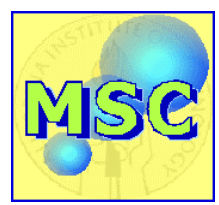
Python and OpenGL

Richard P. Muller
Materials and Process Simulation Center
California Institute of Technology
June 29, 2000



What are 3D Graphics?

- Traditional 2D graphics only store two-dimensional information
  - x,y coordinates
  - Images look like they're drawn on the screen
- 3D graphics hold three-dimensional information
  - x,y,z coordinates
  - Transform image before drawn to screen
  - Graphics boards accelerate the transformations
  - Lots of special features are also enabled, most of which we'll ignore.

What is OpenGL?

- Silicon Graphics (RIP) created the Graphics Library
  - Sometimes referred to as DGL
  - Only ran on SGI Hardware
- SGI made a open standard version of this
  - http://www.opengl.org
  - Licensed and ported to different machines
- There is a free clone of the software available at
  - http://www.mesa3d.org
  - Can port anywhere

OpenGL Family

- GL
  - The basic GL library. Only primitive commands
- GLU
  - "GL Utilities"
  - More complex commands, e.g. drawing cylinder
- GLX
  - "GL for X-Windows"
  - Commands for drawing GL shapes in X
- GLUT
  - "GL Utilities Toolkit"
  - More sophisticated windowing features, spheres, etc.

Aside: OpenGL and Games

- Game manufacturers like OpenGL
  - Quake, Diablo, etc.
- OpenGL-compatible graphics boards are massed produced and become cheaper
- No longer need $20k workstation to do molecular graphics!
- Ending? Microsoft pushing people toward Direct3D…

Caveats before we begin

- We'll only look at a small subset of OpenGL
  - Balls
  - Sticks
  - Lighting
- OpenGL is fairly difficult
  - We'll begin defining libraries to make it a bit easier
  - More help is welcome!

Hello, World

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

glutInit("Hello, World")
glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB |GLUT_DEPTH)
glutInitWindowSize(400,400)
glutCreateWindow("Hello, World")
glClearColor(0.,0.,0.,1.)
glutSetDisplayFuncCallback(display)
glutDisplayFunc()
glutMainLoop()
```

Hello, World Output

- Yawn!

OpenGL Callbacks

- The line

  glutSetDisplayFuncCallback(display)

  defines a callback function.
  - Just like tk used callbacks last week.
  - "display" is the name of the function that draws the screen
- Here's the display callback:

```
def display():
    glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT)
    glutSwapBuffers()
    return
```

  - doesn't do anything yet
  - Uses double buffering

Other OpenGL Callbacks

- Mouse
  - The mouse interaction is setup using:

    glutSetMouseFuncCallback(mouse)

    glutMouseFunc()
- Motion
  - The motion interaction is setup using:

    glutSetMotionFuncCallback(motion)

    glutMotionFunc()
- Keyboard
  - The keyboard interaction is setup using:

    glutSetKeyboardFuncCallback(keyboard)

    glutKeyboardFunc()

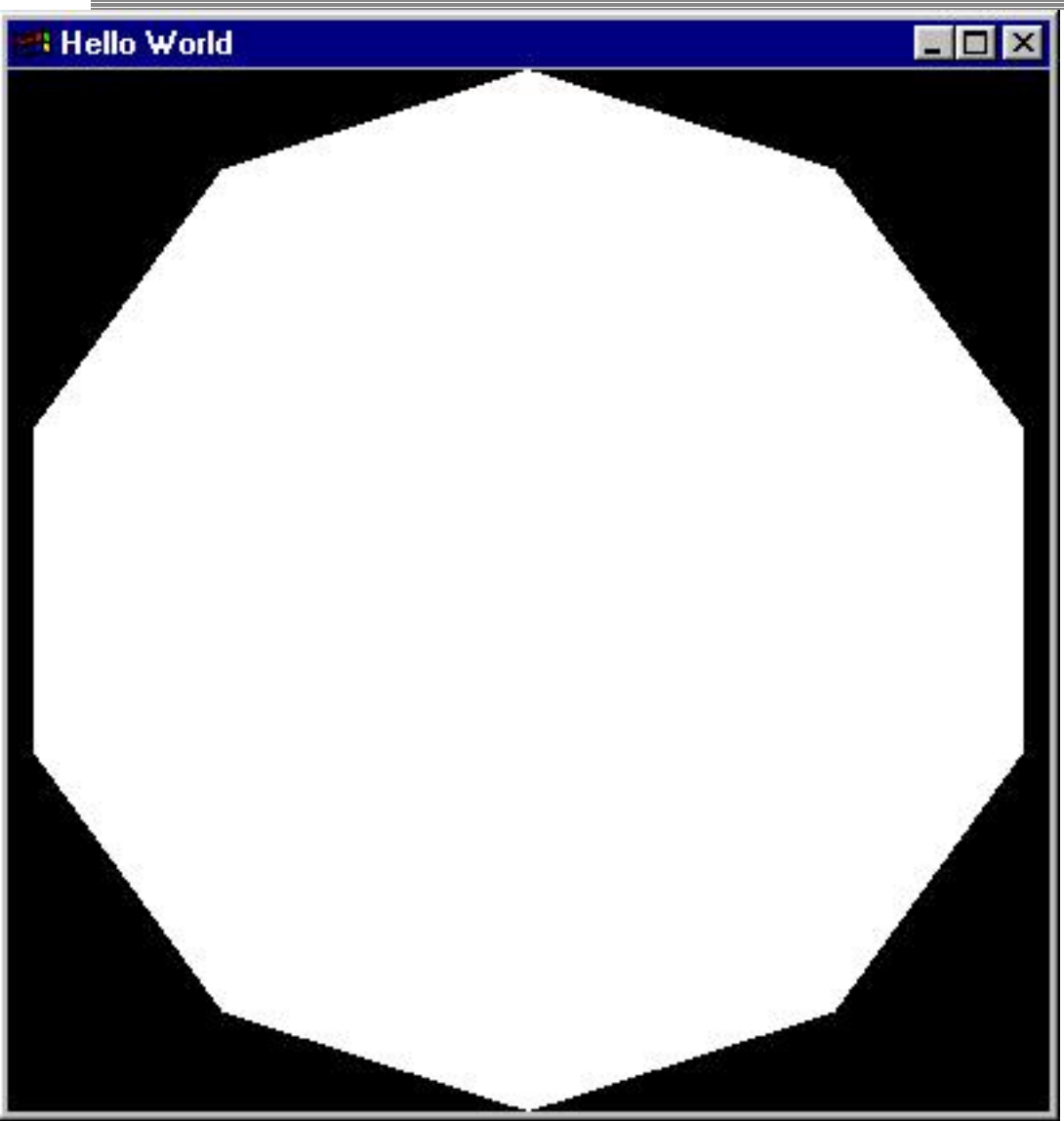
Building on Hello World

- Hello, World didn't do anything other than poping up a window.
- Obviously we want to do more sophisticated graphics
- Display Lists are ways of holding objects to draw and redraw.
  - We can have multiple display lists and flip through them
  - Right now we're just going to create one
  - We'll also only just put one item on the list, which is a little silly, since display lists are made to do complicated renderings.
  - Display lists only have to be constructed once, which means that we can put all kinds of complex stuff on the lists and call it multiple times.
  - Items put on the display lists are also executed in C, and thus render quickly.

Display Lists

```
def init_display_list():
    glNewList(list_number,GL_COMPILE)
    glPushMatrix()
    glTranslatef(0.,1.,-1.) #move to where we want to put object
    glutSolidSphere(1.,5.,5.) # make radius 1 sphere of res 5x5
    glPopMatrix()
    glEndList()
    return

def display():
    glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT)
    glCallList(list_number)
    glutSwapBuffers()
    return
```

Sphere Output

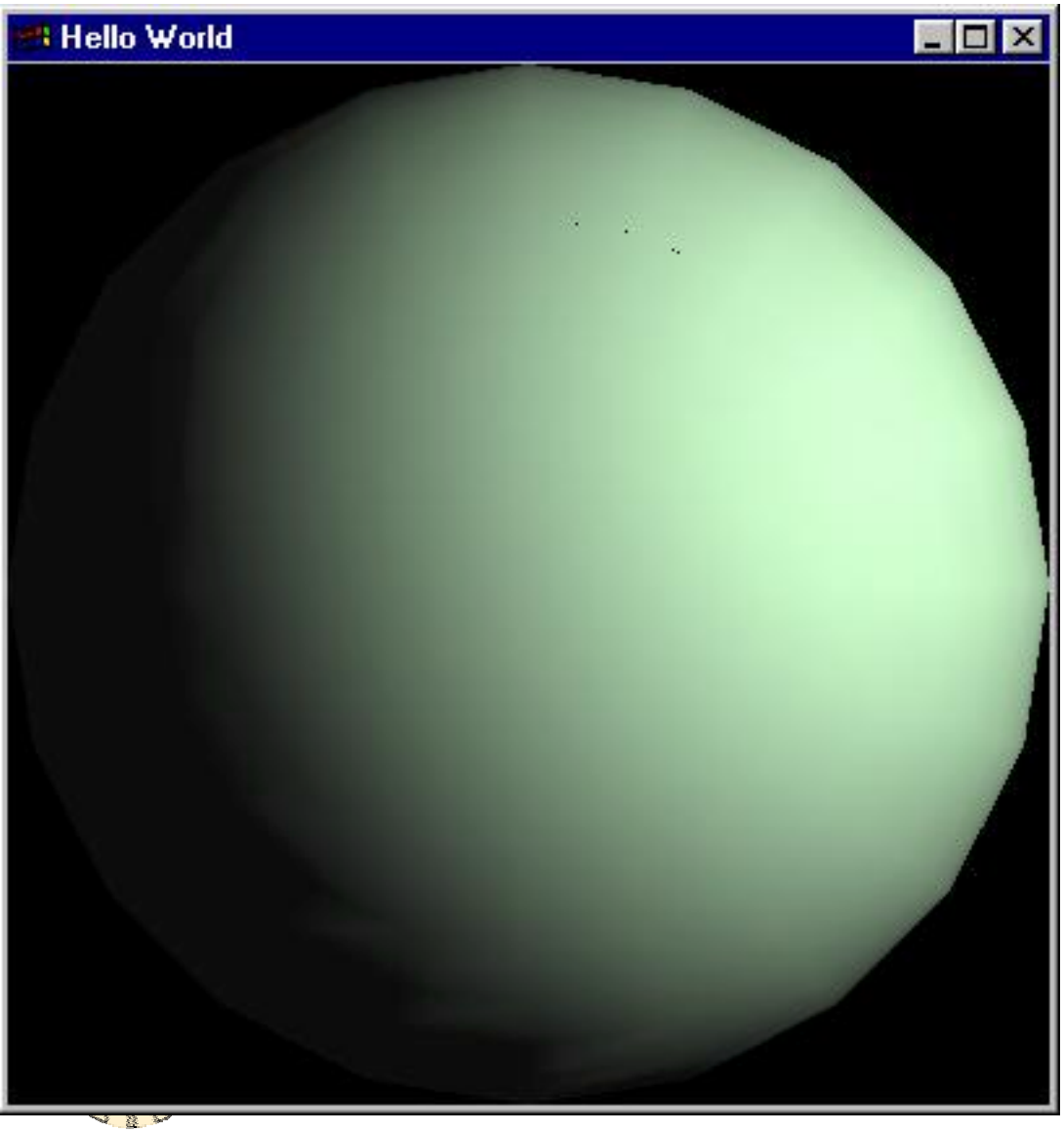


- Yuck!
- What happened?
  - Didn't add lights!

Lighting Models

```
glEnable(GL_CULL_FACE)
glEnable(GL_DEPTH_TEST)
glEnable(GL_LIGHTING)
lightZeroPosition = [10.,4.,10.,1.]
lightZeroColor = [0.8,1.0,0.8,1.0] # greenish
glLightfv(GL_LIGHT0, GL_POSITION, lightZeroPosition)
glLightfv(GL_LIGHT0, GL_DIFFUSE, lightZeroColor)
glLightf(GL_LIGHT0, GL_CONSTANT_ATTENUATION, 0.1)
glLightf(GL_LIGHT0, GL_LINEAR_ATTENUATION, 0.05)
glEnable(GL_LIGHT0)
```

Output of Lit Sphere



- A little more interesting…

More interesting display list

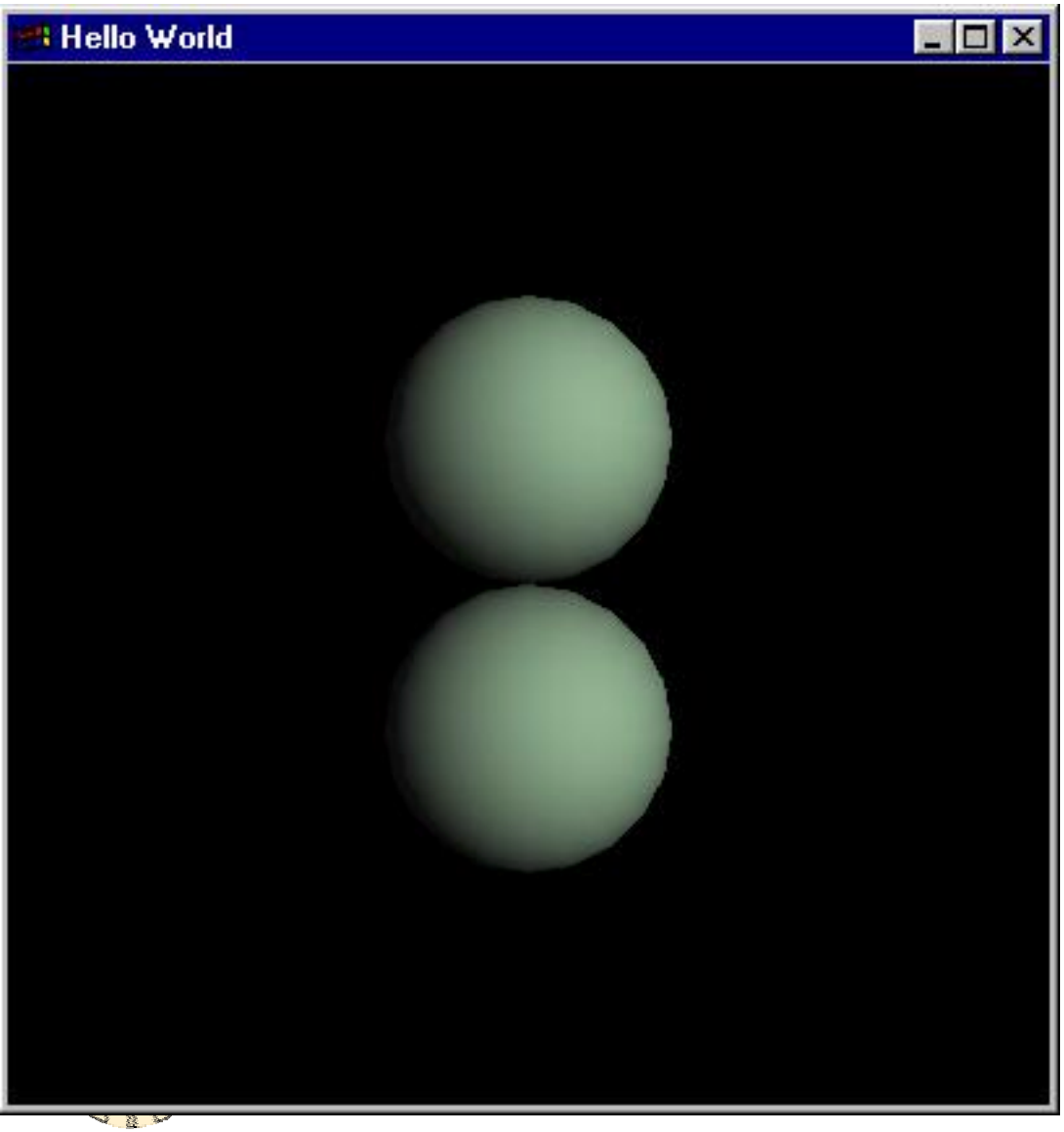
```
glNewList(1,GL_COMPILE)

glPushMatrix()
glTranslatef(0.,1.,0.) #move to where we want to put object
glutSolidSphere(1.,20.,20.) # make radius 1 sphere of res 10x10
glPopMatrix()

glPushMatrix()
glTranslatef(0.,-1.,0.) #move to where we want to put object
glutSolidSphere(1.,20.,20.) # make radius 1 sphere of res 10x10
glPopMatrix()

glEndList()
```

Define Cameras

- Cameras let you define the specific viewpoint from which to look at the scene.
  - Let you do things like rotate, move in/move out, pan, etc.
  - Camera code:

```
glMatrixMode(GL_PROJECTION)
gluPerspective(40.,1.,1.,40.) # angle, aspect ratio, near clip, far clip
glMatrixMode(GL_MODELVIEW)
gluLookAt(
    0,0,10,     # camera position
    0,0,0,      # where camera points
    0,1,0)      # which direction is up
glPushMatrix()
```

Two sphere output



- Finally starting to get a bit interesting
- Now we can look at interacting with the spheres.

Rotating the Graphics

- We want to redefine our display(), mouse() and motion() functions so we can rotate the balls
- Very simple code
- Can also do scaling and translation the same way.

Updated mouse function

```
def mouse(button,state,x,y):
    global beginx,beginy,rotate
    if button == GLUT_LEFT_BUTTON and state == GLUT_DOWN:
        rotate = 1
        beginx = x
        beginy = y
    if button == GLUT_LEFT_BUTTON and state == GLUT_UP:
        rotate = 0
    return
```

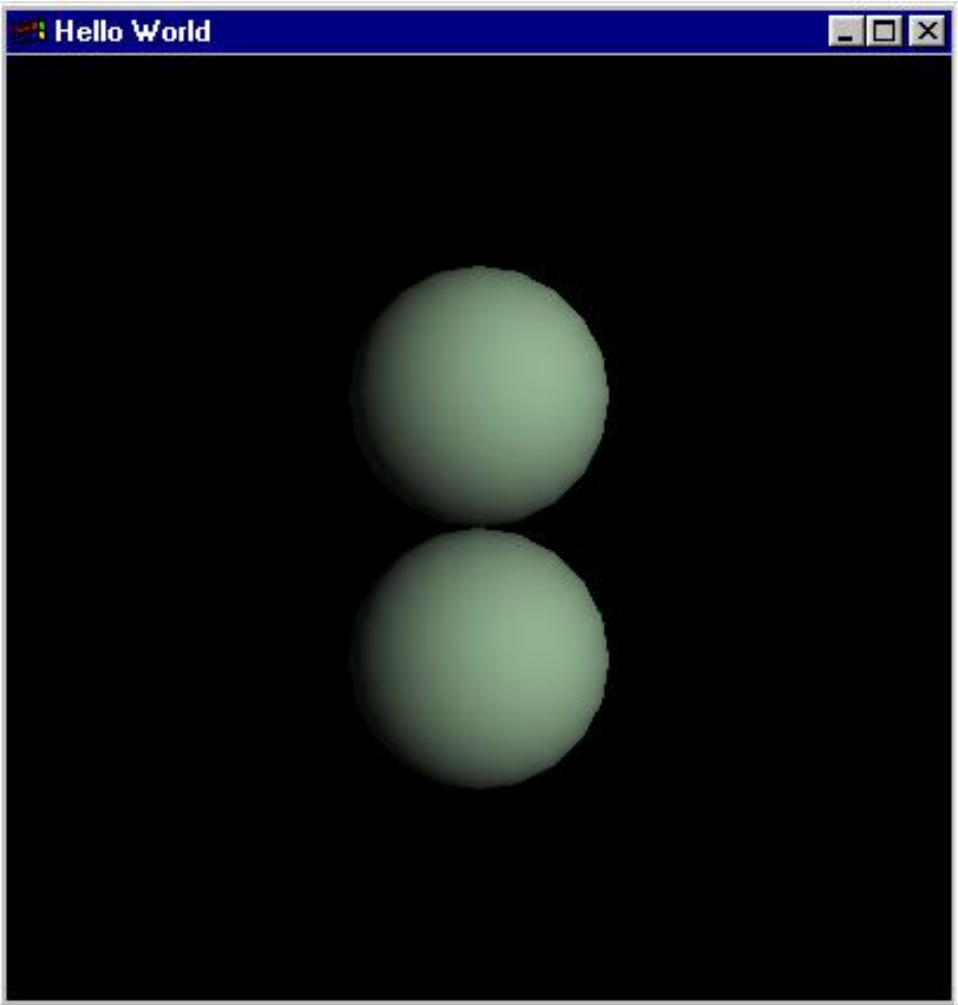
Updated motion function

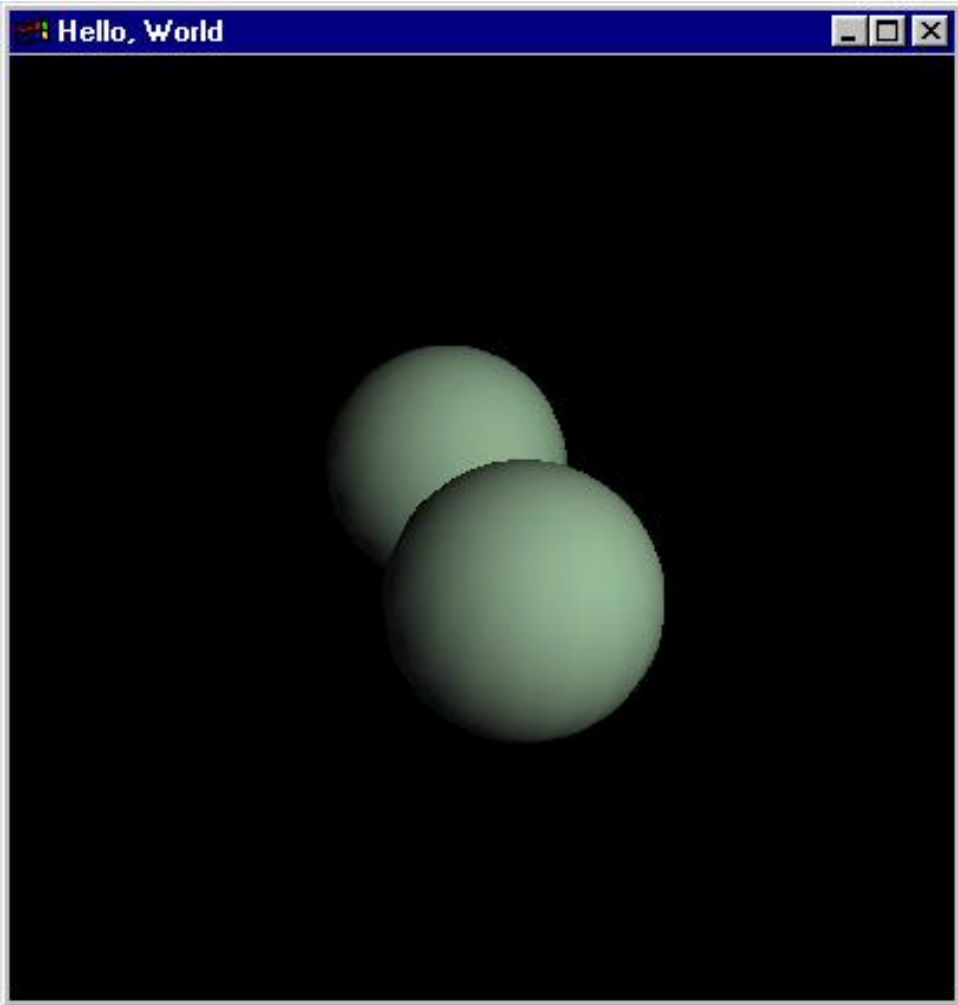
```
def motion(x,y):
    global rotx,roty,beginx,beginy,rotate
    if rotate:
        rotx = rotx + (y - beginy)
        roty = roty + (x - beginx)
        beginx = x
        beginy = y
        glutPostRedisplay()
    return
```

Updated display function

```
def display():
    glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT)
    glLoadIdentity()
    gluLookAt(0,0,10,0,0,0,0,1,0)
    glRotatef(roty,0,1,0)
    glRotatef(rotx,1,0,0)
    glCallList(1)
    glutSwapBuffers()
    return
```

Output with Rotations

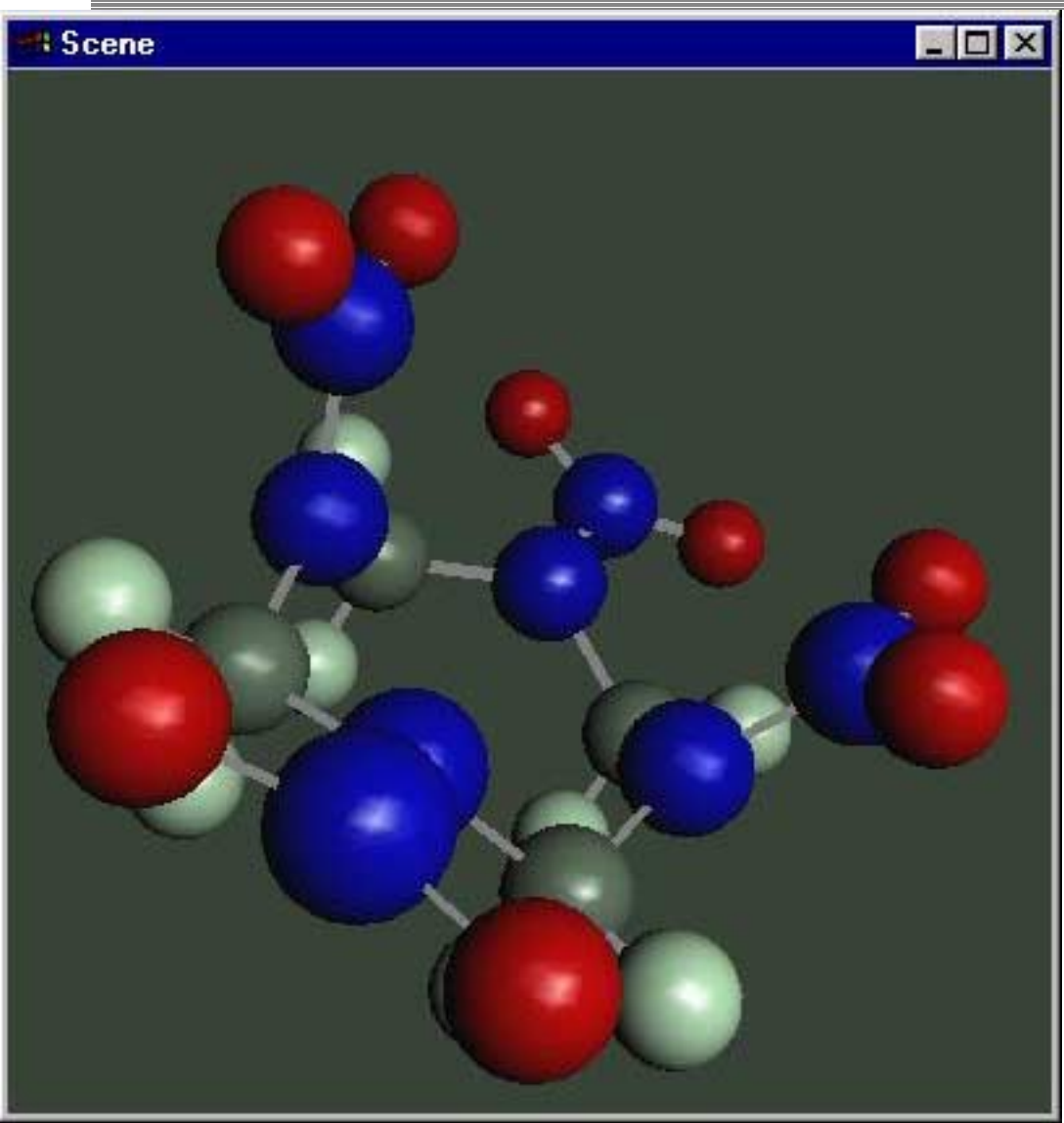




Render Library

- A framework for molecular 3d graphics
- Render module
  - Inputs the files
- RenderLib module
  - Canvas class: top-level drawing object
  - Camera class: handles camera rotations
  - Lights class
  - Objects class: the display list
  - EventManager class: the callbacks
- RenderData module
  - Atom colors, radii, etc.

Output from the Render library



Final Thoughts

- The commands aren't any easier in Python than in C
- However, you don't have to worry about compiling or porting on different platforms
  - I run the same code on Windows98 and Irix-6.4
- This lets you focus on what you're doing with the rendering rather than the technology behind the rendering.

References

- Books
  - OpenGL Programming Guide, OpenGL Architecture Review Board, Addison Wesley
  - OpenGL Programming for the X Window System, Mark Kilgard, Addison Wesley
- Web Pages
  - Python/OpenGL/Tk site: http://www.python.de