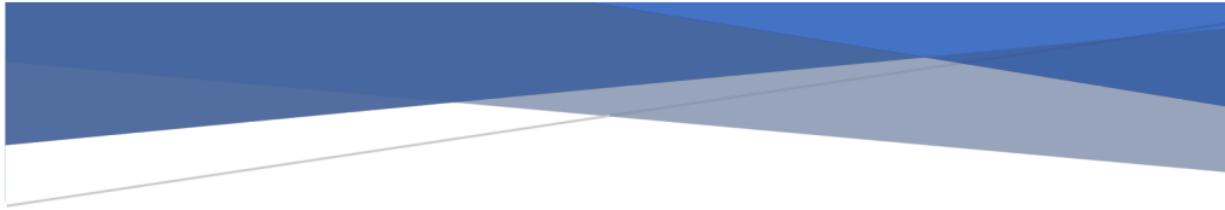


Games Architecture and Mechanics Part 1b

by May Davies

FILE	17857_MAY_DAVIES_GAMES_ARCHITECTURE_AND_MECHANICS_PART_1B_314930_228761812.DOCX (1.24M)		
TIME SUBMITTED	06-JAN-2021 10:21PM (UTC+0000)	WORD COUNT	1714
SUBMISSION ID	141059083	CHARACTER COUNT	8439



GAMES ARCHITECTURE AND MECHANICS

Part 1b

May Davies

Contents

1.Declaration.....	2
2.Abstract.....	2
3.Introduction	2
4.Game loops	2
4.1 Considered Game Loop.....	2
4.2 Chosen Game Loop	3
4.3 Implementation	3
4.4 Considered future implementations.....	5
References	5
Appendix A – Test logs and Extra Descriptions of Features.....	6
Appendix B – Code	13
Source.cpp	13
Display.h.....	14
Display.cpp.....	15
Objects.h	17
Objects.cpp	18
Mesh.h	18
Mesh.cpp.....	19
Transform.h.....	19
Transform.cpp.....	20

1.Declaration

The work of this report has been completed under the supervision of Gordon Dickers. I declare that this report is the product of my own work and is not being submitted for any other degree etc.

May Davies.

2.Abstract

Game loops are discussed in this report, as well as a breakdown of a chosen game loop, how features were implemented, future considerations, and then an appendix containing testing data and code.

3.Introduction

This report is a report that has been made in order to describe a Game Loop, which is typically present in most games. There are several different methods in which such a game loop can be done, two of which are described in this report.

4.Game loops

A game simple loop that loops for as long as a game window is open. At a basic level, this loop can consist of an input function to handle user inputs, an update function, which is responsible for handling object and object updates (i.e physics, and other changes) and then a render function which will display the game to the user.

4.1 Considered Game Loop

The game loop that was initially considered would have looked something similar to this:

```
oldTime = getCurrentTime();
while (game.isStillRunning)
{
    currentTime=getCurrentTime();
    elapsedTime=currentTime – oldTime;
    game.input();
    game.update(elapsedTime);
    game.render();
    oldTime = currentTime;
}
```

For players with faster machines, this is a very rewarding game loop as updates and rendering occurs much more often than it would on lower end machines [1].

This loop is not without fault. It was not chosen because the rate that the game updates will be different on machines with varying system specifications [1]. This is a problem is because if a game is multiplayer, and an object is being updated between two or more players who have different hardware at different speeds, objects that have had updates performed on them will be in different positions or states between each player [1]. For instance, one player might throw a grenade, on one players end it could land next to them, while on the other players end it could land 20 ft behind

them instead.

4.2 Chosen Game Loop

The following algorithm is an example of the Game Loop that was chosen for this particular case.

```
oldTime = getCurrentTime();
msForEachUpdate = 16.67;
lag = 0.0;
while (game.isStillRunning)
{
    currentTime = getCurrentTime();
    elapsedTime = currentTime - oldTime;
    oldTime = currentTime;
    lag = lag + elapsedTime;

    game.input();
    while ( lag > msForEachUpdate)
    {
        game.update();
        lag = lag - msForEachUpdate;
    }
    game.render();
}
```

This game loop was chosen over the previously discussed game loop is because the game is being updated at a fixed rate[1] based on if lag is greater than the value of msForEachUpdate(16) (The reason its about 16 is because the aim in this particular scenario is to update the game consistent to 60FPS, 1000ms/60 happens to be around 16~).

The big problem with the game loop through this approach though, is that on lower end machines, due to the render only being performed after the little “catch up” while loop, the game may appear to be choppy than one would prefer as the machine would be waiting on updates to be performed before rendering the scene once more [1]. However, this is still better than the first example as it means that for the case of multiplayer games things will be less likely to be out of sync between player to player.

4.3 Implementation

So, how did this loop get implemented in practice? Well, the image below is a screenshot of the code.

```

float msForEachUpdate = 16.67; //This is the ms between each update.
float lag = 0.0; //This lag is introduced so that updates are done at a fixed rate.
long timerOldTime = GetTickCount64(); //keeps track of time for game loop.
long timerTimeElapsed; //keeps track of the time since the previous update.
long timerCurrentTime; //The current time.
long fpsStart; //timer for FPS rate
long fpsTime; //timer for FPS rate
while (!gameLoopWindow.isClosed()) // Game loop. As long as the SDL window is open, this loop will repeat over and over.
{
    timerCurrentTime = GetTickCount64(); //sets the game loops time to the current time.
    timerTimeElapsed = timerCurrentTime - timerOldTime; //keeps track of the time between updates, by taking the previous time from the current time.
    lag = lag + timerTimeElapsed;
    //input begins here
    //Registered inputs are A W S D.
    //These will change the XYZ values for the Eye values of glLookAt, by changing the variable used in .update()
    gameLoopWindow.input();

    //This forces the game to update a certain number of times per each game loop. The point of this is to
    while (lag >= msForEachUpdate)
    {
        //our "objects" update here. In this case these "objects" are just dummies, that have some variables set to a random number and an empty for lo
        gameLoopWindow.update();
        lag = lag - msForEachUpdate;
        std::cout << "Update performed." << std::endl;
    }
    std::cout << "Old time is: " << timerOldTime << " Current time is: " << timerCurrentTime << " Elapsed time is: " << timerTimeElapsed << std::endl;
    gameLoopWindow.render(pmesh);
    std::cout << "Render performed." << std::endl;
    timerOldTime = timerCurrentTime;
    //But you already have a delay set up above wrapped around the update!
    //Yes, well, I used a program called FRAPS which has an FPS monitor and found that the actual FPS
    //was hitting much higher than 60 in reality, it was reaching numbers up to 210+ fps on this machine.
    //This led me to looking in to SDL_Delay instead.
    //This is used to cap the SDL windows FPS to 60.
    fpsTime = GetTickCount64() - timerCurrentTime;
    if (msForEachUpdate > fpsTime)
    {
        SDL_Delay(msForEachUpdate - fpsTime);
        std::cout << "Fps has been delayed." << std::endl;
    }
}
return 0;

```

Some changes were made to the game loop as can be seen, `SDL_Delay` was used, as even with the game loop implemented as previously discussed, open using a program called FRAPS which can be used to monitor frame rate [2], the frame rate was higher than 60.

Other features that were implemented included updates to dummy objects, inputs to control the camera (WASD keys), depth, and lighting (though lighting is currently disabled).

For testing and further explanations of some of these implementations, please see Appendix A.

For the full code of this program, please see Appendix B.

4.4 Considered future implementations

Something that would be considered for future would be an object loader. The reason for this is because objects can become much more complicated than the house in this scenario, and it would be impractical to write complex models manually. Thus, a program would need to be used to make models, and then an object loader would be used to load these models in. Here is an example of an object file for a cube, which can be found in this programs files under the name of "box.obj"

```
1  # Blender v2.74 (sub 0) OBJ File: ''
2  # www.blender.org
3  mtl lib box.mtl
4  o Box
5  v 0.500000 0.500000 0.500000
6  v 0.500000 0.500000 -0.500000
7  v 0.500000 -0.500000 0.500000
8  v 0.500000 -0.500000 -0.500000
9  v -0.500000 0.500000 0.500000
10 v -0.500000 0.500000 -0.500000
11 v -0.500000 -0.500000 0.500000
12 v -0.500000 -0.500000 -0.500000
13 vt 0.000000 1.000000
14 vt 0.000000 0.000000
15 vt 1.000000 0.000000
16 vt 1.000000 1.000000
17 vn 1.000000 0.000000 0.000000
18 vn -1.000000 0.000000 0.000000
19 vn 0.000000 1.000000 0.000000
20 vn 0.000000 -1.000000 0.000000
21 vn 0.000000 0.000000 1.000000
22 vn 0.000000 0.000000 -1.000000
23 usemtl b0b0b0
24 s 1
25 f 1/1/1 3/2/1 4/3/1 2/4/1
26 f 5/1/2 7/2/2 8/3/2 6/4/2
27 f 5/1/3 6/2/3 1/3/3 2/4/3
28 f 8/1/4 7/2/4 4/3/4 3/4/4
29 f 6/1/5 8/2/5 3/3/5 1/4/5
30 f 2/1/6 4/2/6 7/3/6 5/4/6
31
```

The letter at the start of each line represents different things. With # being comments, v representing vertex, vt representing texture coordinates of a vertex, vn representing normal coordinates of a vertex, and f which represents a face made up of 3 different vertices of each to create a triangle [3].

The aim of an object loader is to read this file, and then use this data in order to draw objects.

References

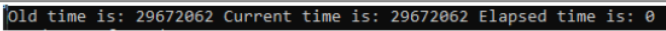
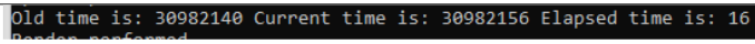
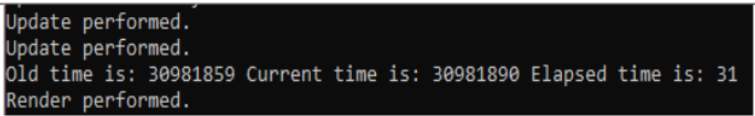
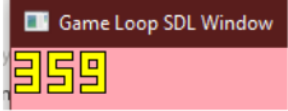
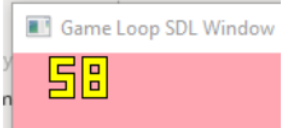
- [1] R. Nystrom, "Game Programming Patterns," 2014. [Online]. Available: <https://gameprogrammingpatterns.com/game-loop.html>. [Accessed 27 12 2020].
- [2] Beepa Pty Ltd, "FRAPS show fps, record video game movies, screen capture software," Beepa Pty Ltd, [Online]. Available: <https://fraps.com/>. [Accessed 6 1 2021].
- [3] opengl-tutorial.org, "Tutorial 7: Model Loading," [Online]. Available: <http://www.opengl-tutorial.org/beginners-tutorials/tutorial-7-model-loading/>. [Accessed 28 12 2020].

Appendix A – Test logs and Extra Descriptions of Features

Timer Tests

The following tests were tests performed on the timer, displaying results and changes made.

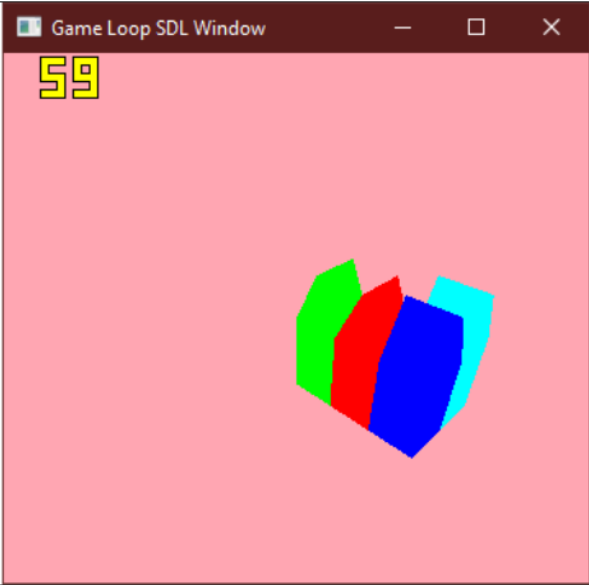
By the end, the program was keeping track of time for the update system, and time for SDL_Delay to make use of.

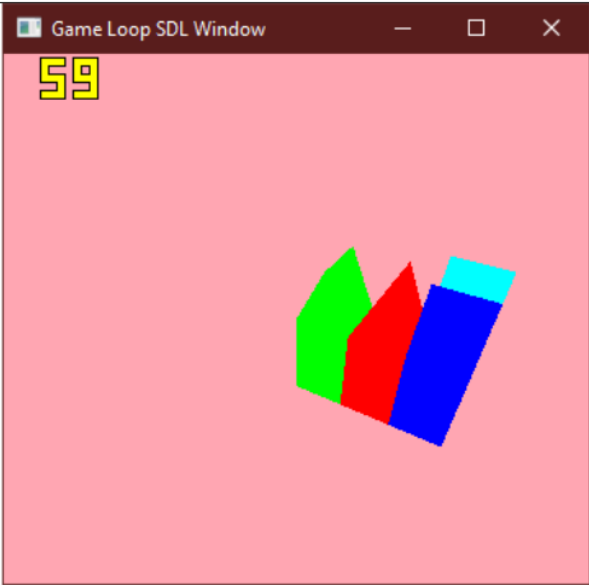
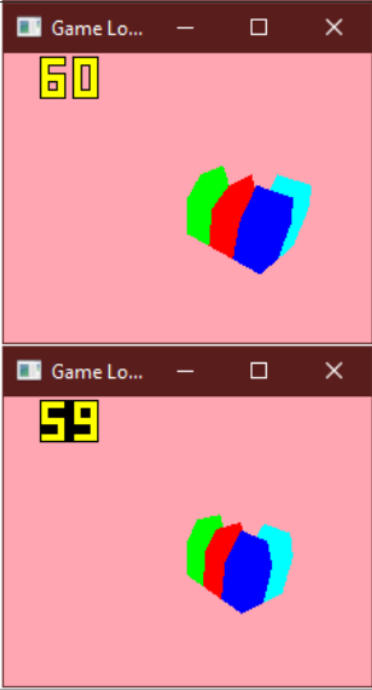
Test No.	Description	Expected Result	Actual Result	Success Y/N? Other notes
1	Check that old time is different to current time	Old time should be different to current time		N. Perhaps changing old time to be set after updates could fix this?
2	Repeat of test 1 with notes discussed as changes.	Old time should be different to current time.		Y
3	Check that multiple updates will occur between frames when lag is occurring	More than one update should occur.		Y Note: the cout takes place after the updates are done, not before.
4	Check that the fps does not go above 60 by using FRAPS	Shouldn't be above 60		N. Looking in to other solutions. So far SDL_Delay() seems like a good idea.
5	Repeat test 4 except with SDL_Delay() implemented	Should stay at around 60 FPS		Y. Note: FPS stayed roughly around 60 going slightly under/over

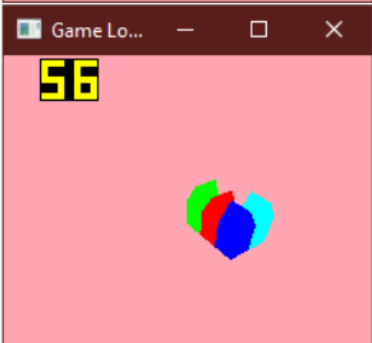
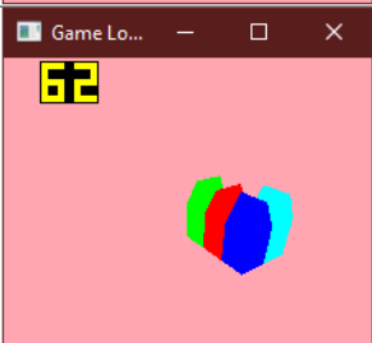
			Render performed. Fps has been delayed. Update performed. Old time is: 31403640 Current time is: 31403656 Elapsed time is: 16 Render performed. Fps has been delayed. Update performed.		for brief moments.
6	Test that multiple updates occur with artificial load by adding a for loop that does nothing inside the objects being updated.	Multiple updates per render should be performed.	Old time is: 31566953 Current time is: 31566968 Elapsed time is: 15 Render performed. Fps has been delayed. Update performed. Old time is: 31566968 Current time is: 31566984 Elapsed time is: 16 Render performed. Update performed. Update performed. Old time is: 31566984 Current time is: 31567015 Elapsed time is: 31 Render performed. Update performed. Update performed. Old time is: 31567015 Current time is: 31567046 Elapsed time is: 31		Y

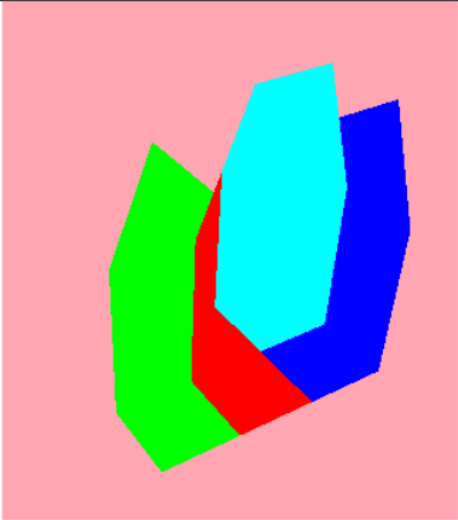
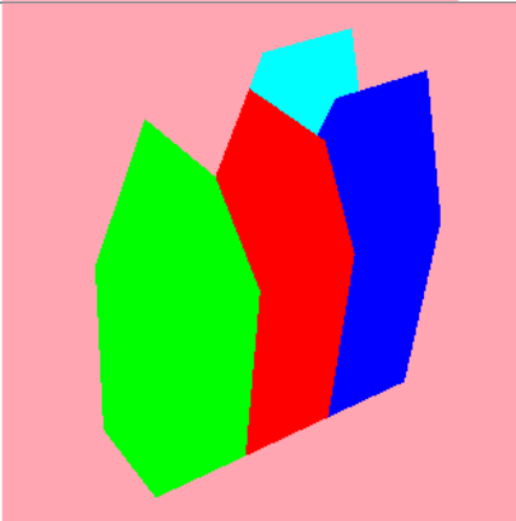
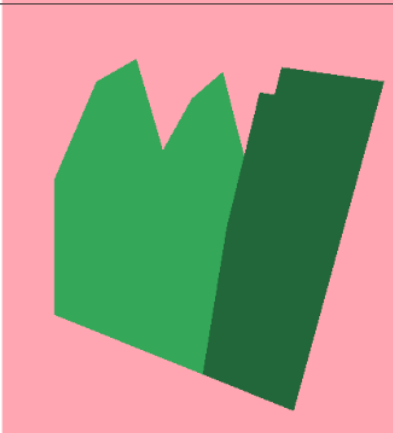
Input, OpenGL, and Camera tests

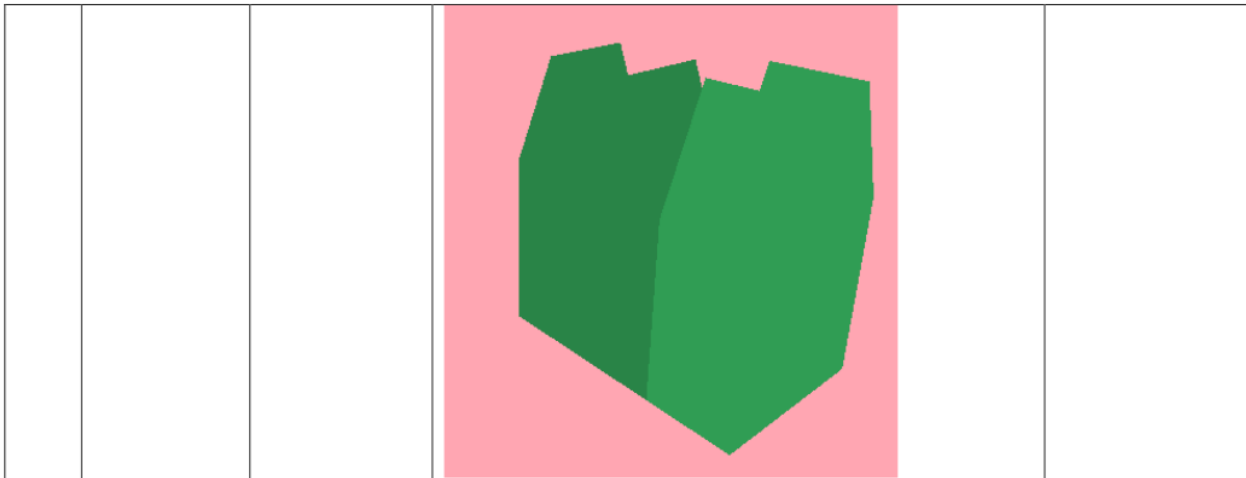
Controls for the camera are A, W, S and D. They change the eyeX, eyeY, and eyeZ values of gluLookAt.

Test No.	Description	Expected Result	Actual Result	Success Y/N? Other notes
1	Check that the camera moves when pressing A.	Camera should move when pressing A		Y Note: Sometimes the camera doesn't always update when pressing a key? Not sure why this is.

				
2	Same as test 1, except this time with the letter D.	Camera should move		Y

3	Same as previous, this time W.	Camera should move	 	
4	Same as previous except this time for S.	Camera should move.	 	Y

5	NOTE: This test was redone as originally it was not logged. Check for depth	Houses should have depth, which can be observed by moving the camera		N enable depth.
6	Previous test, though with depth enabled.	See previous test.		Y
7	Test lighting. Each house should look different	Each house should have a different colour as previously, with lighting effects on them.		N All houses were the same colour, however the lighting did work when changing the cameras position. This will be left disabled for now.



Other bugs discovered during testing:

For some reason, at times, the program will not close when pressing the X. Unsure what the current cause of this bug is, as its been noticed both with and without the delay added to the object class.

Object Testing

Before heading straight in to the tests, the objects in this were designed as dummies. They aren't actual objects with their own physics. In this case they are nothing more than an object with two values within them. One being an int for health, and an int for speed. In addition, these objects also have their own

Test No.	Description	Expected result	Actual result	Success Y/N? Other notes
1	Test that multiple updates occur with artificial load by adding a for loop that does nothing inside the objects being updated.	Multiple updates per render should be performed.	<pre> Old time is: 31566953 Current time is: 31566968 Elapsed time is: 15 Render performed. Fps has been delayed. Update performed. Old time is: 31566968 Current time is: 31566984 Elapsed time is: 16 Render performed. Update performed. Update performed. Old time is: 31566984 Current time is: 31567015 Elapsed time is: 31 Render performed. Update performed. Update performed. Old time is: 31567015 Current time is: 31567046 Elapsed time is: 31 </pre>	Y Yes, this is from the first test. This was just repeated here to show that if we take too long updating our objects, things need to catch up before rendering

				the next frame.
2	Check that objects values (health and speed) can be updated.	Health and speed should be a new number to what they were previously.	<pre>object number: 2 previous health: 77 previous speed: 24 updated object number: 2 has been updated! health: 81 speed: 9</pre>	Y
3	Check that if the object speed and health aren't randomized in objectUpdate that they should be 0, as stated in the constructor. (After test, the objectUpdate will function as normal again, these parts are only being commented out for this test)	Health and speed should be 0, and so should speed.	<pre>updated object number: 13 has been updated! health: 0 speed: 0 object number: 14 previous health: 0 previous speed: 0</pre>	Y Note: Cout's have been disabled after this test, as doing so many cout's per loop kept the time so far behind it was never possible to catch up, resulting in the game loop staying at 0 or 1 FPS due to such a big load.

Appendix B – Code

This appendix contains all of the code found within the project.

Source.cpp

```
#include "mesh.h"
#include "display.h"
#include "objects.h"
#include "glut.h"

#include <Windows.h>
#include <iostream>

int main(int argc, char *args[])
{
    Game gameLoopWindow(800, 800, "Game Loop SDL Window"); //create and open a window

    Vertex vertices[48];

    //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);
```

```

67 vertices[42].pos = glm::vec3(0.5, 1.25, 0.0); //right roof
68 vertices[43].pos = glm::vec3(1.0, 0.0, 0.0);
69 vertices[44].pos = glm::vec3(1.0, 0.0, -1.0);
70 vertices[45].pos = glm::vec3(0.5, 1.25, 0.0);
71 vertices[46].pos = glm::vec3(1.0, 0.0, -1.0);
72 vertices[47].pos = glm::vec3(0.5, 1.25, -1.0);
73
74 Mesh* pMesh = new Mesh(vertices, sizeof(vertices) / sizeof(vertices[0]));
75
76 float msForEachUpdate = 16.67; //This is the ms between each update.
77 float lag = 0.0; //This lag is introduced so that updates are done at a fixed rate.
78 long timerOldTime = GetTickCount64(); //keeps track of time for game loop.
79 long timerTimeElapsed; //keeps track of the time since the previous update.
80 long timerCurrentTime; //The current time.
81 long fpsStart; //timer for FPS rate
82 long fpsTime; //timer for FPS rate
83 while (!gameLoopWindow.isClosed()) // Game loop. As long as the SDL window is open, this loop will repeat over and over.
84 {
85     timerCurrentTime = GetTickCount64(); //sets the game loops time to the current time.
86     timerTimeElapsed = timerCurrentTime - timerOldTime; //keeps track of the time between updates, by taking the previous time from the current time.
87
88     lag = lag + timerTimeElapsed;
89     //Input begins here
90     //Registered inputs are A W S D.
91     //These will change the XYZ values for the Eye values of glLookAt, by changing the variable used in .update()
92     gameLoopWindow.input();
93
94     //This forces the game to update a certain number of times per each game loop. The point of this is to
95     while (lag >= msForEachUpdate)
96     {
97         //our "objects" update here. In this case these "objects" are just dummies, that have some variables set to a random number and an empty for loop in each.
98         gameLoopWindow.update();
99         lag = lag - msForEachUpdate;
100         std::cout << "Update performed." << std::endl;
101     }
102     std::cout << "Old time is: " << timerOldTime << " Current time is: " << timerCurrentTime << " Elapsed time is: " << timerTimeElapsed << std::endl;
103     gameLoopWindow.render(pMesh);
104     std::cout << "Render performed." << std::endl;
105     timerOldTime = timerCurrentTime;
106     //But you already have a delay set up above wrapped around the update!
107     //Yes, well, I used a program called FRAPS which has an FPS monitor and found that the actual FPS
108     //was hitting much higher than 60 in reality. It was reaching numbers up to 218- fps on this machine.
109     //This led me to looking in to SDL_Delay instead.
110     //this is used to cap the SDL window FPS to 60.
111     fpsTime = GetTickCount64() - timerCurrentTime;
112     if (msForEachUpdate > fpsTime)
113     {
114         //Delays by a set number of milliseconds. In this case, for any time that we are going above 60 FPS.
115         SDL_Delay(msForEachUpdate - fpsTime);
116         std::cout << "Fps has been delayed." << std::endl;
117     }
118 }
119
120 return 0;
121 }

```

Display.h

Assignment 2

```

1  #ifndef DISPLAY_H
2  #define DISPLAY_H
3  #include <SDL2/SDL.h>
4  #include <GL/glew.h>
5  #include <string>
6  #include "mesh.h"
7
8  //SDL2 object, contains various things needed for the game loop, and allows the use of OpenGL.
9  class Game
10 {
11 public:
12     Game(int width, int height, const char * title);
13     virtual ~Game();
14     void swapBuffers();
15     bool isClosed();
16     void input();
17     void update();
18     void clear(void);
19     void render(Mesh *pMesh);
20 private:
21     bool closed;
22     SDL_Window* window; //pointer to window
23     SDL_GLContext glContext; //context for drawing OpenGL to SDL window
24 };
25
26 #endif //DISPLAY_H

```

Display.cpp

```

1  #include <GL/glew.h>
2  #include <string>
3  #include <iostream>
4  #include <stdio.h>
5  #include <iio.h>
6  #include "glut.h"
7  #include "display.h"
8  #include "objects.h"
9
10 using namespace std;
11
12 float moveCameraX = 5.0;
13 float moveCameraY = 5.0;
14 float moveCameraZ = 5.0;
15
16 //just creates 30 game object dummies.
17 objectsForGame gameObj[30];
18 Game::Game(int width,int height, const char* title)
19 {
20     SDL_Init(SDL_INIT_EVERYTHING);//uses SDL2 to create
21
22     SDL_GL_SetAttribute(SDL_GL_RED_SIZE, 8);
23     SDL_GL_SetAttribute(SDL_GL_GREEN_SIZE, 8);
24     SDL_GL_SetAttribute(SDL_GL_BLUE_SIZE, 8);
25     SDL_GL_SetAttribute(SDL_GL_ALPHA_SIZE, 8);
26     SDL_GL_SetAttribute(SDL_GL_BUFFER_SIZE, 32);
27     SDL_GL_SetAttribute(SDL_GL_DOUBLEBUFFER, 1);
28
29     window = SDL_CreateWindow(title, 100,100, width, height, SDL_WINDOW_OPENGL | SDL_WINDOW_RESIZABLE );
30
31     glContext = SDL_GL_CreateContext(window);
32     SDL_GL_SetSwapInterval(0);
33     GLenum status = glewInit();
34     if (status != GLEW_OK)
35     {
36         cerr << "Glew failed to initialize" << endl;
37     }
38     closed = false;
39     //Turned on depth, so the houses can look like they are actually 3D when moving the camera.
40     glutInitDisplayMode(GLUT_DEPTH);
41     glEnable(GL_DEPTH_TEST);
42     //Left over from trying to do lighting. Lighting disables the colouring of the houses, making them all the same colour from lighting.
43     GLfloat mat_specular[] = { 0.333333, 0.100000, 0.000000, 1.0 };
44     GLfloat mat_diffuse[] = { 0.7000, 0.2000, 0.8992, 1.0 };
45     GLfloat mat_ambient[] = { 0.2000, 0.8000, 0.4000, 1.0 };
46     GLfloat mat_shininess[] = { 12.8 };
47     glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
48     glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);
49     glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
50     glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_ambient);
51     GLfloat light_position[] = { 1.0, 1.0, 1.0, 0.0 };
52
53     glShadeModel(GL_FLAT); //uses GL_SMOOTH or GL_FLAT
54     /*glEnable(GL_LIGHTING);
55     glEnable(GL_LIGHT0);*/
56
57     //set viewport to screen size and initialize projection matrix. Leave with matrixmode = Modelview
58     glViewport(0, 0, width, height); //make sure openGL view port is scaled to size of new window
59     glMatrixMode(GL_PROJECTION);
60     glLoadIdentity();
61     glFrustum(-1.0, 1.0, -1.0, 1.0, 1.5, 20.0);
62     // 1,r,bot,top,near,far
63     glMatrixMode(GL_MODELVIEW);
64
65 }

```

```

66 //Game deconstructor
67 Game::~Game()
68 {
69     SDL_GL_DeleteContext(glContext);
70     SDL_DestroyWindow(window);
71     SDL_Quit();
72 }
73
74 void Game::input()
75 {
76     SDL_Event e;
77     while (SDL_PollEvent(&e))
78     {
79         //checks if the user is pressing either A, W, S, or D to change the camera's eye values.
80         if(e.type == SDL_KEYDOWN)
81             switch (e.key.keysym.sym)
82             {
83                 case SDLK_a:
84                     moveCameraX = moveCameraX - 1;
85                     break;
86                 case SDLK_d:
87                     moveCameraX = moveCameraX + 1;
88                     break;
89                 case SDLK_w:
90                     moveCameraY = moveCameraY + 1;
91                     break;
92                 case SDLK_s:
93                     moveCameraY = moveCameraY - 1;
94                     break;
95             }
96     }
97 }
98
99 void Game::update()
100 {
101     //this is where our games objects (and SDL), updates.
102     for (int i = 0; i < 30; i++)
103     {
104         //std::cout << std::endl << "object number: " << i;*/
105         gameObjs[i].tellPreviousValue();
106         //std::cout << std::endl << "updated object number: " << i;*/
107         gameObjs[i].objectUpdate();
108     }
109     SDL_Event e;
110     while (SDL_PollEvent(&e))
111     {
112         //cout << "event number:" << e.type << endl;
113         if (e.type == SDL_QUIT)
114             closed = true;
115
116         if (e.window.event == SDL_WINDOWEVENT_RESIZED)
117         {
118             glViewport(0, 0, e.window.data1,e.window.data2);
119             cout << "resized to " << e.window.data1 << " , " << e.window.data2<<endl;
120             glMatrixMode(GL_PROJECTION);
121             glLoadIdentity();
122             glFrustum(-1.0, 1.0, -1.0, 1.0, 1.5, 20.0);
123             //      1,r,bot,top,near,far
124             glMatrixMode(GL_MODELVIEW);
125         }
126     }
127 }
128
129 }
130
131

```

```

131
132 void Game::swapBuffers(void)
133 {
134     SDL_GL_SwapWindow(window);
135 }
136 void Game::render(Mesh* pMesh)
137 {
138
139     clear();// clear frame buffer before redrawing.
140     glLoadIdentity();
141
142     gluLookAt(moveCameraX, moveCameraY, moveCameraZ, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
143     // X,Y,Z depending on variables. Starts off as 5, 5, 5 - place camera ; 0,0,0 - aim lens towards 0,0,0;
144     // 0,1,0 - the up vector defines an unique orientation
145     glColor3f(0.0, 1.0, 0.0);
146     pMesh->draw();
147     glTranslatef(1.0, 0.0, 0.0);
148     glColor3f(1.0, 0.0, 0.0);
149     pMesh->draw();
150
151     glTranslatef(2.0, 0.0, 0.0);
152     glRotatef(90.0, 0.0, 1.0, 0.0);
153     glColor3f(0.0, 0.0, 1.0);
154     pMesh->draw();
155
156     glRotatef(0, 6, 0, 0);
157     glTranslatef(1, 0, 0);
158     glColor3f(0.0, 1.0, 1.0);
159     pMesh->draw();
160     swapBuffers();
161 }
162
163 bool Game::isClosed()
164 {
165     return closed;
166 }
167
168 void Game::clear(void)
169 {
170     glClearColor(1.0f, 0.65f, 0.7f, 1.0f);
171     glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
172 }

```

Objects.h

```

1  #pragma once
2  #include "Display.h"
3  class objectsForGame {
4      // Just a dummy object.
5  public:
6      //constructor with default values. has health and speed set to 0.
7      objectsForGame(void);
8
9      //performs and "Update" on the object.
10     void objectUpdate();
11     //tells values before the update.
12     void tellPreviousValue();
13 private:
14
15     int objectHealth;
16     int objectSpeed;
17 };

```

Objects.cpp

The couts here were used purely for testing.

```
1 #include "objects.h"
2 #include <iostream>
3
4 void objectsForGame::objectsForGame(void)
5 {
6     // sets the default values for our object, in this case, the health and speed.
7     objectHealth = 0;
8     objectSpeed = 0;
9 }
10 void objectsForGame::objectUpdate()
11 {
12     for (int i = 0; i < 150000; i++)
13     {
14         //delay here in order to demonstrate that multiple updates can occur before a render with this game loop.
15     }
16     //sets object health to a random number between 0 to 101.
17     objectHealth = rand()%101;
18     //sets object speed to a random number between 0 and 31.
19     objectSpeed = rand()%31;
20     //std::cout << " has been updated! health: " << objectHealth << " speed: " << objectSpeed << std::endl;
21 }
22 void objectsForGame::tellPreviousValue()
23 {
24     //just tells previous values based on when its called.
25     /*std::cout << " previous health: " << objectHealth << " previous speed: " << objectSpeed << std::endl;*/
26 }
27
28
29
```

Mesh.h

```
1 #ifndef MESH_H
2 #define MESH_H
3
4 #include <glm/glm.hpp>
5 #include <gl/glew.h>
6 #include <SOIL/SOIL.h>
7 #define NUM_BUFFERS 1
8 typedef struct {
9     glm::vec3 pos;
10 }Vertex;
11
12 class Mesh
13 {
14 public:
15     Mesh(Vertex* vertices, unsigned int numVertices);
16     int loadOBJ();
17
18     void draw();
19     virtual ~Mesh();
20 private:
21     Vertex *pVertices;
22     GLuint vertexArrayObject;
23     GLuint vertexArrayBuffers[NUM_BUFFERS];
24     GLuint vertexCount;
25
26 };
27
28 #endif
```

Mesh.cpp

```
1  #include "mesh.h"
2
3
4  Mesh::Mesh(Vertex* vertices, unsigned int numVertices)
5  {
6      vertexCount = numVertices;
7      pVertices = vertices;
8  }
9
10
11 Mesh::~Mesh()
12 {
13 }
14
15 void Mesh::draw()
16 {
17     for (int i = 0; i < vertexCount; i+=3)
18     {
19         glBegin(GL_TRIANGLES);
20
21         glVertex3f(pVertices[i].pos.x, pVertices[i].pos.y, pVertices[i].pos.z);
22
23         glVertex3f(pVertices[i+1].pos.x, pVertices[i+1].pos.y, pVertices[i+1].pos.z);
24
25         glVertex3f(pVertices[i+2].pos.x, pVertices[i+2].pos.y, pVertices[i+2].pos.z);
26
27         glEnd();
28     }
29 }
30
31
32
```

Transform.h

```
1  #ifndef TRANSFORM_H
2  #define TRANSFORM_H
3  #include <glm/glm.hpp>
4  #include <GL/glew.h>
5  #include <string>
6
7
8  class Transform
9  {
10 public:
11     Transform();
12     Transform(glm::mat4x4 m);
13     virtual ~Transform();
14
15     void UpdateMV(glm::mat4x4 m);
16
17 private:
18     glm::mat4x4 mv;
19 };
20
21
22
23 #endif
```

Transform.cpp

Assignment 2

```
1  #include "transform.h"
2  Transform::Transform()
3  {
4  }
5  }
6  Transform::Transform(glm::mat4x4 m)
7  {
8  }
9  }
10 Transform::~~Transform()
11 {
12 }
13 }
14
15 void UpdateMV(glm::mat4x4 m)
16 {
17 }
18 }
19 }
```

Games Architecture and Mechanics Part 1b

GRADEMARK REPORT

FINAL GRADE

GENERAL COMMENTS

Instructor

74/100

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

GRADING FORM: SMU GENERIC_GJ

MAY DAVIES**71****RESEARCH-INFORMED LITERATURE**

Do review what make a good abstract - see comments in previous assignment. Refs good and used appropriately.

7**KNOWLEDGE AND UNDERSTANDING**

Essential ideas identified and discussed. Relative adv/disad of loop patterns discussed. More could be said about container objects and updates.

15**ANALYSIS**

Arguments for the chosen loop provided. Test shedule design reflects requirements here. Well documented. Updating documented via screenshots. Good

14**PRACTICAL APPLICATION AND DEP.**



Some good work here. Obj loader attempted - though not necessary - good attitude here. Well done.

35

SKILLS FOR PROFESSIONAL PRAC.



Some good SE principles demonstrated, Testing documented. REfs appropriate. Report Structured. Well done!

3,5