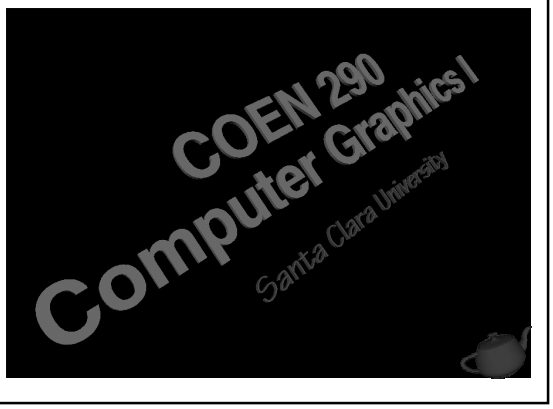


Evening's Goals

- Discuss the mathematical transformations that are utilized for computer graphics
  - projection
  - viewing
  - modeling
- Describe *aspect ratio* and its importance
- Provide a motivation for *homogenous coordinates* and their uses

Mathematical Transformations

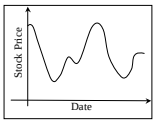
- Use transformations for moving from one coordinate space to another
- The good news
  - only requires multiplication and addition
- The bad news
  - its multiplication and addition of matrices

Mathematical Transformations (cont.)

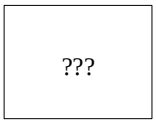
- Coordinate spaces we'll be using
  - model
  - world
  - eye
  - normalized device (NDC's)
  - window
  - screen
  - viewport

Simplified 2D Transform Pipeline

- What if your data is not in viewport coordinates?



World Coordinates



Viewport coordinates

Simplified 2D Transform Pipeline (cont.)

- Need to map world to viewport coordinates
- Simple *linear transformation*
  - linear transformations can be represented by matrices

$$\begin{pmatrix} x \\ y \end{pmatrix}_{viewport} = \begin{pmatrix} x \\ y \end{pmatrix}_{world} \begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

Almost, but not quite

- The 2x2 matrix isn't quite enough to do the whole job
  - think about trying to map a point like (10,10) into the (0,0)
- Enter … *homogenous coordinates*

$$\begin{pmatrix} x & y & 1 \end{pmatrix}$$

  - add an additional "dimension" to your coordinate vector

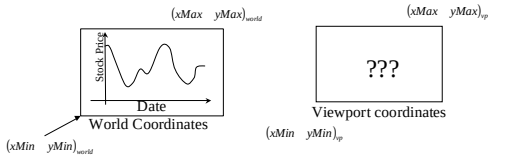
Determining the Matrix Entries

- Matrix forms of linear transforms are shorthand for an "line" equation

$$y = mx + b \Rightarrow \begin{pmatrix} y \\ 1 \end{pmatrix} = \begin{pmatrix} m & b \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ 1 \end{pmatrix}$$

- So what we need is to determine what equations we want to write as matrices

Mapping World to Viewport Coordinates



$$x_{vp} = \frac{xMax_{vp} - xMin_{vp}}{xMax_{world} - xMax_{world}}(x - xMin_{world}) + xMin_{vp}$$

$$y_{vp} = \frac{yMax_{vp} - yMin_{vp}}{yMax_{world} - yMax_{world}}(y - yMin_{world}) + yMin_{vp}$$

Or as a Matrix

- Let

$$m_x = \frac{xMax_{vp} - xMin_{vp}}{xMax_{world} - xMin_{world}} \qquad m_y = \frac{yMax_{vp} - yMin_{vp}}{yMax_{world} - yMin_{world}}$$

then our matrix becomes

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix}_{vp} = \begin{pmatrix} m_x & 0 & xMin_{vp} - m_x \cdot xMin_{world} \\ 0 & m_y & yMin_{vp} - m_y \cdot yMin_{world} \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}_{world}$$

Setting up OpenGL's 2D world

- OpenGL will do this automatically for us

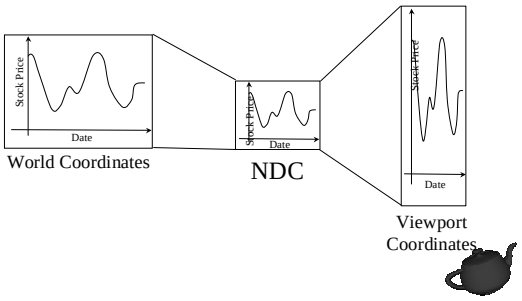
```
gluOrtho2D( xMin, xMax,
            yMin, yMax );
```

- However, it doesn't do it quite as we described
  - first maps world coordinates into *normalized device coordinates (NDC)*
  - then maps from NDC's to viewport coordinates

Normalized Device Coordinates

- Map each dimension linearly into[-1, 1]
  - sometimes mapped to [0, 1]
- Simplifies several things
  - clipping
    - don't need to know viewport for clipping
  - describes a device independent space
    - no concerns about resolution, etc.
  - more things which we'll get to in a minute
    - very useful when we're in 3D

Putting it all together



Err … something doesn't look right

- Need to match *aspect ratio*

$$aspect\ \ ratio = \frac{width}{height}$$

- Aspect ratios of different coordinate spaces need to match

$$ar_{world} = ar_{vp}$$

What's different for 3D

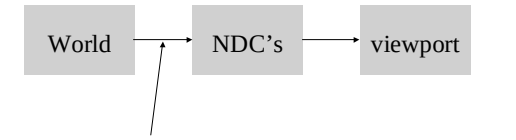
- Add another dimension

$$\begin{pmatrix} x & y & z & w \end{pmatrix}$$

- Our transformation matrices become 4x4
- More options for our projection transform

Where we're at

- What our transformation pipeline looks like so far ...



This is really called a *projection transform*

Projection Transformations

- Map coordinates into NDC's
- Defines our *viewing frustum*
  - sets the position of our *imaging plane*
- Two types for 3D
  - *Orthographic* (or parallel) Projection
    - **`gluOrtho2D`**`()`
  - *Perspective* Projection

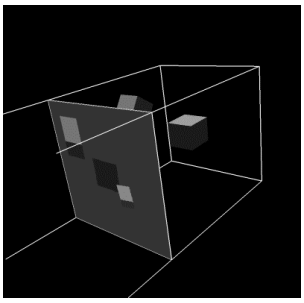
A Few Definitions First …

- A *viewing frustum* is the region in space in which objects can be seen
  - All of the visible objects in our scene will be in the viewing frustum
- The *imaging plane* is a plane in space onto which we project our scene
  - viewing frustum controls where the imaging plane is located

Orthographic Projections

- Project objects in viewing frustum without distortion
  - good for computer aided engineering and design
    - preserves angles and relative sizes

Orthographic Projections (cont.)



Defining an Orthographic Projection

- Very similar to mapping 2D to NDC's
- Use OpenGL's

```
glOrtho( l, r, b, t, n, f );
```

$$\begin{pmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & \frac{-2}{f-n} & -\frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

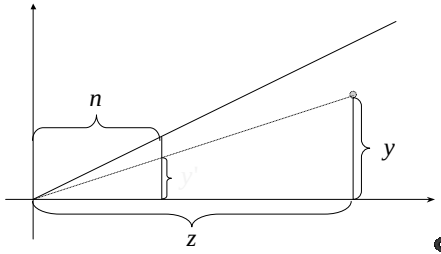
Perspective Projections

- Model how the eye sees
  - objects farther from the eye are smaller
- A few assumptions
  - eye is positioned at the world space origin
  - looking down the world -z axis
- Clipping plane restrictions

$$0 < near < far$$

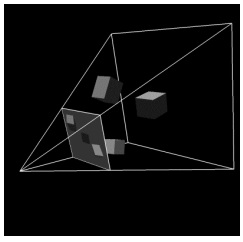
Perspective Projections (cont.)

- Based on similar triangles



Perspective Projections (cont.)

- Viewing frustum looks like a truncated Egyptian pyramid



Defining Perspective Projections

- Two OpenGL versions
  - **`glFrustum`**`( l, r, b, t, n, f );`
    - frustum not necessarily aligned down line of sight
    - good for simulating peripheral vision
  - **`gluPerspective`**`( fovy, aspect, n, f );`
    - frustum centered down line of sight
    - more general form
    - reasonable values: 65.0 < *fovy* < 140.0
    - `aspect` should match aspect ratio of viewport

Defining a Perspective Projection

`glFrustum``( l, r, b, t, n, f );`

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

Defining a Perspective Projection (cont.)

`gluPerspective``( fovy, aspect, n, f );`

$$t = n \cdot \tan(\frac{fovy}{2})$$

$$b = -t$$

$$r = t \cdot aspect$$

$$l = -r$$

- then use **`glFrustum`**`( )`

Clipping in 3D

- Projections transforms make clipping easy
- Use your favorite algorithm
- Clipping region well defined

$$-w \geq x \geq w$$
$$-w \geq y \geq w$$
$$-w \geq z \geq w$$

Normalizing Projected Coordinates

- w is a scaling factor
- *Perspective divide*
  - divide each coordinate by w
  - maps into NDC's

What about z?

$$\begin{pmatrix} \frac{x}{w} \\ \frac{y}{w} \\ \frac{z}{w} \\ 1 \end{pmatrix}$$