

Interactive Graphics Programming

by Jonathan Ieuan Evans

FILE	4766_JONATHAN_IEUAN_EVANS_INTERACTIVE_GRAPHICS_PROGRAM MING_110623_1074184847.DOCX (792K)		
TIME SUBMITTED	05-APR-2019 05:12PM (UTC+0100)	WORD COUNT	1644
SUBMISSION ID	104227039	CHARACTER COUNT	9446

Interactive Graphics Programming

Jonathan Evans

P151713

5th April 2019

Assignment 1

Contents

1. Introduction.....	5
2. The Scene	6
3. User Guide.....	8
3.1 Compiling the program.....	8
3.2 Using the program	8
4 Testing	9
5 Structure and design.....	10
5.1 Functions of program	10
6 Appendices.....	13

1. Introduction

This report will analyse a 3D interactive graphics scene that has been created for the purpose of learning OpenGL. It will consist of four sections. The first will provide a description of the scene that has been created, the second will include a user guide explaining how to compile and use the program, the third will provide testing that has been carried out, and the final section will explain the functionalities and structure of the program.

2. The Scene

Below is a scene created using GLUT (OpenGL Utility Toolkit, a utility that makes use of OpenGL functionality).

The scene that has been created is a gallery that showcases teapots and paintings of teapots. A teapot gallery has been created as GLUT provides a built-in model for a teapot. This is convenient as a teapot model does not have to be manually created. In addition to this, the incorporation of teapots play homage to the Utah teapot, a famous teapot model that was created by Martin Newell as a test for creating a 3D model in computer graphics.



The scene consists of two rooms which have four walls, a ceiling, floor, paintings, and display stands with rotating teapots. The scene also includes a lighting source that illuminates the objects in the scene. Each object (except the teapot model, and stand model) in the scene has been textured using bitmap images.

During the creation of this scene, code samples from the university's Moodle page (an online Virtual Learning Environment 'VLE' platform) have been studied and incorporated where necessary (namely code samples that are used for loading bitmaps, animation, controlling the camera using keys, and code for texturing an object).

In addition, the online documentation of GLUT (from the official OpenGL website) has been studied. This has proved useful when investigating how the functions of GLUT work (as a result, comments have been created preceding some of the functions that have been used in the code).

As well as the code samples referenced on the previous page, some materials and a 3D object has been referenced in the table shown below:

Item	Source
Wood texture (used for Floor) 	https://www.thesolidwoodflooringcompany.com/wood-flooring/extra-wide-wood-flooring
Wall texture 	https://galleryneed.com/pics/wall-paint-drywall-textures-ideas-28.html
Utah Teapot 	Provided through the <i>glutSolidTeapot()</i> function, as part of the GLUT library.

Where necessary, the textures above have been resized to a resolution of 1024x1024. This allowed the textures to be easily mapped onto square objects in the scene with relatively good quality.

3. User Guide

This user guide will describe how to set up and use the program shown in the previous section.

3.1 Compiling the program

The code for this program is available at *Appendix A*. This can be copied into Visual Studio and compiled, providing that the GLUT library has been included in the project (This can be achieved by placing the *glut.h* and related dll files in the same folder as the executable program).

If the platform toolkit is different to the one this program was developed for (Visual Studio 2017), the project may need to be retargeted to the desired version.

3.2 Using the program

As soon as the user launches the program, they (the camera) is placed within a textured box – this is the gallery. The user is allowed to freely move around this space (even outside of it as collision has not been implemented).

Below is a table that shows the control scheme for controlling the camera within the scene.

Key	Action
W	Moves the camera forward
A	Move the camera backward
S	Rotates the view of the camera to the left
D	Rotates the view of the camera to the right
Space	Increases the height of the camera
F	Decreases the height of the camera

There are no controls to strafe left or right, or look up or down, however the given controls should be adequate for exploring the scene and viewing all the objects.

It is worth acknowledging that the rotation of the teapots may seem slightly erratic on displays with refresh rates higher than 60hz (even on 60hz, it may seem too fast). This could be slowed down by implementing a delay in milliseconds per degree of rotation; however, this has not been implemented for this project.

4 Testing

The code was tested to check if every key bind works and every object is present within the scene. Below is a table that was used to validate functionality of the program.

Test	Pass/Fail
#1 'W' key will move the camera forward	Pass
#2 'S' key will move the camera backward	Pass
#3 'A' will rotate the camera view left	Pass
#4 'D' will rotate the camera view right	Pass
#5 'Space' will increase the height of the camera	Pass
#6 'F' will decrease the height of the camera	Pass
#7 Teapot objects are present	Pass
#8 Teapot display stand objects are present	Pass
#9 Teapot objects are rotating	Pass
#10 Canvas paintings are present	Pass
#11 Canvas paintings textures are present	Pass

Not many big issues occurred during development, apart from one noteworthy issue that prevented the object loader code snippet from importing .obj (wavefront) files after they were exported from the Blender graphics program.

By using the debugging tools in Visual Studio, the point in which the program crashed could be viewed. Doing this helped uncover the issue – the object face vertices were in an incorrect format as shown below

```
290 f 62//72 58//72 60//72
291 f 58//31 54//31 56//31
292 f 54//31 50//31 52//31
293 f 50//31 46//31 48//31
294 f 48//31 44//31 46//31
295 f 46//31 42//31 44//31
296 f 44//31 40//31 42//31
297 f 42//31 38//31 40//31
```

Figure 1 Incorrect face vertices using default Blender export options

This was resolved by using the 'Unwrap' feature in Blender after triangulating a given object. As a result, vertices that were compatible with the object loader were then exported from Blender, as shown below.

```
529 f 6//7/3 7//11/3 8//183/3
530 f 8//10/4 9//14/4 10//184/4
531 f 10//13/5 11//17/5 12//185/5
532 f 12//16/6 13//20/6 14//186/6
533 f 14//19/7 15//23/7 16//187/7
534 f 16//22/8 17//26/8 18//188/8
535 f 18//25/9 19//29/9 20//189/9
536 f 20//28/10 21//32/10 22//190/10
537 f 22//31/11 23//35/11 24//191/11
538 f 24//34/12 25//38/12 26//192/12
539 f 26//37/13 27//41/13 28//193/13
540 f 28//40/14 29//44/14 30//194/14
541 f 30//43/15 31//47/15 32//195/15
```

An alternative solution would be to modify the object loader, however, changing the settings in blender proved more time efficient.

5 Structure and design

This section will analyse the functionality of the created scene. The program uses OpenGL to draw graphics to the screen. OpenGL is a specification that has been implemented by graphics card manufacturer's that allow for developers to make use of its functionalities. GLUT is a program that allows developers to program graphics using OpenGL.

The program does not use many classes, as a result a class diagram has not been provided, however, the proceeding section will analyse the functions used within the program.

5.1 Functions of program

Init()

The *Init()* function is responsible for first-time initialisation of the program, this function is called only once in the *Main()* function of the program.

During initialisation, the function sets up properties of the program such as a texture array, background colour, viewport transformation, projection transformation, perspective of camera, lighting, material reflection, and shading properties.

Lighting Variables

Global lighting variables are created to represent ambient lighting, diffuse, and specular lighting.

The ambient lighting represents the lighting of the world in a scene. This affects the colour of every object. This was set to white not to confuse the colours used on the object textures.

Diffuse is the amount of light that is absorbed and reflected by an object. With this an object can effectively light up other objects.

Specular lighting also absorbs some light and reflects light; however, this is more concentrated than diffuse lighting. This can create an effect that illustrates the distance of light to an object.

Incorporating Diffuse and Specular lighting help improve the realism of the objects in the scene. This makes the objects seem 'more real' due to the way the light reflects off the objects.

BitmapLoader()

The *BitmapLoader()* function is responsible for loading bitmaps into a texture array. This array is later used for texturing objects in a scene. The file format for this must be a BMP (Bitmap image).

Resize()

The *Resize()* function adjusts the camera perspective, and relevant global variables (viewport heights and widths) as these should be changed when the window scale changes to reflect the scale of the new window size.

DrawX()

The program has calls to draw different kinds of objects in a scene. These have sometimes been combined to draw bigger objects (for example, using four *drawWall()* functions to draw four walls for a gallery). The code inside such sections specify the relevant transformations, texture mappings, and vertex positions.

Camera()

The *Camera()* function adjusts the values of the *gluLookAt()* function. This is used in conjunction with *KeyboardRead()*, to allow the user to move around the scene.

KeyboardRead()

KeyboardRead() has been used to allow the user to move around the scene as described in the above. It is passed in as an argument to the *glutKeyboardFunc()* function. This ultimately allows the developer to specify key binds for keyboard interaction within the program.

Display()

The Display function is responsible for drawing all objects within the scene. This is where all the *DrawX()* calls are placed, and their relevant textures and colours.

Main()

The *Main()* function of this program is responsible for calling initialisation functionalities (such as *Init()*) and defining relevant properties (such as the display mode, window size, display function).

The last call in the *Main()* function is to begin an infinite loop. This is used for displaying graphics and is beneficial for displaying a glut window. This must be a blocking call as otherwise; the window would close before the user has the opportunity to view their OpenGL scene.

Matrix stacks

The program has three stacks to save operations before carrying out a matrix transformation. Model view (for controlling the model and view components) and the projection matrix. These were used throughout the program for applying transformations on objects within the scene.

6 Appendices

- **Appendix A**

Appendix A is the main.cpp file provided with this assignment. It was too big to include as a text object here unfortunately.

Interactive Graphics Programming

GRADEMARK REPORT

FINAL GRADE

GENERAL COMMENTS

Instructor

67/100

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

GRADING FORM: SMU GENERIC_GJ

JONATHAN IEUAN EVANS

66.50

RESEARCH-INFORMED LITERATURE



You've clearly researched the area well and covered discovered the significance of the glut teapot. But.. no refs present. At this level you need refs!!!!

4.5

KNOWLEDGE AND UNDERSTANDING



A good understanding demonstrated. You could have tried to apply textures to the obj file loader objects. A clear understanding of the OGL API demonstrated.

17.5

ANALYSIS



You solved the camera transform problem - well done, applied lighting, loaded bit maps and applied textures. Good. Testing included - the test documentation could be a little more detailed. The design of the building could be more sophisticated. No collision detection on walls, though.

14.5

PRACTICAL APPLICATION AND DEP.



Some good work here. Obj loader, bit map reader , textures on quads applied too. Good. No textures on obj objects. Scene could be a little more sophisticated. Animations maybe a little simplistic. Camera movement applied well - good.

28

SKILLS FOR PROFESSIONAL PRAC.



Good C++/OGL/ S/W Eng. skills demonstrated and report well structured.No REFS.

2