First Animation + State Machine Introduction

OpenGL

Learning Outcomes

- Review the glutTimer function & its operation
- Understand the relevance of double buffering in an animation context
- Introduce the OpenGL State Machine

Animation Basics

- Move or rotate images and scenes, creating an animated effect.
    - E.g make the square bounce off the sides of the window
- Create a loop that continually changes your object's coordinates before calling the renderScene function.
- This would cause the square to appear to move around within the window.

glutTimerFunc

```
void glutTimerFunc(unsigned int millis, void (*func)(int value), int value);
```

- The GLUT library enables you to register a callback function that makes it easier to set up a simple animated sequence.

- Takes the name of a function to call and the amount of time to wait before calling the function.

- Tells GLUT to wait `millis` milliseconds before calling the function func. You can pass a user-defined value in the value parameter

```
void timerFunction(int value)
```

- When the time expires, this function is fired only once. To effect a continuous animation, you must reset the timer again in the timer function

renderScene

- Change the hard-coded values for the location of our rectangle to variables and then constantly modify those variables in the timer function.

```
void renderScene(void)
{
  glClear( GL_COLOR_BUFFER_BIT);

  glColor3f(1.0f, 0.0f, 0.0f);

  glRectf(-25.0f, 25.0f, 25.0f, -25.0f);

  glFlush();
}
```

- This causes the rectangle to appear to move across the window
- Need to keep track of the position and size of the rectangle as we go along.
- Also - use doubleBuffering

```
void renderScene(void)
{
  glClear( GL_COLOR_BUFFER_BIT);

  glColor3f(1.0f, 0.0f, 0.0f);

  glRectf(x, y, x + rsize, y - rsize);

  glutSwapBuffers();
}
```

Global Variables

- Current position of Rectangle
- Size of Rectangle
- Amount to move on each refresh
- Current size of window

```
GLfloat x = 0.0f;
GLfloat y = 0.0f;
GLfloat rsize = 25;

GLfloat xstep = 1.0f;
GLfloat ystep = 1.0f;

GLfloat windowWidth;
GLfloat windowHeight;
```

timerFunc()

```
void timerFunction(int value)
{
  if (x > windowWidth - rsize || x < -windowWidth)
    xstep = -xstep;

  if (y > windowHeight || y < -windowHeight + rsize)
    ystep = -ystep;

  x += xstep;
  y += ystep;

  glutPostRedisplay();
  glutTimerFunc(33, timerFunction, 1);
}
```

```
void renderScene(void)
{
  glClear( GL_COLOR_BUFFER_BIT);

  glColor3f(1.0f, 0.0f, 0.0f);

  glRectf(x, y, x + rsize, y - rsize);

  glutSwapBuffers();
}
```

changeSize()

```
void changeSize(int w, int h)
{
  GLfloat aspectRatio;

   glViewport(0, 0, w, h);

  glMatrixMode (GL_PROJECTION);
  glLoadIdentity();

  aspectRatio = (GLfloat) w / (GLfloat) h;
  if (w <= h)
  {
    windowWidth = 100;
    windowHeight = 100 / aspectRatio;
    glOrtho(-100.0, 100.0, -windowHeight, windowHeight, 1.0, -1.0);
  }
  else
  {
    windowWidth = 100 * aspectRatio;
    windowHeight = 100;
    glOrtho(-windowWidth, windowWidth, -100.0, 100.0, 1.0, -1.0);
  }

  glMatrixMode( GL_MODELVIEW);
  glLoadIdentity();
}
```

Double Buffering

- Allows you to execute your drawing code while rendering to an offscreen buffer. Then a swap command places your drawing onscreen instantly.

- Double buffering can serve two purposes.

1. Some complex drawings might take a long time to draw, and you might not want each step of the image composition to be visible. Using double buffering, you can compose an image and display it only after it is complete..

2. Animation: Each frame is drawn in the offscreen buffer and then swapped quickly to the screen when ready.

- No longer are we calling glFlush in renderScene.

- This function is no longer needed because when we perform a buffer swap, we are implicitly performing a flush operation

```
int main(int argc, char* argv[])
{
  glutInit(&argc, argv);

  glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA);

  glutInitWindowSize(800, 600);
  glutCreateWindow("Bounce");
  glutDisplayFunc(renderScene);
  glutReshapeFunc(changeSize);
  glutTimerFunc(33, timerFunction, 1);

  setupRC();

  glutMainLoop();

  return 0;
}
```

```
void renderScene(void)
{
  glClear( GL_COLOR_BUFFER_BIT);

  glColor3f(1.0f, 0.0f, 0.0f);

  glRectf(x, y, x + rsize, y - rsize);

  glutSwapBuffers();
}
```

OpenGL State

- Drawing 3D graphics is a complicated affair. For a given piece of geometry, many things can affect how it is drawn:
    - Is a light shining on it?
    - What are the properties of the light?
    - What are the properties of the material?
    - Which, if any, texture should be applied?
    - Etc...
- This collection of variables is called the state of the pipeline

State Machine

- In OpenGL terms a state machine is an abstract model of a collection of state variables, all of which can have various values, be turned on or off, and so on.

- It is not practical to specify all the state variables whenever we try to draw something in OpenGL.

- Instead, OpenGL employs a state model, or state machine, to keep track of all the OpenGL state variables.

- When a state value is set, it remains set until some other function changes it. Many states are simply on or off.

Setting States - e.g. Lighting

- Lighting is either turned on or turned off.

- Geometry drawn without lighting is drawn without any lighting calculations being applied to the colors set for the geometry.

- Any geometry drawn after lighting is turned back on is then drawn with the lighting calculations applied.

```
glEnable(GL_LIGHTING);

//...

glDisable(GL_LIGHTING);
```

- Not all state variables, however, are simply on or off.
- Many of the OpenGL functions set up values that “stick” until changed.
- You can query what these values are at any time

```
void glGetBooleanv(GLenum pname, GLboolean *params);
void glGetDoublev(GLenum pname, GLdouble *params);
void glGetFloatv(GLenum pname, GLfloat *params);
void glGetIntegerv(GLenum pname, GLint *params);
```

Saving and Restoring States

- OpenGL also has a convenient mechanism for saving a whole range of state values and restoring them later.

- The stack is a convenient data structure that allows values to be pushed on the stack (saved) and popped off the stack later to retrieve them.

- Standard Last In First Out (LIFO) data structure.

```
void glPushAttrib(GLbitfield mask);

void glPopAttrib(GLbitfield mask);
```

Bit fields

- Note that the argument to these functions is a bit field.

- This means that you use a bitwise mask, which allows you to perform a bitwise OR (in C using the | operator) of multiple state values with a single function call.

- For example, y save the lighting and texturing state with a single call:

```
glPushAttrib(GL_TEXTURE_BIT | GL_LIGHTING_BIT);
```