Buffers

OpenGL

Learning Outcomes

- Be aware of other buffers within the OpenGL model, including
    - Front and Back
    - Left and Right
    - Depth
    - Scissors
    - Stencil

Front & Back (Colour) Buffers

- OpenGL does not render (draw) primitives directly on the screen. Instead, rendering is done in a buffer, which is later swapped to the screen.
- These two buffers as the front (the screen) and back color buffers.
- By default, OpenGL commands are rendered into the back buffer, and when you call glutSwapBuffers(or your operating system–specific buffer swap function), the front and back buffers are swapped so that you can see the rendering results.
- Directly rendering into the front buffer is possible - useful for displaying a series of drawing commands so that you can see some object or shape actually being drawn.
- 2 Techniques:
  - Buffer Targets
  - Single Buffer

Buffer Targets

```
void glDrawBuffer(Glenum mode);
```

- Specifying GL_FRONT causes OpenGL to render to the front buffer, and GL_BACK moves rendering back to the back buffer.

- OpenGL implementations can support more than just a single front and back buffer for rendering, such as left and right buffers for stereo rendering, and auxiliary buffers.

Requesting a Single Buffer

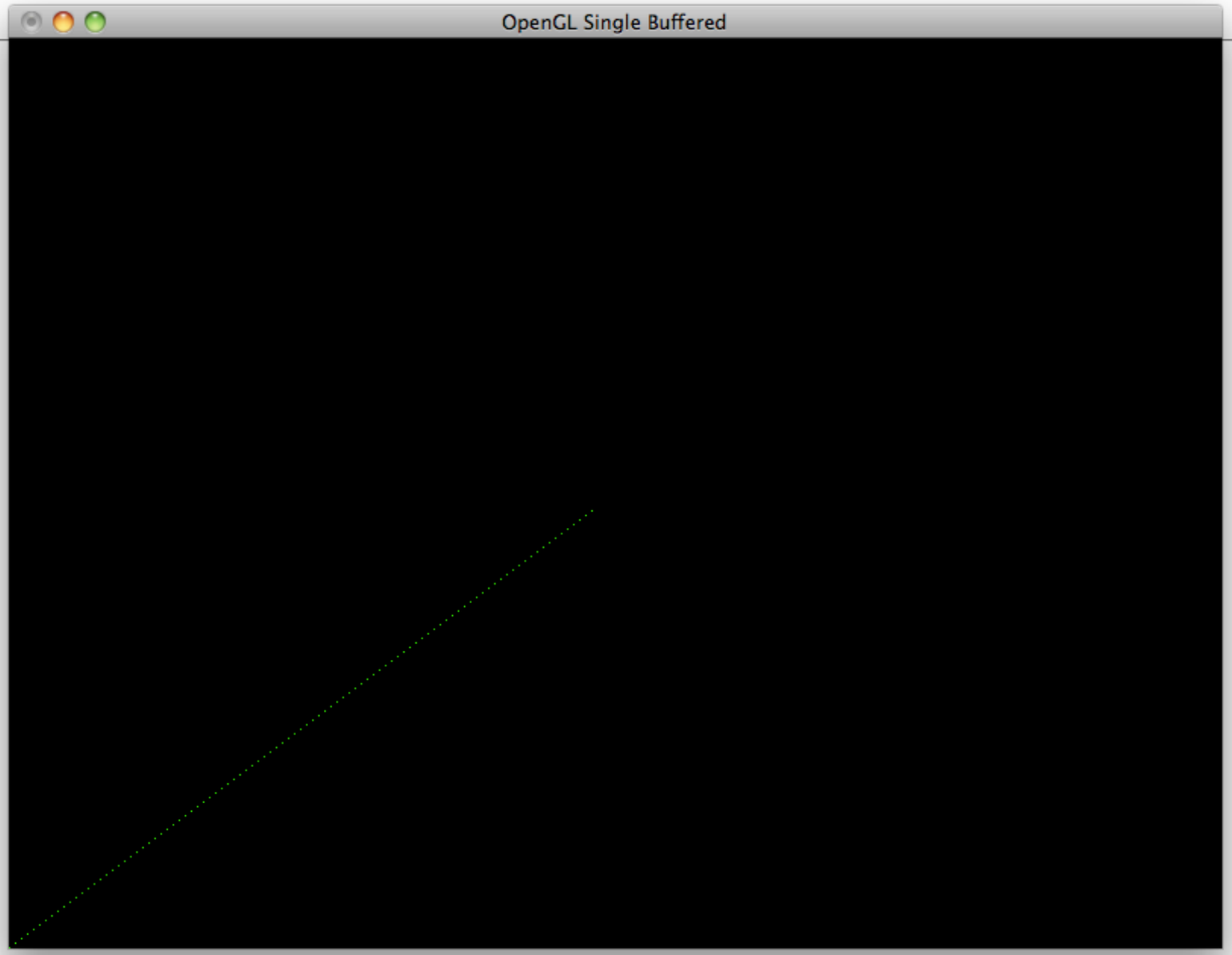
- Do not request double buffered rendering when OpenGL is initialized.

```
glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
```

- OpenGL is initialized differently on each OS plat-form, but with GLUT, we initialize our display mode for RGB color and double-buffered rendering

```
glutInitDisplayMode(GLUT_RGB | GLUT_DEPTH);
```

- Must call either glFlush() or glFinish() whenever you want to see the results actually drawn to screen. A buffer swap implicitly performs a flush of the pipeline and waits for rendering to complete before the swap actually occurs.


OpenGL Single Buffered

```
void timer(int value)
{
  glutTimerFunc(50, timer, 0);
  glutPostRedisplay();
}

int main(int argc, char* argv[])
{
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_RGB);
  glutInitWindowSize(800,600);
  glutCreateWindow("OpenGL Single Buffered");

  glutDisplayFunc(renderScene);
  setupRC();
  timer(50);
  glutMainLoop();

  return 0;
}
```

```
GLfloat x = -100.0;
GLfloat y = -100.0;

void renderScene(void)
{
  if (x == 100.0)
  {
    glClear(GL_COLOR_BUFFER_BIT);
    x = y = -100.0;
  }
  glBegin(GL_POINTS);
    glVertex2f(x,y);
  glEnd();


  x = x + 1.0;
  y = y + 1.0;

  glFlush();
}

void setupRC()
{
  glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
  glColor3f(0.0f, 1.0f, 0.0f);
  glOrtho (-100.0f, 100.0f, -100.0f,
            100.0f, -100.0f, 100.0f);
  glClear(GL_COLOR_BUFFER_BIT);
}
```

The Depth Buffer

- The Depth buffer is filled with depth values instead of color values.
- Available on request:

```
glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
```

- ... and explicitly enabled:

```
glEnable(GL_DEPTH_TEST);
```

- Even when depth testing is not enabled, if a depth buffer is created, OpenGL will write corresponding depth values for all color fragments that go into the color buffer.
- Can be temporarily turn off

```
glDepthMask (GL_FALSE);
```

- disables writes to the depth buffer but does not disable depth testing from being performed using any values that have already been written to the depth buffer.

Scissors

- Can improve rendering performance by updating only the portion of the screen that has changed.

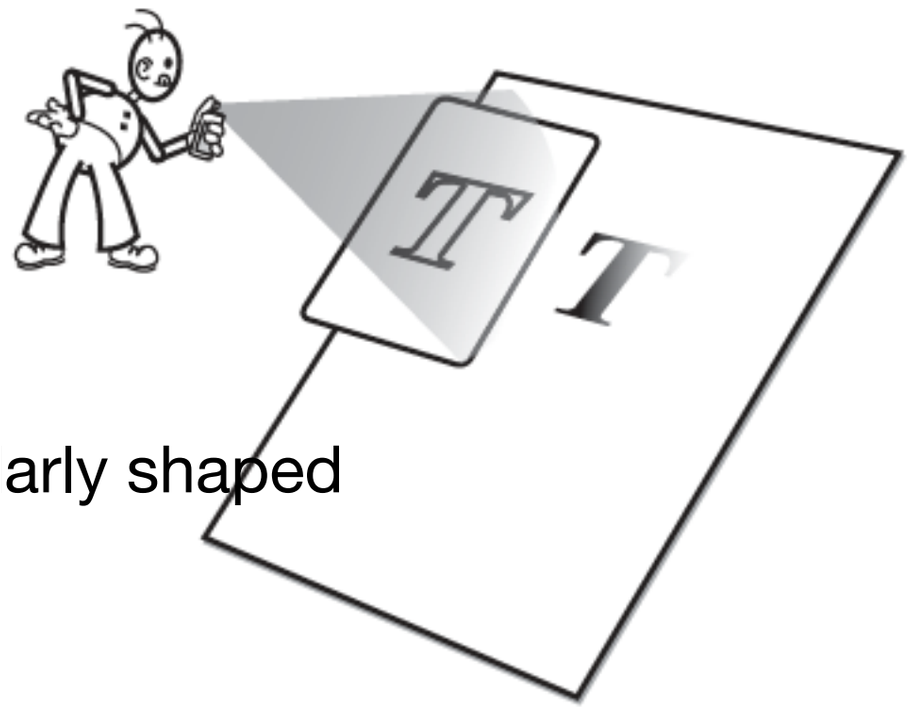
- OpenGL allows you to specify *a scissor rectangle* within your window where rendering can take place. By default, the scissor rectangle is the size of the window, but can be set with:

```
void glScissor(GLint x, GLint y, GLsizei width, GLsizei height);
```

- Scissors can be enabled/disabled:

```
glEnable(GL_SCISSOR_TEST);
glDisable(GL_SCISSOR_TEST);
```

Stencil Buffer



- Similar to Scissors, but used too mask out an irregularly shaped area using a stencil pattern.

- Must request a stencil buffer:

```
glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_STENCIL);
```

- It can also be turned on and off:

```
glEnable(GL_STENCIL_TEST);
```

- With the stencil test enabled, drawing occurs only at locations that pass the stencil test.

```
void glStencilFunc(GLenum func, GLint ref, GLuint mask);
```