



Comparison of Particle System Performance

In Object-Space and Image-Space Within a DirectX11
Implementation

Martin Ohlson

This thesis is submitted to the Faculty of Computing at Blekinge Institute of Technology in partial fulfilment of the requirements for the degree of Bachelor's Thesis in Computer Science. The thesis is equivalent to 10 weeks of full time studies.

The author declares that they are the sole author of this thesis and that they have not used any sources other than those listed in the bibliography and identified as references. They further declare that they have not submitted this thesis at any other institution to obtain a degree.

Contact Information:

Author(s):

Martin Ohlson

E-mail: maoe16@student.bth.se

University advisor: Associate Professor Veronica Sundstedt
Department of Computer Science

Faculty of Computing
Blekinge Institute of Technology
SE-371 79 Karlskrona, Sweden

Internet : www.bth.se
Phone : +46 455 38 50 00
Fax : +46 455 38 50 57

Abstract

Background. Particle systems render groups of points in space that are used to add various effects to a scene. When rendering particles, there are various advantages and disadvantages for rendering an effect in either object-space or image-space. The former uses calculations in a 3D environment, while the latter adds particles to a 2D image. There is a lack of existing research to understand the trade-off in performance and in effort when applying these two techniques.

Objectives. This thesis aims to compare the performance difference between particle systems in object-space versus image-space. The intent is to do this using a C++ DirectX11 implementation.

Methods. The primary methodology of this thesis was an implementation complemented by a performance evaluation. Two particle systems were made to render the same particle effect. The systems were run at five different particle amounts and the FPS was recorded for 30 seconds while the simulations ran.

Results. The data presented displays the average FPS recorded, with different line graphs for the various particle amounts. The results show an overall better performance of the O-system.

Conclusions. While the O-system performed better, the various techniques and optimizations available to particle systems that render in image-space would allow a vastly different result. This thesis would be well complemented with further research. It would also be furthered by comparing performance while additional geometry interacts with the particles.

Keywords: DirectX11, Graphics pipeline, Particle system, Simulation.

Acknowledgments

I am grateful to Veronica Sundstedt, my supervisor. Her guidance and feedback ensured that my focus was kept on what was important in order to accomplish the work done for this thesis.

Contents

Abstract	i
Acknowledgments	iii
1 Introduction	1
1.1 Background	1
1.2 Aim and Objectives	2
1.3 Research Question	2
1.4 Scope	3
1.5 Thesis Outline	3
2 Related Work	5
2.1 Summary of Related Works	6
3 Method	7
3.1 Implementation	8
3.1.1 O-System	8
3.1.2 I-System	8
3.2 Particle Effect	9
3.2.1 Beginning of Simulation	9
3.2.2 Middle of Simulation	10
3.2.3 End of Simulation	11
3.3 Measurements	11
3.3.1 Hardware Setup	12
3.3.2 Performance Evaluation	12
3.4 Considering the Methodology	12
4 Results and Analysis	15
4.1 Relation of the Systems	20
5 Discussion	21
5.1 Possible Optimizations	21
5.1.1 Issues of the Combined Implementation	22
5.2 Other Possible Implementations	22
5.3 Issues of the I-system.	22
5.4 Impact of the Results	23
6 Conclusions and Future Work	25

References	27
A Particle Simulation Screenshots	29

Nomenclature

CPU Central Processing Unit.

dt Delta Time.

FPS Frames Per Second.

GB Gigabyte.

Ghz Gigahertz.

GPU Graphics Processing Unit.

I – system The particle system that uses calculations in image space.

O – system The particle system that uses calculations in object space.

RAM Random-Access Memory.

UV The axis of a 2D texture that a 3D object or space has been projected onto.

VRAM Video Random-Access Memory.

This chapter introduces the research question and background related to the thesis, it also discusses the scope and outline of the thesis.

In real-time applications such as video games, one of the common visual effects used within computer graphics is particle systems. Particle systems render groups of points in space that are used to add various effects to a scene. These effects have a variety of uses, from performance-heavy water simulations and explosions to simple rendering of textures to simulate snowfall or fire. Due to the variety of effects that particles can produce [5], many different techniques exist to facilitate this. This thesis will compare the performance of two different approaches to rendering particles.

1.1 Background

When working in any rendering pipeline of a real-time application, it is essential to understand the various coordinate systems (or spaces) that the pipeline would progress through. Depending on the desired effect, particles can be handled differently and within different spaces. It is therefore relevant to understand the trade-offs these various approaches may have.

When rendering particles, there are various advantages and disadvantages for rendering an effect in either object-space or image-space. The former uses calculations in a 3D environment and comparisons between objects in that space to calculate and accurately add particles in their correct position. The latter adds particles to a 2D image, making needed calculations with depth and UV data, doing calculations pixel per pixel, often to a more discrete effect. The different approaches cannot always achieve the same effects, but there is overlap.

By using compute shaders, an implementation can move outside of traditional render pipelines. These shaders allow the use of the GPU while more abstractly allowing us to decide what space the shader works in [8].

There exists a substantial amount of studies where new approaches are explored in image-space, often in an attempt to speed up or improve on techniques done in object-space [2, 4]. Despite this, there is a lack of existing research to understand the difference in performance on a more rudimentary level. There is a gap in understanding a more direct correlation between performance in object-space versus image-space, especially when employing a particle system effect.

1.2 Aim and Objectives

The common framework of particle systems comes from the 1983 article by William Reeves [5]. For an implementation to be made, this baseline lies in the background of that work. In this thesis, we aim to use and modify this base to compare the performance difference between particle systems in object-space versus image-space. The intent is to do this using a C++ DirectX11 implementation. These two different implementations will henceforth be referred to as *O-system* and *I-system* respectively. The graphical pipelines for these systems can be seen as laid out in Figure 1.1. This Figure shows the more traditional use of shaders and a rasterizer in the O-system, and the I-system's more heavy use of compute shaders and dispatches. These pipelines will be explained more in-depth in Chapter 3.

Achieving this aim would make up for a subpar baseline when understanding the comparative benefits and trade-offs of the two techniques. This baseline would allow for more sound prioritization and decision-making when choosing what techniques to apply, with a more concrete understanding of the sacrifices and benefits the two techniques introduce.

The objectives to achieve the aim are as follows:

- Implement the common base of the particle systems: This involves the general setup of DirectX11 and implementing a common parent class for the two particle systems.
- Implement the O-system: As the most traditional implementation, the O-system will be implemented first. This includes the implementation of a sub-class and the O-system's specific shaders.
- Implement the I-system: This implementation includes a sub-class and the I-system's specific compute shaders.
- Performance Evaluation: A small implementation is added to assist in measurement, and then FPS will be measured while the particle simulations are running.

1.3 Research Question

This thesis will attempt to answer the following research question.

RQ: *"What is the performance difference when comparing a particle effect using a particle system in object-space versus image-space within a DirectX11 implementation?"*

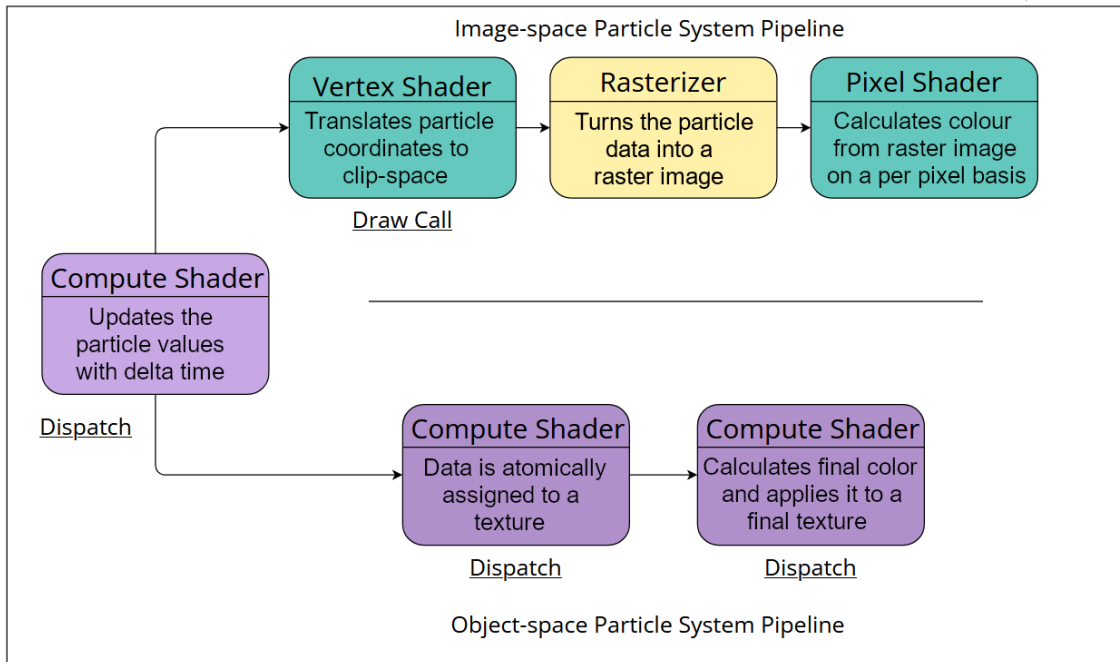


Figure 1.1: A visualization of the graphical pipelines that differ the O-system and I-system.

1.4 Scope

For this thesis, we used an implementation and performance evaluation as the scientific method. An implementation of two particle systems was created with the help of the graphics API DirectX11 and is written in C++.

The performance was evaluated by measuring the FPS of each of the particle systems. The FPS was measured while rendering a specific particle effect created by particles given a starting velocity and then put under the effect of a simulated force.

Implementing a particle system that renders in image-space comes with many potential approaches in and of itself, as many smaller techniques and methods can be used. Chapter 3 will further specify the choices that further limit the application of this thesis.

1.5 Thesis Outline

This thesis contains the following Chapters: Introduction, Related Work, Method, Results and Analysis, Discussion, and Conclusions and Future Work.

1. Chapter 1 gives an overview of the thesis and the background. It will also discuss the scope of the thesis.
2. Chapter 2 goes over related work and summarizes several relevant sources for this thesis.

3. Chapter 3 goes into the details of the particle systems and their performance evaluation. It also discusses the choice of methodology and the motivation for it.
4. Chapter 4 presents the results of the measurements. These results are mainly presented through the use of various line graphs.
5. Chapter 5 engages with the results from the previous chapter. It goes through and discusses the various details of the implementations.
6. Chapter 6 concludes the thesis and gives final answers. It also considers potential future work.

Reeves's article on modeling fuzzy objects [5] is one of the earliest explorations of particle systems as they more or less appear today. While modern hardware and improved rendering techniques have increased computational speeds drastically since the original article was published, the model remains highly relevant. Many techniques have been developed atop this model, and is still often how particle systems are taught [1]. This thesis presents an additional understanding of the possible variations of the model's rendering stage.

On their website, Turitzin has an article that covers their work to better performance for the engine of a Virtual Reality (VR) application they developed [9]. Their engine uses OpenGL 4.5 as its rendering backend, and the article discusses the concerns and trade-offs that lead them to render particles in image-space using compute shaders instead of a rasterizing pipeline. This article is a relevant example of the individual work that image-space solutions are typically applied for and helped in understanding the issues presented by particle systems that render in image-space. Unlike this thesis, Turitzin's work also tests and discusses some variations in hardware and goes more in-depth into the implementations of techniques and their variations.

The work of Falk et al. [3] contains a large amount of work dedicated to the visualization of particle data. Much of the presented work is highly technique-focused and presents various ways of achieving the visual effect one would require. Many of these techniques are explained with advanced uses of image-space and object space. This work is more advanced than the scope of this thesis would require, but a more in-depth understanding of this work could have contributed to a different final implementation that would have yielded different results.

The work of Musawir Shah [7] is an adjacent endeavor to the area of this thesis. Three algorithms are developed to allow realistic rendering to be done with image-space calculations. The three techniques they develop are to help with grass rendering, rendering caustics, and subsurface scattering for rendering of translucent objects. These are used for the context of allowing a realistic representation of a scene. In theory, this is the sort of work that may be assisted by the results this thesis would provide.

2.1 Summary of Related Works

This thesis presents an addition to understanding the progressing work of Reeves's article [5]. Meanwhile, Turitzin's work [9] stands as a similar comparison of implementations of particle systems in object-space or image-space. The work of Falk et al. [3] and Musawir Shah [7] dive into a collection of techniques to achieve visual effects, with many of them focusing on using image-space. This thesis would add a base that allows further understanding of when to use either object-space or image-space for implementations of particle systems. Building up this baseline of knowledge would cover a gap not present within the related work. The use of DirectX11 for such a comparison would cover a gap that is especially wanting.

This chapter discusses the choice of research method and the details of the particle systems and their implementation. It will also discuss the performance evaluation.

The primary methodology we used for this thesis was an implementation, that was complemented with a performance evaluation. This research method allows a simple translation of our research question. We can code the implementation with the help of DirectX11, and to keep the two solutions parallel, they can be programmed to be similar until the divide of the pipelines. As answering the research question only required a direct comparison of performance, a hypothesis seemed superfluous for this thesis.

A case study would have required software that allowed more specific access to a DirectX11 particle system that could switch between an image-space and object-space implementation. When comparing two systems meant to provide a similar particle effect, an implementation allowed more direct control over the process.

A systematic literature review could have been made, however, it would likely have required gathering many different papers that discussed very specific visual effects being implemented with the use of both image-space and object-space. As that is typically where the research of image-space implementations occurs. Shah's dissertation [7] is an example of a desired effect being researched. The results of a systematic literature review would likely have been difficult to generalize for the purpose of this thesis.

3.1 Implementation

A DirectX11 implementation was made, with two particle-system subclasses built from the same parent class. The difference between the two systems is the approach of calculations used for rendering, with the differing calculations being done in object-space or image-space respectively.

3.1.1 O-System

The main difference in implementation between the O-system and the I-system is that the O-system uses a more traditional render pipeline. Figure 1.1 on page 3 shows a basic rendition of the relationship, but more specifics will be discussed in this chapter.

1. **Compute Shader (Particle Updates).** A dispatch of all particles is set to run commands from a compute shader for each particle. The compute shader then calculates the change in value for each particle and progresses them accordingly for the frame. In both implementations, this shader also handles shading by calculating how large the particles would be on screen. Particles further from the camera become darker and those closer become brighter.
2. **Vertex Shader.** The vertex shader handles the transformation of geometry between spaces. It runs once for each vertex and outputs primitives. In the case of this particle system, these primitives are points.
3. **Rasterizer.** The rasterizing step is a non-programmable stage of the pipeline that is still important to address. This step transforms primitives into pixels. The rasterizer calculates the same way it handles triangles, but it identifies the point coordinate as a pixel-wide square [10].
4. **Pixel Shader.** The pixel shader often handles the lighting and post-processing of the fragments from the rasterization process. Since the shading is handled by the compute shader, the pixel shader has little additional function here.

3.1.2 I-System

Instead of a traditional render pipeline, this particle system uses three dispatch calls of compute shaders to work with a set amount of threads and accomplish the task of rendering the particle effect.

1. **Compute Shader (Particle Updates).** This step is the same as the first step of the O-system. Particle values are calculated and progressed for the frame.
2. **Compute Shader (Color Assignment).** A dispatch of all particles is set to run commands from a compute shader for each particle. The particles' UV coordinates are calculated and the data is applied to a texture three times as wide as the program screen. This is done using atomic operations, which make sure no writing issues occur. These operations essentially lock other threads

from accessing the shared memory we are writing to, unlocking it when the operation is complete [6]. The reason for using these operations is derived from Turitzin’s article [9], where they state:

“In order to render with a compute shader, we need either to sort particles by output pixel (or screenspace tile) so that each output pixel is written to by at most one thread, OR we need to update output pixels atomically so that writes from multiple threads don’t conflict.”

Color values are decided by three float values. As there are no interlocked functions that function for our float values in the shader version 5.0 of DirectX11, these values need to be translated to integers. The data is atomically input and then translated back to float values in the next dispatch of a compute shader. For this implementation, we decided to make use of three atomic operations per particle. This is computationally expensive, especially as particle positions overlap. This issue is essential to the outcome of the results and is therefore discussed heavily in Chapter 5.

3. **Compute Shader (Final Texture).** A dispatch of all pixels within the program window is set to run commands from a compute shader for each pixel. The shader receives each color from the texture, culling if no color is found. The shader then applies the color to the final texture that is to be rendered.

3.2 Particle Effect

The particle systems render a particle effect, programmed to be similar in appearance and behavior. As the programs run, each particle moves and changes in real-time as specified by the physics calculations performed within the first dispatch of the render pipeline. These calculations are discussed further in Section 3.2.2.

The color of the particles was chosen to be a pink hue that can be described by the Hex code #b31ab3. Turitzin’s article [9] mentions that a hierarchical depth buffer needed to be built, as depth testing severely lowered performance otherwise. Due to the specific shading that was to be applied to both the O-system and I-system, reducing the number of interlocked values made it challenging to apply shading while the depth issue remained. The final choice was to exclude depth testing in the I-system, instead, the shading was applied to be the brightest colors, as that would be the color of the particle nearest to the camera.

3.2.1 Beginning of Simulation

There are many ways to create a distinct and visually impactful effect, in the case of this thesis, the particles were randomly distributed to eight different spots, that were set up like the corners of a cube with its center in the origin. They were then given a velocity in a random direction.

Parameters	Pseudo Code Symbol	Value
Particle Parameters		
Particle Starting Positions	pPos	X: $\approx \pm 15.0f$ Y: $\approx \pm 15.0f$ Z: $\approx \pm 15.0f$
Physics Parameters		
Maximum Velocity	mV	0.022f
Force Modifier towards Origin	f	0.00001f
Camera Parameters		
Camera Position		X: 35.0f, Y: 40.0f, Z: -62.0f
Camera Forward Vector		X: 1.3f, Y: -0.8f, Z: 1.0f
Camera Up vector		X: 0.0f, Y: 1.0f, Z: 0.0f

Table 3.1: Table showing the parameters from various elements of the particle systems.

3.2.2 Middle of Simulation

Table 3.1 shows the values of various parameters that were set during the measuring of FPS during the simulations. Some of these are assigned symbols as they are used as variables in the pseudo-code presented below.

The pseudo-code below is of the physics calculations within the first compute shader, that runs once for each particle.

1. A vector is created from the particle's position towards the origin.
`Vector = (0.0f, 0.0f, 0.0f) - pPos;`
2. The vector is normalized and multiplied with the force, creating another vector of the force.
`Normalized Vector = normalize(vec);`
`Force Vector = (Normalized Vector) * f;`
3. The position and velocity of the particle is advanced.
`Particle Position += Particle Velocity * dt;`
`Particle Velocity += Force Vector * dt;`
4. If the velocity is higher than mV, it is clamped to have the length of mV.
`if (length(Particle Velocity) > mV)`
`Particles Velocity = normalize(Particle Velocity) * mV;`

This effect appears as eight spherical expansions that then contract towards the origin. The effect continues as several layers of particles progress between various rounded shapes as they begin to orbit the origin. The progression of the particle effect for both systems can be seen in figure 3.1.

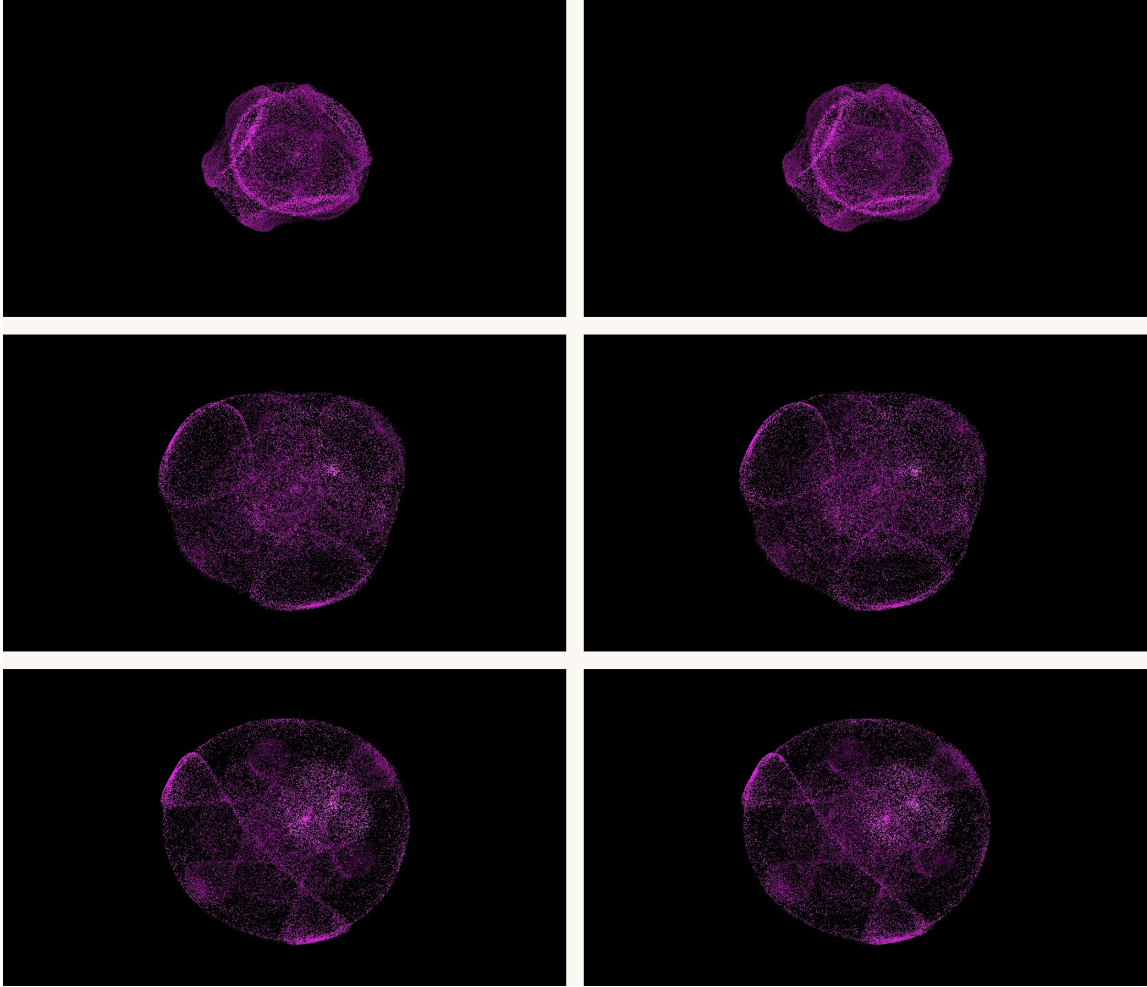


Figure 3.1: Screenshots taken 7, 17 and 23 seconds into particle simulations of 65536 particles. The O-system is to the left, and the I-system to the right.

3.2.3 End of Simulation

Around one to two minutes after the beginning of the simulation, the effect becomes more circular, as the particles begin to find a more consistent orbit around the origin.

3.3 Measurements

During the simulation, the only metric we record is FPS. Measuring frame rate establishes the time it takes for frames to be rendered by the particle systems. As each frame is a single loop of all particle calculations, we get an overview of which of the particle systems spends the most time on the calculations. There are additional metrics that could give insight into the differences between the particle systems. But to limit the scope of this thesis, the only metric we will use is FPS. This is discussed further in section 3.4.

The same simulations were executed with both systems. Five different particle amounts were used, and then the FPS was recorded three different times for each amount. This adds up to a total of 30 simulations being run.

3.3.1 Hardware Setup

The simulations were run and the FPS was measured on a computer with the following specifications:

- **CPU.** Intel Core i9-139000 CPU 2.00 GHz
- **GPU.** Geforce RTX 3080, 10 GB VRAM, 16 GB shared RAM
- **RAM.** 32.0 GB
- **OS.** Windows 10 Education

3.3.2 Performance Evaluation

The FPS was measured while running the simulations with varying particle numbers. This was done to establish any difference in performance behavior as the particle numbers increased. The amounts chosen are, from lowest to highest: $2048(2^{11})$, $65536(2^{16})$, $524288(2^{19})$, $2097152(2^{21})$, and $8388608(2^{23})$. These amounts are powers of two and were used to have a structure in the various particle amounts. The smallest number of 2048 was chosen since it was the lowest power of two which made the collection of rendered particles more cohesively visible. The largest number of 8388608 was chosen as it was the first power of two that gave an FPS consistently below ten. Any larger exponent would make the simulation difficult to work with, and the movement of the particles would also be difficult to make out at such low ranges of FPS. The values in between are additional exponents of two that would give visibly different results during the simulations.

The FPS of the particle simulations was measured for 30 seconds per particle amount.

3.4 Considering the Methodology

This section will discuss the methodology and some of the choices made for the implementation. It will also discuss some of the working conditions and what additions could have been made to increase validity, reliability, and generalizability.

During the work of this thesis, we were provided with certain hardware to use when we were to measure the FPS of the simulations. If we consider threats to validity, It is possible that the hardware used was unknowingly faulty in some way. Running the simulations on several different computers would therefore have helped validate the results. Running the simulations on different operating systems and GPUs would also have helped ensure the generalizability of the results.

Due to the hardware available for use, there is a possibility that background programs running on the hardware may have demanded more or less performance during certain simulations, which would likely affect the final results. Multiple simulations were measured to offset this issue, but additional measurements would further ensure that the impact would be diminished.

To further assess the performance of the O-system and I-system, additional metrics may have contributed. It would for example be helpful to locate where the

particle systems encounter bottlenecks during simulation, as that could give insight into possible optimizations and differences between the systems.

It should be mentioned that simulations were not performed with other geometry. Including other geometry would have necessitated depth calculations that would have further complicated the I-system, and likely affected the overall result. It is not uncommon to render particle effects with other geometry, so this addition could have increased reliability.

Another area to consider is the particle effect. If one applied more varied appearances and behavior to the particle effect, one could then measure the FPS of additional simulations to make the results more generally applicable.

Chapter 4

Results and Analysis

Figures 4.1 to 4.5 display the results from the measurements taken for the simulations of each varying particle amount. The Average FPS is displayed on the Y-axis, while the simulation time is displayed in seconds on the X-axis. The variation of the systems' FPS during the simulations is represented by a blue line for the O-system and by a green line for the I-system.

The first graph presented displays the measurements from the simulations of the lowest particle amount. The particle amounts then increase for each graph presented. Due to the drastic changes in particle numbers, the ranges of FPS shown also change drastically between the graphs.

This chapter also contains two additional line graphs. Figure 4.6 Displays the average FPS recorded at all different particle amounts for the O-system. Figure 4.7. Displays the average FPS recorded at all different particle amounts for the I-system. These graphs display the average FPS on a logarithmic scale on the Y-axis. We use this scale to allow a simple visualization of the FPS, as there is a large difference in FPS when simulating with different particle amounts.

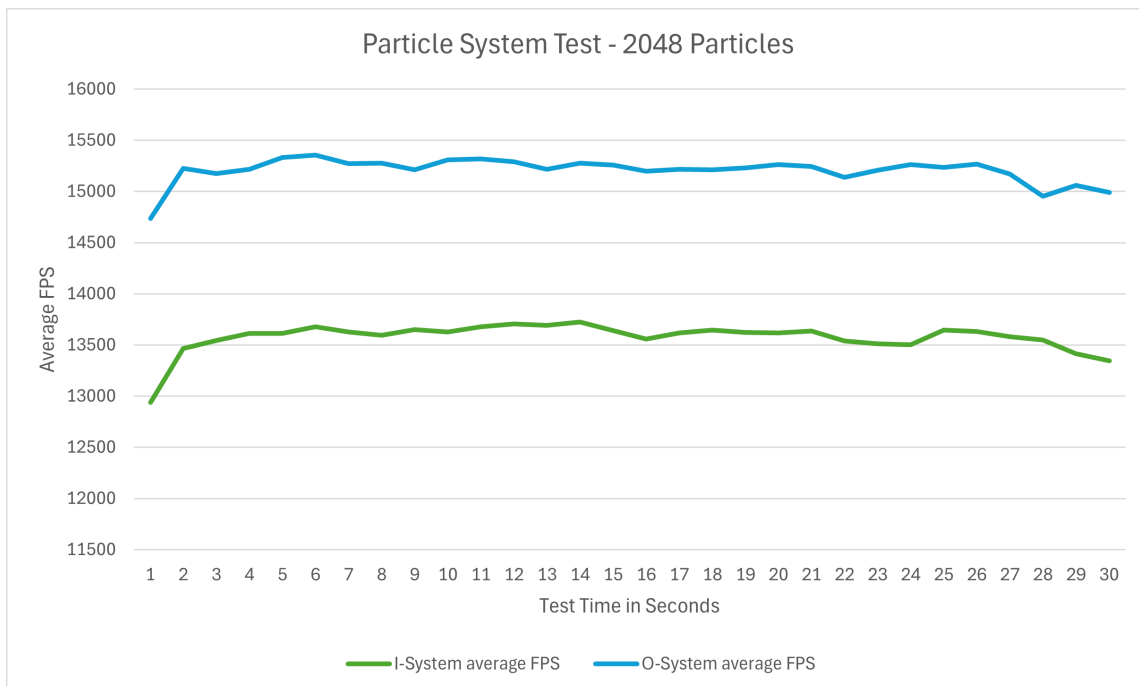


Figure 4.1: Graph showing the average FPS during the simulation period for particle amount 2048.

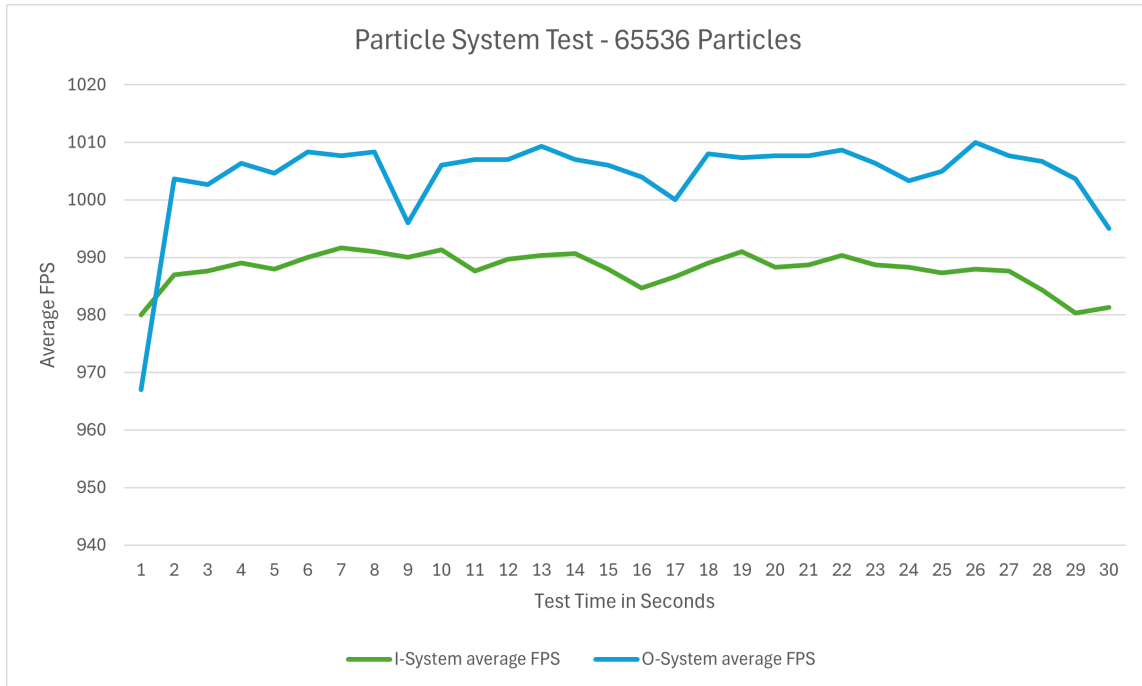


Figure 4.2: Graph showing the average FPS during the simulation period for particle amount 65536.

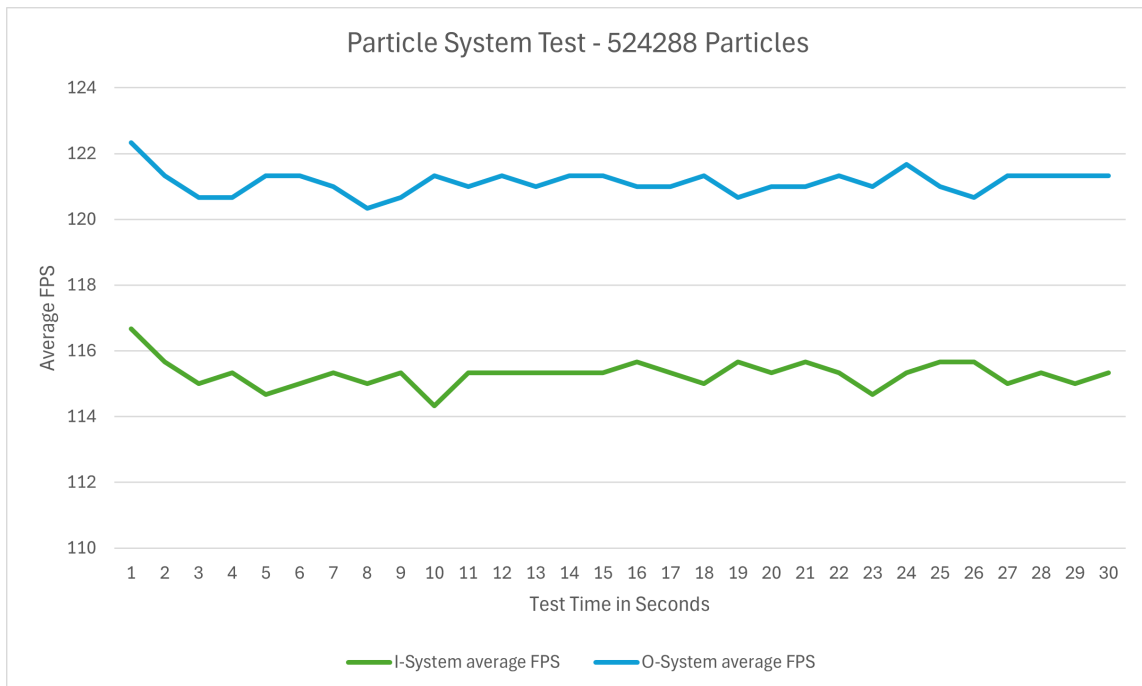


Figure 4.3: Graph showing the average FPS during the simulation period for particle amount 524288.

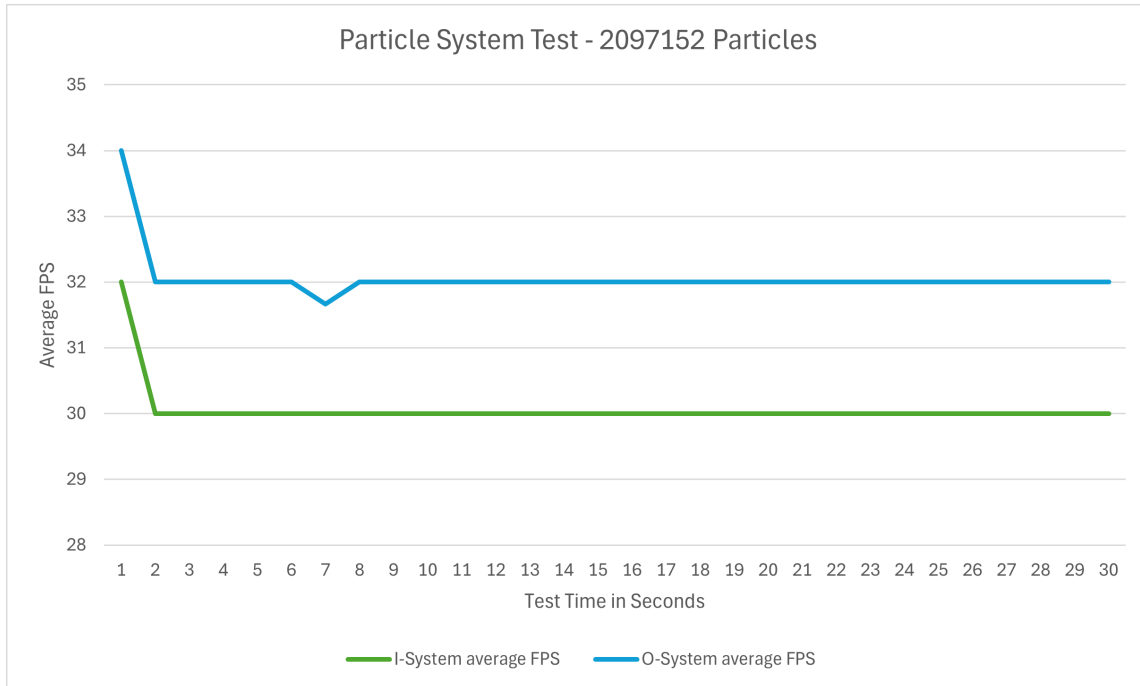


Figure 4.4: Graph showing the average FPS during the simulation period for particle amount 2097152.

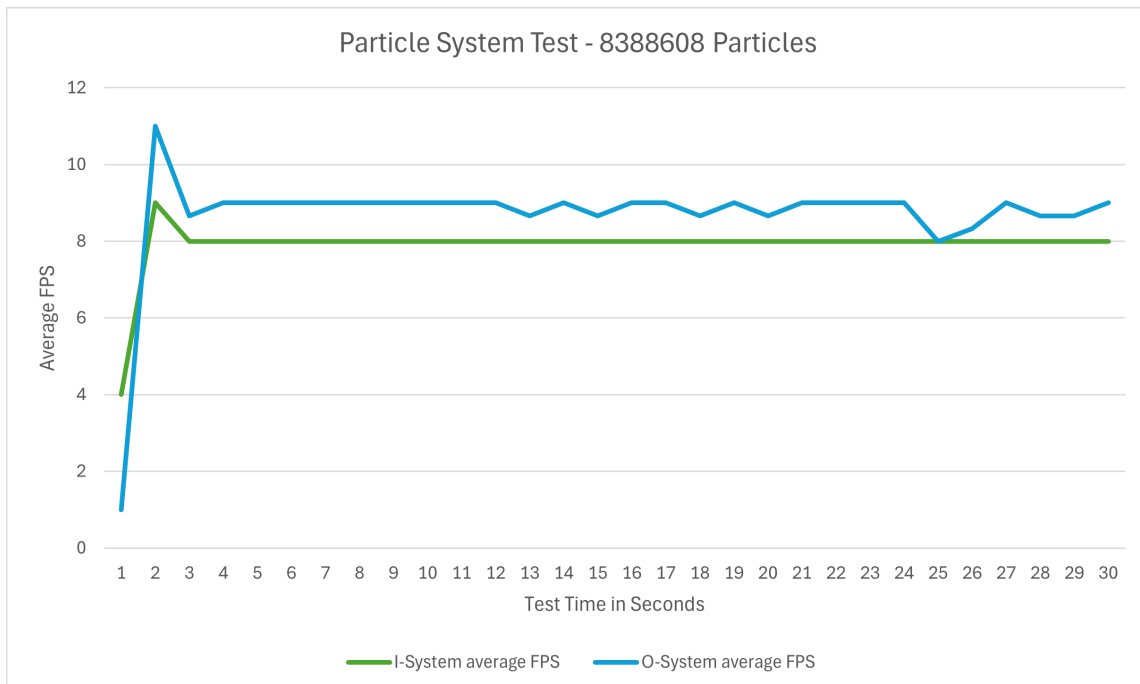


Figure 4.5: Graph showing the average FPS during the simulation period for particle amount 8388608.

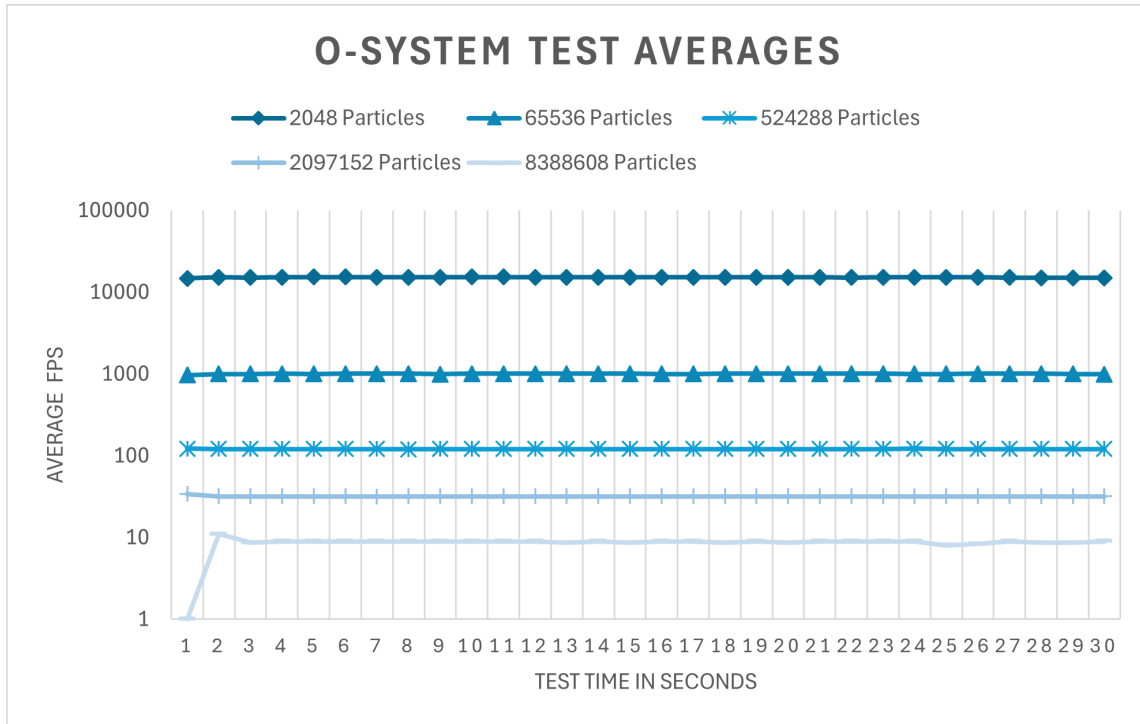


Figure 4.6: Graph showing the Average values of the O-system simulations.

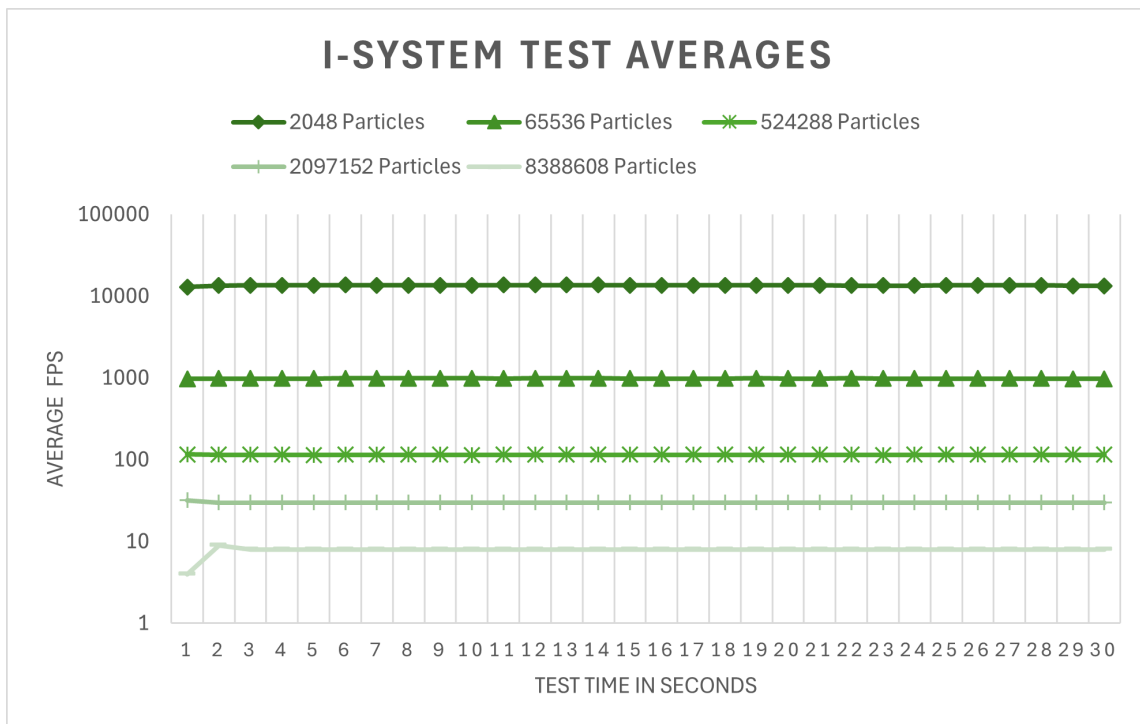


Figure 4.7: Graph showing the Average values of the I-system simulations.

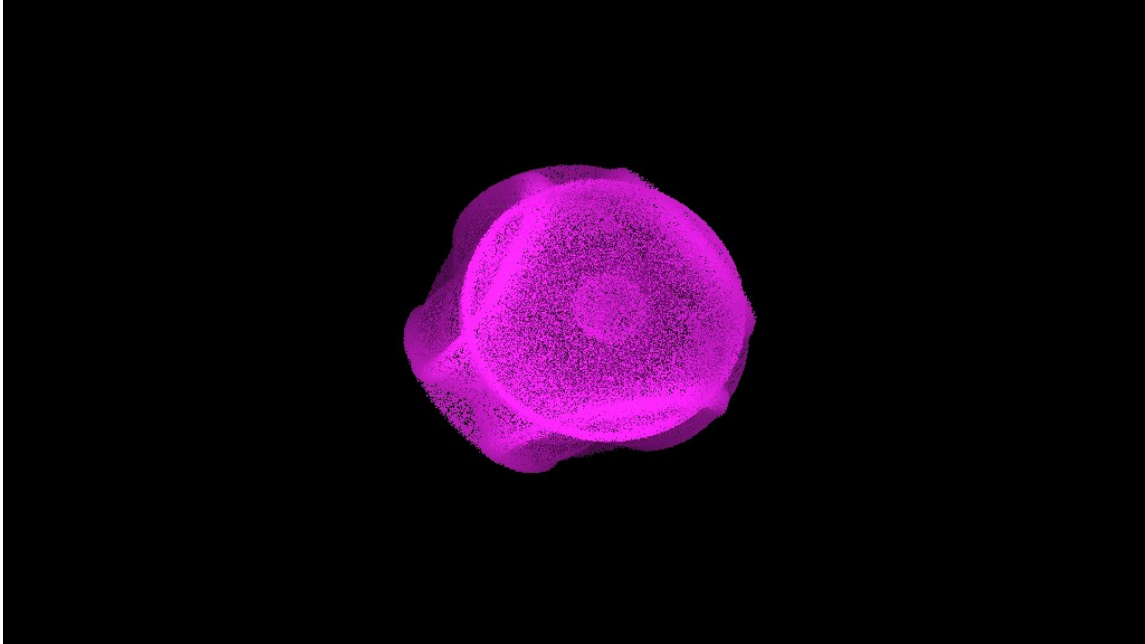


Figure 4.8: Screenshot made after 7 seconds during a simulation of the I-system with 524288 particles.

Figure 4.8 shows one of the screenshots taken of the particle system during its simulation. Screenshots of the particle simulations taken at 7, 17, and 23 seconds for each particle amount can be found in Appendix A.

4.1 Relation of the Systems

Many of the line graphs show that the simulations begin at an FPS that is later rarely reached again, regardless if this FPS was lower or higher than the average.

Table 4.1 shows the overall average FPS and standard deviation of the systems at the different particle amounts. The table also displays what percentage the I-system's average FPS is of the O-system's. We see that the I-system's overall FPS is lower at all different particle amounts. As the particle amounts increase, the I-system's relation lowers steadily. The outlier is the simulation of the lowest amount of 2048 particles, as it has the lowest relation. It is however important to keep in mind that as the particle amount increases, the FPS is lowered drastically. At the highest particle amount, the loss of a single frame would give a noticeable lowering of the relation percentage, while the same loss would barely be reflected in the relation of the lowest particle amount.

Particle Amounts	System		Test Data	System		Relation (%)
	O-System			I-System		
	avg. FPS	Standard Deviation		avg. FPS	Standard Deviation	
2048	15203	≈126	13573	≈146	89.3	
65536	1004	≈8	987	≈3	98.4	
524288	121	≈0.4	115	≈0.4	95.2	
2097152	32	≈0.4	30	≈0.4	93.8	
8388608	9	≈1.5	8	≈0.8	91.2	

Table 4.1: Table showing the overall average FPS for each system, as well as the standard deviation. Because FPS is measured in whole integers, the standard deviations are approximates. The Relation is what percentage the average FPS of the I-system is of the average FPS of the O-system.

This chapter will discuss the results. it will also discuss possible optimizations and other choices that could have been made for the implementations. There will also be a section that will muse on the issues of implementing the I-system.

The research question of this thesis was:

"What is the performance difference when comparing a particle effect using a particle system in object-space versus image-space within a DirectX11 implementation?".

The data presented in 4.1 shows a clear pattern of the O-system reaching better performance at all different particle amounts. By looking at the relation, we also see that overall, the O-system's performance compared to the I-system was better as particle amounts increased.

It should still be considered that the difference of a single frame among the results with lower ranges of FPS would be more drastic. As FPS is measured in whole integers, when the simulations of higher particle amounts result in FPS below 30, a single frame or two shows a larger impact.

It is unclear why there is a drastic change in FPS at the beginning of most simulations. If we were to theorize, it may be related to the hardware re-prioritizing resource distribution. The particle simulations began with the click of a button and not as soon as the program window was created. As the program suddenly demands more resources, it takes a second or two for the hardware to fully adjust, which is then shown in our early measurements of FPS.

5.1 Possible Optimizations

The results of this thesis are not that surprising when the specifics of the implementation are accounted for.

Although the O-system overall performed better, reducing the number of atomic operations could have made the I-system outperform the O-system. Different solutions could be used to address the I-system's problem of rendering particles atop each other, either by sorting or using various atomic functions. Depending on the result wanted from the system, regarding for example shading or interactions with other geometry, the necessary work of implementing a particle system in image-space can change drastically. Doing more research into the different methods and models of shading that could be used in for the two rendering techniques may allow a more thorough comparison.

There are workarounds for some of these issues. For example, as Turitzin discusses in their article [9], it is possible to reduce the number of atomic operations. To do

this, one can reduce the precision of color values and transfer-data. The bits are then split up or compressed into fewer variables before atomic operations are used. One could also use a plugin to supply the program with an interlocked function that allows a higher byte amount. Such additional functionality could greatly affect the results.

5.1.1 Issues of the Combined Implementation

The implementations of the particle systems ran into issues when optimizations were considered. The structure of the systems was built up to be similar until the pipelines began to divide. When an optimization for one of the systems was to be implemented in the shared base of the systems, we realized that this would cause operations for both systems that were unnecessary for one or the other. These optimizations were therefore dismissed to keep the implementation fair to both systems. Since this meant that some simple optimizations were not implemented, the final result may have been affected by this exclusion. This means that there was potential for better performance that could have swayed the results at least slightly.

5.2 Other Possible Implementations

This section discusses the potential alterations or other implementations that could have been made for this thesis.

We did not run our simulations with additional geometry. Including additional geometry would have necessitated depth calculations that would have further complicated the I-system, but would likely have affected the overall result.

In Reeves's article on particle systems [5], the particles are continuously emitted and killed when they reach the end of their life for whatever reason. particles are killed off and then reused within the system, often used to cause a repeating effect. We excluded this functionality in our particle system implementations. As our particle effect does not kill particles, but instead continuously moves for the duration of the simulation.

5.3 Issues of the I-system.

When compared to the traditional pipeline used by the O-system, the implementation of a particle system through the use of image-space can easily become a more specific endeavor. As one attempts to implement more specific and optimized effects, the abstract space of the compute shader encourages the use of more specialized solutions, which can easily escalate the complexity of an implementation.

To clarify this issue, let us say let us consider a theoretical addition to the I-system: The existing structure has likely been built to render the particles in a certain way using certain parameters. In other words, the code is practically optimized for its purpose. As the effort is more independent due to not being built upon the more general code of a traditional render pipeline, an addition to this existing structure may require the code structure to propagate. if the purpose of our I-system is to render only a specific effect. Propagating will be limited, as we do not wish to

implement unnecessary code that hampers performance. Using compute shaders to render particles in image-space allows freedom in implementation, but if performance is to stay low, the propagation will be limited. As such, the implementation can easily become specialized.

The extra effort required for an image-space particle system allows more specific results, and the use of compute shaders and their ability to work in more abstract spaces demand specific solutions. This tension made our implementation of the I-system require more work to implement than the O-system.

One could claim that the traditional render pipeline of the O-system allows general use where performance is held back by its generically applicable nature. Falk's book [3] is overall a large set of techniques that can be implemented if needed. Musawir's work [7] are other techniques developed for a specific purpose. Some works also attempt to use image-space for better results in quality as well, such as Falk's work to visualize volumes of dynamic particle simulations [2]. Overall, existing research that is concerned with the uses of image-space is focused on individual techniques meant to reach very specific results. This subsequently leads to research that is often extensive and complex, but not often generic, at least not in our context of comparison.

5.4 Impact of the Results

The results of this thesis will likely not have a grand impact on society, but as the usage of image-space becomes more complicated, this thesis may contribute to the understanding of what should be considered for such an implementation. The choice of using image-space to render particles may become more common, and a more streamlined understanding will allow proper choices that would help to reach the desired result without unnecessary workload. Additionally, understanding what type of particle system demands more resources would allow informed choices when considering sustainability. To understand the difference between performance in object-space versus image-space on a more rudimentary level, more research is still required.

Chapter 6

Conclusions and Future Work

Between our two particle system implementations, the O-system performed better, but another implementation could easily see vastly different results. Implementing the particle systems with a combined base led to issues when optimizing the systems, and issues with shading led to the use of additional atomic operations that slowed the I-system simulations further. When using image-space to render particles, there are many various techniques available. Depending on the effect wished for and the level of work put into a particle system that render using image-space, it may become specialized and optimized for its purpose.

The work of this thesis would be well complemented by more research done into the various uses of image-space and techniques performed within it, and a systematic literature review may be good for this. The comparisons of performance could also be expanded with more variations of the techniques that are used in the I-system. It would also be good to compare the performance of the systems while interacting with other geometry, as that is a common application of particles.

References

- [1] J. de Vries, “Coordinate systems,” 2014, <https://learnopengl.com/Getting-started/Coordinate-Systems>, accessed 2023-10-04.
- [2] M. Falk, S. Grottel, and T. Ertl, “Interactive image-space volume visualization for dynamic particle simulations,” in *The Swedish Chapter of Eurographics*, 2010. [Online]. Available: <https://api.semanticscholar.org/CorpusID:1369385>
- [3] M. Falk, S. Grottel, M. Krone, and G. Reina, *Interactive GPU-based Visualization of Large Dynamic Particle Data*, 1st ed. Springer, 2022.
- [4] N. Kåvemmark and S. Miloradovic, “Particle systems: A comparison between octree-based and screen space particle collision,” Blekinge Institute of Technology, Karlskrona, Sweden, Tech. Rep., 2018.
- [5] W. T. Reeves, “Particle systems—a technique for modeling a class of fuzzy objects,” *ACM Trans. Graph.*, vol. 2, no. 2, p. 91–108, apr 1983. [Online]. Available: <https://doi.org/10.1145/357318.357320>
- [6] M. Rouse, “Atomic operation,” 2011, <https://www.techopedia.com/definition/3466/atomic-operation>, accessed 2023-05-05.
- [7] M. Shah, “Image-space approach to real-time realistic rendering,” Ph.D. dissertation, University of Central Florida, 2007, <https://stars.library.ucf.edu/etd/3341/>.
- [8] J. Sorgenfrei, “Compute shaders introduction,” 2022, <https://learnopengl.com/Guest-Articles/2022/Compute-Shaders/Introduction>, accessed 2023-05-05.
- [9] M. Turitzin, “Rendering particles with compute shaders,” 2020, <https://miketuritzin.com/post/rendering-particles-with-compute-shaders/>, accessed 2023-04-30.
- [10] S. White, D. Batchelor, D. Coulter, M. Jacobs, and M. Satran, “Rasterizer stage,” 2020, <https://learn.microsoft.com/en-us/windows/win32/direct3d11/d3d10-graphics-programming-guide-rasterizer-stage>, accessed 2023-05-01.

Appendix A

Particle Simulation Screenshots

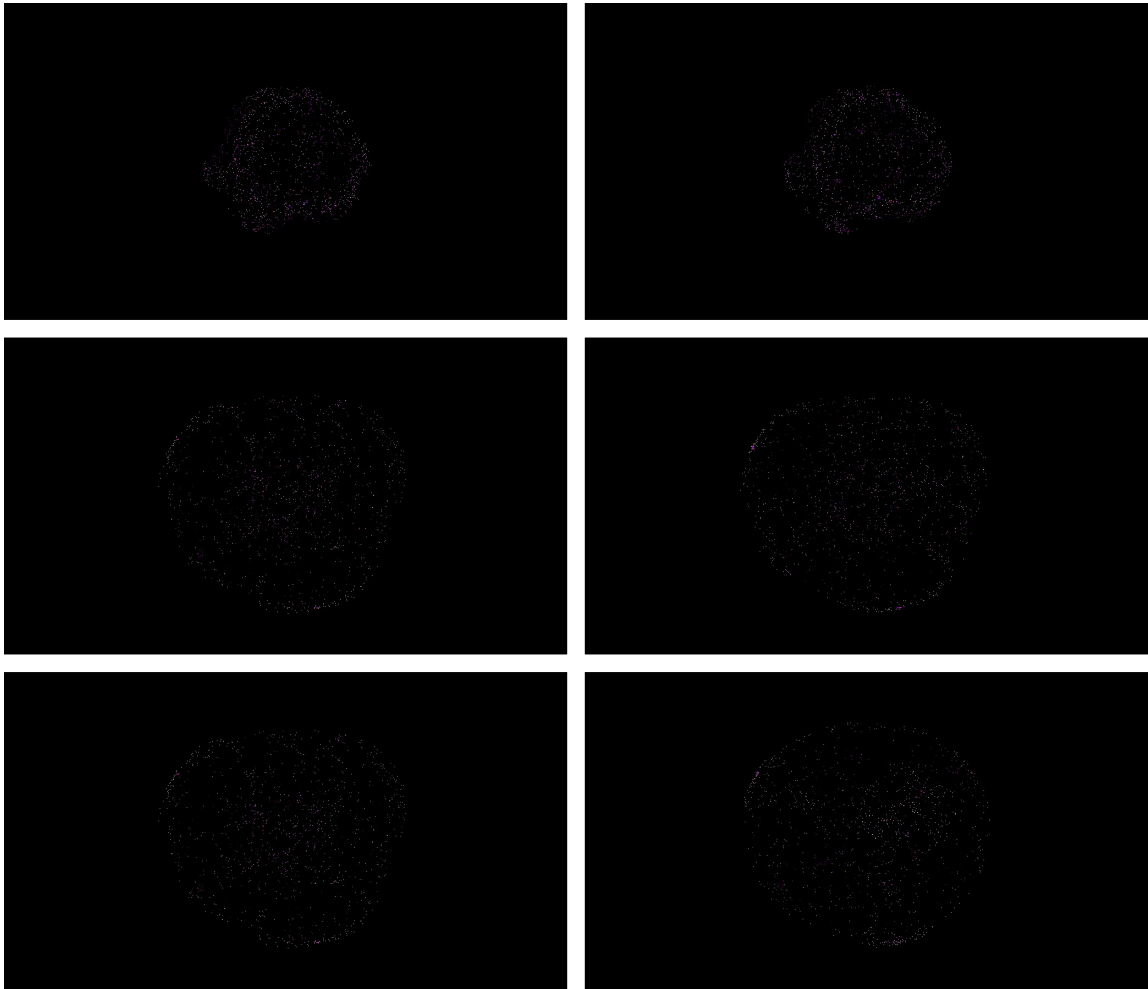


Figure A.1: Screenshots taken 7, 17 and 23 seconds into particle simulations of 2048 particles. The O-system is to the left, and the I-system is to the right. Due to the low particle amount, these pictures have been saturated and their exposure increased to increase readability.

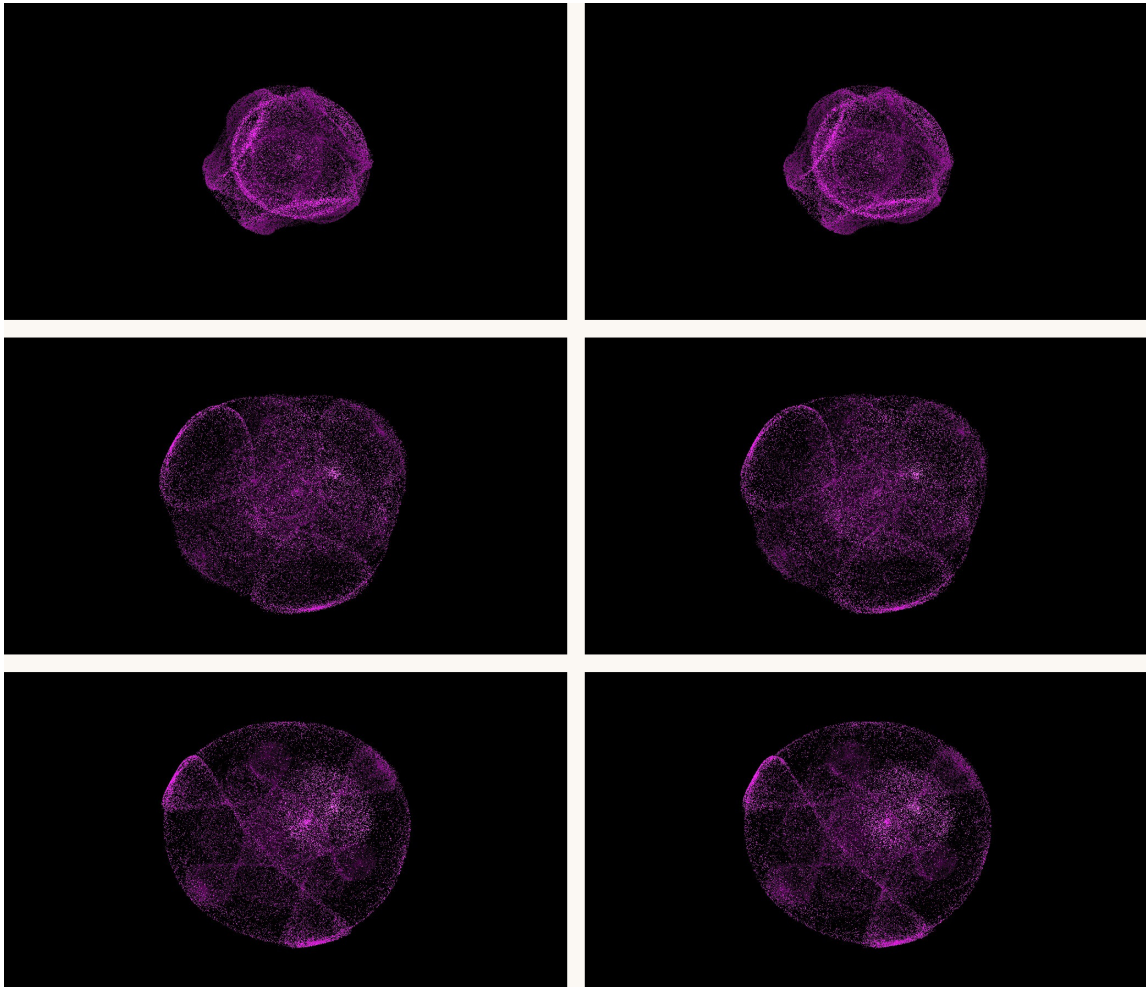


Figure A.2: Screenshots taken 7, 17 and 23 seconds into particle simulations of 65536 particles. The O-system is to the left, and the I-system is to the right.

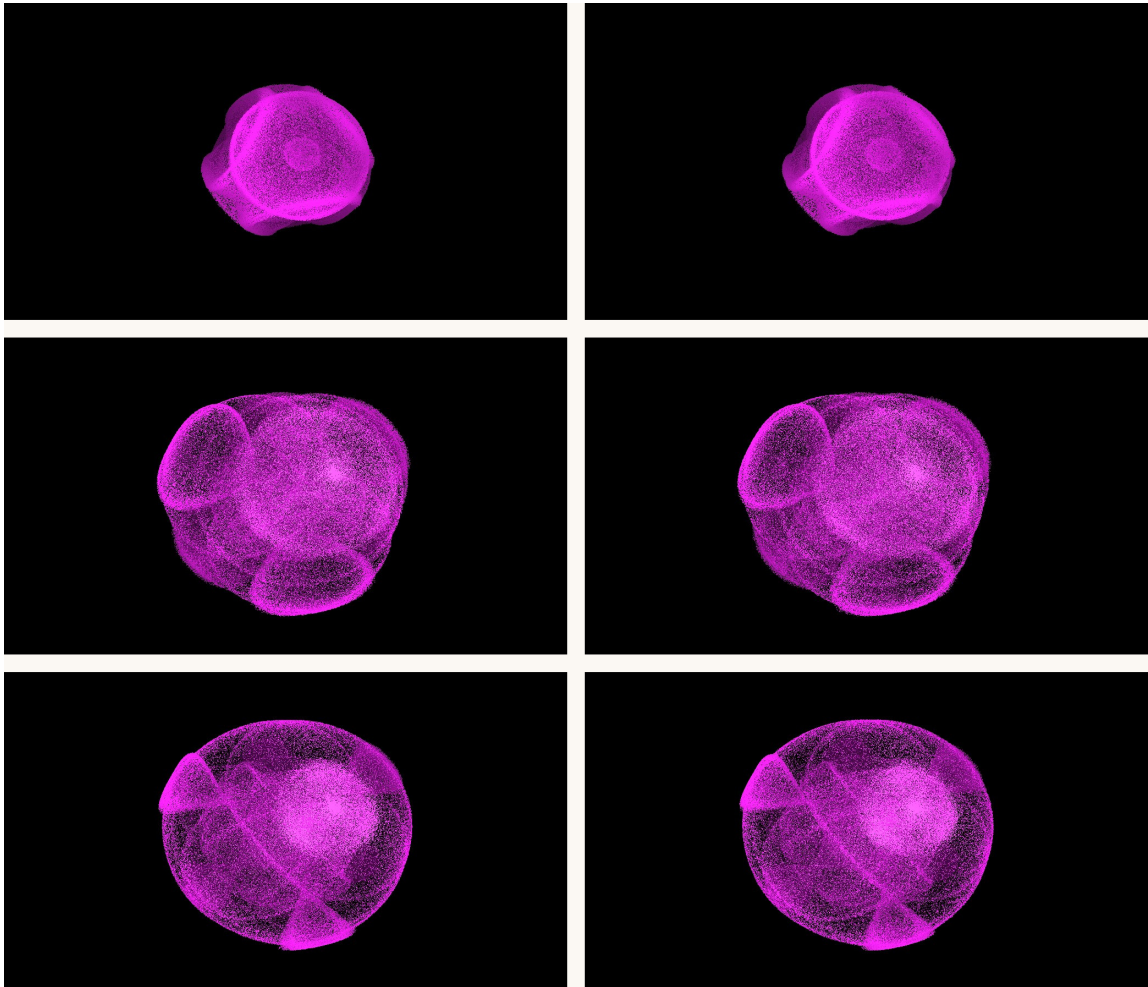


Figure A.3: Screenshots taken 7, 17 and 23 seconds into particle simulations of 524288 particles. The O-system is to the left, and the I-system is to the right.

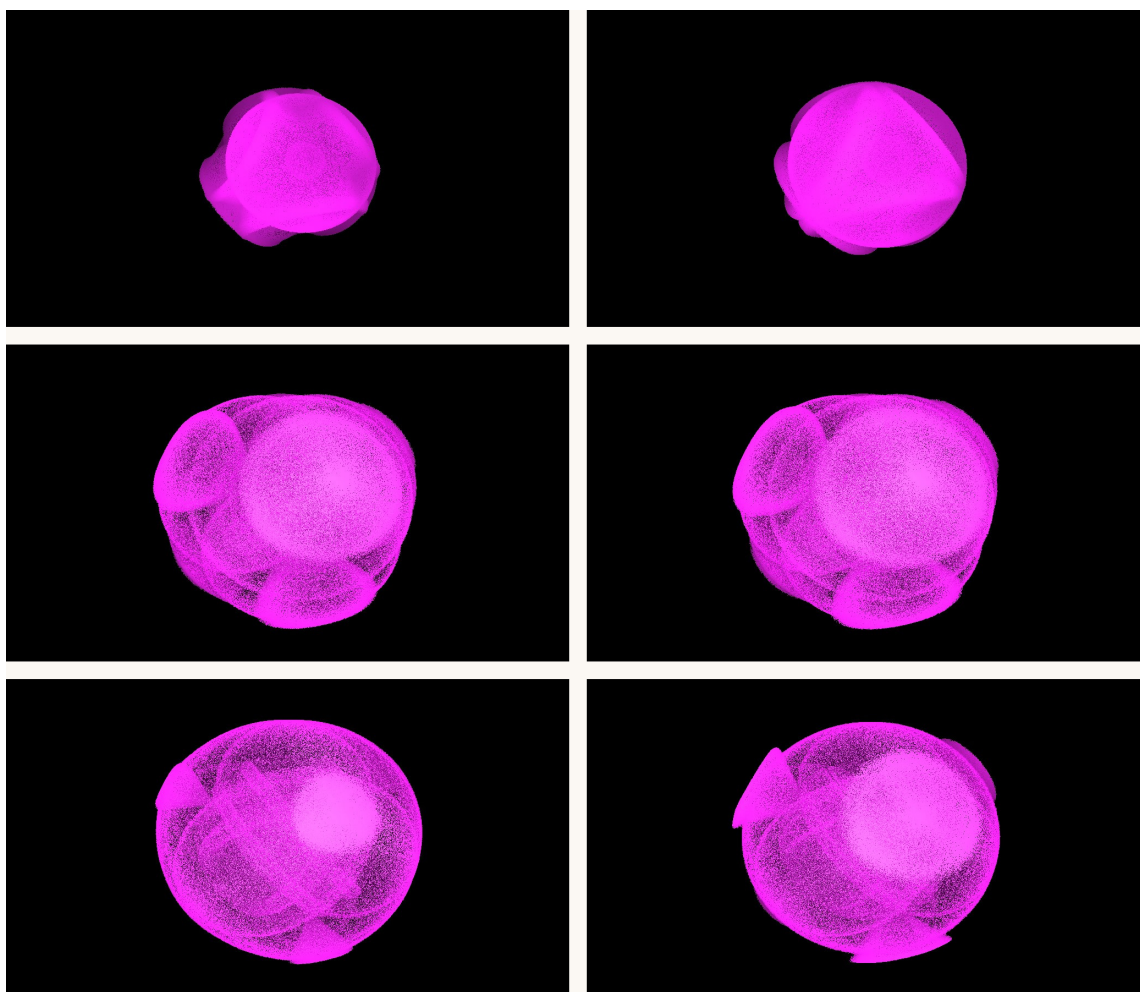


Figure A.4: Screenshots taken 7, 17 and 23 seconds into particle simulations of 2097152 particles. The O-system is to the left, and the I-system is to the right. With this many particles, the difference in frame rate between the systems leads to discernible deviations.

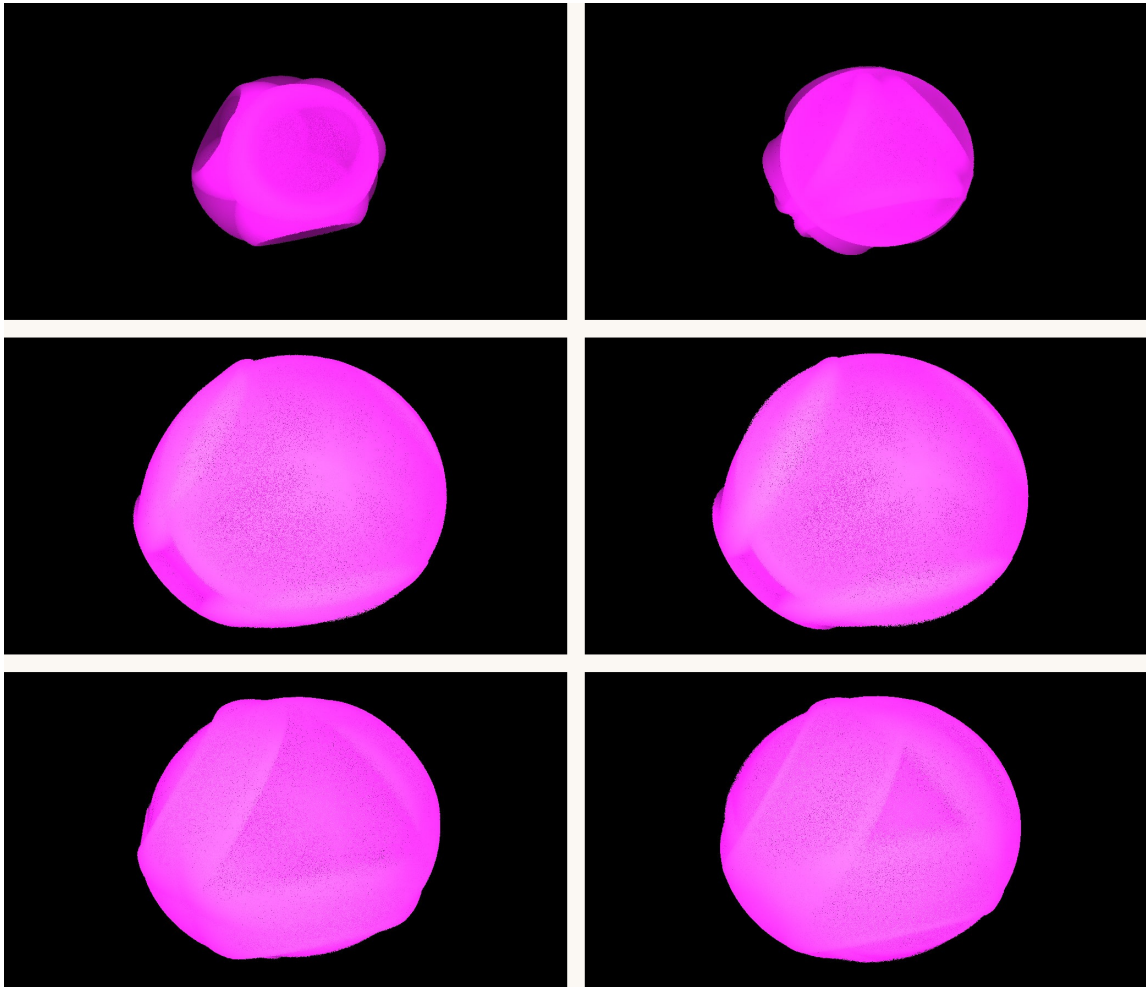


Figure A.5: Screenshots taken 7, 17 and 23 seconds into particle simulations of 8388608 particles. The O-system is to the left, and the I-system is to the right. With this many particles, the difference in frame rate between the systems leads to discernible deviations.

