Viewing

Coordinate Systems
- result of a transformation
- names
  - convenience
  - animal: leg, head, tail
- standard conventions in graphics pipeline
  - object/modelling
  - world
  - camera/viewing/eye
  - screen/window
  - raster/device

Using Transformations
- three ways
  - modelling transforms
    - place objects within scene (shared world)
  - affine transformations
  - viewing transforms
    - place camera
    - rigid body transformations: rotate, translate
    - projection transforms
    - change type of camera
    - projective transformation

Computer Graphics
Reading for This Module
- FCG Chapter 7 Viewing
- FCG Section 6.3.1 Windowing Transforms
- RB rest of Chap Viewing
- RB rest of App Homogeneous Coords
- RB Chap Selection and Feedback
- RB Sec Object Selection Using the Back Buffer
  - (in Chap Now That You Now )

Rendering Pipeline
- Scene graph
- Object geometry
- Modelling
- Transforms
- Viewing
- Transform
- Projection
- Transform

OpenGL Transformation Storage
- modeling and viewing stored together
- possible because no intervening operations
- perspective stored in separate matrix
- specify which matrix is target of operations
- common practice: return to default modelview mode after doing projection operations
  - glmatrixMode(GL_MODELVIEW);
  - glmatrixMode(GL_PROJECTION);

Coordinate Systems
- object
- world
  - OCS - object/model coordinate system
  - WCS - world coordinate system
- viewing
  - VCS - viewing/camera/eye coordinate system
- clipping
  - CCS - clipping coordinate system
  - NDCS - normalized device coordinate system
  - DCS - device/display/screen coordinate system

Projective Rendering Pipeline
- result
- all vertices of scene in shared 3D world coordinate system
- three ways
  - modelling transforms
  - place objects within scene (shared world)
  - affine transformations
  - viewing transforms
  - place camera
  - rigid body transformations: rotate, translate
  - projection transforms
  - change type of camera
  - projective transformation

Basic Viewing
- starting spot - OpenGL
  - camera at world origin
  - probably inside an object
- y axis is up
- looking down negative z axis
  - why? RHS with x horizontal, y vertical, z out of screen
- translate backward so scene is visible
  - move distance d = focal length
- where is camera in P1 template code?
  - 5 units back, looking down -z axis
Convenient Camera Motion
• rotate/translate/scale versus
eye point, gaze/lookat direction, up vector
demo: Robins transformation, projection

OpenGL Viewing Transformation
• rotate/translate/scale versus
eye point, gaze/lookat direction, up vector
• postmultiplies current matrix, so to be safe:
gluLookAt(ex,ey,ez,lx,ly,lz,ux,uy,uz)
• demo: Nate Robins tutorial projection

Deriving V2W Transformation
• translate origin to eye
• rotate view vector (lookat – eye) to w axis
• w: normalized opposite of view/gaze vector g
• w = -g = -\frac{g}{\|g\|}
• u should be perpendicular to vw-plane, thus perpendicular to w and up vector t
• v should be perpendicular to u and w
• u,v,w to bring up into vw-plane

Moving the Camera or the 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
      • first OpenGL camera, models viewing in camera coordinates
    • camera stays put, but world moves to -7

World vs. Camera Coordinates Example
\[\begin{bmatrix}
0 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
1 & 0 & 0 & 0
\end{bmatrix} \times \begin{bmatrix}
0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix} = \begin{bmatrix}
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0
\end{bmatrix}
\]
• possible difference: are lights specified in world or view coordinates?

Deriving V2W Transformation
• rotate from WCS xyz into uvw coordinate system with matrix that has columns u, v, w
\[ M_{V2W} = TR \]
• reminder: rotate from uvw to xyz coord sys with matrix M that has columns u,v,w

Convenient Camera Motion
• rotate/translate/scale versus
treat camera as if it’s just an object
• translate from origin to eye
• rotate view vector (lookat – eye) to w axis
• rotate around w to bring up into vw-plane

Placing Camera in World Coords: V2W
• non-zero sized hole
• blur: rays hit multiple points on film plane

Deriving V2W Transformation
• rotate from WCS xyz into uvw coordinate system with matrix that has columns u, v, w
\[ u = t \times w \]
\[ v = w \times u \]
\[ w = -\frac{g}{\|g\|} \]
• \[ TR \]

Convenient Camera Motion
• rotate/translate/scale versus
treat camera as if it’s just an object
• translate from origin to eye
• rotate view vector (lookat – eye) to w axis
• rotate around w to bring up into vw-plane

Placing Camera in World Coords: V2W
• non-zero sized hole
• blur: rays hit multiple points on film plane

Deriving V2W Transformation
• rotate from WCS xyz into uvw coordinate system with matrix that has columns u, v, w
\[ u = t \times w \]
\[ v = w \times u \]
\[ w = -\frac{g}{\|g\|} \]
• \[ TR \]

Convenient Camera Motion
• rotate/translate/scale versus
treat camera as if it’s just an object
• translate from origin to eye
• rotate view vector (lookat – eye) to w axis
• rotate around w to bring up into vw-plane

Placing Camera in World Coords: V2W
• non-zero sized hole
• blur: rays hit multiple points on film plane

Deriving V2W Transformation
• rotate from WCS xyz into uvw coordinate system with matrix that has columns u, v, w
\[ u = t \times w \]
\[ v = w \times u \]
\[ w = -\frac{g}{\|g\|} \]
• \[ TR \]

Convenient Camera Motion
• rotate/translate/scale versus
treat camera as if it’s just an object
• translate from origin to eye
• rotate view vector (lookat – eye) to w axis
• rotate around w to bring up into vw-plane

Placing Camera in World Coords: V2W
• non-zero sized hole
• blur: rays hit multiple points on film plane

Deriving V2W Transformation
• rotate from WCS xyz into uvw coordinate system with matrix that has columns u, v, w
\[ u = t \times w \]
\[ v = w \times u \]
\[ w = -\frac{g}{\|g\|} \]
• \[ TR \]

Convenient Camera Motion
• rotate/translate/scale versus
treat camera as if it’s just an object
• translate from origin to eye
• rotate view vector (lookat – eye) to w axis
• rotate around w to bring up into vw-plane

Placing Camera in World Coords: V2W
• non-zero sized hole
• blur: rays hit multiple points on film plane

Deriving V2W Transformation
• rotate from WCS xyz into uvw coordinate system with matrix that has columns u, v, w
\[ u = t \times w \]
\[ v = w \times u \]
\[ w = -\frac{g}{\|g\|} \]
• \[ TR \]

Convenient Camera Motion
• rotate/translate/scale versus
treat camera as if it’s just an object
• translate from origin to eye
• rotate view vector (lookat – eye) to w axis
• rotate around w to bring up into vw-plane

Placing Camera in World Coords: V2W
• non-zero sized hole
• blur: rays hit multiple points on film plane

Deriving V2W Transformation
• rotate from WCS xyz into uvw coordinate system with matrix that has columns u, v, w
\[ u = t \times w \]
\[ v = w \times u \]
\[ w = -\frac{g}{\|g\|} \]
• \[ TR \]
Real Cameras
- pinhole camera has small aperture (lens opening)
- minimize blur
- problem: hard to get enough light to expose the film
  - solution: lens
  - permits larger apertures
  - permits changing distance to film plane without actually moving it
    - cost: limited depth of field where image is in focus


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

Simple Perspective Projection Matrix

Projective Transformations
- planar geometric projections
  - planar: onto a plane
  - geometric: using straight lines
    - projections: 3D -> 2D
  - aka projective mappings
  - counterexamples?

Perspective Projection
- our camera must model perspective
  - expressible with 4x4 homogeneous matrix
  - use previously untouched bottom row

Basic Perspective Projection

similar triangles

\[
\begin{align*}
P'(x', y', z') &= P(x, y, z) \\
\frac{y'}{d} &= \frac{y}{z} \\
x' &= x - \frac{d}{z} \cdot x \\
y' &= y - \frac{d}{z} \cdot y \\
z' &= z - \frac{d}{z} \cdot z
\end{align*}
\]

- nonuniform foreshortening
- not affine

Perspective Projection
- desired result for a point \([x, y, z, 1]^T\) projected onto the view plane:

\[
\begin{align*}
x' &= \frac{x}{z/d} \\
y' &= \frac{y}{z/d} \\
z' &= \frac{z}{d}
\end{align*}
\]

- what could a matrix look like to do this?

Simple Perspective Projection Matrix

Moving COP to Infinity
- as COP moves away, lines approach parallel
- when COP at infinity, orthographic view
Orthographic Camera Projection
- camera’s back plane parallel to lens
- infinite focal length
- no perspective convergence
- just throw away z values

Perspective to Orthographic
- transformation of space
- center of projection moves to infinity
- view volume transformed
- from frustum (truncated pyramid) to parallelepiped (box)

View Volumes
- specifies field-of-view, used for clipping
- restricts domain of z stored for visibility test

Canonical View Volumes
- standardized viewing volume representation
  - perspective
  - orthographic orthogonal parallel

Why Canonical View Volumes?
- permits standardization
- clipping
  - easier to determine if an arbitrary point is enclosed in volume
  - with canonical view volume vs. clipping to six arbitrary planes
- rendering
  - projection and rasterization algorithms can be reused

Orthographic Derivation
- scale, translate, reflect for new coord sys
- center of projection moves to infinity
- view volume transformed
- from frustum (truncated pyramid) to parallelepiped (box)

Understanding Z
- near, far always positive in OpenGL calls
  - glOrtho(left,right,bottom,top,near,far);
- convention
  - viewing frustum mapped to specific parallelepiped
  - Normalized Device Coordinates (NDC)
  - same as clipping coords
  - only objects inside the parallelepiped get rendered
  - which parallelepiped?
  - depends on rendering system

Normalized Device Coordinates
- convention
  - viewing frustum mapped to specific parallelepiped
  - Normalized Device Coordinates (NDC)
  - same as clipping coords
  - only objects inside the parallelepiped get rendered
  - which parallelepiped?
  - depends on rendering system

Understanding Z
- why near and far plane?
- near plane:
  - avoid singularity (division by zero, or very small numbers)
- far plane:
  - store depth in fixed-point representation (integer), thus have to have fixed range of values (0 ... 1)
  - avoid/reduce numerical precision artifacts for distant objects

Orthographic Derivation
- scale, translate, reflect for new coord sys
- center of projection moves to infinity
- view volume transformed
- from frustum (truncated pyramid) to parallelepiped (box)

Normalized Device Coordinates
- convention
  - viewing frustum mapped to specific parallelepiped
  - Normalized Device Coordinates (NDC)
  - same as clipping coords
  - only objects inside the parallelepiped get rendered
  - which parallelepiped?
  - depends on rendering system

Understanding Z
- z axis flip changes coord system handedness
- RHS before projection (eye/view coords)
- LHS after projection (clip, norm device coords)

Orthographic Derivation
- scale, translate, reflect for new coord sys
- center of projection moves to infinity
- view volume transformed
- from frustum (truncated pyramid) to parallelepiped (box)

Understanding Z
- why near and far plane?
- near plane:
  - avoid singularity (division by zero, or very small numbers)
- far plane:
  - store depth in fixed-point representation (integer), thus have to have fixed range of values (0 ... 1)
  - avoid/reduce numerical precision artifacts for distant objects

Orthographic Derivation
- scale, translate, reflect for new coord sys
- center of projection moves to infinity
- view volume transformed
- from frustum (truncated pyramid) to parallelepiped (box)

Understanding Z
- why near and far plane?
- near plane:
  - avoid singularity (division by zero, or very small numbers)
- far plane:
  - store depth in fixed-point representation (integer), thus have to have fixed range of values (0 ... 1)
  - avoid/reduce numerical precision artifacts for distant objects

Orthographic Derivation
- scale, translate, reflect for new coord sys
- center of projection moves to infinity
- view volume transformed
- from frustum (truncated pyramid) to parallelepiped (box)
Orthographic Derivation
- Scale, translate, reflect for new coord sys

Orthographic OpenGL
- glMatrixMode(GL_PROJECTION);
- glLoadIdentity();
- glOrtho(left, right, bot, top, near, far);

Asymmetric Frusta
- Our formulation allows asymmetry
- Why bother?

Perspective OpenGL
- glMatrixMode(GL_PROJECTION);
- glLoadIdentity();
- glFrustum(left, right, bot, top, near, far);
  or
- glPerspective(fovy, aspect, near, far);

Demo: Frustum vs. FOV
- Nate Robins tutorial (take 2):

Field-of-View Formulation
- FOV in one direction + aspect ratio (w/h)
  - Determines FOV in other direction
  - Also set near, far (reasonably intuitive)

Perspective Warp
- Perspective viewing frustum transformed to cube
  - Orthographic rendering of cube produces same image as perspective rendering of original
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

Specific Perspective Derivation
• earlier: complete: shear, scale, projection-normalization

Perspective Divide Example
• specific example: assume image plane at z = -1
  • a point $[x,y,z,1]^T$ projects to $[-x/z,-y/z,-z,1]^T = [x,y,z,-z]^T$
  • after homogenizing, once again $w=1$

Origin Location
• yet more (possibly confusing) conventions
  • OpenGL origin: lower left
  • most window coordinates are flipped
  • when interpreting mouse position, have to flip your y coordinates

Perspective Derivation
• matrix formulation
  • warp and homogenization both preserve
    relative depth (z coordinate)

Projective Rendering Pipeline

Perspective To NDCS Derivation
• simple example earlier:
  • complete: shear, scale, projection-normalization

Perspective Derivation

Perspective Derivation
  • similarly for other 5 planes
  • 6 planes, 6 unknowns:

   \[
   \begin{bmatrix}
   0 & 0 & r + 1 & 0 \\
   r & 0 & r + 1 & 0 \\
   0 & r & r + 1 & 0 \\
   0 & 0 & r & r + 1 \\
   -2n & 2n & -2 & 0 \\
   0 & 0 & -f & -n \\
   a & 0 & 1 & 0
   \end{bmatrix}
   \]

NDC to Device Transformation
• map from NDC to pixel coordinates on display
  • NDC range is $x = -1...1, y = -1...1, z = 1...1$
  • typical display range: $x = 0...500, y = 0...300$
  • maximum is size of actual screen
  • $z$ range max and default is $(0, 1)$, use later for visibility

N2D Transformation
• general formulation
  • reflect in y for upper vs. lower left origin
  • scale by width/2, height/2, depth/2
  • for GL includes additional translation for pixel centers at $(.5, .5)$ instead of $(0,0)$
Device vs. Screen Coordinates

- viewport/window location wrt actual display not available within OpenGL
- rarely do this
- use relative information when handling mouse events, not absolute coordinates
- could get actual display width/height, window offsets from OS
- loose use of terms: device, display, window, screen...

Perspective Example

- view volume
- left = -1, right = 1
- bot = -1, top = 1
- near = 1, far = 4

OpenGL Example

- glMatrixMode( GL_PROJECTION );
- glLoadIdentity();
- gluPerspective( 45, 1.0, 0.1, 200.0 );

Coordinate Systems

- device
- NDCS - normalized device coordinate system
- DCS - device coordinate system
- WCS - world coordinate system
- VCS - viewing coordinate system
- CCS - clipping coordinate system
- OCS - object coordinate system

Coordinate Transformations

- modelview
- projection
- View volume

Viewing: More Camera Motion

- fly "through the lens": roll/pitch/yaw

Viewing: Virtual Trackball

- interface for spinning objects around
- drag mouse to control rotation of view volume
- orbit/spin metaphor
- vs. flying/driving
- rolling glass trackball
- center at screen origin, surrounds whole
- hemisphere "sticks up" in z, out of screen
- rotate ball = spin world

Caution: OpenGL Matrix Storage

- OpenGL internal matrix storage is columnwise, not rowwise
- opposite of standard C/C++/Java convention
- possibly confusing if you look at the matrix from glMatrixMode()!
Clarify: Trackball Rotation

• finding location on ball corresponding to click on image plane
  • ball radius \( r \) is 1

Trackball Computation

• user defines two points
  • place where first clicked \( p_1 = (x, y, z) \)
  • place where released \( p_2 = (a, b, c) \)
• create plane from vectors between points, origin
• axis of rotation is plane normal: cross product
  \[ (p_1 - o) \times (p_2 - o) = p_1 \times p_2 \] if origin \( = (0,0,0) \)
• amount of rotation depends on angle between lines
  \[ p_1 \cdot p_2 = \|p_1\| \|p_2\| \cos \theta \]
  \[ \|p_1 \times p_2\| = \|p_1\| \|p_2\| \sin \theta \]
• compute rotation matrix, use to rotate world

Viewport

• small rectangle around cursor
  • change coord sys so fills viewport
  • why rectangle instead of point?
    • people aren’t great at positioning mouse
    • Fitts’ Law: time to acquire a target is function of the distance to and size of the target
    • allow several pixels of slop

Interactive Object Selection

• move cursor over object, click
  • how to decide what is below?
  • inverse of rendering pipeline flow from pixel back up to object
  • ambiguity
    • many 3D world objects map to same 2D point
  • four common approaches
    • manual ray intersection
    • bounding extents
    • backbuffer color coding
    • selection region with hit list

Manual Ray Intersection

• do all computation at application level
  • map selection point to a ray
  • intersect ray with all objects in scene.
• advantages
  • no library dependence
• disadvantages
  • difficult to program
  • slow: work to do depends on total number and complexity of objects in scene

Bounding Extents

• keep track of axis-aligned bounding rectangles
• advantages
  • conceptually simple
  • easy to keep track of boxes in world space
• disadvantages
  • low precision
  • must keep track of object-rectangle relationship
  • extensions
    • do more sophisticated bound bookkeeping
    • first level: box check
    • second level: object check

Backbuffer Color Coding

• use backbuffer for picking
  • create image as computational entity
  • never displayed to user
  • redraw all objects in backbuffer
  • turn off shading calculations
  • set unique color for each pickable object
    • store in table
  • read back pixel at cursor location
  • check against table
• advantages
  • conceptually simple
  • variable precision
• disadvantages
  • introduce 2x redraw delay
  • backbuffer readback very slow

Backbuffer Example

```c
for (int j = 0; j < 2; j++) {
  for (int i = 0; i < 2; ++i) {
    switch (i*2+j) {
      case 0: glColor3ub(255,0,0);break;
      case 2: glColor3ub(0,0,255);break;
      case 3: glColor3ub(250,0,250);break;
      default: glColor3ub(255,255,255);break;
    }
    if (j == 1) {
      gCallList(snowman_display_list);
      gPopMatrix();
    }
    gPushMatrix();
    glColor3ub(0,0,0);
    gTranslatef(i*3.0,0,-j * 3.0);
  }
}
```

Select/Hit

• use small region around cursor for viewpoint
  • assign per-object integer keys (names)
  • redraw in special mode
  • store hit list of objects in region
  • examine hit list
  • OpenGL support

Name Stack

• again, "names" are just integers
  • glInitNames()
• flat list
  • glLoadName(name)
• or hierarchy supported by stack
  • glPushName(name), glPopName
  • can have multiple names per object

Red Book

• Selection and Feedback Chapter
  • all
• Now That You Know Chapter
  • only Object Selection Using the Back Buffer

GL_RENDER: normal color buffer

GL_SELECT: selection mode for picking

(GL_FEEDBACK: report objects drawn)
Hierarchical Names Example

```c
for(int i = 0; i < 2; i++) {
    glPushName(i);
    for(int j = 0; j < 2; j++) {
        glPushMatrix();
        ...
        glPopMatrix();
    }
    glPopName();
}
```

Hit List

- glSelectBuffer(buffersize, "buffer")
- on hit, copy entire contents of name stack to output buffer.

Integrated vs. Separate Pick Function

- integrate: use same function to draw and pick
  - simpler to code
  - name stack commands ignored in render mode
  - separate: customize functions for each
  - potentially more efficient
  - can avoid drawing unpickable objects

Select/Hit

- advantages
  - faster
  - OpenGL support means hardware acceleration
  - avoid shading overhead
  - flexible precision
  - size of region controllable
  - flexible architecture
  - custom code possible, e.g. guaranteed frame rate
- disadvantages
  - more complex

OpenGL Precision Picking Hints

- gluUnproject
- viewing transformation
- object coordinates to window coordinates

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
- multiplied by gluLookAt matrix

Hybrid Picking

- select/hit approach: fast, coarse
- object-level granularity
- manual ray intersection: slow, precise
- exact intersection point
- hybrid: both speed and precision
- use select/hit to find object
- then intersect ray with that object

Projective Rendering Pipeline

- gluLookAt(...)
- gluFrustum(x,y,a,b)
- gluTranslatef(x,y,z)
- gluRotatef(a,x,y,z)

OpenGL Example

- transformations that are applied to object
- first are specified
- go back from end of pipeline to beginning: coord frame POV!