

Rhys Lewis Games Architecture and Mechanics Semester 2 Report

by Rhys Lewis

FILE	12025_RHYS_LEWIS_RHYS_LEWIS_GAMES_ARCHITECTURE_AND_MECHANICS_SEMESTER_2_REPORT_254066_915082755.DOCX (333.41K)		
TIME SUBMITTED	07-MAY-2020 04:15AM (UTC+0100)	WORD COUNT	3120
SUBMISSION ID	126540875	CHARACTER COUNT	20165

Games Architecture and Mechanics Semester 2 Report

ACCA5017 GAMES ARCHITECTURE AND MECHANICS

RHYS LEWIS (1802146)

Table of Contents

Video Link.....	2
Task 1	3
Implementation	3
Testing.....	3
Task 2	5
Implementation	5
Testing.....	5
Task 3	7
Implementation	7
Testing.....	7
Appendix	9
Header Files.....	9
Display.h.....	9
GameObject.h	9
globals.h	10
mesh.h.....	10
objloader.h.....	11
transform.h	13
Source Files	14
Display.cpp.....	14
GameObject.cpp	17
mesh.cpp.....	17
objloader.cpp	19
Source.cpp	21
transform.cpp	25
References	26

Video Link

Here is the link to the video demonstrating the code, the video has been cut down for brevity, but any tests shown in the demonstration can be recreated using the program submitted.

Link: https://uowtsd-my.sharepoint.com/:v/g/personal/1802146_student_uwtsd_ac_uk/EXH7m_TMJnJGoOvZvDX1f9UBTrCFD_Cft2JOxUS8RyQ8fA?e=7FrHM0

Task 1

Implementation

For task 1, the author was tasked to create a timer for the game loop that outputted a frame rate every second, the author chose to make this frame rate fixed, as it allows for consistency between devices. To achieve this, the author makes use of a sleep pattern, whereupon reaching 60 frames per second (FPS) causes the system to sleep until the period of a second (1000 milliseconds) has passed, capping the FPS at 60 [1]. FPS is increased by one for every loop in the system. Once the timer reaches 1 second, the system outputs the total elapsed time in milliseconds and the current FPS (which should always be 60). The FPS and timing are then reset, and the cycle continues. Timing is calculated by using the “GetTickCount” function to calculate the time that has elapsed since the system has started [2]. For the purposes of this program, it is simply used to measure time. Elapsed time is calculated by subtracting the current tick count from the initial tick count, known as “TimeOld”. This results in the number of ticks that have passed since the program began, as “TimeOld” is defined using “GetTickCount” at the beginning of the program and during every reset.

```
float MS_PER_UPDATE = 16.67;
float lag = 0;
static unsigned long TimeOld = GetTickCount();
static unsigned long TimeCurrent = 0;
static unsigned long TimeElapsed = 0;
static int fps = 0;
```

Figure 2: Initial variable declarations

```
//Game Loop
TimeCurrent = GetTickCount();
TimeElapsed = TimeCurrent - TimeOld;

//Increase FPS
if (TimeElapsed > 1000)
{
    //Calculate results
    cout << "lag: " << lag << endl;
    cout << "time elapsed (ms): " << TimeElapsed << endl;

    //Reset variables
    fps = 0;
    TimeOld = TimeCurrent;
    lag = TimeElapsed;

    //Calculate Game
    //Calculate game speed (frames per second)
    lag = MS_PER_UPDATE;

    //FPS = 60
    //Sleep system to sleep
    Sleep(1000 - lag);

    //Increase FPS
    //Increase FPS by 1
    fps++;
}
```

Figure 1: Game Loop

Testing

To test this, the author simply set the program to run, with his setup, the program should output around 1000 milliseconds every second and output a consistent 60 fps.

Test No.	Time Elapsed	Setup Required	Result	Pass/Fail
1	5 seconds	The program needs to run	<pre>fps: 60 time elapsed (ms): 1047 fps: 60 time elapsed (ms): 1031 fps: 60 time elapsed (ms): 1015 fps: 60 time elapsed (ms): 1016 fps: 60 time elapsed (ms): 1016</pre>	Pass

Test No.	Time Elapsed	Setup Required	Result	Pass/Fail
2	10 seconds	The program needs to run	<pre>fps: 60 time elapsed (ms): 1047 fps: 60 time elapsed (ms): 1031 fps: 60 time elapsed (ms): 1015 fps: 60 time elapsed (ms): 1016 fps: 60 time elapsed (ms): 1016 fps: 60 time elapsed (ms): 1021 fps: 60 time elapsed (ms): 1016 fps: 60 time elapsed (ms): 1031 fps: 60 time elapsed (ms): 1015 fps: 60 time elapsed (ms): 1032</pre>	Pass

Test No.	Time Elapsed	Setup Required	Result	Pass/Fail
3	15 seconds	The program needs to run	<pre>fps: 60 time elapsed (ms): 1047 fps: 60 time elapsed (ms): 1031 fps: 60 time elapsed (ms): 1015 fps: 60 time elapsed (ms): 1016 fps: 60 time elapsed (ms): 1016 fps: 60 time elapsed (ms): 1031 fps: 60 time elapsed (ms): 1016 fps: 60 time elapsed (ms): 1031 fps: 60 time elapsed (ms): 1015 fps: 60 time elapsed (ms): 1032 fps: 60 time elapsed (ms): 1031 fps: 60 time elapsed (ms): 1016 fps: 60 time elapsed (ms): 1031 fps: 60 time elapsed (ms): 1047 fps: 60 time elapsed (ms): 1047</pre>	Pass

Task 2

Implementation

For task 2, the author was instructed to modify an existing object loader provided for him by the university to have it work with the new game loop to render .obj files. Originally, the loop would simply assign a set of values to an array of vertices and have the system draw that by assigning the vertices to a mesh class. This approach is incredibly ineffective as the system will only be able to draw this one shape through hard code and to draw another, you would have to hard code another shape.

The provided .obj loader accomplished most of the necessary tasks already, it just needed to be retooled for this loop. The author did this by first re-specifying the mesh class to accept an .obj file with a file name prompt rather than a list of vertices. The inputted file name would be placed into the obj loader's file loading function. It would then call a new function added to the obj loader named "polygonToVertex", this function would loop through the obj file, converting the polygons in the file to vertices to be used by the rendering function.

These steps were all that was needed to integrate the obj loader into the game loop.

Testing

Testing for this setup is rather straightforward. For this assessment, 3 .obj files were provided for use with the program. These files will each be loaded into the program to see if the program loads the file.

```
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 plane
vertices[31].pos = glm::vec3(1.0, 0.0, 0.0);
vertices[32].pos = glm::vec3(1.0, 1.0, 0.0);
```

Figure 3: Section of previous shape implementation

```
void Mesh::polygonToVertex(void)
{
    int jump = 3; //polygon = triangle
    vertexCount = jump * Obj.getNumPolys(); //3 points per polygon
    int polyCount = Obj.getNumPolys(); //number of polygons
    pVertices = new Vertex[vertexCount]; //number of vertices

    for (int i = 0; i < polyCount; i++) //for each polygon
    {
        polygon_type poly = Obj.getPolygon(i);
        pVertices[jump*i + 0].pos.x = Obj.getVertex(poly.v[0] - 1).x;
        pVertices[jump*i + 0].pos.y = Obj.getVertex(poly.v[0] - 1).y;
        pVertices[jump*i + 0].pos.z = Obj.getVertex(poly.v[0] - 1).z;

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

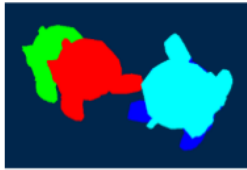
        pVertices[jump*i + 2].pos.x = Obj.getVertex(poly.v[2] - 1).x;
        pVertices[jump*i + 2].pos.y = Obj.getVertex(poly.v[2] - 1).y;
        pVertices[jump*i + 2].pos.z = Obj.getVertex(poly.v[2] - 1).z;

        //copy 3 vertices of a polygon to 3 vertices - each vertex has x,y,z
    }
}
```

Figure 4: polygonToVertex function

Test No.	Obj file	Setup Required	Result	Pass/Fail
1	"house.obj"	The file is present in the project folder. The file name is typed correctly into <pre>Mesh* pmesh = new Mesh("house.obj");</pre> mesh declaration.		Pass

Test No.	Obj file	Setup Required	Result	Pass/Fail
2	"blob.obj"	The file is present in the project folder. The file name is typed correctly into mesh declaration. <pre>Mesh* pmesh = new Mesh("blob.obj");</pre>		Pass

Test No.	Obj file	Setup Required	Result	Pass/Fail
3	"blendermonkeyobj.obj"	The file is present in the project folder. The file name is typed correctly into mesh declaration. <pre>Mesh* pmesh = new Mesh("blendermonkeyobj.obj");</pre>		Pass

The 3 obj files were rendered successfully with no need to alter the loader in any way between tests. Judging by these results, the author assumes the program will be able to render any obj given if the setup is performed correctly.

Task 3

Implementation

For task 3, the author was required to include an update manager for a container class representing game objects. For this task, the author created an entirely new class named “cGameObject”. The class contains an update function and room for future users to implement details. Currently, the update function is empty as it does not need to contain any details as the necessary code would be added by future users.

```
class cGameObject
{
public:
    cGameObject();
    ~cGameObject();
    void UpdateObject();

    //Details for Game Objects can be added in future
private:
}
```

Figure 5: cGameObject class

To update game objects, they must be initialized. At the beginning of the program, a vector of “cGameObject” is created. The author also included a function named “AddNewGameObjects”, which can be used to add new game objects in future.

```
vector <cGameObject> GameObjects(2);

void AddNewGameObject(cGameObject NewObject)
{
    //Add GameObject details
    GameObjects.push_back(NewObject);
}
```

Figure 6: Initialization of Vector and AddNewGameObject function

The only other thing needed is to implement the update function into the game loop’s update pattern. To do this, the existing update function was modified to add the newly initialized vector as an argument, now that the update function has access to the vector, a new function was created named “ObjectUpdate” that looped through every single item in the vector and called its update function. To prove that it is working, the system would then output an update message on each cycle. This function would then be called inside of the update function, meaning so upon every update, the game objects will all update.

```
void ObjectUpdate(vector<cGameObject> GameObjects)
{
    for (int i = 0; i < GameObjects.size(); i++)
    {
        GameObjects[i].UpdateObject();
        cout << "GameObject " << i << " has been updated" << endl;
    }
}

void Display::update(vector<cGameObject> GameObjects)
{
    SDL_Event e;
    while (SDL_PollEvent(&e)) //If an event occurs then process it
    {
        ObjectUpdate(GameObject);
    }
}
```

Figure 7: ObjectUpdate function and it being called inside of Display::update

Testing

To test this setup, the author will set the vector to different initial sizes and will record whether or not the system outputs the desired results.

Test No.	Vector size	Setup Required	Result	Pass/Fail
1	2	The initial Vector size is correct <code>vector <cGameObject> GameObjects(2);</code>	<pre>GameObject 0 has been updated GameObject 1 has been updated event number:1024 GameObject 0 has been updated GameObject 1 has been updated event number:1024 GameObject 0 has been updated GameObject 1 has been updated event number:1024 GameObject 0 has been updated GameObject 1 has been updated event number:1024 GameObject 0 has been updated GameObject 1 has been updated event number:1024</pre>	Pass

Test No.	Vector size	Setup Required	Result	Pass/Fail
2	10	The initial Vector size is correct <pre>vector<cGameObject> GameObjects(10);</pre>	<pre>GameObject 0 has been updated GameObject 1 has been updated GameObject 2 has been updated GameObject 3 has been updated GameObject 4 has been updated GameObject 5 has been updated GameObject 6 has been updated GameObject 7 has been updated GameObject 8 has been updated GameObject 9 has been updated event number:1024 GameObject 0 has been updated GameObject 1 has been updated GameObject 2 has been updated GameObject 3 has been updated GameObject 4 has been updated GameObject 5 has been updated GameObject 6 has been updated GameObject 7 has been updated GameObject 8 has been updated GameObject 9 has been updated event number:1024</pre>	Pass

Test No.	Vector size	Setup Required	Result	Pass/Fail
3	5	The initial Vector size is correct <pre>vector<cGameObject> GameObjects(5);</pre>	<pre>event number:1024 GameObject 0 has been updated GameObject 1 has been updated GameObject 2 has been updated GameObject 3 has been updated GameObject 4 has been updated event number:1024 GameObject 0 has been updated GameObject 1 has been updated GameObject 2 has been updated GameObject 3 has been updated GameObject 4 has been updated event number:1024 GameObject 0 has been updated GameObject 1 has been updated GameObject 2 has been updated GameObject 3 has been updated GameObject 4 has been updated event number:1024 GameObject 0 has been updated GameObject 1 has been updated GameObject 2 has been updated GameObject 3 has been updated GameObject 4 has been updated</pre>	Pass

Appendix

Header Files

Display.h

```
#ifndef DISPLAY_H
#define DISPLAY_H
#include <SDL2/SDL.h>
#include <GL/glew.h>
#include <string>
#include "mesh.h"
#include "globals.h"
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<cGameObject> GameObjects);
    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
```

GameObject.h

```
#pragma once

class cGameObject
{
public:
    cGameObject();
    ~cGameObject();
    void UpdateObject();

    //Details for Game Objects can be added in future
private:
};
```

```

globals.h
#pragma once
#include "GameObject.h"
#include <vector>

mesh.h
#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:
    object_type Obj;
    Mesh(const char* f);
    //int loadOBJ();
    void polygonToVertex(void);
    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

```

```

objloader.h
#ifndef OBJLOAD
#define OBJLOAD
/*****
*
*Loads obj file - limited to vertices, faces, normals, texture maps
*loads to object structure as defined in .h file
*****/

#define MAX_VERTICES 8000 // Max number of vertices (for each object)
#define MAX_POLYGONS 8000 // Max number of polygons (for each object)
#define MAX_NORMALS 8000 // Max number of polygons (for each object)

// Our vertex type
typedef struct {
    float x, y, z;
}vertex_type;

// Our normal type
typedef struct {
    float i, j, k;
}normcoord_type;

// The polygon (triangle), 3 numbers that aim 3 vertices
typedef struct {
    int v[3], t[3], n[3];
}polygon_type;

// The mapcoord type, 2 texture coordinates for each vertex
typedef struct {
    float u, v;
}mapcoord_type;

// The object type
class object_type {
public:
    int id_texture;
    object_type() {}
    ~object_type() {}
    int objloader(const char *p_filename);
    int objdatadisplay();
    int getNumPolys(void) {return polygons_qty;}
    polygon_type getPolygon(int i) { return polygon[i]; }
    vertex_type getVertex(int i) { return vertex[i]; }

private:
    char name[20];
    int vertices_qty;
    int polygons_qty;
    int mapcoord_qty;
    int normcoord_qty;

    vertex_type vertex[MAX_VERTICES];
    mapcoord_type mapcoord[MAX_VERTICES];
    normcoord_type normcoord[MAX_NORMALS];

```

```
    polygon_type polygon[MAX_POLYGONS];  
};  
  
#endif  
  
/*****/
```

```
transform.h
#ifndef TRANSFORM_H
#define TRANSFORM_H
#include <glm/glm.hpp>
#include <GL/glew.h>
#include <string>

//Holds a mesh's transform
class Transform
{
public:
    Transform();
    Transform(glm::mat4x4* m);
    virtual ~Transform();

    void UpdateMV(glm::mat4x4* m);

private:
    glm::mat4x4 mv;
};

#endif
```

Source Files

Display.cpp

```
#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 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 ObjectUpdate(vector<cGameObject> GameObjects)
    {
        for (int i = 0; i < GameObjects.size(); i++)
        {
            GameObjects[i].UpdateObject();
            cout << "GameObject " << i << " has been updated" << endl;
        }
    }

    void Display::update(vector<cGameObject> GameObjects)
    {
        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
        {

```

```

        ObjectUpdate(GameObjects);

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

```

GameObject.cpp

```
#include "globals.h"
```

```
cGameObject::cGameObject()  
{  
}
```

```
cGameObject::~~cGameObject()  
{  
}
```

```
void cGameObject::UpdateObject()  
{  
    //Place GameObject Update Code here  
}
```

mesh.cpp

```
#include "mesh.h"  
#include <iostream>  
#include <fstream>  
#include <string>  
using namespace std;
```

```
Mesh::Mesh(const char* f)
```

```
{  
    Obj.objloader(f);  
    polygonToVertex();  
    cout << "Loaded" << endl;  
    //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::polygonToVertex(void)
{
    int jump = 3; //polygon = triangle
    vertexCount = jump * Obj.getNumPolys(); //3 points per polygon
    int polyCount = Obj.getNumPolys(); //number of polygons
    pVertices = new Vertex[vertexCount]; //number of vertices

    for (int i = 0; i < polyCount; i++) //for each polygon
    {
        polygon_type poly = Obj.getPolygon(i);
        pVertices[jump*i + 0].pos.x = Obj.getVertex(poly.v[0] - 1).x;
        pVertices[jump*i + 0].pos.y = Obj.getVertex(poly.v[0] - 1).y;
        pVertices[jump*i + 0].pos.z = Obj.getVertex(poly.v[0] - 1).z;

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

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

        //copy 3 vertices of a polygon to 3 vertices - each vertex has x,y,z
    }
}

```

```

objloader.cpp
#include <iostream>
#include <fstream>
#include "objloader.h"

using namespace std;
int object_type::objloader(const char * f)
{
    FILE *l_file; //File pointer

    int ipolygon = 0, imap = 0, ivertex = 0, inormal = 0;
    char l_char; //Char variable
    char myString[256];
    // int vertex;
    unsigned short l_face_flags; //Flag that stores some face information

    if (( l_file=fopen(f, "rt")) == NULL) return 0; //Open the file

    while (!feof(l_file)) //Loop to scan the whole file
    {

        fscanf(l_file, "%c", &l_char);
        if (l_char == '\n')//read char if '\n' -skip to next and read
            fscanf(l_file, "%c", &l_char);
        switch (l_char) //parse
        {
            default: fgets(myString, 256, l_file);
                    break;
            case 'v': //a vertex or a normal or a text co-ord
                    fscanf(l_file, "%c", &l_char);
                    switch (l_char)
                    {
                        case ' ': //a vertex -expect and so read 3 floats next
                                fscanf(l_file, "%f %f %f", &vertex[ivertex].x,
                                &vertex[ivertex].y, &vertex[ivertex].z);

                                ivertex++;
                                break;
                        case 'n': //a normal -expect and so read 3 floats next
                                fscanf(l_file, "%f %f %f", &normcoord[inormal].i,
                                &normcoord[inormal].j, &normcoord[inormal].k);
                                inormal++;
                                break;
                        case 't': //a texture map coord-expect and so read 2 floats next
                                fscanf(l_file, "%f %f", &mapcoord[imap].u, &mapcoord[imap].v);
                                imap++;
                                break;
                    } //end switch
                    break;
            case 'f': //a face read next assume format is -> f 1/1/1 2/2/2 3/3/3
                    for (int i = 0; i < 3; i++) //for each vertex of polygon (triangle)
                    {
                        fscanf(l_file, "%c", &l_char); //read space char - ignore this
                        fscanf(l_file, "%d", &polygon[ipolygon].v[i]); //read vertex.
                    }
                }
    }
}

```

```

                                fscanf(l_file, "%c", &l_char); //read space char - ignore this
                                fscanf(l_file, "%d", &polygon[ipolygon].t[i]); //read text
coord.
                                fscanf(l_file, "%c", &l_char); //read space char - ignore this
                                fscanf(l_file, "%d", &polygon[ipolygon].n[i]); //read normal.

                                }
                                ipolygon++;
                                break;

                                } //end switch
                                }
                                fclose(l_file); // Closes the file stream
                                vertices_qty = ivertex;
                                polygons_qty = ipolygon;
                                mapcoord_qty = imap;
                                normcoord_qty = inormal;
                                return 0;
                                }

```

Source.cpp

```
#include "mesh.h"
#include "Display.h"
//
#include "glut.h"
#include <Windows.h>
#include <iostream>
#include "objloader.h"
#include <ctime>
using namespace std;

vector <cGameObject> GameObjects(5);

void AddNewGameObject(cGameObject NewObject)
{
    //Add GameObject details
    GameObjects.push_back(NewObject);
}

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");

float MS_PER_UPDATE = 16.67;
float lag = 0;
static unsigned long TimeOld = GetTickCount();
static unsigned long TimeCurrent = 0;
static unsigned long TimeElapsed = 0;
static int fps = 0;
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;
//static int fps = 0; //frame count;
//dwTimeElapsed = GetTickCount() - dwOldTime;
//fps++;
//if (dwTimeElapsed > 5000)
//{
//    cout << "frames per sec = " << (float)fps / 5.0 << std::endl;
//    fps = 0;
//    dwOldTime = GetTickCount();
//}
//myDisplay.update(GameObjects);
//myDisplay.render(pmesh);

//start = clock();

//int buffer = 0;
////Player Inputs would go here
//for (int i = 0; i < 1000000; i++)
//{
//    buffer++;
//}
//end = clock();
//float diff = end - OldTime;
//OldTime = diff;
//lag += diff;
//static int fps = 0; //frame count;
//fps++;

//while (lag >1/60)
//{
//    myDisplay.update(GameObjects);
//    lag -= 1.0 / 60.0;
//}
//cout << "frames per sec = " << (float)fps << endl;
//fps = 0;
//myDisplay.render(pmesh);

//Sets Timers
TimeCurrent = GetTickCount();
TimeElapsed = TimeCurrent - TimeOld;

```

```

//Increases FPS
fps++;
if (TimeElapsed > 1000)
{
    //Outputs results
    cout << "fps: " << fps << endl;
    cout << "time elapsed (ms): " << TimeElapsed << endl;

    //Resets variables
    fps = 0;
    TimeOld = TimeCurrent;
}

lag += TimeElapsed;

while (lag >= MS_PER_UPDATE)
{
    //Updates Game
    myDisplay.update(GameObjects);
    lag -= MS_PER_UPDATE;
}

if (fps == 59)
{
    //Casues System to sleep
    Sleep(1000 - lag);
}

//Renders scene
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;
}

```

```
transform.cpp
#include "transform.h"
Transform::Transform()
{

}
Transform::Transform(glm::mat4x4* m)
{

}
Transform::~~Transform()
{

}

void Transform::UpdateMV(glm::mat4x4* m)
{

}
```

References

- [1] R. Nystrom, "Game Programming Patterns: Game Loop," [Online]. Available: <https://gameprogrammingpatterns.com/game-loop.html>.
- [2] Microsoft, "GetTickCount function," Microsoft, 5 December 2018. [Online]. Available: <https://docs.microsoft.com/en-us/windows/win32/api/sysinfoapi/nf-sysinfoapi-gettickcount>.

Rhys Lewis Games Architecture and Mechanics Semester 2 Report

GRADEMARK REPORT

FINAL GRADE

GENERAL COMMENTS

Instructor

70/100

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

GRADING FORM: SMU GENERIC_GJ

RHYS LEWIS

69.50

RESEARCH-INFORMED LITERATURE



Refs used are appropriate but try to include more refs to support your investigations. Appropriate topics covered.

6

KNOWLEDGE AND UNDERSTANDING



Sleeping to manage tfps can be dangerous- you're better off allowing the update method to use this time

14

as it see fit, given its constraints.

ANALYSIS



Problems well interpreted on the whole- more detailed testing of update loop within game loop would help. Good SE practices included here.

13

PRACTICAL APPLICATION AND DEP.



Some good work here. Obj loader works and is tested for a variety of obj objects. Good. The update loop is included but please be aware that your sleep loop approach is potentially wasteful. Update task implemented using OO techniques - good. More detailed load testing would have been good to see here.

33.5

SKILLS FOR PROFESSIONAL PRAC.



A conclusion/summary at the end would be nice to see. Otherwise structured with use of some refs. Try to review a wider range texts here.

3