Collage of Computer Science and Technology
Zhejiang University Of Technology(浙江工业大学)

Computer Graphic

Final Project Report

Teacher:Huang Xianping

Topic:3D Animation

Name:MOLLICK SHAHRIAR HOSSAIN

Student ID:L201826100121

Major: Computer Science and Technology

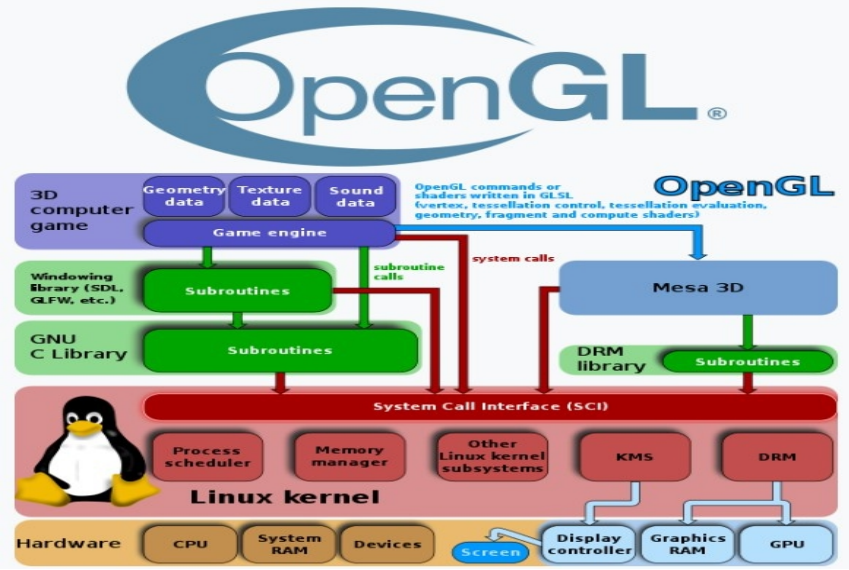
Submission Method:Superstar

ABSTRACT

Computer animation is the process used for digitally generating animated images. The more general term computer-generated imagery (CGI) encompasses both static scenes and dynamic images, while computer animation *only* refers to moving images. Modern computer animation usually uses 3D computer graphics to generate a two-dimensional picture, although 2D computer graphics are still used for stylistic, low bandwidth, and faster real-time renderings. Sometimes, the target of the animation is the computer itself, but sometimes film as well.The main aim of this project is to show the demonstration of working of a doll who can dance for some times. The project completely uses graphics and has been done using OpenGL. The code used in the program is very simple. We can easily understand the working of transformers by seeing the output of this project.

Minimum code is written to show the animation so that the programmers can easily understand the code as well. In the future . I have ensured the minimum use of functions and maximum code reuse. Using the techniques learned, I have tried to make the most efficient code.

CHAPTER 1 INTRODUCTION TO OPENGL



In the context of computer graphics, visuals produced using computers are defined as well as the representation and manipulation of visual data by a computer, more broadly. The advancement of computer graphics has made computers simpler to interact with, as well as more capable of comprehending and interpreting a wide range of data kinds and formats. The advancements in computer graphics have had a significant effect on many other kinds of media and have transformed the animation and video gaming industries, among others. A wide range of elements of our everyday lives are now influenced by computers and computer-generated pictures. Computer-generated imagery may be seen on television, in newspapers, in weather forecasts, and even during surgery. A well-constructed graph may show complicated statistics in a way that is more understandable and interpretable than other forms of presentation. Documents including papers, reports, theses, and other presentation materials are illustrated using graphs of this kind. For users to visualize their data, a variety of tools and capabilities are available, and computer graphics is utilized in a wide range of fields. The OpenGL API is used to create computer graphics in our applications. An API for developing programs that generate 2D and 3D computer graphics is defined by the OpenGL (Open Graphics Library) specification, which is a standard definition for a cross-language, cross- platform API.

OpenGL is a high-level graphics library standard developed by the OpenGL Foundation. It provides a limited collection of geometric primitives - points, lines, polygons, pictures, and bitmaps - accessible to the programmer. OpenGL offers a collection of instructions that enable the definition of geometric objects in two or three dimensions, using the supplied primitives, as well as instructions that govern how these objects are displayed. OpenGL is a graphics programming interface that was developed by Google (drawn). In

order to overcome the limitations of OpenGL drawing commands, which are limited to those that generate simple geometric primitives (such as points, lines, and polygons), the OpenGL Utility Toolkit (GLUT) has been developed to assist in the development of more complicated three-dimensional objects such as a teapot. GLUT may not be sufficient for fully fledged OpenGL applications, but it is a good starting point for understanding the OpenGL language and its capabilities.GLUT is intended to address the requirement for an OpenGL programming interface that is independent of the window system on which it is used. The interface is intended to be simple while yet meeting the requirements of effective OpenGL applications. It was a wise decision to remove window system operations from OpenGL, as doing so allows the OpenGL graphics system to be retargeted to a variety of systems, including powerful but expensive graphics workstations, and low cost mass-production graphics systems, such as video games, set-top boxes for interactive television, and personal computers.

To show a graphical scene produced using OpenGL, the GLUT application-programming interface (API) needs just a small number of functions to be implemented. In addition, the GLUT functions need just a small number of arguments.

1.1 COMPUTER GRAPHICS

The phrase "computer graphics" refers to virtually anything that may be shown on a computer that is not text or sound. In today's world, almost all computers include some kind of visual, and users expect their computers to be controlled by icons and images rather than just by entering commands. The phrase "Computer Graphics" may refer to a variety of things:The display and manipulation of visual data by a computer is known as computer graphics.The many technologies that are used in the creation and manipulation of such visual data.The pictures that have been created in this manner, andComputer graphics is a sub-field of computer science that investigates techniques for digitally synthesizing and manipulating visual information. Computers and computer-generated pictures are now present in almost every area of our lives. Computer- generated imagery may be seen on television, in newspapers, in weather forecasts, and even during surgery. A

well-constructed graph may show complicated statistics in a way that is more understandable and interpretable than other forms of presentation.

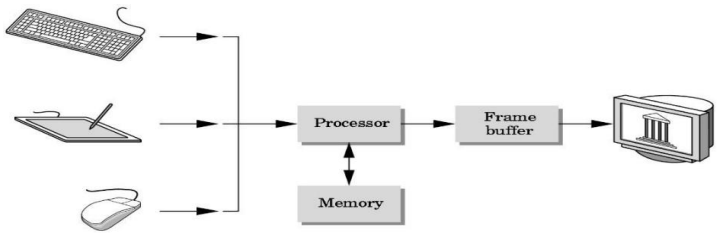


Fig 1.1.1 A Graphics System.

1.2 OPENGL TECHNOLOGY

OpenGL is described as "a software interface to graphics hardware" in its most rigorous sense. The library is a 3D graphics and modeling library that is very portable and very quick, and it is available for free download. Using OpenGL, you can build 3D visuals that are both elegant and attractive, as well as having outstanding visual quality. By far, the most significant benefit of utilizing OpenGL over a ray tracer or software rendering engine is the speed with which it can generate images at light speed. Initial versions of the software made use of algorithms painstakingly designed and refined by Silicon Graphics, Inc. (SGI), a well-known global pioneer in computer graphics and animation. Over time, as other manufacturers have contributed their knowledge and intellectual property to create high-performance versions of their own, OpenGL has developed to become a more complete package.

OpenGL is a software interface to graphics hardware that is used to display visuals. Approximately 120 different commands comprise this interface, and they are used in conjunction with one another to define the objects and actions required to create interactive three-dimensional applications. Unlike other graphics technologies, OpenGL is intended to operate effectively even in situations where the computer that shows the images you produce is not the same machine that is running your graphics application.

Because the syntax for sending OpenGL instructions (known as the protocol) from the client to the server is always the same, OpenGL applications may be run over a network even if the client and server are running on different types of computers (for example, Windows and Linux). If an OpenGL application isn't operating over a network, there is just one machine involved, which serves as both the client and the server at the same time, as described below. In its most basic form, OpenGL is intended to be a simplified, hardware-independent interface that can be implemented on a wide range of hardware platforms.

OpenGL does not offer high-level instructions for defining models of three-dimensional objects, which is a limitation of the technology. Such instructions may enable you to define rather complex forms, such as cars, body parts, aircraft, or molecules, without having to resort to programming. When using OpenGL, you must construct your desired model from a limited number of geometric primitives, such as points, lines, and polygons, before it can be rendered. Computer graphics is concerned with all elements of producing pictures using a computer, including but not limited to:

In this case, the item is an artist's depiction of the sun for use in an animation that will be displayed in a dome-shaped setting (planetarium)

Maya is used for modeling and rendering, however Maya is built on top of the OpenGL graphics API.

Hardware is required for modeling and rendering, which includes a PC with a graphics card.

Figure 1.2.1 Graphics pipeline.

CHAPTER 2 PROJECT DESCRIPTION

The purpose of this project is to use animation to explain the operation in classroom environment.This project maintain Two separate class-based composite objects that each consist of a parent with no geometry, that each have at least 8 attached child objects with geometry, and that each have at least one animation methods that animate some property of child objects, such as position, orientation, or scale. The child objects directly attached to the parent, or they attached indirectly via other child objects. Each class override the hide & show methods so that hide and show calls sent to the composite object are passed to all child objects. Child objects has not hidden inside an object such that they are not visible. This project composite objects must real, recognisable things (such as people, robots, cars, machines, buildings, etc.). That are not be arbitrary collections of shapes. This project's two composite objects are looking significantly different to each other and independent of each other.

Each composite object must have at least 2 independently animatable children. The animation capability of the child components exposed through animation methods in each composite object. This project's composite objects from your own creation.

CHAPTER 3 DESIGN CONSTRAINTS

During the project's design phase, showing the movement of the components proved to be a significant challenge. Because OpenGL provides a wide number of APIs and built-in functions, we were able to overcome this obstacle without much difficulty. The following are the hardware and software limitations:

3.1 HARDWARE CONSTRAINTS

The setup of the machine is not subject to any tight limitations. The editor should be able to function on any machine that is capable of running the most recent version of Microsoft's Windows operating system.

My Device specification:

- Processor : Core i7
- RAM : 32 GB
- Hard Disk: 256 GB (approx)
- Display : Full HD Color Monitor
- GPU : 1060GTX(6GB)

3.2 SOFTWARE CONSTRAINTS

- Operating System : Windows XP/Vista/7/8/10/11
- Language : C/C++
- Compiler : Microsoft Visual Studio (Any version)
- **Must support OpenGL**

Things I didn’t

In implementing the requirements of the project, there are some things that I did not. Specifically:

I did not modify the supplied sweep class or extrusion class in any way.

I did not use any 3D modelling applications to create any object for this project.I did not include any 3rd

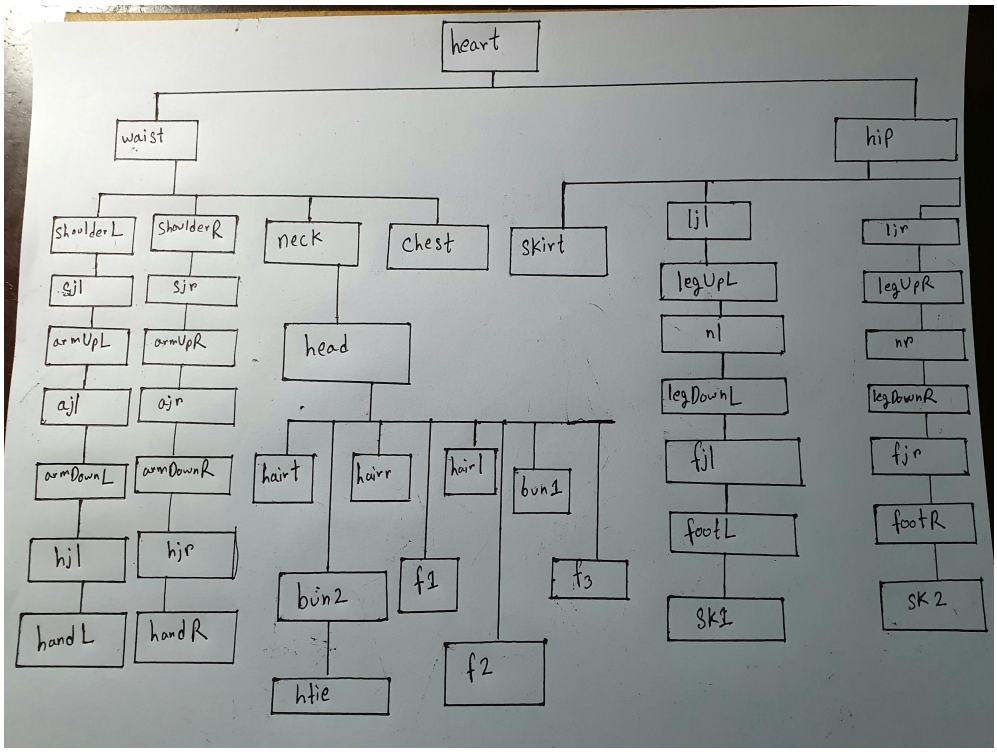
-party software (such as the Quake SDK).

I have not add any sweeps, extrusions, or other objects that are "inside-out".

CHAPTER 4 ARCHITECTURE

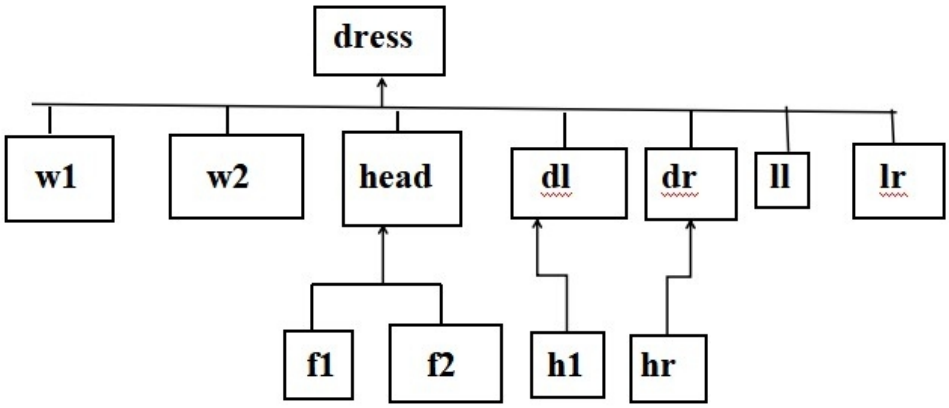
Composite1 Picture:

Hierarchical structure for composite 1:



Composite 2 Picture:

Hierarchical structure for composite 2:



CHAPTER 5 CODE IMPLEMENTATION

Visual Studio is used to carry out the actual execution of this 3D engine project.For Teacher's requirement I should make this animation based on the specified graphics engine in this semester.So,This project is made with many codes.I am giving most important code that are written by me….

scene.cpp:

```
//
**************************************************************************
************
// KXC354 - Computer Graphics & Animation - 2014
// Assignment 1 & 2
// 3D Engine Code
//
**************************************************************************
************
//
// Author: Tony Gray
//
// scene.cpp



// ---------------------------------------------------------------------------
// includes
// ---------------------------------------------------------------------------
#include "prefix.h"
#include "constants.h"                  // system-wide constants
#include "utility.h"                    // general utility functions
#include "light.h"
#include "camera.h"
#include "texture.h"
#include "scene.h"
#include "snd.h"
#include "skybox.h"
#include "fog.h"
#include "viewfrustum.h"

// #includes for geometry
#include "sweep.h"
#include "extrusion.h"
```

```
#include "cube.h"
#include "sphere.h"
#include "torus.h"
#include "terrain.h"
#include "billboard.h"
#include "plane.h"
#include "tsphere.h"
#include "cylinder.h"
#include "disc.h"
#include "tcube.h"

#include "composite1.h"
#include "composite2.h"
// ------------------------------------------------------------------------------------
// global variables
// ------------------------------------------------------------------------------------
extern ProgramMode		gProgramMode;
	// initially, we're not animating

extern cameraClass		gCamera;
	// the camera
extern object3d				*gSky;
		// the global skybox
extern fog				*gFog;
		// the global fog
extern viewfrustum		*gViewFrustum;
	// the global view frustum object
extern float			gCurrentFrameTime;					// the
time that the current animation frame was started

extern vector<object3d*> gShapeVector;
typedef vector<object3d*>::iterator shapeVectorIterator;




// ------------------------------------------------------------------------------------
// variables that represent 3D objects being used in the animation
// ------------------------------------------------------------------------------------
// these are really global variables, but you shouldn't need to access them from anywere
// except in this file
// ------------------------------------------------------------------------------------

// pointer of shapes defined here
```

```
composite1 *f;
composite2 *f1;
terrain *theGround;
tcube *d;
texture *tex ;

// some lights
light *ambient, *light0, *light1, *light2;



// ------------------------------------------------------------------------------------
// constructScene
// ------------------------------------------------------------------------------------
// This function constructs the objects required for the animation.  It is only called once
// when the program is first run.
// Use this to create child objects and bind them to their parents.
// ------------------------------------------------------------------------------------

void constructScene()
{
        // create the lights first
        ambient = new light(GL_LIGHT_MODEL_AMBIENT);
        light0 = new light(GL_LIGHT0);
        light1 = new light(GL_LIGHT1);
   light2 = new light(GL_LIGHT2);
        ///Add 3D objects here

        gSky = new skybox("sky16-");

         f = new composite1;
    f1 = new composite2;


          tex = new texture("122.jpg", kContinuousTone, kMipmaps);
          d= new tcube();
          d->setDeformation(100,1,100);

          d->setTexture(tex);
     d->setPosition(0,-8,0);

        theGround = new terrain("ground.bmp");
        theGround ->setPosition(0, -9, 0);
        theGround ->setScale(2);
        theGround->useSmoothShading();
        theGround->setColour(2,1,1);
```

```
}



// ------------------------------------------------------------------------------------
// resetScene
// ------------------------------------------------------------------------------------
// This function is called whenever the animation is restarted.  Use it to reset objects
// to their starting position or size, or to reset any aspects of the objects that
// might have been altered during an animation sequence.
// ------------------------------------------------------------------------------------

void resetScene()
{
	// initialise the camera
	gCamera.setPosition(0, 1, 10);
	gCamera.setTarget(f);




	ambient->setColour(0.5, 0.5, 0.5, 1.0);

	light0->turnOn();
	light0->setPosition(-10, 10, 5);
	light0->setColour(0.7, 0.7, 0.7, 1.0);
	light0->setSpecularColour(0.7, 0.7, 0.7, 1.0);
	light0->makePositional();
	light0->setLinearAttenuation(0.1);

	light1->turnOn();
	light1->setPosition(10, 10, -5);
	light1->setColour(0.7, 0.7, 0.7, 1.0);
	light1->setSpecularColour(0.7, 0.7, 0.7, 1.0);
	light1->makePositional();
	light1->setLinearAttenuation(0.1);

	light2->turnOn();
	light2->setPosition( 10, 10,-250);
	light2->setColour(0.7, 0.7, 0.7, 1.0);
	light2->setSpecularColour(0.7, 0.7, 0.7, 1.0);
	light2->makePositional();
	light2->setLinearAttenuation(0.1);

	// reset all objects to their starting position
	 f->setPosition(0,0,-8);
	 f1->setPosition(5,0,-8);
```

```
}



// ------------------------------------------------------------------------------------
// animateForNextFrame
// ------------------------------------------------------------------------------------
// This function is called to animate objects ready for the next frame
// ------------------------------------------------------------------------------------

void animateForNextFrame(float time, long frame)
{

        ////Animate 3D objects here

         gCamera.setTarget(f);

         if(21<time && time<=31)
         {
                 gCamera.setPosition( 0,0,-70);
                 gCamera.setTarget(2,0,0);

         }

         f->animate(time);
         f1->animate(time);



         //eventually stop
         if (time >= 60)
           gProgramMode = kpmFinished;

}
```

scene.h:

```
//
//////////////////////////////////////////////////////////////////////////////
// KXC354 - Computer Graphics & Animation - 2014
// Assignment 1 & 2
// 3D Engine Code
//
//////////////////////////////////////////////////////////////////////////////
//
// Author: Tony Gray
```

```
//
// scene.h
//

#ifndef _SCENE_H
#define _SCENE_H


void constructScene();
void resetScene();
void animateForNextFrame(float time, long frame);


#endif // _SCENE_H
```

composite1.cpp:

```
// ----------------------------------------------------------------------------
// includes
// ----------------------------------------------------------------------------
#include "prefix.h"
#include "composite1.h"
#include "utility.h"
#include "camera.h"


// ----------------------------------------------------------------------------
// globals
// ----------------------------------------------------------------------------
extern vector<object3d*> gShapeVector;



// ----------------------------------------------------------------------------
// constructor
// ----------------------------------------------------------------------------

composite1::composite1()
{

                //neck, chest,, waist , armUpL,armUpR,armDownL,armDownR,head
          //legUpL,legUpR,legDownL,legDownR,footL,footR,handR,handL ,
shoulderR,shoulderL;

        setName("Dancing doll");
```

```
        //main core it layes inside her
// so heart sounded like a good name
  heart = new sphere(100);
  heart->setPosition(0,0,0);
  heart->setDeformation(0.1,0.1,0.1);
        heart->setParent(this);

        //waist
        waist = new sphere(100);
        waist->setColour(0.9,0.5,0.6);
        waist->setPosition(0,0,0);
        waist->setDeformation(0.5,1,0.5);
        waist->setParent(heart);

        //chest
        chest= new sphere(100);
        chest->setColour(0.9,0.5,0.6);
        chest->setDeformation(0.8,0.6,0.5);
        chest->setPosition(0,0.5,0.2);
  chest->setParent(waist);

        //neck
        neck= new sphere(100);
        neck->setColour(1,1,1);
        neck->setDeformation(0.3,0.5,0.3);
        neck->setPosition(0,1.35,0);
  neck->setParent(waist);

        //head
        tex = new texture("face1.jpg", kContinuousTone, kMipmaps);
        head= new tsphere(400);
        head->setDeformation(0.6,0.7,0.6);
        head->setPosition(0,0.7,0.1);
        head->setTexture(tex);
  head->setParent(neck);

        //top hair
        hairt= new sweep("hair.txt",100);
        hairt->setColour(1,0,0);
        hairt->setDeformation(0.35,0.1,0.3);
        hairt->setRotation('x',180);
        hairt->setPosition(0,0.75,0);
  hairt->setParent(head);

        //hair right side
        hairr= new sweep("hair.txt",100);
        hairr->setColour(0,0,0);
        hairr->setDeformation(0.2,0.05,0.3);
```

```
        hairr->setRotation('x',90,'y',-90);
        hairr->setPosition(0.7,0,0);
hairr->setParent(head);

        //hair left side
        hairl= new sweep("hair.txt",100);
        hairl->setColour(0,0,0);
        hairl->setDeformation(0.2,0.05,0.3);
        hairl->setRotation('x',90,'y',90);
        hairl->setPosition(-0.7,0,0);
hairl->setParent(head);

        //bun to cover the back of her head
        bun1= new sphere( 100);
        bun1->setColour(0,0,0);
        bun1->setDeformation(0.6,0.6,0.3);
        bun1->setPosition(0,0.1,-0.4);
bun1->setParent(head);

        //the top bun
        bun2= new sphere( 100);
        bun2->setColour(0,0,0);
        bun2->setDeformation(0.7,0.6,0.6);
        bun2->setPosition(0,0.5,-0.5);
bun2->setParent(head);

        //hair tie for the top
        htie= new torus( 100,16,0.5,0.2);
        htie->setColour(0.9,0.5,0.6);
        htie->setRotation('x',-45);
        htie->setPosition(0,-0.2,0.15);
htie->setParent(bun2);



       //flower 1 for the hair
f1 = new extrusion("flower.txt","path2.txt");
       f1->setColour(0.9,0.5,0.6);
       f1->setScale(0.5);
       f1->setParent(head);
       f1->setRotation('y', 90,'z',45,'x',30);
       f1->setPosition( 0.56,0.6,0);

       //flower2 for the hair
       f2 = new extrusion("flower.txt","path2.txt");
       f2->setColour(0.9,0.5,0.6);
       f2->setScale(0.5);
       f2->setParent(head);
```

```
        f2->setRotation('y',-90,'z',-45,'x',30);
        f2->setPosition( -0.56,0.6,0);

        //flower3 for the hair
        f3 = new extrusion("flower.txt","path2.txt");
        f3->setColour(0.9,0.5,0.6);
        f3->setScale(0.5);
        f3->setParent(head);
        f3->setRotation('x',-59);
        f3->setPosition( 0,0.88,0);

        //right shoulder
        shoulderR = new sphere(100);
        shoulderR->setColour(5,5,5);
        shoulderR ->setDeformation(0.7,0.5,0.5);
        shoulderR->setPosition(0.4,0.7,-0.05);
shoulderR->setParent(waist);

        //left shoulder
        shoulderL = new sphere(200);
        shoulderL->setColour(5,5,5);
        shoulderL->setDeformation(0.7,0.5,0.5);
        shoulderL->setPosition(-0.4,0.7,-0.05);
shoulderL->setParent(waist);

        //right shoulder joint
        sjr = new sphere(100);
        sjr->setColour(0.9,0.5,0.6);
        sjr->setDeformation(0.4,0.4,0.4);
sjr->setParent(shoulderR);
        sjr->setPosition(0.6,0,0);

//left shoulder joint
sjl = new sphere(100);
        sjl->setColour(0.9,0.5,0.6);
        sjl->setDeformation(0.4,0.4,0.4);
        sjl->setPosition(-0.6,0,0);
sjl->setParent(shoulderL);

        //upper part of the right arm
        armUpR = new sphere(100);
        armUpR->setColour(1,1,1);
        armUpR->setDeformation(0.3,0.9,0.3);
        armUpR->setParent(sjr);
        armUpR->setPosition(0,-0.7,0);

        //upper part of left arm
        armUpL = new sphere(100);
```

```
        armUpL->setColour(1,1,1);
        armUpL->setDeformation(0.3,0.9,0.3);
        armUpL->setParent(sjl);
        armUpL->setPosition(0,-0.7,0);

//left arm joint == left elbow
        ajl = new sphere(100);
        ajl->setColour(1,1,1);
        ajl->setDeformation(0.25,0.25,0.25);
        ajl->setParent(armUpL);
        ajl->setPosition(0,-0.7,0);

        //right arm joint == right elbow
        ajr = new sphere(100);
        ajr->setColour(1,1,1);
        ajr->setDeformation(0.25,0.25,0.25);
        ajr->setParent(armUpR);
        ajr->setPosition(0,-0.7,0);

        //Down part of left arm
        armDownL = new sphere(100);
        armDownL->setColour(1,1,1);
        armDownL->setDeformation(0.27,0.8,0.27);
        armDownL->setParent(ajl);
        armDownL->setPosition(0,-0.8,0);

//Down part of right arm
        armDownR = new sphere(100);
        armDownR->setColour(1,1,1);
        armDownR->setDeformation(0.27,0.8,0.27);
        armDownR->setParent(ajr);
        armDownR->setPosition(0,-0.8,0);

        //left wrist
        hjl = new sphere(100);
        hjl->setColour(1,1,1);
        hjl->setDeformation(0.2,0.2,0.2);
        hjl->setParent(armDownL);
        hjl->setPosition(0,-0.8,0);

        //right wrist
        hjr = new sphere(100);
        hjr->setColour(1,1,1);
        hjr->setDeformation(0.2,0.2,0.2);
        hjr->setParent(armDownR);
hjr->setPosition(0,-0.8,0);

        //left hand
```

```
        handL = new sphere(100);
        handL->setColour(1,1,1);
        handL->setDeformation(0.2,0.5,0.3);
        handL->setParent(hjl);
        handL->setPosition(0,-0.4,0);

        //right hand
        handR = new sphere(100);
        handR->setColour(1,1,1);
        handR->setDeformation(0.2,0.5,0.3);
        handR->setParent(hjr);
        handR->setPosition(0,-0.4,0);

        //hip
        hip = new sphere(100);
        hip->setColour(0.9,0.5,0.6);
        hip->setDeformation(0.9,0.6,0.6);
        hip->setPosition(0,-1,0);
hip->setParent(heart);

//skirt
skirt =new sweep("skirt.txt",200);
skirt->setColour(4,4,1,0.6);
skirt->useSmoothShading();
skirt->setDeformation(1.95,0.9,1.8);
skirt->setRotation('x',180);
skirt->setPosition(0,0.61,0);
skirt->setParent(hip);


//left leg joint  (hip-leg)
        ljl = new sphere(100);
        ljl->setColour(1,1,1);
        ljl ->setDeformation(0.49, 0.49 ,0.49);
        ljl->setPosition(-0.55,-0.1,0);
ljl->setParent(hip);

        //right leg joint  (hip-leg)
         ljr = new sphere(100);
         ljr->setColour(1,1,1);
         ljr ->setDeformation(0.49, 0.49 ,0.49);
         ljr->setPosition(0.55,-0.1,0);
 ljr->setParent(hip);

         //left leg upper part
 legUpL = new sphere(100);
        legUpL->setColour(1,1,1);
        legUpL->setDeformation(0.49, 1.3 ,0.49);
```

```
	legUpL->setPosition(0,-1.3,0.2);
legUpL->setParent(ljl);

	//right leg upper part
	legUpR = new sphere(100);
	legUpR->setColour(1,1,1);
	legUpR->setDeformation(0.49, 1.3 ,0.49);
	legUpR->setPosition(0,-1.3,0.2);
	legUpR->setParent(ljr);

	//right knee
	nr = new sphere(100);
	nr->setColour(1,1,1);
	nr->setDeformation(0.35, 0.35 ,0.35);
	nr->setPosition(0,-1.2,0);
	nr->setParent(legUpR);

	//left knee
	nl = new sphere(100);
	nl->setColour(1,1,1);
	nl->setDeformation(0.35, 0.35 ,0.35);
	nl->setPosition(0,-1.2,0);
nl->setParent(legUpL);

	//down part of left leg
legDownL = new sphere(100);
	legDownL->setColour(1,1,1);
	legDownL->setDeformation(0.4, 1.3 ,0.43);
	legDownL->setPosition(0,-1,-0.1);
legDownL->setParent(nl);

	//Down part of leg right
	legDownR = new sphere(100);
	legDownR->setColour(1,1,1);
	legDownR->setDeformation(0.4, 1.3 ,0.43);
	legDownR->setPosition(0,-1,-0.1);
legDownR->setParent(nr);

	//right foot joint
	fjr = new sphere(100);
	fjr->setColour(0.9,0.5,0.6);
	fjr->setDeformation(0.25, 0.25 ,0.25);
fjr->setPosition(0,-1.3,0);
fjr->setParent(legDownR);

	//left foot joint
	fjl = new sphere(100);
	fjl->setColour(0.9,0.5,0.6);
```

```
        fjl->setDeformation(0.25, 0.25 ,0.25);
        fjl->setPosition(0,-1.3,0);
    fjl->setParent(legDownL);

        //right foot
        footR = new sphere(100);
        footR ->setColour(1,0,0);
        footR->setDeformation( 0.3, 0.25 ,0.6);
        footR->setPosition(0,-0.1,0.3);
    footR->setParent(fjr);

        //left foot
        footL= new sphere(100);
        footL ->setColour(1,0,0);
        footL->setDeformation( 0.3, 0.25 ,0.6);
        footL->setPosition(0,-0.1,0.3);
    footL->setParent(fjl);

        //two slices sk1 sk2
        //to make each foot look like skating shoe
        sk1= new cube();
        sk1 ->setColour(0.65,0.8,0.85);
        sk1->setDeformation(0.05,0.25,0.53);
        sk1->setPosition(0,-0.3,-0.04);
    sk1->setParent(footL);

        sk2= new cube();
        sk2 ->setColour(0.65,0.8,0.85);
        sk2->setDeformation(0.05,0.25,0.53);
        sk2->setPosition(0,-0.3,-0.04);
    sk2->setParent(footR);


        gShapeVector.push_back(this);

}

void composite1::hide()
{
        head->hide();
        neck->hide();
        chest->hide();
        heart->hide();
        hip->hide();
        armUpL->hide();
        armUpR->hide();
        armDownL->hide();
        armDownR->hide();
```

```
        legUpL->hide();
    legUpR->hide();
        legDownL->hide();
        legDownR->hide();
        footL->hide();
        footR->hide();
        handR->hide();
        handL->hide();
        shoulderR->hide();
        shoulderL->hide();
        ajl->hide();
        ajr->hide();
        ljl->hide();
        ljr->hide();
        nl->hide();
        nr->hide();
        fjr->hide();
        fjl->hide();
        hjr->hide();
        hjl->hide();
        sjr->hide();
        sjl->hide();
        waist->hide();
    hairt->hide();
        hairl->hide();
        hairr->hide();
        skirt->hide();
    bun1->hide();
        bun2->hide();
    f1->hide();
        f2->hide();
        f3->hide();
    htie->hide();
    sk1->hide();
        sk2->hide();

};

void composite1::show()
{

        head->show();
        neck->show();
        chest->show();
        heart->show();
        hip->show();
        armUpL->show();
        armUpR->show();
```

```
        armDownL->show();
        armDownR->show();
        legUpL->show();
    legUpR->show();
        legDownL->show();
        legDownR->show();
        footL->show();
        footR->show();
        handR->show();
        handL->show();
        shoulderR->show();
        shoulderL->show();
        ajl->show();
        ajr->show();
        ljl->show();
        ljr->show();
        nl->show();
        nr->show();
        fjr->show();
        fjl->show();
        hjr->show();
        hjl->show();
        sjr->show();
        sjl->show();
        waist->show();
    hairt->show();
        hairl->show();
        hairr->show();
        skirt->show();
    bun1->show();
        bun2->show();

    f1->show();
        f2->show();
        f3->show();
    htie->show();
    sk1->show();
        sk2->show();

}

void composite1 ::animate(float time)
{

        float x1,z1,xw;

        if(time<=10)
        {
```

```
 x1=time*15;
        z1=time*15;
        xw=time*15;


       if(0<time && time<=4.5)
       {
ljr->setRotation('x',x1);
sjr->setRotation('z',z1,'x',x1);
      sjl->setRotation('z',-z1,'x',x1);
       }

      float z=speedUp(0,10,-50,0);
heart->setPosition(0,0,z);

}




            else if(10<time && time<=20)
  {
              if( time<17 )
           {
     ljr->setRotation('x',-x1/100);
          sjr->setRotation( 'x',-x1/200);
          sjl->setRotation('x',-x1/200);
    }


                     float p1 = 0;
   float p2 = +180;
  float duration = 0.4;

            hjl->setRotation('z', -50);
            hjr->setRotation('z',-30);
            sjr->setRotation('x',-180);
            sjl->setRotation('x',90,'y',200);

            nl->setRotation('x',90);

           float angle = (((p2 - p1) / duration) * time) + p2;
  heart->setRotation('y', angle);


            }
```

```
        else if(20<time && time<=20.5)

                {

                sjr->setRotation('x',0);
                 sjl->setRotation('x',0);
                 nl->setRotation('x',0);

                  hjl->setRotation('z', 0);
                 hjr->setRotation('z',0);

                }



else if(20.5<time && time<=26)
          {
      x1=time*3;
            z1=time*3;
            xw=time*5;

          if( 22 >=time && 20.5<time )
          {
    ljr->setRotation('x',x1);
    sjr->setRotation('z',z1,'x',x1);
          sjl->setRotation('z',-z1,'x',x1);

          }


          float z=speedUp(21,4,0,-40);
    heart->setPosition(0,0,z);

          }



          else if(26<time && time<=36)
   {

                              float p1 = -180;
       float p2 = +180;
      float duration = 0.5;

                  if(time<28)
                  {
                  hjl->setRotation('z', 15);
                  hjr->setRotation('z',-15);
```

```
                sjr->setRotation('z',-10,'x',-180);
                sjl->setRotation('z',10,'x',-180);

                nr->setRotation('x',60);
                ljr->setRotation('x',-60,'z',-40);
                }

                else if(time<28.1)
                {
                hjl->setRotation('z',0);
                hjr->setRotation('z',0);
                sjr->setRotation('z',0,'x',0);
                sjl->setRotation('z',0,'x',0);

                nr->setRotation('x',0);
                ljr->setRotation('x',0,'z',0);
                }

                else if(time<30)
                {

                sjr->setRotation('z',90);
                sjl->setRotation('z',-90);

                nr->setRotation('x',0);
                ljl->setRotation('z',-30);
                }

                float angle = (((p2 - p1) / duration) * time) + p1;
        heart->setRotation('y', angle);

                float z=speedUp(30,4,-40,-10);
        heart->setPosition(0,0,z);


              }

          }
```

composite2.h:

```
//
**********************************************************************************
************
// KXC354 - Computer Graphics & Animation - 2018
// Assignment 1 & 2
// 3D Engine Code
```

```
//
//////////////////////////////////////////////////////////////////////////////
//
// Author: HXP
//
// composite1.h
//
// This file declares the class for composite1 objects, based on the
// generic 3D object


#ifndef _composite1_H
#define _composite1_H



// ----------------------------------------------------------------------------
// includes
// ----------------------------------------------------------------------------
#include "face.h"
#include "object3d.h"
#include "sphere.h"
#include "cube.h"
#include "sweep.h"
#include "extrusion.h"
#include "disc.h"
#include "torus.h"
#include "tsphere.h"
#include "texture.h"


// ----------------------------------------------------------------------------
// class declaration
// ----------------------------------------------------------------------------

class composite1 : public object3d
{
public:
                                                composite1();
                virtual void    hide(void);
                virtual void    show(void);


                void animate(float time);

private:
```

```
sphere *neck ,*chest,*heart, *hip , *armUpL,*armUpR,*armDownL,*armDownR,
            *legUpL,*legUpR,*legDownL,*legDownR,*footL,*footR,*handR,*handL ,
*shoulderR,*shoulderL,

                *ajl , *ajr,*ljl ,*ljr,* nl,*nr,*fjr,*fjl,*hjr,*hjl ,*sjr,*sjl, *waist ;

tsphere *head ;
texture *tex ;

sweep *hairt , *hairl, *hairr ,*skirt ;
sphere *bun1 ,*bun2;

extrusion *f1 ,*f2,*f3;
torus *htie;

cube *sk1 ,*sk2;

};


#endif // _composite1_H
```

composite2.cpp:

```
// ----------------------------------------------------------------------------
// includes
// ----------------------------------------------------------------------------
#include "prefix.h"
#include "composite2.h"
#include "utility.h"



// ----------------------------------------------------------------------------
// globals
// ----------------------------------------------------------------------------
extern vector<object3d*> gShapeVector;



// ----------------------------------------------------------------------------
// constructor
// ----------------------------------------------------------------------------

composite2::composite2()
{
```

```
        setName("composite2");
         dress= new sweep("dress.txt",200);
         dress->setColour(1,0,0);
         dress->setPosition(0,0,0);
         dress->setParent(this);

         head =new sphere(200);
         head->setScale(0.6);
         head->setPosition(0,2.5,0);
         head->setColour(1,1,1);
         head->setParent(dress);

f1 = new extrusion("flower.txt","path2.txt");
        f1->setScale(1);
        f1->setParent(head);
        f1->setRotation('y', 90,'z',45);
        f1->setPosition( 0.8,0.6,0);

        f2 = new extrusion("flower.txt","path2.txt");
        f2->setScale(1);
        f2->setParent(head);
        f2->setRotation('y',-90,'z',-45);
        f2->setPosition( -0.8,0.6,0);

        w1 =new extrusion("wing.txt","path.txt");
        w1->setScale(2);
        w1->setRotation('x',10);
        w1->setPosition(0,1.4,-0.7);
        w1->setParent(dress);


w2 =new extrusion("wing.txt","path.txt");

w2->setScale(2);
        w2->setOffset(0,0,-0.2);
        w2->setRotation('y',180,'x',10);
w2->setPosition(0,1.4,-0.7);
        w2->setParent(dress);

         dr= new sphere(100);
         dr->setScale(0.4);
 dr->setColour(5,1,1);
         dr->setPosition(0.6,1.7,0);
 dr->setParent(dress);

         hr= new sphere(100);
         hr->setDeformation(0.7,0.15,0.15);
```

```
        hr->setScale(2.5);
        hr->setRotation('z',-40);
        hr->setPosition(1,-0.7,0);
        hr->setColour(1,1,1);
        hr->setParent(dr);


        dl= new sphere(100);
        dl->setScale(0.4);
    dl->setColour(5,1,1);
        dl->setPosition(-0.6,1.7,0);
    dl->setParent(dress);


        hl= new sphere(100);
        hl->setDeformation(0.7,0.15,0.15);
        hl->setScale(2.5);
        hl->setRotation('z',40);
        hl->setPosition(-1,-0.7,0);
        hl->setColour(1,1,1);
        hl->setParent(dl);




        lr= new sphere(100);
        lr->setDeformation(0.8,0.2,0.2);
        lr->setPosition(-0.5,0,0);
        lr->setColour(1,1,1);
        lr->setRotation('z',-90);
        lr->setParent(dress);


         ll= new sphere(100);
         ll->setDeformation(0.8,0.2,0.2);
         ll->setPosition(0.5,0,0);
         ll->setColour(1,1,1);
         ll->setRotation('z',90);
         ll->setParent(dress);

       gShapeVector.push_back(this);

}

void composite2::hide()
{

       dress->hide();
```

```
  w1->hide();
        w2->hide();
        f1->hide();
   f2->hide();
   head->hide();
        hl->hide();
        hr->hide();
        lr->hide();
        ll->hide();
        dl->hide();
        dr->hide();




}

void composite2::show()
{

        dress->show();
   w1->show();
        w2->show();
        f1->show();
   f2->show();
   head->show();
        hl->show();
        hr->show();
        lr->show();
        ll->show();
        dl->show();
        dr->show();


}

void composite2 ::animate(float time)
{


        float p1 = -5;
    float p2 = +10;
   float duration = 5;

   float y = (((p2 - p1) / duration) * time) + p1;
   this->setPosition(0, y, 0);

   f1->setRotation('y', 90,'x',time*100,'z',40);
```

```
f2->setRotation('y',-90,'x',time*100,'z',-40);


  w1->setRotation('y',+30+20*sin_d(time*720));
  w2->setRotation('y', 80+70 -45*sin_d(time*720));


        }
```

composite2.h:

```
//
//////////////////////////////////////////////////////////////////////////////
// KXC354 - Computer Graphics & Animation - 2018
// Assignment 1 & 2
// 3D Engine Code
//
//////////////////////////////////////////////////////////////////////////////
//
// Author: HXP
//
// composite2.h
//
// This file declares the class for composite2 objects, based on the
// generic 3D object


#ifndef _composite2_H
#define _composite2_H



// ---------------------------------------------------------------------------
// includes
// ---------------------------------------------------------------------------
#include "face.h"
#include "object3d.h"
#include "sphere.h"
#include "cube.h"
#include "sweep.h"
#include "extrusion.h"


// ---------------------------------------------------------------------------
// class declaration
// ---------------------------------------------------------------------------
```

```
class composite2 : public object3d
{
public:
                                                composite2();
                virtual void    hide(void);
                virtual void    show(void);

                void animate(float time);

private:
sweep *dress;
extrusion *w1 , *w2,*f1 ,*f2;
sphere *head , *hl, *hr, *lr, *ll , *dl,*dr;

};


#endif // _composite2_H
```

tcube.cpp:

```
//
//////////////////////////////////////////////////////////////////////////////
// KXC354 - Computer Graphics & Animation - 2013
// Assignment 1 & 2
// 3D Engine Code
//
//////////////////////////////////////////////////////////////////////////////
//
// Author: Tony Gray
//
// tcube.cpp
//
// This file defines the class for textured cubes



// ----------------------------------------------------------------------------
// application includes
// ----------------------------------------------------------------------------
#include "prefix.h"
#include "tcube.h"
#include "utility.h"
```

```
// --------------------------------------------------------------------------------
// external globals
// --------------------------------------------------------------------------------
extern unsigned long  gPolygonCount;                    // counts how many polygons get
displayed each frame
extern bool           gWireFrameOnly;       // solid or wireframe drawing mode
extern vector<object3d*> gShapeVector;




// --------------------------------------------------------------------------------
// constructor
// --------------------------------------------------------------------------------
tcube::tcube()
{
        // Initialise the object's state
        setColour(1, 1, 1, 1.0);                                    // default colour

        vertexCount = 8;
        faceCount = 6;
        polygonCount = 6;

        // allocate memory for the vertex and face arrays
        vertices.resize(vertexCount);
        faces.resize(faceCount);


        // Initialise the tcube's vertices to create a tcube centered around the origin
        vertices[0].set(-1,  1,  1);                        // front, top, left
        vertices[1].set( 1,  1,  1);                        // front, top, right
        vertices[2].set( 1, -1,  1);                        // front, bot, right
        vertices[3].set(-1, -1,  1);                        // front, bot, left
        vertices[4].set(-1,  1, -1);                        // back,  top, left
        vertices[5].set( 1,  1, -1);                        // back,  top, right
        vertices[6].set( 1, -1, -1);                        // back,  bot, right
        vertices[7].set(-1, -1, -1);                        // back,  bot, left


        // now set up the faces - note that the vertex order is always
        // specified counter-clockwise when that face is viewed front on
        faces[0].init(0, 3, 2, 1);                                  // front
        faces[1].init(1, 2, 6, 5);                                  // right
        faces[2].init(5, 6, 7, 4);                                  // back
        faces[3].init(4, 7, 3, 0);                                  // left
        faces[4].init(4, 0, 1, 5);                                  // top
        faces[5].init(3, 7, 6, 2);                                  // bottom
```

```
	// calculate the face and vertex normals
	calculateNormals();


	// put the shape onto the shapeVector so it gets draw messages
	gShapeVector.push_back(this);
}



// ----------------------------------------------------------------------------
// privateDraw method
// ----------------------------------------------------------------------------
void tcube::privateDraw(bool drawMode)
{

	if (theTexture && !gWireFrameOnly)
	{
		glEnable(GL_TEXTURE_2D);
		glBindTexture(GL_TEXTURE_2D, theTexture->id());
		glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S,
GL_REPEAT);
		glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T,
GL_REPEAT);
		glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
theTexture->bestMagFilter());
		glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
theTexture->bestMinFilter());
		glMatrixMode(GL_TEXTURE);
		glLoadIdentity();
		glScalef(textureScaleX, textureScaleY, 1.0);
		glMatrixMode(GL_MODELVIEW);
	}


	glBegin(GL_QUADS);

	// front face -
	glNormal3fv(faces[0].faceNormal.data);
	glTexCoord2f(0.25,0.5);
	glVertex3fv(vertices[0].coordinate.data);		// top left
	glTexCoord2f(0.25,0);
	glVertex3fv(vertices[3].coordinate.data);		// bottom left
	glTexCoord2f(0.5,0);
	glVertex3fv(vertices[2].coordinate.data);		// bottom right
	glTexCoord2f(0.5,0.5);
	glVertex3fv(vertices[1].coordinate.data);		// top right
```

```
// right face
glNormal3fv(faces[1].faceNormal.data);
glTexCoord2f(0.25,0.5);
glVertex3fv(vertices[1].coordinate.data);          // top left
glTexCoord2f(0.25,0);
glVertex3fv(vertices[2].coordinate.data);          // bottom left
glTexCoord2f(0.5,0);
glVertex3fv(vertices[6].coordinate.data);          // bottom right
glTexCoord2f(0.5,0.5);
glVertex3fv(vertices[5].coordinate.data);          // top right

// back face
glNormal3fv(faces[2].faceNormal.data);
glTexCoord2f(0.25,0.5);
glVertex3fv(vertices[5].coordinate.data);          // top left
glTexCoord2f(0.25,0);
glVertex3fv(vertices[6].coordinate.data);          // bottom left
glTexCoord2f(0.5,0);
glVertex3fv(vertices[7].coordinate.data);          // bottom right
glTexCoord2f(0.5,0.5);
glVertex3fv(vertices[4].coordinate.data);          // top right

// left face
glNormal3fv(faces[3].faceNormal.data);
glTexCoord2f(0.25,0.5);
glVertex3fv(vertices[4].coordinate.data);          // top left
glTexCoord2f(0.25,0);
glVertex3fv(vertices[7].coordinate.data);          // bottom left
glTexCoord2f(0.5,0);
glVertex3fv(vertices[3].coordinate.data);          // bottom right
glTexCoord2f(0.5,0.5);
glVertex3fv(vertices[0].coordinate.data);          // top right

// top face
glNormal3fv(faces[4].faceNormal.data);
glTexCoord2f(0.25,0);
glVertex3fv(vertices[0].coordinate.data);          // bottom (front) left
glTexCoord2f(0.5,0);
glVertex3fv(vertices[1].coordinate.data);          // bottom (front) right
glTexCoord2f(0.5,0.5);
glVertex3fv(vertices[5].coordinate.data);          // top (back) right
glTexCoord2f(0.25,0.5);
glVertex3fv(vertices[4].coordinate.data);          // top (back) left

// bottom face
glNormal3fv(faces[5].faceNormal.data);
glTexCoord2f(0.25,0.5);
glVertex3fv(vertices[3].coordinate.data);          // top (front) left
```

```
        glTexCoord2f(0.25,0);
        glVertex3fv(vertices[7].coordinate.data);          // bottom (back) left
        glTexCoord2f(0.5,0);
        glVertex3fv(vertices[6].coordinate.data);          // bottom (back) right
        glTexCoord2f(0.5,0.5);
        glVertex3fv(vertices[2].coordinate.data);          // top (front) right

        glEnd();

        // disable texturing
        if (theTexture && !gWireFrameOnly) glDisable(GL_TEXTURE_2D);

}
```

tcube.h:

```
//
//////////////////////////////////////////////////////////////////////////////
// KXC354 - Computer Graphics & Animation - 2013
// Assignment 1 & 2
// 3D Engine Code
//
//////////////////////////////////////////////////////////////////////////////
//
// Author: Tony Gray
//
// tcube.h
//
// This file declares the class for textured cubes


#ifndef _tcube_H
#define _tcube_H




// ----------------------------------------------------------------------------
// application includes
// ----------------------------------------------------------------------------
#include "object3d.h"



// ----------------------------------------------------------------------------
```

```
// class declaration
// ----------------------------------------------------------------------------------------
class tcube : public object3d
{
        public:
                                                        tcube();
                        // default constructor

                virtual void            privateDraw(bool drawMode);
};



#endif // _tcube_H
```

CHAPTER 6 SCREEN SNAPSHOTS

'a' - start the test display suite

'f' - toggle fullscreen/window

'p' - pause/unpause

'w' - toggle wireframe/solid drawing

'v' - toggle "always draw vertices" mode

'q', ESC - quit

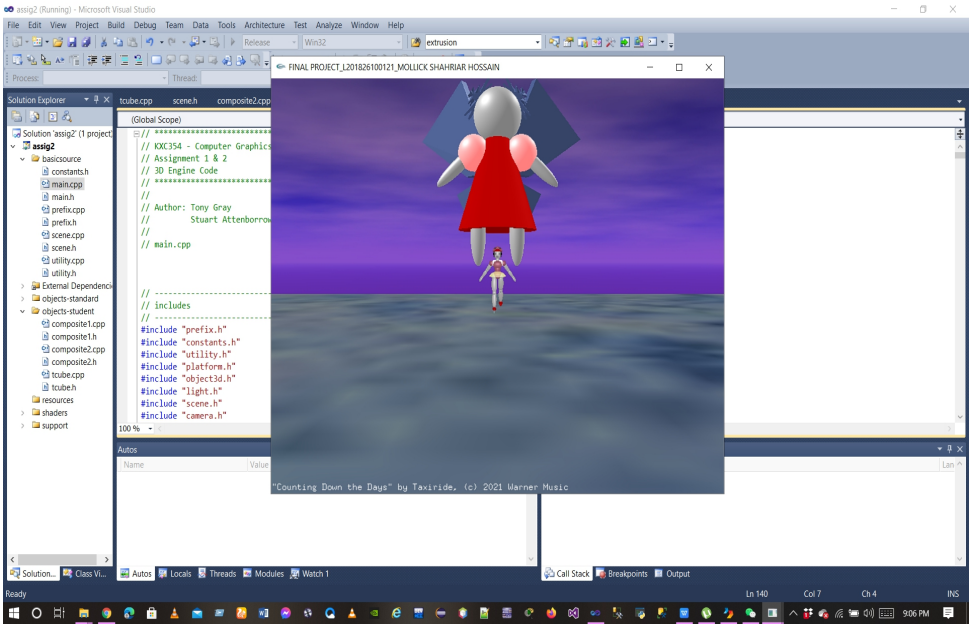
'x' - toggle display of local object axes

'X' - toggle display of global axes

'g' - toggle display of xz plane grid

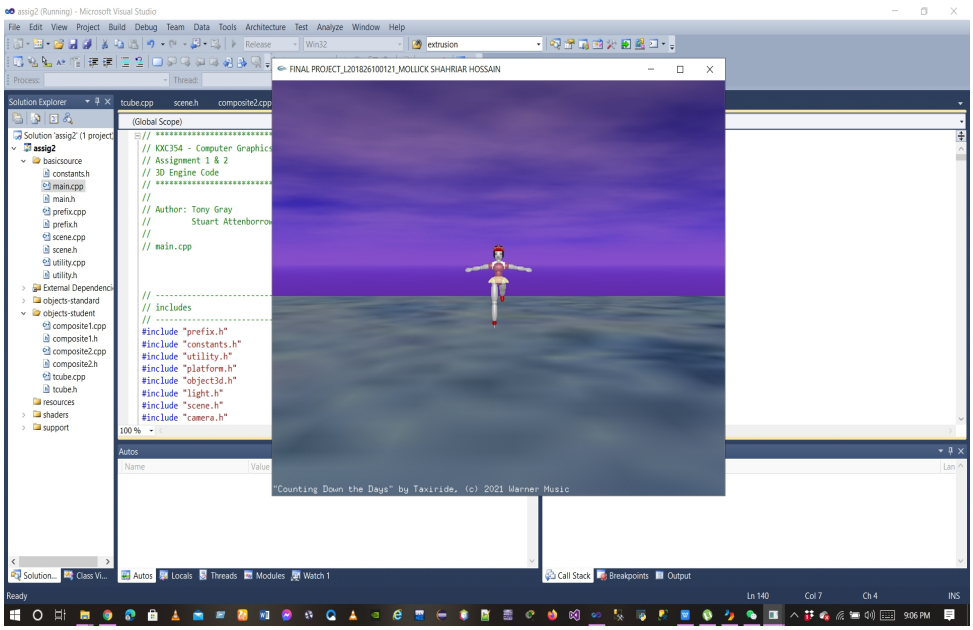
'n' - toggle display of face normals

'a' - start the test display suite:



'f' - toggle fullscreen/window:

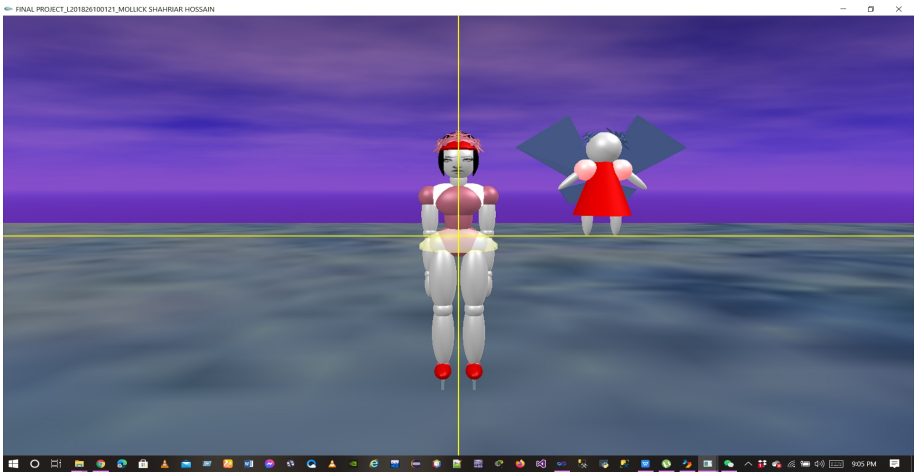
'p' - pause/unpause:



'v' - toggle "always draw vertices" mode:

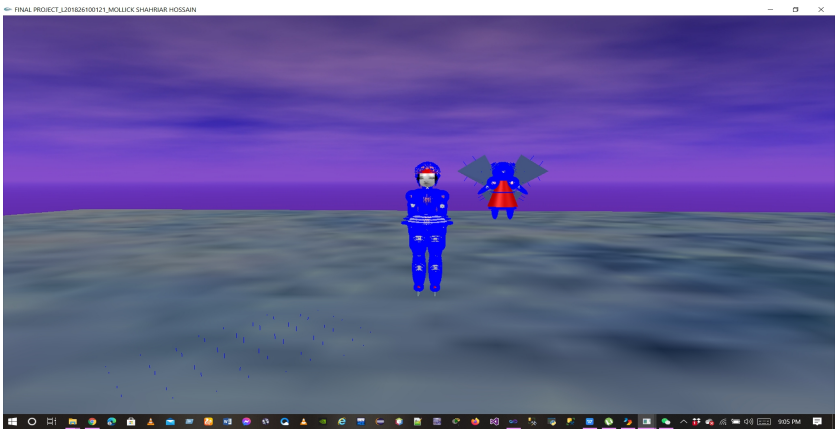
'x' - toggle display of local object axes:

'X' - toggle display of global axes:

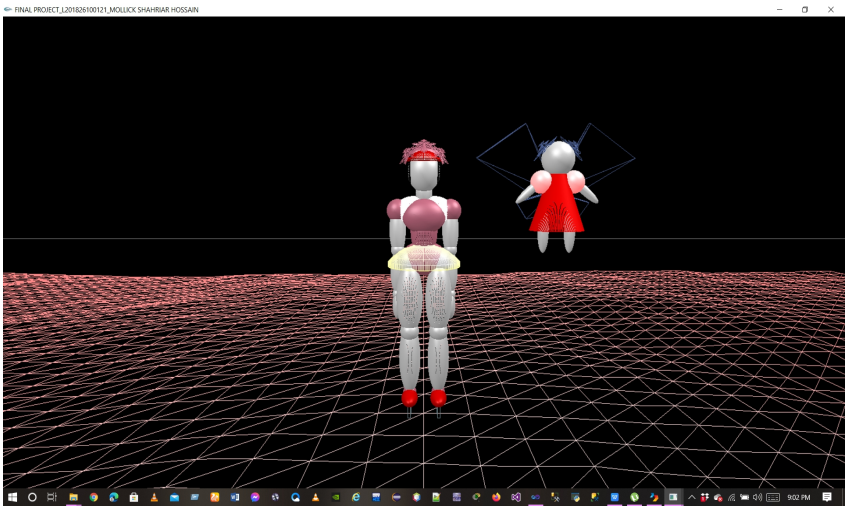


'g' - toggle display of xz plane grid:

'n' - toggle display of face normals:



'w' - toggle wireframe/solid drawing:



'q', ESC - quit:

NO output….press q output will be console display will be close…

This is an animation output so video will be more clear view than still pictures so video will be in zip.file.

Next page have next chapter…
Please be continue for read……..

CHAPTER 7 FUTURE ENHANCEMENTS

The following are some examples of potential future applications:

- The project may be created with fewer codes, and the mathematical functions may also be utilized to calculate the values, reducing the number of codes required.
- Additionally, it may be utilized to illustrate the operation of Dancing doll in an animated movie format.
- The output screen may be captured, and we can use it to create a short animated film to share with others.
- If we display the movement of both composite object for whole time , the animation will look much better.

Next page have next chapter…
Please be continue for read……..

CHAPTER 8 CONCLUSION

The project dubbed is being created with the best of our efforts in a limited amount of available time. Time constraints and a lack of coverage in the curriculum have caused the intricacy of this project to be under-furnished, preventing it from being completed to perfection. In order to guarantee that the package operates efficiently, the features contained inside it have been developed and tested, and they have been determined to be fairly good.

Finally, and most importantly, I want to express my gratitude to our teachers, who were a great source of inspiration as well as technical assistance who guided us throughout the course of the project, my college, which provided us with all of the resources we required, and my parents, who have been supportive in every way.

Constraints that apply generally Because the code was developed in Microsoft Visual Studio, which is only supported by Graphics Interface Operating Systems and not by Command Interface Operating Systems such as MS-DOS or UNIX, the base platform should be considered prior to the code being compiled. It goes without saying that the software must be both strong and quick.

It is expected that the standard output device, which is the monitor, is capable of supporting color. Consider the following: assumptions and dependencies. It is necessary for the user's machine to have the C++ compiler installed in the right versions

CHAPTER 9 BIBLIOGRAPHY

When looking for reference material, a variety of distinct sources were extensively investigated. The majority of our information comes from our reference book and the internet (numerous websites and pages including those of reputed universities and companies). The availability of paper presentations on the internet was a tremendous assistance. The help pages were quite helpful in gaining a thorough knowledge of the principles connected with C++programming.

Books referred:

1.Computer Graphics with Open GL - Hearn Baker Carithers (Fourth Edition)