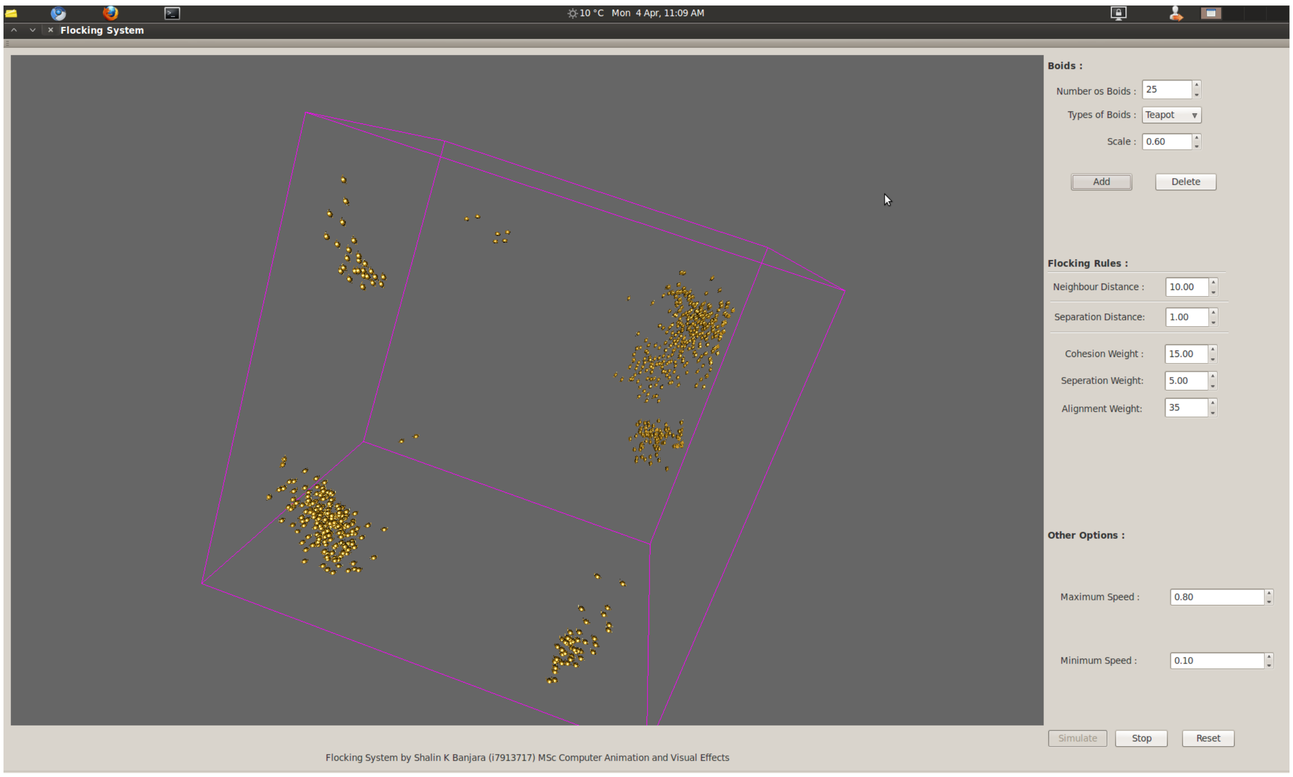
Animation Software Development



Boids :

Number os Boids : 25

Types of Boids : Teapot

Scale : 0.60

Flocking Rules :

Neighbour Distance : 10.00

Separation Distance: 1.00

Cohesion Weight : 15.00

Seperation Weight: 5.00

Alignment Weight: 35

Other Options :

Maximum Speed : 0.80

Minimum Speed : 0.10

Flocking System by Shalin K Banjara (i7913717) MSc Computer Animation and Visual Effects

C++ Flocking System

Shalin K. Banjara

(i7913717)

Introduction

The main objective of the assignment is to create a raw model of flocking system proposed by Reynolds, implementing the basic three rules of flocking along with dynamic creation and deletion of the boids. It was stated in the report handed in the first term that if time permits more complex flocking behaviour like predator-prey, target follow, environment object avoidance, wall follow, many other complex rules would be implemented.

This project serves as the base for creating more complex flocking system with more elaborate rules. It has a basic interface that allows the user to change the values real-time to visualize the three basic flocking behaviour and the way the behave in real-time upon the changes in the value.

Comparison between the first term report and the implementation

The first stage of the application was spent on implementing the system according to the initial design. Many minor aspects of the initial design were changed while implementation, there might be other better ways of implementing them also.

Most of the implementation of this system is refereed from the paper “Steering behaviours” by Craig W. Reynolds to better understand what methods he describes the best one to create a flocking system.

From now onwards the report will explain what methods were used to create a basic flocking system based on the research paper of Craig W. Reynolds.

Implementation

As a first step for implementing flocking system, there was a need to create few boids that can move one place to another. This provided a basic ground and debugging tool for developing and applying the steering rules. A basic locomotive model as suggested in the Reynold's paper with few modifications was used to move the boids in the scene.

Reynolds suggest the implementation of the the boids as a locomotive model with following few rules :

1) If the steering force applied on the boid is greater than the maximum force allowed, limit it to the maximum force limit .

2) Assign the steering vector to the acceleration of the boid.

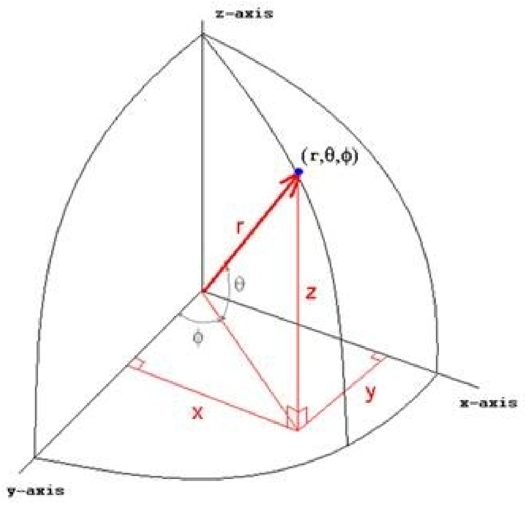
3) The velocity of the boid will be the previous velocity plus the acceleration.

4) If the magnitude of the final velocity vector exceeds the maximum speed value, then limit the vector according to the speed limitations.

These method has been implemented in the this program with a bit of modification. In this system, the mass of each object is considered as 1 unit. For orientation of the boids, a method of converting spherical coordinate system to Cartesian coordinate system is used.

$$\begin{aligned} \rho &= \sqrt{x^2 + y^2 + z^2} \\ S &= \sqrt{x^2 + y^2} \\ \phi &= \arccos\left(\frac{z}{\rho}\right) \\ \theta &= \begin{cases} \arcsin(y/S), & \text{if } 0 \le x; \\ \pi - \arcsin(y/S), & \text{if } x < 0. \end{cases} \end{aligned}$$

$$\Phi = \tan^{-1}\frac{x}{y}$$



1) Each time the new velocity vector is added to the current position of the boid.

2) Before applying the vector its is limited to the maximum maximum speed value of it exceeds it.

3) Also it is checked for the minimum speed limit is checked.

4) After getting the news position of the boid, distance between the current position of the boid and the new position is calculated to get the rotations for the boids aboving to the above stated formulae as seen in the project.

This simple method allows the boids the to move in the scene correctly by updating their positions and rotations.

Implementing the three basic Flocking behaviour

As describes in the Reynold's paper a realistic flock motion can be created by three simple rules of Cohesion, Alignment and Separation.

The cohesion behaviour states that the boid moves towards the centre of the flock. To implement this behaviour the following algorithm was applied.

a) check the positions of the neighbouring boids.

b) Find the average position of all these positions.

c) Steer to the location.

In pseudo code:

```
PROCEDURE rule1(boid $b_J$)

    Vector $pc_J$

    FOR EACH BOID b
        IF b != $b_J$ THEN
            $pc_J = pc_J$ + b.position
        END IF
    END

    $pc_J = pc_J$ / N-1

    RETURN ($pc_J$ - $b_J$.position) / 100

  END PROCEDURE
```

The separation behaviour states that each boid should avoid its fellow boids and should maintain some distance. To implement these rule the following algorithm was used :

a) Check whether the position of the surrounding boids is in separation distance or not.

b) These is done by subtracting a vector from all these boids displacement which are in the sepration range.

c) Adding these vector to the current position moves the boids away from all of these neighbouring boids.

In pseudo code:

```
PROCEDURE rule2(boid $b_J$)

    Vector c = 0;

    FOR EACH BOID b
        IF b != $b_J$ THEN
            IF |b.position - $b_J$.position| < 100 THEN
                c = c - (b.position - $b_J$.position)
            END IF
        END IF
    END

    RETURN c

END PROCEDURE
```

The alignment behaviour states that the member maintains the speed of the flock. It is imporatant to make sure that no boid is left behind for creating a realistic flock.

In pseudocode:

```
PROCEDURE rule3(boid $b_J$)


    Vector $pv_J$

    FOR EACH BOID b
        IF b != $b_J$ THEN
            $pv_J$ = $pv_J$ + b.velocity
        END IF
    END
```

```
        $pv_J = pv_J$ / N-1
        RETURN ($pv_J$ - $b_J$.velocity) / 8
    END PROCEDURE
```

Once these rules are implemented it becomes easy to see that we need a weight factors to multiply them with to create a good looking flock with smooth movements. These weight factors will assign a relevance factor to each rule and by changing their values influencing the outcomes each time the user changes their values.

Finally a mix function is used to add all these forces togather to create a new velocity vector. Even its better to check here that the new velocity does not exceed the maximum force force limit.

Neighbour Finding :: Brute Force

The most complex operation made thorughout the flocking system is finding the neighbours. A brute force algorithm is used, where each boid check its positions of every other boid in the scene to check whether they lie in a certain distance from them or not. This a bit time consuming since its complexity is n^2.

There are certain other techniques that can be used to solve this N^2 complexity.

Areas to improve

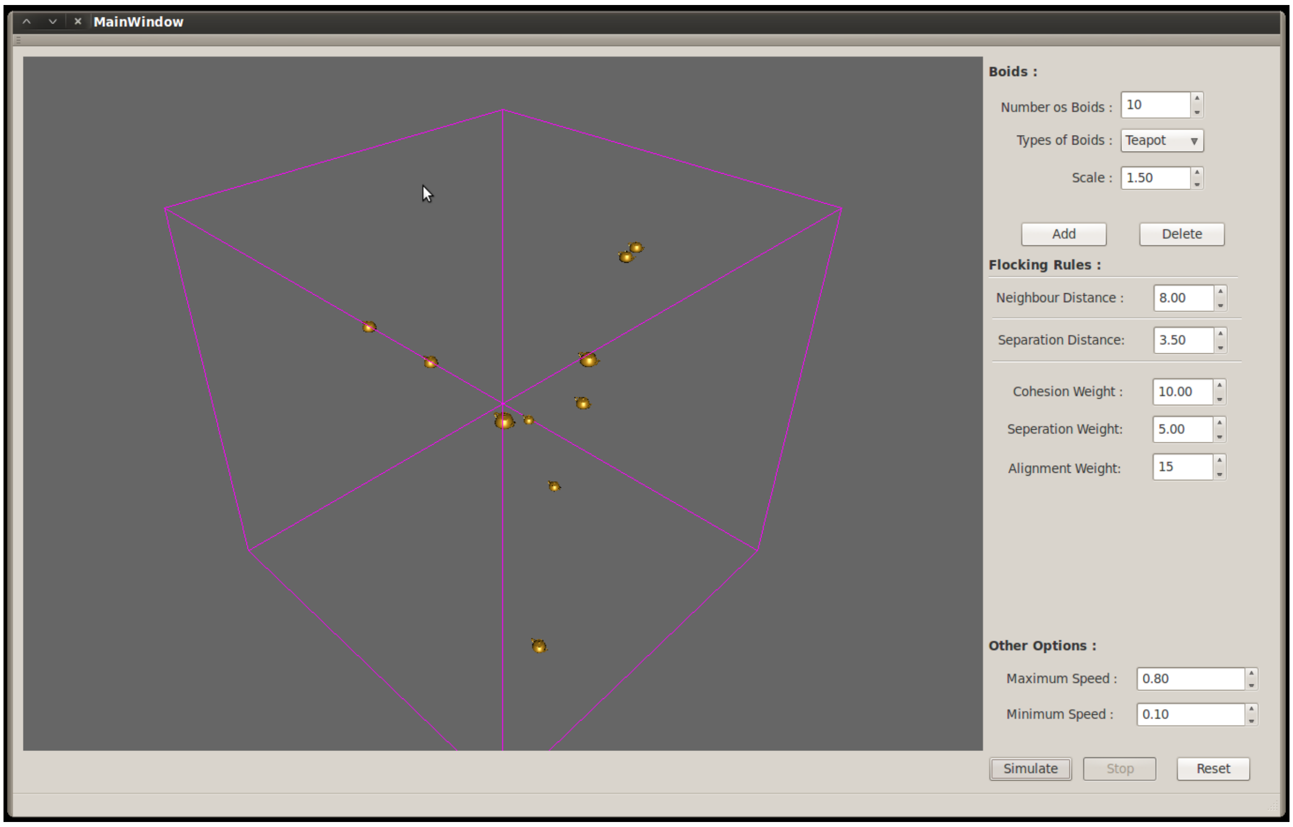
1) Solving N^2 Complexity for finding the neighbours

2) Collision with the environment objects.

3) More behaviours like Flee, Pursuit, Arrival, Seek Evasion, Target Follow, etc.

4) A better GUI with more functionality to the camera which is a prevailing limitation.

Interface



MainWindow

Boids :

Number os Boids : 10

Types of Boids : Teapot

Scale : 1.50

Flocking Rules :

Neighbour Distance : 8.00

Separation Distance: 3.50

Cohesion Weight : 10.00

Seperation Weight: 5.00

Alignment Weight: 15

Other Options :

Maximum Speed : 0.80

Minimum Speed : 0.10

The program has a simple Graphic User Interface. User can change the weight of the three rules to see their effect real-time. User can also specify the neighbour distance and separation distance to be considered.

User has controls to specify the maximum and minimum speed. User can select any one of the three predefined pre-sets from the combo box also see the boids as either teapots, spheres or pyramids. User can add or delete the number of boids relative to see the impact and changes in the shapes of the flock.

Conclusion

By the deadline almost all the things proposed in the initial design was developed except the object avoidance with the environment object. Overall I am happy with what I have achieved with the simulation as this being my first programming experience related to Graphics. It too gave me an idea of C++ in deep along with QT interface, and specially a good experience on using John's NGL library.

There are few areas of the project I would rework on. Those mainly include solving the N^2 complexity arising for finding the neighbours and object avoidance with the environment objects.

While developing these program it also gave me a sense of implementing complex mathematics algorithms in a program. Hopefully these I my first step towards AI.

Finally I would like to conclude by saying that I am happy with what I achieved and hope the flocking behaves properly in different scenarios.

References

1) Craig W. Reynolds, Flocks, herds and schools: A distributed behavioural model, ACM SIGGRAPH Computer Graphics, v.21 n.4, p.25-34, July 1987

2) Parent R, 2002.Computer Animation Algorithms and Techniques, Morgan Kaufmann Publishers, San Francisco, USA

3) Macey J, 2004, The Virtual Camera, Access on 25/03/2011 from http://dec.bournemouth.ac.uk/staff/jmacey/proggraph/VirtualCamera.pdf

4) Macey J, 2004, Collision Detection, Access on 20/03/2011 from http://dec.bournemouth.ac.uk/staff/jmacey/proggraph/Collisions.pdf

5) Vince, J. 2001, Essential Mathematics for Computer Graphics Fast, Springer