COMP 3069 – Computer Graphics
Assignment 2

Varun Sangwan

December 2019

Contents

Chapter 1

Question 2

1.1 Square Fractals

Given a initial square, subdivide each side of square such that a square of side one third of original square. The approach is to store each vertex in order and render them in sequence. There are two structs used Vertex that stores x,y coordinates of a vertex and another struct Edges that contains two consecutive vertex coordinates and orientation which basically tells the orientation of the edge, this will be useful to calculate new edges new small squares form outside of original edge.

```
// Vertex and Edges.
struct Vertex {
    glm::vec2 v;
};

struct Edges {
    glm::vec2  first ;
    glm::vec2 second;
    char orientation;
};
```

The class Pol contains several methods and variables to store the new vertices whenever a object is creates, later whenever the current polygon is subdivided a new Pol object will be formed that will store all edges in sequence.

- Data Types
  - **std::vector** $< Edges >$ **edge -** this vector will store all the edges in sequence. As mentioned above an edge contains two vertices and orientation.

- **std::vector** $< Vertex >$ **vertex;** - this vector stores all the vertices in current polygon in sequence
- **float len** - it stores the side length of current polygon.

- Constructors
  - **Default Constructors Pol()** - it will construct a 2D square and store all call the method construct that will calculate edges and store them,
  - **Parameterized Constructors Pol(std::vector**$< Vertex >$ **v, float l)** - it will construct polygon based on the vector v passed that should contain all the vertices in order, it also calls construct that will calculate edges and store them. It also stores the length of square side.
- Methods
  - **construct()** - this method will fill the edge vector using the vertex vector passed to constructor
  - **findorient()** - this method will find the orientation of each edge using 2 vertices and coordinate geometry. Orientation is necessary to find new edges while subdividing the square. it will return a char which represents the edge orientation. Since all the edges can have only four orientations,
    - 'a' represents right edge of square .
    - 'b' represents bottom edge of square .
    - 'c' represents left edge of square .
    - 'd' represents top edge of square .

```
//Pol definations
Pol::Pol() {
    Vertex ver[4];
    ver[0].v = { 1,1 };
    vertex.push_back(ver[0]);
    ver[1].v = { 1, -1 };
    vertex.push_back(ver[1]);
    ver[2].v = { -1, -1 };
    vertex.push_back(ver[2]);
    ver[3].v = { -1, 1 };
    vertex.push_back(ver[3]);
    len = 2.0;
    std::cout << vertex.size() << std::endl;
    construct();
}
```

```
Pol::Pol(std::vector<Vertex> v, float l) {
    vertex = v;
    len = l;
    construct();
    std::cout << edge.size() << " " << vertex.size() << std::endl;
}

void Pol::construct() {
    std::vector<Vertex>::iterator i;
    for (i = vertex.begin(); i != vertex.end() - 1; ++i)
    {
        Vertex first = *i;
        Vertex second = *(i + 1);
        Edges temp;
        temp.first = first.v;
        temp.second = second.v;
        temp.orientation = find_orient(first.v, second.v);
        edge.push_back(temp);
    }
    Edges temp;
    temp.first = (vertex.back()).v;
    temp.second = (vertex.front()).v;
    temp.orientation = find_orient((vertex.back()).v, (vertex.front()).v);
    edge.push_back(temp);
}

char Pol::find_orient(glm::vec2 a, glm::vec2 b) {
    if (a.x == b.x && a.y > b.y) return 'a';
    else if (a.y == b.y && a.x > b.x) return 'b';
    else if (a.x == b.x && a.y < b.y) return 'c';
    else return 'd';
}
}
```

1.1.1 Source.cpp

There are three global variables a vector listofpols that contains Pol objects hence the ploygons, curr is a Pol object that refers to the current polygon that is being drawn on screen. curpointer is a integer that keeps track of index of Pol object in listofpols and is used to iterate back and fourth over listofpols.

When user presses the key '+' divide() function is called which makes a new polygon based on current polygon edges and orientation of each edge and adds the Polygon to listofpols as well as making the curr point to new polygon.

When user presses the key '-' back() function is called which simply moves the curr pointer one index back in listofpols.

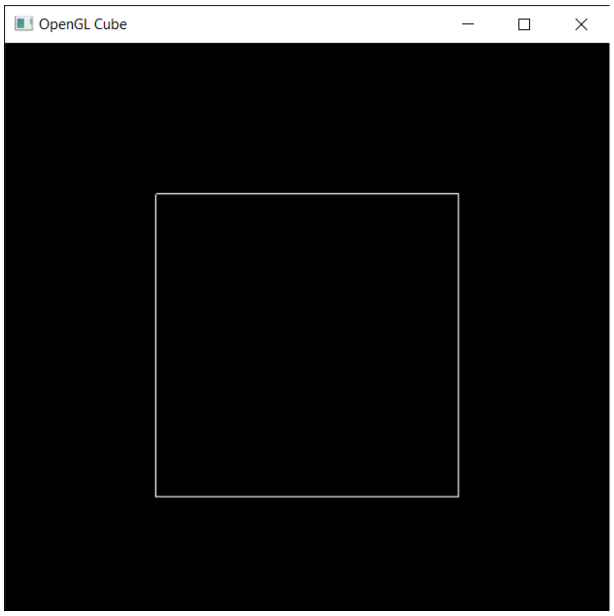
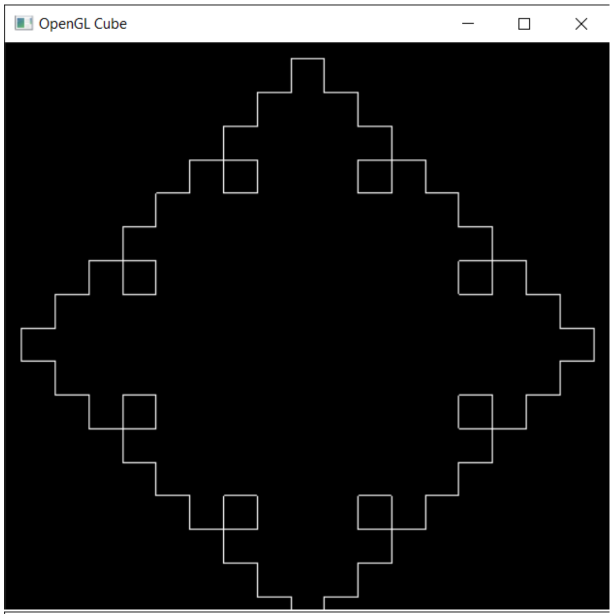


Figure 1.1: Square



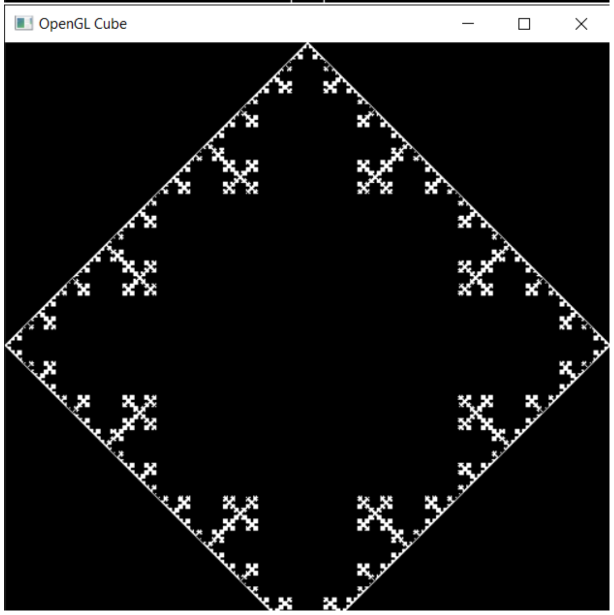


Figure 1.2: Making fractals

1.2 Ball free fall Animation

Question 3

Problem Statement

"In this question, you are required to produce an animation as follows. There is a ball with a specific texture on its surface (each student needs to specify a different texture here) in the space. The ball moves forward horizontally around a square with constant speed. (We remark here that at the corner position, the ball needs to change its moving direction according to the coming edge of the square whereas its speed remains the same.) In the vertical direction, it falls naturally according to the gravity of earth to reach the floor and bounce back to the original height periodically. Particularly, in your scene, the floor needs to be decorated with a texture on its surface (each student needs to specify a different texture here). Further, you are also required to use "+" button to increase the horizontal velocity of the ball, and use "-" button to decrease the horizontal velocity of the ball (note that for obtaining full credits, the velocity may decreased to negative values, which means that the ball changes its moving direction)."

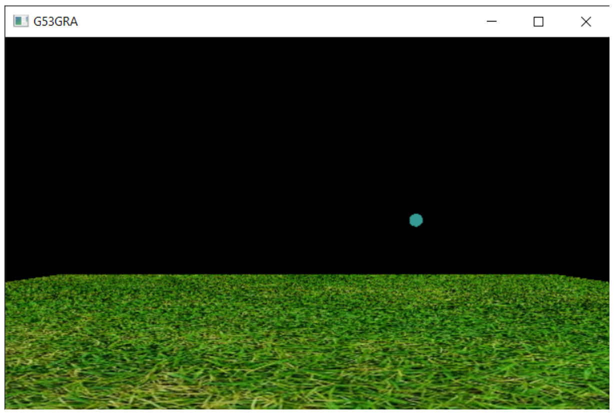


Figure 1.3: Ball Animation

Approach

For this question I used the G53GRA framework provided that comes with various classes and methods as the starter code. The floor is textured with grass image. I have implemented newton equations of motions to mimic the natural free fall under gravity. To animate the ball by overloading Update function the y coordinate of the ball is changed using the change in time between each function fall using below parameters.

```
t = (float)deltaTime;
    //std::cout << deltaTime << std::endl;
    if(sw){
        v = u + g * t;
        ypos -= u * t + (g * t * t)/2;
        //std::cout << ypos << std::endl;
        if (ypos <= -0.999f) {
            sw = false;

        }
    }
    else {
        v = u - g * t;
        ypos += u * t + (g * t * t)/2;
        if (ypos >= 0.999999f) {
            sw = true;
        }
    }
    u = v;
```

Where,

- v = final velocity in m/s.
- u = initial velocity in m/s.
- deltaTime or t = change in time between function calls in seconds.
- ypos = y coordinate of ball or height.
- sw = A boot that acts as switch to change the ball direction at heighest and lowest point.
- g = Acceleration due to gravity on earth i.e 9.8 m/s2

Moving the ball around a square

To make the ball move around the square the x coordinate and z coordinate position is altered in a systematic manner. There is a integer that represents the side of square because the ball is initially placed at x = -1.0f and z = 1.0f the horizontal velocity is changed based on which side of cube ball is currently.

```

    if (ves>0) {
        if (s == 0||s==4) {
            xpos += ves;
            if (xpos >= 1.0f) s = 1;
        }
        else if (s == 1) {
            zpos −= ves;
            if (zpos <= −1.0f) s = 2;
        }
        else if (s == 2) {
            xpos −= ves;
            if (xpos <= −1.0f) s = 3;
        }
        else {
            zpos += ves;
            if (zpos >= 1.0f) s = 4;
        }
    }
    if (ves <= 0) {
        if (s == 4) {
            xpos += ves;
            if (xpos <= −1.0f) s = 3;
        }
        else if (s == 3) {
            zpos += ves;
            if (zpos <= −1.0f) s = 2;
        }
        else if (s == 2) {
            xpos −= ves;
            if (xpos >= 1.0f) s = 1;
        }
        else {
            zpos −= ves;
            if (zpos >= 1.0f) s = 4;
        }
    }
```

Where,

- ves = Velocity that can be increased or decreased by pressing + or -.
- zpos = z coordinate or the ball at any given time and it is altered depending upon the velocity and the current side of square.
- xpos = x coordinate or the ball at any given time and it is altered depending upon the velocity and the current side of square.
- s = It represents the side of square.
  - if s=0 or s=4 that represents the front side of square on floor.
  - if s=1 that represents the right side of square on floor.

  - if s=2 that represents the back side of square on floor.
  - if s=3 that represents the left side of square on floor.

The floor

The floor is made using four coordinates and translated along -ve y axis. Grass texture is applied to the floor.

1.2.1 The ball

Sphere
To make a sphere, the vertices are calculated by projecting the radius on all the axes and because sphere is 3d the radius makes angles with x-z plane and x-y plane. by varying the both the angles in a symmetrical manner and drawing.

The ball is textured with image that can be found "./Textures/liv.bmp". Though only part of image is used to apply the textures.