Lecture 24:
More OpenGL
Two Strategies for Drawing

- **Transform on draw**
  - good for a lot of objects that are transforming often
  - TRRRV

- **Transform then draw**
  - good if only some objects are transforming sometimes
  - TRRRTTTV when transforming
  - just V when drawing!
Transform On Draw

for each object
  push matrix
  set up transformations
  draw object at the origin
  pop matrix
Transform Then Draw

- Code values of vertices in data structures
- When Transforming (independent of draw)
  - Set up transforms
  - transform object
Ordering of Operations

- In OpenGL, operations occur in the reverse order they’re specified:

```
glRotated(RotX, 1, 0, 0);
glRotated(RotY, 0, 1, 0);
glTranslated(TransX, TransY, TransZ);
glBegin(...
g glVertex3d(...
```

$R_x R_y TV$
glTranslated(CT_x, CT_y, CT_z)
for each object i:
   glPushMatrix();
   glTranslate(TX_i, TY_i, TZ_i);
   glRotated(RotX_i, 1, 0, 0);
   glRotated(RotY_i, 0, 1, 0);
   glRotated(RotZ_i, 0, 0, 1);
   DrawObject(i);
   glPopMatrix();

T_cT_iR_{X_i}R_{Y_i}R_{Z_i}V_i
The OpenGL World

- Z is negative into screen:

All ModelView transforms are in this coordinate system

Recall: \( P \cdot MV \cdot V \)
Moving the Camera, 3 DOF

- Translation only - easy, always looking into - Z
- To move camera forward:
  - glTranslate(0,0,positive_value);
- To move camera backward:
  - glTranslate(0,0,negative_value);
Moving the Camera, 6 DOF

- Now, add roll, pitch, and yaw
- or, RotZ, RotX, RotY
- Have to calculate what a “step forward” means
Moving the Camera (con’t)

- first do translations, taking into acc’t rotations
  - \( dX = -\sin(\text{RotY}) \cdot \text{Velocity} \)
  - \( dY = -\sin(\text{RotX}) \cdot \text{Velocity} \);
  - \( dZ = \cos(\text{RotY}) \cdot \cos(\text{RotX}) \cdot \text{Velocity} \);
  - \( \text{glTranslate}(C_x + dX, \ldots) \)
- then do rotations
  - \( \text{glRotated}(\text{theta}, \text{AboutX}, \text{AboutY}, \text{AboutZ}) \);
Rotating the World

- Make a rotation the first transform:
  - `glRotate(...`
  - update camera
  - for each object...

\[
R_{Cx} \cdot R_{Cy} \cdot R_{Cz} \cdot T_C \cdot R_{Wx} \cdot R_{Wy} \cdot R_{Wz} \cdot (T_{Oi} \cdot R_{Oix} \cdot R_{Oiy} \cdot R_{Oiz}) \cdot V_i
\]
Using the OpenGL Engine

- good for Transform then Draw
- use glMultMatrix(type *d)
- only good for 4x4
- example:

  ```
  glPushMatrix();
  glMultMatMatrixf(data);
  //now get the data
  float m[16];
  glGetFloatv(GL_MODELVIEW_MATRIX,m);
  ```
Using Your Own Engine

- float TransMatrix[16];
- float MyVertices[64]; // 4 x 16
- MultMat(TransMatrix, MyVertices,...)
Input

- Mouse
  - glutMouseFunc
  - glutMotionFunc

- Keyboard
  - glutKeyboardFunc
glutMouseFunc

- glutMouseFunc(void (*func)(int button, int state, int x, int y));
- button is GLUT_LEFT_BUTTON, GLUT_MIDDLE_BUTTON, or GLUT_RIGHT_BUTTON
- state is GLUT_UP or GLUT_DOWN
glutMotionFunc

- glutMotionFunc(void (*func)(int x, int y));
- x and y are the cursor coordinates after the move
glutKeyboardFunc

- glutKeyboardFunc(void (*func)(unsigned int key, int x, int y));
- key is character, for example ‘a’
- x and y are mouse cursor location
Remember Your Matrices!

World_Transforms • Camera • Object-Transforms
• Object_Vertices