DH2323 Computer Graphics and Interaction Implementing 3D Boids with Spatial Partitioning

Alexander Danielsson - aldani@kth.se
Jacob Brännström - jacobbra@kth.se
Grade aimed for: A
Link to our blog: https://graintboids2022.blogspot.com/

May 2, 2023

1 Abstract

We present the implementation of a boid simulation which uses spatial partitioning to improve performance. The simulation is rendered using our own renderer which utilizes OpenGL. We studied the simulation's performance as the number of boids increased, as well as when using different side lengths for the bounding box. From this we found that the spatial partitioning provided a significant speedup. Furthermore, we propose a possible perceptual study which can be done to improve the realism of the simulation.

2 Introduction and Background

Animation is one of the largest fields in Computer graphics and the field is always trying to push for even more realistic, easy and fast animation methods. One of the methods of computer animation is behavioural animation, where the animation is controlled by the choices that the animated agents make. These choices are then guided by some rules set by the artist, so to direct the animation to their liking. A famous and well recognized behavioural animation model is the boid simulation, which aims to simulate the behaviour of birdlike objects (boids). However, the original and naive implementation of the simulation is said to be $O(n^2)$, where n is the number of boids, which is very inefficient. In this project we have attempted to improve the efficiency of the boid simulation by using spatial partitioning. We used the code we made in the rasteriser lab in the course DH2323 Computer Graphics and Interaction as the basis to render the boids.

3 Related Work

Craig Reynolds' paper [7] is one of the first published attempts at using behavioral animation to animate the flocking and herding behaviour of birds and fish. The paper describes an agent based simulation model, in which the boids are animated one-by-one through the computation of three basic rules affecting their velocity in 3D space. These rules all depend on the position or velocity of sufficiently nearby boids, and cover collision avoidance, the matching of velocities and the urge to stay near the center of the flock.

As part of their bachelor thesis [3], Jakob Kratz and Viktor Luthman conducted a comparative study of the performance of different spatial partitioning schemes when used for Boid simulations. The work compares the performance of the simulation when using a Grid, Quad Tree or Kd-Tree spatial structure, and how this changes with the spread and number of boids. From this they concluded that the Grid performed best for smaller simulations, but that for larger simulations of more than 5000 boids, tree based data structures should be used instead. Furthermore, all schemes benefited from having the boids evenly distributed in the space, although the grid was most resilient to this.

In their paper, Jae Moon Lee, Se Hong Cho and Rafael Calvo proposed an optimised scheme for Grid based spatial partitioning [5]. The proposed optimisation built on the observation that the nearest neighbours of boids frequently remain the same between two simulation ticks, and thus could frequently be reused for its calculations. By using their proposed algorithm, they found that the performance of the simulation could be improved by 57.7%.

4 Implementation

4.1 GPU Rendering

In our original specification for the project, we had aimed to reuse our rasterisation renderer written for the third render lab in DH2323 Computer Graphics and Interaction. However, as its base performance was rather slow, we opted to instead re-implement the renderer using OpenGL.

The implementation used SDL for handling input and window management, and OpenGL as the backend for rendering. In order to configure SDL to allow for this, we followed what was written in the SDL manual [4]. Initially we changed the number of bits used for colours and the depth buffer to 32 bits, but had to decrease it to 24 bits as we found that our different computers differed in available capacity. We also discovered that we had to use a OpenGL loader to access modern OpenGL functions, and thus added GLAD to the setup.

The renderer itself was based on the ideas of Joey de Wries on his website "Learn OpenGL" [2]. Specifically, we opted to move the code that handled the projection to screen coordinates and lighting calculations to a vertex and fragment shader respectively. As we planed to reuse a single model for all of the boids, we created only a single VBO consisting of each vertex's position

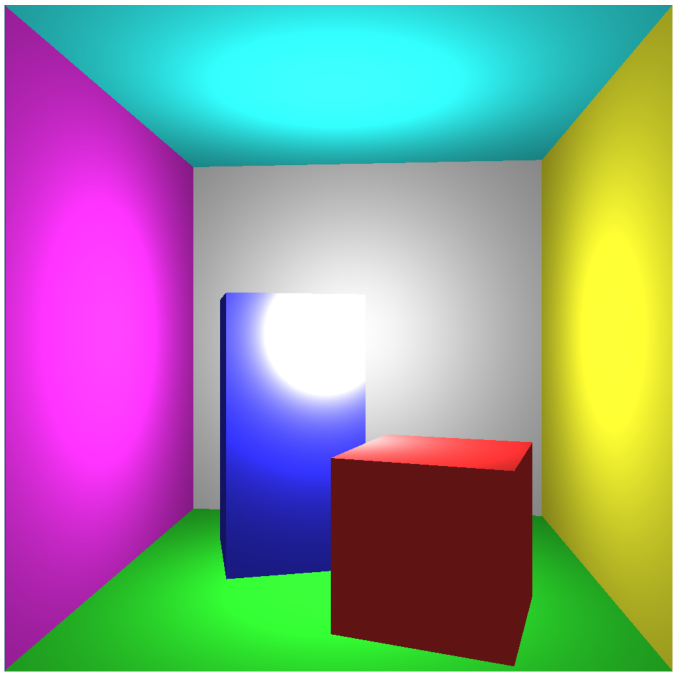

Figure 1: The room from the rasterisation lab of DH2323 Computer Graphics and Interaction when rendered using the OpenGL renderer

and normal. We then used two vertex attributes to instruct the GPU in how to extract these two items from the bound VAO.

For our draw function we would each frame render each boid individually. The vertex shader would take a transformation that rotated and translated the boid model into the correct position for each boid. It would then use this in conjunction with a transformation matrix and projection matrix for the camera to calculate the correct screen position of each vertex. The fragmentation shader would then use this information in combination with information about the position of the scenes light source and the boid's colour to calculate the correct colour of each pixel.

The result of these changes when rendering the scene from the third rasterisation lab can be seen in Fig. 1 and with boids in Fig 2. When comparing the render time of this implementation to the original render time, it was found that the program ran about 78 times faster.

4.2 Boid Simulation

For the boid simulation we based our implementation of of our understanding of Reynolds paper [7], with clarifications provided by Conrad Parker's pseudocode [6] and previous work done for the course DD1354 Models and Simulation.

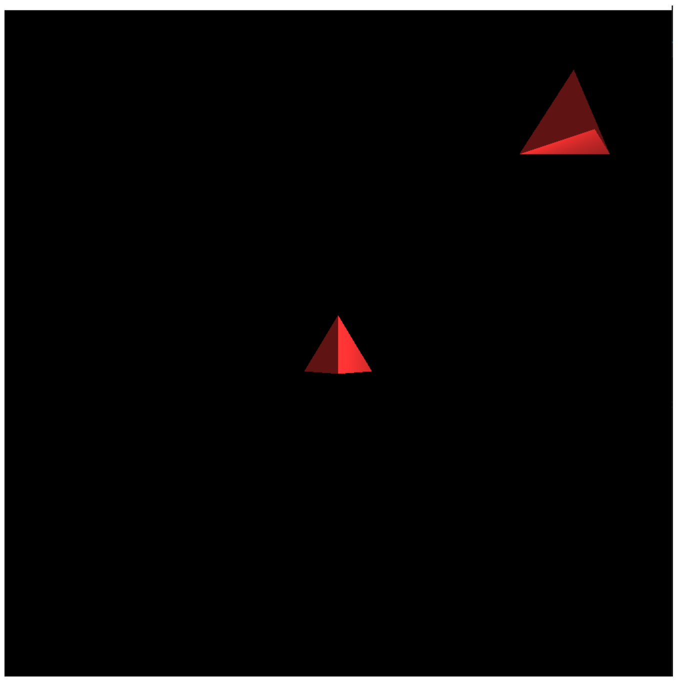
Figure 2: Two boids rendered using the OpenGL renderer

Specifically, the implementation would for each simulation tick go through all of the boids, and for each calculate the change of its velocity based on five rules. These five rules consisted of the original three rules identified by Reynolds, a rule for bounding the boids within a bounding box and a rule applying a slight drag force to the boids. The velocity of the boids were bounded within an interval, and the amount the boids would move for each simulation tick was scaled by the amount of time that had passed in between each frame of the program.

We originally only included Reynolds three rules and a simple rule for bounding the boids within a sphere with a given radius. The motivation for the inclusion of the fourth rule was to ensure that the boids would remain within a given area as to assure that the user could keep track of the boids. Reynolds rules was implemented as had been described, although each rule would only take into consideration the position and velocity of boids within a given radius of the simulated boid. Each rule had an individual radius and strength.

The fourth rule was originally implemented as a check if the boid was outside the given bounding sphere. If so, a scaled vector pointing from the boids position to the center of the bounding sphere would be returned, to simulate an urge for the boid to return inside of the sphere. However, we would later find that the use of a bounding sphere, in the combination with the forces and radiuses used at the time, made the simulation tend to stabilise with the boids forming a form of möbius strip, as can be seen in Fig. 3. Because of this, we opted to instead use a uniform bounding box.

With the first four rules implemented, we had a basic boid simulation. However, the resulting movement and behaviour of the boids didn't seem very realistic. One issue was that large groups of boids seemed to move rather erratically. We reasoned that this might be because of a lack of inclusion of physical laws in the simulation, and hence opted to try to include a rule for simulating drag force on the Boid, as well as to improve how the velocity of the boid changed as to account for inertia. These changes were based on the skeleton code provided for a lab in DD1354. Specifically, the results of the rules were changed from direct changes to the velocity of the boid, and instead treated as forces applied to an object with a mass of 1 kg. These five forces would be summed up and treated as the acceleration of the boid. The new velocity of the boid would then be equal to the sum of its old velocity and its acceleration multiplied by the past time. However, for the initial tick when the forces are applied, the boid would effectively be moved by 1.5 times the velocity given by the acceleration, as to make sure that the impulses of the boid would have an effect.

These changes greatly improved the realism of the boids movements. However, the boids behaviour remained somewhat unrealistic; the boids either tended to stay scattered, form erratic swarms vying for power, or all cramp up into a single swarm as can be seen in Fig. 4. After manually adjusting the strengths and radius of the rules, as well as increasing the bounding box, the boids started behaving as expected, with results such as the one seen in Fig. 5

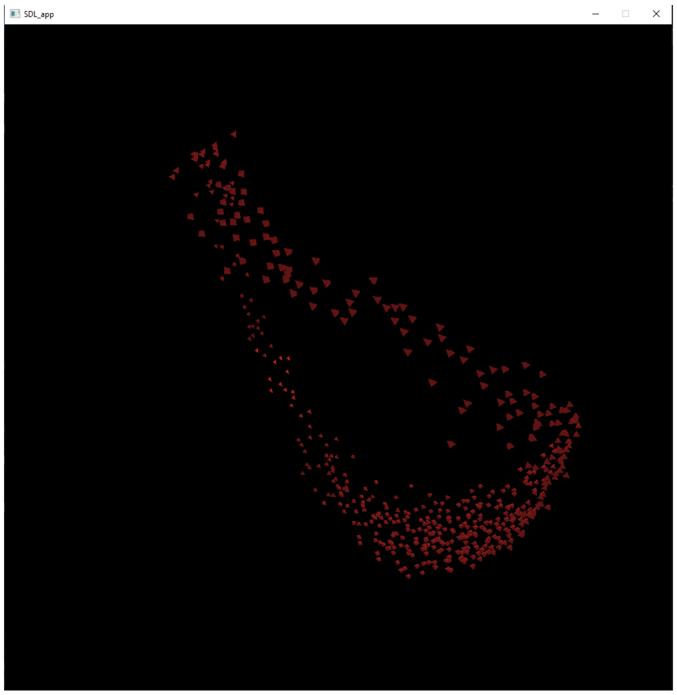

Figure 3: A flock of boids in the midst of traveling along the möbius strip of their making

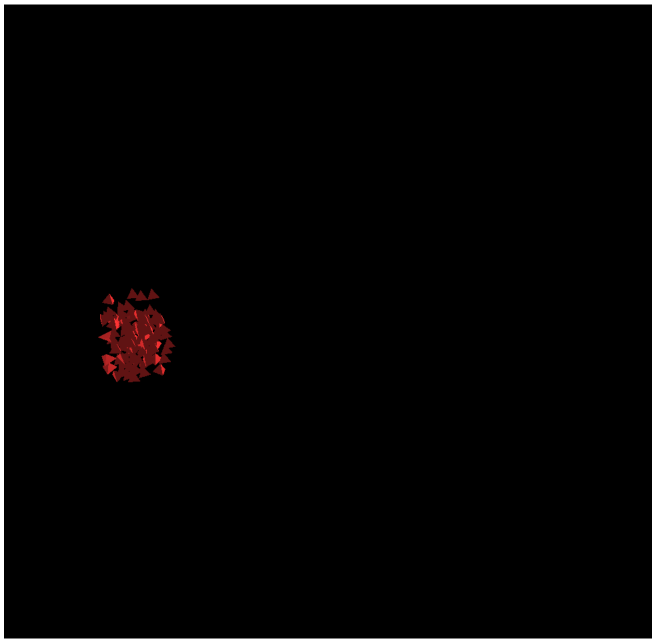

Figure 4: A cramped swarm of boids

4.3 Spatial Partitioning

In order to optimise the Boid Simulation, we choose to implement a spatial partitioning scheme based on a Grid. The choice of using a Grid based scheme over a Quadtree or Kd-tree was motivated by previous works showing that it was more stable and had a better performance when the number of boids were lower than 5000 [3]. As the Boid simulation ceased to be feasible to conduct in real-time when around 1000 boids were being rendered, this seemed appropriate.

Our spatial partitioning scheme was based on a uniform 3D-grid aligned with the boundary box of the simulation. Each grid cell was set to have side lengths equal to the largest radius out of all radius used for the three rules governing the boid behaviours. The motivation for this was that previous works had found that the grid performed the best when the cell size was the same as the stride length of the agent [1]. Whilst the choice of cell size did not correspond directly to the stride length in the simulation, it would be larger than it. Furthermore by choosing a side length equal to the largest radius, we could ensure that the boid would only be able to find relevant neighbours in the 27 cells neighbouring (or being) the cell it was in. Hence we could for each cell predetermine which lists a boid in a given cell would access, and leave pointers to it.

The grid cells were represented using a one dimensional array of vectors that could contain references to boids. To find the position of the cell a given boid was in in the 3D grid, we could simply offset the position of the boid by

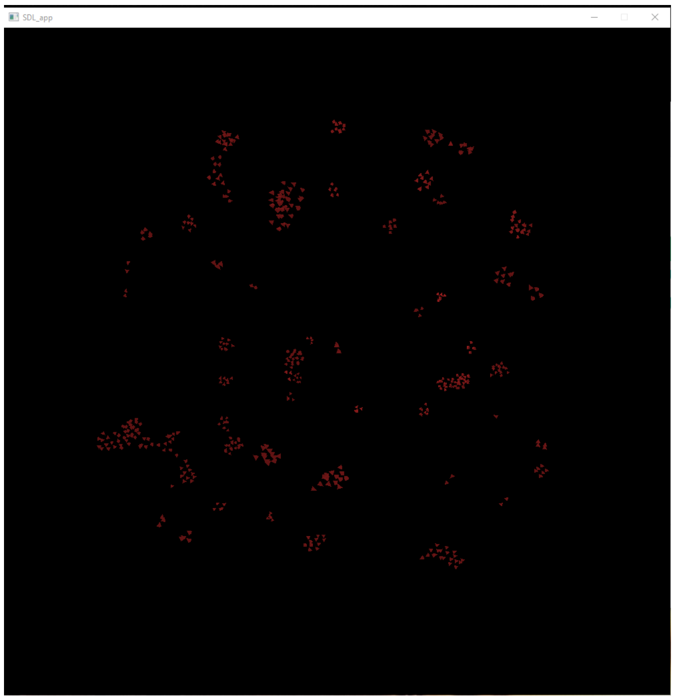

Figure 5: several flocks of boids

half the length of the bounding box edges and then scale these coordinates by the number of cells in each dimension, rounding down the coordinate to the closest cell. Thus to calculate the index of each cell, we would first get its triple consisting of its index in each dimension and then use the following formula to get the actual index of the cell in the one dimensional array

$$index = index_x + index_y * dimension + index_z * dimension^2 \tag{1}$$

By using these two representations of the cells position in the grid, we could easily pre-calculate for each cell its 26 neighbours, and create a function that would for any cell that a Boid was in return it's and it's neighbours lists of boids in the cell. For each tick of the simulation we would then clear the lists of each cell and reinsert all Boids into the new cell they end up with. Then for each boid to be simulated we would only check if any Boid within the 27 nearby cells would affect its behaviour.

5 Results

As part of our project we want to see the difference between the naive implementation of boids and the boid implementation with spatial partitioning, henceforth referred to as non-optimized version and the optimized version. We tested by running the 2 versions and differing the radius of the boundary and the number of boids in the boid simulation. Displayed is just the simulation time, the render time does increase as the number of boids increase, however, it isn't by much and we felt it wasn't relevant to the report. Worth mentioning about the variable radius is that the boundary for our simulation is a uniform cube that is dependent on our radius variable, each side is 2x our radius.

The tests were done on the same desktop computer with these hardware specifications:

- CPU: Intel Core i7 4790K
- GPU: Radeon 5700xt
- RAM: 24GB 1333Mhz

While running the two versions with their testing parameters, we noticed two particular things. When there is too large or too small of a radius, it seems that the boids stop flocking and cease to fly in the same direction. This is probably because there is too much or too little space, respectively, between the boids. We also noticed that when the simulation times became too large and the simulation becomes too slow it makes the boid once again stop flocking. This most likely is because slower simulations gives a larger dt, which leads to boids with larger strides for each frame. This probably leads them to overshoot each other.

In Table 1 we can see all data we gathered for the tests. To visualize the data we also made a number of graphs, these are Figures 6 to 11

In Figure 6 and 7 we see the times for the non-optimized and optimized version, where each line is the radius. We grouped them up like this so to more easily see that difference the radius does for the simulation speed for each frame. We see in Figure 6 that increasing the radius does very minuscule difference, but in our results there seems to be some difference especially when the number of boids increase. The difference could be that with a larger distance there is fewer boids that matter for one specific boids calculations saving a few instructions for each boid.

Figure 7 is visualized with a logarithmic vertical axis because the line representing radius was dominating so much that it was hard to see differences of the different radius which is much clearer with the vertical axis being logarithmic.

The four Figures 8, 9, 10 and 11 is the two versions in the same graph where they have the same radius, so the two versions could be compared for each radius. The differences could be more detailed if we had logarithmic axes. However, we decided not to because it was more clear this way how big the difference is between the two versions.

In this data we can see that as we increase the radius for the optimized version and that keep the same amount of boids, that the time increases and becomes larger than the ones with lower radius, however we win a lot as we increase the number of boids rendered. To note is that you most likely wouldn't use larger radius for very few boids since the space between the boids will become very large and the simulation won't be particularly interesting.

Another curiosity we can see in the data is that if we keep the same amount of boids, but increase the radius, we will most likely see that at some point the non-optimized version will be faster than the optimized version.

6 Future Work

6.1 Limitations

Their exists some limiting factors to both the performance and realism of the current simulation. One limiting factor is that the performance of the simulation is currently connected to the density of the boids, with the simulation being much slower when several boids are confined to a small bounding box than if a larger one was being used. This is an inherent problem of the simulation, as each boid will have to consider every neighbour within range regardless if its affect is minuscule on the boids behaviour. Potentially this could then be mitigated by only considering the up to k nearest neighbours and using more optimised versions of the grid scheme [5], but as the behaviours of the boids seemed rather unrealistic for dense spaces, it might not be a concern in practice.

Another limiting factor concerning the performance of the simulation is that it tends to still degrade as the number of boids increases. From our results, we have found that if more than 5000 boids were present, the simulation tended to become unfeasible to run as a realtime application. As was noted by Kratz and Luthman, grid based partitioning tends to be outperformed by both Quadtrees

Average simulation time each frame			
Number of Boids	Radius	Non-opt Time (ms)	Opt Time (ms)
500	1	22	6
1000	1	93	24
2000	1	360	99
3000	1	851	224
5000	1	2297	621
10000	1	9324	2024
500	3	22	1
1000	3	87	4
2000	3	364	12
3000	3	785	22
5000	3	2268	47
10000	3	8895	157
500	5	22	2
1000	5	88	4
2000	5	354	9
3000	5	783	14
5000	5	2174	25
10000	5	8725	70
500	10	22	7
1000	10	86	9
2000	10	360	13
3000	10	778	17
5000	10	2146	23
10000	10	8573	41

Table 1: The average over the 10 last simulation times we acquired from testing a naive implementation of boid simulation (called non-optimized version) and a boid simulation with spatial partitioning.

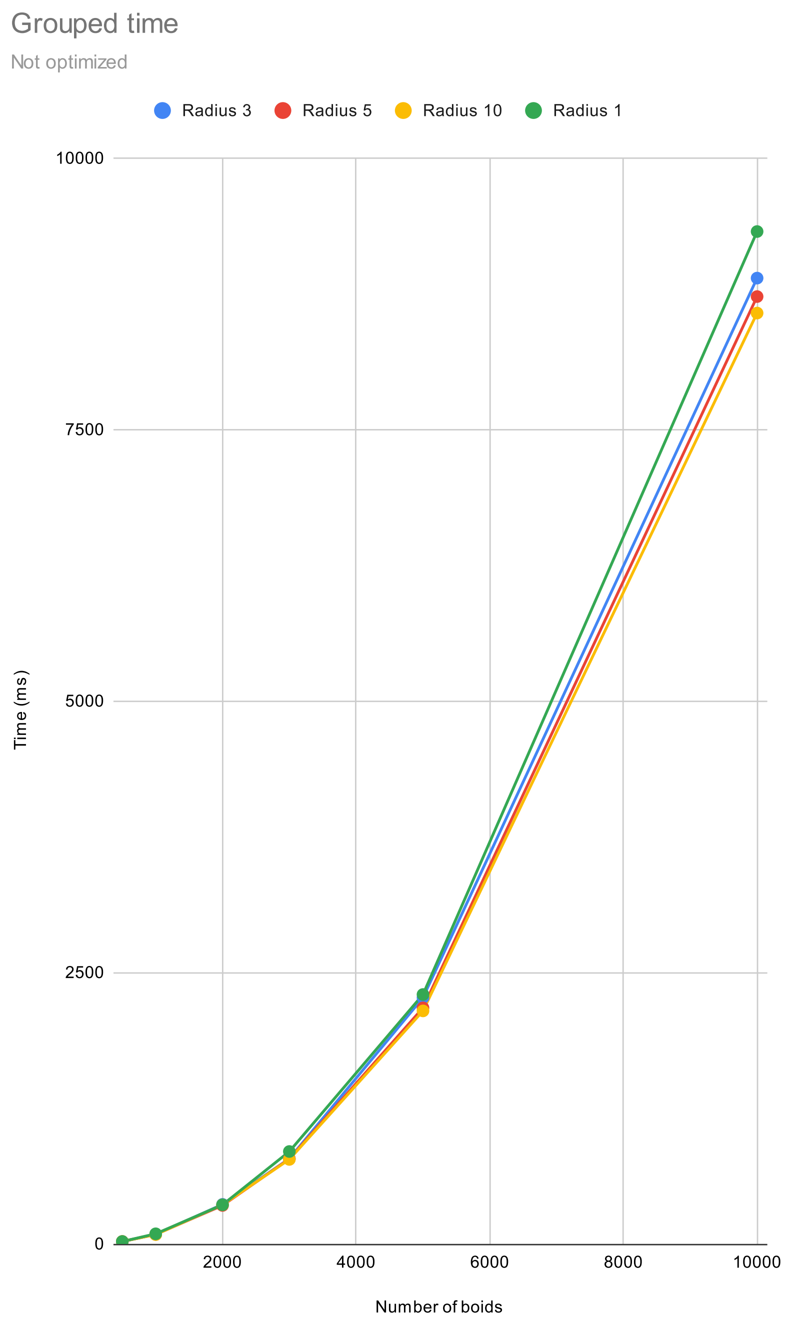

Figure 6: The average simulation times for the non-optimized version of the boid simulations. Each line is the same code running, but with a different radius describing the boundaries of the simulation.

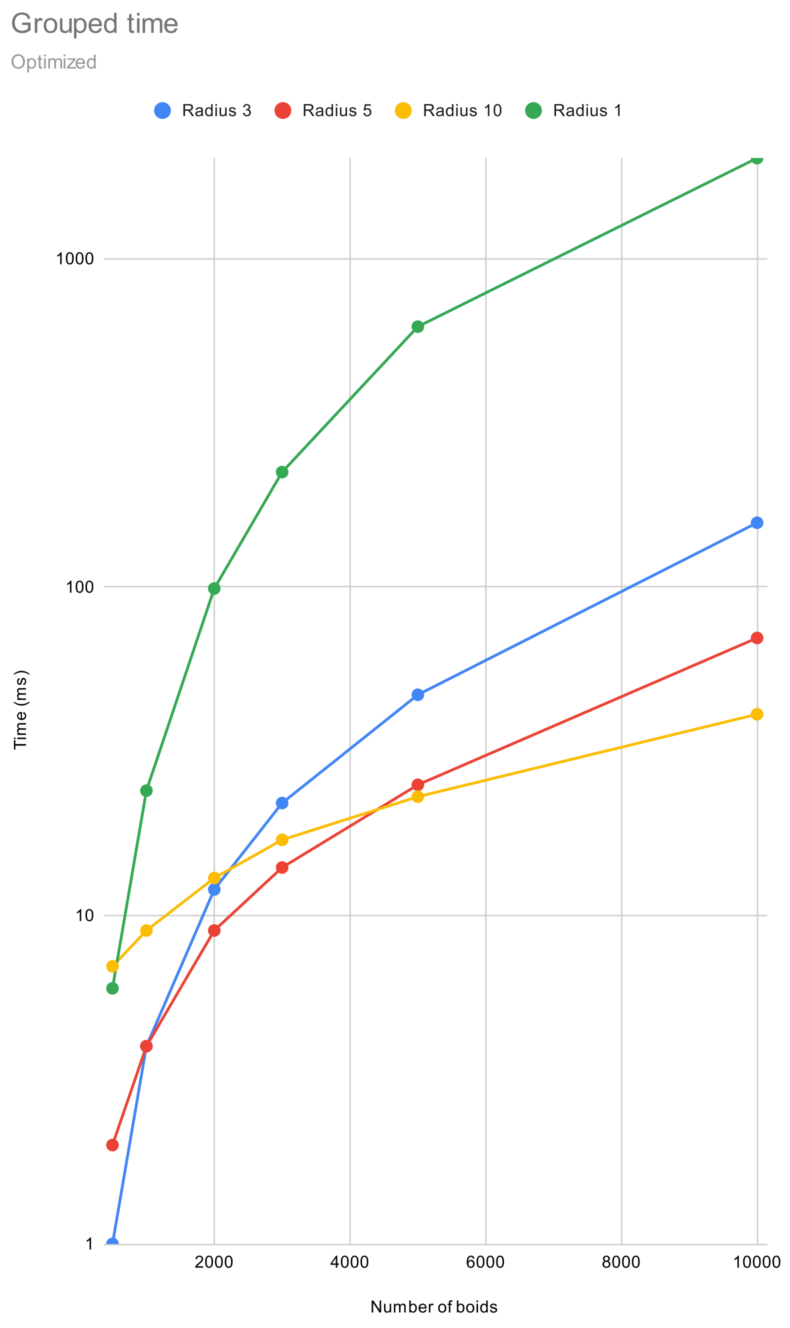

Figure 7: Optimized version average time. Each line represents a different radius describing the boundaries of the simulation. The vertical axis is logarithmic to make the differences between lines more detailed and clear.

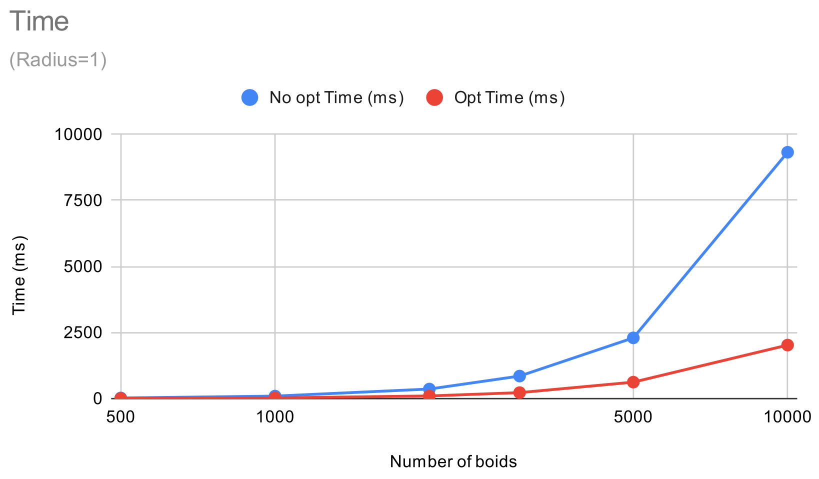

Figure 8: The average times for the optimized and non-optimized version when the radius is 1.

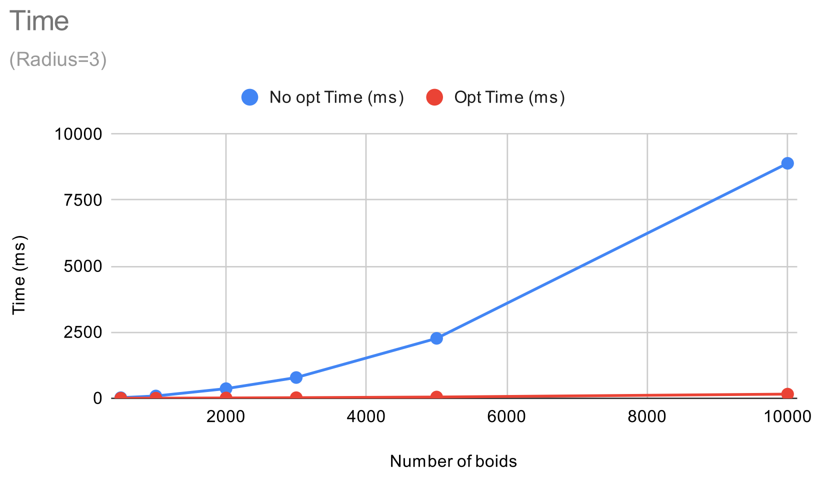

Figure 9: The average times for the optimized and non-optimized version when the radius is 3.

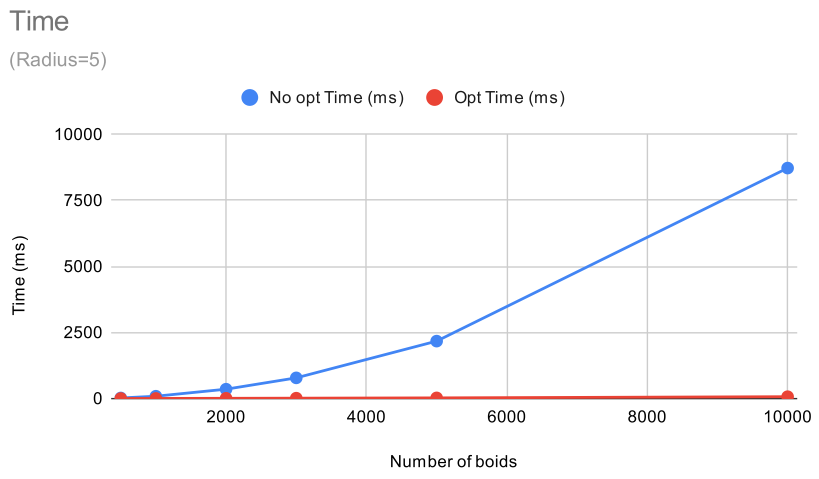

Figure 10: The average times for the optimized and non-optimized version when the radius is 5.

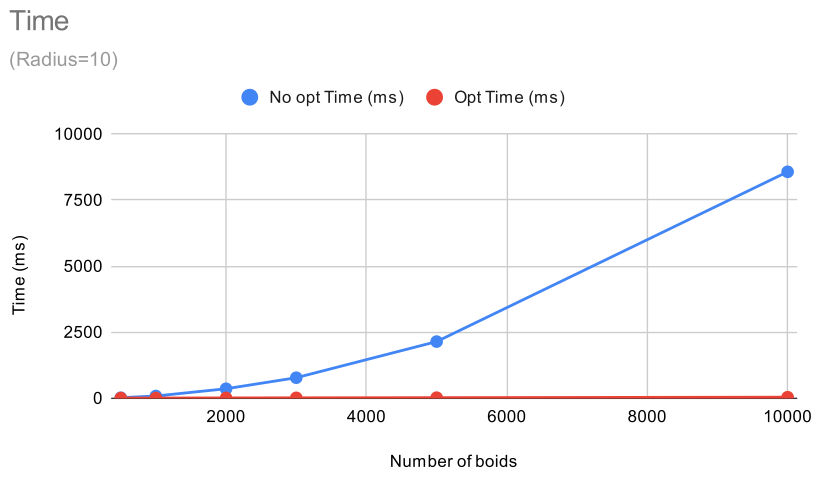

Figure 11: The average times for the optimized and non-optimized version when the radius is 10.

and Kd-trees beyond this point [3], and it might thus have been worth investigating if these schemes would have been a better choice for our application. Furthermore, as the memory needed for the grid grows as $O((\frac{l}{r})^3)$ where l is the bounding box length and r the largest radius used for the rules, memory usage becomes a concern as the boundary box grows. It might hence be worth exploring alternative partitioning schemes for this reason.

For the realism aspect of the simulation, it should noted that only a few physics aspects related to the boids movement has been taken into consideration. Furthermore, the scene has been left empty besides for the boids. As it has previously been noted that the simulation tends to stabilise, as well as not show all of its complex behaviour, if no obstacles is included [7], it might be worth extending the scene to take this into consideration.

6.2 Perceptual Study

One important way the simulation can be improved is by conducting a perceptual study to determine which parameters of the simulation would give the most realistic results. Such a perceptual study would benefit from a larger, well sampled participant pool, although depending on the resources available it might have to be limited in scope. Still, by sending out invites to as many people as possible and ensuring that participation can be done with ease, the reach of the study would be improved.

The project would probably benefit mostly from investigating how users perceive the realism of the simulation as the parameters controlling the different rules varies. Whilst it might also be interesting to study how the number of boids and the size of the bounding box affects the users perception of the simulation, this would most likely be a relatively small aspect in the study. However, if a preferred ratio of boids to bounding box size could be found, it could help inform the design of the actual study.

In order to investigate the relationship between different configurations of the rules' strengths and radius and perceived realism of the scene, and ordinal study could be conducted. For example, an interactive survey could be constructed in which videos of several different configurations of parameters (chosen with systematic increments) have been included. The participant could then be presented by an arbitrary pair of these videos and choose which one seems more realistic to them. By doing so, the survey could then dynamically sort the different configurations into a preferred order, using the participant as the comparison function for the sorting algorithm.

One advantage of using the proposed study would be that ordinal data about several different configurations could be collected, potentially providing insights into how different parameters interact to affect the overall experience of the simulation. Furthermore, the study would directly provide the most preferred configuration out of those presented. However, a disadvantage is that the individual parameters are not considered in isolation, and hence it becomes harder to draw conclusions about their individual impact. With that said, as the number of parameters are quite high and the object of concern is the realism of the overall simulation, it seems fitting to use this approach. Furthermore, it is an established method that has been used in previous works, such as Anders Steen's upcomming Master Thesis. Hence established methods for analysing the data and handling the methods drawbacks should exist.

7 Contributions and Acknowledgments

We want to thank and acknowledge the following people for making this project possible:

- Joey de Vries for his website LearnOpenGL [2] which provided us with an understanding of OpenGL's render pipeline, and a basis for our renderer
- Conrad Parker for his pseudo code for the Boid algorithm [6]
- The course personnel of DD1354 Models and Simulation for providing a skeleton reference implementation of the boid simulation which we used to improve our own

7.1 Our Individual division of labour

Throughout the project we mostly worked as a pair, with the literature study, report and implementation being done collectively. For the programming part we utilized pair programming, with each part of the implementation being done by both members.

References

[1] Bo, L., and Ramakrishnan, M. A comparative analysis of spatial partitioning methods for large-scale, real-time crowd simulation. In *WSCG 2013 Communication Papers Proceedings* (2013), V. Skala, Ed., Vaclav Skala - Union Agency.

[2] de Vries, J. Learn opengl. Website, April 2022. URL: https://learnopengl.com/.

[3] Kratz, J., and Luthman, V. Comparison of spatial partitioning data structures in crowd simulations. Master's thesis, KTH, 2021.

[4] Lantinga, S., et al. *SDL Library Documentation.* SDL, 2013, ch. 2. Graphics and Video. URL: https://www.libsdl.org/release/SDL-1.2.15/docs/html/guidevideoopengl.html.

[5] Lee, J. M., Cho, S. H., and Calvo, R. A. A fast algorithm for simulation of flocking behavior. In *2009 International IEEE Consumer Electronics Society's Games Innovations Conference* (2009), pp. 186–190.

[6] Parker, C. Boids pseudocode. Website, September 2007. URL: http://www.kfish.org/boids/pseudocode.html.

[7] Reynolds, C. W. Flocks, herds and schools: A distributed behavioral model. *SIGGRAPH Comput. Graph. 21*, 4 (aug 1987), 25–34.