University of British Columbia
CPSC 314 Computer Graphics
Jan-Apr 2007

Tamara Munzner

Final Review

Week 13, Wed Apr 11

http://www.ugrad.cs.ubc.ca/~cs314/Vjan2007

Evaluations

- UBC form
- my custom form
  - if you missed class, blanks will be in extra handouts container in lab, can turn in anonymously to the front desk on 2nd floor
  - your feedback helps me improve the course in later years

Getting Help

- extra TA office hours in lab for hw/project Q&A
  - Wed 2-4, Thu 4-6, Fri 9-6
- final review Q&A session
  - Mon Apr 16 10-12

- reminder: my office hours Wed/Fri 11-12 in basement lab

Project 4 Grading

- project 4 grading slots signup
  - Wed Apr 18 10-12
  - Wed Apr 18 4-6
  - Fri Apr 20 10-1

Homework 4

- Q6 corrections, posted very late
  - 8 bins, not 7
  - for part b, give z-values in camera, not world, coordinate system
  - hint on nonuniform depth was for camera, not DCS

- H4 solutions out soon for you to study, check web site. Sunday at latest. Contact me ASAP if you plan to turn in late, we will not accept late homeworks after solutions posted.

Picking Up Work

- still have some marked work not picked up, come grab it!
  - homeworks, midterms
- all extra handouts in lab

Final

- Tue Apr 17 8:30am-11:30am
  - exam will be timed for 2.5 hours, but reserve entire 3-hour block of time just in case
  - closed book, closed notes
  - except for 2-sided 8.5”x11” sheet of handwritten notes
    - fine to staple midterm sheets back to back
  - calculators ok
  - IDs out and face up

Final Emphasis

- covers entire course
- includes material from both midterms
- more than 1/3 on material after last midterm

- clipping
- hidden surfaces
- textures
- procedural approaches
- picking
- collision
- antialiasing
- visualization
- modern hardware
- curves

Reading from OpenGL Red Book

- 1: Introduction to OpenGL
- 2: State Management and Drawing Geometric Objects
- 3: Viewing
- 4: Display Lists
- 5: Color
- 6: Lighting
- 9: Texture Mapping
- 12: Selection and Feedback
- 13: Now That You Know
  - only section Object Selection Using the Back Buffer
- Appendix: Basics of GLUT (Aux in v 1.1)
- Appendix: Homogeneous Coordinates and Transformation Matrices

Reading from Shirley: Foundations of CG

- 1: Intro
- 2: Misc Math
  - except for 2.5.1, 2.5.3, 2.7.1, 2.7.3, 2.8, 2.9
- 3: Raster Algs
- 4: Signal Processing (optional!)
- 5: Linear Algebra
  - only 5.1-5.2.2, 5.2.5
- 6: Transforms
  - except 6.1.6
- 7: Viewing
- 8: Hidden Surfaces
- 9: Surface Shading
- 10: Ray Tracing
  - only 10.1-10.7, 10.9, 10.11.1
- 11: Texture Mapping
- 12: Graphics Pipeline
  - only 12.1-12.4
- 13: Data Structures
  - only 13.3
- 15: Curves and Surfaces
- 17: Hardware
- 18: Color
- 21: Visual Perception
  - only 21.2.2 and 21.2.4
- 25: Image-Based Rendering
- 26: Visualization

Reading

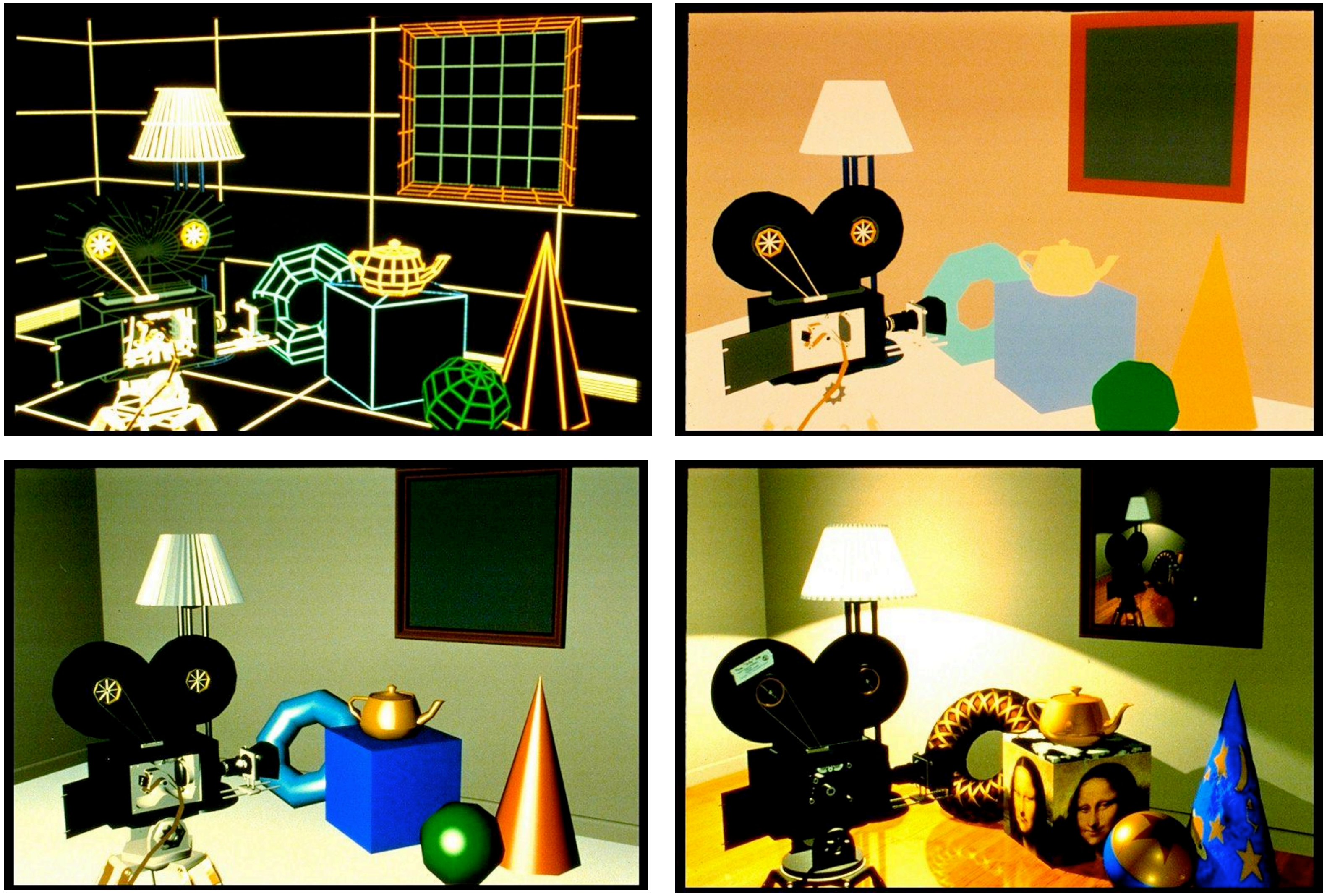
- forgot to post FCG Section 3.4, The Alpha Channel, as reading for Blending last Monday

Studying Advice

- do problems!
  - work through old homeworks, exams

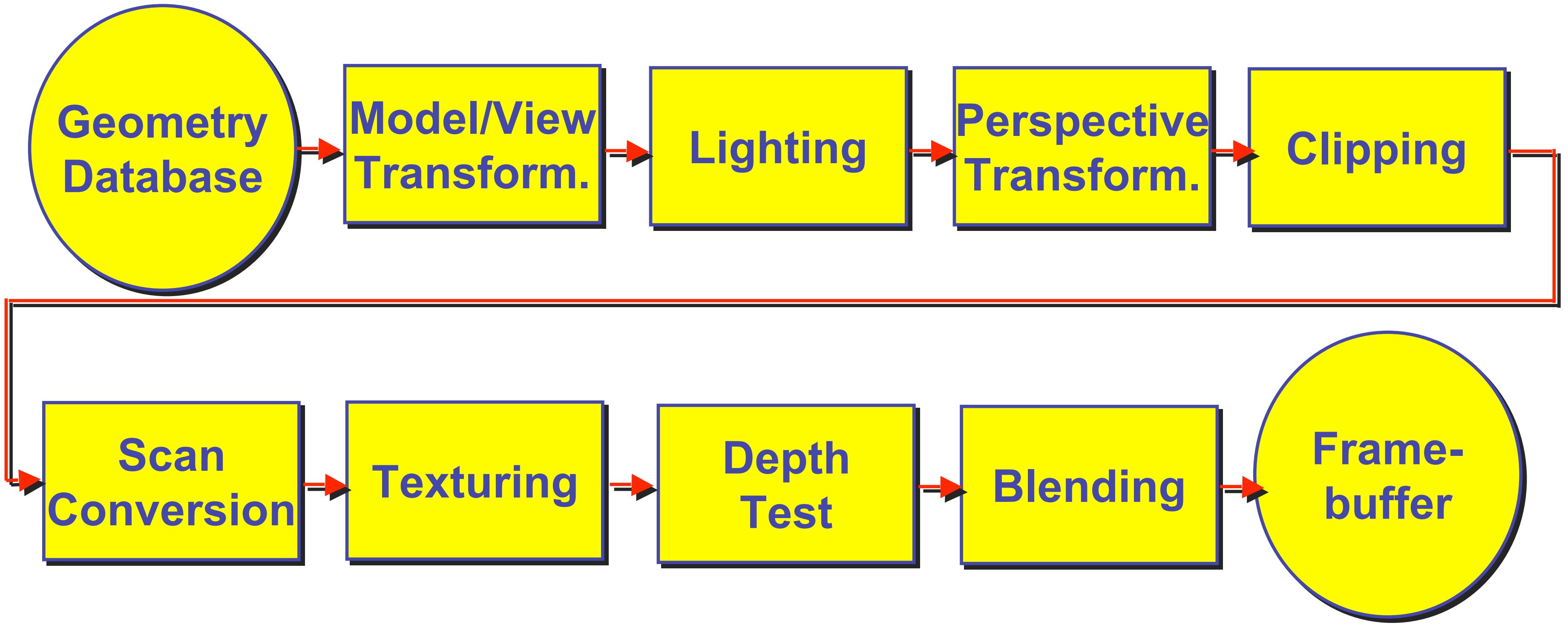
Review – Fast!!

Review: Rendering Capabilities



www.siggraph.org/education/materials/HyperGraph/shutbug.htm

Review: Rendering Pipeline



Review: OpenGL

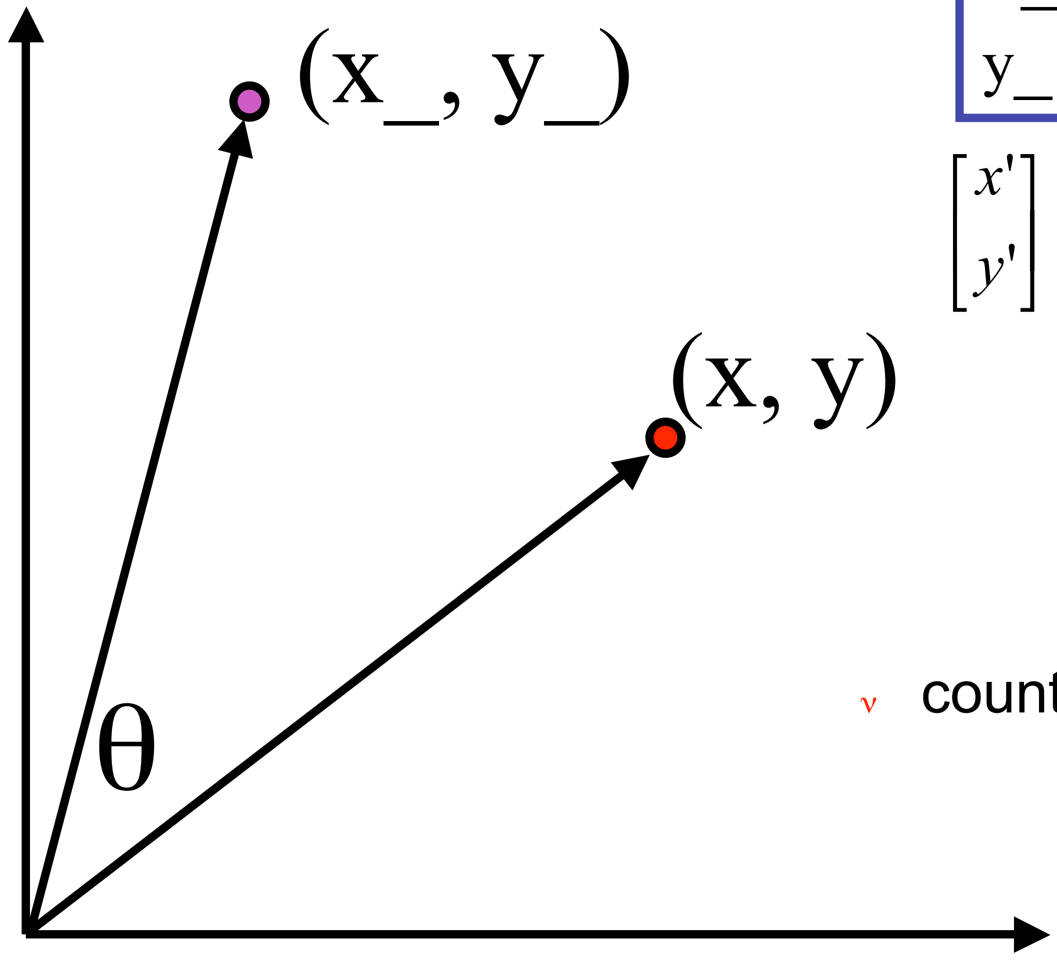
- pipeline processing, set state as needed

```
void display()
{
  glClearColor(0.0, 0.0, 0.0, 0.0);
  glClear(GL_COLOR_BUFFER_BIT);
  glColor3f(0.0, 1.0, 0.0);
  glBegin(GL_POLYGON);
    glVertex3f(0.25, 0.25, -0.5);
    glVertex3f(0.75, 0.25, -0.5);
    glVertex3f(0.75, 0.75, -0.5);
    glVertex3f(0.25, 0.75, -0.5);
  glEnd();
  glFlush();
}
```

Review: Event-Driven Programming

- main loop not under your control
  - vs. procedural
- control flow through event callbacks
  - redraw the window now
  - key was pressed
  - mouse moved
- callback functions called from main loop when events occur
  - mouse/keyboard state setting vs. redrawing

Review: 2D Rotation

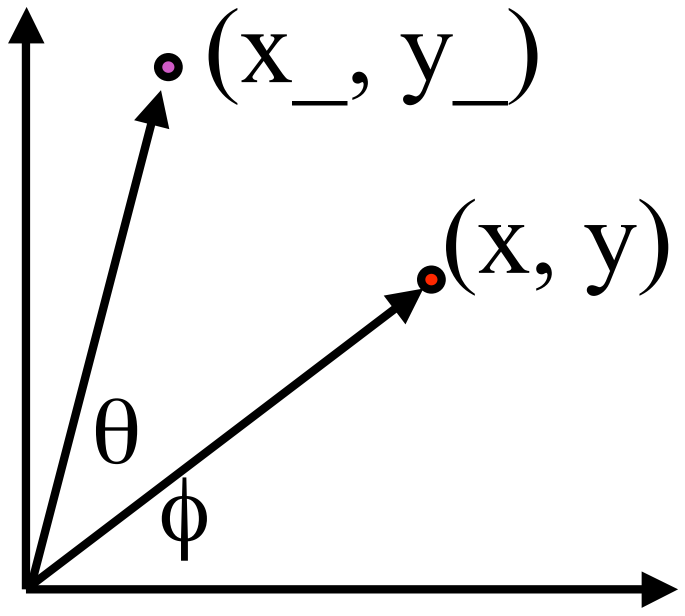


$$\boxed{\begin{aligned} x\_ &= x\,\mathbf{cos}(\theta) - y\,\mathbf{sin}(\theta) \\ y\_ &= x\,\mathbf{sin}(\theta) + y\,\mathbf{cos}(\theta) \end{aligned}}$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

- counterclockwise, RHS

Review: 2D Rotation From Trig Identities



$x = r \cos(\phi)$

$y = r \sin(\phi)$

$x\_ = r \cos(\phi + \theta)$

$y\_ = r \sin(\phi + \theta)$

Trig Identity…

$x\_ = r \cos(\phi) \cos(\theta) - r \sin(\phi) \sin(\theta)$

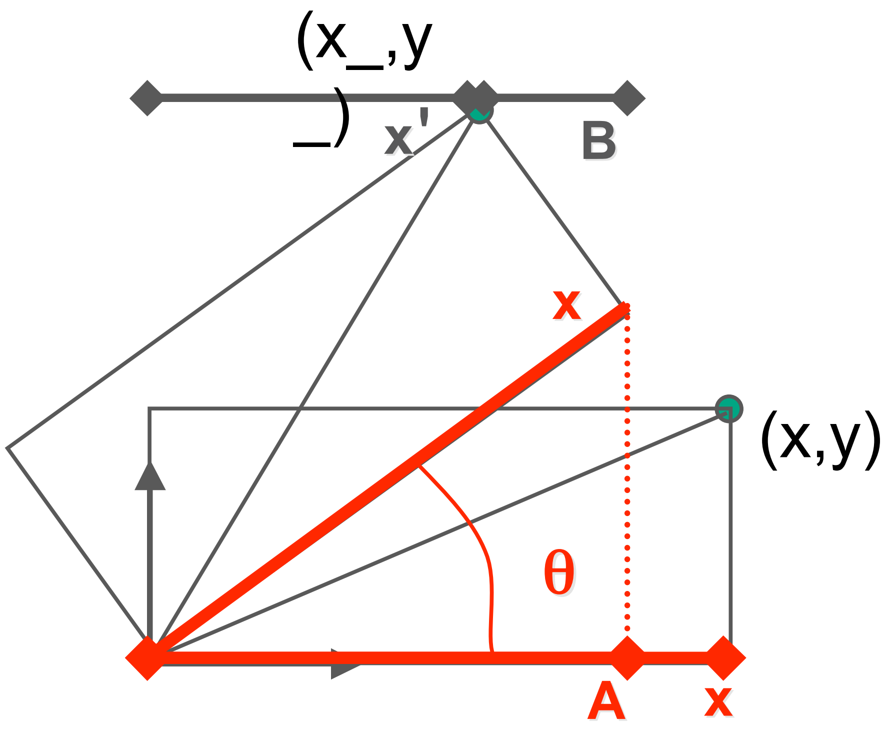
$y\_ = r \sin(\phi) \cos(\theta) + r \cos(\phi) \sin(\theta)$

Substitute…

$x\_ = x\ \mathbf{cos}(\theta) - y\ \mathbf{sin}(\theta)$

$y\_ = x\ \mathbf{sin}(\theta) + y\ \mathbf{cos}(\theta)$

Review: 2D Rotation: Another Derivation



$$x' = x\cos\theta - y\sin\theta$$

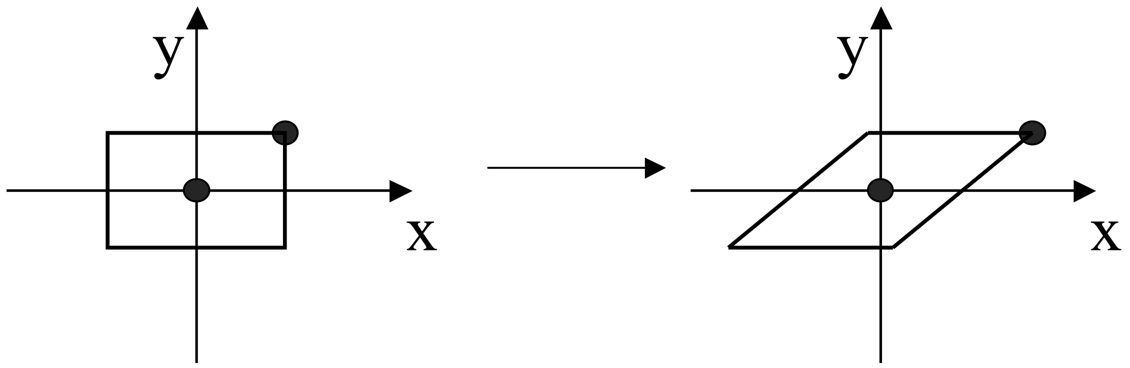
$$y' = x\sin\theta + y\cos\theta$$

$$x' = A - B$$

$$A = x\cos\theta$$

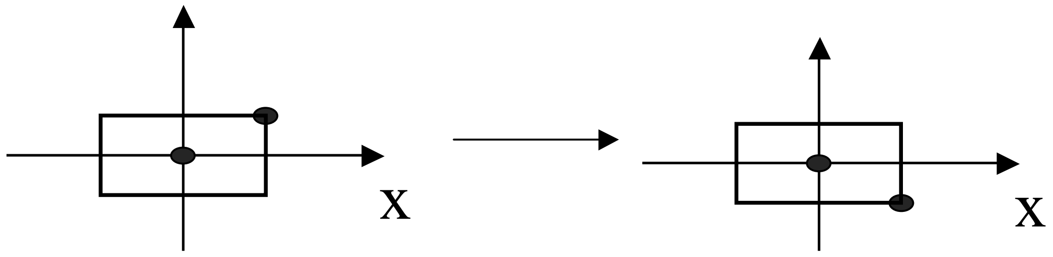
Review: Shear, Reflection

- shear along x axis
  - push points to right in proportion to height



$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & sh_x \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

- reflect across x axis
  - mirror



$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Review: 2D Transformations

matrix multiplication

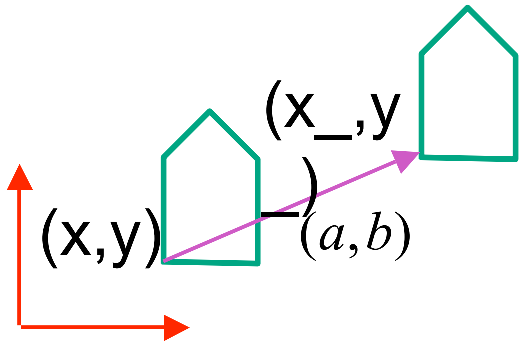
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \underbrace{\begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix}}_{\text{scaling matrix}} \begin{bmatrix} x \\ y \end{bmatrix}$$

scaling matrix

matrix multiplication

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \underbrace{\begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}}_{\text{rotation matrix}} \begin{bmatrix} x \\ y \end{bmatrix}$$

rotation matrix



vector addition

$$\begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} x+a \\ y+b \end{bmatrix} = \begin{bmatrix} x' \\ y' \end{bmatrix}$$

$$\underbrace{\begin{bmatrix} a & b \\ c & d \end{bmatrix}}_{} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x' \\ y' \end{bmatrix}$$

translation multiplication matrix??

Review: Linear Transformations

- linear transformations are combinations of
  - shear
  - scale
  - rotate
  - reflect

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

$$x' = ax + by$$
$$y' = cx + dy$$

- properties of linear transformations
  - satisifes T(*s*$\mathbf{x}$+*t*$\mathbf{y}$) = *s* T($\mathbf{x}$) + *t* T($\mathbf{y}$)
  - origin maps to origin
  - lines map to lines
  - parallel lines remain parallel
  - ratios are preserved
  - closed under composition

Review: 3D Homog Transformations

- use 4x4 matrices for 3D transformations

translate(a,b,c)

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & & & a \\ & 1 & & b \\ & & 1 & c \\ & & & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

scale(a,b,c)

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} a & & & \\ & b & & \\ & & c & \\ & & & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$\text{Rotate}(x,\theta)$

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & & & \\ & \cos\theta & -\sin\theta & \\ & \sin\theta & \cos\theta & \\ & & & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$\text{Rotate}(y,\theta)$

$$\begin{bmatrix} \cos\theta & & \sin\theta & \\ & 1 & & \\ -\sin\theta & & \cos\theta & \\ & & & 1 \end{bmatrix}$$

$\text{Rotate}(z,\theta)$

$$\begin{bmatrix} \cos\theta & -\sin\theta & & \\ \sin\theta & \cos\theta & & \\ & & 1 & \\ & & & 1 \end{bmatrix}$$

Final Correction: 3D Shear

- general shear $shear(hxy, hxz, hyx, hyz, hzx, hzy) = \begin{bmatrix} 1 & hyx & hzx & 0 \\ hxy & 1 & hzy & 0 \\ hxz & hyz & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

- "x-shear" usually means shear along x in direction of some other axis
  - **correction: not** shear along some axis in direction of x
  - to avoid ambiguity, always say "shear along <axis> in direction of <axis>"

$$shearAlongXinDirectionOfY(h) = \begin{bmatrix} 1 & h & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad shearAlongXinDirectionOfZ(h) = \begin{bmatrix} 1 & 0 & h & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$shearAlongYinDirectionOfX(h) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ h & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad shearAlongYinDirectionOfZ(h) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & h & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$shearAlongZinDirectionOfX(h) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ h & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad shearAlongZinDirectionOfY(h) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & h & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

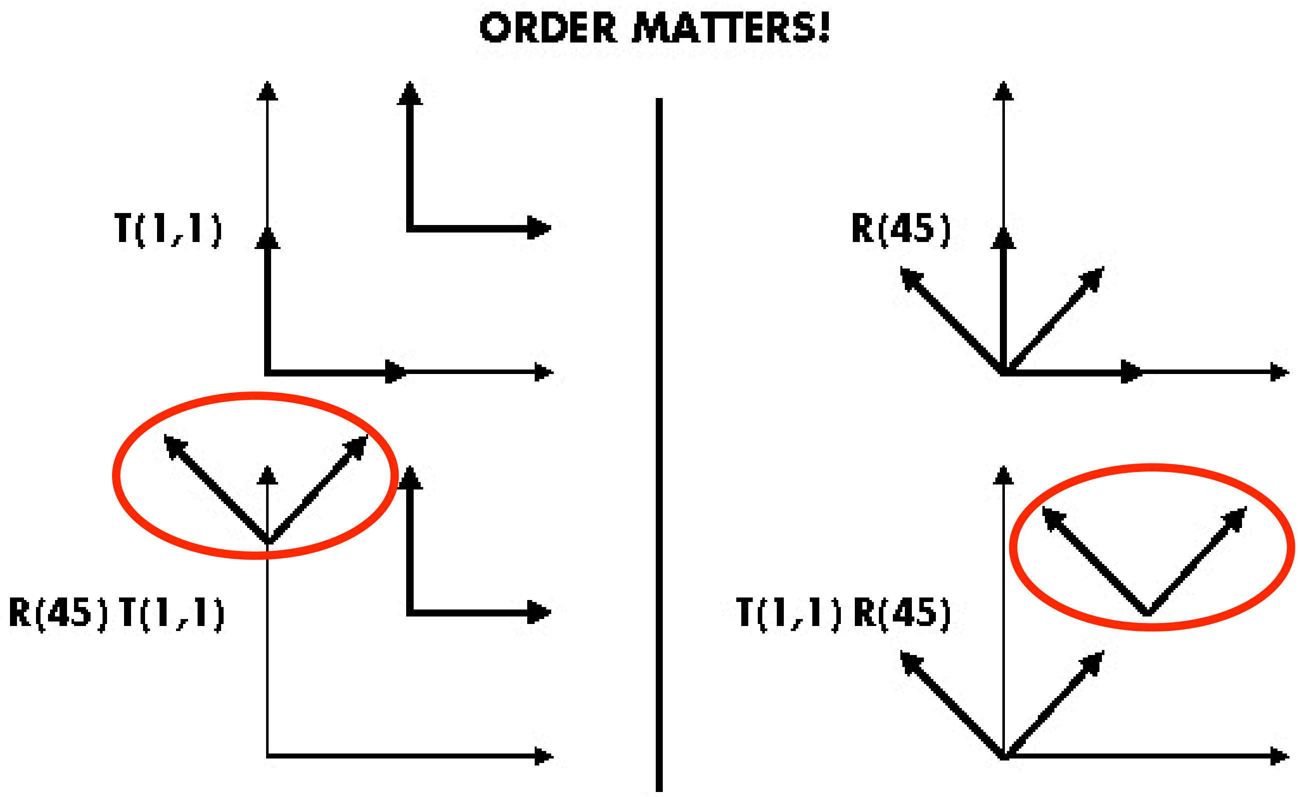
Review: Affine Transformations

- affine transforms are combinations of
  - linear transformations
  - translations

$$\begin{bmatrix} x' \\ y' \\ w \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

- properties of affine transformations
  - origin does not necessarily map to origin
  - lines map to lines
  - parallel lines remain parallel
  - ratios are preserved
  - closed under composition

Review: Composing Transformations



Ta Tb = Tb Ta, but Ra Rb != Rb Ra and Ta Rb != Rb Ta

- rotations around different axes do not commute

Review: Composing Transformations

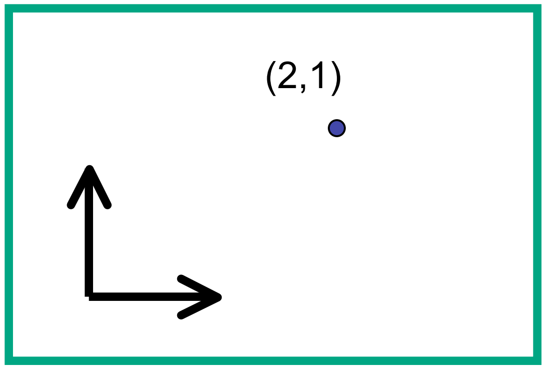
$$\mathbf{p}' = \mathbf{T}\mathbf{R}\mathbf{p}$$

- which direction to read?
  - right to left
    - interpret operations wrt fixed coordinates
    - moving object
  - left to right **OpenGL pipeline ordering!**
    - interpret operations wrt local coordinates
    - changing coordinate system
    - OpenGL updates current matrix with postmultiply
      - glTranslatef(2,3,0);
      - glRotatef(-90,0,0,1);
      - glVertexf(1,1,1);
    - specify vector last, in final coordinate system
    - first matrix to affect it is specified second-to-last

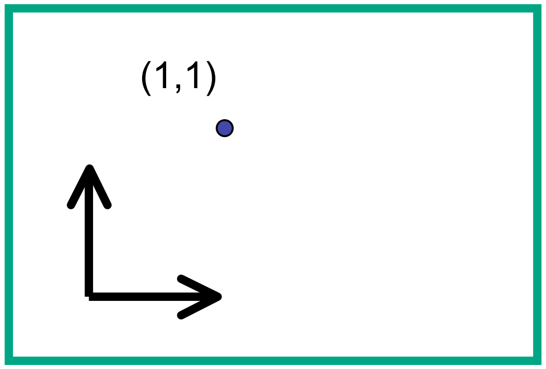
Review: Interpreting Transformations

$$p' = TRp$$

translate by (-1,0)

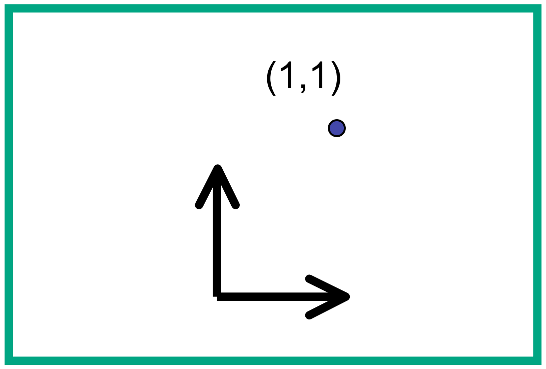


right to left: moving object



intuitive?

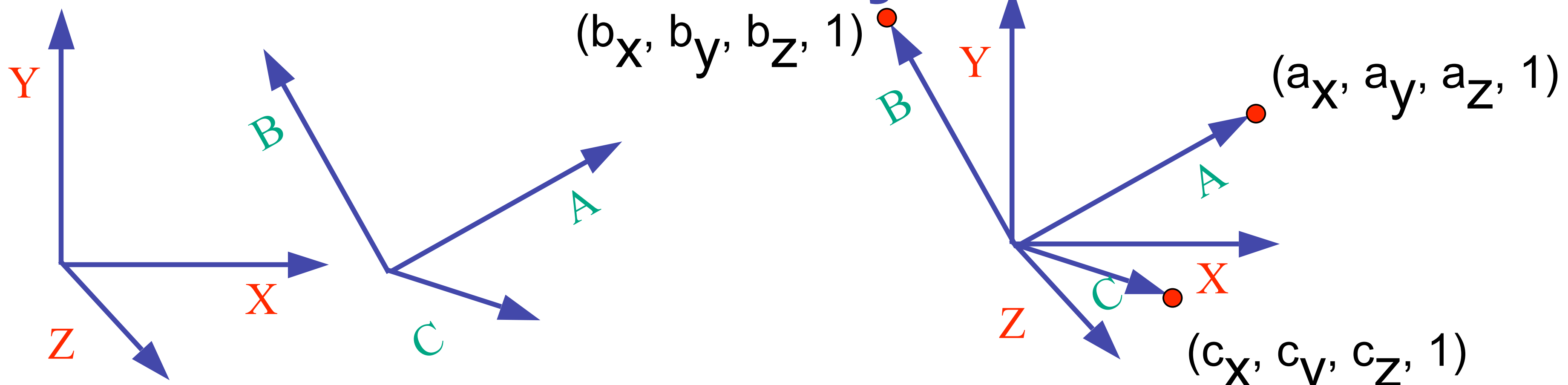
left to right: changing coordinate system



OpenGL

- same relative position between object and basis vectors

Review: Arbitrary Rotation

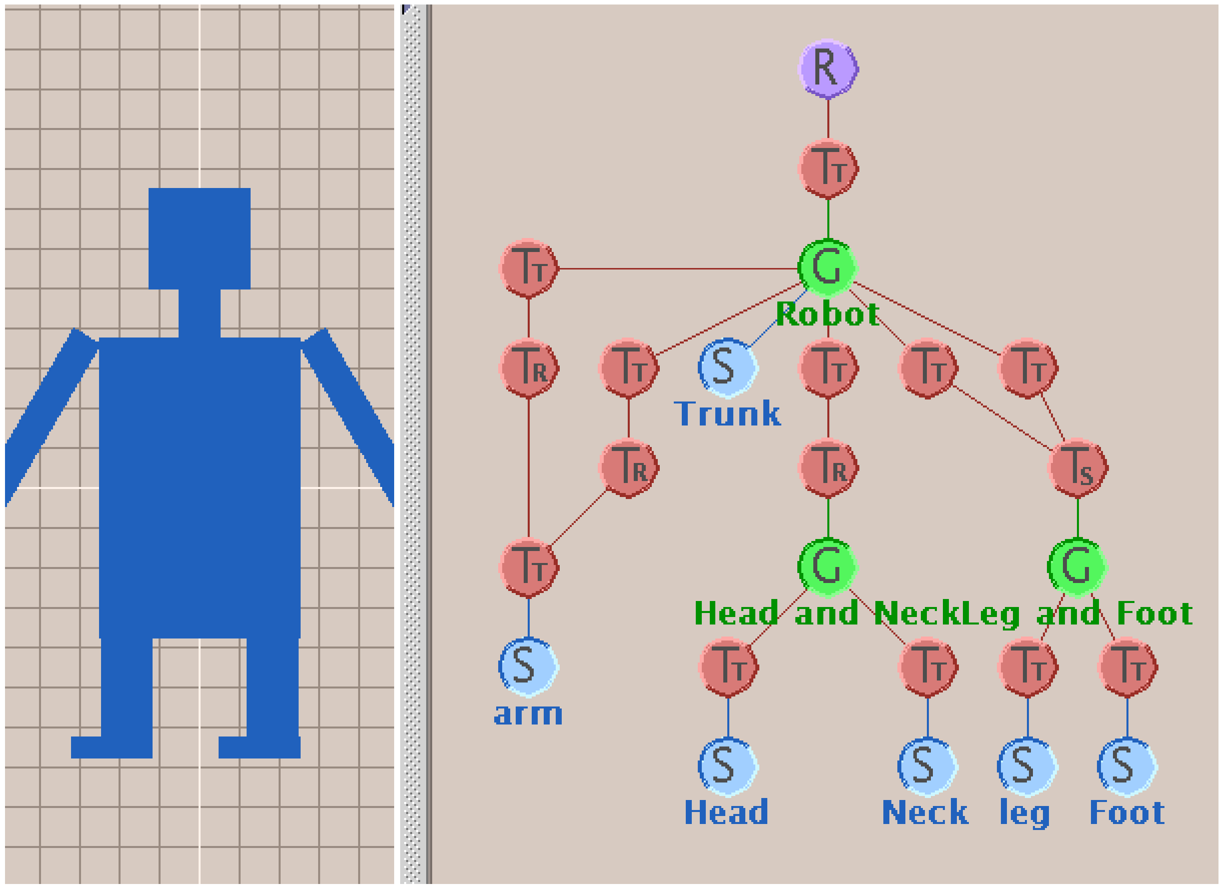


- arbitrary rotation: change of basis
  - given two orthonormal coordinate systems *XYZ* and *ABC*
    - *A*'s location in the XYZ coordinate system is $(a_x, a_y, a_z, 1)$, ...
- transformation from one to the other is matrix R whose columns are *A,B,C*:

$$R(X) = \begin{bmatrix} a_x & b_x & c_x & 0 \\ a_y & b_y & c_y & 0 \\ a_z & b_z & c_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} = (a_x, a_y, a_z, 1) = A$$

Review: Transformation Hierarchies

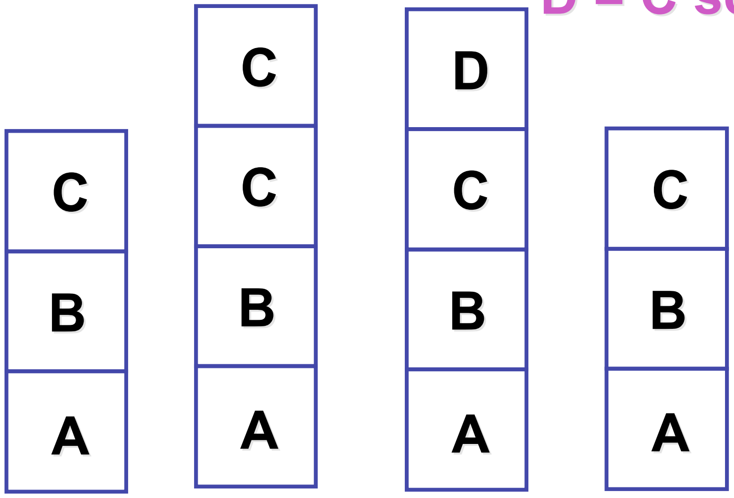
- transforms apply to graph nodes beneath them
- design structure so that object doesn't fall apart
- instancing



Review: Matrix Stacks

- OpenGL matrix calls postmultiply matrix M onto current matrix P, overwrite it to be PM
  - or can save intermediate states with stack
  - no need to compute inverse matrices all the time
  - modularize changes to pipeline state
  - avoids accumulation of numerical errors

D = C scale(2,2,2) trans(1,0,0)



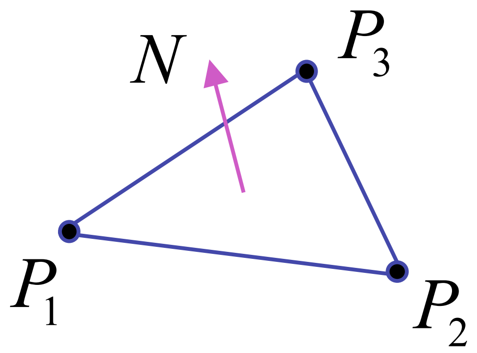
```
DrawSquare()
glPushMatrix()
glScale3f(2,2,2)
glTranslate3f(1,0,0)
DrawSquare()
glPopMatrix()
```

Review: Display Lists

- precompile/cache block of OpenGL code for reuse
  - usually more efficient than immediate mode
    - exact optimizations depend on driver
  - good for multiple instances of same object
    - but cannot change contents, not parametrizable
  - good for static objects redrawn often
    - display lists persist across multiple frames
    - interactive graphics: objects redrawn every frame from new viewpoint from moving camera
  - can be nested hierarchically
- snowman example: 3x performance improvement, 36K polys

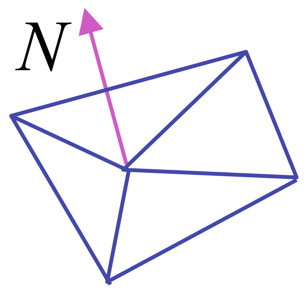
Review: Normals

- polygon:



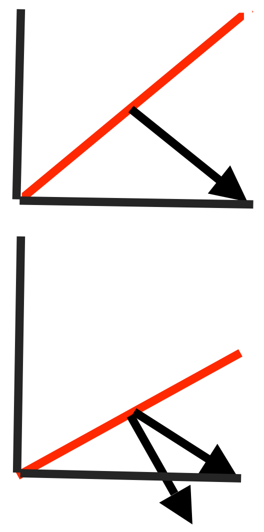
$$N = (P_2 - P_1) \times (P_3 - P_1)$$

- assume vertices ordered CCW when viewed from visible side of polygon
- normal for a vertex
  - specify polygon orientation
  - used for lighting
  - supplied by model (i.e., sphere), or computed from neighboring polygons



Review: Transforming Normals

- cannot transform normals using same matrix as points
  - nonuniform scaling would cause to be not perpendicular to desired plane!



$$\begin{array}{lcl} P & & P' = MP \\ N & \xrightarrow{\hspace{3em}} & N' = QN \end{array}$$

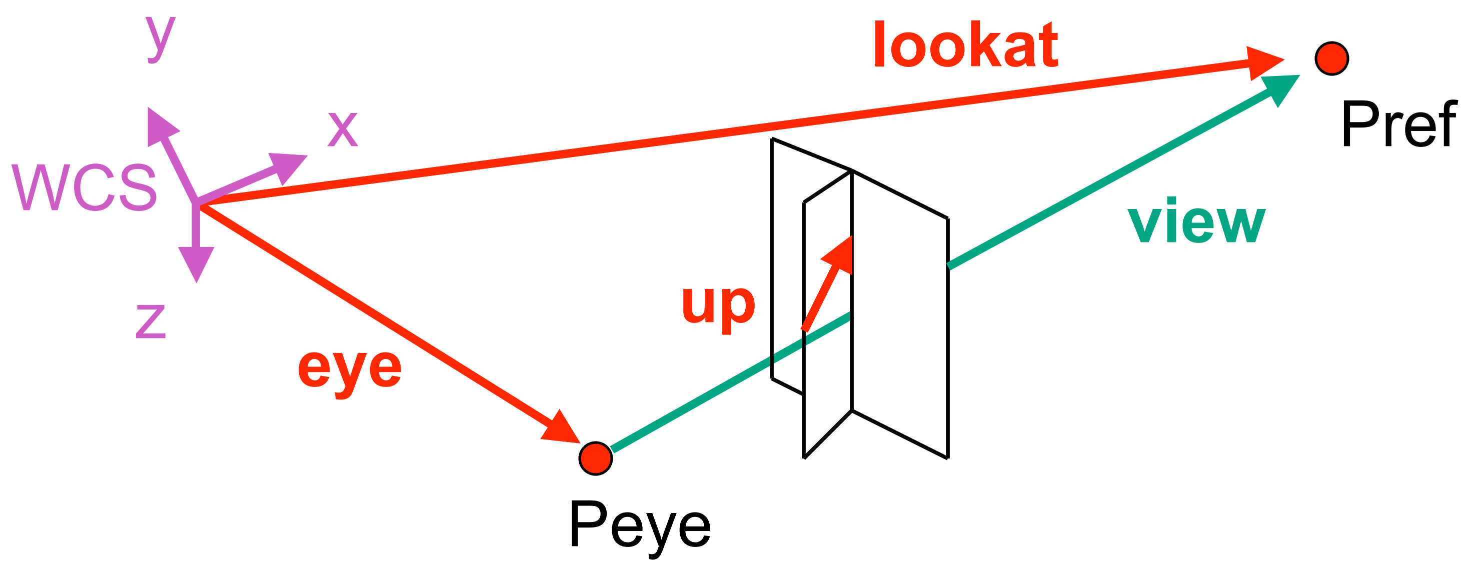
given M,
what should Q be?

$$\boxed{\mathbf{Q} = \left(\mathbf{M}^{-1}\right)^{T}}$$

inverse transpose of the modelling transformation

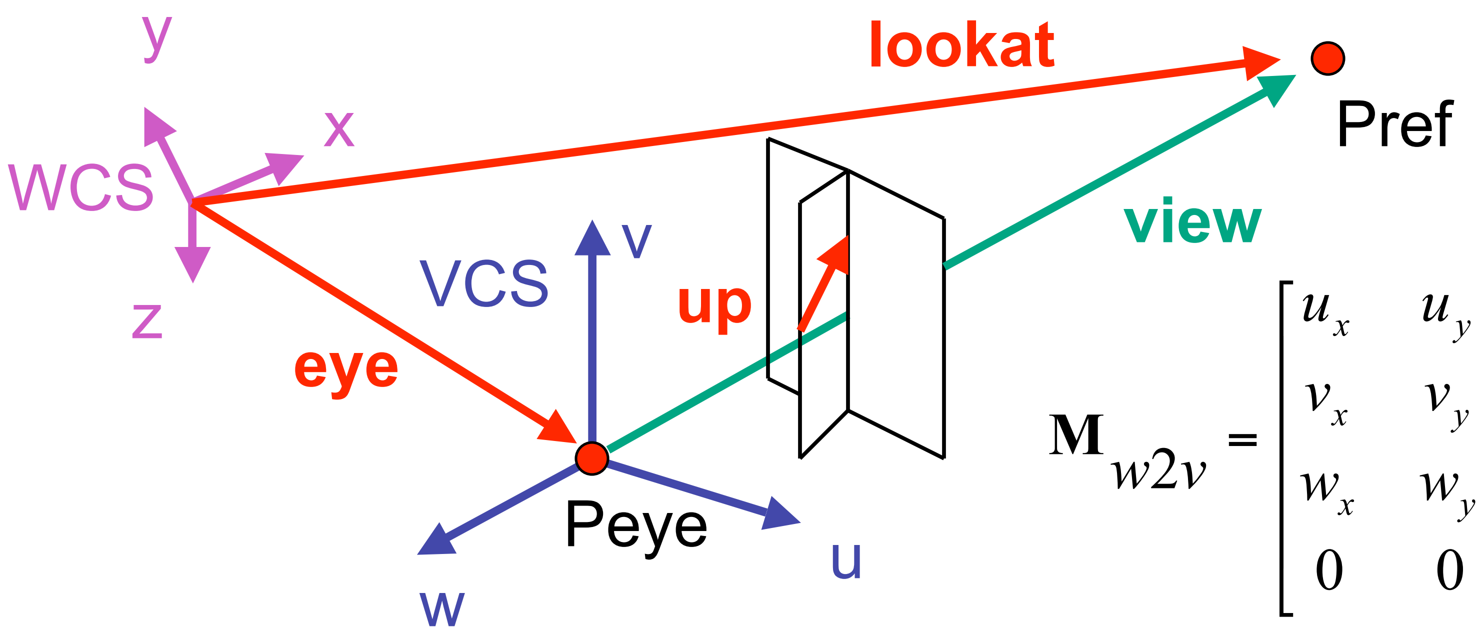
Review: Camera Motion

- 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



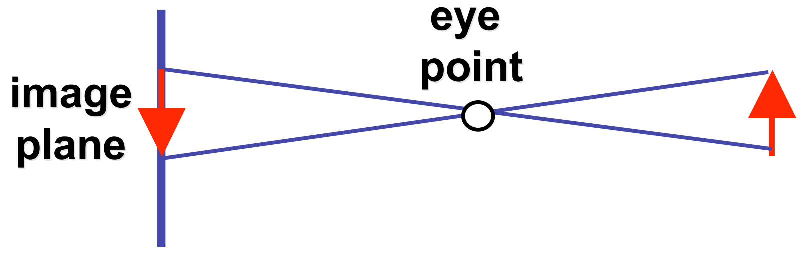
$$\mathbf{M}_{w2v} = \begin{bmatrix} u_x & u_y & u_z & -\mathbf{u}\bullet\mathbf{e} \\ v_x & v_y & v_z & -\mathbf{v}\bullet\mathbf{e} \\ w_x & w_y & w_z & -\mathbf{w}\bullet\mathbf{e} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Review: Moving Camera or World?

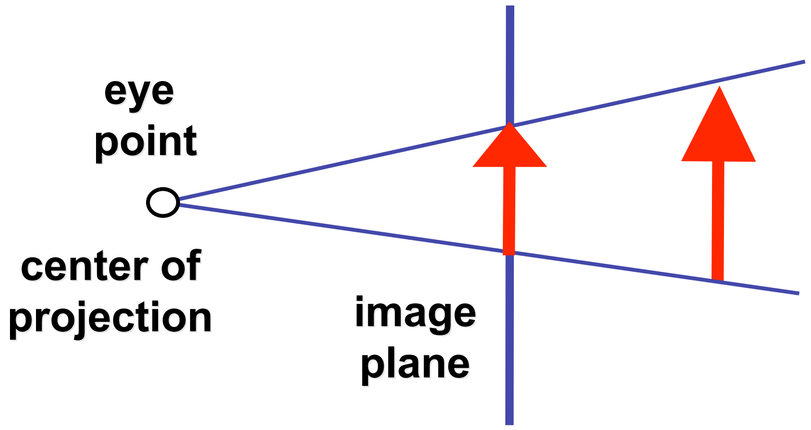
- two equivalent operations
  - move camera one way vs. move world other way
- example
  - initial OpenGL camera: at origin, looking along -z axis
  - create a unit square parallel to camera at z = -10
  - translate in z by 3 possible in two ways
    - camera moves to z = -3
      - Note OpenGL models 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?

Review: Graphics Cameras

- real pinhole camera: image inverted

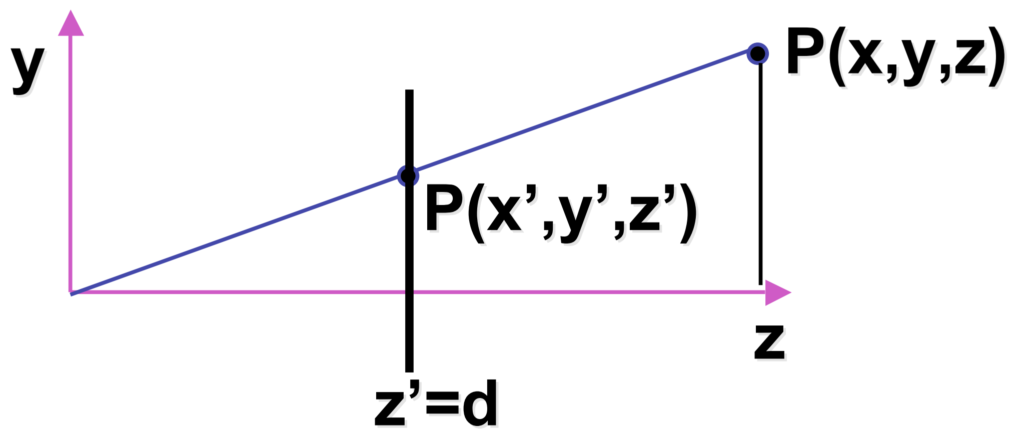


ν computer graphics camera: convenient equivalent



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

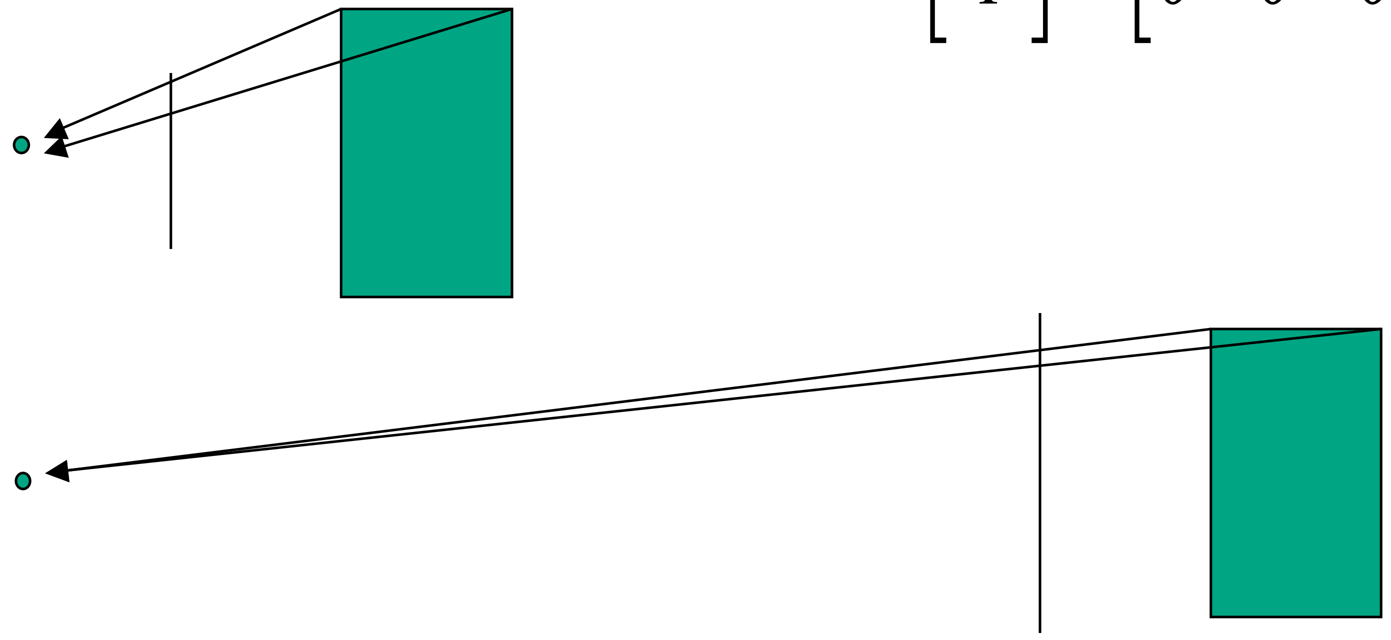
$$\begin{bmatrix} x \\ y \\ z \\ z/d \end{bmatrix}$$

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

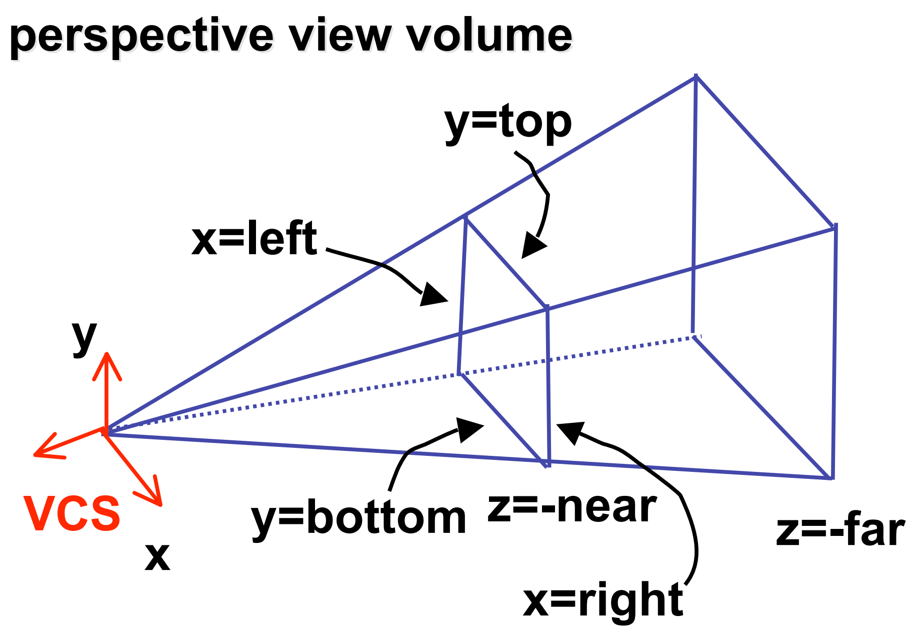
Review: Orthographic Cameras

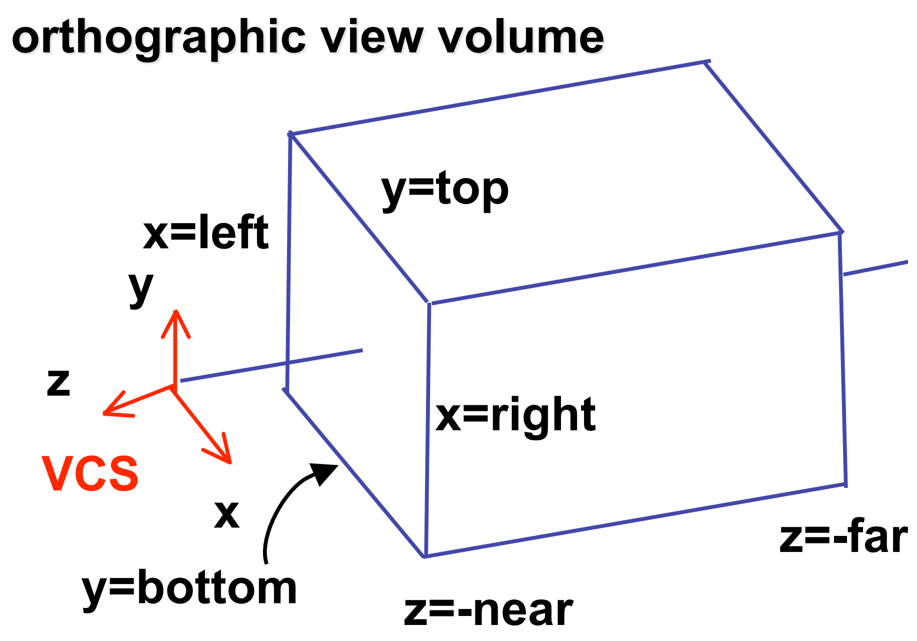
- center of projection at infinity
- no perspective convergence
- just throw away z values

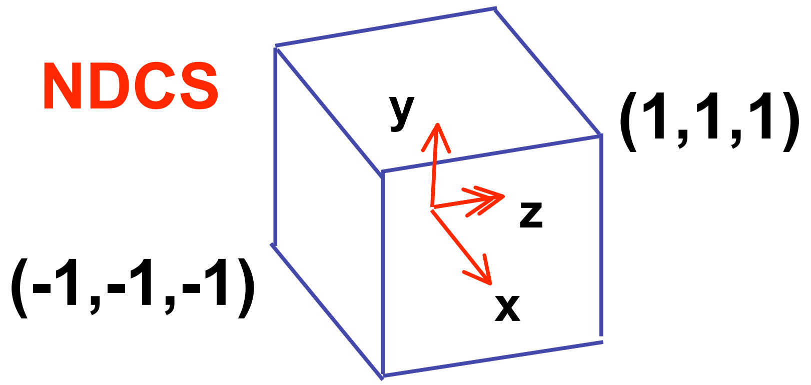
$$\begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



Review: Transforming View Volumes

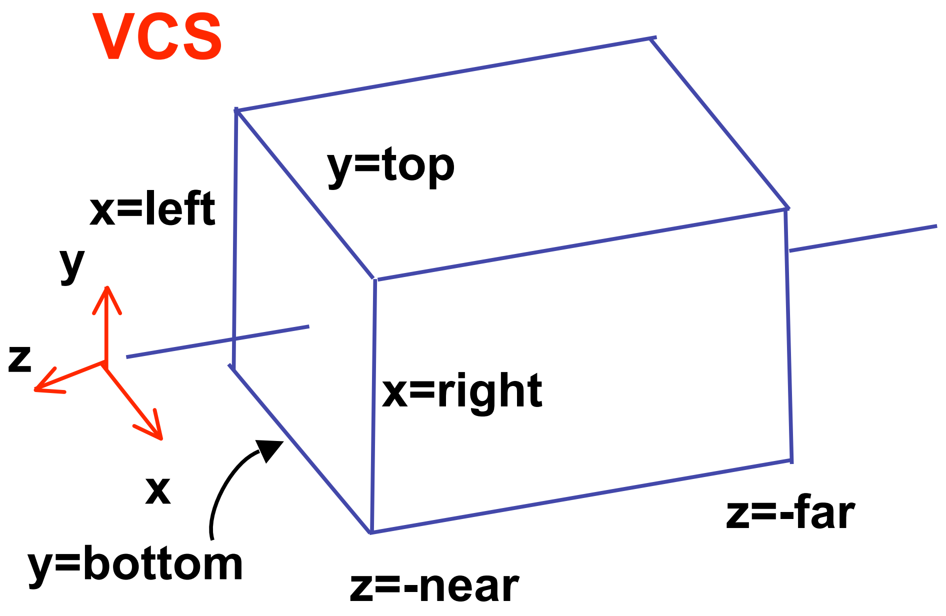


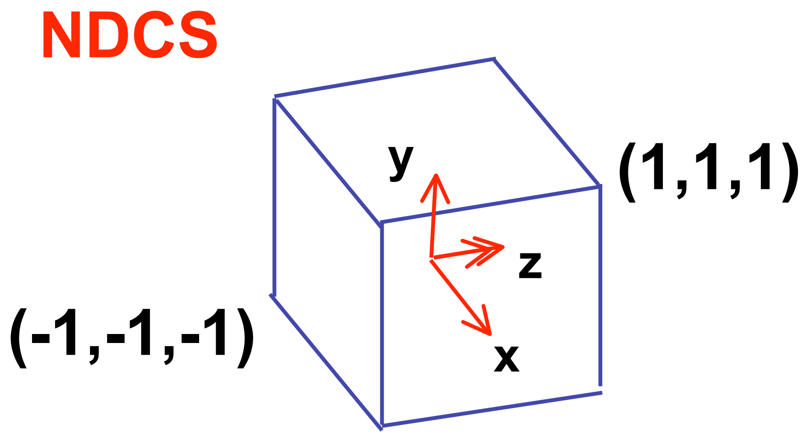




Review: Orthographic Derivation

- scale, translate, reflect for new coord sys





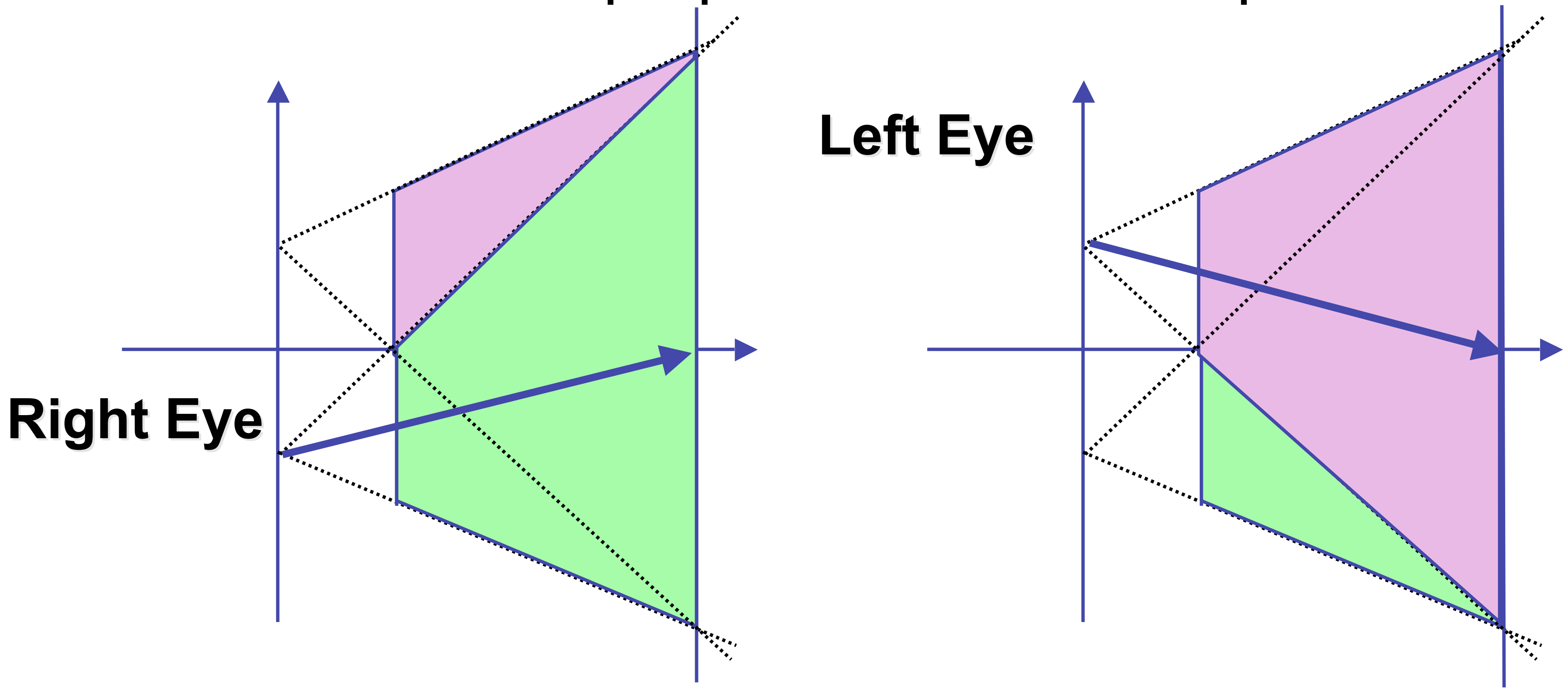
Review: Orthographic Derivation

- scale, translate, reflect for new coord sys

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

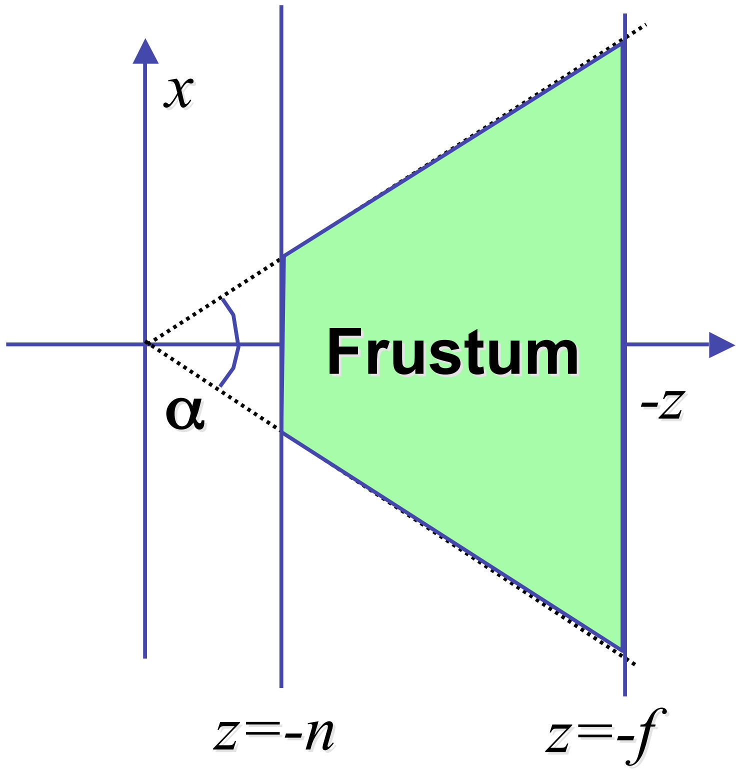
Review: Asymmetric Frusta

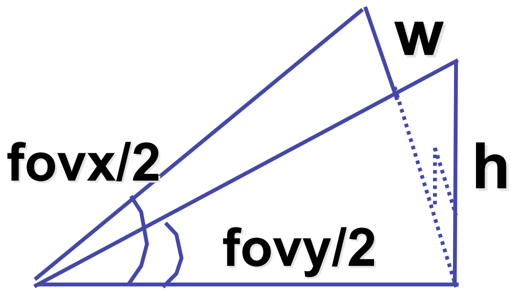
- our formulation allows asymmetry
  - why bother? binocular stereo
    - view vector not perpendicular to view plane



Review: Field-of-View Formulation

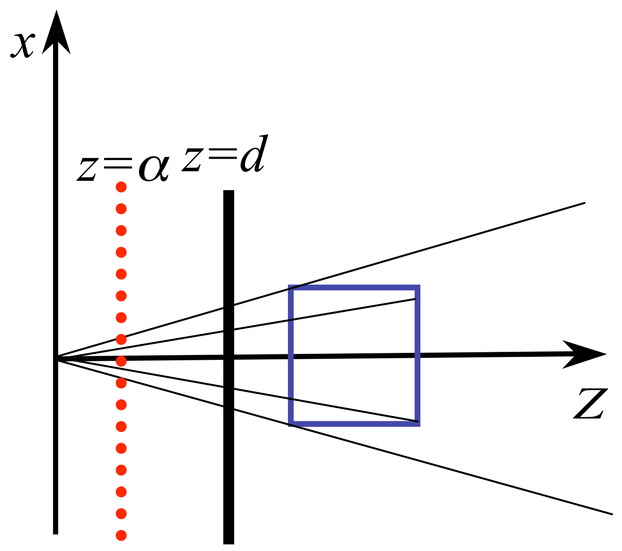
- FOV in one direction + aspect ratio (w/h)
  - determines FOV in other direction
  - also set near, far (reasonably intuitive)

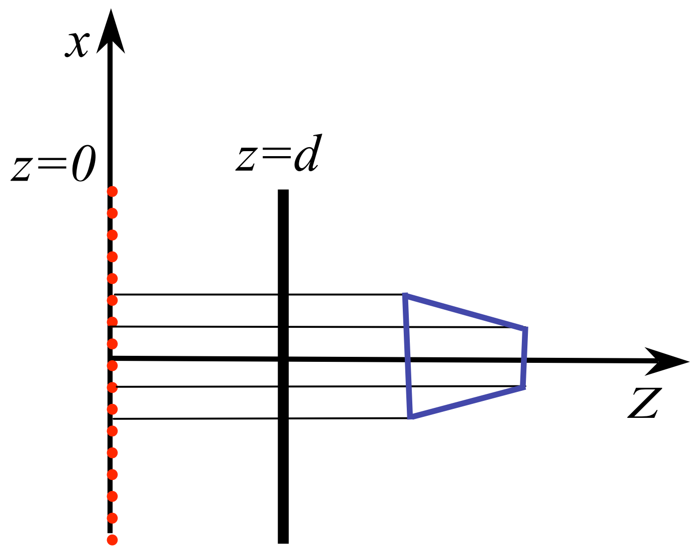




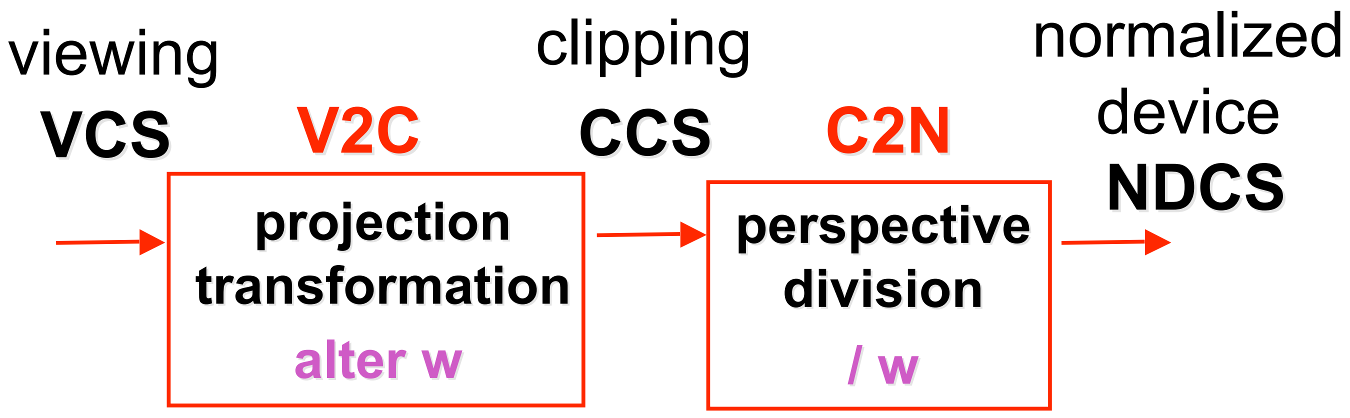
Review: Projection Normalization

- warp perspective view volume to orthogonal view volume
  - render all scenes with orthographic projection!
  - aka perspective warp





Review: Separate Warp From Homogenization

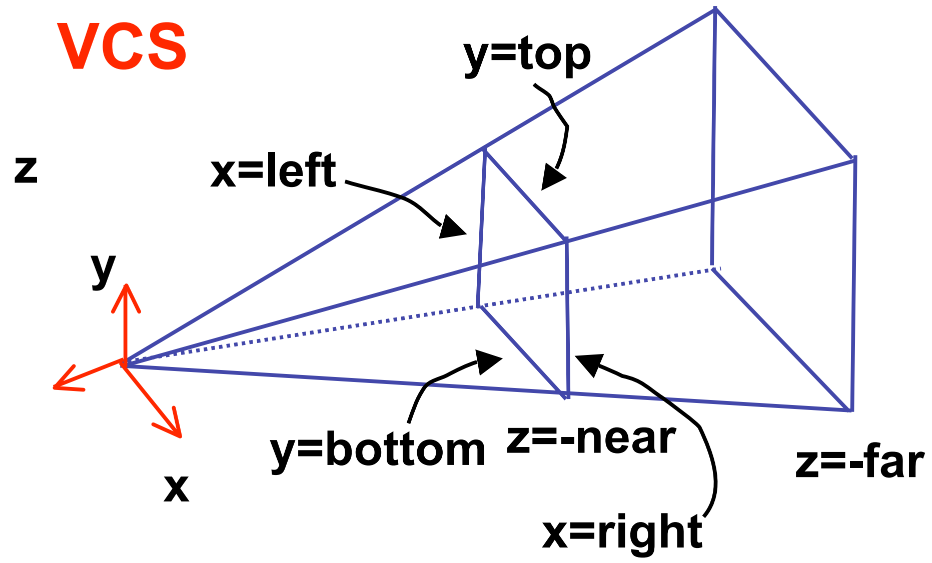


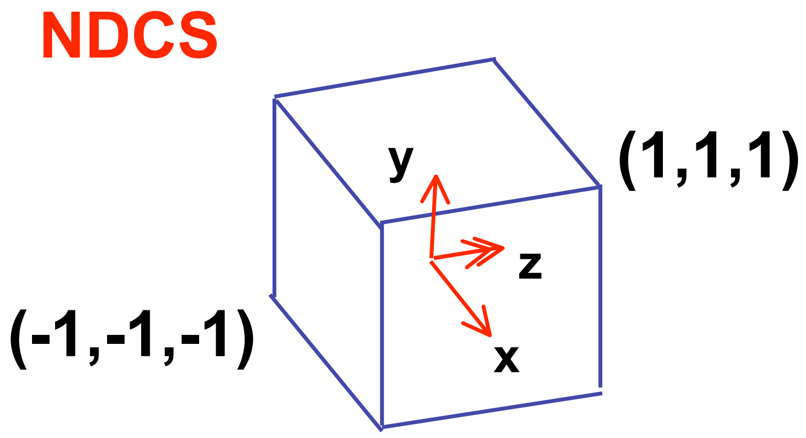
- warp requires only standard matrix multiply
  - distort such that orthographic projection of distorted objects is desired persp projection
    - w is changed
  - clip after warp, before divide
  - division by w: homogenization

Review: Perspective Derivation

- shear
- scale
- projection-normalization

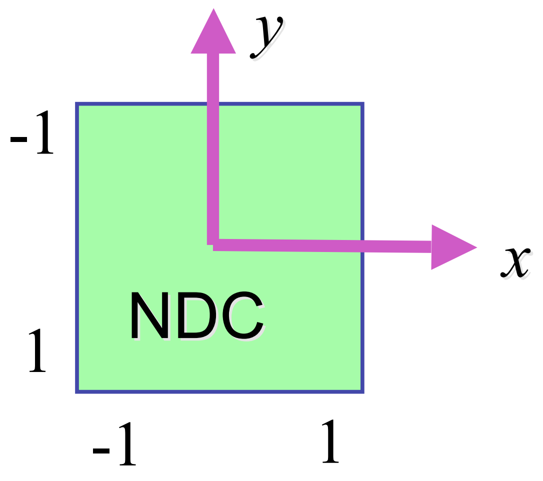
$$\begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

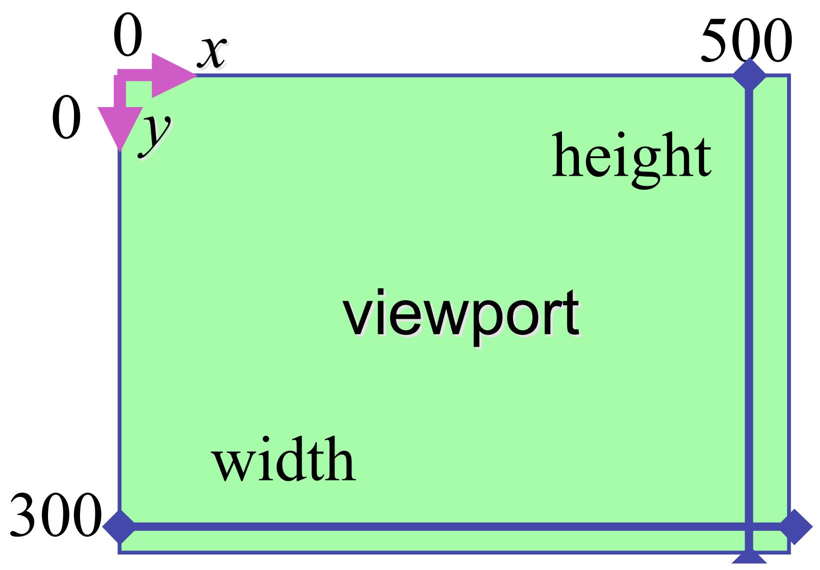




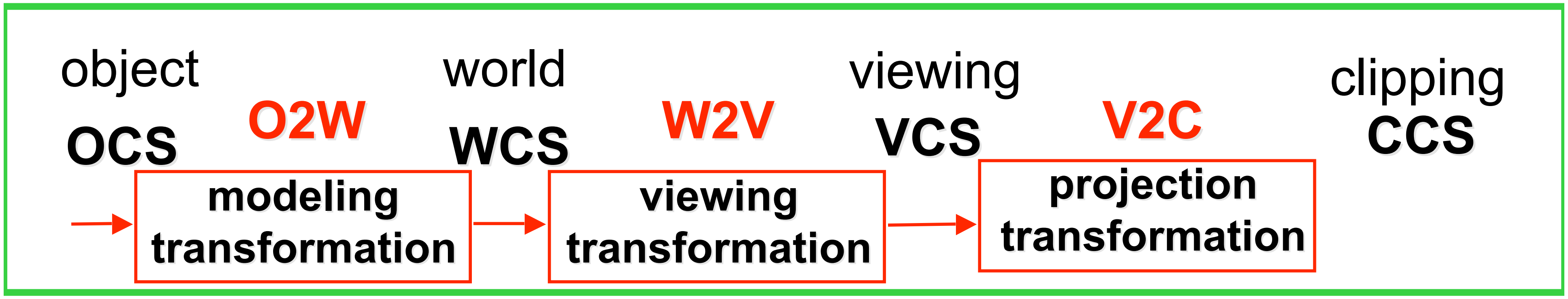
Review: N2D Transformation

$$\begin{bmatrix} x_D \\ y_D \\ z_D \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & \frac{width}{2} - \frac{1}{2} \\ 0 & 1 & 0 & \frac{height}{2} - \frac{1}{2} \\ 0 & 0 & 1 & \frac{depth}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{width}{2} & 0 & 0 & 0 \\ 0 & \frac{height}{2} & 0 & 0 \\ 0 & 0 & \frac{depth}{2} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_N \\ y_N \\ z_N \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{width(x_N+1)-1}{2} \\ \frac{height(-y_N+1)-1}{2} \\ \frac{depth(z_N+1)}{2} \\ 1 \end{bmatrix}$$





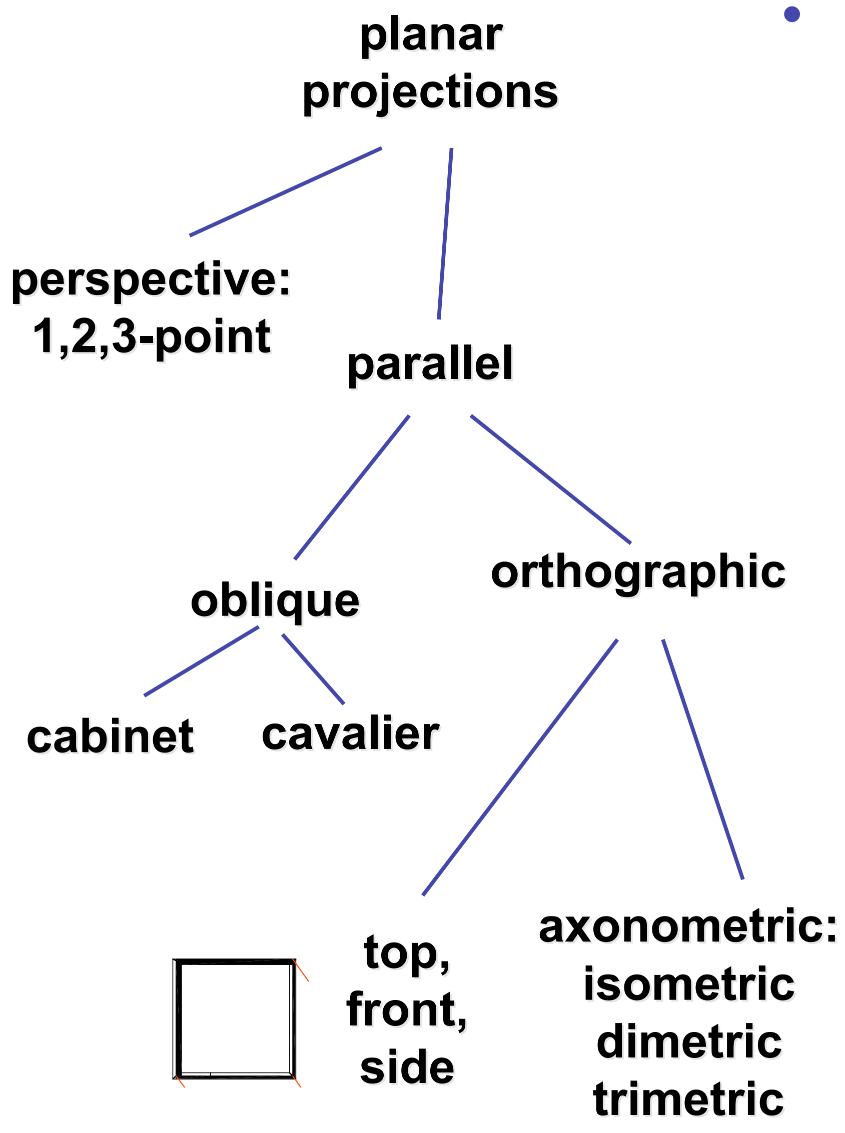
Review: OpenGL Example



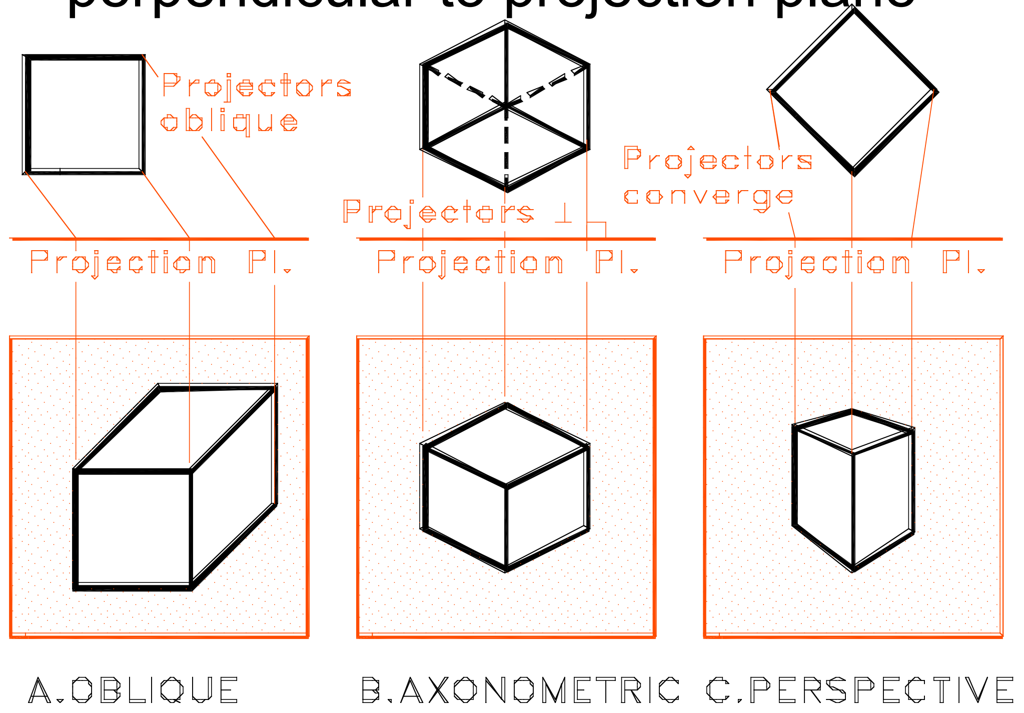
```
CCS   glMatrixMode( GL_PROJECTION );
      glLoadIdentity();
      gluPerspective( 45, 1.0, 0.1, 200.0 );
VCS   glMatrixMode( GL_MODELVIEW );
      glLoadIdentity();
      glTranslatef( 0.0, 0.0, -5.0 );
WCS   glPushMatrix()
      glTranslate( 4, 4, 0 ); W2O
OCS1  glutSolidTeapot(1);
      glPopMatrix();
      glTranslate( 2, 2, 0); W2O
OCS2  glutSolidTeapot(1);
```

- transformations that are applied first are specified last

Review: Projection Taxonomy



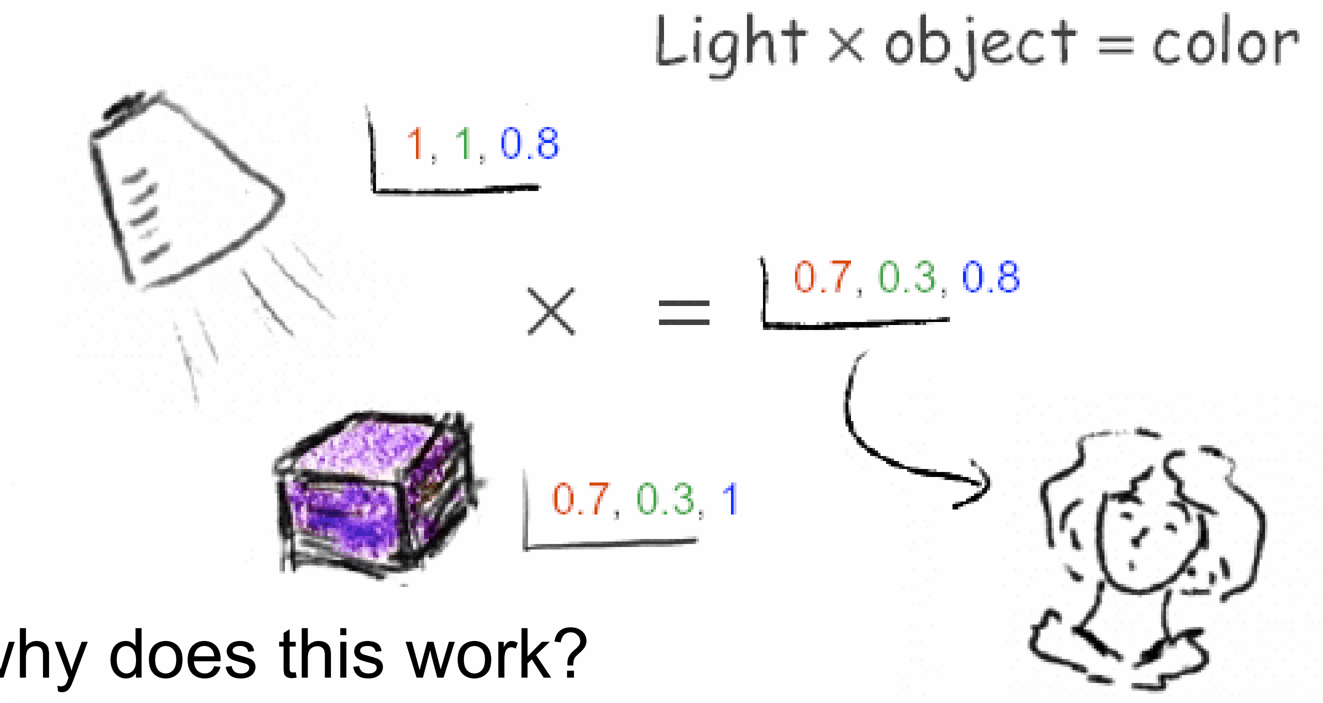
- perspective: projectors converge
  - orthographic, axonometric: projectors parallel and perpendicular to projection plane
  - oblique: projectors parallel, but not perpendicular to projection plane



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

Review: RGB Component Color

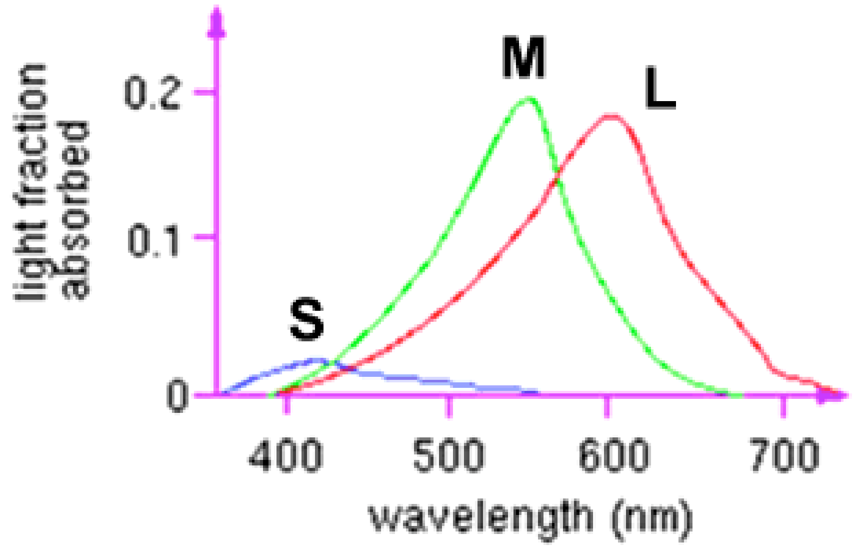
- simple model of color using RGB triples
- component-wise multiplication
  - (a0,a1,a2) * (b0,b1,b2) = (a0*b0, a1*b1, a2*b2)

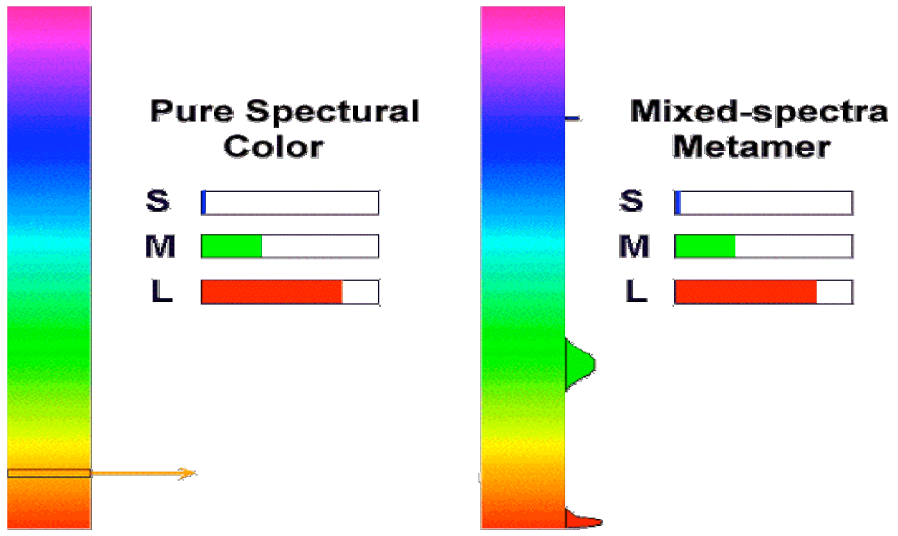


  - why does this work?
    - must dive into light, human vision, color spaces

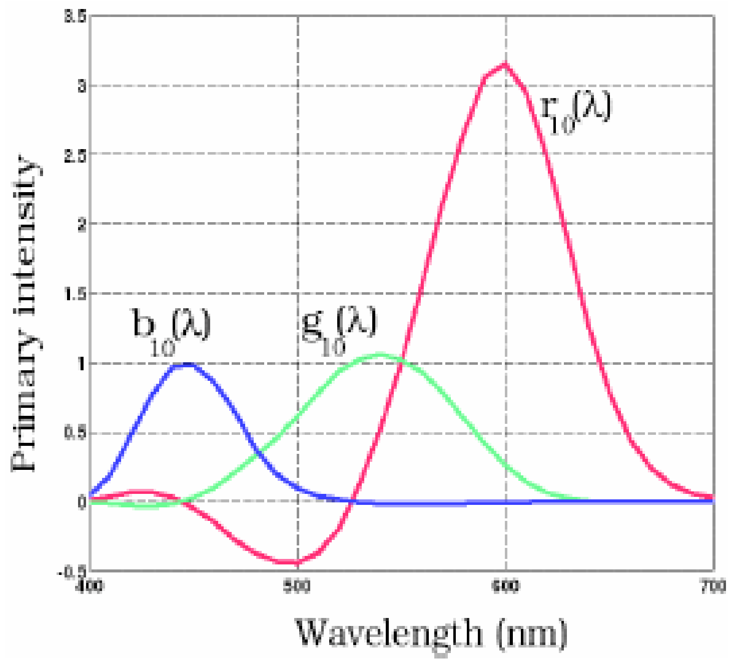
Review: Trichromacy and Metamers

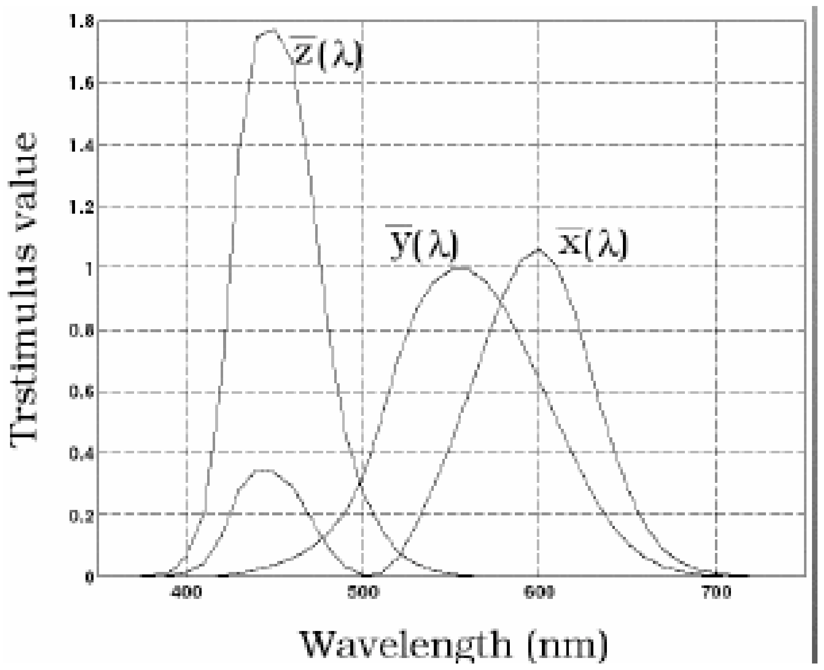
- three types of cones
- color is combination of cone stimuli
  - metamer: identically perceived color caused by very different spectra





Review: Measured vs. CIE Color Spaces



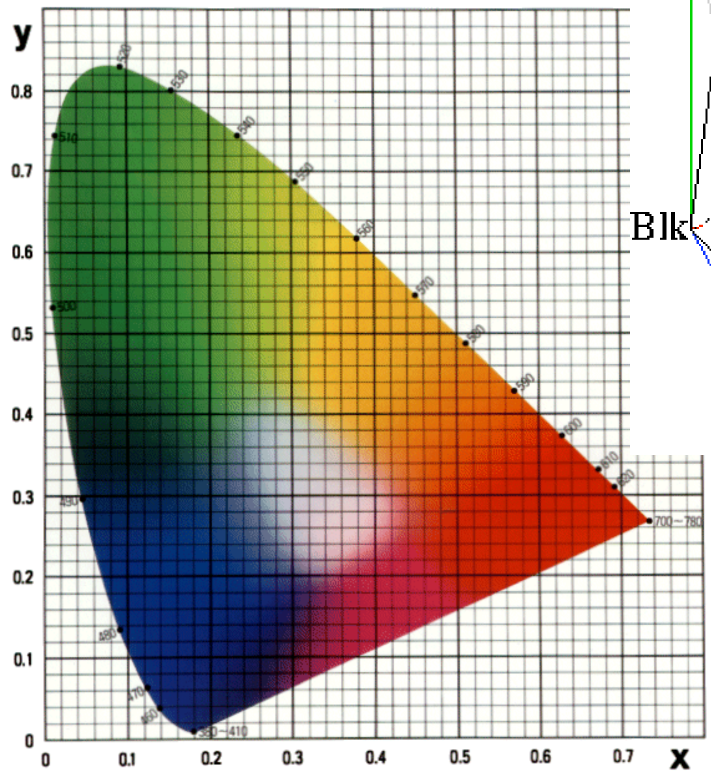


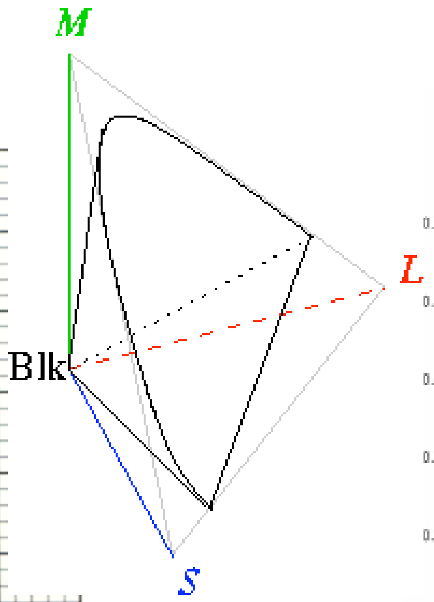
- measured basis
  - monochromatic lights
  - physical observations
  - negative lobes

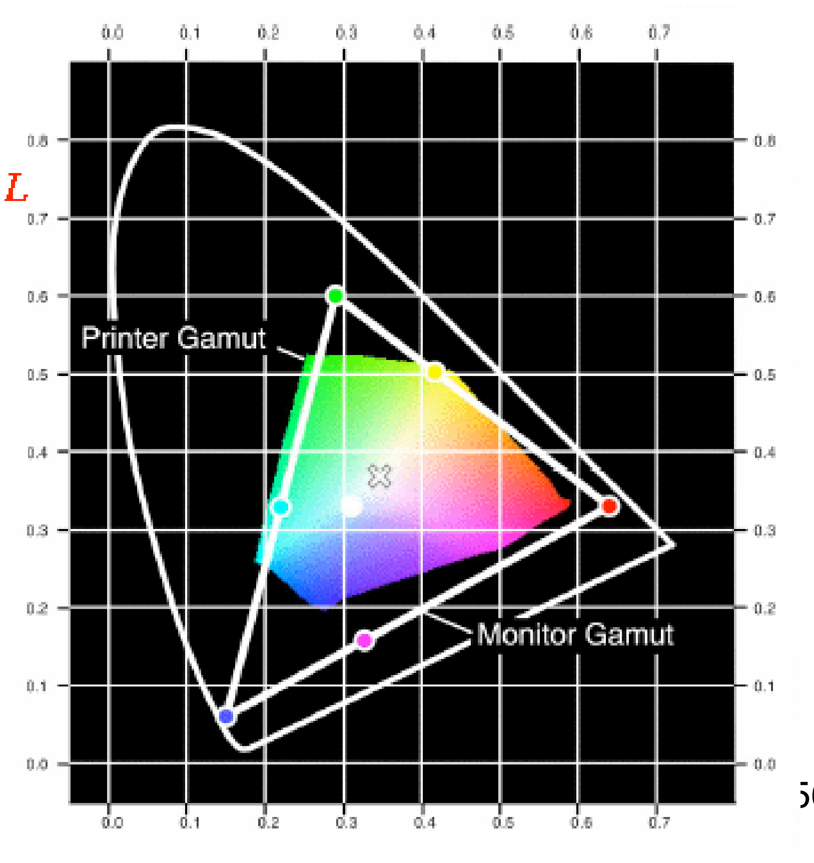
- transformed basis
  - “imaginary” lights
  - all positive, unit area
  - Y is luminance

Review: Chromaticity Diagram and Gamuts

- plane of equal brightness showing chromaticity
- gamut is polygon, device primaries at corners
  - defines reproducible color range

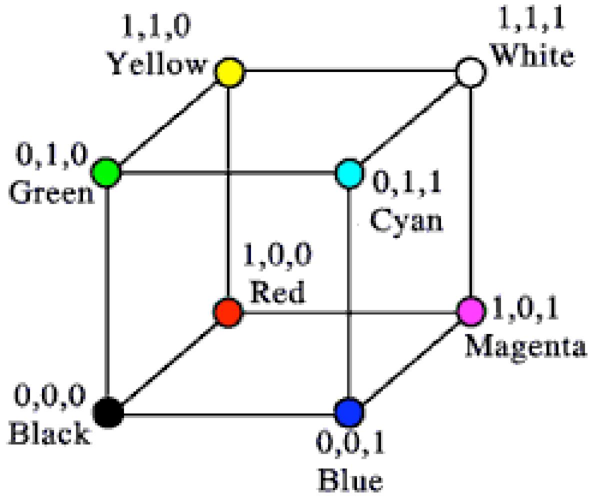




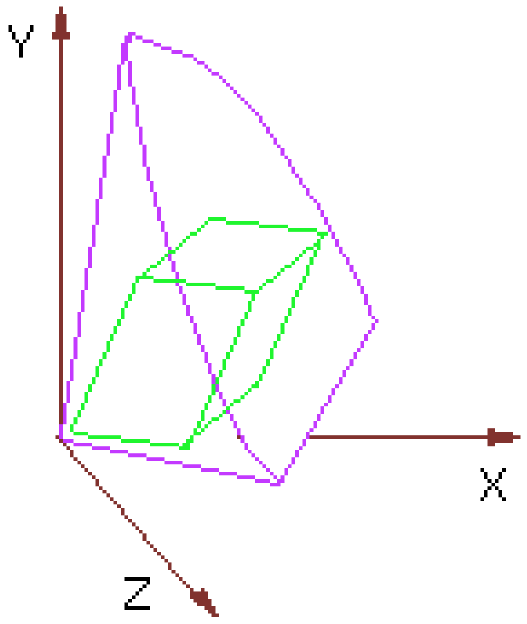


Review: RGB Color Space (Color Cube)

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

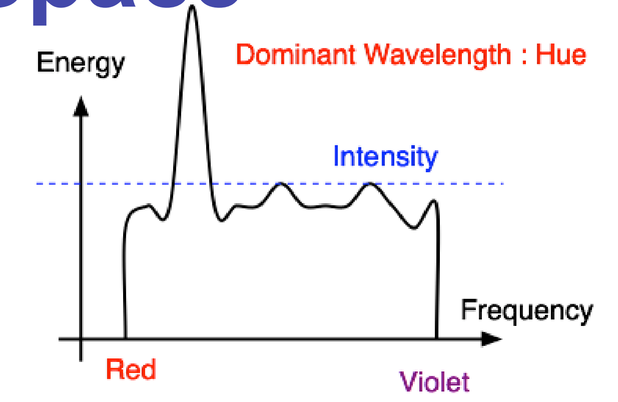


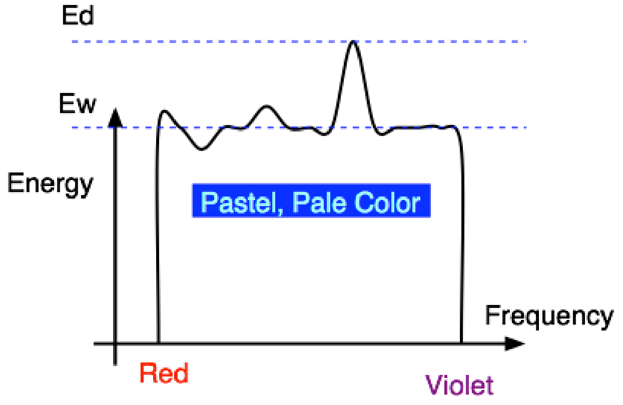
- RGB color cube sits within CIE color space
  - subset of perceivable colors
  - scale, rotate, shear cube

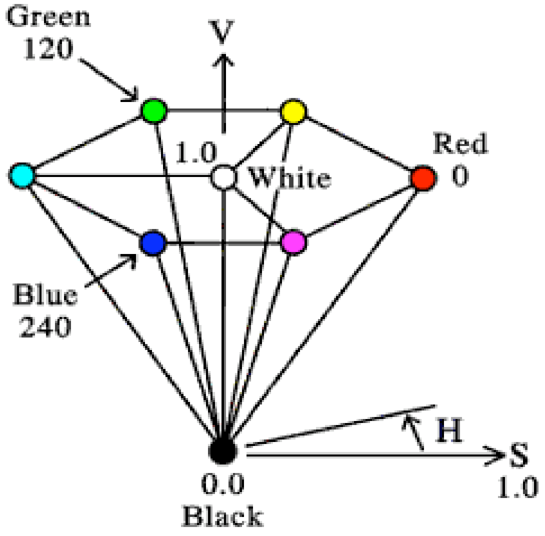


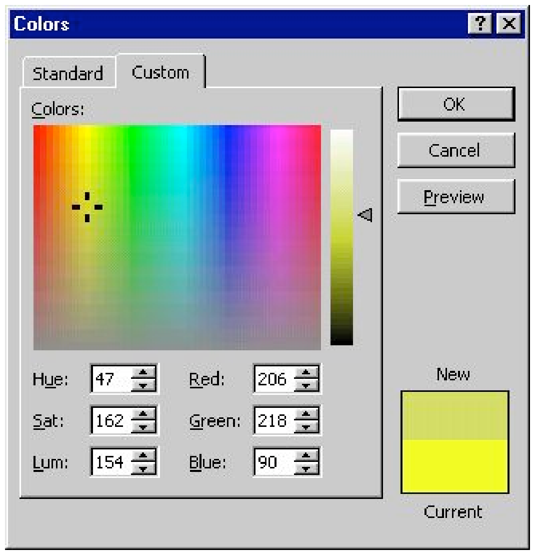
Review: HSV Color Space

- hue: dominant wavelength, “color”
- saturation: how far from grey
- value/brightness: how far from black/white
- cannot convert to RGB with matrix alone

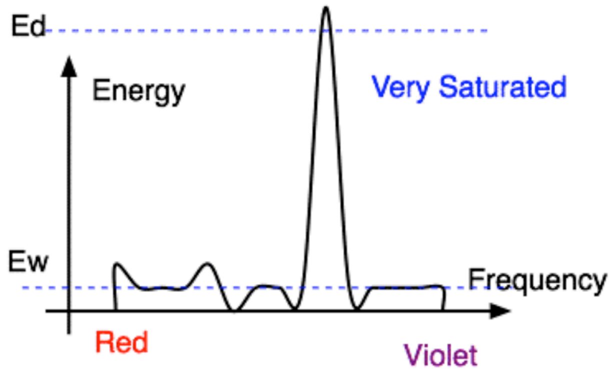








Hue:	47	Red:	206
Sat:	162	Green:	218
Lum:	154	Blue:	90



Correction: HSI/HSV and RGB

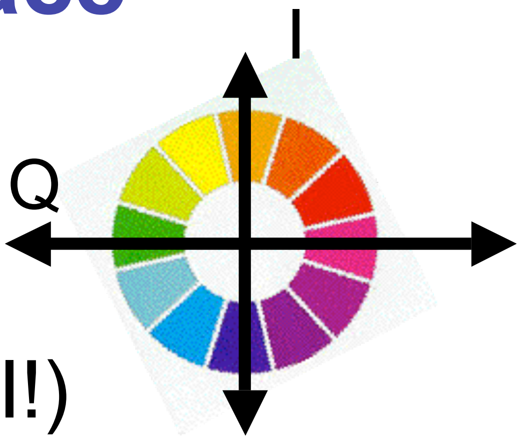
- HSV/HSI conversion from RGB
  - hue same in both
  - value is max, intensity is average

$$H = \cos^{-1}\left[\frac{\frac{1}{2}\left[(R-G)+(R-B)\right]}{\sqrt{(R-G)^2+(R-B)(G-B)}}\right] \quad \text{if (B > G),} \quad H = 360 - H$$

•HSI: $S = 1 - \frac{\min(R,G,B)}{I}$ $\quad I = \frac{R+G+B}{3}$

•HSV: $S = 1 - \frac{\min(R,G,B)}{V}$ $\quad V = \max(R,G,B)$

Review: YIQ Color Space



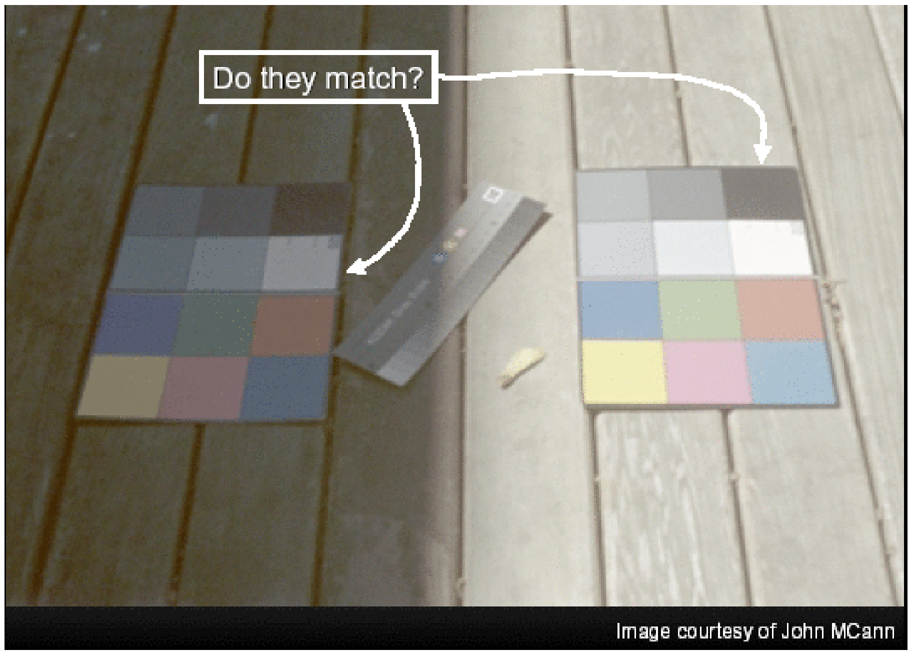
- color model used for color TV
  - Y is luminance (same as CIE)
  - I & Q are color (not same I as HSI!)
  - using Y backwards compatible for B/W TVs
  - conversion from RGB is linear

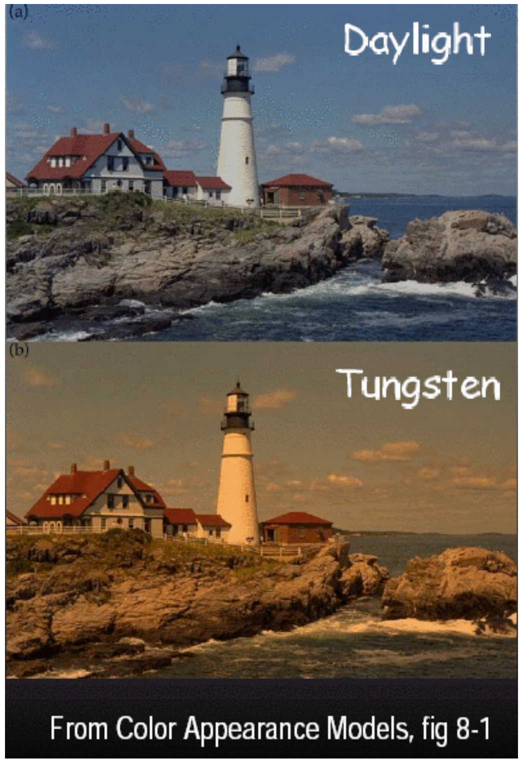
$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.30 & 0.59 & 0.11 \\ 0.60 & -0.28 & -0.32 \\ 0.21 & -0.52 & 0.31 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

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

Review: Color Constancy

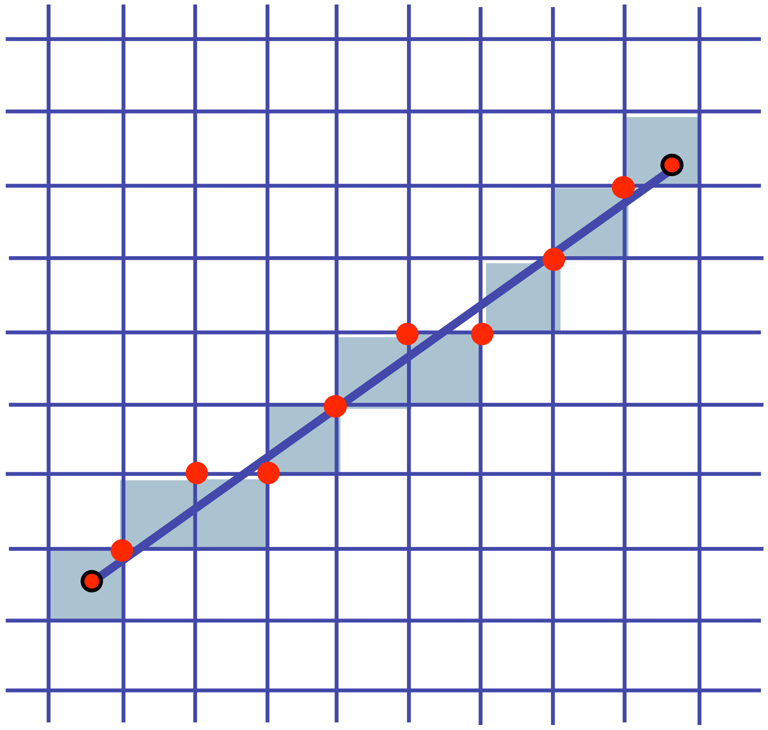
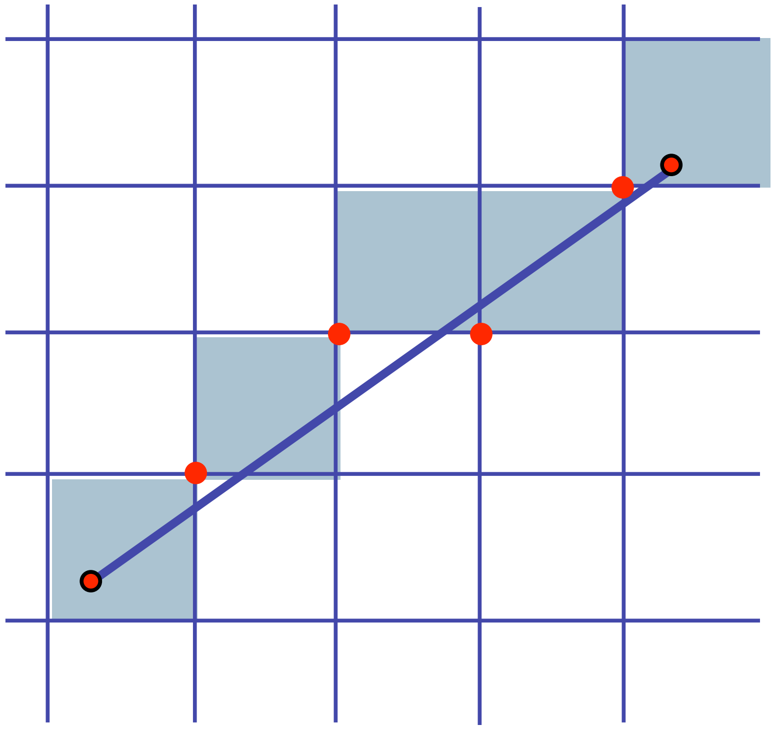
- automatic “white balance” from change in illumination
- vast amount of processing behind the scenes!
- colorimetry vs. perception





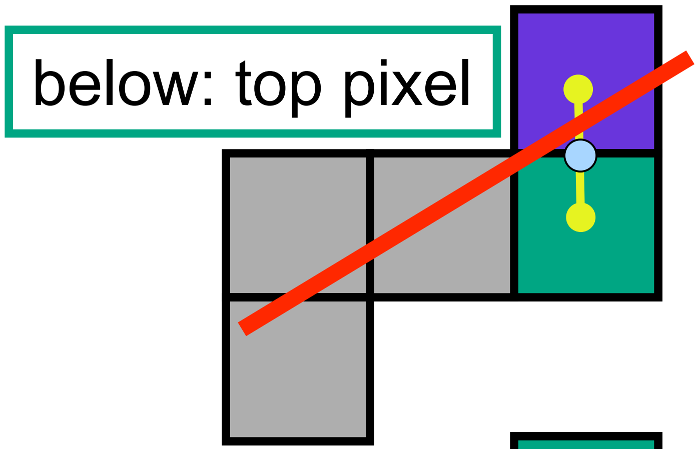
Review: Scan Conversion

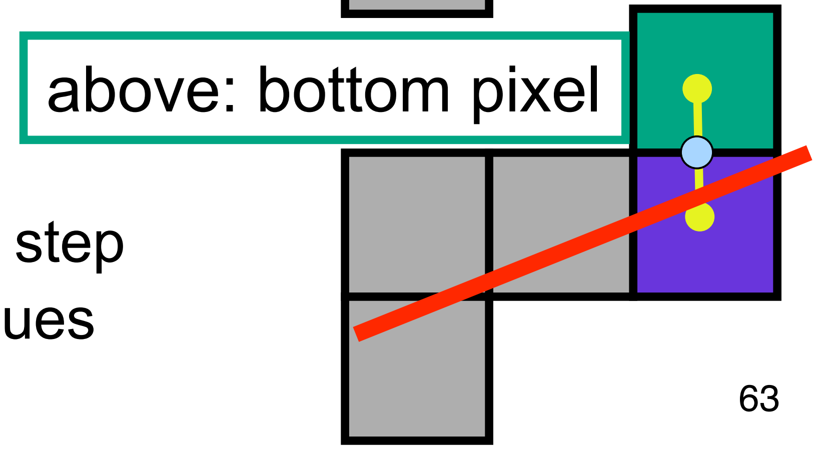
- convert continuous rendering primitives into discrete fragments/pixels
  - given vertices in DCS, fill in the pixels
- display coordinates required to provide scale for discretization

Review: Midpoint Algorithm

- we're moving horizontally along x direction (first octant)
  - only two choices: 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: pick top pixel
  - above: pick bottom pixel
- key idea behind Bresenham
  - reuse computation from previous step
  - integer arithmetic by doubling values





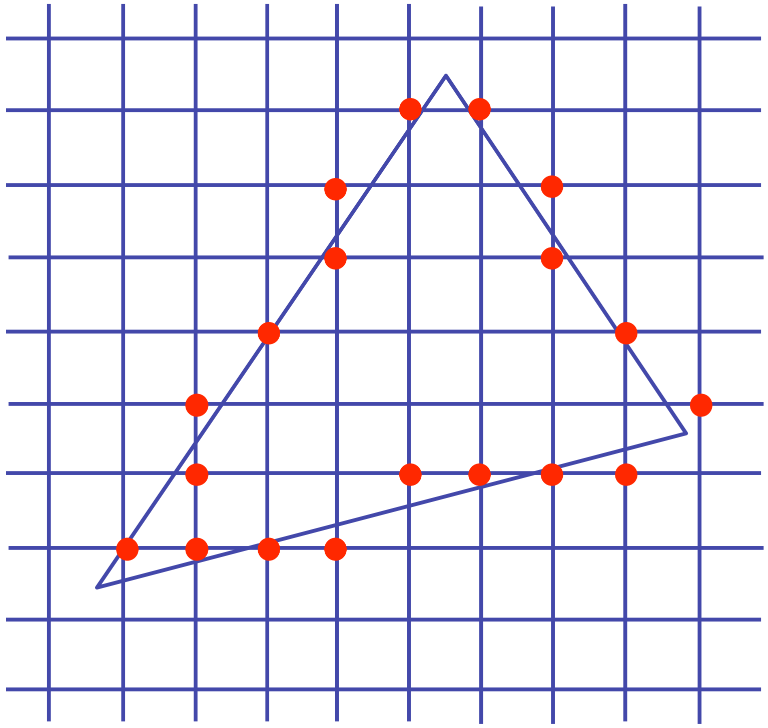
Bresenham's Line Rasterization Algorithm

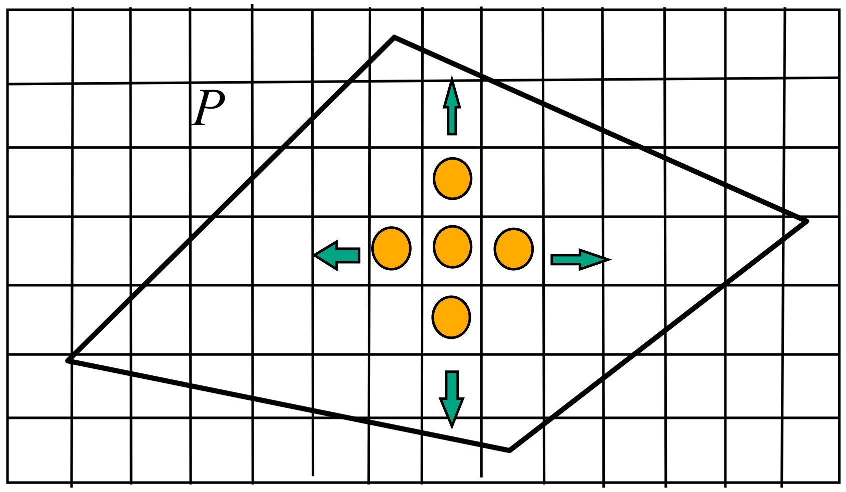
- use error term, integer only

```
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)
}
```

Review: Flood Fill

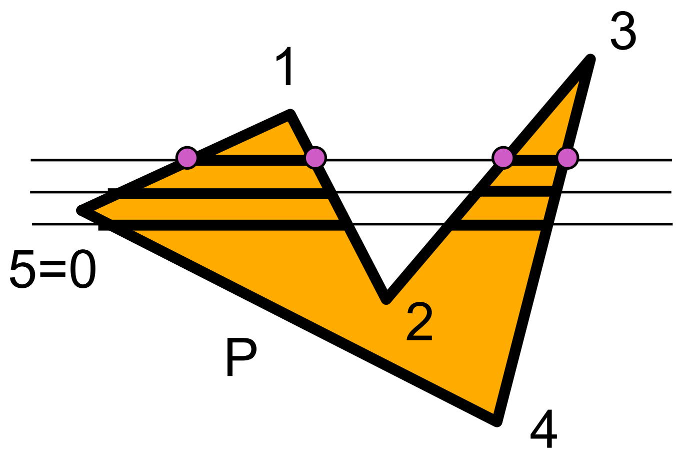
- simple algorithm
  - draw edges of polygon
  - use flood-fill to draw interior

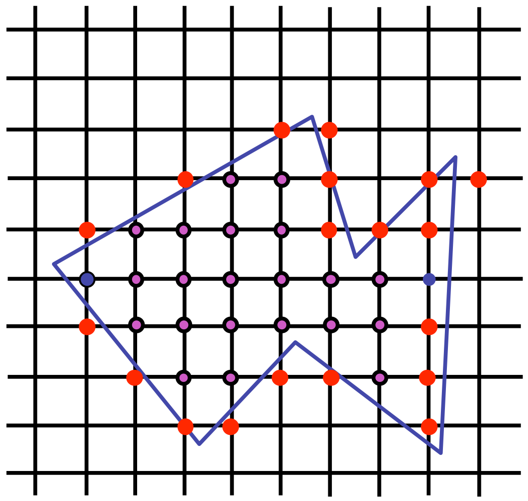


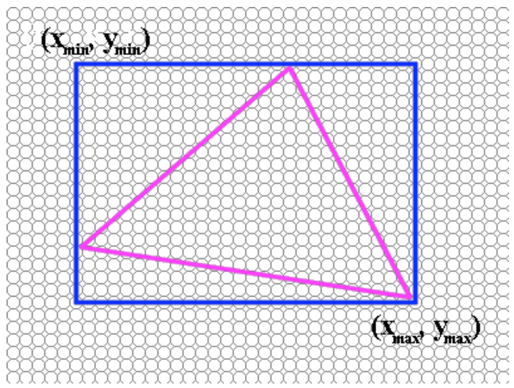


Review: Scanline Algorithms

- scanline: a line of pixels in an image
  - set pixels inside polygon boundary along horizontal lines one pixel apart vertically
    - parity test: draw pixel if edgecount is odd
    - optimization: only loop over axis-aligned bounding box of xmin/xmax, ymin/ymax

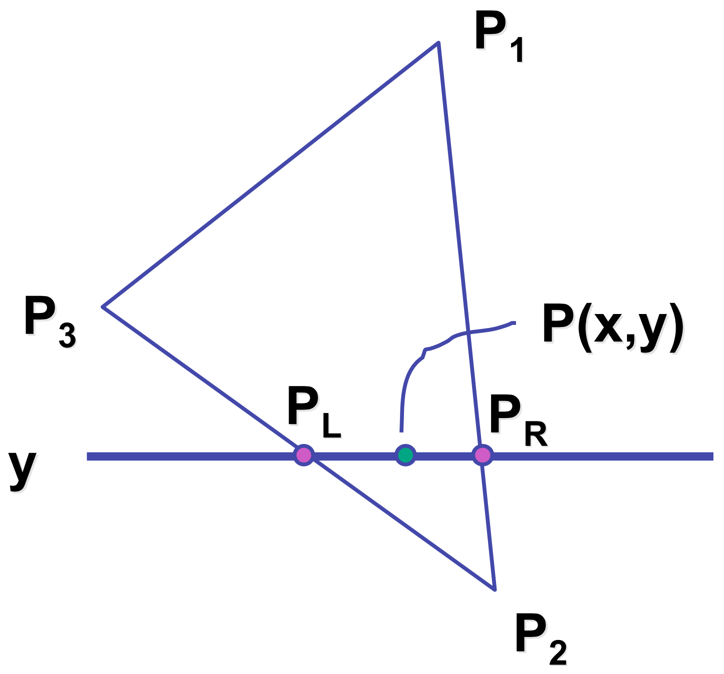






Review: Bilinear Interpolation

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



Review: Barycentric Coordinates

- non-orthogonal coordinate system based on triangle itself
  - origin: P_1, basis vectors: (P_2-P_1) and (P_3-P_1)

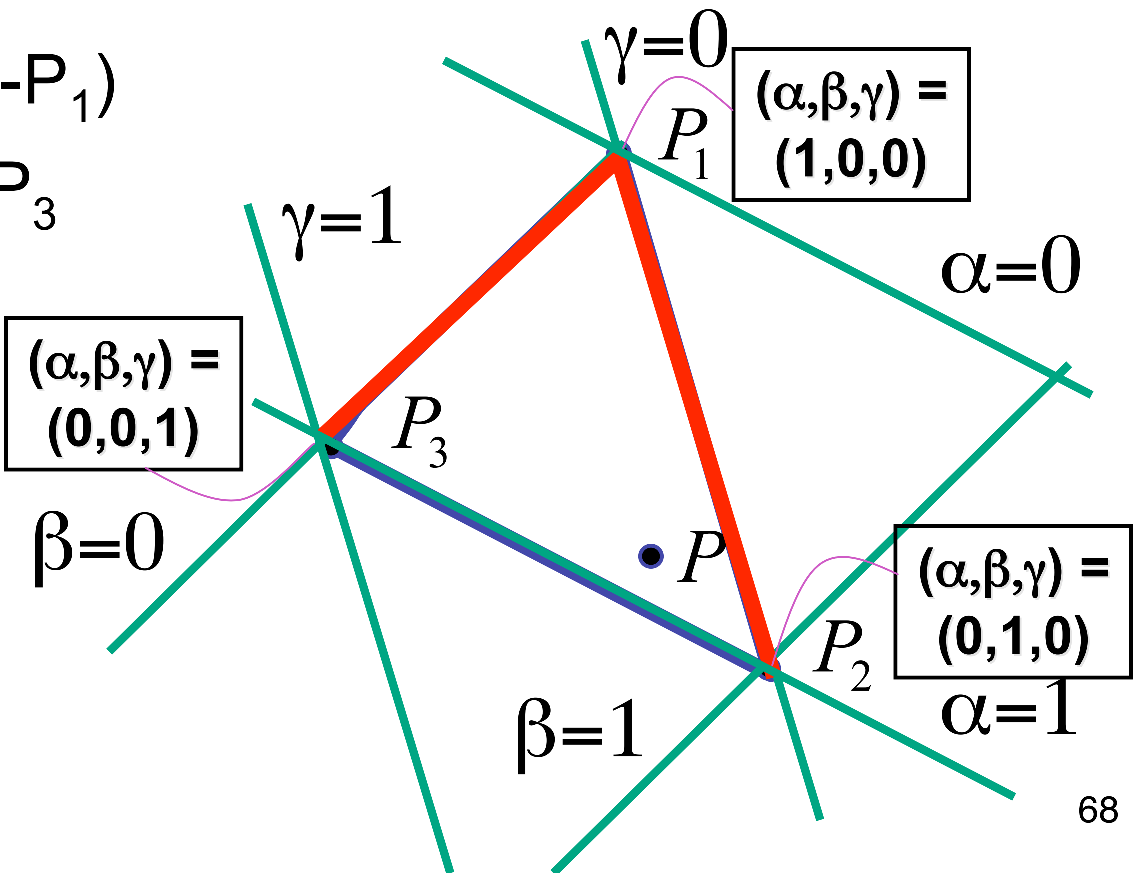
$P = P_1 + \beta(P_2-P_1)+\gamma(P_3-P_1)$

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

$P = \alpha P_1 + \beta P_2+\gamma P_3$

$\alpha + \beta + \gamma = 1$

$0 <= \alpha, \beta, \gamma <= 1$



$\gamma=0$
$(\alpha,\beta,\gamma) = (1,0,0)$
P_1
$\gamma=1$
$\alpha=0$
$(\alpha,\beta,\gamma) = (0,0,1)$
P_3
$\beta=0$
P
$(\alpha,\beta,\gamma) = (0,1,0)$
P_2
$\alpha=1$
$\beta=1$

Review: Computing Barycentric Coordinates

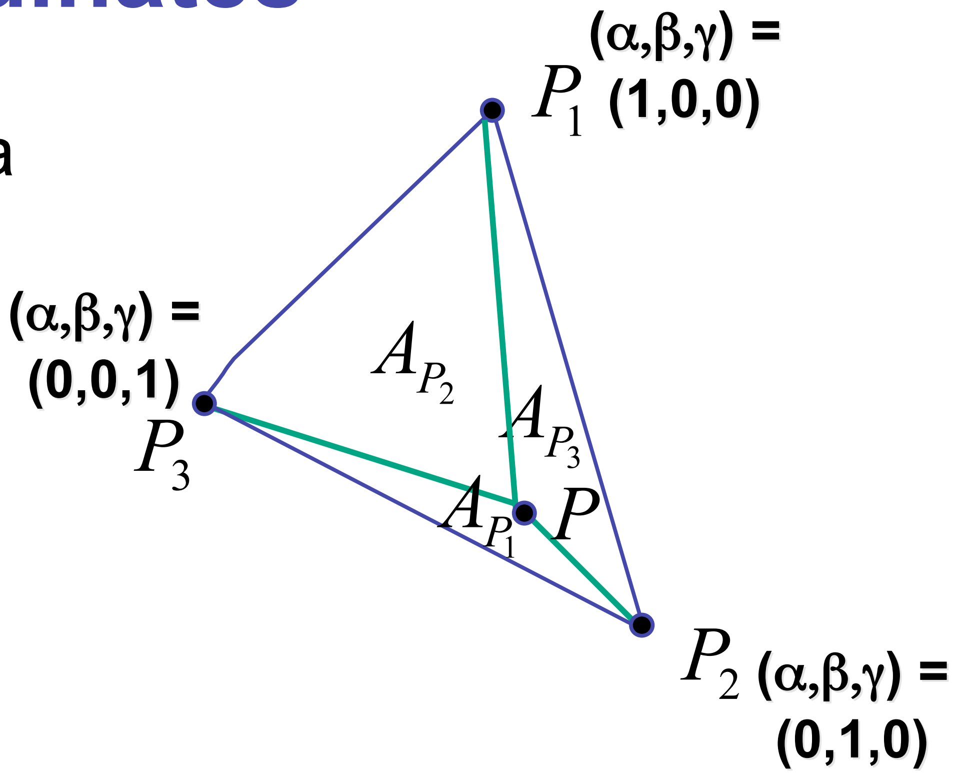
- 2D triangle area
  - half of parallelogram area
    - from cross product

$A = A_{P1} + A_{P2} + A_{P3}$

$\alpha = A_{P1} / A$

$\beta = A_{P2} / A$

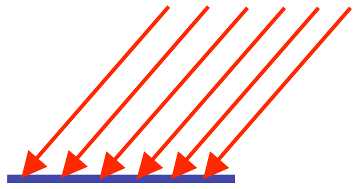
$\gamma = A_{P3} / A$



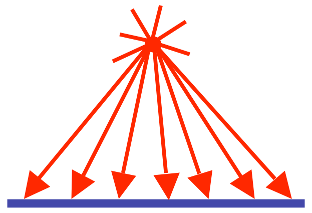
weighted combination of three points
[demo]

Review: Light Sources

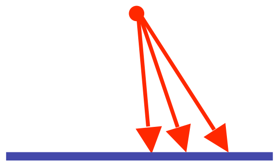
- directional/parallel lights
  - point at infinity: $(x,y,z,0)^T$



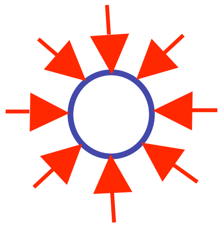
- point lights
  - finite position: $(x,y,z,1)^T$



- spotlights
  - position, direction, angle



- ambient lights

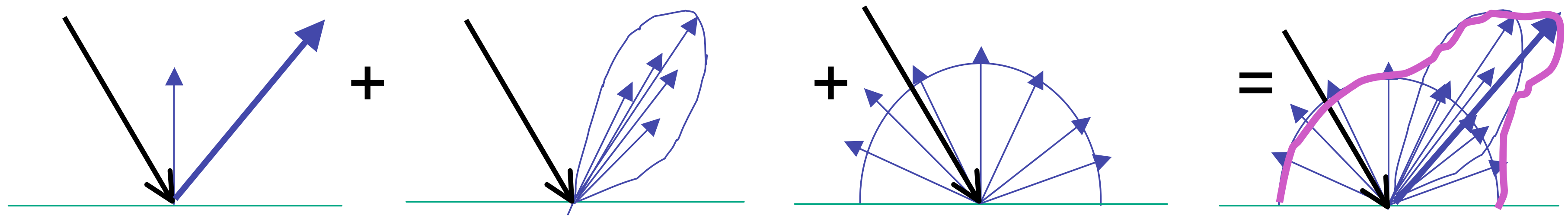


Review: Light Source Placement

- geometry: positions and directions
  - standard: world coordinate system
    - effect: lights fixed wrt world geometry
  - alternative: camera coordinate system
    - effect: lights attached to camera (car headlights)

Review: Reflectance

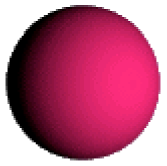
- *specular*: perfect mirror with no scattering
- *gloss*: mixed, partial specularity
- *diffuse*: all directions with equal energy

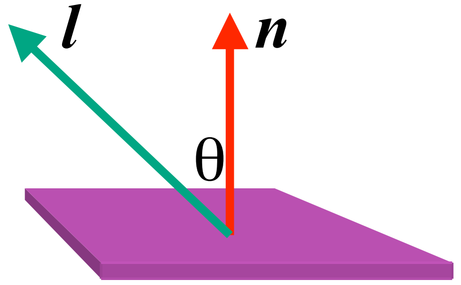


specular + glossy + diffuse =
reflectance distribution

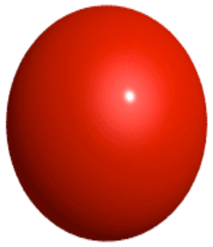
Review: Reflection Equations

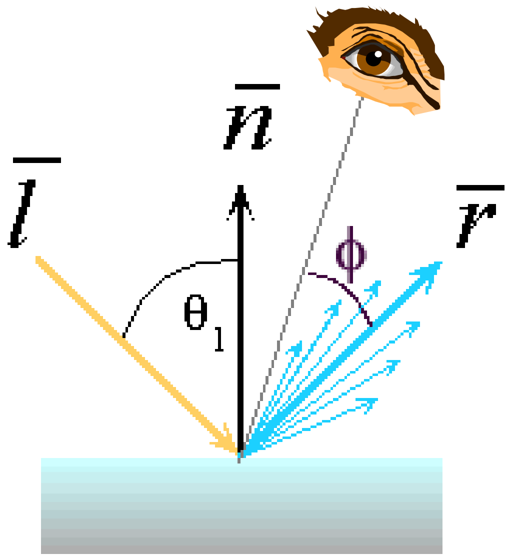
$$I_{diffuse} = k_d \, I_{light} \, (n \bullet l)$$





$$I_{specular} = k_s I_{light} (v \bullet r)^{n_{shiny}}$$

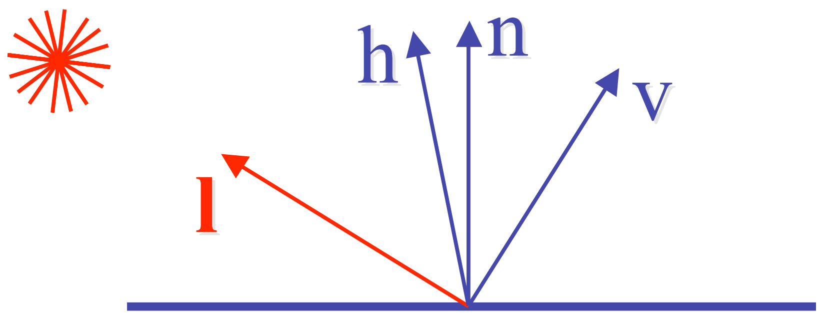




$$2\,(\mathbf{N}\,(\mathbf{N} \cdot \mathbf{L})) - \mathbf{L} = \mathbf{R}$$

Review: Reflection Equations

- Blinn improvement



$$\mathbf{I}_{\mathbf{specular}} = \mathbf{k}_{\mathbf{s}}\mathbf{I}_{\mathbf{light}}(\mathbf{h} \bullet \mathbf{n})^{n_{shiny}}$$

$$\mathbf{h} = (\mathbf{l} + \mathbf{v})/2$$

- full Phong lighting model
  - combine ambient, diffuse, specular components

$$\mathbf{I}_{\mathbf{total}} = \mathbf{k}_{\mathbf{a}}\mathbf{I}_{\mathbf{ambient}} + \sum_{i=1}^{\#\,lights} \mathbf{I}_{\mathbf{i}}(\mathbf{k}_{\mathbf{d}}(\mathbf{n} \bullet \mathbf{l}_{\mathbf{i}}) + \mathbf{k}_{\mathbf{s}}(\mathbf{v} \bullet \mathbf{r}_{\mathbf{i}})^{n_{shiny}})$$

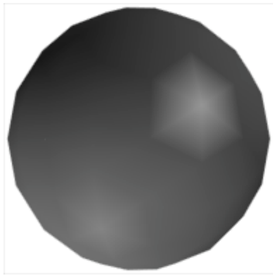
  - don’t forget to normalize all vectors: n,l,r,v,h

Review: Lighting

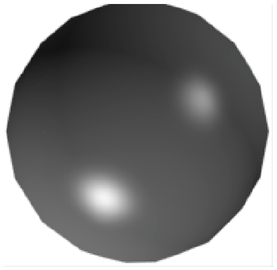
- lighting models
  - ambient
    - normals don’t matter
  - Lambert/diffuse
    - angle between surface normal and light
  - Phong/specular
    - surface normal, light, and viewpoint

Review: Shading Models

- flat shading
  - compute Phong lighting once for entire polygon



- Gouraud shading
  - compute Phong lighting at the vertices and interpolate lighting values across polygon
- Phong shading
  - compute averaged vertex normals
  - interpolate normals across polygon and perform Phong lighting across polygon



Review: Specifying Normals

- OpenGL state machine
  - uses last normal specified
  - if no normals specified, assumes all identical
- per-vertex normals

```
glNormal3f(1,1,1);
glVertex3f(3,4,5);
glNormal3f(1,1,0);
glVertex3f(10,5,2);
```

- per-face normals

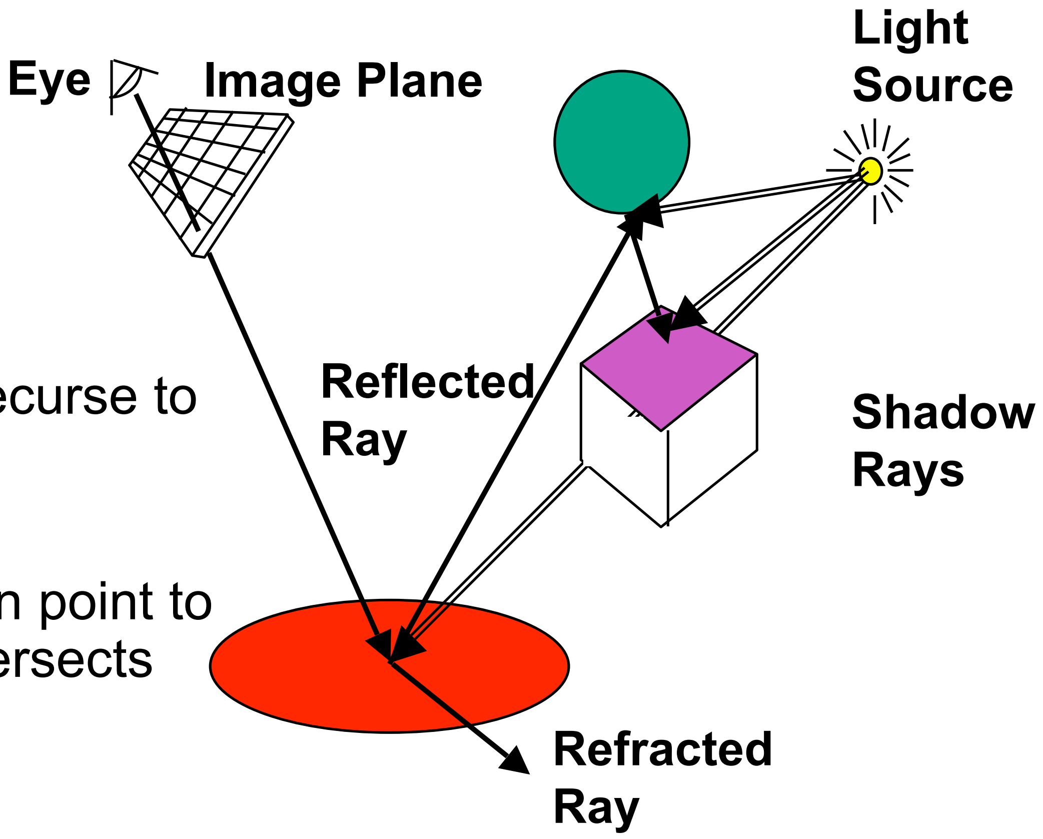
```
glNormal3f(1,1,1);
glVertex3f(3,4,5);
glVertex3f(10,5,2);
```

- normal interpreted as direction from vertex location
- can automatically normalize (computational cost)

```
glEnable(GL_NORMALIZE);
```

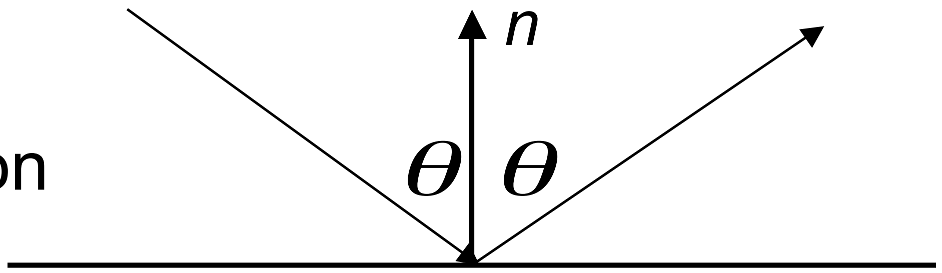
Review: Recursive Ray Tracing

- ray tracing can handle
  - reflection (chrome/mirror)
  - refraction (glass)
  - shadows
- one primary ray per pixel
- spawn secondary rays
  - reflection, refraction
    - if another object is hit, recurse to find its color
  - shadow
    - cast ray from intersection point to light source, check if intersects another object
  - termination criteria
    - no intersection (ray exits scene)
    - max bounces (recursion depth)
    - attenuated below threshold



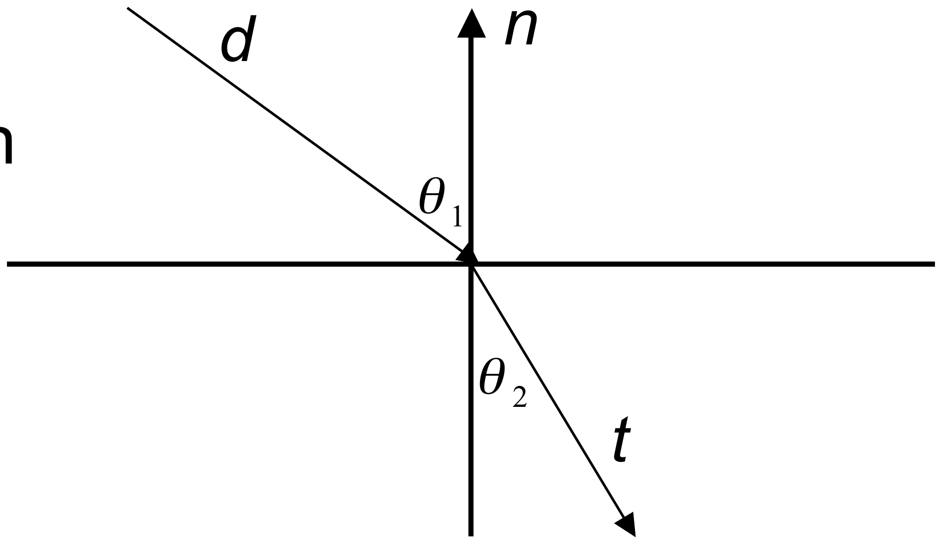
Review: Reflection and Refraction

- refraction: mirror effects
  - perfect specular reflection



- refraction: at boundary
- Snell's Law
  - light ray bends based on refractive indices c_1, c_2

$$c_1 \sin\theta_1 = c_2 \sin\theta_2$$

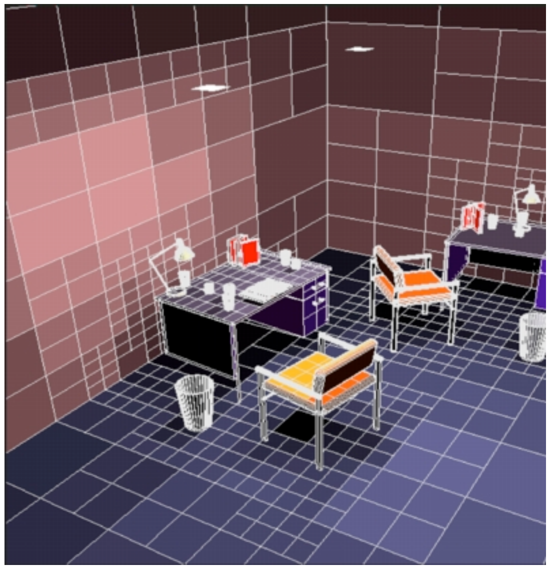


Review: Ray Tracing

- issues:
  - generation of rays
  - intersection of rays with geometric primitives
  - geometric transformations
  - lighting and shading
  - efficient data structures so we don’t have to test intersection with *every* object

Review: Radiosity

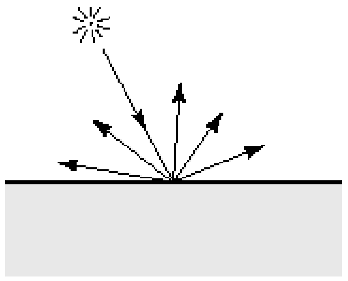
- capture indirect diffuse-diffuse light exchange
- model light transport as flow with conservation of energy until convergence
  - view-independent, calculate for whole scene then browse from any viewpoint
- divide surfaces into small patches
- loop: check for light exchange between all pairs
  - form factor: orientation of one patch wrt other patch (n x n matrix)

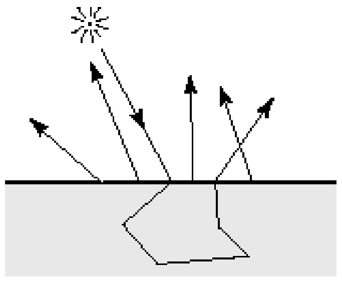

escience.anu.edu.au/lecture/cg/GlobalIllumination/Image/discrete.jpg

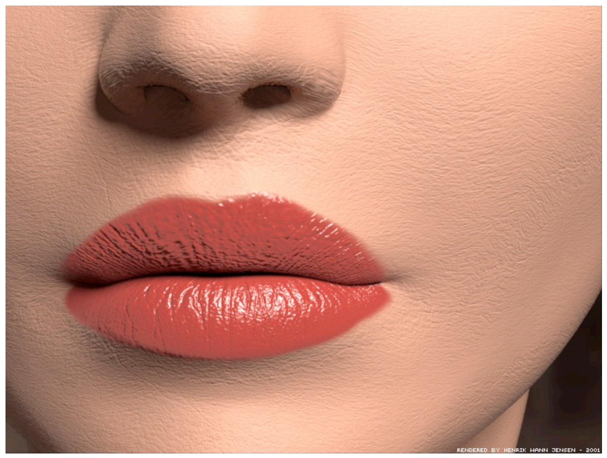
escience.anu.edu.au/lecture/cg/GlobalIllumination/Image/continuous.jpg

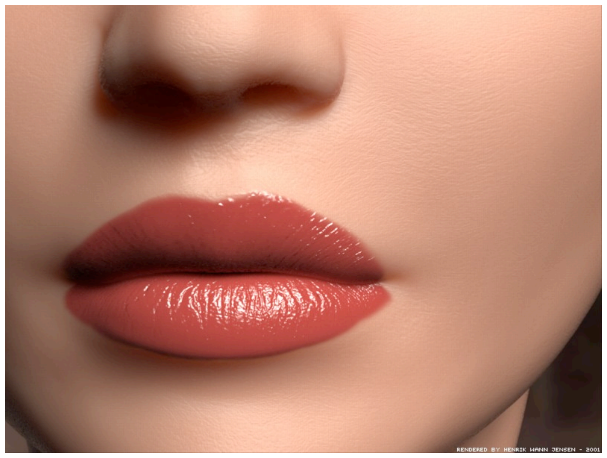
Review: Subsurface Scattering

- light enters and leaves at *different* locations on the surface
    - bounces around inside
- technical Academy Award, 2003
    - Jensen, Marschner, Hanrahan



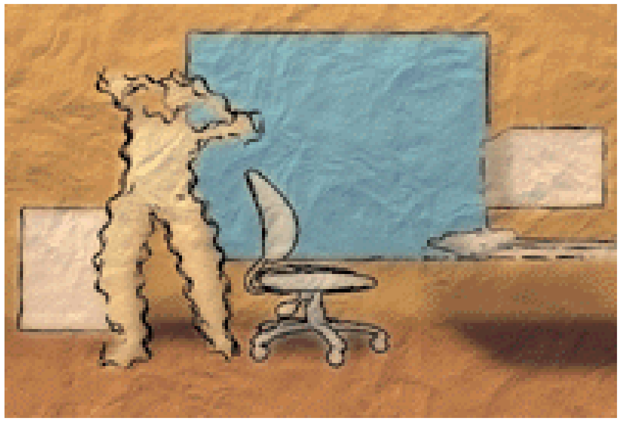
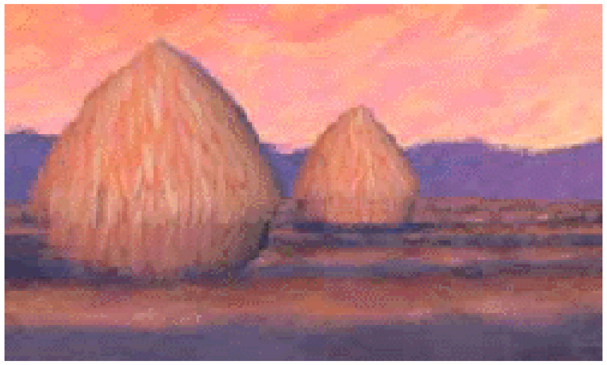






Review: Non-Photorealistic Rendering

- simulate look of hand-drawn sketches or paintings, using digital models

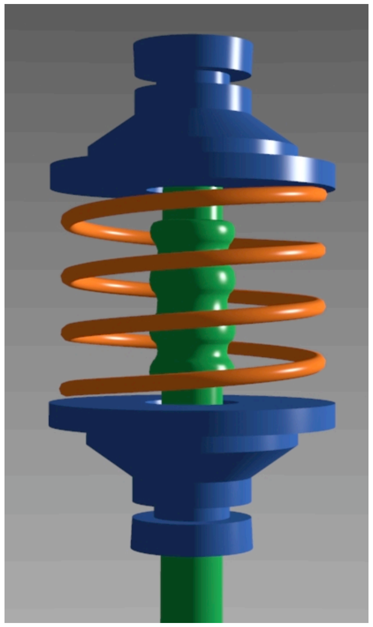



www.red3d.com/cwr/npr/

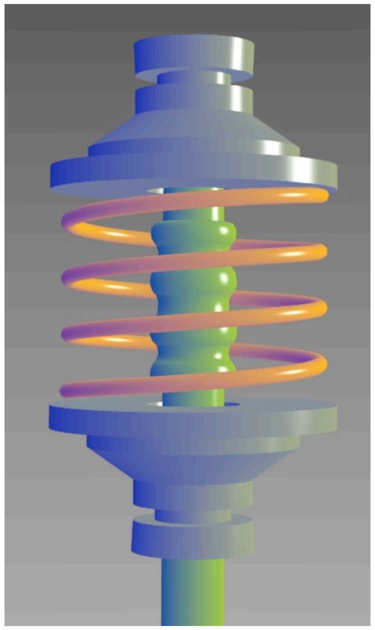
Review: Non-Photorealistic Shading

- cool-to-warm shading: $k_w = \dfrac{1 + \mathbf{n} \cdot \mathbf{l}}{2}, c = k_w c_w + (1 - k_w) c_c$
- draw silhouettes: if $(\mathbf{e} \cdot \mathbf{n_0})(\mathbf{e} \cdot \mathbf{n_1}) \leq 0$, **e**=edge-eye vector
- draw creases: if $(\mathbf{n_0} \cdot \mathbf{n_1}) \leq \mathit{threshold}$

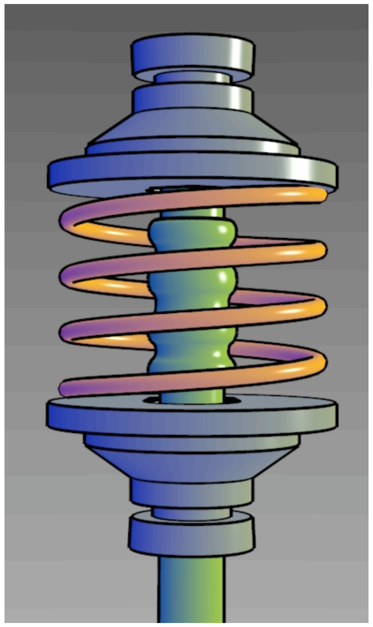
standard



cool-to-warm



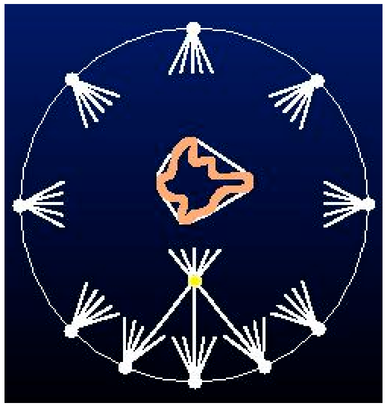
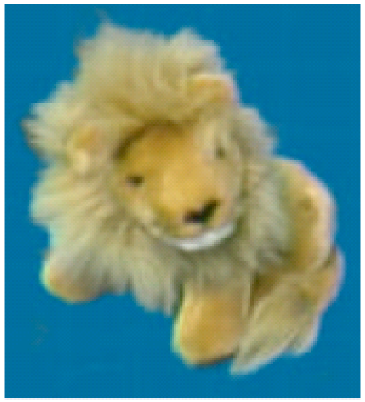
with edges/creases



http://www.cs.utah.edu/~gooch/SIG98/paper/drawing.html

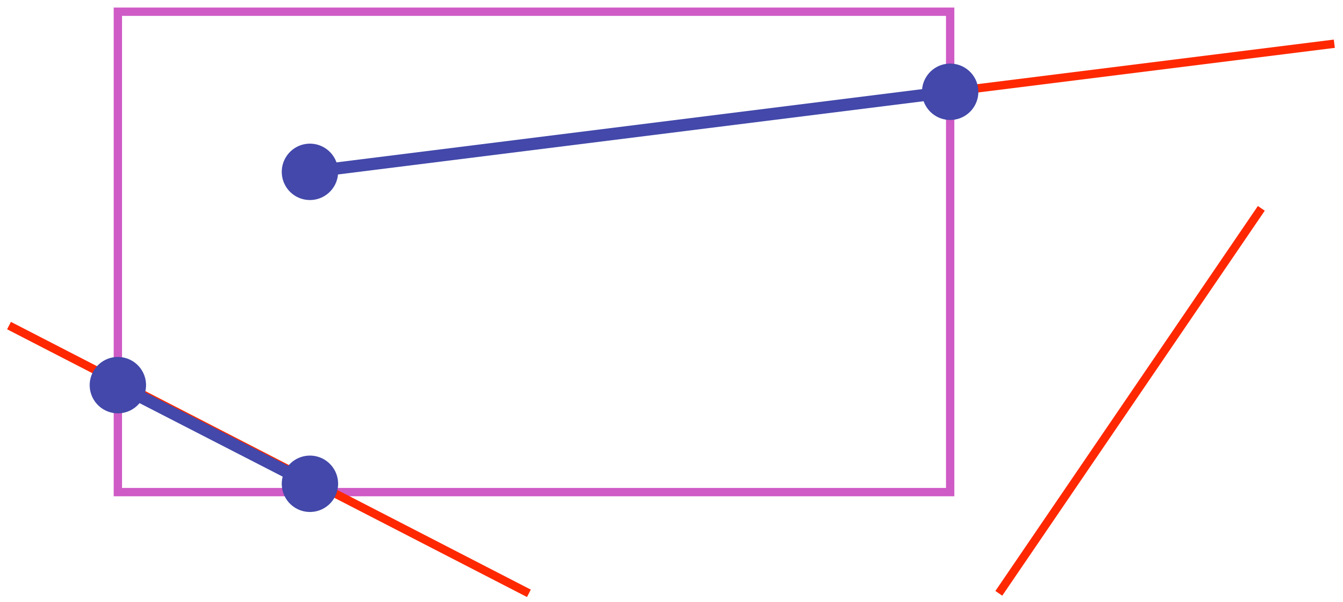
Review: Image-Based Modelling / Rendering

- store and access only pixels
  - no geometry, no light simulation, ...
  - input: set of images
  - output: image from new viewpoint
    - surprisingly large set of possible new viewpoints
- display time not tied to scene complexity
  - expensive rendering or real photographs
- convergence of graphics, vision, photography
  - computational photography

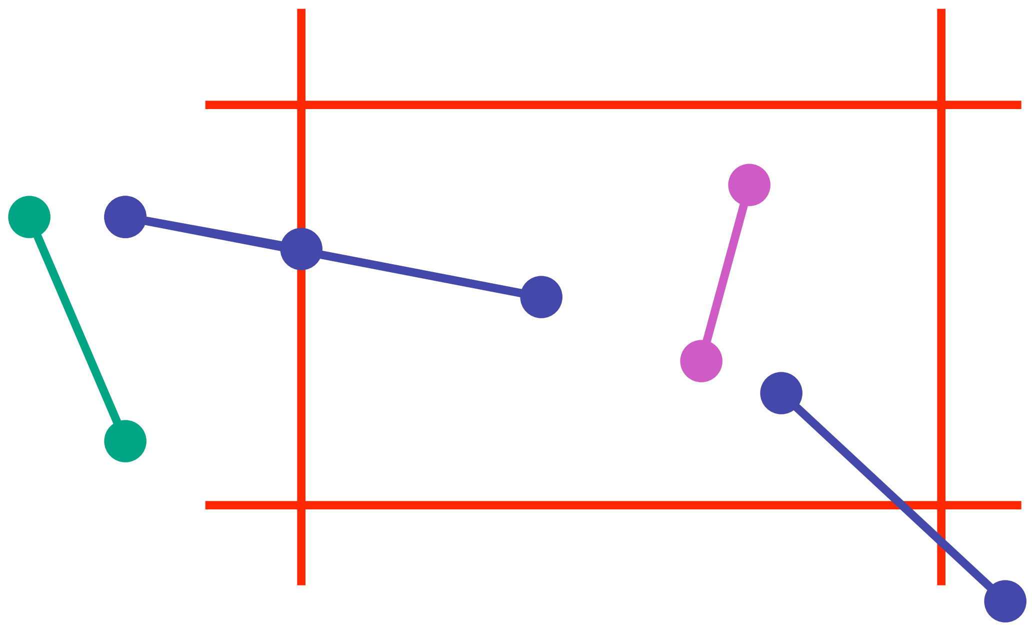
Review: Clipping

- analytically calculating the portions of primitives within the viewport



Review: Clipping Lines To Viewport

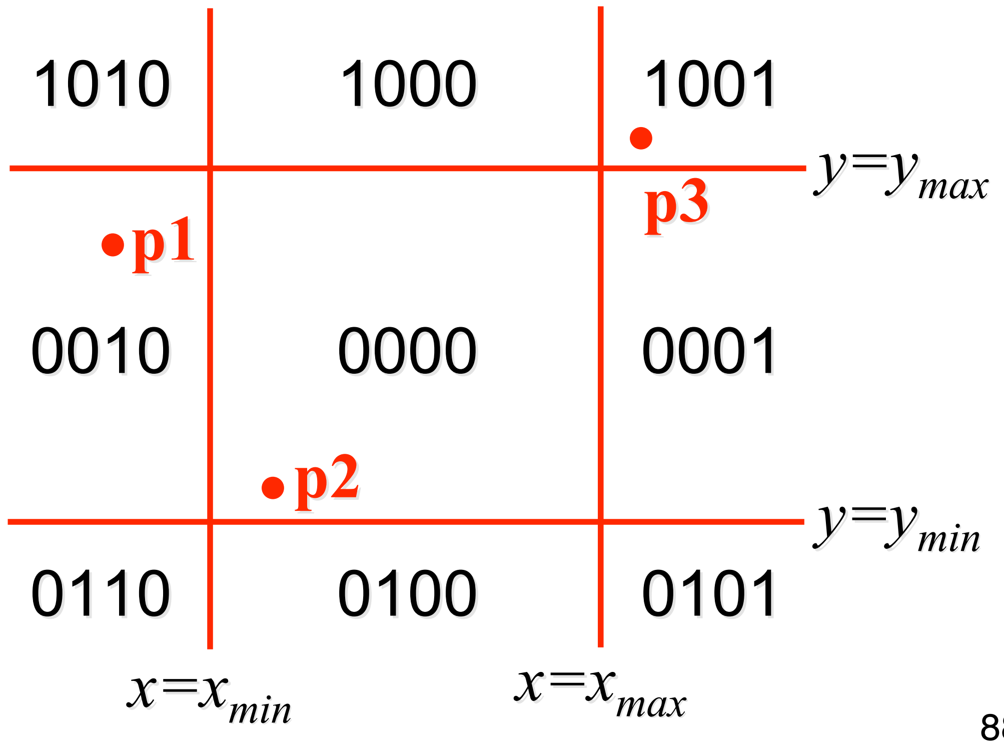
- combining trivial accepts/rejects
  - trivially accept lines with both endpoints inside all edges of the viewport
  - trivially reject lines with both endpoints outside the same edge of the viewport
  - otherwise, reduce to trivial cases by splitting into two segments



Review: Cohen-Sutherland Line Clipping

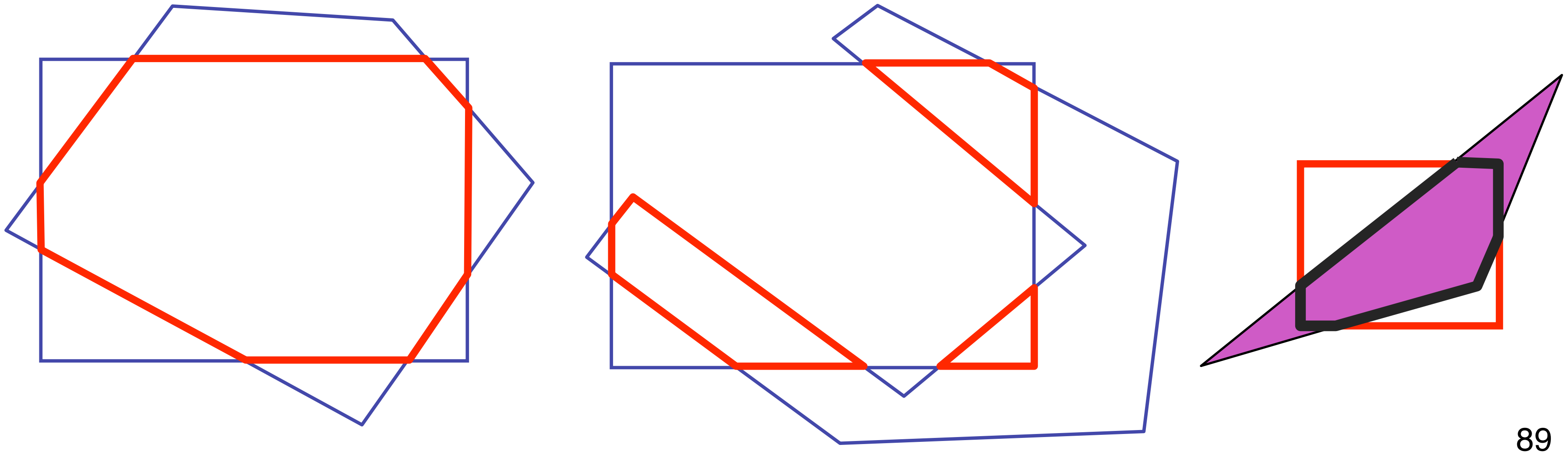
- outcodes
  - 4 flags encoding position of a point relative to top, bottom, left, and right boundary

- OC(p1)== 0 && OC(p2)==0
  - trivial accept
- (OC(p1) & OC(p2))!= 0
  - trivial reject



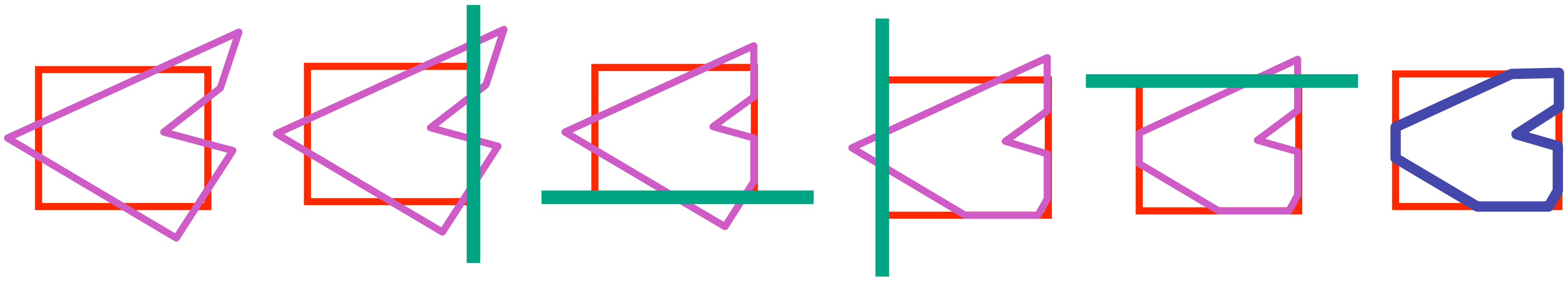
Review: Polygon Clipping

- not just clipping all boundary lines
- may have to introduce new line segments



Review: Sutherland-Hodgeman Clipping

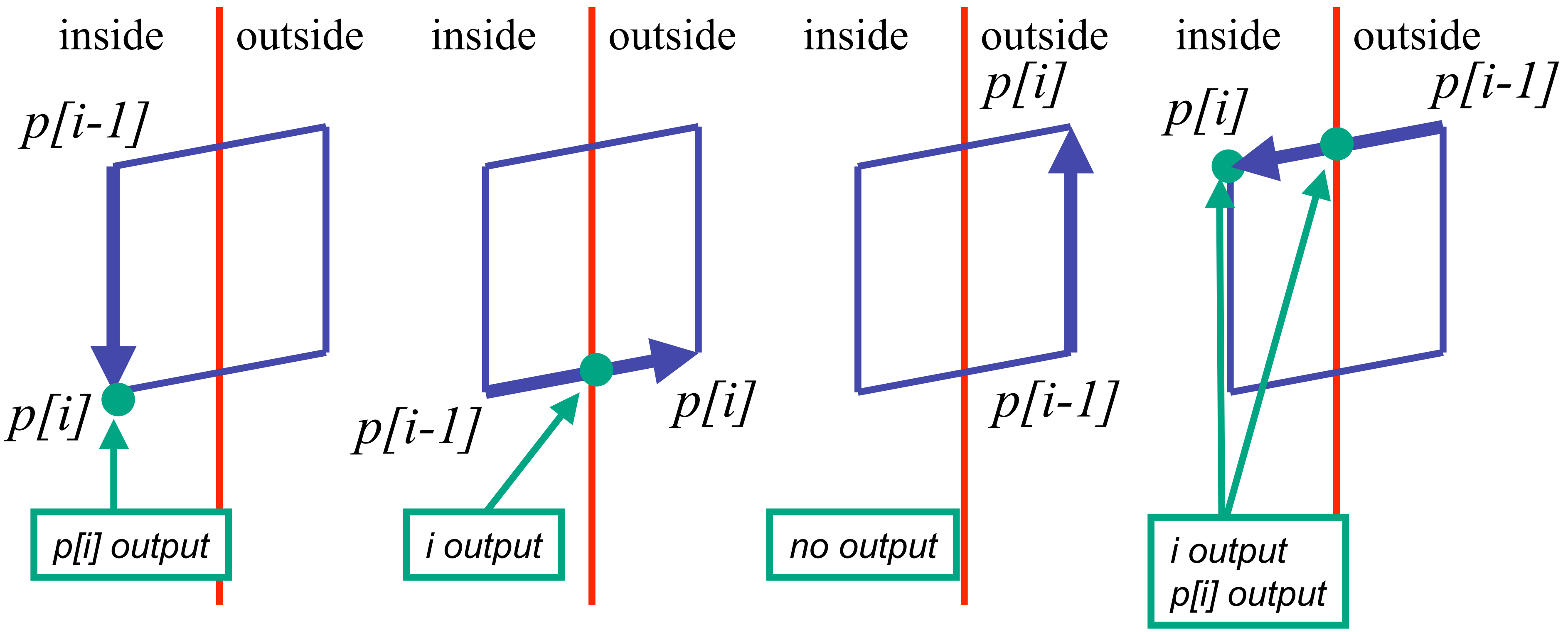
- for each viewport edge
  - clip the polygon against the edge equation for new vertex list
  - after doing all edges, the polygon is fully clipped



- for each polygon vertex
  - decide what to do based on 4 possibilities
    - is vertex inside or outside?
    - is previous vertex inside or outside?

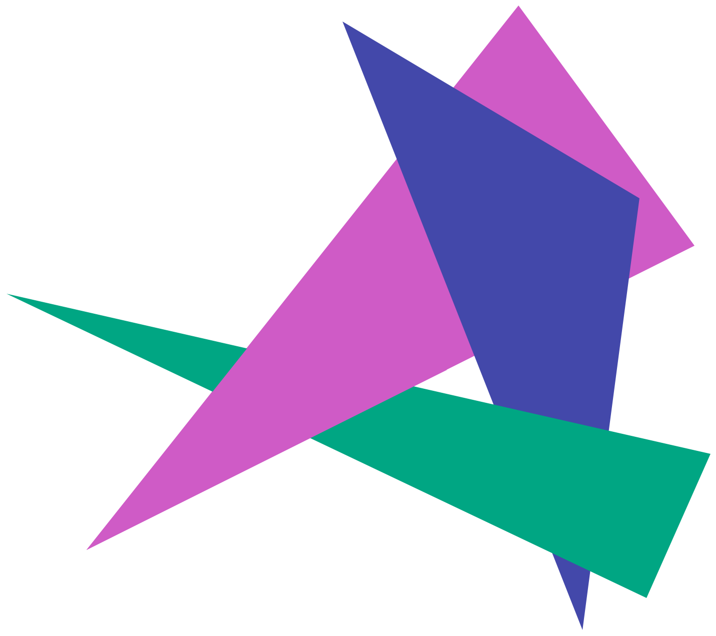
Review: Sutherland-Hodgeman Clipping

- edge from *p[i-1]* to *p[i]* has four cases
  - decide what to add to output vertex list



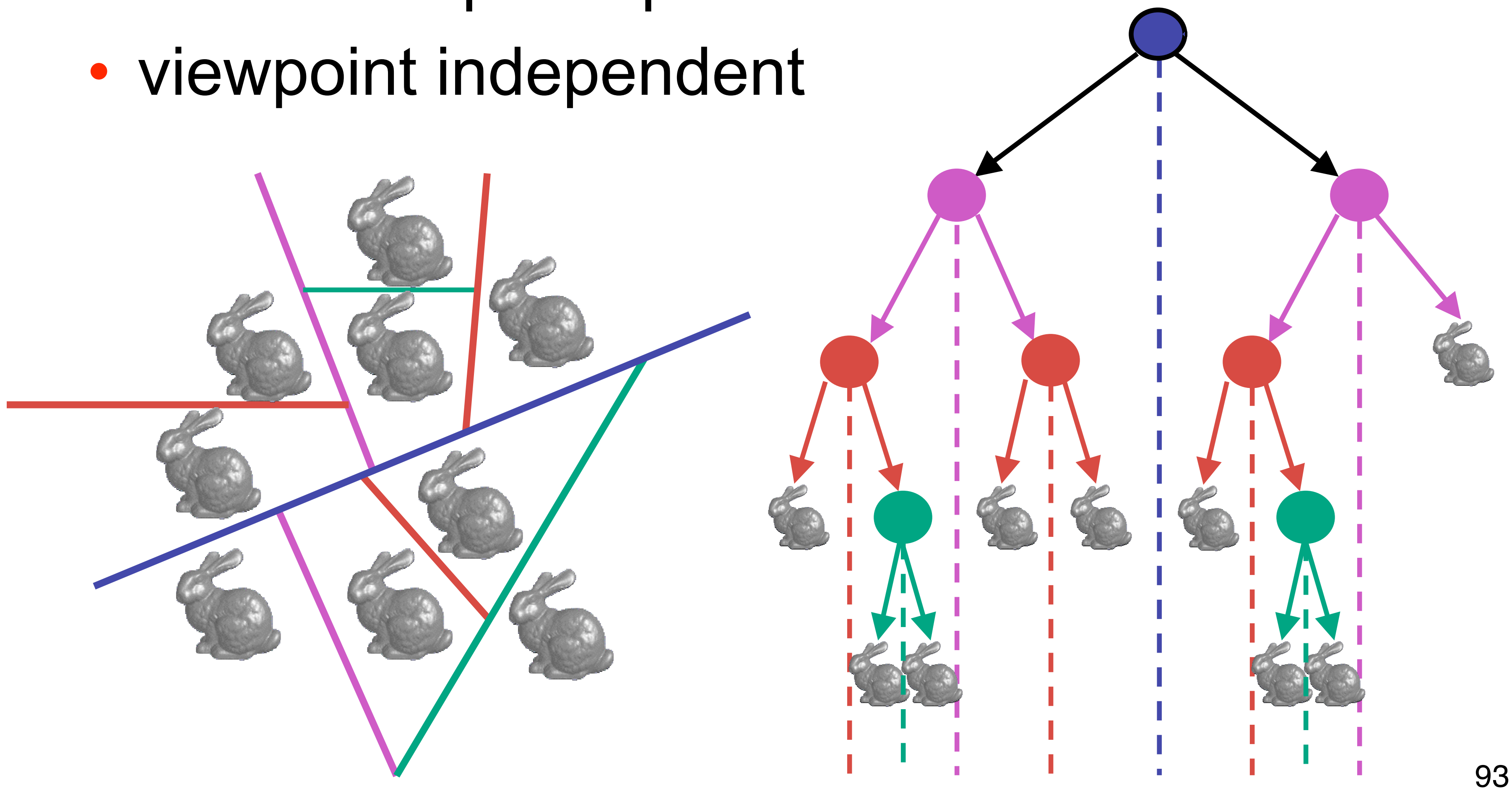
Review: Painter's Algorithm

- draw objects from back to front
- problems: no valid visibility order for
  - intersecting polygons
  - cycles of non-intersecting polygons possible



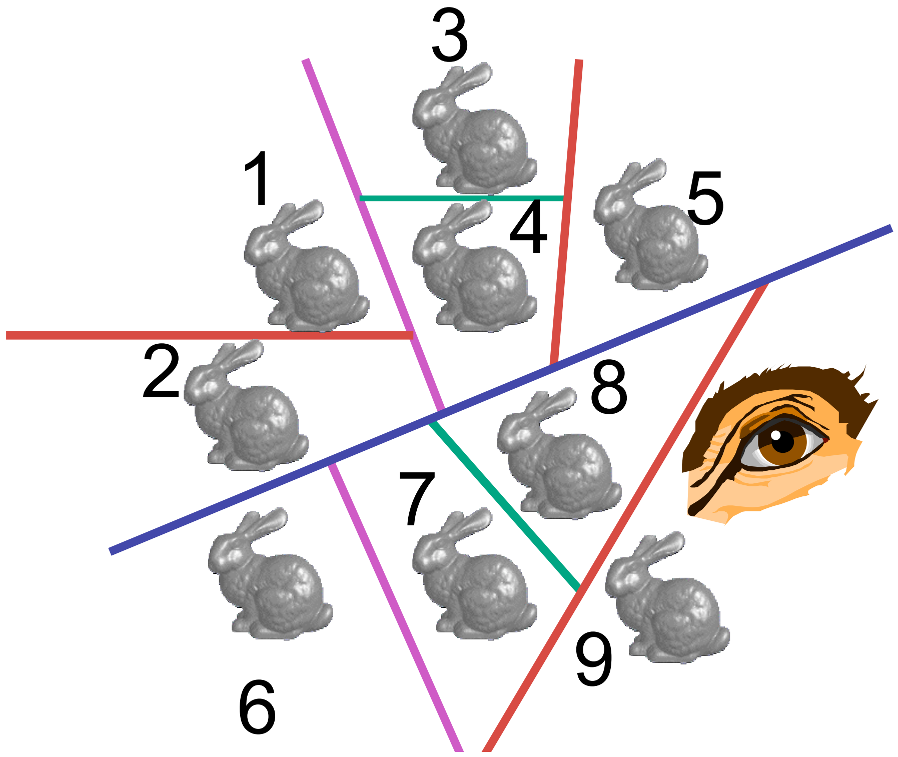
Review: BSP Trees

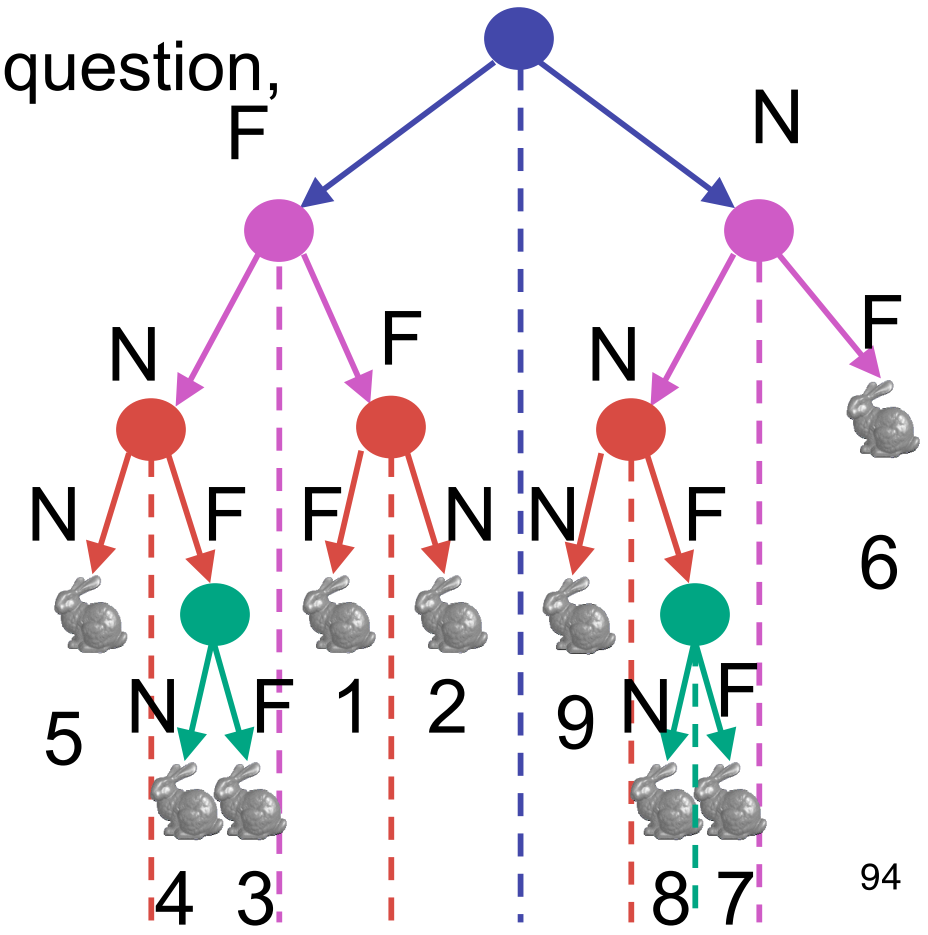
- preprocess: create binary tree
  - recursive spatial partition
  - viewpoint independent



Review: BSP Trees

- runtime: correctly traversing this tree enumerates objects from back to front
  - viewpoint dependent: check which side of plane viewpoint is on **at each node**
  - draw far, draw object in question, draw near



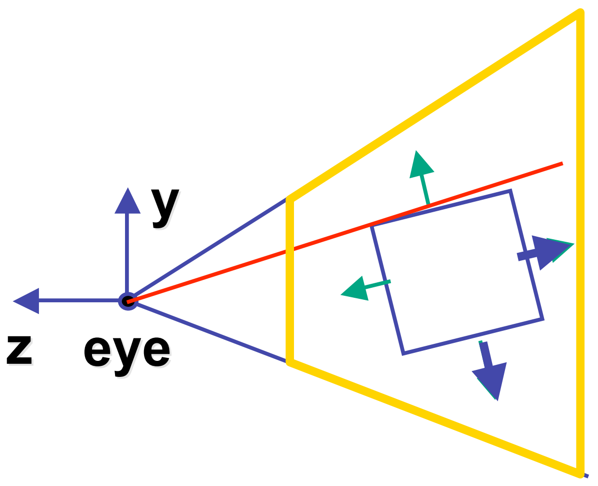


Review: Z-Buffer Algorithm

- augment color framebuffer with Z-buffer or depth buffer which stores Z value at each pixel
    - at frame beginning, initialize all pixel depths to ∞
    - when rasterizing, interpolate depth (Z) across polygon
    - check Z-buffer before storing pixel color in framebuffer and storing depth in Z-buffer
    - don’t write pixel if its Z value is more distant than the Z value already stored there

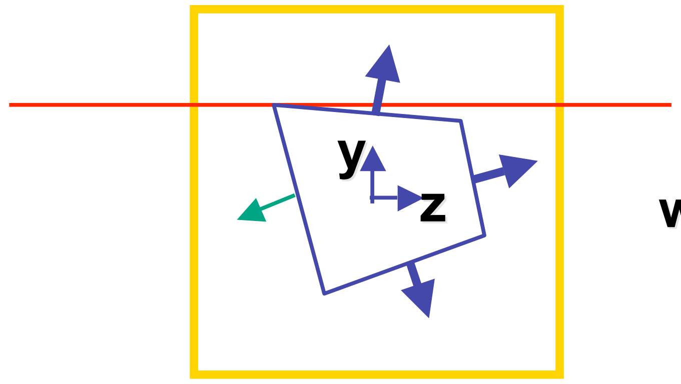
Review: Back-face Culling

VCS



NDCS

eye



works to cull if $N_Z > 0$

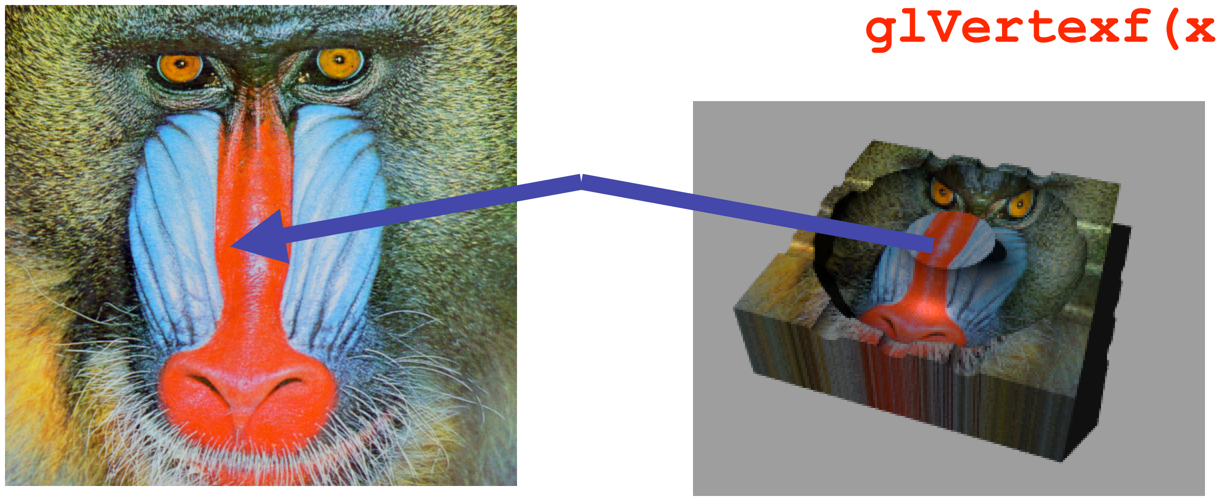
Review: Invisible Primitives

- *why might a polygon be invisible?*
  - polygon outside the *field of view / frustum*
    - solved by clipping
  - polygon is *backfacing*
    - solved by backface culling
  - polygon is *occluded* by object(s) nearer the viewpoint
    - solved by hidden surface removal

Review: Texture Coordinates

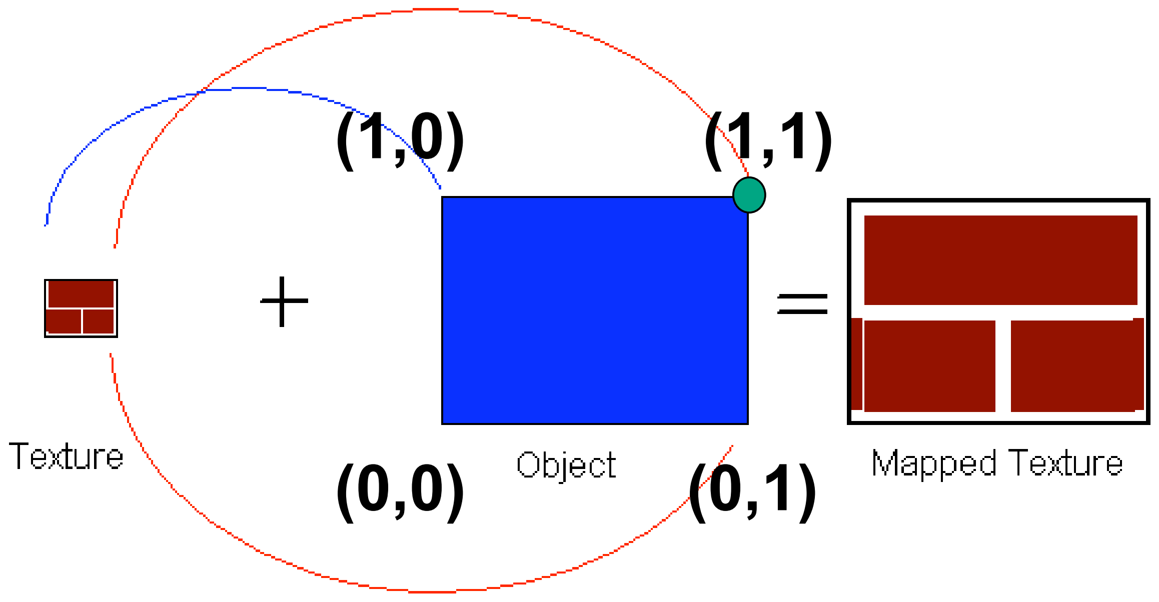
- texture image: 2D array of color values (texels)
- assigning texture coordinates (s,t) at vertex with object coordinates (x,y,z,w)
  - use interpolated (s,t) for texel lookup at each pixel
  - use value to modify a polygon's color
    - or other surface property
  - specified by programmer or artist

```
glTexCoord2f(s,t)
glVertexf(x,y,z,w)
```

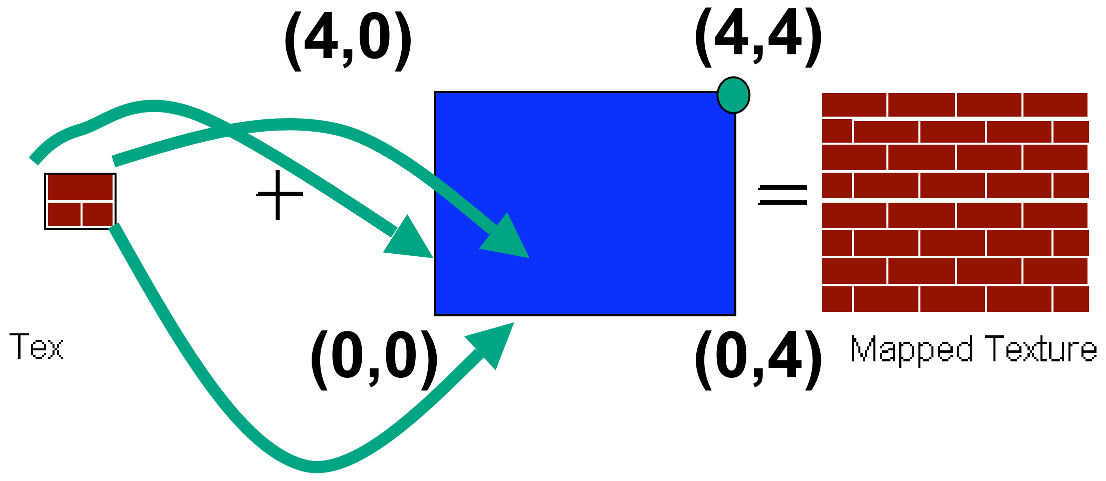


Review: Tiled Texture Map

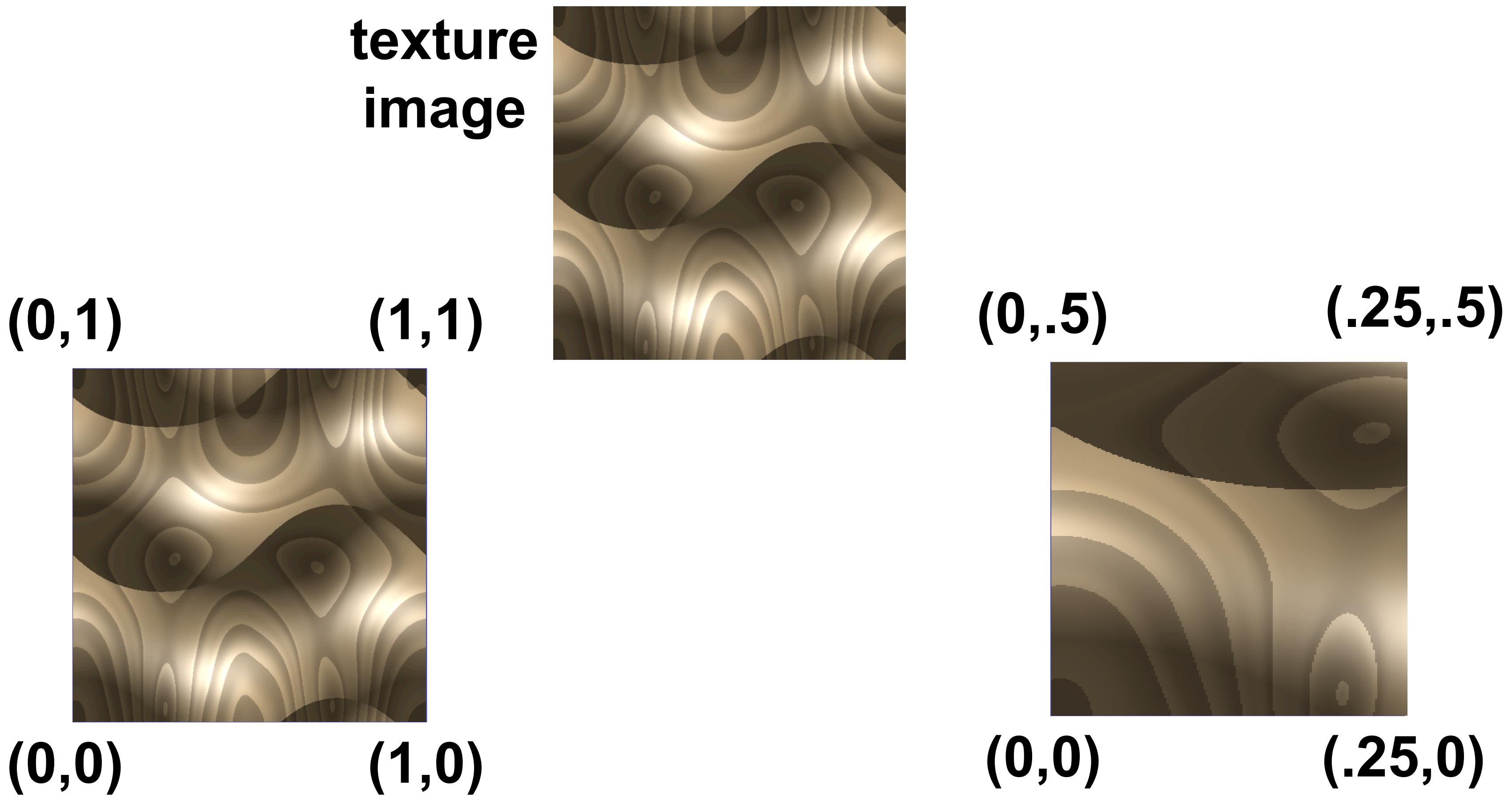
```
glTexCoord2d(1, 1);
glVertex3d (x, y, z);
```



```
glTexCoord2d(4, 4);
glVertex3d (x, y, z);
```



Review: Fractional Texture Coordinates



Review: Texture

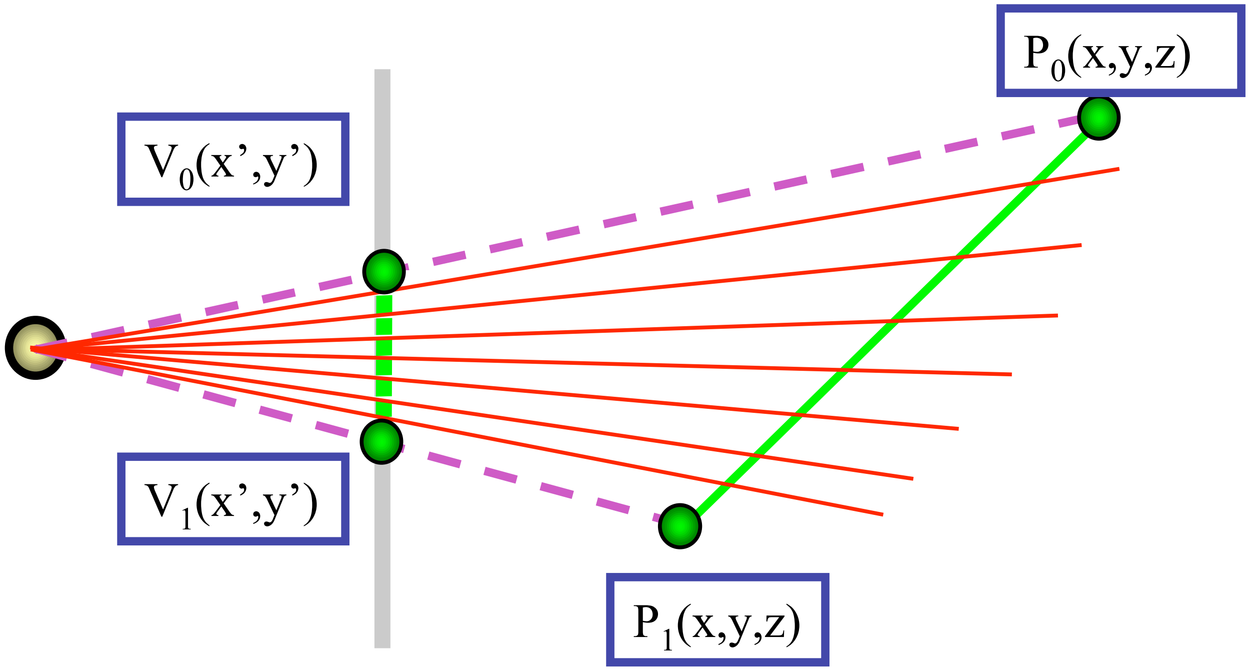
- action when s or t is outside [0…1] interval
  - tiling
  - clamping
- functions
  - replace/decal
  - modulate
  - blend
- texture matrix stack

```
glMatrixMode( GL_TEXTURE );
```

Review: Perspective Correct Interpolation

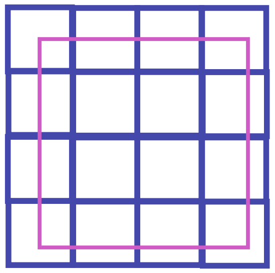
- screen space interpolation incorrect

$$s = \frac{\alpha \cdot s_0 / w_0 + \beta \cdot s_1 / w_1 + \gamma \cdot s_2 / w_2}{\alpha / w_0 + \beta / w_1 + \gamma / w_2}$$

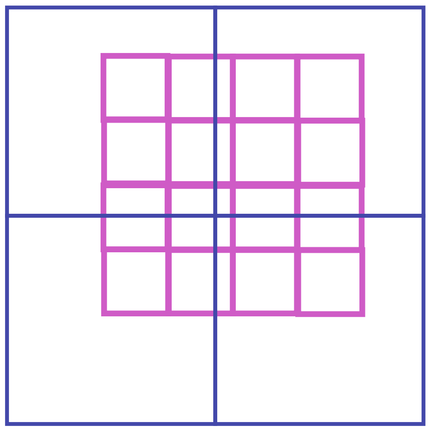


Review: Reconstruction

- how to deal with:
  - pixels that are much larger than texels?
    - apply filtering, “averaging”

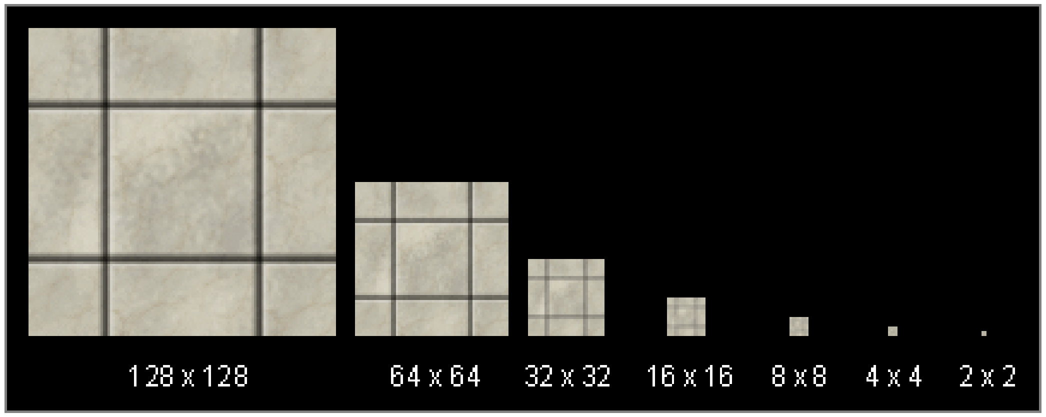


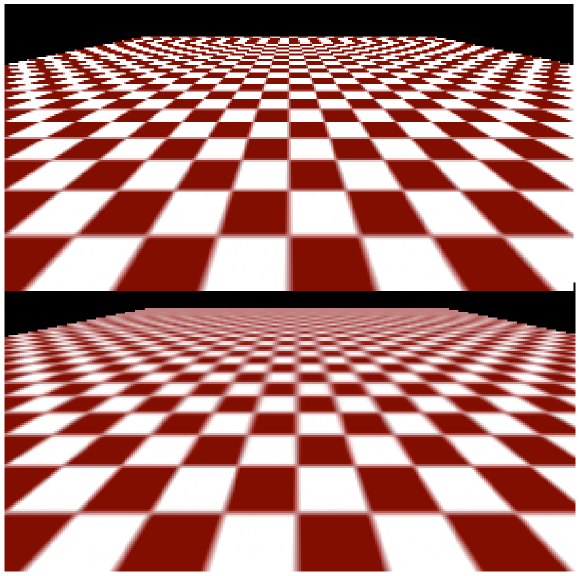
  - pixels that are much smaller than texels ?
    - interpolate

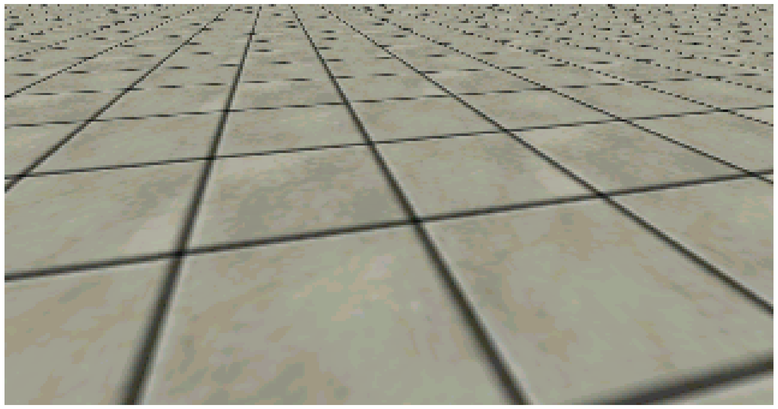


Review: MIPmapping

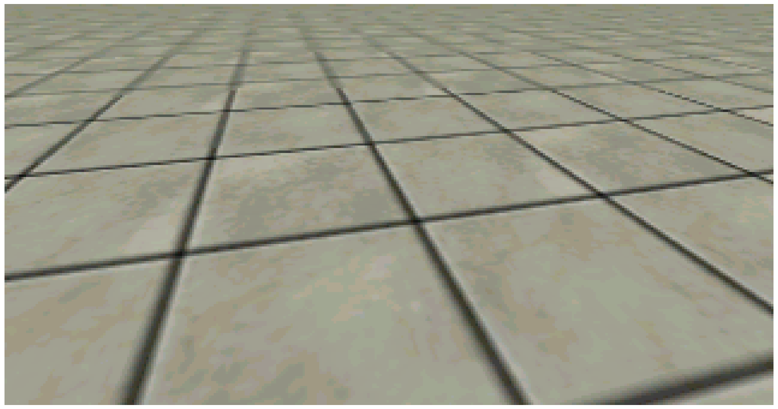
- image pyramid, precompute averaged versions







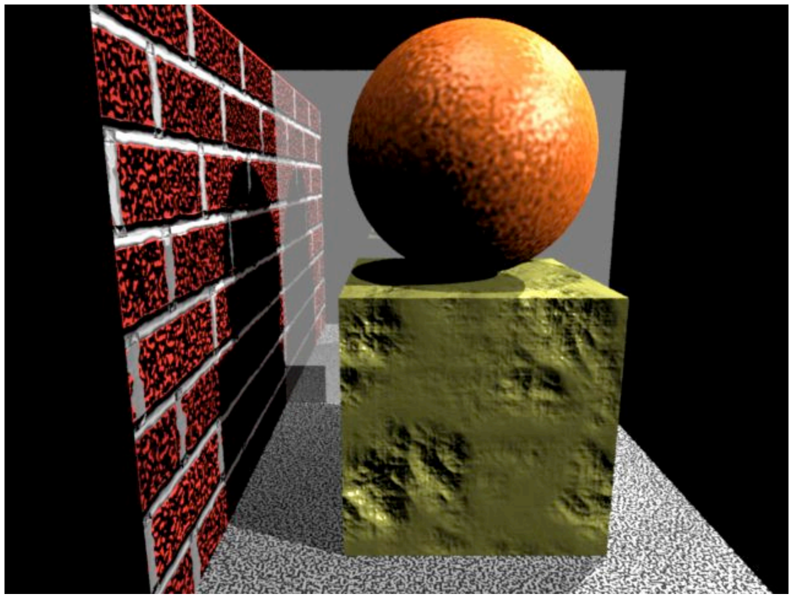
Without MIP-mapping

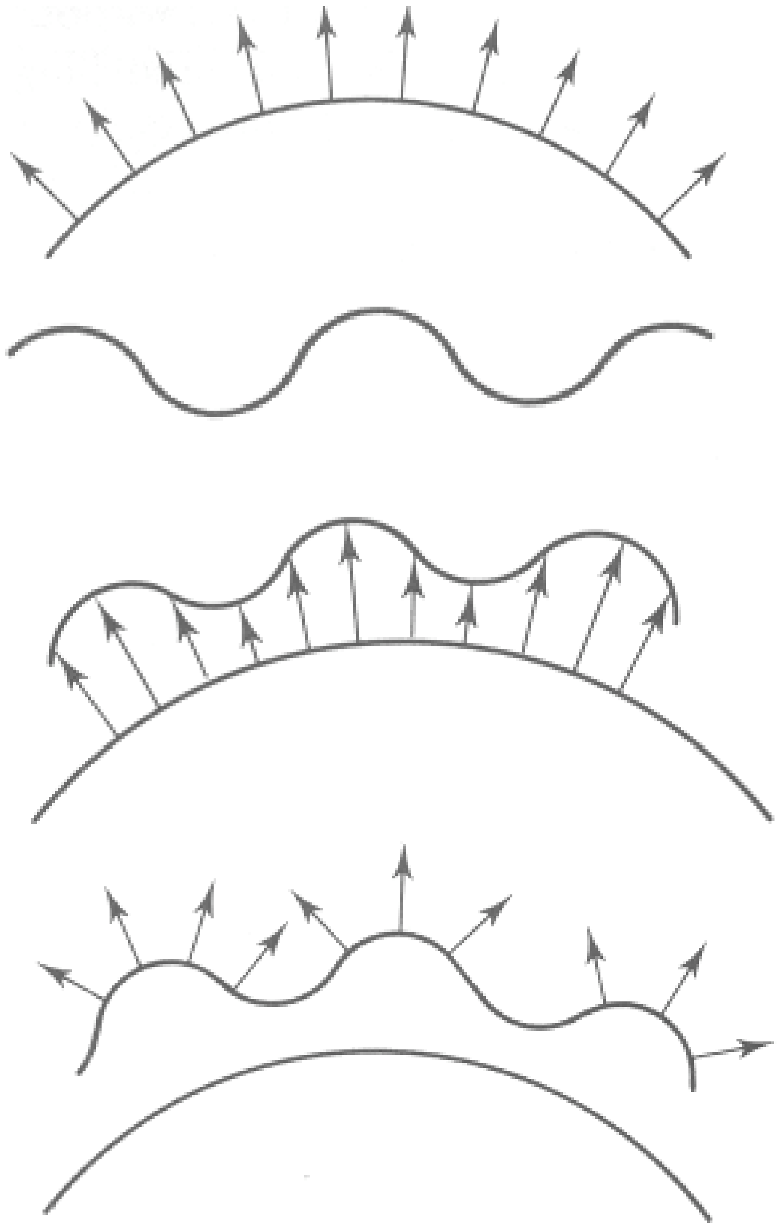


With MIP-mapping

Review: Bump Mapping: Normals As Texture

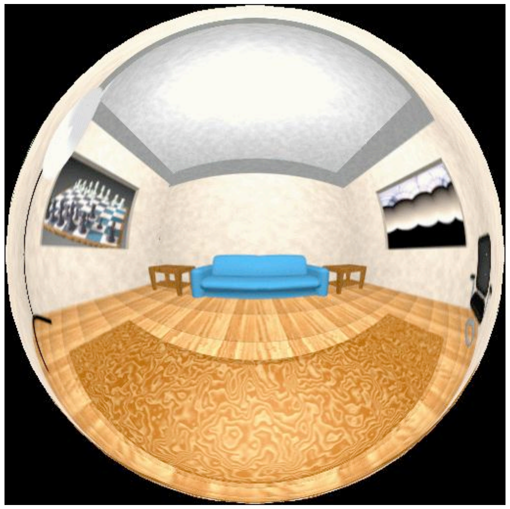
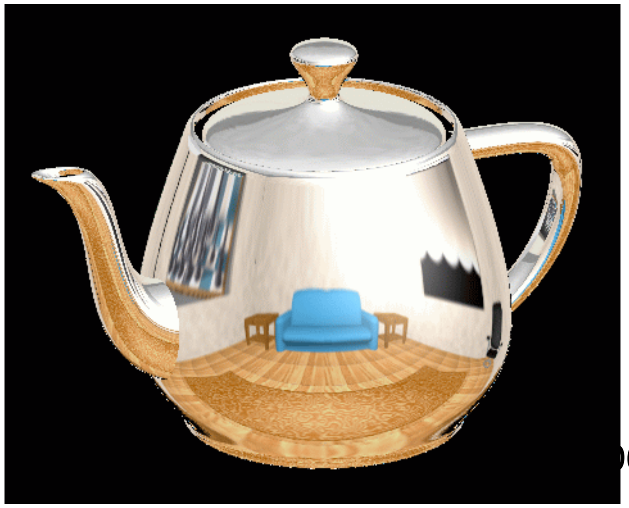
- create illusion of complex geometry model
- control shape effect by locally perturbing surface normal





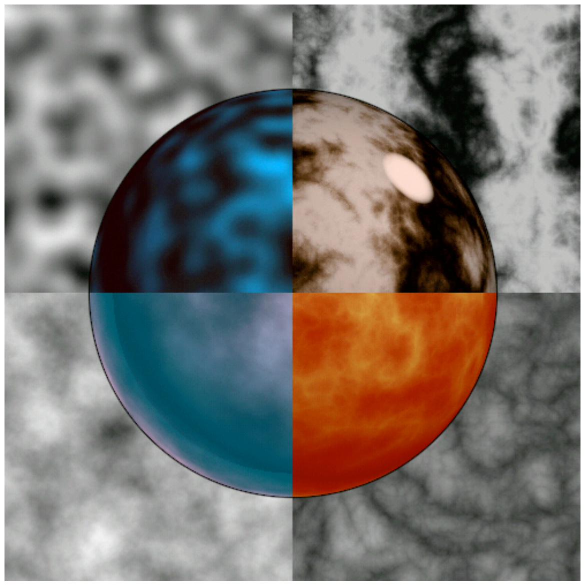
Review: Environment Mapping

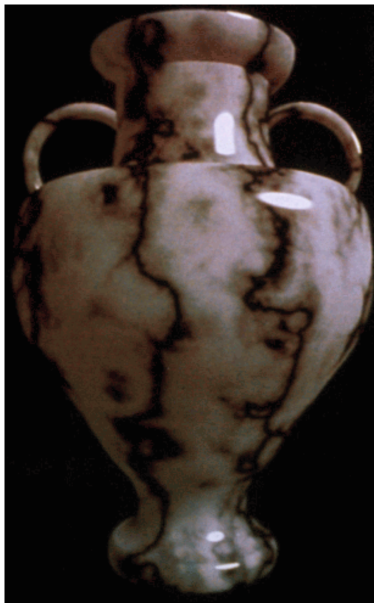
- cheap way to achieve reflective effect
  - generate image of surrounding
  - map to object as texture
- sphere mapping: texture is distorted fisheye view
  - point camera at mirrored sphere
  - use spherical texture coordinates

Review: Perlin Noise: Procedural Textures

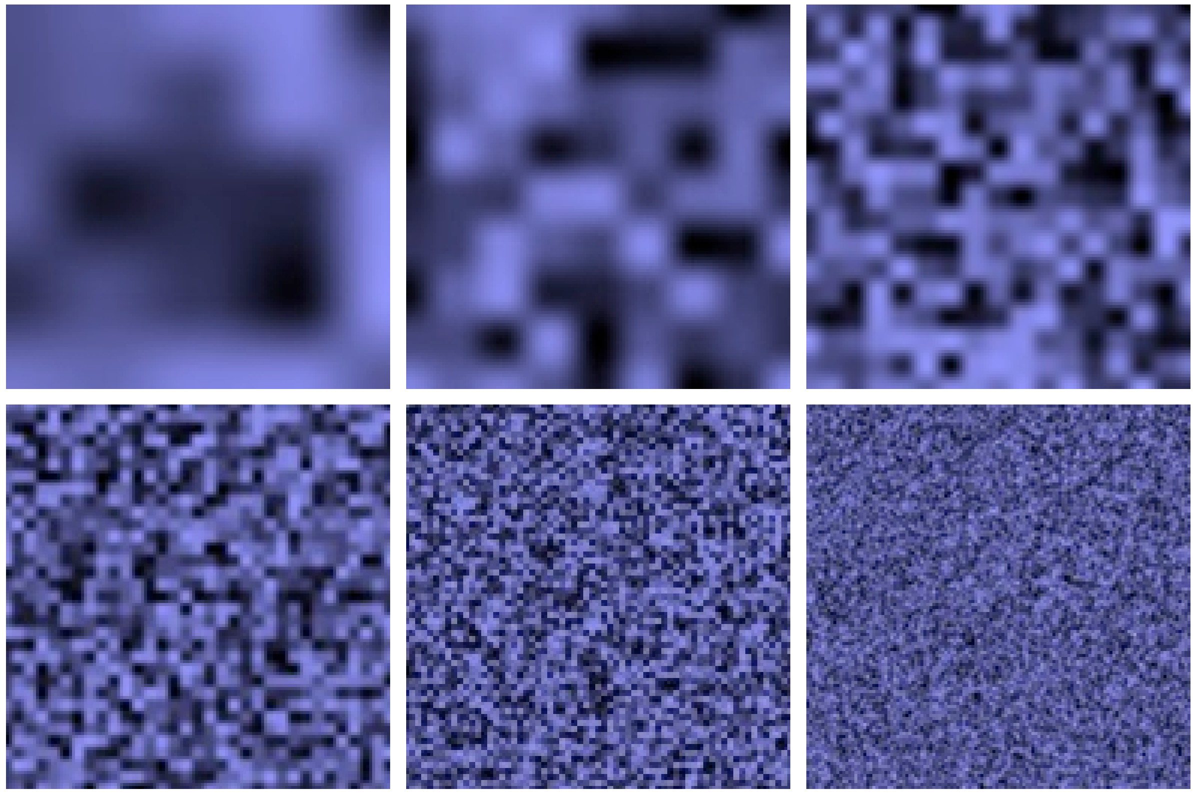
```
function marble(point)
 x = point.x + turbulence(point);
 return marble_color(sin(x))
```

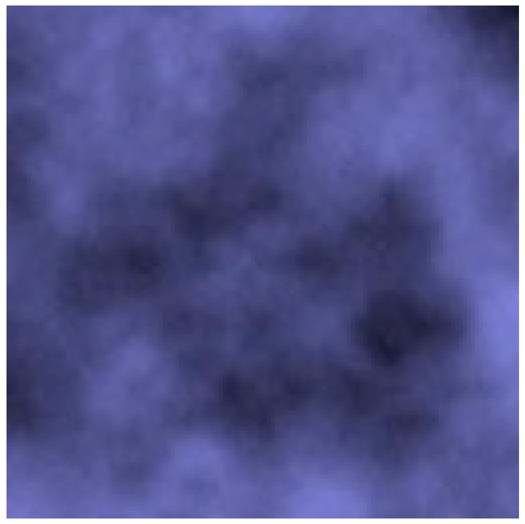




Review: Perlin Noise

- coherency: smooth not abrupt changes
- turbulence: multiple feature sizes



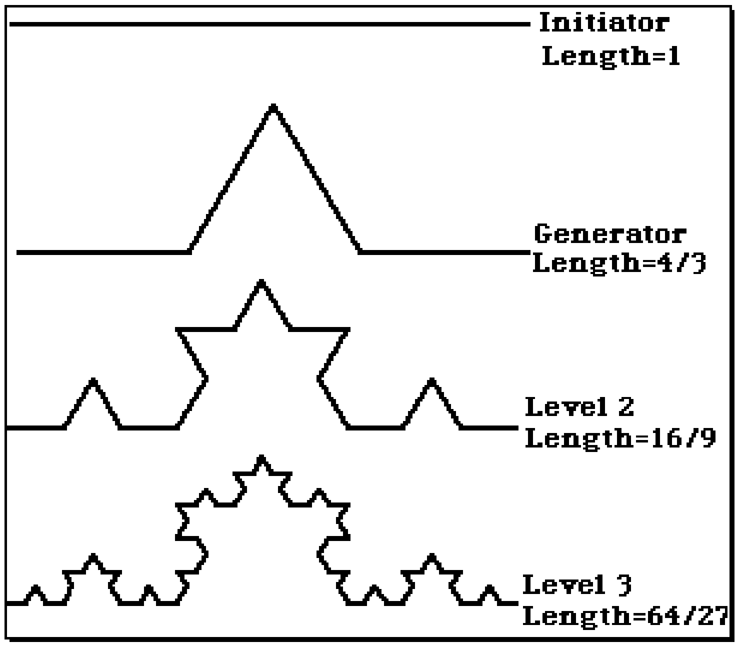


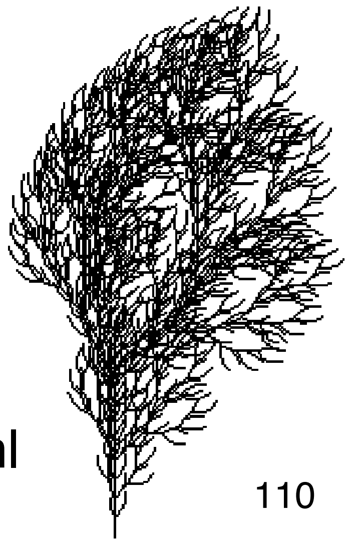
Review: Procedural Modeling

- textures, geometry
    - nonprocedural: explicitly stored in memory
- procedural approach
    - compute something on the fly
        - not load from disk
    - often less memory cost
    - visual richness
        - adaptable precision
- noise, fractals, particle systems

Review: Language-Based Generation

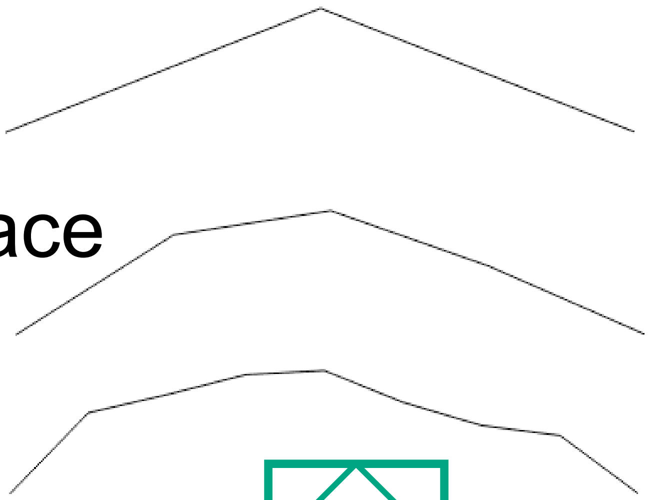
- L-Systems
  - F: forward, R: right, L: left
  - Koch snowflake:
    F = FLFRRFLF
  - Mariano's Bush:
    F=FF-[-F+F+F]+[+F-F-F]
    - angle 16



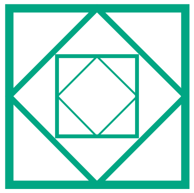


http://spanky.triumf.ca/www/fractint/lsys/plants.html

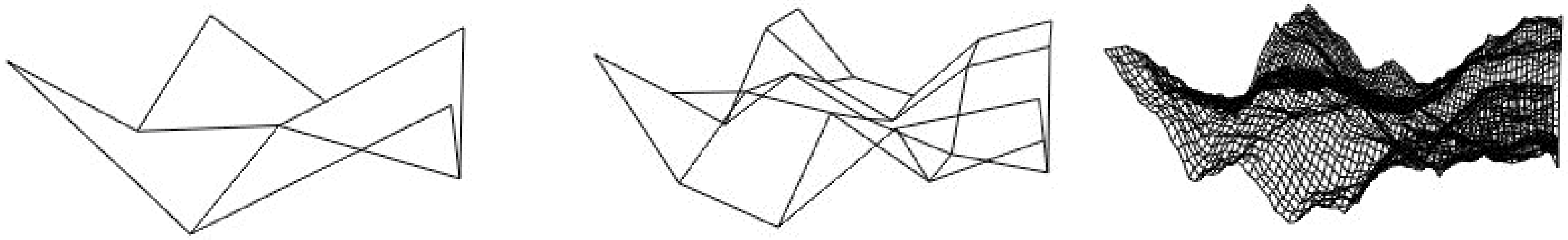
Review: Fractal Terrain



- 1D: midpoint displacement
  - divide in half, randomly displace
  - scale variance by half
- 2D: diamond-square



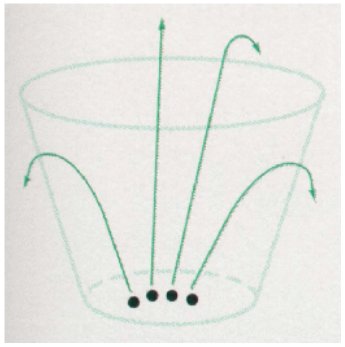
  - generate new value at midpoint
  - average corner values + random displacement
    - scale variance by half each time

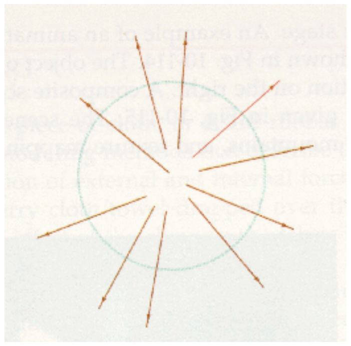


http://www.gameprogrammer.com/fractal.html

Review: Particle Systems

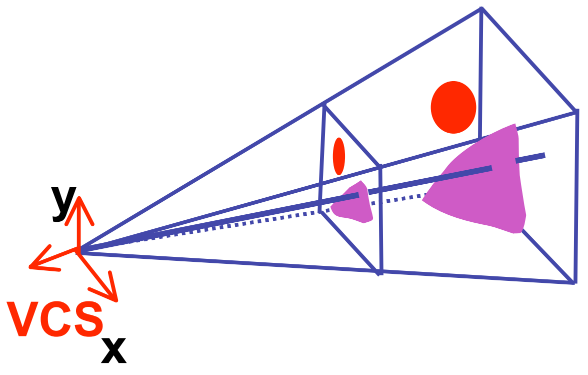
- changeable/fluid stuff
  - fire, steam, smoke, water, grass, hair, dust, waterfalls, fireworks, explosions, flocks
- life cycle
  - generation, dynamics, death
- rendering tricks
  - avoid hidden surface computations



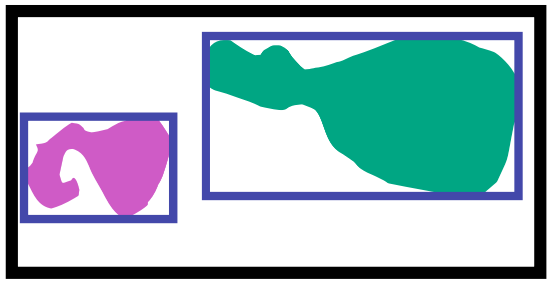


Review: Picking Methods

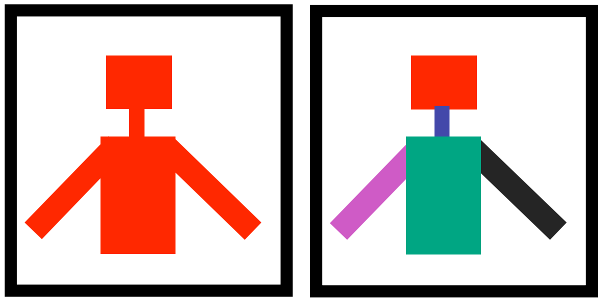
- manual ray intersection

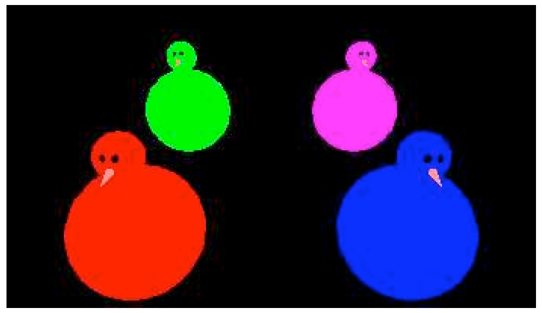


- bounding extents



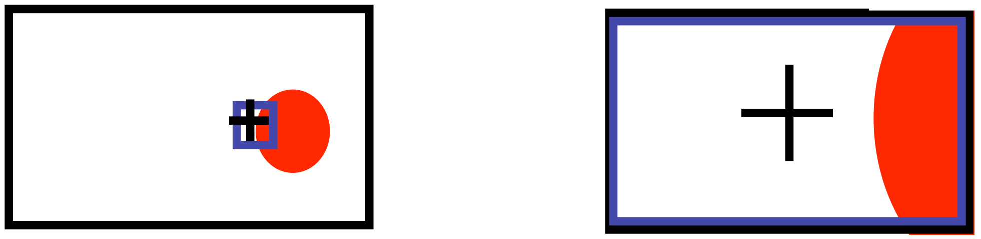
- backbuffer coding





Review: Select/Hit Picking

- assign (hierarchical) integer key/name(s)
- small region around cursor as new viewport



- redraw in selection mode
    - equivalent to casting pick “tube”
    - store keys, depth for drawn objects in hit list
- examine hit list
    - usually use frontmost, but up to application

Review: Hit List

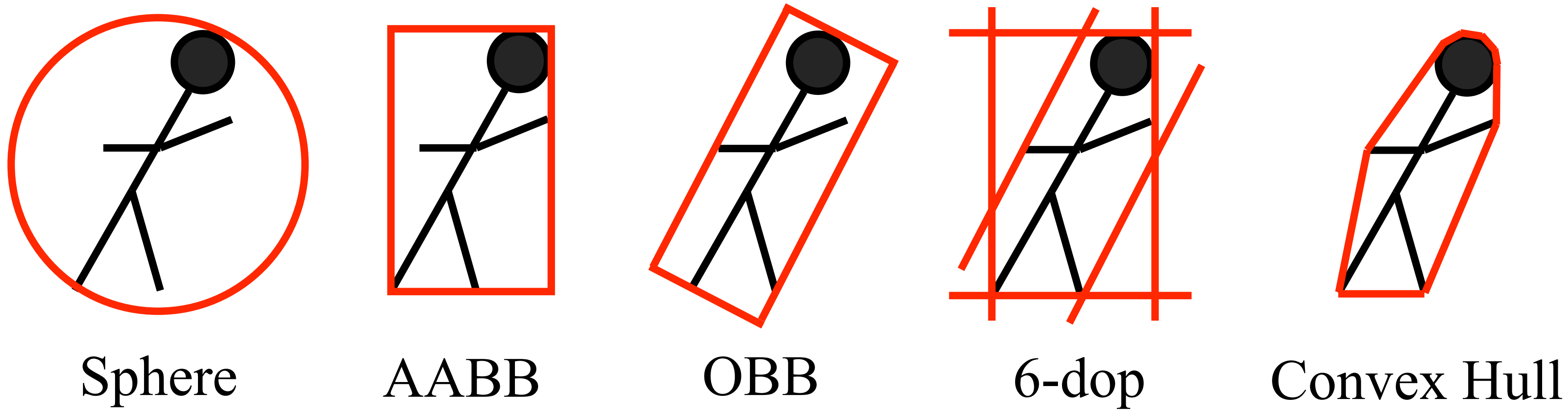
- glSelectBuffer(buffersize, *buffer)
  - where to store hit list data
- on hit, copy entire contents of name stack to output buffer.
- hit record
  - number of names on stack
  - minimum and maximum depth of object vertices
    - depth lies in the z-buffer range [0,1]
    - multiplied by 2^32 -1 then rounded to nearest int

Review: Collision Detection

- boundary check
  - perimeter of world vs. viewpoint or objects
    - 2D/3D absolute coordinates for bounds
    - simple point in space for viewpoint/objects
- set of fixed barriers
  - walls in maze game
    - 2D/3D absolute coordinate system
- set of moveable objects
  - one object against set of items
    - missile vs. several tanks
  - multiple objects against each other
    - punching game: arms and legs of players
    - room of bouncing balls

Review: Collision Proxy Tradeoffs

- collision proxy (bounding volume) is piece of geometry used to represent complex object for purposes of finding collision
- proxies exploit facts about human perception
  - we are bad at determining collision correctness
  - especially many things happening quickly

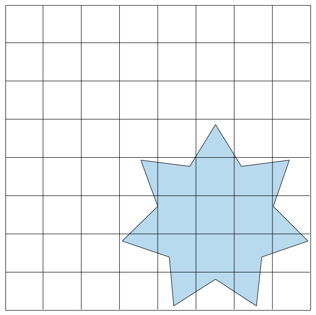


increasing complexity & tightness of fit

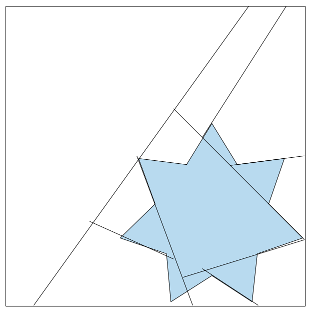
decreasing cost of (overlap tests + proxy update)

Review: Spatial Data Structures

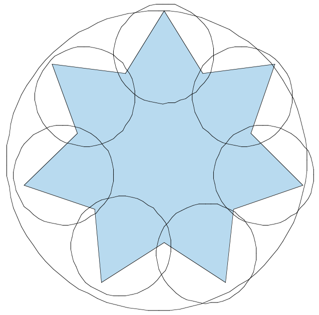
uniform grids



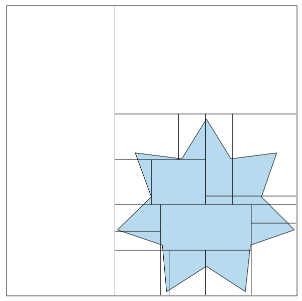
BSP trees



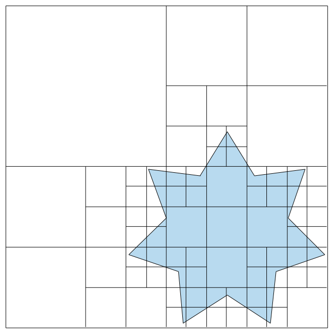
bounding volume hierarchies



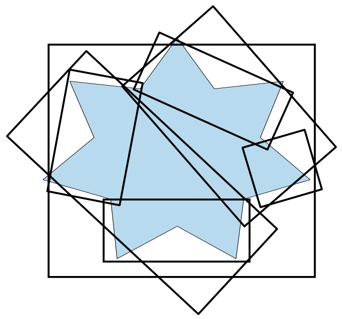
kd-trees



octrees



OBB trees

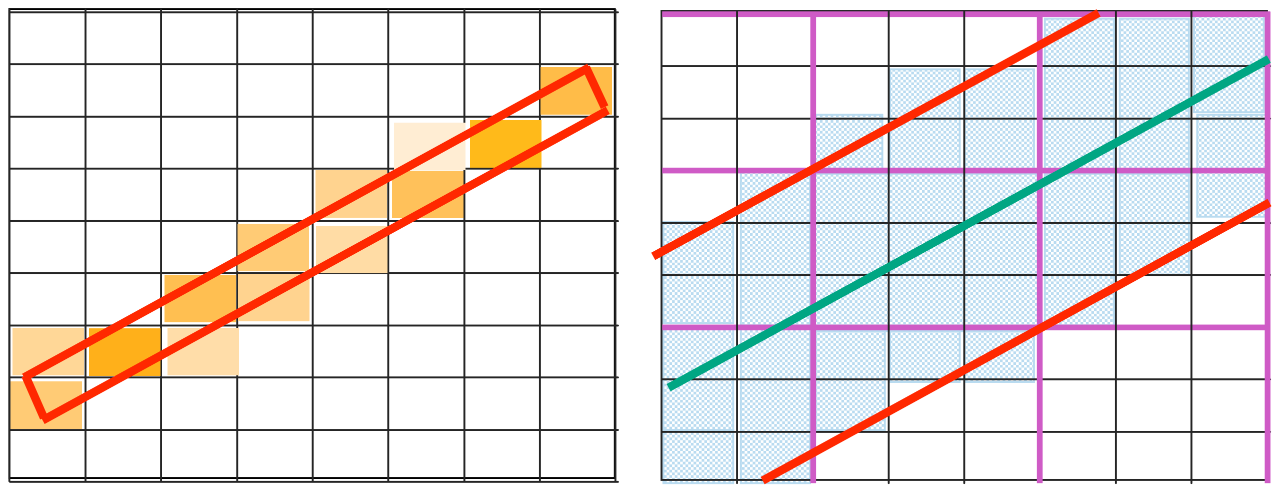


Review: Aliasing

- incorrect appearance of high frequencies as low frequencies
- to avoid: antialiasing
  - supersample
    - sample at higher frequency
  - low pass filtering
    - remove high frequency function parts
    - aka prefiltering, band-limiting

Review: Supersample and Average

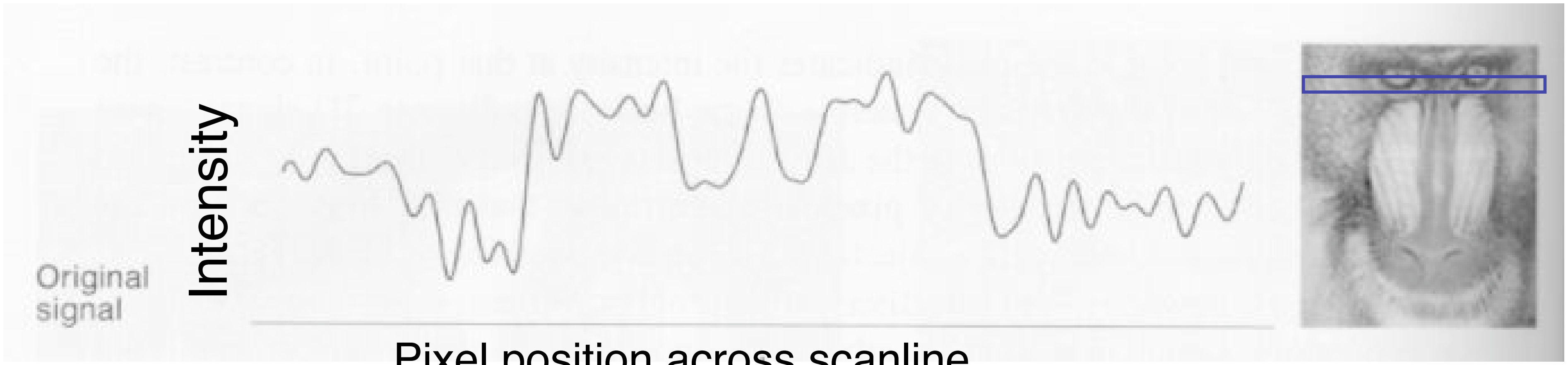
- supersample: create image at higher resolution
  - e.g. 768x768 instead of 256x256
  - shade pixels wrt area covered by thick line/rectangle
- average across many pixels
  - e.g. 3x3 small pixel block to find value for 1 big pixel
  - rough approximation divides each pixel into a finer grid of pixels



5/9	9/9
9/9	6/9
4/9	0/9

Review: Image As Signal

- 1D slice of raster image
  - discrete sampling of 1D spatial signal
- theorem
  - any signal can be represented as an (infinite) sum of sine waves at different frequencies



Pixel position across scanline

Examples from Foley, van Dam, Feiner, and Hughes

Review: Sampling Theorem and Nyquist Rate

- Shannon Sampling Theorem
  - continuous signal can be completely recovered from its samples iff sampling rate greater than twice maximum frequency present in signal
- sample past Nyquist Rate to avoid aliasing
  - twice the highest frequency component in the image's spectrum

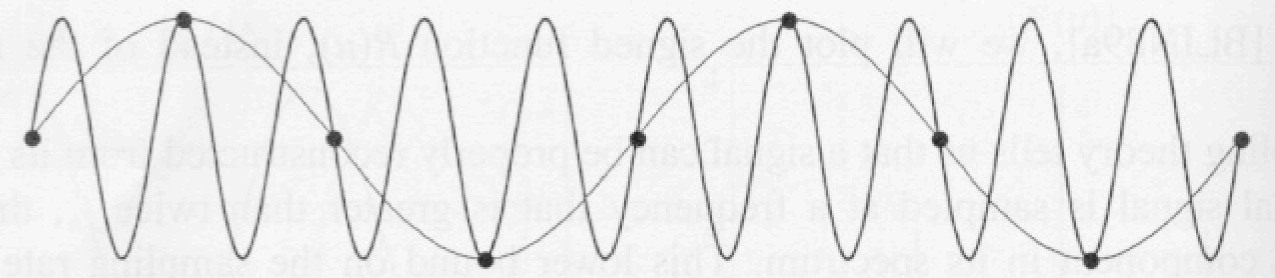
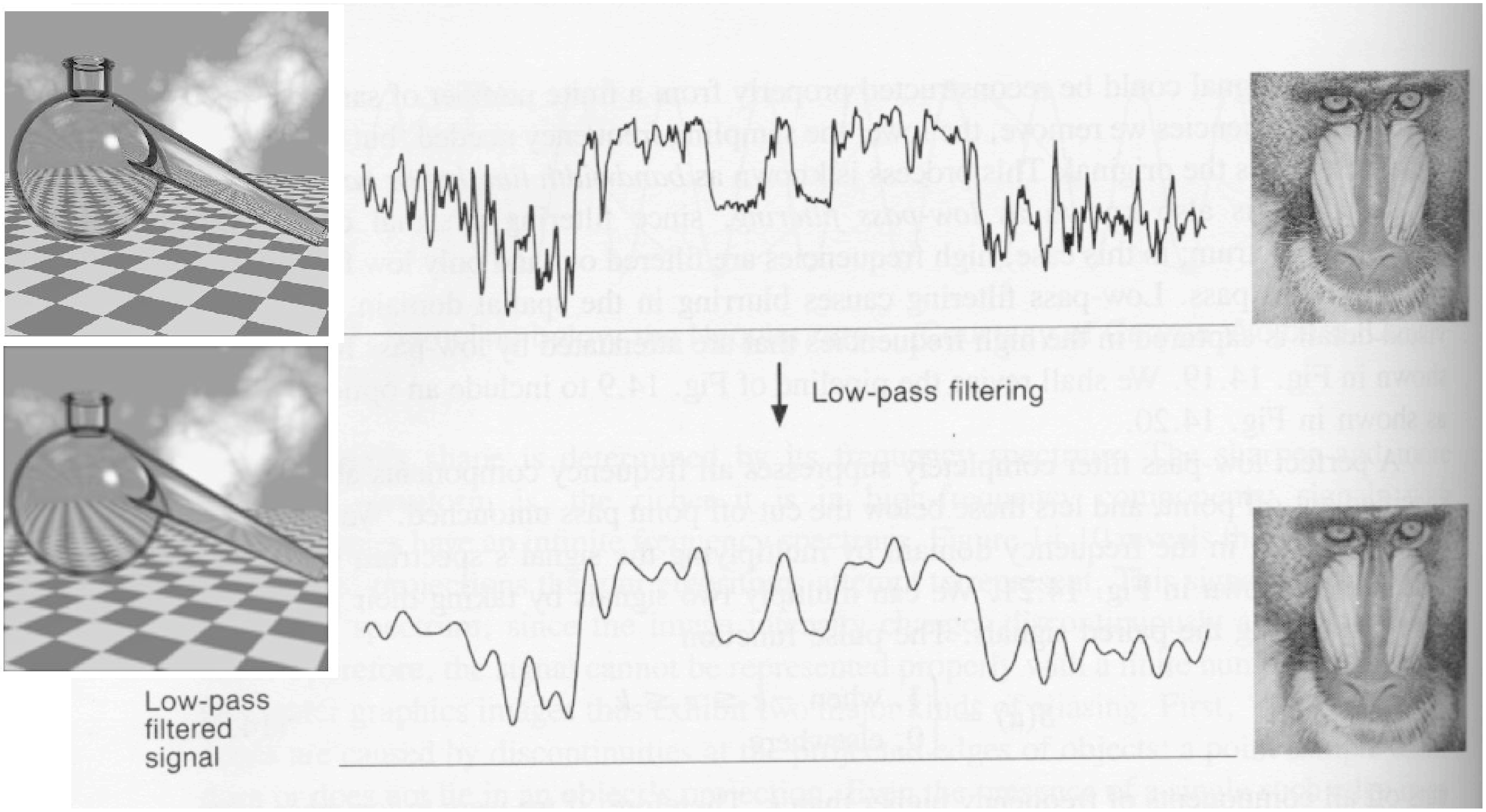


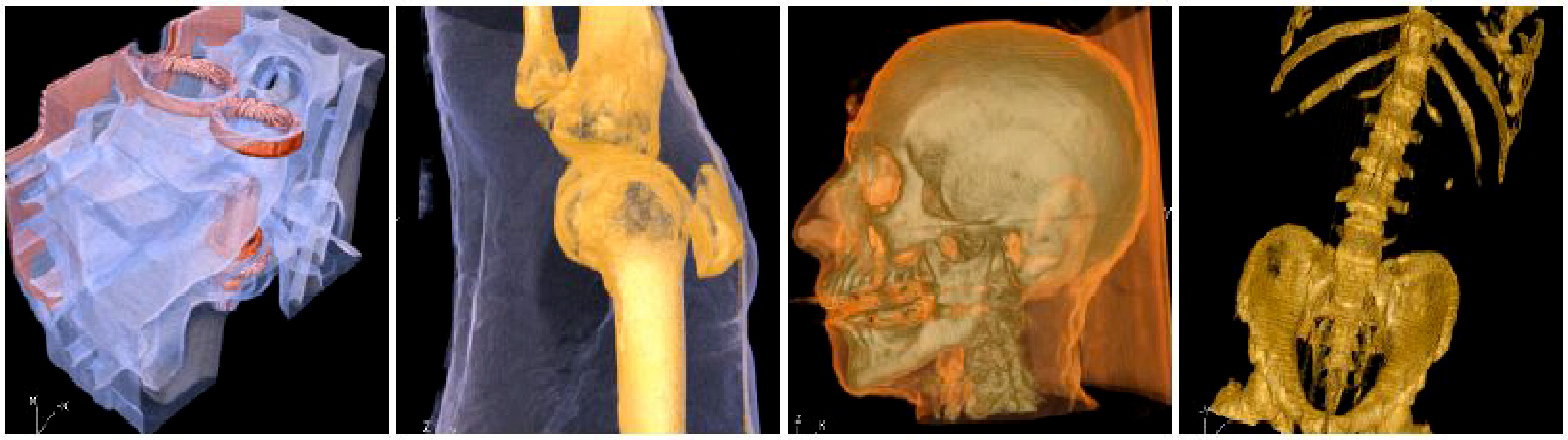
Fig. 14.17 Sampling below the Nyquist rate. (Courtesy of George Wolberg, Columbia University.)

Review: Low-Pass Filtering



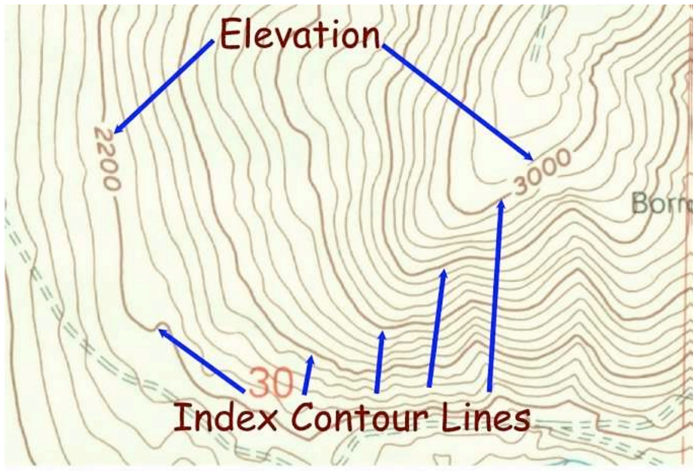
Review: Volume Graphics

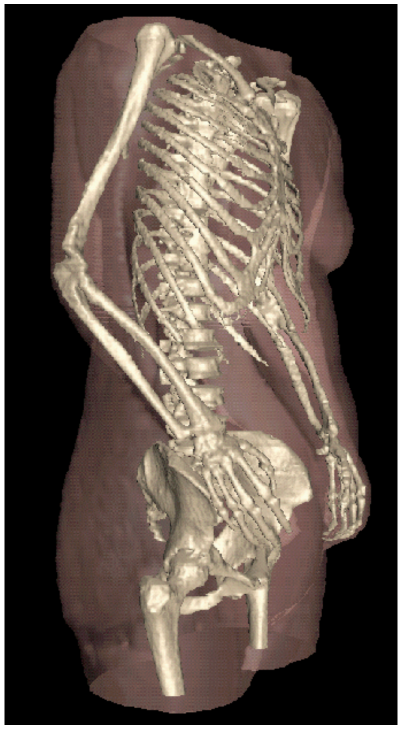
- for some data, difficult to create polygonal mesh
- voxels: discrete representation of 3D object
  - volume rendering: create 2D image from 3D object
- translate raw densities into colors and transparencies
  - different aspects of the dataset can be emphasized via changes in transfer functions

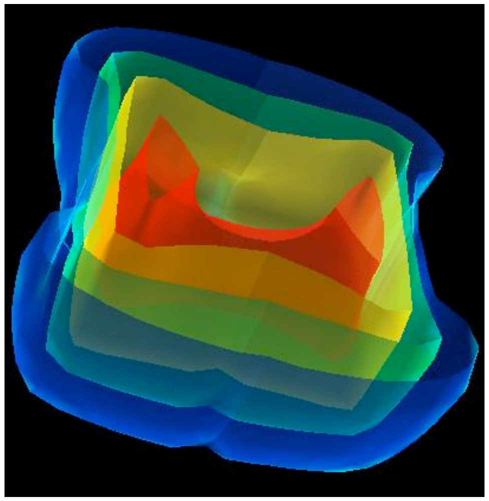


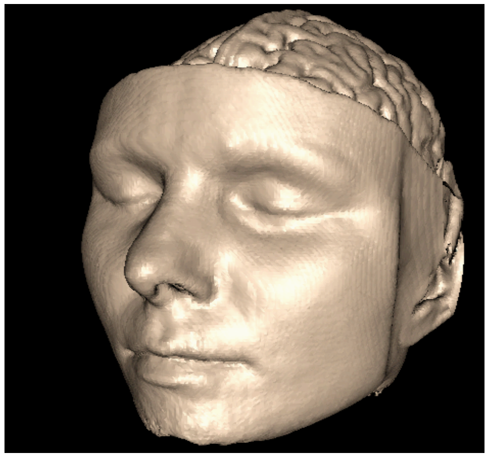
Review: Isosurfaces

- 2D scalar fields: isolines
  - contour plots, level sets
  - topographic maps
- 3D scalar fields: isosurfaces



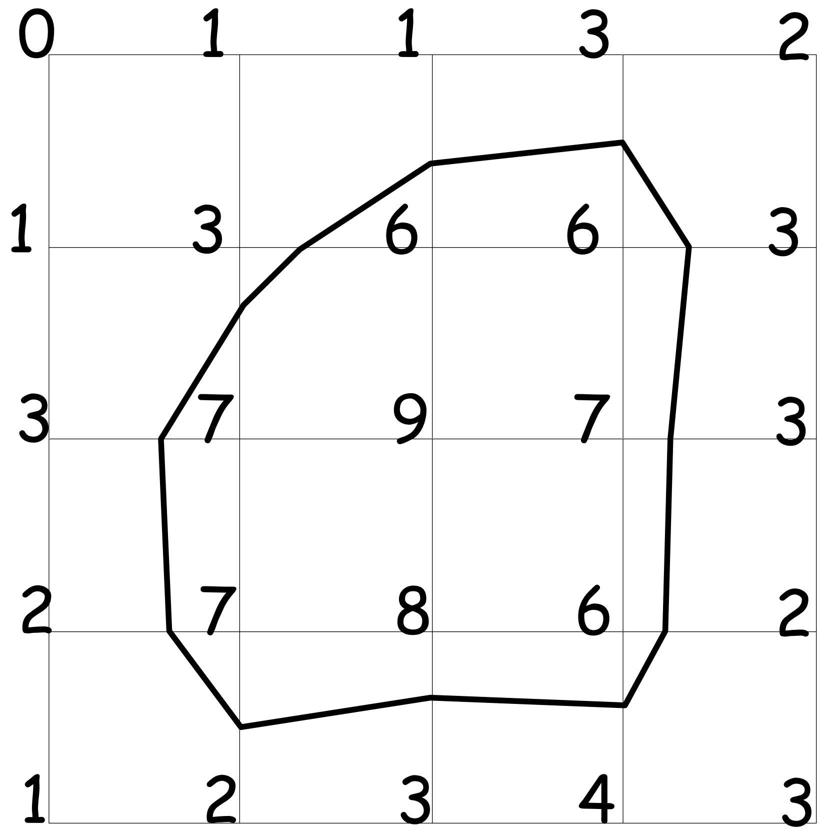






Review: Isosurface Extraction

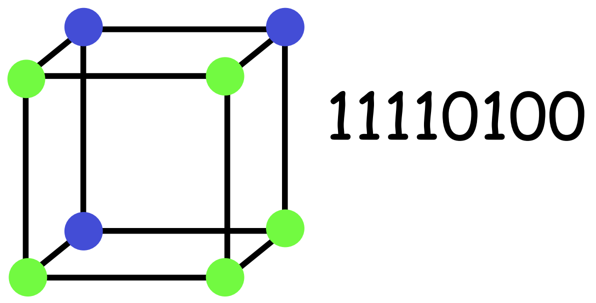
- array of discrete point samples at grid points
  - 3D array: voxels
- find contours
  - closed, continuous
  - determined by iso-value
- several methods
  - marching cubes is most common

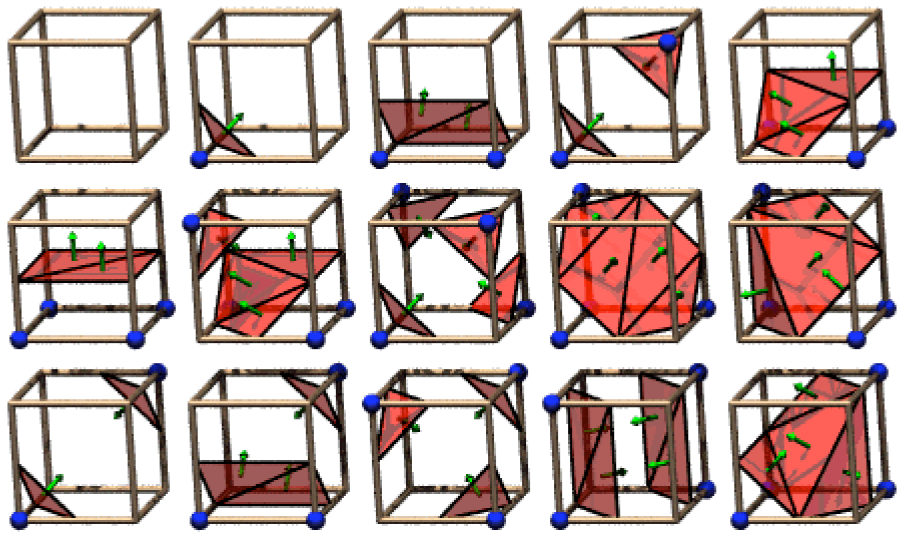


Iso-value = 5

Review: Marching Cubes

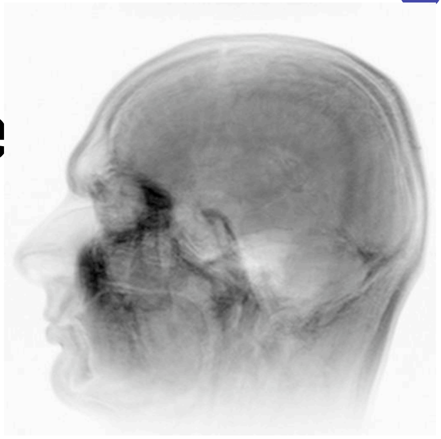
- create cube
- classify each voxel
- binary labeling of each voxel to create index
- use in array storing edge list
  - all 256 cases can be derived from 15 base cases
- interpolate triangle vertex
- calculate the normal at each cube vertex
- render by standard methods

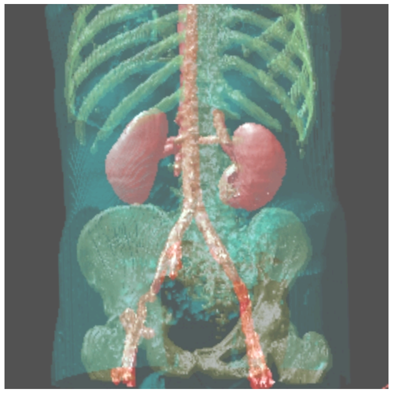


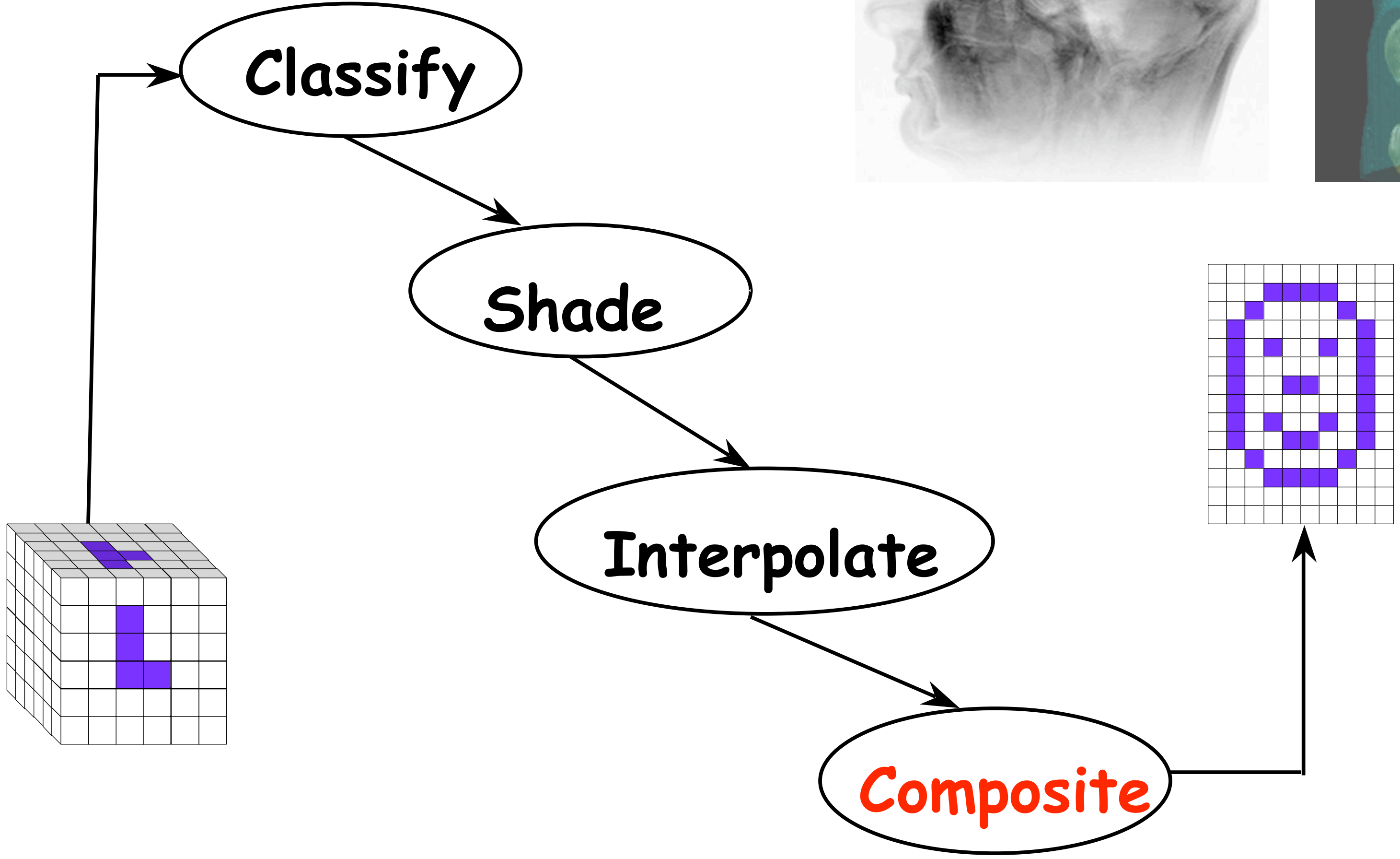

The 15 Cube Combinations

Review: Direct Volume Rendering Pipeline

- do not compute surface

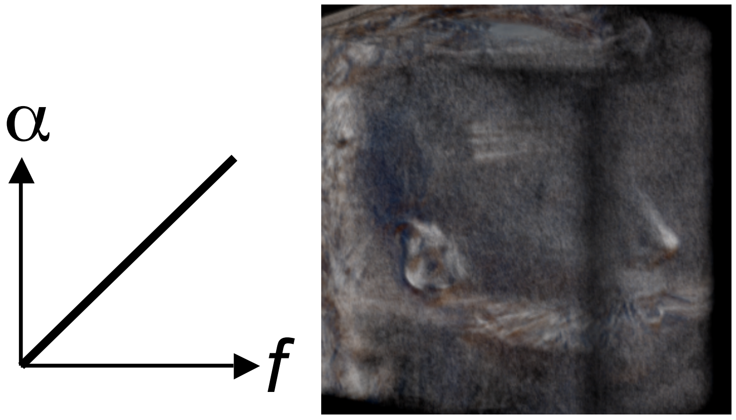


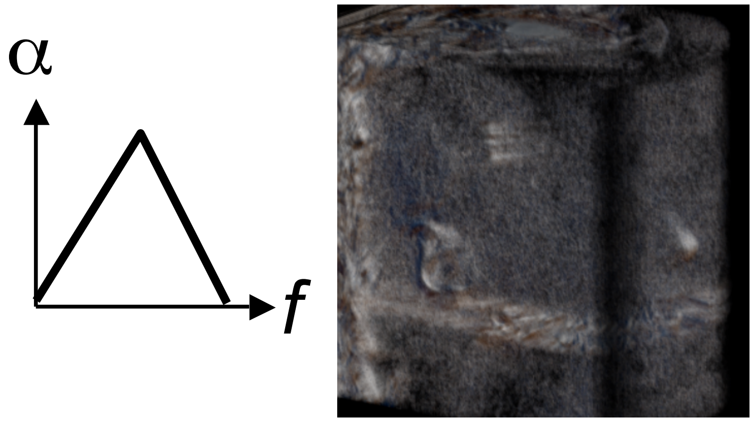


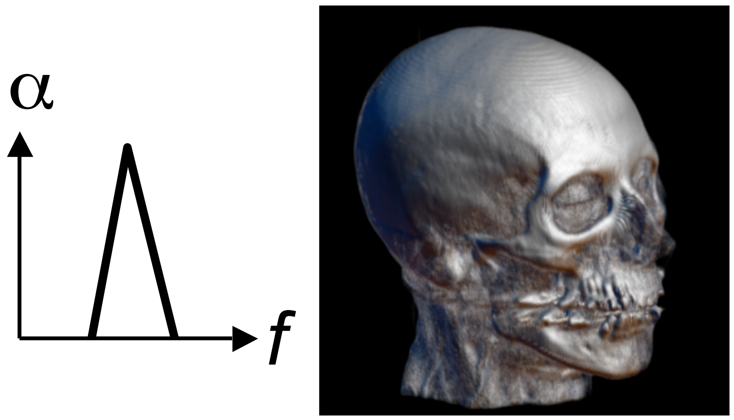


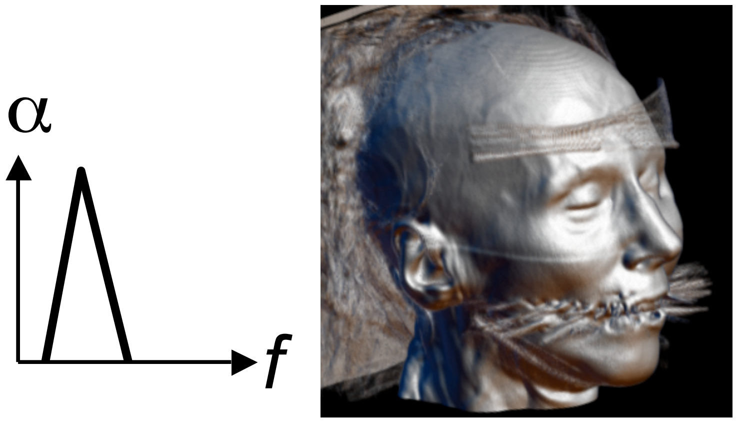
Review: Transfer Functions To Classify

- map data value to color and opacity
  - can be difficult, unintuitive, and slow



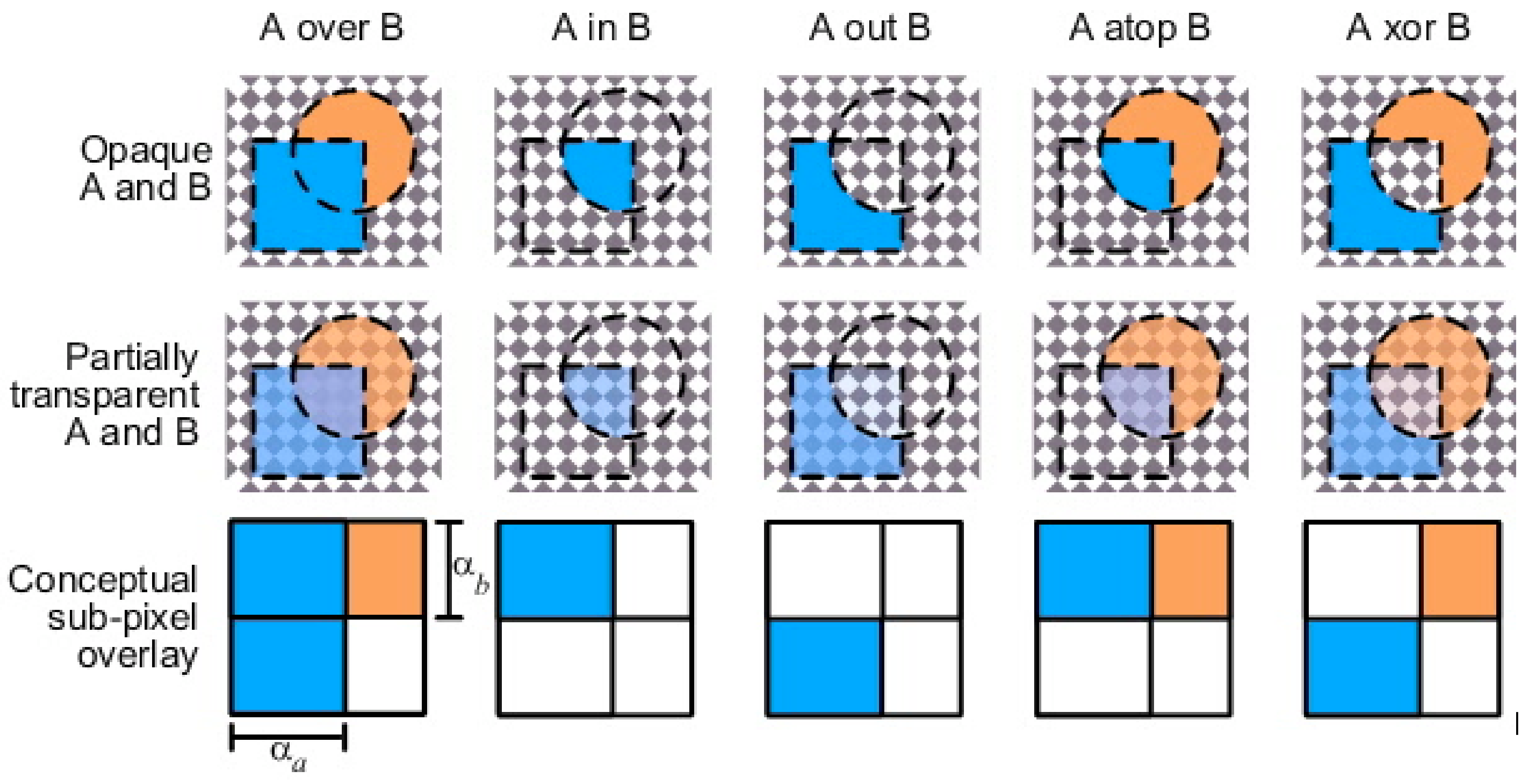






Gordon Kindlmann

Review: Compositing



Review: Premultiplying Colors

- specify opacity with alpha channel: (r,g,b,α)
  - $\alpha=1$: opaque, $\alpha=.5$: translucent, $\alpha=0$: transparent

- **A** *over* **B**
  - $\mathbf{C} = \alpha\mathbf{A} + (1-\alpha)\mathbf{B}$

- premultiplying by alpha
  - $\mathbf{C'} = \gamma\,\mathbf{C},\ \mathbf{B'} = \beta\mathbf{B},\ \mathbf{A'} = \alpha\mathbf{A}$

  - $\mathbf{C'} = \mathbf{B'} + \mathbf{A'} - \alpha\mathbf{B'}$
  - $\gamma = \beta + \alpha - \alpha\beta$
    - 1 multiply to find C, same equations for alpha and RGB

Review: Rendering Pipeline

- so far rendering pipeline as a specific set of stages with **fixed functionality**
- modern graphics hardware more flexible
  - programmable “vertex shaders” replace several geometry processing stages
  - programmable “fragment/pixel shaders” replace texture mapping stage
  - hardware with these features now called Graphics Processing Unit (GPU)
- program shading hardware with assembly language analog, or high level shading language

Review: Vertex Shaders

- replace model/view transformation, lighting, perspective projection
- a little assembly-style program is executed on every individual vertex independently
- it sees:
  - vertex attributes that change per vertex:
    - position, color, texture coordinates…
  - registers that are constant for all vertices (changes are expensive):
    - matrices, light position and color, …
  - temporary registers
  - output registers for position, color, tex coords…

Review: Fragment Shaders

- fragment shaders operate on fragments in place of texturing hardware
  - after rasterization
  - before any fragment tests or blending
- input: fragment, with screen position, depth, color, and set of texture coordinates
- access to textures, some constant data, registers
- compute RGBA values for fragment, and depth
  - can also kill a fragment (throw it away)

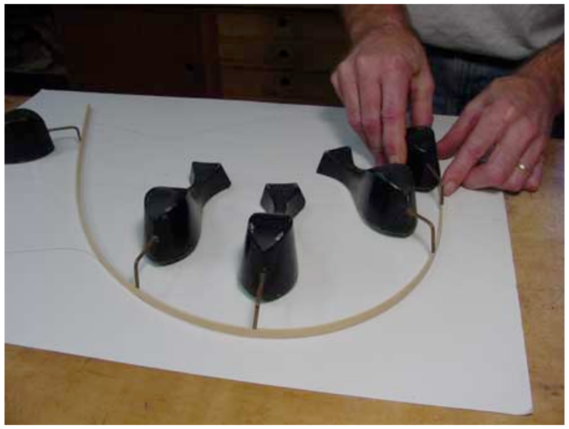
Review: GPGPU Programming

- General Purpose GPU
  - use graphics card as SIMD parallel processor
  - textures as arrays
  - computation: render large quadrilateral
  - multiple rendering passes

Review: Splines

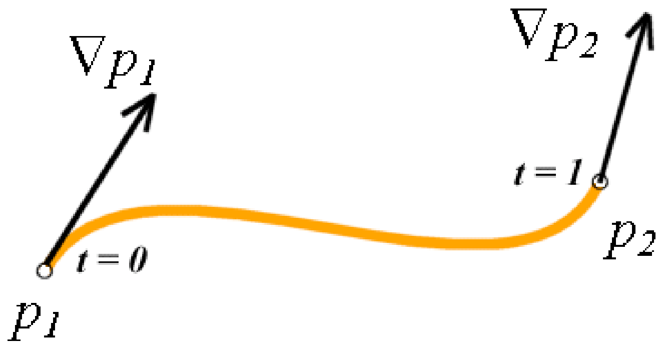
- *spline* is parametric curve defined by *control points*
  - *knots:* control points that lie on curve
  - engineering drawing: spline was flexible wood, control points were physical weights

A Duck (weight)


Ducks trace out curve

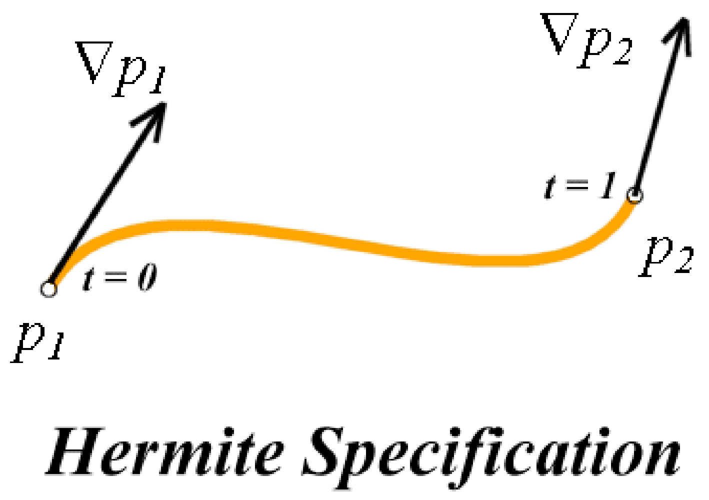
Review: Hermite Spline

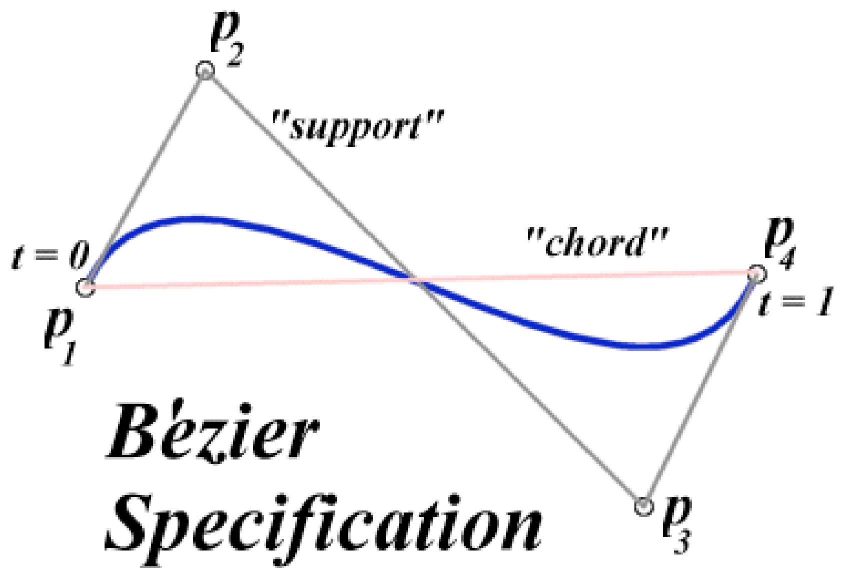
- user provides
  - endpoints
  - derivatives at endpoints



Review: Bézier Curves

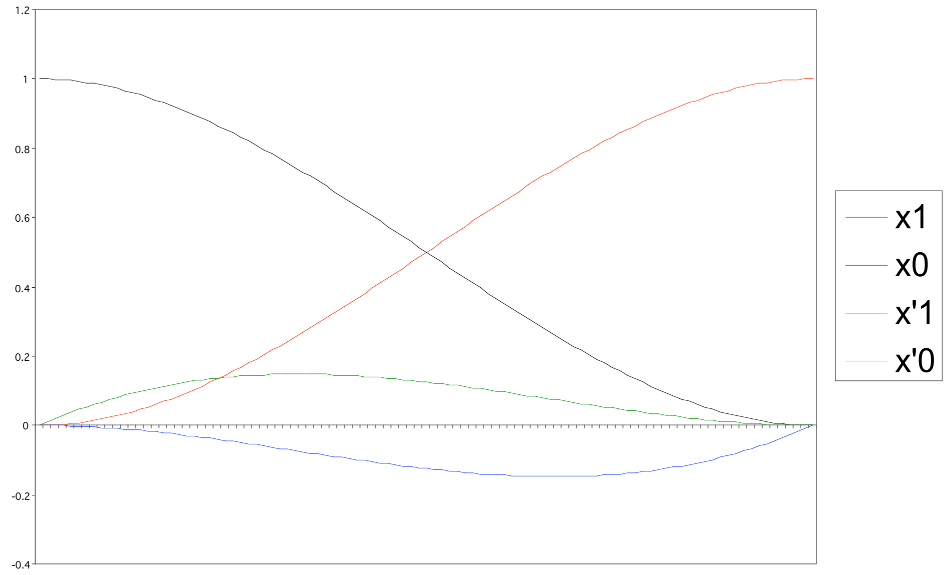
- four control points, two of which are knots
  - more intuitive definition than derivatives
- curve will always remain within convex hull (bounding region) defined by control points





Review: Basis Functions

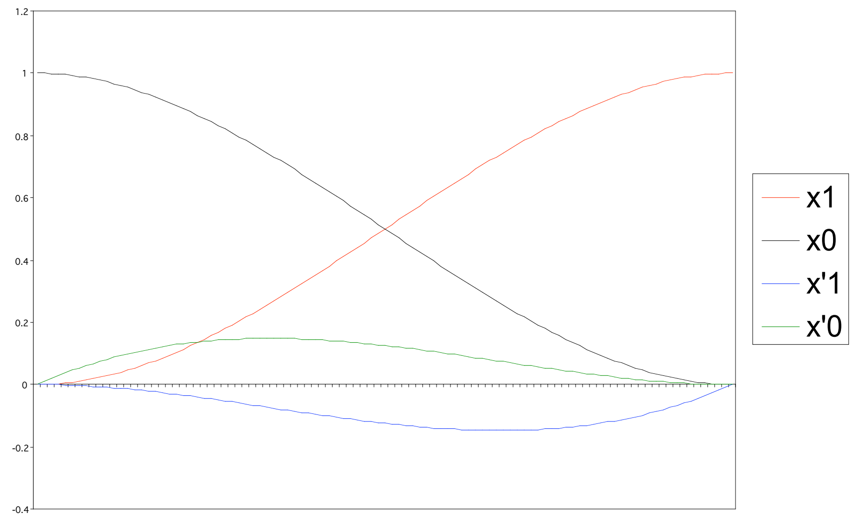
- point on curve obtained by multiplying each control point by some basis function and summing

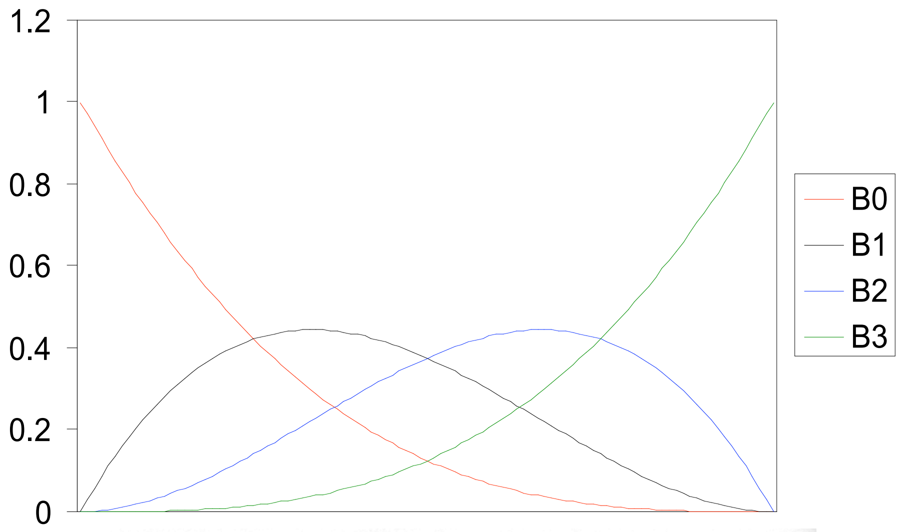


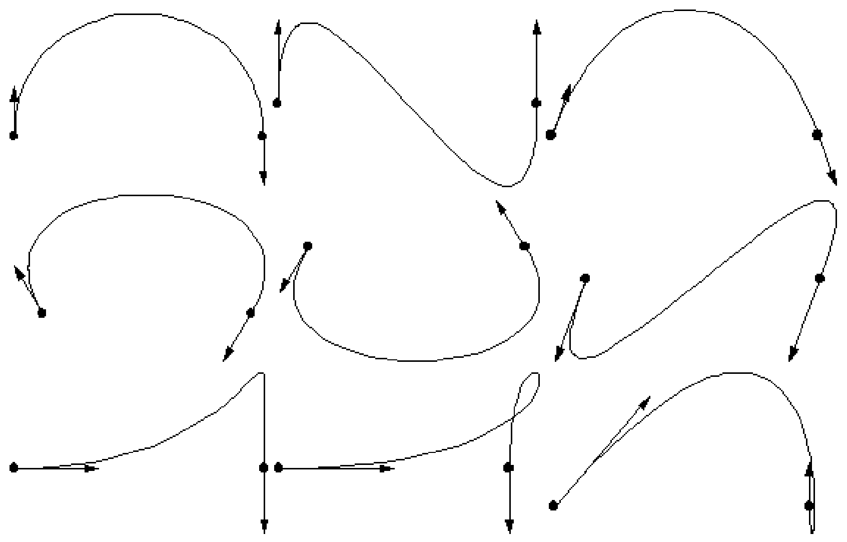
Review: Comparing Hermite and Bézier

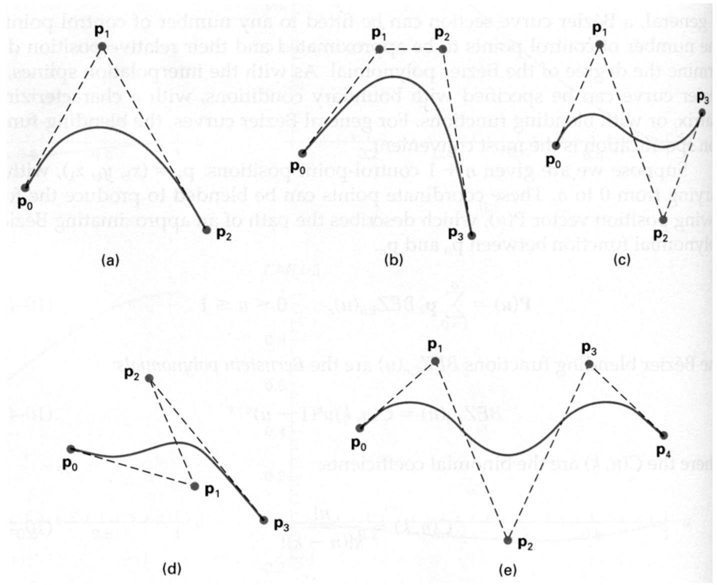
Hermite

Bézier



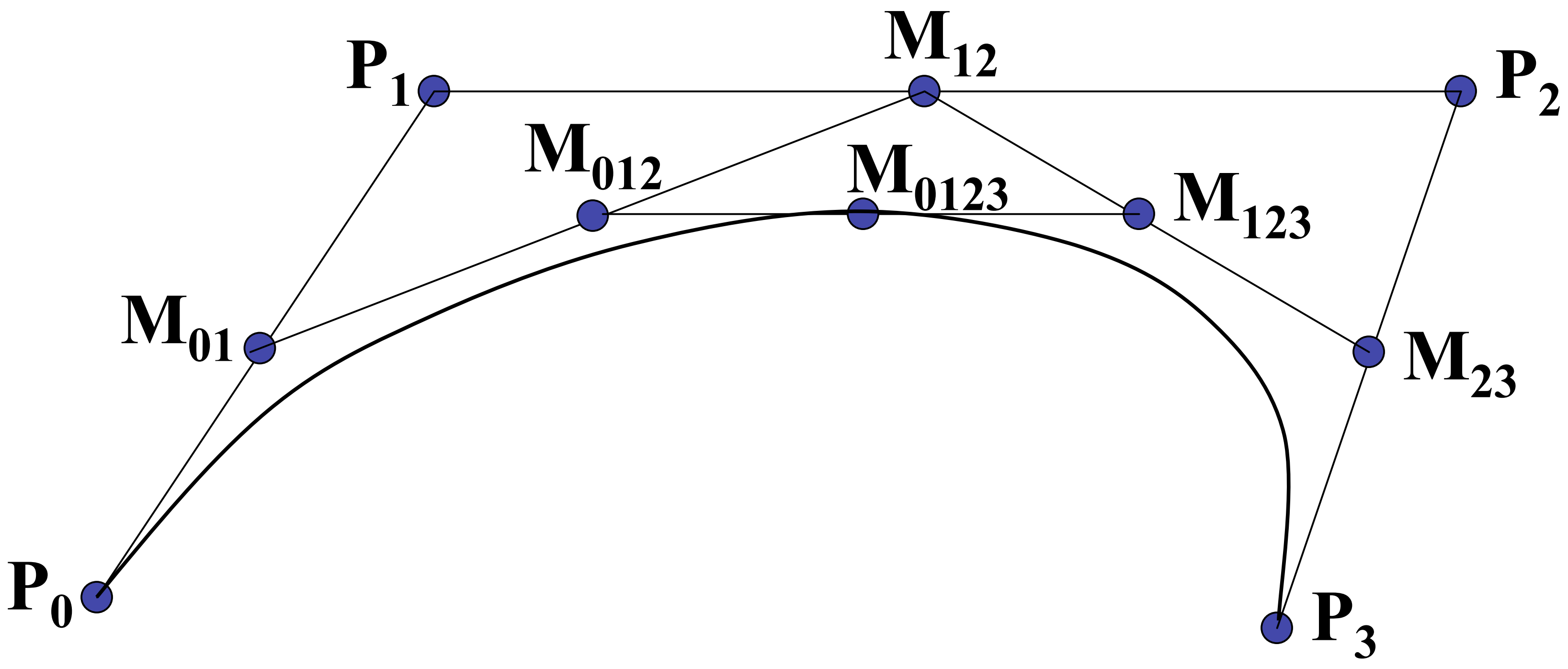






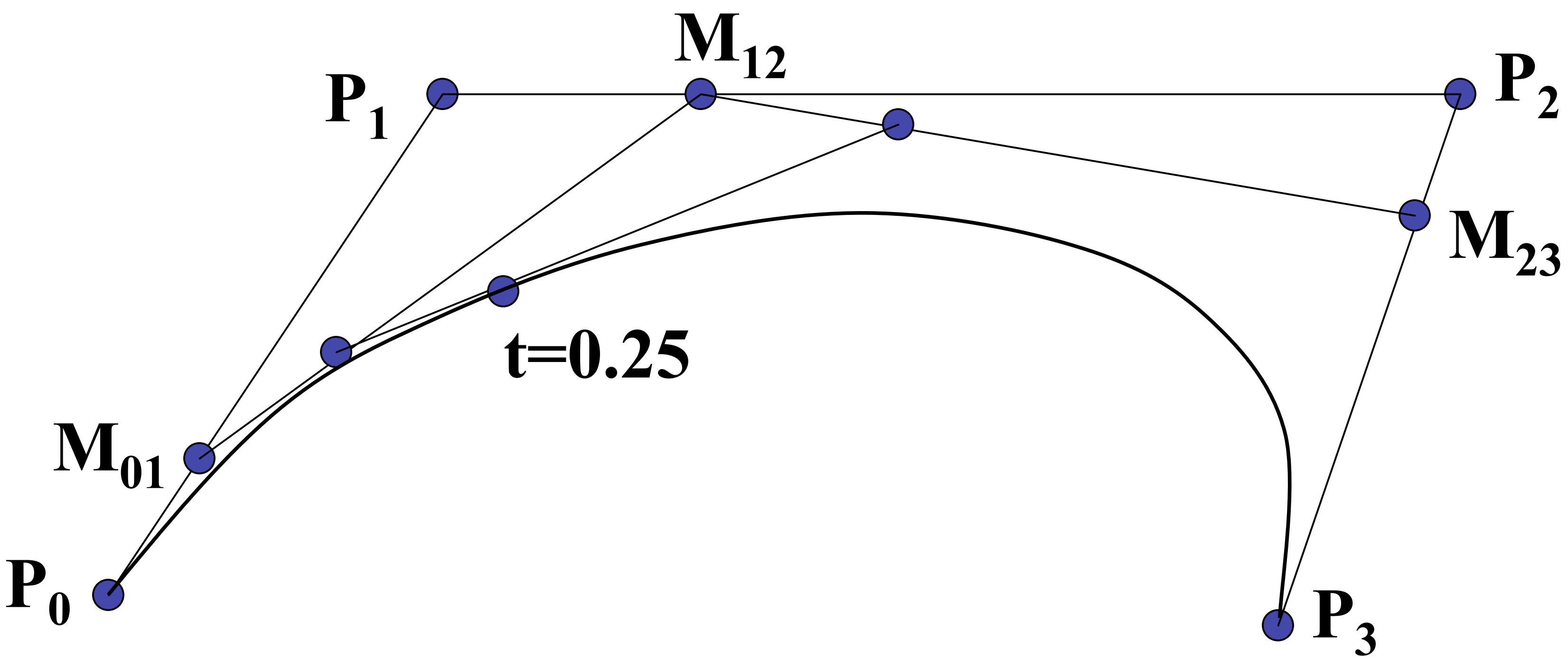
Review: Sub-Dividing Bézier Curves

- find the midpoint of the line joining M_{012}, M_{123}. call it M_{0123}



Review: de Casteljau's Algorithm

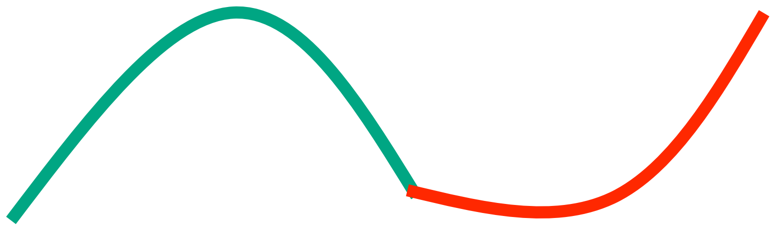
- can find the point on Bézier curve for any parameter value *t* with similar algorithm
  - for *t=0.25*, instead of taking midpoints take points 0.25 of the way



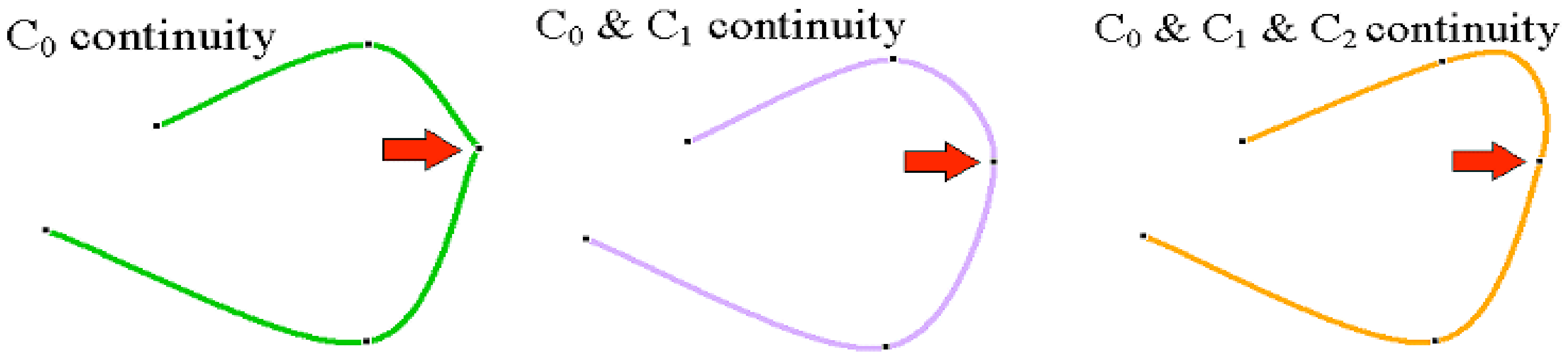
demo: www.saltire.com/applets/advanced_geometry/spline/spline.htm

Review: Continuity

- piecewise Bézier: no continuity guarantees

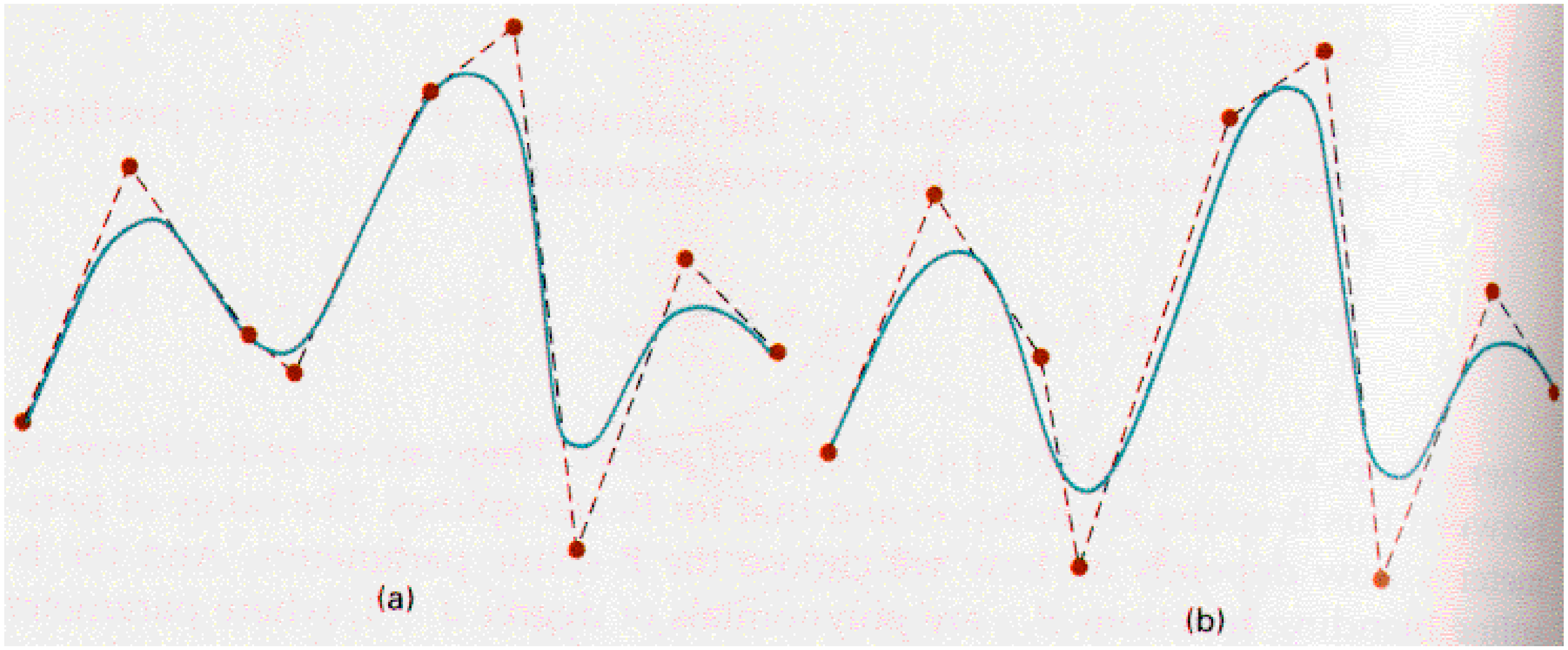


- continuity definitions
  - C^0: share join point
  - C^1: share continuous derivatives
  - C^2: share continuous second derivatives



Review: B-Spline

- C_0, C_1, and C_2 continuous
- piecewise: locality of control point influence



Beyond 314: Other Graphics Courses

- 424: Geometric Modelling
  - was offered this year
- 426: Computer Animation
  - will be offered next year

- 514: Image-Based Rendering - Heidrich
- 526: Algorithmic Animation - van de Panne
- 533A: Digital Geometry - Sheffer
- 533B: Animation Physics - Bridson
- 533C: Information Visualization - Munzner