OpenGL projection, basic viewing and event handling
Coordinate Systems

- The units in `glVertex` are determined by the application and are called *object coordinates*
- In OpenGL object coordinates are first converted to *world coordinates*
- The viewing specifications are also in object coordinates and it is the size of the viewing volume that determines what will appear in the image
- Internally, OpenGL will convert to *camera coordinates* and later to *screen coordinates*
OpenGL Camera

- OpenGL places a camera at the origin pointing in the negative $z$ direction
- The default viewing volume is a box centered at the origin with a side of length 2
Orthographic Viewing

In the default orthographic view, points are projected forward along the $z$ axis onto the plane $z=0$. 

![Diagram of orthographic viewing]

In the orthographic view, points are projected forward along the $z$ axis onto the plane $z=0$. This projection method is useful for representing 3D objects in a 2D space, preserving the relative sizes of objects along the $x$ and $y$ axes.
Transformations are performed through multiplying a matrix onto the current matrix

- `glMatrixMode(GL_PROJECTION);`
- `glLoadIdentity();`

Defines the view volume, i.e. what is visible, and what is to be clipped off.
Orthographical Projection

- Creates a rectangular viewing volume
- Distance from camera does not affect size
- Creates a matrix for projecting 2D coordinates onto the screen and multiply the current projection matrix by it

```c
void gluOrtho2D(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top);
```
Two- and three-dimensional viewing

- In `glOrtho(left, right, bottom, top, near, far)` the near and far distances are measured from the camera.
- Two-dimensional vertex commands place all vertices in the plane \( z=0 \).
- If the application is in two dimensions, we can use the function
  \[
  \text{gluOrtho2D(left,right,bottom,top)}
  \]
- In two dimensions, the view or clipping volume becomes a *clipping window*.
Set up viewing

```c
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glOrtho(-1.0,1.0,-1.0,1.0,-1.0,1.0);
// or glOrtho2D(-1.0,1.0,-1.0,1.0);

glMatrixMode(GL_MODELVIEW);
```
Viewports

- Do not have use the entire window for the image: `glViewport(x, y, w, h)`
- Values in pixels (screen coordinates)
Physical Devices

- mouse
- trackball
- light pen
- data tablet
- joy stick
- space ball
Input Modes

- Input devices contain a trigger which can be used to send a signal to the operating system
  - Button on mouse
  - Pressing or releasing a key
Request Mode

- Input provided to program only when user triggers the device
- Typical of keyboard input
  - Can erase (backspace), edit, correct until enter (return) key (the trigger) is depressed
Event Mode

- Most systems have more than one input device, each of which can be triggered at an arbitrary time by a user.
- Each trigger generates an event whose measure is put in an event queue which can be examined by the user program.
Event Types

- Window: resize, expose, iconify
- Mouse: click one or more buttons
- Motion: move mouse
- Keyboard: press or release a key
- Idle: nonevent
  - Define what should be done if no other event is in queue
Callback functions

- Called when something happens
  - Window resize or redraw
  - User input
  - Animation

- Register callbacks with GLUT
  - glutDisplayFunc(display);
  - glutIdleFunc(idle);
GLUT Event Callbacks

Callback actions:

- `glutDisplayFunc();` // window redraw
- `glutKeyboardFunc();` // a key is struck
- `glutReshapeFunc();` // window resizes
- `glutMouseFunc();` // mouse button press
- `glutMotionFunc();` // mouse moves and // button held
- `glutPassiveMotionFunc();` // mouse moves
- `glutIdleFunc();` // on idle
Important callbacks

- **Display**
  - Called every time the main GL window is drawn/refreshed
  - This is where you do all of your rendering

- **Idle**
  - Use for animation and continuous update
  - Update some variables/data structures and call `glutPostRedisplay()`
Remember that the last line in `main.c` for a program using GLUT must be:

```c
glutMainLoop();
```

which puts the program in an infinite event loop.

In each pass through the event loop, GLUT:

- looks at the events in the queue
- for each event in the queue, GLUT executes the appropriate callback function if one is defined
- if no callback is defined for the event, the event is ignored
Posting redisplay

- Many events may invoke the display callback function
  - Can lead to multiple executions of the display callback on a single pass through the event loop
- We can avoid this problem by instead using `glutPostRedisplay()` which sets a flag.
- GLUT checks to see if the flag is set at the end of the event loop
- If set then the display callback function is executed
Using globals

- The form of all GLUT callbacks is fixed
  - `void display()`
  - `void mouse(GLint button, GLint state, GLint x, GLint y)`
- Must use globals to pass information to callbacks

```c
float t; /*global */

void display() {
  /* draw something that depends on t */
}
```
Mouse

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

- A window is measured in pixels with the origin at the top-left corner
  - Consequence of refresh done top to bottom
- OpenGL uses a world coordinate system with origin at the bottom left
  - Must invert y coordinate returned by callback by height of window
  - \[ y = h - y; \] (0,0)
Terminating a program

- In our original programs, there was no way to terminate them through OpenGL
- We can use the simple mouse callback

```c
void mouse(int btn, int state, int x, int y){
    if(btn==GLUT_RIGHT_BUTTON && state==GLUT_DOWN)
        exit(0);
}
```
Using the keyboard

- `glutKeyboardFunc(keyboard)`
- `Void keyboard(unsigned char key, int x, int y)`
  - ASCII code of key depressed and mouse location
  - Note GLUT does not recognize key release as an event
void keyboard (unsigned char key, int x, int y){
    switch(key) {
        case 'q': case 'Q': case 27:
            exit (0);
            break;
        case 'p': case 'P':
            paused = 1;
            break;
    }
}
Key modifiers and special keys

- **int glutGetModifiers(void);**
  - GLUT_ACTIVE_SHIFT
  - GLUT_ACTIVE_ALT
  - GLUT_ACTIVE_CTRL
- **void glutSpecialFunc(void (*func) (int key, int x, int y));**
  - GLUT_KEY_F1 (F2 ... F12)
  - GLUT_KEY_UP (DOWN, LEFT, RIGHT)
  - GLUT_KEY_PAGEUP (PAGEDOWN, HOME, END, INSERT)
  - passing in **NULL** will cause these keys to be ignored
Reshaping the window

- Resize the OpenGL display window by pulling the corner of the window

- What happens to the display?
  - Must redraw from application
  - Two possibilities
    - Display part of world
    - Display whole world but force to fit in new window
      - Can alter aspect ratio
Reshape possibilities

original

reshaped
Window reshape

- Viewport transformation:
  - Maps image into window coordinates
  - Mostly called in the resize function

```c
void glutReshapeFunc(void (*func)(int width, int height));

void reshape(int w, int h) {
    // Set the viewport to be the entire window
    glViewport(0, 0, (GLint)w, (GLint)h);
}
```
The Reshape callback

- A redisplay is posted automatically at end of execution of the callback.
- GLUT has a default reshape callback but you probably want to define your own.

- The reshape callback is a good place to put camera functions because it is invoked when the window is first opened and every time it is changed.
Example Reshape

- Project the viewport to window coordinate system

```c
void reshape(int w, int h) {
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION); /* switch matrix mode */
    glLoadIdentity();

    gluOrtho2D(0.0, w, 0.0, h);

    glMatrixMode(GL_MODELVIEW); /* return to modelview mode */
}
```