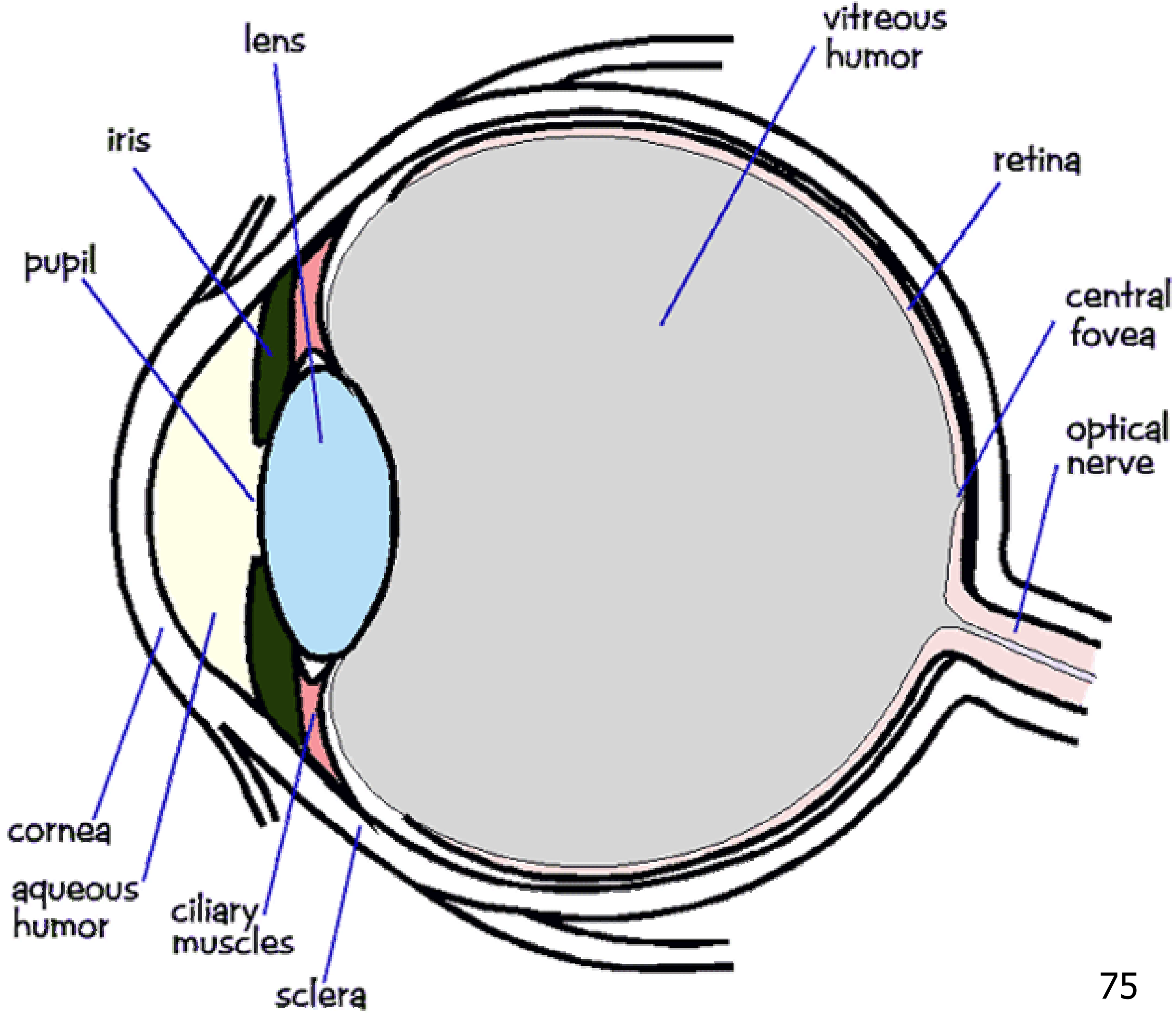
Intensity vs. Brightness

- intensity : measured radiant energy emitted per unit of time, per unit solid angle, and per unit projected area of the source (related to the luminance of the source)

- lightness/brightness : perceived intensity of light
  - nonlinear

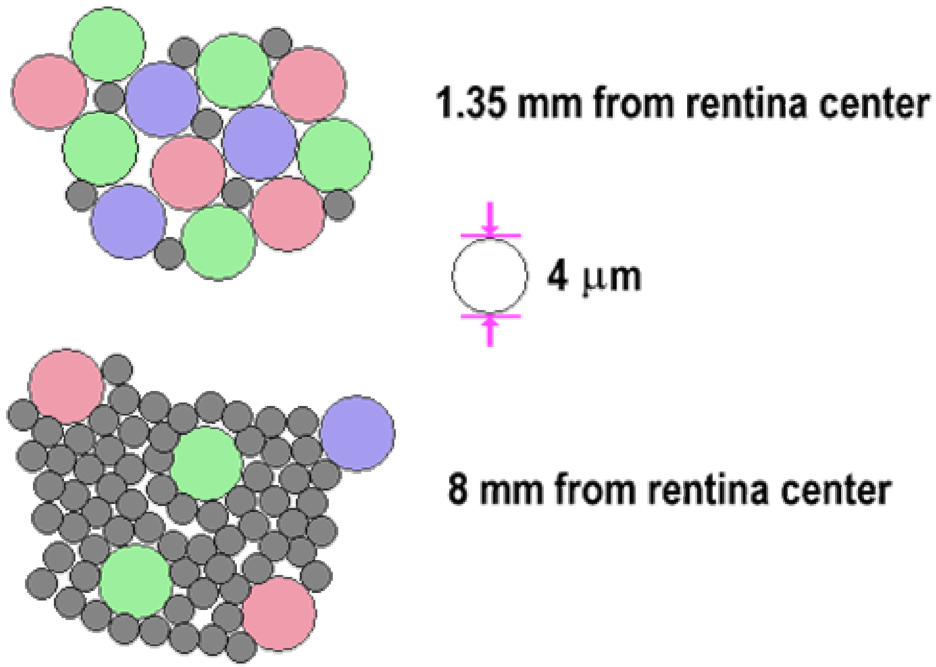
Physiology of Vision

- the retina
  - rods
    - b/w, edges
  - cones
    - color!



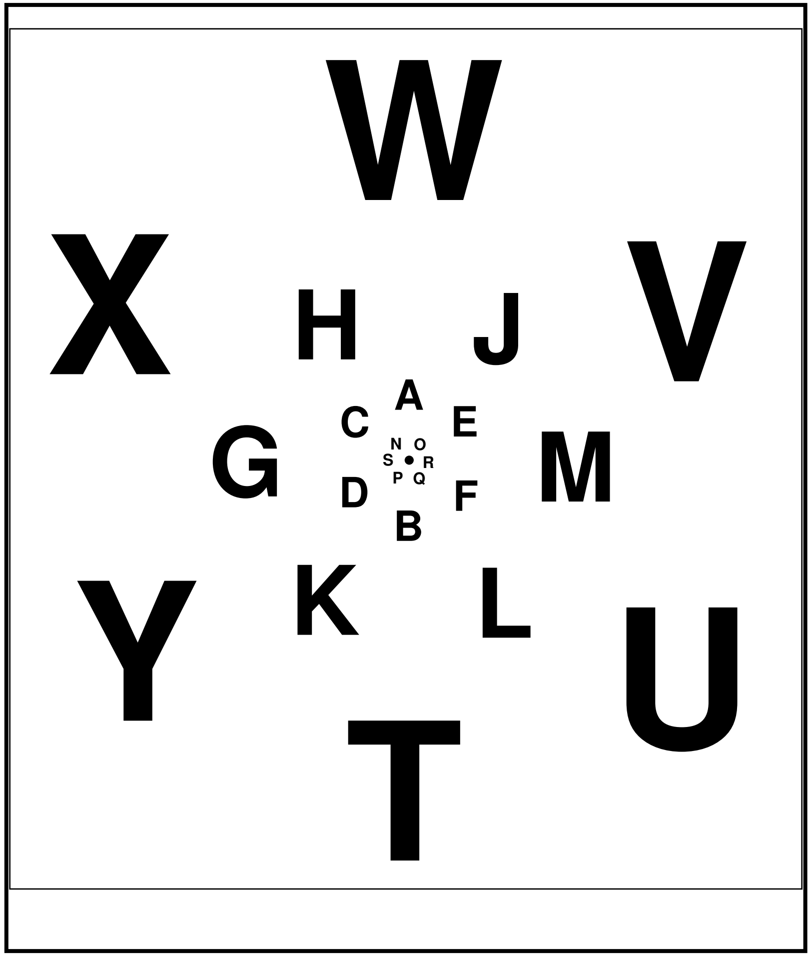
Physiology of Vision

- center of retina is densely packed region called the *fovea*.
  - cones much denser here than the *periphery*



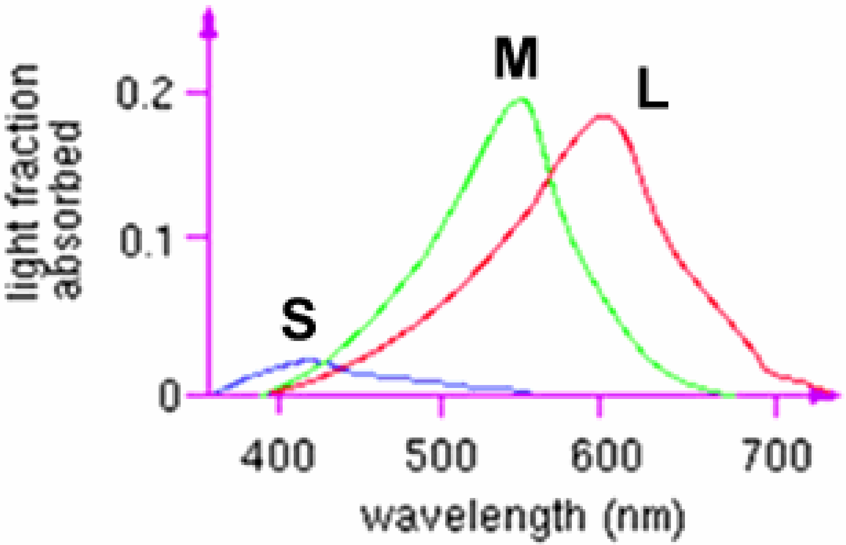
Foveal Vision

- hold out your thumb at arm's length



Trichromacy

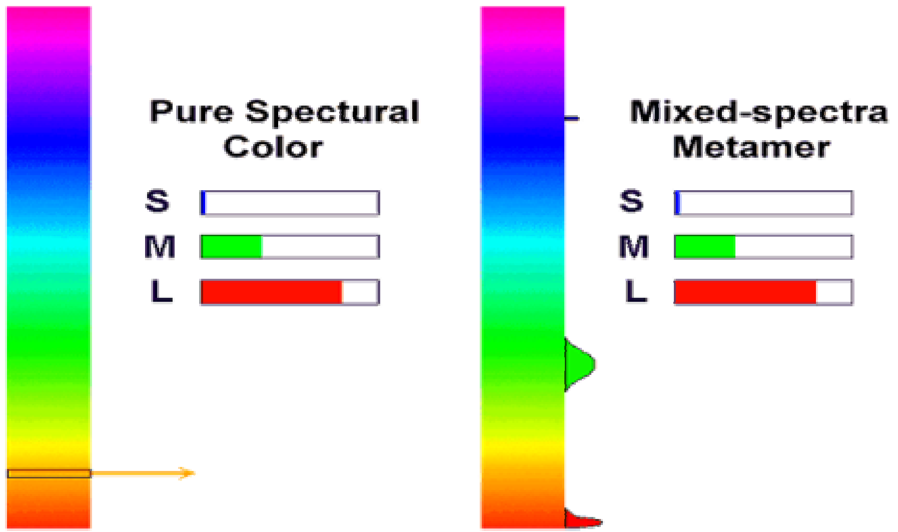
- three types of cones
  - L or R, most sensitive to red light (610 nm)
  - M or G, most sensitive to green light (560 nm)
  - S or B, most sensitive to blue light (430 nm)



  - color blindness results from missing cone type(s)

Metamers

- a given perceptual sensation of color derives from the stimulus of all three cone types



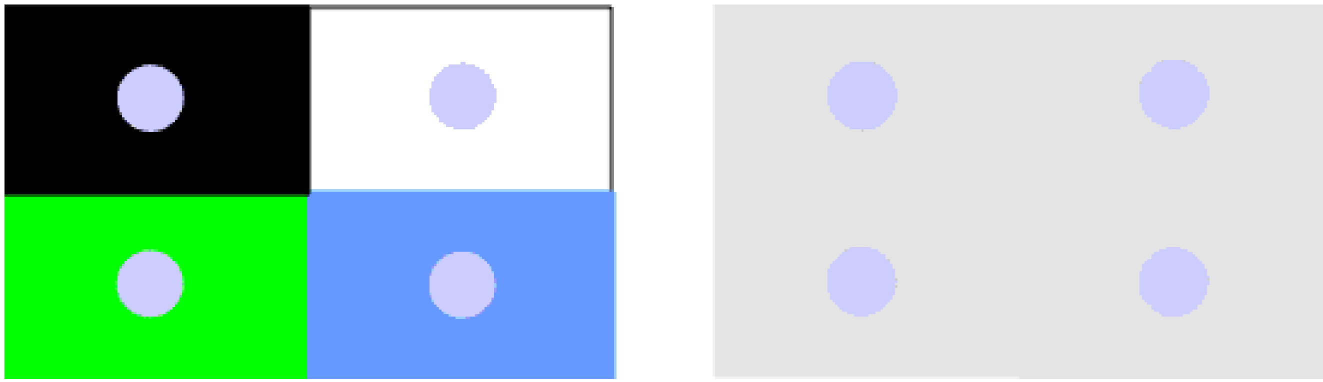
- identical perceptions of color can thus be caused by very different spectra

Metamer Demo

- http://www.cs.brown.edu/exploratories/freeSoftware/catalogs/color_theory.html

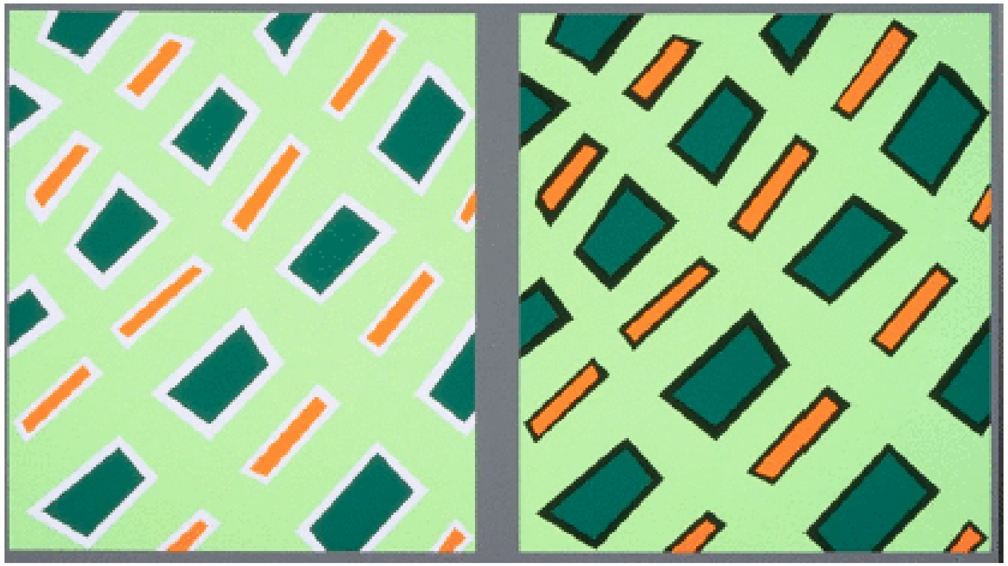
Adaptation, Surrounding Color

- color perception is also affected by
  - adaptation (move from sunlight to dark room)
  - surrounding color/intensity:
    - simultaneous contrast effect

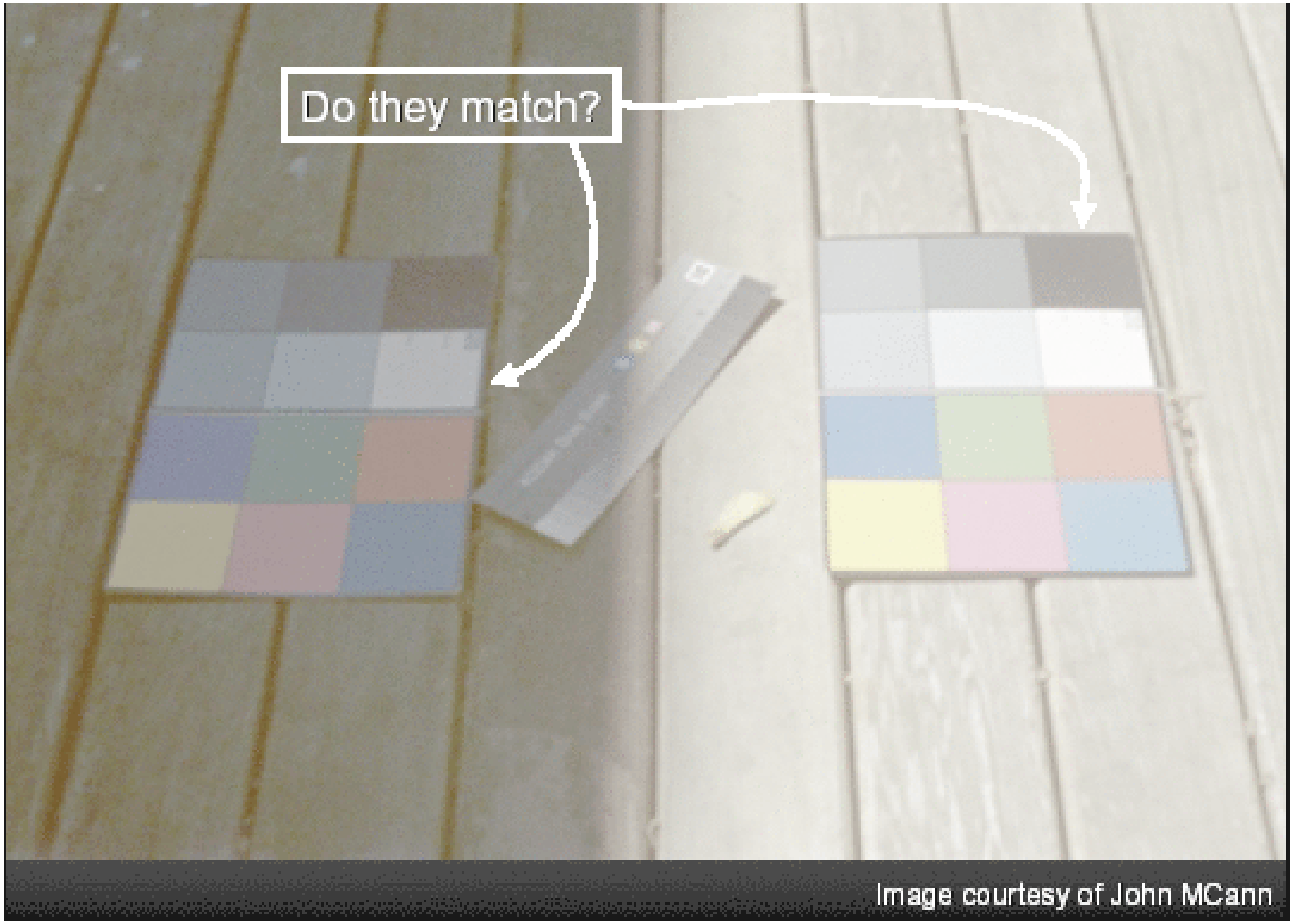


Bezold Effect

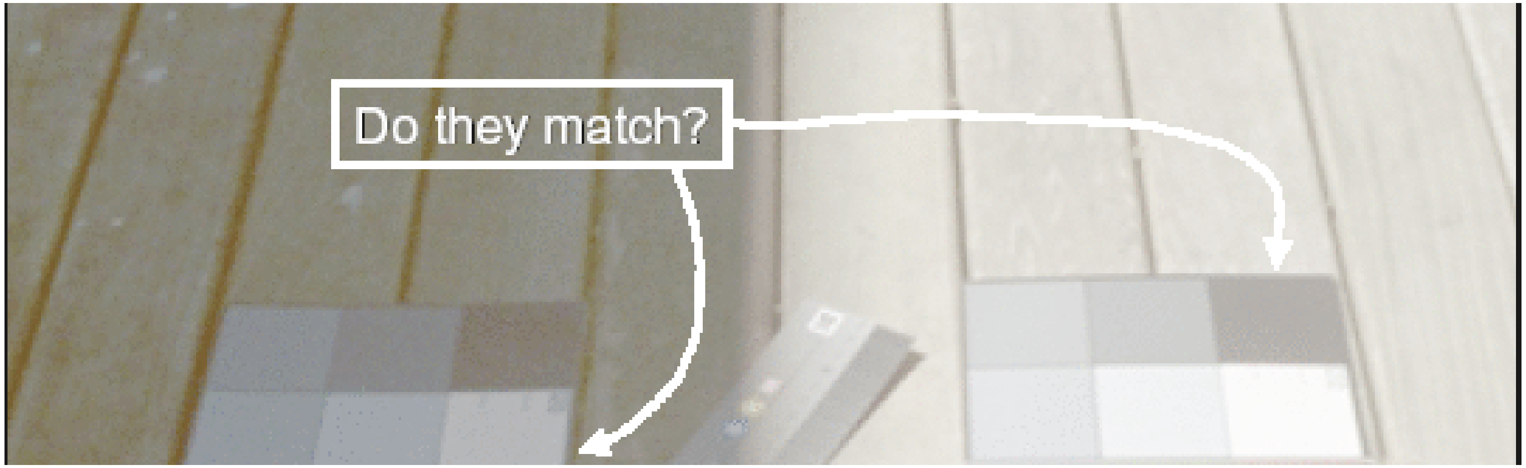
- impact of outlines



Color/Lightness Constancy

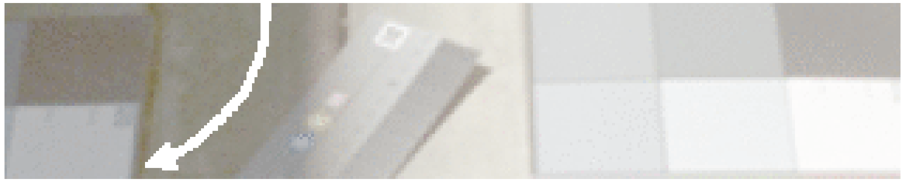


Color/Lightness Constancy

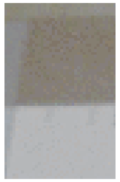


Color/Lightness Constancy

Color/Lightness Constancy



Color/Lightness Constancy



Color/Lightness Constancy

Color Constancy

- automatic “white balance” from change in illumination
- vast amount of processing behind the scenes!
- colorimetry vs. perception

From Color Appearance Models, fig 8-1

Stroop Effect

- red
- blue
- orange
- purple
- green

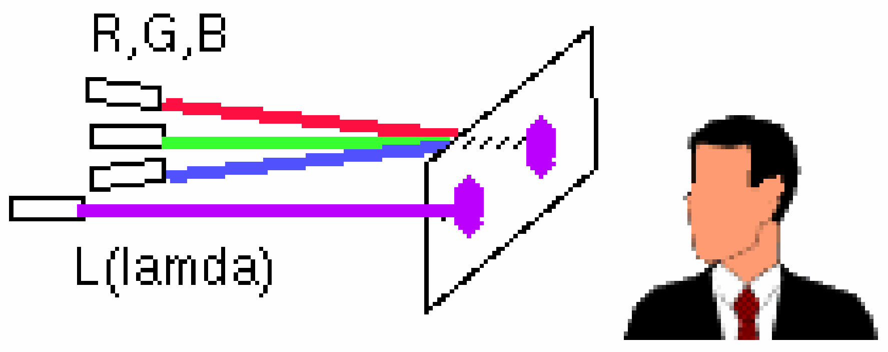
Stroop Effect

- blue
- green
- purple
- red
- orange

- interplay between cognition and perception

Color Spaces

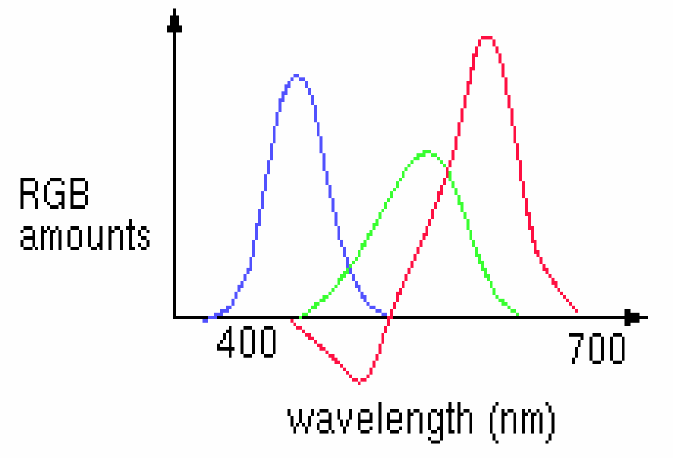
- three types of cones suggests color is a 3D quantity. how to define 3D color space?



- idea: perceptually based measurement
  - shine given wavelength (λ) on a screen
  - user must control three pure lights producing three other wavelengths (say R=700nm, G=546nm, and B=436nm)
  - adjust intensity of RGB until colors are identical
    - this works because of metamers!

Negative Lobes

- exact target match with phosphors not possible



  - some red had to be added to target color to permit exact match using “knobs” on RGB intensity output of CRT
  - equivalently theoretically to removing red from CRT output
  - figure shows that red phosphor must remove some cyan for perfect match
  - CRT phosphors cannot remove cyan, so 500 nm cannot be generated

Negative Lobes

- can’t generate all other wavelenths with any set of three positive monochromatic lights!

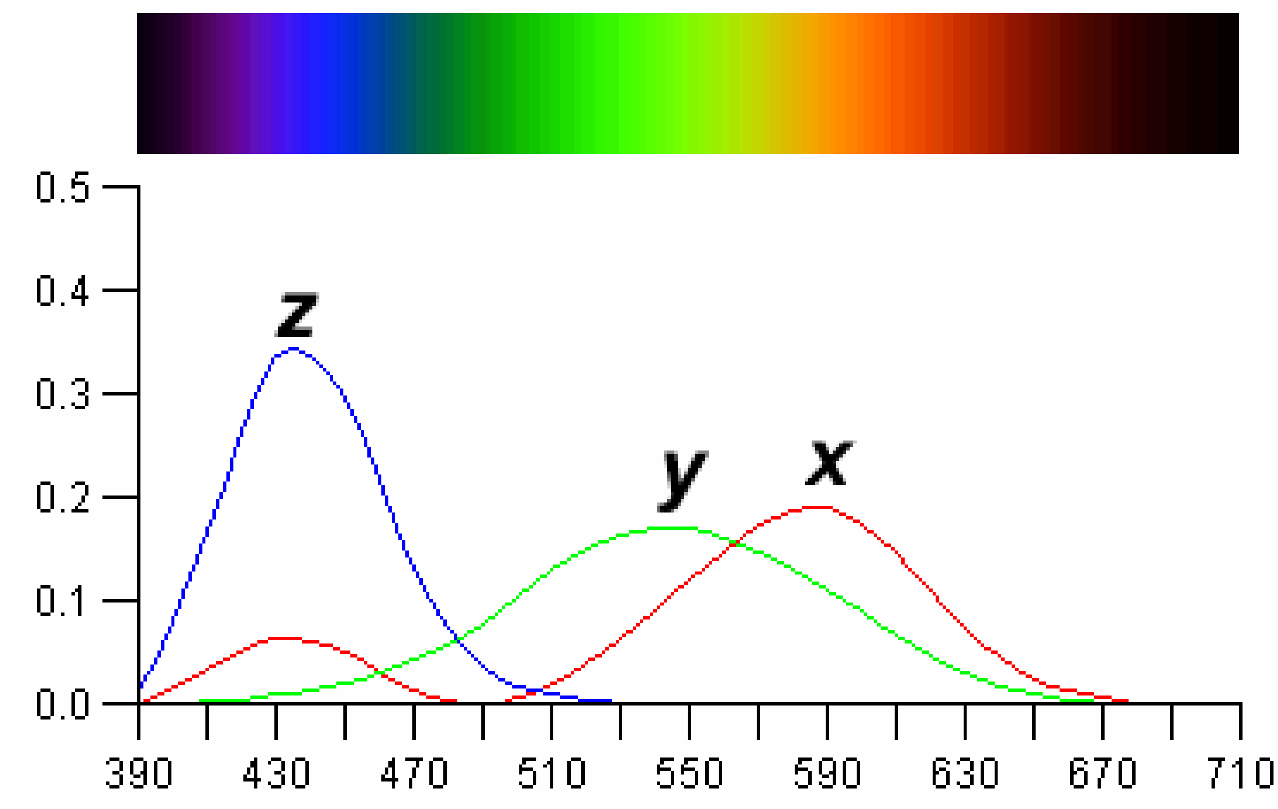
- solution: convert to new synthetic coordinate system to make the job easy

CIE Color Space

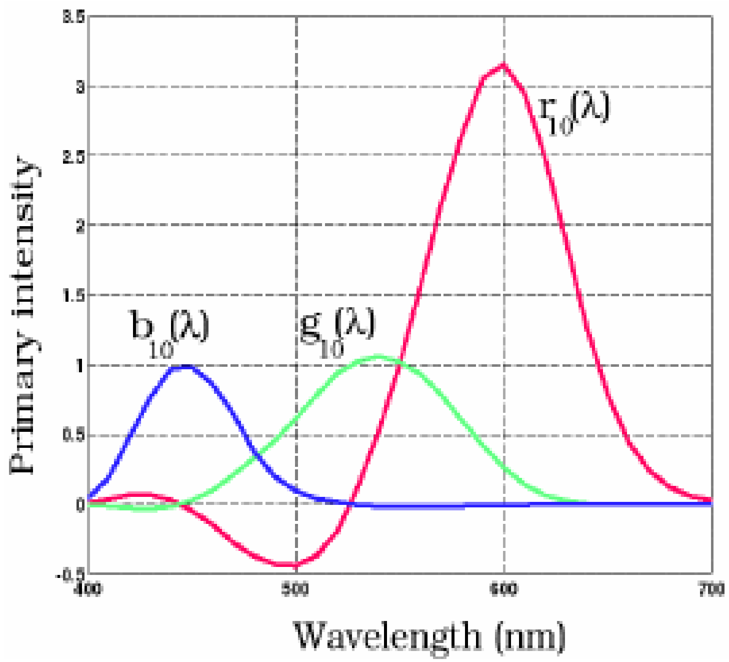
- CIE defined three "imaginary" lights X, Y, and Z, any wavelength λ can be matched perceptually by positive combinations

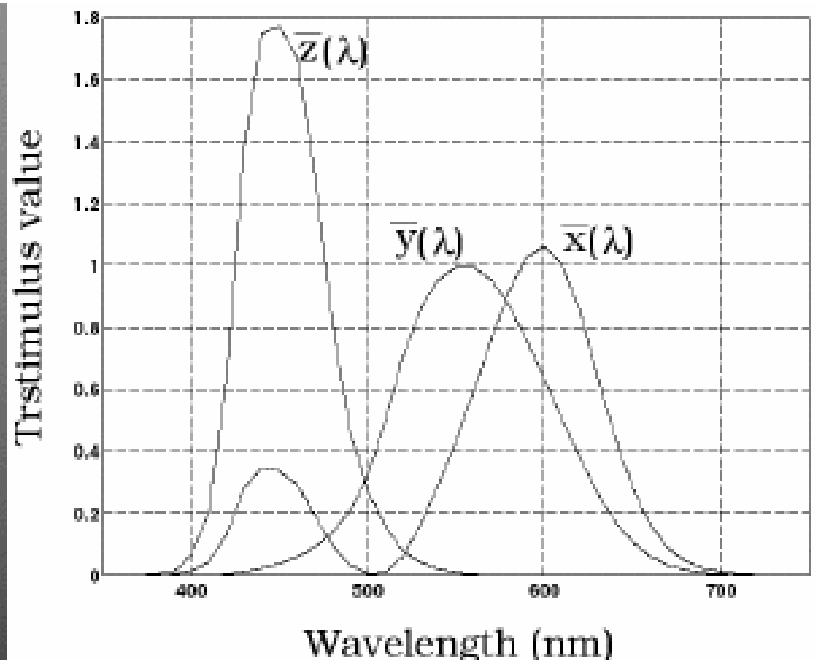
Note that:

X ~ R
Y ~ G
Z ~ B



Measured vs. CIE Color Spaces

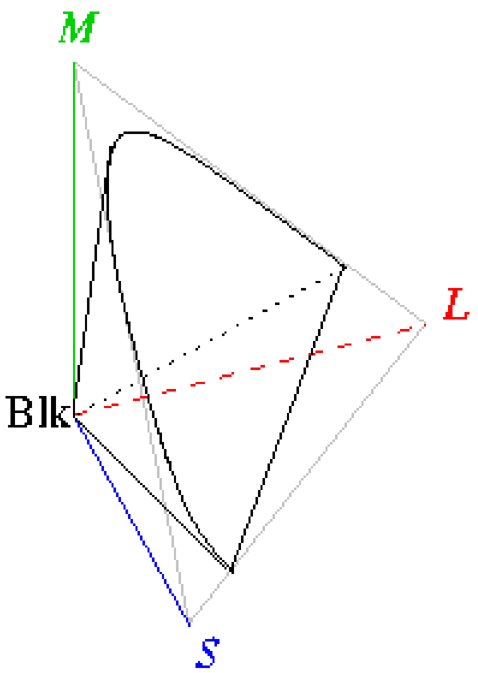




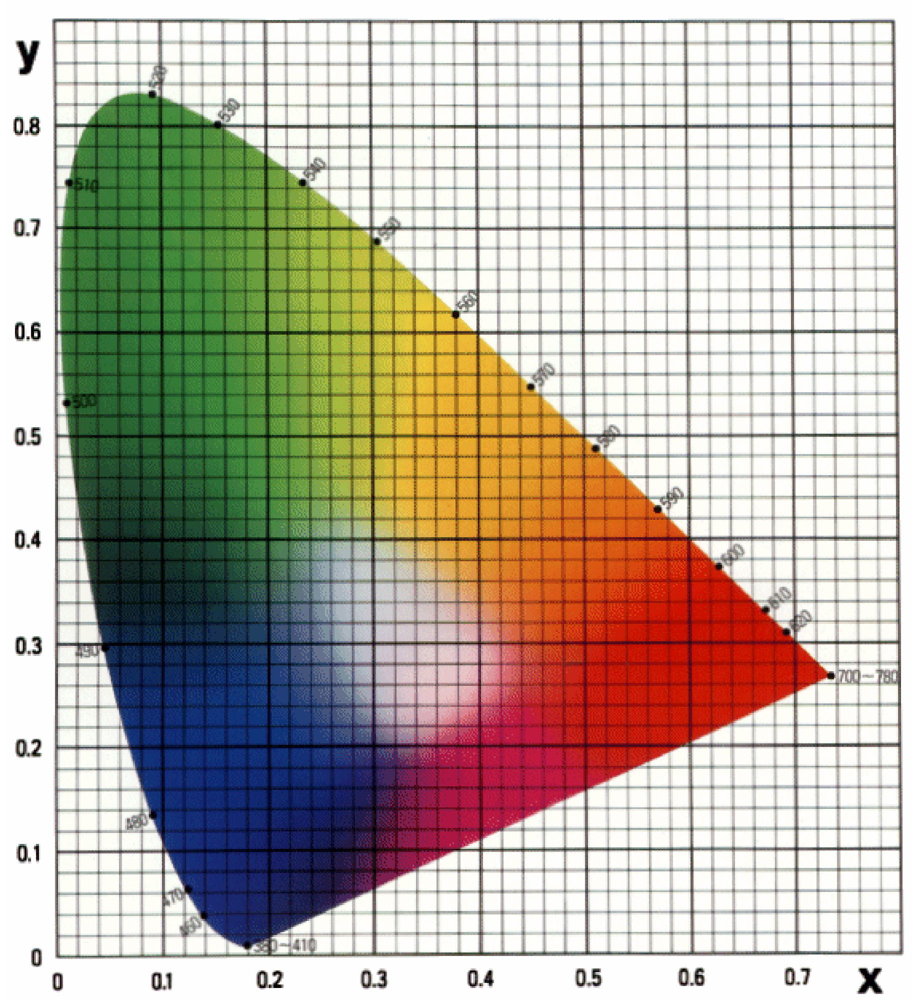
- measured basis
  - monochromatic lights
  - physical observations
  - negative lobes
- transformed basis
  - “imaginary” lights
  - all positive, unit area
  - Y is luminance, no hue
  - X,Z no luminance

CIE Gamut and Chromaticity Diagram

- 3D gamut

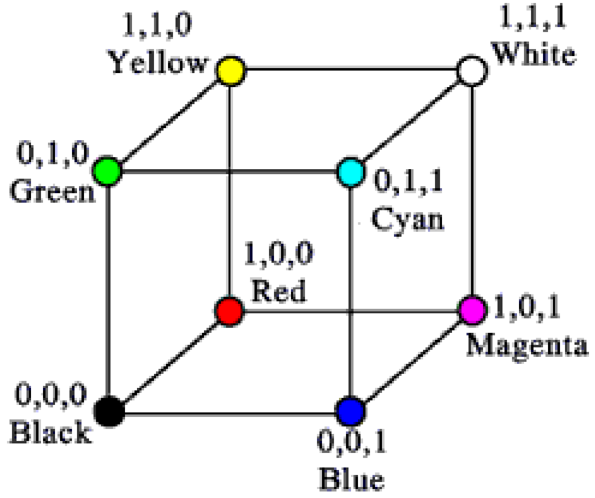


- chromaticity diagram
  - hue only, no intensity

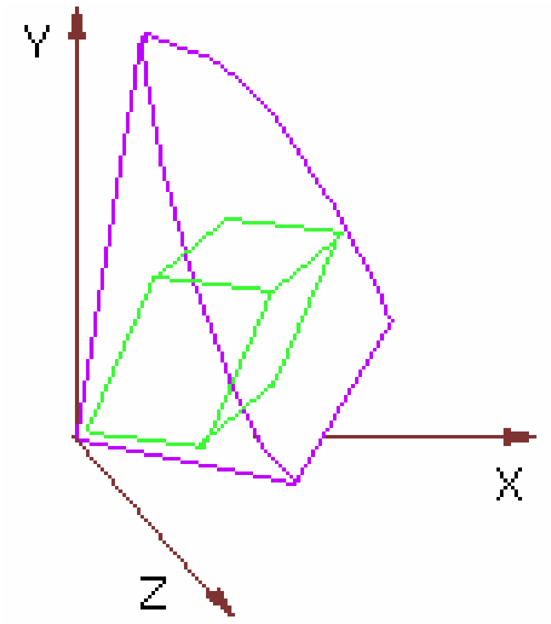


RGB Color Space (Color Cube)

- define colors with (r, g, b) amounts of red, green, and blue
  - used by OpenGL
  - hardware-centric

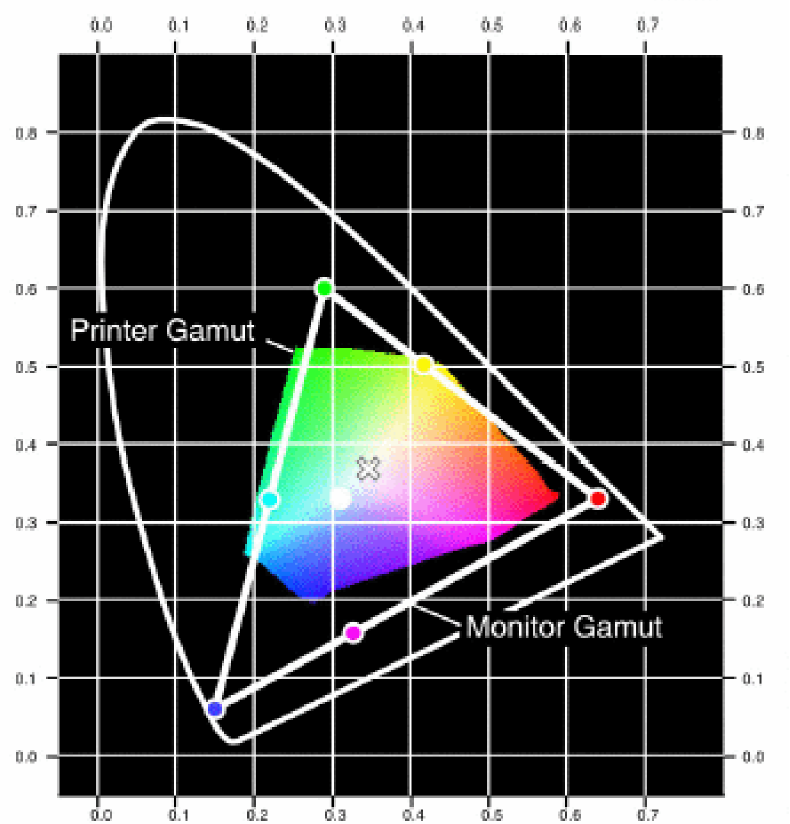


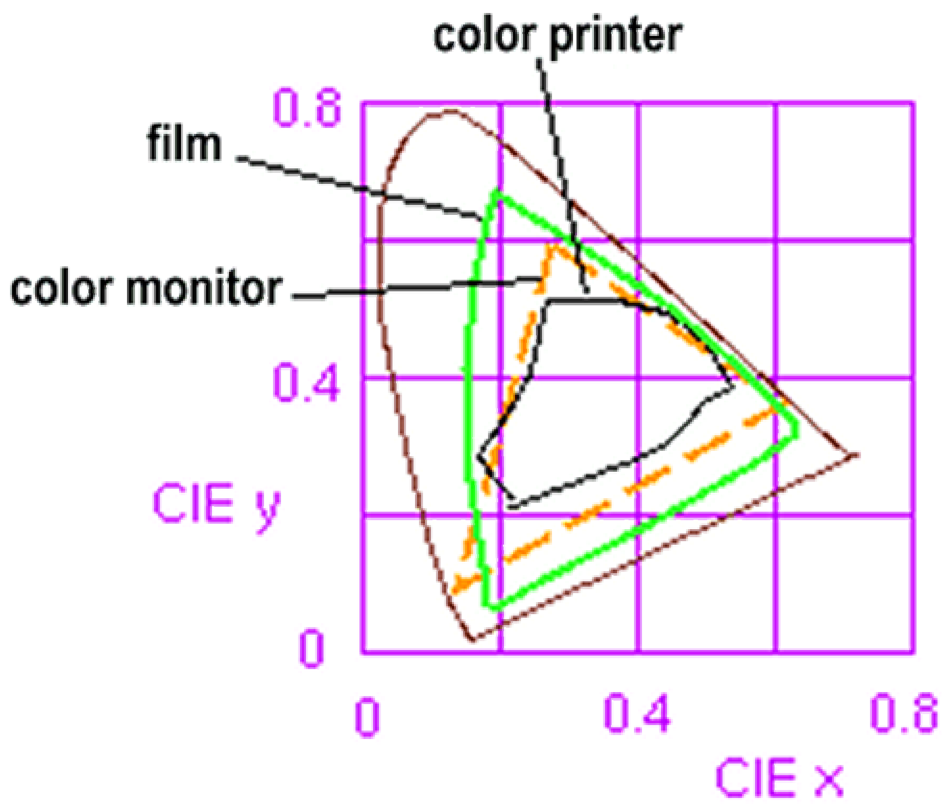
- RGB color cube sits within CIE color space
  - subset of perceivable colors
  - scale, rotate, shear cube



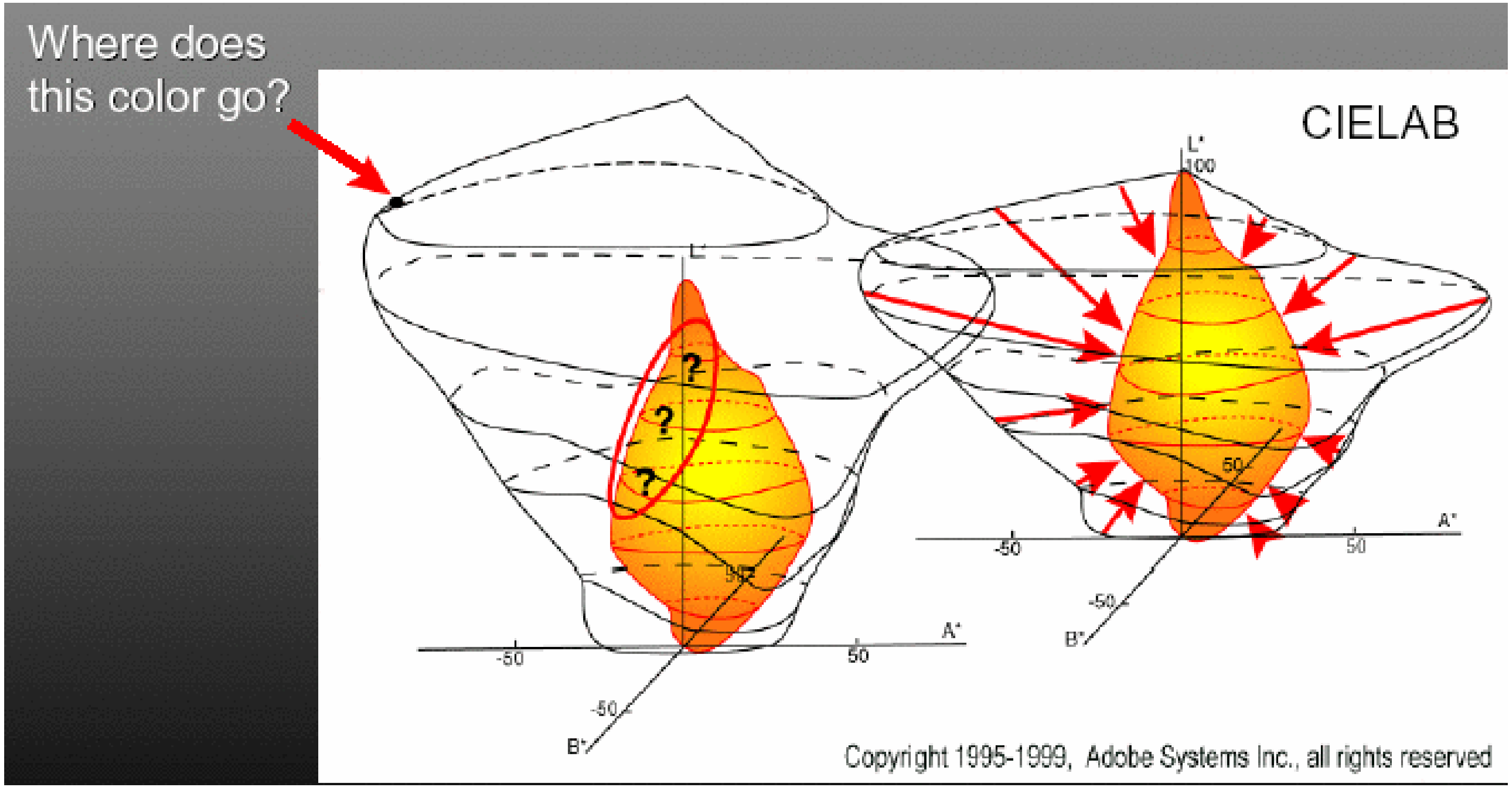
Device Color Gamuts

- use CIE chromaticity diagram to compare the gamuts of various devices
  - X, Y, and Z are hypothetical light sources, no device can produce entire gamut





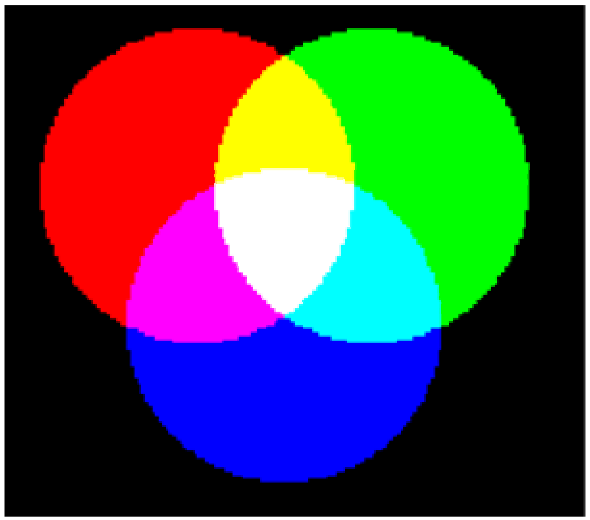
Gamut Mapping

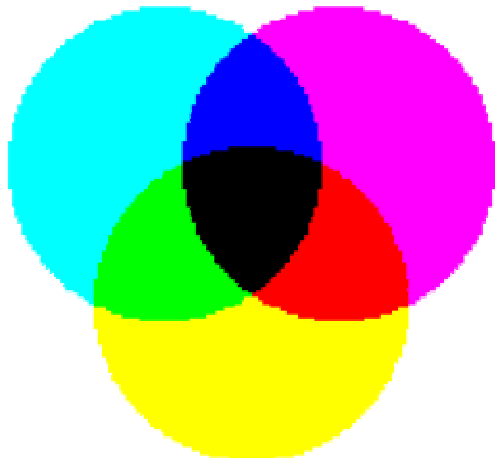


Additive vs. Subtractive Colors

- additive: light
  - monitors, LCDs
  - RGB model
- subtractive: pigment
  - printers
  - CMY model

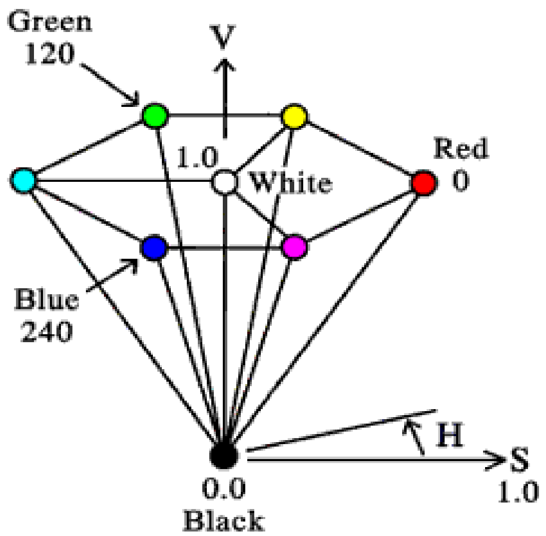
$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

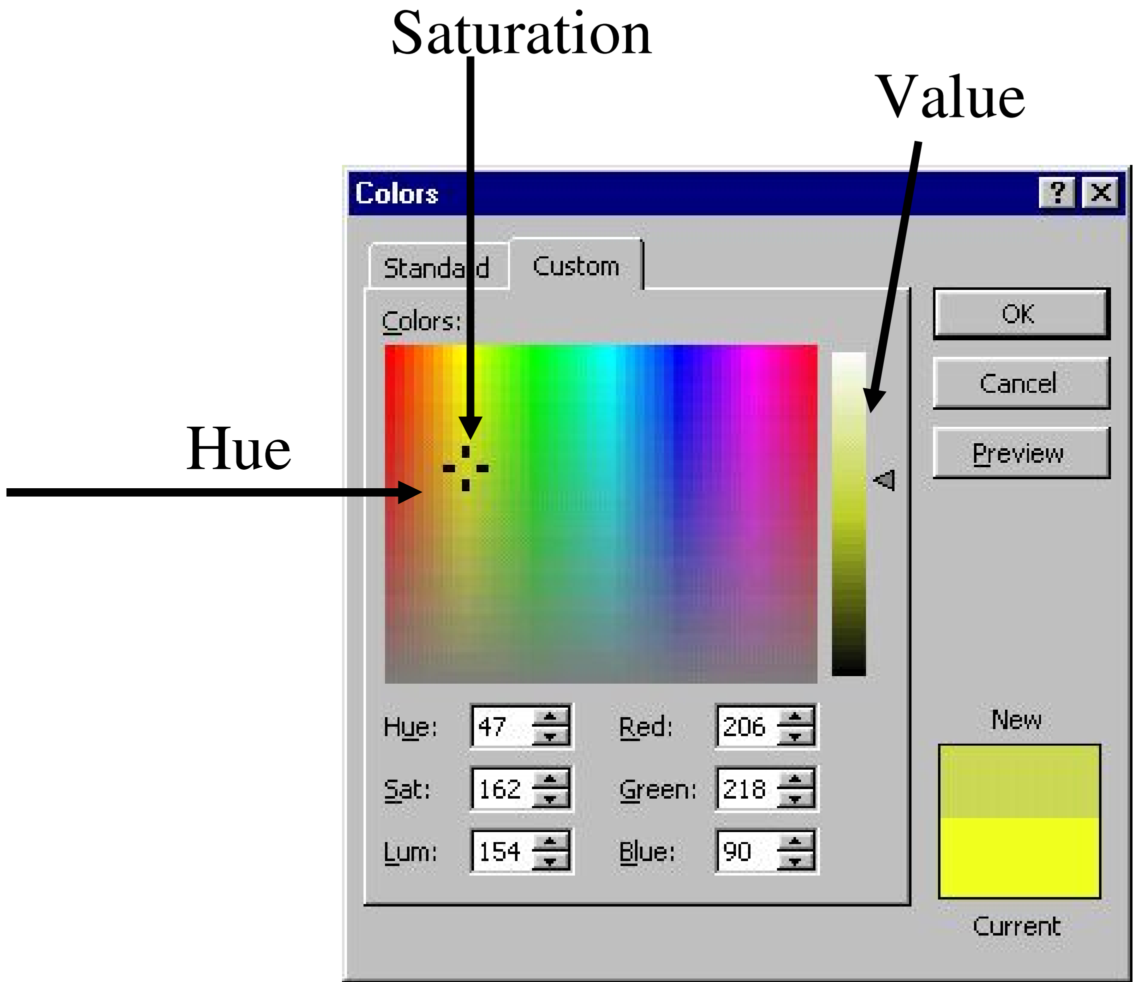




HSV Color Space

- more intuitive color space for people
  - H = Hue
  - S = Saturation
  - V = Value
    - or brightness B
    - or intensity I
    - or lightness L





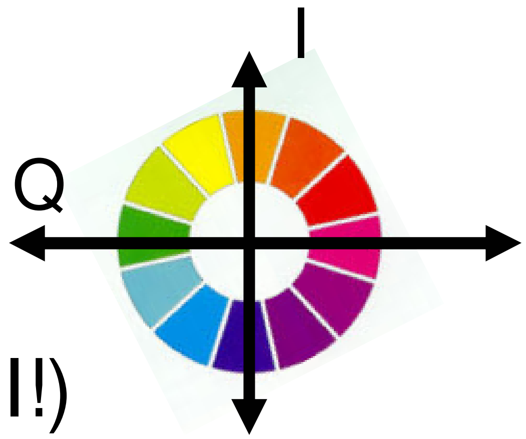
HSI Color Space

- conversion from RGB
  - not expressible in matrix

$$I = \frac{R+G+B}{3} \qquad S = 1 - \frac{\min(R+G+B)}{I}$$

$$H = \cos^{-1}\left[\frac{\frac{1}{2}\left[(R-G)+(R-B)\right]}{\sqrt{(R-G)^2+(R-B)(G-B)}} \right]$$

YIQ Color Space



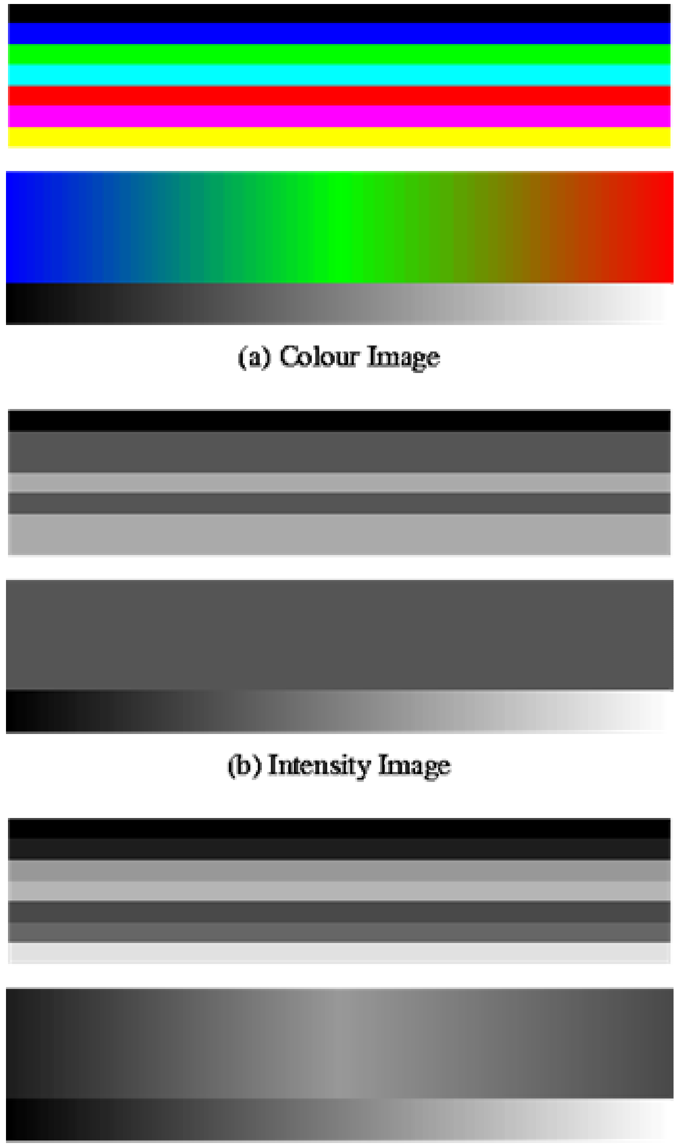
- color model used for color TV
  - Y is luminance (same as CIE)
  - I & Q are color (not same I as HSI!)
  - using Y backwards compatible for B/W TVs
  - conversion from RGB is linear

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.30 & 0.59 & 0.11 \\ 0.60 & -0.28 & -0.32 \\ 0.21 & -0.52 & 0.31 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

  - green is much lighter than red, and red lighter than blue

Luminance vs. Intensity

- luminance
  - Y of YIQ
  - 0.299R + 0.587G + 0.114B
- intensity/brightness
  - I/V/B of HSI/HSV/HSB
  - 0.333R + 0.333G + 0.333B



(a) Colour Image

(b) Intensity Image

(c) Luminance Image

www.csse.uwa.edu.au/~robyn/Visioncourse/colour/lecture/node5.html

Monitors

- monitors have nonlinear response to input
  - characterize by gamma
    - displayedIntensity = a^{γ} (maxIntensity)
- gamma correction
  - displayedIntensity = $\left(a^{1/\gamma}\right)^{\gamma}$(maxIntensity)

    = a (maxIntensity)

Alpha

- transparency
  - (r,g,b,α)
- fraction we can see through
  - $c = \alpha c_f + (1-\alpha)c_b$
- compositing

Program 2: Terrain Navigation

- make colored terrain
  - 100x100 grid
    - two triangles per grid cell
  - face color varies randomly

Navigating

- two flying modes: absolute and relative
- absolute
  - keyboard keys to increment/decrement
  - x/y/z position of eye, lookat, up vectors
- relative
  - mouse drags
  - incremental wrt current camera position
  - forward/backward motion
  - roll, pitch, and yaw angles

Hints: Viewing

- don't forget to flip y coordinate from mouse
  - window system origin upper left
  - OpenGL origin lower left
- all viewing transformations belong in modelview matrix, not projection matrix
  - project 1 template incorrect with this!

Hint: Incremental Motion

- motion is wrt current camera coords
  - maintaining cumulative angles wrt world coords would be difficult
  - computation in coord system used to draw previous frame is simple
  - OpenGL modelview matrix has the info!
    - but multiplying by new matrix gives p'=CIp
    - you want to do p'=ICp
    - trick:
      - dump out modelview matrix
      - wipe the stack with glIdentity
      - apply incremental update matrix
      - apply current camera coord matrix

Demo