DEGREE PROJECT, IN SCIENTIFIC COMPUTING , SECOND LEVEL
STOCKHOLM, *SWEDEN 2015*

Collision Detection between Dynamic Rigid Objects and Static Displacemen Mapped Surfaces in Computer Games

FANGKAI YANG

KTH ROYAL INSTITUTE OF TECHNOLOGY

SCI SCHOOL OF ENGINEERING SCIENCES

Collision Detection between Dynamic Rigid Objects and Static Displacement Mapped Surfaces in Computer Games

F A N G K A I Y A N G

Master’s Thesis in Scientific Computing (30 ECTS credits)
Master's Programme in Applied and Computational Mathematics
Royal Institute of Technology year 2015
Supervisor at Avalanche Studios, Stockholm: Joacim Jonsson
Supervisor at KTH: Christopher Peters
Examiner: Michael Hanke

TRITA-MAT-E 2015:50
ISRN-KTH/MAT/E--15/50--SE

Royal Institute of Technology
School of Engineering Sciences

KTH SCI
SE-100 44 Stockholm, Sweden

URL: www.kth.se/sci

Abstract

Collision detection often refers the detection of the intersection of two or more objects. Collision detection algorithms in Avalanche Studios' game engine need not only to detect the collision, but to get the closest distance and handle penetration. Current algorithms perform well in most cases, but obtain poor accuracy or low efficiency in some cases. This paper will attempt to improve the performance in two ways. First, two new backward projection methods are derived and compared, achieving more accurate backwards projected points. The accurate backwards points are important in collision detection with the terrain surface. Second, multiresolution bounding volumes are constructed in the narrow phase collision detection. These bounding volumes improve the performance when performing collision detection between large complex objects and the terrain. These bounding volumes reduces the number of backward projections needed.

Referat

Kollisionsdetektering mellan dynamiska objekt och statiska terräng

Kollisionsdetektering avser oftast att detektera skärning mellan två eller fler objekt. Kollisionsdetekteringsalgoritmerna i Avalanche Studios spelmotor behöver inte bara upptäcka skärningar, utan även det närmsta avståndet och hantera penetration. Aktuella algoritmer presterar bra i de flesta fall, men erhåller i vissa fall låg noggrannhet eller effektivitet. Detta dokument kommer att förbättra prestandan på två sätt. Först härleds och jämförss två nya bakåtprojektionsmetoder, vilka resulterander i mer exakta bakåtprojicerade punkter. Korrekt bakåtprojektion är viktig för kollisionsdetektering mot terrängytan. Sedan konstrueras och nyttjas flerupplösta begränsnings volymer under den detaljerade kollisionsfasen. Dessa avgränsande volymer förbättrar prestanda när kollisionsdetektering sker mellan stora komplexa objekt och terrängen då de minskar antalet bakåt projiceringar som behöver utföras.

Acknowledgements

I would like to express my sincere gratitude to my 师父 (mentor) Joacim Jonsson for his continuous support of my work, for his greatest patience, immense theoretical and practical knowledge. It is my great luck to have him as a mentor. His guidance not just helped me all the time in this thesis work, but ignite my infinite enthusiasm for game programming.

Besides my mentor, I would like to thank Prof. Christopher Peters, for giving me guidance and reviewing my thesis work.

My sincere thanks also goes to Prof. Xiaoming Hu and Prof. Krister Svanberg, for their help in the optimization methods in this work. I will especially thank Jamie Rinder for his help in language and structure.

Moreover, I would like to thank the foundation of Engineer Ernst Johnson, for offering me full scholarship to support my study in KTH.

Last but not the least, I would like to thank my mother: Yunxian Fang, for giving birth to me, raising me and supporting every choice I have made in my life.

Contents

Chapter 1

Introduction

1.1 Background

The thesis is done on PC platform with Intel Core i7-4790K CPU. The game in this thesis is *Just Cause 3* which will be released soon. Just Cause 3 is an action-adventure game. Player can use a large number of vehicles, including cars, airplanes and naval ships. The vehicle varies in a wide size range, from a motorcycle to a bomber. Dealing with large and complex objects is an important work in this thesis. The environment is large and open, and player can explore and choose what he/she want to do in the game. The terrain system is large and complex, containing basins, tunnels, cliffs.

The volumetric landscape system generated by Avalanche Studios' game engine is approximately 400 square miles, with a resolution of 2 meters. A mesh surface is used as a base mesh for displacement mapping. Each vertex in the base mesh contains a position, a projection vector and coordinates to a displacement map. The displacement lengths in the displacement maps are set so that the forwards projected surface matches the original surface defined by the signed distance field as closely as possible. Small but tiled detailed displacement maps with higher resolution [1] based on painted material selections can also be added to the surface projection function. The result is a very low memory footprint detailed representation of the landscape.

Forward projection from a base mesh is a well known problem within computer graphics, and there are a great number of academic results on the subject [2]. However, backwards projecting arbitrary points in space onto the base mesh efficiently is a more complicated problem which seems to have received very little academic work. The accuracy and efficiency of backward projection will highly influence the performance of collision detection.

Collision detection refers to the detection of the intersection of two or more objects. In Avalanche Studios' game engine, it needs not only to detect the collision, but also obtain the closest distance and handle penetration. If the collision is detected, the objects will very likely penetrate each other. The game engine stops

the penetration (such as a character is partially in a wall or ground) in the former frame, which makes the frame performing collision detection for the predicted one. This predicted frame will never happen in the game, but it is computed to avoid penetration.

Real-time collision detection requires both accuracy and efficiency. There is a lot of research on collision detection among convex rigid objects [3]. Furthermore, [4], [5], [6] illustrate the reason why collision detection is done, i.e. get the closest distance for potentially colliding objects and generate motion constraints during rigid body dynamic simulation. However, very little academic work regarding collision detection between a complex rigid body and displacement mapped terrain exists.

1.2 Problem Statement

Two main problems are solved in this thesis. The old backward projection method used in the game works well for most cases, but results in visual artifacts, like the feet of the character, sink into the terrain in some places in some cases. The cause of this artifact will be shown in Section 2.4. Second, the performance of the collision detection in the game is fine for most scenarios. But large and complex objects added into the scenario will lead a slow performance, such as low *FPS* (frames per second), which will reduce the enjoyment when playing the game. Another problem behind it is that the popular collision detection methods are convex versus convex detection methods. There is no efficient method performing the collision detection with the static terrain, let alone collision detection between large and complex objects with the terrain. Sub-problems behind these two are optimization problems related with objective functions derivation, optimization method selection. These two main problems are tightly related.

The backward projection is the final step of the collision detection. Each vertex of the query dynamic object will be backwards projected to the terrain, and the collision will be detected if the vertex is below the terrain. The error of backward projection will cause error in the collision detection, which lead visual artifacts when playing the game. After the collision detection, the potentially colliding objects together with some places of the terrain will be obtained. In order to handle penetration, the objects and the terrain will be traced back to the former frame, where there is no collision and penetration. The closest distance and the closest points are obtained as constrains, to perform response force.

1.3 Previous Work

Phong Projection is described in [7]. Given an arbitrary point on the surface, the backward Phong projection can find the weights of given anchor points. This projection defines a mapping between two different meshes based on providing point pairs, making better texture transfer and mesh morphing. This method is not

suitable for the problem when an arbitrary point is in the space but not on the surface. However, it offered an optimization idea to find the backward projection.

Methods dealing with non-penetration collision have also been developed to achieve the closest distance for potentially colliding convex pairs. Mihai showed a fixed time-step algorithm for multi-rigid-body dynamical simulation in [6]. Stewart used impulse-momentum equations to deal with simultaneous impacts in [4]. [3] includes research on popular convexity-based methods.

However, the problem we have now is the collision detection between many dynamic objects and the large scale terrain system, and there are always complex objects like vehicles, which contain a large number of components (any object in the game is destructible, and these components will be used in explosion simulation).

Finally, most methods are designed to deal only with small-scale and convex objects and there is very little research on nonconvex collision detection. Nießner, for example, tessellated the objects and ran detection on the GPU in [8], and Katsuaki Kawachi subdivided the Voronoi regions to deal with non-convex polyhedra [9], but they will be costly to tessellate the objects and the terrain.

1.4 Purpose

The overall goal of this project is to increase the accuracy of backward projection and improve the performance and robustness of collision detection.

- The purpose of backward projection is to determine if any vertex of the query object is below the projected surface in the collision detection with the terrain surface. The problem of backwards projecting an arbitrary point in the space onto the base primitive is small but nonlinear. A large number of backward projections will be done in real-time game. The current method used can handle most cases, but performs poorly in some cases. This paper will find two new backward projections to achieve better accuracy.
- The non-convex nature of the displacement mapped surface sets a limit on step-size in ray intersection implementations, and also makes it impossible to exploit vertex connectivity information for convex polytopes in the closest distance or intersection queries. This paper will show a method to handle non-convex problems and to reduce vertices in the query object during the object versus the terrain collision detection. The problem related with collision detection of complex objects will also be solved using multiresolution collision detection with the aim of achieving high efficiency and robustness.

With all these problems solved, the number of detection operations is highly reduced, especially in the collision detection between complex objects and the terrain. The time of collision detection is highly reduced, which allows the game running more smoothly in the same scenario. Moreover, the artifacts in some cases are fixed, generating more realistic simulations in the game. Now the player can run the game smoothly and enjoy better quality.

1.5 Structure of the Work

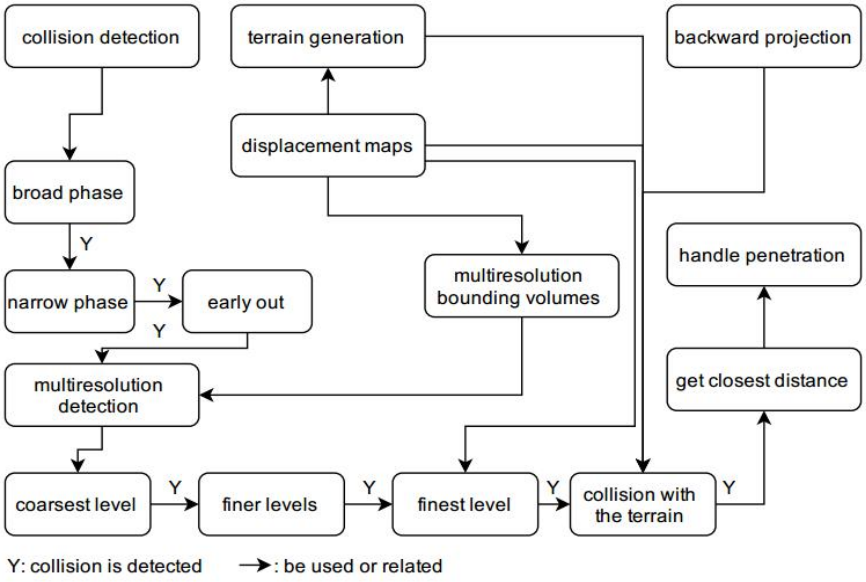


Figure 1.1: The flow chart of this thesis.

All the pieces of work are united as shown in Figure 1.1. The collision detection is performed between dynamic objects and the terrain (Section 2.2). The collision detection consists of the broad phase (Section 2.5) and the narrow phase (Section 2.7). In the narrow phase, the multiresolution collision detection (Section 3.6) is created using multiresolution bounding volumes (Section 3.5). The final collision detection is performed by new backward projections (Section 3.2), to fix the artifacts in some cases.

Chapter 2

Theory

2.1 Overview

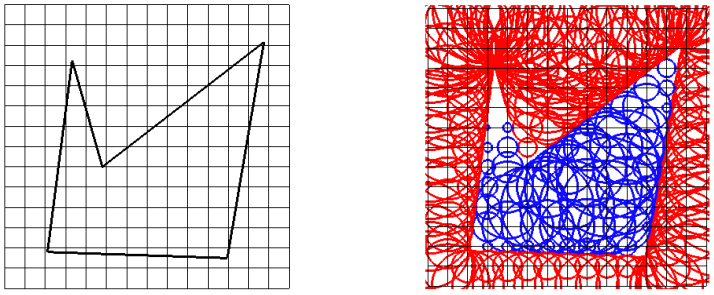
In this chapter, the theories related with this thesis will be discussed. The chapter will be organized in the order: from the generation of the terrain to the collision detection. In Section 2.2, the method to generate the terrain system in the game will be discussed. The commonly used methods will be discussed and compared with the method in the game, to tell the difference and to show the distinctive aspects of the terrain generation in the game. In Section 2.3, the game model used in the collision detection will be shown. The actual collision detection between the dynamics objects and the terrain is performed by the backward projection as shown in Section 2.4. However, it will be extremely costly if the collision detection is performed directly between the objects and the terrain. Most efficient implementations are performed by two phases: a broad phase (as shown in Section 2.5) and a narrow phase (as shown in Section 2.7). After these two phases, two elements in the collision detection are determined: area of the terrain where the collision may happen and the potentially colliding dynamic objects. The collision detection will be performed between these two elements as shown in Section 2.4. After the collision is detected, the closest distance calculation (as shown in Section 2.8) is performed in the former frame to handle penetration.

2.2 Terrain Surface Generation

2.2.1 Iso-surface Extraction

An iso-surface is a 3D surface representing the points in a scalar field with equal values. The volumetric terrain in the game is represented by an iso-surface. The most well known iso-surface extracting method is *Marching Cubes* [10]. The idea is to subdivide the space into uniform voxels and construct surface triangles inside each voxel. The drawback comes from the use of surface configuration of cubes, including wrong surface production and hole generation [11].

The iso-surface in the game is based on editing large uniform scalar grids, representing the signed distance filed. These grids are only used when being edited by artists. Iso-surface extraction is then performed on the signed distance field to obtain a coarse mesh surface [12], [13]. The value at the grid-point is the distance to the nearest point on the surface or the closet distance to the surface, and the sign indicates whether the grid-point is inside or outside of the surface. The interpolated zero iso-contour is the base surface. The triangle primitive in this mesh can have a size 50 times as large as the distance between adjacent values in the edited grid. Figure 2.1 shows a 2D case of surface extraction from a signed distance function. The inner signed distance field (blue circle) and the outer signed distance field (red circle) define a zero iso-surface to approach the input mesh.



(a) Input 2D polygon mesh. (b) Visualized signed distance field.

Figure 2.1: 2D case of isosurface extraction from a signed distance function [14].

2.2.2 Mapping Techniques

Before generating a projected surface, it is necessary to confirm the difference between mapping algorithms, which may cause confusion.

Bump Mapping

Bump mapping is an older way of mapping. The surface created using bump mapping has fake details. It is like drawing a pencil sketch. Bump maps are gray scale images, basically 8 bit color. When the color value in a map is close to 50% gray, there is no detail on the surface. If the value gets brighter, details appear to be pulled out of the surface. Otherwise, a darker value creates details pushing into the surface. Bump mapping is often used to create details on a model, like pores or wrinkles on skin [15]. The advantage is that bump mapping is relatively easy to create, while the disadvantage is that if we look the surface in a different angle, the fake details are quite obvious.

Normal Mapping

Normal mapping is a newer way of bump mapping. The surface created using normal mapping also has fake details. A normal map uses RGB color corresponding to X, Y, Z axis in 3D space. And this RGB color tells the direction of the surface normal (Figure 2.2).

Figure 2.2: 3D model (left), normal map (middle) and applied to a flat surface (right) [16].

Displacement Mapping

If we want to create details on low resolution meshes and actually change the silhouette of the geometry, displacement mapping is needed. Displacement mapping is the mapping technique used in this project. Displacement maps will be discussed in detail the next subsections. Figure 2.3 shows the difference between a bump map and a displacement map. The bump map (left) still has a sphere silhouette, while the displacement map (right) has changed the geometry.

Figure 2.3: Bump Mapping (left) and Displacement Mapping (right) [17].

2.2.3 Displacement Maps Generation

Ray Tracing is used to obtain the relative displacement lengths [18]. A ray is cast from the sampling point on the base primitive along the projection direction of this point. A relative length between the terrain and the base primitive is generated. The computation of normalizing the projection vector will be omitted because of this relative length. After setting the mapping from the points to the displacement maps coordinates, a displacement map is generated. In the displacement map, the reference axes are called s axis and t axis.

2.2.4 Forwards Projected Surface

A base primitive is defined by three vertices and each vertex has its own position, projection direction and s, t coordinates mapping to a displacement map. For a point on this base primitive, its position and projection direction are obtained by interpolation based on barycentric coordinates determined by two parameters u and v as shown in Equation 2.1. We called this point *sampling point*.

$$\begin{aligned} \mathbf{p}(u,v) &= \mathbf{P}_0 + u(\mathbf{P}_1 - \mathbf{P}_0) + v(\mathbf{P}_2 - \mathbf{P}_0), \\ \mathbf{n}(u,v) &= \mathbf{N}_0 + u(\mathbf{N}_1 - \mathbf{N}_0) + v(\mathbf{N}_2 - \mathbf{N}_0). \end{aligned} \tag{2.1}$$

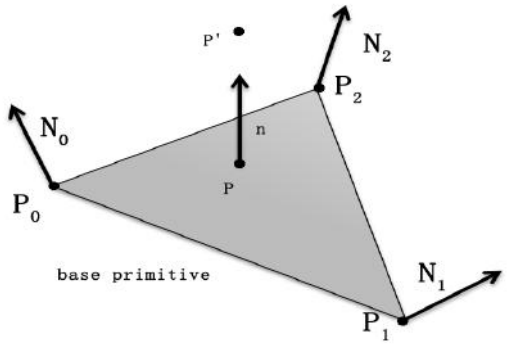


Figure 2.4: A base primitive.

After achieving the base position and the projection direction for the sampling point, the displacement length along the projection direction is retrieved from the displacement maps to forwards project this point to the projected surface (2.2). As shown in Figure 2.4, the point $\mathbf{P}$ is a sampling point and the point $\mathbf{P'}$ is the forwards projected point. Figure 2.5 shows the forward projection in a 2D case. The projected surface is generated by base primitives and displacement maps.

$$\mathbf{p}'(u,v) = \mathbf{p}(u,v) + d(u,v)\mathbf{n}(u,v). \tag{2.2}$$

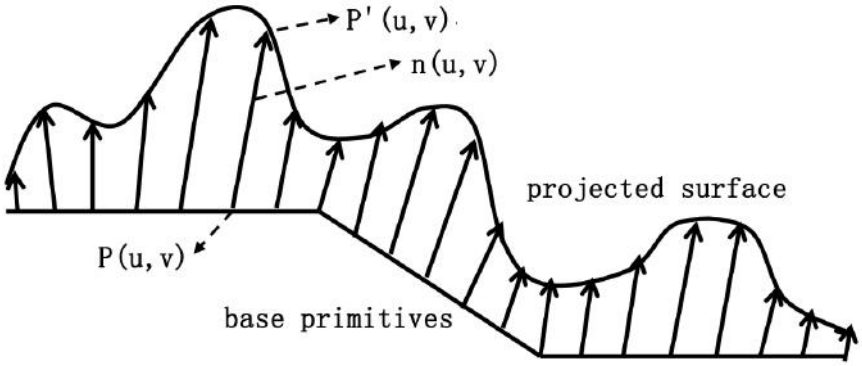
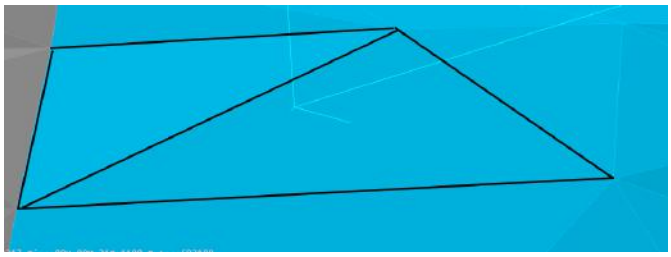
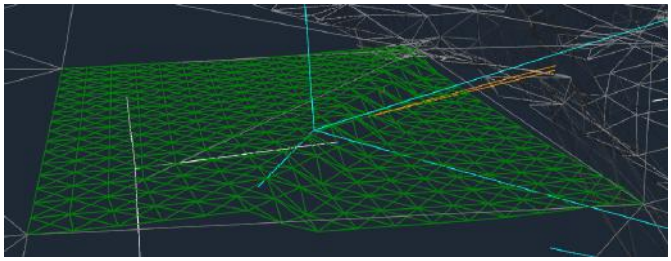


Figure 2.5: Base primitives and the projected surface.



(a) Projected primitives (approximation of projected surface).



(b) Projected surface triangles.

(c) Terrain in the game.

Figure 2.6: Projected surface and primitives [19].

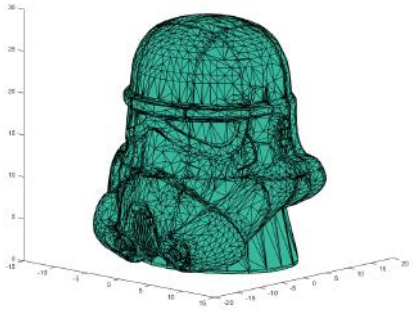
Figure 2.6 shows the process of generating the terrain surface. Figure 2.6(a) shows the primitives. Figure 2.6(b) shows the projected surface triangles. There is

a slope in (b), which can not be seen in the base primitives (a). Figure 2.6(c) shows the projected surface in the game, by adding the detail displacements, texture and vegetation. It is notable that the actual projected surface does not consist of surface triangles, but a list of bilinear patches. It is for eliminating seams on primitive boundaries, it will be described in the following sections.

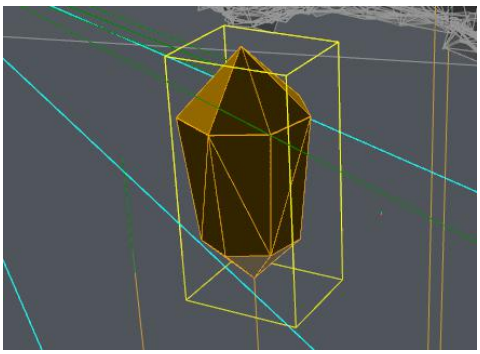
2.3 Collision Model

In the collision detection between the static terrain and the dynamic objects, the dynamic model should have as few points as possible to decrease the number of the backward projections and save time.

The mesh model is the most accurate model constructed by a polygon mesh. Figure 2.7 (a) shows a mesh model of a stormtrooper. This mesh model describes the stormtrooper using vertices, edges and faces. This accurate representation requires a great number of vertices which makes this description quite expensive. An approximation is made to simplify the mesh model and reduce the number of vertices for real-time collision detection (Figure 2.7 (b)). This mesh is computationally efficient due to its simplicity. Another case (shown in Figure A.3) shows the mesh model which approximates a complex warship.



(a)



(b)

Figure 2.7: (a) A surface triangle model of a stormtrooper's head (from `www.stlfinder.com`). (b) Simplified mesh model of the character in the game [19].

2.4 Backward Projection

Backward projection is the inverse problem of forward projection. Backward projection is projecting an arbitrary point in space onto the base primitive. It is important in collision detection because an error will cause visual artifacts in the game (will be discussed in Section 4.2.3).

2.4.1 Aligned Plane method

The method currently used is constructing a plane which passes through the query point $\mathbf{P}'$. This plane is parallel to the base primitive. Three vertices of the primitive triangle are forwards projected to this plane as $\mathbf{P}'_0$, $\mathbf{P}'_1$ and $\mathbf{P}'_2$. Inside $\Delta\mathbf{P}'_0\mathbf{P}'_1\mathbf{P}'_2$, interpolation parameters u, v are obtained. These parameters are used as approximate interpolation parameters in the base triangle to obtain backwards projected point P.

A $2D$ case is taken for example as shown in Figure 2.8. The base point $\mathbf{P}$ is the middle point of the base primitive. Since the interpolation is linear, the forwards projected point $\mathbf{P}'$ is also the middle point of the forwards projected line segment (black line). In the aligned plane method, $\mathbf{P}'$ is the one-third point of the aligned line segment (red line). The same portion $\frac{1}{3}$ is used as the interpolation parameter to find the backwards projected point $\mathbf{P}_a$ on the base primitive. This backwards projected point $\mathbf{P}_a$ is not the same point as $\mathbf{P}$. If forwards project $\mathbf{P}_a$, the forward projected point $\mathbf{P}'_a$ is away from the query point $\mathbf{P}$. That is where error comes from in some cases.

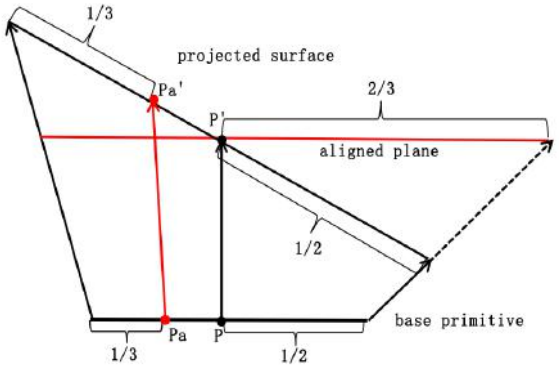
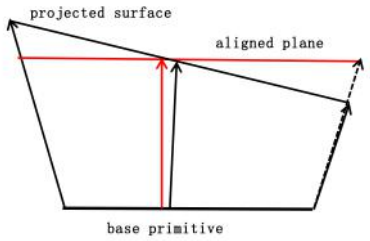


Figure 2.8: Aligned plane method (bad case).

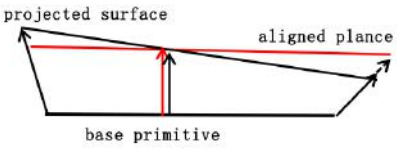
This method works well for most cases in the game. Because in most cases, the lengths along the projection direction of the base primitives are small compared with the size of the base primitive (as Figure 2.9(b) suggests) or the projection direction in one base primitive are close (as Figure 2.9(a) suggests). These reasons enable the aligned plane to be a good approximation of the forwards projected surface. But bad cases can happen and will result in error when doing collision detection with the terrain surface, as shown in Figure 4.6.

2.4.2 Backward Projection in Collision Detection

A vertex is selected in the query object and backwards projected on the base mesh. The distance between this vertex and the backwards projected point is obtained to compare with the displacement length in the backwards projected point. If the distance is smaller than the displacement length, the vertex is under the projected



(a) The projected surface are almost parallel to base primitive.



(b) Most common case in the game.

Figure 2.9: The aligned plane method works well for most cases in the game, because the approximate backwards projected point is close to the exact one.

surface, and the object has collision with the terrain. As shown in Figure 2.10, the projection length of the backwards projected point $\mathbf{p}'$ is smaller than the displacement length, and the collision is detected. Otherwise, there is no collision.

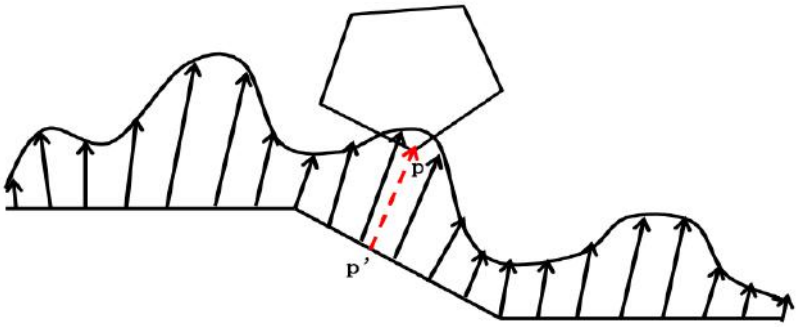


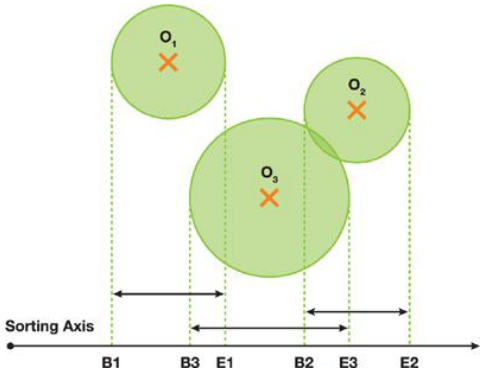
Figure 2.10: The projection length is smaller than the displacement length, and collision is detected.

2.5 Broad Phase

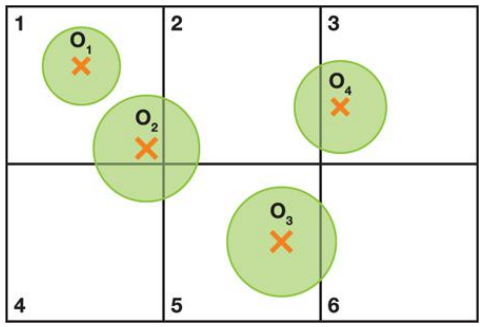
The broad phase collision detection helps to filtrate the potentially colliding object sets. In these colliding candidates, the narrow phase is performed to obtain detail information, like contact points, penetration and normals. The computation in the narrow phase is costly, while the broad phase helps to reject those objects which are not colliding. The brute-force way is doing colliding test all pairs of objects, and the complexity is $\mathcal{O}(n^2)$. Three broad phase algorithms which will reduce the computational complexity are discussed in this section.

2.5.1 Sweep and Prune

This method sorts the starts and the ends of the bounding volume of each object along a number of axes. If the bounding volumes of two objects overlap in all axes, they will have potential to collide [20]. If these two bounding volumes do not overlap in any arbitrary axis, they cannot collide with each other. Figure 2.11 (a) shows x axis projections from three objects. Intersection between intervals of O_1 and O_3 tells to check intersections in other axes. While there is no intersection between intervals of O_1 and O_2, which means they cannot collide and there is no need to check for projections on other axes. The collision test will be performed for each pair of colliding candidates afterwards. This method is combined with a highly efficient tree structure which handles updating colliding candidates in the next frame.



(a) Sweep and prune in one axis.



(b) 2D spatial subdivision.

Figure 2.11: Broad phase algorithms [21].

2.5.2 Spatial Subdivision

Spatial subdivision separates the space into uniform grids. Each grid contains a list of each object whose centroid is inside this grid. The collision test will be performed between two objects which are in the same grid or in the neighbouring grids. In Figure 2.11 (b), a collision test will be performed between O_1 and O_2, but not between O_1 and O_3.

2.5.3 Tree Structure

The method used in the game is by constructing highly efficient trees to record the spatial relationships of the objects. A complex object is record as a parent node and the sub-objects contained in it are set as child nodes. These trees can update efficiently when the dynamic objects move in the game. Figure 2.12 shows the broad phase collision detection between a dynamic object and the terrain. The yellow dash box is the $AABB$ (Axis Aligned Bounding Box) of the chopper. This AABB bounds the *swept volume* (will be discussed in Section 2.7.3) of the chopper from this frame

to the next. The blue dash boxes are the AABBs of the selected forwards projected primitives. These AABBs are fixed due to the static terrain. There will be no collision test since there is no overlapping between the yellow AABB and the blue AABBs.

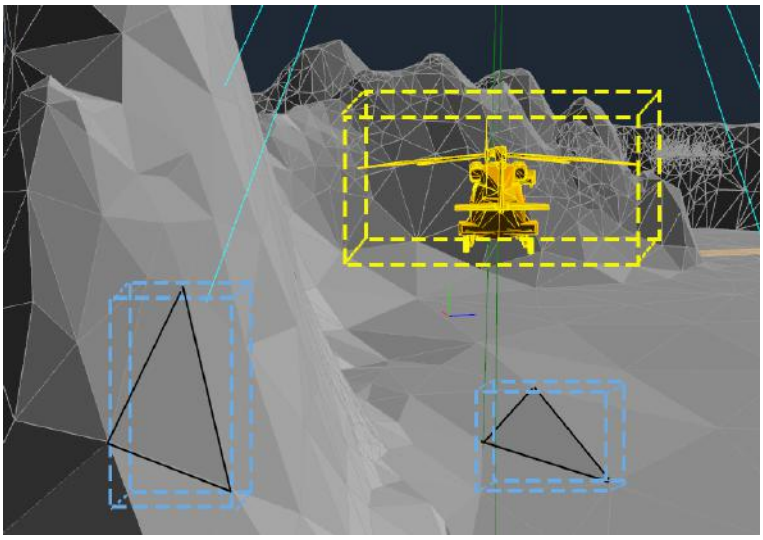


Figure 2.12: The broad phase [19].

2.6 Early Out

Early out is like a middle phase collision detection between the broad phase and the narrow phase. For each forwards projected base primitive, it has an upper bounding plane and a lower bounding plane. Bounding planes are used to bound the projected surface and to reduce the number of backwards projections. The vertices which are between the upper and lower bounding planes are tested in the narrow phase collision detection. The AABB is used to bound the complex object. It can be used with a combination of the bounding planes. If the AABB of the dynamic object does not collide with the bounding plane, there is no need to perform the collision test. The spatial relationship between a vertex and a bounding plane is determined by the sign of the plane equation with the vertex as an input. Figure 2.13 shows the upper bounding planes of two projected base triangles (blue transparent planes). There is no need to perform a narrow phase collision detection since no bounding plane collides with the AABB of the chopper.

2.7 Narrow Phase

In the narrow phase, a list of object pairs will be tested for collision using their actual geometries. The participating colliding shapes are usually restricted to being

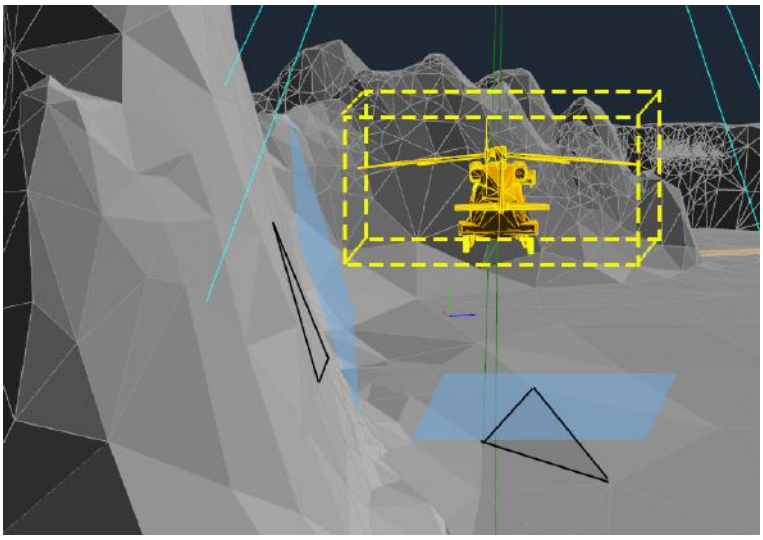


Figure 2.13: Early out check using the bounding planes [19].

convex. For non-convex objects, the convex hull will be used in the collision detection. The problem appears in collision detection between dynamic objects and the static terrain. The large terrain system contains complex geometry, like tunnels, caves and cliffs. Generating sub convex hulls of this terrain system, even for one terrain patch is not straightforward.

2.7.1 Approximate Convex Decomposition

One common idea to handle large complex objects is to partition them into pieces which are easier to handle. [22] shows an approximate convex decomposition algorithm which the user can set the specified tolerance and an elegant hierarchical representation was produced. [23] offered a way to measure the concavity and the shape of detected clusters using a cost function. It also performed automatically detection of structure of the 3D model. However, it is costly to store the approximate convex decomposition or to do it in real-time for the terrain in the game. Although it is not used in this project, the author strongly believe that it is still a method worthy to be used if the data structure can be optimized.

2.7.2 Multiresolution Bounding Volume

Multiresolution bounding volume method is created in this paper for the static terrain collision detection. Figure 2.14 shows the multiresolution upper bounding surface of the terrain in a 2D case. The bounding volume will be constructed by the upper bounding surface and the lower bounding surface (which is not shown in the figure). The bounding volume is a convex polytope, and the convex versus convex collision detection can be performed. If the dynamic object collides with the

coarsest level of bounding volume, a finer resolution bounding volume is checked for collision. It is notable that the bounding volume in the finest resolution will be a tetrahedron formed by four forwards projected points. Nonlinear optimization method will be used to generate the bounding volumes, which will be discussed in the next chapter. All the optimizations are done off-line, the things to be concerned are the memory of data and the bounding accuracy.

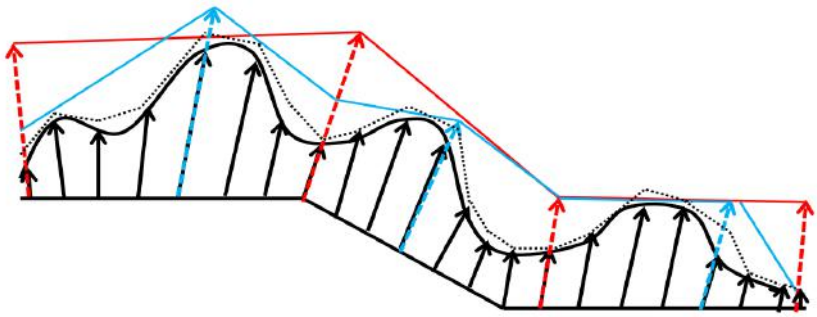


Figure 2.14: The coarsest bounding surface (red line), one level finer bounding surface (blue line) and finest bounding surface (black dot line).

2.7.3 Swept Volume

The AABB in the game bounds not the object, but the *Swept Volume* [24] of the object from one frame to the next.

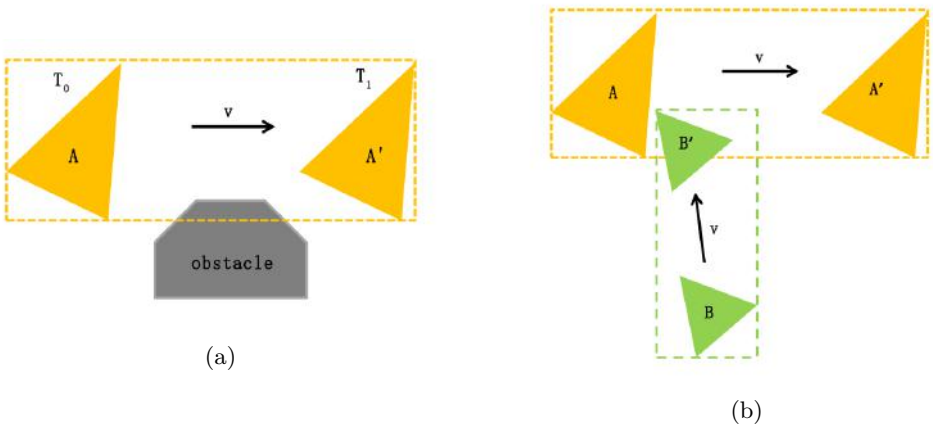


Figure 2.15: (a) Object has collision with obstacle during movement. (b) No collision between objects during movement, while there is collision between the AABBs

In Figure 2.15 (a), the object moves from A to A' from time frame T_0 to T_1 (where T_1 is the predicted frame, and T_0 is the current frame). If the AABB

only bounds the object, collision detection will fail to detect the collision with the obstacle. But if the AABB bounds the swept trajectory, the collision detection can detect the interference between the object and the obstacle. It is notable that if the collision is detected in T_1, this frame will never happen, since there will be response force in current frame (T_0) according to the feedback from the predicted one.

It is also notable that the swept volume detection does not always work between two dynamic objects. As Figure 2.15 (b) suggests, the swept volumes of two objects intersect. But there is no collision between these two objects (two objects reach the same collision position at different time). The AABB of the swept volume is still used in the game since the terrain system is static. For fast moving objects like bullets and missiles, the collision detection is done by ray casting. The bounding volumes of these fast moving objects will be large, resulting costly and inaccurate convex versus convex collision detection.

2.7.4 Convex versus Convex Collision Detection

One of the most commonly used algorithm in game engine is *GJK* (Gilbert-Johnson-Keerthi) algorithm [27]. This algorithm relies on a support function to iteratively get closer simplices to the solution using Minkowski additions.

Minkowski Addition

Given two shapes formed by two sets of points K_A and K_B in Euclidean space. The Minkowski addition of two shapes is formed by adding each point in K_A to each point in K_B,

$$K_A + K_B = \{a + b | a \in K_A, b \in K_B\} \tag{2.3}$$

It is notable that the GJK algorithm uses Minkowski difference in two point sets,

$$K_A - K_B = \{a - b | a \in K_A, b \in K_B\} \tag{2.4}$$

We take a 2D example of

$$K_A = \{(4, 11), (4, 5), (9, 9)\},$$

$$K_B = \{(5, 7), (12, 7), (10, 2), (7, 3)\}.$$

The Minkowski difference is given by

$$\begin{aligned} K_A + K_B = \quad & \{(-1, -4), (-8, 4), (-6, 9), (-3, 8), (-1, -2), (-8, -2), (-6, 3), (-3, 2), \\ & (4, 2), (-3, 2), (-1, 7), (2, 6)\} \end{aligned}$$

.

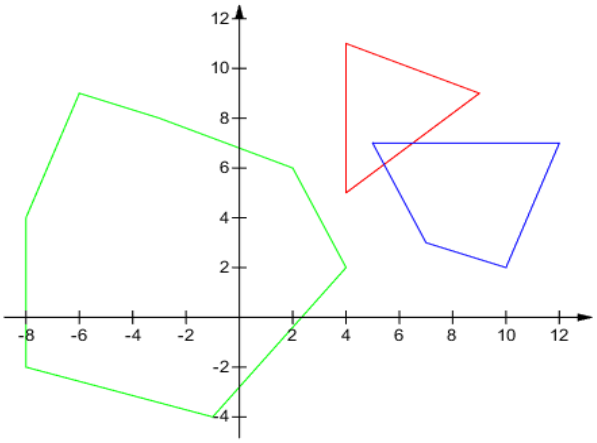
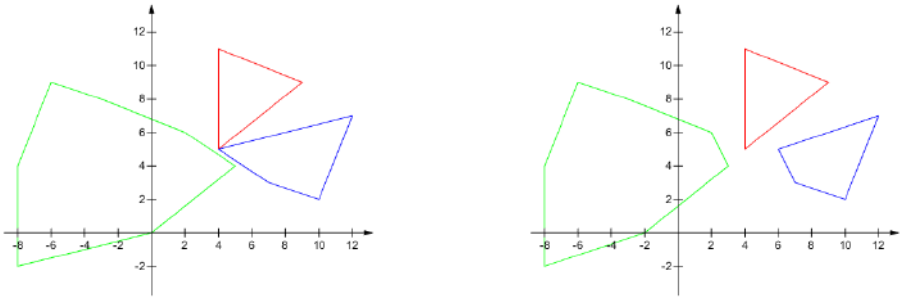


Figure 2.16: Minkowski difference (green) of two shapes (red and blue).

Collision Detection

If two shapes are overlapping or intersecting, the Minkowski sum will contain the origin. The closest distance between two shapes K_A and K_B is the closest distance between the Minkowski difference set $K_A - K_B$ and the origin. The closest point $v(K)$ in the Minkowski difference to the origin corresponds to two closest points in K_A and K_B. The closest distance is $|v(K)|$. Figure 2.16 shows that two shapes and their Minkowski difference when these two shapes intersect. The corresponding Minkowski difference set includes the origin. Figure 2.17 shows two cases: (a) These two shapes collide on boundary, and the origin is on Minkowski boundary. (b) No collision between these two shapes. The origin is outside of the Minkowski difference.



(a) Two shapes collide on one vertex.

(b) No collision.

Figure 2.17: Minkowski difference (green) of two shapes (red and blue).

To find the closest point to the origin in the Minkowski difference point set K, we iteratively build a *simplex* Q (which has $n+1$ points) inside the Minkowski difference that attempts to enclose the origin. If Q contains the origin, then the Minkowski difference contains the origin and the iteration stops. Otherwise, a new simplex Q is generated, guaranteed to contain points closer to the origin than the former one [28]. The iteration will stop with a simplex containing closest point to the origin.

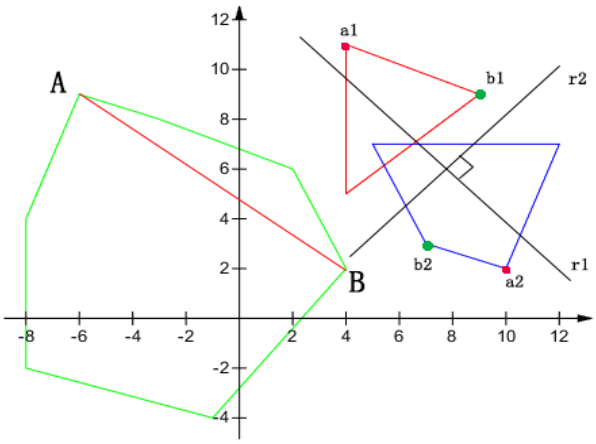


Figure 2.18: The first two vertices in the first simplex.

The first simplex is built using what is called a *support function*. Support function returns one point inside the Minkowski difference given two sets K_A and K_B. From previous discussion, this point is generated by two points, one in K_A and the other in K_B. Choosing the farthest point in a direction and the other farthest one in the opposite ensures that we obtain different points each time, since the point selected only depends on the direction. In Figure 2.18, two farthest points a_1, a_2 (red) in direction r_1 generate point A in the Minkowski difference, while two farthest points b_1, b_2 (green) in direction r_2 (perpendicular to r_1) generate point B. Moreover, all the points returned in this way will locate on the boundary of the Minkowski difference and make a simplex which contains a maximum area.

The iterative search is shown in Figure 2.19. We need to determine one more vertex in simplex after the first two vertices in Figure 2.18. Voronoi region R3 in Figure 2.19 (b) contains the origin, which gives the way to determine the searching direction d in Figure 2.19 (a), i.e. $d = (AC \times AO) \times AC$. The first simplex formed in Figure 2.19 (c) does not contain the origin, but concluding that the two shapes are not intersecting is too early. The first vertex A is discarded, and searching direction d is updated to form the second simplex in Figure 2.19 (e), which contains the origin. Algorithm 1 shows the pseudo code.

Algorithm 1 GJK:

```
function GJK(K_A, K_B)
    choose a searching direction d
    Simplex.add(support(K_A,K_B,d))
    d = −d
    while true do
        Simplex.add(support(K_A,K_B,d))
        if dot(Simplex.lastpoint,d)≤ 0 then
            return false
        else if simplex.isSimplex then
            if containsOrigin(Simplex, d) then
                return true
            else
                update d and discard the first vertex
            end if
        else
            update d and add the third vertex to form a simplex
        end if
    end while
end function
```

2.8 Closest Distance Calculation

After the objects potentially colliding with the terrain are obtained, a closest distance calculation is performed to get the closest distance in the former frame. As [4], [5] suggested, the closest distance is used as a constraint in multi-rigid-body collision optimization. A closest distance calculation considers two objects denoted by two points set K_A and K_B. The closest distance is defined as

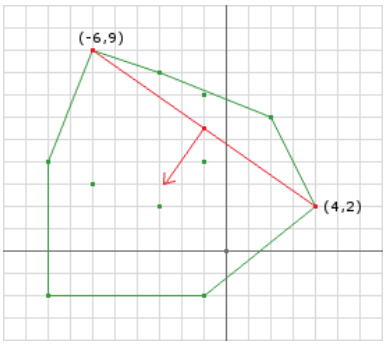
$$d(K_A, K_B) = min\{|x - y| \ |x \in K_A, y \in K_B\}. \tag{2.5}$$

In this section, three collision detection algorithms will be discussed.

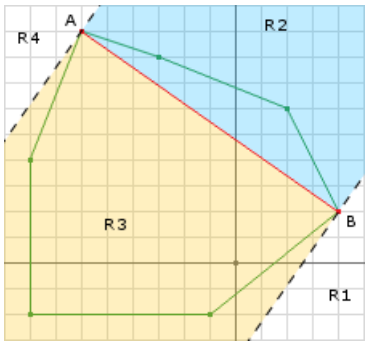
2.8.1 V-Clip Algorithm

V-clip algorithm tracks the closest pair of features between two convex polytopes[25]. The features here include vertices, edges and faces.

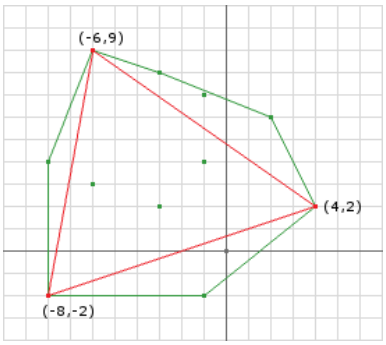
As Figure 2.20 suggests, $V(P_a)$ (transparent blue) denotes the *Voronoi Region* of the point P_a, and $V(E_b)$ (transparent gray) denotes the *Voronoi Region* of the edge E_b. A pair of points P_a and P_b is the closest points between these two features. Because $P_a \in V(E_b)$ and $P_b \in V(P_a)$, this pair of points is the global closest points and the closest distance is obtained between these two points. If they are not in each other's voronoi region, the feature will be updated to a neighbouring feature.



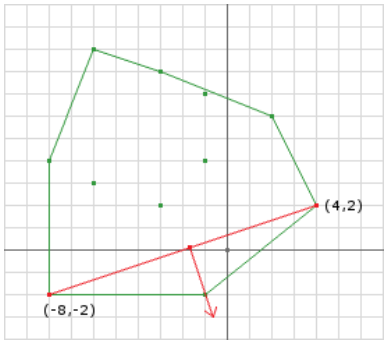
(a) The first iteration.



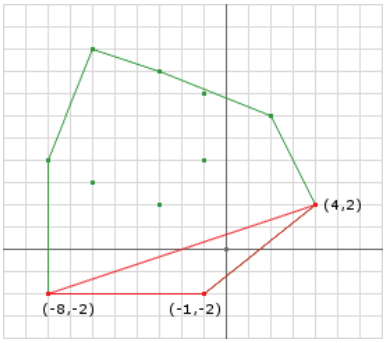
(b) Voronoi regions.



(c) New simplex does not contain the origin.



(d) The second iteration.



(e) New simplex contains the origin.

Figure 2.19: Iterative search of Minkowski difference set K for closest point to the origin by the GJK algorithm [29].

2.8.2 CW Algorithm

The *CW* (Chung-Wang) algorithm [26] is an iterative method to find a separating axis between two polytopes. Algorithm 2 shows the iteration. This algorithm can only check for the intersection between two polytopes by finding a separating axis. The distance calculated along the direction of the separating vector is not accurate. The final iteration step shown in Figure 2.21 (b) suggests the separating axis $d^{\perp}$, but

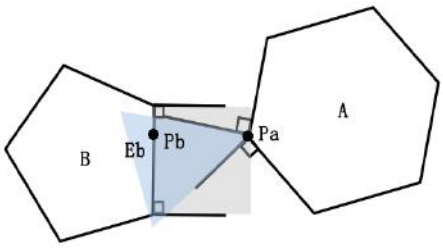
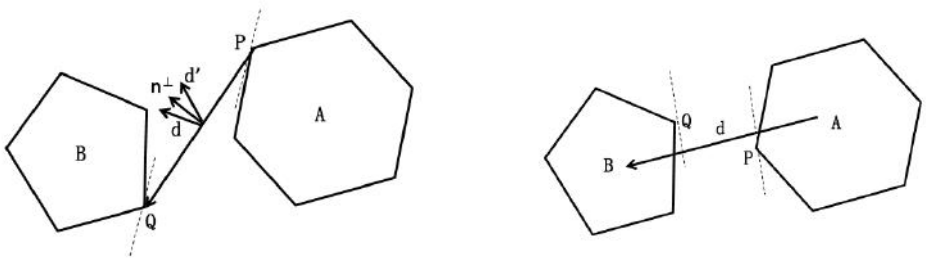


Figure 2.20: V-clip algorithm between two polytopes in a 2D case.

the distance along d is not the closest. However it can be used in the multiresolution bounding volume collision detection.

Algorithm 2 CW:

function $CW(K_A, K_B)$
  choose a separating vector d
  while true **do**
    find extreme point $P \in K_A$ along d, i.e. maximize $P \cdot d$
    find extreme point $Q \in K_B$ along $-d$, i.e. maximize $-Q \cdot d$
    if $P \cdot d < -Q \cdot d$ **then**
      $d^{\perp}$ is the separating axis
      return false
    end if
    update d by $d = d - 2(n \cdot d)n$, where $n = \frac{(Q-P)}{||(Q-P)||}$
  end while
end function



(a) (b)

Figure 2.21: (a) The first iteration of CW algorithm. (b) A separating axis is found in the final step.

2.8.3 GJK Algorithm

Algorithm 3 shows the procedure of calculating the closest distance. This algorithm is mostly identical to detecting collision. The difference is that only one edge of the simplex is kept in each iteration. The point closest to the origin is found on the edge instead of finding a voronoi region which contains the origin.

Algorithm 3 GJK closest distance calculation [29]:

```
function GJK_Dist(K_A, K_B)
    choose a searching direction d
    Simplex.add(support(K_A,K_B,d))
    Simplex.add(support(K_A,K_B,−d))
    while true do
        p=ClosestPointToOrigin(Simplex.a, Simplex.b)
        if norm(p) == 0 then
            return false
        end if
        d=-normlize(p)
        c=support(K_A,K_B,d)
        dc = c · d and da=Simplex.a · d
        if dc − da <tolerance then
            distance = dc
            return true
        end if
        if norm(a)<norm(b) then
            b = c
        else
            a = c
        end if
    end while
end function
```

2.9 Summary

In this chapter, the static terrain and the dynamic object model in the collision detection are clearly shown. The procedure of generating the terrain is illustrated in Section 2.2. The purpose of backward projection in the collision detection is discussed in Section 2.4. However, only the old method used in the game is discussed, the new backward projection methods proposed in this thesis will be described in the next chapter (Chapter 3). The process of collision detection is discussed in two phases: the broad phase (Section 2.5) and the narrow phase (Section 2.7). Different methods used in these two phases are discussed. The idea of the multiresolution bounding volume method is stated, and the detail implementation will be shown in the next chapter (Chapter 3). The following steps after collision detection is discussed in Section 2.8. After obtaining the objects and the terrain which are actually colliding, the closest distance calculation is performed to derive the distances as constrains in the former frames to handle penetration.

Chapter 3

Implementation

3.1 Overview

In this chapter, to solve the problem caused by backward projection as stated in Chapter 2, two new backward projections which improve the accuracy will be shown in Section 3.2. The improvement of bounding planes in the early out part will be shown in Section 3.3. Moreover, Section 3.4 and Section 3.5 will show the procedure of generating the bounding volumes in details, using the derived upper and lower bounding surfaces to construct the bounding volumes. The based mathematical theory of solving inequality-constrained nonlinear programming problems will be discussed in this procedure. The extreme cases and notable details (like data storage, float truncation) in the implementation will also be shown in Section 3.5. In Section 3.6, the implementation of integrating the multiresolution bounding volumes with the collision detection will be discussed.

3.2 New Backward Projections

3.2.1 Aligned Projection Direction

Given a query point $\mathbf{P}'$ in space and the backwards projected point $\mathbf{P}$ on the base primitive. The line $\mathbf{PP}'$ should have the same direction as the interpolated projection direction on $\mathbf{P}$, i.e. the same direction as $\mathbf{n}$ (refer to Figure 2.4).

The objective function is as follows:

$$\begin{aligned} & f = (r \times g) \cdot (r \times g) \\ \text{where} \quad & r = u(\mathbf{N}_1 - \mathbf{N}_0) + v(\mathbf{N}_2 - \mathbf{N}_0) + \mathbf{N}_0 \\ & g = \mathbf{P}' - u(\mathbf{P}_1 - \mathbf{P}_0) - v(\mathbf{P}_2 - \mathbf{P}_0) - \mathbf{P}_0 \end{aligned} \tag{3.1}$$

3.2.2 BFGS Algorithm

The BFGS (Broyden-Fletcher-Goldfarb-Shanno) algorithm is an iterative method to solve unconstrained nonlinear optimization problems [30]. BFGS is a quasi-Newton

method. Hessian matrix is approximated using approximate gradient evaluations. L-BFGS (Limited-memory BFGS) is not used here, because this problem is not a large-scale problem for each query point.

Algorithm 4 BFGS:

function $BFGS(\mathbf{x})$
choose initial guess $\mathbf{x}_0$ and Hessian matrix B_0
while not satisfy terminate condition **do**
get a direction $\mathbf{p}_k$ from $B_k\mathbf{p}_k = -\nabla f(\mathbf{x}_k)$
using line search to find a suitable stepsize α_k in the direction $\mathbf{p}_k$
update $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k\mathbf{p}_k$
$\mathbf{s}_k = \alpha_k\mathbf{p}_k$
$\mathbf{y}_k = \nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k)$
$B_{k+1} = B_k + \frac{\mathbf{y}_k\mathbf{y}_k^T}{\mathbf{y}_k^T\mathbf{s}_k} - \frac{B_k\mathbf{s}_k\mathbf{s}_k^T B_k}{\mathbf{s}_k^T B_k\mathbf{s}_k}$
end while
end function

It is notable that it uses the inverse of the matrix B_k in Algorithm 4. The Sherman-Morrison formula is applied to give the inverse of the matrix B_k.

$$B_{k+1}^{-1} = B_k^{-1} + \frac{(\mathbf{s}_k^T\mathbf{y}_k + \mathbf{y}_k^T B_k^{-1}\mathbf{y}_k)(\mathbf{s}_k\mathbf{s}_k^T)}{(\mathbf{s}_k^T\mathbf{y}_k)^2} - \frac{B_k^{-1}\mathbf{y}_k\mathbf{s}_k^T + \mathbf{s}_k\mathbf{y}_k^T B_k^{-1}}{\mathbf{s}_k^T\mathbf{y}_k} \tag{3.2}$$

Backtracking line search is performed in the line search part. The idea of backtracking line search is that a large step size is set along the searching direction as an initial step size. Then shrink or backtrack the step size until a decrease of objective function is obtained. The BFGS algorithm in use is shown in Algorithm 5.

Algorithm 5 BFGS_modified:

function $BFGS\_modified(\mathbf{x})$
choose initial guess $\mathbf{x}_0$ and Hessian matrix $S_0 = I$
while not satisfy terminate condition **do**
get a direction $\mathbf{p}_k$ from $\mathbf{p}_k = -S_k\nabla f(\mathbf{x}_k)$
using line search to find a suitable stepsize α_k in the direction $\mathbf{p}_k$
update $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k\mathbf{p}_k$
$\mathbf{s}_k = \alpha_k\mathbf{p}_k$
$\mathbf{y}_k = \nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k)$
$S_{k+1} = S_k + \frac{(\mathbf{s}_k^T\mathbf{y}_k + \mathbf{y}_k^T S_k\mathbf{y}_k)(\mathbf{s}_k\mathbf{s}_k^T)}{(\mathbf{s}_k^T\mathbf{y}_k)^2} - \frac{S_k\mathbf{y}_k\mathbf{s}_k^T + \mathbf{s}_k\mathbf{y}_k^T S_k}{\mathbf{s}_k^T\mathbf{y}_k}$
end while
end function

3.2.3 Alpha Plane Method

The idea of this *alpha plane method* is to construct a plane passing through the query point $\mathbf{P}'$, inspired by [31]. Unlike the aligned plane method, this plane is not parallel to the primitive (as shown in Figure 3.1). If the interpolation parameters u, v are obtained in this plane, they will be the same as the ones obtained in the base primitives.

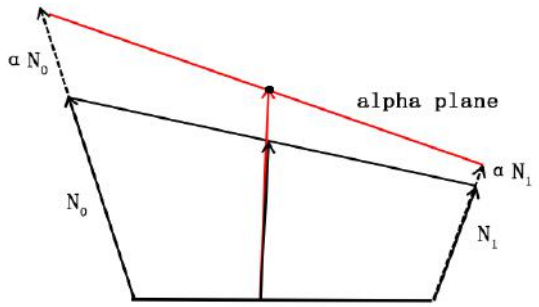


Figure 3.1: Aligned plane method.

The three forward projected vertices on the alpha plane are denoted as $\mathbf{P}'_0$, $\mathbf{P}'_1$ and $\mathbf{P}'_2$, where

$$\begin{aligned} \mathbf{P}'_0 &= \mathbf{P}_0 + \alpha\mathbf{N}_0, \\ \mathbf{P}'_1 &= \mathbf{P}_1 + \alpha\mathbf{N}_1, \\ \mathbf{P}'_2 &= \mathbf{P}_2 + \alpha\mathbf{N}_2. \end{aligned} \tag{3.3}$$

To ensure the alpha plane passing through the query point, the objective function will be:

$$f = ((\mathbf{P}'_1 - \mathbf{P}'_0) \times (\mathbf{P}'_2 - \mathbf{P}'_0)) \cdot (\mathbf{P}' - \mathbf{P}'_0) \tag{3.4}$$

Newton's iteration method is used to obtain α.

3.3 Bounding Plane in Early Out

3.3.1 Bounding Plane of a Forwards Projected Primitive

The upper bounding plane is generated by selecting the maximal projected length on the normal direction of the base triangle, i.e. selecting the maximal $\mathbf{PP}' \cdot (\mathbf{P}_0\mathbf{P}_1 \times \mathbf{P}_0\mathbf{P}_2)$. The method currently used searches points inside the base primitive. Interpolation artifacts will happen in the case that two primitives have large difference in the normal directions, like the cliff case. The adjacent points outside the primitive (but near the boundary) will have an influence on the bounding plane, i.e. if the neighbouring primitive projected to be a cliff, the actual bounding plane will be higher than calculated by just inner points. It is safe to include the projected

lengths of the points on the primitive boundary. Figure 3.2 shows the way to interpolate the points on the primitive edge. Boundary points (green) are the linear interpolation of neighbouring points (yellow), one inside the primitive, the other outside. This interpolation is performed in both s axis and t axis.

The interpolation parameters are determined by the slopes of edges. Figure 3.2 shows the ideal case (no edge of the primitive triangle aligns with s axis or t axis). The extreme cases which the slopes are 0 or infinite will be considered, as shown in Figure 3.3. Algorithm 6 shows the procedure to deal with all conditions.

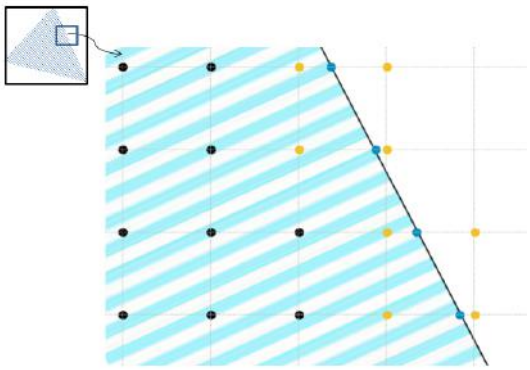


Figure 3.2: Ideal case (no edge of the primitive triangle aligns with s axis or t axis).

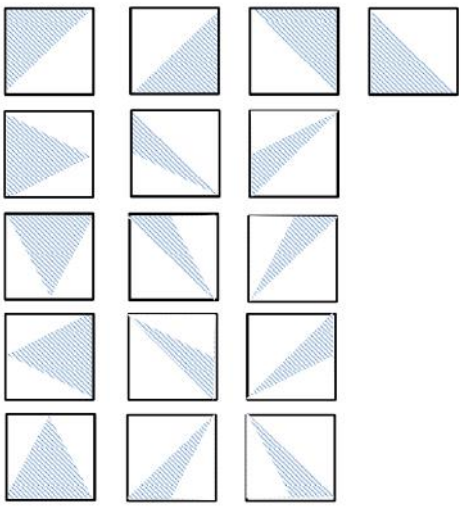


Figure 3.3: Primitive (shadowy triangle) with at least one axis aligned edge.

Algorithm 6 Initial bounding planes of the projected surface:

function *InitialBoundingPlane*(base triangle)
  for each point inside the base primitive **do**
    select the maximal and minimal bounding planes of the projected surface
  end for
  check each edge
  if it is aligned with s axis or t axis **then**
    update maximal and minimal bounding plane accordingly (for an edge aligned with the axis, sampling points are on the edge, and no interpolation is needed for this edge)
  end if
  if no edge is aligned with s axis or t axis (ideal case) **then**
    get interpolated bounding plane along every edge
    update maximal and minimal bounding plane
  end if
end function

3.4 Bounding Surface Triangles

As mentioned in Section 2.2.4, the forwards projected surface consists of a list of bilinear patches for smoother terrain, but not surface triangles.

3.4.1 Bilinear Patch

Bilinear patches are formed by four possibly non-coplanar points, $(p_{00}, p_{01}, p_{10}, p_{11})$, with a weighting of two parameters (u, v), such that any point p can be represented by the equation below with $(u, v) \in [0, 1]^2$:

$$\begin{aligned} p(u,v) &= (1-u)(1-v)p_{00} + (1-u)vp_{01} + u(1-v)p_{10} + uvp_{11} \\ &= uv(p_{11} - p_{10} - p_{01} + p_{00}) + u(p_{10} - p_{00}) + v(p_{01} - p_{00}) + p_{00} \end{aligned} \tag{3.5}$$

This equation can be obtained by multiple linear interpolations. An arbitrary p is derived by linear interpolation of P_{0v} and P_{1v} with weighting $1-u$, u respectively, and P_{0v} and P_{1v} are derived by linear interpolation of p_{00}, p_{01} and p_{10}, p_{11} respectively, with weighting $(1-v)$, v. As equations below:

$$\begin{aligned} P_{0v} &= (1-v)p_{00} + vp_{01} \\ P_{1v} &= (1-v)p_{10} + vp_{11} \end{aligned} \tag{3.6}$$

Similarly, $p(u, v)$ is derived from P_{0v} and P_{1v} as

$$p(u,v) = (1-u)P_{0v} + uP_{1v} \tag{3.7}$$

3.4.2 The Upper and Lower Bounding Triangles

It is notable that the patch has a facing direction, which has as a result that only one type of triangles can bound one bilinear patch.

A test case is discussed as follows: A bilinear patch is formed by four non-coplanar points: $p_{00}([0,0,1])$, $p_{01}([1,0,1])$, $p_{10}([0,1,0.5])$, $p_{11}([1,1,4])$. Figure 3.4 shows a bilinear patch formed by these four points. Assume that the patch faces upwards.

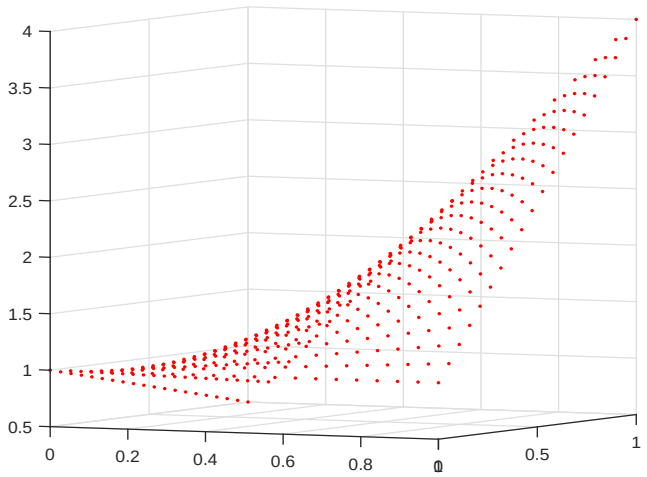
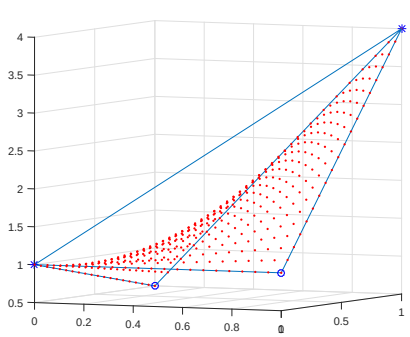
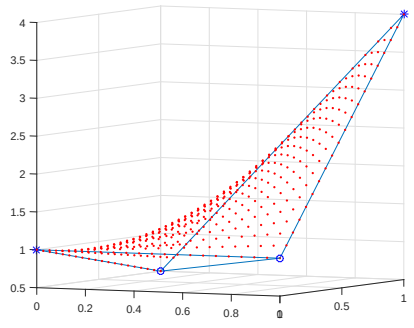


Figure 3.4: Bilinear patch with upward facing direction.

Figure 3.5 shows two types of triangles, case (a) can bound the patch, but case (b) cannot. If the facing direction is set downwards, the bounding triangle type will be reversed.



(a) Two triangles can bound the bilinear patch.



(b) Two triangles cannot bound the bilinear patch.

Figure 3.5: Two types of triangles.

The proof that the two triangles can bound patch is trivial, using the fact that $(u, v) \in [0, 1]^2$, which ensures the interpolation point is always below the triangle plane.

One question comes up naturally: how to determine bounding triangle type? We cannot simply take the midpoint of pair p_{00}, p_{11} and pair p_{01}, p_{10}, to compare the Z-coordinate. Not all bilinear patches face upwards. The patches in a tunnel can face downwards, and the patches in a cliff can face horizontally.

A novel idea is used here (still use the test case above). As Figure 3.6 suggests, a triangle plane is determined by p_{00}, p_{01} and p_{10}. If p_{11} is above the plane, i.e. the dot product of triangle normal vector $\mathbf{N}$ and vector $\mathbf{p}_{00}\mathbf{p}_{11}$ is positive, $\Delta p_{00}p_{11}p_{01}$ and $\Delta p_{00}p_{11}p_{10}$ are taken as two bounding triangles. If the dot product is negative, $\Delta p_{10}p_{01}p_{00}$ and $\Delta p_{10}p_{01}p_{11}$ are used instead.

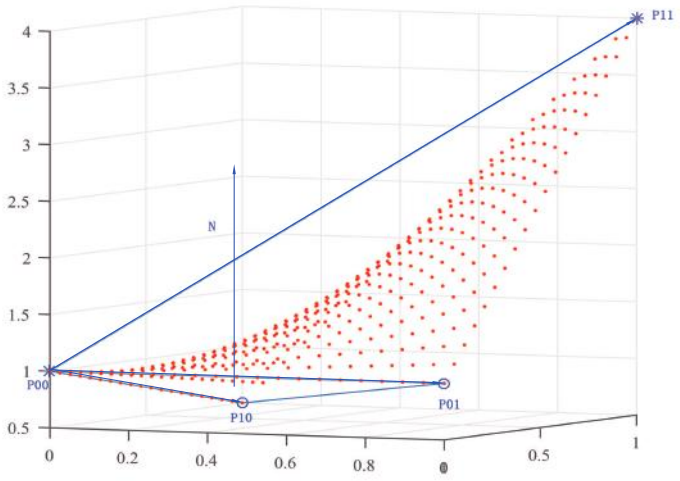


Figure 3.6: Determine triangle type.

3.5 Multiresolution Bounding Volume

3.5.1 Problem Introduction

The terrain map consists of 64×64 terrain patches. Each terrain patch consists of several non-uniform terrain quads. As Figure 3.7 suggests, the quad in the left bottom is a 16×16 (grids) quad. While for the up one, it has 8×8 grids. It is notable that there is no triangle primitive across any quad boundary, and the vertices of triangle primitives all lie in the sampling nodes (small gray dots in Figure 3.9).

Optimization is done separately for each quad. The reason we cannot do it for an entire terrain patch is that location relationship in the terrain patch displacement maps is not corresponds with location relationship the in world space. As Figure 3.8 suggests, the area circled by green line is a 4×4 quad, and the one circled by red

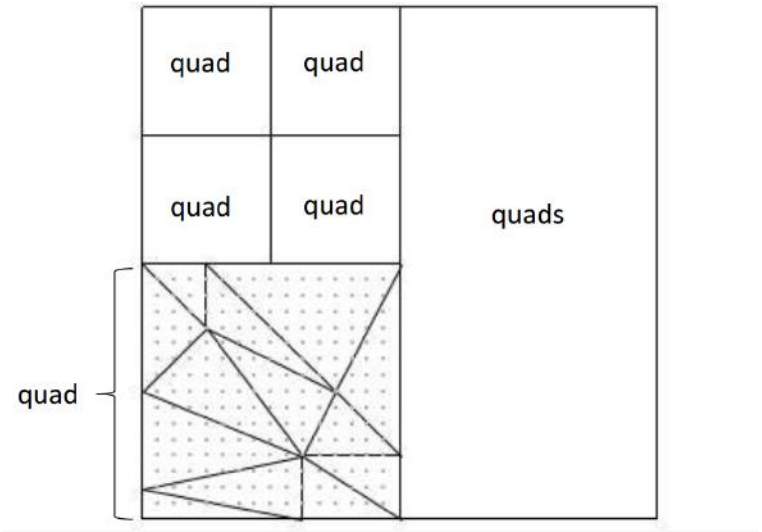


Figure 3.7: Terrain Patch.

line is a 2×2 quad. In the displacement maps (the 4×4 quad and one neighbouring 2×2 quad shown in Figure 3.7), these two quads are adjacent, but they are detached in the world space.

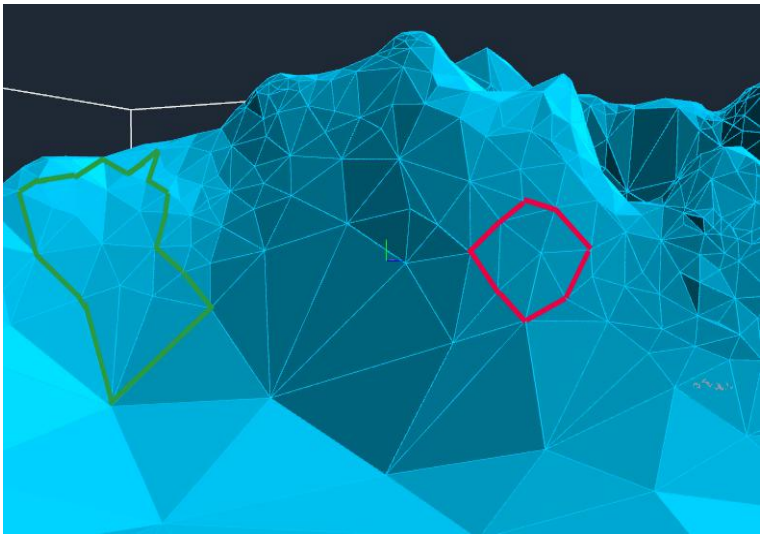


Figure 3.8: These two quads are detached in the world space. 4×4 (green) quad and 2×2 (red) quad [19].

Assume that bounding box in the coarsest level is a 8×8 quad, taking the left-bottom quad in Figure 3.7 for example.

In this quad (Figure 3.9), four bounding boxes in the displacement maps are determined by their vertices (marked with green circle). These vertices are called

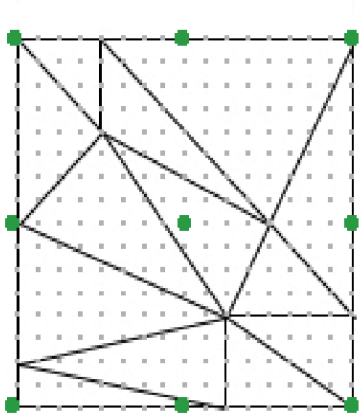


Figure 3.9: A 16×16 quad with constrained points (grey dots) and bounding points (green dots).

bounding points. Those gray points are called *constrained points*. Every four bounding points will form two upper bounding triangles and two lower bounding triangles. The shape of triangles is discussed in Section 3.4. The upper bounding surface is generated to keep all the forwards projected points under the surface or on it. Relatively, the lower bounding surface will keep all the projected points up or on it. Figure 3.10 shows a 2D case of the bounding volumes constructed by the upper and lower bounding surfaces. It is notable that there is a dislocation between bounding volumes in neighbouring quads due to the separating optimizations. Figure 3.11 shows the 3D dislocations in the game.

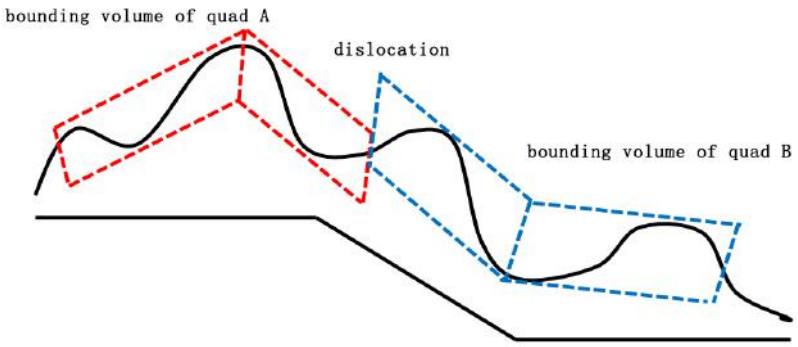


Figure 3.10: There will be a dislocation between bounding volumes between neighbouring quads.

Dislocations in the 2D case is acceptable since there is no gap between two bounding volumes. However 3D bounding volumes will risk this problem, which may cause small objects falling into the gap with no collision detection. It is solved

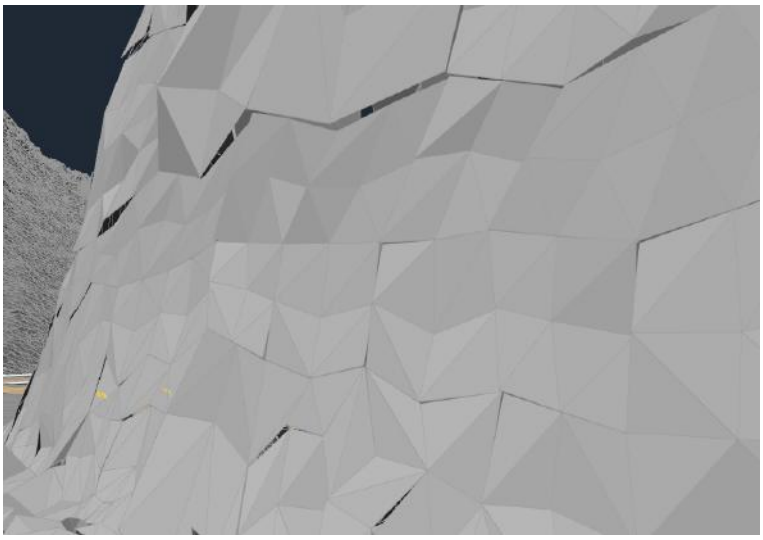


Figure 3.11: Dislocations between upper bounding surface among neighbouring quads [19].

by generating convex bounding volumes from the upper and lower bounding surfaces. For the side faces, which are bilinear patches, the same method generating bounding surface triangles is used to keep the bounding surfaces of side faces convex. Figure 3.12 shows two bounding volumes. The intersection part between these two volumes eliminates the gaps.

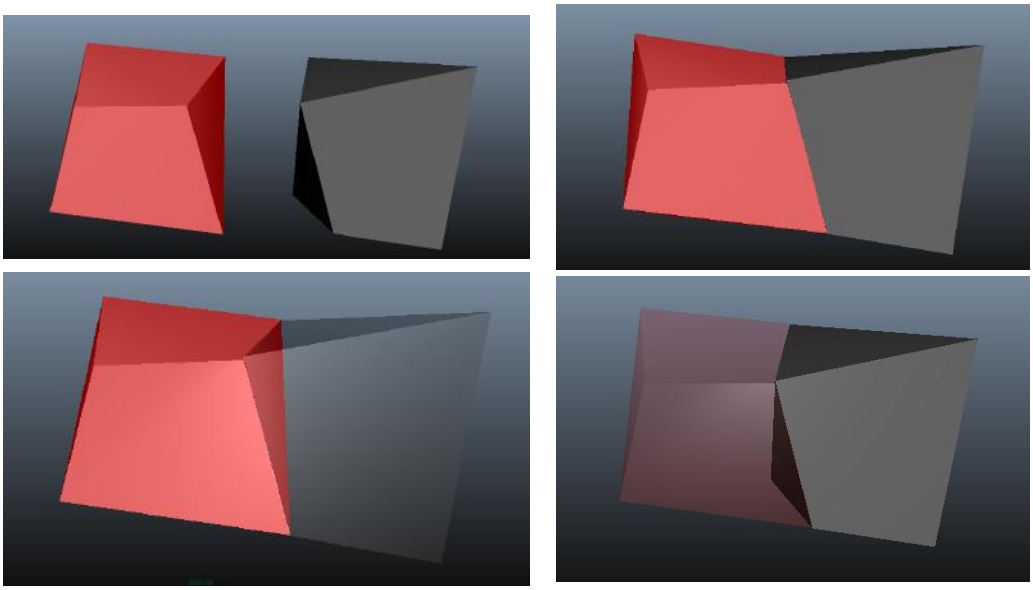


Figure 3.12: There will be no gap between neighbouring convex bounding volumes.

3.5.2 Inequality-constrained Nonlinear Programming Problem

The problem is inequality-constrained nonlinear problem which will be discussed later. At first, definition of the inequality-constrained nonlinear programming problem and the method to solve it will be introduced. Inequality-constrained nonlinear programming problem can be written in the following form:

$$\begin{array}{lll} \text{minimize} & f_0(x) & \\ \text{subject to} & f_i(x) \leq 0, & i = 1, \cdots, m, \\ & x_j^{\min} \leq x_j \leq x_j^{\max} & j = 1, \cdots, n. \end{array} \tag{3.8}$$

where $\mathbf{x} = (x_1, \cdots, x_n)^T \in \mathbb{R}^n$ is the vector of variables, $x_j^{\min}$ and $x_j^{\max}$ are the lower and upper bounds of x_j. Typically, the objective function f_0 and the constraint functions $f_0, f_1, \cdots, f_m$ are twice continuously differentiable.

However, this problem will be transformed into an extended form as following:

$$\begin{array}{lll} \text{minimize} & f_0(x) + \sum_{i=1}^{m} c_i y_i & \\ \text{subject to} & f_i(x) - y_i \leq 0, & i = 1, \cdots, m, \\ & x_j^{\min} \leq x_j \leq x_j^{\max} & j = 1, \cdots, n, \\ & y_j \geq 0, & i = 1, \cdots, m. \end{array} \tag{3.9}$$

Formulation 3.9 is used instead of Formulation 3.8. For Formulation 3.9, there is a feasible solution and at least one optimal solution , and this optimal solution satisfies the KKT conditions [32]. Or we can view from the modelling side. When the right hand side of constraint in Formulation 3.8 increases one unit, how much would the objective function changes to accept this increase.

3.5.3 Optimization Method

The problem is nonlinear problems with nonlinear inequality constraints. The MMA (Method of Moving Asymptotes) is used since the gradients of both the objective function and the nonlinear inequality constraints can be derived.

CCSA (conservative convex separable approximation)

A CCSA (Conservative Convex Separable Approximation) method contains the outer and inner iterations. Within each outer iteration, there will be no or several inner iterations.

Before going into details, the optimization Formulation 3.9 needs to be changed. Formulation 3.10 is a general form of 3.9, with one more artificial variable $z \in \mathbb{R}$.

$$\begin{aligned} \text{minimize} \quad & f_0(x) + a_0 z + \textstyle\sum_{i=1}^{m}(c_i y_i + \frac{1}{2} d_i y_i^2) \\ \text{subject to} \quad & f_i(x) - a_i z - y_i \le 0, && i = 1, \cdots, m, \\ & x_j^{\min} \le x_j \le x_j^{\max} && j = 1, \cdots, n, \\ & z \ge 0 \text{ and } y_i \ge 0, && i = 1, \cdots, m, \end{aligned} \tag{3.10}$$

where a_0, a_i, c_i and d_i are given real numbers such that $a_0 > 0$, $a_i \ge 0$, $c_i \ge 0$, $d_i \ge 0$, and $c_i + d_i > 0$, $a_i c_i > a_0$ for $i = 1, \cdots, m$. The algorithm is described in Algorithm 7. The index k represents the outer iteration number and l represents the inner iteration number. The subproblem takes the form as follows:

$$\begin{aligned} \text{minimize} \quad & g_0^{(k,l)}(x) + a_0 z + \textstyle\sum_{i=1}^{m}(c_i y_i + \frac{1}{2} d_i y_i^2) \\ \text{subject to} \quad & g_i^{(k,l)}(x) - a_i z - y_i \le 0, && i = 1, \cdots, m, \\ & x \in X^{(k)}, z \ge 0 \text{ and } y_j \ge 0, \end{aligned} \tag{3.11}$$

where $X^{(k)} = \{x \in X | x_j \in [x_j^{(k)} - 0.9\sigma_j^{(k)}, x_j^{(k)} + 0.9\sigma_j^{(k)}], j = 1, \cdots, n\}$. The parameter $\sigma_j^{(k)}$ will be specified in MMA subsection. The function $g_i^{(k,l)}(x)$ in this subproblem is chosen as:

$$g_i^{(k,l)}(x) = v_i(x, x^{(k)}, \sigma^{(k)}) + \rho_i^{(k,l)} \omega_i(x, x^{(k)}, \sigma^{(k)}),\ i = 0, 1, \cdots, m. \tag{3.12}$$

where v_i and w_i are real value functions, which will also be specified in next subsection MMA.

MMA(Method of Moving Asymptotes)

It is an improved CCSA (conservative convex separable approximation) variant of original method of moving asymptotes as described in [32]. This method is intended for inequality-constrained nonlinear programming problems. The brief idea is: at each point x, it forms a local approximation based on the gradient of f, the constraints, and a quadratic penalty term to make this approximation conservative (upper bounds for the exact functions). This approximation is both convex and separable, making it possible to solve this optimization problem using a dual method. Solving this problem will generate a new point x. If the approximation is conservative, we start at this new point x. Otherwise, the penalty term will be increased, the approximation will be made more conservative and to be optimized again [33].

The function g_i in subproblem is chosen as:

$$g_i^{(k,l)}(x) = \sum_{j=1}^{n}\left(\frac{p_{ij}^{(k,l)}}{u_j^{(k)} - x_j} + \frac{q_{ij}^{(k,l)}}{x_j - l_j^{(k)}}\right) + r_i^{(k,l)}. \tag{3.13}$$

Algorithm 7 CCSA:

function $CCSA(f_0,\ f_i)$
choose $x^{(0)} \in X$, calculate $y^{(0)}$ and $z^{(0)}$
outer iteration
while true **do**
form subproblem
replace f_i with strictly convex separable function $g_i^{(k,l)}(x)$
s.t. $g_i^{(k,l)}(x^{(k)}) = f_i(x^{(k)})$
get optimal solution $(\hat{x}^{(k,l)}, \hat{y}^{(k,l)}, \hat{z}^{(k,l)})$
inner iteration
while $g_i^{(k,l)}(\hat{x}^{(k)}) < f_i(\hat{x}^{(k)})$ **do**
update $g_i^{(k,l)}(x)$, s.t. $g_i^{(k,l)}(x^{(k)}) = f_i(x^{(k)})$
update optimal solution $(\hat{x}^{(k,l)}, \hat{y}^{(k,l)}, \hat{z}^{(k,l)})$
if reach maximal iteration step or reach tolerance **then**
jump out this inner iteration
end if
end while
if reach maximal iteration step or reach tolerance **then**
break
end if
end while
end function

where $u_j^{(k)}$ and $l_j^{(k)}$ are moving asymptotes, $u_j^{(k)} = x_j^{(k)} + \sigma_j^{(k)}$ and $l_j^{(k)} = x_j^{(k)} - \sigma_j^{(k)}$. The coefficients $p_{ij}^{(k,l)}$, $q_{ij}^{(k,l)}$ and $r_{ij}^{(k,l)}$ are given by

$$\begin{aligned} p_{ij}^{(k,l)} &= (\sigma_j^{(k)})^2 max\{0, \tfrac{\partial f_i}{\partial x_j}(x^{(k)})\} + \tfrac{\rho_i^{(k,l)}\sigma_j^{(k)}}{4}, \\ q_{ij}^{(k,l)} &= (\sigma_j^{(k)})^2 max\{0, -\tfrac{\partial f_i}{\partial x_j}(x^{(k)})\} + \tfrac{\rho_i^{(k,l)}\sigma_j^{(k)}}{4}, \\ r_i^{(k,l)} &= f_i(x^{(k)}) - \textstyle\sum_{j=1}^n \tfrac{p_{ij}^{(k,l)} + q_{ij}^{(k,l)}}{\sigma_j^{(k)}}. \end{aligned} \tag{3.14}$$

If we take the Formulation 3.12, v_i and w_i will be as follows:

$$\begin{aligned} v_i(x,\xi,\sigma) &= f_i(\xi) + \textstyle\sum_{j=1}^n \tfrac{\sigma_j^2 \frac{\partial f_i}{\partial x_j}(\xi)(x_j-\xi_j) + \sigma_j |\frac{\partial f_i}{\partial x_j}(\xi)|(x_j-\xi_j)^2}{\sigma_j^2 - (x_j-\xi_j)^2}, \\ w_i(x,\xi,\sigma) &= \tfrac{1}{2}\textstyle\sum_{j=1}^n \tfrac{(x_j-\xi_j)^2}{\sigma_j^2-(x_j-\xi_j)^2}. \end{aligned} \tag{3.15}$$

Next we determine parameters $\rho_i^{(k,l)}$ and $\sigma_j^{(k)}$ in Formulation 3.12:

$$\begin{aligned} \rho_i^{(1,0)} &= 1, \\ \rho_i^{(k+1,0)} &= max\{0.1\rho_i^{(k,l(k))}, \rho_i^{\min}\}, \end{aligned} \tag{3.16}$$

where $l(k)$ is the number of inner iterations in the kth outer iteration. Inside each inner iteration, $\rho_i^{(k,l)}$ is based on the solution of the last subproblem. If $g_i^{(k,l)}(\hat{x}^{(k,l)}) < f_i(\hat{x}^{(k,l)})$, we choose $\rho_i^{(k,l+1)}$ to make it fulfill $g_i^{(k,l+1)}(\hat{x}^{(k,l)}) = f_i(\hat{x}^{(k,l)})$. Here we set $\rho_i^{(k,l+1)} = \rho_i^{(k,l)} + \sigma_i^{(k,l)}$, where $\sigma_i^{(k,l)} = \frac{f_i(\hat{x}^{(k,l)}) - g_i^{(k,l)}(\hat{x}^{(k,l)})}{w_i(\hat{x}^{(k,l)}, x^{(k)}, \sigma^{(k)})}$ from (3.12). Otherwise, if $g_i^{(k,l)}(\hat{x}^{(k,l)}) \geq f_i(\hat{x}^{(k,l)})$, we keep $\rho_i^{(k,l+1)}$ unchanged, i.e. $\rho_i^{(k,l+1)} = \rho_i^{(k,l)}$.

As for the value of $\sigma_j^{(k)}$:

$$\sigma_j^{(k)} = \begin{cases} 0.5(x_j^{\max} - x_j^{\min}), & \text{for } \ k = 1, 2 \\ \sigma_j^{(k)} = \gamma_j^{(k)} \sigma_j^{(k-1)}, & \text{for } \ k \geq 3 \\ , \end{cases} \tag{3.17}$$

where

$$\gamma_j^{(k)} = \begin{cases} 0.7, & \text{if } \ (x_j^{(k)} - x_j^{(k-1)})(x_j^{(k-1)} - x_j^{(k-2)}) < 0, \\ 1.2, & \text{if } \ (x_j^{(k)} - x_j^{(k-1)})(x_j^{(k-1)} - x_j^{(k-2)}) > 0, \\ 1, & \text{if } \ (x_j^{(k)} - x_j^{(k-1)})(x_j^{(k-1)} - x_j^{(k-2)}) = 0, \end{cases}$$

The parameter determined in $\gamma_j^{(k)}$ is discussed in [33]. The brief idea is: If a certain variable x_j is oscillating, it will be stabilized by decreasing the corresponding σ_j. If the variable x_j is monotonically increasing or decreasing, it will be released by increasing the corresponding σ_j.

3.5.4 Implementation

The optimization problem is formed as follows:

$$\begin{array}{lll} \text{minimize} & \sum_{i=0}^{N} f(x_i) & \\ \text{subject to} & Q_j(x_i) \leq 0 & i = 1, \cdots, N, \\ & & j = 1, \cdots, M. \end{array} \tag{3.18}$$

where $x_i \in \mathbb{R}$ is the variable, which represents the forwards projected length from a bounding point. N is the number of the bounding points. M is the number of the inequality constraints.

To make the bounding surface bound the surface as tightly as possible, the objective function $f(x_i)$ can be derived as follows, i.e. to keep the distances between the bounding surface and the terrain as small as possible:

$$f(x_i) = (x_i - l_i)^2 \tag{3.19}$$

where l_i is the original displacement length in the bounding point $\mathbf{p}_j$. The constrains Q_j can be derived as following (to keep the forwards projected points between the bounding planes):

$$Q_j = (\mathbf{P}_1 - \mathbf{P}_0) \times (\mathbf{P}_2 - \mathbf{P}_0) \cdot (\mathbf{P}_j - \mathbf{P}_0) \tag{3.20}$$

where $\mathbf{P}_0, \mathbf{P}_1$ and $\mathbf{P}_2$ are the forwards projected points of the bounding points $\mathbf{p}_0$, $\mathbf{p}_1$ and $\mathbf{p}_2$. $\mathbf{P}_j$ is the forwards projected point of a constrained point $\mathbf{p}_j$. If substituted by the detailed formulation, Q_j can be written below:

$$Q_j = [\mathbf{p}_1 + \mathbf{r}_1 l_1 - (\mathbf{p}_0 + \mathbf{r}_0 l_0)] \times [\mathbf{p}_2 + \mathbf{r}_2 l_2 - (\mathbf{p}_0 + \mathbf{r}_0 l_0)] \cdot [\mathbf{p}_j + \mathbf{r}_j l_j - (\mathbf{p}_0 + \mathbf{r}_0 l_0)] \quad (3.21)$$

The derivative of Q_j can be derived as:

$$\begin{aligned}
\frac{\partial Q_j}{\partial l_0} &= [-\mathbf{r}_0 \times (\mathbf{P}_2 - \mathbf{P}_0) - (\mathbf{P}_1 - \mathbf{P}_0) \times \mathbf{r}_0] \cdot (\mathbf{P}_j - \mathbf{P}_0) + [(\mathbf{P}_1 - \mathbf{P}_0) \times (\mathbf{P}_2 - \mathbf{P}_0)] \cdot (-\mathbf{r}_0) \\
\frac{\partial Q_j}{\partial l_1} &= [\mathbf{r}_1 \times (\mathbf{P}_2 - \mathbf{P}_0)] \cdot (\mathbf{P}_j - \mathbf{P}_0) \\
\frac{\partial Q_j}{\partial l_1} &= [(\mathbf{P}_1 - \mathbf{P}_0) \times \mathbf{r}_2] \cdot (\mathbf{P}_j - \mathbf{P}_0)
\end{aligned} \quad (3.22)$$

Nonlinear optimization library *NLopt* [34] is used to integrated in current $C++$ project. NLopt is a open-source library providing a common interface for a number of different optimization routines. Algorithm 8 shows the procedure to generate the bounding volumes.

Algorithm 8 Bounding Volume:

```
function BoundingVolume(displacement maps)
    for every terrain patch do
        for every quad inside one terrain patch do
            if bounding size ≥ quad size then
                bounding size = quad size
            end if
            loop the triangle primitives in this quad
            get data of the base positions and the projection directions
            while resolution ≠ 0 do
                upper bounding surface
                set the initial bounding lengths x_i^0 = l_i + 5
                Optimization using MMA
                lower bounding surface
                set the initial bounding lengths x_i^0 = l_i − 5
                Optimization using MMA

                store the bounding map displacements and the resolution
                go to the finer resolution
            end while
        end for
    end for
end function
```

3.5.5 Data Storage

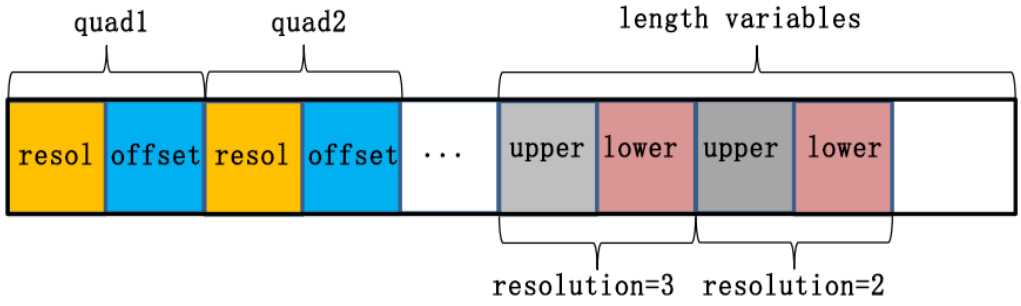


Figure 3.13: The data structure of the bounding volumes. The former part stores the resolution and offset (to retrieve the projection lengths), and the latter part stores the bounding displacement lengths. The lengths in one quad will follow the order: from upper to lower, from the coarsest to the finer.

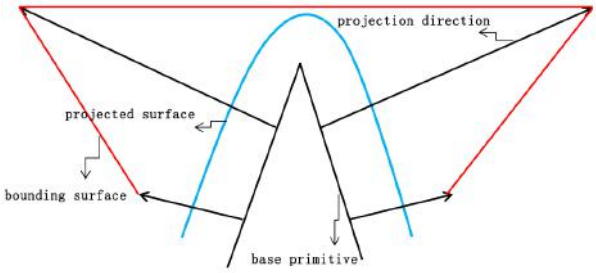
The information of the resolutions and the optimal bounding lengths are kept in the form shown in Figure 3.13.

3.5.6 Truncation from Double to Float

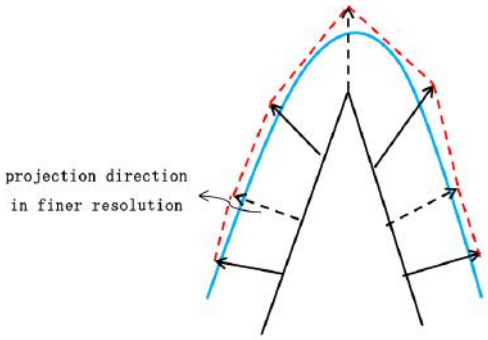
In the implementation, it is necessary to mention that there is truncation error from double to float, which will cause program to run infinitely long. It can be caused in two aspects: a). The updated solution can be truncated to be the same as the former solution. b). If we set the tolerance too small, the variable x will be oscillating in a narrow range, but the trend is converged. Maximal number of the inner and outer iterations is set to avoid infinite iteration.

3.5.7 Extreme Cases

It is notable that there is an extreme case in the implementation. Normally the displacement lengths in the displacement maps are in $[-2, 2]$, but during the optimization process, an extreme case showed by Figure 3.14 should be considered. If we set an initial variable bounding range, like $[-100, 100]$, the optimization process will keep running infinitely in this extreme case, since the optimized lengths will always be larger than 100. It is safe to set an initial bounding range as default, i.e. $[-Inf, Inf]$. After an optimal solution is obtained, we need to check if there exist quite large displacement lengths. If such extreme large lengths exist, they are eliminated, and finer bounding displacement lengths are used instead.



(a) Extreme case makes bounding displacement lengths quite large.



(b) Bounding surfaces in a finer resolution.

Figure 3.14: The bounding displacement lengths will be large and the bounding surfaces do not bound the projected surface well. The finer bounding surface in this area is used instead.

3.6 Multiresolution Collision Detection

The potential primitives whose forwards projected surfaces may collide with the objects are selected in the broad phase. After the early out filtrating part, the bounding volumes which bound the forwards projected surface of one primitive are created using the multiresolution bounding maps. One intention to use this multiresolution collision detection is that the current method used will test all vertices in a dynamic object (M points) with the potential primitives (N primitives) in real-time, and the complexity is $\mathcal{O}(MN)$. The multiresolution collision detection will reduce the vertices to be tested.

3.6.1 Intersection Between a Rectangle and a Triangle

Here comes the question to find the coarsest bounding volumes for one specific primitive. The idea is to find the coarsest bounding rectangles in the displacement maps which have intersection with the triangle. Because the bounding volumes

always bound the projected surface, it is sufficient to select the coarsest bounding rectangles in 2D.

Figure 3.15 shows five different spatial relationships between a rectangle and a triangle. Combined with Table 3.1, a method is derived to determine whether the rectangle intersect with the triangle or not.

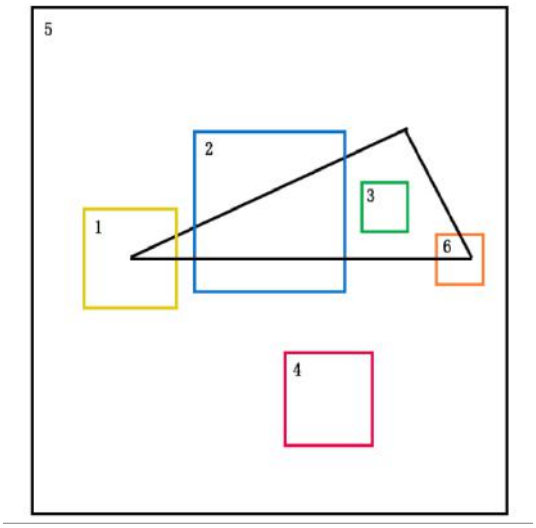


Figure 3.15: Spatial relationships between a rectangle and a triangle.

Rectangle Type	Intersection	Tri Vertex inside Rect	Rect vertex inside Tri	Edge Intersect
1	✓	✓	✗	✓
2	✓	✗	✗	✓
3	✓	✗	✓	✗
4	✗	✗	✗	✗
5	✓	✓	✗	✗
6	✓	✓	✓	✓

Table 3.1: Evaluation of intersections from spatial relationships in edges and vertices

SAT (Separating Axis Theorem) can be used to determine if the triangle and the rectangle is intersected or not. However, it is much faster to use Algorithm 9 since most intersection cases will be determined by checking spatial relationship of vertices, before edge intersection detection. The line segmentation method is specialized of the 3D line intersection algorithm from [35].

Algorithm 9 Intersection between a rectangle and a triangle:

```
function IntersectedTriRect(triangle, rectangle)
    if one vertex of the rectangle is inside the triangle then
        return true
    end if
    if one vertex of the triangle is inside the rectangle then
        return true
    end if
    if the edge of the triangle has intersection with the edge of the rectangle then
        return true
    end if
    return false
end function
```

3.6.2 Coarsest Level Collision Detection

After the filtration is done in the early out part, a list of potential primitives will be obtained. For each potential primitive, the coarsest bounding volumes are derived by the method in Section 3.6.1. It is notable that the collision detection algorithm in this level handles collision between a complex object (or a compound) and a convex bounding volume. The idea is to loop each sub convex object included in this compound and use a convex versus convex collision detection method in each convex pair. If the collision is detected, the sub convex object is kept in a global map. The intention of the global map is to avoid repeated collision detections of one sub convex shape if this convex has been detected collision in the former detections. The algorithm is shown in Algorithm 10.

3.6.3 One Level Down Collision Detection

This level of detection will be performed if there is a collision detection in the former coarser level. As Algorithm 11 suggests, the collision detection algorithm in this level is a convex versus convex detection, since the object pair passed down by former level is a convex pair. This detection is recursive, terminated in no collision detection or in the finest level.

3.6.4 Finest Level Collision Detection

It is notable that in the finest level, the bounding volumes are constructed from the displacement maps, but not from the bounding displacement maps like in the former levels. In this level, a bounding volume is a tetrahedron which bounds a bilinear patch. The displacement lengths to construct the bounding volumes are the same as the lengths in the displacement maps. The global map is updated as shown in Algorithm 12.

Algorithm 10 The coarsest level collision detection:

```
function coarsestCollision(query object, bounding displacement maps)
    get the AABB (2D) of the primitive triangle in the displacement maps
    test sub bounding rectangles inside this bounding rectangle
    Intersection between a rectangle and a triangle
    construct a bounding volume using a rectangle which collides with the triangle
    if the bounding volume has collision with the query object then
        if the query object is a compound then
            get sub convex objects
        else
            get this (convex) object
        end if
        for each sub convex object do
            if this object is not in the global map then
                add to the global map
            else
                skip this object, continue
            end if
            quad_resolution-1
            one level down collision detection
        end for
    end if
end function
```

3.6.5 Overall Multiresolution Collision Detection

The overall multiresolution collision detection is shown in Algorithm 13. If the resolution of the test quad is 0 at the beginning, which means the quad is a 1×1 grid, the finest collision detection will be performed. After the potential sub-objects are determined in the multiresolution collision detection, the collision detection is performed between the sub-object and the projected surface as Section 2.4.2 suggests.

Algorithm 11 One level down collision detection:

```
function oneLevelDown(convex object, bounding displacement maps)
    if quad_resolution ≠ 0 then
        test four sub rectangles
        Intersection between a rectangle and a triangle
        construct a bounding volume using a rectangle which collides with the
triangle
        construct four finer bounding volumes
        for each finer bounding volume do
            convex vs convex (GJK) collision detection
            if collision detected then
                one level down collision detection
            else
                remove this object from the global map
                no need for finer detection
            end if
        end for
    else
        finest level collision detection
    end if
end function
```

Algorithm 12 Finest level collision detection:

```
function finestCollision(convex object, displacement maps)
    test four sub rectangles
    Intersection between a rectangle and a triangle
    construct four bounding tetrahedron with the displacement maps
    for each bounding tetrahedron do
        if collision detected then
            keep this object in the global map
        else
            remove this object in the global map
            no collision
        end if
    end for
end function
```

Algorithm 13 Multiresolution Collision Detection:

```
function MultiCollision(query object, displacement maps, bounding displacement maps)
    construct a global map to store query objects
    early out
    if no collision detection then
        return
    end if
    for each potential primitive detected in early out collision do
        if quad_resolution ≠ 0 then
            coarsest level detection
        else
            finest level collision detection
        end if
    end for
end function
```

3.7 Summary

In this chapter, the implementation of this thesis is illustrated. Two main problems are solved in this chapter: new backward projections and the multiresolution collision detection. The ideas and formulas of deriving the new backward projections are stated in Section 3.2. The results of improvement in accuracy will be shown in Chapter 4. The implementation of the collision detection is illustrated in the following order: a) generation of the upper and lower bounding surfaces by solving inequality-constrained nonlinear programming problems. b) construct the bounding volumes from these bounding surfaces. c) perform collision detection between query objects and these bounding volumes. d) determine collision with the terrain using the backward projection for the relevant potential interactions. The procedure (c) is specified in Section 3.6, from the coarsest level collision detection to the finest one. The results of improvement in collision detection efficiency will be shown in Chapter 4.

Chapter 4

Result

4.1 Overview

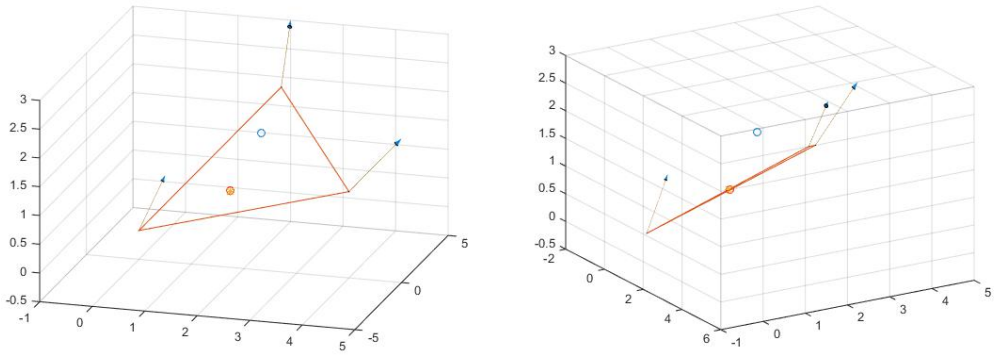
In this chapter, results of the improvement in collision detection will be shown. The results of new backward projections will be shown in Section 4.2 in two aspects, one in test examples and the other in the game. The test examples will show the results clearly under a relatively pure test condition, and helps the reader to understand the mathematical theories stated in Chapter 3. While more visual results will be shown in the game, and the improvement can be clearly seen when playing the game. Section 4.3 will also show the results both in test examples and in the game, helping to understand the implementation both in theory and in practice. The highly improved collision detection will be shown in Section 4.4 by comparing with the old methods used. The comparison diagrams help to illustrate the results more clearly.

4.2 Accuracy of Backwards projections

4.2.1 Common Case

As it was discussed in Subsection 2.4.1, the aligned plane method works well for most cases. An ideal case is presented as follows: $\mathbf{P}_0 = (0, 0, 0)$, $\mathbf{P}_1 = (4, 0, 1)$, $\mathbf{P}_2 = (2, 4, 2)$. $\mathbf{N}_0 = (0.5, 0, 1)$, $\mathbf{N}_1 = (1, 0, 1)$, $\mathbf{N}_2 = (0, 1, 1)$. The query point is set by parameters $u = 0.3$ and $v = 0.2$.

Figure 4.1 shows the aligned plane method in the common case. The query point (blue circle) has a backwards projected point (red star), which almost in the same position as the exact one (red circle). Figure 4.2 and Figure 4.3 show the aligned projection direction method and alpha plane method in this common case. All these methods have good approach to the exact point (shown in Table 4.1). It is notable that the two iterations in the aligned projection method refers to the iterations of the BFGS (first one) and the iterations of the line search (second one).



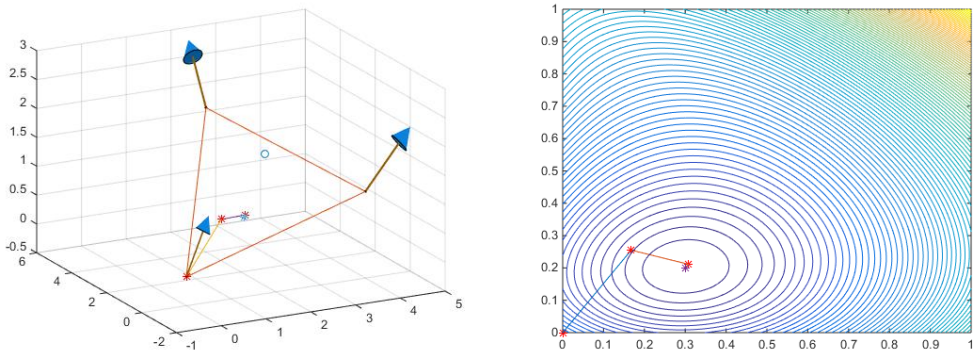
(a) Top view.

(b) Side view.

Figure 4.1: Backward Projection using aligned plane (common case).

	u	v	iteration	f value
Aligned Plane	0.3055	0.1901	N/A	N/A
Aligned Projection	0.3008	0.1999	3+6	1.1178e-05
Alpha Plane	0.3000	0.2000	3	-6.5162e-07

Table 4.1: Comparison between three methods.



(a) Backward projection. (b) Contour of backward projection.

Figure 4.2: Backward projection using aligned projection direction (common case).

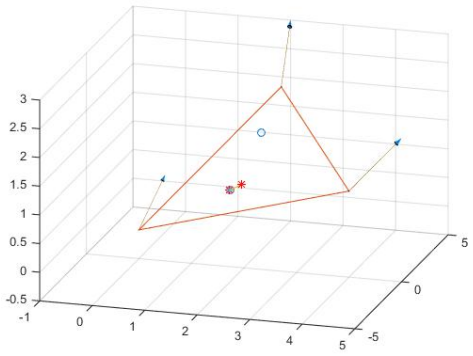


Figure 4.3: Backward projection using alpha plane method (common case).

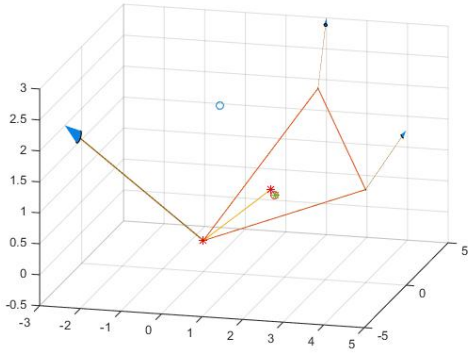
4.2.2 Bad Case

Case I

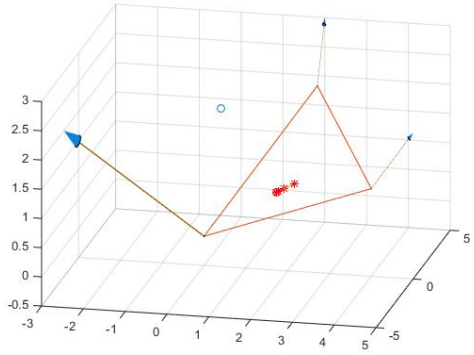
In this case, $\mathbf{N}_0$ is changed to $\mathbf{N}_0 = (-3, -2, 2)$, to have a relatively large length and skew direction. Figure 4.4 shows the two new methods work well for this bad case, but the artifact can be seen using aligned plane method. Details are shown in Table 4.2.

	u	v	iteration	f value
Aligned Plane	0.2067	0.1112	N/A	N/A
Aligned Projection	0.3005	0.2004	2+4	1.3704e-04
Alpha Plane	0.3000	0.2000	4	-4.3434e-05

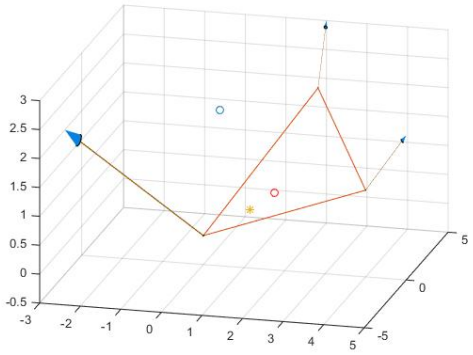
Table 4.2: Comparison between three methods.



(a) Aligned projection direction method.



(b) Alpha plane method.



(c) Aligned plane method.

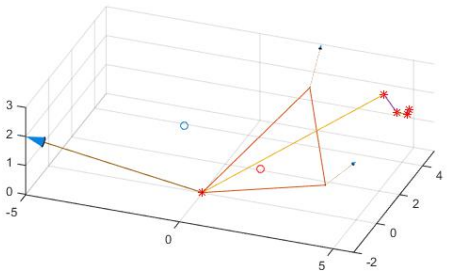
Figure 4.4: Aligned projection direction method and alpha plane method work well, but not the aligned plane method.

Case II

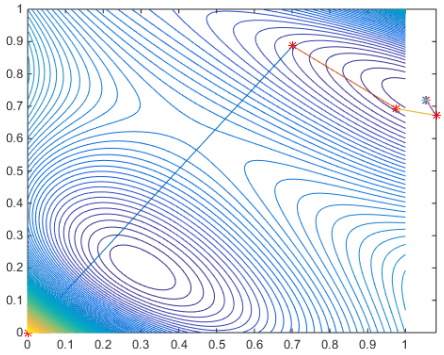
In this case, $\mathbf{N}_0$ is changed to $\mathbf{N}_0 = (-5, -2, 2)$, to have a larger length and more skew direction. Figure 4.5 shows the alpha plane method still works well for this bad case, but the aligned projection direction method cannot find the right minimal value. The objective function has several local minimal values and the BFGS iteration method is an unconstrained iteration method and is not guaranteed to find the minimal inside the triangle. The details are shown in Table 4.3.

	u	v	iteration	f value
Aligned Plane	0.1792	0.1079	N/A	N/A
Aligned Projection	1.0516	0.7196	5+11	8.2312e-06
Alpha Plane	0.3000	0.2000	4	-8.0856e-04

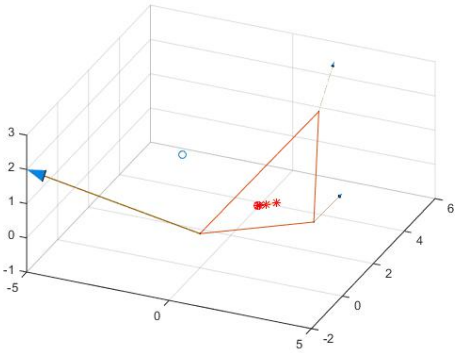
Table 4.3: Comparison between three methods.



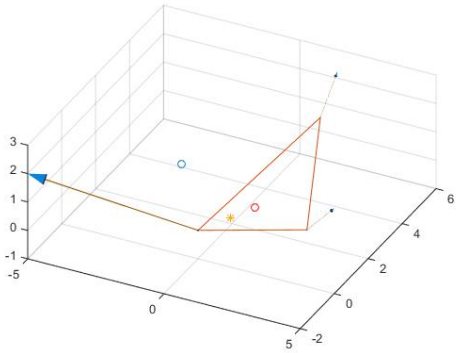
(a) Aligned projection direction method



(b) Contour of aligned projection direction iterations.



(c) Alpha plane method.



(d) Aligned plane method.

Figure 4.5: The alpha plane method still works well, but not the aligned plane method and the aligned projection direction method.

4.2.3 Bad Case in the Game

Figure 4.6 shows the bad case in the game. Backward projection is important in the final collision detection with the terrain surface. The wrongly backwards projected point will lead to wrong detection and distance computation, which make the character sink down into the terrain. Figure 4.7 shows the bad case is fixed using two new iteration methods. In the game test, no bad case using aligned projection direction method has been detected yet.

Figure 4.6: There will be artifacts if the aligned plane method is used.

(a) Alpha Plane Method.

(b) Aligned Projection Direction Method.

Figure 4.7: Two different methods both can handle this problem [19].

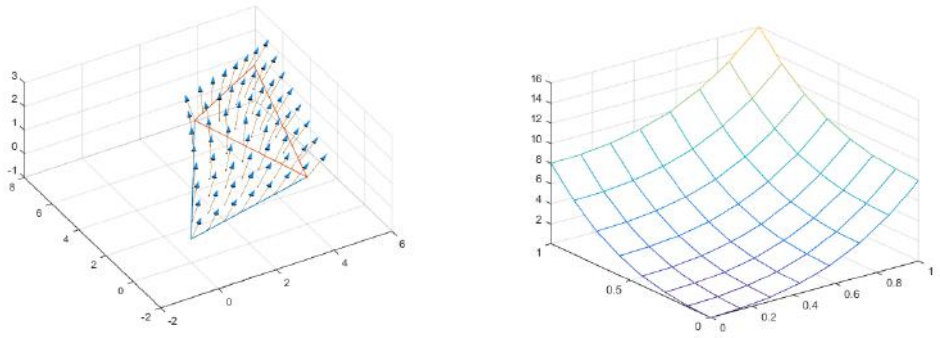
4.3 Multiresolution Bounding Volume Generation

4.3.1 Test Example

One example of generating upper bounding triangles is given here.

Example 4.3.1 Given a quad with two primitive triangles inside, and it has 9×9 constrained points. The number of bounding points is assumed to be four (i.e. the bounding size is eight), for simple test. These four bounding points are mainly four vertices of this quad.

Four quad vertices are $\mathbf{p}_0 = (0, 0, 0)$, $\mathbf{p}_1 = (4, 0, 1)$, $\mathbf{p}_2 = (2, 4, 2)$, $\mathbf{p}_3 = (5, 6, 2)$. And the projection directions of these vertices are $\mathbf{r}_0 = (0.5, -0.5, 2)$, $\mathbf{r}_1 = (1, 0, 1)$, $\mathbf{r}_2 = (0, 1, 1)$, $\mathbf{r}_3 = (2, 1, 2)$ respectively. As Figure 4.8 suggests, the interpolation is performed to obtain the positions and the projection directions of the inner constrained points. Displacements are sampled from surface $f(x, y) = x^3 + y^3$. Figure 4.9 shows the projected surface. It is notable that this surface is not the exact projected surface, but an approximation as mentioned in the former sections.



(a) The projection directions on the constrained points and the bounding points.

(b) Surface $f(x, y) = x^3 + y^3$.

Figure 4.8: The base primitives and the sampling displacement map.

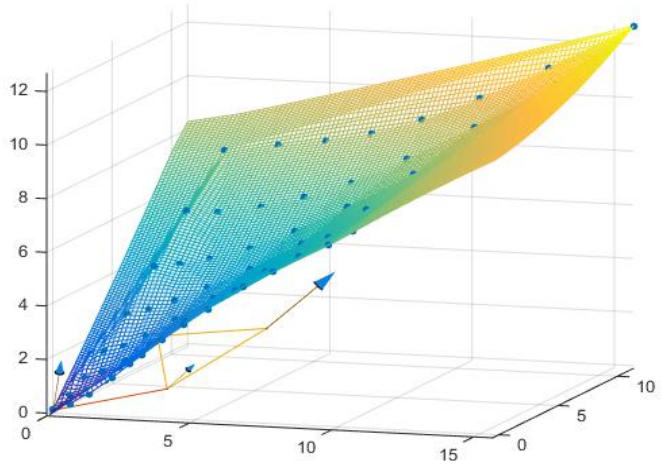
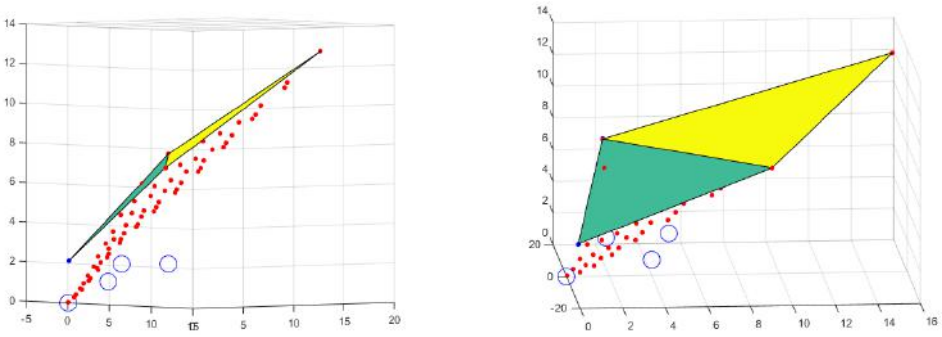


Figure 4.9: Approximation of the Projected Surface.

The optimization algorithm is performed as Section 3.5.4 suggests. In Figure 4.10, red dots represent the projected points on the surface and the blue circles represents the four bounding points. It is clearly seen that two triangles bound these projected points.

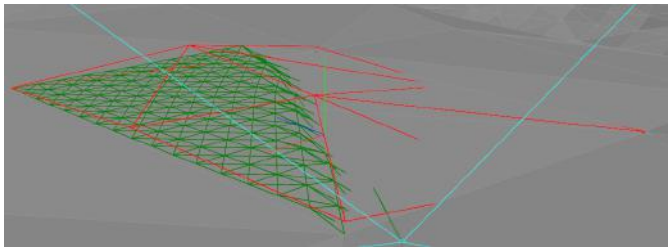


(a) Side view. (b) Top view.

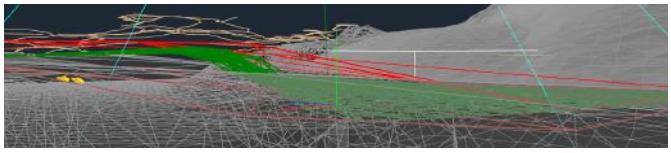
Figure 4.10: Upper bounding triangle surfaces.

4.3.2 Bounding Volumes in the Game

From Figure 4.11 to Figure 4.13, the bounding volumes in different resolutions are shown. One quad is selected to be visualized. The green tetrahedrons represent the bounding volumes in the finest resolution, which bound the forwards projected surface. In the finest resolution, one projected surface is a bilinear patch and can be bounded by a tetrahedron, which is formed by four bounding triangles. For the purpose of clear representation, only the upper bounding and lower bounding surfaces (red surface triangles) are shown. As the resolution goes from coarser to finer, the bounding volumes bound the projected surface more and more tightly.

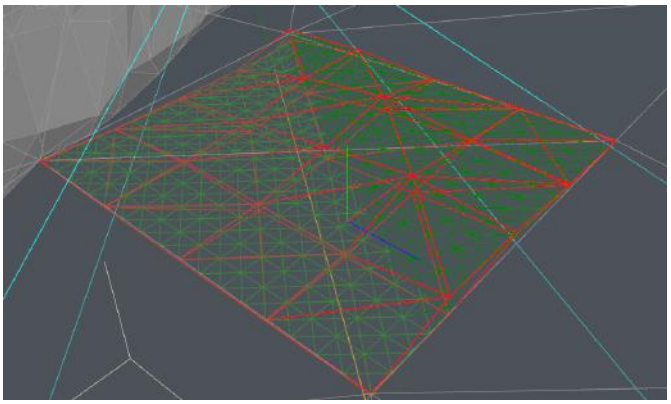


(a) Top view.

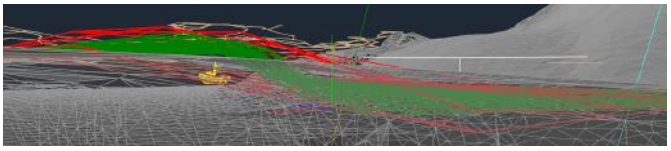


(b) Side view.

Figure 4.11: Upper and lower bounding surfaces with size 8 [19].

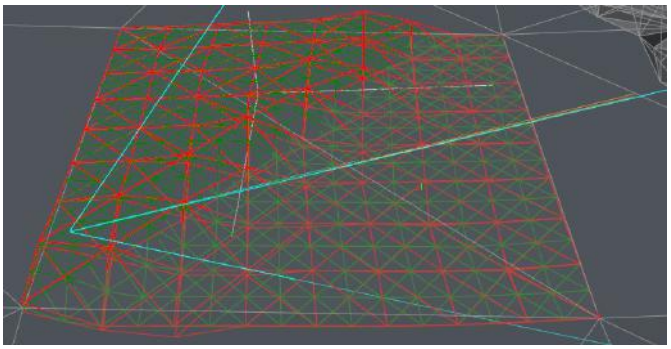


(a) Top view.

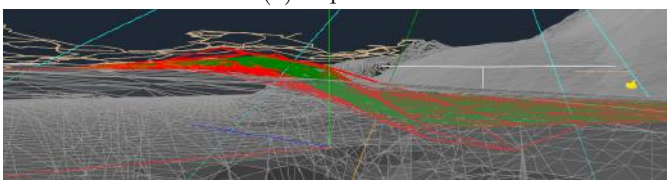


(b) Side view.

Figure 4.12: Upper and lower bounding surfaces with size 4 [19].



(a) Top view.



(b) Side view.

Figure 4.13: Upper and lower bounding surfaces with size 2 [19].

4.4 Multiresolution Collision Detection in the Game

The character is teleported to a remote place without too many dynamic objects for test. The original collision detection method and the multiresolution collision detection method are both enabled at the same time to ensure the comparison is made under exactly the same conditions. As shown in Table 4.4, the multiresolution method highly reduces the number of backward projections in the final step. The number of backward projections is 15, while the number in the original collision detection is 97. The extra time added in the multireolution method is the bounding volume collision detection part. The total time comparison shows the time of multiresoltion collision detection is nearly half of the origin detection time.

	original collision detection	multiresolution collision detection			
		coarsest level	finer level	finest level	collision with the surface
number of calls	97	40	24	12	15
time (ms)	2.103	0.812	0.018	0.007	0.24
		1.077			

Table 4.4: Comparison between the two methods in one certain frame.

Time comparison between these two methods is shown in Figure 4.14 in a time slot with 363 frames. The x axis represents the frame, and the y axis represents the consumed time. It is notable that the time range is $0 - 8(ms)$, and it has a multiplier, which means the time shown in the figure is obtained by multiplying the actual time by the multiplier. There are several dynamic objects in this scenario, like NPC riding on motorcycles and cruise ships near the shore, which cause the noisy shape of the lines. However, it clearly shows that the multiresolution collision detection method is much faster.

An exception happens in the beginning where the yellow line is close to the red one, which means the number of vertices to be tested with the terrain surface is almost the same with ones in the original method. In this case, the multiresolution method is slower with additional time in multiresolution detection. But it is a rare case which there are only small objects detected in current frame, i.e. all dynamic objects to be tested are small and simple. This problem can be fixed if the method can be performed adaptively with the size of objects as discussed in Section 5.2. It is also notable that the program is running on four threads, Figure 4.14 shows the time on the first thread, which cause 0 consumed time in some frames. The collision detections in these 0 time frames are performed in the rest threads as shown in Figure A.1.

The time in the finer bounding volume collision detection is negligible compared with the coarsest bounding volume detection and the surface collision detection. It is still necessary to analysis this time. Detection time in different finer resolutions (resolution=0,1,2) is shown in Figure A.2. The maximal resolution generated in this

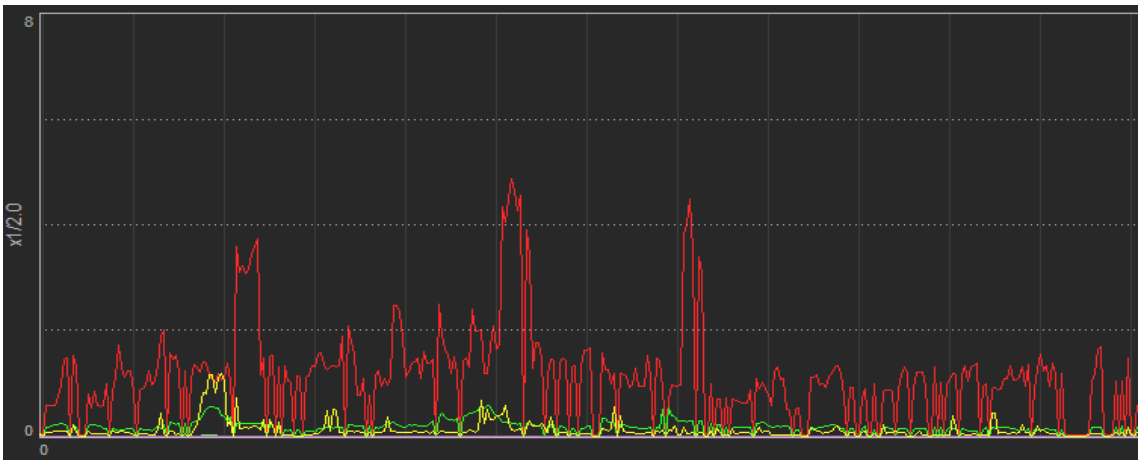


Figure 4.14: Time comparison between two methods. Original collision detection (red), Multiresolution collision detection: coarsest bounding volume detection (green), finer bounding volume detection (violet, close to 0), final surface detection (yellow).

project is 3, i.e. the coarsest bounding size is 8 (2^3). There is no clear relationship between the consumed time on different finer resolutions (any resolution detection can be the most costly in different frames). Given a list of the coarsest bounding volumes for example, each one has 4 finer solution bounding volumes till the finest. If the finest bounding volume colliding with the query object is contained in the first coarsest bounding volume, the time for total finest detection will be larger. However, if the finest colliding bounding volume is contained in the coarsest volume which locates in the end, the time for total coarser resolution detection will be larger.

	original collision detection	multiresolution collision detection			
		coarsest level	finer level	finest level	collision with the surface
number of calls	869	62	18	9	11
time (ms)	9.742	0.236	0.028	0.006	0.472
		0.742			

Table 4.5: Comparison between the two methods in one certain frame when the warship is put in the area with high density of primitives.

To show comparison without much noise, irrelevant complex objects, like the ships near the shore, are moved away from the terrain. Complex objects, like warships, are spawned to test the consumed time. Figure A.4 and Figure A.5 show the consumed time when spawn one and two warships separately. A warship in the game is a huge and complex object containing fort batteries, cabins, conning

towers, etc (as shown in Figure A.3). It is quite costly to test every sub-objects. The multiresolution collision detection filtrates those which are actual colliding and highly reduce the number of collision detections with the terrain surface. When the warship is moved to some place with high density of primitives, the comparison is more obvious as shown in Figure A.6. One randomly selected frame in this scenario is shown in Table 4.5. In this frame, the time using multiresolution detection is 1/13 of the one using the old method.

It is notable that the time of collision detection does not only depend on the number of function calls, but also depend on the scenario, which can be seen from Table 4.4 and Table 4.5. Take the coarsest level collision detection for example, the number of function calls in Table 4.4 is fewer than the number in Table 4.5, but it takes more time. Because the detection places, the sizes of query objects and the base primitives are different in these two scenarios. The crosswise comparison is used to show the trend, and the vertical comparison (it is performed in the same frame and the same scenario, to ensure the comparison is under exact the same conditions) is used to perform a quantitative analysis.

4.5 Summary

In this chapter, the results of new backward projections, multiresolution bounding volumes and the multiresolution collision detection are shown. Each result contains two parts: the test (theoretical) example and the result in the game. The comparison between different backward projections is shown in Section 4.2. Theoretical bad cases are tested to show the reasons of artifacts in the game, and the improvement in the game gives a direct impression to the reader. In Section 4.3, bounding volumes in different resolutions are shown in selected area of the terrain. The comparison diagrams in Section 4.4 shows the significant improvement in accelerating the speed of the collision detection using the multiresolution method.

Chapter 5

Discussion

5.1 Conclusion

The new forms of the backward projections and the multiresolution collision detection method are presented for the first time in this paper. Problems stated in Section 1.2 are solved. This work is done within Avalanche game engine. The comparison will not be performed with other game engines, like Unity 3D or Unreal Engine, since the terrain system is not generated by projection in these engines. And these engines do not support collision detection between dynamic objects and the displacement mapped terrain.

In this project, the collision detection performance is improved in several ways:

- More accurate backward projection methods: aligned projection direction method and the alpha plane method are derived and compared with the old method. Both these new methods perform well in the bad cases where the old backward projection can result in artifacts. However, the aligned projection direction method has errors as shown in test examples in Section 4.2. This test example may not appear in the game, but it is safe to use the alpha plane method. It also has fewer iterations and a higher accuracy. This will ensure the smooth running quality and better reality of the game. The artifacts of the character sinking into the terrain will not happen.

- A better bounding plane is integrated in the early out part, which helps filtrate potential colliding primitives and objects and helps to reduce the objects to be tested in the narrow phase. Multiresolution bounding surfaces are generated by solving inequality-constrained nonlinear programming problems with consideration of extreme cases. With these upper and lower bounding surfaces, multiresolution convex bounding volumes are constructed. Convex versus convex collision detection is performed between the bounding volume and the query object. Collision detection methods differ in resolutions. In the coarsest level, the detection between a compound (consisting in many convex objects, and probably not a convex object) and a bounding volume. In

the finest level, the bounding volume is a tetrahedron obtained from displacement maps, unlike former bounding volumes which are derived from bounding maps. If the collision is detected in the coarsest level, the detection will be performed in one level finer resolution, which are related with finer bounding volumes. Until the collision is detected in the finest level, the sub-objects of the query object are chosen to perform collision detection with the terrain surface. In this stage, the vertices to be tested in the sub-objects are much fewer than ones in the compound.

- The multiresolution bounding volume collision detection highly reduces the number of objects to be tested with the terrain surface. Compared with the old collision detection method which tests every vertex in the query object, the multiresolution detection reduces the number of vertices to be tested, i.e. the number of backward projections. The extra time added in the multiresolution detection is worth comparing with the old method, since it saves much more time in the collision detection with the terrain surface. The efficiency and performance of collision detection is improved in this way. The implication of saving time here means more complex objects can be performed with detection in the same scenario, and the game can still run smoothly. The player can enjoy much more extraordinary quality in grand battles.

5.1.1 Limitation

The limitation of the work lies in the multiresolution detection part. If the query objects are small and simple, it is necessary to test all the vertices of this object when detected with the terrain surface. Multiresolution collision detection here can not reduce the number of backward projections in simple objects (shown in Figure 4.14, where, in some frames, the yellow line is close to the red one). An appropriate threshold should be set by testing to determine the complexity of the query object. If it is regarded as a simple one, the original method, which tests all the vertices, should be used. Otherwise, the multiresolution detection is chosen.

5.2 Future Work

There is much more work to do as extensions or improvements of this thesis work.

- Global optimization of the base mesh. As mentioned in the backward projection, using the aligned plane method causes problems in some cases. However, if the projection directions on the base mesh can be globally optimized to de-

crease the error using the aligned plane method, the original method can be used, since it has no iteration.

- Quadrilateral mesh can be used instead of the triangle base mesh. Quads will ensure the mesh has clean topology and can be deformed properly, and the problem of finding interpolated points on the boundary will not exist. Triangle mesh can cause blemishes or pinch the geometry at render time, and artifacts appear when texturing and especially when deforming for animation.

- As mentioned in Subsection 2.7.1, if a highly efficient data structure can be constructed for the bounding volumes, the approximate decomposition method is worth trying in the generation of convex bounding volumes of the static terrain. The optimization method used to regenerate the bounding displacement maps is slow, even though it is running off-line.

- Efficient caching techniques can be used in collision detection. It can be seen that if the character collides with the terrain in the former frame, it may still collide in the next frame. For the multiresolution detection in this paper, the collision detection will be performed in every frame, from the coarsest to the finest. If these objects can be cached, it will save time for collision detection.

- Generation of adaptive bounding volumes according to the terrain geometry. As shown in Subsection 4.3.2, the coarse bounding volumes are close to the fine ones if the geometry is flat, like a road or a playground. In this situation, only one or two levels of resolution need to be tested.

- Adaptation of the collision detection according to the size of the query objects. The current method tests parts of sub-objects in the query object to perform collision detection. It is efficient if the query object is large and complex. It is probably worth determining an appropriate threshold. If the size of the query object is below this threshold, the original collision detection method is used. If it is above, the multiresolution collision detection method is used.

Appendix A

Time on different threads

Figure A.1: The time comparison on four threads. Original collision detection (red), Multiresolution collision detection: coarsest bounding volume detection (green), finer bounding volume detection (violet, close to 0), final surface detection (yellow).

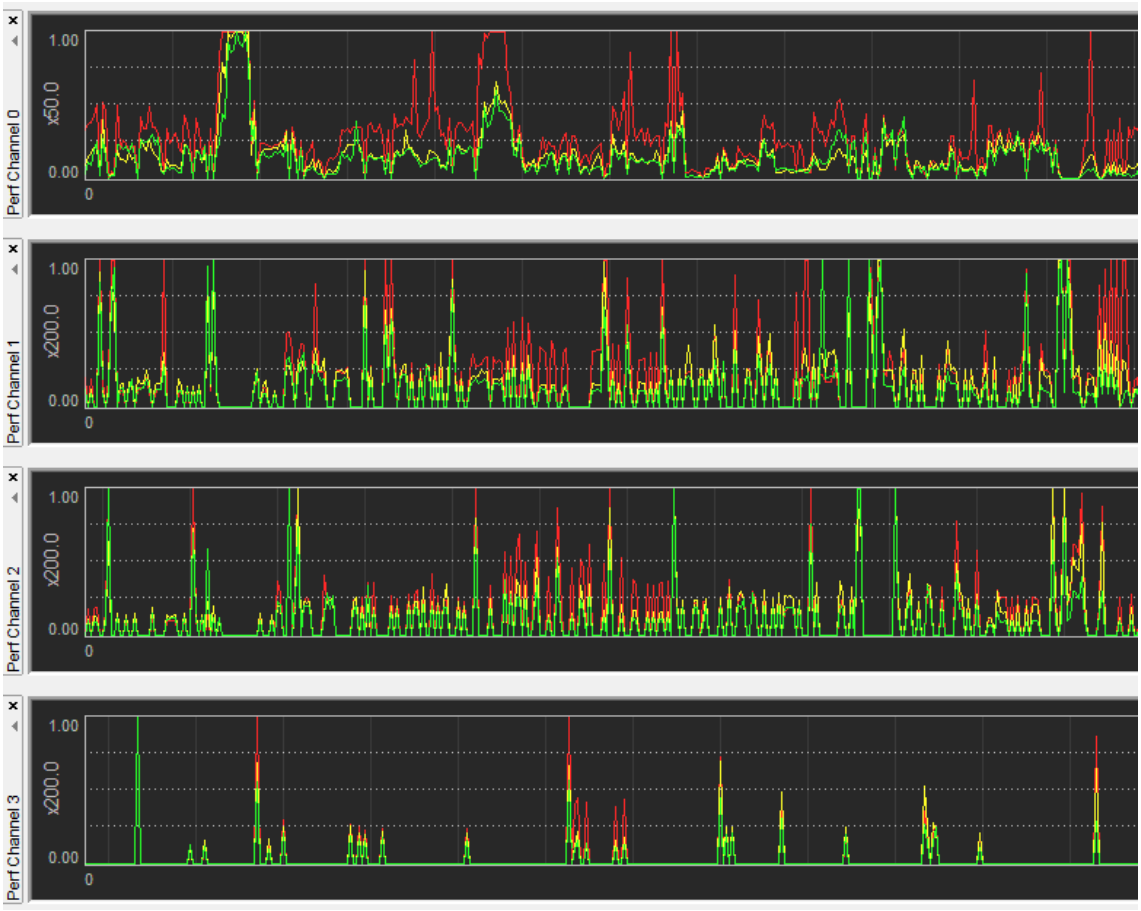


Figure A.2: The time comparison on four threads. Consumed time on finer bounding volume collision detection. Resolution = 2 (red), Resolution = 1 (yellow), Resolution = 0 (green).

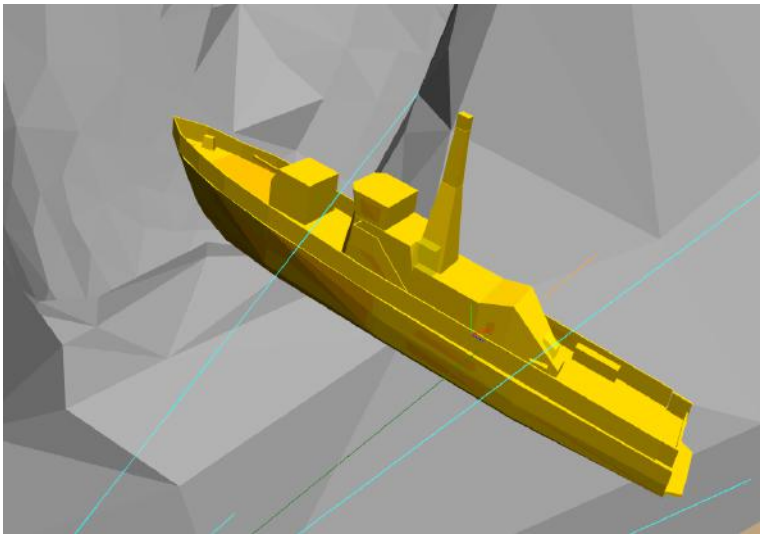


Figure A.3: A warship [19] contains many sub-objects. The multiresolution collision detection method will only detect the collision between the bottom of the warship and the terrain surface. It is notable that the gray mesh in this figure is not the terrain surface, but the projected primitives.

Figure A.4: The time comparison on four threads when spawn one warship. Original collision detection (red), Multiresolution collision detection: coarsest bounding volume detection (green), finer bounding volume detection (violet, close to 0), final surface detection (yellow).

Figure A.5: The time comparison on four threads when spawn two warships. Original collision detection (red), Multiresolution collision detection: coarsest bounding volume detection (green), finer bounding volume detection (violet, close to 0), final surface detection (yellow).

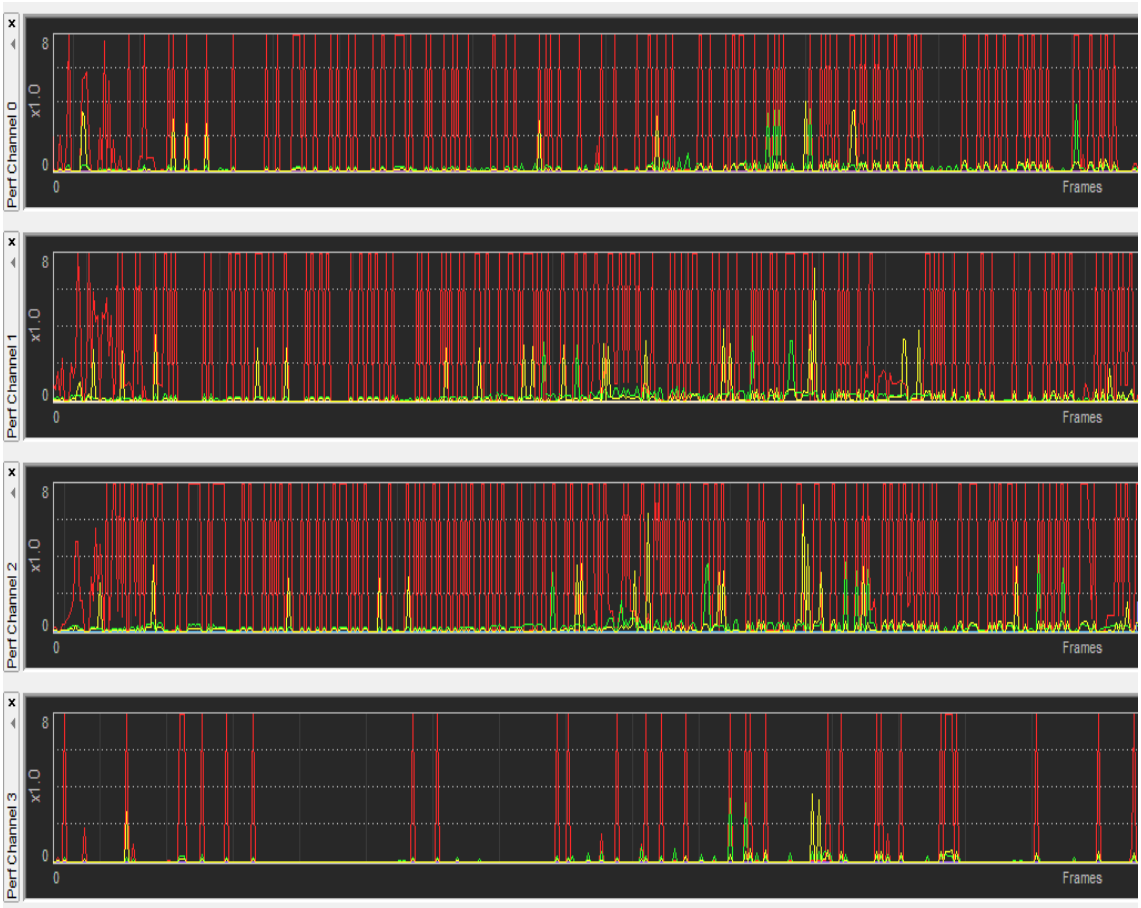


Figure A.6: The time comparison on four threads when the warship is moved to some place with high density of primitives. Original collision detection (red), Multiresolution collision detection: coarsest bounding volume detection (green), finer bounding volume detection (violet, close to 0), final surface detection (yellow).

Bibliography

[1] Stefan Gumhold, Tobias Huttner. *Multiresolution Rendering With Displacement Mapping.* ACM, 1999.

[2] Wright, J. R.. *A Voxel-Based, Forward Projection Algorithm for Rendering Surface and Volumetric Data.* Visualization '92, IEEE Conference, 1992.

[3] Christer Ericson.*Real-Time Collision Detection.* Morgan Kaufmann Publishers, 2005.

[4] D. E. Stewart, J. C. Trinkle. *An Implict Time-Stepping Scheme for Rigid Body Dynamics with Inelastic Collisions and Coulomb Friction.* International Journal for Numerical Methods in Engineering, p. 2673-2691, 1996.

[5] A. Tasora, M. Anitescu. *A Matrix-Free Cone Complementarity approach for Solving Large-Scale, Nonsmooth, Rigid Body Dynamics.* Computer Methods in Applied Mechanics and Engineering, p. 439-453, 2011.

[6] Anitescu M.. *A Fixed Time-Step Approach for Multibody Dynamics with Contact and Friction.* Mathematics and Computer Science, 2003.

[7] Daniele Panozzo, Ilya Baran, Olga Diamanti. *Weighted Averages on Surfaces.* ACM SIGGRAPH, 2013.

[8] M. Nießner, C. Siegl, H. Schafer. *Real-time Collision Detection for Dynamic Hardware Tessellated Objects.* EUROGRAPHICS, 2013.

[9] Katsuaki Kawachi, Hiromasa Suzuki. *Distance Computation between Nonconvex Polyhedra at Short Range Based on Discrete Voronoi Regions.* Geometric Modeling and Processing 2000, Theory and Applications, 2000.

[10] W. E. Lorensen, H. E. Cline. *Marching Cubes: A High Resolution 3D surface Construction Algorithm.* Computer Graphics, vol. 21, no. 4, p. 163-169, 1987.

[11] Jing Jin. *An Improved Marching Cubes Method for Surface Reconstruction of Volume Data.* Intelligent Control and Automation, 2006.

[12] Tamal K. Dey, Joshua A. Levine. *Delaunay Meshing of Isosurfaces.* Proc. Shape Modeling International, 2007.

[13] Yarden Livnat, Han-Wei Shen. *A Near Optimal Isosurface Extraction Algorithm Using the Span Space*. IEEE, 1996.

[14] Christopher Batty. *Reconstructing Meshes with Sharp Features from Sgined Distance Data*. `https://cs.uwaterloo.ca/~c2batty/misc/levelset_meshing/level_set_reconstruction.html`.

[15] Digital tutors. `http://blog.digitaltutors.com/bump-normal-and-displacement-maps/`.

[16] Normal mapping. `http://en.wikipedia.org/wiki/Normal_mapping`.

[17] Wikimedia Commons. `http://commons.wikimedia.org/wiki/File:Bump_map_vs_isosurface3ver2.png`.

[18] Brian Smits, Peter Shirley, Michael M. Stark. *Direct Ray Tracing of Displacement Mapped Triangles*. ACM Proceeding, 2000.

[19] The citation is under the permission of Avalanche Studios.

[20] Baraff D.. *Dynamic Simulation of Non-Penetrating Rigid Bodies*. Computer Science Department, Cornell University, pp52-56, 1992.

[21] Scott Le Grand. *Broad-Phase Collision Detection with CUDA*. NVIDIA Corporation, 2007.

[22] Jyh Ming Lien. *Approximate Convex Decomposition*. Technical Report TR03-001, Texas A&M University, 2003.

[23] Khaled Mamou, Faouzi Ghorbel. *A Simple and Efficient Approach for 3D Mesh Approximate Convex Decomposition*. Image Processing, 16th IEEE international Conference, 2009.

[24] Patrick G. X.. *Fast Swept-Volume Distance for Robust Collision Detection*. International Conference on Robotics and Automation, Proceeding of IEEE, 1997.

[25] Brian Mirtich. *V-Clip: Fast and Robust Polyhedral Collision Detection*. ACM Transactions on Graphics, 1997.

[26] Kelvin Chung, Wenping Wang. *Quick Collision Detection of Polytopes in Virtual Environments*. In Proceedings of the ACM Symposium on Virtual Reality Software and Technology, p. 125-131, 1996.

[27] E.G. Gilbert, D.W. Johnson, and S.S. Keerthi. *A Fast Procedure for Computing the Distance between Complex Objects in Three-Dimensional Space*. IEEE Journal of Robotics and Automation, 4(2):193-203, 1988.

[28] Christer Ericson. *Real-Time Collision Detection*. The Morgan Kaufmann Series in Interactive 3D Technology. Morgan Kaufmann Publishers, 2005.

[29] Java Collision Detection and Physics Engine. `http://www.dyn4j.org/2010/04/gjk-distance-closest-points/`.

[30] Avriel, Mordecal. *Nonlinear Programming: Analysis and Methods*. Dover Publishing, ISBN 0-486-43227-0, 2003.

[31] Robert C. Jagnow. *Virtual Scripting with Haptic Displacement maps*. master thesis paper at MIT, 2001.

[32] Krister Svanberg. *A Class of Globally Convergent Optimization Methods Based on Conservative Convex Separable Approximations*. SIAM J. Optim. 12(2), p. 555-573, 2002p.

[33] Krister Svanberg. *The Method of Moving Asymptotes - A New Method for Structural Optimization*. Internat. J. Numer. Methods Engrg., 24 (1987), pp. 359-373.

[34] Nonlinear Optimization library. `http://ab-initio.mit.edu/wiki/index.php/NLopt`.

[35] Ronald Goldman. *Intersection of Two Lines in Three-Space*. Graphics Gems, Academic Press Professional, p. 304, 1990.

TRITA -MAT-E 2015:50
ISRN -KTH/MAT/E--15/50--SE