Final Review
University of British Columbia
CPSC 314 Computer Graphics
Jan-Apr 2013
Tamara Munzner

Review: 2D Rotation
θ
(x, y) (x′, y′)
\[ x' = x \cos(\theta) - y \sin(\theta) \]
\[ y' = x \sin(\theta) + y \cos(\theta) \]

counterclockwise, RHS

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

Review: Rendering Pipeline
• pipeline processing, set state as needed
void display()
{
    glClearColor(0.0, 0.0, 0.0, 0.0);
    glClear(GL_COLOR_BUFFER_BIT);
    glLoadIdentity();
    glBegin(GL_POLYGON);
    glVertex3f(0.25, 0.75, -0.5);
    glVertex3f(0.75, 0.75, -0.5);
    glVertex3f(0.75, -0.75, -0.5);
    glVertex3f(0.25, -0.75, -0.5);
    glEnd();
    glFlush();
}

Review from Open GL Red Book
• 1: Introduction to Open GL
• 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

Review from Shirley: Foundations of CG
• 1: Intro
• 2: Misc. Math *
• 3: Raster Algos *
• 4: Ray Tracing *
• 5: Linear Algebra *
• 6: Transformations *
• 7: Viewing *
• 8: Graphics Pipeline *
• 9: From 6.1.6 through 6.1.4, 6.2.3, 6.2.4, 6.2.5, 8.3.1, 8.4
• 10: Surface Shading *
• 11: Texture Mapping *
• 12: Data Structure *
• 15: Curves and Surfaces *
• 17: Computer Animation *
• only 17.6-17.7
• 21: Color *
• 22: Visual Perception *
• only 22.2 and 22.4
• 27: Visualization *

Review – Fast!!

Review: 2D Rotation: Another Derivation
\[ x' = x \cos(\theta) - y \sin(\theta) \]
\[ y' = x \sin(\theta) + y \cos(\theta) \]

Substitute…

Review: Rendering Capabilities

Review: 2D Rotation From Trig Identities
\[ x' = r \cos(\phi + \theta) \]
\[ y' = r \sin(\phi + \theta) \]

Trig Identity…
\[ x' = r \cos(\phi) \cos(\theta) - r \sin(\phi) \sin(\theta) \]
\[ y' = r \sin(\phi) \cos(\theta) + r \cos(\phi) \sin(\theta) \]

Shift/warp…
\[ x' = x \cos(\theta) - y \sin(\theta) \]
\[ y' = x \sin(\theta) + y \cos(\theta) \]

Review: 2D Rotation
θ
(x, y) (x′, y′)

counterclockwise, RHS

Review: 2D Rotation From Trig Identities

Review: Event-Driven Programming

Review: Surface Shading *
Review: 2D Transformations

- Linear transformations are combinations of:
  - Shear
  - Scale
  - Rotate
  - Reflect

- Properties of linear transformations:
  - Satisfies $T(sx + sy) = sT(x) + T(y)$
  - Origin maps to origin
  - Lines map to lines
  - Parallel lines remain parallel
  - Ratios are preserved
  - Closed under composition

Matrix multiplication:

```
\begin{pmatrix} a & b \\ c & d \end{pmatrix} \times \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} ax + by \\ cx + dy \end{pmatrix}
```

Review: 3D Homogeneous Coordinates

- Use 4x4 matrices for 3D transformations

```
\begin{pmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{pmatrix}
```

Review: 3D Shear

- General shear
  - "x-shear" usually means shear along x in direction of some other axis

```
\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & k & 0 & 1 \end{pmatrix}
```

Review: Affine Transformations

- Affine transforms are combinations of:
  - Linear transformations
  - Translation

```
\begin{pmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{pmatrix}
```

Review: Linear Transformations

- Scaling:

```
\begin{pmatrix} s & 0 \\ 0 & s \end{pmatrix}
```

- Shearing:

```
\begin{pmatrix} 1 & s \\ 0 & 1 \end{pmatrix} \quad \text{or} \quad \begin{pmatrix} 1 & 0 \\ s & 1 \end{pmatrix}
```

- Rotation:

```
\begin{pmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{pmatrix}
```

Review: Interpreting Transformations

- Composing transformations:

```
\begin{pmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \end{pmatrix} \times \begin{pmatrix} m & n & o & p \end{pmatrix} \times \begin{pmatrix} x \\ y \\ z \end{pmatrix}
```

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
- Modulate changes to pipeline state
- Avoids accumulation of numerical errors

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/openglidisplayslists

Review: Homogeneous Coordinates

- Homogeneous 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

```
\begin{pmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{pmatrix}
```

Review: Interpreting Transformations

- Right to left:
  - Moving object

```
\begin{pmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \end{pmatrix} \times \begin{pmatrix} x \\ y \\ z \end{pmatrix}
```

- Left to right:
  - Changing coordinate system

Review: Arithmetic 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 ABC

```
\begin{pmatrix} a_x & b_x & c_x \\ a_y & b_y & c_y \\ a_z & b_z & c_z \end{pmatrix}
```

Review: Translation Hierarchies

- Transforms apply to graph nodes beneath them
- Design structure so that object doesn't fall apart
- Instancing

Review: Affine Transformations

- Affine transforms are combinations of:
  - Linear transformations
  - Translation

```
\begin{pmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{pmatrix}
```

Review: Normals

- Polygon:

```
N = \frac{(P_1 - P_2) \times (P_2 - P_1)}{|(P_1 - P_2) \times (P_2 - 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 = MP
N' = QN
```

Given M, what should Q be?
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
- \( M_{V2W} = TR \)
- we derived position of camera as object in world
  - invert for glLookAt: go from world to camera!
- \( M_{W2V} = (M_{V2W})^{-1} R^T T^{-1} \)

Review: Graphics Cameras
- real pinhole camera: image inverted
- computer graphics camera: convenient equivalent

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

Review: Graphics Cameras
- real pinhole camera: image inverted
- computer graphics camera: convenient equivalent

Review: Basic Perspective Projection
- similar triangles
- \( P(x', y', z') = \frac{x'}{d} = \frac{y'}{d} = \frac{z'}{d} \)

Review: From VCS to NDCS
- perspective 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

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

Review: N2D Transformation
- \( \begin{bmatrix} x 2 & 0 & z 1 & 0 \\ 0 & \frac{x 3}{z 3} & 0 & 0 \\ \frac{x 4}{z 4} & 0 & 1 & 0 \\ \end{bmatrix} \)
- 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

Review: OpenGL Example
- go back from end of pipeline to beginning: coord frame POV!
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: Texture Coordinates
- texture image: 2D array of color values (texels)
- assigning texture coordinates (s,t) at vertex with object coordinates (x,y,z)
- use interpolated (s,t) for texture lookups at each pixel
- use value to modify a polygon’s color
- or other surface property
- specified by programmer or artist

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
- BSP trees

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: Depth Test Precision
- reminder: perspective transformation maps eye-space (view) z to NDC z
- \( \begin{align*}
    E' &= A E + C \\
    F' &= A F + C \\
    G' &= A G + C \\
    H' &= A H + C \\
    \end{align*} \)
- thus
- depth buffer essentially stores \( 1/z \)
- high precision for near, low precision for distant

Review: Back-face Culling
- why might a polygon be invisible?
- polygon outside the field of view / frustum
- solved by clipping
- polygon is back-facing
- solved by backface culling
- polygon is occluded by object(s) nearer the viewpoint
- solved by hidden surface removal

Review: Invisible Primitives
- why might a polygon be invisible?
- polygon outside the field of view / frustum
- solved by clipping
- polygon is back-facing
- solved by backface culling
- polygon is occluded by object(s) nearer the viewpoint
- solved by hidden surface removal

Review: Integer Depth Buffer
- reminder from picking: depth stored as integer
- depth lies in the DCS z range \([0,1] \)
- where \( n \) is 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: \( (1<<n) \cdot \left( a + b / z \right) \)
- \( a = z_{Near} \cdot b = z_{Far} - z_{Near} \)
- \( z_{Far} \)
- \( z_{Near} \)
- \( z = distance \ from \ the \ eye \ to \ the \ object \)

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

Review: Tiled Texture Map
- \( \text{gTexCoord2f}(x, y) \)
- \( \text{gVertex3f}(x, y, z) \)
- \( \text{glTexCoord2f}(x, y) \)
- \( \text{glVertex3f}(x, y, z) \)
- texture coordinates
- \( \text{texture image} \)

Review: Fractional Texture Coordinates
- action when s or t is outside \([0...1]\) interval
- tiling
- clamping
- functions
- replace/decal
- modulate
- blend
- texture matrix stack
- \( \text{glMatrixMode( GL_TEXTURE );} \)
**Review: MIPmapping**
- image pyramid, precompute averaged versions

**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: Polygon Clipping**
- not just clipping all boundary lines
- may have to introduce new line segments

**Review: Clipping**
- analytically calculating the portions of primitives within the viewport

**Review: Cohen-Sutherland Line Clipping**
- outcodes
  - 4 flags encoding position of a point relative to top, bottom, left, and right boundary

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

**Review: RGB Component Color**
- simple model of color using RGB triples
  - component-wise multiplication
    - \[(a_0, a_1, a_2) \cdot (b_0, b_1, b_2) = (a_0 \cdot b_0, a_1 \cdot b_1, a_2 \cdot b_2)\]

**Review: Bump Mapping: Normals As Texture**
- create illusion of complex geometry model
- control shape effect by locally perturbing surface normal

**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

**Review: Sutherland-Hodgeman Clipping**
- edge from \(p_{i-1}\) to \(p_i\) has four cases
  - decide what to add to output vertex list

**Review: Polygon Clipping**
- coherency: smooth not abrupt changes
- turbulence: multiple feature sizes

**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: 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

**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 = FLFRFFL
  - Mariano’s Bush:
    - F = FF-[F+F+F]+[+F-F-F]
  - angle 16

**Review: RGB Component Color**
- simple model of color using RGB triples
  - component-wise multiplication
    - \[(a_0, a_1, a_2) \cdot (b_0, b_1, b_2) = (a_0 \cdot b_0, a_1 \cdot b_1, a_2 \cdot b_2)\]

**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

**Review: Procedural Modeling**
- textures, geometry
  - nonprocedural: explicitly stored in memory
  - procedural...

**Review: Language-Based Generation**
- L-Systems
  - F: forward, R: right, L: left
  - Koch snowflake:
    - F = FLFRFFL
  - Mariano’s Bush:
    - F = FF-[F+F+F]+[+F-F-F]
  - angle 16

**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

**Review: RGB Component Color**
- simple model of color using RGB triples
  - component-wise multiplication
    - \[(a_0, a_1, a_2) \cdot (b_0, b_1, b_2) = (a_0 \cdot b_0, a_1 \cdot b_1, a_2 \cdot b_2)\]

**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: Sutherland-Hodgeman Clipping**
- for each polygon vertex
  - decide what to do based on 4 possibilities
    - is vertex inside or outside?
    - is previous vertex inside or outside?
**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**
- plane of equal brightness showing chromaticity
- gamut is polygon, device primaries at corners
- defines reproducible color range

**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: 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: 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

**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: HSI/HSV and RGB**
- $H = \cos^{-1}\left(\frac{1}{2}\left((R-G)+(R-B)\right)\right)$ if $(B>G)$,
  \[ H = 360 - H \]
- $S = 1 - \min(R,G,B)$
- $I = \frac{R + G + B}{3}$
- $V = \max(R,G,B)$
- green is much lighter than red, and red lighter than blue

**Review: Chromaticity Diagram and Gamuts**
- RGB color cube sits within CIE color space
- subset of perceivable colors

**Review: Color Constancy**
- automatic "white balance" from change in illumination
- vast amount of processing behind the scenes!
- colorimetry vs. perception

**Review: YIQ Color Space**
- color model used for color TV
- Y is luminance (same as CIE)
- I & Q are color channels: I = monochromatic lights, Q = 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: 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: 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

**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: B-Spline

- C₀, C₁, and C₂ continuous
- piecewise: locality of control point influence

Review: Visual Encoding

- marks: geometric primitives
  - position
  - size
  - grey level
  - texture
  - color
  - orientation
  - shape

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

Review: Channel Ranking By Data Type

- Quantitative
  - Position
  - Length
  - Angle
  - Slope
  - Area
  - Volume
  - Lightness
  - Saturation
  - Hue
  - Texture
  - Connection
  - Containment
  - Angles
  - Slopes
  - Areas
  - Shapes

- Ordered
  - Position
  - Hue
  - Texture
  - Connection
  - Containment
  - Lightness
  - Saturation
  - Shape
  - Length
  - Angle
  - Slope
  - Area
  - Volume

Review: Integral vs. Separable Channels

- not all channels separable


position
size
grey level
texture
color
orientation
shape
marks: geometric primitives
attributes
parameters
control mark
appearance
separable
channels
flowing from retina to brain

Review: Visual Encoding

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

Mackinlay, Automating the Design of Graphical Presentations of Relational Information, ACM TOG 5:2, 1986

Review: Integral vs. Separable Channels

- not all channels separable

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

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

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