Complete PyOpenGL Documentation & Tutorial

Table of Contents

---

1. Introduction to PyOpenGL {#introduction}

PyOpenGL is the Python binding to OpenGL, the standard cross-platform API for rendering 2D and 3D graphics. It allows you to create sophisticated graphics applications using Python.

Key Features:

- Cross-platform (Windows, Linux, macOS)
- Hardware-accelerated graphics
- 2D and 3D rendering capabilities
- Integration with GLUT, GLFW, and other windowing systems

---

2. Installation & Setup {#installation}

Installing PyOpenGL

```
pip install PyOpenGL PyOpenGL_accelerate
pip install glfw  # For window management (recommended)
# OR
pip install PyOpenGL-accelerate freeglut  # Alternative with GLUT
```

Basic Window Setup with GLUT

```
from OpenGL.GL import *
from OpenGL.GLUT import *
from OpenGL.GLU import *

def init():
    glClearColor(0.0, 0.0, 0.0, 1.0)  # Black background
    glMatrixMode(GL_PROJECTION)
    glLoadIdentity()
    gluOrtho2D(-1.0, 1.0, -1.0, 1.0)  # 2D orthographic projection

def display():
    glClear(GL_COLOR_BUFFER_BIT)
    # Your drawing code here
    glFlush()

def main():
    glutInit()
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB)
    glutInitWindowSize(500, 500)
    glutInitWindowPosition(100, 100)
    glutCreateWindow(b"My First PyOpenGL Window")
    init()
    glutDisplayFunc(display)
    glutMainLoop()

if __name__ == "__main__":
    main()
```

---

3. OpenGL Basics {#opengl-basics}

The OpenGL Pipeline

1. **Vertex Specification**: Define vertices (points in space)
2. **Vertex Processing**: Transform vertices (model, view, projection)
3. **Rasterization**: Convert geometric data to fragments (pixels)
4. **Fragment Processing**: Compute color for each pixel
5. **Output**: Display on screen

Coordinate Systems

- **Object Coordinates**: Local to the object
- **World Coordinates**: Position in the scene
- **Eye/Camera Coordinates**: Relative to camera
- **Clip Coordinates**: After projection
- **Screen Coordinates**: Final 2D screen position

---

4. Core Concepts {#core-concepts}

4.1 Buffers

Color Buffer: Stores pixel colors

```
glClear(GL_COLOR_BUFFER_BIT)  # Clear color buffer
glClearColor(r, g, b, a)      # Set clear color
```

Depth Buffer: For 3D depth testing

```
glEnable(GL_DEPTH_TEST)
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
```

4.2 Matrix Modes

```
glMatrixMode(GL_PROJECTION)   # Projection matrix (how 3D→2D)
glMatrixMode(GL_MODELVIEW)    # Model-view matrix (camera & objects)
```

4.3 Projection Types

Orthographic (2D/Isometric)

```
glOrtho(left, right, bottom, top, near, far)
gluOrtho2D(left, right, bottom, top)  # 2D shortcut
```

Perspective (3D Realistic)

```
gluPerspective(fovy, aspect, near, far)
# fovy: field of view (degrees)
# aspect: width/height ratio
# near, far: clipping planes
```

---

5. Drawing Primitives {#drawing-primitives}

Basic Drawing Pattern

```
glBegin(GL_PRIMITIVE_TYPE)
    glVertex2f(x, y)      # 2D vertex
    glVertex3f(x, y, z)   # 3D vertex
glEnd()
```

Primitive Types

GL_POINTS

```
glPointSize(5.0)
glBegin(GL_POINTS)
    glVertex2f(0.0, 0.0)
    glVertex2f(0.5, 0.5)
glEnd()
```

GL_LINES

```
glBegin(GL_LINES)
    glVertex2f(-0.5, 0.0)  # Line 1 start
    glVertex2f(0.5, 0.0)   # Line 1 end
    glVertex2f(0.0, -0.5)  # Line 2 start
    glVertex2f(0.0, 0.5)   # Line 2 end
glEnd()
```

GL_LINE_STRIP

```
glBegin(GL_LINE_STRIP)  # Connected lines
    glVertex2f(-0.5, -0.5)
    glVertex2f(0.0, 0.5)
    glVertex2f(0.5, -0.5)
glEnd()
```

GL_LINE_LOOP

```
glBegin(GL_LINE_LOOP)  # Closed line strip
    glVertex2f(-0.5, -0.5)
    glVertex2f(0.5, -0.5)
    glVertex2f(0.5, 0.5)
    glVertex2f(-0.5, 0.5)
glEnd()
```

GL_TRIANGLES

```
glBegin(GL_TRIANGLES)
    glVertex2f(-0.5, -0.5)
    glVertex2f(0.5, -0.5)
    glVertex2f(0.0, 0.5)
glEnd()
```

GL_TRIANGLE_FAN

```
glBegin(GL_TRIANGLE_FAN)  # Useful for circles
    glVertex2f(0.0, 0.0)  # Center point
    glVertex2f(0.5, 0.0)
    glVertex2f(0.4, 0.3)
    glVertex2f(0.0, 0.5)
glEnd()
```

GL_QUADS

```
glBegin(GL_QUADS)
    glVertex2f(-0.5, -0.5)
    glVertex2f(0.5, -0.5)
    glVertex2f(0.5, 0.5)
    glVertex2f(-0.5, 0.5)
glEnd()
```

GL_POLYGON

```
glBegin(GL_POLYGON)  # For any convex polygon
    glVertex2f(-0.5, -0.5)
    glVertex2f(0.5, -0.5)
    glVertex2f(0.7, 0.0)
    glVertex2f(0.5, 0.5)
    glVertex2f(-0.5, 0.5)
    glVertex2f(-0.7, 0.0)
glEnd()
```

---

6. Transformations {#transformations}

Translation (Moving)

```
glTranslatef(x, y, z)  # Move by (x, y, z)
```

Example:

```
glLoadIdentity()
glTranslatef(0.5, 0.0, 0.0)  # Move right
# Draw object here
```

Rotation

```
glRotatef(angle, x, y, z)  # Rotate 'angle' degrees around axis (x,y,z)
```

Common Rotations:

```
glRotatef(45, 0, 0, 1)   # Rotate 45° around Z-axis (2D rotation)
glRotatef(90, 1, 0, 0)   # Rotate 90° around X-axis
glRotatef(30, 0, 1, 0)   # Rotate 30° around Y-axis
```

Scaling

```
glScalef(sx, sy, sz)  # Scale by factors sx, sy, sz
```

Example:

```
glScalef(2.0, 1.0, 1.0)  # Double width, same height
```

Matrix Stack Operations

```
glPushMatrix()  # Save current matrix
# Apply transformations
# Draw objects
glPopMatrix()   # Restore saved matrix
```

Example: Drawing Multiple Transformed Objects

```
def display():
    glClear(GL_COLOR_BUFFER_BIT)

    # First square
    glPushMatrix()
    glTranslatef(-0.5, 0.0, 0.0)
    glColor3f(1.0, 0.0, 0.0)  # Red
    draw_square()
    glPopMatrix()

    # Second square
    glPushMatrix()
    glTranslatef(0.5, 0.0, 0.0)
    glRotatef(45, 0, 0, 1)
    glColor3f(0.0, 0.0, 1.0)  # Blue
    draw_square()
    glPopMatrix()

    glFlush()
```

---

7. Colors & Lighting {#colors-lighting}

Setting Colors

```
glColor3f(r, g, b)        # RGB values 0.0-1.0
glColor4f(r, g, b, a)     # RGBA with alpha (transparency)
```

Per-Vertex Colors:

```
glBegin(GL_TRIANGLES)
    glColor3f(1.0, 0.0, 0.0)  # Red
    glVertex2f(-0.5, -0.5)
    glColor3f(0.0, 1.0, 0.0)  # Green
    glVertex2f(0.5, -0.5)
    glColor3f(0.0, 0.0, 1.0)  # Blue
    glVertex2f(0.0, 0.5)
glEnd()  # Creates gradient triangle
```

Basic Lighting

```
def setup_lighting():
    glEnable(GL_LIGHTING)
    glEnable(GL_LIGHT0)

    # Light position
    light_pos = [1.0, 1.0, 1.0, 0.0]
    glLightfv(GL_LIGHT0, GL_POSITION, light_pos)

    # Light colors
    ambient = [0.2, 0.2, 0.2, 1.0]
    diffuse = [1.0, 1.0, 1.0, 1.0]
    specular = [1.0, 1.0, 1.0, 1.0]

    glLightfv(GL_LIGHT0, GL_AMBIENT, ambient)
    glLightfv(GL_LIGHT0, GL_DIFFUSE, diffuse)
    glLightfv(GL_LIGHT0, GL_SPECULAR, specular)
```

Material Properties

```
def set_material():
    mat_ambient = [0.2, 0.2, 0.2, 1.0]
    mat_diffuse = [0.8, 0.0, 0.0, 1.0]  # Red
    mat_specular = [1.0, 1.0, 1.0, 1.0]
    mat_shininess = [50.0]

    glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient)
    glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse)
    glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular)
    glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess)
```

---

8. Textures {#textures}

Loading and Applying Textures

```
from PIL import Image

def load_texture(filename):
    img = Image.open(filename)
    img_data = img.tobytes("raw", "RGB", 0, -1)

    texture_id = glGenTextures(1)
    glBindTexture(GL_TEXTURE_2D, texture_id)

    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR)
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR)

    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, img.width, img.height,
                 0, GL_RGB, GL_UNSIGNED_BYTE, img_data)

    return texture_id

def draw_textured_quad(texture_id):
    glEnable(GL_TEXTURE_2D)
    glBindTexture(GL_TEXTURE_2D, texture_id)

    glBegin(GL_QUADS)
        glTexCoord2f(0.0, 0.0); glVertex2f(-0.5, -0.5)
        glTexCoord2f(1.0, 0.0); glVertex2f(0.5, -0.5)
        glTexCoord2f(1.0, 1.0); glVertex2f(0.5, 0.5)
        glTexCoord2f(0.0, 1.0); glVertex2f(-0.5, 0.5)
    glEnd()

    glDisable(GL_TEXTURE_2D)
```

---

9. 3D Graphics {#3d-graphics}

Setting Up 3D Viewport

```
def init_3d():
    glClearColor(0.0, 0.0, 0.0, 1.0)
    glEnable(GL_DEPTH_TEST)

    glMatrixMode(GL_PROJECTION)
    glLoadIdentity()
    gluPerspective(45, 1.0, 0.1, 50.0)  # fov, aspect, near, far

    glMatrixMode(GL_MODELVIEW)
    glLoadIdentity()
    gluLookAt(3, 3, 3,    # Camera position
              0, 0, 0,    # Look at point
              0, 1, 0)    # Up vector
```

Drawing a 3D Cube

```
def draw_cube():
    glBegin(GL_QUADS)

    # Front face
    glColor3f(1.0, 0.0, 0.0)  # Red
    glVertex3f(-1, -1, 1)
    glVertex3f(1, -1, 1)
    glVertex3f(1, 1, 1)
    glVertex3f(-1, 1, 1)

    # Back face
    glColor3f(0.0, 1.0, 0.0)  # Green
    glVertex3f(-1, -1, -1)
    glVertex3f(-1, 1, -1)
    glVertex3f(1, 1, -1)
    glVertex3f(1, -1, -1)

    # Top face
    glColor3f(0.0, 0.0, 1.0)  # Blue
    glVertex3f(-1, 1, -1)
    glVertex3f(-1, 1, 1)
    glVertex3f(1, 1, 1)
    glVertex3f(1, 1, -1)

    # Bottom face
    glColor3f(1.0, 1.0, 0.0)  # Yellow
    glVertex3f(-1, -1, -1)
    glVertex3f(1, -1, -1)
    glVertex3f(1, -1, 1)
    glVertex3f(-1, -1, 1)

    # Right face
    glColor3f(1.0, 0.0, 1.0)  # Magenta
    glVertex3f(1, -1, -1)
    glVertex3f(1, 1, -1)
    glVertex3f(1, 1, 1)
    glVertex3f(1, -1, 1)

    # Left face
    glColor3f(0.0, 1.0, 1.0)  # Cyan
    glVertex3f(-1, -1, -1)
    glVertex3f(-1, -1, 1)
    glVertex3f(-1, 1, 1)
    glVertex3f(-1, 1, -1)

    glEnd()
```

---

10. Common Projects {#common-projects}

Project 1: Bouncing Ball Animation

```
import math

ball_x = 0.0
ball_y = 0.0
velocity_x = 0.02
velocity_y = 0.03

def draw_circle(x, y, radius, segments=30):
    glBegin(GL_TRIANGLE_FAN)
    glVertex2f(x, y)
    for i in range(segments + 1):
        angle = 2.0 * math.pi * i / segments
        dx = radius * math.cos(angle)
        dy = radius * math.sin(angle)
        glVertex2f(x + dx, y + dy)
    glEnd()

def update(value):
    global ball_x, ball_y, velocity_x, velocity_y

    ball_x += velocity_x
    ball_y += velocity_y

    # Bounce off walls
    if ball_x > 0.9 or ball_x < -0.9:
        velocity_x *= -1
    if ball_y > 0.9 or ball_y < -0.9:
        velocity_y *= -1

    glutPostRedisplay()
    glutTimerFunc(16, update, 0)  # ~60 FPS

def display():
    glClear(GL_COLOR_BUFFER_BIT)
    glColor3f(1.0, 0.0, 0.0)
    draw_circle(ball_x, ball_y, 0.1)
    glFlush()
```

Project 2: Simple House Drawing

```
def draw_house():
    glClear(GL_COLOR_BUFFER_BIT)

    # House base
    glColor3f(0.8, 0.5, 0.2)
    glBegin(GL_QUADS)
    glVertex2f(-0.5, -0.5)
    glVertex2f(0.5, -0.5)
    glVertex2f(0.5, 0.2)
    glVertex2f(-0.5, 0.2)
    glEnd()

    # Roof
    glColor3f(0.6, 0.1, 0.1)
    glBegin(GL_TRIANGLES)
    glVertex2f(-0.6, 0.2)
    glVertex2f(0.6, 0.2)
    glVertex2f(0.0, 0.7)
    glEnd()

    # Door
    glColor3f(0.4, 0.2, 0.0)
    glBegin(GL_QUADS)
    glVertex2f(-0.1, -0.5)
    glVertex2f(0.1, -0.5)
    glVertex2f(0.1, 0.0)
    glVertex2f(-0.1, 0.0)
    glEnd()

    # Window
    glColor3f(0.5, 0.8, 1.0)
    glBegin(GL_QUADS)
    glVertex2f(0.25, -0.1)
    glVertex2f(0.45, -0.1)
    glVertex2f(0.45, 0.1)
    glVertex2f(0.25, 0.1)
    glEnd()

    glFlush()
```

Project 3: Rotating 3D Pyramid

```
angle = 0.0

def draw_pyramid():
    glBegin(GL_TRIANGLES)

    # Front
    glColor3f(1.0, 0.0, 0.0)
    glVertex3f(0.0, 1.0, 0.0)
    glVertex3f(-1.0, -1.0, 1.0)
    glVertex3f(1.0, -1.0, 1.0)

    # Right
    glColor3f(0.0, 1.0, 0.0)
    glVertex3f(0.0, 1.0, 0.0)
    glVertex3f(1.0, -1.0, 1.0)
    glVertex3f(1.0, -1.0, -1.0)

    # Back
    glColor3f(0.0, 0.0, 1.0)
    glVertex3f(0.0, 1.0, 0.0)
    glVertex3f(1.0, -1.0, -1.0)
    glVertex3f(-1.0, -1.0, -1.0)

    # Left
    glColor3f(1.0, 1.0, 0.0)
    glVertex3f(0.0, 1.0, 0.0)
    glVertex3f(-1.0, -1.0, -1.0)
    glVertex3f(-1.0, -1.0, 1.0)

    glEnd()

    # Base
    glColor3f(0.5, 0.5, 0.5)
    glBegin(GL_QUADS)
    glVertex3f(-1.0, -1.0, 1.0)
    glVertex3f(1.0, -1.0, 1.0)
    glVertex3f(1.0, -1.0, -1.0)
    glVertex3f(-1.0, -1.0, -1.0)
    glEnd()

def display():
    global angle
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
    glLoadIdentity()

    gluLookAt(5, 3, 5, 0, 0, 0, 0, 1, 0)
    glRotatef(angle, 0, 1, 0)

    draw_pyramid()
    glutSwapBuffers()

def update(value):
    global angle
    angle += 1.0
    if angle > 360:
        angle -= 360
    glutPostRedisplay()
    glutTimerFunc(16, update, 0)
```

Quick Reference: Common Functions

Function	Purpose
`glClear(GL_COLOR_BUFFER_BIT)`	Clear screen
`glColor3f(r, g, b)`	Set drawing color
`glVertex2f(x, y)`	Define 2D vertex
`glVertex3f(x, y, z)`	Define 3D vertex
`glTranslatef(x, y, z)`	Move objects
`glRotatef(angle, x, y, z)`	Rotate objects
`glScalef(x, y, z)`	Scale objects
`glPushMatrix()`	Save transformation state
`glPopMatrix()`	Restore transformation state
`glEnable(GL_DEPTH_TEST)`	Enable 3D depth testing
`gluPerspective(fov, aspect, near, far)`	3D perspective
`gluOrtho2D(l, r, b, t)`	2D orthographic view

Tips for Lab Projects

1. **Start Simple**: Begin with 2D shapes before moving to 3D
2. **Use Functions**: Create reusable drawing functions for complex shapes
3. **Test Incrementally**: Add one feature at a time and test
4. **Matrix Stack**: Always use `glPushMatrix()` / `glPopMatrix()` pairs
5. **Clear Buffers**: Always clear color (and depth for 3D) buffers
6. **Animation**: Use `glutTimerFunc()` for smooth animations
7. **Debugging**: Print values to console to track transformations

Good luck with your computer graphics lab!