MÄLARDALEN UNIVERSITY
SCHOOL OF INNOVATION, DESIGN AND ENGINEERING
VÄSTERÅS, SWEDEN

---

Thesis for the Bachelor of Computer Science Degree – 15hp

# 3D VISIBILITY EMERGENCY STOP SYSTEM FOR AUTOMATED INDUSTRIAL ENVIRONMENTS: AN OPENGL BASED SOLUTION

Evangelia Damasioti
*edi18801*@student.mdu.se

Examiner: Gabriele Capannini
Mälardalen University, Västerås, Sweden

Supervisor: Afshin Ameri
Mälardalen University, Västerås, Sweden

25/05/2022

# Abstract

The advent of industry 4.0 has not only brought innovation and automation with it but also new challenges. Automation in industrial settings is advancing at a rapid pace, thus making the modern industrial workplace all the more stimulating. Highly automated robots and machines work alongside humans in settings that seemed fictional some years ago. However, the shift to a smart industry has brought about certain safety concerns regarding whether the current safety systems can keep up with this ever-changing environment. Emergency stop buttons have long been the industry standard when it comes to classic safety precautions. Nevertheless, researchers examine several possibilities on how they can upgrade the already established safety systems. One such practice is to incorporate visibility as part of an emergency safety system. There has already been a proposal to use 2D visibility as an emergency safety protocol which has shown encouraging results. Thus, making a 3D approach as the logical next step. In this thesis work, a 3D visibility emergency stop system is presented, implemented, and tested. To begin with, a small review of how other researchers have tried to solve similar problems is done. Furthermore, several 3D graphics techniques used in the project are introduced and briefly analysed. Then, the implementation of the project is presented, breaking down each component. After the implementation is completed, several tests that aim to profile the program are performed and examined. In addition, the project is compared to the 2D solution that has already been proposed. The findings from the testing show that a 3D visibility approach is a viable and favourable option. The results reveal a promising outcome when it comes to object visibility as well as response swiftness.

# Table of Contents

# 1. Introduction

When we think about the future, the picture that comes to mind is some form of technological advancement that will upgrade how the world works and fix environmental problems. That picture is not far from the truth, but in order to reach such goals we need to take the first steps towards them. Scientific knowledge is at the base of all technological advances that step by step is changing the way the world works. Technology is the driving force that makes all that change possible. Technology defines the way we interact with one another, spend our free time and most importantly how we work.

The workplace is one of the areas that technology has transformed in recent years. More specifically, the industrial workplace has reached the milestone known as "industry 4.0" [1]. Industry 4.0 is known as the fourth industrial revolution where machines have become intelligent and a lot of the tasks that used to be performed by humans are now automated.

It is widely accepted that there needs to be a type of safety protocol in place, in all industrial environments. One of the most common safety practices is the use of emergency stop buttons (e-stops) which are universally trusted [2] but with the advent of industry 4.0, that safety might not be enough. In an environment where both stationary and mobile robots share the common space with humans, some encounters are bound to happen.

There is a proposal to use visibility with an emergency stop system where only the machines that are visible from a set point of view suspend their function [3]. The visibility proposal shows promising results thus making it a valid solution to the ever-increasing safety problem in highly automated industrial environments.

Visibility has been utilised in different ways in order to improve the functionality of a variety of ambitious projects such as collision avoidance in self-driving cars [4, 5]. Moreover, another practical use of visibility is proposed by Omar and Gu on the subject of airborne path planning [6]. Flying is also an activity overtaken by automation, thus making it a prime candidate for a visibility safety approach especially in three-dimensional space.

3D visibility has widely been an indispensable part of computer graphics that is used to find out which of the surfaces in a given scene are visible [7]. Light is shed in a graphical scene and with the use of visibility techniques it bounces off certain surfaces which then reveals shapes on the screen.

The goal of this thesis work is to create a 3D visibility algorithm that can be used as part of an emergency stop system in highly automated industrial environments, using OpenGL [8]. The first step is to load from the hard drive the models of the machines that the application can recognise as well as the environment. Continuing, the algorithm receives as input, a scene consisting of obstacles and machines in various poses as well as the location of the camera. The next step is to test each machine for visibility. After the implementation is completed, different benchmarks are profiled in order to test the scalability. Then, the different solutions are analysed and compared based on the results.

Some of the terminology used in the implementation, discussion and conclusion parts are briefly explained so that readers with no prior knowledge of the subject, can follow the flow of the report. Also, there are several proposals suggested as emergency stop systems in industrial environments and some of them are presented and analysed in order to understand how others have tried to solve similar situations. Lastly, the findings and discussions are concluded by a brief evaluation of how the work went as well as consideration of any future improvements.

# 2. Background

This section is dedicated to information that will assist the reader to better grasp the subject at hand. This extra material is intended as supplementary reading for those who want to learn the subject material more in depth. As such vital information in the first section, the subject of how safety in industrial environments is explained along with some more specific safety systems. The next part of this section handles the technical background that needs to be noted. In the last part, the topic of visibility in 3D is defined.

## 2.1. Safety in smart industries

As previously stated, Industry 4.0 is the new standard when it comes to the current state of smart industries. One of the most crucial elements of this new automation era is that both humans and smart machines work in a safe and protected environment. The goal is to minimize negative encounters by implementing safety systems that are modified to fit the highly automated environment.

This has prompted a discussion around how much pain do humans tolerate in case of a collision with a robot [9, 10]. The idea behind an emergency stop mechanism has been utilized in industrial manufacturing to avoid the risk of injury. In their article, Suita et al. propose a solution where they cover the robots with a viscoelastic material and the safety system is equipped with stop capabilities [3]. The goal of their experiment was to measure the human pain threshold and when a high point is reached, the robot is going to cease all activity with the use of an emergency stop mechanism.

There are many ways to implement safety in a traditional industrial environment such as safety mats, light curtains, laser scanners, software solutions and emergency stop buttons [11].

### 2.1.1. ” Kill switch” systems

A “kill switch system” in the context of an industrial environment is an emergency stop device, usually a button or lever, which disables all machine activity. According to International Organization for Standardization (ISO) [12] almost all kinds of machines must be equipped with an emergency stop system.

Emergency stop buttons are the most common safety systems that have been widely used in industrial settings but with the advent of industry 4.0 that safety might not be enough. While a “kill switch” can effectively de-energize all machines in the work floor that might not be ideal if all the machines must be switched off. Such situation might be time consuming and halter productivity while the problem is being addressed. The proposal to use a visibility emergency stop system where only the machines that are visible by a point cease working [3] can be useful. Such visibility safety allows all the machines and humans to continue with their activities by moving unhindered throughout the work floor. The moment a worker activates the “kill switch” all visible machines will stop moving.

## 2.2. Visibility emergency stop system

As for other emergency stop systems, the goal of a visibility emergency stop system is to pause all activities in the surrounding area to avoid accidents and collisions. As explained above, an e-stop button is pressed by a nearby person in order to stop the activities of all machines. That might not always be ideal since the person that is responsible for pressing the button might miss important events happening before, they become dangerous. With a visibility system, it is the computer through a visibility point that detects any dangerous situation as soon as it happens and automatically sends the signal that something is wrong.

## 2.3. The visibility problem

In computer graphics, the visibility problem has a significant point of research and many researchers have tried different ways to solve it in several ways [13, 14]. The problem is defined by the determination of which parts of certain objects are presented in a scene from certain points of view. In short, the goal is to determine the hidden surfaces or certain bits of an object from e specific point otherwise known as occlusion culling [15].

Occlusion culling is used to exclude objects from being drawn when they are hidden due to obstructions in the scene. In a highly automated environment where we want to use a visibility-based safety system we can use this technique in order to determine which machines are in front and which are behind a wall. This approach gives a more efficient signal on which machines should be shut down and which are not in line of sight and can continue working. However, there are other ways in which an object/machine might not be visible which have to do with the fields of view of the camera.

When objects are excluded from drawing based on the angle from the camera forward vector it is called frustum culling [16]. In automated industrial environments where machines and humans move around the working space freely the frustum culling technique can be immensely helpful. This technique helps to determine the objects that are relevant to be drawn according to where they stand with respect to the camera position, orientation, and field of view.

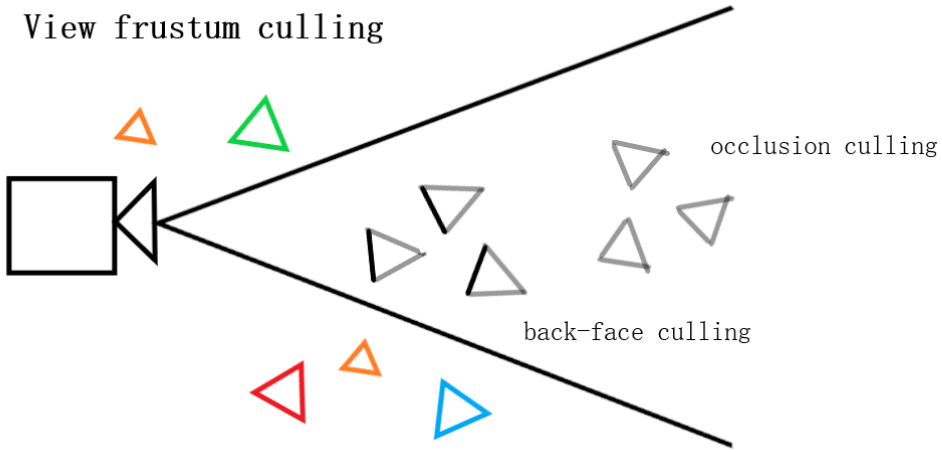


**Figure 2. This figure depicts both occlusion and frustum culling. The colourful objects are not visible in the frustum. The black lines are the visible sides from the objects while the gray lines are the occluded parts.**

## 2.4. Object Oriented programming

There are many methods of organising data in programming. Object oriented programming (OOP) is a method of managing and grouping data around the objects instead of the functionality of the program [17]. The main concept of OOP is that the program is divided into:

- Classes or methods which contain the scheme of how the objects are structured. These classes are abstract interpretations of objects that can be used again and again.
- Objects which are the focus of an OOP are instances of the classes or methods mentioned above. Each object can be represented by a class from which it receives its attributes.

In this thesis work, C++ [18] is used as the primary programming language which is a popular programming language that is often used in conjunction with OOP.

## 2.5. Rendering

Rendering is a method with which a program illustrates an image by sending information to the graphics card [19]. It is used both for two- and three-dimensional images. According to the work that needs to be done, there are different approaches a programmer can take in order to display the image on the screen. Rendering is widely used in computer game graphics, movie animations and in anything that needs to be visualised digitally.

With respect to scene rendering, in this thesis work, the preferred tool is Visual studio [20]. Visual studio (VS) is a software development tool shortly referred to as an IDE which stands for "integrated development environment" [21]. VS is a popular platform with which users create and troubleshoot computer programs.

- **Meshes**

3D representation of objects is done by an assortment of dots and lines that describe an item [22]. Essentially, this collection of dots and lines form triangles. That object is then rendered and drawn on the screen representing an object in 3D graphics. Each shape can have a low or high polygon count according to how and where it going to be used. For the purpose of this thesis work, only low polygon shapes were used because the accuracy of the objects is not of great importance to the results.

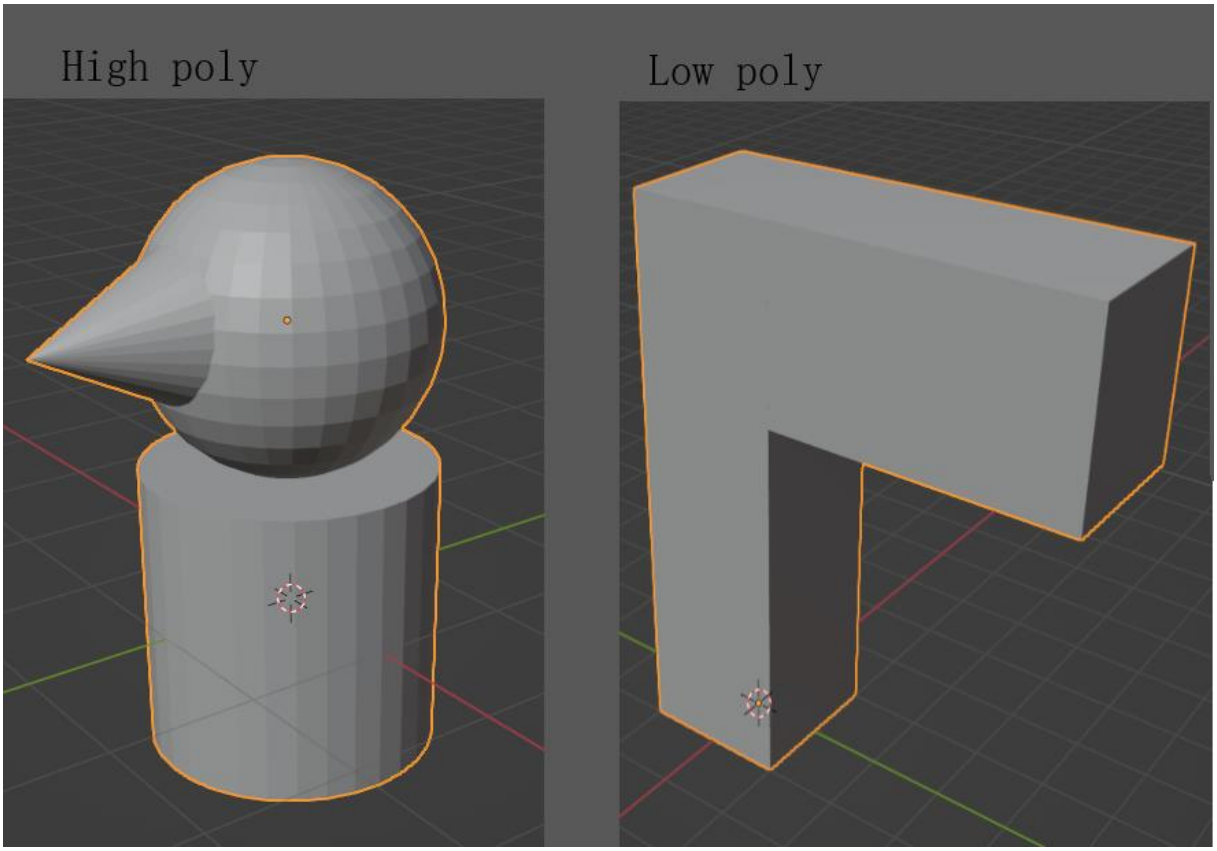


**Figure 3. An example of low and high polygon objects looks like.**

- **Blender 3D**

Blender 3D is the program used to create the polygon objects, the floor and plain that they stand on. Blender 3D is an open-source program that offers tools with which the user can create objects, animations and general 3D graphical applications like video game graphics [23]. Meshes created with

this program can be exported in various formats and for the scope of this project, these meshes were exported in Object (OBJ) file format.

Like everything in computer programming, the components that are used to assemble a fully working project need to be introduced to each other because they have no prior connections, nor do they recognize other components automatically. Such is the case for the work at hand where the meshes created must be imported in order to be utilised throughout the project. Since none of the libraries that are used for this thesis project have an OBJ loader, it was important that one was created to import the files.

## 2.6. Camera

A camera is an object that describes a viewpoint in a scene, and it is required to render that scene. The camera can sometimes be allowed to move in a pattern or freely in 3D space or alternatively to move the world with respect to the camera [24]. The camera is also divided into viewports creating a 360-degree view. A viewport is a rectangle on which we project a 3D scene and represents the area that the camera is directed on [24]

## 2.7. Runtime rendering

Real time rendering is when the program loops through the code and draws the scene in more than 30 frames per second [25, pp. 9-24]. There are many tools and operations that are used to make real-time rendering possible. The ones that are utilised in this thesis work are explained in brief below.

- **OpenGL**

Graphics libraries like OpenGL are an important part of any computer graphics project. OpenGL was used for this project not only because it is open source but also portable, can be used between different platforms and offers a broad spectrum of features. The features that are on offer are predefined functions that help with the rendering and illustration of two- and three-dimensional images.

- **Shaders**

A shader is a program that runs on the GPU and is divided into two parts:

The vertex shader, which is code executed for each vertex independently. It can modify and then pass that data on to the fragment shader [25, pp. 181-288].

The fragment shader contains the code that is being executed for every pixel independently. It receives flat or interpolated information from the vertex shader and writes its output to a render target that could be a texture or the back buffer [25, pp. 181-288].

- **Instancing**

Instancing is a technique that allows for the rendering of multiple objects at the same time when they share common geometry and material characteristics [26].

- **Depth testing**

Depth testing is a method of discarding pixels that are occluded behind other pixels. In a scenario where one machine is hidden behind another one, it compares the depth of the pixels that belong to each machine and only the one that is closer to the camera is kept [27].

- **Mipmaps**

Mipmaps are downscaled versions of high-resolution texture [28].

## 2.8. Render pipeline

A rendering pipeline is the part which interacts with the graphics library [29] and it consists of the following data and code:

In the data portion of the pipeline there are two file types:

- The models which contain vertex information of an object.
- The shader files which contain the vertex and fragment shader code.

In the code portion of the pipeline there are several steps that must be taken in order to render a scene:

- The loading of the models from the object files.
- The loading of the shader files.
- Moreover, there is the main rendering loop which consists of:
    - The “begin scene” call which prepares the device to start drawing.
    - The actual drawing of the models.
    - The “end scene” which notifies the device that it is done with the drawing phase.
    - Lastly the “present call” that flips the back and front buffers.
- The creation and release of render device.
- The create and release of models and shaders.
- The create and release of render targets.

# 3. Related Work

Many researchers have been engaged in the effort to tackle emergency situations in industrial environments. At the same time there are various articles that are using 3D visibility techniques in order to undertake several problems that arise in diverse environments. However, the combination of the two has not yet been explored which gives the opportunity of examining the task at hand quite interesting.

To begin with, there is an article by Ebrt et al. [30] where they propose an emergency stop system using a chip that allows for vision tracking which can detect collisions at rapid speeds. They argue that since humans and robots work in proximity then there needs to be an emergency stop system that detects when a human is too close to a robot. The vision chip they propose is going to utilise images in binary format which are going to represent two points: One that is furthest from the viewer and one that is the most near. Then they attempt to measure the distance between those two binary points in order to determine if the robot is at a dangerously close distance to the human. Their argument is that a camera would take more time to determine the distance between human and robot than the binary information from the tracking chip. In the end they had to make the robot shape slightly bigger because the collision was unavoidable due to the lack of brake controls on the robots. The paper highlights how vision can be helpful in an emergency stop system which makes the 3D visibility solution an interesting counter solution to the system they propose.

Continuing, Kaur and Bhatia wrote an article where they recommend using Internet of Things (IoT) in order to detect disasters early in industrial environments [31]. They propose a stochastic game net (SGN) type of solution that could be beneficial as an emergency safety system for highly automated industrial environments. The solution makes use of strategically placed sensors that are in constant communication with each other. One sensor activates a chain reaction that activates the other relevant sensors as well as notify the responsible workers that are needed to handle an emergency. This Iot network is skilfully programmed to make critical decisions with the use of the SGN approach. The goal is to detect an emergency as early as possible and notify the pertinent personnel in order to prevent monetary and material losses. One disadvantage that this proposal has is the wait between the detection of an emergency and the decision to trigger the sensor network and workers. Despite that unfavourable delay, their results showed a high ratio of accuracy of over 90% on all cases along with high reliability scores and high stability results. Taking everything into account, IoT can be quite an advantageous safety system for smart industries but at the same time it can be quite an arduous job to set up. There are many points where the network can break down and the cost can prove prohibitive for some businesses.

Moreover, Capannini, Carlson and Mellander have introduced a 2D visibility emergency stop system from which this thesis work was stimulated [3]. They discuss that highly automated industries where robots and humans work in the same space could benefit from an additional safety system that is based on 2D visibility. The visibility problem is examined, and the researchers end up finding two distinct points where their scenario does not fit the theory of the problem. The first is that the polygons used to define the objects in their scenario are linked while the second is that some of these objects allow the camera to see through them. Continuing, they explain that in order to define the objects, they use AABB bounding boxes that the objects are enclosed in. The algorithm that is suggested starts with a reserved point from where the visibility is going to be based on. The next step is to pass through all the edges of each item one by one thus determining which objects are behind walls or other machines. If an object's edges are behind a wall, then that object is considered non-visible. The results of their experiments show promising speeds despite the complexity of the algorithm. The algorithm is proven to be satisfactory to be used as an emergency stop system. This approach is an interesting addition to emergency stop systems in industrial environments especially now that these environments become even more complex. Since this 2D version shows promising results, it is even more appealing to build a 3D approach and compare the timing and performance.

# 4. Problem Formulation

Nowadays, highly automated smart industries are evolving which makes the need for new and improved safety systems all the more relevant. The traditional emergency stop buttons might have been a good solution so far, but a new visibility emergency stop system has been introduced [3]. Capannini et al propose a visibility emergency system that renders a 2D scene and stops all the machines that are visible from a specific point of view. However, there are some issues to this solution because of the limited precision of a 2D AABB based analysis. There might be issues with some of the longer objects which could potentially not be very accurately represented when placed in diagonal or other unfavourable positions.

The aim of this thesis is to build an algorithm solution using 3D rendering techniques which in turn is going to be analysed and ultimately compared to the 2D solution. The comparison of the two solutions is going to examine and determine whether the 3D solution is an upgrade in efficiency and veracity.

The questions that are answered in the thesis work are:

**Q1.** Can a 3D solution be implemented as an emergency stop sytem in highly automated industrial environments?

**Q2.** Can a 3D approach produce accurate results for the visibility problem in industrial settings?

**Q3.** How does the 3D approach compare to the 2D AABB solutions in terms of performance?

# 5. Method

For this thesis the method that is going to be used is Constructive research which uses different research methods bundled together to answer convoluted questions [32]. This technique can be very helpful when solving computer science problems because they tend to be quite complex in nature. In constructive research there are several steps that the researcher must follow according to the needs of the research being done. In this thesis the steps are as follows:

- **Research topic selection**
  This part is fundamental to the whole research procedure because it determines whether the argument of the thesis is valuable and will lead to constructive conclusions. In this work, the chosen topic is the visibility problem in highly automated industrial environments. The proposal is a 3D visibility emergency stop system that is going to be compared to the already publicized 2D solution.

- **Literature review**
  The next part is a literature review where the aim is to examine how other researchers have approached emergency stop sytems in highly automated insutrial environments and 3D visibility solutions. This part is essential to the thesis project because it can determine the approach that will be taken in this work.

- **Design and mathematics**
  This part deals with the design of the theoretical aspects of the project where all the algorithm specifics are determined and explained in detail. Along with the algorithm details, the theoretical aspect of the algorithm solution is also explained and presented.

- **Implementation**
  Since the goal of this thesis is to create a 3D solution, there are some steps that need to be done:
  - Build a specific starting scene on which the testing of both solutions is going to be done.
  - Create a render pipeline which will communicate all the objects of the scenes to the computer screen.
  - Save the scenes in a file format and create a file loader in order to access the same scenes again for testing purposed.
  - Create the 3D algorithm.

- **Testing**
  Testing is an essential part of every research project where all the algorithms and code that is created in the implementation phase is tested. In order to test the solution for the 3d approach, 10 fixed scenes are created and saved. Then a loading algorithm is going to be utilized in order to load the scenes and test using the same information. This way ensures that the results can be repeated not only by 2D or 3D but also by any other potential reseracher.

- **Data collection and results**
  The last step of the constructive research method is to gather all the results from the experiments and determine if the are fit to solve the problem at hand. In this case, the information gathered in the end needs to be able to answer the research questions.

# 6. Ethical and Societal Considerations

During the thesis project, all the implementation and testing are not going to involve any information regarding classified or sensitive data. All data is generated by the program that is implemented in the scope of the 3D solution which is not classified and openly available to the public.

However, the project requires data results from a previous 2D project [3] in order to test them against the 3D solution of this project. During the testing on the 2D approach, only the results of the tests of the scenes generated are going to be published in this paper. This means that the code and information of the article of Capannini et al. is not in danger of being leaked or revealed.

Furthermore, there could be some ethical implications when it comes to the nature of emergency systems in industrial environments. Such systems are of critical importance because they keep workers safe and away from danger. Since the solution that is proposed in this thesis work could be used in a real industrial production line, then the ethical implication of malfunctions should be seriously considered. On one hand, such a solution can have promising monetary advantages for the company that is going to use it due to the elevated safety solutions. On the other hand, a mishap could have devastating implications. In case the solution does not work properly then the results might lead to serious accidents. A 3D visibility solution along with the traditional e-stop buttons has the potential to decrease work accidents thus making safety in highly automated industrial environments better.

# 7. 3D visibility emergency stop algorithm

To create the rendering pipeline of the project, the first thing that had to be done was to start with a rendering pipeline template. For this project, a rudimentary template from a graphics course is used [33] as a startin point. The basic mathematics and simple functionality are then enriched with more specific functionalities applicable to the project at hand. In this part of the project, each essential component of the implementation is described and explained.

## 7.1. Mesh

The mesh consists of the geometry, the materials, and the handle resources. The mesh represents all the 3D objects in the scene, which in this case are the machines and walls.

The information that are included in the mesh are:

- The geometry: Describes the external surface of the machines and walls.
- The render material: The handle to the shader program. A number that is used to communicate with Opengl about an object that has been created. There are no other parameters due to all the objects are monochrome and textureless because there is no need for such information. The choice to omit this kind of information was made because it is not in the scope of the project plus it makes the programm lighter.
- The handles for the render resources of the geometry: These are numerical ids used to refer to objects (buffer, buffer arrays, textures, texture arrays etc) created by OpenGL. Each object that is created by OpenGL is then referred to with the help of these handles.

Each mesh is a type of shape that appears on the scene which means that all machines that look the same share the same mesh. In the project at hand, there is only one type of mesh (figure 1) which is then scaled and rotated randomly with the use of the mesh instance.

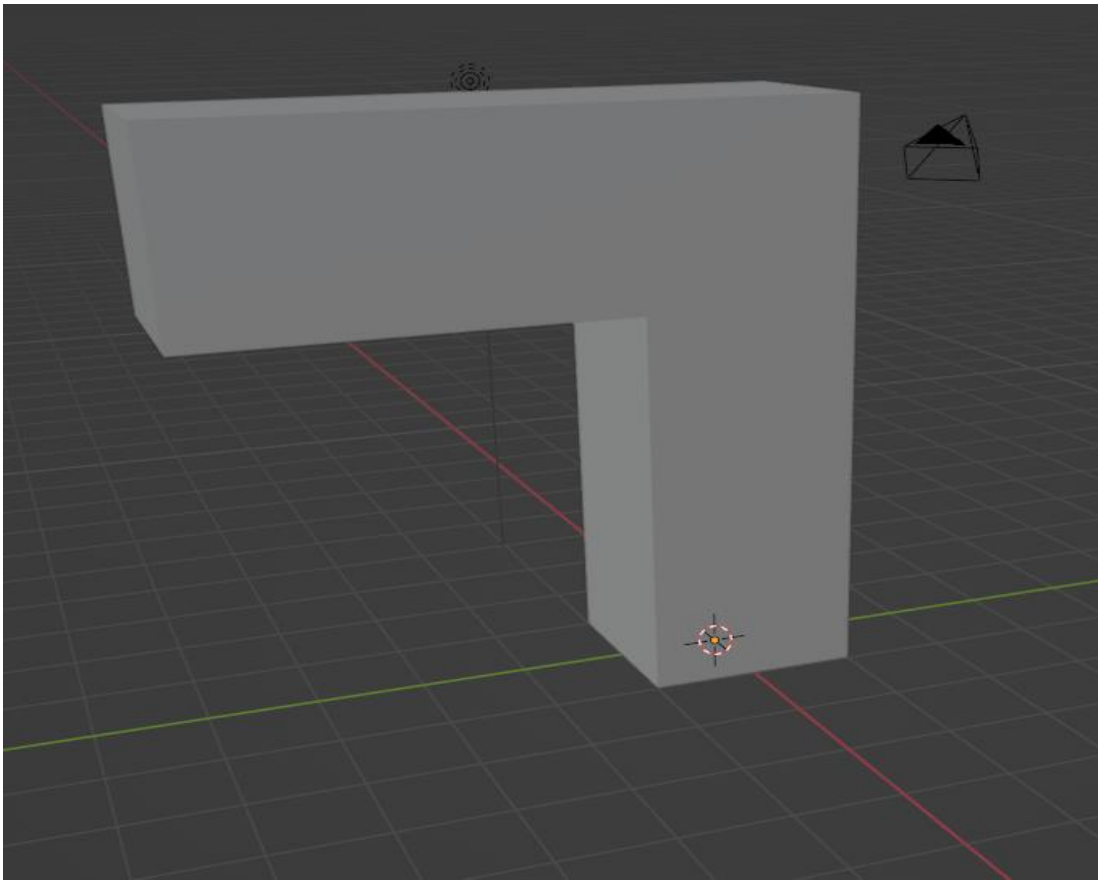

**Figure 4. The mesh object used to represent the machines created in Blender 3D.**

## 7.2. Mesh instance

The Mesh Instance represents an individual object in the scene that can be positioned, rotated, and scaled independently. This information is useful because the program needs to remember where each machine is located, how big it is and if it is rotated. The point is that in an industrial environment not all machines are in the same place, have the same size or the same rotation. So, there is a need to represent that in the scene. Each mesh instance has a mesh property which it shares with all other objects of the same shape.

## 7.3. Camera

For the camera portion of the project the template from a previous project [34] that was completed for the graphics course [33] was used. The main functionality is used and enriched with additional components required for this specific project. The camera is an essential part of the project because it is the point of view and helps to position the viewport and navigate through the rendered scene. It can rotate on two axes, pitch and spin but not roll. The camera represents the point of view from where the visibility is going to be tested.

## 7.4. Viewport

The rendering is done in a 360-degree view, so it is split in four 90-degree fields of view called viewports. The visibility test is conducted in a 360-degree view and not only from a front point of view, but that is also why it is important to set up four different viewports: One in the front, one on the left of the camera view, one on the right and lastly one o the back. There is the possibility to set up viewports for the top and bottom sides but for the purpose of this project it is not necessary.

## 7.5. Object loader

The models are created using the Blender 3D program and the meshes are simple with minimal triangles. The models are then saved in obj format and loaded using a parser.

## 7.6. Renderer

The renderer is a wrapper class for generic OpenGl functionalities and is used to centralise the access to OpenGL functions. To begin with, it can create a mesh render device resource which is used for the scene objects. In other words, it sends the geometry of the objects to the graphics card. Continuing, it sends draw calls to the render device making sure that the correct global parameters have been updated. In addition, it loads the shaders which are needed for the drawing of the objects and to test the visibility.

## 7.7. 3D visibility solver

For the 3D visibility solver there were two versions that were implemented during the project work. The reason for the two versions is that the first version was very vulnerable to float precision errors

### 7.7.1. Version one (obsolete)

The 3D visibility is done using depth testing on a per pixel basis. The first step is to render all the objects using a custom depth buffer. Then each machine is rendered on its own against the combined

depth buffer using less or equal depth test function. Then, the render target is downscaled using mipmapping and copied to the main memory. If any pixel contains color, it means that the object is, to some degree, visible. The render target is then cleared after each draw.

```
runVisibilityTest
        Mark all machines as non-visible
        set custom depth buffer
        set custom render target
        set depth function to "less"

        for each angle
            clear depth buffer
            draw all machines
            clear render target
            set depth function to "less or equal"

            for each machine
                if !machine.is_visible
                    if machine is in front of the current angle
                        draw machine
                        generate render target mipmap
                        copy high level mipmap to main memory
                        if any pixel is non-black
                            machine.is_visible = true
                            clear render target
                            break;
```

**Figure 5. The pseudocode that represents the first 3D visibility algorithm testing.**

#### 7.7.2. Vesrion two (current)

The second 3D visibility algorithm relies on objects painting unique integer ids on to the colour buffer. After all the objects have been rendered, the colour buffer is copied back to the main memory and each pixel is treated as a visible object. The intricate parts are the threaded processing of the colour buffer and its parallel operation simultaneously with the rendering. The processing of the render target has a large fixed initial cost and linear time complexity of O(n)with respect to resolution. The rendering has a high initial cost, but it scales flat with respect to the number of objects. The complexity is:

*fixed cost * number of objects / 255 + number of objects * insignificant multiplier*

The main memory usage is linear to the number of objects plus linear to the resolution

*Number of objects + resolution*

because the memory allocation needs to fit the render target.

*Size of int * resolution^2*

```
runVisibilityTest
        Mark all machines as non-visible
        set custom depth buffer
        set custom render target
        set depth function to "less"

        for each angle
            clear depth buffer and render target
            collect machines within camera frustum
            draw frustum collected machines (255 at a time)
            draw walls and floors
            wait for worker threads to exit
            copy render target to main memory

            [split between worker threads]
            for each pixel in the render target
                convert pixel value to instance id
                convert instance id to machine index
                set respective machine to visible
```

**Figure6. The pseudocode that represents the current 3D visibility algorithm testing**

## 7.8. Scene generator

This part of the project creates a list of mesh Instances at random positions and rotations on a square field. There are two objects being generated in this scene and both have the same mesh but are of different sizes. It can also save a 2D representation of that scene. The reason for the 2D representation is because the same scene is going to be tested with a 2D visibility algorithm in order to compare the 3D and the 2D solutions.

## 7.9. Single scene renderer

The scene renderer draws the scene from a single viewport based on the camera orientation. The ground is painted gray, the backdrop is black, and the visible machines are green while the occluded machines are red.

## 7.10. Panoramic scene renderer

This part is similar to the single scene renderer, but it draws four orthogonal angles and cascades them from left to right inside the window client area.

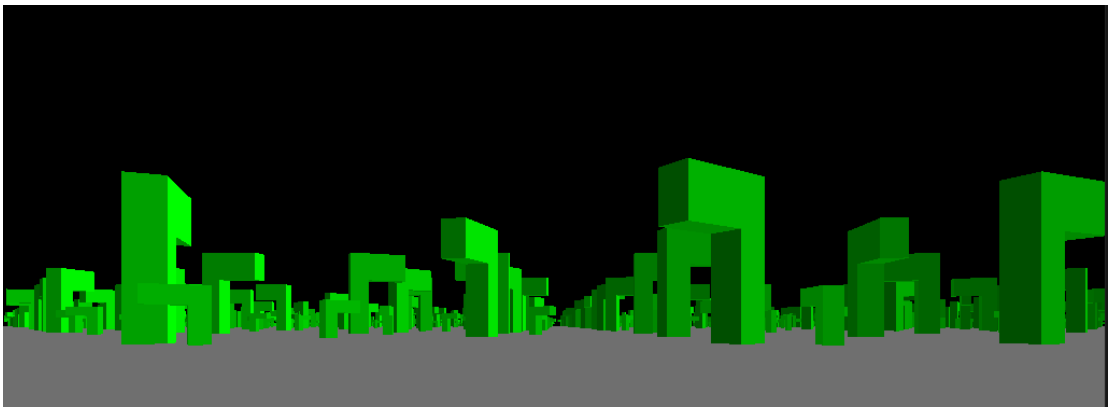

**Figure 7. Panoramic scene that shows all four orthogonal angles of the scene. The objects are drawn in green because they are visible from the camera view.**

## 7.11. Saving the scenes with 2D representation

Next the scene file is saved with the additional 2D representation which is done with convex hull gift wrapping technique. The mesh is flattened on the x-z plane (horizontal plane) and reduced to a convex polygon of arbitrary number of edges. Starting from the vertex furthest away from the centre of the machine mesh, segments are drawn to all other vertices and the second vertex is added to the hull unless there is another vertex to the left of the segment. This ensures that for each segment of the hull, there is no vertex that is outside the enclosed area.

# 8. Testing

The testing is done using 10 scenes that are composed of 10.000 total tiles and each tile contains zero to two machine meshes. I pick a number of machines to spawn and randomise the type of machine and grid position. The goal is to create a scene that contains 10.000 machine objects which are going to be tested for visibility. The default values for all tests are:

- Resolution: 512 x 512
- Frustum culling: ON
- Number of worker threads: 4
- Top and bottom viewports: included

For some of the tests these values are changed depending on the test.

## 8.1. Realistic scenario

A realistic scenario would be a large indoors environment with stairs, walls, windows, and doors. As a realistic testing scenario, I have created a room with randomly positioned walls that each have an open door and window. The room is divided into four sections and each section has one to three randomly positioned machine objects.

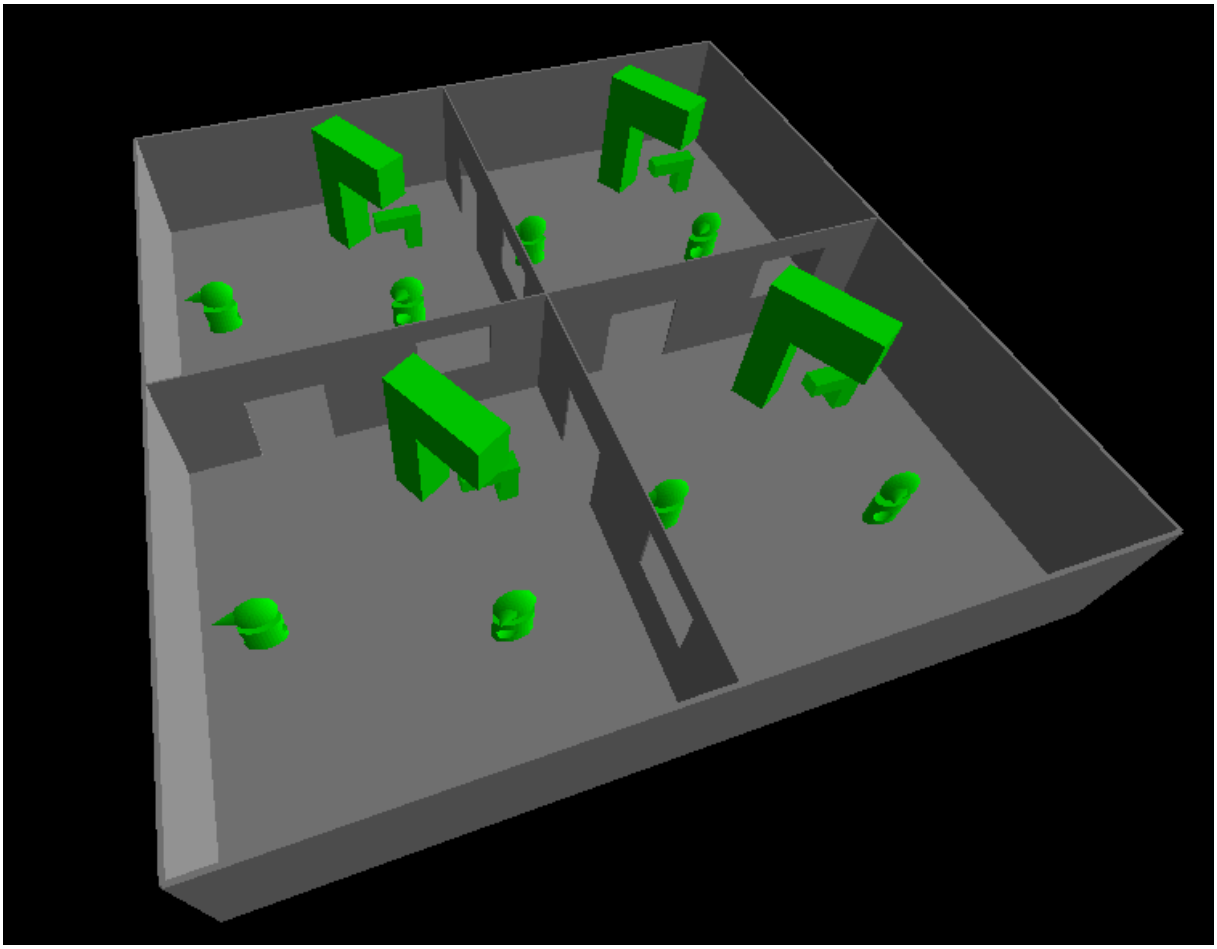

**Figure 8. A top-down view of the room in a relaistic scenario along with walls and machines.**

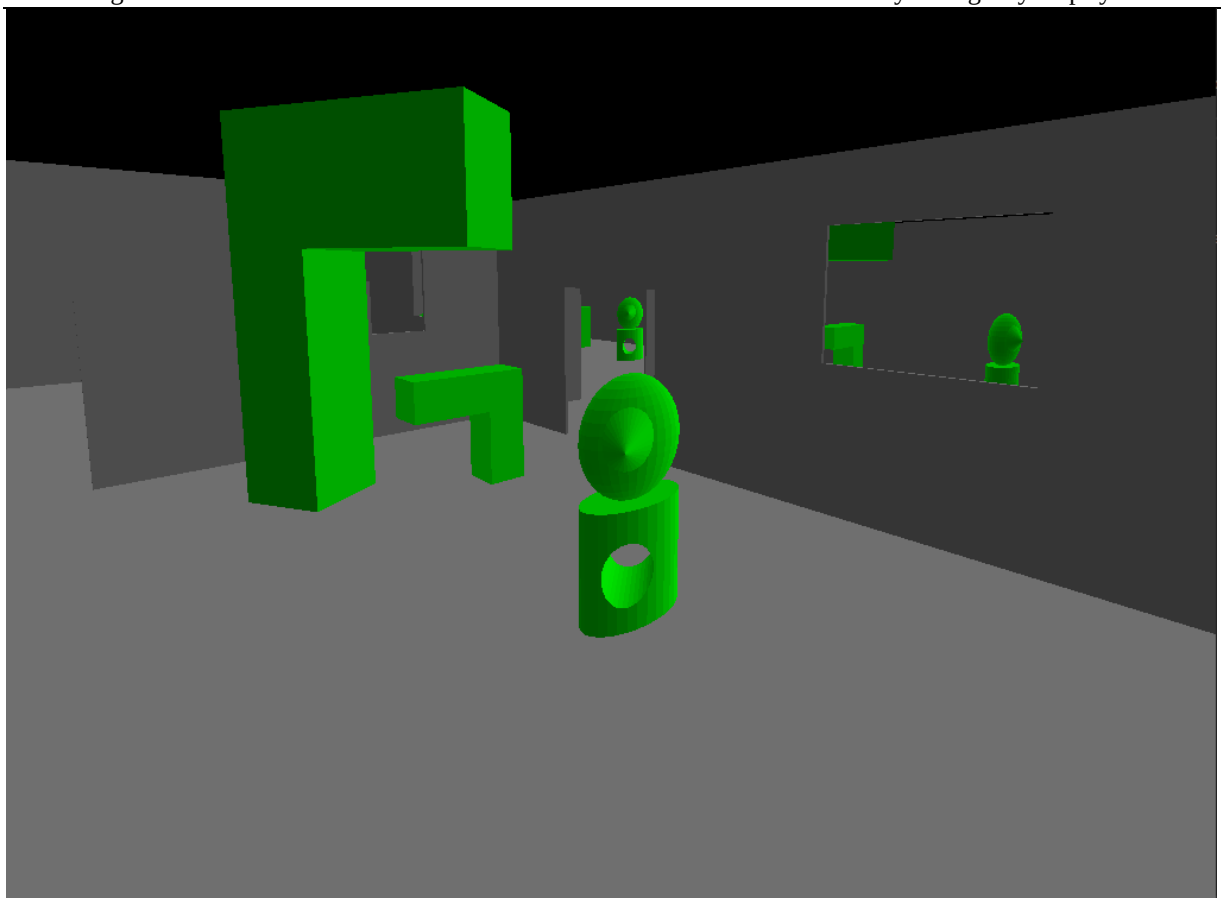

**Figure 9. The view from inside the realistic scenario room, where some of the machines are visible. The machines that are occluded by the walls and other machines are not considered visible by the visibility algorithm.**

## 8.2. Synthetic benchmark (stress testing)

The reason that so many machines are created in the scene is that I need to create an artificial scenario where the program is going to be stress tested. This scene does not represent a realistic industrial setting, but it is used for testing purposes only to profile for performance and accuracy.

## 8.3. Time and visibility

For the stress test, there are 10 scenes consisting of a grid plane that measures 100 by 100 squares with the possibility of zero to two machines as mentioned above.

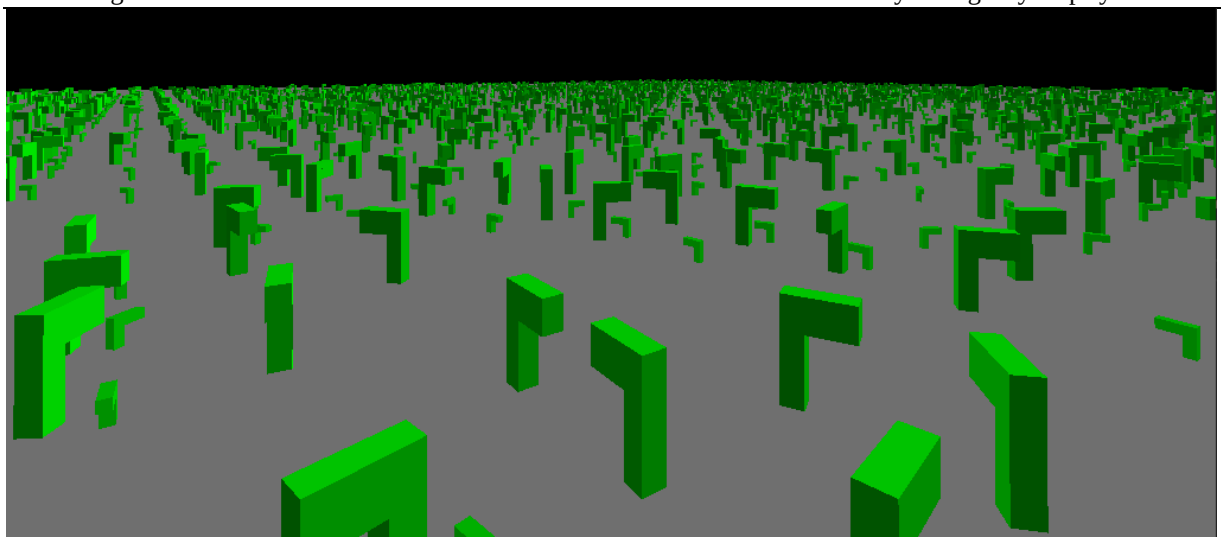

**Figure 10. Aerial view of the front viewport of the scene that depicts a portion of the 10.000 machines.**

To test how much time, it takes for the visibility algorithm to complete all the tasks, a timer has been placed in several points throughout the 3D visibility function. The timer marks are set at the following places:

- t0- At the beginning of the function.
- t1- After the initialization of the render states and memory allocations.
- t2- At the beginning of each frustum.
- t3- After rendering.
- t4- After each frustum visibility test.
- t5- The last one is at the end of the algorithm.

The following intervals are measured:

- Initialization (t1 - t0).
- Sum of rendering (t3 - t2).
- Sum of visibility tests (t4 - t3).
- Total algorithm runtime (t5 - t0).

## 8.4. Performance

To test for optimal performance there are several variations of tests being done. It is important to tweak and optimise the algorithm when experimenting with any kind of project. So, for this algorithm I decided to test if there are any significant changes using different techniques.

The techniques used are:

- Testing with and without the upwards and downwards viewports.
- Testing with and without the use of frustum culling.
- Testing with different resolutions.

## 8.5. Comparison of 3d alternatives

While trying to perfect the 3D visibility testing, there were two different versions of the testing being done. In this section these two versions will be compared as to how fast they complete the task.

## 8.6. Comparison with 2D

As stated previously, one of the goals of this project is to test the algorithm against another project [2] that solves the same problem using a two-dimensional approach. For the purpose of this comparison, two factors will be measured:

- Total execution time.
- Visibility tests results. These results will be measured according to how many objects are visible.

To make sure that the test cases are equivalent, the same 10 scenes are used for the tests. Since I do not have access to the code of the 2D project, the scene format along with the code to deserialize the file scene contents were sent to the responsible representative of the project.

# 9. Results

The tests were completed according to the information given in the testing chapter of this project. In this section all the test results are displayed according to the category they belong to.

## 9.1. Time and visibility

| Time and visibility (3D version) | | |
|---|---|---|
| Scene | Visible machines | Total time (ms) |
| 1 | 1009 | 7.465 |
| 2 | 959 | 7.871 |
| 3 | 869 | 8.01 |
| 4 | 926 | 7.897 |
| 5 | 986 | 8.049 |
| 6 | 897 | 8.187 |
| 7 | 1007 | 8.993 |
| 8 | 920 | 7.662 |
| 9 | 911 | 8.315 |
| 10 | 900 | 8.328 |

**Figure 11. The results from the tests of the 3D version.**

## 9.2. Performance

All the results from the performance tests are exhibited in this part of the paper. There are three different tests being done to examine the performance of the 3D algorithm.

- **With and without upwards and downwards viewports**

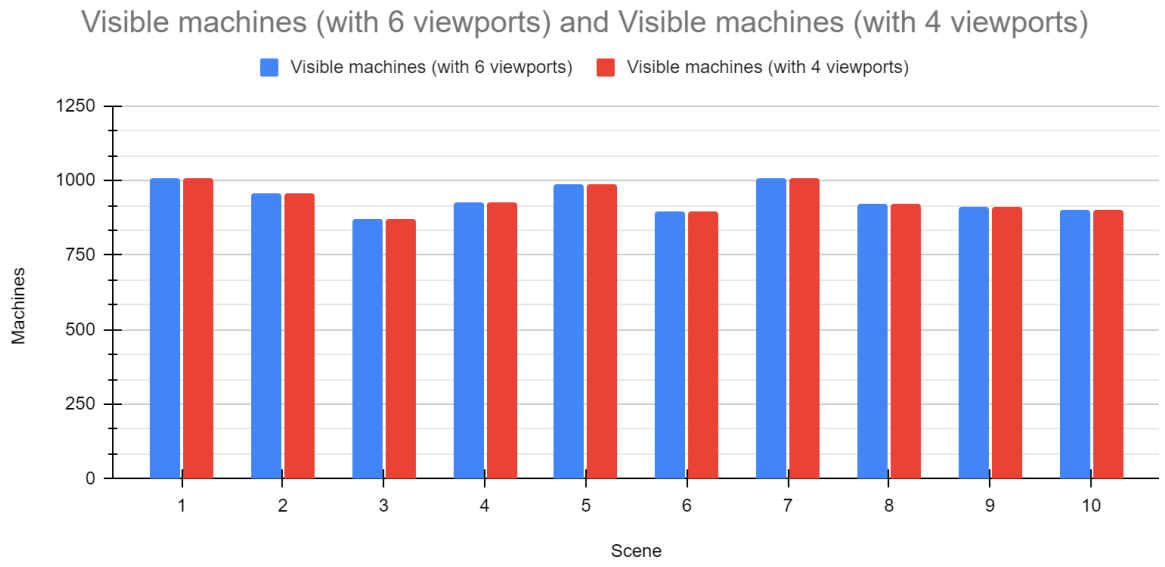


**Figure 12. The results of how many machines are visible according to how many viewports are used in the test**

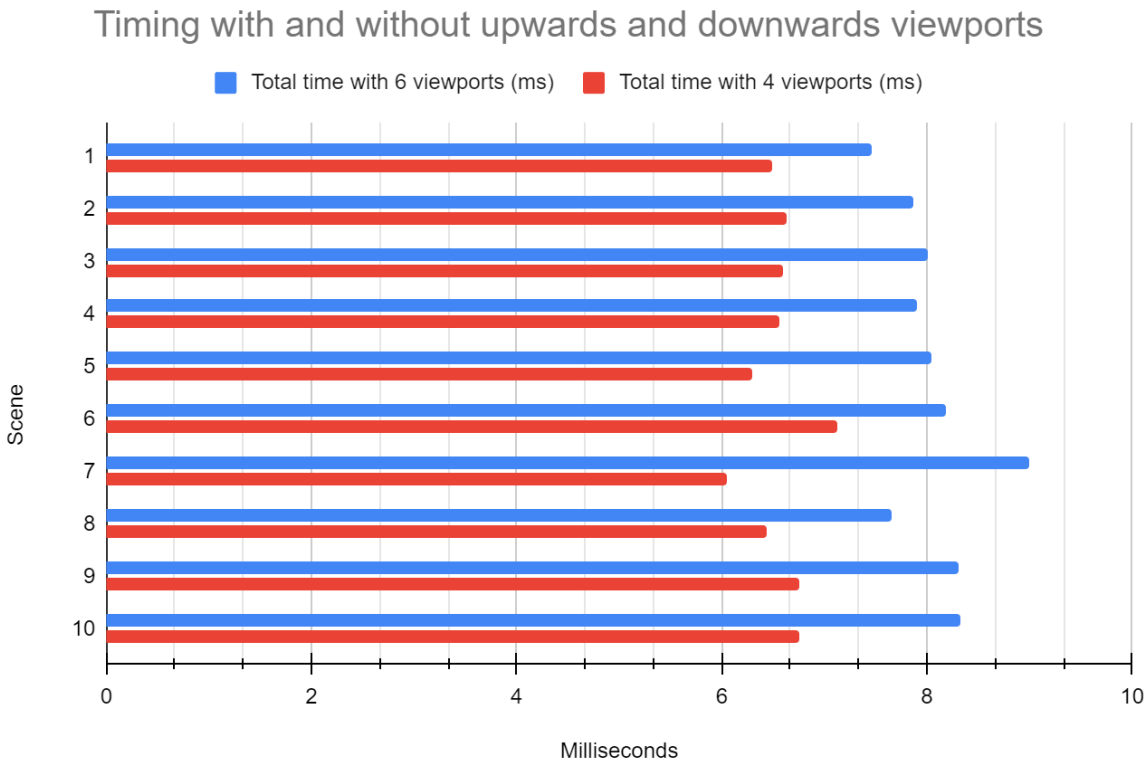


**Figure 13. The results of the test on how much time it takes for each version to finish.**

It is clear from the results that excluding the upwards and downwards viewports from the algorithm does not make any difference to the visible machines because the camera is facing in front and tests are at a 360 view. There are no machines in the ground directly below the camera nor are there any machines in the sky directly upwards from the camera view. However, when testing for how long it takes for the algorithm to complete there is a significant difference.

- **With and without frustum culling**

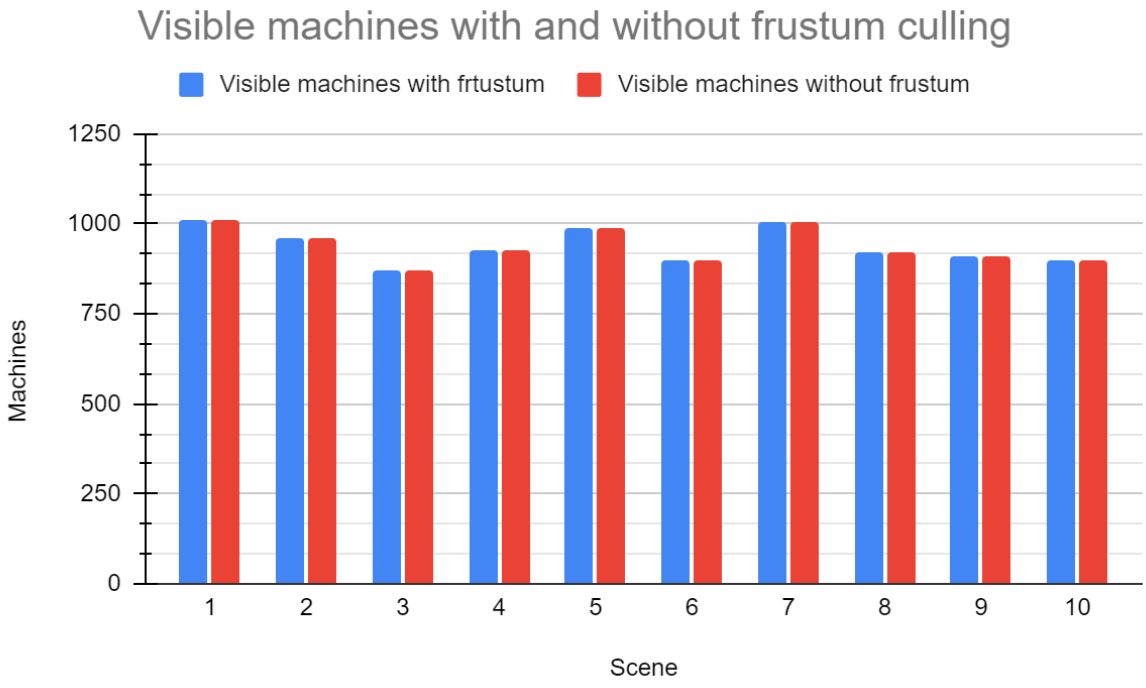


**Figure 14. The results of how many machines are visible with and without frustum culling.**

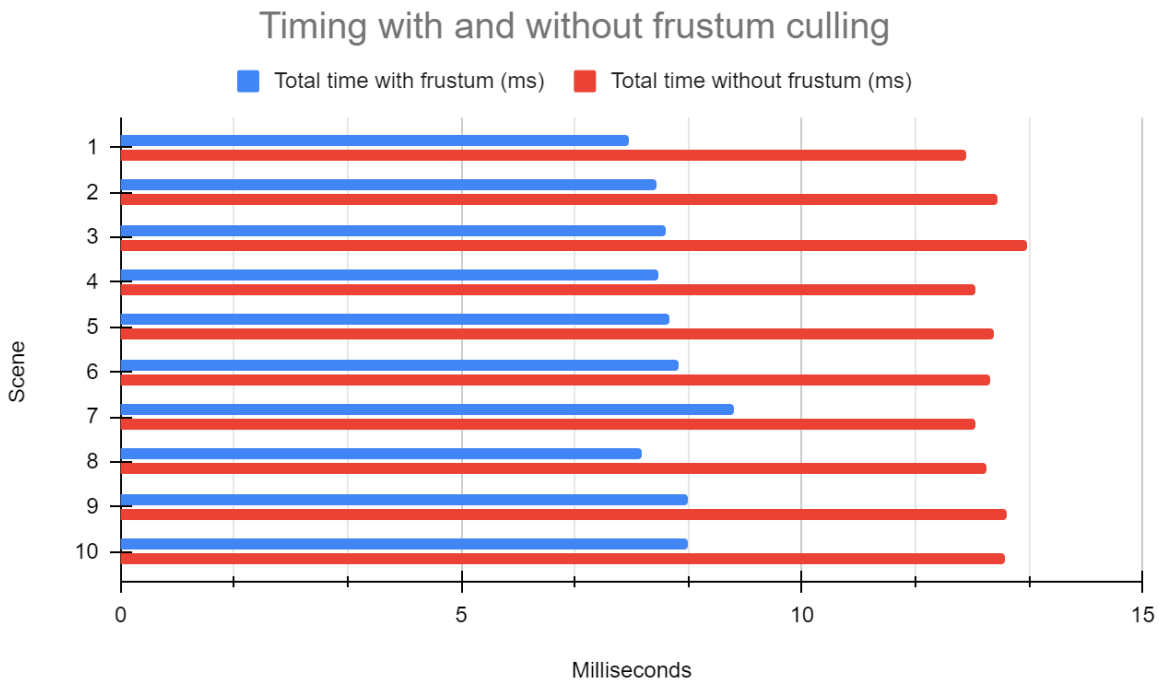


**Figure 15. The results of how much time it takes for the algorithm to finish with and without frustum culling.**

From the results of the frustum culling, it is apparent that the visible machines are the same, but the timing is considerably improved with the use of frustum. The reason is that the algorithm skips some machines that are outside the field of view thus having to render fewer targets which in turn makes the runtime much lower.

- **Different resolutions**

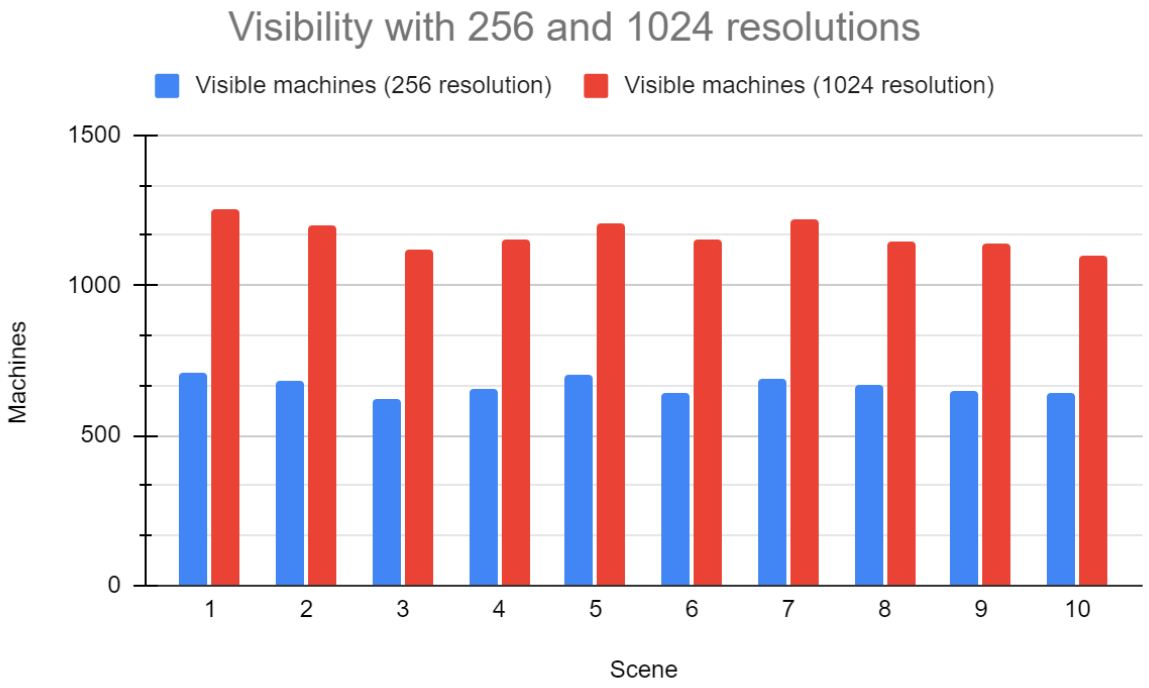


**Figure 16. The results of how many machines are visible with 256 and 1024 resoltions alternatives.**

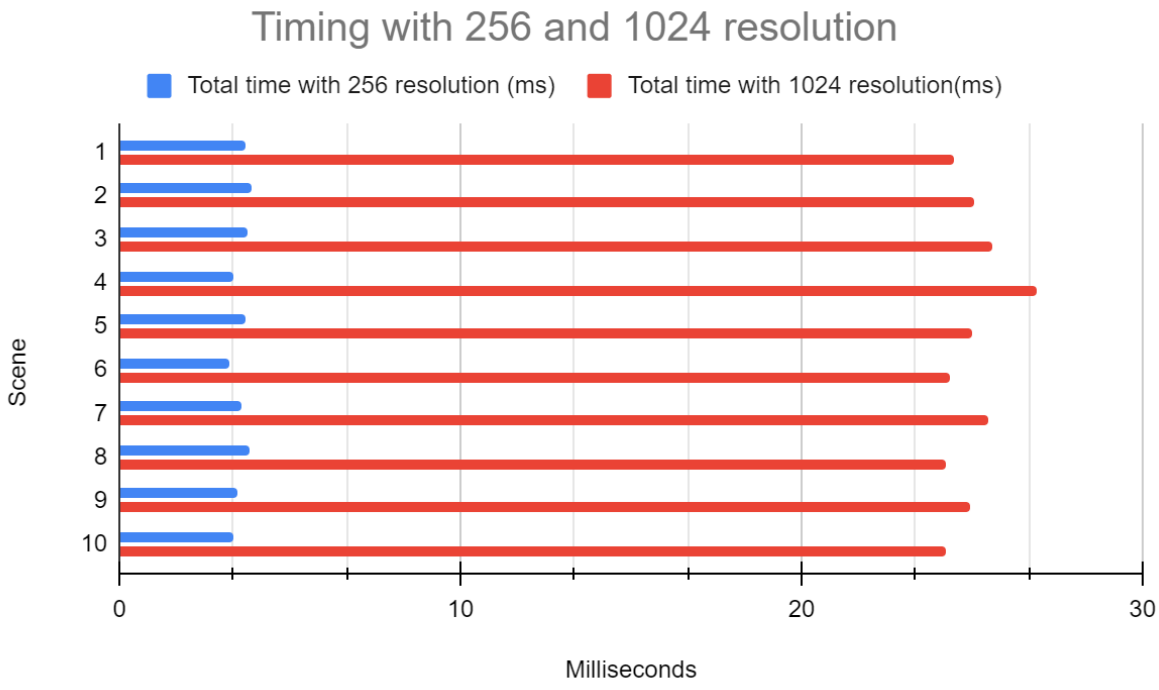


**Figure 17. The results of how much time it takes for the algorithm to finish in 256 and 1024 resolutions.**

The results from the different resolutions show interesting deviations from the low spectrum to the high one. The number of visible machines is much lower in the low-resolution test compared to the high resolution.

## 9.3. Comparison of 3D alternatives

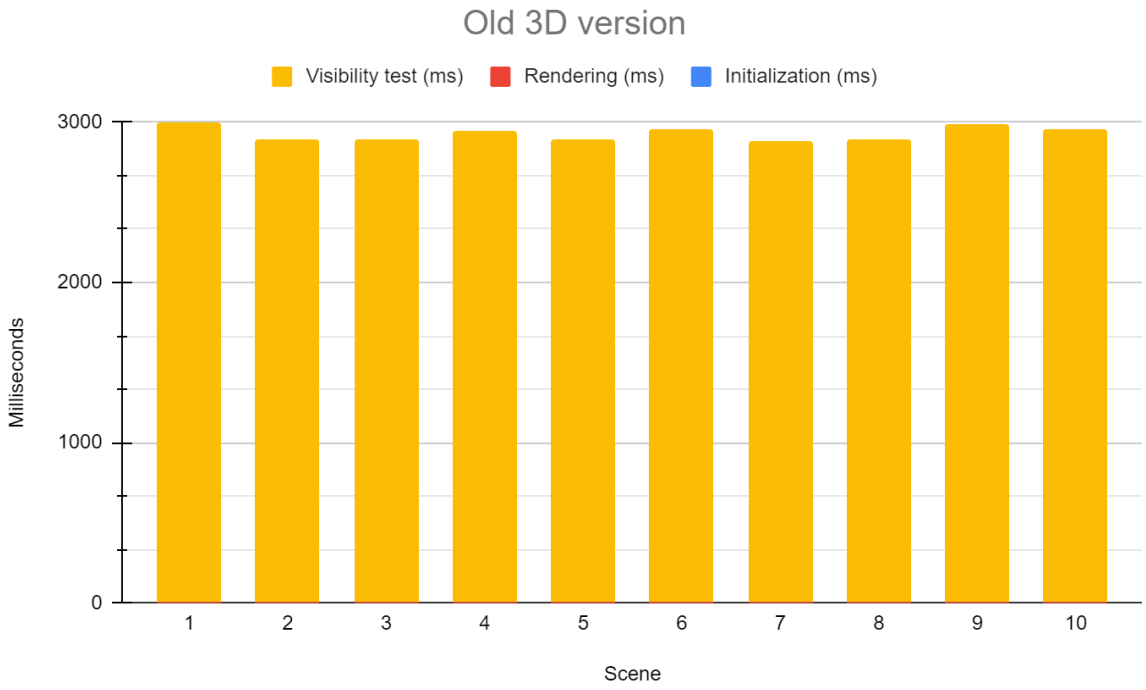


**Figure 18. The results from the Old 3D version split by taks.**

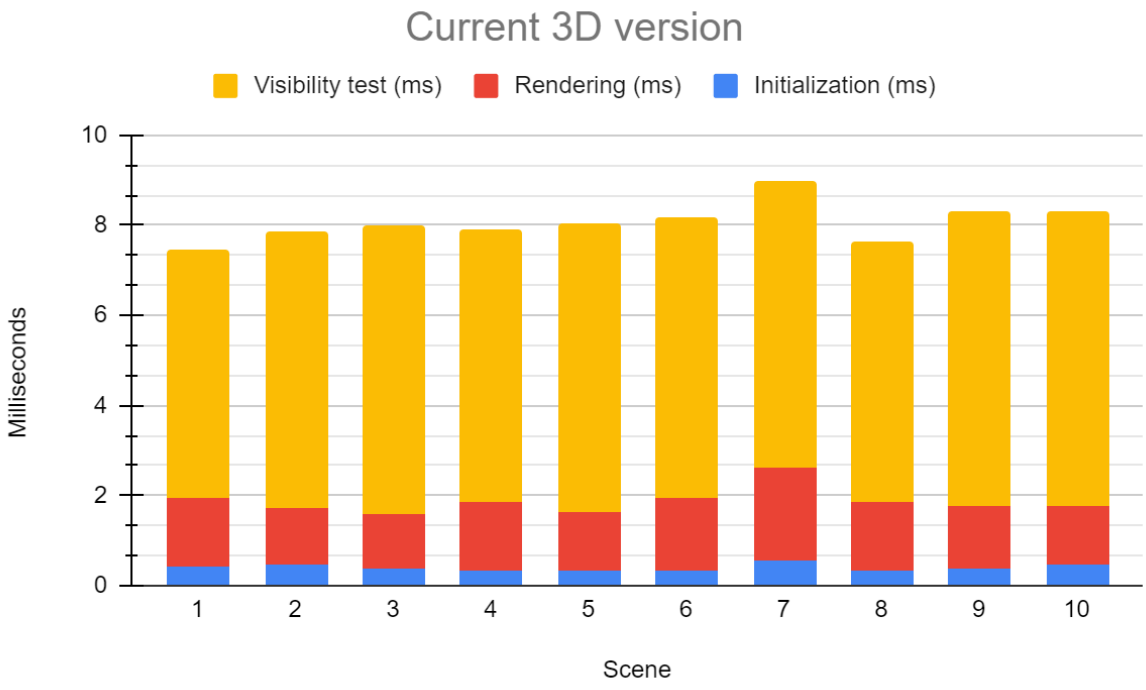


**Figure 19. The results from the current 3D version split by task.**

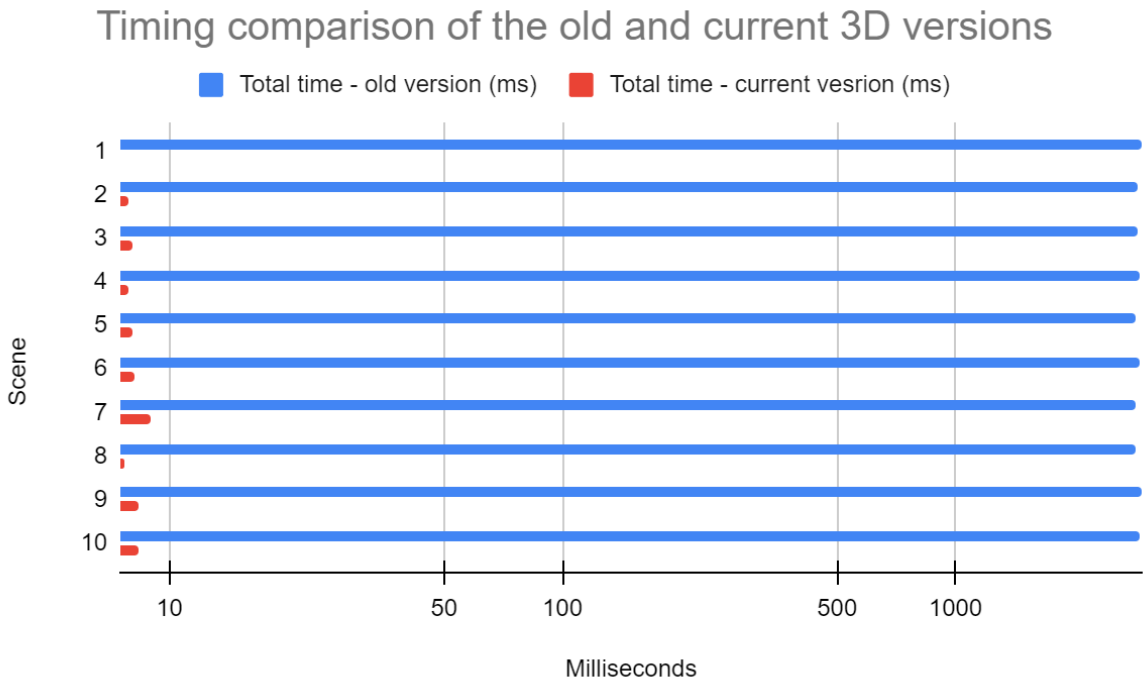


**Figure 20. The results of the time comparison between the old and current version of the 3D algorithm.**

The results of the two 3D versions show a serious difference in runtime with the newer version being much faster than the older one. Also, the distribution of the different tasks appears to be different. In both versions, the rendering and initialization take way less time than the visibility test which seems to take the most time to complete.

## 9.4. Comparison with 2D

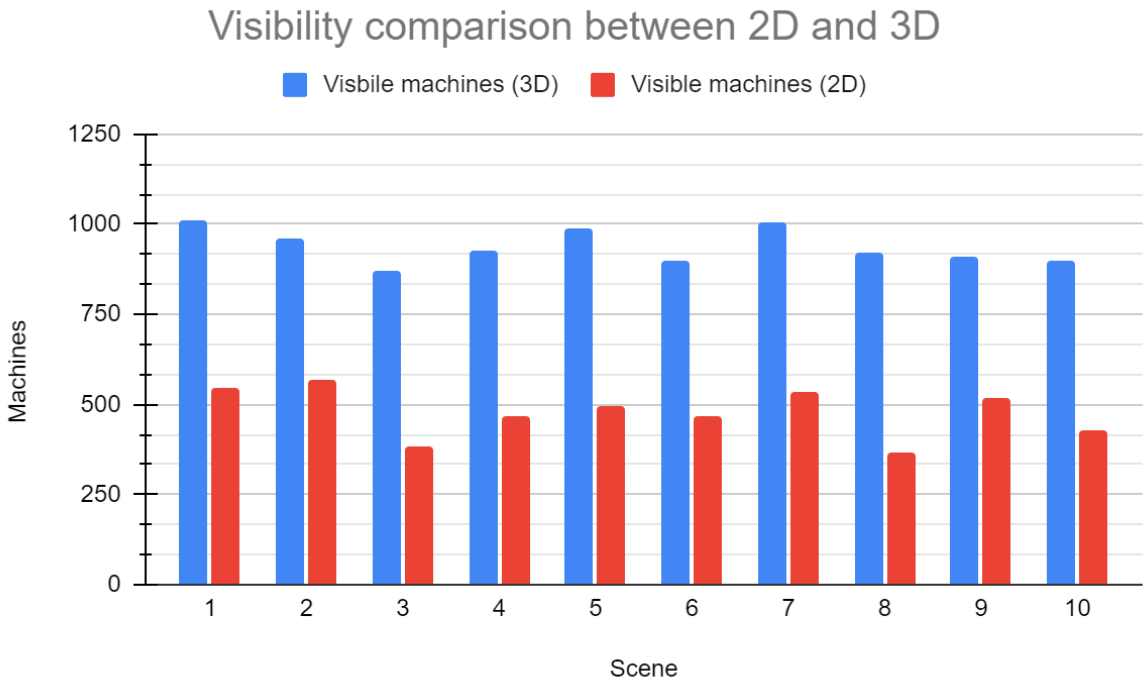


**Figure 21. The results from the visibility comparison between the 2D and 3D approaches.**

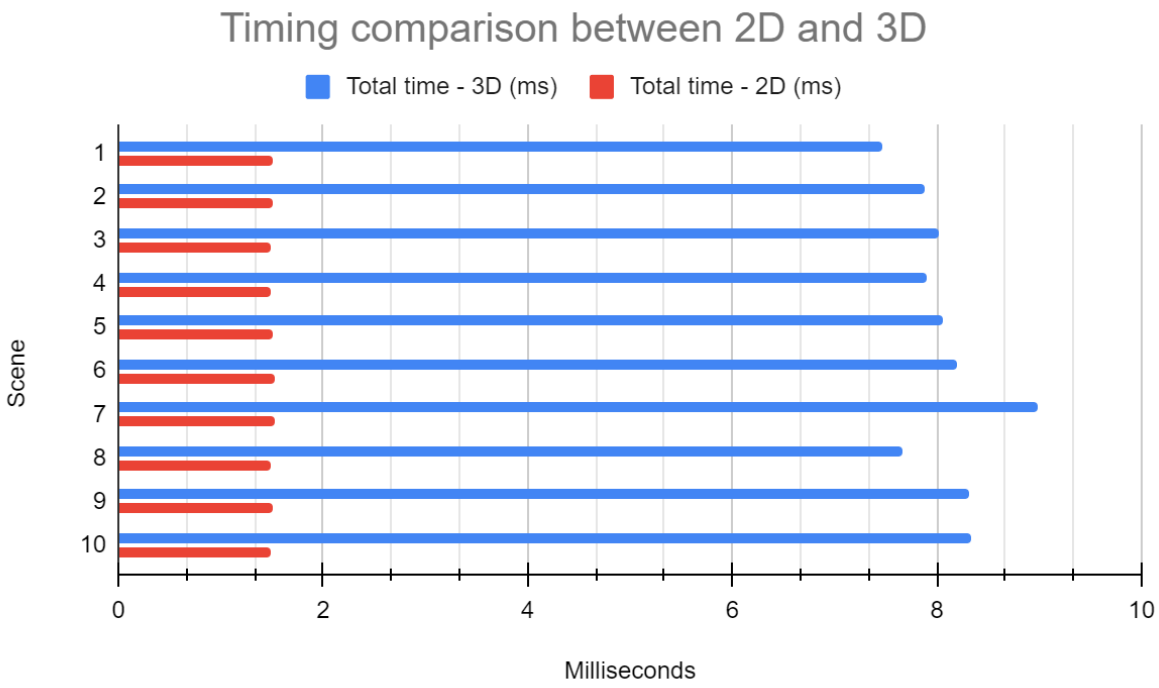


**Figure 22. The results from the timing comparison between the 2D and 3D approaches.**

The results from the 2D and 3D versions appear to be quite different to each other which is understandable since the two techniques have little in common. However, at first glance the 3D algorithm seems to be slower than the 2D but has a higher visibility rate.

# 10. Discussion

The goal of the project is to create a 3d visibility solution that can be used in a highly automated industrial environment. In addition, this solution needs to be examined against a 2D visibility solution comparing the timing and visibility.

- **Implementation**

To begin with, the 3D visibility algorithm works as intended which means that it can detect a machine even if it is partially visible. Moreover, it can use autonomous models that can be placed and animated. The program can also detect to what degree an object is visible. Thus, if an object is too close to the camera it would have a very big fill ratio which means it occupies more pixels in the frustum. It scans spherically, which allows the camera to be placed on any surface, vertical or horizontal. The models representing the machines can be of any degree of detail, from simple AABBs to complicated meshes. In addition, there is a parallel 3D render scene that can be used for verification or even surveillance. The code is optimised for machines of identical shape, making it ideal for an assembly line with machines that share characteristics. It is also important to note that the program uses GPU acceleration when available that makes it fast. The image analysis makes use of multiple cores working fast through even high resolutions. Furthermore, there is a free flight camera that makes it easy for the user to inspect the scene from any angle at real time. Another useful feature is the panoramic view which displays the scene at 360 degrees from the camera position. Thus, being helpful when the user tries to visualise what the camera shows from all angles. Both, free flight, and panoramic views display the machine visibility status with a colour code: Green for visible, red for occluded and displays the walls and floors in grey. This is handy for visualising the visibility test results. Finally, the program can also calculate 2D convex hulls that can be used for alternate two-dimensional analysis.

- **Results**

To begin with, the time and accuracy test of the 3D algorithm has very similar results for each scene because the distribution of the machines over the grid is uniform. Each scene takes 7 to 8 milliseconds to finish which is divided into three steps. The initialization step is mostly setting up global device parameters and allocating buffers taking around 0.4 ms to complete. This step does not take that much time because it does not involve any logic or lookup operations. In fact, there is hardly any branching while initialising these global parameters. In the rendering step there are 10.000 machines to draw that are split in badges of 255 items because this is the OpenGL 2.1 limit. A more modern API would allow for all models to be rendered in a single draw call. This takes double the time that the initialization step takes because it has many items to render. Lastly, the visibility test is the costliest out of the three steps taking around 8 ms. It was expected that this part would be the heaviest because it needs to read the texture pixels one at a time and flag objects as visible, bearing in mind that a 512 x 512 resolution texture has 262.144 pixels.

In general, the accuracy is directly defined by the resolution. The higher the resolution, the more detailed the analysis. Taking a 512 x 512 resolution provides an accuracy of approximately 12 cm at a distance of 100m, meaning that at that distance, a surface smaller than 12 cm across has a 50% chance of being missed by the renderer. The effect of the resolution in accuracy is best demonstrated in the test of figures 16 and 17 where we can see how many more objects become visible once we quadruple the resolution.

All the tests were done using all 6 viewports of the camera which makes the algorithm 2ms slower than when we skip the top and bottom viewports. Depending on where the camera is attached in a real-

life scenario, it might be beneficial to ignore blind angles. The test results in figures 12 and 13 measure the performance gain by ignoring two out of six viewports.

Moreover, the frustum culling optimization that is performed before rendering appears to be saving a good amount of bandwidth as shown in figures 14 and 15. Since the machines are very evenly distributed in space, the gains of applying frustum culling are almost constant regardless of the angle.

- **Comparison of 3D versions**

The first attempt at a three-dimensional visibility solution based on rendering required as many draw calls as there are machines in the scene, making it unscalable when the number of machines were in the thousands. In fact, the profile times were a few seconds long as it can be observed from the test results in figure 18. As if that wasn't disappointing enough, the floating-point errors and optimizations created false positives which required a threshold to be set for when an object can be considered visible. Contrary to the second attempt, where all results are in integer format and not subject to error or optimization. Moreover, the second technique required only a fraction of the draw calls because it could utilise instancing. Had the OpenGL version been newer, it would require even less draw calls. The second technique is heavy on the CPU during the analysis stage but still magnitudes less than the initial one.

- **Comparison of 2D versus 3D**

The 2D approach uses concepts that are specific to its technique and not relevant to the 3D approach. They are thus hardly comparable from a profiling perspective. However, it is apparent that with both techniques, it is possible to produce useful results in real time. The points where the two approaches are the most different are:

- Transparency

The 2D approach has the feature of treating selected objects as transparent meaning that they are visible, but they do not block vision. Transparency is set on objects whose volume are not well represented by a convex or those that hang above the ground leaving everything behind them visible. The 3D approach uses 3D meshes that can have tremendous detail and can support any shape of object. It thus relies on the detail of the mesh to determine to what degree one can see through it or not.

- Supported shapes

The 2D approach supports convex polygons. This means that a non-convex form needs to be replaced with a convex hull before it can be analysed. The 3D approach can support concave meshes.

- Hardware dependency

The 2D approach does not have specific hardware requirements. The faster the hardware, the quicker the analysis. The 3D approach on the other hand, even though it does not require GPU acceleration it would struggle without it in a real-life scenario.

# 11. Conclusions

Taking everything into consideration, it is possible to create a real time 3D emergency stop system that works as well as a 2D approach would. It would be able to solve scenarios where a 2D approach would not even support, such as when there are objects on different heights, windows, or complex shapes. Even though the 3D approach is slower compared to the 2D, the object visibility is much higher. That can be attributed to the intrinsic nature of the 3D approach that reads pixels instead of AABB shapes making it ideal for complex and hollow shapes.

In a real industrial environment, machines are never square but on the contrary, they always have odd shapes with arms, wheels, and wires. In addition, machines can carry objects that effectively change their volume, they can be hanging from the roof or attached to a wall, or they can be on top of other larger machines. None of the aforementioned situations pose a concern to the 3D solution because it is built to work in such circumstances.

In the implementation there was a decision that machines should occlude one another which is not always the case in a real-life scenario. Static environment objects like walls occluding the machines would be a more sensible approach.

In hindsight, OpenGL is not really a plug and play library and it requires a lot of effort to organise and keep stable. An object-oriented graphics library would have been more practical for this project. Almost all the maths functions were written anew for this project which was not time worthy. An intrinsics accelerated, third party library would have been a better choice. In addition, the project description took a few turns during the time frame of the course, which caused some backtracking.

Lastly, one should not use a complicated solution where a simple solution suffices. The 3D solution that was developed and presented in this thesis work was not meant to compete with the 2D solution but to cover cases where extra detail is necessary.

# 12. Future work

Having encountered quite a few hindrances with OpenGL, it could be more useful that in a future version of this project, to use another library, for example DirectX.

Having run into a few obstacles with OpenGL, it could be wise in a future version of this project, to use another library, for example DirectX. Directx has more high-level features and requires less coding making it more suitable for a small scope project. Alternatively, an upgrade to a newer version of OpenGL could improve performance.

Moreover, the machine-to-machine occlusion could be changed in a future version of this project. During the implementation, static objects that exist in the scene should occlude the machines instead of machines occluding one another. This definition could be proven more practical when testing for a real-life emergency stop scenario. In addition, machines that are made up of complex geometries should be profiled in order to test triangle count scalability. In the current profiling tests, only low polygon meshes are tested. Other rendering techniques might be useful for example transparency or skinning.

Also, it would be useful if the application could optionally be supplied with occlusion volumes, so it can focus only on a part of a certain environment. In a scenario where several areas exist in the industrial floor, the user can choose which volume to test by whitelisting the areas to be tested and blacklisting the areas not to be tested.

Another task that can be implemented in a future version is that the worker threads could be reused by pausing and unpausing them using system events, instead of being destroyed and created again at every iteration. Creating threads is a slow operation and it could make a significant difference in high resolutions.

Furthermore, a hybrid solution, featuring both 2D and 3D would be more complete. A 2D approach could be integrated into the same application so it can provide both 2D and 3D techniques on a case-by-case basis.

# References


[1] V. Gobinath, "An Overview of Industry 4.0 Technologies and Benefits and Challenges That Incurred While Adopting It", in *Advances in industrial automation and smart manufacturing: select proceedings of ICAIASM 2019*, A. Arockiarajan, M. Duraiselvam and R. Raju, Ed. Gateway East, Singapore: Springer, pp. 1-12, 2021.

[2] Anonymous,"Understanding and Using E-Stops", *Machine Design*, vol. 90, no. 11, p. 92, 2018. Available: https://www.proquest.com/trade-journals/understanding-using-e-stops/docview/2132638663/se-2.

[3] G. Capannini, J. Carlson and R. Mellander, "Thou Shalt Not Move: A Visibility-based Emergency Stop System for Smart Industries", *7th Conference on the Engineering of Computer Based Systems*, 2021

[4] Y. Zhang, J. Wang, X. Wang and J. Dolan, "Road-Segmentation-Based Curb Detection Method for Self-Driving via a 3D-LiDAR Sensor", *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 12, pp. 3981-3991, 2018. Available: 10.1109/tits.2018.2789462.

[5] J. Zhang, H. Liu and J. Lu, "A semi-supervised 3D object detection method for autonomous driving", *Displays*, vol. 71, p. 102117, 2022. Available: 10.1016/j.displa.2021.102117.

[6] R. Omar and D. Gu, "3D Path Planning for Unmanned Aerial Vehicles using Visibility Line based Method", *ICINCO*, pp. 80-85, 2010.

[7] J. Bittner and P. Wonka, "Visibility in Computer Graphics", *Environment and planning. B, Planning & design*, vol. 30, no. 5, pp. 729-755, 2003. Available: 10.1068/b2957

[8] K. Group, "OpenGL - The Industry Standard for High Performance Graphics", Opengl.org, 2022. [Online]. Available: https://www.opengl.org/.

[9] K. Suita, Y. Yamada, N. Tsuchida, K. Imai, H. Ikeda and N. Sugimoto, "A failure-tosafety "Kyozon" system with simple contact detection and stop capabilities for safe humanautonomous robot coexistence", *Proceedings of 1995 IEEE International Conference on Robotics and Automation*, vol. 3, pp. 3089-3096, 1995.

[10] A. De Santis, B. Siciliano, A. De Luca and A. Bicchi, "An atlas of physical human–robot interaction", *Mechanism and Machine Theory*, vol. 43, no. 3, pp. 253-270, 2008.

[11] Digi-Key's North American Editors, "Reducing Robot Risk: How to Design a Safe Industrial Environment", *https://www.digikey.com/*, 2018. [Online]. Available: https://www.digikey.com/en/articles/reducing-robot-risk-how-to-design-a-safe-industrialenvironment.

[12] "ISO 13850:2015 Safety of machinery — Emergency stop function — Principles for design", *ISO*, 2022. [Online]. Available: https://www.iso.org/standard/59970.html.

[13] J. Bittner and P. Wonka, "Visibility in Computer Graphics", *Environment and planning. B, Planning & design*, vol. 30, no. 5, pp. 729-755, 2003.

[14] C. Hornung, "A Method for Solving the Visibility Problem", *IEEE Computer Graphics and Applications*, vol. 4, no. 7, pp. 26-33, 1984.

[15] G. Lee, M. Jeong, Y. Seok and S. Lee, "Hierarchical Raster Occlusion Culling", *Computer Graphics Forum*, vol. 40, no. 2, pp. 489-495, 2021.

[16] U. Assarsson and T. Moller, "Optimized View Frustum Culling Algorithms for Bounding Boxes", *Journal of Graphics Tools*, vol. 5, no. 1, pp. 9-22, 2000.

[17] G. Lee, Modern Programming: Object Oriented Programming and Best Practices. Packt Publishing, Limited, 2019, pp. 2-50.

[18] "Standard C++", Isocpp.org, 2022. [Online]. Available: https://isocpp.org

[19] J.X. Chen, "Rendering in computer graphics", *Wiley Interdisciplinary Reviews: Computational Statistics 2*, vol. 2, no. 1, pp. 120-126, 2010.

[20] "Visual Studio: IDE and Code Editor for Software Developers and Teams", *Visual Studio*, 2022. [Online]. Available: https://visualstudio.microsoft.com/

[21] "Overview of Visual Studio", *Docs.microsoft.com*, 2022. [Online]. Available: https://docs.microsoft.com/en-us/visualstudio/get-started/visual-studio-ide?view=vs-2022.

[22] J. Rossignac, "Edgebreaker: connectivity compression for triangle meshes", *IEEE Transactions on Visualization and Computer Graphics*, vol. 5, no. 1, pp. 47-61, 1999. Available: 10.1109/2945.764870.

[23] B. Foundation, "blender.org - Home of the Blender project - Free and Open 3D Creation Software", *blender.org*, 2022. [Online]. Available: https://www.blender.org/.

[24] M. Haigh-Hutchinson, *Real-time cameras: a guide for game designers and developers*. Amsterdam: Morgan Kaufmann/Elsevier, 2009.

[25] E. Haines and T. Akenine-Möller, "The Graphics Rendering Pipeline", in *Real-Time Rendering*, 2nd ed., CRC Press LLC, 2002.

[26] "LearnOpenGL - Instancing", *Learnopengl.com*, 2022. [Online]. Available: https://learnopengl.com/Advanced-OpenGL/Instancing.

[27] "LearnOpenGL - Depth testing", *Learnopengl.com*, 2022. [Online]. Available: https://learnopengl.com/Advanced-OpenGL/Depth-testing.

[28] L. Williams, "Pyramidal parametrics", *ACM SIGGRAPH Computer Graphics*, vol. 17, no. 3, pp. 1-11, 1983. Available: 10.1145/964967.801126.

[29] V. Gonakhchyan, "Performance Model of Graphics Pipeline for a One-Pass Rendering of 3D Dynamic Scenes", *Programming and Computer Software*, vol. 47, no. 7, pp. 522-533, 2021.

[30] D. Ebert, T. Komuro, A. Namiki and M. Ishikawa, "Safe human-robot-coexistence: emergency-stop using a high-speed vision-chip", *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2923-2928, 2005. Available: 10.1109/IROS.2005.1545242.

[31] A. Kaur and M. Bhatia, "Stochastic game network based model for disaster management in smart industry", *Journal of Ambient Intelligence and Humanized Computing*, 2021. Available: 10.1007/s12652-021-03090-3.

[32] B. Pasian and R. Turner, "Design Methods and Practices for Research of Project Management", in *Designs, Methods and Practices for Research of Project Management*, Surrey, England: Taylor & Francis Group, pp. 125-136, 2022

[33] "Kursplan - Datorgrafikens grunder", *https://www.mdu.se*, 2022. [Online]. Available: https://www.mdu.se/utbildning/kursplan?id=27374.

[34] *github.com*, 2022. [Online]. Available: https://github.com/Ludds/Cave-Generator/tree/master/Cave-Generator.

# Appendix

In this appendix, the tables from the results are displayed in order to give a more in depth understanding of the resulting figures. Moreover, the source code of the project can be found in GitHub at: https://github.com/Didourbidi/Bachelor_Thesis_Git

| Time and visibility with upwards and downwards viewports | | |
|---|---|---|
| Scene | Visible machines | Total time (ms) |
| 1 | 1009 | 7.465 |
| 2 | 959 | 7.871 |
| 3 | 869 | 8.01 |
| 4 | 926 | 7.897 |
| 5 | 986 | 8.049 |
| 6 | 897 | 8.187 |
| 7 | 1007 | 8.993 |
| 8 | 920 | 7.662 |
| 9 | 911 | 8.315 |
| 10 | 900 | 8.328 |

**Table 1. Results of time and visibility with 6 viewports.**

| Time and visibility without upwards and downwards viewports | | |
|---|---|---|
| Scene | Visible machines | Total time (ms) |
| 1 | 1009 | 6.498 |
| 2 | 959 | 6.628 |
| 3 | 869 | 6.602 |
| 4 | 926 | 6.564 |
| 5 | 986 | 6.293 |
| 6 | 897 | 7.133 |
| 7 | 1007 | 6.056 |
| 8 | 920 | 6.446 |
| 9 | 911 | 6.749 |
| 10 | 900 | 6.757 |

**Table 2. Results of time and visibility with 4 viewports.**

| Time and visibility with frustum culling | | |
|---|---|---|
| Scene | Visible machines | Total time (ms) |
| 1 | 1009 | 7.465 |
| 2 | 959 | 7.871 |
| 3 | 869 | 8.01 |
| 4 | 926 | 7.897 |
| 5 | 986 | 8.049 |
| 6 | 897 | 8.187 |
| 7 | 1007 | 8.993 |
| 8 | 920 | 7.662 |
| 9 | 911 | 8.315 |
| 10 | 900 | 8.328 |

**Table 3. Results of time and visibility with frustum culling.**

| Time and visibility without frustum culling | | |
|---|---|---|
| Scene | Visible machines | Total time (ms) |
| 1 | 1009 | 12.404 |
| 2 | 959 | 12.866 |
| 3 | 869 | 13.305 |
| 4 | 926 | 12.54 |
| 5 | 986 | 12.806 |
| 6 | 897 | 12.766 |
| 7 | 1007 | 12.557 |
| 8 | 920 | 12.709 |
| 9 | 911 | 12.999 |
| 10 | 900 | 12.978 |

**Table 4. Results of time and visibility without frustum culling.**

| Time and visibility with 256 resolution | | |
|---|---|---|
| Scene | Visible machines | Total time (ms) |
| 1 | 708 | 3.678 |
| 2 | 680 | 3.901 |
| 3 | 623 | 3.782 |
| 4 | 655 | 3.352 |
| 5 | 703 | 3.697 |
| 6 | 644 | 3.21 |
| 7 | 686 | 3.581 |
| 8 | 671 | 3.788 |
| 9 | 645 | 3.433 |
| 10 | 640 | 3.312 |

**Table 5. Results of time and visibility for 256 resolution.**

| Time and visibility with 1024 resolution | | |
|---|---|---|
| Scene | Visible machines for 1024 resolution | Total time (ms) |
| 1 | 1251 | 24.468 |
| 2 | 1201 | 25.082 |
| 3 | 1119 | 25.592 |
| 4 | 1155 | 26.903 |
| 5 | 1206 | 24.997 |
| 6 | 1151 | 24.345 |
| 7 | 1217 | 25.46 |
| 8 | 1147 | 24.218 |
| 9 | 1136 | 24.93 |
| 10 | 1098 | 24.233 |

**Table 6. Results of time and visibility for 1024 resolution.**

| Time and visibility - 3D version | | | | | |
|---|---|---|---|---|---|
| Scene | Visbile machines | Initialization (ms) | Rendering (ms) | Visibility test (ms) | Total time (ms) |
| 1 | 1009 | 0.437 | 1.493 | 5.533 | 7.465 |
| 2 | 959 | 0.459 | 1.252 | 6.158 | 7.871 |
| 3 | 869 | 0.376 | 1.228 | 6.404 | 8.01 |
| 4 | 926 | 0.335 | 1.52 | 6.04 | 7.897 |
| 5 | 986 | 0.337 | 1.318 | 6.392 | 8.049 |
| 6 | 897 | 0.349 | 1.599 | 6.237 | 8.187 |
| 7 | 1007 | 0.568 | 2.057 | 6.367 | 8.993 |
| 8 | 920 | 0.323 | 1.534 | 5.803 | 7.662 |
| 9 | 911 | 0.391 | 1.367 | 6.555 | 8.315 |
| 10 | 900 | 0.47 | 1.296 | 6.56 | 8.328 |

**Table 7. Results of the timing for the different parts of the 3D algorithm.**

| Time and visibility - 2D version | | | | | | |
|---|---|---|---|---|---|---|
| Scene | Visible machines | Collision detection (ms) | Preprossesing (ms) | Sorting (ms) | Sweeping (ms) | Total time (ms) |
| 1 | 544 | 0.42 | 0.506 | 0.262 | 0.321 | 1.509 |
| 2 | 566 | 0.416 | 0.509 | 0.262 | 0.329 | 1.516 |
| 3 | 384 | 0.412 | 0.506 | 0.263 | 0.303 | 1.484 |
| 4 | 466 | 0.409 | 0.5 | 0.257 | 0.322 | 1.487 |
| 5 | 497 | 0.428 | 0.51 | 0.263 | 0.314 | 1.515 |
| 6 | 468 | 0.425 | 0.521 | 0.264 | 0.317 | 1.527 |
| 7 | 536 | 0.426 | 0.509 | 0.267 | 0.331 | 1.533 |
| 8 | 366 | 0.424 | 0.511 | 0.264 | 0.296 | 1.495 |
| 9 | 516 | 0.425 | 0.509 | 0.262 | 0.313 | 1.509 |
| 10 | 428 | 0.425 | 0.509 | 0.259 | 0.301 | 1.494 |

**Table 8. Results of the timing for the different parts of the 2D algorithm.**