University of British Columbia
CPSC 314 Computer Graphics
Jan-Apr 2013

Tamara Munzner

Final Review

http://www.ugrad.cs.ubc.ca/~cs314/Vjan2013

Final

- exam notes
  - 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
    - ok to staple midterm sheet + new one back to back
  - calculator: a good idea, but not required
    - graphical OK, smartphones etc not ok
  - IDs out and face up

Final Emphasis

- covers entire course
- includes material from before midterm
  - transformations, viewing/picking
- but heavier weighting for material after last midterm

- post-midterm topics:
  - lighting/shading
  - advanced rendering
  - collision
  - rasterization
  - hidden surfaces / blending
  - textures/procedural
  - clipping
  - color
  - curves
  - visualization

Sample Final

- solutions now posted
  - Spring 06-07 (label was off by one)
- note some material not covered this time
  - projection types like cavalier/cabinet
    - Q1b, Q1c,
  - antialiasing
    - Q1d, Q1l, Q12
  - animation
  - image-based rendering
    - Q1g
  - scientific visualization
    - Q14

Studying Advice

- do problems!
  - work through old homeworks, exams

Reading from OpenGL Red Book

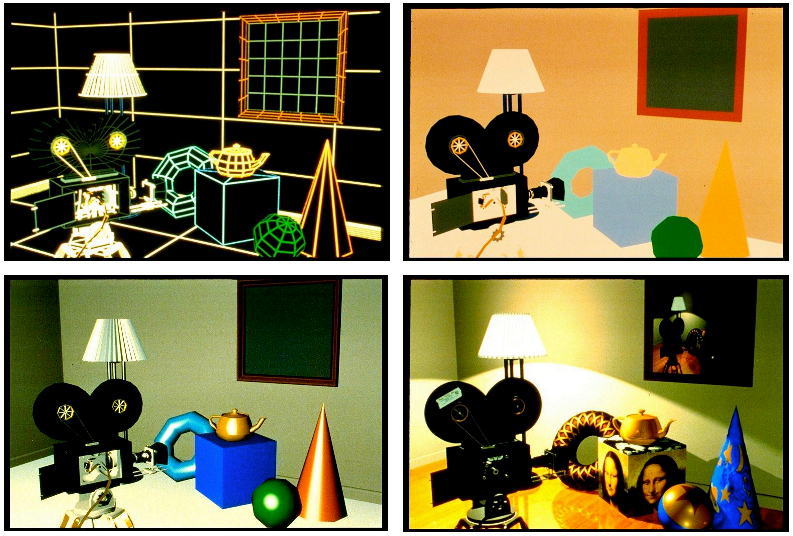
- 1: Introduction to OpenGL
- 2: State Management and Drawing Geometric Objects
- 3: Viewing
- 4: Display Lists
- 5: Color
- 6: Lighting
- 9: Texture Mapping
- 12: Selection and Feedback
- 13: Now That You Know
  - only section Object Selection Using the Back Buffer
- Appendix: Basics of GLUT (Aux in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

Reading from Shirley: Foundations of CG

- 1: Intro *
- 2: Misc Math *
- 3: Raster Algs *
  - through 3.3
- 4: Ray Tracing *
- 5: Linear Algebra *
  - except for 5.4
- 6: Transforms *
  - except 6.1.6
- 7: Viewing *
- 8: Graphics Pipeline *
  - 8.1 through 8.1.6, 8.2.3-8.2.5, 8.2.7, 8.4
- 10: Surface Shading *
- 11: Texture Mapping *
- 13: More Ray Tracing *
  - only 13.1
- 12: Data Structures *
  - only 12.2-12.4
- 15: Curves and Surfaces *
- 17: Computer Animation *
  - only 17.6-17.7
- 21: Color *
- 22: Visual Perception *
  - only 22.2.2 and 22.2.4
- 27: Visualization *

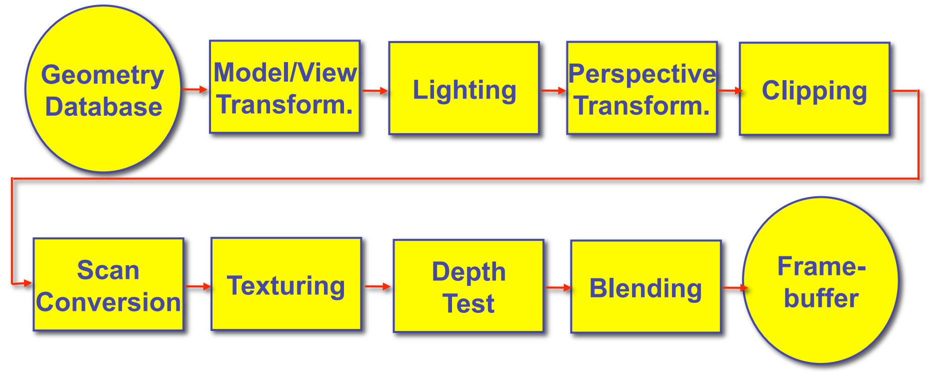
Review – Fast!!

Review: Rendering Capabilities



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

Review: Rendering Pipeline



Review: OpenGL

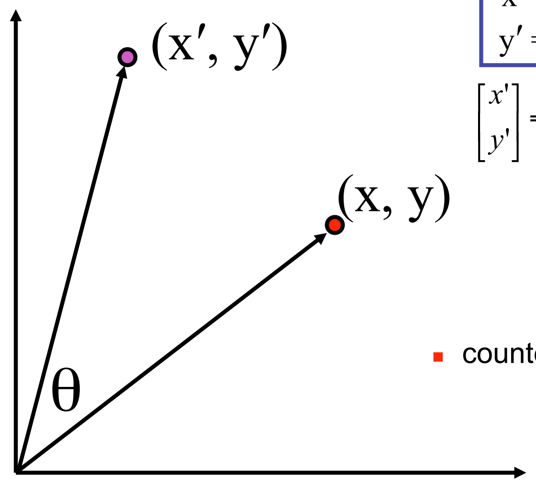
- pipeline processing, set state as needed

```
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

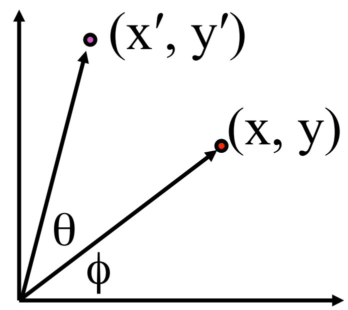


$x' = x\ \mathbf{cos}(\theta) - y\ \mathbf{sin}(\theta)$
$y' = x\ \mathbf{sin}(\theta) + y\ \mathbf{cos}(\theta)$

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

- counterclockwise, RHS

Review: 2D Rotation From Trig Identities

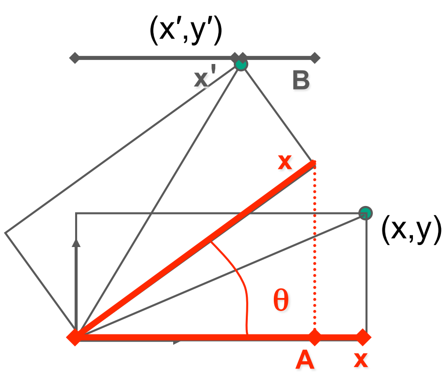


$x = r\cos(\phi)$
$y = r\sin(\phi)$
$x' = r\cos(\phi + \theta)$
$y' = r\sin(\phi + \theta)$

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

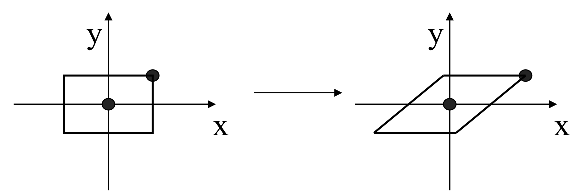


$$x' = x\cos\theta - y\sin\theta$$
$$y' = x\sin\theta + y\cos\theta$$

$$x' = A - B$$
$$A = x\cos\theta$$

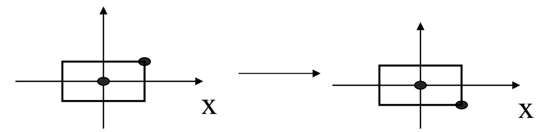
Review: Shear, Reflection

- 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

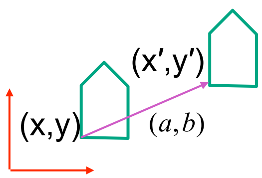
$$\begin{bmatrix}x'\\y'\end{bmatrix}=\begin{bmatrix}a&0\\0&b\end{bmatrix}\begin{bmatrix}x\\y\end{bmatrix}$$

scaling matrix

matrix multiplication

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

rotation matrix



vector addition

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

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

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**x**+t**y**) = s T(**x**) + t T(**y**)
  - origin maps to origin
  - lines map to lines
  - parallel lines remain parallel
  - ratios are preserved
  - closed under composition

Review: Affine Transformations

- affine transforms are combinations of
  - linear transformations
  - translations

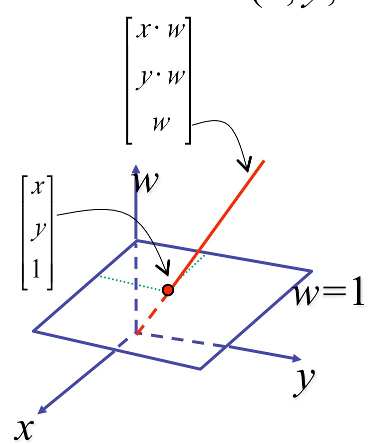
$$\begin{bmatrix}x'\\y'\\w\end{bmatrix}=\begin{bmatrix}a&b&c\\d&e&f\\0&0&1\end{bmatrix}\begin{bmatrix}x\\y\\w\end{bmatrix}$$

- 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: Homogeneous Coordinates

homogeneous — cartesian

$$(x,y,w) \xrightarrow{/\,w} \left(\frac{x}{w},\frac{y}{w}\right)$$



- homogenize to convert homog. 3D point to cartesian 2D point:
  - divide by w to get (x/w, y/w, 1)
  - projects line to point onto w=1 plane
  - like normalizing, one dimension up
- when w=0, consider it as direction
  - points at infinity
  - these points cannot be homogenized
  - lies on x-y plane
- (0,0,0) is undefined

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)

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

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

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

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

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

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

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

Review: 3D Shear

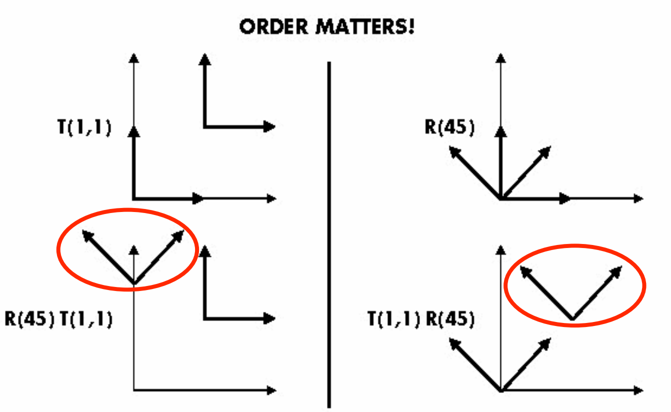
- general shear $shear(hxy, hxz, hyx, hyz, hzx, hzy) = \begin{bmatrix} 1 & hyx & hzx & 0 \\ hxy & 1 & hzy & 0 \\ hxz & hyz & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
- "x-shear" usually means shear along x in direction of some other axis
  - **correction: not** shear along some axis in direction of x
  - 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: Composing Transformations



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

- translations commute
- rotations around same axis commute
- rotations around different axes do not commute
- rotations and translations do not commute

Review: Composing Transformations

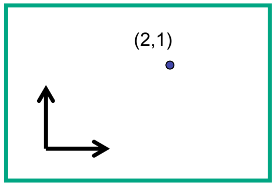
$$\mathbf{p'} = \mathbf{TRp}$$

- 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

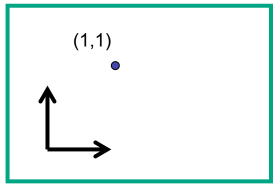
Review: Interpreting Transformations

$$\mathbf{p}' = \mathbf{TRp}$$

translate by (-1,0)

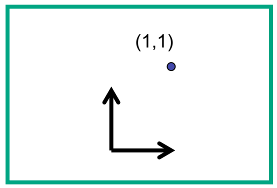


right to left: moving object



intuitive?

left to right: changing coordinate system



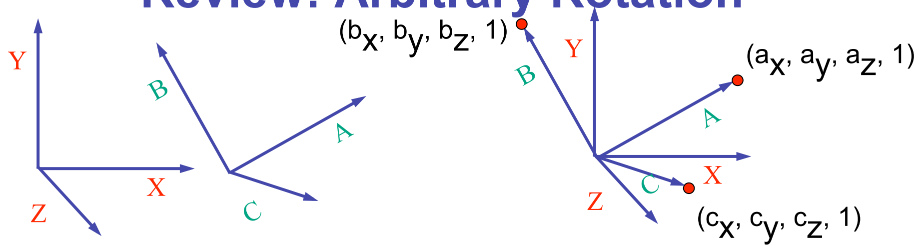
OpenGL

- same relative position between object and basis vectors

Review: General Transform Composition

- transformation of geometry into coordinate system where operation becomes simpler
  - typically translate to origin
- perform operation
- transform geometry back to original coordinate system

Review: Arbitrary Rotation

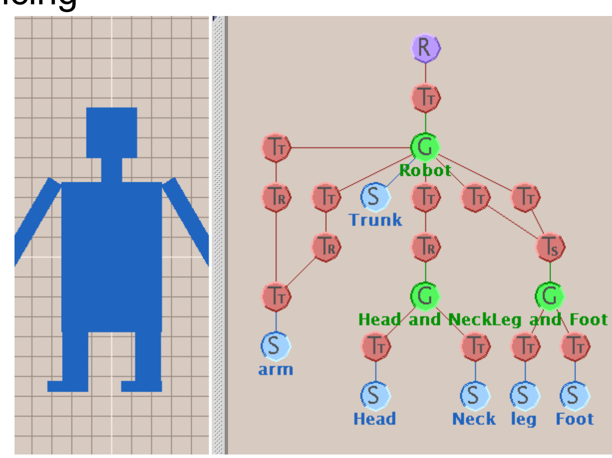


- arbitrary rotation: change of basis
  - given two orthonormal coordinate systems *XYZ* and *ABC*
    - *A*'s location in the XYZ coordinate system is $(a_x, a_y, a_z, 1)$, ...
- transformation from one to the other is matrix R whose columns are *A,B,C:*

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

Review: Transformation Hierarchies

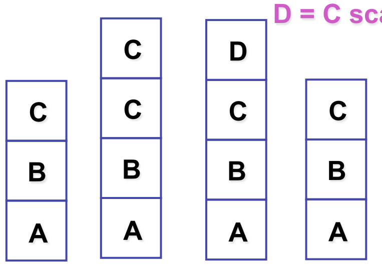
- transforms apply to graph nodes beneath them
- design structure so that object doesn't fall apart
- instancing



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)



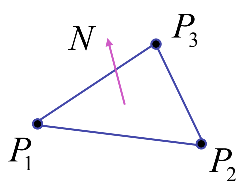

```
DrawSquare()
glPushMatrix()
glScale3f(2,2,2)
glTranslate3f(1,0,0)
DrawSquare()
glPopMatrix()
```


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
  - http://www.lighthouse3d.com/opengl/displaylists

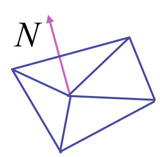
Review: Normals

- polygon:



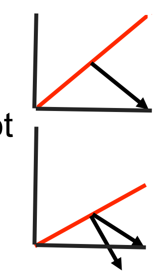
$$N = (P_2 - P_1) \times (P_3 - P_1)$$

- 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!



$$P \rightarrow P' = MP$$
$$N \rightarrow N' = QN$$

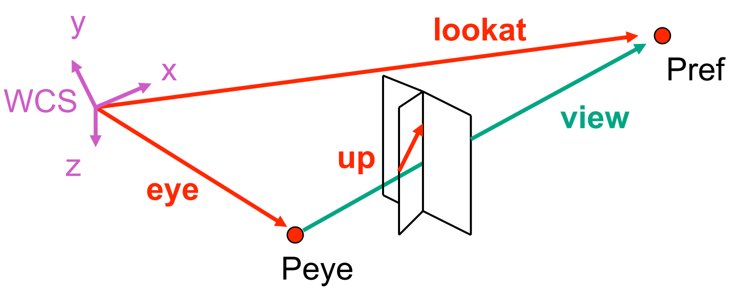
given M, what should Q be?

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

inverse transpose of the modelling transformation

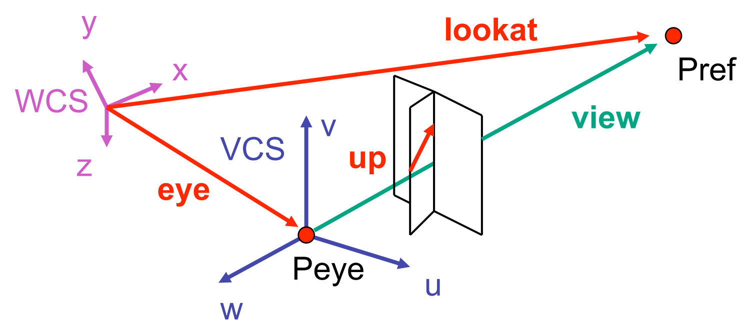
Review: Camera Motion

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



Review: Constructing Lookat

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



Review: V2W vs. W2V

- M_{V2W}=TR

$$\mathbf{T} = \begin{bmatrix} 1 & 0 & 0 & e_x \\ 0 & 1 & 0 & e_y \\ 0 & 0 & 1 & e_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \mathbf{R} = \begin{bmatrix} u_x & v_x & w_x & 0 \\ u_y & v_y & w_y & 0 \\ u_z & v_z & w_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

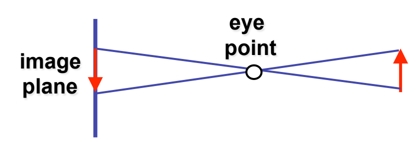
- we derived position of camera as object in world
  - invert for gluLookAt: go from world to camera!
- $M_{W2V}=(M_{V2W})^{-1}=R^{-1}T^{-1}$

$$\mathbf{R}^{-1} = \begin{bmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ w_x & w_y & w_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \mathbf{T}^{-1} = \begin{bmatrix} 1 & 0 & 0 & -e_x \\ 0 & 1 & 0 & -e_y \\ 0 & 0 & 1 & -e_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

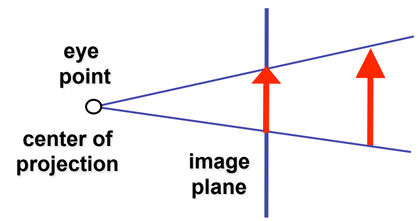
$$\mathbf{M}_{W2V} = \begin{bmatrix} u_x & u_y & u_z & -\mathbf{e}\bullet\mathbf{u} \\ v_x & v_y & v_z & -\mathbf{e}\bullet\mathbf{v} \\ w_x & w_y & w_z & -\mathbf{e}\bullet\mathbf{w} \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} u_x & u_y & u_z & -e_x*u_x+-e_y*u_y+-e_z*u_z \\ v_x & v_y & v_z & -e_x*v_x+-e_y*v_y+-e_z*v_z \\ w_x & w_y & w_z & -e_x*w_x+-e_y*w_y+-e_z*w_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Review: Graphics Cameras

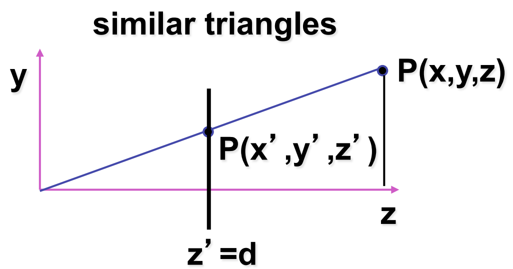
- real pinhole camera: image inverted



- computer graphics camera: convenient equivalent



Review: Basic Perspective Projection

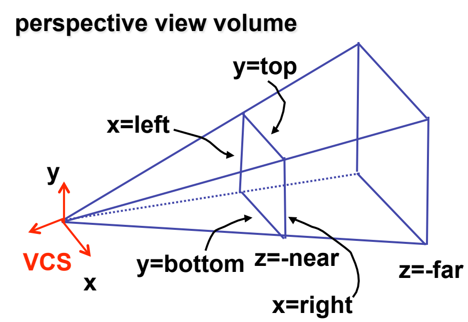


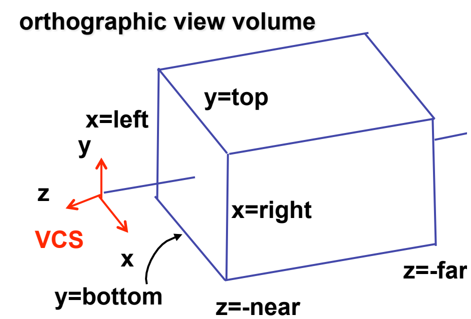
$$\frac{y'}{d} = \frac{y}{z} \rightarrow y' = \frac{y \cdot d}{z}$$

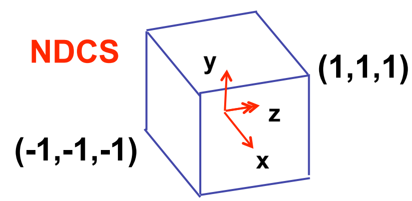
$$x' = \frac{x \cdot d}{z} \qquad z' = d$$

$$\begin{bmatrix} \frac{x}{z/d} \\ \frac{y}{z/d} \\ d \end{bmatrix} \xrightarrow{\text{homogeneous coords}} \begin{bmatrix} x \\ y \\ z \\ z/d \end{bmatrix} \qquad \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix}$$

Review: From VCS to NDCS



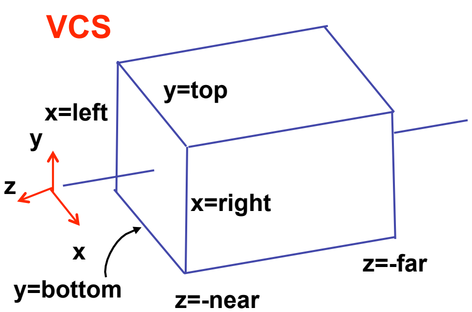


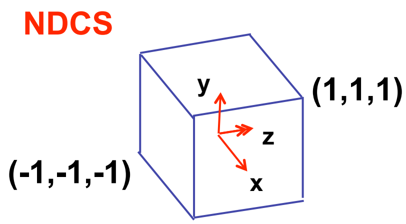


- orthographic camera
  - center of projection at infinity
  - no perspective convergence

Review: Orthographic Derivation

- scale, translate, reflect for new coord sys





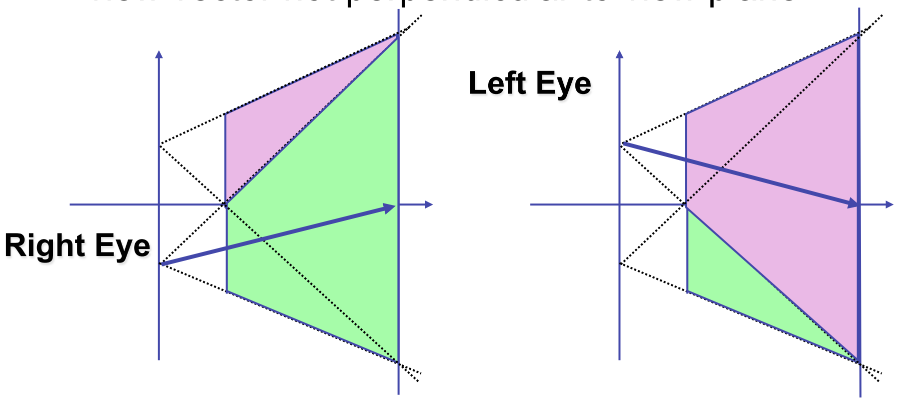
Review: Orthographic Derivation

- scale, translate, reflect for new coord sys

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

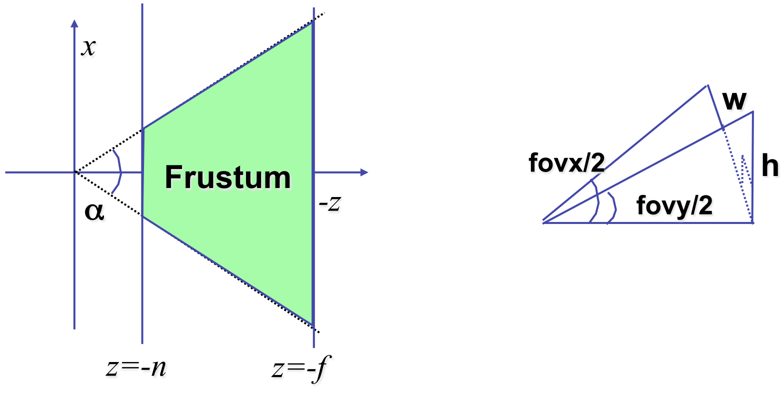
Review: Asymmetric Frusta

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



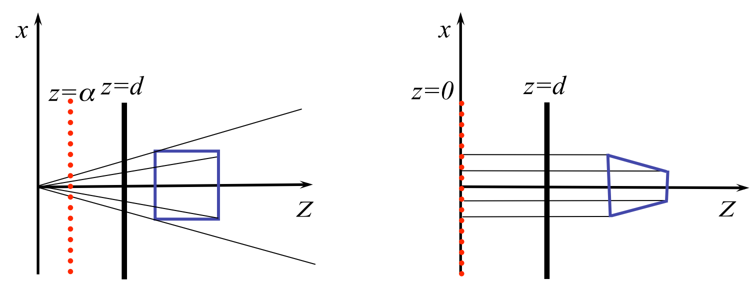
Review: Field-of-View Formulation

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

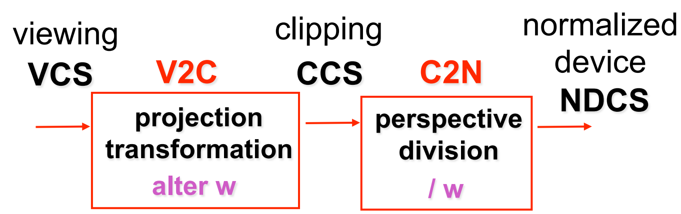


Review: Projection Normalization

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



Review: Separate Warp From Homogenization

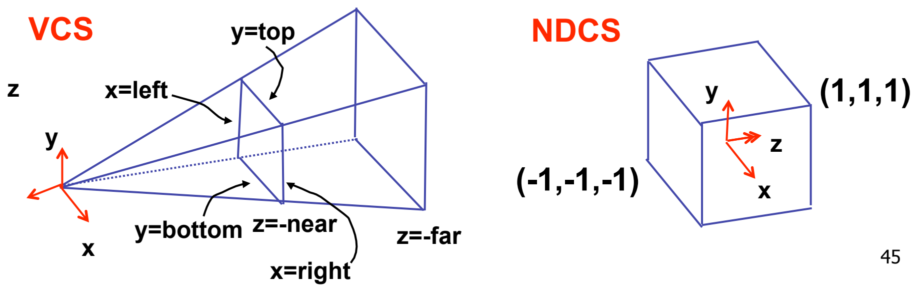


- warp requires only standard matrix multiply
  - distort such that orthographic projection of distorted objects is desired persp projection
    - w is changed
  - clip after warp, before divide
  - division by w: homogenization

Review: Perspective Derivation

- shear
- scale
- projection-normalization

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

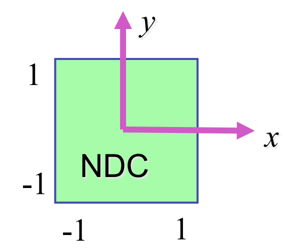


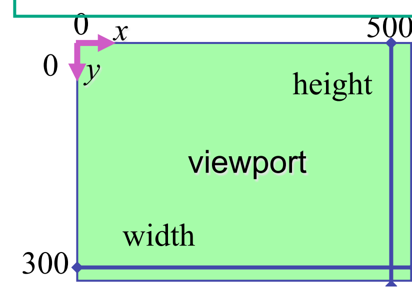
Review: N2D Transformation

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

reminder:
NDC z range is -1 to 1

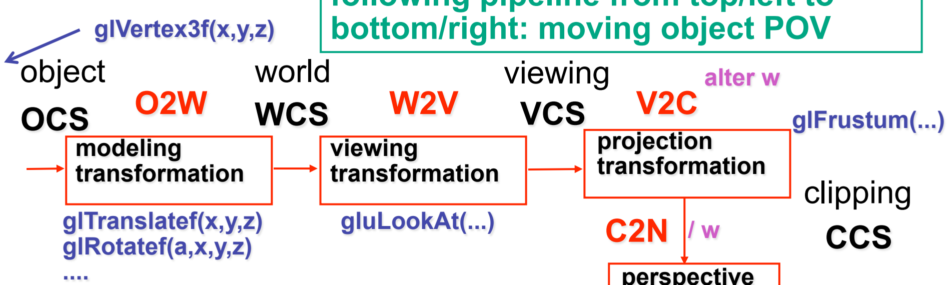
Display z range is 0 to 1. glDepthRange(n,f) can constrain further, but *depth* = 1 is both max and default

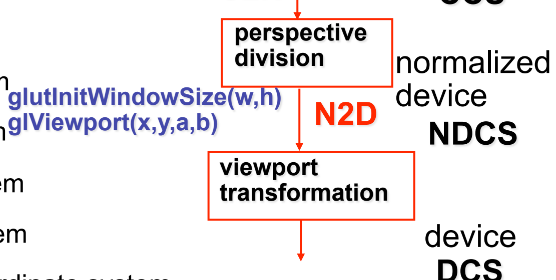




Review: Projective Rendering Pipeline

following pipeline from top/left to bottom/right: moving object POV





OCS - object coordinate system

WCS - world coordinate system

VCS - viewing coordinate system

CCS - clipping coordinate system

NDCS - normalized device coordinate system

DCS - device coordinate system

Review: OpenGL Example

go back from end of pipeline to beginning: coord frame POV!


```
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 );   V2W
WCS   glPushMatrix()
      glTranslate( 4, 4, 0 );   W2O
OCS1  glutSolidTeapot(1);
      glPopMatrix();
      glTranslate( 2, 2, 0);   W2O
OCS2  glutSolidTeapot(1);
```

- transformations that are applied to object first are specified last

Review: Coord Sys: Frame vs Point

read down: transforming between coordinate frames, from frame A to frame B

read up: transforming points, up from frame B coords to frame A coords

OpenGL command order

OpenGL command order			pipeline interpretation
D2N	**DCS** display	glViewport(x,y,a,b)	N2D
N2V	**NDCS** normalized device	glFrustum(...)	V2N
V2W	**VCS** viewing	gluLookAt(...)	W2V
W2O	**WCS** world	glRotatef(a,x,y,z)	O2W
	OCS object	glVertex3f(x,y,z)	

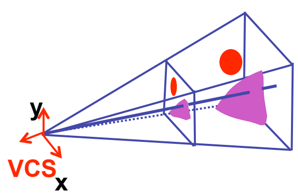
pipeline interpretation

Review: Coord Sys: Frame vs Point

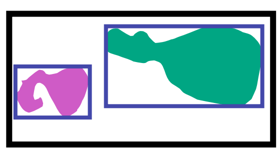
- is gluLookat viewing transformation V2W or W2V? depends on which way you read!
  - coordinate frames: V2W
    - takes you from view to world coordinate frame
  - points/objects: W2V
    - point is transformed from world to view coords when multiply by gluLookAt matrix
- H2 uses the object/pipeline POV
  - Q1/4 is W2V (gluLookAt)
  - Q2/5-6 is V2N (glFrustum)
  - Q3/7 is N2D (glViewport)

Review: Picking Methods

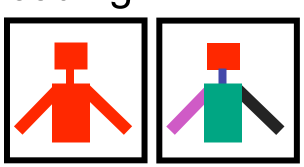
- manual ray intersection

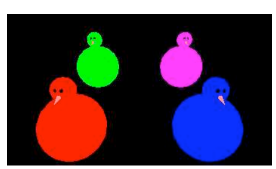


- bounding extents



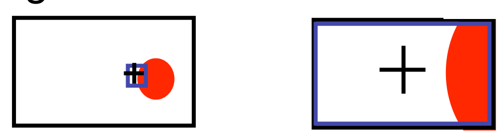
- backbuffer coding





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

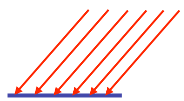
Review: Hit List

- 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

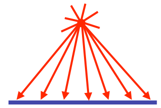
Post-Midterm Material

Review: Light Sources

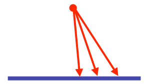
- directional/parallel lights
  - point at infinity: $(x,y,z,0)^T$



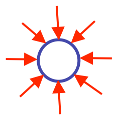
- point lights
  - finite position: $(x,y,z,1)^T$



- spotlights
  - position, direction, angle



- ambient lights

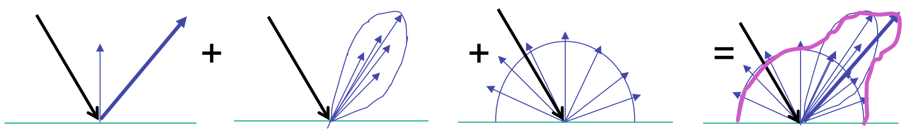


Review: Light Source Placement

- 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

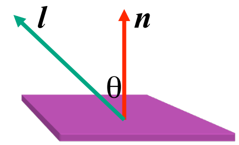
- *specular*: perfect mirror with no scattering
- *gloss*: mixed, partial specularity
- *diffuse*: all directions with equal energy



specular + glossy + diffuse =
reflectance distribution

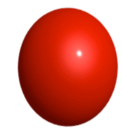
Review: Reflection Equations

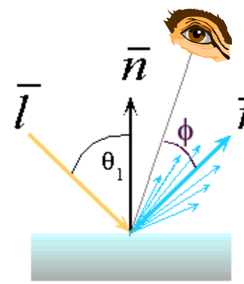
$$\mathbf{I}_{\mathbf{diffuse}} = \mathbf{k_d}\, \mathbf{I}_{\mathbf{light}}\, (\mathbf{n} \bullet \mathbf{l})$$



$$\mathbf{I}_{\mathbf{specular}} = \mathbf{k}_s \mathbf{I}_{\mathbf{light}} (\mathbf{v} \bullet \mathbf{r})^{n_{shiny}}$$

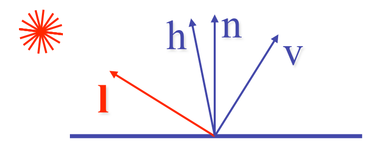
R = 2 (**N** (**N** · **L**)) – **L**





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

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



Review: Reflection Equations

full Phong lighting model

- combine ambient, diffuse, specular components

$$\mathbf{I}_{\mathbf{total}} = \mathbf{k}_a \mathbf{I}_{\mathbf{ambient}} + \sum_{i=1}^{\#lights} \mathbf{I}_i (\mathbf{k}_d (\mathbf{n} \bullet \mathbf{l}_i) + \mathbf{k}_s (\mathbf{v} \bullet \mathbf{r}_i)^{n_{shiny}})$$

- Blinn-Phong lighting

$$\mathbf{I}_{\mathbf{total}} = \mathbf{k}_a \mathbf{I}_{\mathbf{ambient}} + \sum_{i=1}^{\#lights} \mathbf{I}_i (\mathbf{k}_d (\mathbf{n} \bullet \mathbf{l}_i) + \mathbf{k}_s (\mathbf{h} \bullet \mathbf{n}_i)^{n_{shiny}})$$

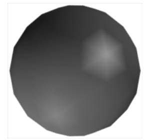
- **don't forget to normalize all lighting vectors!! n,l,r,v,h**

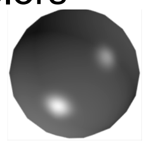
Review: Lighting

- 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

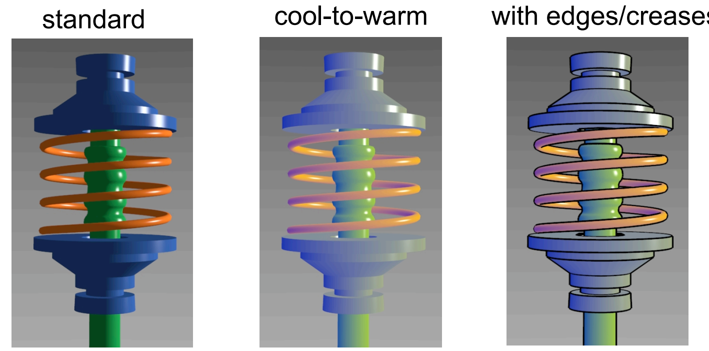
- flat shading
  - for each polygon
    - compute Phong lighting just once
- Gouraud shading
  - compute Phong lighting at the vertices
  - for each pixel in polygon, interpolate colors
- Phong shading
  - for each pixel in polygon
    - interpolate normal
    - perform Phong lighting





Review: Non-Photorealistic Shading

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



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

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

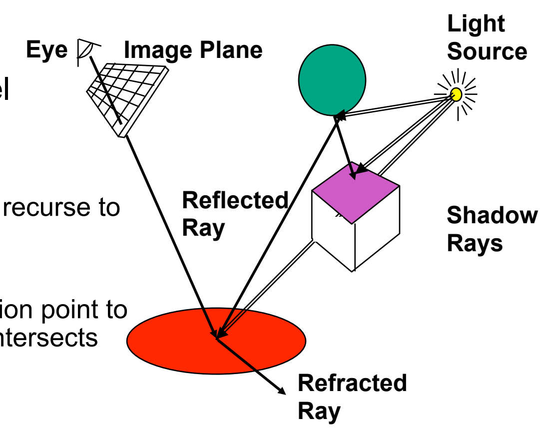
```
glNormal3f(1,1,1);
glVertex3f(3,4,5);
glVertex3f(10,5,2);
```

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

```
glEnable(GL_NORMALIZE);
```

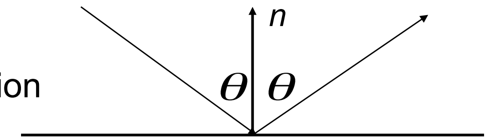

Review: Recursive Ray Tracing

- 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



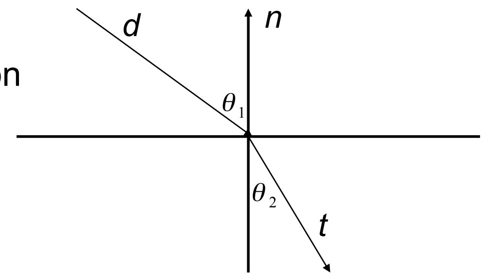
Review: Reflection and Refraction

- reflection: mirror effects
  - perfect specular reflection



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

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

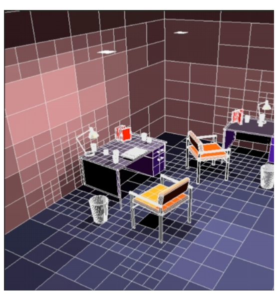


Review: Ray Tracing

- 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)

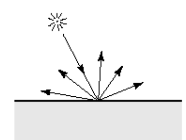


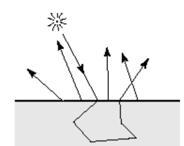
escience.anu.edu.au/lecture/cg/GlobalIllumination/Image/discrete.jpg

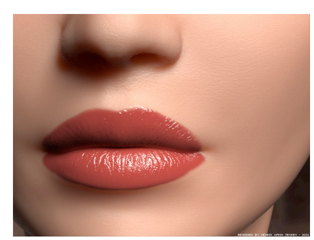
escience.anu.edu.au/lecture/cg/GlobalIllumination/Image/continuous.jpg

Review: Subsurface Scattering

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

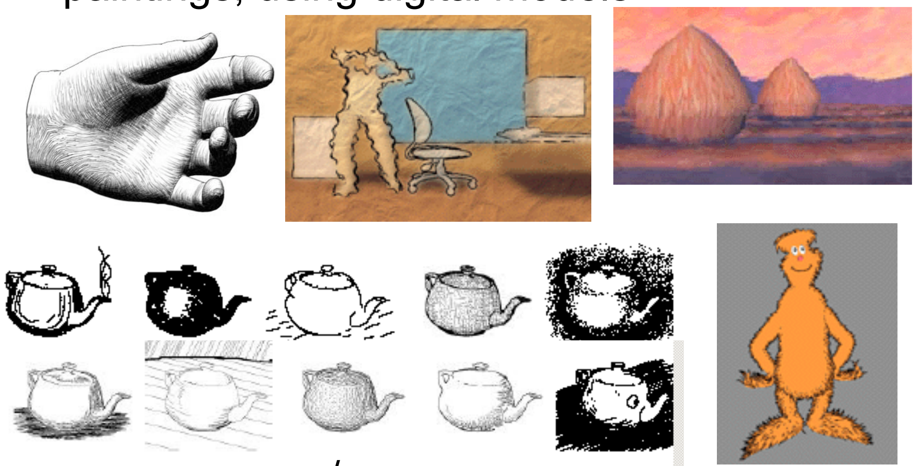






Review: Non-Photorealistic Rendering

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



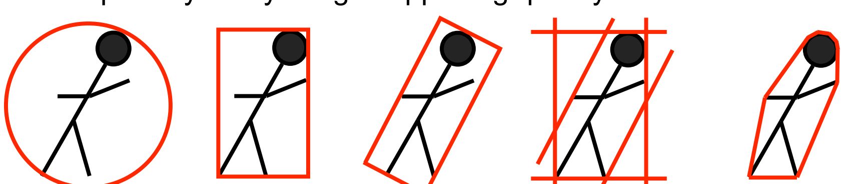
www.red3d.com/cwr/npr/

Review: 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

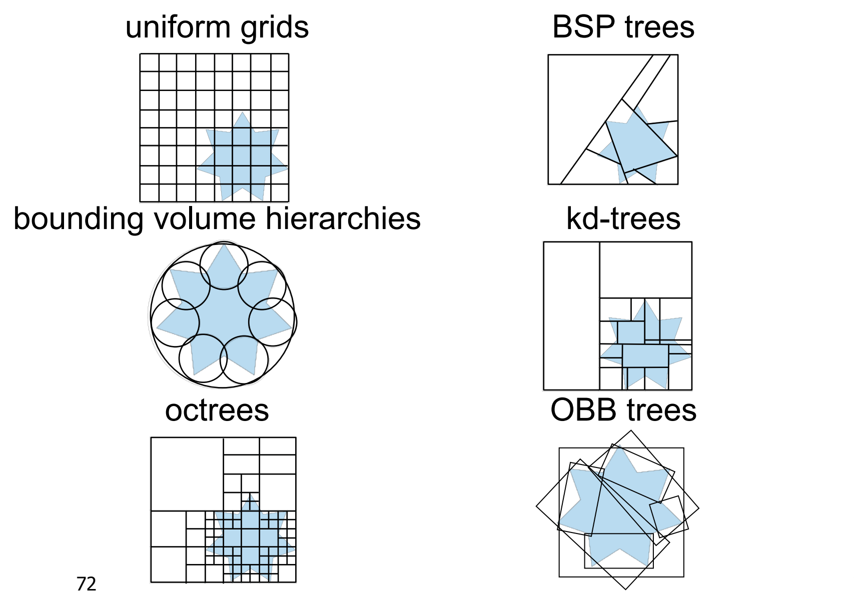


Sphere AABB OBB 6-dop Convex Hull

increasing complexity & tightness of fit →

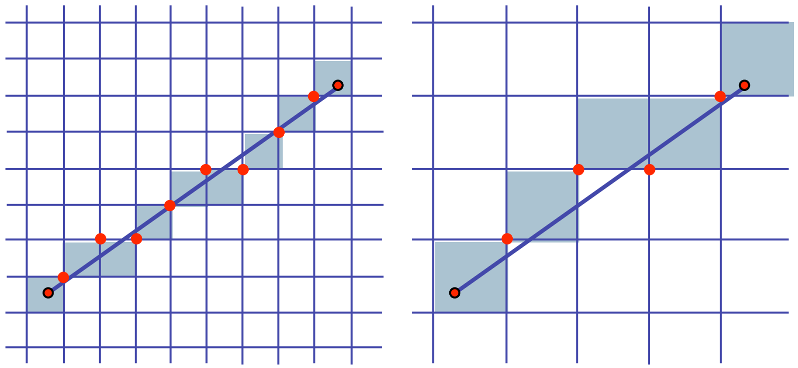
← decreasing cost of (overlap tests + proxy update)

Review: Spatial Data Structures



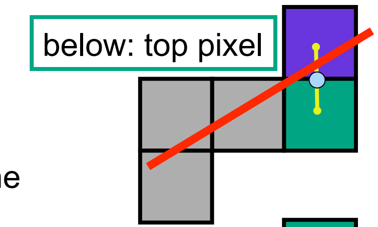
Review: Scan Conversion

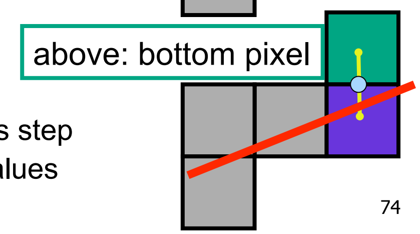
- convert continuous rendering primitives into discrete fragments/pixels
  - given vertices in DCS, fill in the pixels
- display coordinates required to provide scale for discretization



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



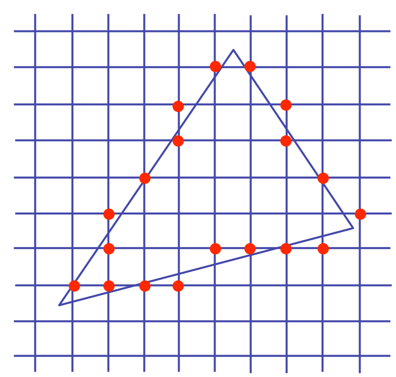


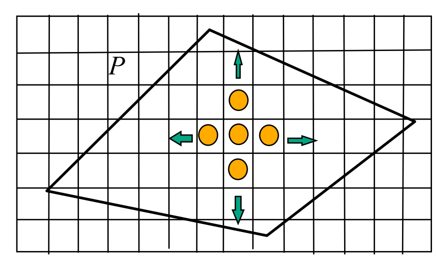
Review: Bresenham - Reuse Computation, Integer Only

```
y=y0;
dx = x1-x0;
dy = y1-y0;
d = 2*dy-dx;
incKeepY = 2*dy;
incIncreaseY = 2*dy-2*dx;
for (x=x0; x <= x1; x++) {
  draw(x,y);
  if (d>0) then {
   y = y + 1;
   d += incIncreaseY;
  } else {
   d += incKeepY;
}
```


Review: Flood Fill

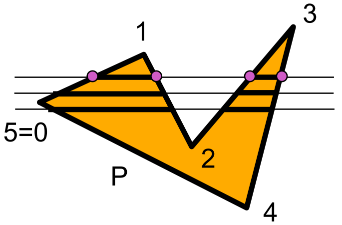
- simple algorithm
  - draw edges of polygon
  - use flood-fill to draw interior

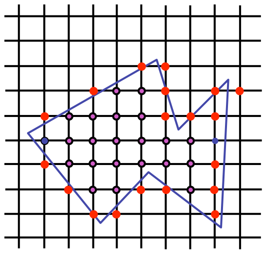


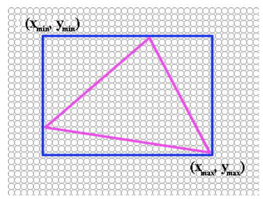


Review: Scanline Algorithms

- 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

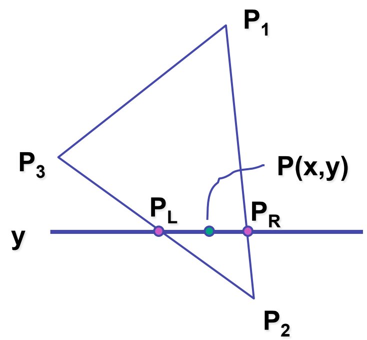






Review: Bilinear Interpolation

- 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)

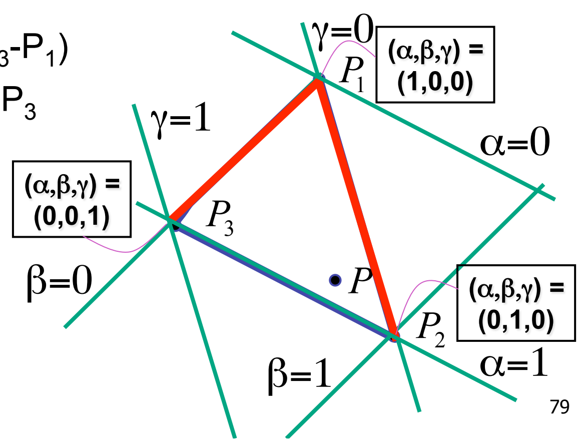
$P = P_1 + \beta(P_2-P_1)+\gamma(P_3-P_1)$

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

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

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

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



Review: Computing Barycentric Coordinates

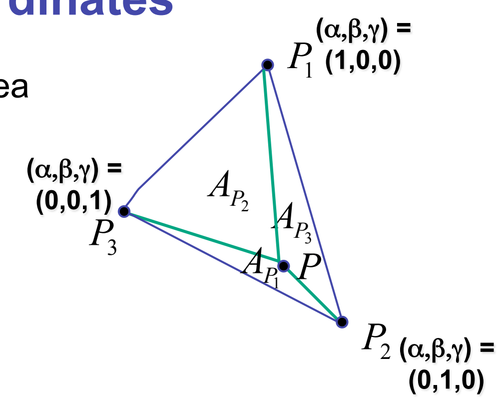
- 2D triangle area
  - half of parallelogram area
    - from cross product

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

$\alpha = A_{P1} / A$

$\beta = A_{P2} / A$

$\gamma = A_{P3} / A$



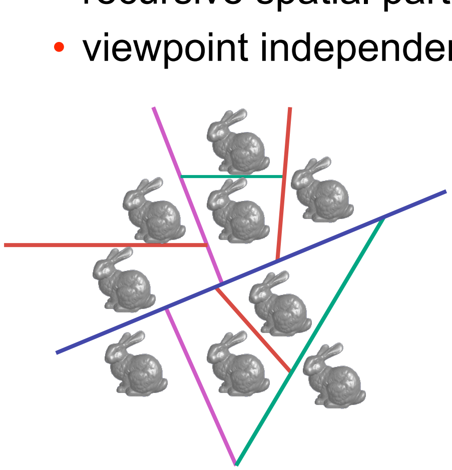
weighted combination of three points

Review: Painter's Algorithm

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

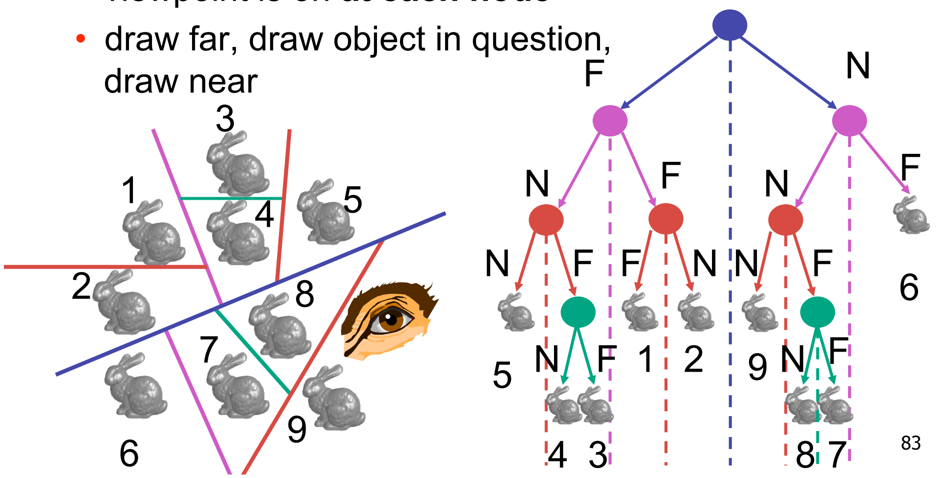
Review: BSP Trees

- 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: Depth Test Precision

- reminder: perspective transformation maps eye-space (view) *z* to NDC *z*

$$\begin{bmatrix} E & 0 & A & 0 \\ 0 & F & B & 0 \\ 0 & 0 & C & D \\ 0 & 0 & -1 & 0 \end{bmatrix}\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} Ex + Az \\ Fy + Bz \\ Cz + D \\ -z \end{bmatrix} = \begin{bmatrix} -\left(\frac{Ex}{z} + Az\right) \\ -\left(\frac{Fy}{z} + Bz\right) \\ -\left(C + \frac{D}{z}\right) \\ 1 \end{bmatrix}$$

- thus $z_{NDC} = -\left(C + \frac{D}{z_{eye}}\right)$
  - depth buffer essentially stores 1/z
    - high precision for near, low precision for distant

Review: Integer Depth Buffer

- reminder from picking: depth stored as integer
  - depth lies in the DCS z range [0,1]
  - format: multiply by 2^n -1 then round to nearest int
    - where n = number of bits in depth buffer
- 24 bit depth buffer = 2^24 = 16,777,216 possible values
  - small numbers near, large numbers far
- consider depth from VCS: (1<<N) * (a + b / z)
  - N = number of bits of Z precision
  - a = zFar / (zFar - zNear)
  - b = zFar * zNear / (zNear - zFar)
  - z = distance from the eye to the object

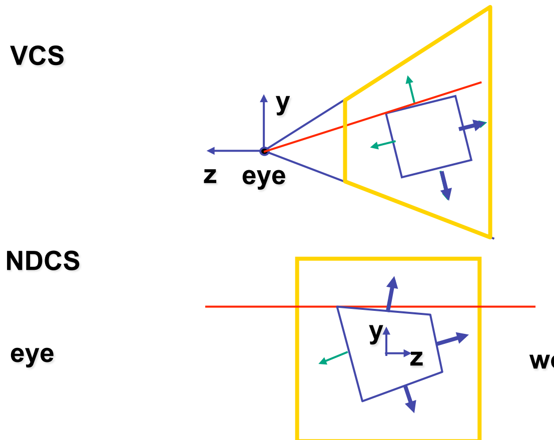
Review: Object Space Algorithms

- determine visibility on object or polygon level
  - using camera coordinates
- resolution independent
  - explicitly compute visible portions of polygons
- early in pipeline
  - after clipping
- requires depth-sorting
  - painter's algorithm
  - BSP trees

Review: Image Space Algorithms

- perform visibility test for in screen coordinates
  - limited to resolution of display
  - Z-buffer: check every pixel independently
- performed late in rendering pipeline

Review: Back-face Culling



works to cull if $N_Z > 0$

Review: Invisible Primitives

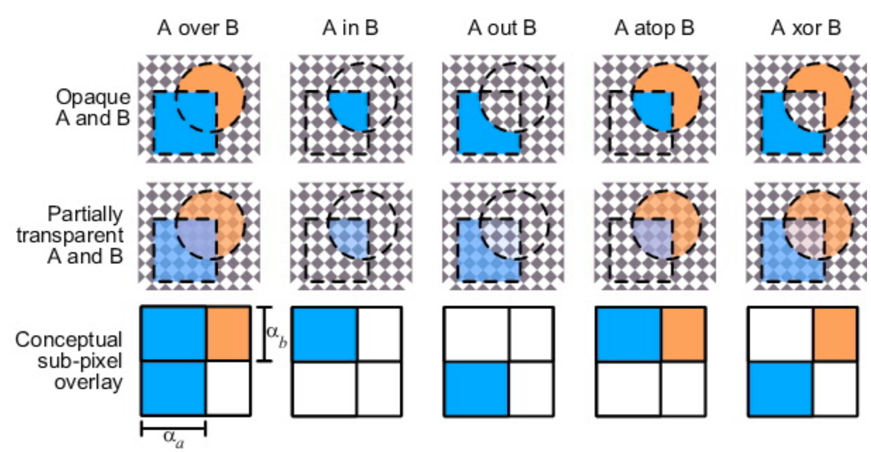
- *why might a polygon be invisible?*
  - polygon outside the *field of view / frustum*
    - solved by clipping
  - polygon is *backfacing*
    - solved by backface culling
  - polygon is *occluded* by object(s) nearer the viewpoint
    - solved by hidden surface removal

Review: Alpha and Premultiplication

- specify opacity with alpha channel α
  - α=1: opaque, α=.5: translucent, α=0: transparent
- how to express a pixel is half covered by a red object?
  - obvious way: store color independent from transparency (r,g,b,α)
    - intuition: alpha as transparent colored glass
      - 100% transparency can be represented with many different RGB values
    - pixel value is (1,0,0,.5)
    - upside: easy to change opacity of image, very intuitive
    - downside: compositing calculations are more difficult - not associative
  - elegant way: premultiply by α so store (αr, αg, αb,α)
    - intuition: alpha as screen/mesh
      - RGB specifies how much color object contributes to scene
      - alpha specifies how much object obscures whatever is behind it (coverage)
      - alpha of .5 means half the pixel is covered by the color, half completely transparent
      - only one 4-tuple represents 100% transparency: (0,0,0,0)
    - pixel value is (.5, 0, 0, .5)
    - upside: compositing calculations easy (& additive blending for glowing!)
    - downside: less intuitive

Review: Complex Compositing

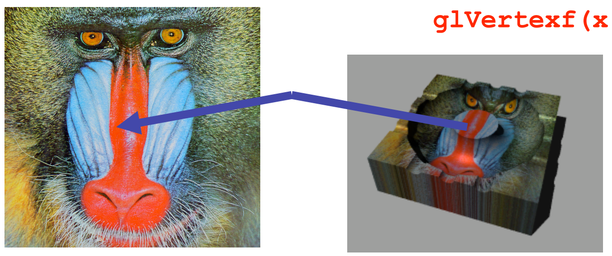
- foreground color **A**, background color **B**
- how might you combine multiple elements?
  - Compositing Digital Images, Porter and Duff, Siggraph '84
  - pre-multiplied alpha allows all cases to be handled simply



Review: Texture Coordinates

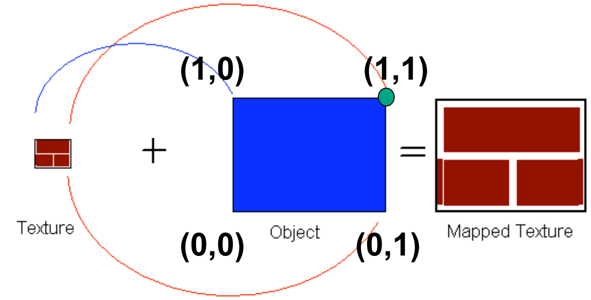
- texture image: 2D array of color values (texels)
- assigning texture coordinates (s,t) at vertex with object coordinates (x,y,z,w)
  - use interpolated (s,t) for texel lookup at each pixel
  - use value to modify a polygon's color
    - or other surface property
  - specified by programmer or artist

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

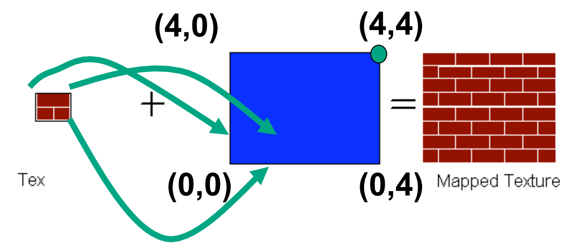


Review: Tiled Texture Map

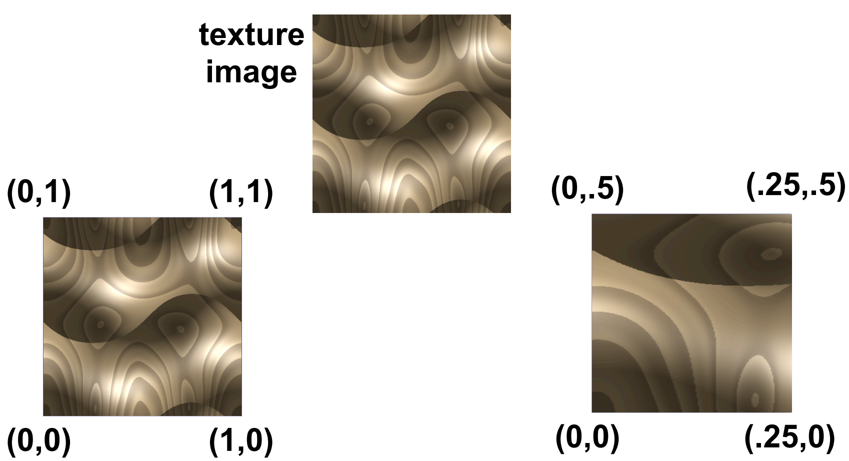
glTexCoord2d(1, 1);
glVertex3d (x, y, z);



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



Review: Fractional Texture Coordinates



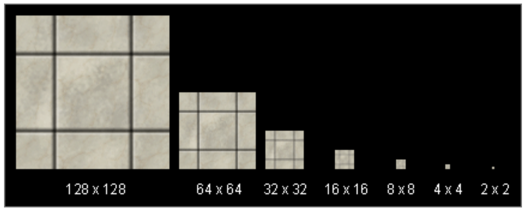
Review: Texture

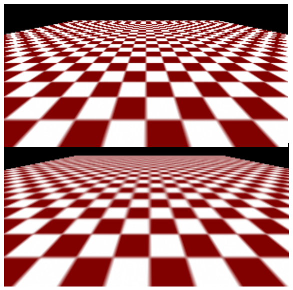
- action when s or t is outside [0…1] interval
  - tiling
  - clamping
- functions
  - replace/decal
  - modulate
  - blend
- texture matrix stack

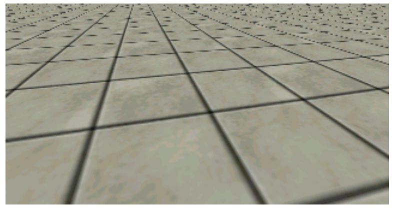
```
glMatrixMode( GL_TEXTURE );
```

Review: MIPmapping

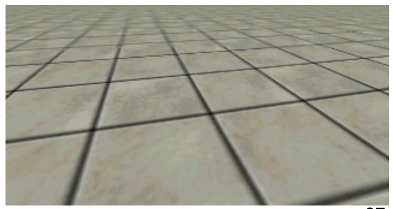
- image pyramid, precompute averaged versions







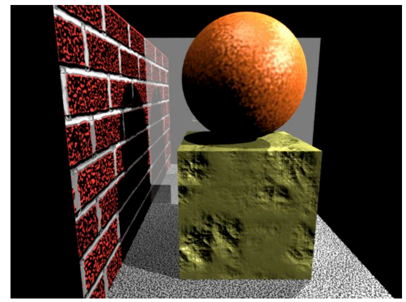
Without MIP-mapping

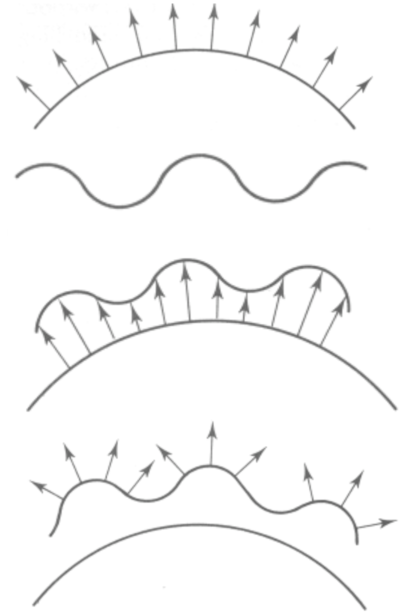


With MIP-mapping

Review: Bump Mapping: Normals As Texture

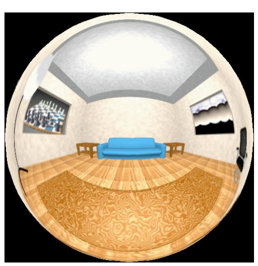
- create illusion of complex geometry model
- control shape effect by locally perturbing surface normal





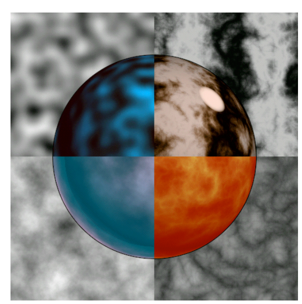
Review: Environment Mapping

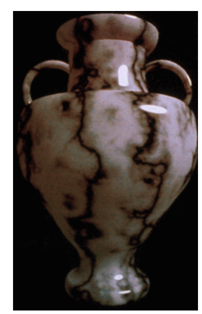
- cheap way to achieve reflective effect
  - generate image of surrounding
  - map to object as texture
- sphere mapping: texture is distorted fisheye view
  - point camera at mirrored sphere
  - use spherical texture coordinates



Review: Perlin Noise: Procedural Textures

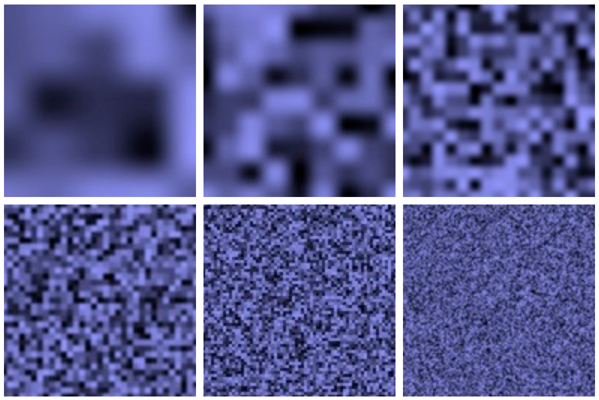
```
function marble(point)
 x = point.x + turbulence(point);
 return marble_color(sin(x))
```

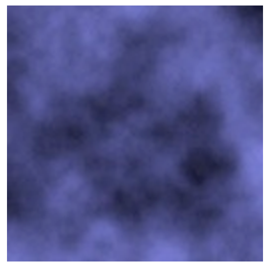




Review: Perlin Noise

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



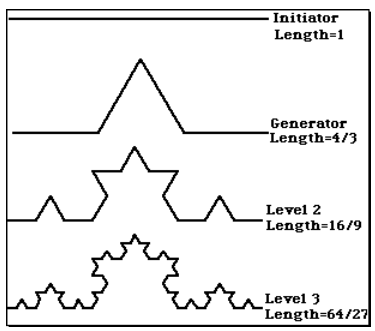


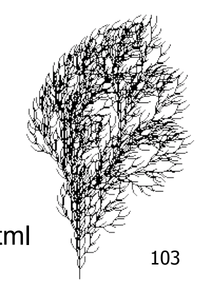
Review: Procedural Modeling

- 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

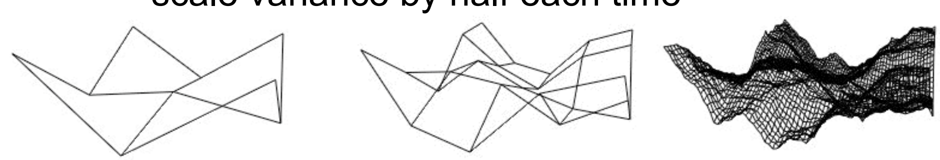




http://spanky.triumf.ca/www/fractint/lsys/plants.html

Review: Fractal Terrain

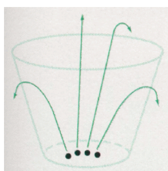
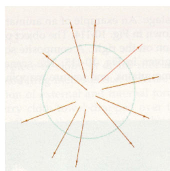
- 1D: midpoint displacement
  - divide in half, randomly displace
  - scale variance by half
- 2D: diamond-square
  - generate new value at midpoint
  - average corner values + random displacement
    - scale variance by half each time



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

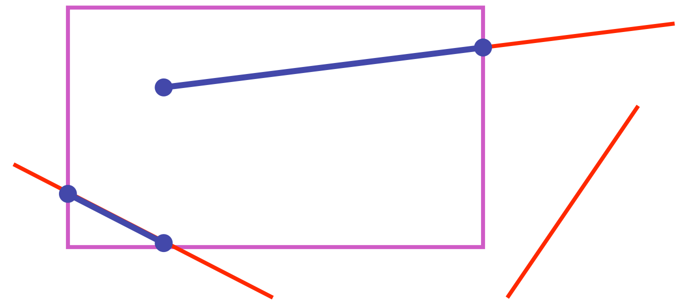
Review: Particle Systems

- 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

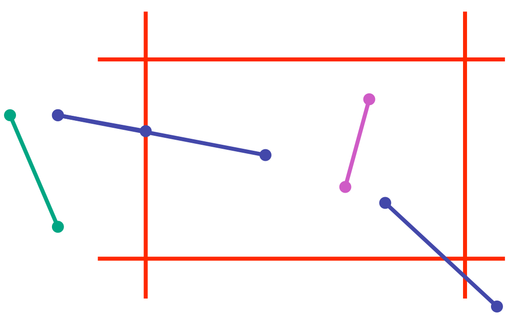
Review: Clipping

- analytically calculating the portions of primitives within the viewport



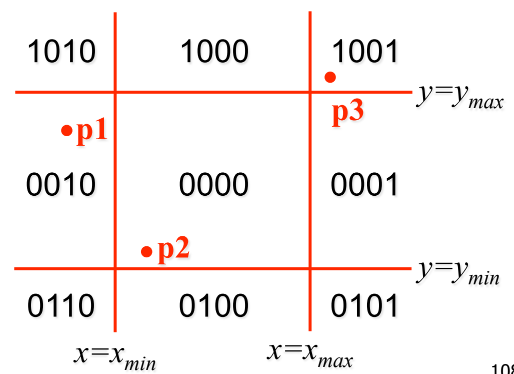
Review: Clipping Lines To Viewport

- 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



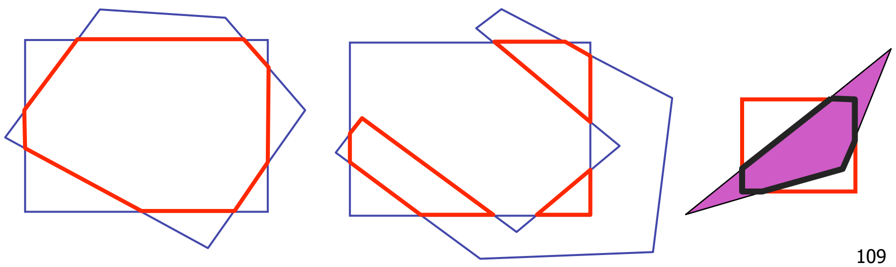
Review: Cohen-Sutherland Line Clipping

- 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



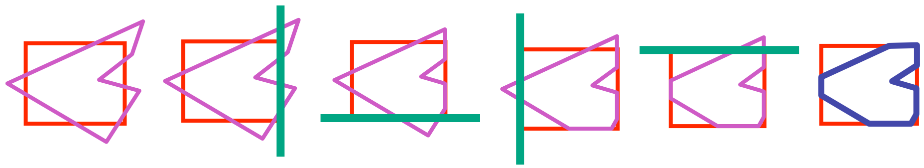
Review: Polygon Clipping

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



Review: Sutherland-Hodgeman Clipping

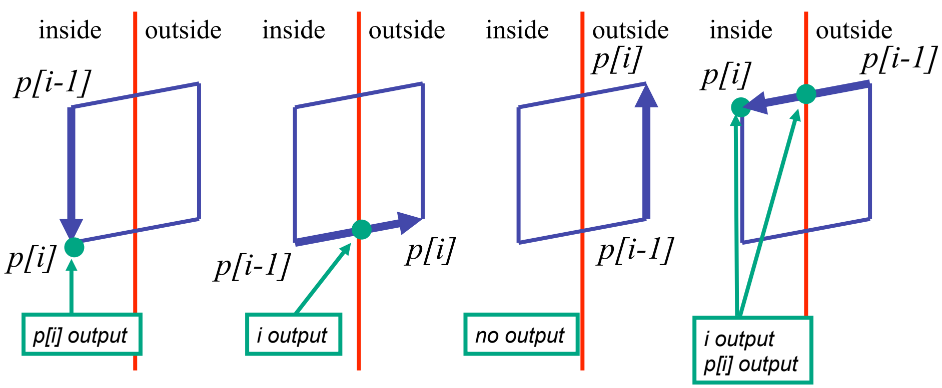
- for each viewport edge
  - clip the polygon against the edge equation for new vertex list
  - after doing all edges, the polygon is fully clipped



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

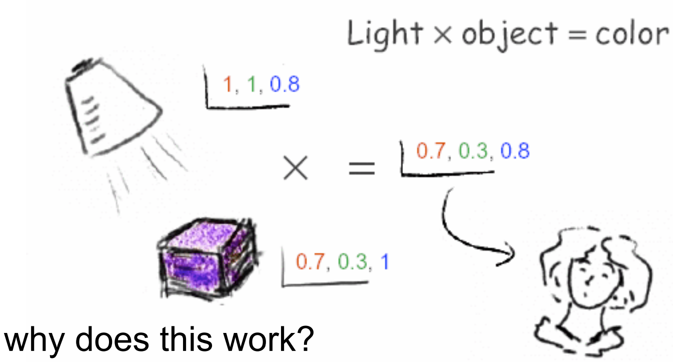
Review: Sutherland-Hodgeman Clipping

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



Review: RGB Component Color

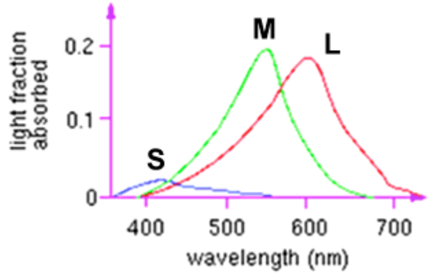
- simple model of color using RGB triples
- component-wise multiplication
  - (a0,a1,a2) * (b0,b1,b2) = (a0*b0, a1*b1, a2*b2)

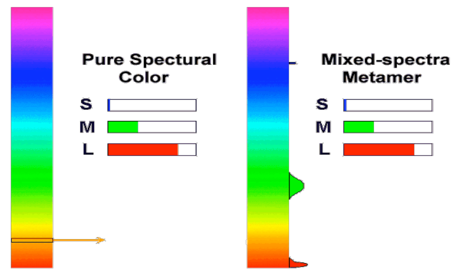


  - why does this work?
    - must dive into light, human vision, color spaces

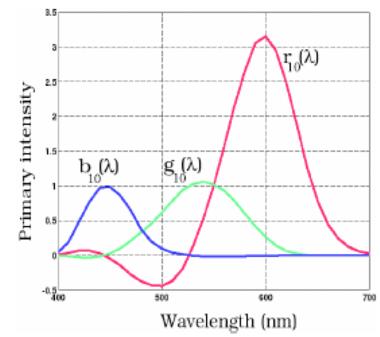
Review: Trichromacy and Metamers

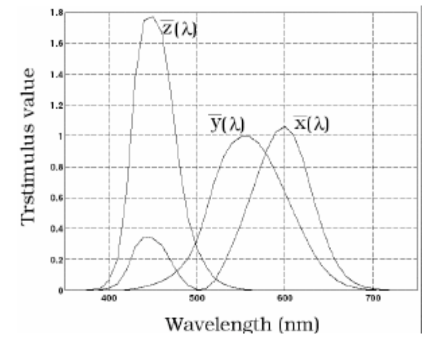
- three types of cones
- color is combination of cone stimuli
  - metamer: identically perceived color caused by very different spectra





Review: Measured vs. CIE Color Spaces

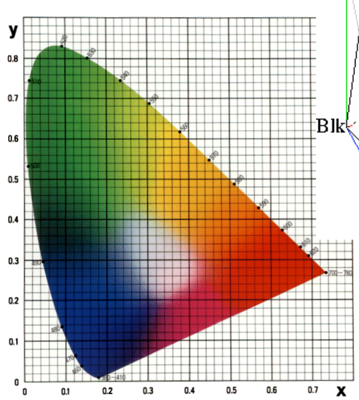


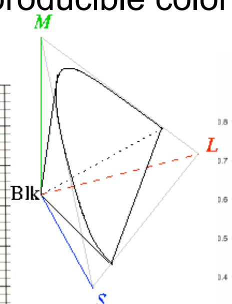


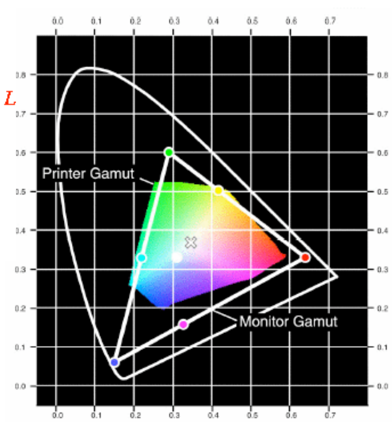
- measured basis
  - monochromatic lights
  - physical observations
  - negative lobes
- transformed basis
  - “imaginary” lights
  - all positive, unit area
  - Y is luminance

Review: Chromaticity Diagram and Gamuts

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

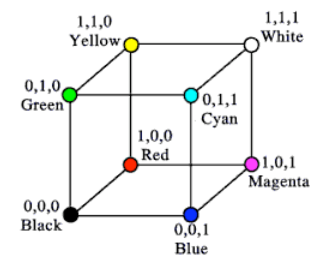




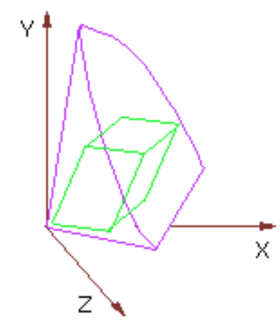


Review: RGB Color Space (Color Cube)

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

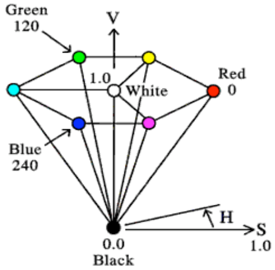


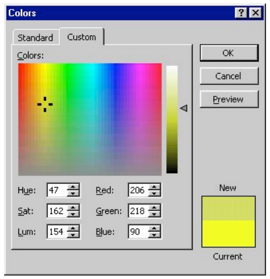
- RGB color cube sits within CIE color space
  - subset of perceivable colors
  - scale, rotate, shear cube

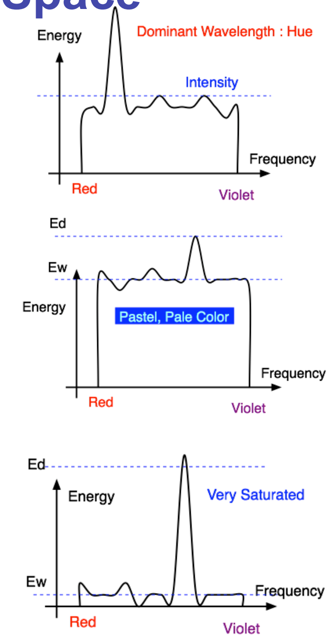


Review: HSV Color Space

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







Review: HSI/HSV and RGB

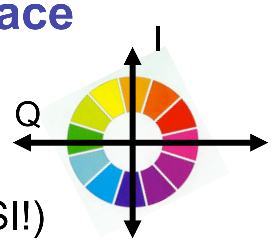
- HSV/HSI conversion from RGB
  - hue same in both
  - value is max, intensity is average

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

•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



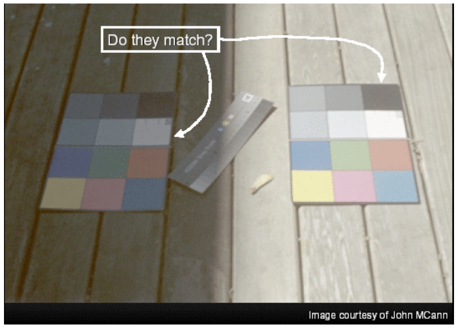
- color model used for color TV
  - Y is luminance (same as CIE)
  - I & Q are color (not same I as HSI!)
  - using Y backwards compatible for B/W TVs
  - conversion from RGB is linear

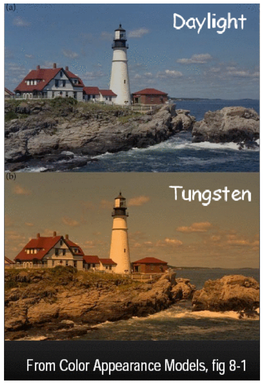
$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.30 & 0.59 & 0.11 \\ 0.60 & -0.28 & -0.32 \\ 0.21 & -0.52 & 0.31 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

  - 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



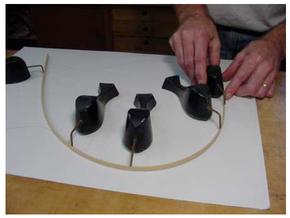


From Color Appearance Models, fig 8-1

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

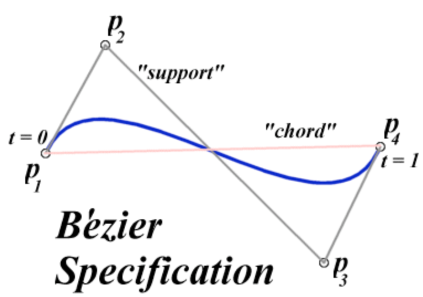
Review: Hermite Spline

- user provides
  - endpoints
  - derivatives at endpoints

Review: Bézier Curves

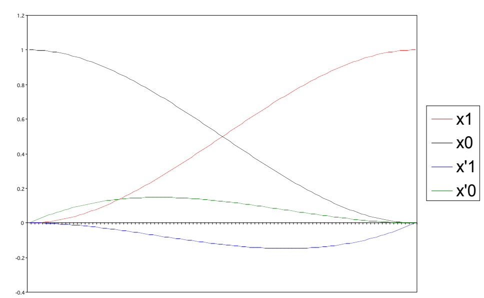
- four control points, two of which are knots
  - more intuitive definition than derivatives
- curve will always remain within convex hull (bounding region) defined by control points

Hermite Specification



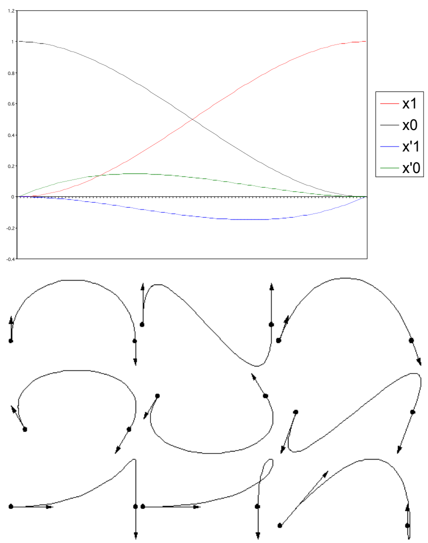
Review: Basis Functions

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

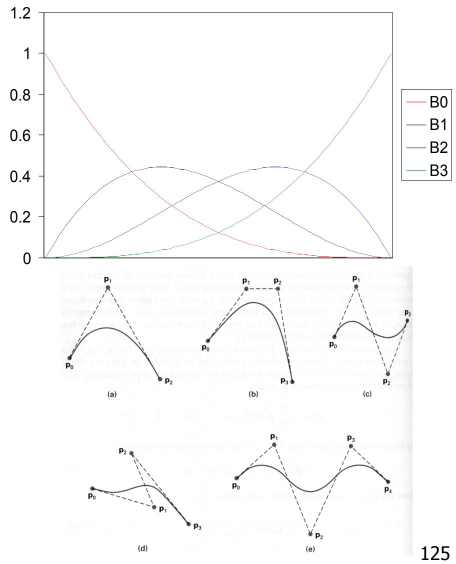


Review: Comparing Hermite and Bézier

Hermite

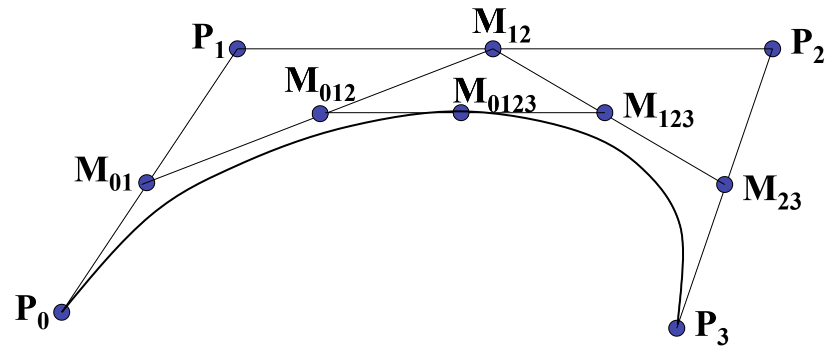


Bézier



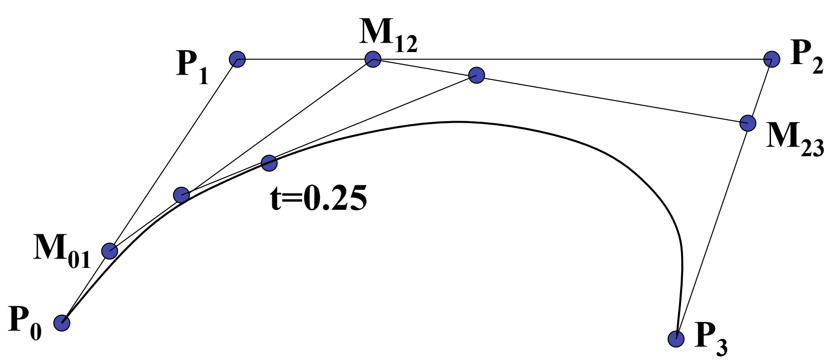
Review: Sub-Dividing Bézier Curves

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



Review: de Casteljau's Algorithm

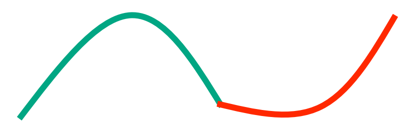
- can find the point on Bézier curve for any parameter value *t* with similar algorithm
  - for *t=0.25*, instead of taking midpoints take points 0.25 of the way



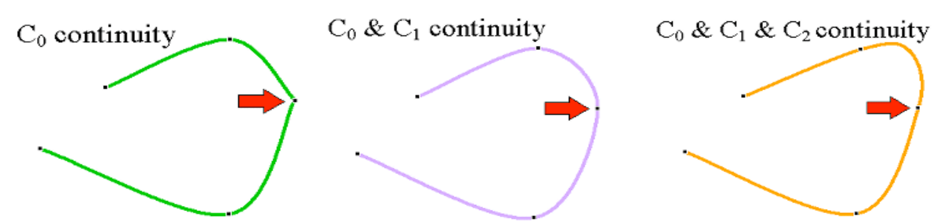
demo: www.saltire.com/applets/advanced_geometry/spline/spline.htm

Review: Continuity

- piecewise Bézier: no continuity guarantees

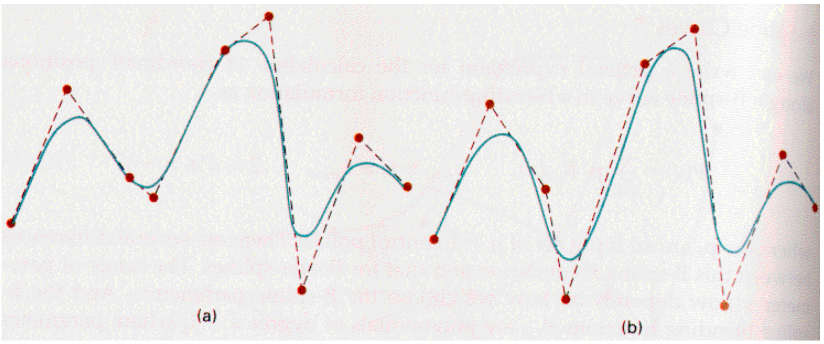


- continuity definitions
  - C^0: share join point
  - C^1: share continuous derivatives
  - C^2: share continuous second derivatives

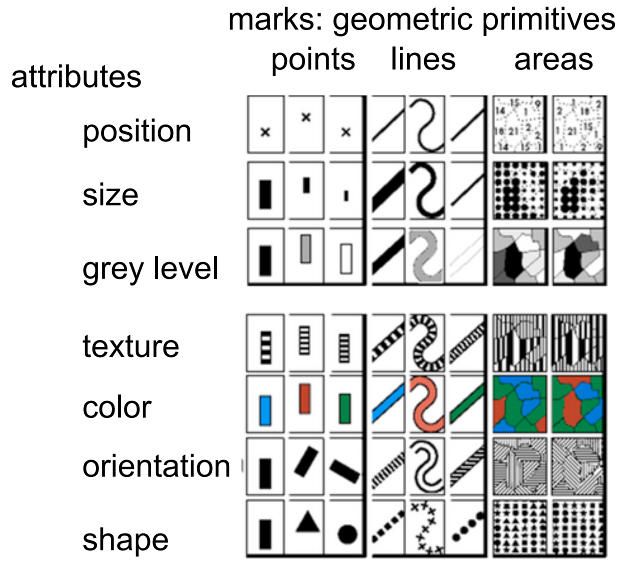


Review: B-Spline

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



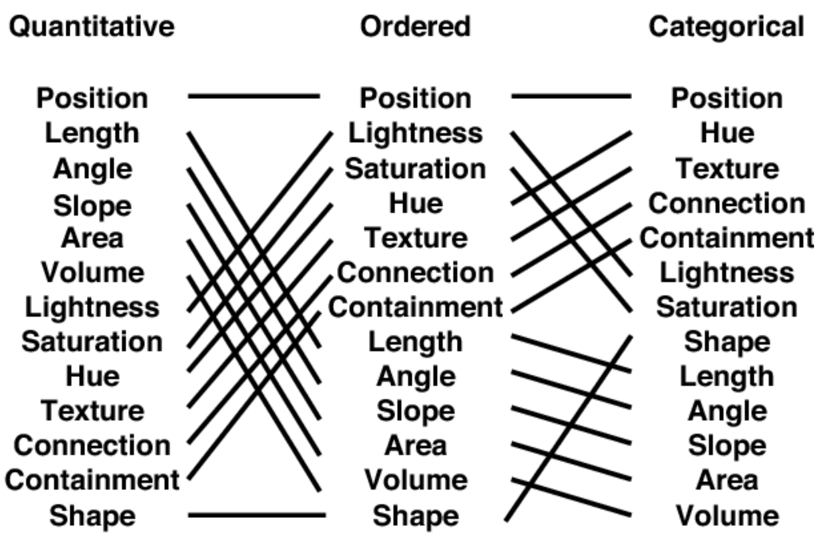
Review: Visual Encoding



- attributes
  - parameters control mark appearance
  - separable channels flowing from retina to brain

Semiology of Graphics. Jacques Bertin, Gauthier-Villars 1967, EHESS 1998

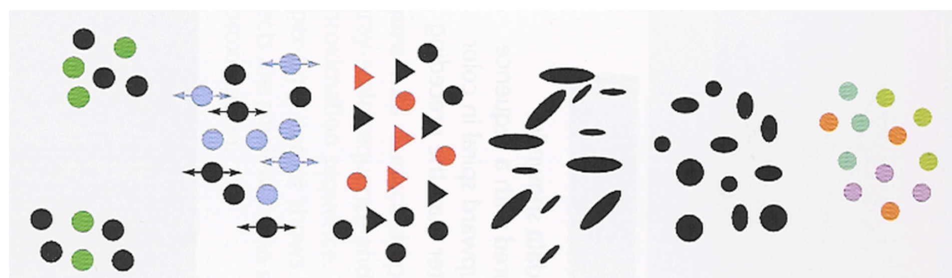
Review: Channel Ranking By Data Type



[Mackinlay, Automating the Design of Graphical

Review: Integral vs. Separable Channels

- not all channels separable

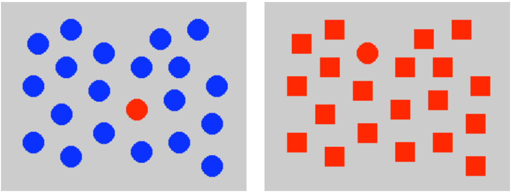


color location | color motion | color shape | size orientation | x-size y-size | red-green yellow-blue

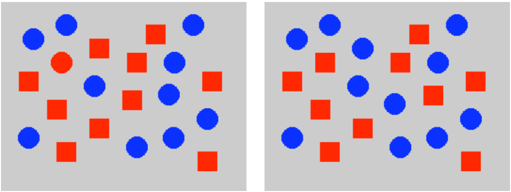
[Colin Ware, Information Visualization: Perception for Design. Morgan Kaufmann 1999.]

Review: Preattentive Visual Channels

- color alone, shape alone: preattentive



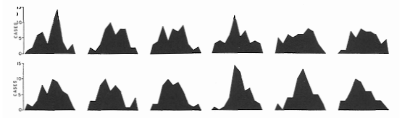
- combined color and shape: requires attention
  - search speed linear with distractor count



[Christopher Healey, [www.csc.ncsu.edu/faculty/healey/PP/PP.html]

Review: InfoVis Techniques

- 3D often worse then 2D for abstract data
  - perspective distortion, occlusion
  - transform, use linked views
- animation often worse than small multiples



- aggregation and filtering
  - focus+context
- dimensionality reduction
- parallel coordinates

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
- 530P: Sensorimotor Computation - Pai
- 533A: Digital Geometry – Sheffer
- 547: Information Visualization - Munzner