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

Author: Fangkai Yang, KTH
Mentor: Joacim Jonsson, Avalanche Studios
Supervisor: Prof. Christopher Peters, KTH

26 June, 2015

Overview

Purpose: Improve the performance and robustness of collision detection in Avalanche game engine.

Contribution:

- Fix artifacts in some cases by creating new backward projection methods.
- Highly reduce the time of collision detection between dynamic objects and the terrain by creating multiresolution method.

Outline

Outline

Collision Detection

Definition

Collision detection refers the detection of the intersection of two or more objects.

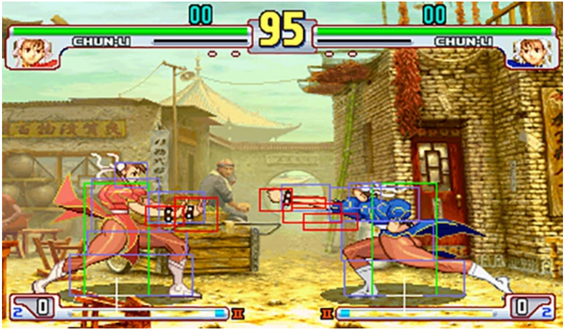


Figure: Collision detection in Street Fighter

Penetration

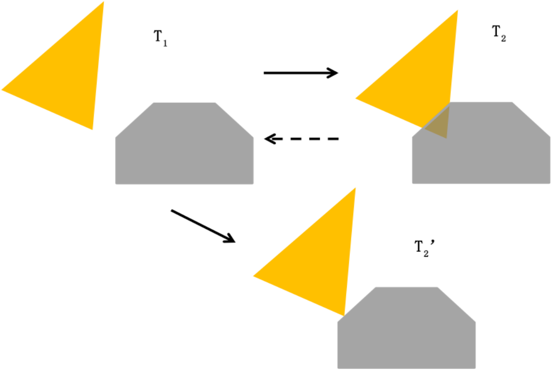
When collision is detected, the penetration follows.

Two forms of collision detection:

- Continuous: very expensive. Simulate solid objects in real life.
- Discrete: objects will end up with penetrating each other.

Backtracking

Like Tom Cruise in *Edge of Tomorrow*, it turns time backwards and fix the penetration that occurred in the last frame.



How to determine the collision between the dynamic objects and the terrain?

First, how is the terrain formed?
It is formed by base triangle primitives with displacement maps.

Outline

Base Primitive

$$\begin{aligned} \mathbf{p}(u,v) &= \mathbf{P}_0 + u(\mathbf{P}_1 - \mathbf{P}_0) + v(\mathbf{P}_2 - \mathbf{P}_0), \\ \mathbf{n}(u,v) &= \mathbf{N}_0 + u(\mathbf{N}_1 - \mathbf{N}_0) + v(\mathbf{N}_2 - \mathbf{N}_0). \end{aligned} \tag{1}$$

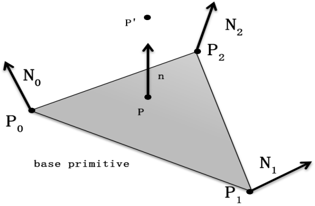


Figure: A base primitive.

2D Terrain Surface

$$\mathbf{p}'(u,v) = \mathbf{p}(u,v) + d(u,v)\mathbf{n}(u,v). \tag{2}$$

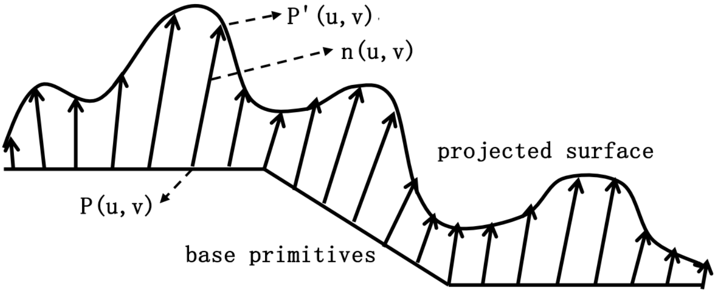


Figure: Base primitives and the projected surface.

Displacement maps

The terrain map consists of 64 $\times$ 64 terrain patches. Each terrain patch consists of several non-uniform terrain quads.

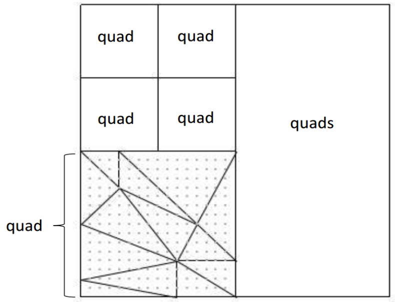
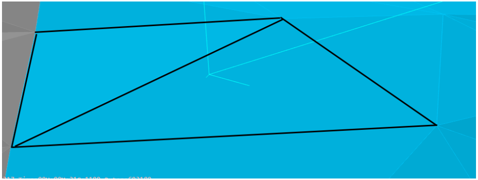
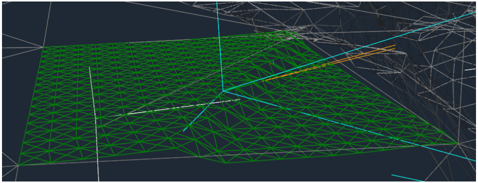


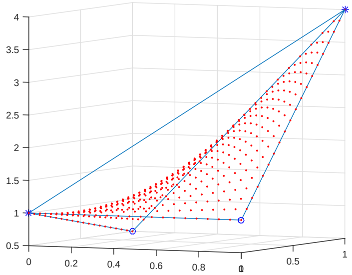
Figure: Displacement map of a terrain patch.(Slide 27,Slide 28)



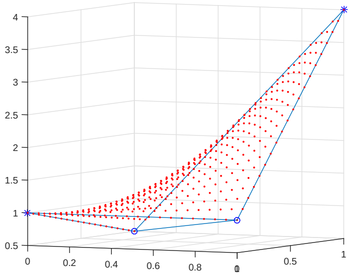


Projected Surface

A bilinear patch is formed by four non-coplanar points: $p_{00}([0,0,1])$, $p_{01}([1,0,1])$, $p_{10}([0,1,0.5])$, $p_{11}([1,1,4])$. Assume that the patch faces upwards.



(a) Two triangles can bound the bilinear patch.



(b) Two triangles cannot bound the bilinear patch.

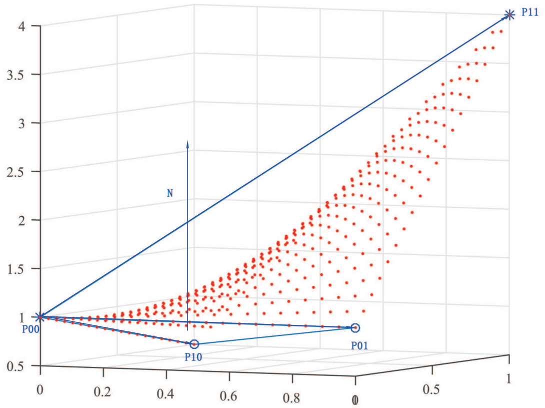


Figure: Determine triangle type. (Slide 30)

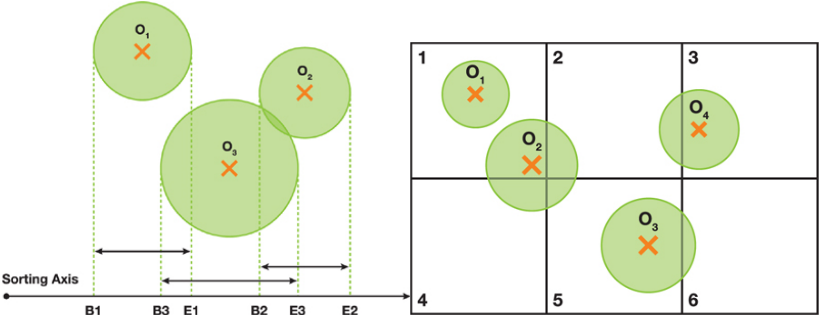
How to determine the collision between the dynamic objects and the terrain?

Second, how is the collision detection performed?
It is performed by two phases: the broad phase and the narrow phase.

Outline

Broad Phase

The broad phase collision detection helps to filtrate the potentially colliding object sets.
The brute-force way is doing collision test for all pairs of objects, and the complexity is $\mathcal{O}(n^2)$.



(a) Sweep and prune. (b) Spatial subdivision.

Tree Structure

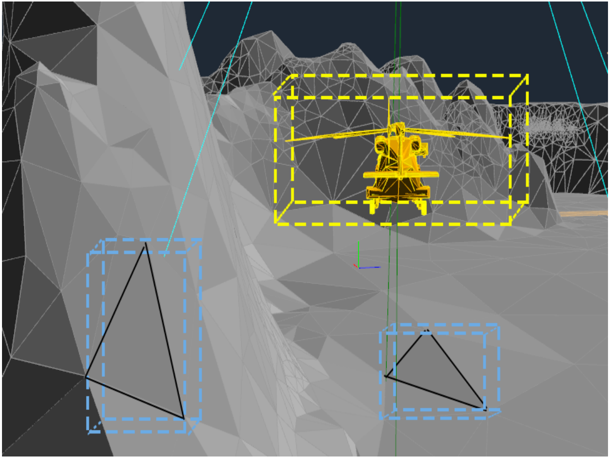
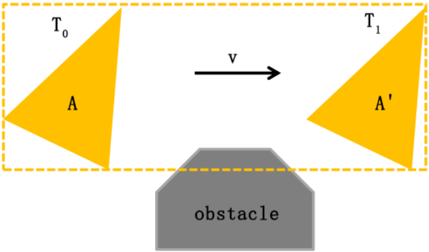


Figure: Collision detection between AABBs (Axis Aligned Bounding Box).

Swept Volume

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



Early Out

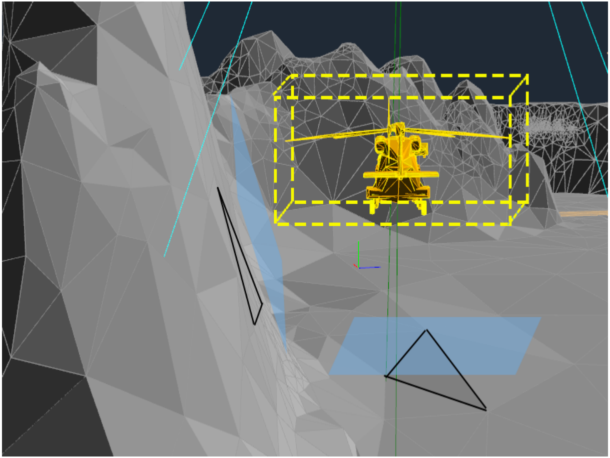


Figure: Early out check using the bounding planes.

Outline

Problem

The large terrain system contains complex geometry, like tunnels, caves and cliffs. The participating colliding shapes are usually restricted to being convex.

Multiresolution Bounding Volume

Multiresolution bounding volume method is created in this thesis for the static terrain collision detection.

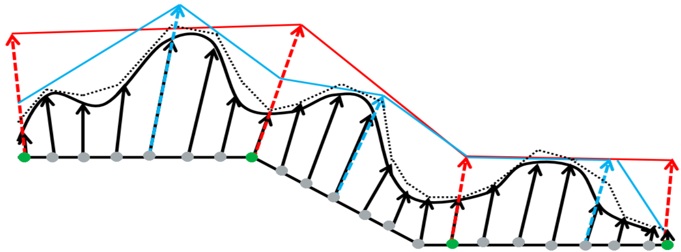


Figure: A 2D case. The coarsest bounding surface (red line), one level finer bounding surface (blue line) and finest bounding surface (black dot line). (Slide 30)

Bounding Volumes

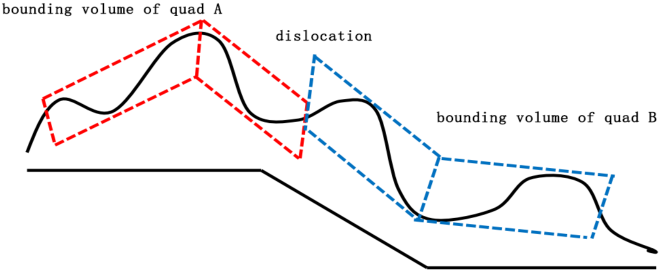


Figure: There will be a dislocation between bounding volumes between neighbouring quads.

Bounding Surface (2D)

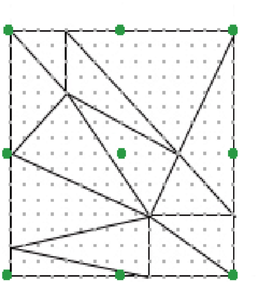


Figure: A 16 $\times$ 16 quad with constrained points (grey dots) and bounding points (green dots).(slide 13)

Optimization is done separately for each quad. Location relationship in the displacement maps (slide 13) is not corresponds with that in the world space.

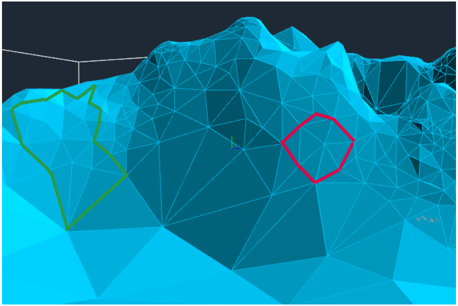


Figure: These two quads are detached in the world space. 4×4 (green) quad and 2×2 (red) quad

Inequality-constrained Nonlinear Programming Problem

The optimization problem is formed as follows:

$$\begin{array}{lll} \text{minimize} & \sum_{i=0}^{N} f(x_i) & \\ \text{subject to} & Q_j(x_i) \leq 0 & i = 1, \cdots, N, \\ & & j = 1, \cdots, M. \end{array} \tag{3}$$

where $x_i \in \mathbb{R}$ is the variable, which represents the forwards projected length from a bounding point.

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 (Slide 25, Slide 16):

$$f(x_i) = (x_i - l_i)^2 \tag{4}$$

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):

$$Q_j = (\mathbf{P}_1 - \mathbf{P}_0) \times (\mathbf{P}_2 - \mathbf{P}_0) \cdot (\mathbf{P}_j - \mathbf{P}_0) \tag{5}$$

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 $\mathbf{p}_j$.

Optimization Method

The problem is nonlinear problems with nonlinear inequality constraints.

$$
\begin{aligned}
&\text{minimize} && f_0(x) \\
&\text{subject to} && f_i(x) \leq 0, && i = 1, \cdots, m, \\
& && x_j^{\text{min}} \leq x_j \leq x_j^{\text{max}} && j = 1, \cdots, n.
\end{aligned} \tag{6}
$$

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.

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 [1].

$$
\begin{array}{lll}
\text{minimize} & f_0(x) + a_0 z + \sum_{i=1}^{m}(c_i y_i + \frac{1}{2} d_i y_i^2) & \\
\text{subject to} & f_i(x) - a_i z - y_i \leq 0, & i = 1, \cdots, m, \\
 & x_j^{\text{min}} \leq x_j \leq x_j^{\text{max}} & j = 1, \cdots, n, \\
 & z \geq 0 \text{ and } y_i \geq 0, & i = 1, \cdots, m,
\end{array}
\tag{7}
$$

where a_0, a_i, c_i and d_i are given real numbers such that $a_0 > 0$, $a_i \geq 0$, $c_i \geq 0$, $d_i \geq 0$, and $c_i + d_i > 0$, $a_i c_i > a_0$ for $i = 1, \cdots, m$. The index k represents the outer iteration number and l represents the inner iteration number.

The subproblem takes the form as follows:

$$\begin{array}{ll} \text{minimize} & g_0^{(k,l)}(x) + a_0 z + \sum_{i=1}^{m}(c_i y_i + \frac{1}{2} d_i y_i^2) \\ \text{subject to} & g_i^{(k,l)}(x) - a_i z - y_i \leq 0, \qquad i = 1, \cdots, m, \\ & x \in X^{(k)}, z \geq 0 \text{ and } y_j \geq 0, \end{array} \tag{8}$$

The function $g_i^{(k,l)}(x)$ in this subproblem is chosen as:

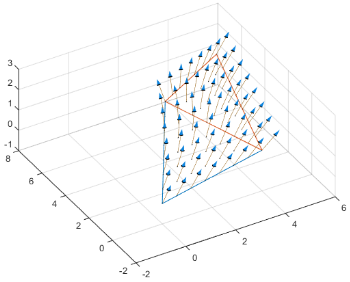
$$g_i^{(k,l)}(x) = v_i(x, x^{(k)}, \sigma^{(k)}) + \rho_i^{(k,l)} \omega_i(x, x^{(k)}, \sigma^{(k)}), \; i = 0, 1, \cdots, m. \tag{9}$$

where v_i and w_i are real value functions.

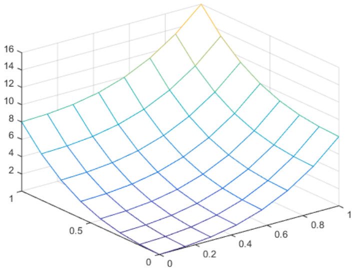
Algorithm 7 CCSA:

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

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.

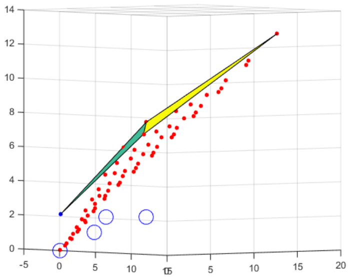


(a) The projection directions on the constrained points and the bounding points.

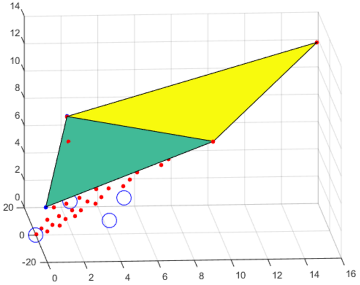


(b) Surface $f(x, y) = x^3 + y^3$.

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.

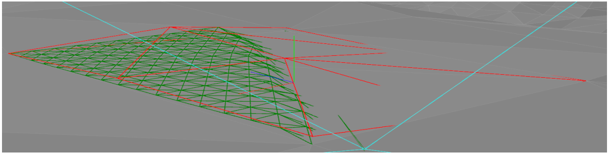


(a) Side view.

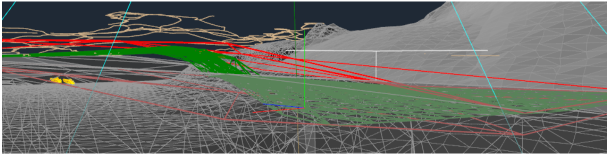


(b) Isometric view.

Bounding Volumes in the Game



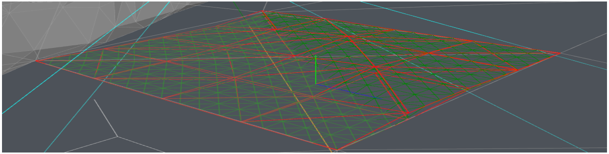
(a) Isometric view.



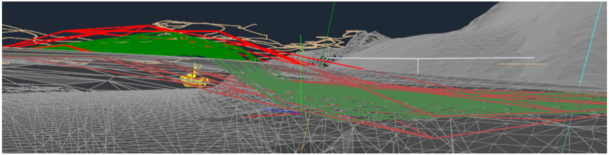
(b) Side view.

Figure: Upper and lower bounding surfaces with size 8.

Bounding Volumes in the Game



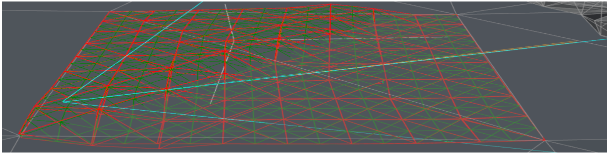
(a) Isometric view.



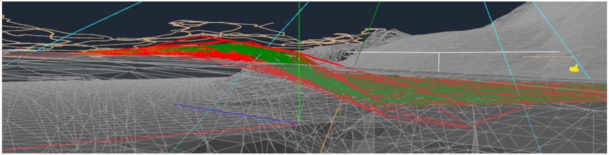
(b) Side view.

Figure: Upper and lower bounding surfaces with size 4.

Bounding Volumes in the Game



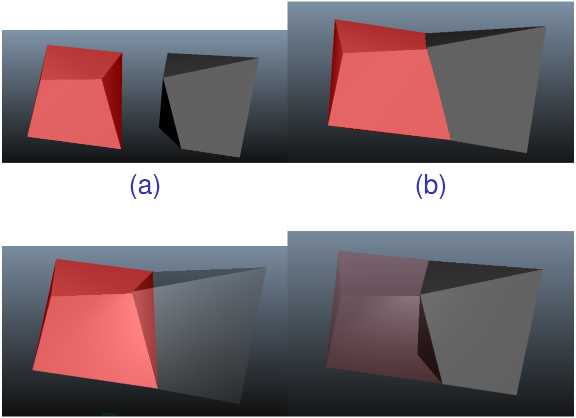
(a) Isometric view.



(b) Side view.

Figure: Upper and lower bounding surfaces with size 2.

Bounding Volumes



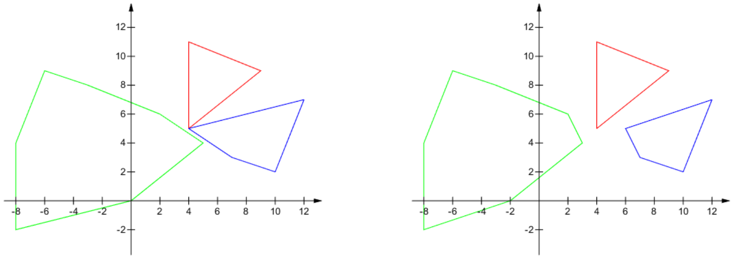
(a) (b)

(c) (d)

Figure: There will be no gap between neighbouring convex bounding

Convex versus Convex Collision Detection

One of the most commonly used algorithm in game engine is *GJK* (Gilbert-Johnson-Keerthi) algorithm [2]. This algorithm relies on a support function to iteratively get closer simplices to the solution using Minkowski difference.



(a) Two shapes collide on one vertex.

(b) No collision.

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.

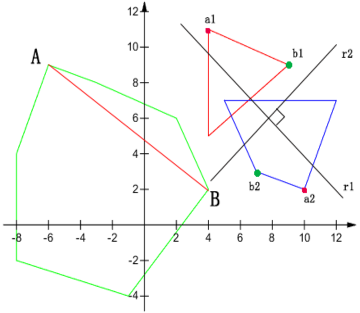
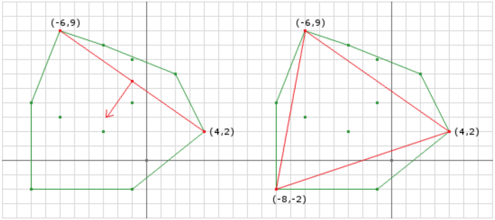
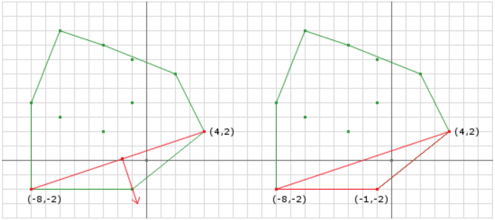


Figure: The first two vertices in the first simplex.



(a) The first iteration. (b) New simplex does not contain the origin.



(c) The second (d) New simplex

Multiresolution Collision Detection

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 $\neq 0$ **then**
      coarsest level detection
    else
      finest level collision detection
    end if
  end for
end function

Coarsest Level Collision Detection

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
```

One Level Down Collision Detection

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
```

Finest Level Collision Detection

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
```

Outline

Backward Projection

Definition

Backward projection is the inverse problem of forward projection, by projecting an arbitrary point in space onto the base primitive.

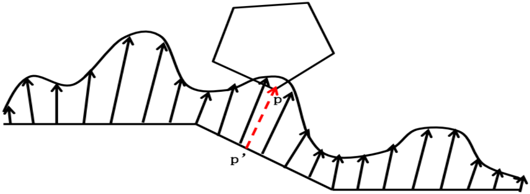


Figure: The projection length is smaller than the displacement length, and collision is detected.

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.

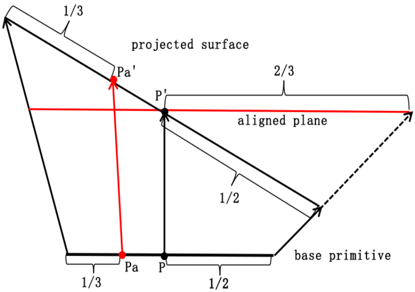
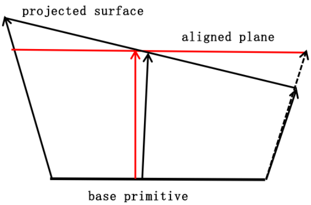
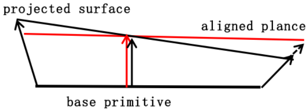


Figure: Aligned plane method (bad case).

This method works well for most cases in the game. In most cases, the lengths along the projection direction of the base primitives are small compared with the size of the base primitive (as (b) suggests) or the projection direction in one base primitive are close (as (a) suggests).



(a)



(b)

Bad Case

But bad cases can happen and will result in error when doing collision detection with the terrain surface.

(a) (b)

Aligned Projection Direction Method

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 $\mathbf{n}$.

The objective function is as follows:

$$
\begin{aligned}
f &= (r \times g) \cdot (r \times g) \\
\text{where} \qquad r &= u(\mathbf{N}_1 - \mathbf{N}_0) + v(\mathbf{N}_2 - \mathbf{N}_0) + \mathbf{N}_0 \\
g &= \mathbf{P}' - u(\mathbf{P}_1 - \mathbf{P}_0) - v(\mathbf{P}_2 - \mathbf{P}_0) - \mathbf{P}_0
\end{aligned} \tag{10}
$$

The BFGS (Broyden-Fletcher-Goldfarb-Shanno) algorithm is used to solve this unconstrained nonlinear optimization problem.

Alpha 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

$$\begin{aligned}\mathbf{P}'_0 &= \mathbf{P}_0 + \alpha\mathbf{N}_0,\\ \mathbf{P}'_1 &= \mathbf{P}_1 + \alpha\mathbf{N}_1,\\ \mathbf{P}'_2 &= \mathbf{P}_2 + \alpha\mathbf{N}_2.\end{aligned} \tag{11}$$

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

$$f = ((\mathbf{P}'_1 - \mathbf{P}'_0) \times (\mathbf{P}'_2 - \mathbf{P}'_0)) \cdot (\mathbf{P}' - \mathbf{P}'_0) \tag{12}$$

Newton's iteration method is used to obtain α.

Alpha Plane Method

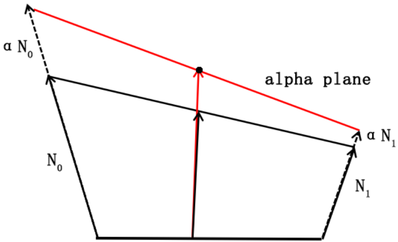


Figure: Aligned plane method.

Bad Case Fixed

(a) Alpha Plane Method.

(b) Aligned Projection Direction Method.

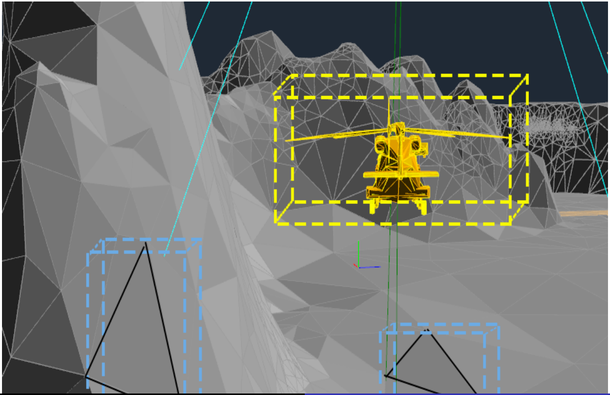
Outline

Test Environment

The character is teleported to a remote place without too many dynamic objects for test. Irrelevant complex objects, like the ships near the shore, are moved away from the terrain. Complex objects, like warships, are spawned to test the consuming time.

Original Method

Original method performs backward projection in each vertex of the dynamic object after the broad phase is done (refer to Slide 49). It is quite expensive compared with multiresolution collision detection.



Improvement using Multiresolution Collision Detection

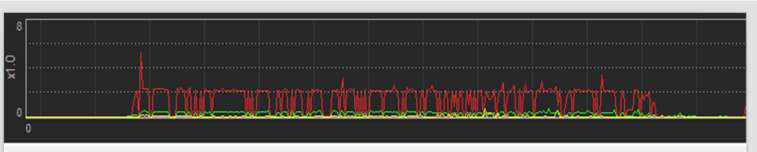


Figure: The time comparison when spawn one warship. Original collision detection (red), Multiresolution collision detection: coarsest bounding volume detection (green), finer bounding volume detection (violet, close to 0), final surface detection (yellow).

Improvement using Multiresolution Collision Detection

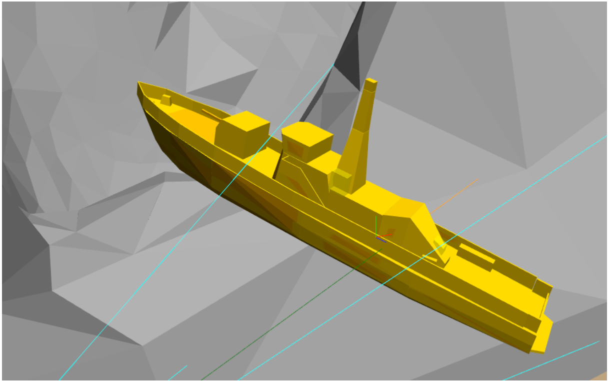


Figure: A warship contains many sub-objects. The multiresolution collision detection method will only detect the collision between the bottom of the warship and the terrain surface. It is notable that the gray mesh in this figure is not the terrain surface, but the projected primitives.

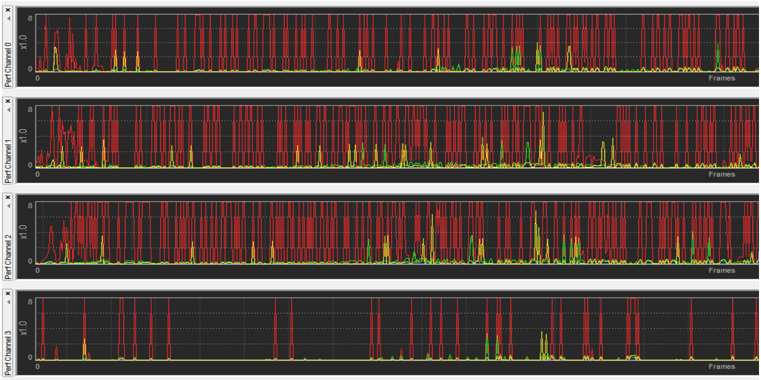


Figure: The time comparison on four threads when the warship is moved to some place with high density of primitives. Original collision detection (red), Multiresolution collision detection: coarsest bounding volume detection (green), finer bounding volume detection (violet, close to 0), final surface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

	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			

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

Outline

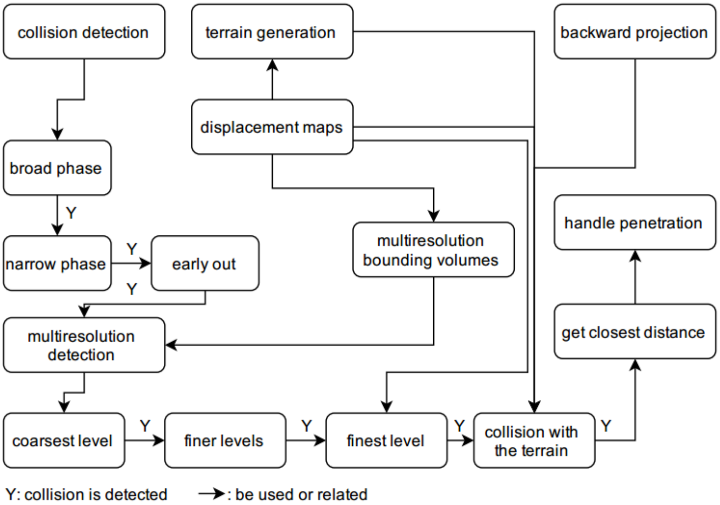


Figure: The flow chart of this thesis.

Reference

- Krister Svanberg. *A Class of Globally Convergent Optimization Methods Based on Conservative Convex Separable Approximations*. SIAM J. Optim. 12(2), p. 555-573, 2002p.
- 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.
- Java Collision Detection and Physics Engine. `http://www.dyn4j.org/2010/04/gjk-distance-closest-points/`.