Projections

Ed Angel

Transformations in OpenGL

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

Camera Analogy and 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

Coordinate Systems and Transformations

- 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)

Programming Transformations

- Prior to rendering, view, locate, and orient:
  - eye/camera position
  - 3D geometry
- Manage the matrices
  - including matrix stack
- Combine (composite) transformations
An Interactive Introduction to OpenGL Programming

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

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

Perspective Projection
\textit{glFrustum(l, r, b, t, n, f)}

Orthographic Projection
\textit{glOrtho(l, r, b, t, n, f)}
An Interactive Introduction to OpenGL Programming

**gluLookAt**

- void gluLookAt(GLdouble ex, GLdouble ey, GLdouble ez, GLdouble cx, GLdouble cy, GLdouble cz, GLdouble px, GLdouble py, GLdouble pz);

  Defines a viewing matrix and multiplies it to the right of the current matrix. The desired viewpoint is specified by ex, ey, and ez. The cx, cy, and cz arguments specify any point along the desired line of sight, but typically they’re some point in the center of the scene being looked at. The px, py, and pz arguments indicate which direction is up (that is, the direction from the bottom to the top of the viewing volume).

**Viewport Transformation**

- void glViewport(GLint x, GLint y, GLsizei width, GLsizei height);

  Defines a pixel rectangle in the window into which the final image is mapped. The (x, y) parameter specifies the lower-left corner of the viewport, and width and height are the size of the viewport rectangle. By default, the initial viewport values are (0, 0, winWidth, winHeight), where winWidth and winHeight are the size of the window.

**Transformation Pipeline**

- object — modelview matrix — projection matrix — clip — normalized device coordinates — window

  - other calculations here
    - material, color
    - shade model (flat, gouraud, phong)
    - polygon rendering mode
    - polygon culling
    - clipping

**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_x, eye_y, eye_z,
    aim_x, aim_y, aim_z,
    up_x, up_y, up_z )

  - up vector determines unique orientation

  - careful of degenerate positions
Connection: Viewing and Modeling

- 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

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 vertexes to be rendered

Common Transformation Usage

- 3 examples of `resize()` routine
  - Restate projection & viewing transformations
  - Usually called when window resized
  - Registered as callback for `glutReshapeFunc()`

Resize(): Perspective & LookAt

```c
void resize( int w, int h )
{
    glViewport( 0, 0, (GLsizei) w, (GLsizei) h);
    glMatrixMode( GL_PROJECTION );
    glLoadIdentity();
    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 );
}
```

Resize(): Perspective & Translate

- Same effect as previous LookAt
```c
void resize( int w, int h )
{
    glViewport( 0, 0, (GLsizei) w, (GLsizei) h);
    glMatrixMode( GL_PROJECTION );
    glLoadIdentity();
    gluPerspective( 65.0, (GLfloat) w/h,
                    1.0, 100.0 );
    glMatrixMode( GL_MODELVIEW );
    glLoadIdentity();
    glTranslatef( 0.0, 0.0, -5.0 );
}
```

Relate to Msys

- Msys = Mscreen Mpersp Mview
**resize(): Ortho (part 1)**

```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 ...
```

**resize(): Ortho (part 2)**

```c
if ( aspect < 1.0 ) {
    left /= aspect;
    right /= aspect;
} else {
    bottom *= aspect;
    top *= aspect;
}
gluOrtho( left, right, bottom, top, near, far );
glMatrixMode( GL_MODELVIEW );
glLoadIdentity();
```

**Additional Clipping Planes**

- At least 6 more clipping planes available
- Good for cross-sections
- Modelview matrix moves clipping plane
  - \( Ax + By + Cz + D \leq 0 \)
  - \( glClipPlane( GL_CLIP_PLANE1 ) \)
  - \( glClipPlane( GL_CLIP_PLANE1, GLdouble* coeff ) \)

**Reversing Coordinate Projection**

- Screen space back to world space
  - \( glGetIntegerv( GL_VIEWPORT, GLint viewport[4] ) \)
  - \( glGetDoublev( GL_MODELVIEW_MATRIX, GLdouble* mvmatrix[16] ) \)
  - \( glGetDoublev( GL_PROJECTION_MATRIX, GLdouble* projmatrix[16] ) \)
  - \( gluUnProject( GLdouble winx, winy, winz, mvmatrix[16], projmatrix[16], GLint viewport[4], GLint* objx, GLint* objy, GLint* objz ) \)
  - gluProject goes from world to screen space

**Compositing Modeling Transformations**

- **Problem 1: hierarchical objects**
  - one position depends upon a previous position
  - robot arm or hand: sub-assemblies
- **Solution 1: moving local coordinate system**
  - modeling transformations move coordinate system
  - post-multiply column-major matrices
  - OpenGL post-multiplies matrices
- **Problem 2: objects move relative to absolute world origin**
  - my object rotates around the wrong origin
  - make it spin around its center or something else
- **Solution 2: fixed coordinate system**
  - modeling transformations move objects around fixed coordinate system
  - pre-multiply column-major matrices
  - OpenGL post-multiplies matrices
  - must reverse order of operations to achieve desired effect