News
- extra lab coverage with TAs
  - 12-2 Mondays, Wednesdays
  - for rest of term
  - just for answering questions, no presentations

Reading: Today
- FCG Chapter 6
- FCG Section 5.3.1
- RB rest of Chap Viewing
- RB rest of App Homogeneous Coords

Reading: Next Time
- FCG Section 2.11
- FCG Chap 3
  - except 3.8
- FCG Chap 17 Human Vision (pp 293-298)
- FCG Chap 18 Color pp 301-311
  - until Section 18.9 Tone Mapping

Textbook Errata
- list at http://www.cs.utah.edu/~shirley/fcg/errata
  - p 113
    - last matrix, last column denominators
      1. D-a -> A-a
      2. E-b -> B-b
      3. F-c -> C-c
  - p 120
    - "Sometimes we will want to take the inverse of P" should be "M_p" instead of "P"

Correction²: Vector-Vector Subtraction
- subtract: vector - vector = vector
  \[ \mathbf{u} - \mathbf{v} = \begin{bmatrix} u_1 - v_1 \\ u_2 - v_2 \\ u_3 - v_3 \end{bmatrix} \]

- (3,2) - (6,4) = (−3,−2)
- (2,5,1) - (3,1,−1) = (−1,4,2)

argument reversal
- \[ \mathbf{u} + \mathbf{v} = \mathbf{v} + \mathbf{u} \]
Review: 2D Rotation

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

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

- 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…

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

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

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

- reflect across x axis
  - mirror

Review: 2D Transformations

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

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

Review: Linear 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

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

- rotation matrix

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

- translation multiplication matrix

- vector addition

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

- matrix multiplication
**Review: Composing Transformations**

- order matters
  - 4x4 matrix multiplication not commutative!

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

---

**Review: Affine Transformations**

- affine transforms are combinations of
  - linear transformations
  - translations

- 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 Geometrically**

<table>
<thead>
<tr>
<th>homogeneous</th>
<th>cartesian</th>
</tr>
</thead>
<tbody>
<tr>
<td>((x, y, w))</td>
<td>((\frac{x}{w}, \frac{y}{w}))</td>
</tr>
</tbody>
</table>

- point in 2D cartesian + weight \(w\) = point \(P\) in 3D homogeneous coordinates
- multiples of \((x, y, w)\)
- all homogeneous points on 3D line \(L\) represent same 2D cartesian point
- homogenize to convert homog. 3D point to cartesian 2D point
- divide by \(w\) to get \((\frac{x}{w}, \frac{y}{w}, 1)\)
- \(w=0\) is direction; \((0,0,0)\) is undefined

---

**Review: 3D Homog Transformations**

- use 4x4 matrices for 3D transformations

**Example Transforms**

**Translate(a,b,c)**

\[
\begin{bmatrix}
    x' \\
    y' \\
    z' \\
    w'
\end{bmatrix} = \begin{bmatrix}
    1 & a & 0 & 0 \\
    b & 1 & 0 & 0 \\
    c & 0 & 1 & 0 \\
    0 & 0 & 0 & 1
\end{bmatrix} \begin{bmatrix}
    x \\
    y \\
    z \\
    1
\end{bmatrix}
\]

**Scale(a,b,c)**

\[
\begin{bmatrix}
    x' \\
    y' \\
    z' \\
    w'
\end{bmatrix} = \begin{bmatrix}
    a & 0 & 0 & 0 \\
    0 & b & 0 & 0 \\
    0 & 0 & c & 0 \\
    0 & 0 & 0 & 1
\end{bmatrix} \begin{bmatrix}
    x \\
    y \\
    z \\
    1
\end{bmatrix}
\]

**Rotate(x,\theta)**

\[
\begin{bmatrix}
    x' \\
    y' \\
    z' \\
    w'
\end{bmatrix} = \begin{bmatrix}
    \cos \theta & -\sin \theta & 0 & 0 \\
    \sin \theta & \cos \theta & 0 & 0 \\
    0 & 0 & 1 & 0 \\
    0 & 0 & 0 & 1
\end{bmatrix} \begin{bmatrix}
    x \\
    y \\
    z \\
    1
\end{bmatrix}
\]

---

**Review: Composing Transformations**

- 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

---

**Additional Notes**

- Review: 3D Homogeneous Transformations
- properties of affine transforms
  - origin does not necessarily map to origin
  - lines map to lines
  - parallel lines remain parallel
  - ratios are preserved
  - closed under composition

---

**Example Transformations**

- **Translate(a,b,c)**
- **Scale(a,b,c)**
- **Rotate(x,\theta)**
- **Order Matters**
  - \(Ta Tb = Tb Ta, \ but \ Ra Rb \neq Rb Ra\ and \ Ta Rb \neq Rb Ta\)

---

**Further Exploration**

- 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: Arbitrary Rotation

- problem:
  - given two orthonormal coordinate systems \(XYZ\) and \(UVW\)
  - find transformation from one to the other
- answer:
  - transformation matrix \(R\) whose columns are \(U, V, W\):
    \[
    R = \begin{bmatrix}
    u_x & v_x & w_x \\
    u_y & v_y & w_y \\
    u_z & v_z & w_z
    \end{bmatrix}
    \]

Review: Interpreting Transformations

\[
p' = TRp
\]

- right to left: moving object
- left to right: changing coordinate system

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

Review: Transforming Normals

- shear, nonuniform scale makes normal nonperpendicular
- need to use inverse transpose matrix instead

Review: Display Lists

- precompile/cache block of OpenGL code for reuse
  - efficiency
    - exact optimizations depend on driver
  - multiple instances of same object
  - static objects redrawn often
  - exploit hierarchical structure when possible
- set up list once with \(glNewList/glEndList\)
  - call multiple times
Using Transformations
- three ways
  - modelling transforms
    - place objects within scene (shared world)
  - viewing transforms
    - place camera
  - projection transforms
    - change type of camera

Viewing and Projection
- need to get from 3D world to 2D image
- projection: geometric abstraction
  - what eyes or cameras do
- two pieces
  - viewing transform:
    - where is the camera, what is it pointing at?
  - perspective transform: 3D to 2D
    - flatten to image

Rendering Pipeline

Rendering Pipeline
Rendering Pipeline

- result
  - all vertices of scene in shared 3D world coordinate system

Coordinate Systems

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

Projective Rendering Pipeline

- object world viewing
  - O2W WCS V2C
  - modeling transformation
  - viewing transformation
  - projection transformation
  - OCS - object/model coordinate system
  - WCS - world coordinate system
  - VCS - viewing/camera/eye coordinate system
  - CCS - clipping coordinate system
  - NDCS - normalized device coordinate system
  - DCS - device/display/screen coordinate system

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 $d = \text{focal length}$
  - can use rotate/translate/scale to move camera
    - demo: Nate Robins tutorial transformations
Viewing in Project 1
- where is camera in template code?
  - 5 units back, looking down -z axis

Convenient Camera Motion
- rotate/translate/scale not intuitive
- arbitrary viewing position
  - eye point, gaze/lookat direction, up vector

Convenient Camera Motion
- rotate/translate/scale not intuitive
- arbitrary viewing position
  - eye point, gaze/lookat direction, up vector

From World to View Coordinates: W2V
- translate eye to origin
- rotate view vector (lookat – eye) to w axis
- rotate around w to bring up into vw-plane

OpenGL Viewing Transformation
- gluLookAt(ex, ey, ez, lx, ly, lz, ux, uy, uz)
  - postmultiplies current matrix, so to be safe:
    
    \[
    \begin{bmatrix}
    1 & 0 & 0 & -ex \\
    0 & 1 & 0 & -ey \\
    0 & 0 & 1 & -ez \\
    0 & 0 & 0 & 1
    \end{bmatrix}
    \]

  - demo: Nate Robins tutorial  projection

Deriving W2V Transformation
- translate eye to origin
  
  \[
  T = \begin{bmatrix}
  1 & 0 & 0 & -e_x \\
  0 & 1 & 0 & -e_y \\
  0 & 0 & 1 & -e_z \\
  0 & 0 & 0 & 1
  \end{bmatrix}
  \]
Deriving W2V Transformation

- rotate view vector (lookat – eye) to w axis
  - w is just opposite of view/gaze vector g
  \[ w = -\hat{g} = -\frac{g}{\|g\|} \]

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
    - Note OpenGL models viewing in left-hand coordinates
    - camera stays put, but square moves to -7
  - resulting image same either way
- possible difference: are lights specified in world or view coordinates?
Projections I

Pinhole Camera
- ingredients
  - box
  - film
  - hole punch
- results
  - pictures!

Pinhole Camera
- theoretical perfect pinhole

Pinhole Camera
- non-zero sized hole

Pinhole Camera
- field of view and focal length

Pinhole Camera
- field of view and focal length
**Real Cameras**

- pinhole camera has small aperture (lens opening)
  - hard to get enough light to expose the film

- lens permits larger apertures
- lens permits changing distance to film plane without actually moving the film plane

price to pay: limited depth of field

**Graphics Cameras**

- real pinhole camera: image inverted

- computer graphics camera: convenient equivalent

**General Projection**

- image plane need not be perpendicular to view plane

**Perspective Projection**

- our camera must model perspective
Perspective Projections
- classified by vanishing points

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

Projective Transformations
- properties
  - lines mapped to lines and triangles to triangles
  - parallel lines do NOT remain parallel
    - e.g. rails vanishing at infinity
  - affine combinations are NOT preserved
    - e.g. center of a line does not map to center of projected line (perspective foreshortening)

Perspective Projection
- project all geometry
  - through common center of projection (eye point)
  - onto an image plane

Basic Perspective Projection
- similar triangles
  \[
  \frac{y'}{y} = \frac{z}{d} \Rightarrow y' = \frac{y \cdot d}{z}
  \]
- nonuniform foreshortening
- not affine
Perspective Projection

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

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

- what could a matrix look like to do this?

Simple Perspective Projection Matrix

\[
\begin{bmatrix}
x \\
z/d \\
y \\
z/d \\
d
\end{bmatrix}
\]

is homogenized version of

where \(w = z/d\)

Simple Perspective Projection Matrix

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

Perspective Projection

- expressible with 4x4 homogeneous matrix
- use previously untouched bottom row
- perspective projection is irreversible
  - many 3D points can be mapped to same \((x, y, d)\) on the projection plane
  - no way to retrieve the unique \(z\) values

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

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

**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

**View Volume**

- 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**

left/right \( x = +/- 1 \), top/bottom \( y = +/- 1 \), near/far \( z = +/- 1 \)

**Understanding Z**

- z axis flip changes coord system handedness
- RHS before projection (eye/view coords)
- LHS after projection (clip, norm device coords)
Understanding Z
near, far always positive in OpenGL calls
- `glOrtho(left, right, bot, top, near, far);`
- `glFrustum(left, right, bot, top, near, far);`
- `glPerspective(fovy, aspect, near, far);`

Orthographic Derivation
- scale, translate, reflect for new coord sys

\[
\begin{align*}
  y' &= a \cdot y + b \\
  y &= \text{top} \Rightarrow y' &= 1 \\
  y &= \text{bot} \Rightarrow y' &= -1
\end{align*}
\]

\[
\begin{align*}
  b &= 1 - a \cdot \text{top}, b = -1 - a \cdot \text{bot} \\
  1 - a \cdot \text{top} &= -1 - a \cdot \text{bot} \\
  1 - (-1) &= -a \cdot \text{bot} - (-a \cdot \text{top}) \\
  2 &= a(-\text{bot} + \text{top}) \\
  a &= \frac{2}{\text{top} - \text{bot}}
\end{align*}
\]

same idea for right/left, far/near
Orthographic Derivation

- scale, translate, reflect for new coord sys

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

Orthographic OpenGL

```c
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glOrtho(left, right, bot, top, near, far);
```

Projections II
NDC to Viewport Transformation

- generate pixel coordinates
- map x, y from range –1…1 (NDC) to pixel coordinates on the display
- involves 2D scaling and translation

 Origin Location

- yet more possibly confusing conventions
  - OpenGL: lower left
  - most window systems: upper left
  - often have to flip your y coordinates
    - when interpreting mouse position

Perspective Example

- tracks in VCS:
  - left x=-1, y=1
  - right x=1, y=1
- view volume
  - left = -1, right = 1
  - bot = -1, top = 1
  - near = 1, far = 4

Viewing Transformation

Projective Rendering Pipeline
**Perspective Projection**

- specific example
  - assume image plane at $z = -1$
  - a point $[x, y, z, 1]^T$ projects to $[-x/z, -y/z, -z/z, 1]^T$ or $[x, y, z, -z]^T$

**Canonical View Volumes**

- standardized viewing volume representation

  - orthographic
    - orthogonal
    - parallel
    - $x$ or $y = \pm z$
  - front plane $z = -1$
  - back plane $z = 1$

**Why Canonical View Volumes?**

- permits standardization
- clipping
- easier to determine if an arbitrary point is enclosed in volume
- consider clipping to six arbitrary planes of a viewing volume versus canonical view volume
- rendering
- projection and rasterization algorithms can be reused

**Projection Normalization**

- one additional step of standardization
- warp perspective view volume to orthogonal view volume
- render all scenes with orthographic projection!

**Predistortion**

- projection transformation after $w$
- perspective division $/w$
Perspective Normalization
- perspective viewing frustum transformed to cube
- orthographic rendering of cube produces same image as perspective rendering of original

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

Perspective Warp
- matrix formulation preserves relative depth (third coordinate)
- what does \( \alpha = 0 \) mean?

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

Projective Rendering Pipeline
Coordinate Systems

Perspective Derivation

earlier:

\[
\begin{bmatrix}
  x' \\
  y' \\
  z' \\
  w'
\end{bmatrix} = \begin{bmatrix}
  E & 0 & A & 0 \\
  0 & F & B & 0 \\
  0 & 0 & C & D \\
  0 & 0 & -1 & 0
\end{bmatrix} \begin{bmatrix}
  x \\
  y \\
  z \\
  w
\end{bmatrix}
\]

complete: shear, scale, projection-normalization

\[
\begin{bmatrix}
  x' \\
  y' \\
  z' \\
  w'
\end{bmatrix} = \begin{bmatrix}
  E & 0 & A & 0 \\
  0 & F & B & 0 \\
  0 & 0 & C & D \\
  0 & 0 & -1 & 0
\end{bmatrix} \begin{bmatrix}
  x \\
  y \\
  z \\
  w
\end{bmatrix}
\]

Perspective Derivation

\[
\begin{align*}
x' &= Ex + Az \\
y' &= Fy + Bz \\
z' &= Cz + D \\
w' &= -z
\end{align*}
\]

\[
\begin{align*}
x &= \text{left} \quad \rightarrow \quad x'/w' = 1 \\
x &= \text{right} \quad \rightarrow \quad x'/w' = -1 \\
y &= \text{top} \quad \rightarrow \quad y'/w' = 1 \\
y &= \text{bottom} \quad \rightarrow \quad y'/w' = -1 \\
z &= -\text{near} \quad \rightarrow \quad z'/w' = 1 \\
z &= -\text{far} \quad \rightarrow \quad z'/w' = -1
\end{align*}
\]

\[
\begin{align*}
y &= Fy + Bz, \quad y' &= \frac{Fy + Bz}{w'}, \quad w &= \frac{Fy + Bz}{-z}, \\
1 &= F \frac{y}{-z} + B \frac{z}{-z}, \quad 1 &= F \frac{y}{-z} - B, \quad 1 = F \frac{\text{top}}{-(-\text{near})} - B, \\
1 &= F \frac{\text{top}}{\text{near}} - B
\end{align*}
\]

Perspective Derivation

- similarly for other 5 planes
- 6 planes, 6 unknowns

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

Perspective Example

view volume

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

\[
\begin{bmatrix}
\frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\
0 & \frac{2n}{t-b} & \frac{r+l}{t-b} & 0 \\
0 & 0 & -(f+n) & -2fn \\
0 & 0 & -1 & f-n
\end{bmatrix} \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & -5/3 & -8/3 \\
0 & 0 & -1 & 0
\end{bmatrix}
\]
**Perspective Example**

\[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
-5z_{VCS}/3 - 8/3 & -5/3 - 8/3 & -z_{VCS} & 1 \\
-3z_{VCS} & -1 & 1 & 0
\end{bmatrix} = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
-5/3 - 8/3 & -1 & z_{VCS} & 1 \\
-1 & 0 & 0 & 1
\end{bmatrix}
\]

\[
x_{NDCS} = -1/z_{VCS}
\]

\[
y_{NDCS} = 1/z_{VCS}
\]

\[
z_{NDCS} = \frac{5}{3} + \frac{8}{3z_{VCS}}
\]

**Asymmetric Frusta**

- our formulation allows asymmetry
- why bother?

**Simpler Formulation**

- left, right, bottom, top, near, far
- nonintuitive
- often overkill
- look through window center
  - symmetric frustum
- constraints
  - left = -right, bottom = -top

**Field-of-View Formulation**

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

**Perspective OpenGL**

```c
glMatrixMode(GL_PROJECTION);
gLoadIdentity();
glFrustum(left,right,bot,top,near,far);
or
glPerspective(fovy,aspect,near,far);
```

**Demo: Frustum vs. FOV**

- Nate Robins tutorial (take 2):
### Projection Taxonomy

- **Planar Projections**
  - Perspective: classified by vanishing points
    - One-point perspective
    - Two-point perspective
    - Three-point perspective

- **Parallel Projection**
  - Projectors are all parallel
  - vs. perspective projectors that converge
  - Orthographic: projectors perpendicular to projection plane
  - Oblique: projectors not necessarily perpendicular to projection plane

- **Axonometric Projections**
  - Projectors perpendicular to image plane
  - Select axis lengths
    - Isometric: equal axes
    - Dimetric: equal angles
    - Trimetric: equal angles

- **Oblique Projections**
  - Projectors oblique to image plane
  - Select angle between front and z axis
    - Lengths remain constant
    - Both have true front view
      - Cavalier: distance true
      - Cabinet: distance half

### Demos

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