Flock Simulation of Birds

A comparison between two boid algorithms

AGNES SKATTMAN UDD

KTH Computer Science
and Communication

Bachelor of Science Thesis
Stockholm, Sweden 2010

Flock Simulation of Birds

A comparison between two boid algorithms

AGNES SKATTMAN UDD

Bachelor's Thesis in Computer Science (15 ECTS credits)
at the School of Computer Science and Engineering
Royal Institute of Technology year 2010
Supervisor at CSC was Cristian Bogdan
Examiner was Mads Dam

URL: www.csc.kth.se/utbildning/kandidatexjobb/datateknik/2010/
skattman_udd_agnes_K10033.pdf

Royal Institute of Technology
School of Computer Science and Communication

KTH CSC
SE-100 44 Stockholm, Sweden

URL: www.kth.se/csc

Abstract

The aggregated motion of a flock of birds originates from the decisions of individuals in the flock. No bird is leading the flock. Rather each bird follows its closest neighbours and thereby keeping the topology of the flock. Similarly a simulated bird chooses its own course and navigates according to its local perception of the surroundings. Craig W Reynolds was first to introduce this way of simulating birds. He called the simulated individuals boids, and let each individual boid follow three simple rules to decide their movement. The rules were: cohesion – to keep the boid close to its nearby flock mates, collision avoidance – to avoid collision with nearby flock mates and obstacles, and alignment – to adjust the speed and heading to nearby boids. This study evaluates two algorithms based on Reynolds ideas. The study compares the algorithms regarding cohesion, collision avoidance and alignment of boids during grouping, flight in group and when exposed to obstacles. It discusses some issues with these algorithms and suggests improvements.

Sammanfattning

En fågelflocks samordnade rörelse uppkommer av individernas beslut i flocken. Ingen fågel leder flocken, utan varje individ följer sina närmsta grannar. Detta bibehåller topologi i flocken. På samma sätt bestämmer simulerade fåglar sin flygkurs och navigerar enligt uppfattning av omvärlden. Craig W Reynolds var först att introducera detta sätt att simulera fåglar. Han kallade dessa simulerade individer för Boids och lät dem följa tre enkla regler för att bestämma deras rörelse. De tre reglerna var: sammanhållning – för att hålla samman Boids som befinner sig när varann, undvika kollision – för att undvika krockar in i andra fåglar eller hinder, och anpassning – för att anpassa hastighet och färdriktning mellan Boids som befinner sig nära varann. Denna studie behandlar två algoritmer som baserar sig Craig W Reynolds idéer. Studien jämför algoritmerna med avseende på sammanhållning, undvikande av kollision och anpassning mellan Boids, under gruppering, gruppflygning och möte av hinder. Den diskuterar problem med dessa algoritmer samt anger förslag på förbättringar.

Table of Contents

1. Introduction

When simulating flock behaviour of birds, objects called boids are used. These boids are simulated individuals of the flock and follows rules to make their decisions of where to go. Boids decide their movement based on cohesion, separation and alignment of the boids in their close surroundings. This study examines two different boid algorithms that simulate these birds. It compares the solutions of the problem and their different results in the simulations.

2. Background

2.1 Flocking of Birds

When living in a group of individuals, a flock, there are several risks that a bird can be exposed to. There is an increased intensity of competition of the resources in the group and the individuals have to share the amount of food with the rest of the flock and contest them to find a mate. While living dense the risk of parasites and other disease transmissions are increasing. In some cases also predators conspicuousness increase since a large group is more visible and noisy [1].

Why do individuals choose to live in a flock? According to Rob Nelson [1] there is two main reasons for flocking. Primarily it contributes to protection from predators, the flock can both keep predators away by using aggressive group defence and individuals can use the group as cover. Secondly it is easier to find food and detect predators the more eyes that are looking. As a result the birds can reduce their vigilance and therefor spend more time finding food.

2.2 Motion of a flock

Collective behaviour of large groups of animals, or flocking, is a natural behaviour that has puzzled many people. Examples of this is not only flocks of birds, but even fish schools and mammal herds [2][3]. The motion of flocks of birds seem quite randomly ordered and yet well arranged and synchronised. In 1987 Craig Reynolds thought that the flock motion was a result from actions of individual animals [3]. He thought that each individual acted on its own decisions from its perception of the world, which today is a common notion on how collective behaviour emerge [2].

To keep cohesion in a flock of birds, without colliding into each other, it is required to have some kind of interaction between the individuals of the flock. In a field study [2] it is shown that the interaction does not depend on metric distances to other bird as most previous models and theories assumed [3], but rather that each bird on average interacts with six or seven of its neighbours. This topological way of interacting means that every bird in larger group keeps its eye on six or seven of its closest neighbours and follow their every motion [2]. If a predator appear, the bird first to see it will change the direction to avoid becoming the prey. Instantaneously the flock will change direction in a manoeuvre wave [4].

Wayne Potts started 1984 [4] to do frame-by-frame analysis of high-speed film of sandpiper flocks and found that the reaction time of the individual birds in a flock, except for the first on to turn, actually were almost three times more than the measured visual reaction time of a bird. This he later tried to explained by that rehearsed manoeuvres among human chorus lines, initiated without warning, spread almost twice as fast as the human visual reaction time.

When a flock is not under attack it can fly around quite aimlessly since single individuals easily generate changes in movement, depending if the majority of the flock has a goal and is motivated to get there [4].

2.3 Flock Simulations Today

Today behavioural models such as simulations of flocks of birds are used in the entertainment industry to increase the realism in simulations of large groups of individuals [8]. These simulations are used in many movies, for example The Lord of the Rings trilogy. But are also used in digital games as background scenes, such in GTA – Grand Theft Auto, or as a part of gameplay itself, in Pikimin from Nintendo.

2.4 Particle Systems

Flock simulations is an elaboration of interacting between the objects in a particle system. Therefor flock behaviour simulations can be considered similar to a particle system [3]. Particle systems are used for “fuzzy” objects that do not have any smooth and well-defined surfaces. Object that are irregular, complex and ill-defined such as clouds, smoke, water and fire are defined by using particles that form their volume by changing shape and move during the passage of time. New particle are “born” and old particles “die” similar to a life-cycle [5]. The particles have behaviour that can alter their own state, which could consist of colour, opacity, location and velocity. When modelling a flock of bird a slight generalisation is made to the particle system. The original dot-like particles is changed to geometrical object with a full co-ordinate system and references to shapes that symbolises birds. The outcome is generally that more complex behaviour of the particles is used when modelling birds, as well as the particles interacting with each other and thereby have orientation. None of the simulated behaviour is nearly as complex as that of a real bird [3].

2.5 Boids

Craig W Reynolds was the first to make a flock simulation of birds. He named these animated birds to boids [3]. To build a simulated flock he started with a boid that supported geometric flight with the basic behaviours that avoided collision and supported cohesion and separation of the flock. There are three rules that the boids follow:

1. Collision avoidance: avoid collision with nearby flock-mates and obstacles
2. Alignment: attempt to match the velocity and heading to nearby flock-mates
3. Cohesion: attempt to stay close to nearby flock-mates

Collision avoidance and alignment is complementary and both ensure that the members of the simulated flock are free to fly within the flock interior without running into each other. The collision avoidance is determined by the relative position of a nearby flock-mate and ignores velocity. It is a static distance that make sure that the boids keep a minimum distance. Conversely the alignment bases on velocity and heading, and ignores position. By matching velocity the separation between the boids remain approximately invariant with respect to ongoing geometric flight.

Cohesion makes a boid want to be nearby its flock-mates, or in fact in centre of the nearest of them. If a boid is in inside the flock enclosed by flock-mates, the density will be approximately the same in all directions and the urge to centring is small. If however the boid is on the boundary of the flock, more of its neighbouring boids is on one side. The centroid of the neighbourhood boids are thereby placed toward the body of the flock and the centring urge is stronger and the flight path is adjusted somewhat toward that local flock centre.

The boid model allows separation in contrast to designated leader models. If an obstacle appears, a boid does not care if the flock is separated, as long as it stays close to its nearby flock-mates. The model has some problems with separated flocks that can not see each other until they are within a certain distance.

If the three rules are seen as forces into different direction with acceleration, and are weighted against each other, they can cancel each other and become a potential risk of collision to the boid. Therefor the rules are prioritised and in an emergency the acceleration would be allocated to satisfy the most pressing need first.

3. Problem and Hypothesis

This study will compare two different boid algorithms based on Craig W Reynolds [3]. The algorithms in the study will be chosen to be as simple as possible to be easier to compare. A simple boid algorithm in this case, simulates the boids in a two dimensional frame without walls and let them fly around aimlessly. But they still need to have all the features that are going to be examined. The features that will be compared are:

- Normal flight; the flight of a boid without influence of other boids or obstacles.
- Grouping; the cohesion, collision avoidance and alignment when perceiving other boids
- Flight in group; the cohesion, collision avoidance and alignment when flying in a flock.
- Encounter obstacles; how the boids manage to veer to obstacles

The first three criterions are critical to get a working flock of boids that are flying around aimlessly in a frame, that has a resemblance to real birds. The last criteria gives some additional features to the boids as well as examines the other criterions when the boids are influenced by the surrounding of the flock. All criterions together should show if the weights in the algorithms are balanced and if there are any major problems with the algorithms. The study will also argument for improvements of the algorithms.

4. The Implementation of the Boid Algorithms

The two algorithms chosen are both based on the original ideas of Craig W Reynolds [3] and are composed as applets. They simulate the boids as two-dimensional shapes that have indicated heading, moving around within a frame without walls. When a boid reaches the edge it appears again on the opposite side of the frame. The boids are originally placed at random and are thereafter flocking according to the forces of the algorithm. Both applets has a panel to adjust features such as number of boids, speed, detection distance and separation distance. These features have a predefined default value when the simulation starts and resets. In both applets the detection and separation distance can be shown as a circle around the boids. They also provide obstacles and food objects that can be placed in the frame during simulation. When describing the algorithms the Cartesian co-ordinate system is used.

4.1 The Lalena Algorithm

The first algorithm is written in Java by Michael Lalena [6]. Except from the properties already mentioned, this algorithm also provides boids of different types (flocks) that will keep separated from each other and a predator type that kills the boids of other types if reaching them. These features are not used in this study however. In figure 1 a screen shot of the applet is shown.

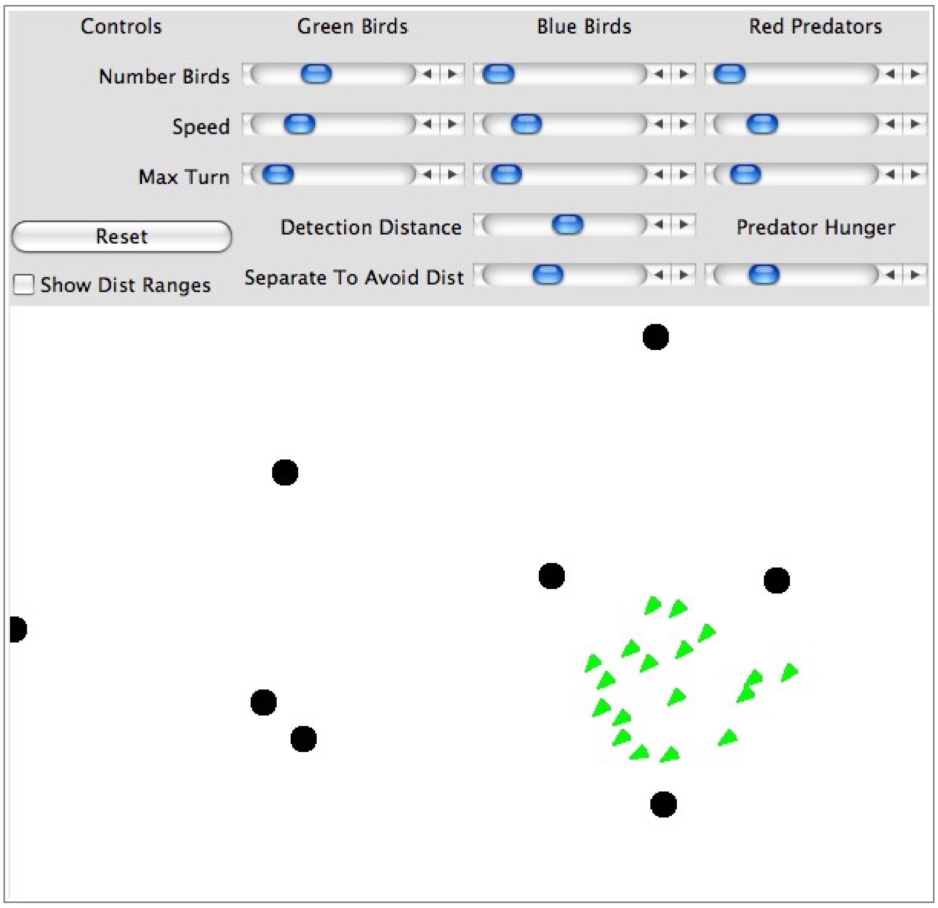


Figure 1. A screen shot from the Lalena applet [6]. Only green boids are generated, shown as green triangles. The black circles are obstacles.

Some modifications from the Reynolds algorithm are made. All boids of the same flock has the same speed and therefor do not need to adjust it to their surroundings. Also the cohesion and collision avoidance is only based on weighted decisions and are not prioritised in strained situations, even if they are changing with different distances.

The applet has a panel to adjust some features within a predefined boundary. Features relevant to this study are the number of boids, the maximum turn angle, the speed, and the detection and separation distance of the boids. If the number of boids is reduced, the first boid in the list of boids is removed independent of its location. If a new boid is added it is placed at a random location in the frame. The maximum turn angle and speed is initiated in the next iteration of the algorithm. When detection or separation distance is changed, the boids try to adjust themselves to the applied forces. Here follows some pseudocode with a more detailed description of the decisions of the boids.

All boids are made as objects with certain properties. All properties except location and heading are customisable during simulation.

```
//A Boid object consists of:
location            //A point with the current x and y co-ordinate of the boid
heading             //The current heading of the boid
speed               //The speed of the boid
maxTurnAngle        //The maximum turn angle
detectionDistance   //The detection distance to other boids and obstacles
avoidDistance       //The distance when to avoid collision
```

In every iteration of the applet, the algorithm updates the position of the boids. This is made by the function moveFlock that iterates through the boids and call the functions calculateNewHeading and moveBoids that are described below.

```
//Move all boids
Function moveFlock()
    for all boids b do
        newHeading = calculateNewHeading(b)
        moveBoid(b, newHeading)
```

The function calculateNewHeading iterates through the boids and obstacles within detection distance of the input boid. The function calculates the new heading by take the mean force of the forces that these boids and obstacles contribute to. The function uses normalisePoint that is described below.

```
Function calculateNewHeading(boid)
    neighbourBoids      //The sum of the general location of boids within detection distance
    numberOfBoids       //The sum of boids within detection distance
    for all boids b do
        //The distance between boids are sometimes closer if you go off the edge of the map.
        //Then recalculate the location of the other boid b, even if it will be outside the frame
        otherLocation = recalculateLocation(boid.location, b.location)
        distance = calculateDistance(boid.location, otherLocation)
```

Here follows calculations of the forces from other boids and obstacles within detection distance of the boid. If the boid is outside the avoid distance of a neighbour boid, it will add an align and cohesion force according to calculated weights. If it is within avoid distance of a neighbour boid or detection distance of an obstacle it will instead of using cohesion, avoid collision and be repulsed according to calculated weights.

```
        if b != boid && distance < boid.detectionDistance do
            weight = 100            //Alignment weight is 100
            align = (weight * cos(b.heading), weight * sin(b.heading))
            align = normalisePoint(align, weight)
            //Calculate cohesion weight, positive for collision avoidance, negative for
            cohesion
            weight = 200
            //When within avoidDistance; the closer to the other boid, the greater weight
            if (distance < boid.avoidDistance)
                weight *= (1 – (distance / boid.avoidDistance))^2
            //When outside avoidDistance; the longer from it the greater negative weight
            else
                weight *= - ((distance – boid.avoidDistance) / (boid.detectionDistance -
                boid.avoidDistance))^2
            cohesion = boid.getLocation - otherLocation
            cohesion = normalisePoint(cohesion, weight)
            dist = cohesion + align
            dist = normalisePoint(dist, 100)        //Final weight is 100
            neighbourBoids += dist                  //Summarise the points
        //If b is an obstacle and have to be avoided:
        else if b = obstacle && distance < boid.detectioDistance do
            dist = boid.getLocation - otherLocation
            dist = normalisePoint(dist, 1000)

            weight = (1 -  distance / boid.detectionDistance)^2
            dist = dist * weight
            neighbourBoids += dist
        numberOfBoids++
```

Now the calculated forces are used to get a new heading.

```
    if numberOfBoids == 0
        return boid.heading
    else
        neighbourBods *= (1 / numberOfBoids)        //Calculate mean
        neighbourBoids += boid.getLocation
    //Use the target point to calculate a direction angle within 360
    newHeading = (calculateAngle(boid.getLocation - neighbourBoids) + 360) % 360
    return newHeading
```

The function normalisePoint, normalises a point p with an x and y co-ordinate with a specified weight.

```
Function normalisePoint(p, weight)
    weight = weight / (p.x^2 + p.y^2)
    p = (p.x * weight, p.y * weight)
    return p
```

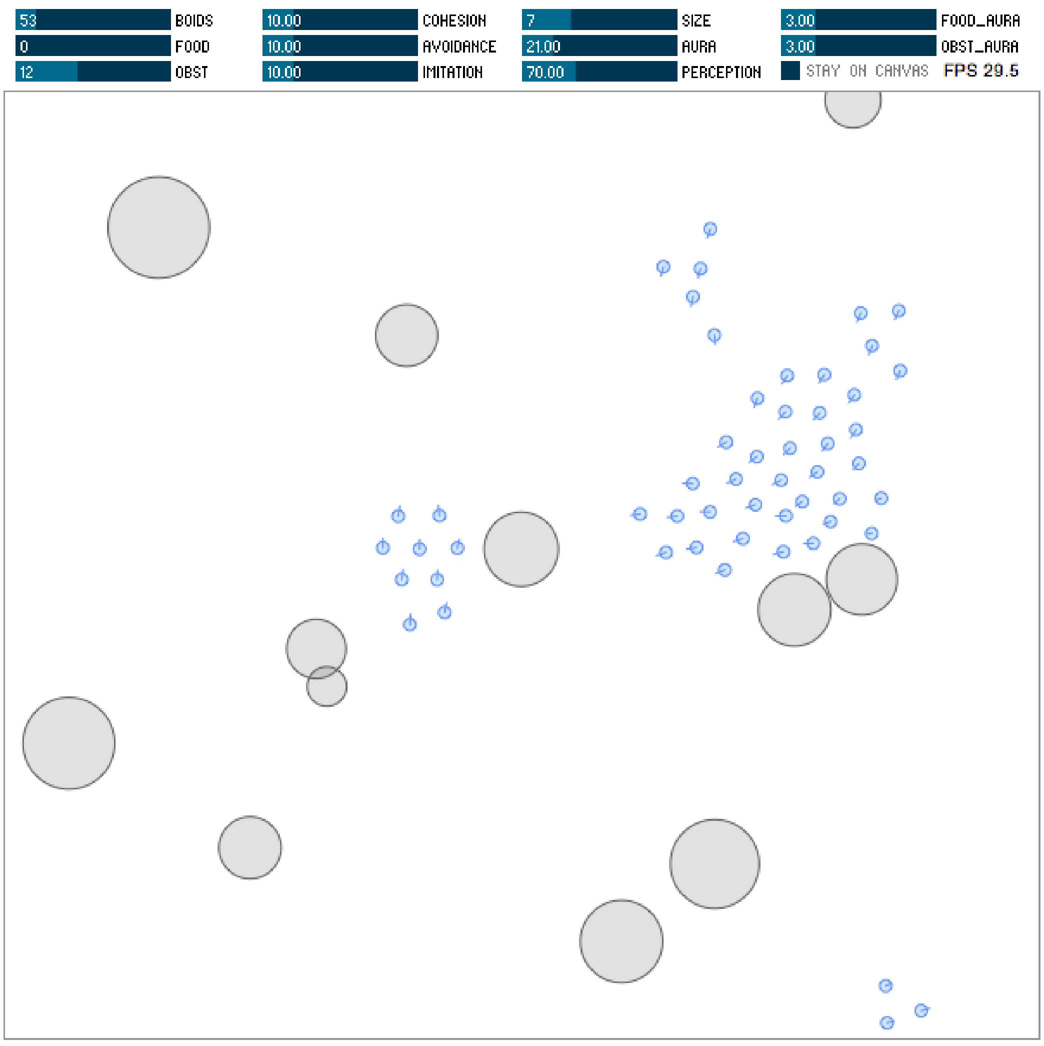
The function moveBoid takes a boid and a heading as input and calculates a new heading for the boid, by turning it the number of degrees specified by the input heading.

```
Function moveBoid(boid, newHeading)
    //Decide if to turn left (positive) or right (negative) within maximum turn boundaries
    turn = decideTurn(newHeading);
    //Calculate new heading within 360
    heading = (heading + turn + 360) % 360
    //Calculate new co-ordinates within the boundaries
    location = ((speed * cos(heading) + width) % width, (speed  * sin(heading) + height) % height)
```

4.2 The Liautaud Algorithm

The second algorithm is written by Nicolas Liautaud [7] in the Processing language. This algorithm has additional features such as keeping the boids within the walls of the frame and avoidance of the mouse.

The algorithm is using Reynolds three rules and have weights to affect them. The weights are customisable, except when the boids comes into contact with an obstacle and are affected by a huge weight to avoid collision. Customisable features relevant to this study are number of boids, weights for cohesion, avoidance and imitation, detection and separation distance of the boids and number of obstacles. In figure 2 a screen shot from the applet is shown.



```
53     BOIDS   10.00  COHESION    7      SIZE         3.00  FOOD_AURA
0      FOOD    10.00  AVOIDANCE   21.00  AURA         3.00  OBST_AURA
12     OBST    10.00  IMITATION   70.00  PERCEPTION   STAY ON CANVAS  FPS 29.5
```

Figure 2. A screen shot from the Liautaud applet [7]. The blue dots are boids and the grey circles are obstacles.

Adding and removing boids and adjusting detection and separation distance acts the same way as in the Lalena algorithm. Adding obstacles generates obstacles with random size at a random location in the frame. Removing an obstacle, removes the obstacle in the frame that was first generated. The rest of the features is changed instantaneously during next iteration of the algorithm.

The boids are initiated with random speed and are then adjusting themselves to its surrounding according to the weights of the algorithm. The boids do not have any perception of each other over the edges of the frame. More details about the decision of the boids follows in pseudo code.

All boids are made as object with certain properties. All properties except location and speed are customisable during simulation.

```
//Boids consists of
location            //A point with the x and y co-ordinate of the boid
speed               //A vector with the current speed in x and y direction
maxSpeed            //The maximum speed of the boid
detectionDistance   //The detection distance to other boids or obstacles
avoidDistance       //The distance when to avoid collision
cohesion            //The weight for cohesion among the boids
avoidance           //The weight for avoidance among the boids
imitation           //The weight for imitation among the boids
```

In every iteration of the algorithm it updates the position of the boids. The new position is calculated by the function moveFlock that iterates through the boids and call the functions goToCenter, keepDistance and matchVelocity. It also keeps the boid within the frame, limits the speed of the boids and use the function effector to keep the boids from colliding with obstacles. The mentioned functions are described below.

```
Function moveFlock()
    for all boids b do
        //Move basic rules
        goToCenter(b)
        keepDistance(b)
        matchVelocity(b)

        //When the boids reaches the end of the frame it is teleported to the other side
        keepInFrame(b);

        //Check that the boid is within speed limit and adjust it
        if(magnitude(b.speed) > b.speedLimit)
            b.speed = (b.speed / magnitude(b.speed)) * b.maxSpeed
        //Avoid obstacles
        for all obstacles o do
            //Small repulsion when perceived
            effector(b, o.location, o.detectionDistance + b.detectionDistance, -0,2)
            //Big repulsion on contact
            effector(b, o.location, o.size + b.size, -20)
        b.pos.add(b.speed)
```

To keep the flock together the boids need a cohesion force. The function goToCenter iterates through the neighbour boids within detection distance of a boid. The function takes the average position of the neighbour boids and multiply it with the cohesion weight and add it to the speed of the input boid.

```
//Cohesion with other flock mates
Function goToCenter(boid)
    neighbourBoids              //The sum of locations of the boids within detection distance
    for all boids b
        if (b != boid && withinDetectionDistance(b, boid))
            neighbourBoids += b.location
    if (neighbourBoids)
        //Let the boid cohere  with the neighbours
        neighbourBoids = average position of neighbourBoids
        neighbourBoids = neighbourBoids * cohesion * 0.001        //Set intensity
        speed += neighbourBoids
```

To keep the boids in the flock separated the boids need a collision avoidance force. The function keepDistance iterates through the neighbour boids within avoid distance of a boid. The function adds or subtract speed from the force depending of the location of a neighbour boid. When the sum of speed adjustments is calculated, the force is multiplied with the avoidance weight and added to the speed of the input boid.

```
//Collision avoidance with other flock mates
Function keepDistance(boid)
    neighbourBoids              //The sum of locations of the boids within avoid distance
    for all boids b
        if (b != boid && withinAvoidDistance(b, boid))
            //NeighbourBoids will add or reduce speed in a direction depending on it
            location in comparison with the boid
            neighbourBoids += normalise(boid.location – b.location) / distance(b, boid)
            //Normalise divides with the norm of the vector (point)
        neighbourBoids *= avoidance
        speed += neighbourBoids
```

To make the flock adapt their speed and heading, the boids need an alignment force. The function matchVelocity iterates through the neighbour boids within detection distance of a boid. The function takes the average speed of the neighbour boids and multiply it with the imitation weight and add it to the speed of the input boid.

```
//Alignement with other flock mates
Function matchVelocity(boid)
    neighbourBoids              //The location of the boids within detection distance
    for all boids b
        if (b != boid && withinDetectionDistance(b, boid))
            neighbourBoids += b.speed
    if (neighbourBoids)
        //Let the boid cohere  with the neighbours
        neighbourBoids = average speed of neighbourBoids
        neighbourBoids = neighbourBoids *  imitation * 0.01        //Set intensity
        speed += neighbourBoids
```

The function effector iterates through the obstacles within detection distance of a boid. The function will increase the speed in the x respectively the y direction if this is turning the boid away from the obstacle, and decrease the speed if there is a risk of collision.

```
//Collision avoidance with obstacles
//oLocation, oDetectionDistance and oIntensity is coordinates and effecting area/intensity of the
obstacle
//The oIntensity should be negative for repulsion
Function effector(boid, oLocation, oDetectionDistance, oIntensity)
    dist = distance(oLocation, boid.location)
    if (dist < oDetectionDistance)
        vector v = (0, 0)               //The speed adjustment to be calculated
        if (boid.location.x >= oLocation.x)
            v.x += (oDetectionDistance – dist) * oIntencity * 0.01
        else
            v.x -= (oDetectionDistance – dist) * oIntencity * 0.01
        if (boid.location.y >= oLocation
            v.y += (oDetectionDistance – dist) * oIntencity * 0.01
        else
            v.y -= (oDetectionDistance – dist) * oIntencity * 0.01
        boid.speed -= v
```

5. Performance of the Boids

These results are a combination of reading the source code of the algorithms and observation from live simulations. When simulating the boids several attempts were made to observe the most interesting cases. Special cases, where for example the exact same but opposite speed and heading of the boids could cancel the different forces out are ignored. The customisable settings are investigated when relevant, except for detection and avoidance distance that remain with predefined values.

5.1 Normal Flight

One boid is initiated with the predefined properties of the algorithm (except for number of boids).

The Lalena algorithm initiates a boid with random heading and location. Thereafter it is moved in the heading with the default value of speed, until reaching the end of the frame and it reappears at the opposite side. This is repeated until any conditions change.

The Liautaud algorithm has instead of heading, speed in x and y-direction. When initiating the boid it gets a random position and speed. Then it continues as the Lalena algorithm.

5.2 Grouping

Two boids are initiated with the predefined and random properties of the algorithms and get time to intercept each other and group. Thereafter more boids are added. Customisable features that have impact on grouping forces are investigated.

With the Lalena algorithm, the boids sometimes get headings which unable them to intercept each other. In these cases the boids are regenerated. When successful and the boids come into the detection distance of each other, they start to cohere and align with each another. The change in heading is fast and executed long before a boid gets within the separation distance. If the change is small, the turn is completed faster and the boids will stay at a longer distance from each other. When the turn is completed the boids continues with previous speed in the new collective heading, with the same distance from each other as when the turn was completed.

When a third boid gets into detection distance with one of the first two boids, the new boid and the one who perceives it align and cohere within maximum turn angle during one iteration, then the third boid perceives this change and will also influence the heading. This often results in a small sick-sack movement before a collective heading is decided. This sometimes occurs even when the first two boids group, depending on which boid that turn first. The more boids that are group together, the less they are influenced of a new boid that is joining them.

If the maximum turn angle is reduced, they boids get much softer turns that look more natural. This sometimes results in failing to group together and getting inside the avoid distance of each other. Increasing maximum turn angle does not make any difference in grouping. Changing the speed of the boids does not impact their decision.

When the boids of the Liautaud algorithm perceives each other, the function goToCenter adding speed towards the location of the other boid. The matchVelocity function should neutralise this force, but its imitation weight is to small. This eventuates the boids to start charge against each other. When the boids come into separation distance of one another, the keepDistance function start to slow down the boids and making a bumping effect to avoid them from colliding. Together with the keepDistance function, matchVelocity can make the boids decide a new collective heading. Since the goToCenter function still adding the position of the other boid to its speed, the boids speed will increase until nearly maximum. If the two boids originally have similar course, this “bumping” problem will not occur, since matchVelocity only need to make a small job to match the horizontal and vertical speed, while goToCenter helping to get the same magnitude of the speed. The speed of the boids will still increase since goToCenter boosts it.

When a third boid is added the same bumping into the separation distance occur. If the imitation weight is increased the boids are grouping more slowly and the bumping disappears.

Since the algorithm does not have any perception between boid over the edges of the frame, the boids sometimes lose track of each other when reaching an edge.

If the cohesion weight is increased the boids ending up flying within the separation distance of the other boids. If it is reduced a little the force in goToCenter is diminished, if zero the birds do not cohere but will stay together according to the matchVelocity.

5.3 Flight in Group

A group of five, ten and thirty boids are generated and studied.

With the Lalena algorithm, the flock of boids get denser the more boids that are added. Already in a flock of five members the boids sometimes get into the detection distance of each other. By ten boids the group is very dense, and by thirty the flock is a mass with boids on top of each other. After the flock has finished grouping the different forces from the algorithm are stabilised and the distances and headings in the group are not changed.

If the separation distance are increased, the boids keep longer distances to each other, but some boids still end up within other boids separation distance and this problem still increasing with a larger flock.

The Liautaud algorithm keeps the distance between the boids regardless of the size of the flock. Sometimes a boid gets into another boids separation distance, but it correcting this quite fast and keeping enough distance to not collide. The boids is adjusting their speed and heading all the time, the general flock keeps the heading and speed quite stable, but small individual changes are present. This make the flock look more alive.

When adjusting the separation distance, the flock is adapting to the change almost instantaneously and keeps working as before except for a more spacious appearance. If decreased the opposite result is maintained. The cohesion and imitation weights work as described during grouping.

The flock keeps losing contact when reaching the edges of the frame.

5.4 Encounter Obstacles

25 obstacles are placed scattered over the frame. The boids are then generated in groups of five, ten and thirty individuals.

The Lalena algorithm keeps in most cases the boids from colliding with the obstacles. When moving in flocks of five and ten boids, they still avoiding the obstacle well, but sometimes bump into each other as described during normal flight. Sometimes they split to avoid the obstacle and are rejoining on the other side. When thirty boids are generated the group gets like a mass with boids on top of each other and the boids start to skitter over the obstacle instead of moving around them. Sometimes they even fly straight over the obstacles instead of veering.

When the obstacles are placed with the Liautaud algorithm, the customisable feature is used and generates random sized obstacles at random locations. Quite often when the boids are heading straight at an obstacle they fail to veer and bump into it and the greater weight is used to get away. This occur more often, the more boids there are to bump into the obstacles. When a flock of thirty boids are generated, they are sick sacking between the obstacles with some boids bumping into them in every turn. The boids seem somewhat better to veer when they are few.

If the avoidance weight is increased the flock in general is better to veer to the obstacles and keeping a longer distance to them. But some boids still colliding.

6. Discussion

The Liautaud algorithm looks more natural than the Lalena algorithm due to more movement among the boids, when adjusting the speed to each other. The Lalena algorithm should add individual speed to the boids for a more natural appearance. The boids in the Lalena algorithm also look unnatural when turning. They change their direction very fast and together with their collective speed it looks lifeless. When the maximum turn angle are reduced it looks better and the sick sacking among the boids decreased. Unfortunately the boids started to fail to group instead. This problem could be solved by fit the speed so the boids have longer time to adjust their heading (by slowing down) or with a greater detection distance.

The Lalena algorithm also has problems with boids that end up within detection range of each other. This is a result of too much cohesion as well as to little collision avoidance among the boids. This should be solved trying to increase weight that is used when the boids come into avoid distance of each other and try to lower the alignment weight and the weight that is used when the boids are outside of the avoid distance. The easiest way to get the weights balanced should be to make them customisable and try different weights during several simulations.

The major problems with the Liautaud algorithm are that the boids charge at each other and that they collide with obstacles. The charging can be solved by increasing the imitation weight. This reduces the occurrence of the boids bump into each other as well. The effector algorithm that is used to avoid colliding with obstacles does not work when the boids are heading straight at obstacles. This depends on the function, that is only turning a boid that is already turning away from an obstacle. The function should add features that make the boids that heading straight or almost straight at an obstacle to pick a side to veer.

The boids in the Liautaud algorithm also increasing speed after they have grouped, since the function goToCenter is adding speed to the boids. This could be solved by add some more orientation to the boids. If a boid is in front of another boid, it should slow down or stay at the same speed, instead of increasing it to adjust it to the other boid.

The Liautaud algorithm does not have any perception between the boids over the edges. This should be added to keep the boids together even in these areas.

6.1 Conclusion

Both algorithms have some problems to work according to Craig W Reynolds ideas [3]. Most of them are quite easily adjusted, when they depend on unbalanced weights. To add individual speed among the boids in the Lalena algorithm probably need some more effort to get working however, since most of the forces have to be adjusted. If the problems in the algorithms were corrected I think they would be equal in their performance.

Both algorithms were chosen for being simple and are therefor harder to compare to real birds. That the simulations are two dimensional make them far from looking like real birds, but disregarding the simpleness they still need some more features before they can look like real birds. To add some randomness should really help, since that would make them less computer like. Thereafter I think they need a more natural environment that require some intelligence of the boids and gives them a reason for acting the way they do. An environment that for example creates an urge to find somewhere to land or make the flock slow down to look for food in an area.

7. References

[1] Nelson, R. “Flocking” [www]. From <http://www.thewildclassroom.com/biodiversity/birds/aviantopics/flockingbehavior.html > 7 Mars 2010

[2] Ballerini, M., Cabibbo, N., Candelier, R., Cavagna, A., Cisbani, E., Giardina, I., Lecomte, V., Oriandi, A., Parisi, G., Procaccini, A., Viarle, M., Zdravkovic, V. (2007). “Interaction ruling animal collective behaviour depends on topological rather than metric distance: Evidence from a field study”. Vol 105, no. 4. Yvette, France: PNAS. pp. 1232-1237

[3] Reynold, C. W. (1987). “Flocks, Herds, and Schools: A Distributed Behavioural Model”. Computer Graphics, 21, July 1987, pp. 25-34.

[4] Potts, Wayne K. (1984). "The chorus-line hypothesis of coordination in avian flocks". Nature 309, 24 May 1984, pp. 344-345.

[5] Reeves, W. T. (1983). “Particle Systems—a Technique for Modeling a Class of Fuzzy Objects”. ACM Transactions on Graphics, Vol. 2, No. 2, April 1983, pp. 91-108.

[6] Lalena, Michael. “Flocking Behavior Simulator” [www]. From <http://www.lalena.com/AI/Flock/> 7 Mars 2010

[7] Liautaud, Nicolas (2009). “Boids” [www]. From <http://www.openprocessing.org/visuals/?visualID=8676 > 17 April 2010

[8] Da Silva, A. R, Lages, W. S., Chaimowicz, L., (2009). “Boids that See: Using Self-Occlusion for Simulating Large Groups on GPUs”. ACM Computers in Entertainment, Vol. 7, No. 4, Article 51, December 2009.