

1602835 Jake Russell Games Arch Assignment 2

by Jake Russell

FILE	12019_JAKE_RUSSELL_1602835_JAKE_RUSSELL_GAMES_ARCH_ASSIGNMENT_2_254066_71445080.DOCX (252.21K)		
TIME SUBMITTED	07-MAY-2020 06:00PM (UTC+0100)	WORD COUNT	1794
SUBMISSION ID	126661265	CHARACTER COUNT	11875

Games Architecture and Mechanics

Assignment 2

Jake Russell

1602835

Table of Contents

Task 1	3
Task 2	3
Task 3	6
Appendix	10
Display.h.....	10
GameObject.h	10
Mesh.h	11
Objloader.h	12
Transform.h.....	13
Display.cpp	14
GameObject.cpp	16
Mesh.cpp.....	17
Objloader.cpp	18
Source.cpp	20
Transform.cpp	20

Task 1

For task 1, in order to update at a fixed rate and have it render at a potentially variable rate, first the render() method and update method() were separated by placing the update() method in a contained while loop which loops at a fixed rate. This was done with the following code:

```
const float ms_per_update = 1000/60; //60fps
float lag = 0.0;
static unsigned long dwOldTime = GetTickCount();
static unsigned long dwTimeElapsed;
static unsigned long dwCurrentTime;
while (!myDisplay.isClosed()) // Application loop
{
    dwCurrentTime = GetTickCount();
    dwTimeElapsed = dwCurrentTime - dwOldTime;
    dwOldTime = dwCurrentTime;
    lag += dwTimeElapsed;
    while (lag >= ms_per_update)
    {
        myDisplay.update(Objects);
        lag -= ms_per_update;
    }
    myDisplay.render(pmesh);
}
```

Figure 1: Code for fixed rate timing

In order for the loop to run at 60 fps, a variable called ms_per_update was made, and it was set the value of 1000/60 which is 16.666 recurring being the millisecond delay between updates when it is updating at 60 fps. This calculation being how many milliseconds in a second (1000) divided by the desired fps, being in this case 60. The lag variable is made as a way to track how far behind the loop is compared to real time, the inner while loop with the update() method then loops until the main loops clock has caught up.

Task 2

For task 2 to integrate the obj loader into the prototype an object of class obj and a new constructor called Mesh(const char *f) were made within the mesh class. A function was also made to convert the polygons from the obj file into a vertex list called objPoly2Vertex:

```

#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);
    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

```

Figure 2: task 2 mesh.h

In the mesh.cpp class, the object is called from the object loader through obj.objloader(f), and it is then converted by calling objPoly2Vertex:

```

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

```

Figure 3: task 2 mesh.cpp

```

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;
    }
}

```

Figure 4: mesh.cpp convert function

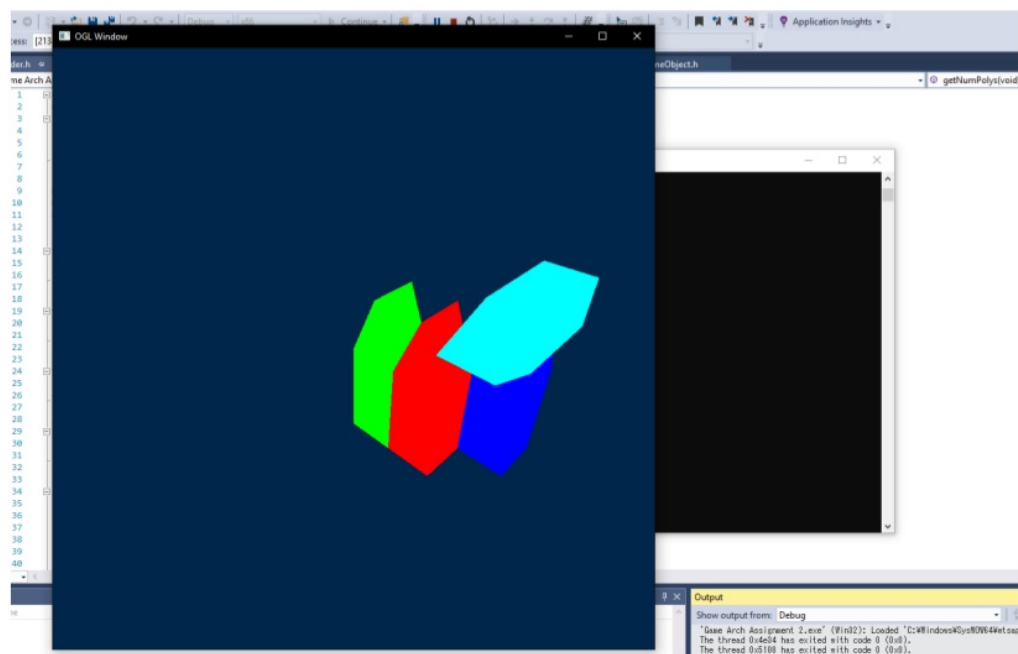


Figure 5: Task 2 House.obj

Task 3

For task 3 a class called GameObject was made as the container class. A vector was then made called Objects which takes in objects of type GameObject with the initial amount set as 3. Within the GameObject class, an empty function was made named GUpdate to be called later. Next in the display class, the display classes update function was adjusted to be to take in the vector "Objects". A new function was inside display.cpp named ObjectUpdate which outputs to the user that objects have been updated. This function is then made to be called within the display classes update function. Lastly in the source.cpp, the gameloop was made to send the vector "Objects" when calling myDisplay.update.

```
vector<GameObject>Objects(3);
int main(int argc, char *args[])
{
    Display myDisplay(800, 800, "OGL Window"); //create and open a window

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

    const float ms_per_update = 1000/60; //60fps
    float lag = 0.0;
    static unsigned long dwOldTime = GetTickCount();
    static unsigned long dwTimeElapsed;
    static unsigned long dwCurrentTime;
    while (!myDisplay.isClosed()) // Application loop
    {
        dwCurrentTime = GetTickCount();
        dwTimeElapsed = dwCurrentTime - dwOldTime;
        dwOldTime = dwCurrentTime;
        lag += dwTimeElapsed;
        while (lag >= ms_per_update)
        {
            myDisplay.update(Objects);
            lag -= ms_per_update;
        }

        myDisplay.render(pmesh);
    }
    return 0;
}
```

Figure 6:task 3 source.cpp

```

1      #include "GameObject.h"
2
3      GameObject::GameObject()
4      {
5      }
6
7      GameObject::~~GameObject()
8      {
9      }
10
11     void GameObject::GUpdate()
12     {
13     }
14

```

Figure 7:task 3 GameObject.cpp

```

1      #pragma once
2      class GameObject
3      {
4      public:
5          GameObject();
6          ~GameObject();
7          void GUpdate();
8      private:
9
10     };
11
12

```

Figure 8:task 3 GameObject.h

```

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

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

```

Figure 9: task 3 Display.cpp

```
event number:1024  
Object 0 has been updated  
Object 1 has been updated  
Object 2 has been updated  
event number:1024  
Object 0 has been updated  
Object 1 has been updated  
Object 2 has been updated  
event number:1024  
Object 0 has been updated  
Object 1 has been updated  
Object 2 has been updated  
event number:1024  
Object 0 has been updated  
Object 1 has been updated  
Object 2 has been updated  
event number:1024  
Object 0 has been updated  
Object 1 has been updated  
Object 2 has been updated  
event number:1024  
Object
```

Figure 10: task 3 test

Appendix

Display.h

```
#ifndef DISPLAY_H
#define DISPLAY_H
#include <SDL2/SDL.h>
#include <GL/glew.h>
#include <string>
#include "mesh.h"
#include <vector>
#include "GameObject.h"
//uses SDL2 to create a window to display our OGL stuff
using namespace std;
class Display
{
public:
    Display(int width, int height, std::string title);
    virtual ~Display();
    void swapBuffers();
    bool isClosed();
    void update(vector <GameObject>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
```

GameObject.h

```
#pragma once
class GameObject
{
public:
    GameObject();
    ~GameObject();
    void GOupdate();
private:
};
```

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

```

Display.cpp

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

#include <GL/glew.h>
#include <string>
#include <iostream>
#include <vector>
#include "GameObject.h"
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 ObjectUpdate(vector<GameObject>Objects)
    {
        for (int i = 0; i < Objects.size(); i++)
        {
            Objects[i].GOupdate();
            cout << "Object " << i << " has been updated" << endl;
        }
    }

    void Display::update(vector<GameObject>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
        {
            ObjectUpdate(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, 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);
    }

```

GameObject.cpp

```

#include "GameObject.h"

GameObject::GameObject()
{
}

GameObject::~GameObject()
{
}

void GameObject::G0update()
{
}

```

Mesh.cpp

```
#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;
}

Mesh::~Mesh()
{
}

void Mesh::draw()
{
    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;
    }
}

```

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_s(l_file, "%c", &l_char);
        if (l_char == '\n')//read char if '\n' -skip to next and read
            fscanf_s(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_s(l_file, "%c", &l_char);
                    switch (l_char)
                    {
                        case ' ': //a vertex -expect and so read 3 floats next
                                fscanf_s(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_s(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_s(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
    {
        fscanf_s(l_file, "%c", &l_char); //read space char -
        fscanf_s(l_file, "%d", &polygon[ipolygon].v[i]); //read
        fscanf_s(l_file, "%c", &l_char); //read space char -
        fscanf_s(l_file, "%d", &polygon[ipolygon].t[i]); //read
        fscanf_s(l_file, "%c", &l_char); //read space char -
        fscanf_s(l_file, "%d", &polygon[ipolygon].n[i]); //read

        }
        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 <vector>
#include "GameObject.h"

using namespace std;

vector <GameObject>Objects(3);
int main(int argc, char *args[])
{
    Display myDisplay(800, 800, "OGL Window"); //create and open a window

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

    const float ms_per_update = 1000/60; //60fps
    float lag = 0.0;
    static unsigned long dwOldTime = GetTickCount();
    static unsigned long dwTimeElapsed;
    static unsigned long dwCurrentTime;
    while (!myDisplay.isClosed()) // Application loop
    {
        dwCurrentTime = GetTickCount();
        dwTimeElapsed = dwCurrentTime - dwOldTime;
        dwOldTime = dwCurrentTime;
        lag += dwTimeElapsed;
        while (lag >= ms_per_update)
        {
            myDisplay.update(Objects);
            lag -= ms_per_update;
        }

        myDisplay.render(pmesh);
    }
    return 0;
}
```

Transform.cpp

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

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

Transform::~~Transform()
{
}

void UpdateMV(glm::mat4x4* m)
{
}
```

1602835 Jake Russell Games Arch Assignment 2

GRADEMARK REPORT

FINAL GRADE

GENERAL COMMENTS

Instructor

52/100

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

GRADING FORM: SMU GENERIC_GJ

JAKE RUSSELL**52****RESEARCH-INFORMED LITERATURE**

No references used here. Please use standard referencing that uses the school's standard (IEEE). It's clear that you have investigated the relevant topic areas- reference your sources.

4.5**KNOWLEDGE AND UNDERSTANDING**

Obj loader incorporated. Try testing with multiple files of differing objects. fps manager within game loop reasonably sophisticated - reference this pattern! Straightforward Updater included and demonstrated.

11**ANALYSIS**

Some good choices made but no formal test design in evidence. Some integration of updater and game loop - good.

11**PRACTICAL APPLICATION AND DEP.**



Obj file loader integrated but testing of this with various files not attempted (not in evidence in report). Updater created and incorporated. Good. fps designed and code shown but no screen shots or testing shown! Code does not display fps. No testing apparent here. Overall little testing or test results in evidence here.

23

SKILLS FOR PROFESSIONAL PRAC.



Structured. Testing is very light. No refs. A summary would help here.

2.5