CS312

OpenGL
Modeling Transformations
OpenGL Matrices

- In OpenGL matrices are part of the state
- Three types
  - Model-View (GL_MODELVIEW)
  - Projection (GL_PROJECTION)
  - Texture (GL_TEXTURE) (ignore for now)
- Single set of functions for manipulation
- Select which to manipulated by
  - glMatrixMode(GL_MODELVIEW);
  - glMatrixMode(GL_PROJECTION);
Conceptually there is a 4 x 4 homogeneous coordinate matrix, the *current transformation matrix* (CTM) that is applied to all vertices that pass down the pipeline.

The CTM is defined in the user program and loaded into a transformation unit.
CTM operations

The CTM can be altered either by loading a new CTM or by postmultiplication

Load an identity matrix: \( C \leftarrow I \)
Load an arbitrary matrix: \( C \leftarrow M \)
Load a translation matrix: \( C \leftarrow T \)
Load a rotation matrix: \( C \leftarrow R \)
Load a scaling matrix: \( C \leftarrow S \)

Postmultiply by an arbitrary matrix: \( C \leftarrow CM \)
Postmultiply by a translation matrix: \( C \leftarrow CT \)
Postmultiply by a rotation matrix: \( C \leftarrow CR \)
Postmultiply by a scaling matrix: \( C \leftarrow CS \)
CTM in OpenGL

- OpenGL has a model-view and a projection matrix in the pipeline which are concatenated together to form the CTM
- Can manipulate each by first setting the correct matrix mode
Matrix Operations

- Specify current matrix stack
  ```c
  glMatrixMode(GL_MODELVIEW) or
  glMatrixMode(GL_PROJECTION)
  ```

- Matrix operations
  - `glLoadIdentity()`
  - `glPushMatrix()`
  - `glPopMatrix()`
  - `glLoadMatrix()`
  - `glMultMatrix()`
Modeling Transformations

- **Translation**
  - `glTranslatef(fd)(x,y,z)`

- **Rotation around arbitrary axis**
  - `glRotate(fd)(angle, x,y,z)`

- **Scaling**
  - `glScale(fd)(x,y,z)`

- Multiplies onto the current matrix (use `GL_MODELVIEW`)
Order of Transformations

- OpenGL post-multiples matrices
- Operations occur in reverse order

```c
glLoadIdentity();
glMultMatrix(M);
glMultMatrix(N);    CIMNO(v)
glMultMatrix(O);
glBegin(GL_POINTS);
glVertex3fv(v);
glEnd();
```
Post-multiplication: Rotation about a Fixed Point

- Start with identity matrix: \( C \leftarrow I \)
- Move fixed point to origin: \( C \leftarrow CT \)
- Rotate: \( C \leftarrow CR \)
- Move fixed point back: \( C \leftarrow CT^{-1} \)
- Result: \( C = TR T^{-1} \) which is backwards.
- This result is a consequence of doing postmultiplications.
Reversing the Order

- We want \( C = T^{-1} R T \)
  - \( C \leftarrow I \)
  - \( C \leftarrow CT^{-1} \)
  - \( C \leftarrow CR \)
  - \( C \leftarrow CT \)
  - Each operation corresponds to one function call in the program.

- Note that the last operation specified is the first executed in the program.
Example

- Rotation about z axis by 30 degrees with a fixed point of (1.0, 2.0, 3.0)

```c
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glTranslatef(1.0, 2.0, 3.0);
glRotatef(30.0, 0.0, 0.0, 1.0);
glTranslatef(-1.0, -2.0, -3.0);
```

- Remember that last matrix specified in the program is the first applied
Arbitrary Matrices

- Can load and multiply by matrices defined in the application program

  ```
  glLoadMatrixf(m)
  glMultMatrixf(m)
  ```

- The matrix \( m \) is a one dimension array of 16 elements which are the components of the desired 4 x 4 matrix stored by columns

- In `glMultMatrixf`, \( m \) multiplies the existing matrix on the right
Matrix Stacks

- In many situations we want to save transformation matrices for use later
  - Traversing hierarchical data structures
  - Avoiding state changes when executing display lists (introduced later)

- OpenGL maintains stacks for each type of matrix
  - Access present type (as set by `glMatrixMode`) by `glPushMatrix()`, `glPopMatrix()`
Matrix Stack

- Code often looks like this:

```c
glPushMatrix();
glTranslatef(...);
glRotatef(...);
/* draw object */
glPopMatrix();
```
Reading Back Matrices

- Can also access matrices (and other parts of the state) by *query* functions
  
  \[
  \text{glGetIntegerv,}\ \text{glGetFloatv,}\ \text{glGetBooleanv,}\ \text{glGetDoublev,}\ \text{glIsEnabled}\n  \]

- For matrices, we use as

  ```c
  double m[16];
  \text{glGetFloatv(GL\_MODELVIEW, } m);\n  ```
Smooth Rotation

- From a practical standpoint, we are often want to use transformations to move and reorient an object smoothly.
  - Problem: find a sequence of model-view matrices $M_0, M_1, \ldots, M_n$ so that when they are applied successively to one or more objects we see a smooth transition.

- For orientating an object, we can use the fact that every rotation corresponds to part of a great circle on a sphere.
  - Find the axis of rotation and angle.
Incremental Rotation

- Consider the two approaches
  - For a sequence of rotation matrices $R_0, R_1, \ldots, R_n$, find the Euler angles for each and use $R_i = R_{iz} R_{iy} R_{ix}$
  - Not very efficient
  - Use the final positions to determine the axis and angle of rotation, then increment only the angle
void draw() {
    glPushMatrix();
    glRotatef(angle, 0,0,1);
    // draw
    glPopMatrix();
    glutSwapBuffers();
}
void animate() {
    angle += 2.0;
    glutPostRedisplay();
}

glutIdleFunc(animate);
Double buffering

- Two color buffers so that when one is displayed, the other is being redrawn.
- When drawing is complete, buffers are swapped.
- The viewer never sees an incompletely drawn buffer.
- Eliminates flickering.
Animation using Double Buffering

- Requests a double buffered color buffer
- Clear color buffer
  
  `glClear(GL_COLOR_BUFFER_BIT)`
- Render scene
- Request swapping of front and back buffers
Double buffering in GL

- `glInitDisplayMode(GLUT_DOUBLE);`
- `void display() {
   glClear(GL_COLOR_BUFFER_BIT);
   ...
   glutSwapBuffers();
}
- `glutSwapBuffers()` flushes automatically