

1801164 Daniel O'Dwyer

by Daniel O'dwyer

FILE	12011_DANIEL_ODWYER_1801164_DANIEL_ODWYER_254066_74136757.DOCX (342.33K)		
TIME SUBMITTED	07-MAY-2020 08:42PM (UTC+0100)	WORD COUNT	1990
SUBMISSION ID	126677890	CHARACTER COUNT	13331

Games Architecture and Mechanics

ACCA5017

-Assignment Report-

Daniel O'Dwyer

1801164

Introduction

In this report the author will detail the source code that he developed; this report will also include tests of said code. The source code has been provided by the authors lecturer; the author has modified the source code as required. The code is an object loader that loads and displays .obj files.

Report

The author has integrated and implemented a functioning .obj file loader. Originally the source code used GL commands to draw a set of houses. Now the program loads whatever .obj file the user asks it to if it's in the same folder as the code. The author also added an update manager for 10 game objects. The author tried to implement a game loop that follows a fixed update rate but failed to integrate it properly. The object loader contains a command called `fscanf`, which isn't secure and can lead to problems when reading a file. This ideally should be `fscanf_s` but the author has made the decision to leave it as it is. To rectify this, the author has added the following line to the pre-processor argument `_CRT_SECURE_NO_WARNINGS;%(PreprocessorDefinitions)`. This line is added so that the compiler doesn't try to warn the user about the use of insecure `fscanf`. The object loader was then integrated with the mesh class, which allowed it to be called from the source C++ file. Instead the .obj file is called by typing it into the brackets of the new integrated file loader mesh class.

The author also implemented an update manager for a certain amount of game objects. For this task, the author chose to have 10 objects. The update manager has its own class and is integrated into the display C++ file and the source C++ file. However, most of the code resides in the display C++ file. The update manager, called game object, has its own function in the display C++ file. This function is called inside the display update loop. The author thought it would be better to have the update manager loop be outside the display update function. Every time an event occurs within the OGL window, like cursor movement, the update manager updates all the game objects. The game manager uses a vector to store the objects. Below are some screenshots of the implemented and updated code.

Mesh class with integrated .obj loader functions

```
#ifndef MESH_H
#define MESH_H

#include <glm/glm.hpp>
#include <gl/glew.h>
#include <SOIL/SOIL.h>
#include "objloader.h"
#define NUM_BUFFERS 1

typedef struct {
    glm::vec3 pos;
}Vertex;

class Mesh
{
public:
    Mesh(Vertex* vertices, unsigned int numVertices);
    Mesh(const char *f);
    object_type obj;
    void objPoly2Vertex(void);
    int loadOBJ();

    void draw();
    virtual ~Mesh();
private:
    Vertex *pVertices;
    GLuint vertexArrayObject; //for use with Modern OGL
    GLuint vertexArrayBuffers[NUM_BUFFERS]; //for use with Modern OGL
    GLuint vertexCount;
};

#endif
```

Mesh class loading the chosen .obj file

```
Mesh* pmesh = new Mesh("house.obj");
```

Render and Display functions

```
myDisplay.update(Objects);
myDisplay.render(pmesh);
```

Vector used with the Game Object class

```
vector <GameObj> Objects(10);
```

Game Object class

```
class GameObj
{
public:
    GameObj();
    ~GameObj();
    void UpdateGameObj();
private:
};
```

Game Object loop

```
void Gameobject(vector<GameObj>Objects)
{
    for (int i = 0; i < Objects.size(); i++)
    {
        Objects[i].UpdateGameObj();
        cout << "Object" << " " << i << " " << "has now been updated" << endl;
    }
}
```

Game Object loop being called in the Display Update function

```
void Display::update(vector<GameObj>Objects)
{
    SDL_Event e;
    while (SDL_PollEvent(&e)) //if an event occurs then process it and repeat til e
    {
        Gameobject(Objects);

        cout << "event number:" << e.type << endl;
        if (e.type == SDL_QUIT)
            closed = true;

        if (e.window.event == SDL_WINDOWEVENT_RESIZED)
        {
            glViewport(0, 0, e.window.data1,e.window.data2); //make sure openGL view po
            cout << "resized to " << e.window.data1 << " , " << e.window.data2<<endl;
            glMatrixMode(GL_PROJECTION);
            glLoadIdentity();
            glFrustum(-1.0, 1.0, -1.0, 1.0, 1.5, 20.0);
            //          l,r,bot,top,near,far

            glMatrixMode(GL_MODELVIEW);
        }
    }
}
```

Updated Display Class

```
#ifndef DISPLAY_H
#define DISPLAY_H
#include <SDL2/SDL.h>
#include <GL/glew.h>
#include <string>
#include "mesh.h"
#include "GameObj.h"
#include <vector>

using namespace std;

//uses SDL2 to create a window to display our OGL stuff
class Display
{
public:
    Display(int width, int height, std::string title);
    virtual ~Display();
    void swapBuffers();
    bool isClosed();
    void update(vector<GameObj>Objects);
    void clear(void);
    void render(Mesh *pMesh);
private:
    bool closed;
    SDL_Window* window;    //pointer to window
    SDL_GLContext glContext; //context for drawing OGL to SDL window
};

#endif //DISPLAY_H
```

Mesh constructor with object loader function and a polygon to vertex function

```
Mesh::Mesh(const char* f)
{
    obj.objloader(f);
    objPoly2Vertex();
}
```

Polygon to vertex function

```
void Mesh::objPoly2Vertex(void)
{
    int stride = 3;
    vertexCount = stride * obj.getNumPolys();
    int polyCount = obj.getNumPolys();
    pVertices = new Vertex[vertexCount];

    for (int i = 0; i < polyCount; i++)
    {
        polygon_type poly = obj.getPolygon(i);

        pVertices[stride*i + 0].pos.x = obj.getVertex(poly.v[0] - 1).x;
        pVertices[stride*i + 0].pos.y = obj.getVertex(poly.v[0] - 1).y;
        pVertices[stride*i + 0].pos.z = obj.getVertex(poly.v[0] - 1).z;

        pVertices[stride*i + 1].pos.x = obj.getVertex(poly.v[1] - 1).x;
        pVertices[stride*i + 1].pos.y = obj.getVertex(poly.v[1] - 1).y;
        pVertices[stride*i + 1].pos.z = obj.getVertex(poly.v[1] - 1).z;

        pVertices[stride*i + 2].pos.x = obj.getVertex(poly.v[2] - 1).x;
        pVertices[stride*i + 2].pos.y = obj.getVertex(poly.v[2] - 1).y;
        pVertices[stride*i + 2].pos.z = obj.getVertex(poly.v[2] - 1).z;
    }
}
```

Testing

Test 1

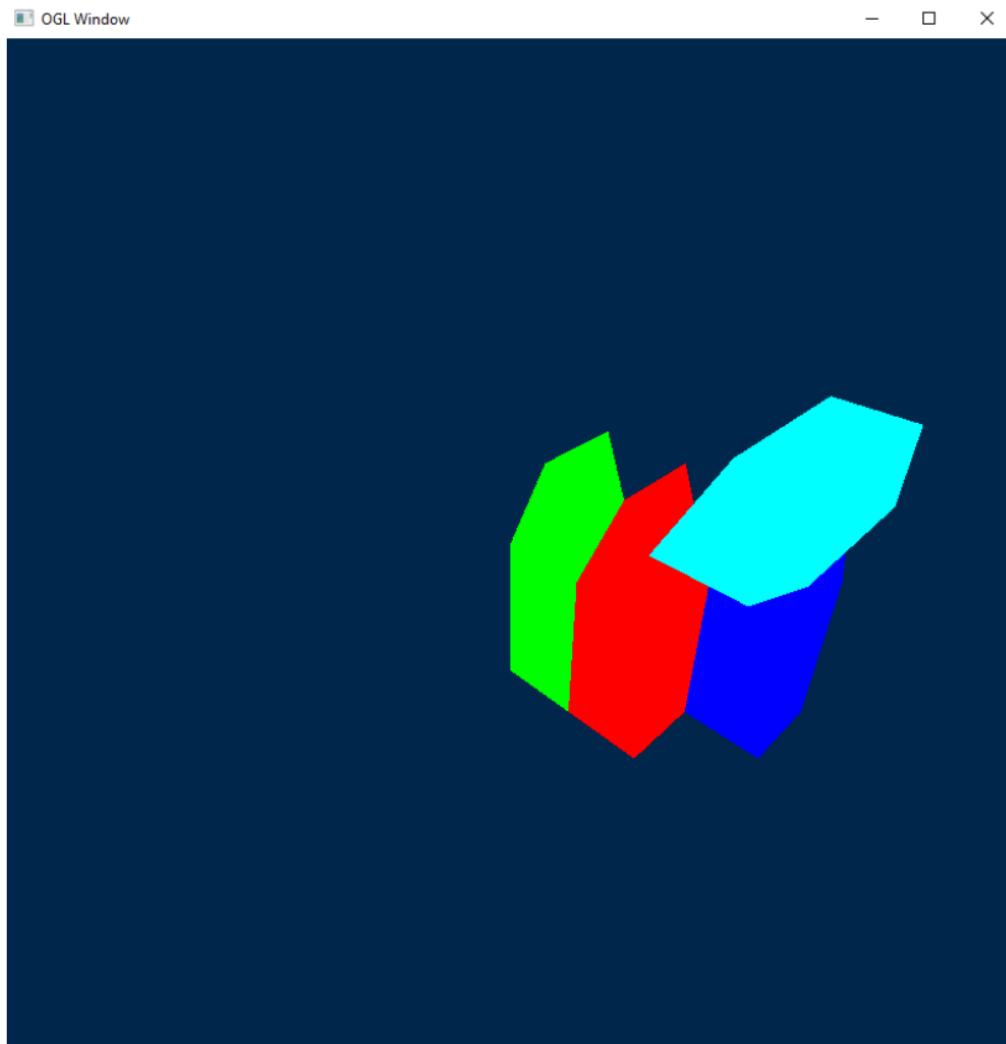
Test info: This first test will see if the object file loader can successfully load a .obj file of the authors choosing.

Expected results: The object file loader will successfully load the chosen .obj file

Actual results: The object file loader has successfully loaded the chosen .obj file

Pass/Fail: Pass

```
Mesh* pmesh = new Mesh("house.obj");
```



Test 2

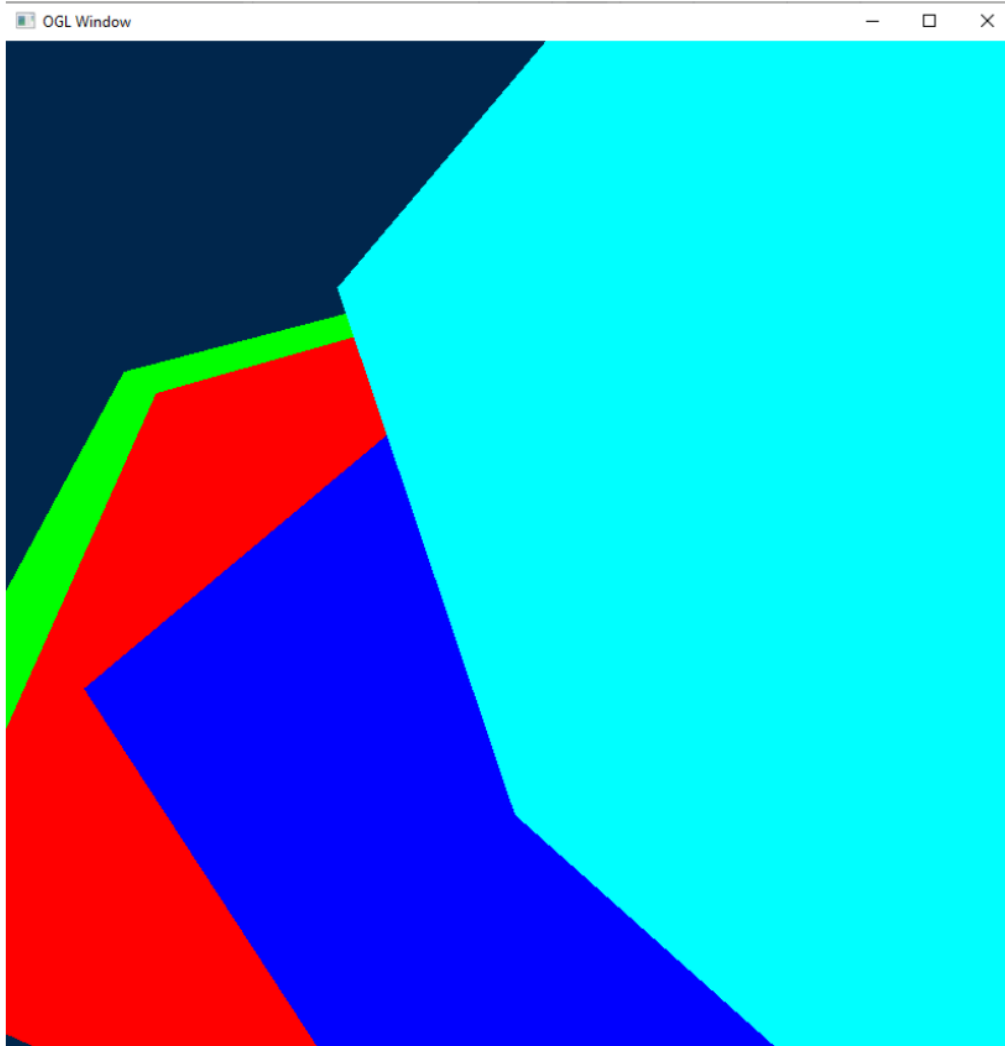
Test info: This second test will see if the object file loader can successfully load a different .obj file of the authors choosing.

Expected results: The object file loader will successfully load the chosen .obj file

Actual results: The object file loader has successfully loaded the chosen .obj file

Pass/Fail: Pass

```
Mesh* pmesh = new Mesh("blob.obj");
```



Test 3

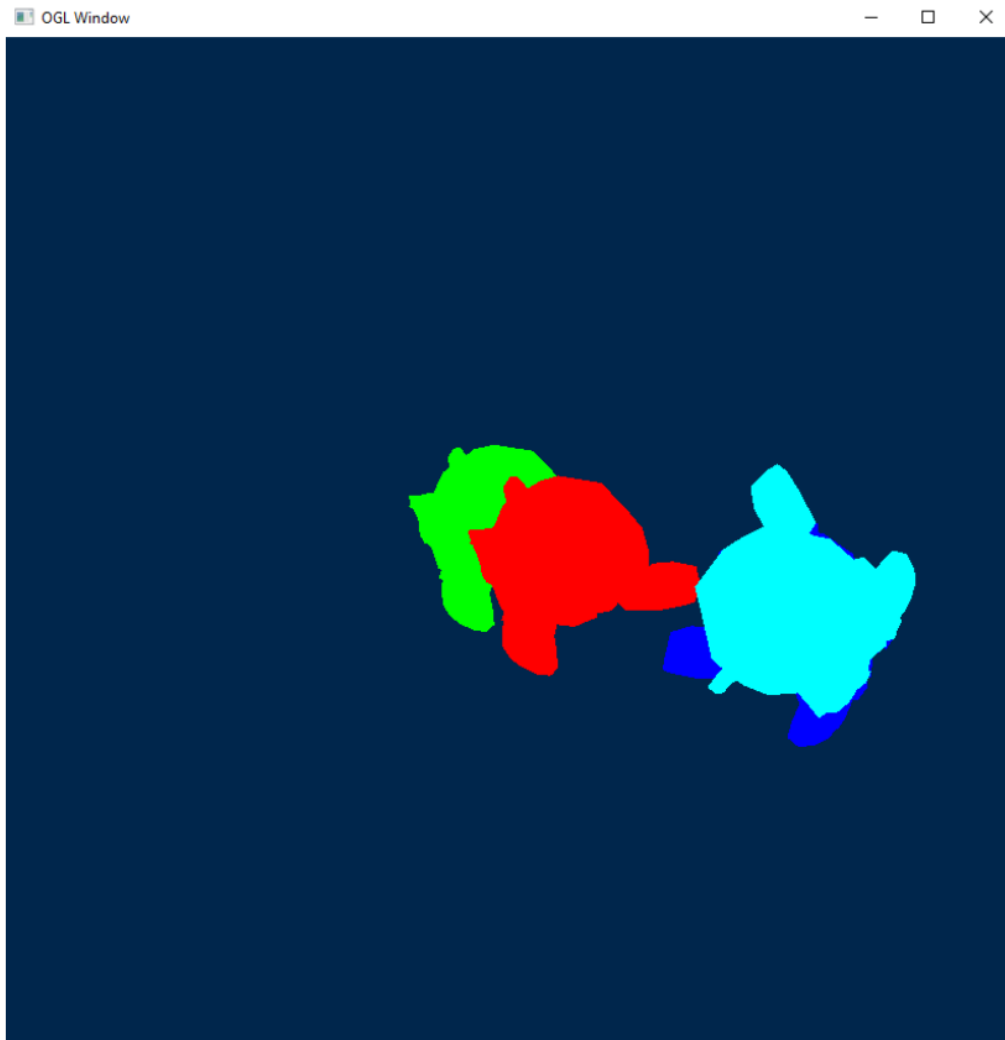
Test info: This third test will see if the object file loader can successfully load a different .obj file of the authors choosing.

Expected results: The object file loader will successfully load the chosen .obj file

Actual results: The object file loader has successfully loaded the chosen .obj file

Pass/Fail: Pass

```
Mesh* pmesh = new Mesh("blendermonkeyobj.obj");
```



Test 4

Test info: This last test will see if the update manager updates all the objects correctly whenever an event occurs. For this test the event will be the clicking of the left mouse button. This event is represented by the number 1025.

Expected results: The update manager will update all 10 game objects correctly when the event occurs.

Actual results: The update manager updated all 10 game objects correctly when the event occurred.

Pass/Fail: Pass

```
Object 0 has now been updated
Object 1 has now been updated
Object 2 has now been updated
Object 3 has now been updated
Object 4 has now been updated
Object 5 has now been updated
Object 6 has now been updated
Object 7 has now been updated
Object 8 has now been updated
Object 9 has now been updated
event number:1025
```

Appendix

```
#include "mesh.h"
#include "display.h"
//
#include "glut.h"
#include <Windows.h>
#include <iostream>
#include "GameObj.h"
#include <vector>

using namespace std;

vector <GameObj> Objects(10);

int main(int argc, char *args[])
{
    Display myDisplay(800, 800, "OGL Window"); //create and open a window

    //specify a list of vertices to draw - this will be loaded into our mesh object
    //we'll replace this array with a file mesh loader later - see .obj loader
    notes.

    //Vertex vertices[48];
    //Vertex myV;
    ///a triangle
    ///*vertices[0].pos = glm::vec3(-0.5, -0.5, 0.0);
    //vertices[1].pos = glm::vec3(0.0, 0.886-0.5, 0.0);
    //vertices[2].pos = glm::vec3(0.5, -0.5, 0.0);*/
    ///A House
    //vertices[0].pos = glm::vec3(0.0, 0.0, 0.0); //front
    //vertices[1].pos = glm::vec3(0.0, -2.0, 0.0);
    //vertices[2].pos = glm::vec3(1.0, 0.0, 0.0);
    //vertices[3].pos = glm::vec3(0.0, -2.0, 0.0);
    //vertices[4].pos = glm::vec3(1.0, -2.0, 0.0);
    //vertices[5].pos = glm::vec3(1.0, 0.0, 0.0);
```

```

//vertices[6].pos = glm::vec3(1.0, 0.0, 0.0);//right
//vertices[7].pos = glm::vec3(1.0, -2.0, 0.0);
//vertices[8].pos = glm::vec3(1.0, -2.0, -1.0);
//vertices[9].pos = glm::vec3(1.0, 0.0, 0.0);
//vertices[10].pos = glm::vec3(1.0, -2.0, -1.0);
//vertices[11].pos = glm::vec3(1.0, 0.0, -1.0);

//vertices[12].pos = glm::vec3(0.0, 0.0, 0.0);//left
//vertices[13].pos = glm::vec3(0.0, 0.0, -1.0);
//vertices[14].pos = glm::vec3(0.0, -2.0, -1.0);
//vertices[15].pos = glm::vec3(0.0, 0.0, 0.0);
//vertices[16].pos = glm::vec3(0.0, -2.0, -1.0);
//vertices[17].pos = glm::vec3(0.0, -2.0, 0.0);

//vertices[18].pos = glm::vec3(0.0, -2.0, 0.0); //bottom
//vertices[19].pos = glm::vec3(0.0, -2.0, -1.0);
//vertices[20].pos = glm::vec3(1.0, -2.0, -1.0);
//vertices[21].pos = glm::vec3(0.0, -2.0, 0.0);
//vertices[22].pos = glm::vec3(1.0, -2.0, -1.0);
//vertices[23].pos = glm::vec3(1.0, -2.0, 0.0);

//vertices[24].pos = glm::vec3(0.0, 0.0, -1.0);//back
//vertices[25].pos = glm::vec3(1.0, -2.0, -1.0);
//vertices[26].pos = glm::vec3(1.0, 0.0, -1.0);
//vertices[27].pos = glm::vec3(0.0, 0.0, -1.0);
//vertices[28].pos = glm::vec3(0.0, -2.0, -1.0);
//vertices[29].pos = glm::vec3(1.0, -2.0, -1.0);

//vertices[30].pos = glm::vec3(0.0, 0.0, 0.0);//front gable
//vertices[31].pos = glm::vec3(1.0, 0.0, 0.0);
//vertices[32].pos = glm::vec3(0.5, 1.25, 0.0);

//vertices[33].pos = glm::vec3(0.0, 0.0, -1.0);//back gable
//vertices[34].pos = glm::vec3(0.5, 1.25, -1.0);
//vertices[35].pos = glm::vec3(1.0, 0.0, -1.0);
//

//vertices[36].pos = glm::vec3(0.0, 0.0, 0.0);//left roof
//vertices[37].pos = glm::vec3(0.5, 1.25, 0.0);
//vertices[38].pos = glm::vec3(0.5, 1.25, -1.0);
//vertices[39].pos = glm::vec3(0.0, 0.0, 0.0);
//vertices[40].pos = glm::vec3(0.5, 1.25, -1.0);
//vertices[41].pos = glm::vec3(0.0, 0.0, -1.0);

//vertices[42].pos = glm::vec3(0.5, 1.25, 0.0);//right roof
//vertices[43].pos = glm::vec3(1.0, 0.0, 0.0);
//vertices[44].pos = glm::vec3(1.0, 0.0, -1.0);
//vertices[45].pos = glm::vec3(0.5, 1.25, 0.0);
//vertices[46].pos = glm::vec3(1.0, 0.0, -1.0);
//vertices[47].pos = glm::vec3(0.5, 1.25, -1.0);

Mesh* pmesh = new Mesh("blendermonkeyobj.obj");

while (!myDisplay.isClosed()) // Application loop - it's ok as a first
attempt
{
    //myDisplay.clear();// makes the point that the frame buffer is usually
    cleared before redrawing
    //glLoadIdentity();
    ///glTranslatef(-5, -2, -10);//

```

```

        //gluLookAt(5.0, 5.0, 5.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
        //// 5,5,5 - place camera ; 0,0,0 - aim lens towards 0,0,0;
        //// 0,1,0 - the up vector defines an unique orientation
        //pmesh->draw();
        ////glTranslatef(1.0, 0.0, 0.0);
        ////glColor3f(0.0, 1.0, 0.0);
        ////mesh.draw();
        ////glTranslatef(2.0, 0.0, -1.0);
        ////glRotatef(90.0, 0.0, 1.0, 0.0);
        ////glColor3f(0.0, 0.0, 1.0);
        ////mesh.draw();//mesh.draw(); // draw the model (s)
        ////glScalef(0.5f, 0.5f, 0.5f);
        ////glRotatef(90, 0, 0, 1);
        ////glTranslatef(0.5, 0, 0);
        ////mesh.draw();

        static unsigned long dwOldTime = GetTickCount();
        static unsigned long dwTimeElapsed;

/*
    float MS_PER_UPDATE = 16.67;
    dwOldTime = GetTickCount();
    float lag = 0.0;
    while (!gameOver)
    {
        currentTime = getCurrentTime();
        dwTimeElapsed = GetTickCount() - dwOldTime;
        dwOldTime = currentTime;
        lag += elapsed;
        processInput();
        while (lag >= MS_PERUPDATE)
        {
            update();
            lag -= MS_PER_UP_DATE;
        }
        render();
    }*/

    static int fps = 0; //frame count;
    dwTimeElapsed = GetTickCount() - dwOldTime;
    fps++;
    if (dwTimeElapsed > 5000) {

        std::cout << "frames per sec = " << (float)fps / 5.0 <<
std::endl;

        fps = 0;
        dwOldTime = GetTickCount();

    }
    myDisplay.update(Objects);
    myDisplay.render(pmesh);

    //This will soon need to be revised so we can update model positions,
    look after timers/ do physics/ deal with events.
}
return 0;
}

```

```

#ifndef MESH_H
#define MESH_H

#include <glm/glm.hpp>
#include <gl/glew.h>
#include <SOIL/SOIL.h>
#include "objloader.h"
#define NUM_BUFFERS 1
typedef struct {
    glm::vec3 pos;
}Vertex;

class Mesh
{
public:
    Mesh(Vertex* vertices, unsigned int numVertices);
    Mesh(const char *f);
    object_type obj;
    void objPoly2Vertex(void);
    int loadOBJ();

    void draw();
    virtual ~Mesh();
private:
    Vertex *pVertices;
    GLuint vertexArrayObject; //for use with Modern OGL
    GLuint vertexArrayBuffers[NUM_BUFFERS]; //for use with Modern OGL
    GLuint vertexCount;

};

#endif

#include "mesh.h"
#include <iostream>
#include <fstream>
#include <string>

using namespace std;

Mesh::Mesh(const char* f)
{
    obj.objloader(f);
    objPoly2Vertex();
}

Mesh::Mesh(Vertex* vertices, unsigned int numVertices)
{
    vertexCount = numVertices;
    pVertices = vertices;
    //glGenVertexArrays(1, &vertexArrayObject);
    //glBindVertexArray(vertexArrayObject);
    //glGenBuffers(NUM_BUFFERS, vertexArrayBuffers);
    //glBindBuffer(GL_ARRAY_BUFFER, vertexArrayBuffers[0]);
    //glBufferData(GL_ARRAY_BUFFER, numVertices * sizeof(vertices[0]), vertices,
    GL_STATIC_DRAW);
    //
    //glEnableVertexAttribArray(0);
    //glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 0, 0);
    //glBindVertexArray(0);

```

```

}

Mesh::~Mesh()
{
    //glDeleteVertexArrays(1, &vertexArrayObject);
}

void Mesh::draw()
{
    //glBindVertexArray(vertexArrayObject);
    //glDrawArrays(GL_TRIANGLES, 0, vertexCount);
    for (int i = 0; i < vertexCount; i+=3)
    {
        glBegin(GL_TRIANGLES);

        glVertex3f(pVertices[i].pos.x, pVertices[i].pos.y, pVertices[i].pos.z);

        glVertex3f(pVertices[i+1].pos.x, pVertices[i+1].pos.y,
pVertices[i+1].pos.z);

        glVertex3f(pVertices[i+2].pos.x, pVertices[i+2].pos.y,
pVertices[i+2].pos.z);

        glEnd();
    }
}

void Mesh::objPoly2Vertex(void)
{
    int stride = 3;
    vertexCount = stride * obj.getNumPolys();
    int polyCount = obj.getNumPolys();
    pVertices = new Vertex[vertexCount];

    for (int i = 0; i < polyCount; i++)
    {
        polygon_type poly = obj.getPolygon(i);

        pVertices[stride*i + 0].pos.x = obj.getVertex(poly.v[0] - 1).x;
        pVertices[stride*i + 0].pos.y = obj.getVertex(poly.v[0] - 1).y;
        pVertices[stride*i + 0].pos.z = obj.getVertex(poly.v[0] - 1).z;

        pVertices[stride*i + 1].pos.x = obj.getVertex(poly.v[1] - 1).x;
        pVertices[stride*i + 1].pos.y = obj.getVertex(poly.v[1] - 1).y;
        pVertices[stride*i + 1].pos.z = obj.getVertex(poly.v[1] - 1).z;

        pVertices[stride*i + 2].pos.x = obj.getVertex(poly.v[2] - 1).x;
        pVertices[stride*i + 2].pos.y = obj.getVertex(poly.v[2] - 1).y;
        pVertices[stride*i + 2].pos.z = obj.getVertex(poly.v[2] - 1).z;
    }
}

```

```

#include "display.h"
//#include<GL/GLU.h>

#include <GL/glew.h>
#include <string>
#include <iostream>

using namespace std;

Display::Display(int width,int height, std::string title)
{
    SDL_Init(SDL_INIT_EVERYTHING);//uses SDL2 to create

    //std::cout << "hello world" << std::endl;

    SDL_GL_SetAttribute(SDL_GL_RED_SIZE, 8);
    SDL_GL_SetAttribute(SDL_GL_GREEN_SIZE, 8);
    SDL_GL_SetAttribute(SDL_GL_BLUE_SIZE, 8);
    SDL_GL_SetAttribute(SDL_GL_ALPHA_SIZE, 8);
    SDL_GL_SetAttribute(SDL_GL_BUFFER_SIZE, 32);    //THESE SIZES ARE AT LEAST
    THESE VALUES.
    SDL_GL_SetAttribute(SDL_GL_DOUBLEBUFFER, 1);    //1 turns double buffering on
    in SDL.

    window = SDL_CreateWindow(title.c_str(), 100,100/* SDL_WINDOWPOS_CENTERED,
    SDL_WINDOWPOS_CENTERED*/, width, height, SDL_WINDOW_OPENGL | SDL_WINDOW_RESIZABLE );

    glContext = SDL_GL_CreateContext(window); //create the context
    SDL_GL_SetSwapInterval(0);
    GLenum status = glewInit(); //glew searches to find every single OGL function
    supported.
    if (status != GLEW_OK)
    {
        cerr << "Glew failed to initialize" << endl;
    }

    closed = false;
    glShadeModel(GL_FLAT);    //uses GL_SMOOTH or GL_FLAT

    //set viewport to screen size and initialize projection matrix. Leave with
    matrixmode = Modelview
    glViewport(0, 0, width, height); //make sure openGL view port is scaled to size
    of new window (if you want!) - why didn't I use freeglut?
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glFrustum(-1.0, 1.0, -1.0, 1.0, 1.5, 20.0);
    //          l,r,bot,top,near,far
    glMatrixMode(GL_MODELVIEW);
}

Display::~Display()
{
    SDL_GL_DeleteContext(glContext);

    SDL_DestroyWindow(window);
    SDL_Quit();
}

```

```

        //std::cout << "bye bye" << std::endl;
    }

    void GameObject(vector<GameObj>Objects)
    {
        for (int i = 0; i < Objects.size(); i++)
        {
            Objects[i].UpdateGameObj();
            cout << "Object" << " " << i << " " << "has now been updated" << endl;
        }
    }

    void Display::update(vector<GameObj>Objects)
    {
        SDL_Event e;
        while (SDL_PollEvent(&e)) //if an event occurs then process it and repeat til
event queue is empty - catches SDL_QUIT if it is in the queue
        {
            GameObject(Objects);

            cout << "event number:" << e.type << endl;
            if (e.type == SDL_QUIT)
                closed = true;

            if (e.window.event == SDL_WINDOWEVENT_RESIZED)
            {
                glViewport(0, 0, e.window.data1,e.window.data2); //make sure openGL view
port is scaled to size of new window (if you want!) - why didn't I use freeglut?
                cout << "resized to " << e.window.data1 << " , " <<
e.window.data2<<endl;
                glMatrixMode(GL_PROJECTION);
                glLoadIdentity();
                glFrustum(-1.0, 1.0, -1.0, 1.0, 1.5, 20.0);
                //          l,r,bot,top,near,far

                glMatrixMode(GL_MODELVIEW);
            }
        }
    }

    void Display::swapBuffers(void)
    {
        SDL_GL_SwapWindow(window);
    }

    void Display::render(Mesh* pMesh)
    {
        clear();// makes the point that the frame buffer is usually cleared before
redrawing
        glLoadIdentity();
        //glTranslatef(-5, -2, -10);//
        gluLookAt(5.0, 5.0, 5.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
        // 5,5,5 - place camera ; 0,0,0 - aim lens towards 0,0,0;
        // 0,1,0 - the up vector defines an unique orientation
        glColor3f(0.0, 1.0, 0.0);
        pMesh->draw();
        glTranslatef(1.0, 0.0, 0.0);
        glColor3f(1.0, 0.0, 0.0);
    }

```

```

        pMesh->draw();
        glTranslatef(2.0, 0.0, -1.0);
        glRotatef(90.0, 0.0, 1.0, 0.0);
        glColor3f(0.0, 0.0, 1.0);
        pMesh->draw();//mesh.draw(); // draw the model (s)
        //glScalef(0.5f, 0.5f, 0.5f);
        glRotatef(90, 0, 0, 1);
        glTranslatef(0.5, 0, 0);
        glColor3f(0.0, 1.0, 1.0);
        pMesh->draw();
        swapBuffers();
    }
    bool Display::isClosed()
    {
        return closed;
    }

    void Display::clear(void)
    {
        glClearColor(0.0f, 0.15f, 0.3f, 1.0f);
        glClear(GL_COLOR_BUFFER_BIT);
    }

#endif

#define DISPLAY_H
#include <SDL2/SDL.h>
#include <GL/glew.h>
#include <string>
#include "mesh.h"
#include "GameObj.h"
#include <vector>

using namespace std;

//uses SDL2 to create a window to display our OGL stuff
class Display
{
public:
    Display(int width, int height, std::string title);
    virtual ~Display();
    void swapBuffers();
    bool isClosed();
    void update(vector<GameObj>Objects);
    void clear(void);
    void render(Mesh *pMesh);
private:
    bool closed;
    SDL_Window* window;    //pointer to window
    SDL_GLContext glContext; //context for drawing OGL to SDL window
};

#endif //DISPLAY_H

```

FINAL GRADE

GENERAL COMMENTS

Instructor

57 / 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

DANIEL
O'DWYER

56.50

RESEARCH-INFORMED LITERATURE



No references used here although you do state you have used given materials when necessary. Please use standard referencing that uses the school's standard (IEEE). It's clear that you have investigated the relevant topic areas.

5

KNOWLEDGE AND UNDERSTANDING



Key areas addressed though you could have spent more time investigating the the fps code. Update works as does the obj file loader. The updater and game loop could be better integrated.

11

ANALYSIS



Key areas addressed. The updater task needs further work to integrate it with the game loop.. Evidence of test schedule design here, but this could be more sophisticated.

11

PRACTICAL APPLICATION AND DEP.



Obj loader incorporated - good. updater included but could have been better integrated into the game loop. Basic fps manager.

27

SKILLS FOR PROFESSIONAL PRAC.



Report structured. Testing could be more detailed.
Report structured. No bibliography/ ref. section.

2.5