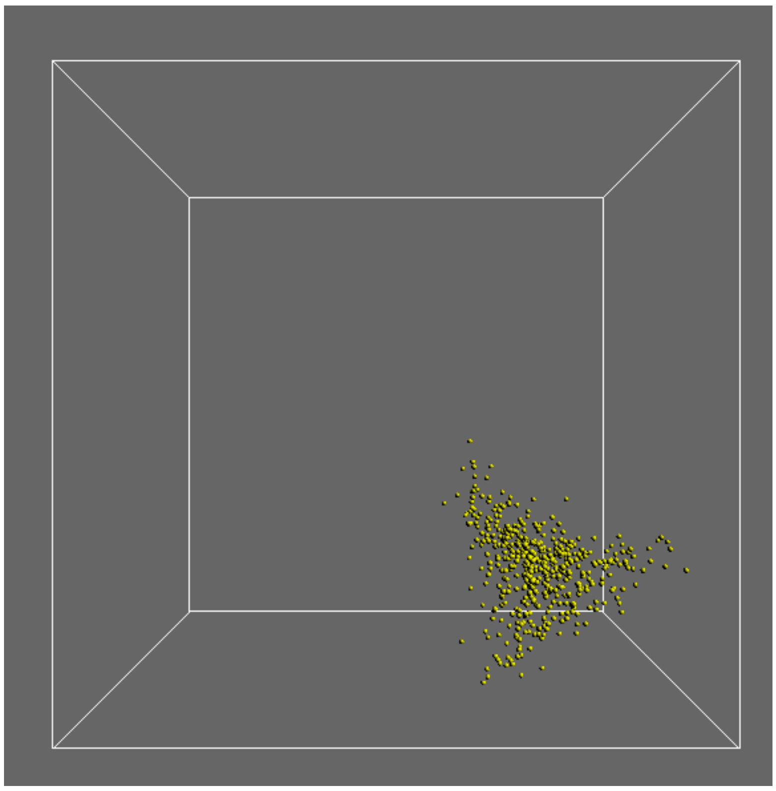
Computing for Animation 1
Flocking System



BA Computer Visualisation and Animation
Sihyeon Gu
(s4927639)

Introduction
The main goal of the project is to build a simple flocking simulation system referred to Reynolds's theory. Basically, the simulation system is able to check the change of the boid through Reynold's rule (separation, alignment and cohesion). In addition, as I change a basic option of a boid, I can observe the difference through the programme.

Implementation
Before I start to make a programme, first of all, It is an essential to understand the properties of a basic element which effects on the movement of a boid. Through Reynold's theory, I divided the step to create a movement.

1. <u>Boid</u>
   First of all, I have a moving boid with a cerntain location and velocity in order to play the flocking system. In addition, since I have to set the simulation with confined space, I need to create confined box (bounding box). I referred to the code what we learned on lecture for this basic step.

2. <u>Steer Behaviour</u>
   To complete flocking system, I need to set each boids with expected movement. There are three rules to control each boids.

   - Separation: steer behaviour to avoid crowded boids.
   - Alignment: steer behavior towards the average heading point.
   - Cohesion: steer behavior to move towoard the average position.

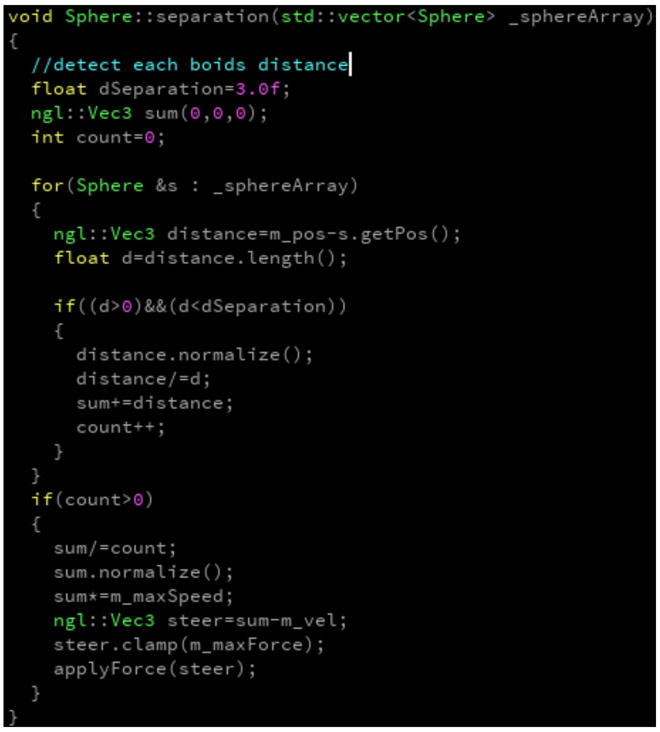
```
Data: A group of boids.
Result: Simulates flocking behaviour with an animation.
foreach Frame do
    foreach boid do
        separation(boid);
        cohesion(boid);
        alignment(boid);
    end
    foreach boid do
        boid.x ← $cos(boid.course) * b.velocity * dTime$;
        boid.y ← $sin(boid.course) * b.velocity * dTime$;
        draw(boid);
    end
end
```

An overview of the boids algorithm

1) Separation

```
Data: A boid.
Result: The course of the boid is updated.
goal ← (0,0);
neighbours ← getNeighbours(boid);
foreach nBoid in neighbours do
    goal ← goal + positionOf(boid) - positionOf(nBoid);
end
goal ← goal / neighbours.size();
steerThoward(goal, boid);
```

In flocking system, boids must avoid collisions between others. This rule makes boids away as much as possible. For each nearby boids, a repulsive fore is computed by subtracting the postion of each boids.

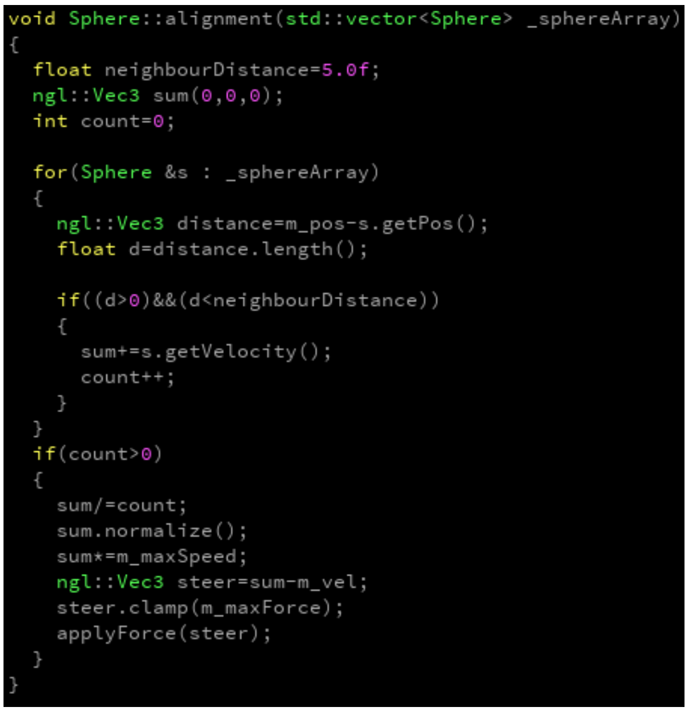


```
void Sphere::separation(std::vector<Sphere> _sphereArray)
{
  //detect each boids distance
  float dSeparation=3.0f;
  ngl::Vec3 sum(0,0,0);
  int count=0;

  for(Sphere &s : _sphereArray)
  {
    ngl::Vec3 distance=m_pos-s.getPos();
    float d=distance.length();

    if((d>0)&&(d<dSeparation))
    {
      distance.normalize();
      distance/=d;
      sum+=distance;
      count++;
    }
  }
  if(count>0)
  {
    sum/=count;
    sum.normalize();
    sum*=m_maxSpeed;
    ngl::Vec3 steer=sum-m_vel;
    steer.clamp(m_maxForce);
    applyForce(steer);
  }
}
```

2) Alignment

```
Data: A boid.
Result: The course and velocity of the boid is updated.
dCourse ← 0;
dVelocity ← 0;
neighbours ← getNeighbours(boid);
foreach nBoid in neighbours do
    dCourse ← dCourse + getCourse(nBoid) - getCourse(boid);
    dVelocity ← dVelocity + getVelocity(nBoid) - getVelocity(boid);
end
dCourse ← dCourse / neighbours.size();
dVelocity ← dVelocity / neighbours.size();
boid.addCourse(dCourse);
boid.addVelocity(dVelocity);
```

For calculating alignment, I need to get the average velocity because velocity have both direction and force. Therefore, alignment can be increased if velocity is increased.

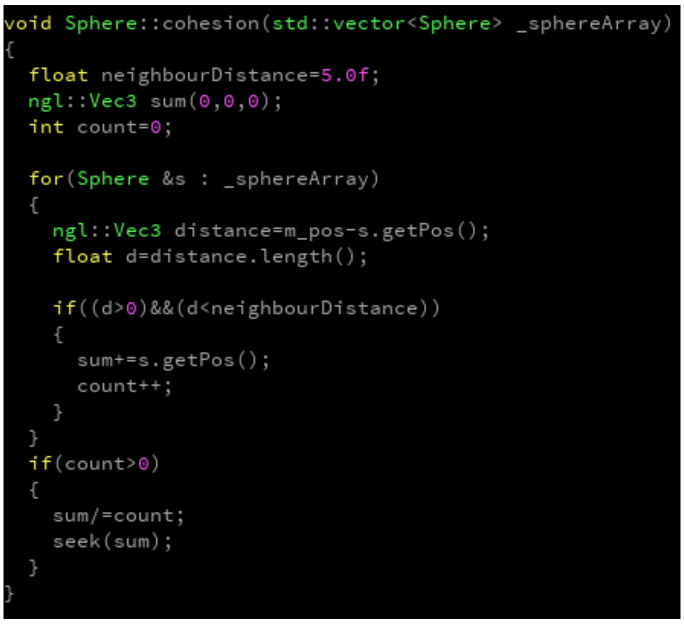


```
void Sphere::alignment(std::vector<Sphere> _sphereArray)
{
  float neighbourDistance=5.0f;
  ngl::Vec3 sum(0,0,0);
  int count=0;

  for(Sphere &s : _sphereArray)
  {
    ngl::Vec3 distance=m_pos-s.getPos();
    float d=distance.length();

    if((d>0)&&(d<neighbourDistance))
    {
      sum+=s.getVelocity();
      count++;
    }
  }
  if(count>0)
  {
    sum/=count;
    sum.normalize();
    sum*=m_maxSpeed;
    ngl::Vec3 steer=sum-m_vel;
    steer.clamp(m_maxForce);
    applyForce(steer);
  }
}
```

3) Cohesion

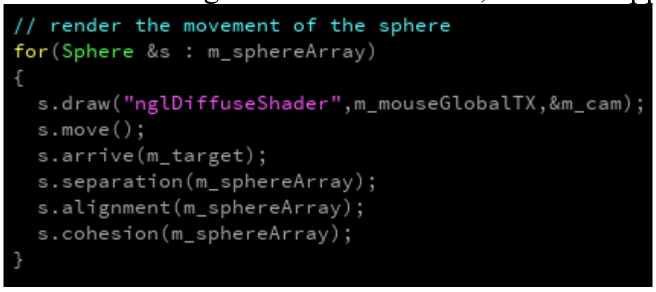
```
Data: A boid.
Result: The course of the boid is updated.
goal ← (0,0);
neighbours ← getNeighbours(boid);
foreach nBoid in neighbours do
    goal ← goal + positionOf(nBoid);
end
goal ← goal / neighbours.size();
steerThoward(goal, boid);
```

Cohesion is steer to keep boids together, I need to find the average position. It basically similar to calculate alignment, but I need to put position instead of velocity.



```
void Sphere::cohesion(std::vector<Sphere> _sphereArray)
{
  float neighbourDistance=5.0f;
  ngl::Vec3 sum(0,0,0);
  int count=0;

  for(Sphere &s : _sphereArray)
  {
    ngl::Vec3 distance=m_pos-s.getPos();
    float d=distance.length();

    if((d>0)&&(d<neighbourDistance))
    {
      sum+=s.getPos();
      count++;
    }
  }
  if(count>0)
  {
    sum/=count;
    seek(sum);
  }
}
```

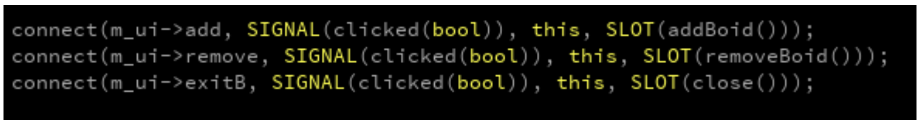
After calculating each rules for boids, I need to apply all rules on same boids.



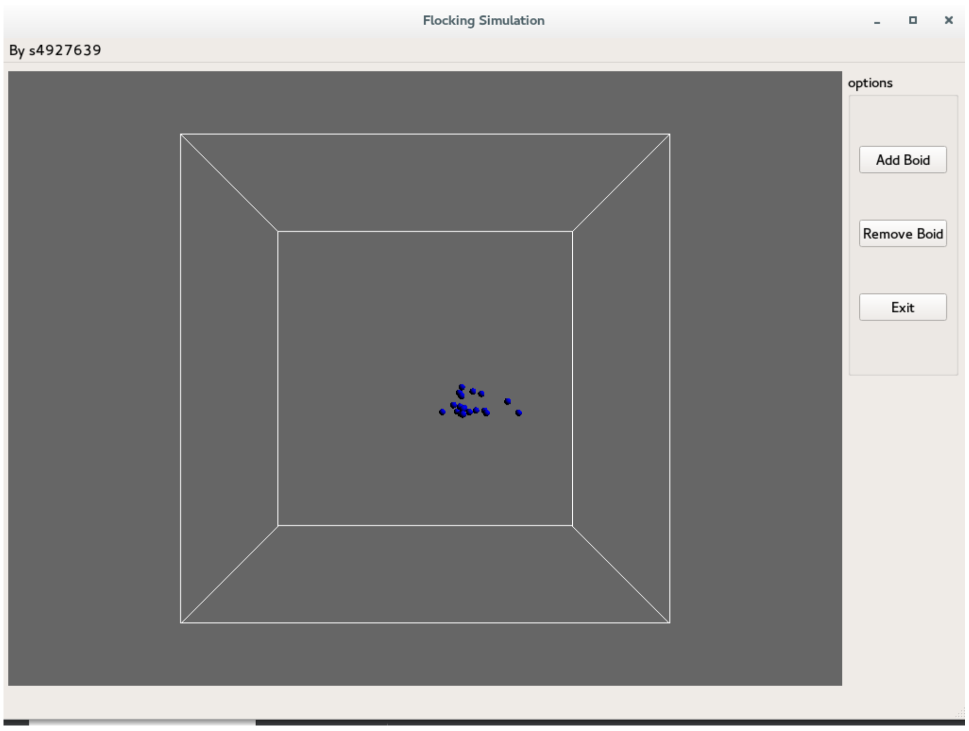
```
// render the movement of the sphere
for(Sphere &s : m_sphereArray)
{
  s.draw("nglDiffuseShader",m_mouseGlobalTX,&m_cam);
  s.move();
  s.arrive(m_target);
  s.separation(m_sphereArray);
  s.alignment(m_sphereArray);
  s.cohesion(m_sphereArray);
}
```

I added arrive function on render scene cause it has similar value as cohesion. If I added s.arrive, it keeps boids together. Whilst, if I didn't add s.arrive, boids tend to spread as small group.

I added a simple gui in order to control the number of boids. I applied push button on my window and linked it on MainWindow.cpp. I already built a main function on NGLScene.cpp. Therefore, I just linked to it from MainWindow.



```
connect(m_ui->add, SIGNAL(clicked(bool)), this, SLOT(addBoid()));
connect(m_ui->remove, SIGNAL(clicked(bool)), this, SLOT(removeBoid()));
connect(m_ui->exitB, SIGNAL(clicked(bool)), this, SLOT(close()));
```



- Add boid: add one boid when I click the button.
- Remove boid: remove one boid when I click the button.
- Exit: exit from the window.

Conclusion

Overall, I was really happy that I was able to understand how to move group through flocking algorithm in terms of birds, animals and games as well as c++ and ngl library. Also, I was able to build detailed programme compared to last year.
However, there is something to be desired during the project. Since I just built a basic algorithm of flocking, I could not add some complicated gui on my window. Also, I might be fun if I create a game with flocking system like I said on initial design.
During making the programme, It was variable time to understand and create different c++ and openGL programme unlike last year.

References

- Carl. E, 2011. Simulation of the Flocking Behavior of Birds with the Boids Algorithm. Sweden. Available from: http://www.red3d.com/cwr/steer/gdc99/ [Accessed 17 May 2017].
- Choi. Y, 2012. Flock. Available from: http://dogfeet.github.io/articles/2012/flocking-algorithm.html [Accessed 17 May 2017].
- Davison. A, 2003. Java Prog. Techniques for Games. Chapter 13. Flocking Boids. Available from: https://fivedots.coe.psu.ac.th/~ad/jg/ch13/chap13.pdf [Accessed 17 May 2017].
- Pemmaraju. V, 2013. 3 Simple Rules fo Flocking Behaviors: Alignment, Cohesion, and Separation. Envatotuts+. Avaliable from: https://gamedevelopment.tutsplus.com/tutorials/3-simple-rules-of-flocking-behaviors-alignment-cohesion-and-separation--gamedev-3444 [Accessed 17 May 2017].
- Reynolds. C, 1999. Steering Behaviors For Autonomous Characters[online]. California. Available from: http://www.red3d.com/cwr/steer/gdc99/ [Accessed 17 May 2017].