CENTRO *UNIVERSITÁRIO* DE BARRA MANSA
ACADEMIC PRO-RECTORY

COMPUTER ENGINEERING COURSE

CONSTRUCTION AND SIMULATION OF A ROBOT ARM WITH OPENGL

By:

Leniel Braz de Oliveira Macaferi
Wellington Magalhães Leite

Barra Mansa
May 16, 2007

CENTRO *UNIVERSITÁRIO* DE BARRA MANSA
ACADEMIC PRO-RECTORY

COMPUTER ENGINEERING COURSE

CONSTRUCTION AND SIMULATION OF A ROBOT ARM WITH OPENGL

By:

Leniel Braz de Oliveira Macaferi
Wellington Magalhães Leite

Paper presented to the Computer Engineering course at Centro Universitário de Barra Mansa, as a partial requisite to the obtention of the second grade in the Computer Graphics discipline, under prof. Ronaldo Dias Corrêa supervision.

Barra Mansa
May 16, 2007

ABSTRACT

The importance of projects related to the field of Computer Graphics in simulations has been growing a lot during the last years. Therefore it brings to life the necessity of mastering the concepts and techniques inherent to the process of elaboration, construction and simulation of a given graphical project.

The OpenGL API specification tries to help us when we are programming the graphical details of a given project.

In this article we’re showing the necessary steps and routines to the proper codification and simulation of a robotic arm in 3D, which is the most employed robot in the manufacturing industry and in areas that require a high precision rate.

With a simulation (virtual) model, we can have a closer vision of the object of study in contrast with reality, what make us capable of foreseeing how a determined object will look like and how it will behave after its proper construction in the physical world.

Keywords: robot arm, OpenGL, 3D simulation, computer graphics

TABLE OF FIGURES

LIST OF TABLES

CONTENTS

1 INTRODUCTION

1.1 Objective

Our objective is to create a robot arm with claws that simulate the opening and closing motions. The model will be created in a virtual way through OpenGL programming.

1.2 Definition

1.2.1 Robot arm

A robotic arm can be classified as articulated and not articulated [1].

It's more autonomous than a simple mechanic arm and can be used to lift small parts with high precision and velocity. It's generally used in tasks such as: welding, painting, assembling, packaging, storage, product inspection and test, using to that purpose final actuators.

The robot arm can be built in a fix or mobile fashion (e.g.: with wheels) to be used in industries or in home (where it's also called micro-arm).

In the industrial field it's defined by ISO (International Standards Organization) as an automatically controlled, reprogrammable, multiuse programmable manipulator with three or more pivots.

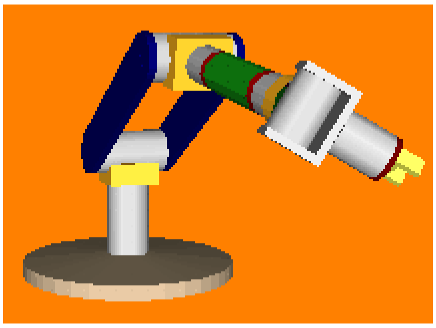

Figure 1 - Articulated robot arm

1.2.2 OpenGL

OpenGL (Open Graphics Library) [2] is a standard specification that defines an API (Application Program Interface) that is multi-language and multi-platform and that enables the codification of applications that output computerized graphics in 2D and 3D.

The interface consists in more than 250 different functions, which can be used to draw complex tridimensional scenes with simple primitives.

OpenGL was developed by Silicon Graphics Inc. (SGI) on 1992 and is popular in the gaming industry where it competes with the Direct3D in the Microsoft Windows platform. OpenGL is broadly used in CAD (Computer Aided Design), virtual reality, scientific visualization, information visualization, flight simulation and video games development.

Figure 2 - OpenGL logo

2 DEVELOPMENT

Our development started from the basic idea of a robot arm that we found at the site [3] of the Computer Science department from the University of North Caroline - Chapel Hill campus. As a matter of fact, we build upon a homework elaborated by the professors of the department that is related to the topics of Computer Graphics in respect to graphics scenes, hierarchical transformations, instantiation, stack of matrices and OpenGL transformations.

2.1 Provided files

A file called robot.cpp, which contains the primitives to the construction of the robot arm, that is, the display functions to all the parts of a robot arm and an executable file called robot1.exe, which contains the robot arm already constructed were initially provided.

2.2 Robot arm construction

With the given files described in 2.1 we executed the following steps to program (assemble) the robot arm as stated in the provided guide at [3]:

- Compile and execute the file robot.cpp, which draws in the screen only a flat grey surface.
- Execute the file robot1.exe to see the final result of the robot arm, what also serves to verify the code library glut32.dll is necessary in the folder of the executable file.
- Draw the robot arm’s parts to determine the correct dimensions. This gave us the local coordinates of each part of the robot arm.
- Visualize the executable robot1.exe to determine how the parts must be connected and what transformations are necessary to support the control of the variables.

The user interface that changes the variables’ control (as shown in the file robot1.exe) was already configured and our objective was to complement the display routine DrawRobotArm that wasn’t implemented. To accomplish that we used a stack of matrices from the OpenGL and other necessary transformations.

An advice that was given: read the third chapter of the book OpenGL RedBook so that we could have a better understanding to execute the construction of the robot arm.

As a restriction: the output program must be equal to the provide program, that is, the position and motion of the robot arm’s part must match the provided program.

2.3 Framework employed

To elaborate this work we used the Tao framework [4], which provides to the .Net programmers (Microsoft) access to the common OpenGL code libraries.

2.4 Code conversion from C to C#

As an extra motivation, we converted the C code to C# and we used the Microsoft C++ 2005 Express Edition and Microsoft Visual C# 9.0 Express Edition to do so.

2.5 The DrawRobotArm function

```
static void DrawRobotArm(int numSegs)
{
  GL.glMatrixMode(GL.GL_MODELVIEW);

  GL.glTranslatef(baseTransX, 0, baseTransZ);
  GL.glRotatef(baseSpin, 0f, 360f, 0f);
  DrawBase(64);

  GL.glTranslatef(0, 0.4f, 0);
  GL.glRotatef(shoulderAng, 0f, 0f, 90f);
  DrawArmSegment(64);

  GL.glTranslatef(0, 0.5f, 0);
  GL.glRotatef(elbowAng, 0f, 0f, 90f);
  DrawArmSegment(64);

  GL.glTranslatef(0, 0.5f, 0);
  GL.glRotatef(wristAng, 0.0f, 0f, 90f);
  DrawWrist(16);

  GL.glRotatef(wristTwistAng, 0.0f, 180f, 0f);

  GL.glPushMatrix();
  GL.glTranslatef(0f, 0.2f, 0f);
  GL.glRotatef(fingerAng1, 0f, 0f, -180f);
  DrawFingerBase(16);

  GL.glTranslatef(0f, 0.3f, 0f);
  GL.glRotatef(fingerAng2, 0f, 0f, -90f);
  DrawFingerTip(16);

  GL.glPopMatrix();

  GL.glPushMatrix();

  GL.glTranslatef(0f, 0.2f, 0f);
  GL.glRotatef(fingerAng1, 0f, 0f, 90f);
  DrawFingerBase(16);

  GL.glTranslatef(0f, 0.3f, 0);
  GL.glRotatef(fingerAng2, 0f, 0f, 90f);
  DrawFingerTip(16);

  GL.glPopMatrix();
}
```

2.6 Complete source code

```
//
//  Construction and Simulation of a Robot Arm with OpenGL
//  Copyright ©2007 Leniel Braz de Oliveira Macaferi & Wellington Magalhães
Leite.
//
//  UBM COMPUTER ENGINEERING - 9TH TERM [http://www.ubm.br/]
//  This program sample was developed and turned in as a term paper for
Computer Graphics
//  The source code is provided "as is" without warranty.
//
```

```
//  The original code can be found at:
//  http://www.cs.unc.edu/~dm/UNC/COMP236/Homeworks/hw1b/

using System;
using System.Windows.Forms;
using OpenGL;
using Tao.FreeGlut;

namespace ComputerGraphics
{
  /// <summary>
  /// Creates a robot arm
  /// </summary>
  class RobotArm
  {
    // Robot's arm controls
    static float baseTransX = -0.5f; // 0
    static float baseTransZ = 0;
    static float baseSpin = 0;        // 1
    static float shoulderAng = -10;   // 2
    static float elbowAng = -120;
    static float wristAng = 90;       // 3
    static float wristTwistAng = 10;
    static float fingerAng1 = 45;     // 4
    static float fingerAng2 = -90;

    // Robot's colors
    static byte[] arms = { 128, 128, 128 };
    static byte[] joints = { 0, 68, 119 };
    static byte[] fingers = { 150, 0, 24 };
    static byte[] fingerJoints = { 128, 128, 128 };

    // User interface global variables
    static bool leftButtonDown = false;     // Mouse stuff
    static float oldX, oldY, newX, newY;
    static int robotControl = 1;

    static void DrawUnitCylinder(int numSegs)  // x,y,z in [0,1], Y-axis is
up
    {
      int i;
      float[] Px = new float[numSegs];
      float[] Py = new float[numSegs];
      float AngIncr = (2.0f * 3.1415927f) / (float)numSegs;
      float Ang = AngIncr;
      Px[0] = 1;

      Py[0] = 0;

      for (i = 1; i < numSegs; i++, Ang += AngIncr)
      {
        Px[i] = (float)Math.Cos(Ang);
        Py[i] = (float)Math.Sin(Ang);
      }

      GL.glMatrixMode(GL.GL_MODELVIEW);
      GL.glPushMatrix();
      GL.glTranslatef(0.5f, 0.5f, 0.5f);
      GL.glScalef(0.5f, 0.5f, 0.5f);

      // Top
```

```
      GL.glNormal3f(0, 1, 0);
      GL.glBegin(GL.GL_TRIANGLE_FAN);
      GL.glVertex3f(0, 1, 0);
      for (i = 0; i < numSegs; i++)
        GL.glVertex3f(Px[i], 1, -Py[i]);
      GL.glVertex3f(Px[0], 1, -Py[0]);
      GL.glEnd();

      // Bottom
      GL.glNormal3f(0, -1, 0);
      GL.glBegin(GL.GL_TRIANGLE_FAN);
      GL.glVertex3f(0, -1, 0);
      for (i = 0; i < numSegs; i++)
        GL.glVertex3f(Px[i], -1, Py[i]);
      GL.glVertex3f(Px[0], -1, Py[0]);
      GL.glEnd();

      // Sides
      GL.glBegin(GL.GL_QUAD_STRIP);
      for (i = 0; i < numSegs; i++)
      {
        GL.glNormal3f(Px[i], 0, -Py[i]);
        GL.glVertex3f(Px[i], 1, -Py[i]);
        GL.glVertex3f(Px[i], -1, -Py[i]);
      }
      GL.glNormal3f(Px[0], 0, -Py[0]);
      GL.glVertex3f(Px[0], 1, -Py[0]);
      GL.glVertex3f(Px[0], -1, -Py[0]);
      GL.glEnd();

      GL.glPopMatrix();
    }

    static void DrawUnitSphere(int numSegs)  // x,y,z in [0,1]
    {
      GL.glMatrixMode(GL.GL_MODELVIEW);
      GL.glPushMatrix();
      GL.glTranslatef(0.5f, 0.5f, 0.5f);
      Glut.glutSolidSphere(0.5f, numSegs, numSegs);
      GL.glPopMatrix();
    }

    static void DrawUnitCone(int numSegs)  // x,y,z in [0,1], apex is in +Y
direction
    {
      GL.glMatrixMode(GL.GL_MODELVIEW);
      GL.glPushMatrix();
      GL.glTranslatef(0.5f, 0, 0.5f);
      GL.glRotatef(-90, 1, 0, 0);
      Glut.glutSolidCone(0.5f, 1, numSegs, numSegs);
      GL.glPopMatrix();
    }

    static void DrawGroundPlane(int numSegs)
    {
      GL.glColor3f(0.7f, 0.7f, 0.7f);
      GL.glBegin(GL.GL_QUADS);
      GL.glNormal3f(0f, 1f, 0f);
      GL.glVertex3f(-1f, 0f, 1f);
      GL.glVertex3f(1f, 0f, 1f);
      GL.glVertex3f(1f, 0f, -1f);
```

```
  GL.glVertex3f(-1f, 0f, -1f);
  GL.glEnd();
}

static void DrawJoint(int numSegs)
{
  GL.glMatrixMode(GL.GL_MODELVIEW);
  GL.glPushMatrix();
  GL.glScalef(0.15f, 0.15f, 0.12f);
  GL.glRotatef(90, 1, 0, 0);
  GL.glTranslatef(-0.5f, -0.5f, -0.5f);
  GL.glColor3ubv(joints);
  DrawUnitCylinder(numSegs);
  GL.glPopMatrix();
}

static void DrawBase(int numSegs)
{
  GL.glMatrixMode(GL.GL_MODELVIEW);
  GL.glPushMatrix();
  GL.glScalef(0.2f, 0.025f, 0.2f);
  GL.glTranslatef(-0.5f, 0, -0.5f);
  GL.glColor3ubv(joints);
  DrawUnitCylinder(numSegs);
  GL.glPopMatrix();
  GL.glPushMatrix();
  GL.glTranslatef(-0.05f, 0, -0.05f);
  GL.glScalef(0.1f, 0.4f, 0.1f);
  GL.glColor3ubv(arms);
  DrawUnitCylinder(numSegs);
  GL.glPopMatrix();
  GL.glPushMatrix();
  GL.glTranslatef(0, 0.4f, 0);
  DrawJoint(numSegs);
  GL.glPopMatrix();
}

static void DrawArmSegment(int numSegs)
{
  GL.glMatrixMode(GL.GL_MODELVIEW);
  GL.glPushMatrix();
  GL.glTranslatef(-0.05f, 0, -0.05f);
  GL.glScalef(0.1f, 0.5f, 0.1f);
  GL.glColor3ubv(arms);
  DrawUnitCylinder(numSegs);
  GL.glPopMatrix();
  GL.glPushMatrix();
  GL.glTranslatef(0, 0.5f, 0);
  DrawJoint(numSegs);
  GL.glPopMatrix();
}

static void DrawWrist(int numSegs)
{
  GL.glMatrixMode(GL.GL_MODELVIEW);
  GL.glPushMatrix();
  GL.glTranslatef(-0.04f, 0, -0.04f);
  GL.glScalef(0.08f, 0.2f, 0.08f);
  GL.glColor3ubv(fingers);
  DrawUnitCylinder(numSegs);
  GL.glPopMatrix();
```

```
  GL.glPushMatrix();
  GL.glTranslatef(0, 0.2f, 0);
  GL.glScalef(0.12f, 0.12f, 0.12f);
  GL.glTranslatef(-0.5f, -0.5f, -0.5f);
  GL.glColor3ubv(fingerJoints);
  DrawUnitSphere(numSegs);
  GL.glPopMatrix();
}

static void DrawFingerBase(int numSegs)
{
  GL.glMatrixMode(GL.GL_MODELVIEW);
  GL.glPushMatrix();
  GL.glTranslatef(-0.025f, 0, -0.025f);
  GL.glScalef(0.05f, 0.3f, 0.05f);
  GL.glColor3ubv(fingers);
  DrawUnitCylinder(numSegs);
  GL.glPopMatrix();
  GL.glPushMatrix();
  GL.glTranslatef(0, 0.3f, 0);
  GL.glScalef(0.08f, 0.08f, 0.08f);
  GL.glTranslatef(-0.5f, -0.5f, -0.5f);
  GL.glColor3ubv(fingerJoints);
  DrawUnitSphere(numSegs);
  GL.glPopMatrix();
}

static void DrawFingerTip(int numSegs)
{
  GL.glMatrixMode(GL.GL_MODELVIEW);
  GL.glPushMatrix();
  GL.glScalef(0.05f, 0.25f, 0.05f);
  GL.glTranslatef(-0.5f, 0, -0.5f);
  GL.glColor3ubv(fingers);
  DrawUnitCone(numSegs);
  GL.glPopMatrix();
}

static void DrawRobotArm(int numSegs)
{
  GL.glMatrixMode(GL.GL_MODELVIEW);

  GL.glTranslatef(baseTransX, 0, baseTransZ);
  GL.glRotatef(baseSpin, 0f, 360f, 0f);
  DrawBase(64);

  GL.glTranslatef(0, 0.4f, 0);
  GL.glRotatef(shoulderAng, 0f, 0f, 90f);
  DrawArmSegment(64);

  GL.glTranslatef(0, 0.5f, 0);
  GL.glRotatef(elbowAng, 0f, 0f, 90f);
  DrawArmSegment(64);

  GL.glTranslatef(0, 0.5f, 0);
  GL.glRotatef(wristAng, 0.0f, 0f, 90f);
  DrawWrist(16);

  GL.glRotatef(wristTwistAng, 0.0f, 180f, 0f);

  GL.glPushMatrix();
```

```
  GL.glTranslatef(0f, 0.2f, 0f);
  GL.glRotatef(fingerAng1, 0f, 0f, -180f);
  DrawFingerBase(16);

  GL.glTranslatef(0f, 0.3f, 0f);
  GL.glRotatef(fingerAng2, 0f, 0f, -90f);
  DrawFingerTip(16);

  GL.glPopMatrix();

  GL.glPushMatrix();

  GL.glTranslatef(0f, 0.2f, 0f);
  GL.glRotatef(fingerAng1, 0f, 0f, 90f);
  DrawFingerBase(16);

  GL.glTranslatef(0f, 0.3f, 0);
  GL.glRotatef(fingerAng2, 0f, 0f, 90f);
  DrawFingerTip(16);

  GL.glPopMatrix();
}

static void myDisplay()
{
  GL.glClear(GL.GL_COLOR_BUFFER_BIT | GL.GL_DEPTH_BUFFER_BIT);

  GL.glMatrixMode(GL.GL_MODELVIEW);
  GL.glLoadIdentity();
  GLU.gluLookAt(0f, 2f, 4f, 0f, 0.5f, 0f, 0f, 1f, 0f);

  DrawGroundPlane(16);
  DrawRobotArm(16);

  Glut.glutSwapBuffers();
}

static void myReshape(int w, int h)
{
  GL.glViewport(0, 0, w, h);

  GL.glMatrixMode(GL.GL_PROJECTION);
  GL.glLoadIdentity();
  GLU.gluPerspective(30, w / h, 0.1, 10);

  GL.glMatrixMode(GL.GL_MODELVIEW);
  GL.glLoadIdentity();
  GL.glTranslatef(1.0f, 0.5f, -7.0f);
}

static void myIdle()
{
  Glut.glutPostRedisplay();
}

static void KeyboardFunc(byte Key, int x, int y)
{
  char c = (char)Key;

  if (c >= '1' && c <= '5')
```

```
        robotControl = c - '1';
      if (Key == 27)
        Application.Exit();         // ESC
    }

    static void MouseFunc(int button, int state, int x, int y)
    {
      newX = x;
      newY = y;

      if (button == Glut.GLUT_LEFT_BUTTON)
        leftButtonDown = !leftButtonDown;
    }

    static void MotionFunc(int x, int y)
    {
      oldX = newX;
      oldY = newY;
      newX = x;
      newY = y;
      float RelX = (newX - oldX) / Glut.glutGet(Glut.GLUT_WINDOW_WIDTH);
      float RelY = (newY - oldY) / Glut.glutGet(Glut.GLUT_WINDOW_HEIGHT);
      if (leftButtonDown)
        switch (robotControl)
        {
          case 0:
            baseTransX += RelX;
            baseTransZ += RelY;
            break;
          case 1:
            baseSpin += RelX * 180;
            break;
          case 2:
            shoulderAng += RelY * -90;
            elbowAng += RelX * 90;
            break;
          case 3:
            wristAng += RelY * -180;
            wristTwistAng += RelX * 180;
            break;
          case 4:
            fingerAng1 += RelY * 90;
            fingerAng2 += RelX * 180;
            break;
        };
    }

    [STAThread]
    public static void Main(string[] args)
    {
      Glut.glutInit();
      Glut.glutInitDisplayMode(Glut.GLUT_DOUBLE | Glut.GLUT_RGB |
Glut.GLUT_DEPTH);
      Glut.glutGetWindow();
      Glut.glutInitWindowSize(512, 512);
      Glut.glutInitWindowPosition(180, 100);
      Glut.glutCreateWindow("The Robot Arm");

      GL.glEnable(GL.GL_COLOR_MATERIAL);
      GL.glEnable(GL.GL_LIGHTING);
      GL.glEnable(GL.GL_LIGHT0);
```

```
            GL.glEnable(GL.GL_DEPTH_TEST);
            GL.glEnable(GL.GL_NORMALIZE);
            GL.glEnable(GL.GL_CULL_FACE);

            Glut.glutDisplayFunc(myDisplay);
            Glut.glutReshapeFunc(myReshape);
            Glut.glutIdleFunc(myIdle);

            Glut.glutKeyboardFunc(KeyboardFunc);
            Glut.glutMouseFunc(MouseFunc);
            Glut.glutMotionFunc(MotionFunc);

            Glut.glutMainLoop();
        }
    }
}
```

You can get a copy of the source code and executables at: http://lenielmacaferi.blogspot.com/2008/02/robot-arm-with-opengl-in-csharp.html

3 APPLICATION

The robot arm that we implemented operates with the keyboard and mouse support. Pressing the keys from 1 to 5 we have the control of the different parts of the robot arm that are: the base, the shoulder, the elbow, the fist and the fingers.

The robot arm's motion occurs according to the following table:

Key	Motion with mouse support
1	Moves the base in the flat surface
2	Rotates the arm in 360º
3	Moves the shoulder and the elbow
4	Moves the fist
5	Moves the fingers

Table 1 - Robot arm keyboard and mouse bindings

The following are some screenshots of the robot arm according to the user's accomplished operation.

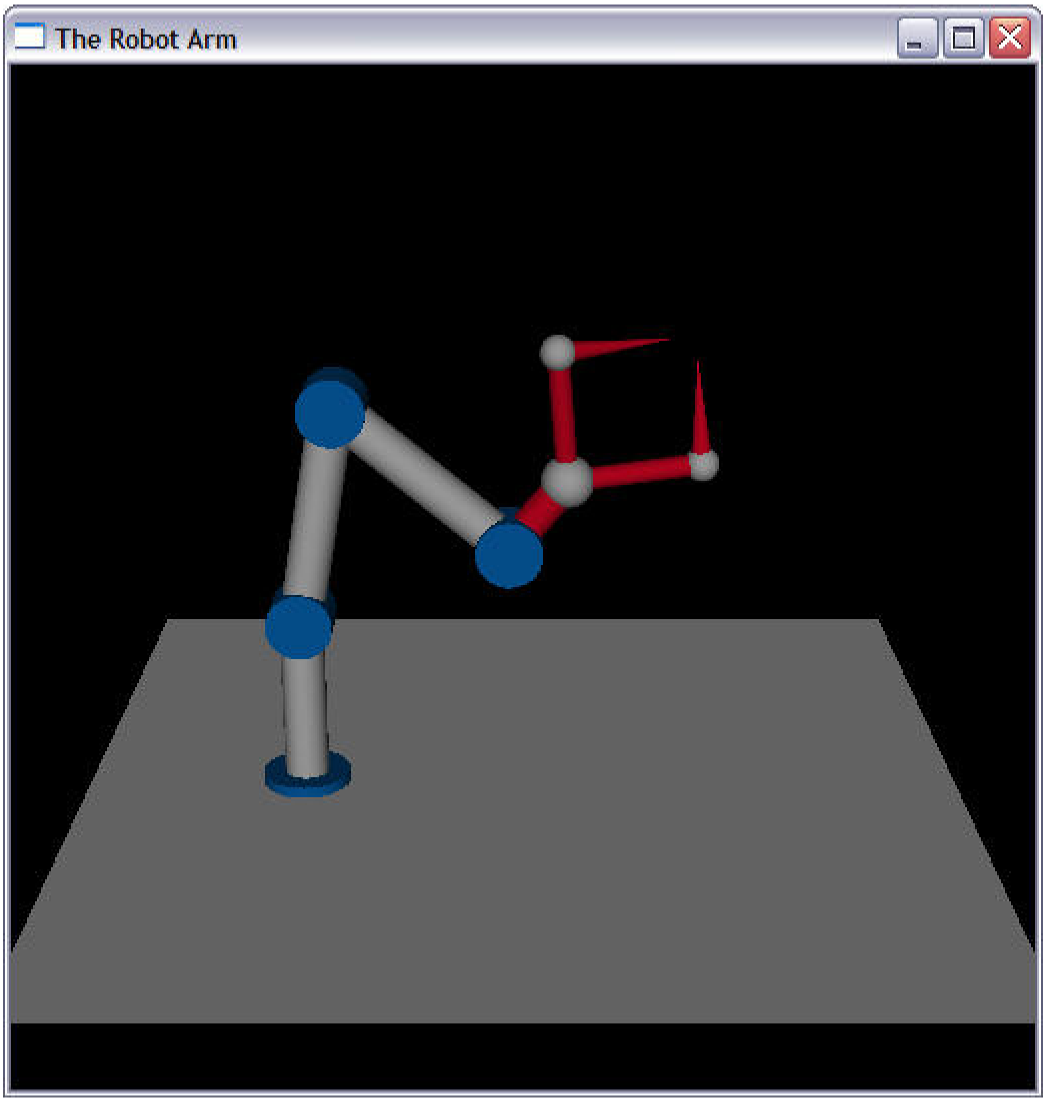

Figure 3 - Initial position of the robot arm

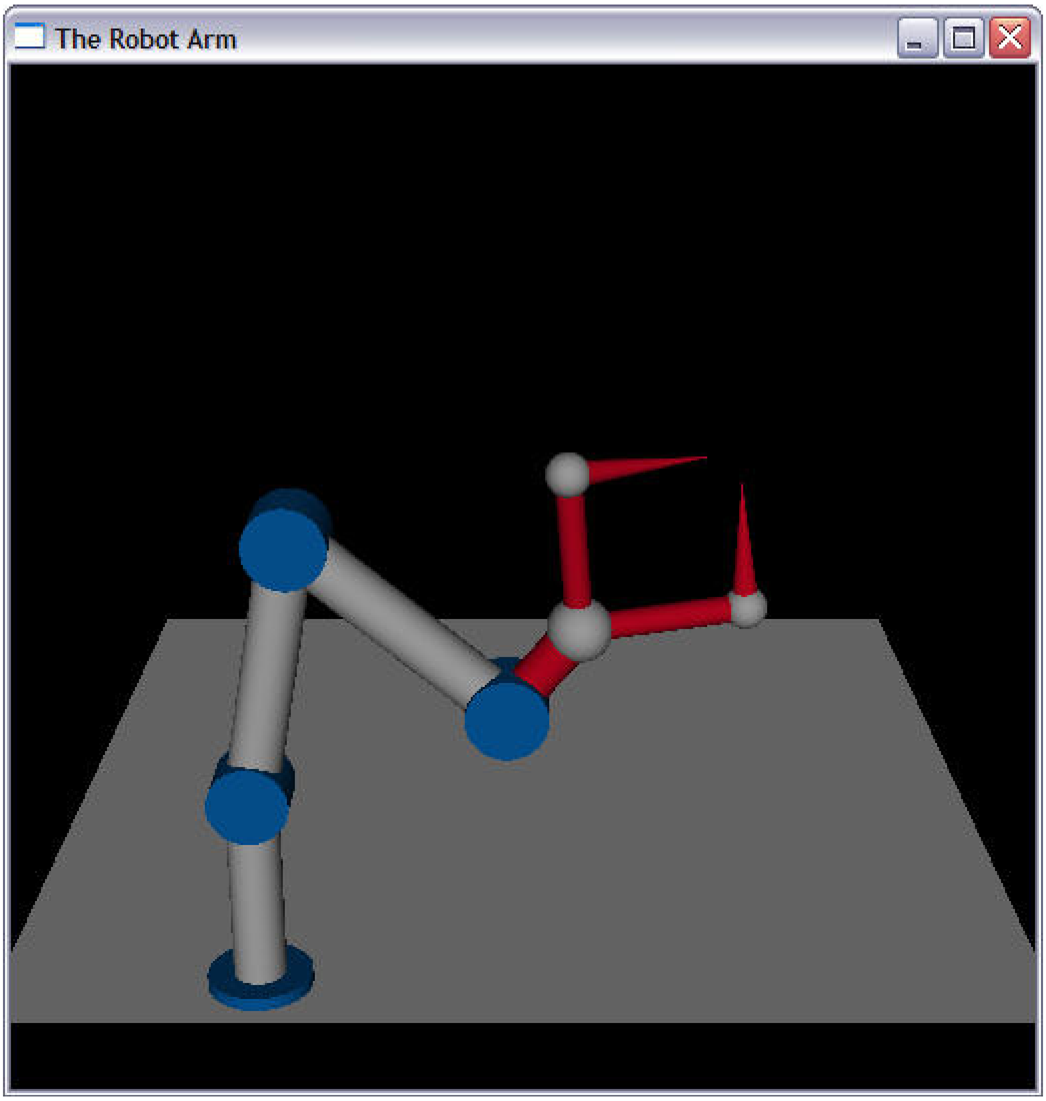

Figure 4 - Motion of the robot arm’s base in the flat surface

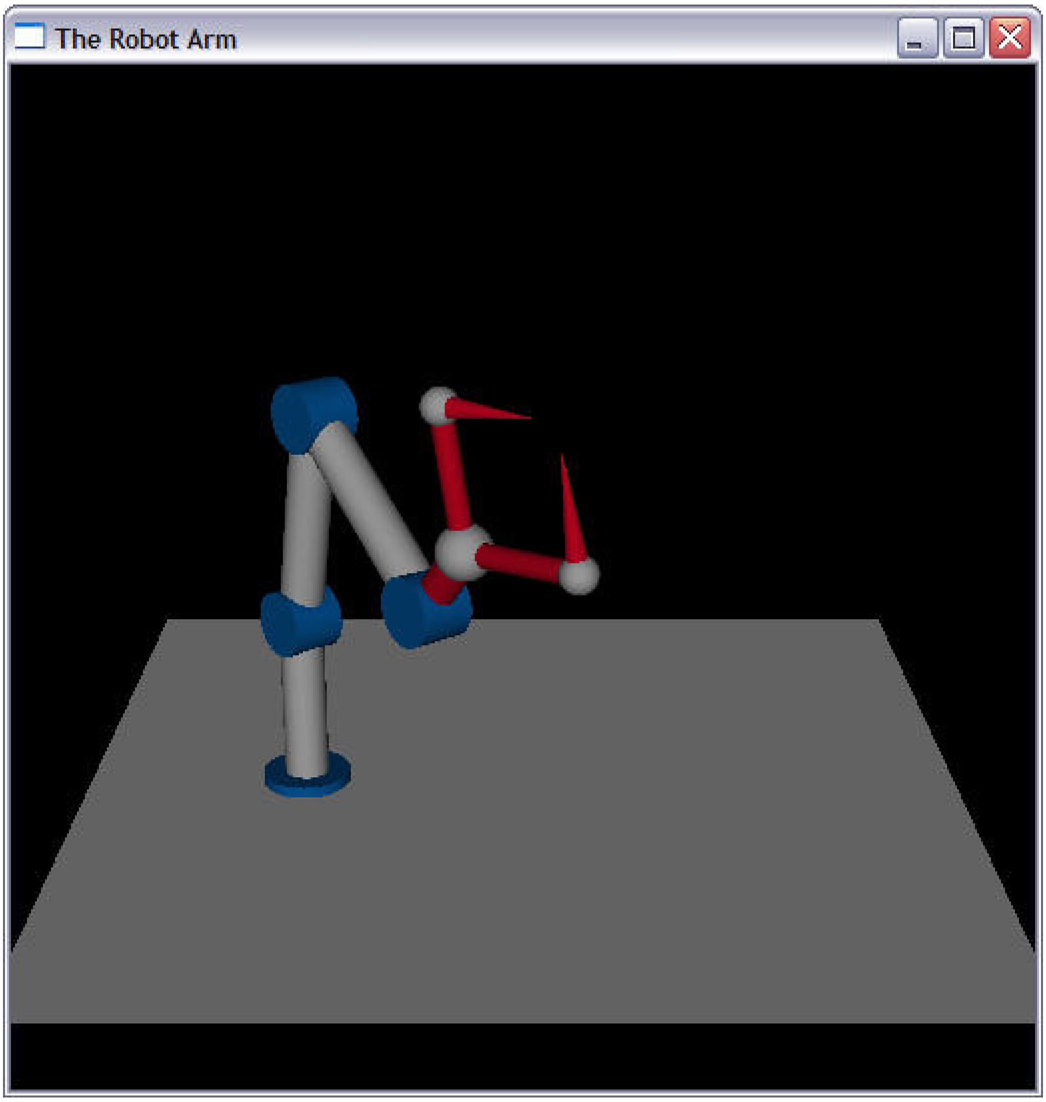

Figure 5 - Robot arm’s rotation in the base pivot

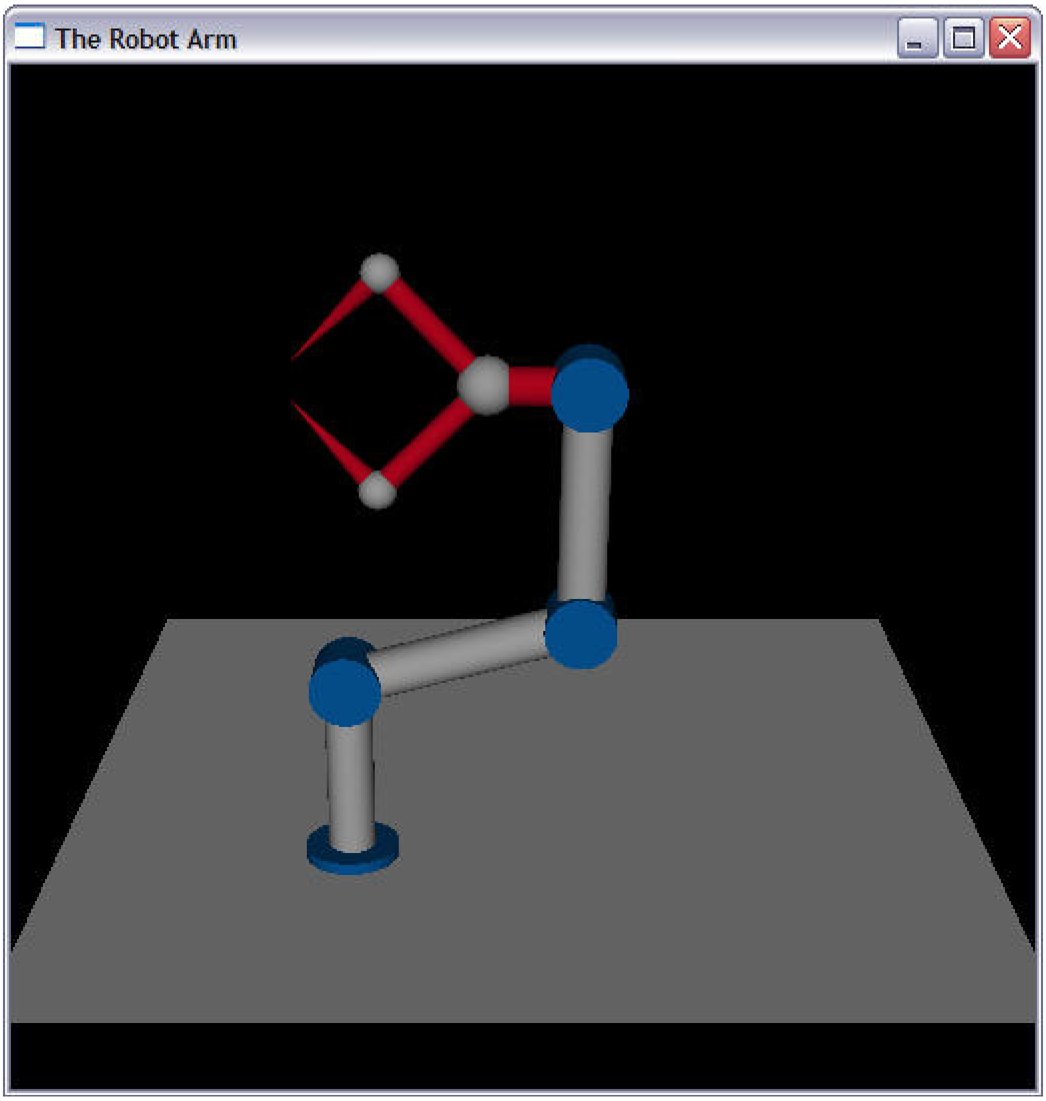

Figure 6 - Robot arm’s shoulder and elbow motion

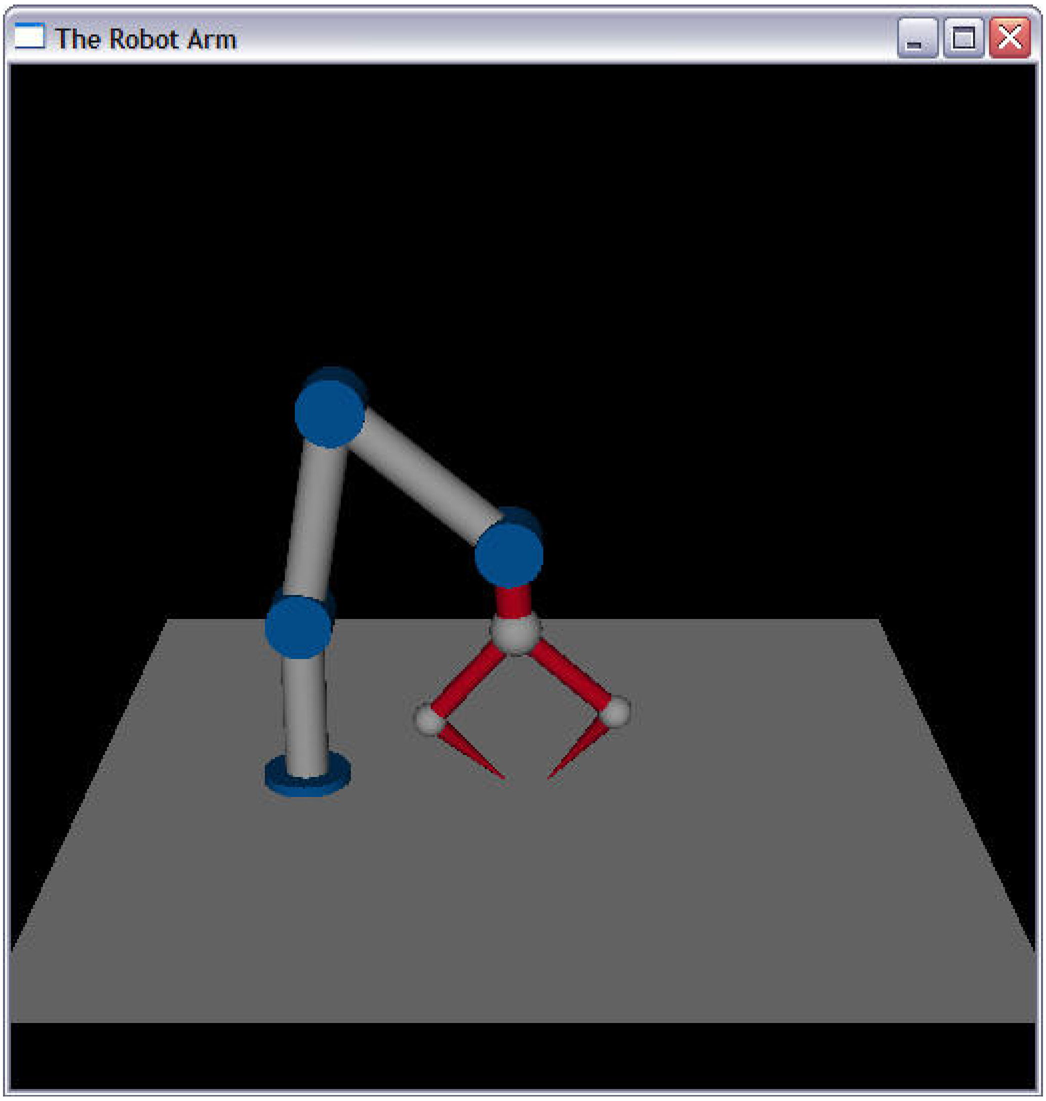

Figure 7 - Robot arm’s fist motion

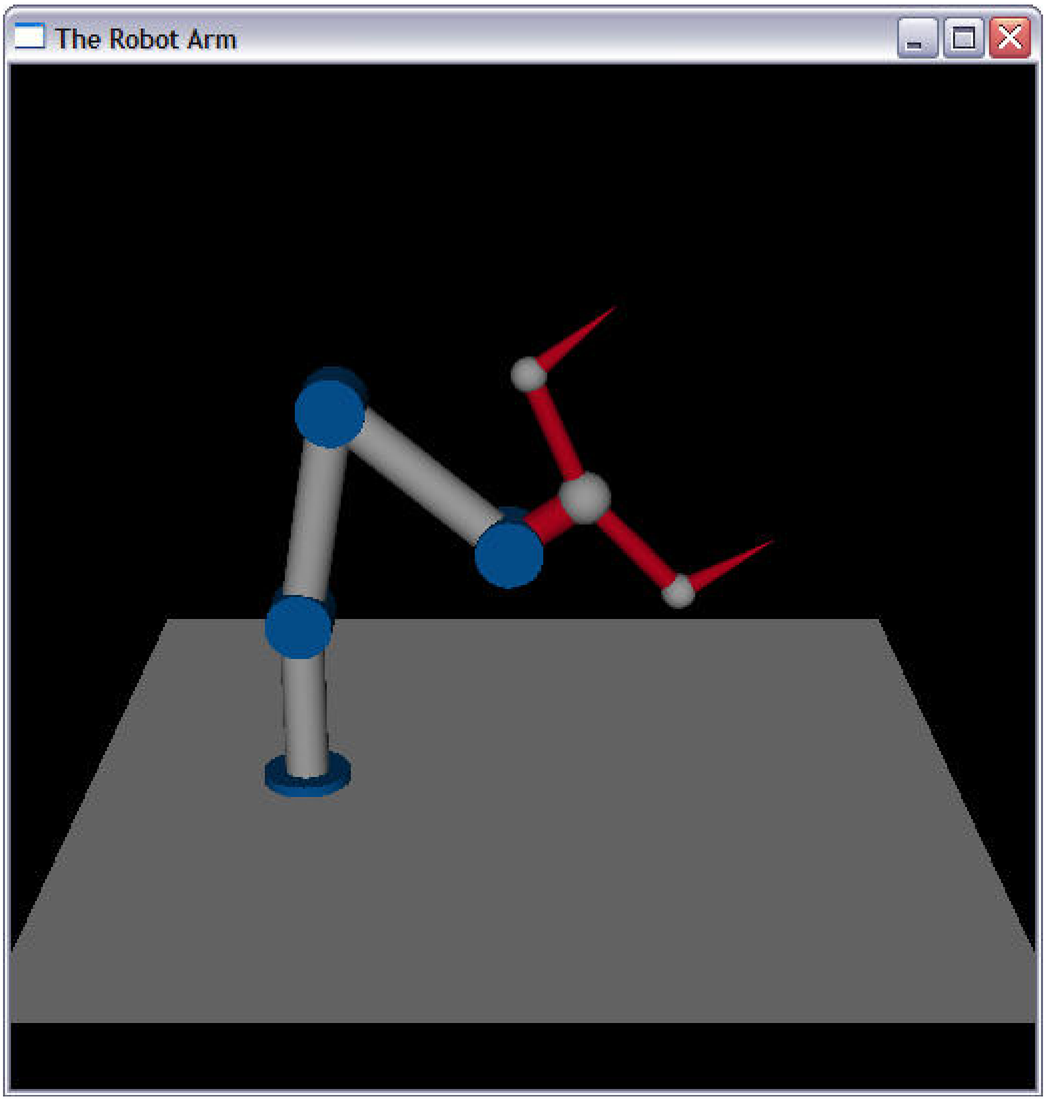

Figure 8 - Robot arm’s fingers motion

4 CONCLUSION

We acquired knowledge about terms, concepts and questions related to the subject what make us capable of working with the commands of the OpenGL language and consequently we are prepared to elaborate 3D graphical projects of the more diverse types.

This paper gave us a satisfactory view of graphics scenes, hierarchical transformations, instantiations, stack of matrices and OpenGL transformations.

The virtual simulation is an excellent way of avoiding unnecessary outgoings and at the same time enables the designer architect to improve his work. All that said, we conclude that the study of such concepts is of great value in the Computer Engineering course.

5 REFERENCES

[1] **Robot arm**. Available at <http://en.wikipedia.org/wiki/Robot_arm>. Accessed May 18, 2007.

[2] **OpenGL**. Available at <http://en.wikipedia.org/wiki/OpenGL>. Accessed May 18, 2007.

[3] Manocha, Dinesh and Salomon, Brian. **The Robot Arm**. COMP-236 - Homework 1, part b, January 08, 2006. Available at <http://www.cs.unc.edu/~dm/UNC/COMP236/Homeworks/hw1b/>. Accessed May 18, 2007.

[4] **The Tao Framework**. Available at <http://taoframework.com/>. Accessed May 18, 2007.

[5] **Industrial robot**. Available at <http://en.wikipedia.org/wiki/Industrial_robot>. Accessed May 18, 2007.

[6] Harris, Tom. **How a Robot Arm works**. Howstuffworks. Available at <http://science.howstuffworks.com/robot2.htm>. Accessed May 18, 2007.