Lecture Outline

- Reading for Last Class: §2.3 (esp. 2.3.4), Foley et al. Slides
- Reading for Today: Chapters 2, 16, Eberly 2
- Reading for Next Class: §2.3 (esp. 2.3.7), 2.6, 2.7, Eberly 2
- Last Time: Matrix Stack for 3-D Viewing Transformation
  \[ N = M_{\text{proj}} S_{\text{clip}} S_{\text{eye}} M_{\text{model}} \]
  - Perspective: optical principles, terminology

Today: Highlights from First of Three Tutorials on OpenGL (Three Parts)
1. OpenGL and GL Utility Toolkit (GLUT) – V. Shreiner
2. Basic rendering – V. Shreiner
3. 3-D viewing setup – E. Angel

Next Class: Scan Conversion (Rasterization) of Lines, Polygons

Where We Are

Review:

- Viewing Transformation
  \[ P = \text{CTM} \cdot \text{Eye} \]
  - \( \text{CTM} = \text{Norm} \cdot \text{Model} \cdot \text{Proj} \)
  - \( \text{Proj} = \text{Persp} \cdot \text{Proj} \)
  - \( \text{Model} = \text{Trans} \cdot \text{Rot} \cdot \text{Scale} \)

CTM for “Polygons-to-Pixels” Pipeline

- Entire problem can be reduced to a composite matrix multiplication of vertices, clipping, and a final matrix multiplication to produce screen coordinates.
- Final composite matrix (CTM) is composed of all modeling (instance) transformations (CMTM) accumulated during scene graph traversal from root to leaf, combined with the final composite normalization transformation \( N \) applied to the root/world coordinate system.
  \[ N = \begin{pmatrix} D_{\text{proj}} & 0 & 0 & 0 \\ 0 & S_{\text{clip}} & 0 & 0 \\ 0 & 0 & T_{\text{eye}} & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \]
  \[ \text{CTM} = \begin{pmatrix} C_{\text{model}} & T_{\text{model}} \\ 0 & 1 \end{pmatrix} \]
  \[ P = \text{CTM} \cdot P \] for every vertex or triangle in the root coordinate system
  \[ P_{\text{new}} = 512 \cdot P_{\text{old}} + 1 \] for a clipped vertex

Lab 1 of 7: Part A [1]:
Setup, OpenGL/Mesa, & XWindows

CIS 616 Interactive Computer Graphics
CIS 716 Computer Graphics

Lab 1 of 7: OpenGL Setup and Basics
Lab 1 of 7, Part A [2]: Perspective View Volume Specification

4. (10%) Step 1: Perspective Viewing with OpenGL
   glMatrixMode( GL_PROJECTION );
   glLoadIdentity();
5. (10%) Step 2: Making things move in Perspective
   glMatrixMode( GL_MODELVIEW );
   glLoadIdentity();
   gluPerspective( field_of_view, aspect, near, far );
6. (10%) Step 3: Specifying the Viewing matrix
   glmatrix( eyep, eyep, eyep,
             eyep, eyep, eyep,
             centerX, centerZ, centerZ,
             eyeX, eyeY, eyeZ );
7. (10%) Step 4: Instantiating Objects

OpenGL Rendering
Application Programmer Interface

- Geometric primitives
  - points, lines and polygons
- Image Primitives
  - images and bitmaps
    - separate pipeline for images and geometry
      - linked through texture mapping
- Rendering depends on state
  - colors, materials, light sources, etc.

Preliminaries

- Headers Files
  - #include <GL/gl.h>
  - #include <GL/glu.h>
- Libraries
- Enumerated Types
  - OpenGL defines numerous types for compatibility
    - GLfloat
    - GLint
    - GLenum
    - etc.

OpenGL & Related APIs

- AGL, GLX, WGL
  - glue between OpenGL and windowing systems
- GLU (OpenGL Utility Library)
  - part of OpenGL
  - NURBS, tessellators, quadric shapes, etc.
- GLUT (OpenGL Utility Toolkit)
  - portable windowing API
  - not officially part of OpenGL

GLUT Callback Functions

- Application Structure
  - Configure and open window
  - Initialize OpenGL state
  - Register input callback functions
    - render
    - resize
    - input: keyboard, mouse, etc.
  - Enter event processing loop

Open GL Architecture

CPU
- Display List
- Polynomial Evaluator
- Per Vertex Operations & Primitive Assembly
- Rasterization
- Per Fragment Operations
- Frame Buffer
- Texture Memory
- Pixel Operations

© 2000 Shreiner, D., Angel, E., Shreiner, V.
void main( int argc, char** argv )
{
    int mode = GLUT_RGB|GLUT_DOUBLE;
    glutInitDisplayMode( mode );
    glutCreateWindow( argv[0] );
    init();
    glutDisplayFunc( display );
    glutReshapeFunc( resize );
    glutKeyboardFunc( key );
    glutIdleFunc( idle );
    glutMainLoop();
}

© 2000 Shreiner, D., Angel, E., Shreiner, V.
Simple Example

```c
void drawRhombus( GLfloat color[] )
{
    glBegin( GL_QUADS );
    glColor3fv( color );
    glVertex2f( 0.0, 0.0 );
    glVertex2f( 1.0, 0.0 );
    glVertex2f( 1.5, 1.118 );
    glVertex2f( 0.5, 1.118 );
    glEnd();
}
```

OpenGL Command Formats

```
glVertex3fv( v )
```

- **Number of components**
  - 2 - (x,y)
  - 3 - (x,y,z)
  - 4 - (x,y,z,w)

- **Data Type**
  - b - byte
  - ub - unsigned byte
  - s - short
  - us - unsigned short
  - i - int
  - ui - unsigned int
  - f - float
  - d - double

- Vector
  - omit "v" for scalar form
  - `glVertex2f( x, y )`

Specifying Geometric Primitives

- Primitives are specified using
  ```
  glEnd();
  ```

- `primType` determines how vertices are combined

```
GLfloat red, green, blue;
GLfloat coords[3];
glBegin( primType );
for ( i = 0; i < nVerts; ++i ) {
    glColor3f( red, green, blue );
    glVertex3fv( coords );
}
glEnd();
```

Transformations in OpenGL

- Modeling
- Viewing
  - orient camera
  - projection
- Animation
- Map to screen

```
Camera Analogy

- 3D is just like taking a photograph (lots of photographs!)
```

```
Camera Analogy & Transformations

- Projection transformations
  - adjust the lens of the camera
- Viewing transformations
  - tripod--define position and orientation of the viewing volume in the world
- Modeling transformations
  - moving the model
- Viewport transformations
  - enlarge or reduce the physical photograph
```

```
```
Steps in Forming an Image
- specify geometry (world coordinates)
- specify camera (camera coordinates)
- project (window coordinates)
- map to viewport (screen coordinates)

Each step uses transformations
Every transformation is equivalent to a change in coordinate systems (frames)

Matrix Operations
- Specify Current Matrix Stack
  `glMatrixMode( GL_MODELVIEW or GL_PROJECTION );`
- Other Matrix or Stack Operations
  `glLoadIdentity();`  `glPushMatrix();`  `glPopMatrix();`
- Viewport
  - usually same as window size
  - viewport aspect ratio should be same as projection transformation or resulting image may be distorted
  `glViewport( x, y, width, height );`

Projection Transformation
- Shape of viewing frustum
- Perspective projection
  `gluPerspective( fovy, aspect, zNear, zFar );`
  `glFrustum( left, right, bottom, top, zNear, zFar );`
- Orthographic parallel projection
  `glOrtho( left, right, bottom, top, zNear, zFar );`
  `gluOrtho2D( left, right, bottom, top );`  
calls `glOrtho` with z values near zero

Applying Projection Transformations
- Typical use (orthographic projection)
  `glMatrixMode( GL_PROJECTION );`
  `glLoadIdentity();`
  `glOrtho( left, right, bottom, top, zNear, zFar );`

Viewing Transformations
- Position the camera/eye in the scene
  - place the tripod down; aim camera
- To “fly through” a scene
  - change viewing transformation and redraw scene
  `gluLookAt( eye, eye, eye, aim, aim, aim, up, up, up );`  
  - up vector determines unique orientation
  - careful of degenerate positions
### Projection Tutorial

- Move object:
  \[
  \text{glTranslate}(x, y, z)
  \]
- Rotate object around arbitrary axis:
  \[
  \text{glRotate}(\text{angle}, x, y, z)
  \]
- Dilate (stretch or shrink) or mirror object:
  \[
  \text{glScale}(x, y, z)
  \]

### Modeling Transformations

- Move object:
  \[
  \text{glTranslate}(x, y, z)
  \]
- Rotate object around arbitrary axis:
  \[
  \text{glRotate}(\text{angle}, x, y, z)
  \]
  (x y z)
- Dilate (stretch or shrink) or mirror object:
  \[
  \text{glScale}(x, y, z)
  \]

### Transformation Tutorial

- Moving camera is equivalent to moving every object in the world towards a stationary camera.
- Viewing transformations are equivalent to several modeling transformations.
  - `gluLookAt()` has its own command.
  - Can make your own polar view or pilot view.

### Connection: Viewing and Modeling

- Projection transformations (`gluPerspective`, `glOrtho`) are left handed.
  - Think of `zNear` and `zFar` as distance from viewpoint.
- Everything else is right handed, including the vertices to be rendered.

### Projection is Left-Handed

-Projection transformations (`gluPerspective`, `glOrtho`) are left handed.
  - Think of `zNear` and `zFar` as distance from viewpoint.
-Everything else is right handed, including the vertices to be rendered.

### resize(): Perspective & LookAt

```c
void resize( int w, int h )
{
  glViewport( 0, 0, (GLsizei) w, (GLsizei) h );
  glMatrixMode( GL_PROJECTION );
  gluPerspective( 65.0, (GLfloat) w / h,
                  1.0, 100.0 );
  glMatrixMode( GL_MODELVIEW );
  glLoadIdentity();
  gluLookAt( 0.0, 0.0, 5.0,
             0.0, 0.0, 0.0,
             0.0, 1.0, 0.0 );
}
```
```c
void resize( int width, int height )
{
    GLdouble aspect = (GLdouble) width / height;
    GLdouble left = -2.5, right = 2.5;
    GLdouble bottom = -2.5, top = 2.5;
    glViewport( 0, 0, (GLsizei) w, (GLsizei) h );
    glMatrixMode( GL_PROJECTION );
    glLoadIdentity();
    … continued …
    if ( aspect < 1.0 ) {
        left /= aspect;
        right /= aspect;
    } else {
        bottom *= aspect;
        top *= aspect;
    }
    glOrtho( left, right, bottom, top, near, far );
    glMatrixMode( GL_MODELVIEW );
    glLoadIdentity();
}
```

**Summary**

- Three Tutorials from SIGGRAPH 2000
  - Overall architecture
  - Initialization
  - Viewport management
  - Part 2: Basic Rendering – Vicki Shreiner
  - Part 3: 3-D Viewing – Edward Angel
  - Math background (see CG Basics 1)
  - Viewing and normalization transformations (see CG Basics 4)
  - More on viewing in CG Basics 4
  - View volume specification
  - Automated part: clipping

**Terminology**

- **OpenGL and GL Utility Toolkit (GLUT)**
- **State machine**
- **Using GLUT**
- **Specifying perspective, parallel projections**
- **Transformations**
  - Fixed function pipeline: modelview, normalizing, viewing
  - **Rigid body**: preserves distance (e.g., translation, rotation)
  - **Linear**
    - Preserves vector addition, scalar multiplication
    - e.g., rotation, scaling
  - **Affine**: linear transformation followed by translation
  - **Non-affine**: all others (e.g., perspective-to-parallel transformation)