

Graphics

by Ruaidhri David Cook

FILE	4148_RUAIDHRI_DAVID_COOK_GRAPHICS_110623_580802118.DOCX (8.98M)		
TIME SUBMITTED	05-APR-2019 08:47 PM (UTC+0100)	WORD COUNT	5709
SUBMISSION ID	104240422	CHARACTER COUNT	44769

Animated Fly-through Scene

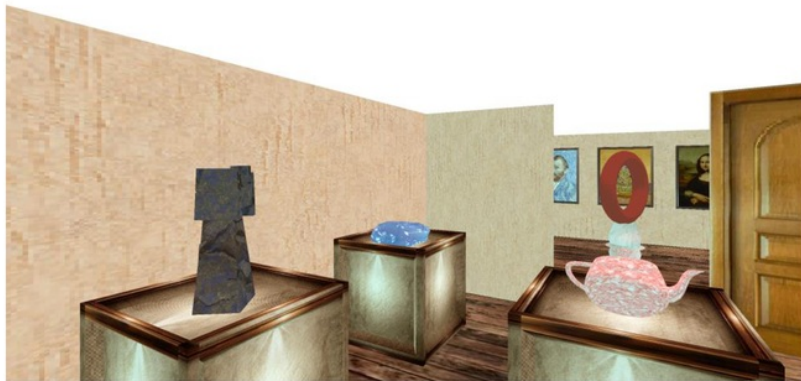
Student Number: P148970

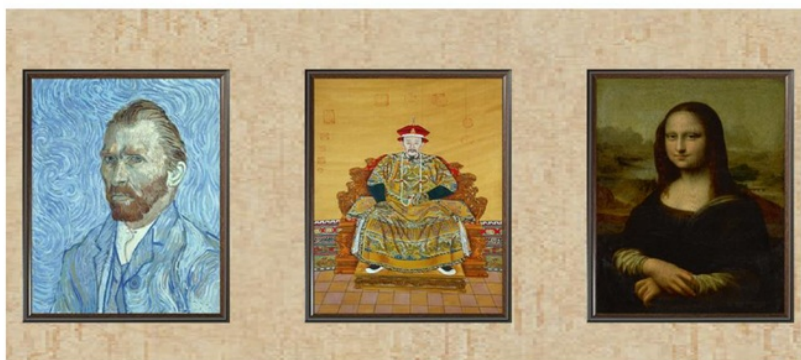
Student Name: Ruaidhri Cook

Lecturer: Gordon Dickers

Description

The scene consists of two rooms with textured walls and floors, the first of these rooms is the ornament gallery which has 5 different ornaments placed on plinths, the ornaments and the plinths are objects that have been rendered in, apart from the teapot which is an object that can be created through OpenGL – all of these objects use unique textures. At the back of the ornament gallery there is also an open textured door that leads the room with paintings. This room also has a rotating donut in the centre that is being suspended on stand below it. In the painting gallery there are three portrait paintings on the back wall, with 2 landscape paintings on the right and left walls. All the paintings have a painting frame in addition to the painting itself.





Source code used

The main source code that was studied to create the program was from the `ogl_scene_dev_examples` on moodle. Code from these examples was chopped and changed so it was suitable for the scene that was created. In addition to this the `obj_loader` from moodle was utilised and the `objloader` source used so that unique objects could be included within the scene.

Project Source Code

Printout of code

The code for the source.cpp is in the appendix, it shows the code that creates the specified scene.

How to compile

The code is pre-compiled, to open the scene the user must run the Graphics – Shortcut.exe file found in the root folder.

How to drive

The navigation only has limited controls. That is the W key to move forwards, and S key to move backwards. The user can also press the A key to move left and the D key to move right. The Z key always the user to rotate the camera left and the X key allows the user to rotate the camera right.

Structure and design

Description of code design

The structure for the code is inspired by the examples that are on moodle, namely the file named '6-lit scene with textures and animation' in the og_scene_dev_examples zip file.

This structure starts with definitions, such as the Universal bitmap ID which is needed for the scene.

After this the Includes are used, making use of the glut header and objloader header so the scene can be created, and the objects can be loaded into the scene.

The variables needed for the texture information are then initialised.

This is followed by the initialisation of the global variables, such as the global angle and the array that contains the objects used in the scene.

The functions are then initialised.

The variables needed to change the colour of the light are then initialised, such as the ambient light, diffuse light and the light position itself.

The first function is the initialisation function. This function first calls the GL functions that do the setup for the lighting, it then calls the function to enable the 2D texturing of the objects within the scene. After this the first 24-bit bitmap file is loaded and the 20 texture objects needed for the scene are generated. Following this the first texture is enabled through the `glBindTexture` and `glTexParameterf` functions. The texture image is then generated using the `glTexImage2D` function. This process is then repeated for all 20 textures, the difference being that the texture objects aren't generated each time. Following this, the 7 objects that will be rendered into the scene are loaded in using the functions specified in the `objloader.cpp`.

The next function is the one used to draw a box by passing in width, height and length. First the texture object is enabled. For each plane the coordinates and vertex are specified, these being the front, back, left, right, top and bottom planes.

The next function follows this same structure, however with the textures for a door instead.

The next seven functions are for each painting in the scene, consisting of just one plane and the texture for a painting.

The following function is for a picture frame, which follows the same structure as the painting, but with a picture frame texture instead.

The next function is for the display, which includes the GL function calls that draw the scene. The walls and floor of the first room are drawn first, each wall is drawn by first specifying the draw position, the rotation orientation and the material properties, such as ambient and diffuse, the wall is then drawn using the `drawBox` function. The door is then drawn using the same method but using the `drawDoor` function instead. After this the plinth objects are loaded in, the texture for the following object is first specified, then the draw position, the object is specified and rendered. The ornaments are loaded in the same way with only the object being rendered changing. A teapot is also drawn using the `glutSolidTeapot` function instead of object render. The second room walls and floor are then drawn afterwards in the same way as the first room. The centre stand is then rendered in the same way as the plinths and ornaments. The pictures and their frames are then drawn using the same format as the walls and floor, the difference being that `drawPicture` and

drawFrame are the functions called, with the position of the picture frame slightly behind the picture itself. After this the donut is rendered in in the same fashion as the other objects, however it uses the global angle when glRotatef is called which allows for idle rotation. The function glFlush is then called to force the draw. The back buffer is then brought to the foreground.

The next function handles reshaping the window. It calls glut functions to change the display accordingly when the window is resized, so the viewport fills the window.

The next function is for navigation. It handles keyboard input by using cases that will change the camera x and z axis, as well as the viewing angle, based upon input; this is done by adding or subtracting from the x or z or angle variables and then calling the camera function.

The next function, spinDisplay, changes the global angle, which in turn causes the donut to spin.

There is then the main function that calls all of the other functions to setup the scene from launch. One such function is glutIdleFunc, which uses the spinDisplay the spin the donut whilst the scene is idle.

The last functions deal with the Loading and Writing of Bitmaps.

Testing

Test Num	Test	Test Data	Expected Outcome	Actual Outcome	Pass/Fail ?
1	Test donut rotation	glRotatef(globangle, 0, 0, 1);	Donut rotates right on Z axis	Donut rotated right on Z axis	Pass
2	Test donut rotation	glRotatef(globangle, 1, 0, 1);	Donut rotates right on X and Zaxis	Donut rotated right on X and Zaxis	Pass
3	Test object render	objarray[5]->render();	Renders donut into scene	Rendered donut into scene	Pass
4	Test plane draw	drawPicture7(2, 0.01, 2.5);	Draws painting with texture on back wall	Drew painting with texture on back wall	Pass
5	Test pinth texture	glBindTexture(GL_TEXTURE_2D, texnum[0]);	Changes Texture of plinth object	Changed Texture of plinth object	Pass
6	Test rotate cat ornament	glRotatef(-90, 0, 1, 0);	Cat ornament rotates 90 degrees to the left	Object rotated 90 degrees to the left	Pass
7	Test spin speed	globangle = globangle + 0.3f;	Donut will rotate faster than if it were 0.1f	Donut rotated faster than if it were 0.1f	Pass

Appendix

```
//Defines

#define BITMAP_ID 0x4D42    // the universal bitmap ID


// Includes

#include "objloader.h"

#include <math.h>

#include <stdio.h>

#include <stdlib.h>

#include "glut.h"

#include <windows.h>


// Texture Information

BITMAPINFOHEADER bitmapInfoHeader; // bitmap info header (needs windows.h to
define type

unsigned char*    bitmapData;      // the texture data

static GLuint    texnum[2];        // array of 2 texture object names


                                // Global Variables

float globangle = 0;

object_type *objarray[8]; //objects container for the museum scene


                                //function prototypes

unsigned char *LoadBitmapFile(char *filename, BITMAPINFOHEADER *bitmapInfoHeader);

int WriteBitmapFile(char *filename, int width, int height, unsigned char *imageData);

void init(void);

void display(void);

void drawBox(float w, float h, float l);

void drawDoor(float w, float h, float l);
```

```
void drawPicture1(float w, float h, float l);  
void drawPicture2(float w, float h, float l);  
void drawPicture3(float w, float h, float l);  
void drawPicture4(float w, float h, float l);  
void drawPicture5(float w, float h, float l);  
void drawPicture6(float w, float h, float l);  
void drawPicture7(float w, float h, float l);  
void drawFrame(float w, float h, float l);  
void reshape(int w, int h);  
void camera(float, float, float, float);
```

```
//FOR CHANGING THE COLOUR OF THE LIGHT
```

```
//// Lighting variables
```

```
float ambientLight[] = { 0.6f, 1.0f, 1.0f, 1.0f }; // ambient light
```

```
float diffuseLight[] = { 1.0f, 1.0f, 1.0f, 1.0f }; // diffuse light
```

```
float lightPosition[] = { 10.0f, 100.0f, 10.0f, 0.0f }; // the light position
```

```
//// Material variables
```

```
float matAmbient[] = { 1.0f, 1.0f, 1.0f, 1.0f };  
float matDiff[] = { 1.0f, 1.0f, 1.0f, 1.0f };  
float matAmbient1[] = { 1.0f, 1.0f, 1.0f, 1.0f };  
float matDiff1[] = { 1.0f, 1.0f, 1.0f, 1.0f };  
float matAmbient2[] = { 1.0f, 1.0f, 1.0f, 1.0f };  
float matDiff2[] = { 1.0f, 1.0f, 1.0f, 1.0f };  
  
void init()  
// Initialize
```

```

// desc: initializes OpenGL
{
    glClearColor(1.0f, 1.0f, 1.0f, 0.0f);    // clears colour buffer to white
    glEnable(GL_DEPTH_TEST);                // hidden surface removal
    glEnable(GL_CULL_FACE);                  // do not calculate inside of poly's
    glFrontFace(GL_CCW);                     // counter clock-wise polygons are out

                                //***** lighting
    glEnable(GL_SMOOTH);
    glEnable(GL_LIGHTING);                   // enable lighting

                                // Setup the materials for LIGHT0
    glMaterialfv(GL_FRONT, GL_AMBIENT, matAmbient);
    glMaterialfv(GL_FRONT, GL_DIFFUSE, matDiff);

    // sets up LIGHT0
    glLightfv(GL_LIGHT0, GL_AMBIENT, ambientLight);    // sets up the ambient element
    glLightfv(GL_LIGHT0, GL_DIFFUSE, diffuseLight);    // sets up the diffuse element
    glLightfv(GL_LIGHT0, GL_POSITION, lightPosition);  // place the light in the world

                                // enables the light
    glEnable(GL_LIGHT0);

    // enables 2D texturing
    glEnable(GL_TEXTURE_2D);

    //TEXTURES USED
    // loads bitmap file

```

```
bitmapData = LoadBitmapFile("wall.bmp", &bitmapInfoHeader);

// generates 20 texture objects
glGenTextures(20, texnum);

// enable first texture object
glBindTexture(GL_TEXTURE_2D, texnum[0]);

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);

// generate the texture image
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, bitmapInfoHeader.biWidth,
             bitmapInfoHeader.biHeight, 0, GL_RGB, GL_UNSIGNED_BYTE, bitmapData);

/*2nd texture*****8

bitmapData = LoadBitmapFile("floor.bmp", &bitmapInfoHeader);

glBindTexture(GL_TEXTURE_2D, texnum[1]);    // enables texture object

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);

// generate the texture image
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, bitmapInfoHeader.biWidth,
             bitmapInfoHeader.biHeight, 0, GL_RGB, GL_UNSIGNED_BYTE, bitmapData);
```

```

//*****end of 2nd
tex*****g

//*3rd texture*****g

bitmapData = LoadBitmapFile("teapot.bmp", &bitmapInfoHeader);

glBindTexture(GL_TEXTURE_2D, texnum[2]);    // enables texture object

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);

// generate the texture image
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, bitmapInfoHeader.biWidth,
    bitmapInfoHeader.biHeight, 0, GL_RGB, GL_UNSIGNED_BYTE, bitmapData);

//*****end of 3rd
tex*****g

//*4th texture*****g

bitmapData = LoadBitmapFile("stone.bmp", &bitmapInfoHeader);

glBindTexture(GL_TEXTURE_2D, texnum[3]);    // enables texture object

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);

```

```

// generate the texture image
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, bitmapInfoHeader.biWidth,
             bitmapInfoHeader.biHeight, 0, GL_RGB, GL_UNSIGNED_BYTE, bitmapData);

//*****end of 4th
tex*****g

//*5th texture*****g

bitmapData = LoadBitmapFile("museumWall.bmp", &bitmapInfoHeader);

glBindTexture(GL_TEXTURE_2D, texnum[4]);    // enables texture object

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);

// generate the texture image
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, bitmapInfoHeader.biWidth,
             bitmapInfoHeader.biHeight, 0, GL_RGB, GL_UNSIGNED_BYTE, bitmapData);

//*****end of 5th
tex*****g

//*6th texture*****g

bitmapData = LoadBitmapFile("crystal.bmp", &bitmapInfoHeader);

```

```

glBindTexture(GL_TEXTURE_2D, texnum[5]);    // enables texture object

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);

// generate the texture image
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, bitmapInfoHeader.biWidth,
             bitmapInfoHeader.biHeight, 0, GL_RGB, GL_UNSIGNED_BYTE, bitmapData);

//*****end of 6th
tex*****8

//*7th texture*****8

bitmapData = LoadBitmapFile("precious.bmp", &bitmapInfoHeader);

glBindTexture(GL_TEXTURE_2D, texnum[6]);    // enables texture object

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);

// generate the texture image
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, bitmapInfoHeader.biWidth,
             bitmapInfoHeader.biHeight, 0, GL_RGB, GL_UNSIGNED_BYTE, bitmapData);

//*****end of 7th
tex*****8

//*8th texture*****8

```

```

bitmapData = LoadBitmapFile("gold.bmp", &bitmapInfoHeader);

glBindTexture(GL_TEXTURE_2D, texnum[7]);    // enables texture object

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);

// generate the texture image
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, bitmapInfoHeader.biWidth,
             bitmapInfoHeader.biHeight, 0, GL_RGB, GL_UNSIGNED_BYTE, bitmapData);

//*****end of 8th
tex*****g

//*9th texture*****g

bitmapData = LoadBitmapFile("door.bmp", &bitmapInfoHeader);

glBindTexture(GL_TEXTURE_2D, texnum[8]);    // enables texture object

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);

// generate the texture image
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, bitmapInfoHeader.biWidth,
             bitmapInfoHeader.biHeight, 0, GL_RGB, GL_UNSIGNED_BYTE, bitmapData);

```

```

//*****end of 9th
tex*****

//*10th texture*****g

bitmapData = LoadBitmapFile("painting1.bmp", &bitmapInfoHeader);

glBindTexture(GL_TEXTURE_2D, texnum[9]);    // enables texture object

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);

// generate the texture image
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, bitmapInfoHeader.biWidth,
             bitmapInfoHeader.biHeight, 0, GL_RGB, GL_UNSIGNED_BYTE, bitmapData);

//*****end of 10th
tex*****

//*11th texture*****g

bitmapData = LoadBitmapFile("painting2.bmp", &bitmapInfoHeader);

glBindTexture(GL_TEXTURE_2D, texnum[10]);    // enables texture object

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);

// generate the texture image

```

```

glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, bitmapInfoHeader.biWidth,
             bitmapInfoHeader.biHeight, 0, GL_RGB, GL_UNSIGNED_BYTE, bitmapData);

//*****end of 11th
tex*****

//*12th texture*****8

bitmapData = LoadBitmapFile("painting3.bmp", &bitmapInfoHeader);

glBindTexture(GL_TEXTURE_2D, texnum[11]);    // enables texture object

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);

// generate the texture image
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, bitmapInfoHeader.biWidth,
             bitmapInfoHeader.biHeight, 0, GL_RGB, GL_UNSIGNED_BYTE, bitmapData);

//*****end of 12th
tex*****

//*13th texture*****8

bitmapData = LoadBitmapFile("painting4.bmp", &bitmapInfoHeader);

glBindTexture(GL_TEXTURE_2D, texnum[12]);    // enables texture object

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);

```

```

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);

// generate the texture image
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, bitmapInfoHeader.biWidth,
    bitmapInfoHeader.biHeight, 0, GL_RGB, GL_UNSIGNED_BYTE, bitmapData);

//*****end of 13th
tex*****

//*14th texture*****8

bitmapData = LoadBitmapFile("painting5.bmp", &bitmapInfoHeader);

glBindTexture(GL_TEXTURE_2D, texnum[13]);    // enables texture object

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);

// generate the texture image
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, bitmapInfoHeader.biWidth,
    bitmapInfoHeader.biHeight, 0, GL_RGB, GL_UNSIGNED_BYTE, bitmapData);

//*****end of 14th
tex*****

//*15th texture*****8

bitmapData = LoadBitmapFile("painting 6.bmp", &bitmapInfoHeader);

```

```

glBindTexture(GL_TEXTURE_2D, texnum[14]);    // enables texture object

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);

// generate the texture image
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, bitmapInfoHeader.biWidth,
             bitmapInfoHeader.biHeight, 0, GL_RGB, GL_UNSIGNED_BYTE, bitmapData);

//*****end of 15th
tex*****

//*16th texture*****8

bitmapData = LoadBitmapFile("painting7.bmp", &bitmapInfoHeader);

glBindTexture(GL_TEXTURE_2D, texnum[15]);    // enables texture object

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);

// generate the texture image
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, bitmapInfoHeader.biWidth,
             bitmapInfoHeader.biHeight, 0, GL_RGB, GL_UNSIGNED_BYTE, bitmapData);

//*****end of 16th
tex*****

//*17th texture*****8

```

```

bitmapData = LoadBitmapFile("frame.bmp", &bitmapInfoHeader);

glBindTexture(GL_TEXTURE_2D, texnum[16]);    // enables texture object

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);

// generate the texture image
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, bitmapInfoHeader.biWidth,
             bitmapInfoHeader.biHeight, 0, GL_RGB, GL_UNSIGNED_BYTE, bitmapData);

//*****end of 17th
tex*****

//*18th texture*****g

bitmapData = LoadBitmapFile("bone.bmp", &bitmapInfoHeader);

glBindTexture(GL_TEXTURE_2D, texnum[17]);    // enables texture object

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);

// generate the texture image
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, bitmapInfoHeader.biWidth,
             bitmapInfoHeader.biHeight, 0, GL_RGB, GL_UNSIGNED_BYTE, bitmapData);

//*****end of 18th
tex*****

```

```

// *19th texture*****8

bitmapData = LoadBitmapFile("painting7.bmp", &bitmapInfoHeader);

glBindTexture(GL_TEXTURE_2D, texnum[18]);    // enables texture object

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);

// generate the texture image
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, bitmapInfoHeader.biWidth,
             bitmapInfoHeader.biHeight, 0, GL_RGB, GL_UNSIGNED_BYTE, bitmapData);

// *****end of 19th
tex*****

// *20th texture*****8

bitmapData = LoadBitmapFile("painting 1.bmp", &bitmapInfoHeader);

glBindTexture(GL_TEXTURE_2D, texnum[19]);    // enables texture object

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);

// generate the texture image

```

```
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, bitmapInfoHeader.biWidth,
             bitmapInfoHeader.biHeight, 0, GL_RGB, GL_UNSIGNED_BYTE, bitmapData);

//*****end of 20th
tex*****
```

```
//OBJECTS USED
```

```
printf("*****\n");
objarray[0] = new (object_type);
int fp = objarray[0]->objloader("box.obj");
objarray[0]->objdatadisplay();

objarray[1] = new (object_type);
fp = objarray[1]->objloader("ornament1.obj");
objarray[1]->objdatadisplay();

objarray[2] = new (object_type);
fp = objarray[2]->objloader("rock1.obj");
objarray[2]->objdatadisplay();

objarray[3] = new (object_type);
fp = objarray[3]->objloader("rockset.obj");
objarray[3]->objdatadisplay();

objarray[4] = new (object_type);
fp = objarray[4]->objloader("cat.obj");
objarray[4]->objdatadisplay();
```

```

objarray[5] = new (object_type);
fp = objarray[5]->objloader("donut.obj");
objarray[5]->objdatadisplay();

objarray[6] = new (object_type);
fp = objarray[6]->objloader("vase.obj");
objarray[6]->objdatadisplay();

}

void drawBox(float w, float h, float l)
//Draws a box of width w, height h, length l
{
    glPushMatrix();    //save modelview
    w = w / 2.0; h = h / 2.0; l = l / 2.0; //adjust values so centre is in middle of box
    glBindTexture(GL_TEXTURE_2D, texnum[1]);    // enables texture object
    glBegin(GL_POLYGON);
    glNormal3f(0.0f, 1.0f, 0.0f);    // top face
    glTexCoord2f(0.0f, 0.0f); glVertex3f(w, l, h);
    glTexCoord2f(1.0f, 0.0f); glVertex3f(w, l, -h);
    glTexCoord2f(1.0f, 1.0f); glVertex3f(-w, l, -h);
    glTexCoord2f(0.0f, 1.0f); glVertex3f(-w, l, h);
    glEnd();

    glBindTexture(GL_TEXTURE_2D, texnum[4]);    // enables texture object
    glBegin(GL_POLYGON);

```

```
glNormal3f(0.0f, 0.0f, 1.0f); // front face
glTexCoord2f(0.0f, 0.0f); glVertex3f(w, l, h);
glTexCoord2f(1.0f, 0.0f); glVertex3f(-w, l, h);
glTexCoord2f(1.0f, 1.0f); glVertex3f(-w, -l, h);
glTexCoord2f(0.0f, 1.0f); glVertex3f(w, -l, h);
glEnd();

glBegin(GL_POLYGON);
glNormal3f(-1.0f, 0.0f, 0.0f); // left face
glTexCoord2f(0.0f, 0.0f); glVertex3f(-w, l, h);
glTexCoord2f(1.0f, 0.0f); glVertex3f(-w, l, -h);
glTexCoord2f(1.0f, 1.0f); glVertex3f(-w, -l, -h);
glTexCoord2f(0.0f, 1.0f); glVertex3f(-w, -l, h);
glEnd();

glBegin(GL_POLYGON);
glNormal3f(1.0f, 0.0f, 0.0f); // right face
glTexCoord2f(0.0f, 0.0f); glVertex3f(w, l, h);
glTexCoord2f(1.0f, 0.0f); glVertex3f(w, -l, h);
glTexCoord2f(1.0f, 1.0f); glVertex3f(w, -l, -h);
glTexCoord2f(0.0f, 1.0f); glVertex3f(w, l, -h);
glEnd();

glBegin(GL_POLYGON);
glNormal3f(0.0f, 0.0f, -1.0f); // back face
glTexCoord2f(0.0f, 0.0f); glVertex3f(w, -l, -h);
glTexCoord2f(1.0f, 0.0f); glVertex3f(-w, -l, -h);
glTexCoord2f(1.0f, 1.0f); glVertex3f(-w, l, -h);
glTexCoord2f(0.0f, 1.0f); glVertex3f(w, l, -h);
glEnd();
```

```

glBindTexture(GL_TEXTURE_2D, texnum[8]);
glBegin(GL_POLYGON);
glNormal3f(0.0f, -1.0f, 0.0f); // bottom face
glTexCoord2f(0.0f, 0.0f); glVertex3f(-w, -l, h);
glTexCoord2f(1.0f, 0.0f); glVertex3f(-w, -l, -h);
glTexCoord2f(1.0f, 1.0f); glVertex3f(w, -l, -h);
glTexCoord2f(0.0f, 1.0f); glVertex3f(w, -l, h);
glEnd();

glPopMatrix(); //restore previous modelview matrix
}

void drawDoor(float w, float h, float l)
//Draws a door of width w, height h, length l
{
    glPushMatrix(); //save modelview

    glBindTexture(GL_TEXTURE_2D, texnum[8]); // enables texture object

    glBegin(GL_POLYGON);
    glNormal3f(0.0f, 0.0f, 1.0f); // front face
    glTexCoord2f(0.0f, 0.0f); glVertex3f(w, l, h);
    glTexCoord2f(1.0f, 0.0f); glVertex3f(-w, l, h);
    glTexCoord2f(1.0f, 1.0f); glVertex3f(-w, -l, h);
    glTexCoord2f(0.0f, 1.0f); glVertex3f(w, -l, h);
    glEnd();

    glBegin(GL_POLYGON);
    glNormal3f(-1.0f, 0.0f, 0.0f); // left face

```

```

    glTexCoord2f(0.0f, 0.0f); glVertex3f(-w, l, h);
    glTexCoord2f(1.0f, 0.0f); glVertex3f(-w, l, -h);
    glTexCoord2f(1.0f, 1.0f); glVertex3f(-w, -l, -h);
    glTexCoord2f(0.0f, 1.0f); glVertex3f(-w, -l, h);
    glEnd();

    glBegin(GL_POLYGON);
    glNormal3f(1.0f, 0.0f, 0.0f); // right face
    glTexCoord2f(0.0f, 0.0f); glVertex3f(w, l, h);
    glTexCoord2f(1.0f, 0.0f); glVertex3f(w, -l, h);
    glTexCoord2f(1.0f, 1.0f); glVertex3f(w, -l, -h);
    glTexCoord2f(0.0f, 1.0f); glVertex3f(w, l, -h);
    glEnd();

    glBegin(GL_POLYGON);
    glNormal3f(0.0f, 0.0f, -1.0f); // back face
    glTexCoord2f(0.0f, 0.0f); glVertex3f(w, -l, -h);
    glTexCoord2f(1.0f, 0.0f); glVertex3f(-w, -l, -h);
    glTexCoord2f(1.0f, 1.0f); glVertex3f(-w, l, -h);
    glTexCoord2f(0.0f, 1.0f); glVertex3f(w, l, -h);
    glEnd();

    glPopMatrix(); //restore previous modelview matrix
}

void drawPicture1(float w, float h, float l)
//Draws a painting of width w, height h, length l
{

```

```

glPushMatrix();    //save modelview
w = w / 2.0; h = h / 2.0; l = l / 2.0; //adjust values so centre is in middle of box
glBindTexture(GL_TEXTURE_2D, texnum[9]);    // enables texture object

glBegin(GL_POLYGON);
glNormal3f(0.0f, 0.0f, -1.0f);    // back face
glTexCoord2f(0.0f, 0.0f); glVertex3f(w, -l, -h);
glTexCoord2f(1.0f, 0.0f); glVertex3f(-w, -l, -h);
glTexCoord2f(1.0f, 1.0f); glVertex3f(-w, l, -h);
glTexCoord2f(0.0f, 1.0f); glVertex3f(w, l, -h);
glEnd();

glPopMatrix();    //restore previous modelview matrix
}

void drawPicture2(float w, float h, float l)
//Draws a painting of width w, height h, length l
{
    glPushMatrix();    //save modelview
    w = w / 2.0; h = h / 2.0; l = l / 2.0; //adjust values so centre is in middle of box
    glBindTexture(GL_TEXTURE_2D, texnum[10]);    // enables texture object

    glBegin(GL_POLYGON);
    glNormal3f(0.0f, 0.0f, -1.0f);    // back face
    glTexCoord2f(0.0f, 0.0f); glVertex3f(w, -l, -h);
    glTexCoord2f(1.0f, 0.0f); glVertex3f(-w, -l, -h);
    glTexCoord2f(1.0f, 1.0f); glVertex3f(-w, l, -h);
    glTexCoord2f(0.0f, 1.0f); glVertex3f(w, l, -h);
    glEnd();
}

```

```

    glPopMatrix(); //restore previous modelview matrix
}

void drawPicture3(float w, float h, float l)
//Draws a painting of width w, height h, length l
{
    glPushMatrix();    //save modelview
    w = w / 2.0; h = h / 2.0; l = l / 2.0; //adjust values so centre is in middle of box
    glBindTexture(GL_TEXTURE_2D, texnum[11]);    // enables texture object

    glBegin(GL_POLYGON);
    glNormal3f(0.0f, 0.0f, -1.0f); // back face
    glTexCoord2f(0.0f, 0.0f); glVertex3f(w, -l, -h);
    glTexCoord2f(1.0f, 0.0f); glVertex3f(-w, -l, -h);
    glTexCoord2f(1.0f, 1.0f); glVertex3f(-w, l, -h);
    glTexCoord2f(0.0f, 1.0f); glVertex3f(w, l, -h);
    glEnd();

    glPopMatrix(); //restore previous modelview matrix
}

void drawPicture4(float w, float h, float l)
//Draws a painting of width w, height h, length l
{
    glPushMatrix();    //save modelview
    w = w / 2.0; h = h / 2.0; l = l / 2.0; //adjust values so centre is in middle of box
    glBindTexture(GL_TEXTURE_2D, texnum[12]);    // enables texture object

    glBegin(GL_POLYGON);
    glNormal3f(0.0f, 0.0f, -1.0f); // back face

```

```

    glTexCoord2f(0.0f, 0.0f); glVertex3f(w, -l, -h);
    glTexCoord2f(1.0f, 0.0f); glVertex3f(-w, -l, -h);
    glTexCoord2f(1.0f, 1.0f); glVertex3f(-w, l, -h);
    glTexCoord2f(0.0f, 1.0f); glVertex3f(w, l, -h);
    glEnd();
    glPopMatrix(); //restore previous modelview matrix
}

```

```

void drawPicture5(float w, float h, float l)
//Draws a painting of width w, height h, length l
{
    glPushMatrix();    //save modelview
    w = w / 2.0; h = h / 2.0; l = l / 2.0; //adjust values so centre is in middle of box
    glBindTexture(GL_TEXTURE_2D, texnum[13]);    // enables texture object

    glBegin(GL_POLYGON);
    glNormal3f(0.0f, 0.0f, -1.0f); // back face
    glTexCoord2f(0.0f, 0.0f); glVertex3f(w, -l, -h);
    glTexCoord2f(1.0f, 0.0f); glVertex3f(-w, -l, -h);
    glTexCoord2f(1.0f, 1.0f); glVertex3f(-w, l, -h);
    glTexCoord2f(0.0f, 1.0f); glVertex3f(w, l, -h);
    glEnd();
    glPopMatrix(); //restore previous modelview matrix
}

```

```

void drawPicture6(float w, float h, float l)
//Draws a painting of width w, height h, length l
{
    glPushMatrix();    //save modelview

```

```

w = w / 2.0; h = h / 2.0; l = l / 2.0; //adjust values so centre is in middle of box
glBindTexture(GL_TEXTURE_2D, texnum[14]);    // enables texture object

glBegin(GL_POLYGON);
glNormal3f(0.0f, 0.0f, -1.0f); // back face
glTexCoord2f(0.0f, 0.0f); glVertex3f(w, -l, -h);
glTexCoord2f(1.0f, 0.0f); glVertex3f(-w, -l, -h);
glTexCoord2f(1.0f, 1.0f); glVertex3f(-w, l, -h);
glTexCoord2f(0.0f, 1.0f); glVertex3f(w, l, -h);
glEnd();
glPopMatrix(); //restore previous modelview matrix
}

```

```

void drawPicture7(float w, float h, float l)
//Draws a painting of width w, height h, length l
{
    glPushMatrix();    //save modelview
    w = w / 2.0; h = h / 2.0; l = l / 2.0; //adjust values so centre is in middle of box
    glBindTexture(GL_TEXTURE_2D, texnum[18]);    // enables texture object

    glBegin(GL_POLYGON);
    glNormal3f(0.0f, 0.0f, -1.0f); // back face
    glTexCoord2f(0.0f, 0.0f); glVertex3f(w, -l, -h);
    glTexCoord2f(1.0f, 0.0f); glVertex3f(-w, -l, -h);
    glTexCoord2f(1.0f, 1.0f); glVertex3f(-w, l, -h);
    glTexCoord2f(0.0f, 1.0f); glVertex3f(w, l, -h);
    glEnd();
    glPopMatrix(); //restore previous modelview matrix
}

```

```
void drawFrame(float w, float h, float l)
//Draws a painting frame of width w, height h, length l
{
    glPushMatrix();    //save modelview
    w = w / 2.0; h = h / 2.0; l = l / 2.0; //adjust values so centre is in middle of box
    glBindTexture(GL_TEXTURE_2D, texnum[16]);    // enables texture object

    glBegin(GL_POLYGON);
    glNormal3f(0.0f, 0.0f, -1.0f);    // back face
    glTexCoord2f(0.0f, 0.0f); glVertex3f(w, -l, -h);
    glTexCoord2f(1.0f, 0.0f); glVertex3f(-w, -l, -h);
    glTexCoord2f(1.0f, 1.0f); glVertex3f(-w, l, -h);
    glTexCoord2f(0.0f, 1.0f); glVertex3f(w, l, -h);
    glEnd();
    glPopMatrix();    //restore previous modelview matrix
}

// display
//*****
// desc: handles drawing of scene

void display()
{
    glPushMatrix();

    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);    // clear screen and depth
    buffer

    // reset modelview matrix
```

//FIRST ROOM

//FLOOR

glColor3f(0.8, 0.8, 1.0); //set colour not used. Automatically disabled when lighting Enabled

glTranslatef(0, 0, 0); //move by x,y and z units

glMaterialfv(GL_FRONT, GL_AMBIENT, matAmbient); //set backdrop material properties

glMaterialfv(GL_FRONT, GL_DIFFUSE, matDiff);

drawBox(10.0, 10.0, 0.01); //draw floor

glColor3f(1.0, 0.8, 0.8); //set colour not used. Automatically disabled when lighting Enabled

//LEFT

glPushMatrix(); //save current position/orientation, etc.

glTranslatef(-5, 2.5, 0); //move by x,y and z units

glRotatef(90, 0, 1, 0);

glMaterialfv(GL_FRONT, GL_AMBIENT, matAmbient1); //set backdrop material properties

glMaterialfv(GL_FRONT, GL_DIFFUSE, matDiff1);

drawBox(10, 0.01, 5); //draw left wall

glPopMatrix(); //restore previous "current" position/orientation etc.

//RIGHT

glPushMatrix(); //save current position/orientation, etc.

glTranslatef(5, 2.5, 0); //move by x,y and z units

glRotatef(-90, 0, 1, 0);

glMaterialfv(GL_FRONT, GL_AMBIENT, matAmbient1); //set backdrop material properties

glMaterialfv(GL_FRONT, GL_DIFFUSE, matDiff1);

drawBox(10, 0.01, 5); //draw right wall

glPopMatrix(); //restore previous "current" position/orientation etc.

```
//FRONT

glPushMatrix(); //save current position/orientation, etc.
glTranslatef(0, 2.5, 5); //move by x,y and z units
glRotatef(180, 0, 1, 0);
glMaterialfv(GL_FRONT, GL_AMBIENT, matAmbient1); //set backdrop material properties
glMaterialfv(GL_FRONT, GL_DIFFUSE, matDiff1);
drawBox(10, 0.01, 5); //draw front wall
glPopMatrix(); //restore previous "current" position/orientation etc.
```

```
//BACK LEFT

glPushMatrix(); //save current position/orientation, etc.
glTranslatef(-3.25, 2.5, -5); //move by x,y and z units
glMaterialfv(GL_FRONT, GL_AMBIENT, matAmbient1); //set backdrop material properties
glMaterialfv(GL_FRONT, GL_DIFFUSE, matDiff1);
drawBox(3.5, 0.01, 5); //draw left part of back wall
glPopMatrix(); //restore previous "current" position/orientation etc.
```

```
//BACK RIGHT

glPushMatrix(); //save current position/orientation, etc.
glTranslatef(3.25, 2.5, -5); //move by x,y and z units
glMaterialfv(GL_FRONT, GL_AMBIENT, matAmbient1); //set backdrop material properties
glMaterialfv(GL_FRONT, GL_DIFFUSE, matDiff1);
drawBox(3.5, 0.01, 5); //draw right part of back wall
glPopMatrix(); //restore previous "current" position/orientation etc.
```

```
//DOOR

glPushMatrix(); //save current position/orientation, etc.
```

```
glTranslatef(3, 2.5, -4.73); //move by x,y and z units
glMaterialfv(GL_FRONT, GL_AMBIENT, matAmbient1); //set backdrop material properties
glMaterialfv(GL_FRONT, GL_DIFFUSE, matDiff1);
glRotatef(-10, 0, 1, 0);
drawDoor(3, 0.01, 5); //draws door
glPopMatrix(); //restore previous "current" position/orientation etc.
```

```
//PLINTHS
```

```
glBindTexture(GL_TEXTURE_2D, texnum[0]); //Changes Texture for following objects
```

```
//Center Plinth
```

```
glPushMatrix(); //save current position/orientation, etc.
glBindTexture(GL_TEXTURE_2D, texnum[0]);
glTranslatef(0, 0, 0); //move by x,y and z units
objarray[0]->render();
glPopMatrix(); //restore previous "current" position/orientation etc.
```

```
//Back Right Plinth
```

```
glPushMatrix(); //save current position/orientation, etc.
glTranslatef(4, 0, -2); //move by x,y and z units
objarray[0]->render();
glPopMatrix(); //restore previous "current" position/orientation etc.
```

```
//Back Left Plinth
```

```
glPushMatrix(); //save current position/orientation, etc.
glTranslatef(-4, 0, -2); //move by x,y and z units
objarray[0]->render();
glPopMatrix(); //restore previous "current" position/orientation etc.
```

```
//Front Right Plinth  
  
glPushMatrix(); //save current position/orientation, etc.  
glTranslatef(4, 0, 2); //move by x,y and z units  
objarray[0]->render();  
glPopMatrix(); //restore previous "current" position/orientation etc.
```

```
//Front Left Plinth  
  
glPushMatrix(); //save current position/orientation, etc.  
glTranslatef(-4, 0, 2); //move by x,y and z units  
objarray[0]->render();  
glPopMatrix(); //restore previous "current" position/orientation etc.
```

```
//ORNAMENTS  
  
glBindTexture(GL_TEXTURE_2D, texnum[3]);  
  
//Front Left Ornament  
  
glPushMatrix(); //save current position/orientation, etc.  
glTranslatef(-4, 2.5, 2); //move by x,y and z units  
objarray[1]->render();  
glPopMatrix(); //restore previous "current" position/orientation etc.
```

```
glBindTexture(GL_TEXTURE_2D, texnum[5]);  
  
//Back Left Ornament  
  
glPushMatrix(); //save current position/orientation, etc.  
glTranslatef(-4, 1.5, -2); //move by x,y and z units  
objarray[2]->render();  
glPopMatrix(); //restore previous "current" position/orientation etc.
```

```
glBindTexture(GL_TEXTURE_2D, texnum[6]);
```

```
//Back Right Ornament
```

```
glPushMatrix(); //save current position/orientation, etc.
```

```
glTranslatef(4, 1.6, -2); //move by x,y and z units
```

```
objarray[3]->render();
```

```
glPopMatrix(); //restore previous "current" position/orientation etc.
```

```
glBindTexture(GL_TEXTURE_2D, texnum[7]);
```

```
//Front Right Ornament
```

```
glPushMatrix(); //save current position/orientation, etc.
```

```
glTranslatef(4, 2, 2); //move by x,y and z units
```

```
glRotatef(-90, 0, 1, 0);
```

```
objarray[4]->render();
```

```
glPopMatrix(); //restore previous "current" position/orientation etc.
```

```
glBindTexture(GL_TEXTURE_2D, texnum[2]); //Changes Texture for following objects
```

```
//Teapot
```

```
glColor3f(0.8, 1.0, 0.8); //set colour not used. Automatically disabled when lighting  
Enabled
```

```
glPushMatrix(); //save current position/orientation, etc.
```

```
glTranslatef(0, 2.25, 0); //move by x,y and z units
```

```
glMaterialfv(GL_FRONT, GL_AMBIENT, matAmbient1); //set teapot light material properties
```

```
glMaterialfv(GL_FRONT, GL_DIFFUSE, matDiff1);
```

```
glutSolidTeapot(0.4); //param is size of teapot
```

```
glPopMatrix(); //restore previous "current" position/orientation etc.
```

```
//SECOND ROOM
```

```
//FLOOR
```

```
glColor3f(0.8, 0.8, 1.0); //set colour not used. Automatically disabled when lighting
Enabled

glTranslatef(0, 0, -10); //move by x,y and z units

glMaterialfv(GL_FRONT, GL_AMBIENT, matAmbient); //set backdrop material properties

glMaterialfv(GL_FRONT, GL_DIFFUSE, matDiff);

drawBox(10.0, 10.0, 0.01); //draw floor

glColor3f(1.0, 0.8, 0.8); //set colour not used. Automatically disabled when lighting
Enabled
```

//LEFT

```
glPushMatrix(); //save current position/orientation, etc.

glTranslatef(-5, 2.5, 0); //move by x,y and z units

glRotatef(90, 0, 1, 0);

glMaterialfv(GL_FRONT, GL_AMBIENT, matAmbient1); //set backdrop material properties

glMaterialfv(GL_FRONT, GL_DIFFUSE, matDiff1);

drawBox(10, 0.01, 5); //draw left wall

glPopMatrix(); //restore previous "current" position/orientation etc.
```

//RIGHT

```
glPushMatrix(); //save current position/orientation, etc.

glTranslatef(5, 2.5, 0); //move by x,y and z units

glRotatef(-90, 0, 1, 0);

glMaterialfv(GL_FRONT, GL_AMBIENT, matAmbient1); //set backdrop material properties

glMaterialfv(GL_FRONT, GL_DIFFUSE, matDiff1);

drawBox(10, 0.01, 5); //draw right wall

glPopMatrix(); //restore previous "current" position/orientation etc.
```

//BACK

```
glPushMatrix(); //save current position/orientation, etc.

glTranslatef(0, 2.5, -5); //move by x,y and z units
```

```
glRotatef(180, 0, 1, 0);  
glMaterialfv(GL_FRONT, GL_AMBIENT, matAmbient1); //set backdrop material properties  
glMaterialfv(GL_FRONT, GL_DIFFUSE, matDiff1);  
drawBox(10, 0.01, 5); //draw back wall  
glPopMatrix(); //restore previous "current" position/orientation etc.
```

```
//CENTER STAND
```

```
glPushMatrix(); //save current position/orientation, etc.  
glBindTexture(GL_TEXTURE_2D, texnum[17]);  
glTranslatef(0, 0, 0); //move by x,y and z units  
objarray[6]->render();  
glPopMatrix(); //restore previous "current" position/orientation etc.
```

```
//PAINTINGS & THEIR FRAMES
```

```
//Center Picture
```

```
glPushMatrix(); //save current position/orientation, etc.  
glTranslatef(0, 3, -4.99); //move by x,y and z units  
glRotatef(180, 0, 1, 0);  
glMaterialfv(GL_FRONT, GL_AMBIENT, matAmbient1); //set backdrop material properties  
glMaterialfv(GL_FRONT, GL_DIFFUSE, matDiff1);  
drawPicture7(2, 0.01, 2.5); //draw picture  
glPopMatrix(); //restore previous "current" position/orientation etc.
```

```
//Center Picture frame
```

```
glPushMatrix(); //save current position/orientation, etc.  
glTranslatef(0, 3, -4.991); //move by x,y and z units  
glRotatef(180, 0, 1, 0);  
glMaterialfv(GL_FRONT, GL_AMBIENT, matAmbient1); //set backdrop material properties  
glMaterialfv(GL_FRONT, GL_DIFFUSE, matDiff1);
```

```
drawFrame(2.2, 0.01, 2.7); //draw picture frame  
glPopMatrix(); //restore previous "current" position/orientation etc.
```

```
    //Picture 1  
    glPushMatrix(); //save current position/orientation, etc.  
    glTranslatef(3, 3, -4.99); //move by x,y and z units  
    glRotatef(180, 0, 1, 0);  
    glMaterialfv(GL_FRONT, GL_AMBIENT, matAmbient1); //set backdrop material properties  
    glMaterialfv(GL_FRONT, GL_DIFFUSE, matDiff1);  
    drawPicture1(2, 0.01, 2.5); //draw picture  
    glPopMatrix(); //restore previous "current" position/orientation etc.
```

```
    //Picture 1 frame  
    glPushMatrix(); //save current position/orientation, etc.  
    glTranslatef(3, 3, -4.991); //move by x,y and z units  
    glRotatef(180, 0, 1, 0);  
    glMaterialfv(GL_FRONT, GL_AMBIENT, matAmbient1); //set backdrop material properties  
    glMaterialfv(GL_FRONT, GL_DIFFUSE, matDiff1);  
    drawFrame(2.2, 0.01, 2.7); //draw picture frame  
    glPopMatrix(); //restore previous "current" position/orientation etc.
```

```
    //Picture 2  
    glPushMatrix(); //save current position/orientation, etc.  
    glTranslatef(-3, 3, -4.99); //move by x,y and z units  
    glRotatef(180, 0, 1, 0);  
    glMaterialfv(GL_FRONT, GL_AMBIENT, matAmbient1); //set backdrop material properties  
    glMaterialfv(GL_FRONT, GL_DIFFUSE, matDiff1);
```

```
drawPicture2(2, 0.01, 2.5); //draw picture  
glPopMatrix(); //restore previous "current" position/orientation etc.
```

```
    //Picture 2 frame  
glPushMatrix(); //save current position/orientation, etc.  
glTranslatef(-3, 3, -4.991); //move by x,y and z units  
glRotatef(180, 0, 1, 0);  
glMaterialfv(GL_FRONT, GL_AMBIENT, matAmbient1); //set backdrop material properties  
glMaterialfv(GL_FRONT, GL_DIFFUSE, matDiff1);  
drawFrame(2.2, 0.01, 2.7); //draw picture frame  
glPopMatrix(); //restore previous "current" position/orientation etc.
```

```
    //Picture 3  
glPushMatrix(); //save current position/orientation, etc.  
glTranslatef(-4.99, 3, 2); //move by x,y and z units  
glRotatef(270, 0, 1, 0);  
glMaterialfv(GL_FRONT, GL_AMBIENT, matAmbient1); //set backdrop material properties  
glMaterialfv(GL_FRONT, GL_DIFFUSE, matDiff1);  
drawPicture3(3, 0.01, 2.5); //draw picture  
glPopMatrix(); //restore previous "current" position/orientation etc.
```

```
    //Picture 3 frame  
glPushMatrix(); //save current position/orientation, etc.  
glTranslatef(-4.991, 3, 2); //move by x,y and z units  
glRotatef(270, 0, 1, 0);  
glMaterialfv(GL_FRONT, GL_AMBIENT, matAmbient1); //set backdrop material properties  
glMaterialfv(GL_FRONT, GL_DIFFUSE, matDiff1);
```

```
drawFrame(3.2, 0.01, 2.7); //draw picture frame  
glPopMatrix(); //restore previous "current" position/orientation etc.
```

```
//Picture 4
```

```
glPushMatrix(); //save current position/orientation, etc.  
glTranslatef(-4.99, 3, -2); //move by x,y and z units  
glRotatef(270, 0, 1, 0);  
glMaterialfv(GL_FRONT, GL_AMBIENT, matAmbient1); //set backdrop material properties  
glMaterialfv(GL_FRONT, GL_DIFFUSE, matDiff1);  
drawPicture4(3, 0.01, 2.5); //draw picture  
glPopMatrix(); //restore previous "current" position/orientation etc.
```

```
//Picture 4 frame
```

```
glPushMatrix(); //save current position/orientation, etc.  
glTranslatef(-4.991, 3, -2); //move by x,y and z units  
glRotatef(270, 0, 1, 0);  
glMaterialfv(GL_FRONT, GL_AMBIENT, matAmbient1); //set backdrop material properties  
glMaterialfv(GL_FRONT, GL_DIFFUSE, matDiff1);  
drawFrame(3.2, 0.01, 2.7); //draw picture frame  
glPopMatrix(); //restore previous "current" position/orientation etc.
```

```
//Picture 5
```

```
glPushMatrix(); //save current position/orientation, etc.  
glTranslatef(4.99, 3, -2); //move by x,y and z units  
glRotatef(90, 0, 1, 0);  
glMaterialfv(GL_FRONT, GL_AMBIENT, matAmbient1); //set backdrop material properties
```

```
glMaterialfv(GL_FRONT, GL_DIFFUSE, matDiff1);  
drawPicture5(3, 0.01, 2.5); //draw picture  
glPopMatrix(); //restore previous "current" position/orientation etc.
```

//Picture 5 frame

```
glPushMatrix(); //save current position/orientation, etc.  
glTranslatef(4.991, 3, -2); //move by x,y and z units  
glRotatef(90, 0, 1, 0);  
glMaterialfv(GL_FRONT, GL_AMBIENT, matAmbient1); //set backdrop material properties  
glMaterialfv(GL_FRONT, GL_DIFFUSE, matDiff1);  
drawFrame(3.2, 0.01, 2.7); //draw picture frame  
glPopMatrix(); //restore previous "current" position/orientation etc.
```

//Picture 6

```
glPushMatrix(); //save current position/orientation, etc.  
glTranslatef(4.99, 3, 2); //move by x,y and z units  
glRotatef(90, 0, 1, 0);  
glMaterialfv(GL_FRONT, GL_AMBIENT, matAmbient1); //set backdrop material properties  
glMaterialfv(GL_FRONT, GL_DIFFUSE, matDiff1);  
drawPicture6(3, 0.01, 2.5); //draw picture  
glPopMatrix(); //restore previous "current" position/orientation etc.
```

//Picture 6 frame

```
glPushMatrix(); //save current position/orientation, etc.  
glTranslatef(4.991, 3, 2); //move by x,y and z units  
glRotatef(90, 0, 1, 0);  
glMaterialfv(GL_FRONT, GL_AMBIENT, matAmbient1); //set backdrop material properties
```

```
glMaterialfv(GL_FRONT, GL_DIFFUSE, matDiff1);  
drawFrame(3.2, 0.01, 2.7); //draw picture frame  
glPopMatrix(); //restore previous "current" position/orientation etc.
```

```
        //ROTATING DONUT  
  
glBindTexture(GL_TEXTURE_2D, texnum[19]); //5  
glPushMatrix(); //save current position/orientation, etc.  
glTranslatef(0, 2.75, 0); //move by x,y and z units  
glRotatef(90, 1, 0, 0);  
glMaterialfv(GL_FRONT, GL_AMBIENT, matAmbient2); //set box light material properties  
glMaterialfv(GL_FRONT, GL_DIFFUSE, matDiff2);  
glRotatef(globangle, 0, 0, 1);  
objarray[5]->render();  
glPopMatrix(); //restore previous "current" position/orientation etc.
```

```
glPopMatrix();  
glFlush(); //force drawing
```

```
glutSwapBuffers();    // bring backbuffer to foreground
```

```
        //any errors then display error codes
```

```
GLenum errCode;  
const GLubyte *errString;  
if ((errCode = glGetError()) != GL_NO_ERROR)  
{  
    errString = gluErrorString(errCode);  
    fprintf(stderr, "OpenGL error : %s\n", errString);  
}
```

```

    }

}

void reshape(int w, int h)

//OpenGL reshape called when the window is resized
{
    glViewport(0, 0, (GLsizei)w, (GLsizei)h); //respecify size of view port to match new w and h
    glMatrixMode(GL_PROJECTION); //go into projection mode
    glLoadIdentity(); //reset projection matrix
    gluPerspective(60.0, (GLfloat)w / (GLfloat)h, 1.0, 200.0); //redefine projection matrix to
    match changes in w and h
    glMatrixMode(GL_MODELVIEW); //switch back to modelview mode
    glLoadIdentity(); //reset modelview matrix
    gluLookAt(0.0, 2.0, 5.0, 0.0, 2.0, -1.0, 0.0, 1.0, 0.0);
}


void keyboard(unsigned char key, int x, int y)
{
    static float vx = 0, vy = 0, vz = 5, angle = 0;
    switch (key) {

    case 'a' | 'A':
        vx = vx - 0.1;
        glMatrixMode(GL_MODELVIEW);
        break;
    case 'd' | 'D':

```

```
    vx = vx + 0.1;

    glMatrixMode(GL_MODELVIEW);

    break;

case 'w' | 'W':

    vz = vz - 0.1;

    glMatrixMode(GL_MODELVIEW);

    break;

case 's' | 'S':

    glMatrixMode(GL_MODELVIEW);

    vz = vz + 0.1;

    break;

case 'Z' | 'z':

    angle = angle - 10;

    glMatrixMode(GL_MODELVIEW);

    break;

case 'X' | 'x':

    glMatrixMode(GL_MODELVIEW);

    angle = angle + 10;

//Camera up and down

//case 'q' | 'Q':

//    vy = vy - 0.1;

//    glMatrixMode(GL_MODELVIEW);

//    break;

//case 'e' | 'E':

//    glMatrixMode(GL_MODELVIEW);

//    vy = vy + 0.1;
```

```

        break;
case 27:
    exit(0);
    break;
default:
    break;

}

glLoadIdentity();
camera(vx, vz, vy, angle); //use glulookat() if you want
glutPostRedisplay();
}

void camera(float x, float z, float y, float angle)
{
    glRotatef(angle, 0, 1, 0);
    glTranslatef(-x, -3.5, -z);
}

void spinDisplay(void)
{
    globangle = globangle + 0.3f;
    if (globangle >= 360.0f)
        globangle = 0.0f;
    glutPostRedisplay();
}

int main(int argc, char** argv)

```

```

{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(100, 100);
    glutCreateWindow(argv[0]);
    init();
    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutKeyboardFunc(keyboard);
    glutIdleFunc(spinDisplay);
    glutMainLoop();
    return 0;
}

//*****

// LoadBitmapFile
// desc: Returns a pointer to the bitmap image of the bitmap specified
//      by filename. Also returns the bitmap header information.
//      No support for 8-bit bitmaps.
unsigned char *LoadBitmapFile(char *filename, BITMAPINFOHEADER *bitmapInfoHeader)
{
    FILE *filePtr;          // the file pointer
    BITMAPFILEHEADER bitmapFileHeader; // bitmap file header
    unsigned char *bitmapImage; // bitmap image data
    int imageIdx = 0; // image index counter
    unsigned char tempRGB; // swap variable

```

```
        // open filename in "read binary" mode
filePtr = fopen(filename, "rb");
if (filePtr == NULL)
    return NULL;

// read the bitmap file header
fread(&bitmapFileHeader, sizeof(BITMAPFILEHEADER), 1, filePtr);

// verify that this is a bitmap by checking for the universal bitmap id
if (bitmapFileHeader.bfType != BITMAP_ID)
{
    fclose(filePtr);
    return NULL;
}

// read the bitmap information header
fread(bitmapInfoHeader, sizeof(BITMAPINFOHEADER), 1, filePtr);

// move file pointer to beginning of bitmap data
fseek(filePtr, bitmapFileHeader.bfOffBits, SEEK_SET);

// allocate enough memory for the bitmap image data
bitmapImage = (unsigned char*)malloc(bitmapInfoHeader->biSizeImage);

// verify memory allocation
if (!bitmapImage)
{
    free(bitmapImage);
    fclose(filePtr);
    return NULL;
}
```

```

    }

    // read in the bitmap image data
    fread(bitmapImage, 1, bitmapInfoHeader->biSizeImage, filePtr);

    // make sure bitmap image data was read
    if (bitmapImage == NULL)
    {
        fclose(filePtr);
        return NULL;
    }

    // swap the R and B values to get RGB since the bitmap color format is in BGR
    for (imageldx = 0; imageldx < bitmapInfoHeader->biSizeImage; imageldx += 3)
    {
        tempRGB = bitmapImage[imageldx];
        bitmapImage[imageldx] = bitmapImage[imageldx + 2];
        bitmapImage[imageldx + 2] = tempRGB;
    }

    // close the file and return the bitmap image data
    fclose(filePtr);
    return bitmapImage;
}

// WriteBitmapFile()
// desc: takes image data and saves it into a 24-bit RGB .BMP file
//      with width X height dimensions
int WriteBitmapFile(char *filename, int width, int height, unsigned char *imageData)

```

```

{
    FILE      *filePtr;    // file pointer

    BITMAPFILEHEADER bitmapFileHeader; // bitmap file header
    BITMAPINFOHEADER bitmapInfoHeader; // bitmap info header

    int      imgIdx; // used for swapping RGB->BGR
    unsigned char tempRGB; // used for swapping


    // open file for writing binary mode
    filePtr = fopen(filename, "wb");
    if (!filePtr)
        return 0;


    // define the bitmap file header
    bitmapFileHeader.bfSize = sizeof(BITMAPFILEHEADER);
    bitmapFileHeader.bfType = 0x4D42;
    bitmapFileHeader.bfReserved1 = 0;
    bitmapFileHeader.bfReserved2 = 0;
    bitmapFileHeader.bfOffBits = sizeof(BITMAPFILEHEADER) + sizeof(BITMAPINFOHEADER);


    // define the bitmap information header
    bitmapInfoHeader.biSize = sizeof(BITMAPINFOHEADER);
    bitmapInfoHeader.biPlanes = 1;
    bitmapInfoHeader.biBitCount = 24; // 24-bit
    bitmapInfoHeader.biCompression = BI_RGB; // no compression
    bitmapInfoHeader.biSizeImage = width * abs(height) * 3; // width * height * (RGB bytes)
    bitmapInfoHeader.biXPelsPerMeter = 0;
    bitmapInfoHeader.biYPelsPerMeter = 0;
    bitmapInfoHeader.biClrUsed = 0;
    bitmapInfoHeader.biClrImportant = 0;

```

```
    bitmapInfoHeader.biWidth = width;           // bitmap width
    bitmapInfoHeader.biHeight = height;         // bitmap height

    // switch the image data from RGB to BGR
    for (imageldx = 0; imageldx < bitmapInfoHeader.biSizeImage; imageldx += 3)
    {
        tempRGB = imageData[imageldx];
        imageData[imageldx] = imageData[imageldx + 2];
        imageData[imageldx + 2] = tempRGB;
    }

    // write the bitmap file header
    fwrite(&bitmapFileHeader, 1, sizeof(BITMAPFILEHEADER), filePtr);

    // write the bitmap info header
    fwrite(&bitmapInfoHeader, 1, sizeof(BITMAPINFOHEADER), filePtr);

    // write the image data
    fwrite(imageData, 1, bitmapInfoHeader.biSizeImage, filePtr);

    // close our file
    fclose(filePtr);

    return 1;
}
```

Graphics

GRADEMARK REPORT

FINAL GRADE

GENERAL COMMENTS

Instructor

64/100

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

PAGE 15

PAGE 16

PAGE 17

PAGE 18

PAGE 19

PAGE 20

PAGE 21

PAGE 22

PAGE 23

PAGE 24

PAGE 25

PAGE 26

PAGE 27

PAGE 28

PAGE 29

PAGE 30

PAGE 31

PAGE 32

PAGE 33

PAGE 34

PAGE 35

PAGE 36

PAGE 37

PAGE 38

PAGE 39

PAGE 40

PAGE 41

PAGE 42

PAGE 43

GRADING FORM: SMU GENERIC_GJ

RUAIDHRI DAVID COOK

63.50

RESEARCH-INFORMED LITERATURE



No refs included. you need to include them here -
you've obviously explored each topic in detail.

4

KNOWLEDGE AND UNDERSTANDING



A sound understanding of the majority of concepts -
camera transforms need to be investigated more as
do the application of textures to obj file formats.
Otherwise a v good submission.

16

ANALYSIS



You've interpreted the design brief appropriately. Lots of objects loaded in obj format but you haven't quite solved the texturing to the load obj objects. A more sophisticated animation would be even better. You haven't solved the camera problem exercise - you need to investigate camera transforms - see note on moodle for this.

14.5

PRACTICAL APPLICATION AND DEP.



Some good work here. You've demonstrated a good understanding of the graphics pipeline and the application of OGL API, obj loaders., etc. Textures loaded but some obj objects need further work to attach textures.

26

SKILLS FOR PROFESSIONAL PRAC.



No refs. Report structured and good discussion.

3