CHAPTER 1

INTRODUCTION

1.1 Overview

1.1.1 Introduction to Graphics

Graphics provides one of the most natural means of communicating with a computer, since our highly developed 2D and 3D pattern-recognition, abilities allow us to perceive and process pictorial data rapidly and efficiently. Interactive computer graphics is the most important means of producing pictures since the invention of photography and television. It has the added advantage that, with the computer, we can make pictures not only of concrete real-world objects but also of abstract, synthetic objects, such as mathematical surfaces and of data that have no inherent geometry, such as survey results.

Using this editor, you can draw and paint using the mouse. It can also perform a host of other functions like drawing lines, circles, polygons and so on. Interactive picture construction techniques such as basic positioning methods, rubber-band methods, dragging and drawing are used. Block operations like cut, copy and paste are supported to edit large areas of the workspace simultaneously. It is user friendly and intuitive to use.

1.1.2 Overview of Computer Graphics

In this era of computing, computer graphics has become one of the most powerful and interesting fact of computing. It all started with display of data on hardcopy and CRT screen. Now computer graphics is about creation, retrieval, manipulation of models and images.

Graphics today is used in many different areas. Graphics provides one of the most natural means of communicating within a computer, since our highly developed 2D and 3D pattern-recognition abilities allow us to perceive and process pictorial data rapidly and effectively. Interactive computer graphics is the most important means of producing pictures since the invention of photography and television. It has the added advantage that, with the computer, we can make pictures not only of concrete real-world objects but also of abstract, synthetic objects, such as mathematical surfaces and of data that have no inherent geometry, such as survey results.

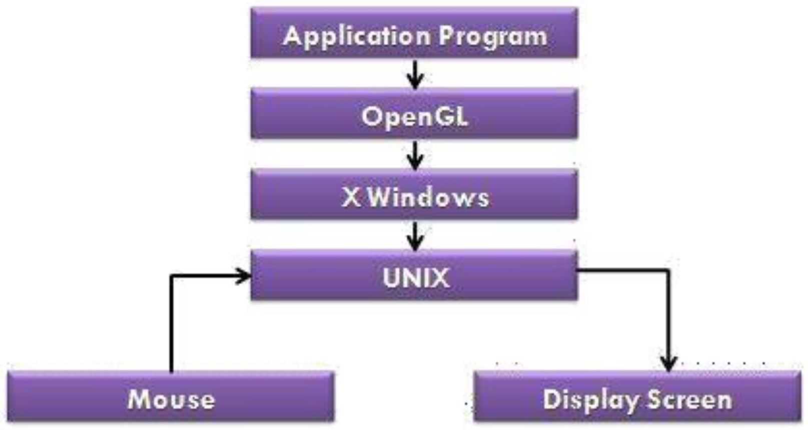


Fig. 1.1.2: Overview of Computer Graphics

OpenGL is an application program interface (API) offering various functions to implement primitives, models and images. This offer functions to create and manipulate render lighting, coloring, viewing the models. OpenGL offers different coordinate system and frames. OpenGL offers translation, rotation and scaling of objects refer fig 1.1.2.

1.2 OpenGL Concept

1.2.1 Interface

OpenGL is an application program interface (API) offering various functions to implement primitives, models and images. This offer functions to create and manipulate render lighting, coloring, viewing the models. OpenGL offers different coordinate system and frames. OpenGL offers translation, rotation and scaling of objects. Functions in the main GL library have names that begins with *gl* and are stored in a library usually referred to as GL. The second is the **OpenGL Utility Library (GLU)**. The library uses only GL functions but contains code for creating common objects and simplifying viewing.

All functions in GLU can be created from the core GL library but application programmers prefer not to write the code repeatedly. The GLU library is available in all OpenGL implementations; functions in the GLU library begin with the letter *glu*.

Rather than using a different library for each system we use a readily available library called OpenGL Utility Toolkit (GLUT), which provides the minimum functionality that should be expected in any modern windowing system.

1.2.2 Overview of OpenGL

OpenGL (open graphics library) is a standard specification defining a cross language cross platform API for writing applications that produce 2D and 3D computer graphics. The interface consists of over 250 different function calls which can be used to draw complex 3D scenes from simple primitives. OpenGL was developed by silicon graphics Inc. (SGI) in 1992 and is widely used in CAD, virtual reality, scientific visualization, information visualization and flight simulation. It is also used in video games, where it competes with direct 3D on Microsoft Windows platforms. OpenGL is managed by the non-profit technology consortium, the khronos group, Inc OpenGL serves two main purposes:

• To hide the complexities of interfacing with different 3D accelerators, by presenting programmer with a single, uniform API

• To hide the differing capabilities of hardware platforms, by requiring that all Implementations support the full openGL, feature set.

- OpenGL (Open Graphics Library) is the interface between a graphics program and graphics hardware. *It is streamlined*. In other words, it provides low-level functionality. For example, all objects are built from points, lines and convex polygons. Higher level objects like cubes are implemented as six four-sided polygons.
- OpenGL supports features like 3-dimensions, lighting, anti-aliasing, shadows, textures, depth effects, etc.
- It is system-independent. It does not assume anything about hardware or operating system and is only concerned with efficiently rendering mathematically described scenes. As a result, it does not provide any windowing capabilities.
- It is a state machine. At any moment during the execution of a program there is a current model transformation.
- It is a rendering pipeline. The rendering pipeline consists of the following steps:
    - Defines objects mathematically.
    - Arranges objects in space relative to a viewpoint.

* Calculates the color of the objects.

- OpenGL has historically been influential on the development of 3D accelerator, promoting a base level of functionality that is now common in consumer level hardware:
  - * Rasterized points, lines and polygons are basic primitives.
  - * A transform and lighting pipeline.
  - * Z buffering.
  - * Texture Mapping.
  - * Alpha

1.2.3 OpenGL Architecture

1) Pipeline Architecture

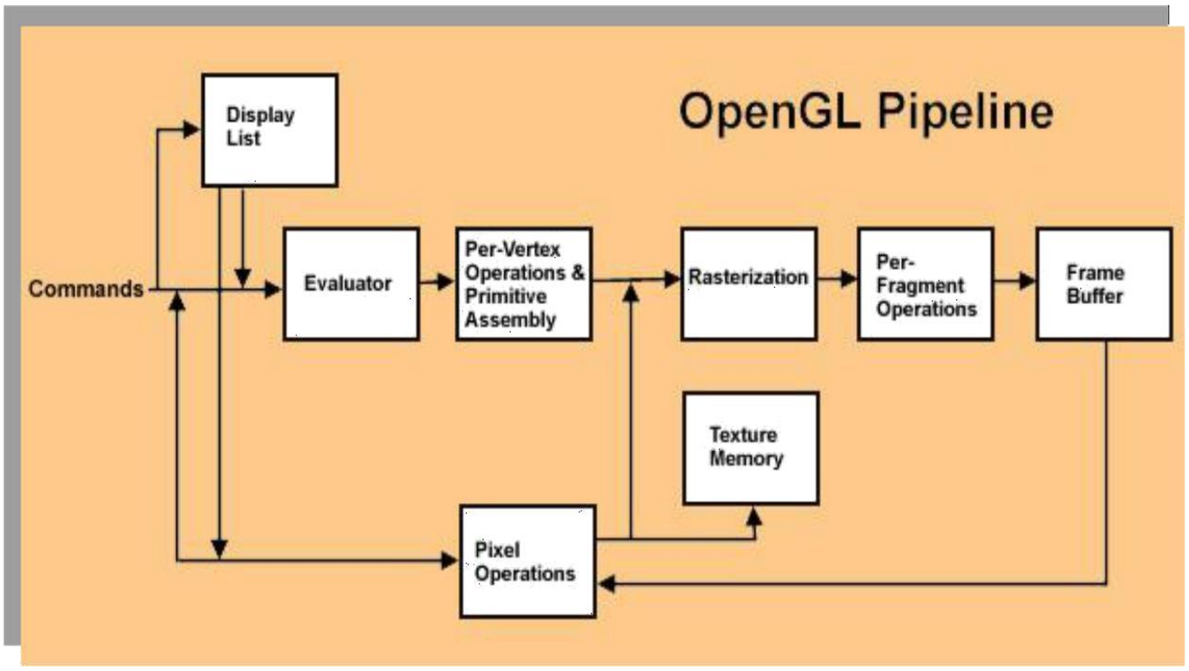


FIG:1.2.3.1 OpenGL Pipeline Architecture

2) OpenGL Engines and Drivers

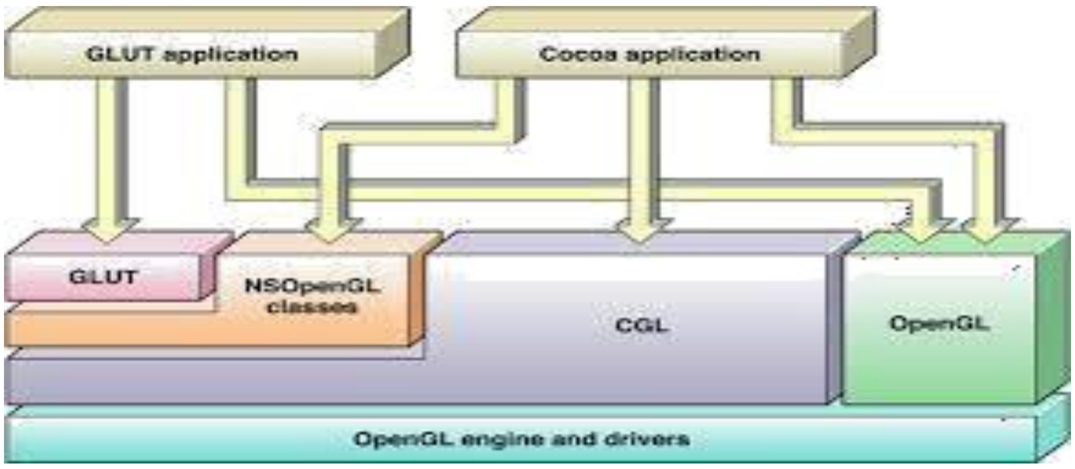
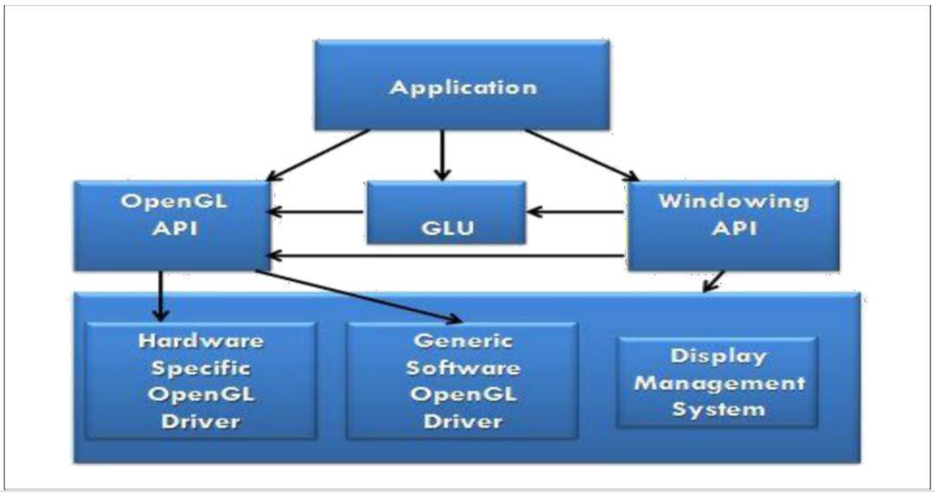


FIG:1.2.3.2 OpenGL Engine and Drivers

3) Application Development API's


Application
OpenGL API
GLU
Windowing API
Hardware Specific OpenGL Driver
Generic Software OpenGL Driver
Display Management System

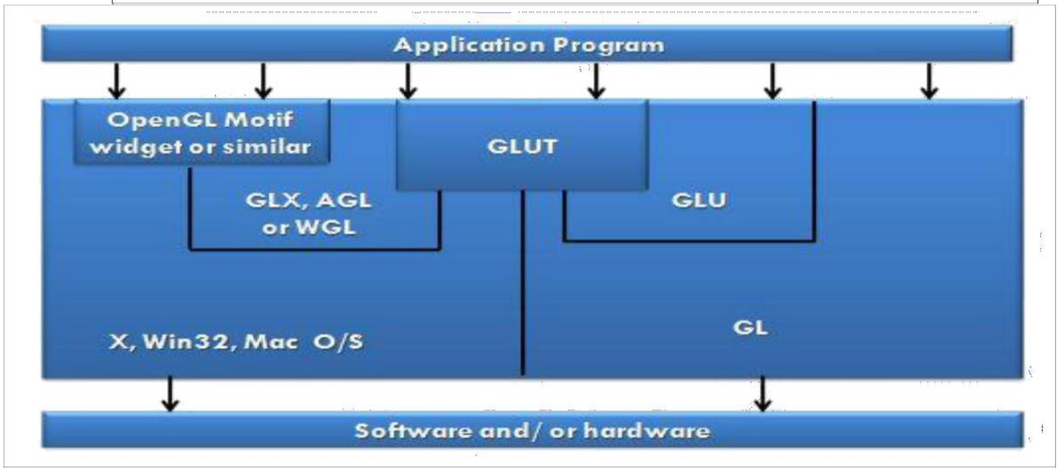


FIG:1.2.3.3 APPLICATIONS DEVELOPMENT(API'S)

The above diagram illustrates the relationship of the various libraries and window system components.

Generally, applications which require more user interface support will use a library designed to support those types of features (i.e., buttons, menu and scroll bars, etc.) such as Motif or the Win32 API.

Prototype applications, or ones which do not require all the bells and whistles of a full GUI, may choose to use GLUT instead because of its simplified programming model and window system independence.

Display Lists:

All data, whether it describes geometry or pixels, can be saved in a display current or later use. (1 alternative to retaining data in a displaylist is processing the data immediately - also known as immediate mode.) When a display list is executed, the retained data is sent from the display list just as if it were sent by the application in immediate mode refer fig 1.2.3.1 and fig 1.2.3.2.

Evaluators:

All geometric primitives are eventually described by vertices. Parametric curves and surfaces may be initially described by control points and polynomial functions called basis functions. Evaluators provide a method to derive the vertices used to represent the surface from the control points. The method is a polynomial mapping, which can produce surface normal, texture coordinates, colors, and spatial coordinate values from the control points refer fig 1.2.3.1.

Per-Vertex Operations

For vertex data, next is the "per-vertex operations" stage, which converts. The vertices into primitives. Some vertex data (for example, spatial coordinates) are transformed by 4 x 4 floating-point matrices. Spatial coordinates are projected from a position in the 3D world to a position on your screen.

If advanced features are enabled, this stage is even busier. If texturing is used, texture coordinates may be generated and transformed here. If lighting is enabled, the lighting calculations are performed using the transformed vertex, surface normal, light source position, material properties, and other lighting information to produce a color value refer fig 1.2.3.1.

Primitive Assembly

Clipping, a major part of primitive assembly, is the elimination of portions of geometry which fall outside a half-space, defined by a plane. Point clipping simply passes or rejects vertices; line or polygon clipping can add additional vertices depending upon how the line or polygon is clipped.

In some cases, this is followed by perspective division, which makes distant geometric objects appear smaller than closer objects. Then view portand depth (z coordinate) operations are applied. If culling is enabled and the primitive is a polygon, it then may be rejected by a culling test. Depending upon the polygon mode, a polygon may be drawn as points or lines. The results or this stage are complete geometric primitives, which are the transformed and clipped vertices with related color, depth, and sometimes texture-coordinate values and guidelines for the rasterization step refer fig 1.2.3.1.

Pixel Operations

While geometric data takes one path through the OpenGL rendering pipeline, pixel data takes a different route. Pixels from an array in system memory are first unpacked from one of a variety of formats into the proper number of components. Next the data is scaled, biased, and processed by a pixel map. The results are clamped and then either written into texture memory or sent to the rasterization step. If pixel data is read from the frame buffer, pixel-transfer operations (scale, bias, mapping, and clamping) are performed. Then these results are packed into an appropriate format and returned to an array in system memory. There are special pixel copy operations to copy data in the frame buffer to other parts of the frame buffer or to the texture memory. A single pass is made through the pixel transfer operations before the data is written to the texture memory or back to the frame buffer refer fig 1.2.3.1.

Texture Assembly

An OpenGL application may wish to apply texture images onto geometric objects to make them look more realistic. If several texture images are used, it's wise to put them into texture objects so that you can easily switch among them. Some OpenGL implementations may have special resources to accelerate texture performance. There may be specialized, high-performance texture memory. If this memory is available, the texture objects may be prioritized to control the use of this limited and valuable resource refer fig 1.2.3.1.

Rasterization

Rasterization is the conversion of both geometric and pixel data into fragments. Each fragment square corresponds to a pixel in the frame buffer. Line and polygon stipples, line width, point size, shading model, and coverage calculations to support ant aliasing are taken into consideration as vertices are connected into lines or the interior pixels are calculated for a filled polygon. Color and depth values are assigned for each fragment square refer fig 1.2.3.1.

1.3 Motivation

In the 1980s, developing software that could function with a wide range of graphics hardware was a real challenge. By the early 1990s, Silicon Graphics (SGI) was a leader in 3D graphics for workstations. SGI's competitors (including Sun Microsystems, Hewlett-Packard and IBM) were also able. In addition, SGI had a large number of software customers; by changing to the OpenGL API, they planned to keep their customers locked onto SGI (and IBM) hardware for a few years while market support for OpenGL matured to bring to market 3D hardware, supported by extensions made to the PHIGS standard. In 1992, SGI led the creation of the OpenGL architectural review board (OpenGL ARB), the group of companies that would maintain and expand the OpenGL specification took for years to come. On 17 December 1997, Microsoft and SGI initiated the Fahrenheit project, which was a joint effort with the goal of unifying the OpenGL and Direct3D interfaces (and adding a scene-graph API too). In 1998 Hewlett-Packard joined the project. It initially showed some promise of bringing order to the world of interactive 3D computer graphics APIs, but on account of financial constraints at SGI, strategic reasons at Microsoft, and general lack of industry support, it was abandoned in1999.

Our work with the interactive 3D Dancing Robot demonstrates help with all three categories:

- A Driving Simulation, helps in visualization of Dance moves
- Making Simulation look more realistic and a dynamic platform for dynamic dancing environment.

Compelling Interactivity: Allows user to select from menu driven options

1.4 Objective

- Practice with 3D transformations in OpenGL
- Implement a hierarchical transformation system
- Practice with specifying animations
- Continue working with the OpenGL APIs
- Implement Dance moves according to music

CHAPTER 2
SYSTEM REQUIREMENTS

2.1 Hardware Requirements

- Laptop or PC
    - i3 Processor Based Computer or higher
    - 8 GB RAM
    - 1 TB Hard Disk
- Android Phone or Tablet
    - 1.2 Quad core Processor or higher
    - 1 GB RAM

2.2 Software Requirements

- Laptop or PC
    - Windows 7 or higher.
    - JDK (Java Development Kit)
    - Android Studio
- Android Phone or Tablet
    - Android v5.0 or Higher

2.3 External Interface Requirements

- User Interface:

The interface for the 2D package requires for the user to have a mouse connected, and the corresponding driver’s software and header files installed. For the convenience of the user, there are menus and sub -menus displayed on the screen.

- Menus:

The Menus consists of various operations related to drawing on the area specified

- Hardware Interface:

The standard output device, as mentioned earlier has been assumed to be a color monitor. It is quite essential for any graphics package to have this, as provision of color options to the user is a must. The mouse, the main input device, has to be functional. A keyboard is also required. Apart from these hardware requirements, there should be sufficient hard disk space and primary memory available for proper working of the package.

- Software Interface:

The editor has been implemented on the DOS platform and mainly requires an appropriate version of the compiler to be installed and functional

CHAPTER 3
DESIGN

3.1 Existing System

The existing system is walking robot, with the help of translate and rotate function 3D robot can be assembled into one form. And by changing coordinates one can move the torso and other body parts.

3.2 Proposed System

To achieve three dimensional effects, open GL software is proposed. It is software which provides a graphical interface. It is a interface between application program and graphics hardware. The advantages are:

1. Open GL is designed as a streamlined
2. It's a hardware independent interface i.e., it can be implemented on many different hardware platforms
3. With OpenGL we can draw a small set of geometric primitives such as points, lines and polygons etc
4. With the help of GLUquadricObj we can construct and assemble cylindrical structure (like torso, arm ,legs) and spherical structures (like joints to join lower and upper parts of body)
5. With the help of system and windows library we can implement songs in this program
6. Combination of Ambient, Diffusion, Specular and Shininess are used for following reasons:

- For a point light source that is inside a scene, we need the following data to model the light:

    * The light's position
    * The light's color
    * The ambient percentages for background light

- Light interacts with the surfaces of an object. To model light we also need basic properties of the surfaces, such as:

    * The surface's color

* The surface's location
* The surface's orientation (its normal vector)
* The surface's shininess exponent

3.3 Program Graphics

Most of the objects in the program, such as buttons, grids or cells, are drawn using OpenGL primitives.

Most of the control buttons used through the program were made as textures in an image editor and exported as .raw files. The class *"Textures"* (in *Textures.h* & *Textures.cpp*) handles the loading of these images into memory and drawing them on the screen using OpenGL's texture related function.

Text is drawn after the back buffer is flushed to the front buffer, by hijacking the window handle via Win32 and writing text onto the window via the Win32ThrowText () function defined in ***Util.h***

CHAPTER 4

IMPLEMENTATION

4.1 Functions

- glColor3f (float, float, float) :- This function will set the current drawing color
- gluOrtho2D (GLdouble left, GLdouble right, GLdouble bottom, GLdouble top):- which defines a two dimensional viewing rectangle in the plane z=0.
- glClear (): -Takes a single argument that is the bitwise OR of several values indicating which buffer is to be cleared
- glClearColor (): -Specifies the red, green, blue, and alpha values used by glClear to clear the color buffers
- GlLoadIdentity (): -the current matrix with the identity matrix
- glMatrixMode(mode): -Sets the current matrix mode, mode can be GL_MODELVIEW, GL_PROJECTION or GL_TEXTURE
- Void glutInit (int *argc, char**argv): -Initializes GLUT, the arguments from main are passed in and can be used by the application
- Void glutInitDisplayMode (unsigned int mode): -Requests a display with the properties in mode. The value of mode is determined by the logical OR of options including the color model and buffering
- Void glutInitWindowSize (int width, int height):- Specifies the initial position of the topleft corner of the window in pixels
- Int glutCreateWindow (char *title): -A window on the display. The string title can be used to label the window. The return value provides references to the window that can be used when there are multiple windows
- Void glutMouseFunc (void *f (int button, int state, int x, int y):-Register the mouse callback function f. The callback function returns the button, the state of button after the event and the position of the mouse relative to the top-left corner of the window.
- Void glutDisplayFunc (void (*func) (void)): -Register the display function func that is executed when the window needs to be redrawn
- glut PostReDisplay (): -which requests that the display callback be executed after the current callback returns

- Void glutMainLoop () Cause the program to enter an event-processing loop. It should be the last statement in main
- glutCreateMenu() and glutAddMenuEntry() these are used for menu options
- PlaySound() This function enables song to be played at background

4.1.2 Algorithm

(1) init objects for every parts of the robot GLUquadricObj*t, /// body *gl, /// glass bot *h, /// head *lua, /// left upper aram *lla, /// left lower aram *rua, /// right upper aram *rla, /// right lower aram *lul, /// left upper leg *rul, /// right upper leg *lll, /// left lower leg *rll; /// right lower leg

(2) Build body parts: Then by using gluCylinder() or gluSpherefunction and these objs (in (1)) to build body parts for using in display function

(3) Combine body parts together We can use glTranslate and glRotate functions to move these body parts of the robot into the right position to build a robot

(4) Init lighting things In light_init() function: Init ambient, diffuse, specular and position arrays. There are two lights in different positions, and we will random choose the arguments of these light arguments to make different color on the robot. random choose arguments for color: srand(time(NULL)); mat_specular[0] = (rand()% 10) * 0.09; mat_specular[1] = (rand()% 10) * 0.06; mat_specular[2] = (rand()% 10) * 0.03; mat_ambient[0] = (rand()% 10) * 0.09; mat_ambient[1] = (rand()% 10) * 0.06; mat_ambient[2] = (rand()% 10) * 0.03; mat_diffuse[0] = (rand()% 10) * 0.09; mat_diffuse[1] = (rand()% 10) * 0.06; mat_diffuse[2] = (rand()% 10) * 0.03;

(5) Arguments of rotation angles: (i) frequency: to control the frequency of the rotation. (ii) max & min to control the scope of rotation

(6) Get rotation angle of each body parts: By using following function: float getAngle(float freq, float min, float max, float time){ return (max-min) * sin(freq * PI * time) + 0.5 * (max + min);} This function takes four arguments: frep, min, max and time. The fourth argument time is the time expired the program started. So the angle can be automatically changed by using this function

(7) Idle function: So according to (6), we can use getAngle() function in idle(void) function, and use glutIdleFunc(idle) to call the idle() , in this way we can use the change of the time to make the robot dancing automatically

(8) Implement multiple dancing style: Besides changing the angle, we can also change the axis these body parts rotate for, including x, y and z axis. We use a flag called flag_dance to control with axis to rotate for. So we set each axis to 0 or 1, then we can have 8 dancing style in total. And we can random choose the value of flag_dance to make it do multiple dancing style automatically

(9) Create a Menu using GlutCreateMenu() and GlutAddMenuEntry()

4.2 Implementation Code

```
#include<GL/glut.h>
#include<conio.h>
#include<Gl/GL.h>
#include<GL/GLU.h>
#include<math.h>
#include<stdlib.h>
#include<stdio.h>
#include<time.h>
#include <windows.h>
#include <mmsystem.h>
#define PI 3.14159
#define TORSO_HEIGHT 5.0
#define TORSO_RADIUS 1.5
#define HEAD_HEIGHT 2.0
#define HEAD_RADIUS 1.5
#define UPPER_ARM_HEIGHT 3.0
#define LOWER_ARM_HEIGHT 2.0
#define UPPER_ARM_RADIUS  0.6
#define LOWER_ARM_RADIUS  0.5
#define LOWER_LEG_HEIGHT 3.0
#define UPPER_LEG_HEIGHT 3.0
#define UPPER_LEG_RADIUS  0.6
#define LOWER_LEG_RADIUS  0.5
#define SHOULDER_RADIUS 0.65
```

```
#define JOINT_RADIUS 0.75
GLfloat theta[11] = { 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 180.0, 0.0, 180.0, 0.0 };
GLfloat theta_freq[11];
GLfloat theta_max[11];
GLfloat theta_min[11];
GLfloat mat_ambient[] = { 0.1, 0.1, 0.1, 1.0 };
GLfloat mat_diffuse[] = { 0.7, 0.7, 0.7, 1.0 };
GLfloat mat_specular[] = { 0.4, 0.4, 0.4, 1.0 };
GLfloat mat_shininess = { 100.0 };
//int flag_dance = 0;
GLUquadricObj* t, /// body
* gl, /// glass bot
* h,  /// head
* lua, /// left  upper aram
* lla, /// left  lower aram
* rua, /// right upper aram
* rla, /// right lower aram
* lul, /// left  upper leg
* rul, /// right upper leg
* lll, /// left  lower leg
* rll; /// right lower leg
int where_to_rotate=0;
void body() {
   glPushMatrix();
   glRotatef(-90.0, 1.0, 0.0, 0.0);
   ///(*obj, base, top, height, slices, stacks)
   gluCylinder(t, TORSO_RADIUS, TORSO_RADIUS * 1.5, TORSO_HEIGHT, 10, 10);
   glPopMatrix();
}
void head() {
   glPushMatrix();
   glTranslatef(0.0, 0.5 * HEAD_HEIGHT, 0.0);
   glScalef(HEAD_RADIUS, HEAD_HEIGHT, HEAD_RADIUS);
```

```
    gluSphere(h, 1.0, 10, 10);
    glPopMatrix();
}
void glass_bot() {
    glPushMatrix();
    glTranslatef(0.0, 0.5 * HEAD_HEIGHT, 0.075);
    glRotatef(-90.0, 1.0, 0.0, 0.0);
    gluCylinder(gl, HEAD_RADIUS, HEAD_RADIUS, HEAD_HEIGHT / 2, 10, 10);
    glPopMatrix();
}
void waist() {
    glPushMatrix();
    gluCylinder(gl, 1.2 * TORSO_RADIUS, 1.2 * TORSO_RADIUS, TORSO_RADIUS / 2, 10,
10);
    glPopMatrix();
}
void shoulder_joints()
{
    glPushMatrix();
    glScalef(SHOULDER_RADIUS, SHOULDER_RADIUS, SHOULDER_RADIUS);
    gluSphere(h, 1.0, 10, 10);
    glPopMatrix();
}
void elbow_joints() {
    glPushMatrix();
    glScalef(SHOULDER_RADIUS / 1.2, SHOULDER_RADIUS / 1.2, SHOULDER_RADIUS
/ 1.2);
    gluSphere(h, 1.0, 10, 10);
    glPopMatrix();
}
void palms() {
    glPushMatrix();
    glScalef(SHOULDER_RADIUS / 1.3, SHOULDER_RADIUS / 1.3, SHOULDER_RADIUS
/ 1.3);
    gluSphere(h, 1.0, 10, 10);
    glPopMatrix();
}
void leg_joints() {
    glPushMatrix();
    glScalef(JOINT_RADIUS, JOINT_RADIUS, JOINT_RADIUS);
    gluSphere(h, 1.0, 10, 10);
    glPopMatrix();
}
void knee_joints()
{
    glPushMatrix();
    glScalef(JOINT_RADIUS, JOINT_RADIUS, JOINT_RADIUS);
    gluSphere(h, 1.0, 10, 10);
    glPopMatrix();
}
void left_upper_arm() {
    gluCylinder(lua, UPPER_ARM_RADIUS * 1.2, UPPER_ARM_RADIUS,
UPPER_ARM_HEIGHT, 10, 10);
}
void left_lower_arm() {
    gluCylinder(lla, LOWER_ARM_RADIUS * 1.1, LOWER_ARM_RADIUS,
LOWER_ARM_HEIGHT, 10, 10);
}
void right_upper_arm() {
    glPushMatrix();
    glRotatef(-90.0, 1.0, 0.0, 0.0);
    gluCylinder(rua, UPPER_ARM_RADIUS * 1.2, UPPER_ARM_RADIUS,
UPPER_ARM_HEIGHT, 10, 10);
    glPopMatrix();
}
```

```
void right_lower_arm() {
    glPushMatrix();
    glRotatef(-90.0, 1.0, 0.0, 0.0);
    gluCylinder(rla, LOWER_ARM_RADIUS * 1.1, LOWER_ARM_RADIUS,
LOWER_ARM_HEIGHT, 10, 10);
    glPopMatrix();
}

void left_upper_leg() {
    glColor3f(1.0, 0.0, 1.0);
    glPushMatrix();
    glRotatef(-120.0, 1.0, 0.0, 0.0);
    gluCylinder(lul, UPPER_LEG_RADIUS * 1.2, UPPER_LEG_RADIUS,
UPPER_LEG_HEIGHT, 10, 10);
    glPopMatrix();
}
void left_lower_leg()
{
    glColor3f(1.0, 0.0, 0.0);
    glPushMatrix();
    glTranslatef(0.0, -0.25, -UPPER_LEG_HEIGHT / 2);
    glRotatef(-70.0, 1.0, 0.0, 0.0);
    gluCylinder(lll, LOWER_LEG_RADIUS, LOWER_LEG_RADIUS * 1.5,
LOWER_LEG_HEIGHT, 10, 10);
    glPopMatrix();
}
void right_upper_leg()
{
    glColor3f(1.0f, 0.0f, 1.0f);
    glPushMatrix();
    glRotatef(-120.0, 1.0, 0.0, 0.0);
    gluCylinder(rul, UPPER_LEG_RADIUS * 1.2, UPPER_LEG_RADIUS,
UPPER_LEG_HEIGHT, 10, 10);
```

```
    glPopMatrix();
}
void right_lower_leg()
{
    glColor3f(1.0, 0.0, 0.0);
    glPushMatrix();
    glTranslatef(0.0, -0.25, -UPPER_LEG_HEIGHT / 2);
    glRotatef(-70.0, 1.0, 0.0, 0.0);
    gluCylinder(rll, LOWER_LEG_RADIUS, LOWER_LEG_RADIUS * 1.5,
LOWER_LEG_HEIGHT, 10, 10);
    glPopMatrix();
}
void feets() {
    glPushMatrix();
    glScalef(SHOULDER_RADIUS, SHOULDER_RADIUS, SHOULDER_RADIUS);
    gluSphere(h, 1.2, 10, 10);
    glPopMatrix();
}

void display(void) {


    GLfloat rot_x = 0.0, rot_y = 0.0, rot_z = 0.0;
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    glColor3f(0.0, 0.0, 0.0);
    glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
    glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient);
    glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
    glMaterialf(GL_FRONT, GL_SHININESS, mat_shininess);
    /// set camera
    gluLookAt(0.0, 0.0, 10.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
    glTranslatef(0.0, 0.0, 10);
    /// ************************ body************************
    glTranslatef(0.0, -2.0, 0.0);
    glRotatef(theta[0], 0.0, 1.0, 0.0);
    body();
    /// ************************** waist  *************************
    glPushMatrix();
    glTranslatef(0.0, 1.0, 0.075);
    glRotatef(-90.0, 1.0, 0.0, 0.0);
    waist();
    glPopMatrix();
    /// ************************** head  *************************
    glPushMatrix();
    glTranslatef(0.0, TORSO_HEIGHT + 0.5 * HEAD_HEIGHT, 0.0);
    glRotatef(theta[1], 1.0, 0.0, 0.0);
    glRotatef(theta[2], 0.0, 1.0, 0.0);
    glTranslatef(0.0, -0.5 * HEAD_HEIGHT, 0.0);
    head();
    glass_bot();
    glPopMatrix();
    /// ************************** left and right shoulder joints
*************************
    glPushMatrix();
    glTranslatef(1.5 * TORSO_RADIUS, 0.9 * TORSO_HEIGHT, 0.0);
    shoulder_joints();
    glTranslatef(-3.0 * TORSO_RADIUS, 0.0, 0.0);
    shoulder_joints();
    glPopMatrix();
    /// ************************** left and right leg  joints
*************************
    glPushMatrix();
    glTranslatef(TORSO_RADIUS, 0.0, 0.0);
    leg_joints();
    glTranslatef(-2 * TORSO_RADIUS, 0.0, 0.0);
    leg_joints();
    glPopMatrix();
    /// ************************** whole left arm *************************
    glPushMatrix();
    glTranslatef(-(TORSO_RADIUS + UPPER_ARM_RADIUS), 0.9 * TORSO_HEIGHT,
0.0);
    glRotatef(theta[3], rot_x, rot_y, rot_z);
    left_upper_arm();
    glTranslatef(0.0, 0.0, UPPER_ARM_HEIGHT);
    elbow_joints();
    glRotatef(theta[4], rot_x, rot_y, rot_z);
    left_lower_arm();
    glTranslatef(0.0, 0.0, LOWER_ARM_HEIGHT);
    palms();
    glPopMatrix();
    /// ************************** whole right arm *************************
    glPushMatrix();
    glTranslatef(TORSO_RADIUS + UPPER_ARM_RADIUS, 0.9 * TORSO_HEIGHT, 0.0);
    glRotatef(theta[5], rot_x, rot_y, rot_z);
    right_upper_arm();
    glTranslatef(0.0, UPPER_ARM_HEIGHT, 0.0);
    elbow_joints();
    glRotatef(theta[6], rot_x, rot_y, rot_z);
    glColor3f(1.0, 1.0, 1.0);
    right_lower_arm();
    glTranslatef(0.0, LOWER_ARM_HEIGHT, 0.0);
    palms();
    glPopMatrix();
    /// ************************** whole left leg *************************
    glPushMatrix();
    glTranslatef(-(TORSO_RADIUS), 0.1 * UPPER_LEG_HEIGHT, 0.0);
    glRotatef(theta[7], rot_x, 0.0, 0.0);
    left_upper_leg();
    glTranslatef(0.0, UPPER_LEG_HEIGHT, -1.5);
    knee_joints();
    glTranslatef(0.0, 0.0, 1.5);
    glRotatef(theta[8], 1.0, 0.0, 0.0);
    //    glRotatef(theta[8], rot_x, rot_y, rot_z);
    left_lower_leg();
    glTranslatef(0.0, LOWER_LEG_HEIGHT, 0.0);
    feets();
    glPopMatrix();
    /// ************************** whole right leg *************************///
    glPushMatrix();
    glTranslatef((TORSO_RADIUS), 0.1 * UPPER_LEG_HEIGHT, 0.0);
    glRotatef(theta[9], rot_x, 0.0, 0.0);
    right_upper_leg();
    glTranslatef(0.0, UPPER_LEG_HEIGHT, -1.5);
    knee_joints();
    glTranslatef(0.0, 0.0, 1.5);
    glRotatef(theta[10], 1.0, 0.0, 0.0);
    right_lower_leg();
    glTranslatef(0.0, LOWER_LEG_HEIGHT, 0.0);
    feets();
    glPopMatrix();
    /// ************************** end *************************///
    glFlush();
    glutSwapBuffers();
    /// glDisable(GL_TEXTURE_2D);
}
/// Caluculate angle
float getAngle(float freq, float min, float max, float time) {
    return (max - min) * sin(freq * PI * time) + 0.5 * (max + min);
}

static void idle(void) {
    GLfloat seconds = glutGet(GLUT_ELAPSED_TIME) / 1000.0;
```

```
srand(time(NULL));
mat_specular[0] = (rand() % 10) * 0.09;
mat_specular[1] = (rand() % 10) * 0.06;
mat_specular[2] = (rand() % 10) * 0.03;
mat_ambient[0] = (rand() % 10) * 0.09;
mat_ambient[1] = (rand() % 10) * 0.06;
mat_ambient[2] = (rand() % 10) * 0.03;
mat_diffuse[0] = (rand() % 10) * 0.09;
mat_diffuse[1] = (rand() % 10) * 0.06;
mat_diffuse[2] = (rand() % 10) * 0.03;

/// moving right arm
if (where_to_rotate == 1) {

    theta[6] = getAngle(theta_freq[6], theta_min[6], theta_max[6], seconds);
    theta[5] = getAngle(theta_freq[5], theta_min[5], theta_max[5], seconds);
}
/// moving left arm
else if (where_to_rotate == 2) {
    theta[3] = getAngle(theta_freq[3], theta_min[3], theta_max[3], seconds);
    theta[4] = getAngle(theta_freq[4], theta_min[4], theta_max[4], seconds);
}
/// moving right leg
else if (where_to_rotate == 3) {
    theta[9] = getAngle(theta_freq[9], theta_min[9], theta_max[9], seconds);
    theta[10] = getAngle(theta_freq[10], theta_min[10], theta_max[10], seconds);
}
/// moving left leg
else if (where_to_rotate == 4) {
    theta[7] = getAngle(theta_freq[7], theta_min[7], theta_max[7], seconds);
    theta[8] = getAngle(theta_freq[8], theta_min[8], theta_max[8], seconds);
}
/// moving body
```

```
else if (where_to_rotate == 5) {
    theta[0] = getAngle(theta_freq[0], theta_min[0], theta_max[0], seconds);
}
/// moving head
else if (where_to_rotate == 6) {
    theta[1] = getAngle(theta_freq[1], theta_min[1], theta_max[1], seconds);
}
else if (where_to_rotate == 7) {
    theta[2] = getAngle(theta_freq[2], theta_min[2], theta_max[2], seconds);
}
//moving body with left arm
else if (where_to_rotate == 8) {
    theta[0] = getAngle(theta_freq[0], theta_min[0], theta_max[0], seconds);
    theta[3] = getAngle(theta_freq[3], theta_min[3], theta_max[3], seconds);
    theta[4] = getAngle(theta_freq[4], theta_min[4], theta_max[4], seconds);
}
//moving body with right arm
else if (where_to_rotate == 9) {
    theta[0] = getAngle(theta_freq[0], theta_min[0], theta_max[0], seconds);
    theta[6] = getAngle(theta_freq[6], theta_min[6], theta_max[6], seconds);
    theta[5] = getAngle(theta_freq[5], theta_min[5], theta_max[5], seconds);
}
//moving body with left leg
else if (where_to_rotate == 10) {
    theta[0] = getAngle(theta_freq[0], theta_min[0], theta_max[0], seconds);
    theta[7] = getAngle(theta_freq[7], theta_min[7], theta_max[7], seconds);
    theta[8] = getAngle(theta_freq[8], theta_min[8], theta_max[8], seconds);
}
//moving body with right leg
else if (where_to_rotate == 11) {
    theta[0] = getAngle(theta_freq[0], theta_min[0], theta_max[0], seconds);
    theta[9] = getAngle(theta_freq[9], theta_min[9], theta_max[9], seconds);
    theta[10] = getAngle(theta_freq[10], theta_min[10], theta_max[10], seconds);
```

```
}
//moving body with head
else if (where_to_rotate == 12) {
    theta[0] = getAngle(theta_freq[0], theta_min[0], theta_max[0], seconds);
    theta[1] = getAngle(theta_freq[1], theta_min[1], theta_max[1], seconds);
    theta[2] = getAngle(theta_freq[2], theta_min[2], theta_max[2], seconds);
}
//moving body with both arms
else if (where_to_rotate == 13) {
    theta[0] = getAngle(theta_freq[0], theta_min[0], theta_max[0], seconds);
    theta[3] = getAngle(theta_freq[3], theta_min[3], theta_max[3], seconds);
    theta[4] = getAngle(theta_freq[4], theta_min[4], theta_max[4], seconds);
    theta[0] = getAngle(theta_freq[0], theta_min[0], theta_max[0], seconds);
    theta[6] = getAngle(theta_freq[6], theta_min[6], theta_max[6], seconds);
    theta[5] = getAngle(theta_freq[5], theta_min[5], theta_max[5], seconds);
}
//moving body with both legs
else if (where_to_rotate == 14) {
    theta[0] = getAngle(theta_freq[0], theta_min[0], theta_max[0], seconds);
    theta[7] = getAngle(theta_freq[7], theta_min[7], theta_max[7], seconds);
    theta[8] = getAngle(theta_freq[8], theta_min[8], theta_max[8], seconds);
    theta[9] = getAngle(theta_freq[9], theta_min[9], theta_max[9], seconds);
    theta[10] = getAngle(theta_freq[10], theta_min[10], theta_max[10], seconds);
}
//moving body with left arm,left leg and head
else if (where_to_rotate == 15) {
    theta[0] = getAngle(theta_freq[0], theta_min[0], theta_max[0], seconds);
    theta[3] = getAngle(theta_freq[3], theta_min[3], theta_max[3], seconds);
    theta[4] = getAngle(theta_freq[4], theta_min[4], theta_max[4], seconds);
    theta[7] = getAngle(theta_freq[7], theta_min[7], theta_max[7], seconds);
    theta[8] = getAngle(theta_freq[8], theta_min[8], theta_max[8], seconds);
    theta[1] = getAngle(theta_freq[1], theta_min[1], theta_max[1], seconds);
    theta[2] = getAngle(theta_freq[2], theta_min[2], theta_max[2], seconds);
```

```
}
//moving body with right arm,right leg and head
else if (where_to_rotate == 16) {
    theta[0] = getAngle(theta_freq[0], theta_min[0], theta_max[0], seconds);
    theta[6] = getAngle(theta_freq[6], theta_min[6], theta_max[6], seconds);
    theta[5] = getAngle(theta_freq[5], theta_min[5], theta_max[5], seconds);
    theta[9] = getAngle(theta_freq[9], theta_min[9], theta_max[9], seconds);
    theta[10] = getAngle(theta_freq[10], theta_min[10], theta_max[10], seconds);
    theta[1] = getAngle(theta_freq[1], theta_min[1], theta_max[1], seconds);
    theta[2] = getAngle(theta_freq[2], theta_min[2], theta_max[2], seconds);
}
//moving body with left arm,right leg and head
else if (where_to_rotate == 17) {
    theta[0] = getAngle(theta_freq[0], theta_min[0], theta_max[0], seconds);
    theta[3] = getAngle(theta_freq[3], theta_min[3], theta_max[3], seconds);
    theta[4] = getAngle(theta_freq[4], theta_min[4], theta_max[4], seconds);
    theta[9] = getAngle(theta_freq[9], theta_min[9], theta_max[9], seconds);
    theta[10] = getAngle(theta_freq[10], theta_min[10], theta_max[10], seconds);
    theta[1] = getAngle(theta_freq[1], theta_min[1], theta_max[1], seconds);
    theta[2] = getAngle(theta_freq[2], theta_min[2], theta_max[2], seconds);
}
else if (where_to_rotate == 18) {
    theta[0] = getAngle(theta_freq[0], theta_min[0], theta_max[0], seconds);
    theta[6] = getAngle(theta_freq[6], theta_min[6], theta_max[6], seconds);
    theta[5] = getAngle(theta_freq[5], theta_min[5], theta_max[5], seconds);
    theta[7] = getAngle(theta_freq[7], theta_min[7], theta_max[7], seconds);
    theta[8] = getAngle(theta_freq[8], theta_min[8], theta_max[8], seconds);
    theta[1] = getAngle(theta_freq[1], theta_min[1], theta_max[1], seconds);
    theta[2] = getAngle(theta_freq[2], theta_min[2], theta_max[2], seconds);
}


glEnable(GL_LIGHT0);
```

```
    glEnable(GL_LIGHT3);
    glutPostRedisplay();
}
/// My reshape
void reshape(int w, int h) {
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    if (w <= h)
        glOrtho(-10.0, 10.0, -10.0 * (GLfloat)h / (GLfloat)w,
            10.0 * (GLfloat)h / (GLfloat)w, -10.0, 10.0);
    else
        glOrtho(-10.0 * (GLfloat)w / (GLfloat)h,
            10.0 * (GLfloat)w / (GLfloat)h, -10.0, 10.0, -10.0, 10.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}
/// init light things
void light_init() {
    /// light 1
    GLfloat light_ambient1[] = { 0.0, 0.0, 0.0, 1.0 };
    GLfloat light_diffuse1[] = { 1.0, 1.0, 1.0, 1.0 };
    GLfloat light_specular1[] = { 1.0, 1.0, 1.0, 1.0 };
    GLfloat light_position1[] = { 10.0, 10.0, 10.0, 0.0 };
    glLightfv(GL_LIGHT0, GL_POSITION, light_position1);
    glLightfv(GL_LIGHT0, GL_AMBIENT, light_ambient1);
    glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse1);
    glLightfv(GL_LIGHT0, GL_SPECULAR, light_specular1);
    ///light 2
    GLfloat light_ambient2[] = { 0.0, 0.0, 0.0, 1.0 };
    GLfloat light_diffuse2[] = { 1.0, 1.0, 1.0, 1.0 };
    GLfloat light_specular2[] = { 1.0, 1.0, 1.0, 1.0 };
    GLfloat light_position2[] = { -10.0, 10.0, 10.0, 0.0 };
    glLightfv(GL_LIGHT3, GL_POSITION, light_position2);
    glLightfv(GL_LIGHT3, GL_AMBIENT, light_ambient2);
    glLightfv(GL_LIGHT3, GL_DIFFUSE, light_diffuse2);
    glLightfv(GL_LIGHT3, GL_SPECULAR, light_specular2);
    glShadeModel(GL_SMOOTH);
    glEnable(GL_LIGHTING);
    glDepthFunc(GL_LEQUAL);
    glEnable(GL_DEPTH_TEST);
    glClearColor(0.0, 0.0, 0.0, 1.0);
    glColor3f(0.0, 0.0, 0.0);
}
/// init rotation angles
void rotate_init() {
    /// Setting the min, max and frequency for body parts
    for (int i = 0; i < 11; i++) {
        theta_min[i] = 0.0;
        theta_max[i] = 0.0;
        theta_freq[i] = 0.0;
    }
    /// assigning values for theta
    theta_freq[0] = 0.8; theta_max[0] = 30;    theta_min[0] = -30;
    theta_freq[1] = 2.0; theta_max[1] = 15.0;  theta_min[1] = -5.0;
    theta_freq[2] = 2.0; theta_max[2] = 20.0; theta_min[2] = 0.0;
    theta_freq[3] = 2.0; theta_max[3] = -100.0; theta_min[3] = 0.0;
    theta_freq[5] = 2.0; theta_max[5] = 100.0;  theta_min[5] = 0.0;
    theta_freq[4] = 2.0; theta_max[4] = -35.0; theta_min[4] = -10.0;
    theta_freq[6] = 2.0; theta_max[6] = -35.0; theta_min[6] = -10.0;
    theta_freq[7] = 2.0; theta_max[7] = 200.0; theta_min[7] = 160.0;
    theta_freq[9] = 2.0; theta_max[9] = 160.0; theta_min[9] = 200.0;
    theta_freq[8] = 2.0; theta_max[8] = -35.0; theta_min[8] = -10.0;
    theta_freq[10] = 2.0; theta_max[10] = -35.0; theta_min[10] = -10.0;

}
```

```
/// init quadrics style : GLU_FILL
void style_init() {
    /// allocate quadrics with filled drawing style: GLU_FILL
    h = gluNewQuadric(); gluQuadricDrawStyle(h, GLU_FILL);
    t = gluNewQuadric(); gluQuadricDrawStyle(t, GLU_FILL);
    gl = gluNewQuadric(); gluQuadricDrawStyle(gl, GLU_FILL);
    lua = gluNewQuadric(); gluQuadricDrawStyle(lua, GLU_FILL);
    lla = gluNewQuadric(); gluQuadricDrawStyle(lla, GLU_FILL);
    rua = gluNewQuadric(); gluQuadricDrawStyle(rua, GLU_FILL);
    rla = gluNewQuadric(); gluQuadricDrawStyle(rla, GLU_FILL);
    lul = gluNewQuadric(); gluQuadricDrawStyle(lul, GLU_FILL);
    lll = gluNewQuadric(); gluQuadricDrawStyle(lll, GLU_FILL);
    rul = gluNewQuadric(); gluQuadricDrawStyle(rul, GLU_FILL);
    rll = gluNewQuadric(); gluQuadricDrawStyle(rll, GLU_FILL);
}
///********* menu things ***********///
void menu(int id) {
    if (id == 0)
        exit(0);
    if (id == 1)
        where_to_rotate = 1;
    if (id == 2)
        where_to_rotate = 2;
    if (id == 3)
        where_to_rotate = 3;
    if (id == 4)
        where_to_rotate = 4;
    if (id == 5)
        where_to_rotate = 5;
    if (id == 6)
        where_to_rotate = 6;
    if (id == 7)
        where_to_rotate = 7;
    if (id == 8)
        where_to_rotate = 8;
    if (id == 9)
        where_to_rotate = 9;
    if (id == 10)
        where_to_rotate = 10;
    if (id == 11)
        where_to_rotate = 11;
    if (id == 12)
        where_to_rotate = 12;
    if (id == 13)
        where_to_rotate = 13;
    if (id ==14)
        where_to_rotate = 14;
    if (id ==15)
        where_to_rotate = 15;
    if (id == 16)
        where_to_rotate = 16;
    if (id == 17)
        where_to_rotate = 17;
    if (id ==18)
        where_to_rotate = 18;

}
void make_menu() {
    glutCreateMenu(menu);
    glutAddMenuEntry("Move right arm", 1);
    glutAddMenuEntry("Move left arm", 2);
    glutAddMenuEntry("Move right leg", 3);
    glutAddMenuEntry("Move left leg", 4);
    glutAddMenuEntry("Move Body", 5);
    glutAddMenuEntry("Move head towards up-down", 6);
    glutAddMenuEntry("Move head towards sideways", 7);
    glutAddMenuEntry("Move Body with left arm", 8);
    glutAddMenuEntry("Move Body with right arm", 9);
    glutAddMenuEntry("Move Body with left leg", 10);
    glutAddMenuEntry("Move Body with right leg", 11);
    glutAddMenuEntry("Move Body with head", 12);
    glutAddMenuEntry("Move Body with both arms", 13);
    glutAddMenuEntry("Move Body with both legs", 14);
    glutAddMenuEntry("Move Body with left arm,left leg and head", 15);
    glutAddMenuEntry("Move Body with right arm,right leg and head", 16);
    glutAddMenuEntry("Move Body with left arm,right leg and head", 17);
    glutAddMenuEntry("Move Body with right arm,left leg and head", 18);
    glutAddMenuEntry("exit", 0);
    glutAttachMenu(GLUT_RIGHT_BUTTON);
}
///********* main function ***********///
int main(int argc, char** argv) {
    const wchar_t* path = L"C:\\Users\\rosha\\Downloads\\blinding_lights1.wav";
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize(800, 800);
    glutCreateWindow("dancing robot");
    PlaySound(path, NULL, SND_FILENAME | SND_ASYNC |SND_LOOP);
    //PlaySound("", NULL, SND_ASYNC | SND_FILENAME | SND_LOOP);
    light_init();
    style_init();
    rotate_init(); /// init light, style and rotate
    glutReshapeFunc(reshape);
    glutDisplayFunc(display);
    glutIdleFunc(idle);
    make_menu();
    glutMainLoop();
    return 0;
}
```

CHAPTER 5

SNAPSHOTS

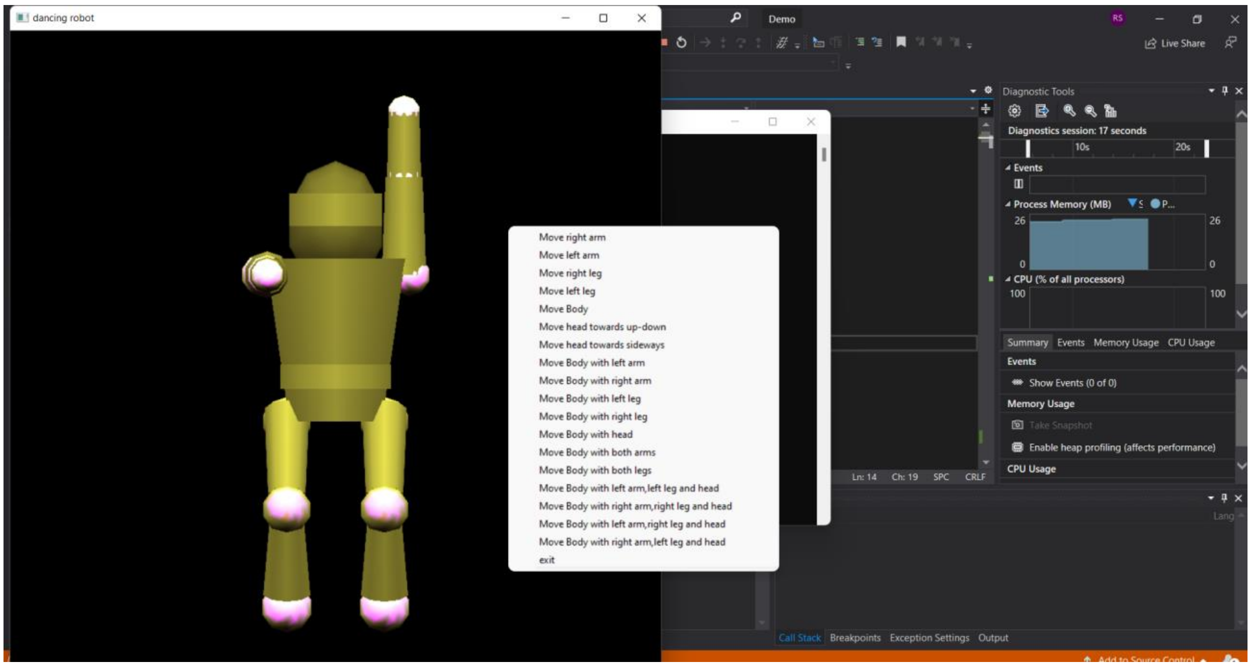


Fig 5.1 Menu Page

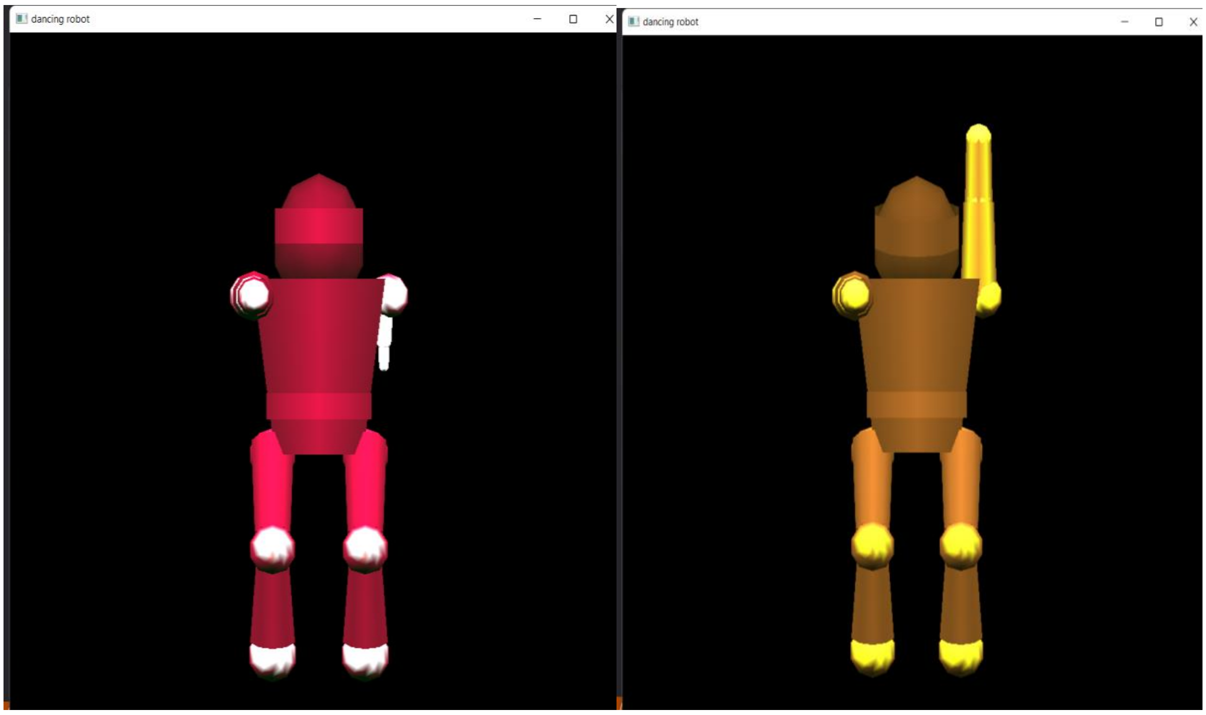


Fig 5.2 Change of Color

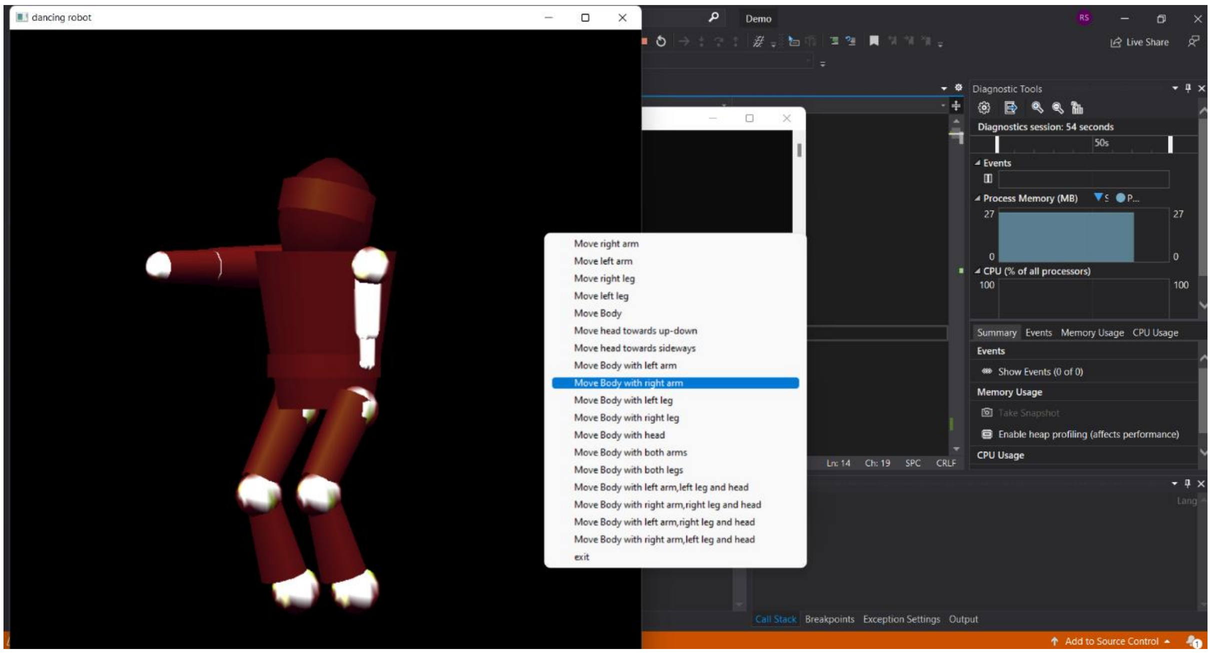

dancing robot
Move right arm
Move left arm
Move right leg
Move left leg
Move Body
Move head towards up-down
Move head towards sideways
Move Body with left arm
Move Body with right arm
Move Body with left leg
Move Body with right leg
Move Body with head
Move Body with both arms
Move Body with both legs
Move Body with left arm,left leg and head
Move Body with right arm,right leg and head
Move Body with left arm,right leg and head
Move Body with right arm,left leg and head
exit
Diagnostic Tools
Diagnostics session: 54 seconds
Events
Process Memory (MB)
CPU (% of all processors)
Summary Events Memory Usage CPU Usage
Show Events (0 of 0)
Take Snapshot
Enable heap profiling (affects performance)
Call Stack Breakpoints Exception Settings Output
Add to Source Control

Fig 5.3 Dance Moves

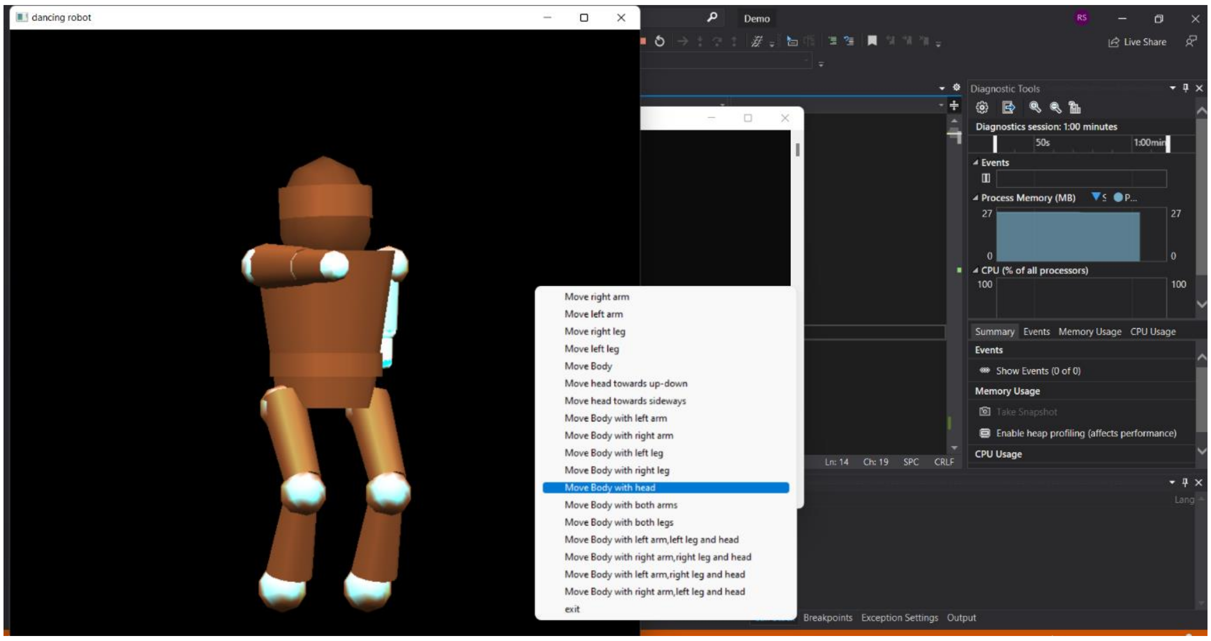

dancing robot
Move right arm
Move left arm
Move right leg
Move left leg
Move Body
Move head towards up-down
Move head towards sideways
Move Body with left arm
Move Body with right arm
Move Body with left leg
Move Body with right leg
Move Body with head
Move Body with both arms
Move Body with both legs
Move Body with left arm,left leg and head
Move Body with right arm,right leg and head
Move Body with left arm,right leg and head
Move Body with right arm,left leg and head
exit
Diagnostic Tools
Diagnostics session: 1:00 minutes
Events
Process Memory (MB)
CPU (% of all processors)
Summary Events Memory Usage CPU Usage
Show Events (0 of 0)
Take Snapshot
Enable heap profiling (affects performance)
Breakpoints Exception Settings Output

Fig 5.4 Dance Moves w.r.t head and body

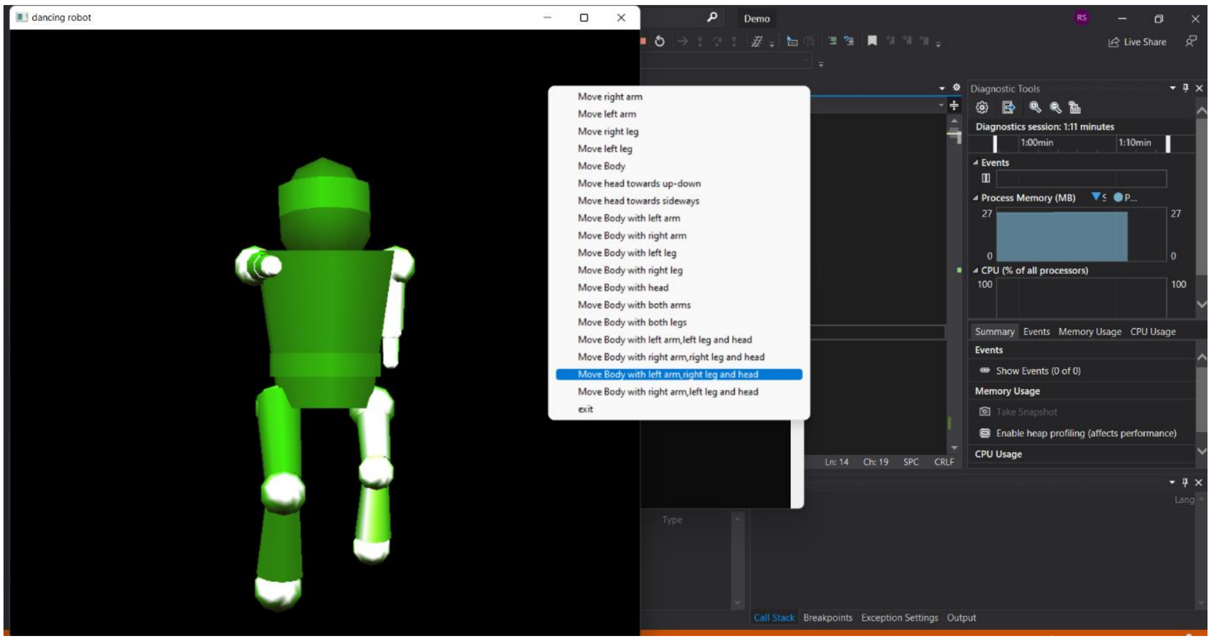


Fig 5.5 Dance Moves w.r.t Both legs and arms

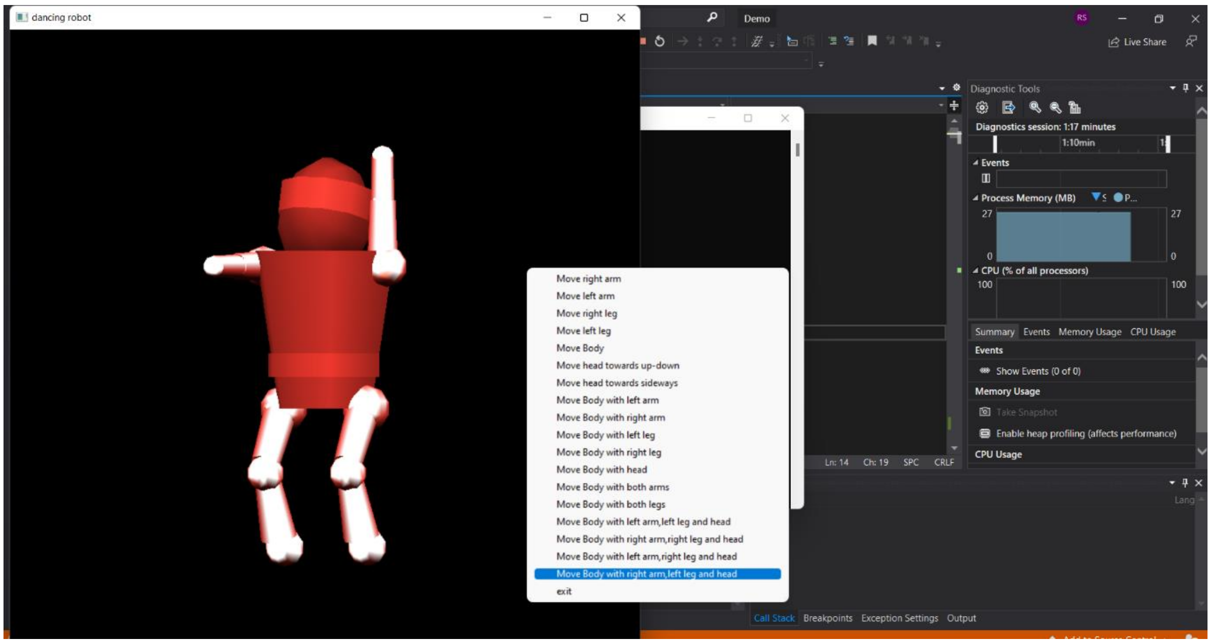


Fig 5.6 Dance Moves w.r.t body, right arm, left leg and head

CHAPTER 6

CONCLUSION

The simulation of Dancing Robot gives an interesting point of view on how objects like spherical and cylindrical shape can be used to form a robot, We had much fun in assigning and making dance moves with the help of rotation of angle with coordinates, dance moves are based on rotate function with respect to x, y, z axis.

The challenging part of this project is in assigning the values for ambient, specular, diffusion and shininess for various parts of body and lighting the objects with different color.

PlaySound function is an interesting one as it gives a life to dance moves, Since dance without music is not fun, here we are making use of blinding lights music as background music.

CHAPTER 7

FUTURE ENHANCEMENTS

Further Development with this project can be done by adding extra dance moves such as Hip-hop etc, and extra menus can be created for this dance moves.
Additional Songs can be used according to dance moves.
Robot can have more creative design by adding finger like structure instead of sphers for realistic view of hand.
Walls , surfaces for stages can be created with the help of Cube related OpenGL functions.
This Simulation can be made further use in app to simulate various dance moves with a matching songs.

REFERENCES

[1] Donald D Hearn and M.Pauline Baker, "Computer Graphics with OpenGL", 3rd Edition

[2] Edward Angel’s Interactive Computer Graphics Pearson Education 5th Edition

[3] Interactive computer Graphics --A top-down approach using open GL--by Edward Angle

[4] Motion Control of Dancing Character with Music