Assignment #3 – Star rotation

Yoke Kai Wen, 2020280598

April 20, 2021

1 Introduction

In this assignment, I created an animation of star particles rotating and spreading out in three types of spiral paths, with each star coloured differently according to their angular position. Figure 1 shows screenshots of the three types of spiral paths.

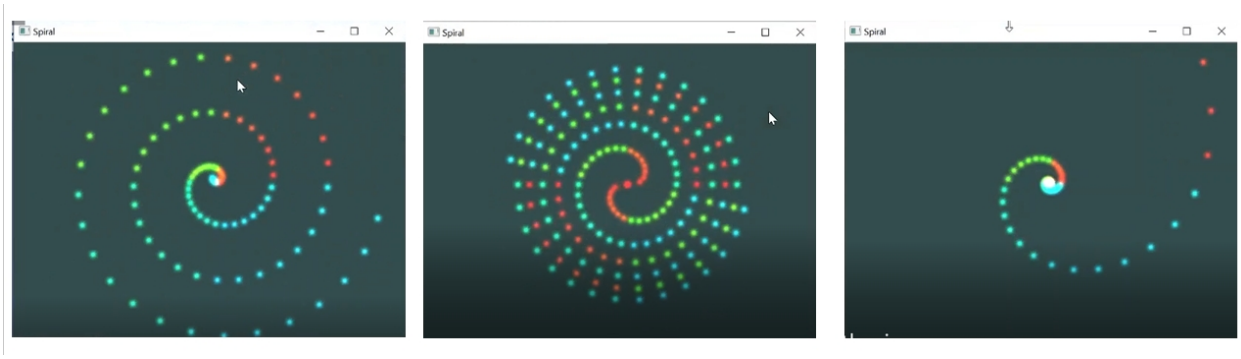

Figure 1: Screenshots of three types of spirals:Archimedes, Fermat, Logarithmic

2 Files and directory structure

1. Main script: main.cpp
2. Header file for star spiral: SpiralStars.h (in include/tut_headers/ folder)
3. Shader files: main.vert.glsl, main.frag.glsl
4. Star texture: Star.bmp
5. Other header files: include/tut_headers/shader.h
6. Set include/ and lib/ folders as include and library directories respectively in VS.

3 Usage

In `main.cpp`, the user can set parameters to determine type of spiral path, number of stars in each spiral path, and other parameters as seen in code fragment below. After setting the parameters, the animation of the star spirals will be displayed continuously.

```
int starNum = 60;
glm::vec3 centre = glm::vec3(WIDTH / 2, HEIGHT / 2, 0);
double a = 0.5; // spiral param: 3.0 for archi, 7.0 for fermat, 0.5 for log (
    determines extent of spreading out)
double b = 0.4; // spiral param for logarithmic spiral
int type = 2; // 0 for archi, 1 for fermat, 2 for log
int numSpirals = 10;

spiral = new BunchofStars(starNum, centre, a , b, type, numSpirals);
```

4 Design

4.1 Generating spirals

4.1.1 Spiral equation

The distance of each particle from the centre r in the spiral path (i.e. the radius) is calculated based on its equation, with the angular position θ and other constant parameters a, b as input.

$$\text{Archimedes spiral: } r = a\theta$$
$$\text{Fermat spiral: } r = \pm a\sqrt{\theta}$$
$$\text{Logarithmic spiral: } r = ae^{b\theta}$$

The position of the particle can then be computed using $x = rcos\theta$ and $y = rsin\theta$ as seen in the code fragment below.

```
//calc position according to spiral formula
void calc_pos()
{
  GLfloat theta_rad = glm::radians(GLfloat(theta));
  //archimedes spiral
  if (type == 0) {
    r = a * theta / 180.0 * PI;

  }
  //fermat spiral
  else if (type == 1) {
    r = a * sqrt(theta / 180.0 * PI);
  }
  //logarithmic spiral
  else if (type == 2) {
    r = a * exp(b * theta / 180.0 * PI);
  }
  position = glm::vec3(GLfloat(r) * glm::cos(theta_rad), GLfloat(r) * glm::sin(
    theta_rad), 0) + centre;
}
```

For the Fermat spiral, for each particle rendered, another counterpart particle with radius negative of that has to be rendered in accordance to the spiral equation. This is executed in the game loop of the `main.cpp` code as seen below.

```
 for (int i = 0; i < numSpirals*spiral->getStarNum(); ++i) {
    GLint offsetLoc = glGetUniformLocation(particleShader.Program, "offset");
    glUniform2f(offsetLoc, spiral->getStar(i)->position[0], spiral->getStar(i)->
    position[1]);
    //... setting uniforms and binding vertices and textures ...//
```

```
    glDrawArrays(GL_TRIANGLES, 0, 6);
    glBindVertexArray(0);

    if (type == 1) {
        //get negative radius for fermat spiral
        glUniform2f(offsetLoc, WIDTH - spiral->getStar(i)->position[0], HEIGHT -
    spiral->getStar(i)->position[1]);
        //... setting uniforms and binding vertices and textures ...//
        glDrawArrays(GL_TRIANGLES, 0, 6);
        glBindVertexArray(0);
    }
}
```

4.1.2 Setting angular positions θ for each particle in spiral

To create each spiral path, we assign the `starNum` = 60 particles an angle θ from `mintheta` = -180deg to `maxtheta` = 1080deg in a uniform distribution. This creates a spiral path with 3.5 circular loops which gives a nicer visual effect as compared to using an angular range of 0 to 360 deg, and enhances the spiralling outward effect.

```
mintheta = -180.0;
maxtheta = 1080.0;

for (int i = 0; i < starNum; ++i)
{
  theta = i * (maxtheta-mintheta) / starNum + mintheta; //assign angles for each
    particle
}
```

4.2 Creating effect of spiral rotating and spreading out

4.2.1 10 spirals of varying radius

To create the effect of the spiral rotating and spreading out, at each instant, I have `numSpirals`= 10 spirals with different radius as determined by different a parameter. Each spiral is associated with a different initial `life` parameter from {6, 7, 8, 9, 10, 11, 12, 13, 14, 15}. The code fragment below shows how the ten spirals are initialised.

```
for (int j = 0; j < numSpirals; ++j) {
  a_new = (j + 1.0) * a; //set a params for expanding spirals
  life = 6.0 + j; //set life params for expanding spirals
  for (int i = 0; i < starNum; ++i)
  {
    theta = i * (maxtheta-mintheta) / starNum + mintheta; //assign angles for each
     particle
    stars[j*starNum + i] = new Star(centre, theta, a_new, b, type, life);
  }
}
```

4.2.2 Controlling spiral transparency based on `life` parameter

When the life of the spiral is large (> 5), the alpha parameter of the spiral is zero (i.e. spiral is invisible). Every 0.1s, the `life` of each spiral decreases by 1, and when the life of a spiral equals 5, the alpha parameter is set to 1 and the spiral becomes visible, otherwise alpha is zero. Then when the life of a spiral goes below zero, the `life` parameter is reset to 10. This causes each

spiral to light up one by one, from the smallest radius to the largest radius, hence creating the spreading out effect.

```
void update(double dt)
{
  num_sec += dt;
  if (num_sec > 0.1) { //change display every 0.1s
    life -= 1;
    num_sec = 0;
  }

  // set visibility of points based on life remaining
  if (life > 5) {
    color.a = 0.0;
  }
  else if (life == 5) {
    color.a = 1.0;
  }
  else if (life <= 0) {
    life = 10;
  }
  else {
    color.a = 0.0;
  }
}
```

4.3 Setting star colours and texture

The colour of each star particle is determined by its θ angle (restricted to 0-360), and converted to an RGB vector in the code fragment below. The colour is then combined with the 'Star.bmp' texture to render stars of different colours.

```
//0 <= scalar <= 1
glm::vec4 scalar2col(double scalar)
{
  glm::vec4 out_col;

  if (scalar < 0.25) {
    out_col = glm::vec4(1.0, GLfloat(scalar), 0.0, 1.0);
  }
  else if (scalar < 0.5) {
    out_col = glm::vec4(GLfloat(scalar), 1.0, 0.0, 1.0);
  }
  else if (scalar < 0.75) {
    out_col = glm::vec4(0.0, 1.0, GLfloat(scalar), 1.0);
  }
  else {
    out_col = glm::vec4(0.0, GLfloat(scalar), 1.0, 1.0);
  }

  return out_col;
}
```