FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

ASSIGNMENT OF BACHELOR’S THESIS

Title:	Optimization of explosion computing in VBS4
Student:	Radovan Netík
Supervisor:	Ing Filip Hřebačka
Study Programme:	Informatics
Study Branch:	Web and Software Engineering
Department:	Department of Software Engineering
Validity:	Until the end of summer semester 2020/21

Instructions

VBS4 is a realtime army simulation game that is being developed by Bohemia Interactive Simulations, k. s. This program has a slow algorithm to calculate explosions and effects of these explosions on objects placed on the game map. The main goal is to create a better algorithm to improve the user experience from playing VBS4.

1. Perform a search for existing ways to calculate explosions.
2. Analyze physics behind explosions.
3. Design a plug-in using Gears Studio that will manage explosion calculations.
   a) Focus on physical accuracy and performance.
   b) Users have to see the difference between the old implementation and the created implementation.
4. Analyze hardware possibilities.
5. Appropriately test the implementation.
6. Demonstrate the VBS4 improvements with the created plug-in.

References

Will be provided by the supervisor.

Ing. Michal Valenta, Ph.D.
Head of Department

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
Dean

Prague February 6, 2020

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Bachelor's thesis

Optimization of explosion computing in VBS4

Radovan Netík

Department of Software Engineering
Supervisor: Ing. Filip Hřebačka

June 4, 2020

Acknowledgements

I would like to thank my supervisor, Ing. Filip Hřebačka, for his guidance through each stage of the process.

Declaration

In Prague on June 4, 2020 . . . .

Citation of this thesis

Netík, Radovan. *Optimization of explosion computing in VBS4*. Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2020.

Abstrakt

Tato práce prezentuje základní chemické a fyzikální vlastnosti explozí společně s algoritmy které mají využití v počítačové grafice. Tyto znalosti jsou poté využity k popisu algoritmu, který dokáže simulovat efekty exploze na okolní objekty. Výzkum obsahuje studie a matematické definice ohledně problémů, které se musejí řešit při těchto simulacích. Prezentované algoritmy se poté dají vyžít k implementaci přesných a rychlích výpočtů efektů které způsobují výbušné zbraně.

Klíčová slova VBS4, Exploze, Fragmentace, Počítačová grafika, Válečné hry

Abstract

This work declares the basics of physical and chemical properties of explosions and the algorithms used in computer graphics to create the real-time simulation of explosions. The research contains studies about the problems of these simulations and also math equations to help implement algorithms presented in the realization part. The algorithms presented can be used to simulate precise and relatively fast calculations of damage done by an explosion.

Keywords VBS4, Explosions, Fragmentation, Computer Graphics, Wargames

Contents

List of Figures

List of Tables

Introduction

Bohemia Interactive Simulations (BISim) is an Australian software company founded in 2001 that is specialized in creating battlespace simulations. The first designed product was VBS1 back in 2004. VBS1 (Virtual battlespace) was an army simulation game that aimed for tactics and military training. After that, the company developed VBS2 and VBS3, and today the company is developing VBS4. All these simulation games stand on the VBS1 engine, and because VBS4 is a significant step from all previous games, it needs lots of upgrades. One of the upgrades is the simulation of explosions and the effects of explosions on objects placed on a game map. The maps in previous virtual battlespace games were limited so the user can play only in already created areas, but VBS4 is using whole-earth rendering that allows users to move all across the globe. Because of this, the game has more objects to deal with, and the algorithm that was used to calculate explosion effects is not capable of handling this in a reasonable time.

Chapter 1

Goal

The goal of this bachelor's thesis explained in one sentence is to create the algorithm for computation of what objects are affected by the explosion.

The first category, named State-of-the-art, presented in this thesis is about resources used to create a plugin where the algorithm was developed. In the second category called Analysis and design, the goal is separated into two parts. The first part is about to study the physical and chemical properties of explosions along with other effects that are present during the detonation of explosive army weapons (fragmentation, ...). The second part is about the research of algorithms used in computer graphics together with already existing ideas used to calculate explosions effects.

The third category is The Realisation chapter, where the algorithm created for this thesis is presented as the primary goal. The purpose of the algorithm is to calculate what happens to objects around the explosion and how much they are affected. The algorithm will then be compared with the previous VBS4 implementation in areas like time complexity and precision. This will be demonstrated in real-time user experience or by automated testing.

Chapter 2

State-of-the-art

This chapter will explain the technology and interface used in this thesis. First, the history and the actual state of the military simulations will be covered. Second, there will be a description of the VBS engine and Gears Studio.

2.1 Military Simulations

Military simulations, commonly known as war games, are games where the main goal is to create a simulation to perform warfare theories. They are a useful way to develop tactical and strategic solutions without the need for actual hostilities. They can simulate a broad spectrum of army operations, such as field exercise or command post-exercise. The first known game modeled on war was a board game called Petteia created in the 5th century B.C [1]. Then, in the 6th century A.D., the commonly known game called Chess was built [1]. In the 19th century, the Prussian army made the Kriegssipiel, which is the first game that can be named as a wargame [1]. Now, we have more advanced wargames that are using new technologies to introduce the player into military conflicts. In 1995, the U.S.Marines developed a game called Marine Doom - the first computer-simulated combat simulator, which was a hack of the popular video game Doom II [1].

Figure 2.1: Screenshot from the Doom II modification called Marine Doom.

Because technology became more and more powerful, the needs for larger-scale and reality-like army simulations became standard. The software companies realize this and start developing more sophisticated wargames, where the armies can simulate war exercise on the whole earth, and players can use virtual reality headsets.

2.2 VBS4 (Virtual Battle Space 4)

VBS4 is a wargame that was first released in December of 2019. It is a successor of VBS3. VBS has been continuously developed since Operation Flashpoint (OFP). VBS1 was created using code from OFP, and VBS2 was created by modifying ArmA2. VBS3 is a successor of VBS2 and first introduced the implementation of PhysX and some other features, like procedurally generated snow or large maps with detailed graphics (2000 km x 2000 km). The main difference between VBS3 and VBS4 is that VBS4 starts using VBS Blue IG. VBS Blue IG is a high-performance, CIGI-compliant, 3D whole-earth image generator designed to support the full spectrum of land, sea, air and space use cases for collective and individual training [2].

The game itself allows a user to create any imaginable scenario. It can create "Battlespaces', which is a collection of terrain edits, missions plans, scenario files, and after-action reviews [3]. A great feature is that a large portion of attributes can be modified in real-time. Because of the after-action review, missions can be replayed many times for lessons to learn [3].

Figure 2.2: View on the air fighter flying over mountains in VBS4.[3]

About technology, the game itself is written in C++. The DirectX library handles the graphic, and Nvidia PhysX takes responsibility for physics. The game should be officially finished before the summer of 2021.

2.3 Gears Studio

Gears is a software development framework that defines a standard way for components to communicate through formal interfaces [4]. It uses component-based architecture to build applications for systems. Individual components communicate thought formally interfaces [4]. This results in that functionality of components can be reused for multiple systems.

Bohemia Interactive Simulations are developing Gears Studio since 2015, and it is widely used in VBS4.

Chapter 3

Analysis and design

3.1 Definition of Explosions

As the first part of this thesis, it is needed to introduce the reader to the physical model and the chemical behavior of blasts and explosions. Overall, the physical model is the core thing to have for almost every simulation of the real world. Without it, there is a small probability that the final solution will be accurate. On the start, a short description of chemical reactions is covered, and then the physical model is explained.

Figure 3.1: Example of warhead explosion that intersects with tank.[5]

3.1.1 What is explosion

An explosion is a rapid increase in volume and release of energy in an extreme manner [6]. In moments of explosion, the gases are released and, depending on the type of blast, the supersonic shock wave can happen. There exist different types of explosions, but for this thesis are important chemical explosions known as exothermic reactions. They are mostly used in explosive weapons.

When the exothermic reaction happens, the great quantity of energy can be deposited into a relatively small volume, then the manifest itself by a rapid expansion of hot gases, which in turn can create a shock wave or propel fragments outwards at high-speed [7]. The first explosives use gunpowder as the main force of power, but the first widely used explosive weapon was dynamite patented in 1867 by the Swedish chemist and engineer Alfred Nobel.

In wargames, there are many types of explosive weapons, from light armaments (hand-held weapons) to long-range weapons (missiles, rockets, ...). These differ by the ways damage is dealt with surrounding objects. For example, the grenade can use fragmentation to cause harm, and missiles can damage the same way or use a blast or shaped charge.

3.1.2 TNT Equivalent

Chemical reactions and physics of explosions will be demonstrated using secondary explosive material called TNT (2,4,6 — trinitrotoluene). Because the behavior of TNT is well explored, the energy density of other types of explosions (nuclear, thermonuclear, ...) is measured using TNT equivalent. TNT equivalent is a convention for expressing energy [9]. One t (a ton of TNT) has an energy of 1.168 MW h or 4.184 GJ. Conversion factors for other types of explosive materials are given in table 3.1.

Explosive	TNT equivalent
TNT	1.000
RDX (Cyclonite)	1.185
PETN	1.282
Compound B (60% RDX 40% TNT)	1.148
Pentolite 50/50	1.129
Dynamite	1.300
Semtex	1.250

Table 3.1: Table of TNT equivalent examples

This means, for example, that 200 kg of Semtex has a TNT equivalent about 200 * 1.250 = 250 kg TNT.

3.1.3 Chemical properties of explosion

Damage can be caused by different types of warheads but in the most general sense, the damage is caused by the transfer of energy from the warhead to the target [7]. The power takes the form of a shock wave or the kinetic energy of fragments [7]. In most warheads, the energy is stored in the form of chemical reactions. Here is an example of what will happen upon detonation of TNT that goes through a decomposition equivalent to the reaction:

$$2\,C_7H_5N_3O_6 \longrightarrow 3\,N_2 + 7\,CO + 5\,H_2O + 7\,C$$

A significant positive entropy change drives the explosion. Which means a release of gases such as Nitrogen (N2), Hydrogen (H2) and Carbon Monoxide (CO). Production of Carbon (C) adds a sooty appearance to TNT explosions [10]. Rapidly expanded gas compress the surrounding air and creates a shock wave that travel outwards near the detonation velocity of approximately 6.4 km s [10]. There is also high exothermic change (great release of heat) that makes the process thermodynamically favorable.

Released shock wave (blast wave) is almost the same as an ordinary acoustic wave created by sound, but it has great energy and limited duration. The shock wave effect goes through a few phases over time. The start phase is a positive phase that has risen to a peak pressure (up to 19.13 GPa for 1 TNT [11]) and linear decay to atmospheric pressure. As seen in figure 3.2, the peak pressure is achieved almost instantly after detonation. The second is negative phase and has pressure that is lower than atmospheric pressure and then converges gradually back to atmospheric pressure [8].

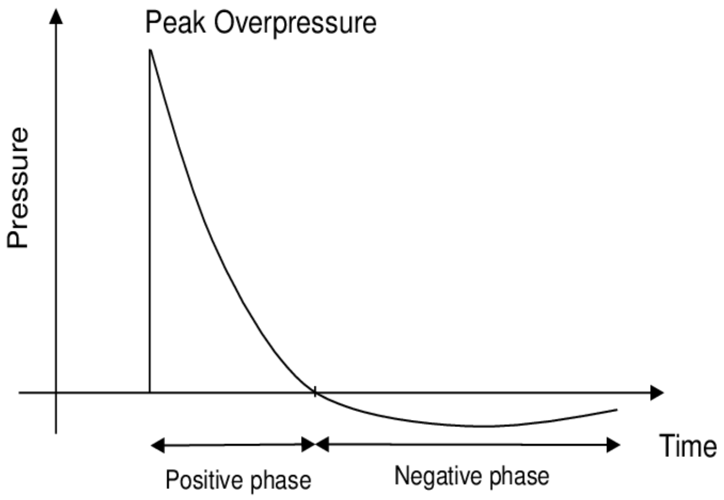

Figure 3.2: Pressure in time after detonation

3.1.4 Shockwave

The damage that has the most significant impact on fixed objects is done by blast shockwave [1] . After detonation, the shockwave travels to all directions in shape based on the strength of the explosion and distance from the epicenter. It is also influenced by the objects and terrain it collides. Those objects are then also affected by it, and it can result in different scenarios - from nothing special up to total destruction of collided objects. Besides, the reflected shock wave is created after the impact.

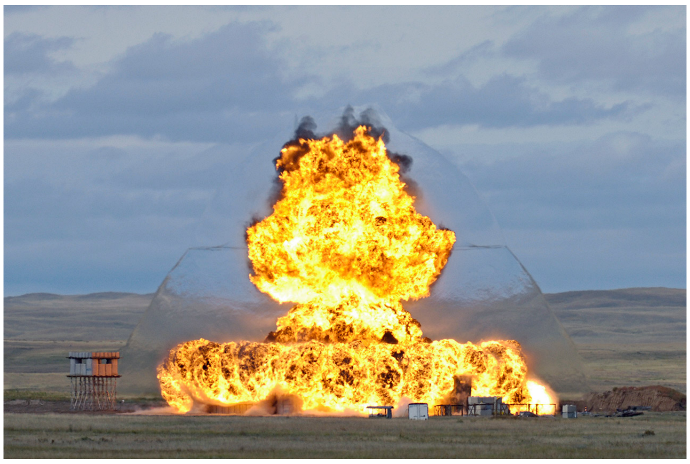
Figure 3.3: Demonstration of the of shockwave after explosion.

Fortunately, there exist ways to calculate these effects without the need to simulate it in the real world. The most known and widely used calculations are from Charles Kingery, who developed with his coworker Gerald Bulmash equation called Kingery-Bulmash equation. This equation is a polynomial function that can approximate properties listed with symbols and units in table 3.2.

[1] To kill a man, it is needed around 200 kPa compared to just 83 kPa that is needed for destroying a building.

Symbol	Name	Unit
P_s	Peak Incident Overpressure	MPa
P_r	Peak Reflected Overpressure	MPa
$I_s/W^{1/3}$	Incident Impulse	$\mathrm{MPa} \cdot \mathrm{ms/kg}^{1/3}$
$I_r/W^{1/3}$	Reflected Impulse	$\mathrm{MPa} \cdot \mathrm{ms/kg}^{1/3}$
$T_a/w^{1/3}$	Arrival Time	$\mathrm{ms/kg}^{1/3}$
$T_o/w^{1/3}$	Positive Phase Duration	$\mathrm{ms/kg}^{1/3}$
U	Shock Velocity	m/ms

Table 3.2: Table of units defined by Kingery.

Kingery-Bulmash equations are defined as

$$Y = 10^{(C_0 + C_1 U + C_2 U_2 + \ldots + C_n U_n)}$$

Where Y is one of the Units listed in list 1, C_0 C_n are constants and U is

$$U = K_0 + K_1 \log Z$$

Where $K_0 \ldots K_1$ are constants and Z is the common logarithm of scaled distance described by:

$$Z = \frac{R}{W^{1/3}}$$

Where R is distance from epicenter in meters, W is the TNT weight, which can be calculated using the already introduced TNT equivalent.

Because the Kingery-Bulmash equations were created based on real life observations, they involve a lot of constants and also the polynomial degree for properties can differ. This can be unpleasantly to implement it using a programming language. So in 2017, a Kingery-Bulmash modified equation of shock wave parameters was created by Doojin Jeon, KiTae Kim and SengEul Han at Inha University in Korea. There were established conditions for this modified equation. Some of them were that the range of scaled distances must be the same as in the conventional equation, the calculation of variable U must be omitted and the number of constants in the modified equation must be less or equal to five and the constants [8].
Modified Equation is then defined as:

$$Y = 10^{(C_0 + C_1 \log Z + C_2 (\log Z)^2 + C_3 (\log Z)^3 + C_4 (\log Z)^4)}$$

Where: $C_0 \ldots C_n$ are constants, and Z is called scaled distance.

3.1.5 Fragmentation

Blast waves can cause a lot of damage, but the strength of it decays fast. Considering this, the military using fragmentation. The fragmentation term refers to the act or process of breaking or making something break into small pieces or parts [12]. Weapons that use fragmentation have a relatively small explosive charge necessary to fragment the metallic shell of the ammunition and to give the fragments the required initial speed to fly to a relatively long distance, to hit and incapacitate the objects in the distance [13]. The probability of lethal effect of pieces on a random person can be achieved in a few seconds if that person is hit by at least one fragment that has a kinetic energy of at least 78 Joules [13]. This value is called the lethal threshold value, and it is an essential value for the construction of fragmentation ammunition. Example of fragmentation is demonstrated using computer on Figure 3.4.

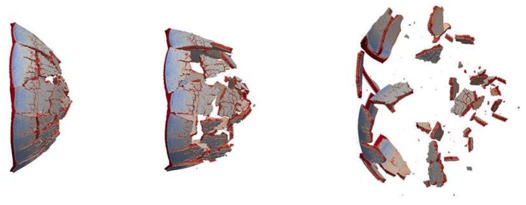

Figure 3.4: Simulation of shell fragmentation.[14]

The fragments are projectiles. The damage they inflict on impact and the distance they travel are functions of their initial kinetic energy [15].The nature of the explosive charge and its case are used to calculate the theoretical velocity of the fragments.

$$v^2 = 2\Delta E \frac{m_c/m_e}{1 + K(m_c/m_e)}$$

In this equation the quantity ΔE is the heat of explosion of the explosive charge [15]. This value is for example $2.175 * 10^6$ J/kg for TNT. The value m_c is the mass of explosive charge and m_e is the mass of the charge case. The quantity K is a geometrical constant [15]. The values of this constant are shown in table 3.3 and are based on the shape of the case.

Once the initial velocity of the fragments is known, they can be treated like any other projectile [15]. The aerodynamics and gravity forces will slow down the fragment the same way as a projectile.

Shape	K
Flat plate	1/3
Cylinder	1/2
Sphere	3/5

Table 3.3: Table of K constant based on shape

There are some assumptions that can be made to decide if the projectile hits the target. The distribution of fragments is assumed to be spherical and also that they are distributed uniformly around the sphere. The sphere has surface area equaled to

$$A = 4\pi r_2$$

The number of fragments per unit area, n is then equal to the total number of fragments, N, divided by surface area of the sphere [15].

$$n = \frac{N}{4\pi r^2}$$

Using this equation is then possible to approximate the total number of fragments that hit an object. It is done by multiplying the total area of that object by the number of pieces in one area unit.

3.2 Data structures and algorithms

This part is describing computer graphics algorithms that can be used to solve the problem of searching the objects influenced by the explosion.

3.2.1 Octree

An octree is a hierarchical spatial structure which is widely used in Computer Graphics algorithms. [16]. It is used to speed up many applications, from searching objects in a given space or computation of intersections between octree voxels and straight lines.

The study of octrees was first published in 1980 by Donald Meagher at Rensselaer Polytechnic Institute. The following description is based on his research.

3.2.1.1 Representation

Octree utilizes a hierarchical N-dimensional binary tree that represents an N-dimensional space [17]. Since VBS4 is a three-dimensional game, a 3-D binary tree can be used and will be described here. Each internal node is the parent of 8 children nodes at the next lower level forming an 8-ary tree or “octree” [17]. Region of space is represented by a node and has one or more values that define the region. If the value of the node completely defines the area, it is a terminal node (leaf). If not, an ambiguity exists and the node contains eight children, which represent the eight subregions or octants of the parent node [17].

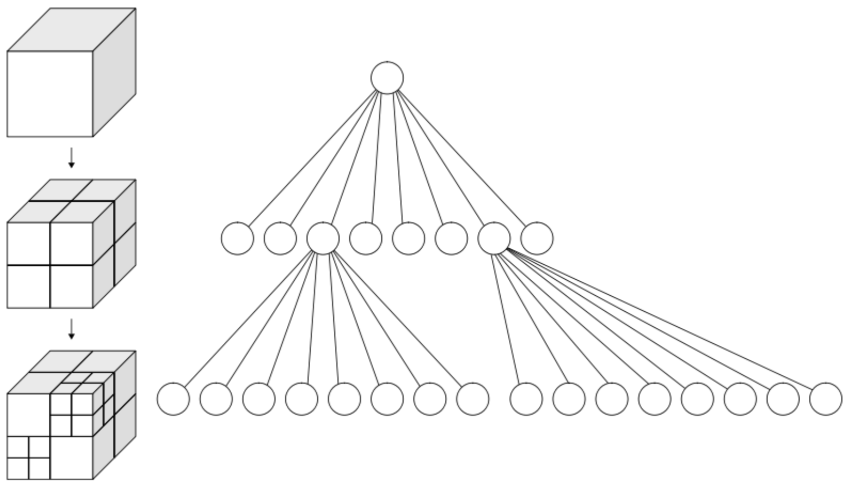

Figure 3.5: Representation of first three levels of octree.

3.2.1.2 Adventages

There are several advantages of this data structure. First, only a compact set of manipulation algorithms is required for separate space to small regions and all these regions have a single primitive shape, the cube. A second advantage arises because of the hierarchical structure [17]. The root node represents the entire object [17]. The other nodes at some level, together with the higher nodes, completely describe space as the whole. Thus, algorithms can operate at a level appropriate for the task and avoid the bulk of data contained in a lower level. Another advantage is that objects are partially pre-sorted. By traversing the tree in proper sequence, the regions of the tree will be visited in a uniform direction.

3.2.1.3 Examples of usage

Collision checks
: Octree can be used to quickly check collision between objects such as dynamic entities (soldiers, animals, etc.) and static objects (house, wall, fence, etc.).

Color quantization
: The advantage of the octree for color quantization is that it is simple to generate both a good partitioning of the color space and a fast inverse color table to find the color index for each pixel in the image [18].

3.2.2 Quadtree

An octree is a hierarchical spatial structure that is widely used in fields like image processing. The stand-alone structure will not be explained in detail because it is similar to the Octree data structure described before. The difference is that quadtree uses two-dimensional space. This means that every node of the structure resembles the square. Every parent node of area N has four sons area $N/4$. The hierarchy is visible in Figure 3.6.

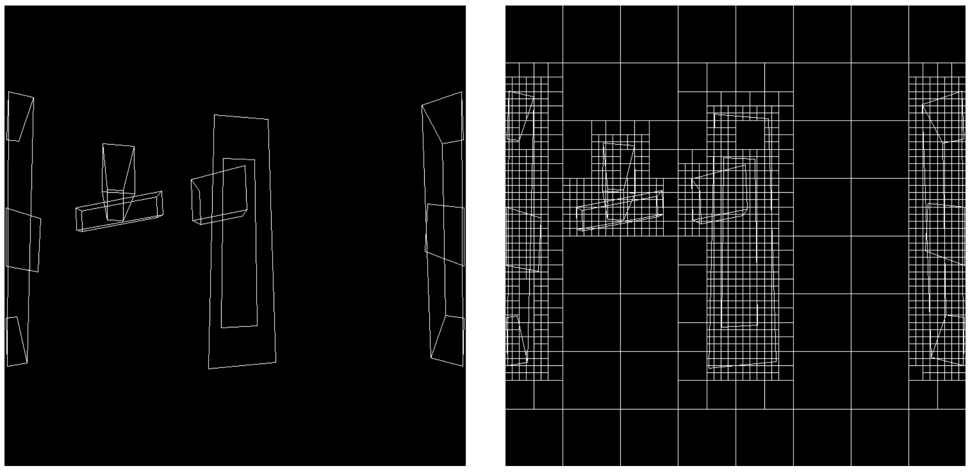

Figure 3.6: Vizualization of the quadtree during rasterization.

The advantages of quadtree are almost the same as octree advantages but can be applied only on two-dimensional space. Mentioned image processing uses quadtrees for operations that need to locate some areas of the image quickly.

3.2.3 Line-Plane Intersection

This section is about finding a cross point between the line and the plane. The method presented is using basic linear algebra to calculate the point coordinates in 3D space. The result of an intersection can be determined by one equation. But to fully understands it, it is needed to explain how to have line and plane defined and also what means the term vector projection.

Many ways can define the position and direction of the line. Yet for the presented method is essential to describing the line either by two points or a point and a vector. The plane will be defined using two properties—the normal vector n of this plane and the point c that lies on this plane.

The vector projection is an equation that takes two vectors $\vec{a}$,$\vec{b}$, and project one of them on the second, for example, *veca* to $\vec{b}$. This can be better understood by looking at figure 3.7 .

As can be seen, the result of vector projection gives a new vector $\vec{a}$ on the vector $\vec{a}$ or $\vec{b}$. Vector projection equation:

$$\vec{a_1} = \vec{b}\frac{\vec{a} \cdot \vec{b}}{||\vec{b}|| \quad ||\vec{b}||}$$

The point of the intersection I is then calculated using equation:

$$I = x_0 + k\vec{v}$$

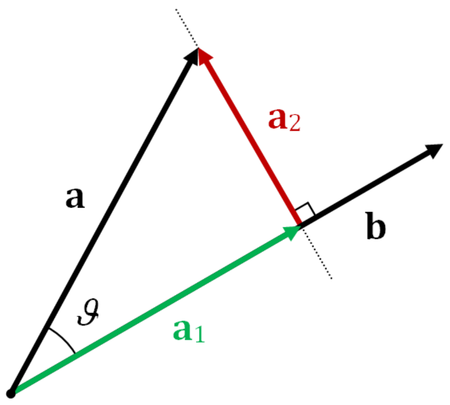

Figure 3.7: Vector Projection

Where the x_0 is the starting point of line and $\vec{v}$ is the is the vector difference between the line startpoint and endpoint.

The variable k is calculated using the ratio between the vector $\vec{w'}$ and the vector $\vec{v'}$ which are results of vector projections defined as:

$$\vec{v'} = \vec{v}\frac{\vec{v}\cdot\vec{n}}{||\vec{n}|| \quad ||\vec{n}||} \quad and \quad \vec{w'} = \vec{w}\frac{\vec{w}\cdot\vec{n}}{||\vec{n}|| \quad ||\vec{n}||}$$

The $\vec{n}$ is defined as the normal vector of the plane and the $\vec{w}$ is the vector difference between the line startpoint and the point c that is on the plane.

If this is substituted into $I = x_0 + k * \vec{v}$ and then simplified, the final definition of I is

$$I = X_0 + \frac{\vec{w}\cdot\vec{n}}{\vec{v}\cdot\vec{n}}\vec{v}$$

3.2.4 Point in polygon

Now, that the process of getting the point that intersects the plane and line is known. The line-convex polygon intersection needs to be introduced. Firstly, the term polygon will be explained and then take place the algorithm that can determine if a random point is inside a convex polygon.

By definition, the polygon is the closed plane figure bounded by three or more line segments [23]. Convex polygon means that every angle between the connected line segments is lower than 180 degrees. One of the useful properties of a polygon is that is should always be planar. That means all line segments are on the same plane. In VBS4 are used convex polygons that have three or four segments. Because of that, the presented algorithm is most useful for a three-sided convex polygon, alias triangle. But can be easily adapted for a four-sided convex polygon.

The algorithm itself works with a triangle using the barycentric coordinates. Barycentric coordinates are triples of numbers (a, b, c) corresponding to masses placed at the vertices of a triangle ΔABC [24]. The point P determined by the (a, b, c) is the geometric centroid. The vertices of the triangle are given by (1,0,0),(0,1,0),(0,0,1).

When it is needed to determine if the point I lie within the boundary of the triangle ΔABC. The barycentric coordinates are calculated for this point. The first coordinate t_1 can be calculated by the equation:

$$a = 1 - \frac{\vec{v} \cdot \vec{AI}}{\vec{v} \cdot \vec{AB}}$$

Where the $\vec{AI}$ is the difference vector between the point A and the point I. The $\vec{AB}$ is calculated the same way but the point I is replaced by the point B. The vector $\vec{v}$ is defined as:

$$\vec{v} = \vec{AB} - \frac{\vec{AB} \cdot \vec{CB}}{||\vec{CB}|| \quad ||\vec{CB}||}$$

Where the vector $\vec{CB}$ is calculated like $\vec{AB}$ but the points A and B are replaced by points C and B . As can be seen, the already explained sum called vector projection is used in this equation.

When the variable a is known, the variables c and b are calculated analogically.

Each variable of the (a, b, c) triple is then tested if it has a value between 0 and 1. If this test is true for all values, the point I lies within the triangle's boundary. If the test is false, then at least one variable has the value outside the 0,1 interval, and thus the point I lie outside the triangle's boundary.

To determine if the point is in the four-segments polygon (A, B, C, D). The test is done for each opposite triangle. In this case, it could be tested with the triangle (A, B, C) and the triangle (C, D, A).

3.2.5 Ray-tracing

This part is describing method called ray-tracing that is widely used in computer graphics algorithms. Because of the attitude to ray-tracing differ in many ways in algorithms. It is presented here in a way that is closest to the solution presented in this thesis.

The idea behind ray tracing use in this thesis is not complicated. It is a method to find or display objects that are in a line of sight from a point in space. To determine if the objects intersect the line, the ray is cast from the point to the position of that object. Two scenarios can come from this process. One of them is that object is not intersecting, and thus there can be some other obstacle in front, or the object is not in that direction. The second scenario is that object can be visible and thus intersect on the ray.

This usage was widely present in older first-person shooting games to determine if the shot bullet hit the target.

3.2.6 Cylindrical projection

For the second solution algorithm, it is needed to introduce a cylindrical projection. By definition, the cylindrical projection is a map projection in which the surface of a globe is depicted as if projected onto a cylinder [23]. Is it is often used to view the world map, as shown on Figure 3.8.

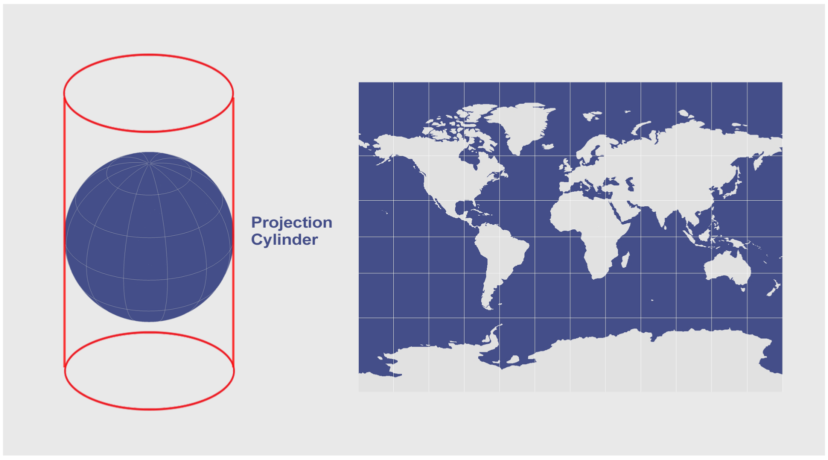

Figure 3.8: Cylindrical projection of the world map.[19]

The projection is made by calculating the angle within 0 and 180 degrees between the projected object's height. The y coordinate is then set to that angle. The x coordinate is equal to the angle within 0 and 360 degrees between the projected object's width. The resulted projection has a rectangular shape in the ratio 2:1.

3.2.7 Cohen–Sutherland algorithm

Cohen–Sutherland algorithm is a line clipping algorithm. Clipping, which is a fundamental operation to several aspects of computer graphics, includes two elements: the clipping window, which could be a rectangle, circle, convex window, concave window, or open window, And the object to be clipped, which could be a line, polygon, circle, character, or irregular curves [22]. The Cohen-Sutherland algorithm is clipping against a rectangular window, which is the most used shape to clip against.

It is almost essential for processes like rendering visible parts of a scene with a specific field of view.

The algorithm uses nine regions set up about the boundaries, as shown in 3.9. Every area has assigned a four-digit binary code, called a region code. This code identifies the location of every line endpoint relative to the boundaries of the clipping rectangle. Each bit position in the region code is used to indicate one of the four relative coordinate areas of the endpoint concerning the clip window: to the left, right, top, or bottom [22].

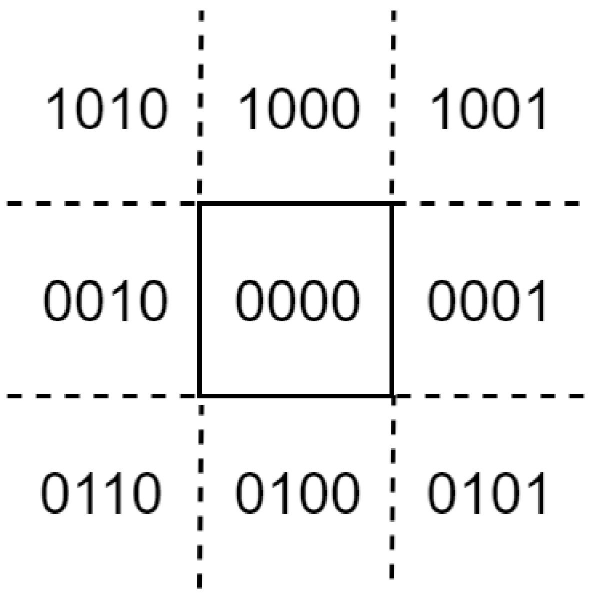

Figure 3.9: Set up regions in Cohen-Sutherland algorithm.

After the assignation of region codes for both lines endpoints, the determination which lines are entirely inside the clip window and witch are clearly outside can be done quickly. If both endpoints of a line have region code 0000, the line is inside the window. If the endpoints have at least one true bit in the result of bitwise AND operator between region codes, the line does not cross the clipping window and thus is rejected.

Lines that cannot be identified as entirely inside or entirely outside a clip window by these tests are checked for intersection with the window boundaries [22]. As can be seen in Fig 3.10, these lines may or may not cross into the window interior.

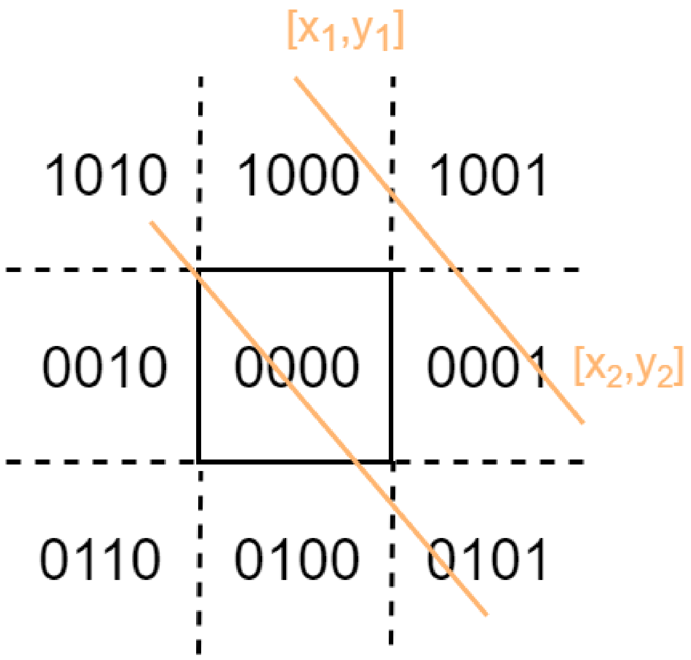

Figure 3.10: Example of possible lines tested using Cohen-Shutherland algorithm.

For finding an intersection within a clipping boundary, the slopt-intercept form of the line equation is used. For example, the Y_1 coordinate between the line with endpoints (x_1, y_1), (x_2, y_2), and a vertical boundary can be calculated using $y = y_1 + m(x - x_1)$. Where the X1 value is either Xw_{min} or Xw_{max}. The m variable is calculated using $m = (y_2 - y_1)/(x_2 - x_1)$. After we get the (X_1, Y_1) coordinate of the first endpoint, the second coordinates (X_2, Y_2) can be calculated analogically.

Next, the processes presented at the start of this section are used to test line with (X_1, Y_1) and (X_2, Y_2) endpoints. That gives us the final result if the line $(x_1, y_1), (x_2, y_2)$ is in the window boundary.

3.3 Research of already existing algorithms

Because the problem presented in this thesis occurs in many games, some solutions already exist. This chapter will explain solutions presents in games Rainbow Six Siege, VBS3 and also in game engines like Unreal Engine 4 and Unity.

3.3.1 Tom Clancy's Rainbow Six Siege

Tom Clancy's Rainbow Six Siege, referred to as RSS, is a game published in 2015. It is developed by Ubisoft Montreal studio. Since it is the first-person shooter, the explosions has a significant role in the gameplay [21]. In this section, all ideas presented are from this game.

Figure 3.11: Image from the Tom Clancy's Rainbow Six Siege.[20]

In RSS, the damage and destruction that happens once an explosion goes off are calculated in a matter of milliseconds. Players will receive visual and audio cues and take damage if they are caught in the blast [21]. To make all of this possible, RSS engine must go through several steps.

When the blast occurs, the game engine uses the origin of detonation and the explosion shape to return a list of entities (operators and objects) present in the blast radius. The shape and the radius is defined by multiple data points defined for each type of the explosion. The final list is then full of potential entities to be affected by the explosion [21].

With the list of potential entities from within the blast AOE, RSS physics engine executes ray-casts [2] [21]. These ray-casts will then travel from the epicenter towards any entity from the list. Each entity has many individual query points that are used to determine whether or not damage is taken [21]. These points are special points used for this use-case. They help to decide better if the entity should be vulnerable to blast damage while still optimizing performance and minimalizing latency [21]. Ray-casts travel to all these points for all entities on the list. Data returned from this process help determine the damage output.

[2]Described in Ray-tracing section.

All objects in Siege contain metadata that define what the object is and how it interacts with other elements in the game. Particular objects contain metadata that block the explosion damage. (Castle walls, shields, etc.). If the ray-cast path collides with an object that has the metadata to block explosive damage before reaching the query points on the entity, the entity is safe [21]. If the ray-cast hits nothing, or objects without metadata to block damage, the entity takes damage. Example of this process is shown on picture 3.12.

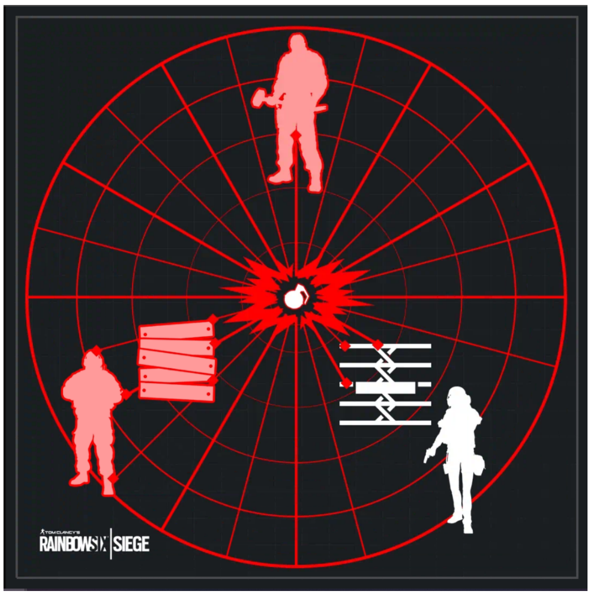

Figure 3.12: Example of ray casts path from epicenter. [21]

The damage output to entity is then calculated based on the distance from the epicenter of blast together with the data from ray casts. Explosions commonly have two effects - destruction and damage. Damage deals damage to players and destruction is what causes any environmental destruction . The shapes and ranges can be different depending on explosion. These effects are limited based on the data returned from ray-casts. However, this is not ideal for some cases. For example if the blast happend inside a building. The walls can cap the damage radius and there is no simulation of reflection wave.

With version Y5S1, the shrapnel concept was introduced into explosions in RSS. That means that destructible objects no longer limits the damage to player. So the blasts will be more realistic as the damage from explosion will be reduced by the number of destructible objects between the entity and the epicentre. Explosions will also be more transparent as shrapnel holes will provide much more visual feedback on the direction from which the explosion damage is oriented.

3.3.2 Discussion of RSS algorithm

The core of the solution used by RSS is present in many FPS games. If it is used with some computer graphics structure like octrees, it is a relatively fast and precise solution. But it can be slow with massive explosions. The main problem is that every object is used more than once when iterating over all ray-casts. And the precision depends on the number of points that are used as the destination of ray-casts. In VBS4, the problem would be that it expects the ray-casts to intersects with the ground. And as was written, the VBS4 uses the whole earth as a map. To get the information about the intersection with it could take a higher time, even with the implementation of computer graphics algorithms.

3.3.3 VBS3

As VBS4 is a game based on VBS3, the variation of the algorithm that solves explosion damage calculations is implemented from there. In the next section the algorithm is described.

VBS3 uses ray casting implementation alike RSS. But it also adds few optimizations that help the time complexity. When are all objects in radius loaded from a map, the iteration through them start. While the iteration runs, every center of the object is checked if it is in a direct hit. While the checking is in action, every other object visited is added to the sorted array by distance. The next iteration, the first controlled objects are taken from the sorted array. That improves the time complexity because the selection of objects is made from more to less important.

For a damage calculation, the methods described in the first part of State-of-the-art are used. One difference is that it use the non-enhanced version of Kingery-Bulmash equation.

3.3.4 VBS3 algorithm discussion

Part of the VB3 solution was discussed in the RSS algorithm discussion. The reason why the implementation of this algorithm to VBS4 is the same as described there. Even with the storing already iterated objects by distance. What is good about the VBS3 process is that it optimizes the algorithm and still is saves precision. It is also fast in cases when the explosion takes place near the massive wall or house because it blocks a large portion of the hit area. An example of this case is shown in figure 3.13.

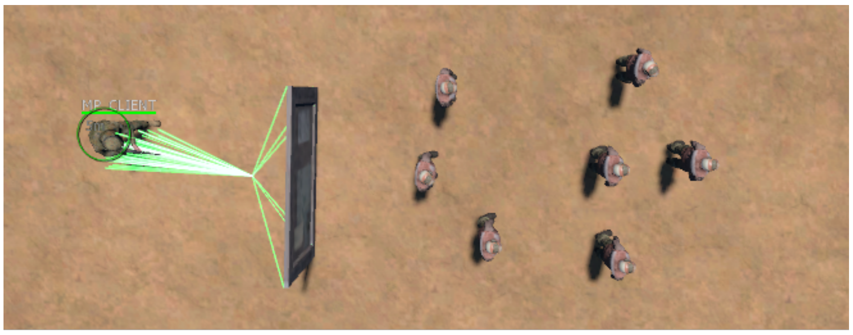

Figure 3.13: Example of explosion near near wall.

3.3.5 Unity and Unreal Engine 4

Because many games are created using game engines, it can be assumed that the mentioned problem is solved in them. For this reason, the two best-known game engines were tested - Unity and Unreal Engine. There is no discussion about presented solutions because they are similar to RSS solution that was discussed before.

The resolution from testing[3] both of the game engines was surprising. By default, neither one of them takes care of what happens when some target is behind a bulletproof object. They basically select all objects in radius and does damage to them appropriately or creates only visual effect and ends. An example of only visual effect is shown in Figure 3.14 demonstrated in Unity. The Figure 3.15 shows the explosion in Unreal Engine 4. As can be seen, the power of explosion affect also the blocks behind the wall.

[3]The testing was done using the basic possibilities that game engines provide. No expanding plugins were used.

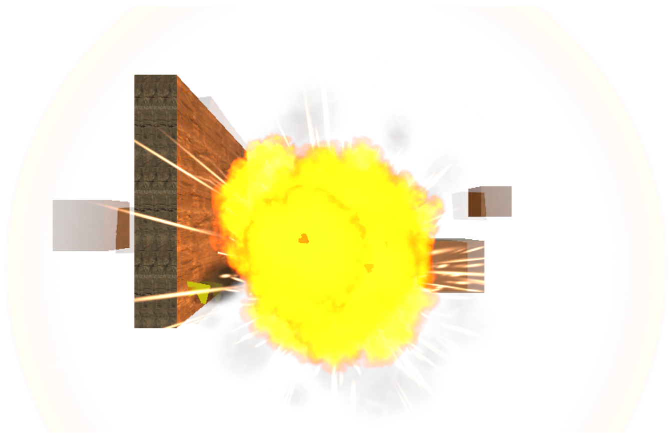

Figure 3.14: Example of unity explosion.

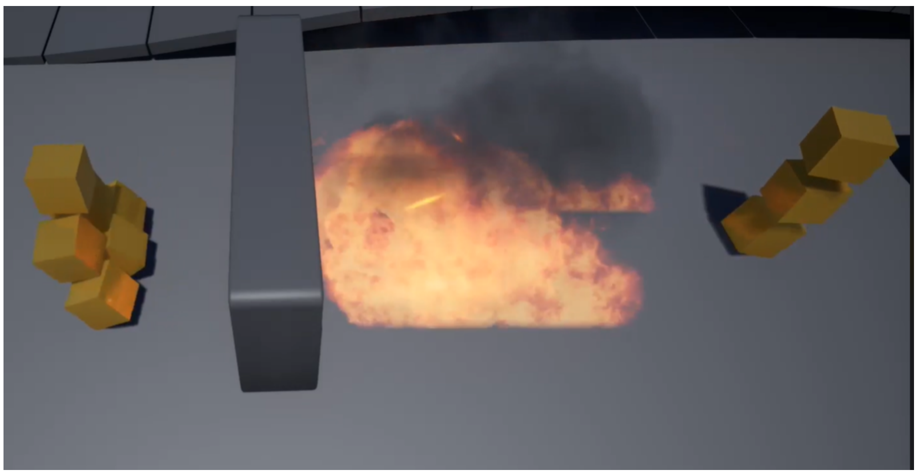

Figure 3.15: Example of Unreal Engine 4 explosion.

When game developers want to implement occlusion, they need to create their algorithms. As far as unity implementation goes, the most solutions discussed on the official unity forum was similar to the method used in RSS - cast rays from the epicenter to every hitpoint of every object and checked for a direct hit. The same is offered as a solution for projects in Unreal Engine 4.

Chapter 4

Realisation

This chapter explains the process of developing the explosion calculator plugin for VBS4. The first part is about a class view and design of the plugin. The second part describes the API interface and the rest sections explains the damage calculator and the two algorithms and their application as the result of this bachelor's thesis. The damage calculator will use the knowledge described in the first part of the Analysis and design chapter. It will do the exact thing as the name says - calculate the damage done to the object based on the defined parameters.

The result of the two algorithms is the value of how much the object gets damaged, which is used as one of the arguments to the explosion damage calculator. Both of the algorithms have a similar core, and both are useful for different scenarios. The first algorithm presented, called Octree algorithm. is more inspired by Tom Clancy's Rainbow Six Siege algorithm explained in the previous chapters. The second algorithm, called Quadtree algorithm, is more innovative but requires more steps to be done to calculate the result. Both algorithms will also have a discussion part where the advantages and disadvantages take place.

4.1 Class diagram of the plugin

This section shows the class diagram 4.1 of the created plugin. It mainly describes the relations between the classes and structs. For the algorithm that calculates the occlusion is selected the Quadtree algorithm that will be introduced later.

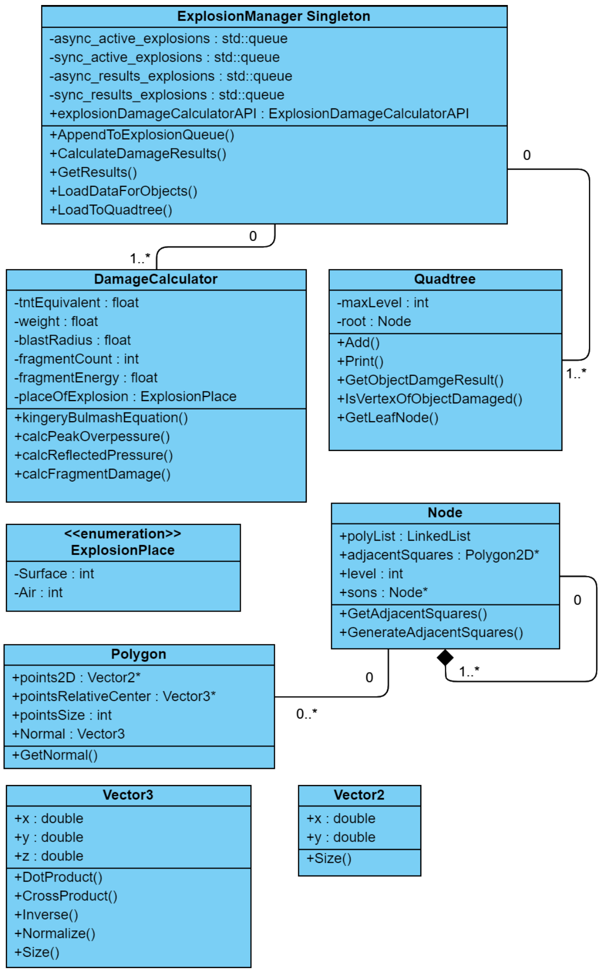

Figure 4.1: Class diagram of the plugin.

4.2 API Implementation

This part presents the API created for the Explosion Damage Calculator Plugin. First, the functions for communication are presented using sequence diagram in figure 4.2. Then, the description of API functions is presented.

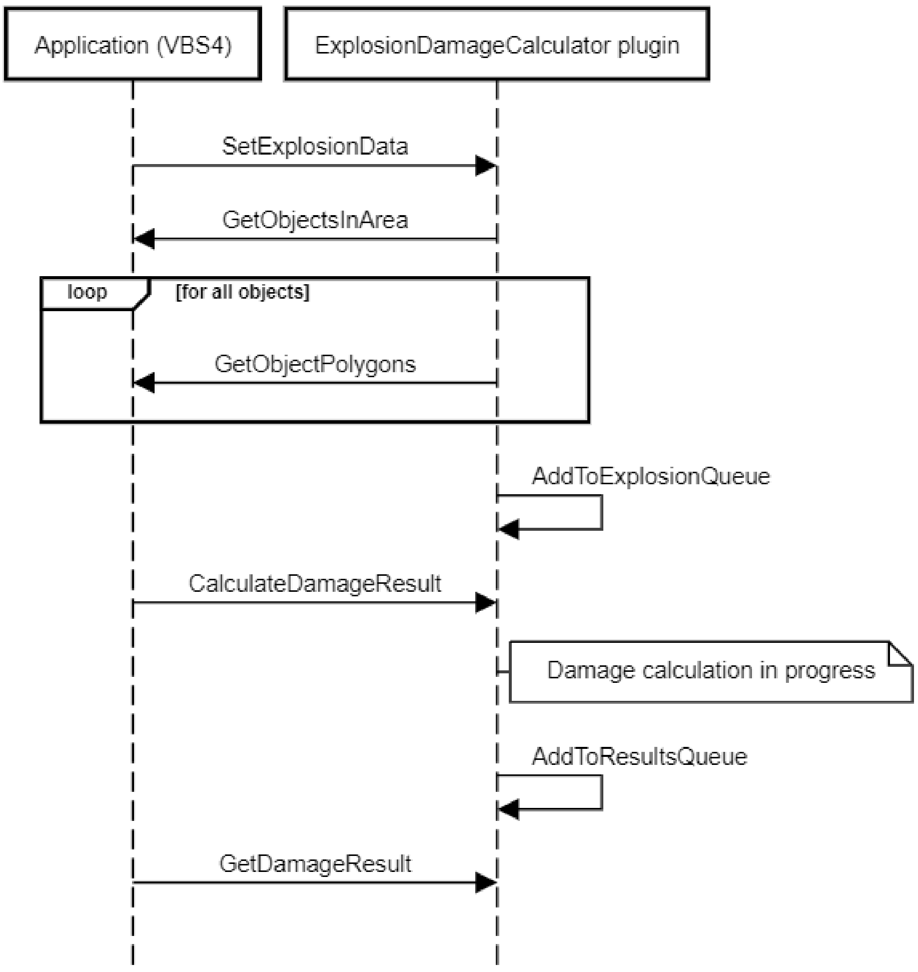

Figure 4.2: Communication between application and plugin.

There are also two debug functions that could, for example, create visible lines between explosion and objects vertexes to control if the hit points are correctly aimed. The second debug function can draw the collision boxes that are used to check bullet or shrapnel collisions. The implementation of these functions is done on the application side. In the following text, the detailed description of the for listed functions takes place. They are also described in the same order as are called in the runtime. One exception is function to calculate damage results async.

At first, when the application (VBS4) wants to calculate explosion damage, the function to set explosion parameters needs to be called. The parameters that are passed to it are the epicenter, explosion weight, blast radius, TNT equivalent, number of fragments, the initial energy of fragments, and the explosion position. These values are passed using the structure called Explosion Info. In the scope of the function, the Explosion Manager singleton class is called, and all named data are passed to it. Behind that, it also loads the necessary data like polygons or objects in area.

The second function, called CalculateDamageResult, takes no parameters. It is because it reads One explosion data from the explosion manager, that were posted to it by service that sets explosion parameters. This, by the way, implies that the task to set explosion data needs to be called every time before calculation of the explosion takes place.

In the scope of this API function is a call to the explosion manager singleton class that calculate damage results. The process of calculating is the most time consuming, and thus has the most significant impact on gameplay from the whole API. It does, what its says, calculate everything that's needed when an explosion occurs.

The third function GetObjectsInArea is called from the plugin to the application. That is done after it is check in the explosion manager that the calculation should be synchronous. This function returns all objects that are in the explosion radius. The returned value is an array of DataOfOneObject structures together with the size of that array. Because the Gears Studio uses API based on the C programming language, a pointer passes these values.

The four function, called GetObjectsPolygons, is more complicated. The values passed to it are the C array and size of all objects previously loaded from the map. It requires the particular format of data to be loaded to buffer from the application. The format could be harder to understand and thus is better described in the Figure 4.3 below:

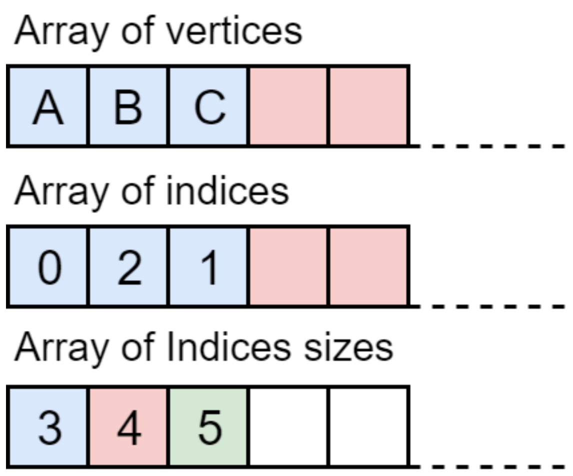

Figure 4.3: Format of loading object data

As can be seen, the format data contains three arrays, each of the different lengths.

The array of vertices is filled with the world position of the vertices of the object. If that array were taken and rendered, the displayed object would be transparent and visible will only be the dots that represent the vertices. That is a problem, and thus it the function needs to load indexes called indices. These are saved to the indices array and are used to determine what vertices from the vertex array make a polygon. To decide how many vertexes have one polygon, the array full of polygon sizes is created.

One additional crucial variable is the size of the polygon sizes array. It is used to loop through the polygon sizes array and create individual polygons.

As was said, the format can be non-intuitive, and thus, the Figure 4.4 shows the process of creating polygons from introduced arrays.

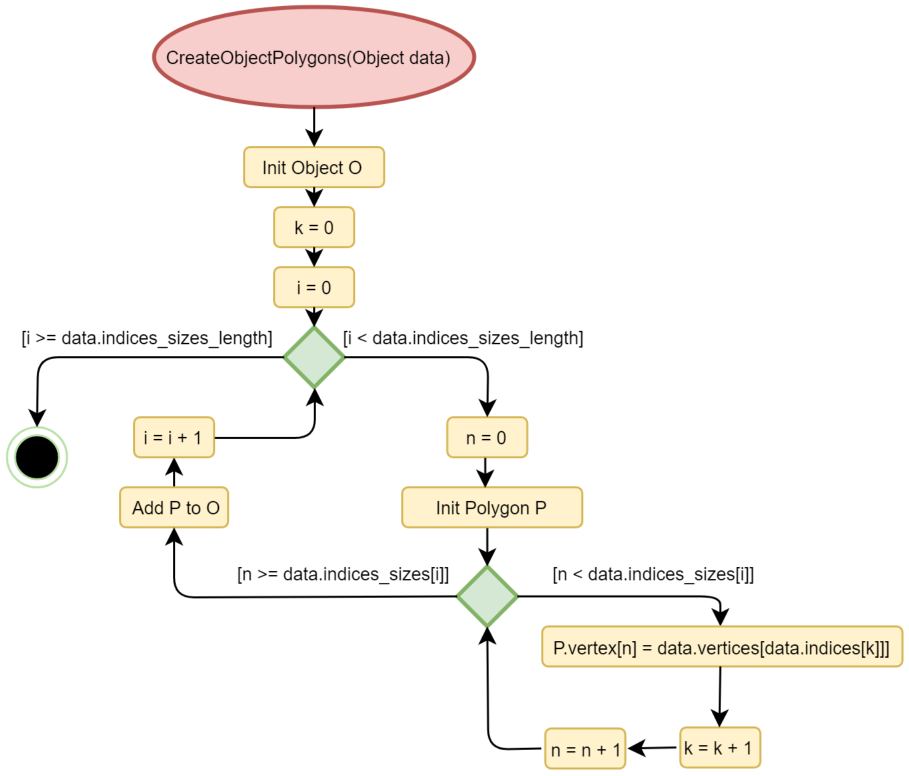
CreateObjectPolygons(Object data)
Init Object O
k = 0
i = 0
[i >= data.indices_sizes_length]
[i < data.indices_sizes_length]
i = i + 1
n = 0
Add P to O
Init Polygon P
[n >= data.indices_sizes[i]]
[n < data.indices_sizes[i]]
P.vertex[n] = data.vertices[data.indices[k]]
n = n + 1
k = k + 1

Figure 4.4: Process of creating polygons from set of arrays.

The arrays presented needs follow a few rules. All ranges need to be non-empty, and the minimal value of one element in indices sizes array has to be more significant or equal to three.

There is also one additional array not shown in figure 4.3. It is a thickness array. It needs to have one value inside for each polygon. That implies that its size is the same as the size of the polygons sizes array. Every value of this array should be between zero and one. Where the one says that the object is non-permeable and zero says that is permeable and destructive.

Overall, the thickness is used to calculate the effects on the object that is behind some permeable structure—for example, the glass or thin wood.

Last but not least, the function GetResults needs to be introduced. This function is used from the application, and the primary purpose is to load all pieces of information about damage dealt to objects. It takes array of damage results and size of that array as arguments. These arguments are filled with the results calculated in the function that calculates the damage. The application of these results depends on the application side. It could then be loaded to a physics engine, for example.

Because the calculating process of the plugin can be time-consuming for massive explosions, the asynchronous implementation is added to guarantee flawless gameplay. Asynchronous workflow has a similar scheme as sync workflow. There are two main differences - the function to set explosion parameters is not called, the task to calculate damage result is asynchronous, and the results are saved to the thread-safe structure.

The set center position is not called because from the developer's perspective to calls two functions where one is asynchronous could be unpleasant.

The advantages of asynchronous implementation are that it could be called every time during the frame from the parent thread, and the game does not stutter. The disadvantage is that the developer who uses it needs to be careful. If the get results function is not called from the parent thread after the calculation ends, then the zombie threads could occur.

4.3 Geometry namespace

In namespace called Geometry are declared crucial parts of this plugin used many times in algorithms described in the next section. The namespace consists mainly of the math functions and structures.

The most basic structures used are Vector2 and Vector3 that present primary math vectors. The central part they contain are variables that define the vector positions. It is x,y for Vector2 and x,y,z for Vector3. They also include some math operations that can be done with them. They are all shown in the table 4.1

Operation name	Returns
Size	Magnitude of the vector
DotProduct	Angle between two vectors
Normalize	Vector that has a magnitude size 1
CrossProduct	Vector that is perpendicular two vectors
Inverse	Vector that is inverted

Table 4.1: Table of operations with vectors implemented in plugin

It's good to say that only the operation Size is implemented for the Vector2 because it is unnecessary to have other functions implemented to Vector2 — they are not used elsewhere in the code.

Vector2 and Vector3 are used in many parts of the code. Overall are used for almost every math operation in the plugin. But for example, they are used to represent polygons and to calculate normals.

Another math-like operation is the implementation of the enhanced Cohen-Sutherland algorithm. In the plugin, it is called PolygonSquareIntersection. It returns true if the square crosses the line segments or is inside the polygon. The scope of the function goes thought the same process as is described in the theoretical part with only one difference. At the start, it defines the variable called isInside and sets it to number fifteen. That is because the number fifteen is represented as 1111 in binary. On the isInside variable is then performed a binary "And" operation with every code that is assigned to every line start point. The outcome of isInside variable is then checked on the end of the PolygonSquareIntersection's scope. If its value is more than zero, the square is not entirely inside the polygon. If its zero, the polygon is entirely inside. An example of how it works is presented using polygon P in figure 4.5.

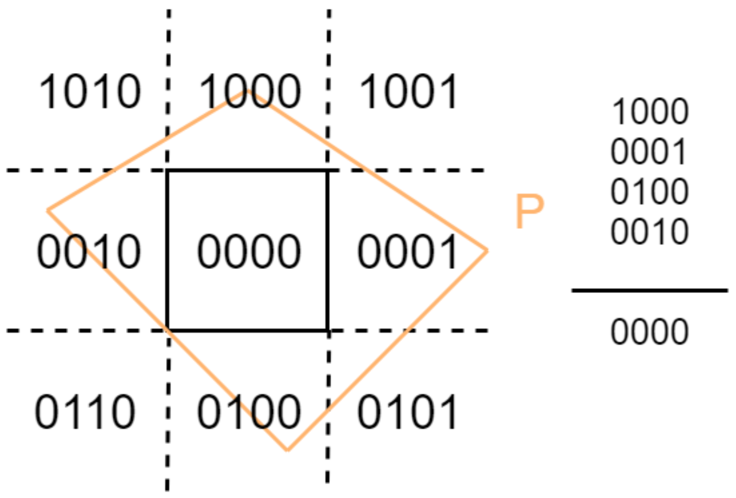

Figure 4.5: Enhanced Cohen-Sutherland algorithm example.

Last thing to say about PolygonSquareIntersection operation is that is used only in Quadtree algorithm.

Lots of math equations are also used in the function called GetDistance, which is crucial for both algorithms described next. It calculates the distance from the epicenter to the point that crosses ray and polygon. Thus the two arguments are needed to be passed to the function—the ray from the epicenter and a requested polygon. In the scope, it uses two algorithms/math equations described in the theoretical part.

The first is the ray-plane intersection. For the calculation of point I, it uses the last equation described in that chapter just with a small difference. There is no need to add up the point X_0 because the epicenter is always centered on the coordinates (0, 0, 0).

The second is the point in polygon algorithm. It has an implementation of this algorithm for two situations that are selected based on the number of polygon points. If the polygon is composed of three nodes, it goes by the standard way defined in the named chapter. When it has four nodes, it is separated into two triangles, and the operation is done for both.

As far as structures go, one basic structure is called a sorted linked list. It is the structure that can sort polygons by their distance from the center. It has four main variables called first, last, prev, and next. These are used to determine where the polygon should be added. The structure has three methods. They are called Insert, CheckFirst and IsEmpty.

Insert function does what the name says. It inserts the required element to the linked list based on its attributes. In the case of this plugin, the item is a polygon handle. On the start, it checks if the first attribute from the linked list is filled. If no, it is set to be the inserted element and the function ends. Otherwise, it continues to select the right place for that element. That means it will find the site where the epicenter's distance is smaller than the next element and place it in that position. If the element has a gap more significant then the other items in the list, it is placed to the variable called last.

The functions called CheckFirst and IsEmpty has an effortless course. They are both used in the condition statements. The check first function checks, if the first element is empty. If yes, the value passed with the function is assigned to the first element, and the CheckFirst returns true. In another case, the function ends right away and returns false. The IsEmpty checks if the variable first is empty. If yes, its return true. Otherwise, it returns false.

4.4 Octree Algorithm

As was said, this algorithm is inspired by Tom's Clancy's Rainbow Six Siege. It is inspired in the way that it uses raycasting to determine if the object we want to check is in some direction. To better explain how the whole algorithm works, the next list describes the steps for getting damage deal to all objects in the scene. The steps will then be explained further.

1. Select all objects that are in blast radius.
2. Select which polygons of those objects will be used.
3. Add those polygons to octree.
4. Iterate through selected polygons and, using octree, check, if it is visible from the point of explosion.
5. If the polygon is at least partially visible, calculate the damage dealt.

Step one and two are the same for both algorithms presented here. The first step is vital because it helps the performance to have selected objects that can be damaged. There can be an edge case that only one object is selected. If so, the algorithm jumps over the third and four steps because it is unnecessary. The selection speed highly depends on the program that uses the plugin presented here. The best thing would be it uses some faster structures like R-Tree or Octree. The VBS4, for example, uses R-Tree.

When are all possibly affected objects selected, the algorithm then iterates through every object polygons. While doing this, it will select all polygons that are facing in the direction of the explosion. To decide if some polygon (A, B, C) meets this requirement, it calculates the normal vector of this polygon and the vector difference between the epicenter and point A. If the angle of these two vectors is less than 90 degrees, it is selected to process next. This process is called polygon culling and is essential for rendering. In this case, it is done because of the performance, and because it is useless to have back faces.

The third step is one of the most computationally hard to do. Every selected polygon needs to be added to octree for faster searching in the next level. But it is unnecessary to add a polygon to octree on the maximal depth every time. The process of adding an algorithm to octree is revealed in the Flow chart 4.6:

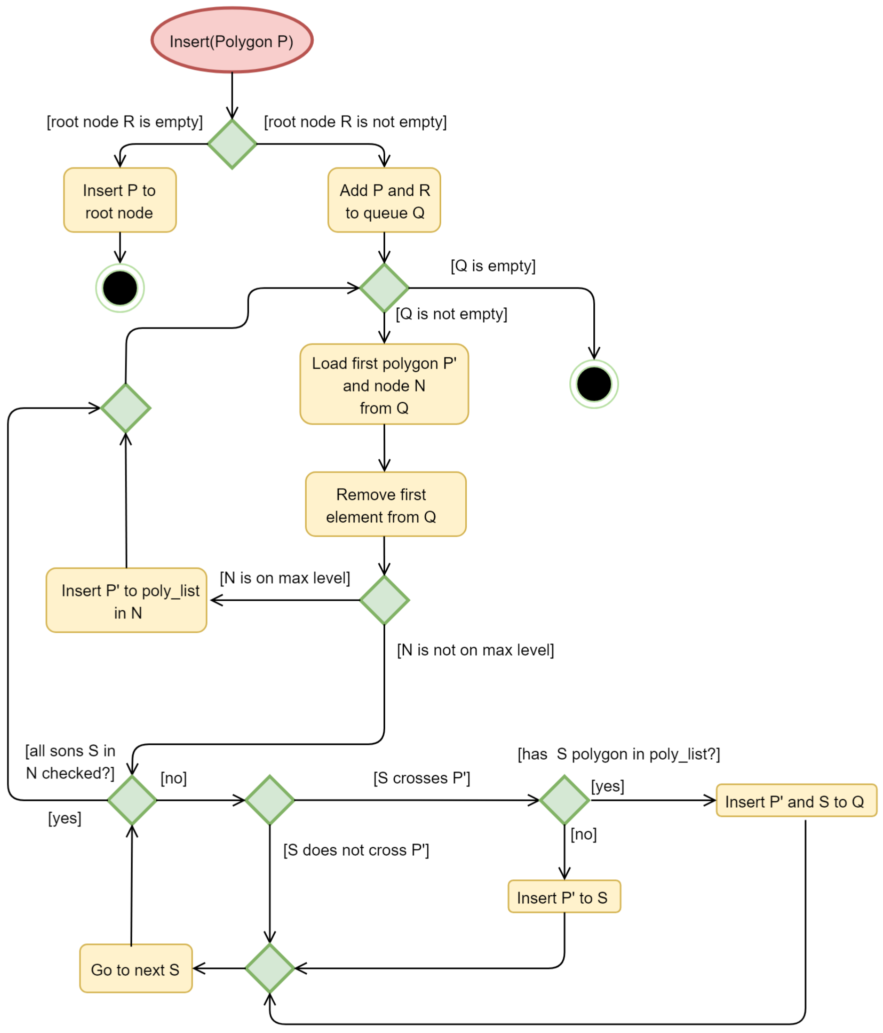

Figure 4.6: Flow chart that describes the process of inserting polygon.

Step four is to decide which object is wholly or partially hit. That is done by iterating through vertexes of selected polygons and testing them if they are in direct sight or behind some bulletproof objects. For this purpose is needed to get polygons from octree nodes that intersect with the ray R defined by two points - the epicenter of the explosion and the vertex. In order to achieve this, the standard recursive or iterative function can be used.

After the algorithm obtain that polygons from octree. The iteration through them starts. In that phase is every polygon tested with r using the ray-plane and ray-triangle algorithm. If it is confirmed that polygon lies on r, the magnitude m of the vector difference between the epicenter and ray-plane intersection point is calculated. The magnitude of m is then compared with the s, which is the magnitude of r. If m has a smaller value than s, the tested vertex is behind the polygon and is needed to take further actions. Such as calculate the percentage from full damage based on the toughness of polygon material. These results are then saved to the array of pairs. The first value of the pair is the ray R, and the second value is the damage percentage. These are used for determining what should happen with the object that contains tested vertex if it should be moved or destroyed. The whole process is outlined in the Figure 4.7.

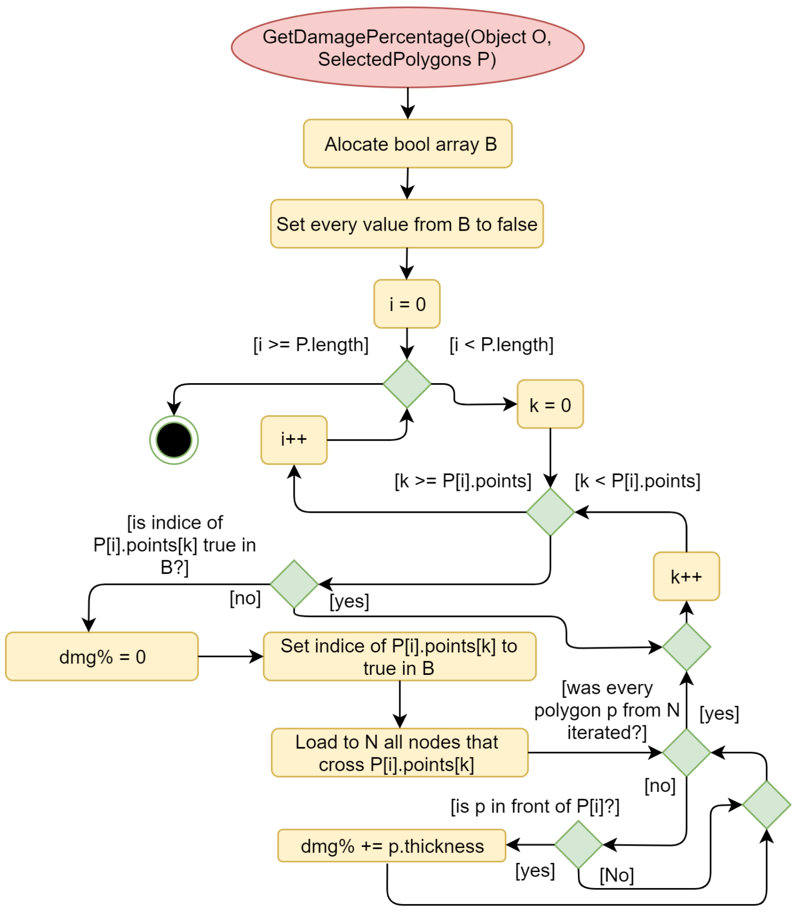
GetDamagePercentage(Object O, SelectedPolygons P)
Alocate bool array B
Set every value from B to false
i = 0
[i >= P.length]
[i < P.length]
k = 0
i++
[k >= P[i].points]
[k < P[i].points]
[is indice of P[i].points[k] true in B?]
[no]
[yes]
k++
dmg% = 0
Set indice of P[i].points[k] to true in B
[was every polygon p from N iterated?]
[yes]
Load to N all nodes that cross P[i].points[k]
[no]
[is p in front of P[i]?]
dmg% += p.thickness
[yes]
[No]

Figure 4.7: Algorithm that counts the damage dealt percentage to object.

As can be seen on the start, the algorithm saves the vertices that were already calculated. That is because of the vertices can be repetitive in polygons. And to keep what vertices were used is faster than to calculate the damage again. Also, it needs to be mentioned that the thickness represents the value of how much the object is bulletproof. Thus it is between 0 and 1.

4.4.1 Discussion about the Octree algorithm

The presented algorithm is relatively fast if it is done with some preparations. One of the preparation is to have a scene already inserted to octree. In that case, the third step can be completely removed, and the algorithm is then just checking the intersection only with the different epicenter. The advantage of this algorithm against the quadtree algorithm presented next is that it has the third dimension. This means it is possible to simulate other effects of an explosion as reflected shockwave. But the implementation could cause a large overhead to the time complexity. Also, one advantage was already written, and it is a possibility to create octree before the explosion occurs.

4.5 Quadtree Algorithm

The Quadtree algorithm is the innovative part of this thesis. The idea behind this is that it is almost possible to remove the third space from algorithms to calculate explosions. That means that it is achievable to use only the vertical angle and horizontal engle to determine where the object is in perspective to the epicenter. The aspects can then be used to display an object in a two-dimensional space. This project will be called a 2D projection in the further text. An example could be seen in the Figure 4.8.

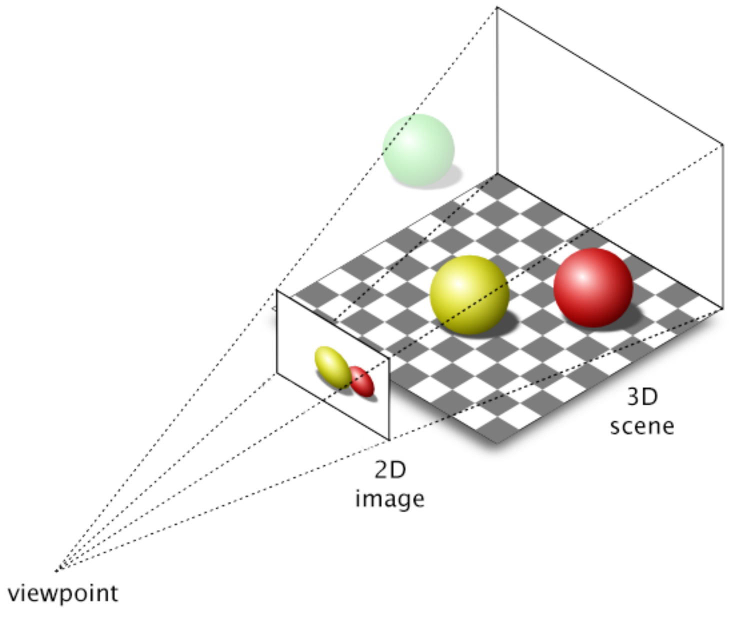

Figure 4.8: Example of the 3D to 2D projection.[25]

As in the Octree algorithm, this algorithm goes through some stages that are sometimes alike. All the stages are listed below:

1. Select all objects that are in blast radius
2. Select which polygons of those objects will be used.
3. Calculate the projection of those polygons.
4. Add projected polygons to the quadtree.
5. Iterate thought the selected polygons.
6. If the polygon is at least partially visible, calculate the damage dealt.

The first two steps are described in the Octree algorithm section. The third step needs more explanation that is presented next.

The stand-alone calculation of the one polygons projection is straightforward. The result of one polygon point projection of the vertex (x, y, z) is two values (α, β). These values are calculated using basics trigonometric functions.

The β point has a value between 0 and 180 and equals to the angle between the up vector and the vector between the point and epicenter. The angle is calculated using the dot product that is then transferred from radians to degrees.

The α point equals to the angle ratio between the x value and z value. To do this, the function atan2 is used because it is needed to have an interval of x1 from zero to three sixty degrees. Atan2 is a function that is implemented in most programming languages. It works as an arctangent function but moreover takes care of some edge cases. For example, the divisor in arctg role might be zero. That could result in unhandled behavior. But, the atan2 will return a $\pi/2$ or $3/2\pi$ based on the dividend value.

The example with picture of this transformation can be seen on Figure 4.9.

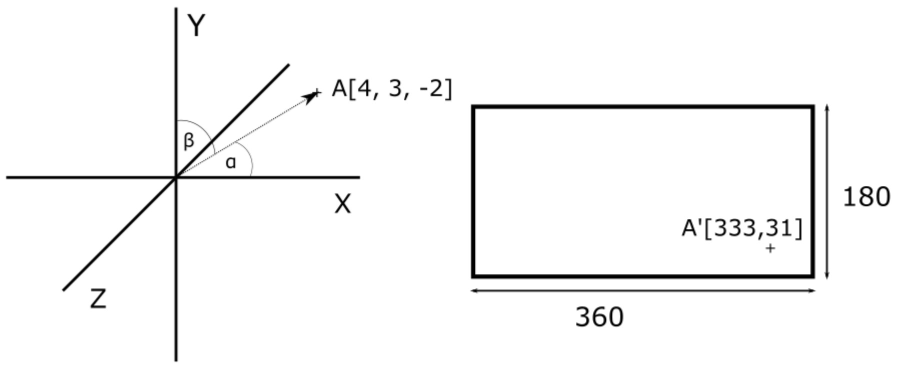

Figure 4.9: Example of the transformation.[25]

The example 4.9 shows point A, defined by coordinates (4, 3, -2). The following math operations describe how to get its 2D projection A' represented by (333, 31).

$$\alpha = atan2(-2,4),\ \beta = \frac{(0,\vec{1},0)\cdot(4,\vec{3},-2)}{||(0,\vec{1},0)||\ ||(4,3,-2)||}$$

The α equals 333 and the β equals 31 after the results are rounded.

Against Octree implementation, there is needed to do one other process. It is the premise that cylindrical projection works perfectly for all polygons that do not cross the up vector. It is because the nature of this method is directed to control all objects around the explosion but not directly inside the explosion. The solution to this problem is that all these polygons need to be tested whenever the algorithm calculates some polygons' direct hit. The testing is achieved by storing these polygons to some structure, for example, vector and iterate over them every time in the function GetDamageResults.

Every polygon is then inserted into the quadtree. The process of adding is similar to octree insertion. The only difference is that it is inserted into nodes with square shape. The inserted polygon is checked using the Polygon-SquareIntersection function described in the geometry section. If it returns true, it continues to another node or creates a new node. When it travels to the last node, it is inserted into the polygon list of that node. As in the octree algorithm, the nodes in the polygon list are sorted by the distance of the closest point of the polygon.

The algorithm to check damage results is also almost the same as in the octree Algorithm 4.7. The main difference was already mentioned. It is that is needed to check every polygon that intersects up vector.

The beautiful thing about quadtree implementation is that it can be visualized by rendering the whole projection to one picture. One example is shown in the figure 4.10

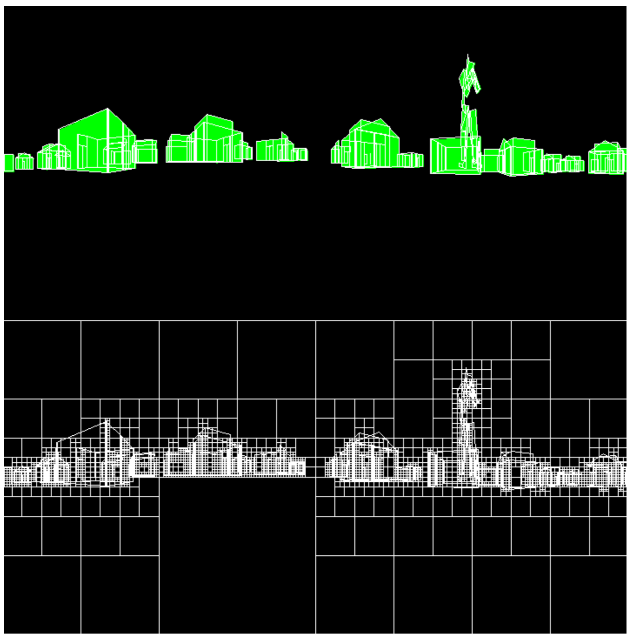

Figure 4.10: Example of the projection and quadtree visualization.

The figure 4.10 is from a mission with many warehouses and one soldier, shown in Figure 4.11. The soldier, surrounded by warehouses, is nicely visible in the left part. The green lines shown on Figure 4.11 are hit points. Also the occlusion is nicely visible.

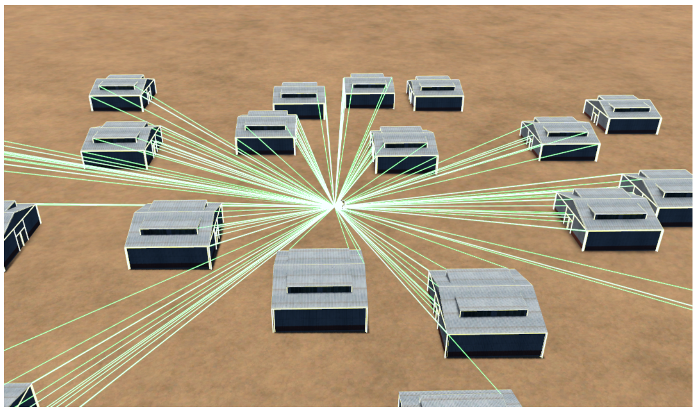

Figure 4.11: Example of explosion around houses.

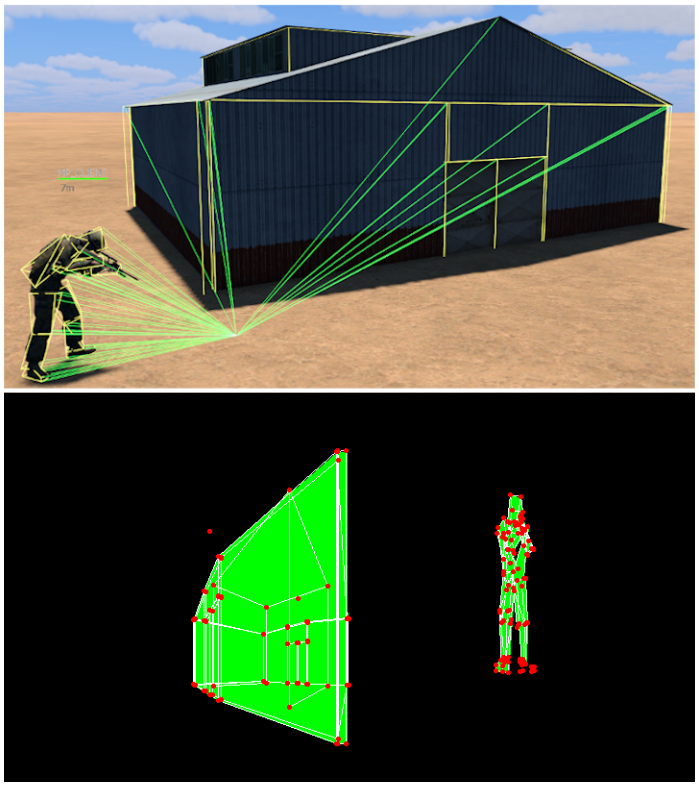

Figure 4.12: Example of the hitpoints damaged by the explosion compared to the game visualization.

4.5.1 Discussion about the quadtree algorithm

The quadtree algorithm presented is an innovative way to calculate occlusion and explosion damage. The advantages are that the precision and time complexity can be adjusted by changing the depth of quadtree. The 2D visualization can be used for many cases, like showing the soldier where has been hit and where he makes mistakes, as shown in Figure 4.12.

It also reduces the time complexity based on the range of the explosion - the larger the explosion gets, the quadtree build time gets shorter against its ratio. That is because the quadtree was already allocated, and no new nodes are created. It is also proof that the algorithm would work even when the calculations were made on GPU.

The disadvantages are that the 2D projection works only for one position, and against octree implementation, it cant be used again with the same quadtree. There is also needed to use some additional data, like storing all projected polygons and polygons that cross the center in a vertical direction. The algorithm also has some limitations as calculating the reflected shockwaves or fend fragments.

4.6 Damage calculator

The damage calculator is a class that takes results from the octree or quadtree algorithm. Mainly the distance from the epicenter to the hitpoint and the percentage of damage reduction. The reduction is based on how many objects prevents the direct hit.

The class alone calculates the power that affects objects using the equations presented at the start of the State-of-the-art chapter. The enhanced Kingery-Bulmash equation is used to calculate blast overpressure and reflected pressure. The fragment energy, together with the fragment count, is used to calculate damage from fragment impact.

The class is allocated in the function that calculates the damage result with settings that differ for every explosion. The calculator also needs to set up parameters that are used for damage estimating as TNT equivalent, fragment energy, fragment count, etc.

4.7 Testing

The performance of the presented quadtree algorithm was tested against the existing VBS4 algorithm. The tests were done on three missions that have ascended complexity. The first mission, called HouseAndSoldier, is placed in the desert, where there are only one house and one soldier. The second mission, called CityInDesert, is also on the desert, where are multiple buildings and one soldier. And the third mission, called Forest, is in the dense forest with one soldier. As can be seen, the number of objects surrounding the epicenter is higher in every next mission. If the exact numbers were taken to account, the maps have 6,28,1104 objects, sequentially. The results are in miliseconds and are shown in Table 4.2

Mission	Number of objects	VBS4 algorithm	Quadtree algorithm
HouseAndSoldier	6	0	12
CityInDesert	28	1	84
Forest	1104	18	1296

Table 4.2: Time complexity of Quadtree algorithm and VBS4 algorithm.

As can be seen on table, the results of testing are unsatisfactory. The time complexity of the Quadtree algorithm is higher than the complexity of the already existing VBS4 algorithm. The reason for that is that the VBS4 algorithm uses only bounding boxes for collisions. That means the algorithm should not be so precise. Let's take a look at figures that compare the algorithms by precision.

The first figure 4.13 shows the soldier that is hiding behind a house. The green lines show the soldier's and house's hitpoints generated by the Quadtree algorithm, and the red line shows the hitpoint calculated by the VBS4 algorithm. As can be seen, the VBS4 algorithm decided that the soldier is wholly hiding and dealt damage based on that. The Quadtree algorithm decided that only part of the soldier is hiding and calculates damage dealt only by the approximation of the area that is shown. The yellow polygons shows collision polygons used to calculate hitpoints in Quadtree algorithm.

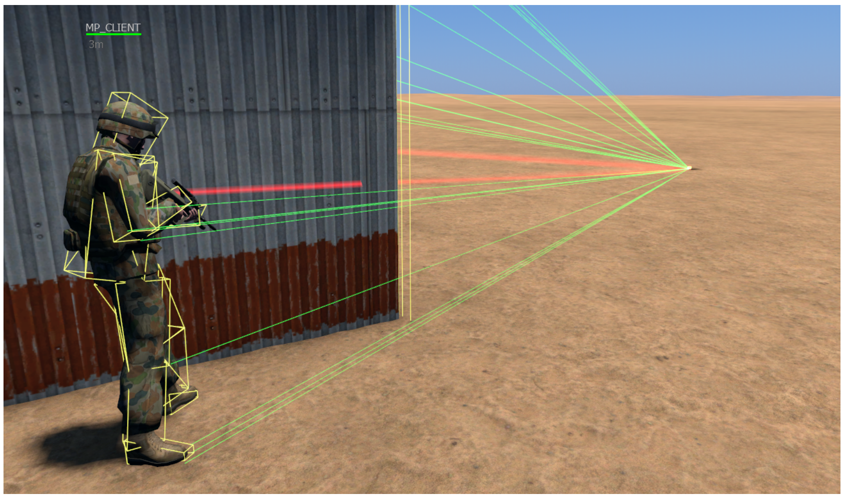

Figure 4.13: Example of the precision on small mission.

The second figure 4.14 demonstrates the damaged CityInDesert map. It is ruined because a high explosive with a large radius was dropped to the center. The hitpoints were already defined in the description for figure 4.13. As can be seen, the occlusion of the Quadtree algorithm is more precise. An example of that is especially on the left building. The VBS4 algorithm recognizes that the building is wholly hit (red line). But the Quadtree algorithm (green lines) shows, that it is hitted only partially.

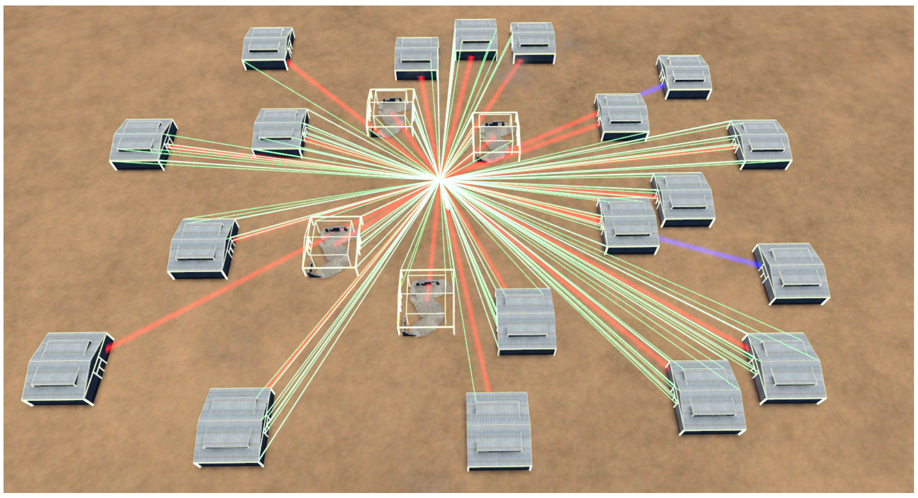

Figure 4.14: CityInDesert map hitpoints asnd occlusion.

The hitpoints calculated by the Quadtree algorithm are visible also on the 2D projection of the map as can be seen on Figure 4.15.

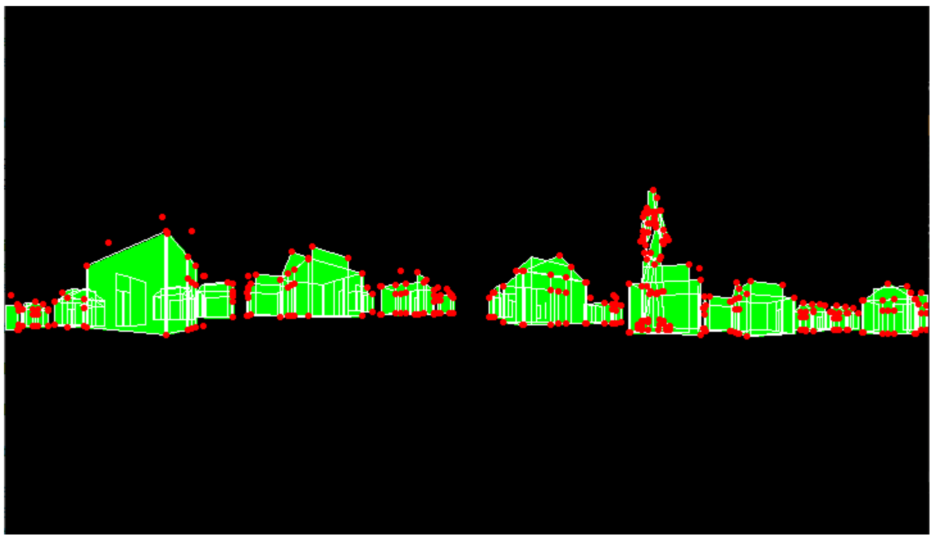

Figure 4.15: Rendered hitpoints on 2D projection.

The third figure 4.16 presenting the explosion in the mission Forest. The soldier is standing behind the tree and thus it will be damaged only to some parts of body. The red line, as was said, shows the VBS4 algorithm. It finds that the soldier is behind the tree and deals reduced damage based on the thickness of the tree.

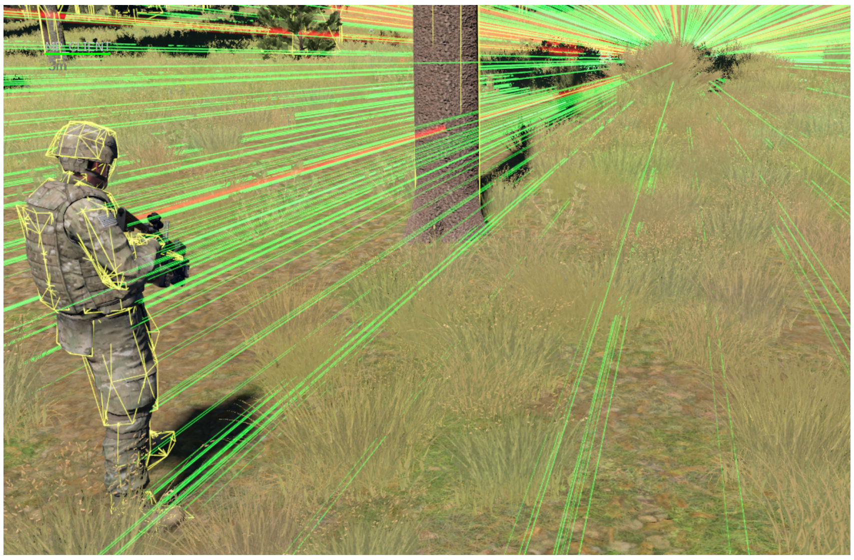

Figure 4.16: Screen of the soldier from the forest with visible hitpoints.

The precision testing proves that the Quadtree algorithm is more precise than the VBS4 algorithm. And thus, it helps to simulate a better real-life experience.

Conclusion

The conclusion of this thesis is interesting. The testing proved that the main algorithm of this thesis, Quadtree algorithm is slower than the algorithm used in VBS4. But it is also more precise to calculate what area was hit by the explosion. Because the VBS4 is the game that aims for real-life simulation, accurate calculations are essential. The algorithm alone is innovative, and all ideas and methods used to create it should compress the time complexity. The lesson from it is that during game development, it is necessary to account that the data preparation for processes is a crucial part of any algorithm and should be fast. The Quadtree algorithm also proves that it is possible to calculate explosions effects using the rendering method. That means that it could run on the GPU for better time complexity. Also, the theoretical part introduces examples of explosion damage calculations in other games together with needed math and explosion equations. That gives an excellent summary of these problems to everyone who will come across similar interests.

Bibliography

[1] Homans, Charles. “War Games: A Short History.” Foreign Policy, Foreign Policy, 31 Aug. 2011, foreignpolicy.com/2011/08/31/war-games-a-short-history/. [online]

[2] “Bohemia Interactive Simulations.” BISim, Bohemia Interactive Simulations, bisimulations.com/products/vbs-blue-ig.[online][03.03.2020]

[3] “Bohemia Interactive Simulations.” BISim, Bohemia Interactive Simulations, bisimulations.com/products/vbs4.[online][03.03.2020]

[4] Alexion, A. (2020). Simplify and Standardize Your Software Development Process.[online] Retrieved May 25, 2020, from https://gears.studio/

[5] Baker, E. (2017). Warhead Technology. Retrieved May 28, 2020, from https://www.msiac.nato.int/areas-of-expertise/warhead-technology [online]

[6] MADIGAN, Michael L., 2017, First Responders Handbook: An Introduction, Second Edition. Boca Raton, FL : CRC Press.

[7] Payne, Craig. Principles of Naval Weapons Systems. Vol. 2, Naval Institute Press, 2010.

[8] JEON, Doojin, KIM, Kitae and HAN, Sangeul, 2017, Modified Equation of Shock Wave Parameters. Computation. 2017. Vol. 5, no. 4p. 41. DOI 10.3390/computation5030041.

[9] Šumbera, Michal, Nuclear Explosion[lecture][02.05.2020]

[10] Girard, James. Criminalistics: Forensic Science and Crime. Jones and Bartlett Publishers, 2008.

[11] BAJIC, Zoran, BOGDANOV, Jovica and JEREMIĆ, Radun, [no date], Blast-Effects-Evaluation-Using-TNT-Equivalent. Academia.edu. [Accessed 2 May 2020]. Available from: https://www.academia.edu/8055261/Blast-Effects-Evaluation-Using-TNT-Equivalent [online]

[12] Oxford advanced learners dictionary, 2005. , Oxford : Oxford University Press.

[13] Klimi, George. (1990). Estimation of Casualties from Fragmentation Ammunitions.

[14] Shrestha, R. (2019, November 24). Camera Calibration in OpenCV-python. Retrieved May 24, 2020, from https://medium.com/@rashikshrestha01/camera-calibration-in-opencv-python-dffed958f7e4 [online]

[15] PALMER, Grant, 2005, Physics for game programmers. Berkeley, CA : Apress.

[16] Revelles, Jorge Ureña, Carlos Lastra, M. Lenguajes, Dpt Informaticos, Sistemas Informatica, E.. (2000). An Efficient Parametric Algorithm for Octree Traversal.

[17] Meagher, Donald. (1980). Octree Encoding: A New Technique for the Representation, Manipulation and Display of Arbitrary 3-D Objects by Computer.

[18] BLOOMBERG, Sam S., 2008, Color quantization using octrees. [online]. 4 September 2008. [Accessed 3 May 2020]. Available from: http://leptonica.org/papers/colorquant.pdf

[19] Cylindrical Projection: Mercator, Transverse Mercator and Miller. (2020, March 06). Retrieved May 26, 2020, from https://gisgeography.com/cylindrical-projection/

[20] Mccaffrey, R. (2015, January 24). Rainbow Six Siege: 5v5 or Bust – IGN First. Retrieved May 26, 2020, from https://sea.ign.com/rainbow-six-siege/85961/feature/rainbow-six-siege-5v5-or-bust-ign-first

[21] DEV BLOG: EXPLOSIONS SHRAPNEL IN Y5S1, [no date]. DEV BLOG: EXPLOSIONS SHRAPNEL IN Y5S1 [online], [Accessed 30 April 2020]. Available from: https://www.ubisoft.com/en-us/game/rainbow-six/siege/news-updates/1QkezaGoRkDWqcQ6duGvtk/dev-blog-explosions-shrapnel-in-y5s1 [online]

[22] Iraji, ms Mazandarani, Ayda Motameni, Homayun. (2011). An Efficient Line Clipping Algorithm based on Cohen-Sutherland Line Clipping Algorithm. 65-71.

[23] The American Heritage dictionary of the English language, 2016. , Boston : Houghton Mifflin Harcourt.

[24] Weisstein, Eric W. ”Barycentric Coordinates.” From MathWorld–A Wolfram Web Resource. https://mathworld.wolfram.com/BarycentricCoordinates.html [online]

[25] Shrestha, R. (2019, November 24). Camera Calibration in OpenCV-python. Retrieved May 24, 2020, from https://medium.com/@rashikshrestha01/camera-calibration-in-opencv-python-dffed958f7e4 [online]

Appendix **A**

Acronyms

AOE Area of effect

RSS Tom Clancy's Rainbow Six Siege

TNT Trinitrotoluen

Appendix B

Contents of enclosed CD

```
 README.txt ........................... the file with description of plugin
├── bin ................................. the directory with the plugin dll
├── build ................................ the directory of source codes
├── src .............................. the directory of API source codes
├── thesis ............... the directory of LaTeX source codes of the thesis
│   ├── thesis.pdf ......................... the thesis text in PDF format
│   └── thesis.ps ............................ the thesis text in PS format
└── include ...................... the directory with the plugin header files
```