Extraction of Multiple Stance Navigation Data in Arbitrary Polygonal Environments

Studienarbeit
im Rahmen des Diplomstudiengangs Informatik

eingereicht am Institut für Informatik
der Humboldt-Universität zu Berlin

von .....Sara Budde
und .....Leonard Kausch

Betreuer: .....Prof. Dr.-Ing. Peter Eisert

eingereicht am: 18.5.2012

Contents

1 Introduction **1**

2 Problem Definition **2**

3 Navigation Graph Definition **4**

4 Navigation Graph Generation **4**
- 4.1 Brief Concept . . . 4
- 4.2 Import of World Geometry . . . 5
- 4.3 Preliminary Welding of Input Polygons . . . 5
- 4.4 Steepness Test of Polygons . . . 5
- 4.5 Sort Polygons into Spatial Partitioning System . . . 6
- 4.6 Volume Collision Decomposition . . . 6
- 4.7 Smooth Welding of Traversable Polygons . . . 6
- 4.8 Optimization of the Navigation Graph . . . 7

5 Volume Collision Decomposition **7**
- 5.1 Idea . . . 7
- 5.2 Algorithm . . . 9
- 5.3 Slicing by Vertical Blocker . . . 12
- 5.4 Parallel Traversable and Blocker . . . 14
- 5.5 Cut off Blocker beneath Traversable . . . 14
- 5.6 Cut across Blocker in Upper Blocker and Lower Blocker . . . 15
- 5.7 Traversable Decomposition . . . 16

6 Blocker Slicing **18**
- 6.1 Idea . . . 18
- 6.2 Algorithm . . . 18
- 6.3 Distance Calculation . . . 21
- 6.4 Polygon-Plane Intersection Cases . . . 23
- 6.5 Blocker-Traversable Plane Intersection Point Calculation . . . 27
- 6.6 Blocker Slicing in Lower Blocker and Upper Blocker . . . 31

7 Traversable Decomposition **32**
7.1 Definition . . . 32
7.2 Algorithm . . . 33
7.3 Projected Outline Intersection . . . 36
7.3.1 Definition . . . 36
7.3.2 Line-Line Intersection . . . 38
7.3.3 Intersection Point Definition . . . 41
7.4 Traversable Totally Contained in Blocker . . . 43
7.5 Blocker Totally Contained in Traversable . . . 43
7.5.1 Definition . . . 43
7.5.2 Start Vertex Selection . . . 44
7.5.3 Convex Decomposition . . . 46
7.6 Coplanar Polygon Cutting . . . 53
7.6.1 Intro/Overview . . . 53
7.6.2 Blocker and Traversable are Coplanar in 3D-Space . . . 53
7.6.3 Intermediate Vertices . . . 53
7.6.4 Coplanar Polygon Cutting Example . . . 56
7.6.5 Coplanar Polygon Cutting (CPC) Automaton . . . 59
7.6.6 CPC Convex Decomposition Call . . . 64
7.6.7 CPC Result Processing . . . 66

8 Slicing by Vertical Blocker **67**

9 Implementation **69**
9.1 Introduction . . . 69
9.2 Conservative Vertex Creation . . . 69
9.3 Minimal Edge Length and Vertex Snapping . . . 70

10 Conclusions **71**

11 Acronyms **73**

12 References **73**

This document describes an efficient way to extract multiple stance navigation data with complete coverage for human like agents from arbitrary polygon clouds in three-dimensional space. The multiple stance information is obtained by layered volume obstruction tests realized in the Volume Collision Decomposition.

1 Introduction

To be able to have agents planning complex navigation paths you first need to have detailed data about how and where different movements are possible. So to have complex movement of agents, one needs to have a precise and detailed navigation graph (see section 3) with complete coverage of the traversable surface.

Our every day surroundings are build to be easily traversable, so the ability for complex movement shines when the agents are put into more chaotic environments. To give a striking example, to move through the ruins of a collapsed skyscraper requires detailed knowledge of the traversability of thousands of pieces of debris together with a complex pathfinding that incorporates different methods of movement.

The field of use ranges from path planning in robotics to the use in AI engines for the entertainment industry. Any kind of simulation involving virtual path planning is most likely to greatly benefit from the ability for complex movement on a perfectly detailed navigation graph.

The section 2, section 3, section 4 and section 7 were primarily written by Leonard

Kausch and the section 5, section 6, section 8 and section 9 were primarily written by Sara Budde.

2 Problem Definition

In the past Navigation Graphs (see section 3) for agents in virtual worlds were made by hand, which is a very time consuming work, and the work is obsolete when the level is modified. Automating this process with high precision and speed is essential [Toz02; Toz03] to be able to manage rapidly changing worlds or pathfinding in worlds that are explored in real time.

State of the art navigation mesh generation is using Space Filling Volumes [HYD08; HY09] or voxel molds [Zha+07] as an intermediate world representation. From these intermediate models the actual polygonal navigation mesh is then extracted. An example of a voxel based implementation of navigation mesh generation is *Recast* [Mon].

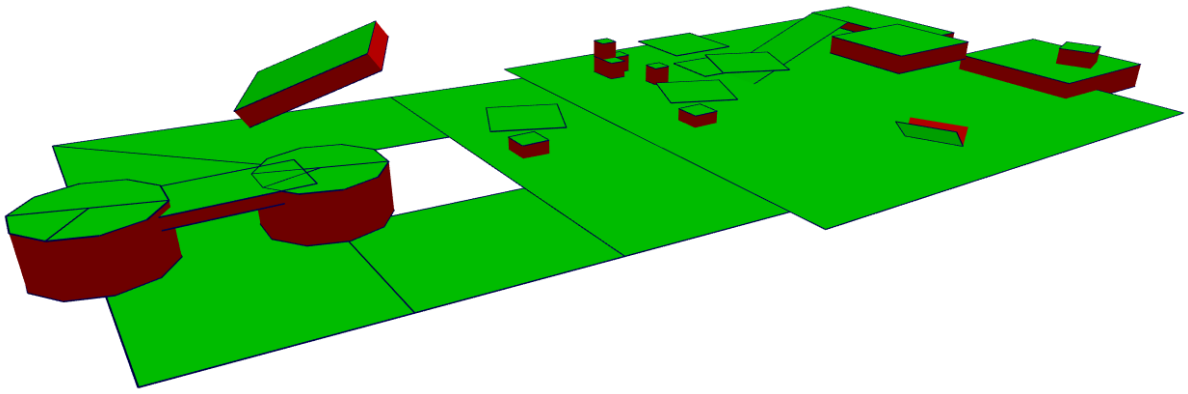


Figure 1: *Level geometry after the steepness test*
The red surfaces can not be traversed by the agent because it is too steep. The green surfaces on the other hand are potentially traversable in any stance.

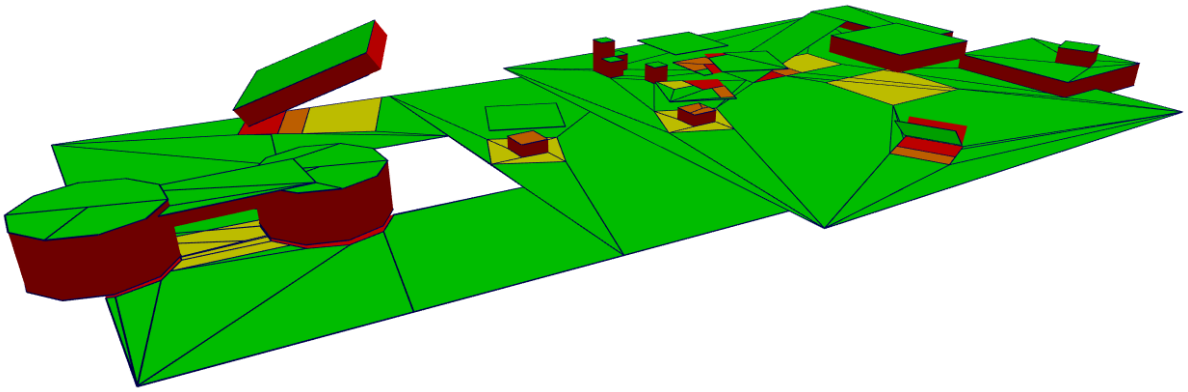


Figure 2: *Extracted multi stance navigation data for a complete test level*
The red surface can not be traversed by the agent, orange can be traversed crouching, yellow can be traversed ducked and on the green surface the agent is able to stand.

In our work, we target 100% coverage of the navigable surface. According to "Building a Near-Optimal Navigation Mesh" [Toz02] this "completeness" is one of the important goals of building a optimal navigation mesh. This is important for the raw navigation data to be agent independent, the smallest agent defines the minimal relevant navigatable space. Techniques like Space Filling Volumes [HYD08; HY09] have a natural inaccuracy, because they converge towards 100% but never reach it. Because we directly work on the world geometry itself, there is no theoretic inaccuracy, only the inaccuracy depending on the precision of the arithmetic calculations during the navigation graph generation.

We also aim for the navigation data to represent the abilities for different movement methods of the agents. These movement stances are defined by a necessary unobstructed height above a traversable surface. Stances will be put into a linear ordering and will then be incorporated into the Navigation Graph Generation by layer based obstruction tests. This will enable artificial intelligence algorithms to employ different movement methods for different agents to traverse complex terrain. Basically we improve the information

richness of the AI world representation so the agents can improve on the foundation of this richer knowledge [You+11].

3 Navigation Graph Definition

A Navigation Graph, which is also called navigation mesh, is defined by Tozour as "a representation that covers the walkable surfaces of the world with convex polygons"[Toz03, page 95]. Like any graph, this navigation graph is defined by nodes and edges. The nodes are polygons and "the links between adjacent polygons are the edges of the graph"[Toz03, page 96]. One important thing about navigation meshes is that: "Convex polygons allow us to guarantee that a character can freely walk from any point within a polygon to any other point within that same polygon"[Toz03, page 96]. The ideal navigation graph covers 100% of the navigable surface and decomposes this surface into a minimal number of convex polygons.

4 Navigation Graph Generation

4.1 Brief Concept

The Navigation Graph Generation is composed of seven steps shown in the figure below. The biggest part of this document is about the Volume Collision Decomposition, which is the core around which the other topics are arranged. The goal of the Navigation Graph Generation is to extract the traversable space according to variable agent properties from the source world geometry.

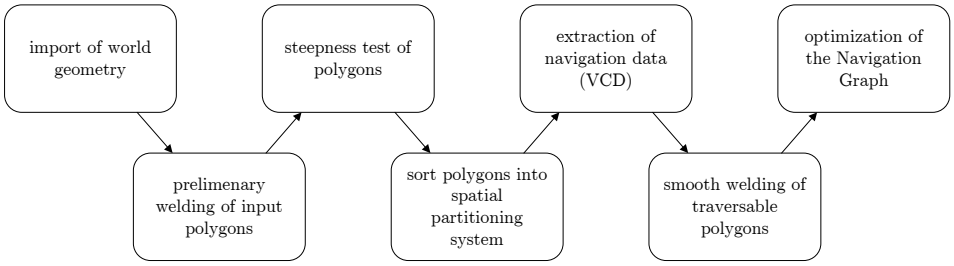


Figure 3: *Navigation Graph Generation*

4.2 Import of World Geometry

The world geometry is imported from OBJ files. We do not preserve mesh structures, everything is decomposed to the polygon level. The Normals in the OBJ files are ignored, because we recalculate them, but we presume that all other preliminaries, like planarity and convexity, are met.

4.3 Preliminary Welding of Input Polygons

Preliminary welding is extremely useful when the import file strictly only contains triangles. In this case multiple triangles defining one planar polygonal plane can be welded together into one single polygon. The benefit of this welding depends on the actual positioning of all polygons, but is most likely significant for common geometry.

4.4 Steepness Test of Polygons

We do take every polygon as a two sided entity, which means that we only use normals with a z-component that is zero or positive to define the steepness. If the z-component is negative, it is flipped. Polygons that are steeper than the predefined maximum traversable steepness are made a blocker and the other are made traversables.

4.5 Sort Polygons into Spatial Partitioning System

All polygons are now sorted into a spatial partitioning system. This spatial data structure is supposed to reduce the amount of obstruction tests, by applying a heuristic to the spatial separation. The simplest way would be a octree with a modified iterator. The topic on optimal spatial data structures is covered well in many papers. In the following we presume that the blockers and the traversables can be separately accessed in the spatial data structure.

4.6 Volume Collision Decomposition

The Volume Collision Decomposition (VCD) is the core of this document. The VCD is the one part of Navigation Graph Generation that makes the traversable space coverage 100%. All traversables identified in the steepness test are iterated and checked against all possible blockers. If a blocker obstructs the traversable in any kind, the traversable will be decomposed into appropriate new convex traversables and/or blockers.

4.7 Smooth Welding of Traversable Polygons

The obstruction calculations precisely cover which surface is traversable and how, but there is no tribute to the context in which all the traversables stand to each other. In example, if you have a bridge with crosswise boards and there are small gaps between all boards, then the VCD would not make this bridge one traversable surface. But because the gaps in between the boards are smaller than the foot size of a human Agent, they are irrelevant for the movement problem. And that is what smooth welding is doing, it merges traversable polygons together where the gaps are small enough to be neglected. When two edges are welded together the basic effect for the Navigation Graph is the creation of a new navigation edge between two existing polygons.

After this step the navigation graph is practically ready for use, the polygons are the nodes and the shared edges are the links between these nodes. The agent is already taken into account considering the maximum weld distance depending on the agent size.

4.8 Optimization of the Navigation Graph

As an optional last step the navigable surface can now be optimized to fit certain criteria that algorithms either need or make them faster. This optimization affects how the surface is decomposed into convex polygons. Standard criteria are having minimal number of nodes or having the most regular polygons.

5 Volume Collision Decomposition

5.1 Idea

The Volume Collision Decomposition calculates the obstructed surface of a traversable polygon by projecting the obstruction onto the traversable polygon and then decomposing the result into new convex polygons.

The basic idea is described in [Far06] , where they extruded the blockers along the negative up axis (negative z-axis) according to agent height and checked if this volume intersects with any traversables and if so, marked the areas of the traversable as obstructed. So they calculated where it is possible to stand.

Our idea is to extrude the traversables along the positive up axis (positive z-axis) and check if any blockers intersect with this volume (see Figure 4). If a blocker intersects with a volume of an extruded traversable, then the traversable will be decomposed into one new obstructed convex polygon and a finite number of new traversable convex polygons

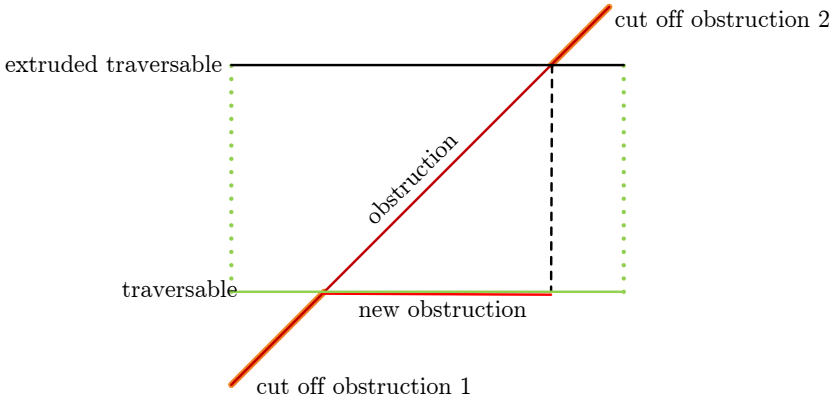


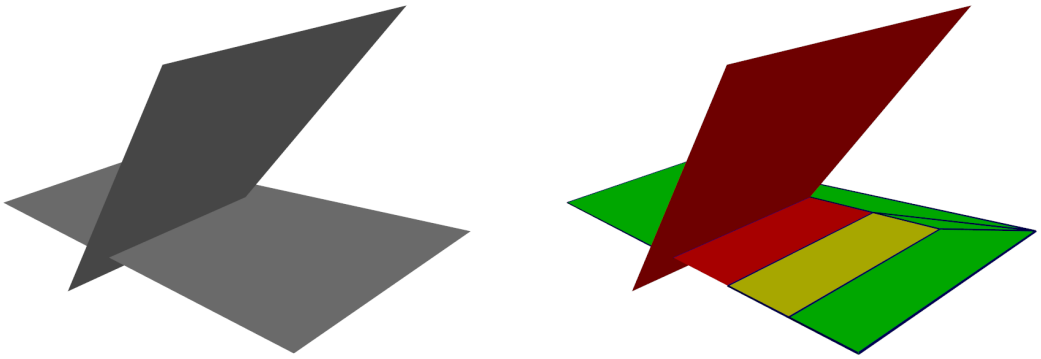
Figure 4: *Simplified example of the Volume Collision Decomposition concept*

according to the height of the stances. Meaning if the distance between the traversable and the intersecting blocker is smaller than the smallest stance height, then it is not possible to traverse this area. If the distance is higher than the smallest stance, then a new traversable with the according stance restriction will be extracted from the original traversable.

To support the information of how an area is traversable, the blocker needs to be sliced into separate parts according to its distance from the traversable. This way it is possible to represent crawling, crouching and standing in the data model. A part of the blocker that corresponds to one stance is called a slice of that blocker.

It is easy to get confused which distance interval, that defines one slice, corresponds to which stance. The first slice will be made from zero distance to the minimum height of the first stance, which means it is the slice corresponding to a blocked area (see "new obstruction" in Figure 4). The end of one slice is always defined by the start of the next one. So the lower end of a slice interval is always the minimum height for the corresponding stance and the higher end of the slice interval is always the minimum height of the next stance corresponding to a more freely traversable area.

The last stance is the one where the agent can move without any stance restrictions, so it has no maximum height, meaning it always contains the rest of the blocker in its slice.



(a) Input geometry before VCD

(b) Visualization of the navigation data after the VCD

Figure 5: *Screenshots of one of our example cases*
(a) The lower quadrangle is a potentially traversable polygon and the upper quadrangle is too steep to traverse and is therefore a blocker. **(b)** The red surface can not be traversed by the agent, yellow can be traversed ducked and on the green surface the agent is able to stand.

5.2 Algorithm

The VCD always works on one traversable polygon and one blocker polygon which might obstruct the traversable. Only the traversable polygon might get altered during the VCD. The blocker polygon is never changed.

Important to understand for some of the special cases in the following explanations is that the blocker inside the VCD can also be a traversable. The Navigation Graph Generation that calls the VCD iterates over all traversables. Every traversable is checked against all other traversables and blockers that are returned by the spatial partitioning

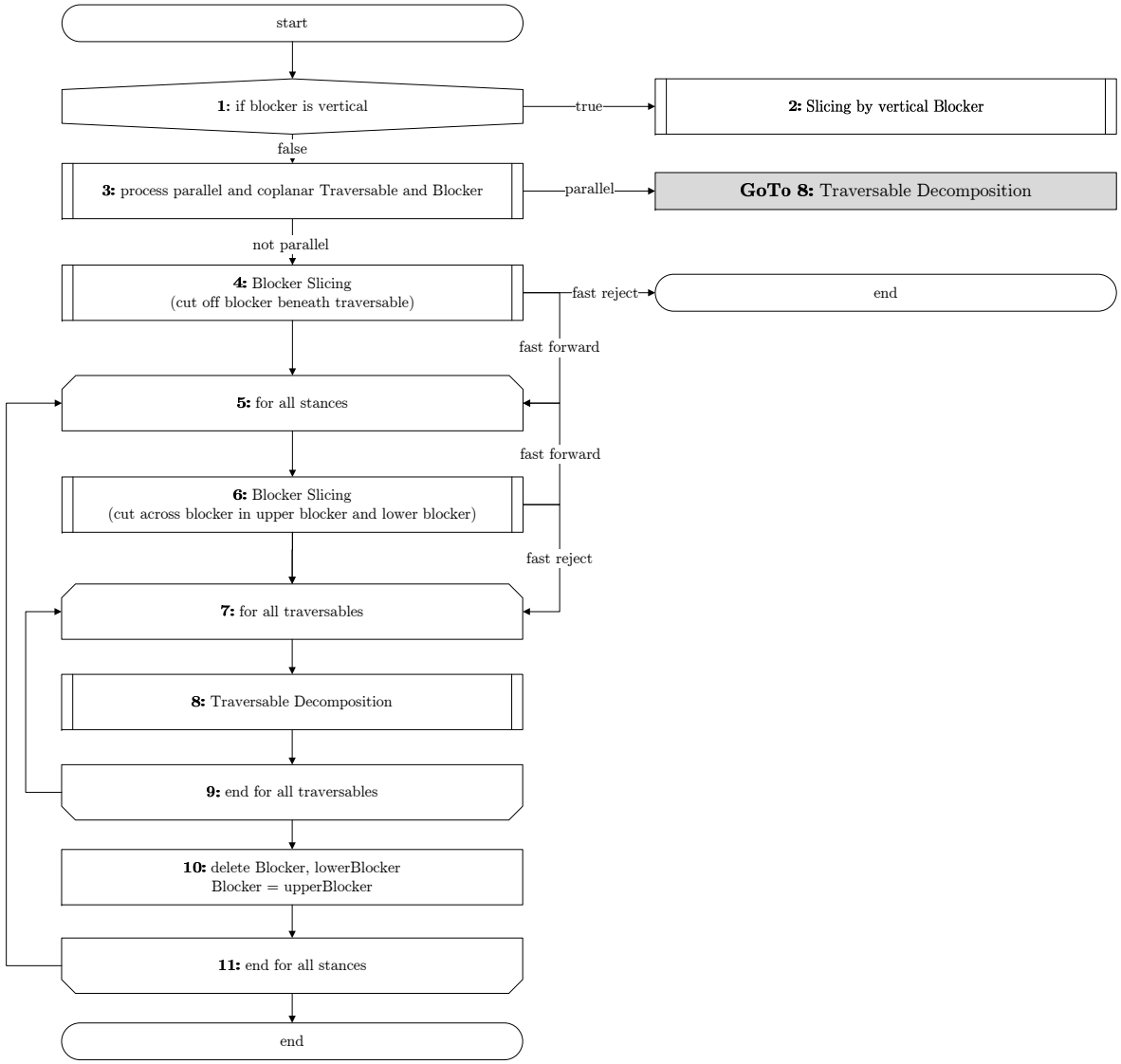


Figure 6: *Volume Collision Decomposition*

system as heuristically possible collisions. The reason is that for the traversable it makes no difference if the polygon obstructing the space above is a blocker or a traversable.

We assume that all polygon normals are pointing upwards. With this preliminary condition, it is certain that two polygons that are projected into the horizontal-plane are always both clockwise defined towards the positive z-axis. This is important for the VCD because the ordering of the vertex list directly relates to the way the intersections are handled in the VCD.

The following paragraph explains the flow through the VCD. The VCD consists of five main parts, which will be described briefly here and in more detail in the following subsections.

The first step in the VCD is to check if the blocker is vertical. There is a procedure that handles vertical blockers, which is described in subsection 5.3.

If the blocker is not vertical, we check if the blocker plane and traversable plane are parallel. What is done if the traversable plane and the blocker plane are parallel is described in subsection 5.4 and after this we continue with the Traversable Decomposition (TD).

If the traversable plane and the blocker plane are not parallel, there could be a part of the blocker which is below the traversable plane. Because this lower part of the blocker would not obstruct anything of the traversable, we cut it off. This is done by the "Blocker Slicing", which is also labeled with "cut off blocker beneath traversable" (see subsection 5.5). We continue with the upper part of the blocker, which lies above the traversable plane, as blocker for the rest of the VCD.

The "for all stances"-loop iterates over the stances by changing the slicing plane in each iteration according to the next stance. The initial obstruction level of the traversable

is the upper border of the stance tests. That is because there is no sense in testing a stronger obstructed surface against a weaker obstructing blocker, because it is already in a stronger obstruction state.

The changed slicing plane is then used in the "Blocker Slicing", which is also labeled with "cut across blocker in upper blocker and lower blocker", to get the obstruction for the current stance (lower blocker) and the obstruction for the remaining stances (upper blocker). This is explained in more detail in subsection 5.6.

The "for all traversables"-loop is required, because one stance obstruction might require the initial traversable to be decomposed into a number of convex polygons, so the following stance obstructions have to be applied to a number of convex polygons representing the rest of the initial traversable.

The TD is calculating the surface of the traversable which is obstructed by the lower blocker (see subsection 5.7). If necessary the rest of the traversable will be decomposed into several polygons to match the convexity constraint.

5.3 Slicing by Vertical Blocker

A vertical blocker obviously does not cover an area if projected into horizontal-plane. Instead the projected blocker is a line segment. This special case is handled by cutting the traversable along the line segment defined by the projected blocker. The resulting two traversable polygons are always convex. The line along which the traversable is cut needs to be separated into up to three edges, depending on the case of the intersection between the vertical blocker and the traversable. This is because the projected blocker line segment is a obstruction and this need to be saved in the edge. The part of the line segment representing the cut which is not part of the projected blocker line segment is a traversable edge. This is further explained in section 8.

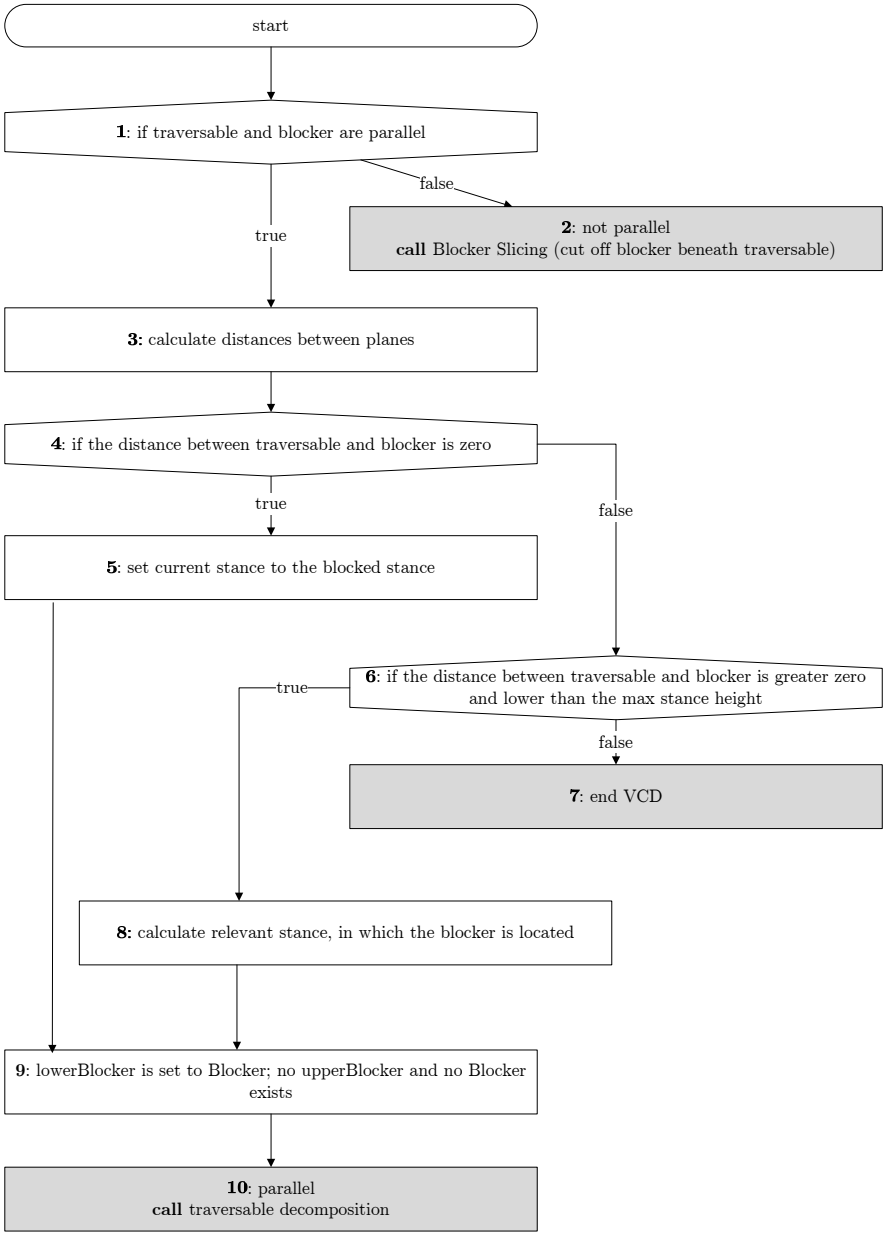


Figure 7: *Parallel Traversable and Blocker*

5.4 Parallel Traversable and Blocker

If the blocker and the traversable are parallel, any obstructed surface of the traversable has the same distance to the blocker, meaning there can only be one relevant slice. This means that there is only one stance relevant for the obstruction of this traversable. This stance can be determined by the distance between the traversable plane and the blocker plane.

If the distance is zero, the traversable and the blocker are coplanar, which is a special case of the parallel planes. In this case the stance which is needed for the obstruction calculation is the blocked stance. If the distance is greater than the stance of the traversable, this VCD run can be stopped and the traversable will be returned, because nothing there is no stronger obstruction. So if the distance between the plane is between zero and the traversable stance height, the appropriate stance will be calculated and then the VCD can continue with the TD. The TD will be described in subsection 5.7.

5.5 Cut off Blocker beneath Traversable

The process of "cut off blocker beneath traversable" cuts the blocker along the plane in which the traversable lies into a lower part and an upper part. The lower part can not obstruct the traversable in any way, so it is discarded. The upper part of the blocker is used for the further obstruction calculation in the VCD. After this step it is considered the blocker in the VCD. The "cut off blocker beneath traversable" is one form of the blocker slicing. What distinguishes the different forms of the Blocker Slicing is described in more detail in section 6. The different exits and where they lead to is described in the following paragraph.

In this Blocker Slicing we want to cut off the blocker beneath the traversable plane, that is why we have three different situations how the blocker can be positioned relative to the

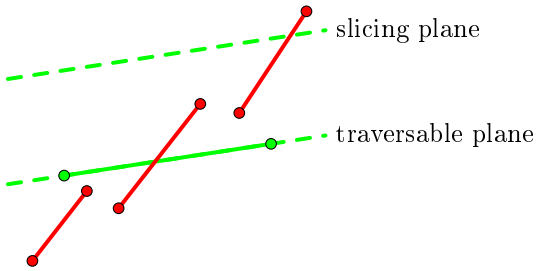


Figure 8: *Three different situations between blocker and traversable in "cut off blocker beneath traversable"*

traversable, more precisely the traversable plane. If the blocker is below the traversable plane, the Blocker Slicing quits through the exit which is labelled with "fast reject". In this case the traversable is not obstructed in any way by the blocker and the VCD returns the traversable as a sign that nothing was changed by this VCD run. If the Blocker is above the traversable plane, there is nothing to cut off and the VCD can progress to the obstruction calculation for the first stance. In this situation the Blocker Slicing outcome is the exit labelled with "fast forward". In the case that the blocker pierces through the traversable plane, the Blocker Slicing cuts off the part of the blocker which is below the traversable plane. Then the VCD continues with the obstruction calculation for the first stance.

5.6 Cut across Blocker in Upper Blocker and Lower Blocker

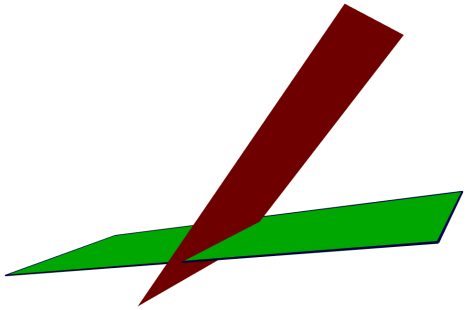
The process "cut across blocker in upper blocker and lower blocker" is the second form of the Blocker Slicing. In this form the blocker is cut across into two parts, a lower blocker and an upper blocker, by a slicing plane. A slicing plane is a parallel plane to the traversable plane and always corresponds to one stance. A slicing plane is calculated by shifting the traversable plane along the up-axis by the height of the current stance. This will be done in the VCD for several stances and for every stance the lower part of the

blocker will be used to compute the obstruction of the traversable for this stance. The upper part of the blocker will be used for the obstruction calculation of the next stance.

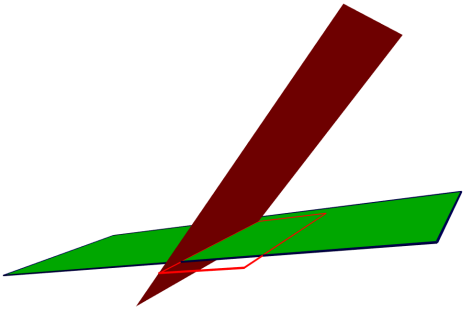
Like "cut off blocker beneath traversable" this Blocker Slicing has also three exits. The exit "fast reject" activates if the blocker is totally contained in the volume between the last slicing plane and the current slicing plane and there is no upper blocker. In this situation there will be the obstruction calculation for this stance, but there is no obstruction calculation for further stances. The exit "fast forward" means that the blocker is above the slicing plane, so the VCD can proceed with the obstruction calculation for the next stance. In the case that the blocker pierces through the slicing plane, the blocker is cut across by the slicing plane in a lower blocker and an upper blocker. The lower blocker is used to calculate the obstruction of the traversable for this stance and the upper blocker is used for the obstruction calculation for the next stances (see Figure 9 (b)).

5.7 Traversable Decomposition

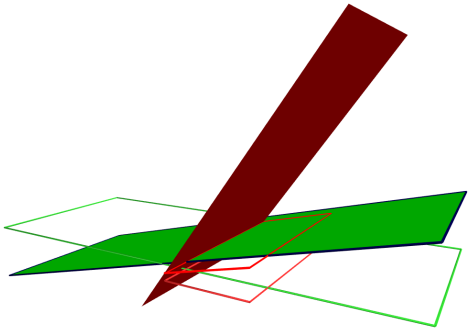
The traversable is decomposed in new traversables and one new blocker. The new blocker covers the surface of the traversable which is obstructed by the lower blocker, which is determined by the current stance via the Blocker Slicing (BS). The rest of the traversable, which is not obstructed by the lower blocker, is covered by the new traversables (see Figure 9 (c)). There is normally more than one new traversable, because all polygons have to be convex. If a temporary new traversable is not convex, it will be decomposed in several convex polygons. This is also the reason why there is a loop in the VCD which iterates over all traversables, despite the fact that the VCD initially only works with a pair of one traversable and one blocker. The TD is described in more detail in section 7.



(a) Rendering of a potential traversable and a blocker



(b) Rendering of a lower blocker projected into the traversable plane



(c) Rendering of the projected current traversable and the projected lower blocker

Figure 9: *Three step visualization of the projections done in the VCD*
(a) The red quadrangle is a blocker, because it is too steep. The green quadrangle is a potential traversable polygon, which has not yet been processed by the VCD. **(b)** The red outline is the projection of the first lower blocker into the plane of the traversable after the blocker has been cut off beneath the traversable. **(c)** The green and the second red outline are the projections of the current traversable and the lower blocker into the horizontal-plane to compute the part of the current traversable which is obstructed by the lower blocker.

6 Blocker Slicing

6.1 Idea

The blocker slicing means that a convex blocker polygon is being cut into two new convex polygons along a plane. This plane is called "slice plane", and is in our case a parallel plane to the traversable polygon. The blocker slicing is used to divide the blocker into different parts according to the stances of the agents. The first slice plane is always defined by the traversable itself, and the following slice planes are always planes defined by parallel shifting the traversable plane upwards according to the next stance. In the following we will always refer to slice planes, so it is important to keep in mind where they originate from.

The blocker slicing is used in the VCD in two slightly different ways. It always returns a lower and an upper part that together represent the input blocker. In the VCD the first use of the blocker slicing is to cut off the part of the blocker that lies beneath the traversable, because that part is never obstructing the space above the traversable. This means that the lower polygon returned by the blocker slicing is discarded and the upper polygon is kept for further calculations. After the first usage of the blocker slicing, the lower part always represents a stance specific obstruction and the upper part is the rest of the blocker that is kept for the next stance slices of the VCD.

6.2 Algorithm

Under Blocker Slicing we understand all calculations that have to be done to compute the intersection points between a slice plane and the blocker as well as the actual slicing of the blocker in a lower blocker, which is below the slicing plane, and an upper blocker, which is above the slicing plane. The Blocker Slicing has three different exits. The first one is a

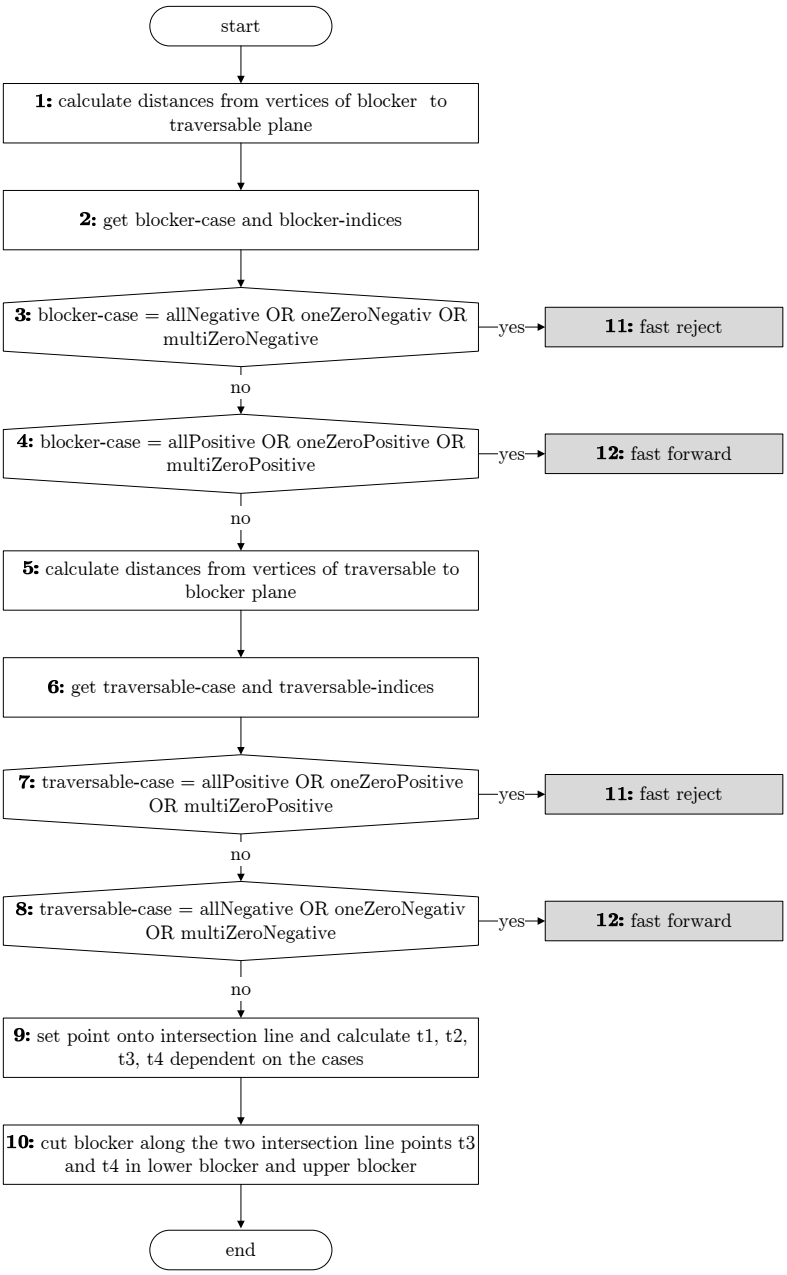


Figure 10: *Blocker Slicing*

reject, which means there is nothing to cut because the blocker is below the slicing plane. What we do after this depends on where we are in the VCD. There is also a fast forward, which means that there is nothing to cut, but it is possible that there is something to cut in the remaining stance obstruction calculations, because the blocker is above the slicing plane. And of course there is that case where the blocker is actually cut in lower blocker and upper blocker, which means that the blocker is below and above the slicing plane. What these different exits mean for the VCD has been described in subsection 5.5 and subsection 5.6.

The first step is to calculate the intersection points between the blocker and the slicing plane. To do this we compute the distance of each vertex of the blocker to the slicing plane (Figure 10 state 1) and later the distances between the parallel shifted traversable vertices and the blocker plane. After we have computed the distances of the blocker vertices, we determine the blocker case. The blocker case describes the position of the blocker vertices relative to the slice plane, this is explained in detail in subsection 6.4. Then we check if the blocker lies below the slicing plane, if it does, we can fast reject. If the blocker lies above the slicing plane, there is nothing to cut and we can fast forward to the next stance obstruction calculation. If the blocker pierces the slicing plane, we continue with the distance calculation of the parallel shifted traversable vertices (Figure 10 state 5) and the traversable case determination (Figure 10 state 6). The situation for the fast reject and the fast forward is still the same as above. But because we are looking at the traversable and not at the blocker, the traversable case for the fast reject and the fast forward are not the same as above (see Figure 10 state 7 and state 8).

If the blocker pierces the slicing plane and the parallel shifted traversable pierces the blocker plane, we calculate the intersection points between the blocker and the slicing plane. These intersection points are calculated by a scalar to a particular line equation. The line equation is defined with a point on the line, which we determine in this step (Figure 10 state 9) and the direction vector is calculated by the cross product of both

planes' normals.

With the calculated intersection points we slice the blocker in a lower part and an upper part. The lower part of the blocker is also called "lower blocker" and lies below the slicing plane. The upper part, which lies above the slicing plane, is called "upper blocker".

6.3 Distance Calculation

The Distance Calculation computes the signed distances between all vertices of a polygon and a plane given by another polygon. A signed distance is the minimal distance and the sign implies on which side of the plane the point lies. These distances are used to determine an intersection case which is directly used for a rejection test, but the calculated distances are also used in the calculation of the intersection points between the blocker and the slicing plane.

Let the traversable have n_t vertices and the blocker have n_b vertices. We denote the traversable vertices with $\mathbf{v}_0^t, \ldots, \mathbf{v}_{n_t-1}^t$ and analog the blocker vertices with $\mathbf{v}_0^b, \ldots, \mathbf{v}_{n_b-1}^b$. The plane in which the traversable lies is described in Hessian normal form $\hat{\mathbf{n}}_t \cdot \mathbf{x} = -p_t$, where $\hat{\mathbf{n}}_t$ is the unit normal vector and p_t is the minimal distance of the traversable plane to the origin. Analog the plane in which the blocker lies is given by $\hat{\mathbf{n}}_b \cdot \mathbf{x} = -p_b$.

The distance between a point $\mathbf{x}_0$ and a plane given in Hessian normal form $\hat{\mathbf{n}} \cdot \mathbf{x} = -p$ can be calculated by

$$D = \hat{\mathbf{n}} \cdot \mathbf{x}_0 + p$$

where $\hat{\mathbf{n}}$ is the unit normal vector and p is the distance of the plane from the origin. If the point $\mathbf{x}_0$ is in the half-space determined by the direction of the normal, then $D > 0$, or else it is in the other half-space. Because our normal is pointing upwards, the point would be below the plane if $D < 0$ and over the plane otherwise (see [Weia]).

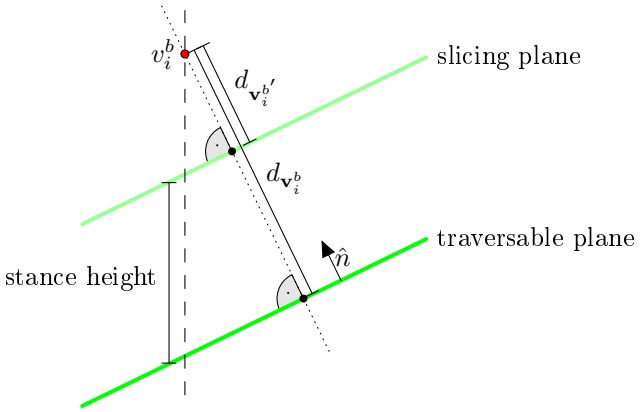


Figure 11: *Sketch of a blocker vertex and the traversable plane and a slicing plane*

To compute the distances between the blocker vertices and the traversable plane and between the traversable vertices and the blocker plane, we use the formula described above. But to calculate the distances between the blocker vertices and a slicing plane and between the parallel shifted traversable vertices and the blocker plane we use this formula to calculate the distance between one vertex and a plane. With the distances calculated before we can now calculate the distances by computing a delta distance and adding this to all already calculated distances. The delta distance Δ is calculated for any traversable vertex $\mathbf{v}_i^t$ by

$$\Delta = (\hat{\mathbf{n}}_t \cdot \mathbf{v}_i^t + p_t) - (\hat{\mathbf{n}}_t \cdot \mathbf{v}_i^t + (p_t + \mathit{stance\ height} \cdot \hat{\mathbf{n}}_t.z)) \ ,$$

where *stance height* is the height of the current stance and $\hat{\mathbf{n}}_t.z$ is the z-component of the unit normal vector of the traversable plane. So $\mathit{stance\ height} \cdot \hat{\mathbf{n}}_t.z$ is the distance between the traversable plane and the parallel shifted traversable plane. This is because the parallel shifted traversable plane is, like the name says, parallel to the traversable plane, and the shifted distance is the *stance height*. With this delta Δ we calculate the distances between the parallel shifted traversable vertices and the blocker plane by adding the delta to the distances between traversable vertices and blocker plane. The savings

are that so only one traversable vertex have to be parallel shifted, because we do not need the positions of the parallel shifted vertices for the further calculations. This can be done similar for the computation of the distances between blocker vertices and the slicing plane.

We denote that $d_{\mathbf{v}_i^t}$ is the distance of the ith traversable vertex $\mathbf{v}_i^t$ and that $d_{\mathbf{v}_i^{t\prime}}$ is the distance of the ith parallel shifted traversable vertex to the plane in which the blocker lies. Analog $d_{\mathbf{v}_i^b}$ is the distance of the ith blocker vertex to the traversable plane and $d_{\mathbf{v}_i^{b\prime}}$ the current slicing plane. So the delta distance Δ can also be calculated with a saved distance of the traversable vertex to the blocker plane by

$$\Delta = d_{\mathbf{v}_i^t} - d_{\mathbf{v}_i^{t\prime}} .$$

6.4 Polygon-Plane Intersection Cases

Polygon-Plane Intersection Cases describe the different ways in which a polygon can lie in space relative to a plane. In the following list all different cases are listed and described:

- `allZero` means that the polygon lies in the plane.
- `allNegative` stands for the fact that the polygon lies under the plane.
- `allPositive` is the opposite, meaning the polygon lies over the plane.
- `oneZeroNegative` means that the polygon lies under the plane, but one point lies on the plane.
- `oneZeroPositive` implies that the polygon lies over the plane, but one point lies on the plane.
- `multipleZeroNegative` is the case, where two or more consecutive points of the polygon lie in the plane and the other points lie below the plane.
- `multipleZeroPositive` is analog to the one above, with the difference that the other points of the polygon are above the plane.

- `twoZero` stands for the case that two not adjacent points of the polygon lie in the plane, so there have to be points above and below the plane.
- `oneZeroOneSignSwitch` means that one point of the polygon lies in the plane and one edge intersects the plane, the other points have to lie above and below the plane.
- `twoSignSwitch` implies that there are two edges which intersect the plane and some points lie above and some below the plane.

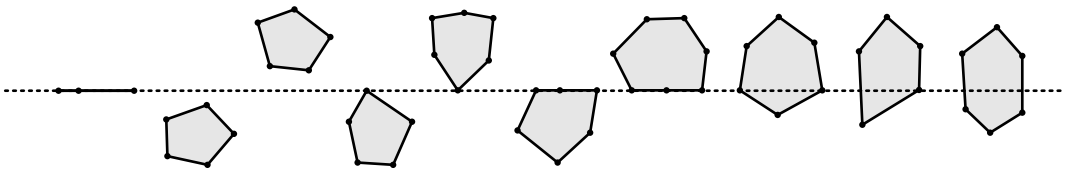


Figure 12: *Polygon against plane intersection cases, the plane is indicated by the dotted line*
f.l.t.r.: `allZero`, `allNegative`, `allPositive`, `oneZeroNegative`, `oneZeroPositive`, `multipleZeroNegative`, `multipleZeroPositive`, `twoZero`, `oneZeroOneSignSwitch` and `twoSignSwitch`

These different cases are used to determine if there is an intersection possibility or not. If all blocker vertices have a negative distance to the traversable plane or lie in the plane this means that the blocker lies below the traversable plane and therewith below the traversable. This implies that there is no surface of the traversable obstructed by this blocker and we do not have to do anything with this traversable blocker pair. In this case the blocker case is `allNegative`, `oneZeroNegative` or `multipleZeroNegative`. This is one of the two fast rejects. The other one is that the traversable is above the blocker, so the traversable case is `allPositive`, `oneZeroPositive` or `multipleZeroPositive`. If the traversable case is for example `allPositive` this does not imply that the blocker case is `allNegative` (see Figure 13).

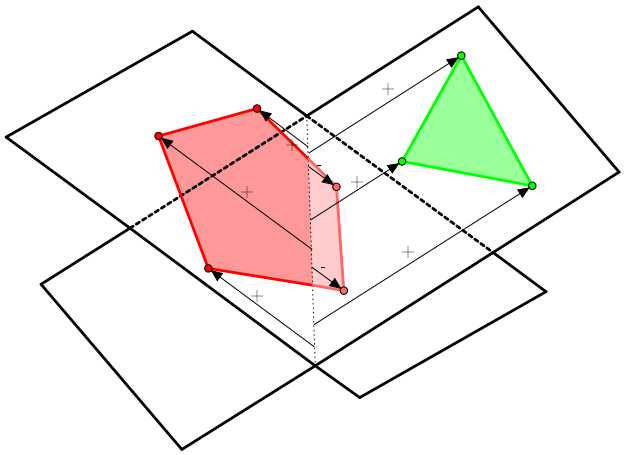


Figure 13: *Blocker case and traversable case sketch*
The traversable is visualized in green and the blocker in red and for both polygons the planes are sketched in which they lie. The arrows from the intersection line of these two planes to the vertices of the polygons are labeled with a "+" if the distance between the vertex and the plane is positive. It is labeled with a "-" if the distance between the vertex and the other plane is negative.

There is another class of cases where the blocker is above the traversable plane and the traversable is below the blocker plane, these are the blocker cases `allPositive`, `oneZeroPositive` and `multipleZeroPositive` and the traversable cases `allNegative`, `oneZeroNegative` or `multipleZeroNegative`. In these cases we know that there is no intersection yet, but there could be an intersection in the next slice, so we fast forward to the calculation for the next stance.

The blocker case `allZero` and the traversable case `allZero` means that the blocker and the traversable plane lie in the same plane, so they are coplanar. Because it is checked if the traversable and blocker are coplanar at the beginning of the VCD, this state is not reachable.

The other cases are used in further calculations. An essential fact is that during the determination of the case we also save information about the edges, which intersect the plane of the other polygon, which will be used to calculate the actual intersection points.

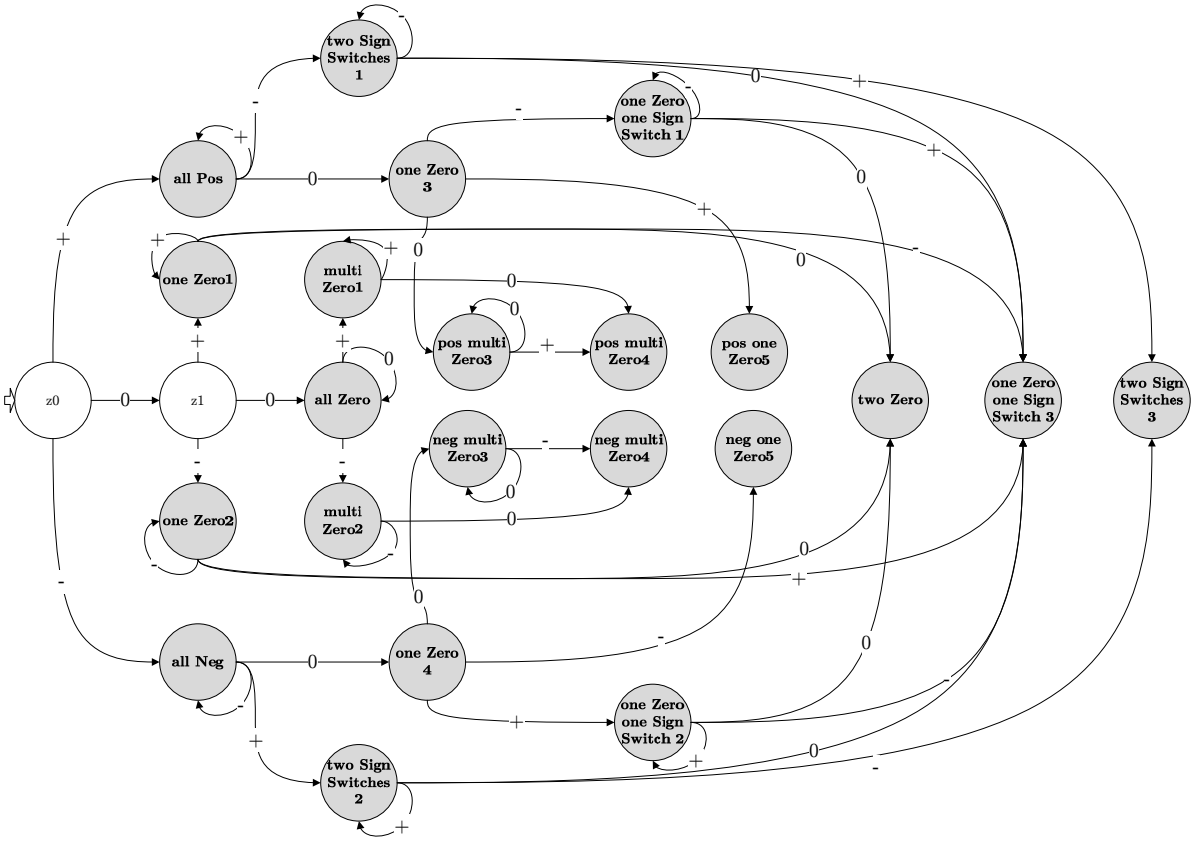


Figure 14: *Automaton to determine intersection cases*

The polygon-plane intersection cases are determined with an automaton (see Figure 14) which iterates through all vertices of a polygon with their distances. If a vertex has a distance of zero to the plane, this vertex will be saved as plane intersecting. If an edge intersects the plane, which happens if the sign of distances changes, then one point is below and the other point is above the plane. This edge will be saved as well. So after the automaton run determined which intersection case this polygon has and which vertices or which edges intersect the plane. This information is used in further calculations to determine the intersection points.

As described in the paragraph above, the polygon-plane intersection cases were just described between the blocker and the traversable. This can be generalized to blocker and slicing plane or parallel shifted traversable and blocker plane. The only thing that has to be done is the calculation of the distances between the appropriate polygon and the appropriate plane and then use these for the polygon-plane intersection case determination.

6.5 Blocker-Traversable Plane Intersection Point Calculation

We calculate the intersection points between the blocker and the slicing plane, because they define the line at which the blocker is cut along, and the intersection points are the new points that we need for the construction of the lower and upper blocker as result polygons.

Assume that the slicing plane and the blocker do not intersect, the blocker slicing would have rejected this situation with the help of the polygon-plane intersection case, which has been described above (see subsection 6.4). So if we calculate the intersection between the blocker and the slicing plane, we know that the blocker plane and slicing plane do intersect each other.

If two planes intersect each other, we know that the intersection is a line. We denote the intersection line between the blocker plane and the slicing plane with L. As any other line this line L can be described by

$$L = \mathbf{x}_0 + t \cdot \mathbf{v}\,,$$

where $\mathbf{x_0}$ is a point on the line, $\mathbf{v}$ is a directional vector and t is a real number. The direction vector of the line L can be calculated with

$$\mathbf{v} = \hat{\mathbf{n}}_b \times \hat{\mathbf{n}}_t\,,$$

where $\hat{\mathbf{n}}_b$ is the unit normal vector of the blocker polygon and plane, and $\hat{\mathbf{n}}_t$ is the unit normal vector of the traversable, which is also the unit normal vector of the slicing planes, that are parallel shifted to the traversable plane. The intersection points between the blocker and the slicing plane have to lie on the line L, so they can be described by the scalars t_3 and t_4. To calculate the intersection points we also need to know a point $\mathbf{x_0}$ on the line.

If one of the polygon-plane intersection cases indicates a distance of zero to the plane of the other polygon, it means that we can use this point as the point on the intersection line L. So only if the blocker case and the traversable case are `twoSignSwitches`, we have to calculate a point on the intersection line (see Table 1 where no scalar is set to zero).

To compute the point on the line $\mathbf{x_0}$, we perform a line-plane intersection calculation. The edge, that intersects the intersection line, is the line from the line-plane intersection test, which is given by the vertex $\mathbf{v}_i^t$ and the vertex $\mathbf{v}_{i'}^t$. Furthermore the plane in the line-plane intersection test is the plane, in which the polygon lies, which does not include the edge. This intersection has to lie on the intersection line because the intersection line L is defined so that it is the intersection of the blocker plane and the traversable plane and the edge lies in one of these planes. The intersection point is calculated like described

in reference [Pap03, on page 65] by

$$\mathbf{x}_0 = \mathbf{v}_i^t + \frac{\hat{\mathbf{n}}_b \cdot (\mathbf{v}_\star^b - \mathbf{v}_i^t)}{\hat{\mathbf{n}}_b \cdot (\mathbf{v}_{i'}^t - \mathbf{v}_i^t)} \cdot (\mathbf{v}_{i'}^t - \mathbf{v}_i^t) \ ,$$

where $\mathbf{v}_\star^b$ is any vertex of the blocker.

Now we have a point on the intersection line L and we can calculate the scalars t_3 and t_4, which define the intersection points between the blocker and the traversable plane.

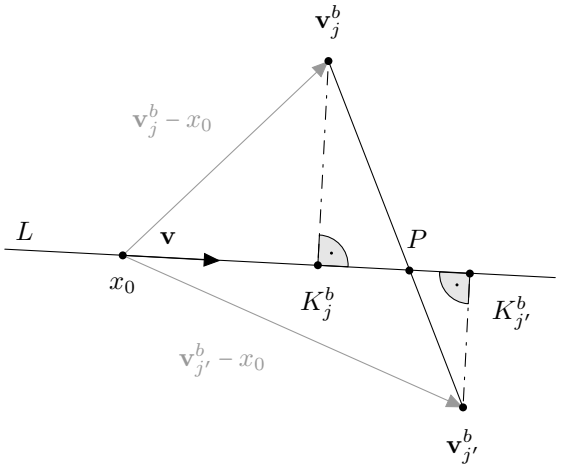


Figure 15: *Calculation of the intersection line direction vector scalar for the intersection point P of a polygon edge with the plane in which the other polygon lies*

If we have a point on the line and the line directional vector, we can simply calculate the scalar to a given point on the same line. This can be done by calculating the length of the vector between the point on the line and the point to which we want to calculate the scalar and divide this by the length of the line directional vector.

In the situation that a blocker edge with the vertex $\mathbf{v}_j^b$ and the vertex $\mathbf{v}_{j'}^b$ intersects the

intersection line L, we can compute the scalar as it is described in reference [Möl97].

$$t_4 = p_{\mathbf{v}_j^b} + \left(p_{\mathbf{v}_{j'}^b} - p_{\mathbf{v}_j^b}\right) \cdot \frac{d_{\mathbf{v}_j^b}}{d_{\mathbf{v}_j^b} - d_{\mathbf{v}_{j'}^b}},$$

where vertex $\mathbf{v}_j^b$ is projected onto L with $p_{\mathbf{v}_j^b} = \mathbf{v} \cdot \left(\mathbf{v}_j^b - \mathbf{x}_0\right)$ and as above $\mathbf{v}$ is the directional vector of the intersection line, $\mathbf{x}_0$ is a point on the intersection line L and $d_{\mathbf{v}_j^b}$ is the distance of the blocker vertex $\mathbf{v}_j^b$ to the slicing plane (see Figure 15).

So we have two different methods of how the scalars can be computed depending on the traversable intersection case and the blocker intersection case. Which method we use for which combination of blocker and traversable intersection cases can be looked up in Table 1.

traversable case \ blocker case	`oneZeroNegative` `oneZeroPositive`	`multipleZeroNegative` `multipleZeroPositive` `twoZero`	`OneZeroOneSignSwitch`	`twoSignSwitches`
`oneZeroNegative` `oneZeroPositive`	$t_1 = 0$ $t_2 = t_1$ $t_3 = \mapsto$ $t_4 = t_3$	$t_1 = 0$ $t_2 = t_1$ $t_3 = \mapsto$ $t_4 = \mapsto$	$t_1 = 0$ $t_2 = t_1$ $t_3 = \mapsto$ $t_4 = \bowtie$	$t_1 = 0$ $t_2 = t_1$ $t_3 = \bowtie$ $t_4 = \bowtie$
`multipleZeroNegative` `multipleZeroPositive` `twoZero`	$t_1 = 0$ $t_2 = \mapsto$ $t_3 = \mapsto$ $t_4 = t_3$	$t_1 = 0$ $t_2 = \mapsto$ $t_3 = \mapsto$ $t_4 = \mapsto$	$t_1 = 0$ $t_2 = \mapsto$ $t_3 = \mapsto$ $t_4 = \bowtie$	$t_1 = 0$ $t_2 = \mapsto$ $t_3 = \bowtie$ $t_4 = \bowtie$
`OneZeroOneSignSwitch`	$t_1 = 0$ $t_2 = \bowtie$ $t_3 = \mapsto$ $t_4 = t_3$	$t_1 = 0$ $t_2 = \bowtie$ $t_3 = \mapsto$ $t_4 = \mapsto$	$t_1 = 0$ $t_2 = \bowtie$ $t_3 = \mapsto$ $t_4 = \bowtie$	$t_1 = 0$ $t_2 = \bowtie$ $t_3 = \bowtie$ $t_4 = \bowtie$
`twoSignSwitches`	$t_1 = \bowtie$ $t_2 = \bowtie$ $t_3 = 0$ $t_4 = t_3$	$t_1 = \bowtie$ $t_2 = \bowtie$ $t_3 = 0$ $t_4 = \mapsto$	$t_1 = \bowtie$ $t_2 = \bowtie$ $t_3 = 0$ $t_4 = \bowtie$	

Table 1: *Scalar computation by length calculation or line-line intersection calculation depending on the intersection cases*

The scalar t_1 and t_2 are the scalars which define the two intersection points of the traversable and the blocker plane, analog for the scalar t_3 and t_4. $\mapsto$ means length calculation and $\bowtie$ means line-line intersection calculation.

The computation via the scalar is described by reference [Möl97]. We decided to implement it, so that we can compute the intersection points between two polygons in three-dimensional space.

6.6 Blocker Slicing in Lower Blocker and Upper Blocker

The actual blocker slicing in lower and upper blocker, by the slicing plane, can now be performed, because we know the intersection point or intersection points. If there is only one intersection point, this means that the blocker touches the slicing plane, but does not pierce through. In this case the lower blocker or the upper blocker is equal to the blocker and the other part of the blocker does not exist. If there are two intersection points, the blocker pierces through the slicing plane or an edge of the blocker lies in the slicing plane. The second case has the same result as described for one intersection point. If the blocker is piercing through the slicing plane, then the blocker is cut into lower blocker and upper blocker. Both consist only of vertices of the blocker and the two intersection points, but non of them contains all blocker vertices.

There are two slightly different ways in which the Blocker Slicing is used in the VCD. The first occurrence is to cut off the blocker beneath the traversable plane. In this situation the slicing plane is the traversable plane, the lower blocker is discarded and the upper blocker is used for the further obstruction computations.

The next occurrence in the VCD is in "cut across blocker in upper blocker and lower blocker" (see Figure 6 state 6), where the blocker is sliced by a slicing plane which is defined by the traversable and the current stance. The resulting lower blocker is used in the Traversable Decomposition to determine the part of the traversable which is obstructed for a stance by the blocker. The upper blocker is used as blocker in the obstruction computation for the next stance.

7 Traversable Decomposition

7.1 Definition

The Traversable Decomposition (TD) is the subdivision of a current traversable into one new blocker and zero or more new traversables.

$$currentTraversable = newBlocker \cup \bigcup_{i \in \{1,\dots,n\}} newTraversables_i$$

The new blocker is the part of the current traversable that is obstructed by the lower blocker. The lower blocker is a slice from the initial blocker of the VCD which represents one specific stance and is handed down into the Traversable Decomposition. From now on the vertical projection of the lower blocker in the traversable plane is just called projected lower blocker.

$$newBlocker = currentTraversable \cap projectedLowerBlocker$$

The unobstructed part of the current traversable surface can, in most cases, not be represented by one convex polygon, so it has to be decomposed into a number of convex new traversables, representing the unobstructed traversable surface.

$$\bigcup_{i \in \{1,\dots,n\}} newTraversables_i = currentTraversable \setminus projectedLowerBlocker$$

7.2 Algorithm

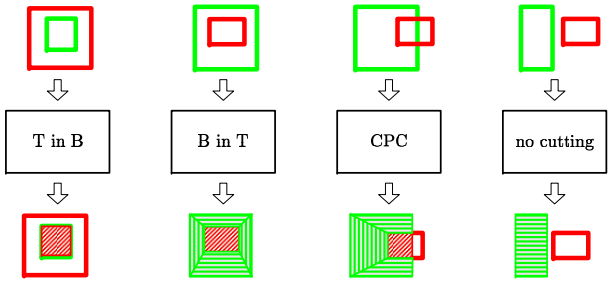


Figure 16: *Traversable Decomposition Cutting Cases*

Two coplanar convex polygons have four different cases how they could intersect. Here these polygons are the current traversable and the projected lower blocker. The first two cases are that one polygon is totally contained in the other polygon. So the first case is that the current traversable is totally contained in the projected lower blocker. The second case is that the projected lower blocker is totally contained in the current traversable. The next case is that there is a non empty intersection between both polygons and non of the polygons is totally contained in the other polygon. Finally there is the case where there is no intersection between these two polygons.

The TD starts with determining the intersection between current traversable and projected lower blocker. This intersection has to be one of the four cases mentioned above. Depending on the case, we use a special cutting method to decompose the current traversable into an obstructed part and a convex decomposed rest (see Figure 16). Of course if there is no intersection between the current traversable and the projected lower blocker there is no need for any kind of cutting.

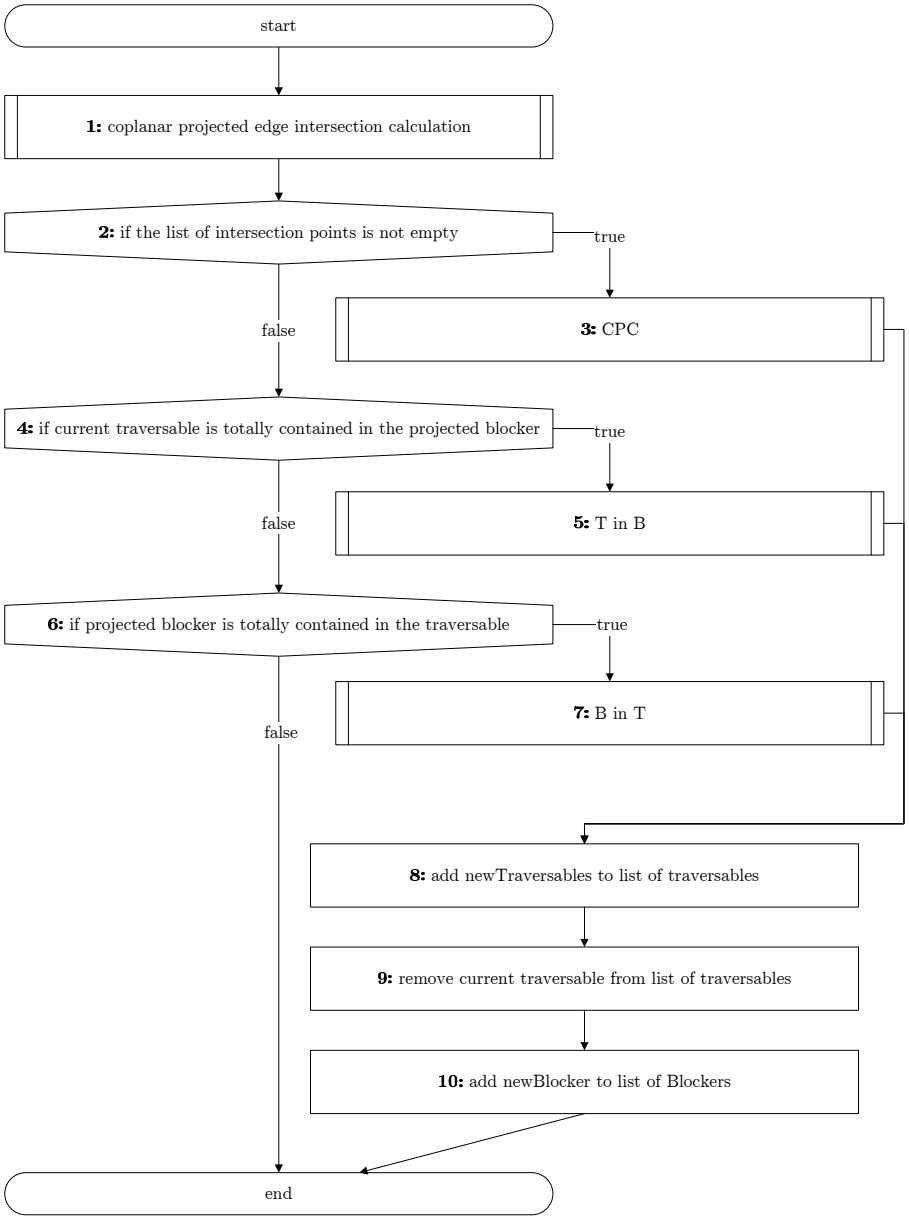


Figure 17: *Traversable Decomposition*

We use the Projected Outline Intersection, which is described in detail in subsection 7.3, to get a list of edge intersection points (Figure 17 state 1). If we do not find any edge intersection points, we have to identify which of the three possible cases we have to process (Figure 17 state 2).

Either the current traversable is totally contained in the projected lower blocker (Figure 17 state 5) or the projected lower blocker is totally contained in the current traversable (Figure 17 state 7). If both cases are false, then we know that the lower blocker is not obstructing the current traversable and stop the TD.

To test if the current traversable is totally contained in the projected lower blocker, we use a point inside polygon test which is implemented in two-dimensional space. For the test we simply ignore the z-coordinate of the current traversable and the lower blocker. Then we test for one vertex of the traversable if it lies inside the lower blocker. Testing this for only one vertex is enough, because we know there are no edge intersections, which implies that this test has the same outcome for all vertices of the current traversable.

If the vertex lies inside the projected lower blocker, we know the traversable is totally contained in the projected lower blocker (Figure 17 state 5). This case is described in subsection 7.4.

If the vertex does not lie inside the projected lower blocker, we have to test for a vertex of the projected lower blocker if it lies inside the current traversable. If the vertex of the projected lower blocker lies outside of the current traversable, then the lower blocker does not obstruct the current traversable at all.

If the vertex of the projected lower blocker is tested positively for lying inside the current traversable, the projected lower blocker must be totally contained in the current traversable (Figure 17 state 7). That case is explained in subsection 7.5.

If we find an Intersection Point (IP) (Figure 17 state 2), the actual cutting and construction of the new polygons will be done in the subsection 7.6 (Figure 17 state 3).

Regardless of the case we processed, the result polygons are treated the same way as covered below. The new traversables will be added to the list of traversables of the section 5 (Figure 17 state 8). The current traversable is now obsolete, because its surface is now represented by the new traversables and the new blocker, so it is removed from the list of traversables (Figure 17 state 9). If a new blocker was created and if the initial traversable and the initial blocker are not coplanar, the new blocker is added to the list of blockers of the section 5 (Figure 17 state 10).

7.3 Projected Outline Intersection

7.3.1 Definition

The Projected Outline Intersection calculates the outline intersections between the current traversable and the projected lower blocker.

First both polygons will be projected into the horizontal-plane, so the following coplanar polygon intersection calculation has a better performance. Every projected edge of the current traversable will perform a line-line intersection calculation against every projected edge of the projected lower blocker, which is described in detail in subsubsection 7.3.2. If an intersection is found, we save it as an Intersection Point (IP). What IPs are and how they differ from intersections will be explained in subsubsection 7.3.3. These IPs have to be ordered clockwise on the traversable, which will happen automatically, because we check one edge of the traversable against all edges of the projected lower blocker before we take the next traversable edge. So we only have to check the ordering if two intersections are found on one traversable edge. This is done by a distance calculation from the counterclockwise vertex of the traversable to the IPs. The IP that has a higher distance lies clockwise from the IP with the lower distance.

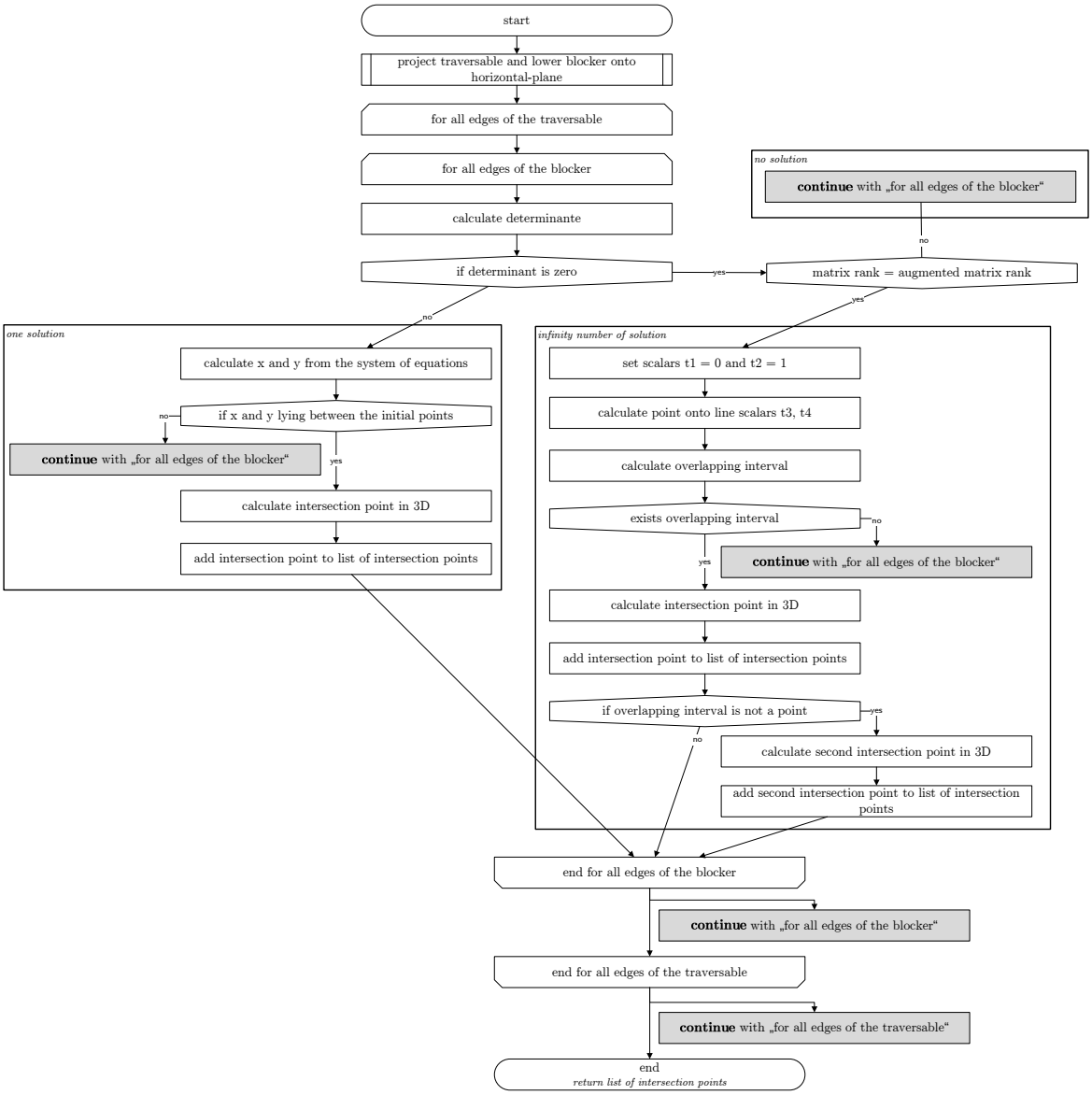


Figure 18: *Projected Outline Intersection*

7.3.2 Line-Line Intersection

Let the projected edge of the current traversable have the start (x_1, y_1) and the end (x_2, y_2) and the projected edge of the lower blocker have the start (x_3, y_3) and the end (x_4, y_4). We first consider these line segments as lines and calculate the intersection point, after this we check if this intersection point lies on both of these line segments. The two lines given by the projected polygon edges can intersect, be parallel or lie on each other. If they intersect, let (x, y) be the intersection point.

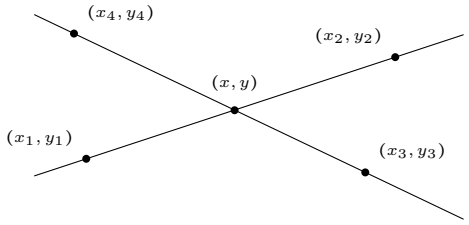


Figure 19: *Sketch of the edge-edge intersection situation*

The line given by the projected traversable edge can be described as

$$\left\{ \begin{pmatrix} x' \\ y' \end{pmatrix} \in \mathbb{R}^2 \middle| \begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} + \alpha \cdot \left(\begin{pmatrix} x_2 \\ y_2 \end{pmatrix} - \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} \right) \wedge \alpha \in \mathbb{R} \right\}.$$

Analogue the projected line given by the lower blocker edge can be described as

$$\left\{ \begin{pmatrix} x' \\ y' \end{pmatrix} \in \mathbb{R}^2 \middle| \begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x_3 \\ y_3 \end{pmatrix} + \beta \cdot \left(\begin{pmatrix} x_4 \\ y_4 \end{pmatrix} - \begin{pmatrix} x_3 \\ y_3 \end{pmatrix} \right) \wedge \beta \in \mathbb{R} \right\}.$$

If there is an intersection point, it has to lie on both lines, which implies that the point has to fulfill both line equations. The resulting system of linear equations for the intersection point of these lines is

$$\left| \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} + \alpha \cdot \left(\begin{pmatrix} x_2 \\ y_2 \end{pmatrix} - \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} \right) = \begin{pmatrix} x_3 \\ y_3 \end{pmatrix} + \beta \cdot \left(\begin{pmatrix} x_4 \\ y_4 \end{pmatrix} - \begin{pmatrix} x_3 \\ y_3 \end{pmatrix} \right), \right.$$

$$\Leftrightarrow \left| \begin{pmatrix} x_2 - x_1 & x_4 - x_3 \\ y_2 - y_1 & y_4 - y_3 \end{pmatrix} \cdot \begin{pmatrix} \alpha \\ -\beta \end{pmatrix}^{\top} = \begin{pmatrix} x_1 - x_3 \\ y_1 - y_3 \end{pmatrix}, \right.$$

which can be rewritten, like any other equation system, as $\mathbf{A} \cdot \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \mathbf{c}$.

The coefficient determinant

$$D = \det \mathbf{A} = \begin{vmatrix} x_2 - x_1 & x_4 - x_3 \\ y_2 - y_1 & y_4 - y_3 \end{vmatrix} = \begin{vmatrix} x_1 - x_2 & y_1 - y_2 \\ x_3 - x_4 & y_3 - y_4 \end{vmatrix}$$

is needed to know if there is one solution for the system of equations, an endless number of solutions or no solution. If the coefficient determinant is not zero, there is exactly one solution, which means the lines intersect each other, which does not necessarily mean that the line segments intersect. Otherwise if the determinant is zero, we have to compare the rank of the matrix $\mathbf{A}$ to the rank of the augmented matrix $\mathbf{A}|\mathbf{c}$.

If the ranks are not equal, then there is no solution of the equation system, which means that the projected edges are parallel and this means there is no intersection, so we can continue with the next lower blocker edge.

If the ranks are equal, we have an endless number of solutions of the equation system, which means that the lines are equal. So the line segments are the same, are overlapping each other, are included or are separate. But we know that the four points $(x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_4)$ are collinear. So we can calculate the overlapping line segment, if it exists, by using $(x_2, y_2) - (x_1, y_1)$ as directional vector and calculate the scalars t_3 and t_4 for the points (x_3, y_3) and (x_4, y_4). So that the scalar t_3 holds

$$\begin{pmatrix} x_3 \\ y_3 \end{pmatrix} = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} + t_3 \cdot \begin{pmatrix} x_2 - x_1 \\ y_2 - y_1 \end{pmatrix}$$

and the analog equation holds for t_4. Determine the overlapping interval, overlapping point or disjoint of the interval $[0, 1]$ and the interval $[t_3, t_4]$. If the two intervals are

disjoint, there is nothing to be done and we continue with the next blocker edge. In the case that there is an overlapping point we create an IP and insert it into the list of already found IPs. The two points which define the overlapping are the points we create IPs for and add them to the other IPs.

In the case that the coefficient determinant D is zero, which means there is not more than one intersection, we calculate the solution of the equation system as described in [Weib] by

$$x = \frac{\begin{vmatrix} \begin{vmatrix} x_1 & y_1 \\ x_2 & y_2 \end{vmatrix} & x_1 - x_2 \\ \begin{vmatrix} x_3 & y_3 \\ x_4 & y_4 \end{vmatrix} & x_3 - x_4 \end{vmatrix}}{\begin{vmatrix} x_1 - x_2 & y_1 - y_2 \\ x_3 - x_4 & y_3 - y_4 \end{vmatrix}} \qquad y = \frac{\begin{vmatrix} \begin{vmatrix} x_1 & y_1 \\ x_2 & y_2 \end{vmatrix} & y_1 - y_2 \\ \begin{vmatrix} x_3 & y_3 \\ x_4 & y_4 \end{vmatrix} & y_3 - y_4 \end{vmatrix}}{\begin{vmatrix} x_1 - x_2 & y_1 - y_2 \\ x_3 - x_4 & y_3 - y_4 \end{vmatrix}} .$$

The denominator determinant is the coefficient determinant D, which has already been computed. On the other hand the numerator determinant has to be calculated, by which we do not have to calculate α or β explicitly.

Now that we know (x, y), we have to check if (x, y) lies on both projected edges. If so, then we can create an IP for this intersection and insert it into the list of already found IPs. Then we continue with the next projected lower blocker edge.

This check is done by a componentwise comparison as follows

$$x \geq \min(x_1, x_2) \wedge x \leq \max(x_1, x_2) \wedge x \geq \min(x_3, x_4) \wedge x \leq \max(x_3, x_4) \ .$$

If this formula is satisfied for the x-component and the y-component, then the intersection point (x, y) lies on the line segment defined by (x_1, y_1) and (x_2, y_2) and also lies on the line segment defined by (x_3, y_3) and (x_4, y_4).

7.3.3 Intersection Point Definition

Intersection Points (IPs) are more than just points in three-dimensional space. They consist of a type, pointers and a point. The pointers point to the vertices of the edges that are intersecting.

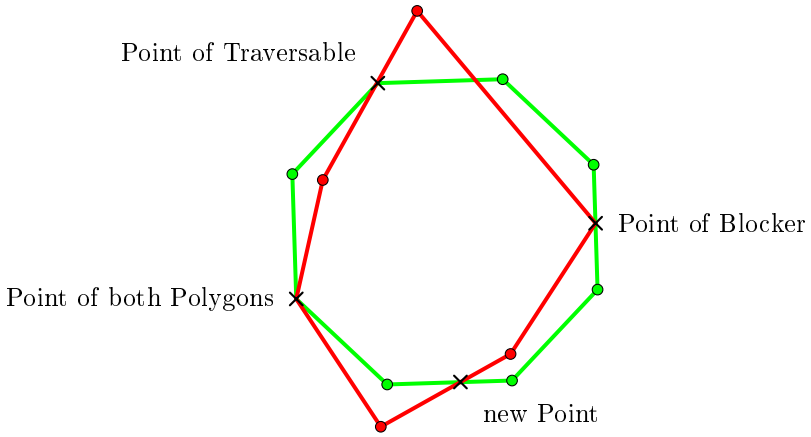


Figure 20: *Intersection Point types*

There are four different types of IPs (see Figure 20). The first is the type "new point", which means that the IP is not part of the current traversable nor the projected lower blocker. There is the type "point of traversable", which means the IP is a vertex of the current traversable. If the IP has the type "point of blocker", this means that the IP is part of the lower blocker or at least part of the projected lower blocker. The last type is "point of both polygons", which means that it is part of the current traversable and the projected lower blocker.

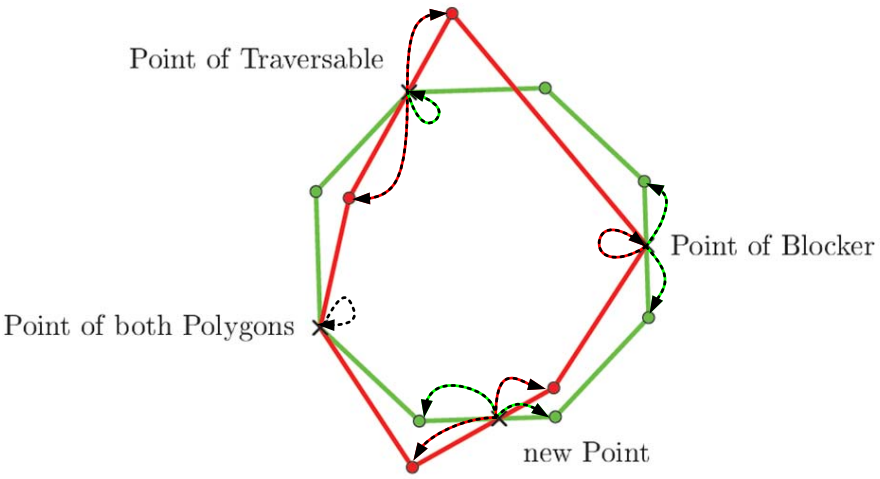


Figure 21: *Different Intersection Points and the vertices which are linked to each Intersection Point*

The pointers of the IPs point to vertices of the current traversable and the projected lower blocker that are relevant for the sorting of the IP into these polygons. If an IP is a "point of both polygons", then there are two pointers. One of the pointers points to a vertex of the current traversable, which has the same position as the IP. The other pointer points to a vertex of the lower blocker, which projection in the traversable plane has the same position as the IP. Is the IP a "point of traversable", then there are three pointers. There is one pointer that points to a vertex of the current traversable, which has the same position as the IP. The two other pointer are connected with the two vertices of the lower blocker, which form the edge that has caused this IP. Analogue if the IP is a "point of blocker", this means there is one pointer to a lower blocker vertex and there are two pointers to one edge of the current traversable. An IP which has the type "new point" has four pointers. Two of them pointing to an edge of the current traversable and the other two pointing to an edge of the lower blocker. These two edges are the edges which have caused this IP.

7.4 Traversable Totally Contained in Blocker

This case is pretty simple to handle, because we know that the Traversable is completely obstructed, so we do not need to alter any geometry. Instead we just change the current traversable to be a new blocker.

In case that the current traversable and the lower blocker are coplanar, then the new blocker would be a redundant piece of obstruction geometry. So in the case of coplanarity, the current traversable is simply deleted and no new geometry is constructed.

7.5 Blocker Totally Contained in Traversable

7.5.1 Definition

In the case that the current traversable and the lower blocker are coplanar, no new blocker has to be created, because it would be a redundant piece of obstruction geometry. This is very similar to the coplanar handling in subsection 7.4.

If the current traversable and the lower blocker are not coplanar, the new blocker is very easy to construct. It is simply the lower blocker projected into the current traversable plane.

The real problem here is that the current traversable is definitely not convex when the lower blocker is subtracted from it. Specifically, because we know there are no edge intersections, the current traversable without the lower blocker is always a polygonal ring structure.

As defined early the VCD requires convex polygons to work. Because of that the ring structure has to be decomposed into the smallest possible number of convex new traversable polygons. This process is described in detail in subsubsection 7.5.3, with the preliminary selection of start vertices which is explained in subsubsection 7.5.2.

Because we know that both polygons are convex we can conclude that a convex decomposition of the polygonal ring structure can be done very efficiently by just incrementally building polygons until the ring is completely represented through new convex polygons.

7.5.2 Start Vertex Selection

To start the Convex Decomposition, we need to select one start vertex v_t from the current traversable and one start vertex v_b from the projected lower blocker in that way, that the edges of the triangle $\triangle(v_b, v_t, (v_b \oplus 1))$ do not intersect with any of the edges of the current traversable and/or the projected lower blocker. Only then it is guaranteed, that there is a clean starting triangle with which the Convex Decomposition can set off.

The triangle $\triangle(v_b, v_t, (v_b \oplus 1))$ does not intersect with any of the edges of the current traversable, because the projected lower blocker is totally contained in the projected current traversable. So it just has to be tested if the triangle $\triangle(v_b, v_t, (v_b \oplus 1))$ does not intersect with any edge of the projected lower blocker. Because the lower blocker is a convex polygon and not a vertical blocker, the lower blocker, which is projected into the plane of the traversable, is also a convex polygon. That is why we know that the edge between the vertices $(v_b, (v_b \oplus 1))$ does not intersect any other edge of the projected lower blocker. So we just have to check that the edge (v_b, v_t) and the edge $(v_t, (v_b \oplus 1))$ do not intersect any edge of the projected lower blocker. This is done by a simplicity check of the polygon $\bigcirc(v_b, v_t, (v_b \oplus 1), \ldots, (v_b \ominus 1))$. If this polygon is simple, it does not self intersect. This means that the edge (v_b, v_t) and the edge $(v_t, (v_b \oplus 1))$ do not intersect any edge of the projected lower blocker.

We choose an arbitrary but fixed vertex v_b from the projected lower blocker. Then we iterate through the vertices of the current traversable $T_{vertices} \coloneqq [t_0, \ldots, t_{n_t-1}]$ and test for every vertex t_i of the current traversable if the polygon

$$\bigcirc P \coloneqq \bigcirc(v_b, t_i, (v_b \oplus 1), \ldots, b_{n_b-1}, b_0, \ldots, (v_b \ominus 1))$$

is simple, where $B_{vertices} \coloneqq [b_0, \ldots, b_{n_b-1}]$ are the projected lower blocker vertices. The polygon P is constructed from the projected lower blocker with the vertex i inserted between v_b and $v_b \oplus 1$. Testing this temporary polygon for simplicity is a sufficient test for the necessary conditions mentioned above. The first vertex t_i that fulfills the condition is chosen as v_t.

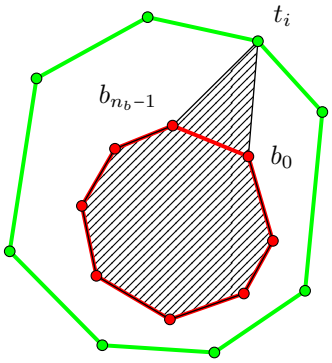


Figure 22: *Start vertex pair selection*

We reorder the vertex list of the current traversable's vertices so that is starts with the vertex v_t. The new vertex list of the current traversable is $[t_i, \ldots, t_{n_t-1}, t_0, \ldots, t_{i-1}]$. Analog we reorder the vertex list of the project lower blocker so that it starts with v_b and then the projected lower blocker vertices are $[b_{n_b-1}, b_0, \ldots, b_{n_b-2}]$. At this point we renumber the traversable vertices list and the blocker vertices list, in such a way that both list starts with the index 0 and the last current traversable list index is $n_t - 1$ and the last projected lower blocker vertex index is $n_b - 1$, wih the result that the traversable vertices list is $T_{vertices} = [t_0, \ldots, t_{n_t-1}]$ and the projected lower blocker vertices list is $B_{vertices} = [b_0, \ldots, b_{n_b-1}]$.

The Convex Decomposition (subsubsection 7.5.3) is started with $T_{vertices}$ and $B_{vertices}$.

7.5.3 Convex Decomposition

The goal of the Convex Decomposition is to decompose the surface of *current Traversable* \ *projected Blocker* into a minimal number of convex polygons.

The Convex Decomposition is started with two lists of vertices, one is the vertex list defining the projected lower blocker and the other is the vertex list defining the current traversable.

These lists are the input for the Convex Decomposition, so they are defined the same way as in subsubsection 7.5.2 and subsection 7.6:

- the traversable's list of vertices is defined as $T_{vertices} \coloneqq [t_i, \ldots, t_{n_t-1}, t_0, \ldots, t_{i-1}]$
- the blocker's list of vertices is defined as $B_{vertices} \coloneqq [b_{n_b-1}, b_0, \ldots, b_{n_b-2}]$

When we add vertices from the input polygons to the polygon in construction, we imply that moving through the vertex lists represents a clockwise movement along the polygon's edges. Furthermore it is implied that vertices are always added into the cyclic structure of the new polygon between the vertex last added from the traversable and the vertex last added from the blocker. This is as important as it is intuitively simple and will not be mentioned at every inserting operation reference in the following text.

The first triangle to be constructed is $\triangle T_0 = \triangle(b_j, t_i, b_{j+1})$ with $i = 0$ and $j = 0$ and this is known to not intersect with any edges of input polygons. Now we add as many vertices from the blocker to the triangle T_0 as possible. This means we incrementally add one vertex after another and check every time if the resulting polygon is still simple and convex. If it is either not simple any more or not convex any more, then the last added vertex is removed and no more blocker vertices will be added to this polygon.

For the blocker vertices, it is only possible that more than one vertex is added if the following vertices all lie on the same line as the line defined by b_j and b_{j+1}. Then as many traversable vertices as possible are added in the same way as for the blocker vertices. These vertices do not have to lie on one line. It makes no difference if the traversable vertices are added first or the blocker vertices, because the constraints of simpleness and convexity will be violated at the same vertices regardless of the order in which the vertices are added.

This is because the line L_j defined by b_j and b_{j+1} splits the space in two partitions, one we call inside which fully contains the blocker, and one we call outside which contains no blocker surface at all. As long as the added vertices are on the outside of the line L_j or on the line L_j the resulting polygon is convex. The line L_j is in that regard the convexity border for the adding of the vertices and this is not changed by the adding of more projected lower blocker vertices as mentioned above, so the condition for the convexity test stays the same.

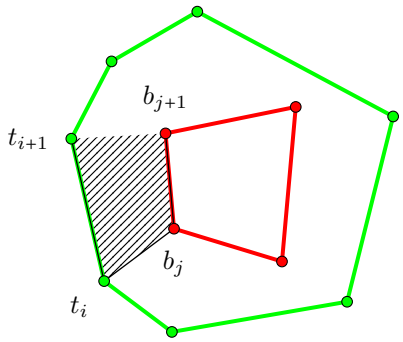


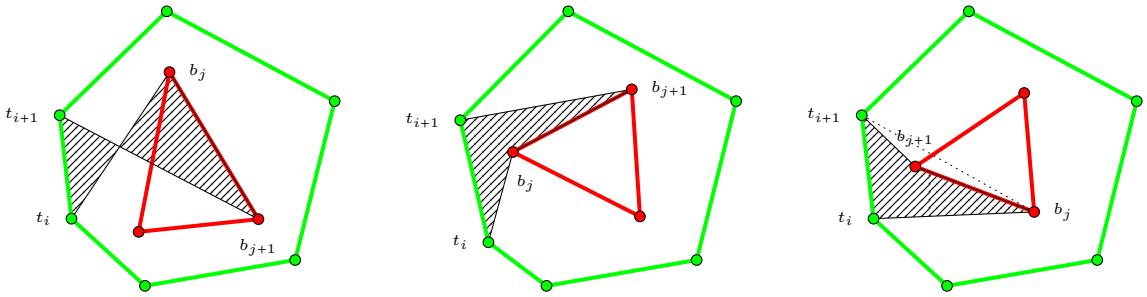
Figure 23: *Middle/Loop polygon simple and convex*

The last vertex of the projected lower blocker which has been added to the new polygon and the last vertex of the current traversable which has been added to the new polygon

are the two start vertices b_j and t_i for the following loop of polygon constructions of the Convex Decomposition.

The difference to the first polygon that got constructed is, that the triangle $\triangle(b_j, t_i, b_{j+1})$ is not guaranteed to be correct, it was just correct in the first construction because of the conditions for the preliminary start vertex calculation. In the loop the first thing that is done is to construct a quadrangle $Q = \square(b_{j+1}, b_j, t_i, t_{i+1})$, which is tested for simplicity and convexity.

If Q is simple and convex we proceed as with the first triangle and add as many blocker vertices as possible and then as many traversable vertices as possible. Then the loop continues with newly set j and i.



(a) *not simple*

(b) *not convex and* b_{j+1} *outside*

(c) *not convex and* b_{j+1} *inside*

Figure 24: *Middle/Loop polygon during convex decomposition which is not simple(1) and the other which is not convex(2) and (3) and in subfigure (3) is* b_{j+1} *inside the triangle* $\triangle(b_j, t_i, t_{i+1})$

If one of the tests fails the blocker vertex b_{j+1} is removed from the quadrangle $\square Q$ and we define that as triangle $\triangle Q'$. Then we test if b_{j+1} lies inside the triangle $\triangle Q'$, if it does, it means that the triangle $\triangle Q'$ is intersecting edges of the blocker (see Figure 24 (c)).

The test if b_{j+1} lies inside the triangle $\triangle Q'$ can be done by just checking the vertex b_{j+1} against the edge $t_{i+1}b_j$. Where an ordinary inside test, which has no further information, would have to check the vertex against all edges.

We know that either the triangle $\triangle Q' = \triangle(b_j, t_i, t_{i+1})$ or the $\triangle Q'' \coloneqq \triangle(b_{j+1}, b_j, t_i)$ is not intersecting the projected lower blocker in more points than the shared edge. So now that we have ruled out Q', we know that the other triangle Q'' does not intersect any blocker edges and thus is added to the list of new traversables and the loop continues.

If b_{j+1} lies outside the triangle $\triangle Q'$, then it does not intersect any edges of the blocker. Because the triangle Q' is obviously simple and convex, it is then added to the list of new traversables and the loop continues.

In the case that the quadrangle $\square Q$ is not simple or not convex it is relevant if the first vertex to be removed is t_{i+1} from the traversable or b_{j+1} from the projected lower blocker. The difference lies in the complexity of the necessary tests and is explained in more detail in the following paragraphs.

It is possible to implement it the way that not vertex b_{j+1} from the projected lower blocker is removed from the quadrangle $\square Q = \square(t_i, t_{i+1}, b_{j+1}, b_j)$, but the traversable vertex t_{i+1}. After the first created quadrangle $\square Q$ the triangle would be $\triangle Q'' = \triangle(b_{j+1}, b_j, t_i)$. Then it has to be checked that there is no line intersection between any line of the triangle $\triangle Q''$ and the projected lower blocker other than the line segment $b_j b_{j+1}$ itself. If there is no intersection than the polygon can be added to the list of new traversables. But if there is an intersection then there has to be a new polygon with two traversable vertices, which would be the triangle $\triangle Q' = \triangle(b_j, t_i, t_{i+1})$.

So there is a different complexity in the tests. The test we implemented tests the vertex in polygon condition and the alternate way needs to test if the line segment $t_i b_{j+1}$

intersects with any blocker edge. This can be done by testing if the triangle $\triangle Q''$ is clockwise defined or not, if it is counterclockwise defined in the two dimensional space, then it is implied that the triangle intersects with edges of the blocker. Therefore we decided to implement it the first way.

The loop is stopped when the newest constructed polygon contains either the vertex b_0 of the blocker or the vertex t_0 of the current traversable. In this case we are finished with one of the cyclic structures which makes the rest of the constructions much more simple.

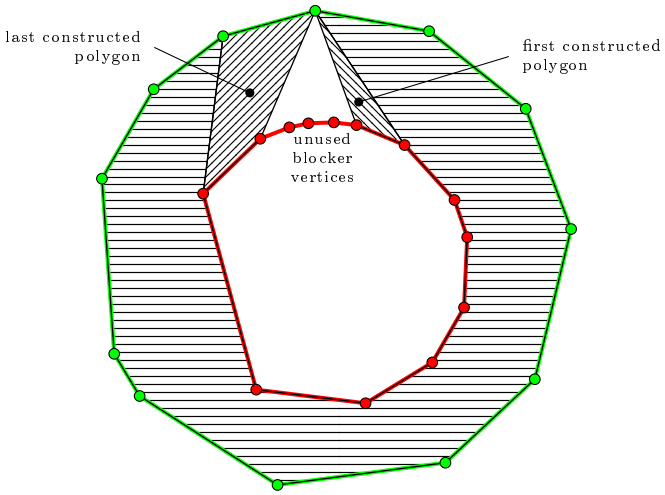


Figure 25: *Sketch of a situation, where blocker vertices are unused*

In case that the blocker's list of vertices is finished, the last polygon is created with the vertex b_0 and all the rest of the traversables with the vertex t_0 added as the last one (see Figure 25).

If the traversable's list of vertices is iterated to its end first, then there is the need for another loop, which is very similar to the one described above. Just that it does not start

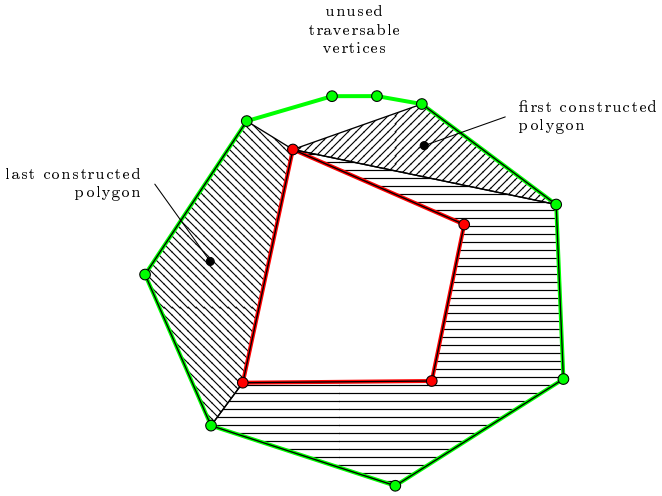


Figure 26: *Sketch of a situation, where traversable vertices are unused*

with a quadrangle, instead with the triangle $\triangle(b_{j+1}, b_j, t_i)$ (see Figure 26). Obviously no more vertices of the traversable are added to the polygon in construction. But the incremental addition of blocker vertices with the test for simplicity and convexity stays the same.

When all vertices of both the traversable and the blocker are consumed, the list of new traversables represents the surface that we wanted to be decomposed. The decomposition is near optimal, but the first and the last polygon that got created can in some cases be welded along their shared edge. That is because the first polygon is created in clockwise direction and is not expanded into counterclockwise as well. When the weld test is done, the list of new traversables is returned as the result of the Convex Decomposition.

All the vertex and list operations can be reviewed in more detail in Figure 27.

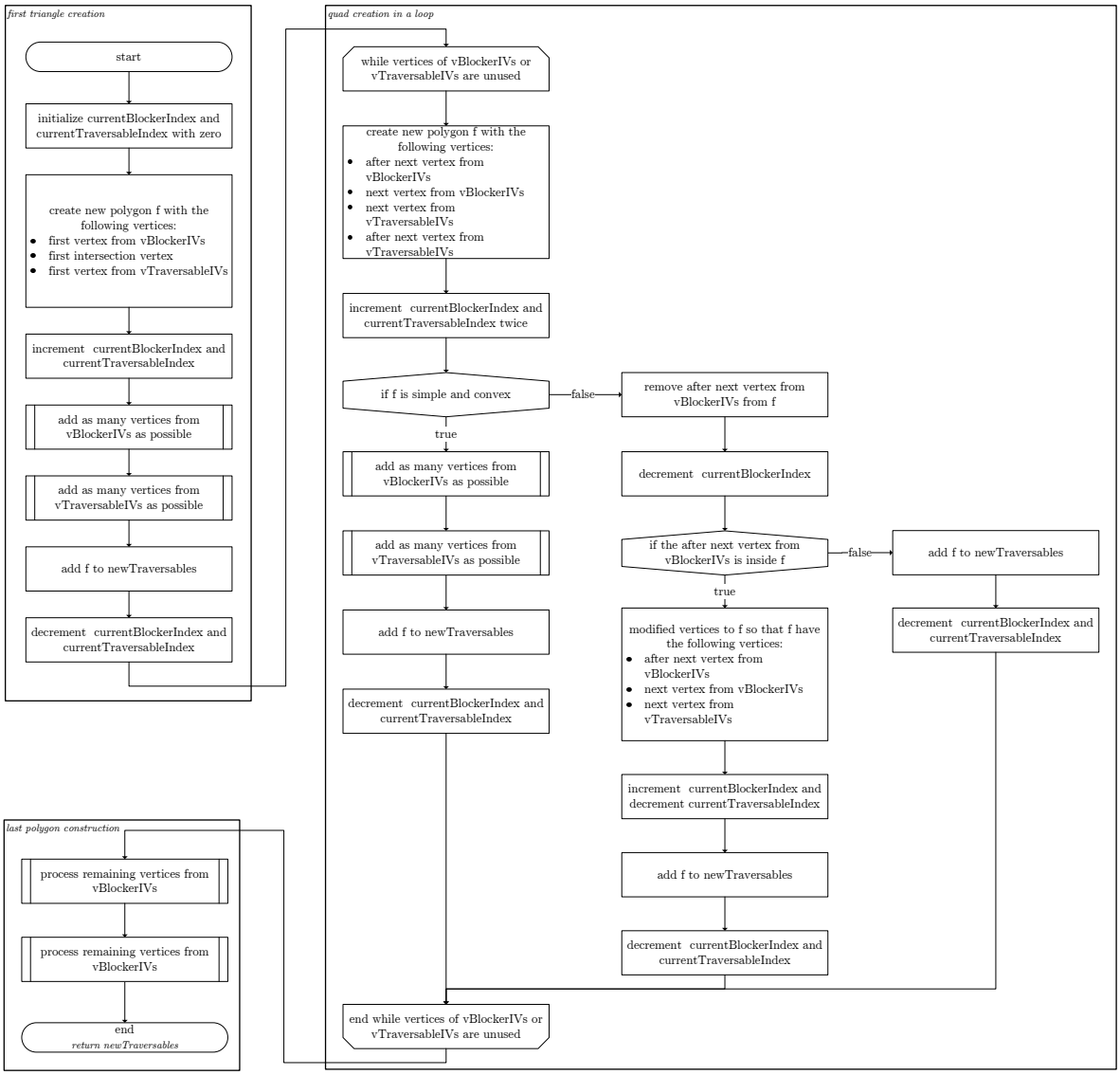


Figure 27: *Convex decomposition*

7.6 Coplanar Polygon Cutting

7.6.1 Intro/Overview

The task of the CPC is to construct the new Traversables and the new Blocker with the IPs of the current traversable and projected lower blocker as well as the current traversable and the projected lower blocker itself as input. Especially the CPC is constructing all the new polygonal data in linear complexity without the need for any sort of history cross checking of the constructed polygons. The vertices of the blocker and the traversable are only iterated once, and during this iteration all polygons are created. With the given IPs, the CPC only requires one geometric operation. Except that operation, everything else is just fast list operations.

7.6.2 Blocker and Traversable are Coplanar in 3D-Space

There is one case for the CPC where no new blocker is supposed to be created. Normally cutting of the current traversable is done on the bases of IPs that were calculated from a lower blocker that is somewhere in the obstructable space above the current traversable. So the obstructed part of the current traversable becomes a new blocker. But if the current traversable and the lower blocker are coplanar from the start, then the new blocker would be just equal to a part of the lower blocker, which means the new blocker would be redundant obstruction geometry, thus should never be created. If the current traversable and the lower blocker are coplanar, all operations on the new blocker that are described below are ignored and only the new traversables will be created.

7.6.3 Intermediate Vertices

The vertices between two IPs are called Intermediate Vertices (IVs). The IVs always either belong to the projected lower blocker or the current traversable, which will also be

referred to as blocker IVs or traversable IVs. The IVs are used by the CPC to construct the new traversables and the new blocker.

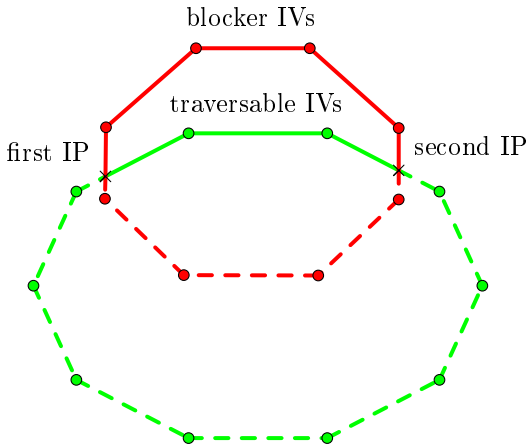


Figure 28: *Intermediate Vertices (IVs)*

The determination if IVs exist depends on the type of the IPs and of course the polygon we are looking at. An IP can be a vertex of the polygon we are looking at or not (see subsubsection 7.3.1). So there are four different situations of the two IPs.

Let IP_1 be the first Intersection Point and IP_2 be the second Intersection Point. Each IP is embedded in both polygons, so we say an IP has a previous vertex and a successor vertex, which is alway a vertex of the polygon. So if there are two IPs on an edge of a polygon, they have the same previous and successor vertex. The previous vertex of an IP is given by the function *prev* and the successor by *succ*. If an IP is part of the polygon, the previous and the successor of this IP are set to the IP itself.

In the case ① (of Table 2) where both IPs are part of the polygon we are looking at, there are no IVs if the two vertices of the IPs are adjacent and the first is clockwise directly before the second. Here applies that $prev(IP_1) = IP_1 = succ(IP_1)$ and the same for IP_2.

case	first IP	second IP	condition for no IVs
①	$\in$ poly	$\in$ poly	$IP_1 = IP_2 \ominus 1$
②	$\in$ poly	$\notin$ poly	$succ(IP_1) = IP_2$
③	$\notin$ poly	$\in$ poly	$IP_1 = prev(IP_2)$
④	$\notin$ poly	$\notin$ poly	$prev(IP_1) = prev(IP_2) \wedge$ $succ(IP_1) = succ(IP_2) \wedge$ $\lvert\overline{prev(IP_1)\,IP_1}\rvert < \lvert\overline{prev(IP_1)\,IP_2}\rvert$

Table 2: *Conditions that no IVs exists in the different cases*
The symbol "'$\in$ poly"' means that the IP belongs to the polygon and the symbol "'$\notin$ poly"' means that the IP does not belong to the polygon.

There are no IPs when $IP_1 = IP_2 \ominus 1$, which is equivalent to

$$IP_1 \oplus 1 = IP_2.$$

If the second IP does not belong to the polygon we are looking at (Table 2 case ②), there are no IVs if the second IP lies on the edge which starts with IP_1 and ends with the clockwise successor of IP_1. This context is described by

$$succ(IP_1) = IP_2.$$

The situation that the first IP does not belong to the polygon we are looking at and the second is part of the polygon, is the case ②(see Table 2). This is similar to the case described above. The first IP lies on the edge which ends with the second IP and starts with the previous clockwise vertex of the second IP. If this holds, so does

$$IP_1 = prev(IP_2).$$

If both IPs do not belong to the polygon we are looking at (see Table 2 case ①), there are no IVs where both IPs are lying on the same edge of the polygon and the first IP lies

clockwise directly before the second IP.

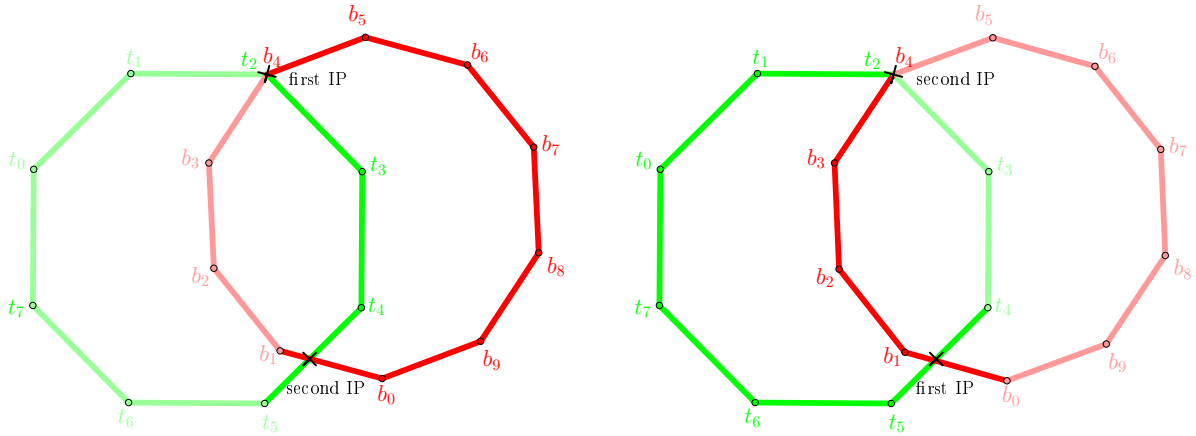
$$prev(IP_1) = prev(IP_2) \wedge succ(IP_1) = succ(IP_2) \wedge \left|\overline{prev(IP_1)\,IP_1}\right| < \left|\overline{prev(IP_1)\,IP_2}\right|$$

In the case we are looking at the traversable, we compare the indices of the IPs to check the ordering, because the IPs are ordered clockwise on the traversable. In the case we are looking at the blocker, we have the problem that the IPs are not ordered. It could be that they are ordered clockwise or counter clockwise. So we have to order them by taking the vector from the *prev* vertex of the first IP to the first IP itself and compare its length to the length of the vector from the *prev* vertex of the first IP to the second IP. This more complex test is only necessary for this case because in all other cases at least one vertex is element of the polygon we are looking at, so the ordering of the IPs is implied.

7.6.4 Coplanar Polygon Cutting Example

The basic idea of the CPC is that when two convex polygons intersect each other in two-dimensional space, you only need to calculate the intersection points in order to construct the polygon that represents the overlapping surface of the two polygons. The thing to keep in mind is, that both polygons are clockwise defined respective to the two-dimensional space they are defined in. The same has to be true for the newly constructed polygons. So when constructing new polygons from the IPs and IVs, you need to make sure that the vertices are added in the correct order. When the inside of the new polygon lies on the outside of the vertices' source polygon, then the ordering of the vertices has to be switched.

In the following example a simple polygon cut that only needs two iterations in the CPC will be used to explain the way the CPC Automaton works and how the vertices are added to the new polygon structures so they remain consistent with the clockwise definition.



(a) first run of CPC *(b) second run of CPC*

Figure 29: *Example CPC Automaton run*
The traversable is the green polygon and the blocker is the red polygon. The traversable IVs and blocker IVs have a brighter green or red than the other vertices in the respective CPC runs.

IPs are ordered clockwise along the current traversable's border. This is a matter of definition, they could as well be ordered clockwise along the projected lower blocker's border, but then the ordering of the CPC Automaton calls would change. The IPs are iterated in pairs, so every IP is used two times. Like in a cyclic structure the first pair is (IP_0, IP_1) and the last is (IP_n, IP_0).

In our example the IP_0 is a vertex of both polygons and the IP_1 is a new vertex (see subsubsection 7.3.3). Because in the example we only have two IPs, it is good to point out that it is irrelevant which IP is the first, only the clockwise ordering on the current traversable matters.

What we want in this example is that the overlapping part becomes the new blocker $\hexagon(IP_0, t_3, t_4, IP_1, b_1, b_2, b_3)$ and the rest of the current traversable, that lies outside the projected lower blocker, to become the new traversable $\hexagon(IP_1, t_5, t_6, t_7, t_0, t_1, IP_0, b_3, b_2, b_1)$.

In the first CPC run we have the IP pair (IP_0, IP_1). We get the traversable IVs between IP_0 and IP_1 by moving clockwise from IP_0 to IP_1 on the current traversable and we get the blocker IVs by moving clockwise from IP_0 to IP_1 on the projected lower blocker. This way we get traversable IVs $[t_3, t_4]$ and blocker IVs $[b_5, b_6, b_7, b_8, b_9, b_0]$.

The Blocker IVs lie outside the current traversable, so they are of no further interest for the CPC, because blocker vertices that lie outside the traversable are never part of the new geometry that we are trying to construct. The traversable IVs are part of the new blocker we want to construct, so we add IP_0, the traversable IVs and IP_1 in clockwise order to the new blocker. So we get $newBlocker = \bigcirc(IP_0, t_3, t_4, IP_1)$.

There are only two IPs, but they still have to be iterated like a cyclic structure, so the next run of the CPC is done with the IP pair (IP_1, IP_0). Now, like in the first run, we move clockwise on the current traversable from IP_1 to IP_0 and get the traversable IVs $[t_5, t_6, t_7, t_0, t_1]$. As blocker IVs we get $[b_1, b_2, b_3]$. Now we add the blocker IVs to the new blocker in clockwise order and get the finished obstruction $newBlocker = \bigcirc(IP_0, t_3, t_4, IP_1, b_1, b_2, b_3)$. Note that normally we would have to add IP_1 and IP_0 too, but when adding IPs to the polygons in construction, it always has to be checked for duplicates. If this check would not be performed, we would end up with a blocker $\bigcirc(IP_0, t_3, t_4, IP_1, IP_1, b_1, b_2, b_3, IP_0)$ (the red marked area in Figure 30).

Because the inside of the new traversable polygon lies on the outside of the projected lower blocker, we have to add the blocker IVs in counterclockwise order to the new traversable. Now to construct the new traversable, we move from IP_1 in clockwise direction on the current traversable to IP_0 and then from IP_0 in counterclockwise direction on the projected lower blocker to IP_1. This way we get the new traversable $\bigcirc newTraversable = \bigcirc(IP_1, t_5, t_6, t_7, t_0, t_1, IP_0, b_3, b_2, b_1)$ (the green marked area in Figure 30).

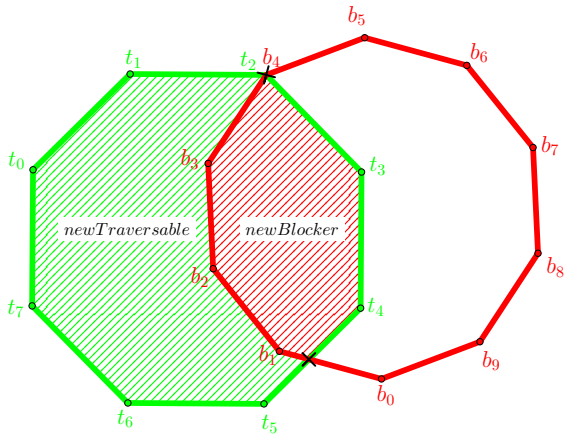


Figure 30: *The CPC results for the example without the convex decomposition of the* $newTraversable$

The traversable is the green polygon and the blocker is the red polygon. The $newTraversable$ is the green marked area and the $newBlocker$ is visualized by the red marked area.

Obviously the constructed new traversable $\hexagon newTraversable$ is not convex, how this is handled is explained in detail in subsubsection 7.6.6. For our example the CPC Convex Decomposition Call will decompose $\hexagon newTraversable$ into a set of convex new traversables

$$newTraversables = \{\hexagon(t_7, t_6, t_5, IP_1, b_1), \hexagon(b_1, b_2, t_7), \hexagon(t_1, t_0, t_7, b_2, b_3), \hexagon(t_1, IP_0, b_3)\} .$$

7.6.5 CPC Automaton

The CPC Automaton is used to calculate the intersection of the current traversable and the projected lower blocker, which is called “newBlocker”, and it constructs a set of new traversables which are the parts of current traversable without the projected lower blocker. The CPC is mainly based on rules how to construct the new traversables and the new blocker and a repeat loop over all adjacent IPs in clockwise order. To determine which rule we use to construct the new polygons we check if there are blocker IVs and if they

are inside the current traversable and if there are traversable IVs and if they are inside the projected lower blocker.

The inside/outside test for IVs, that is referenced in Table 3, is always tested against the other polygon. So blocker IVs are tested if they are inside or outside the current traversable and traversable IVs are tested if they are inside or outside the projected lower blocker.

If the overlapping surface of two convex polygons is always a convex polygon, then of course this polygon contains all IPs. The rules to construct the new blocker based on the part between two IPs has to contain the IVs of the polygon which is inside the other one. The new traversables then have to contain these inside IVs and the outside IVs of the other polygon. The outside IVs have to be the traversable IVs because of the definitions of the new traversables, which says that every new traversable is a subset of the current traversable.

Below we discuss all the different situations of the blocker IVs and the traversable IVs between two IPs (see Table 3). It does not matter with which pair of adjacent IPs we start, if we take care of the problem that the new blocker should not have the same vertex twice. So we assume that the new blocker vertices are all added so that there are no duplicates.

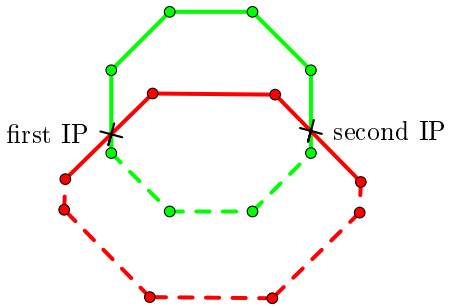


Figure 31: *Sketch of two IPs with traversable and blocker IVs and the traversable IVs are outside*

blocker IVs	inside / outside	traversable IVs	inside / outside	resulting actions
no		no		add [1. IP, 2. IP] to newBlocker
no		yes	inside	not possible
no		yes	outside	add [1. IP, 2. IP] to newBlocker & add ○(1. IP, traversable IVs (clockwise), 2. IP) to list of newTraversables
yes	inside	no		not possible
yes	inside	yes	inside	not possible
yes	inside	yes	outside	add [1. IP, traversable IVs (clockwise), 2. IP] to newBlocker & add convex decomposition of ○(1. IP, traversable IVs (clockwise), 2. IP, blocker IVs (counterclockwise)) to list of newTraversables
yes	outside	no		add [1. IP, 2. IP] to newBlocker
yes	outside	yes	inside	add [1. IP, traversable IVs (clockwise), 2. IP] to newBlocker
yes	outside	yes	outside	not possible

Table 3: *Different situations in the CPC and their processing*

If the blocker IVs are inside and any traversable IVs exist, they have to be outside, because the blocker IVs are inside (see Figure 31). The blocker IVs are inside, which means that they are also in the current traversable, that is why they are added to the new blocker. Because there are traversable IVs which are outside, there is a surface which has to be covered by new traversables. This surface consists of the first IP, the traversable IVs in clockwise ordering, the second IP and the blocker IVs in counterclockwise ordering. This surface usually looks like a half moon, which is normally not convex, so we construct the new traversable with the help of the Convex Decomposition, which is described in subsubsection 7.6.6.

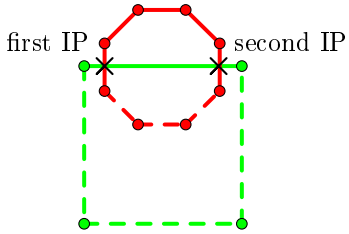


Figure 32: *Sketch of two IPs with blocker IVs, which are outside, and no traversable IVs*

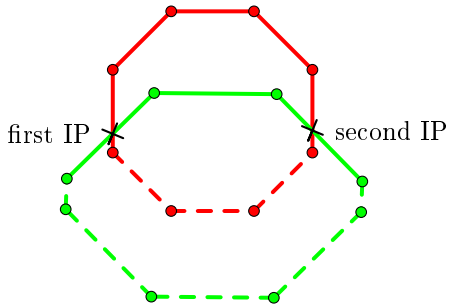
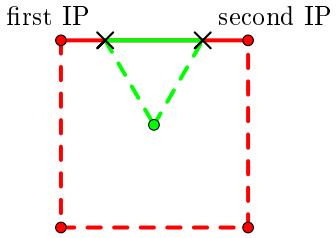
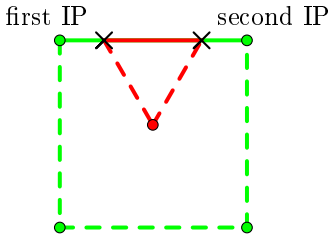


Figure 33: *Sketch of two IPs with blocker IVs, which are outside, and traversable IVs*

In the situations in which blocker IVs exist and are outside, there is no way that a new traversable should be constructed. So we add to the new blocker the first IP, the traversable IVs and the second IP, in case there are traversable IVs (see Figure 33). In the case there are no traversable IVs we add the first IP and then the second IP (see Figure 32).



(a) *A traversable edge lies on an edge of the blocker*



(b) *A blocker edge lies on an edge of the traversable*

Figure 34: *Two sketches of two IPs without IVs*

In the following situations there are no blocker IVs, which means that the traversable intersects an edge of the projected lower blocker.

If there are no IVs at all, an edge of the current traversable lies on an edge of the projected lower blocker or vice versa (see Figure 34). So we just have to add both IPs to the new blocker.

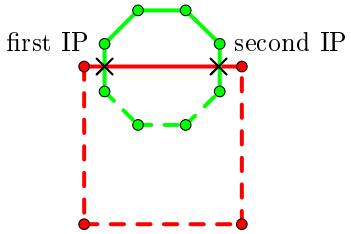
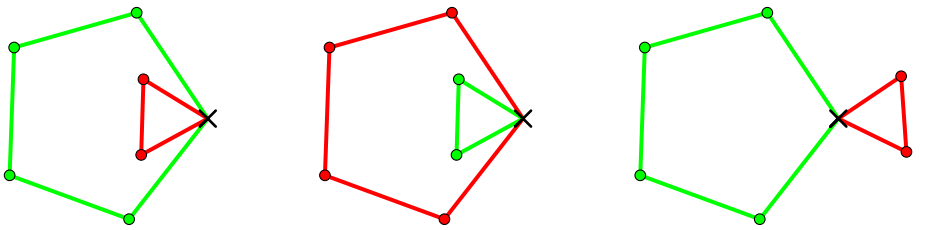


Figure 35: *Sketch of two IPs with traversable IVs, which are outside, and no blocker IVs*

In the situation that there are traversable IVs, they have to be outside (see Figure 35), because if they would be inside, then they would be IPs and our second IP would not be the second IP, because there are IVs which would also be IPs. So the traversable IVs have to be outside. A part of the traversable is cut off by the edge of the projected lower blocker, this a convex new traversable which has the vertices first IP, traversable IVs in clockwise ordering and the second IP. Also the two IPs are added to the new blocker.



(a) *Blocker inside the traversable*

(b) *Traversable inside the blocker*

(c) *Traversable and blocker with one boundary point*

Figure 36: *Three sketches for situation with one IP*

The case that only one IP exists is separately handled. Only one IP implies a boundary point, so the IVs together with the IP are always all vertices of the polygon. But to decide the correct operations we have to make some inside/outside tests. In the following consider $IP_0 = IP_1$.

The blocker is fully contained in the traversable (see Figure 36 (a)), if the blocker IVs are inside, which implies that the traversable IVs are outside. This case is handled like there were two different IPs, with traversable and blocker IVs and the traversable IVs are outside.

The traversable is fully contained in the blocker (see Figure 36 (b)), if the blocker IVs are outside and the traversable IVs are inside. In this case the traversable simply is converted into the new blocker and we are finished.

If the polygons have no overlapping surface (see Figure 36 (c)), there is nothing to do, because no part of the geometry needs to be modified. No overlapping surface is the case when traversable IVs and blocker IVs are positive for outside.

7.6.6 CPC Convex Decomposition Call

The CPC Convex Decomposition (CD) is a Decomposition of a non convex polygon into a minimal amount of convex polygons. The CD is nearly the same as the one described in subsubsection 7.5.3. There are just three differences that will be described in the following section.

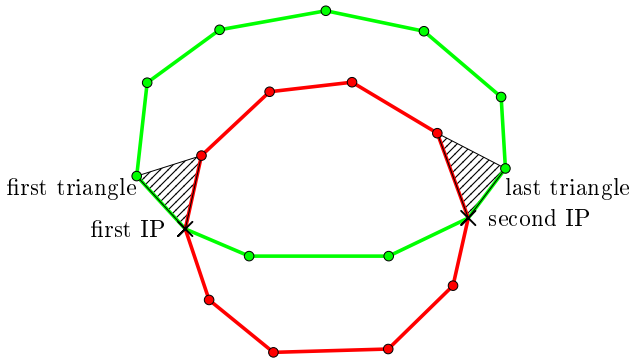


Figure 37: *The first and the last triangle construction of the Convex Decomposition in the CPC*

The first difference is the construction of the first triangle. In the Blocker totally contained in Traversable (BinT) we have to find a constellation of the current traversable and the projected lower blocker, where we have a triangle with two adjacent blocker vertices and one from the current traversable. And this triangle is not allowed to intersect the projected lower blocker in any thing other then the shared blocker edge. In the CPC CD we can construct the first triangle with the first blocker IV, the first IP and the first traversable IV (see Figure 37). We know that this triangle does not intersect the projected lower blocker in any other point then the points of the shared edge, because this CD is called if the traversable IVs are outside. After this first triangle is processed, like described above, the CD continues with the adding of as many blocker and traversable vertices as possible.

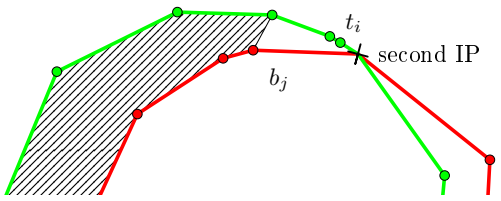


Figure 38: *Sketch to show that no blocker IVs can remain*

The next difference is that after the loop, which constructs the quadrangles in the CD, it is clear that only blocker IVs can remain when all traversable IVs have already been used. This is because the last blocker IV b_j and the vertex of the second IP, which is handled like a blocker IV (see Figure 38), always incorporate all remaining traversable IVs in one convex polygon.

At last the construction of the last triangle is different. Which is normally constructed with the remaining vertices, is now constructed manually with the last vertex of the traversable IVs, the last vertex of the blocker IVs and the second IP(see Figure 37). It is possible that the second IP is already used as the last blocker IV in the loop that constructs quadrangles and then it is not necessary to construct the last triangle separately.

7.6.7 CPC Result Processing

The CPC Automaton return results have to be post processed to be made valid for incorporation into the spatial partitioning system. It is possible that the constructed new blocker is not a polygon and then it should not be returned to the VCD and should not be inserted into the spatial partitioning system. This happens for example if two polygons only have one shared edge, then this shared edge will be the new blocker but it is a line-segment and not a polygon. Not every new blocker with more than two vertices is a polygon because we allow that polygons have three or more collinear vertices. So at the end of the CPC Automaton we have to check if the new blocker is a polygon, which means the polygon has to have more than two vertices and these are not allowed to be all collinear. The way we construct the new blocker guarantees, if it is a polygon, that it is simple and convex.

The new traversables are not required to go through post processing, because we know that they are all polygons, simple and convex. There are just two situation in which a new

traversable is constructed. In the first one the traversable intersects with a blocker edge and we construct a polygon with the traversable IVs and the two IPs. That guarantees to be a polygon, which is simple and convex, because the current traversable is convex and a line decomposes a convex polygon into two convex polygons. If the traversable intersects with the projected lower blocker in a way that there are blocker IVs, and of course traversable IVs, we have a surface like a half moon. This form is usually not convex and then we use the convex decomposition described in subsubsection 7.6.6 to get a set of new convex polygons which cover this surface. So all new traversables are convex and do not have to be checked by any sort of post processing.

8 Slicing by Vertical Blocker

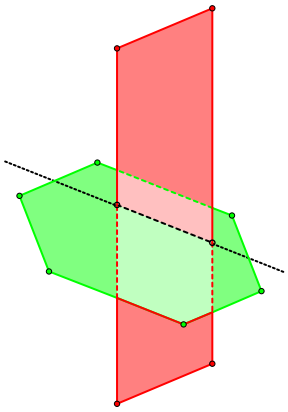


Figure 39: *Perspective sketch of a vertical blocker piercing through a traversable*

The slicing by a vertical blocker decomposes a traversable into two new traversables and separates them by an edge which holds the information that there is an obstruction. The separation can be done by a modified version of the BS. The BS cuts a blocker into two parts by a plane given by a polygon. Now we want to cut the traversable into two parts along a plane given by the blocker.

To be accurate it is necessary to not just cut along the plane of the blocker, but rather divide that cut into a part where the blocker is and potentially in two parts where the blocker is not. It is divided so that three edge and four vertices lie in the plane which is defined by the blocker. These four points can also be determined by the BS (see Table 1).

If the blocker does not pierce the traversable plane, the edge is still not traversable, if the blocker distance to the traversable is smaller than the smallest traversable stance. But if the blocker hovers higher than the smallest traversable stance over the traversable, the information how the edge can be traversed, meaning how it can be crossed, is added to the edge. If there is an intersection between traversable and blocker, this intersection does not have to be the part of the cut that is obstructed by the blocker. It is possible that the part which is obstructed by the blocker is larger than the intersection. Then this part has to be determined. To get the part that is obstructed by the blocker, all blocker vertices which are above or in the plane of the traversable are projected onto the intersection line between traversable plane and blocker plane. After this it is possible to calculate the distances of the projected blocker vertices to one point on the intersection line and the points with the smallest and the biggest distance represent the largest possible obstruction by the blocker. But you could also take different stances into account and then you would slice the blocker along the slicing planes, that are parallel shifted to the traversable plane. And you would just project the lower parts analogue to the way it is done in the VCD. After you projected all lower parts, you have a collection of distances and stances and you can cut the traversable along the intersection line between traversable plane and blocker plane. Afterwards the new edge has to be split according to the distances of the projected vertices and their stances.

9 Implementation

9.1 Introduction

The implementation was programmed in C++ and OGRE (see [Str]) was used as the rendering engine. It should be noted that one of the most important things is the integration of a fast spatial data system. Obviously using something fast is always good, but reducing the amount of VCD calls by using a well thought out spatial data structure is the best way to get faster practical performance. In our Implementation we used a very simple octree as a spatial data structure. More sophisticated spatial data structures like AABB trees [XLX10] are most likely the easiest way to speed up the overall performance of the practical implementation.

The overall speed of the algorithm when consuming a complete polygonal environment depends heavily on the way the polygons are positioned. For example, if very thin layers of polygons lie on top of each other, the algorithm will check each polygon against numerous of the other layers just to find that only the top one is traversable. If the environment geometry is very clean, without polygons sticking in each other, the algorithm is much faster in relation to the navigation space that is extracted.

9.2 Conservative Vertex Creation

Our data model supports shared vertices and shared edges between multiple polygons. In a normal polygonal world this means that most vertices and edges are part of more than one polygon.

We only create a new vertex when necessary, meaning we do not create a new vertex if there is already a vertex at this position in space. This leads to a much cleaner data structure over all because it naturally leads to shared edges between the traversables which

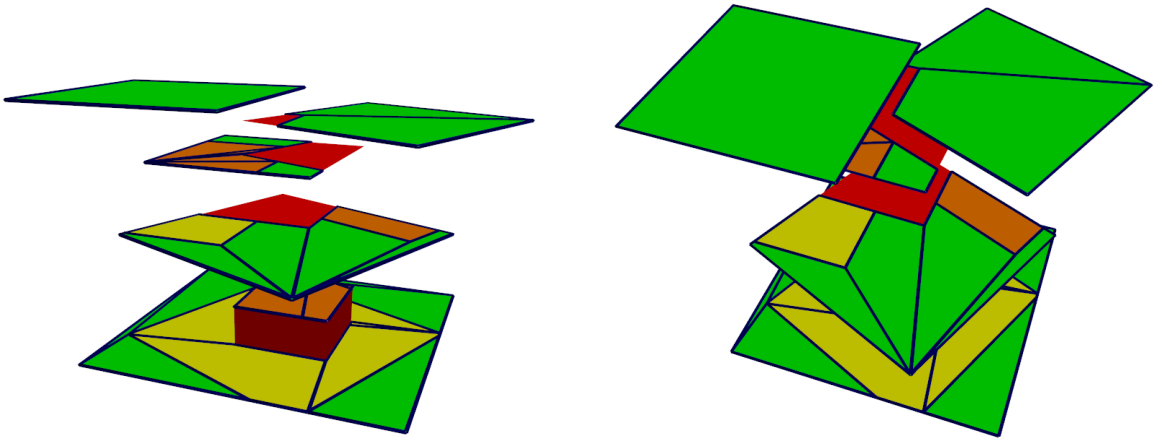
in turn means that edge welding is much easier to implement. In fact it is likely that any post processing and optimization greatly benefits from conservative vertex creation.

9.3 Minimal Edge Length and Vertex Snapping

We encountered a lot of problems with the precision of our calculations. Edges that were too short would not work because the intersection point calculations sometimes returned points that did not even lie on the edge. These precision problems will always exist, it just depends how small you make the edge, at some point you will run into trouble. To find solutions that would scale with the achievable precision was quite time consuming. In the end the main constraint was to define a minimal edge length, if in edge was shorter, it had to be collapsed. This constraint was sometimes met in advance to the edge creation. For example there is vertex snapping in the Blocker Slicing and in the Projected Outline Intersection. This means we snap the calculated intersection to an existing vertex, which prevents the creation of an edge which would be too short.

The repercussions of the minimal edge length constraint turned out to be another huge problem. When you start collapsing edges and snapping vertices onto one another, you end up changing the polygonal structure. This means there are cases were you violate the other constraints we put on the polygons, like convexity and planarity, to name the most important. So when we potentially changed the polygonal structure by collapsing or snapping, we had to check all polygons that shared the changed vertices or edges if they still fulfilled all the constraints. Then if any of the properties of one of these polygons changed, we had to fix it. This means either decomposing it into polygons that are planar again or modify the vertices so everything was fulfilled again. Depending how a case could be solved, the "wave" of structure changes extended further through the world.

10 Conclusions



(a) Low angle view *(b) High angle view*

Figure 40: *Complex multi layered extraction of multi stance navigation data. The red surface can not be traversed by the agent, orange can be traversed crouching, yellow can be traversed ducked and on the green surface the agent is able to stand.*

We achieved our goal to get to the 100% coverage of the navigatable surface we aimed for by using the precise downward projection of the obstruction geometry in the Traversable Decomposition (see section 7). We also managed to integrate the different stance heights into the main algorithm so the everything is calculated in one path instead of revisiting the navigatable surface to determine the maximum stance height (see section 6).

The rendering in Figure 40 illustrates the strength of our implementation. The rather complex multi layered level geometry is decomposed into precise polygons containing the information where the agent can move and in which stance.

However we did not manage to get an absolutely robust implementation. The problem with the arithmetic inaccuracy and the need to keep all polygons planar as described in

subsection 9.3 turned out to cause a lot of problems. And the huge number of tests and corrections to retain polygon planarity ended up costing a lot of computational power.

Using only triangles instead of polygons would have dramatically increased the triangle count compared to the polygon count, but the planarity issue just does not exist with triangles, which saves a lot of computations. Overall using triangles would have led to a much more robust implementation and most of the problematic cases of polygon-polygon intersection would have been avoided.

11 Acronyms

BinT Blocker totally contained in Traversable

BS Blocker Slicing

CD Convex Decomposition

CPC Coplanar Polygon Cutting

IP Intersection Point

IV Intermediate Vertex

TD Traversable Decomposition

TinB Traversable totally contained in Blocker

VCD Volume Collision Decomposition

12 References

[FSH94] Jr. F. S. Hill. "The Pleasures of "Perp Dot" Products". In: *Graphics Gems IV*. Ed. by Paul S. Heckbert. Academic Press, 1994, pp. 138–148.

[Far06] Fredrik Farnstrom. "Improving on Near-Optimality: More Techniques for Building Navigation". In: *AI Game Programming Wisdom 3*. Ed. by Steve Rabin. Charles River Media, Inc., 2006.

[FTU95] Francisco R. Feito, Juan Carlos Torres, and A. Ureña. "Orientation, simplicity, and inclusion test for planar polygons". In: *Computers & Graphics* 19.4 (1995), pp. 595–600.

[HY09] D. Hunter Hale and G. Michael Youngblood. "Full 3D Spatial Decomposition for the Generation of Navigation Meshes". In: *AIIDE*. 2009.

[HYD08] D. Hunter Hale, G. Michael Youngblood, and Priyesh N. Dixit. "Automatically-generated Convex Region Decomposition for Real-time Spatial Agent Navigation in Virtual Worlds". In: *AIIDE*. 2008.

[KS02] J. Mark Keil and Jack Snoeyink. "On the Time Bound for Convex Decomposition of Simple Polygons". In: *Int. J. Comput. Geometry Appl.* 12.3 (2002).

[Möl97] Tomas Möller. "A Fast Triangle-Triangle Intersection Test". In: *journal of graphics tools* 2.2 (1997), pp. 25–30.

[Mon] Mikko Mononen. *Recast*. URL: `http://code.google.com/p/recastnavigation/` (visited on 01/16/2012).

[O'R+82] Joseph O'Rourke et al. "A new linear algorithm for intersecting convex polygons". In: *Comput. Graph. Image Process.* 19 (1982), pp. 384–391.

[Pap03] Lothar Papula. *Mathematische Formelsammlung*. Vieweg, 2003.

[Str] Steve Streeting. *OGRE*. Torus Knot Software Ltd. URL: `http://www.ogre3d.org/` (visited on 01/16/2012).

[Tou85] Godfried T. Toussaint. "A Simple Linear Algorithm for Intersecting Convex Polygons". In: *The Visual Computer* 1 (1985), pp. 118–123.

[Toz02] Paul Tozour. "Building a Near-Optimal Navigation Mesh". In: *AI Game Programming Wisdom*. Ed. by Steve Rabin. Charles River Media, Inc., 2002, pp. 171–185. ISBN: 1-58450-077-8,

[Toz03] Paul Tozour. "Search Space Representation". In: *AI Game Programming Wisdom 2*. Ed. by Steve Rabin. Charles River Media, Inc., 2003.

[Weia] Eric W. Weisstein. *Hessian Normal Form*. MathWorld–A Wolfram Web Resource. URL: `http://mathworld.wolfram.com/HessianNormalForm.html` (visited on 12/15/2011).

[Weib] Eric W. Weisstein. *Line-Line Intersection*. MathWorld–A Wolfram Web Resource. URL: http://mathworld.wolfram.com/Line-LineIntersection.html (visited on 12/09/2011).

[XLX10] Yi-Si Xing, Xiaoping P Liu, and Shao-Ping Xu. "Efficient collision detection based on AABB trees and sort algorithm". In: *Ieee Icca 2010*. Ieee, 2010, pp. 328–332.

[You+11] G Michael Youngblood et al. "Embedding Information into Game Worlds to Improve Interactive Intelligence". In: (2011).

[Zha+07] Long Zhang et al. "Conservative Voxelization". In: *The Visual Computer* 23.9-11 (2007), pp. 783–792.

Selbstständigkeitserklärung

Hiermit versicheren wir, die vorliegende Arbeit selbstständig und unter ausschließlicher Verwendung der angegebenen Literatur und Hilfsmittel erstellt zu haben.

Ort, Datum	Sara Budde
Ort, Datum	Leonard Kausch