# Matrix Stacks

OpenGL

# Learning Outcomes

- Explore simple animations using glRotate and glTranslate

- Understand the role of glPushMatrix and glPopMatrix in this context

- Explore simple extension to the Color and Vector3 classes

# Pushing & Popping Matrix Stacks

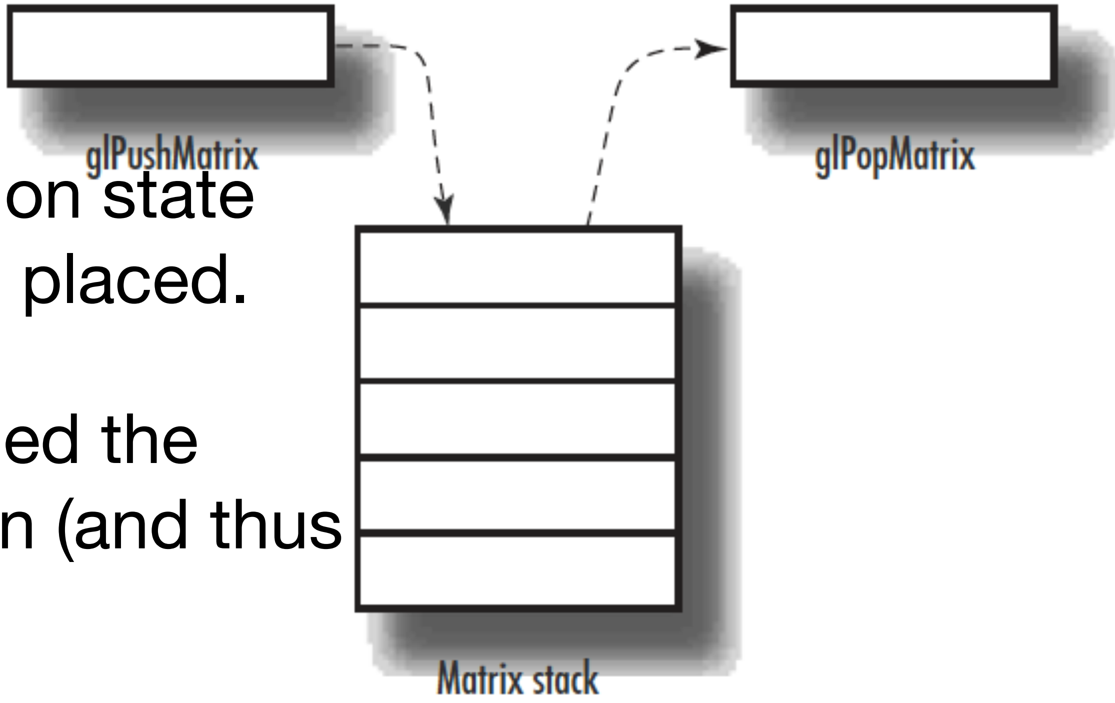
- Resetting the modelview matrix to identity before placing every object is not always desirable.

- Often, you want to save the current transformation state and then restore it aftersome objects have been placed.

- Most common when you have initially transformed the modelview matrix as your viewing transformation (and thus are no longer located at the origin).

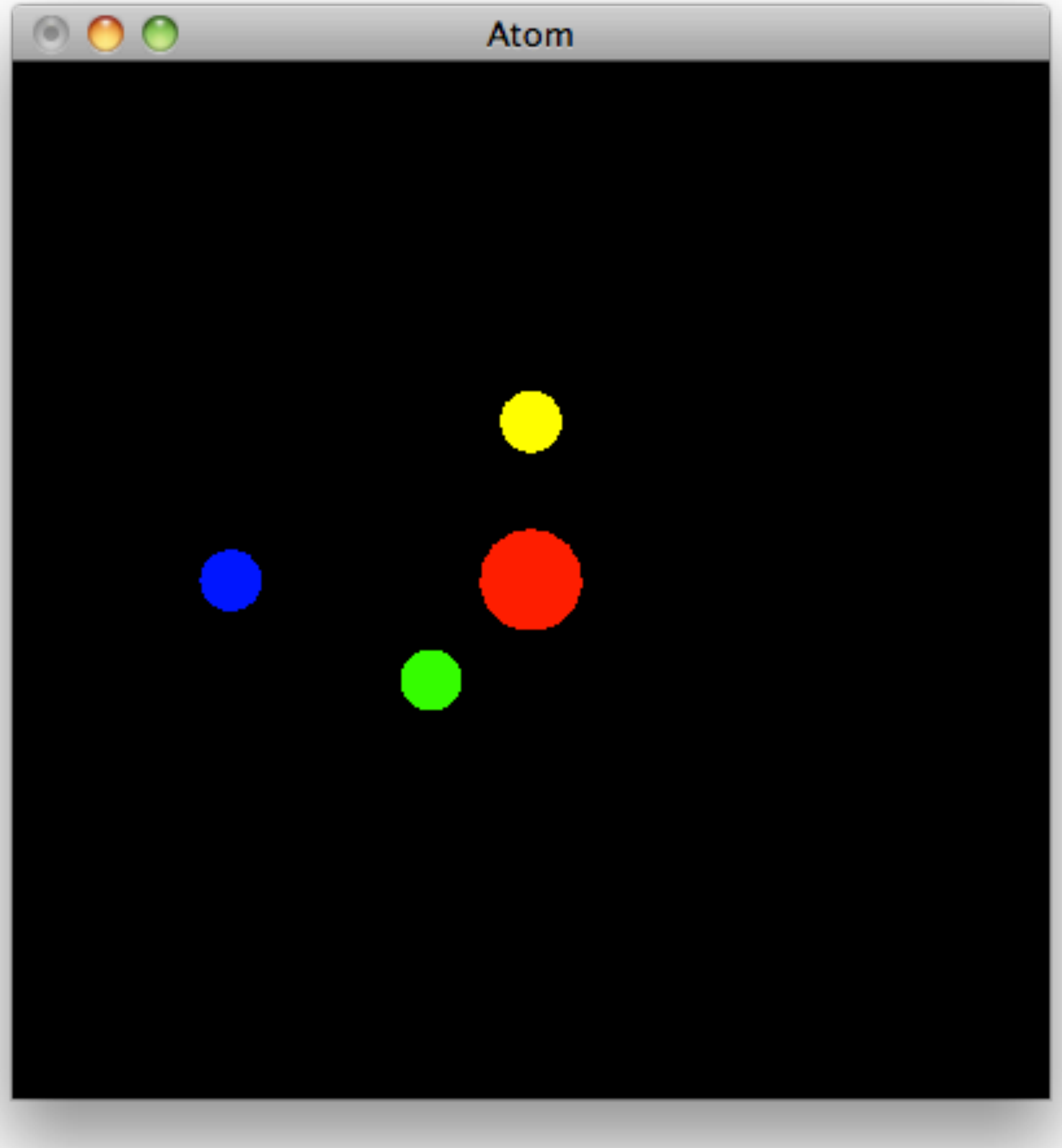
- To facilitate this procedure, OpenGL maintains a matrix stack for both the modelview an projection matrices.

- Push the current matrix onto the stack with glPushMatrix to save it and then make your changes to the current matrix. Popping the matrix off the stack with glPopMatrix then restores it



# A Nuclear Example

- An animated model of an atom with a single sphere at the center to represent the nucleus and three electrons in orbit about the atom.
- Use an orthographic projection.
- Use a timer callback mechanism to redraw the scene about 20 times per second.
- Each time the renderScene function is called, the angle of revolution about the nucleus is incremented.
- Each electron lies in a different plane



# Color Class

```
struct Color
{
  float R;
  float G;
  float B;
  float A;

  static Color White;
  static Color Yellow;
  static Color Red;
  static Color Magenta;
  static Color Cyan;
  static Color Green;
  static Color Black;
  static Color Blue;


  Color();
  Color(float r, float g, float b, float a=1.0f);
  Color(int r, int g, int b, int a=255);

  void render();
  void renderClear();
};
```

```
Color Color::Black   (0,     0,    0);
Color Color::Blue    (0,     0,  255);
Color Color::Green   (0,   255,    0);
Color Color::Cyan    (0,   255,  255);
Color Color::Red     (255,   0,    0);
Color Color::Magenta (255,   0,  255);
Color Color::Yellow  (255, 255,    0);
Color Color::White   (255, 255,  255);

Color::Color()
{
  R = G = B = A = 1.0f;
}
Color::Color(float r, float g, float b, float a)
{
  R = r;
  G = g;
  B = b;
  A = a;
}
Color::Color(int r, int g, int b, int a)
{
  R = (float) r / 255.0f;
  G = (float) g / 255.0f;
  B = (float) b / 255.0f;
  A = (float) a / 255.0f;
}

void Color::render()
{
  glColor4f(R,G,B,A);
}

void Color::renderClear()
{
  glClearColor(R,G,B, 1.0f);
}
```

# Vector Class

```
struct Vector3
{
  float X;
  float Y;
  float Z;

  static Vector3 UnitX;
  static Vector3 UnitY;
  static Vector3 UnitZ;

  Vector3(float x, float y, float z);
  Vector3(float value);
  Vector3();
  Vector3(std::istream& is);

  void translate();
  void rotate (float angle);

  void render();
};
```

```
Vector3 Vector3::UnitX(1.0f, 0.0f, 0.0f);
Vector3 Vector3::UnitY(0.0f, 1.0f, 0.0f);
Vector3 Vector3::UnitZ(0.0f, 0.0f, 1.0f);

Vector3::Vector3(float x, float y, float z)
: X(x)
, Y(y)
, Z(z)
{}
Vector3::Vector3(float value)
: X(value)
, Y(value)
, Z(value)
{}
Vector3::Vector3()
: X(0)
, Y(0)
, Z(0)
{}
Vector3::Vector3(istream &is)
{
  skipComment(is);
  is >> X >> Y >> Z;
}

void Vector3::render()
{
  glVertex3f(X, Y, Z);
}

void Vector3::translate()
{
  glTranslatef(X,Y,Z);
}

void Vector3::rotate (float angle)
{
  glRotatef(angle, X,Y,Z);
}
```

# glutSolidSphere & glutWireSphere

Render a solid or wireframe spheres.

**Usage**

void glutSolidSphere(GLdouble radius, GLint slices, GLint stacks);

void glutWireSphere(GLdouble radius, GLint slices, GLint stacks);

- radius The radius of the sphere.

- slices The number of subdivisions around the Z axis (similar to lines of longitude).

- stacks The number of subdivisions along the Z axis (similar to lines of latitude).

**Description**

Renders a sphere centered at the modeling coordinates origin of the specified radius. The sphere is subdivided around the Z axis into slices and along the Z axis into stacks.

# Atom Simulation - Main + Timer

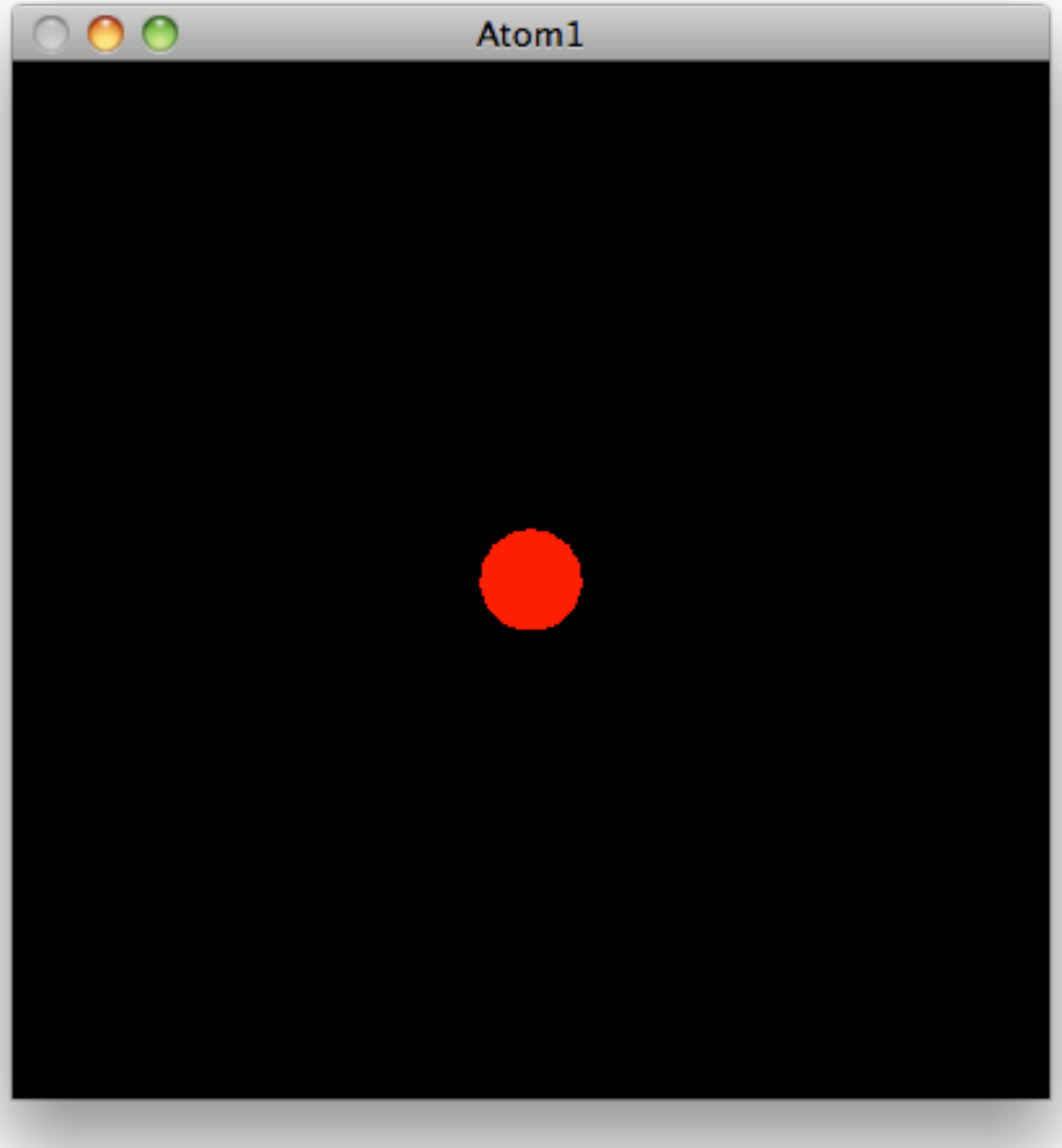
```
void timerFunc(int value)
{
  glutPostRedisplay();
  glutTimerFunc(50, timerFunc, 1);
}

int main(int argc, char* argv[])
{
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);

  glutInitWindowSize(400,400);
  glutCreateWindow("Atom1");
  glutDisplayFunc(renderScene);
  setupRC();
  timerFunc(50);
  glutMainLoop();

  return 0;
}
```

# setupRC

```
void setupRC()
{
  Color::Black.renderClear();
  glEnable(GL_DEPTH_TEST);
  glFrontFace(GL_CCW);
  glEnable(GL_CULL_FACE);

  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  glOrtho (-100.0f, 100.0f, -100.0f, 100.0f, -100.0f, 100.0f);

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

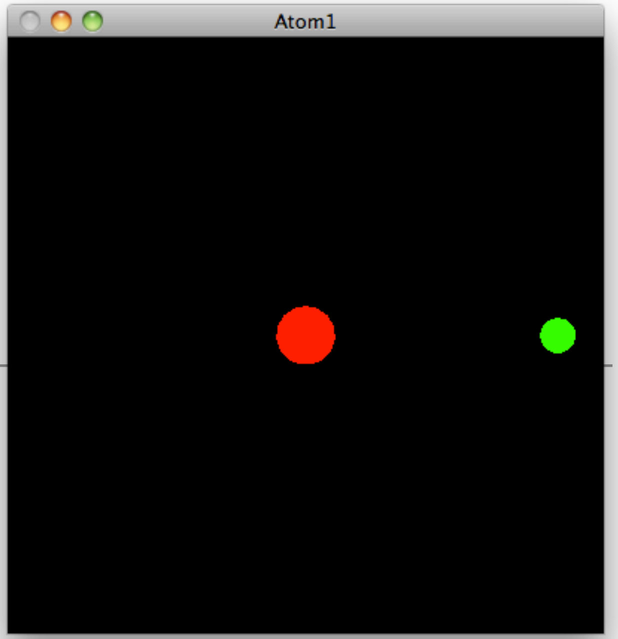
# renderScene + renderNucleus

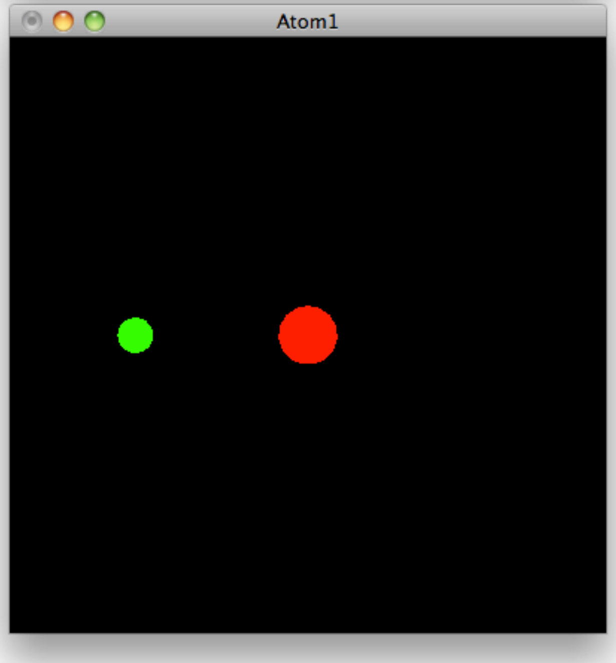
```
void renderNucleus()
{
  Color::Red.render();
  glutSolidSphere(10.0f, 15, 15);
}

void renderScene(void)
{
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

  renderNucleus();
  //...
  //...
  glutSwapBuffers();
}
```



# Single Electron Rotating around the origin

```
void renderScene(void)
{
  static int angle = 0;

  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

  renderNucleus();

  glPushMatrix();
    Color::Green.render();
    Vector3::UnitY.rotate(angle);
    Vector3(90,0,0).translate();
    glutSolidSphere(6.0f, 15, 15);
  glPopMatrix();

  angle = (angle + 10) % 360;

  glutSwapBuffers();
}
```

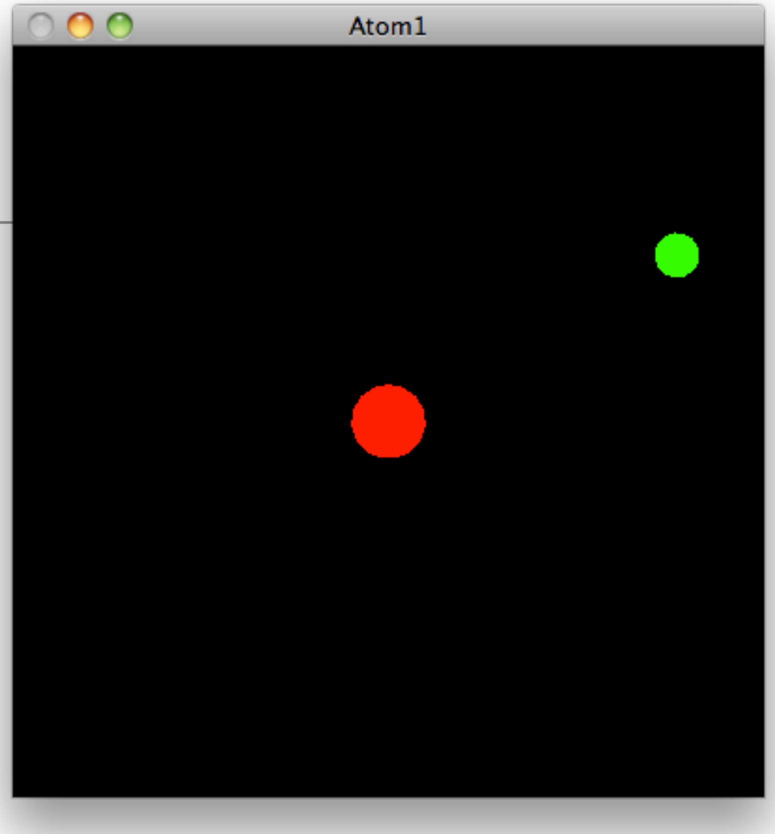


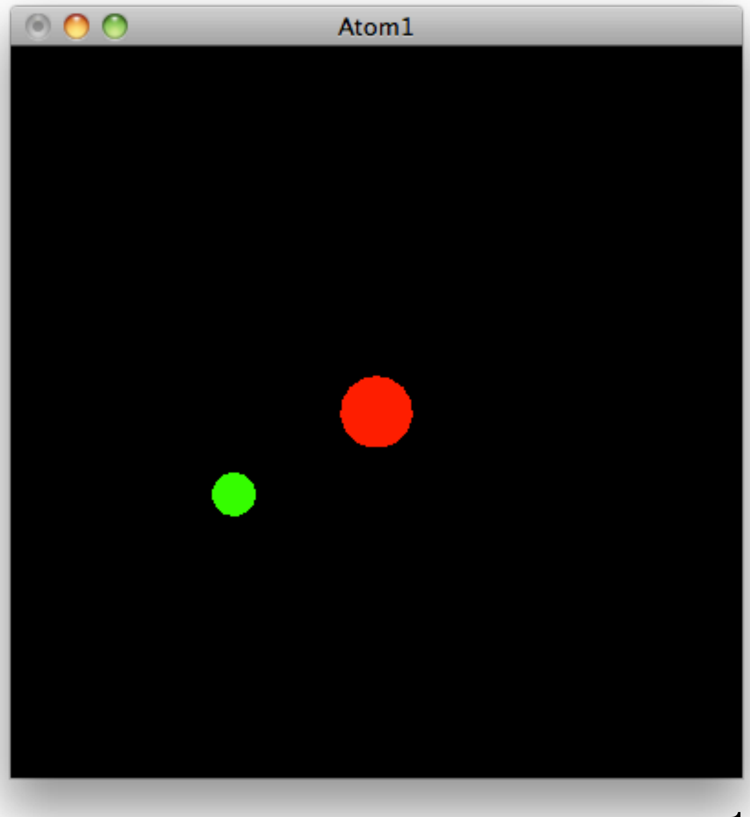


- Green Electron
- 90 units from nucleus

# Skewing the Rotation around the Z axis

```
glPushMatrix();
  Color::Green.render();
  Vector3::UnitZ.rotate(30);
  Vector3::UnitY.rotate(angle);
  Vector3(90,0,0).translate();
  glutSolidSphere(6.0f, 15, 15);
glPopMatrix();
```





# Abstracting to renderElectron

- Simple procedural abstraction

```
void renderElectron(Color color,  float orbitRadius, float orbitAngle, float zSkew)
{
  color.render();
  glPushMatrix();
    Vector3::UnitZ.rotate(zSkew);
    Vector3::UnitY.rotate(orbitAngle);
    Vector3(orbitRadius,0,0).translate();
    glutSolidSphere(6.0f, 15, 15);
  glPopMatrix();
}
```

```
  renderElectron(Color::Green,  90,  angle, 40);
```

# Multiple Electrons, Different Orbits

```
void renderScene(void)
{
  static int angle = 0;
  static int angle2 = 0;

  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

  renderNucleus();

  renderElectron(Color::Green,  90,  angle, 40);
  renderElectron(Color::Cyan,   40,  angle2, 20);

  angle = (angle + 10) % 360;
  angle2 = (angle2 + 5) % 360;

  glutSwapBuffers();
}
```