Clipping

What happens to objects drawn outside the viewing box?

Let's insert the following square

```
glBegin(GL_POLYGON)
    glVertex3f(120.0, 120.0, 0.0);
    glVertex3f(180.0, 120.0, 0.0);
    glVertex3f(180.0, 180.0, 0.0);
    glVertex3f(120.0, 180.0, 0.0);
glEnd();
```

Where did the new square go? OpenGL clips the scene to the viewing box before rendering

Clipping is a stage in the rendering pipeline

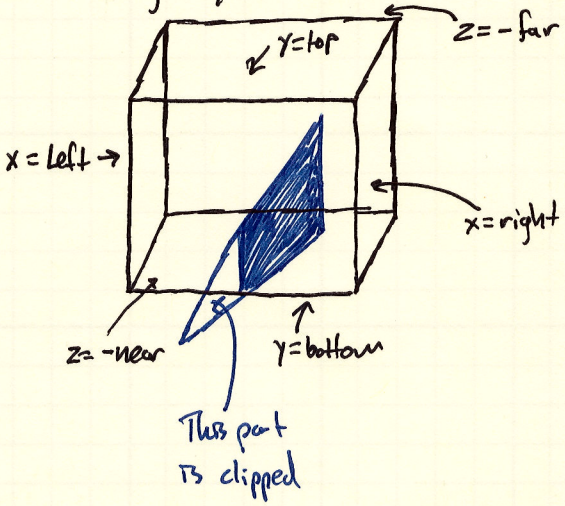
Ex: Redefine the viewing box via glOrtho(..) such that both squares are visible.

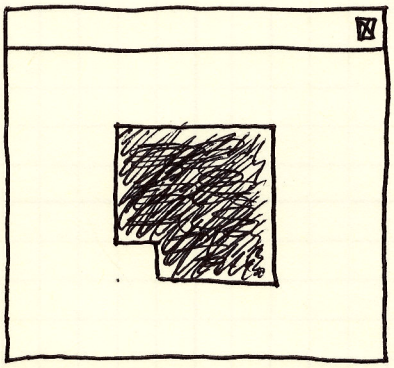
Exp: 1) Replace the original square w/a triangle by commenting out its last vertex

2) Lift the first vertex up the z-axis to 0.5, then further to 1.5, then 2.5 and finally 10.0

The viewing box defines six clipping planes

- Though you can further define additional clipping boundaries





clipping after raising first vertex

OpenGL Geometric Primitives

- Primitives are used to form the most basic to complex objects in OpenGL
- So far we've seen the Polygon (GL_POLYGON) but there are several more
  - GL_POINTS - draws a point at each vertex
  - GL_LINES - draws a disconnected set of straight line segments between vertices, taken two at a time.
  - GL_LINE_STRIP - draws the connected sequence of segments defined by $v_0v_1, v_1v_2, v_2v_3, \ldots, v_{n-2}v_{n-1}$ this is called a polygonal line or polyline
  - GL_LINE_LOOP - same as GL_LINE_STRIP, but includes the additional segment $v_{n-1}v_0$
  - GL_TRIANGLES - draws a sequence of triangles using the vertices 3 at a time. If the number of vertices is not a multiple of 3, the remaing 1 or 2 vertices are not used.
  - GL_TRIANGLE_STRIP - draws a sequence of triangles, as follows:
    $v_0v_1v_2, v_1v_3v_2, v_2v_3v_4, \ldots, v_{n-3}v_{n-2}v_{n-1}$ (if n is odd)
    or
    $v_0v_1v_2, v_1v_3v_2, v_2v_3v_4, \ldots, v_{n-3}v_{n-1}v_{n-2}$ (if n is even)
  - GL_TRIANGLE_FAN - draws a sequence of triangles around the first vertex, as follows

    $v_0v_1v_2, v_0v_2v_3, \ldots, v_0v_{n-2}v_{n-1}$

- NOTE: GL_POLYGON has been dropped from the core profile starting in OpenGL 4.0

- Drawing Orientation
  - The order of vertices determines a polygon's orientation
  - Orientation can be either CW or CCW (as seen by the user)
  - Orientation is used by OpenGL to determine which face, front or back, the viewer sees.

Polygon Modes

- Drawing mode is set by calling glPolygonMode(face, mode)
- face can be: GL_FRONT, GL_BACK or GL_FRONT_AND_BACK
- mode can be: GL_FILL, GL_LINE or GL_POINT
- Whether a polygon is front or back depends on its orientation

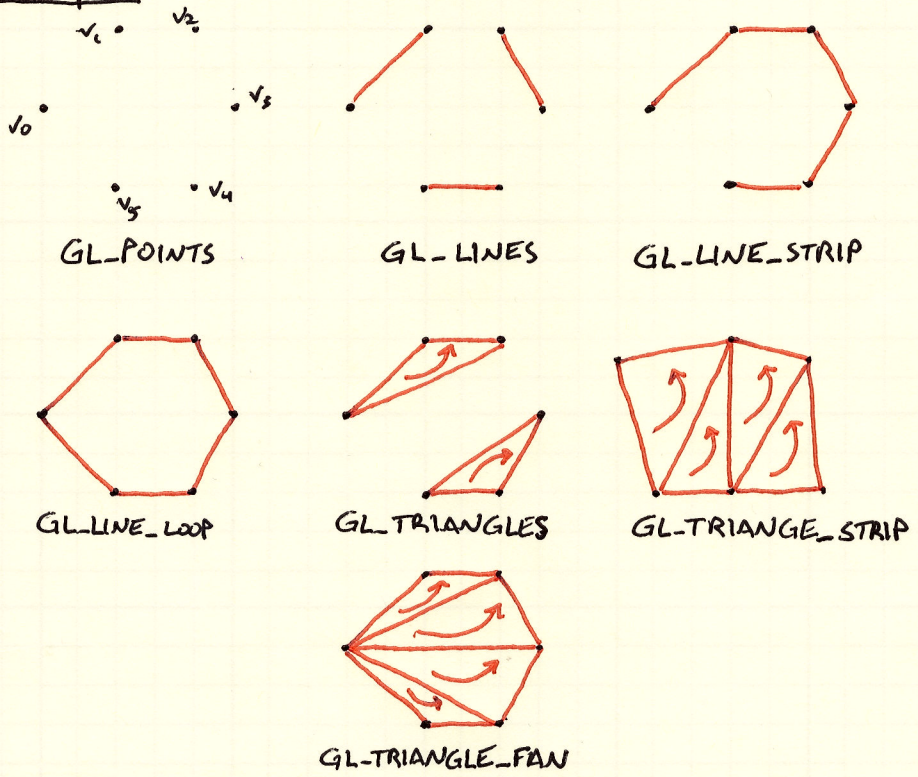
OpenGL Geometric Primatives (cont'd)

- Point size for GL_POINTS can be set by calling `glPointSize(size)` the default value is 1.0
- Line width for GL_LINE* primatives and GL_LINE drawing mode is set by calling `glLineWidth(width)`. The default value is 1.0
- General Definition of a Primitive object

```
glBegin(primitive type);
  glVertex3f(*,*,*); // v0
  glVertex3f(*,*,*); // v1
  ...
  glVertex3f(*,*,*); // vn-1
glEnd();
```

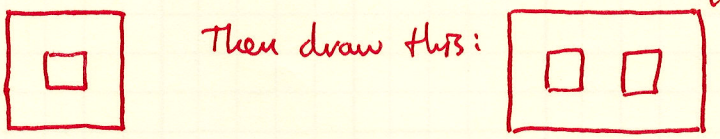
Note: Any configuration commands such as glLineWidth, glPolygonMode, glPointSize must be called outside of glBegin() ... glEnd() and before the block they perform to.

- Examples



Ex: Replace the GL_POLYGON primitive in square.cpp with each of the above. What happens?

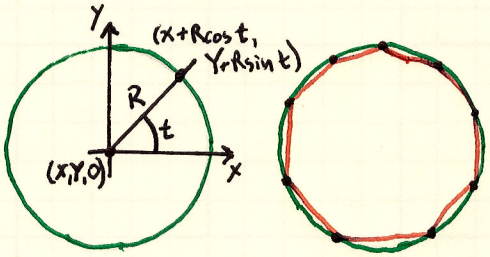
Ex: Using only 2 GL_TRIANGLE_FAN's draw a square annulus.



Approximating Curved Objects

Q: If all of our primitives are composed of points, straight lines and flat segments, then how can we draw curved objects such as circles, ellipses, or beer cans?

- Let us begin by considering a circle (download + review circle.cpp)



- We construct a circle by sampling points using its parametric equations:

$$x = X + R\cos t \qquad 0 \le t \le 2\pi$$
$$y = Y + R\sin t$$

- Implementation:

- The larger the number of samples the more precise of a circle that can be drawn

```
glBegin(GL_LINE_LOOP);
  for (i = 0; i < numVertices; i++)
  {
    glColor3f((float) rand()/(float) RAND_MAX,
              (float) rand()/(float) RAND_MAX,
              (float) rand()/(float) RAND_MAX);
    glVertex3f(X + R * cos(t), Y + R * sin(t), 0.0);
    t += 2 * PI / numVertices;
  }
glEnd();
```

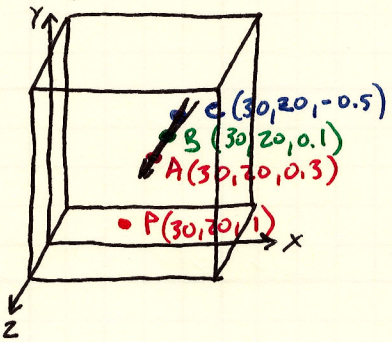
- EXP: Modify this to construct a flat 3-turn spiral

- EXP: Modify this to construct a circular annulus

-

The Depth Buffer

- The depth buffer (or z-buffer) allows OpenGL to eliminate obscured parts of objects prior to rendering
- That is, a point of an object is not drawn if its projection is hidden by another object.
- This process goes by many names
  - Hidden Surface Removal
  - Depth Testing
  - Visibility Determination



- For Orthographic projection this works as follows.
  1. We fix an x,y value as X,Y from the viewing box
  2. $S \leftarrow$ set of all points of objects in viewing box along line $L = (X, Y, 0)$
  3. S is of form $S = \{(X, Y, z)\}$, where z varies depending on points
  4. $Z \leftarrow \arg\max_z S$
  5. $P \leftarrow rgb(X, Y, Z)$ // all points in S project to $P = (X, Y, -near)$, thus

~~the Closest~~

- In the GPU the z-buffer is a block of memory containing z-values (one per pixel)
  - As primitives are processed their z-values are compared against the one currently in the buffer.
  - If the incoming z-value is greater it replaces the current one and the rgb values of the new pixel replace the current values

Note: The GPU uses a scaled value between 0 and 1 where 0 is the near face and 1 is the far face. But, in OpenGL we use World Space, which reverses this relationship

- To enable this add the following code:
  - `glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)` in drawScene to clear the depth buffer
  - `glEnable(GL_DEPTH_TEST)` in drawScene() to enable HSR
    use `glDisable(..)` to disable it
  - `glutInitDisplayMode(.... | GLUT_DEPTH)` in main() will initialize the depth buffer.

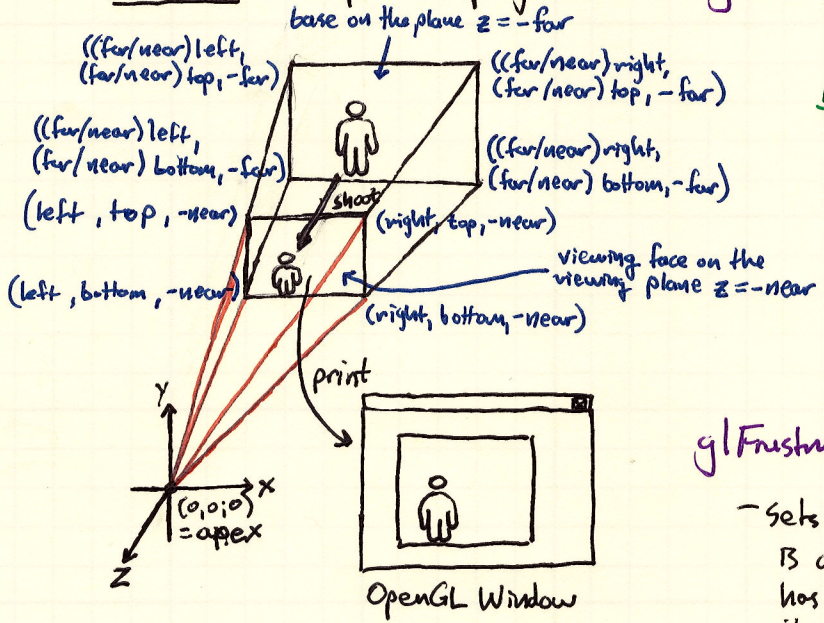
Ex: Make a bullseye with increasing diameter disks of differing colors. Use at least 5 disks

Perspective Projection

- Motivating Example: a helix governed by the following equations

$$x = R\cos t$$
$$y = R\sin t \quad \text{for } -10\pi \le t \le 10\pi$$
$$z = t - 60$$

- Download and review helix.cpp from moodle
- Run helix.cpp, what do you see?
- Issue: We apparently see only a circle
- Perhaps we can fix this by: replacing `glVertex3f(R*cos(t), R*sin(t), t-60.0)` with `glVertex3f(R*cos(t), t, R*sin(t)-60.0)`
- But this still doesn't seem to do much better!
- Why are these problems occurring?
  - Orthographic projection squashes a dimension => Bad for 3D scenes
- Solution: Perspective projection via `glFrustum()`



Note: We can think of the apex as the location of a point camera

`glFrustum(left, right, bottom, top, near, far)`

- Sets up the viewing frustum which is a truncated pyramid whose top has been cut off by a plane parallel to its base.

- Values of `glFrustum()` are typically set
  - to be symmetrical about the z-axis
  - right and top are positive
  - left and bottom are their negatives
  - near and far should both be positive and near < far

Ex: Replace the `glOrtho(...)` call in helix.cpp with:

`glFrustum(-5.0, 5.0, -5.0, 5.0, 5.0, 100.0)`

Perspective vs. Orthographic

- Both use a two-step "print and shoot" rendering process
- Perspective is no longer perpendicular, rather each point is projected along the line joining it to the apex.
- Perspective causes foreshortening because objects further from the apex appear smaller
  - This is called perspective transformation
- The second step is exactly the same for both modes
- Perspective is closer to reality, in that it mimics how images are formed on the retina

~~[illegible]~~

- Both orthographic and perspective projections converge to a center of projection (COP)
  - For perspective, this is a regular point with finite coordinates
  - For orthographic, this is a point infinitely far away and thus the lines are parallel.
- Both of these projections allow OpenGL to implement the synthetic-camera model

Ex: Try the following glFrustum settings in helix.cpp

```
glFrustum(-5.0, 5.0, -5.0, 5.0, 5.0, 120.0)
glFrustum(-5.0, 5.0, -5.0, 5.0, 10.0, 100.0)
glFrustum(-5.0, 5.0, -5.0, 5.0, 2.5, 100.0)
glFrustum(-10.0, 10.0, -10.0, 10.0, 5.0, 100.0)
```

What are the effects?

Def: Take a moment and work out an algorithm for HSR in perspective projection.

OpenGL Program Walkthrough (square.cpp)

- main()
  - glutInit(--); - initializes the FreeGLUT Library
  - glutInitContextVersion(--);
    glutInitContextProfile(GLUT_COMPATIBILITY_PROFILE);

    sets the OpenGL context used by FreeGLUT and allows for backwards compatibility

  - glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA); - single-buffered frame with each pixel having red, blue, green, and alpha values.
  - glutInitWindowSize(500, 500);] set initial window size and position conditions
    glutInitWindowPosition(100, 100);]
  - glutCreateWindow(--); - actually create the OpenGL context and its window with the supplied title
  - glutDisplayFunc(drawScene);]
    glutReshapeFunc(resize);] register callback routines
    glutKeyboardFunc(keyInput);]
  - glewExperimental = GL_TRUE;] initializes GLEW to load extensions
    glewInit();]
  - setup() – invoke initialization function
  - glutMainLoop() - begin event-processing loop, calling registered callbacks as needed

- drawScene() - already covered in-depth

- resize(int w, int h) - handles when the window is resized
  - glViewport(0,0,w,h) – specifies the portion of the window in which rendering will take place.
  - glMatrixMode(GL_PROJECTION);] activate the projection matrix stack
    glLoadIdentity();] - place identity matrix on top of stack
    glOrtho(--);] multiply identity matrix by result of glOrtho command
  - glMatrixMode(GL_MODEL_VIEW);] activate modelview matrix stack
    glLoadIdentity();] place identity matrix on top of this stack

- keyInput(unsigned char key, int x, int y)
  - when a key is pressed if ASCII its value is passed to the char parameter along with the current value of the x,y mouse location

For Next Time

- Review Lectures 02 and 03
- Read Chapter 3 Sections 3.1 – 3.7
- Review Lecture 04 when it becomes available
- Begin working on Assignment 01
- Come to class!

Additional Notes

- Can you create these shapes?