Usage of OpenGL for the Representation of a Atom Trajectory

John E. Morales García
University of Puerto Rico at Humacao
100 Teja Avenue
Humacao 00791

Faculty Adviser: José Sotero Esteva

INTRODUCTION

MoSDAS is a easy to use Graphical User Interface (GUI) developed for the study of hybrid system as a plug-in for the VMD to develop Molecular Dynamics (MD) simulations. This work is part of the beginning of the graphic representation of the molecular trajectory of a atom or molecules. The main purpose is to use OpenGL to make the representation of this trajectory real, in 3D using as base PyOpenGL.

SOFTWARE AND METHODS

In this work the softwares utilize were Dr.Python (python 2.5 programming language) and PyOpenGL (GLUT library).

MAIN PURPOSE

To find a way in which a atom trajectory can be represented in a three dimensional view, using OpenGL as a base.

PyOpenGL

What is PyOpenGL and the GLUT library?

PyOpenGL is a standard specification defining a cross-language cross-platform API for writing applications that produce 2D and 3D computer graphics. The interface have over 250 different functions calls which can be used to draw from complex three dimensional scenes from simple

primitives. Almost all modern computers have it installed. Glut (pronounced like the glut in gluttony) is the OpenGL Utility Toolkit, a window system independent toolkit for writing OpenGL programs. It implements a simple windowing application programming interface (API) for OpenGL.

FROM C++ TO PYTHON

Since python is a object oriented computer language, most of the initial approach was based on the translation of C++ language code to Python language. We chose examples that shoold result in code that we expect to use in the final implementation of MASDAS-GUI.

Here is part of the pseudo code from the C++ Glutlib library, created by Andrew William Proksel of the Northwestern University:

This piece of code do all the initialization and setup needed to use the graphic library.

```
void InitGraphics()
{
      {int argc=0; char **argv=(char **)0;
      glutInit(&argc, argv);
      }

      glutInitDisplayMode(GLUT_DOUBLE| GLUT_RGBA);

      glutInitWindowPosition(NU_SCREEN_XPOS, NU_SCREEN_YPOS);
      glutInitWindowSize(NU_SCREENWIDTH,NU_SCREENHEIGHT);
      glutCreateWindow("Northwestern University - EECS-110 ");

      glClearColor(1.0,1.0,1.0,0.0);
      glColor3d(0.0,0.0,0.0);
      glPointSize(3.0);

      glMatrixMode(GL_PROJECTION);
      glLoadIdentity();
      gluOrtho2D(0.0, (GLdouble)NU_SCREENWIDTH, 0.0, (GLdouble)NU_SCREENHEIGHT);

      glutDisplayFunc(DisplayCallback);
      glutKeyboardFunc(KeyboardCallback);
      glutSpecialFunc(SpecialCallback);
      glutMouseFunc(MouseCallback);
      glutIdleFunc(myIdle);

      glutMainLoop();
```

which its equivalent in Python is:

```
def InitGraphics(DisplayCallback):
```

```
      argv= []
      glutInit(argv)


      glutInitDisplayMode(GLUT_DOUBLE| GLUT_RGBA)

      glutInitWindowPosition(NU_SCREEN_XPOS, NU_SCREEN_YPOS)
      glutInitWindowSize(NU_SCREENWIDTH,NU_SCREENHEIGHT)
      glutCreateWindow("UPRH ")

      glClearColor(1.0,1.0,1.0,0.0)
      glColor3d(0.0,0.0,0.0)
      glPointSize(3.0)

      glMatrixMode(GL_PROJECTION)
      glLoadIdentity()
      gluOrtho2D(0.0, NU_SCREENWIDTH, 0.0, NU_SCREENHEIGHT)


      glutDisplayFunc(DisplayCallback)
      glutKeyboardFunc(KeyboardCallback)
      glutSpecialFunc(SpecialCallback)
      glutMouseFunc(MouseCallback)
      glutIdleFunc(myIdle)

      glutMainLoop()


angle = 0.0
NU_ANGLESTEP = pi / 180.0

rom10  = GLUT_BITMAP_TIMES_ROMAN_10
rom24  = GLUT_BITMAP_TIMES_ROMAN_24
helv10 = GLUT_BITMAP_HELVETICA_10
helv12 = GLUT_BITMAP_HELVETICA_12
helv18 = GLUT_BITMAP_HELVETICA_18
```

SIMULATIONS

Heat Transfer.

The heat transfer simulation consist of showing how a metal plate behave when heat its apply into one of the corners. Using of reference some example of Matrix without Linear Algebra of the course MATE 3009 given at the University of Puerto Rico at Humacao in 2006.

This part of the pseudo code defines dibujaPlaca(). To define this function, we most first scan the matrix that have the information and then paint a square with a color that represent the temperature assigned.

implemented in C++

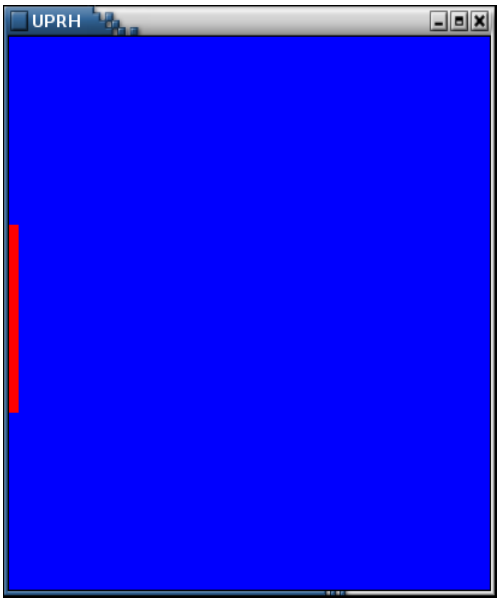
```
void dibujaPlaca(){
   int anchoCuadrito = 400 / N;
   for(int n=0; n < N; n++)
   for(int m=0; m < M; m++){
      double intensidad = (placa[n][m]tempMin)/(tempMaxtempMin);
      SetPenColor(intensidad, 0.0, 1.0intensidad);
      int baseX = n * anchoCuadrito, baseY = m * anchoCuadrito;
      DrawFillBox(baseX, baseY, baseX+anchoCuadrito,
                  baseY+anchoCuadrito);
   }
}
```

In Python since dibujaPlaca is declare void, theres no need for a formal function call dibujaPlaca(), we just can add the definition into the display function and we still getting the same results.

implemented in Python with PyOpenGL

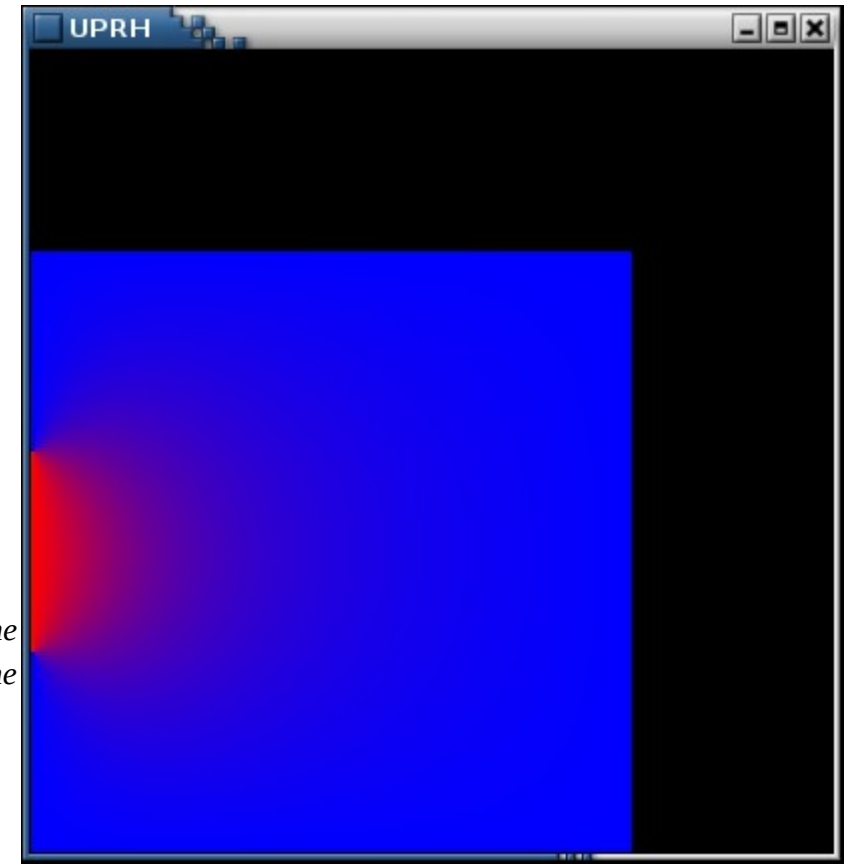
```
def myDisplay():
      global placa, tInterior, NMAX, tempMax, NU_SCREENWIDTH
      calculaTemperaturas(2.2, placa, NMAX)

      # dibuja
      anchoCuadrito = NU_SCREENWIDTH / NMAX
      intensidad = random.randrange(0.0,1.0)

      for m in range(NMAX):
            for n in range(NMAX):
                  intensidad = (placa[n][m] - tInterior) / (tempMax - tInterior)
                  SetPenColor(intensidad, 0.0, 1.0 - intensidad)
                  baseX = n * anchoCuadrito
                  baseY = m * anchoCuadrito
                  DrawFillBox(baseX, baseY, baseX + anchoCuadrito,baseY + \
anchoCuadrito)
```

Here are some pictures of the Heat Transfer Simulation.



In here, heat is apply to the middle left corner of the plate. (red area represent the heat, and the blue area is the plate).

In this other picture we can see the propagation of the heat over the plate.(the red haze area).



Movement of Spherical Bodies.

The main purpose of this simulation was to study the movement of spherical bodies and they're surface. To learn more of spherical objects I used as reference the OpenGL Programming Guide , better know as The Red Book , which is in C++ and not in Python.

Step 1. Building a wired solar system:

Since the Red Book is write in C++, we need to translate it into Python language.

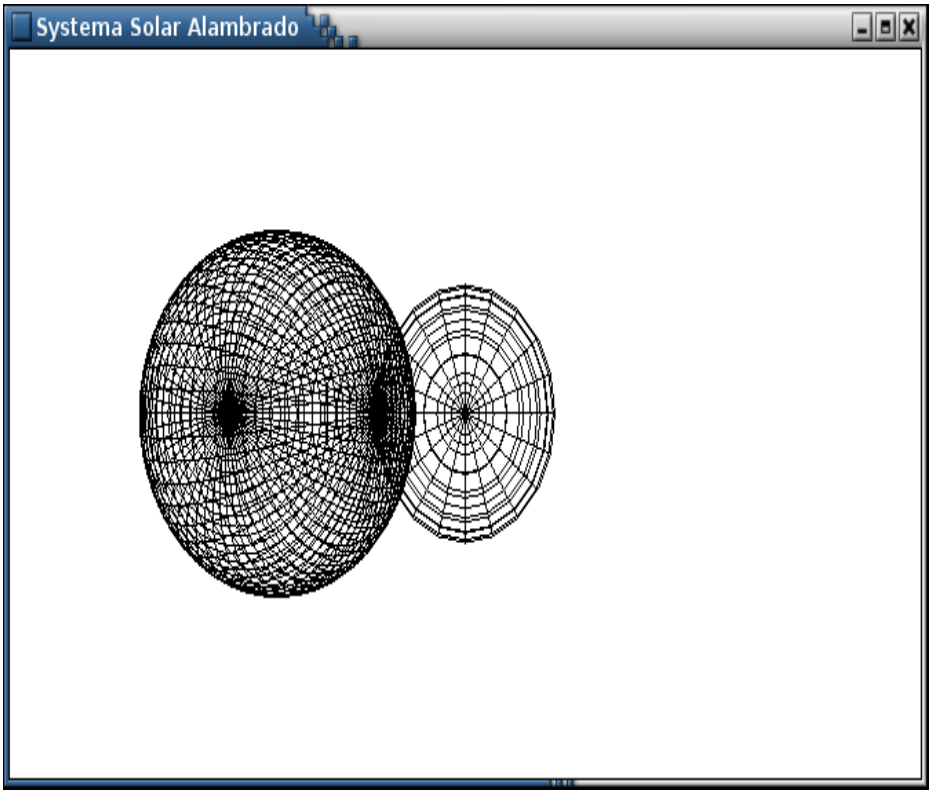
Since we are interested in Spherical Bodies, heres the function that display the spheres, this the display function of the program.

```
def display():
      glClear(GL_COLOR_BUFFER_BIT)
      glColor3f(0.0,0.0,0.0)

      glPushMatrix()
      glutWireSphere(1.0,20,16)
      glRotatef(year,0.0,1.0,0.0)
      glTranslatef(2.0,0.0,0.0)
      glRotatef(day,0.0,1.0,0.0)
      glutWireSphere(1.0,60,40)
      glPopMatrix()
      glutSwapBuffers()
```

This part of code is the responsible for giving movement to the solar system. It have the keyboard command to make the sphere that represent the planet to rotate in its own axis and around the sphere that represents the sun.

```
def keyboard(key, x, y):
      global year, day
      if key == 'd':
            day = (day + 10) % 360
            glutPostRedisplay()

      elif key == 'D':
            day = (day - 10) % 360
            glutPostRedisplay()

      elif key == 'y':
            year = (year + 5) % 360
            glutPostRedisplay()

      elif key == 'Y':
            year = (year - 5) % 360
            glutPostRedisplay()
```

This how the wired solar system looks at the End.

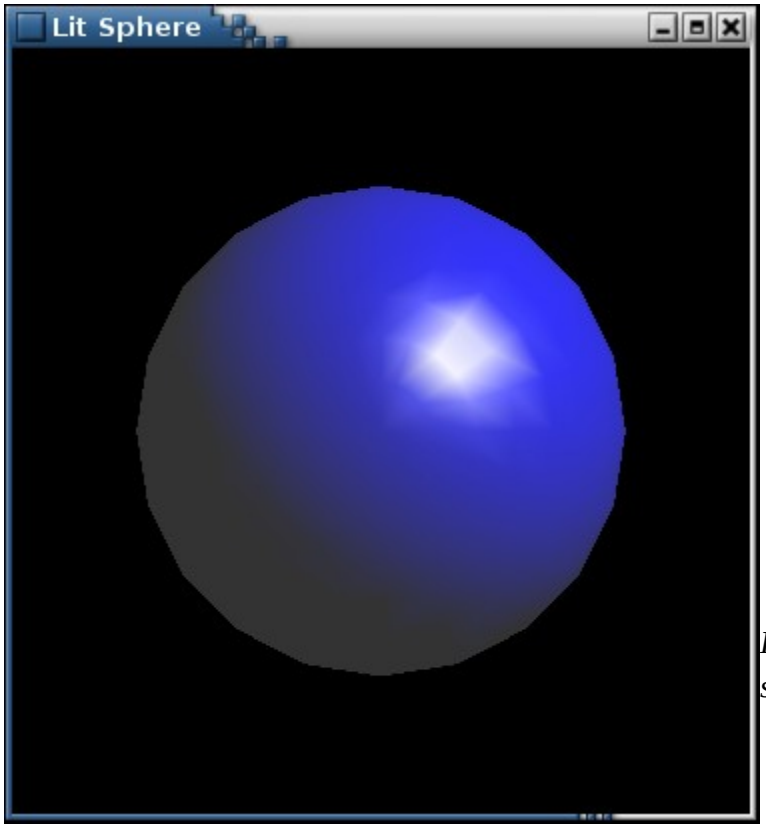


Step 2. Applying texture to a sphere, the Lit Sphere :

Since we now know how to make a wired sphere, we can now try to put some texture, light, color and light. The next function is the initialization for the rendering of a lit sphere.

```
def init():
      mat_specular = [1.0,1.0,1.0,1.0]
      mat_shininess = [50.0]
      light_position = [1.0,1.0,1.0,0.0]
      white_light = [0.0, 0.0,1.0,0.0]
      lmodel_ambient = [1.0,1.0,1.0,0.0]
      glClearColor(0.0,0.0,0.0,0.0)
      glShadeModel(GL_SMOOTH)
      glMaterialfv(GL_FRONT,GL_SPECULAR,mat_specular)
      glMaterialfv(GL_FRONT,GL_SHININESS,mat_shininess)
      glLightfv(GL_LIGHT0,GL_POSITION, light_position)
      glLightfv(GL_LIGHT0,GL_DIFFUSE, white_light)
      glLightModelfv(GL_LIGHT_MODEL_AMBIENT,lmodel_ambient)

      glEnable(GL_LIGHTING)
      glEnable(GL_LIGHT0)
      glEnable(GL_DEPTH_TEST)
```

With this code we are able to position one or more light source. It also define material properties for the object in the scene and define the level of global ambient light and the effective location of the viewpoint.



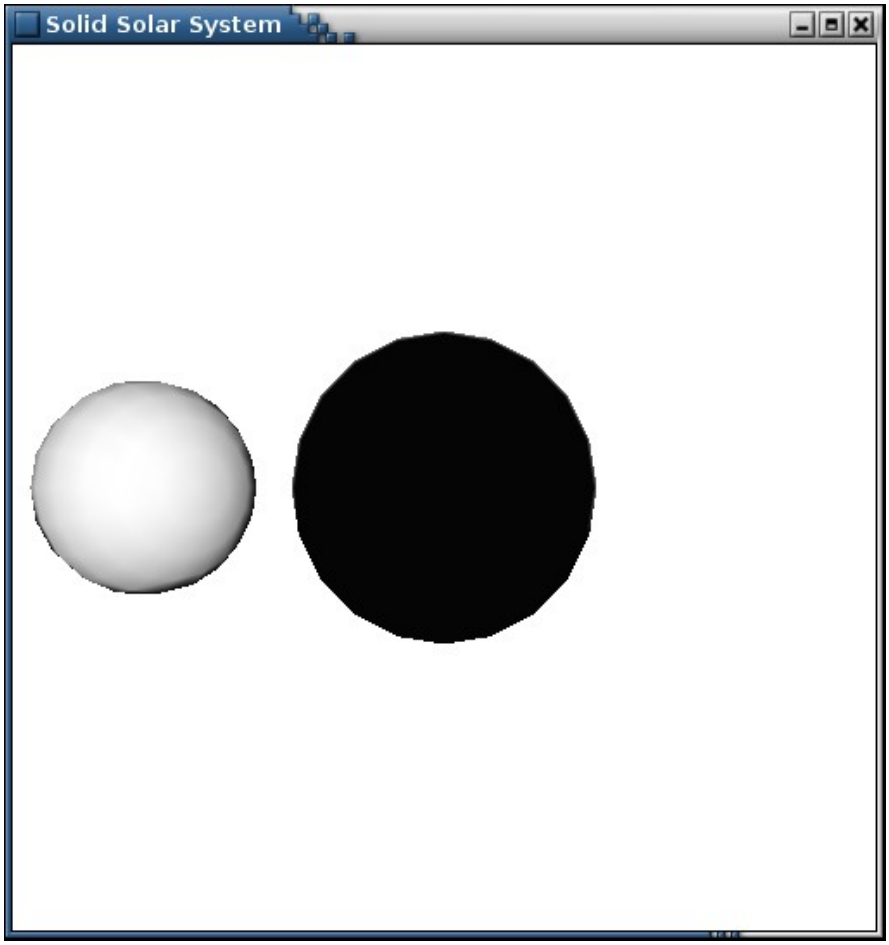
Here are the final results of the Lit sphere program.

Step 3. A solid solar system:

Now that we have the wired solar system, lets try to apply the Lit sphere method to it. If we add the previous code from the Lit sphere program and also change the display function of the solar system (previously shown) we can make a solid solar system.

```
def display():
      glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
      glColor3f(0.0,0.0,0.0)

      glPushMatrix()
      glutSolidSphere(1.0,20,16)
      glRotatef(year,0.0,1.0,0.0)
      glTranslatef(2.0,0.0,0.0)
      glRotatef(day,0.0,1.0,0.0)
      glutSolidSphere(0.5,20,10)
      glPopMatrix()
      glutSwapBuffers()
      glFlush()
```

The key definition here is glutSolidSphere(), which is the one responsible for solidifying the spheres.



This figure show the results of the solid solar system. (The Grey ball is the planet and the black one the sun).

REVIEW

PyOpenGL is a cross language, cross-platform that let us use the object oriented language Python, work well with the GLUT library. The C++ codes for some programs can be easily translate into Python without trouble. Simulations such like heat transfer and solar systems, can be of help in the development of new approach to MD. The behavior of spherical bodies is a good approach to the representation of a atom behavior.

CONCLUSION

The main purpose of all this simulations in OpenGL, is to apply them to the MoSDAS to create a way to represent the atom trajectory in a three dimensional view. With this methods of movement of spherical bodies we can give the atoms a sphere behavior to make the work of molecule trajectory more easier and since the adaptation of C++ language to Python works very well, we are able to work in a new methods and ideas in the future of Molecular Dynamics.

REFERENCE

Mason Woo, Jackie Neider, Tom Davis, DaveShreiner. OpenGL Programming Guide. Third Edition.Pearson Educatin Corporate Sales Division. 201 W.103rd Street

Tom McReynolds, David Blythe. Advanced Graphics Programming Using OpenGL . 500 Sansome Street, Suite 400, San Francisco, CA 94111.

Graphical User Interface to Run modelecular Dynamics Simulations of CNT-Polymer Hybrids in VMD , Myrna I. Merced, Paper, December 2007.