X

# Automated Static and Dynamic Obstacle Avoidance in Arbitrary 3D Polygonal Worlds

J.M.P. van Waveren, Drs. dr. L.J.M. Rothkrantz
*Delft University of Technology*
*The Netherlands*

## 1. Introduction

In common applications, a robot will have no prior knowledge of its environment and must use sensory information to build a cognitive model for path and route finding. The process of building a good cognitive model from sensory information alone is complex and time consuming. Uncertainty of the input data can also cause the resulting paths and routes to be suboptimal or erroneous. In many of these applications, however, it is possible to augment a robot's sensory information with existing static information about the environment.

Detailed and accurate digital blueprints of buildings are a fortuitous by-product of the prevalence of computer aided design in architectural design today. These data provide some information about the layout of the environment, but typically do not include more dynamic elements such as furniture, people, or other robots. Additional records may indicate what type of furniture is in a given room, but its placement may have changed over time. A robot navigating an office building will have to deal with the uncertainties of dynamic and unexpected objects in real-time, even if the layout of the building is known. However, the situation is vastly improved if the sensory data is used to identify object types, whose precise dimensions are already known and can be retrieved as needed.

The use of existing information of the environment is important to make a robot resource-efficient, while achieving near optimal path and route finding. However, processing existing information in the form of geometric models with hundreds of thousands of polygons can be expensive if a robot system is not carefully crafted to deal with such large amounts of information.

An automated system is presented for path and route finding through arbitrary 3D polygonal environments. The system can process a polygonal representation of an environment with hundreds of thousands of polygons within a few minutes on a small grid of today's computers. The processed information allows a robot to efficiently find routes and paths through the environment in real-time. Additionally, when a robot has identified and recognized dynamic obstacles in the environment based on sensory information, the

system is able to create near optimal paths in real-time around arbitrary configurations of dynamic obstacles.

The system presented here has been battle hardened in several generations of computer games, such as the triple-A titles: QUAKE III Arena, DOOM III and Enemy Territory QUAKE Wars. These computer games provide virtual environments for a robot (or artificial player), which are by no means inferior to real-life environments in that sophisticated physics simulations are used to move the robot and dynamic obstacles. These latest games show that the system can be applied to not only indoor but also outdoor environments when a polygonal representation of the terrain is available.

## 1.1 Previous Work

The objective of obstacle avoidance is to find collision-free trajectories between a start and a goal configuration in static and/or dynamic environments containing obstacles. A rich variety of algorithms for obstacle avoidance can be found in literature (Ribeiro, 2005).

The Certainty Grid method for probabilistic representation of obstacles in a grid-type world model has been developed at Carnegie-Mellon University (CMU) (Moravec, Elfes, 1985), (Elfes, 1987), (Moravec, 1988). This world model is especially suited to the accommodation of inaccurate sensor data such as range measurements from ultrasonic sensors. The work area is represented by a two-dimensional array of square elements, denoted as cells. Each cell contains a certainty value that indicates the measure of confidence that an obstacle exists within the cell area. The environment is scanned to update the certainty grid and local movement of the robot may be required for multiple scans. A global path-planning method is then employed for off-line calculations of subsequent paths.

Potential fields were introduced by Khatib (Khatib, 1985) for robot path-planning. Using the potential field method, every obstacle exerts a repelling force on the robot while the goal exerts an attractive force. Using this method, however, the robot might fall into local minima where it achieves a stable configuration before reaching a goal. A similar approach using repulsion vectors is used in (Johnson, 2003). A combination of the Certainty Grid method and potential fields, named Virtual Force Field, can be found in (Borenstein, Koren, 1989).

The Vector Field Histogram (VFH) approach was introduced by Borenstein (Borenstein, Koren, 1991), (Ulrich, Borenstein, 1998), (Ulrich, Borenstein, 2000). The VFH approach generates a polar histogram of the space occupancy in the vicinity of a robot. This polar histogram is then checked to select the most suitable sector out of all polar histogram sectors which have a low polar obstacle density. The steering of the robot is then aligned with that direction.

Using the Bugs algorithm (Lumelsky, Skewis, 1990), (Lumelsky, Stepanov, 1990), (Choset et al., 2005), (Kamon, 1998), also known as edge detection and wall following (Bauzil et al., 1981), (Iijima et al., 1983), (Giralt, 1984) a robot moves directly towards the goal unless an obstacle is found, in which case the obstacle is contoured until motion to the goal is again possible. However, it may be necessary to take all static and dynamic obtacles into account

at the same time to predict optimal paths towards the goal. When groups of objects are contoured the path may become jagged and non-optimal. Connolly (Connolly et al., 1990), (Connolly, Grupen, 1993) presents methods for planning smooth robot paths using Laplace's equation. Techniques such as "string-pulling" also known as "line-of-sight testing" (Snook, 2000) can also be used to optimize the paths.

Most of these systems are setup to create a cognitive model from sensory information without any prior information about the environment. (Waveren, Rothkrantz, 2006) presents a system for automated path and route finding in static polygonal environments. The system uses an off-line compilation process to derive a cognitive model of the environment that is suitable for efficient route and path finding. However, the system does not deal with dynamic obstacles in real-time.

Many other automated systems that use an off-line compilation process can be found in literature such as (Farnstrom, 2006), (McAnlis, Stewart, 2008), (Hamm 2008), (Axelrod, 2008) and (Ratcliff, 2008). Some of these systems use a discrete approach and others an analytical approach to create data structures that are suitable for route and path finding. The analytical approaches are complex or suffer from floating-point rounding errors on computers. The discrete approaches are prone to inaccuracies and errors that are inherent to discretely sampling the environment.

## 2. Static Obstacle Avoidance

Static obstacle avoidance deals with obstacles in the environment with a shape and position that never changes. Static obstacles may present themselves in many different forms and situations. Obvious examples of static obstacles are mountains and fences, on terrain, and the walls of a building. However, obstacles may also present themselves as gaps or pits a robot can fall into, or perhaps borders between countries that are not even physically marked. Unless some serious destruction takes place, it is safe to assume such obstacles always stay the same, or at least for an extended period of time. In many applications, pre-calculated data structures can be constructed based on existing information about the environment in order to speed up real-time route and path finding around a variety of different static obstacles.

### 2.1 Area System

The system presented here for dealing with static obstacles is similar to the system presented in (Waveren, Rothkrantz, 2006). The robot is assumed to live inside a simple bounding volume, and the robot has a limited number of degrees of freedom. In particular, the robot lives inside an axis-aligned bounding box that only translates through the world. This bounding box can be defined by six axis-aligned bounding planes relative to a reference point on the robot. The system can be implemented to work with different polygonal bounding volumes, but by using an axis-aligned bounding box the complexity is kept to a minimum. The system can also be extended to deal with rotations of the bounding volume of the robot. However, using more than 3 degrees of freedom significantly increases

the complexity, where the movement of the robot is bounded by hyper-surfaces that are much harder to visualize and conceive.

During an off-line compilation process, the system automatically derives a cognitive model for route and path-finding, from a polygonal representation of a world with static obstacles. The first step in this off-line compilation process involves the construction of a boundary representation of configuration space (C-Space). The boundary representation of C-Space consists of one or more two-manifold triangle meshes that describe the Minkowsky sum of the polygonal world geometry and the bounding volume in which the robot resides. This boundary representation describes the extents of the complete movement freedom of the robot in an environment with only static obstacles.

The configuration space is calculated using Constructive Solid Geometry (CSG) operations on two-manifold triangle meshes, in which meshes are subtracted from each other and welded together. The system presented in (Waveren, Rothkrantz, 2006) uses a three-dimensional (3D) Binary Space Partitioning (BSP) algorithm with portalization to calculate the boundaries of configuration space. However, 3D BSP algorithms are prone to floating-point rounding errors on computers when dealing with many polygons. These rounding errors may cause the constructed boundary representation of configuration space to not accurately describe the actual extents of the complete movement freedom of the robot. The floating-point rounding errors can be localized and minimized by subdividing polygonal environments into smaller blocks that are moved into a well defined floating-point range centered about the origin. Nevertheless, 3D BSP algorithms are prone to rounding errors when dealing with a mixture of very high and very low density polygonal geometry.

Before using CSG operations to calculate the boundary representation of configuration space, each polygon that is used to describe the environment is turned into a convex polytope. Such a convex polytope is described by a two-manifold triangle mesh that represents the Minkowsky sum of a single polygon and the bounding volume of the robot.

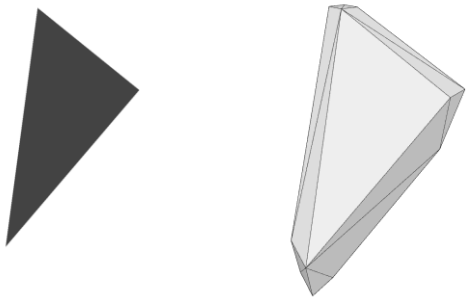

Fig. 1. A triangle in 3D space and the convex polytope that describes the Minkowsky sum.

There are several ways to construct such convex polytopes. One way is to place a copy of the axis-aligned bounding box in which the robot resides at each vertex position of a polygon, such that, the point of reference on the robot, relative to which the bounding box is defined, coincides with a vertex position. The convex polytope for a polygon is then constructed by

calculating the convex hull of all the corners of all the positioned bounding boxes. This approach assumes the point of reference on the robot, relative to which the bounding box is defined, is centered inside the bounding box. Once a convex polytope is defined for each polygon in the environment, the two-manifold meshes that describe these convex polytopes can be welded together using CSG operations in order to construct a complete boundary representation of configuration space.

Fig. 2. Boundary representation of C-Space for an outdoor environment with buildings.

In the next step of the off-line compilation process, traversable surfaces are identified on the boundary representation of configuration space. First, the slope of each triangle is determined. A triangle is considered traversable if the slope is less than 45 degrees. Each triangle edge between a traversable and non-traversable triangle can now be classified and flagged as either a "wall" edge or a "ledge" edge based on whether the triangles meet at an inward or outward angle respectively. The robot is assumed to be able to cover small height differences. For instance, a robot may be able to walk up stairs. Whenever a "ledge" edge is positioned directly above a "wall" edge, the overlapping part, where the edges are not too far apart, can be considered traversable. The "ledge" and "wall" flags are removed from the parts of the edges that overlap within the height difference that is considered traversable. As such, the overlapping edges of the steps of a staircase are not flagged. By processing all edges this way, a lot of geometric detail is ignored which helps to simplify the cognitive model used for path and route finding.

The edges that are flagged as "ledge" and "wall" edges can now be used to subdivide the traversable surfaces into the least number of traversable areas, such that a robot can move in a straight line between any two points in an area. A two-dimensional (2D) Binary Space Partitioning (BSP) algorithm is used to subdivide the traversable surfaces into areas where vertical planes through the "ledge" and "wall" edges are used as splitters. This 2D BSP algorithm is significantly more robust and numerically stable than a 3D BSP algorithm with partalization because only vertical planes are used to split two-manifold meshes. The 2D BSP algorithm effectively creates convex vertical columns in space that contain one or more convex traversable surface fragments. Within a column each fragment covers the whole area of the column from a top-down perspective. However, each fragment within a single column is part of a different floor. The different floors are separated in the BSP tree by

introducing additional split planes between the floors that are approximately level with the floor surface.

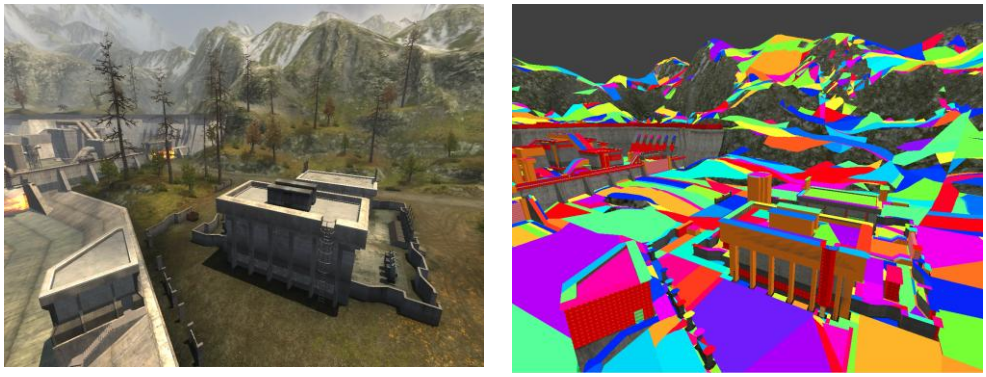

Fig. 3. Areas for an outdoor environment with buildings.

Because different floors of a building are not first separated, a floor on one level may be split by "ledge" or "wall" edges from a floor on a different level. In effect this causes unnecessary fragmentation of floors. However, convex traversable surface fragments from different columns can be trivially merged back together if they meet at an edge and the merged fragments also form a convex area in which a robot can move in a straight line between any two points. Merging traversable surface fragments together may cause multiple branches of the BSP tree to point to the same area. This is, however, not a problem because the BSP "tree" is typically only used to quickly find the area a robot is in. The inner edges of a traversable surface fragment can now be discarded so that an area is described by a single polygon.

The calculation of the configuration space is the most expensive step in the off-line compilation process. Fortunately, the polygonal representation of the environment can be broken up into small blocks, and the boundary representation of configuration space for each block can be calculated separately. A small grid of computers can be used to process these blocks in parallel, which significantly reduces the time it takes to complete the off-line compilation process. After all the blocks have been processed, the results from the individual blocks can be merged to form one or more two-manifold meshes that describe the complete boundary representation of configuration space for the whole environment.

## 2.2 Avoiding Ledges

The robot movement is modelled as an axis-aligned bounding box that translates through the environment. Even if most of this bounding box is hanging over a cliff or ledge, and only a corner of the box actually rests upon an edge, the box will not tumble as far as the area system is concerned. Figure 4 shows a bounding box which is hanging over a ledge.

Fig. 4. A robot partly standing over a ledge while the bounding box rests upon the edge.

So called "ledge" areas are introduced to keep the legs of a robot from partly dangling over ledges. The robot may travel through such areas if absolutely necessary but when calculating routes the robot will try to avoid these areas. The ledge areas are introduced at edges that are not traversable because the height difference is too large. A vertical plane is constructed through a ledge facing away from the gap. This plane is then moved away from the ledge such that if the robot stays in front of the plane, the robot bounding volume cannot dangle over the ledge. The areas of the area system are then cut up with this plane to create areas on both sides of the plane. Areas that are behind the plane can then be marked as ledge areas and the robot can avoid these areas when calculating routes, or knows to navigate through such areas with care.

## 2.3 Reachability

Areas describe spaces in which a robot can travel between any two points in a straight line. To travel between areas a robot needs additional information. For this purpose so-called reachabilities are introduced that describe how a robot can travel from one area to the next. In particular, a reachability describes how a robot can travel from one area to an adjacent area or an area in close proximity. The reachabilities can then be used to calculate routes through the environment.

A reachability is stored in the area system as a link between two areas and a point in space where the robot can transition from one area to the next. Reachabilities are easy to define using the polygons that represent the areas in the area system. Many polygon edges of adjacent areas are exactly on top of each other and can be used directly to define reachabilities where there is a smooth transition from one area to the next. The robot may, however, also be able to cover small height differences such as the steps of a staircase. A lot of these small height differences are already filtered and ignored when constructing the

areas, but the 2D BSP algorithm may have cut up the traversable two-manifold mesh such that there are still small height differences between areas. Because the robot is assumed to live inside an axis-aligned bounding box, these small steps always present themselves as edges from adjacent areas that are directly above each other. Areas in close proximity can be tested for edges that overlap when projected onto a horizontal plane. The vertical distance between the overlapping parts of such edges can then be calculated to find the places where a robot can reasonably be expected to cover the height distance and consequently navigate across the edges from one area to the next.

## 2.4 Routing

Routes are calculated between areas by following the reachabilities from one area to another. The number of areas, however, can grow quite large when the area system is employed in an environment described by hundreds of thousands of polygons. Environments that are mapped by ten thousand areas or more are not uncommon. Conventional routing algorithms can be time consuming when dealing with such large numbers of areas in real-time. All routes could be pre-calculated, but this consumes a lot of storage space and does not allow for adjustments to dynamic changes, such as areas that are temporarily disabled.

A hierarchical routing system is used to calculate routes with the same accuracy as a conventional routing algorithm. However, due to the hierarchical nature of the system, it takes significantly less time and space to calculate routes. The calculated routes are temporarily saved (cached) to avoid having to recalculate routes repeatedly. The caches can be freed and routes are recalculated, when the total size of all cached routes exceeds a threshold or when routes need to be adjusted due to dynamic changes in the environment. Routes are typically calculated and cached per goal area, because the goal area tends to stay the same over a longer period of time than the area the robot is in.

The hierarchical routing system creates a set of clusters of area. The areas within a cluster are connected through reachabilities. The areas are considered nodes of a graph and the reachabilities are considered the edges between the nodes. The clusters are separated by "cluster portals". These cluster portals are areas themselves and represent passages from one cluster to another. The clusters and portals are setup such that the only way to travel from one cluster to another is through a cluster portal. Each cluster portal separates no more and no less than two clusters.

The hierarchical routing system can save a lot of computation time and storage space by limiting routing calculations to the areas within clusters. Calculating all distances or travel times from all areas in a cluster to a specific goal area in the same cluster is significantly faster than performing routing calculations over all areas in the environment, because a cluster contains only a subset of all the areas. The upper bound for route calculations in a cluster is the square of the number of areas in the cluster; whereas the upper bound for calculating routes through the whole environment is square in the total number of areas in the environment.

The cluster portals are used to calculate routes through the whole environment. This works by interpreting the cluster portals as the nodes of a graph, and the routes through clusters as the edges between the nodes. The goal area of a route is also interpreted as a node of the same graph. The goal area is connected to cluster portals with routes through the cluster the goal area is in.

Routes through clusters are calculated with a simple breadth first routing algorithm. The breadth first algorithm starts at the goal area and uses the reversed reachabilities to flood to other areas. The algorithm never floods to areas outside the cluster, but does flood into cluster portals that touch the cluster. While flooding through a cluster, the algorithm calculates and temporarily saves travel times to the goal area for each area in the cluster. Routes through the whole environment are also calculated with a breadth first routing algorithm. The algorithm starts at the goal area and floods to all portals in the environment through the routes within clusters. For each portal the algorithm calculates and temporarily saves the travel time of the portal to the goal area.

At any given time the next area along a route can be calculated by first determining the portal of the current cluster that is closest to the goal area, and then calculating the route to this portal. If the goal area is in the same cluster as the source area, the direct route within the cluster is also considered and chosen instead if determined to be shorter.

## 2.5 Path Optimization

A route retrieved from the area system is a sequence of areas connected through reachabilities. The reachabilities can be used to define a path as a sequence of points in space. These intermediate points along such a path are typically at the centers of edges between areas. A line through these centers, however, may not describe the most optimal path around static obstacles. The arrows in figure 5 show a path retrieved from the area system around two wall sections. Obviously this is not the most optimal path.

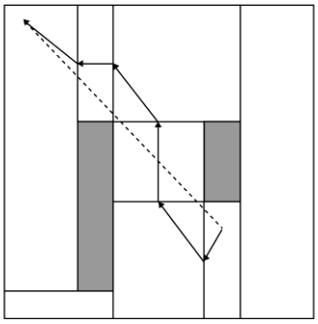

Fig. 5. Top down view of path optimization around two wall sections.

The path shown by the arrows in figure 5 can be optimized by plotting a line from the start position directly to one of the intermediate points that is further away, possibly several

areas down the path. The robot must be able to travel along this line without being obstructed. The polygons that describe the areas from the area system can be used to test whether or not the line goes through free space without obstructions. At any given time the robot can choose the intermediate point on the path that is furthest away and to which the robot can move in a straight line. Instead of only choosing intermediate points that are on the edges between areas, the lines between successive intermediate points can be sub-sampled. A line from the start position to such a sub-sampled point can also be tested against the polygons of areas to determine if the sub-sampled point can be reached by the robot without being obstructed. Sub-sampling avoids discontinuities that may arise when a robot follows a route through areas with vastly different sizes. Optimizing paths in this manner can be done in real-time. By continuously optimizing paths while moving, the robot will follow a smooth and close to optimal path to its destination.

## 3. Real-Time Dynamic Obstacle Avoidance

The area system provides a robust solution for static obstacle avoidance. However, there may also be many dynamic obstacles in the environment. When the positions and dimensions of nearby dynamic obstacles are known, the areas of the area system could be recalculated for a particular configuration of such dynamic obstacles. However, recalculating the areas is typically too time consuming on today's computers to be done in real-time. Instead, the area system only handles the static part of the environment. Another system which is built on top of the area system is used to calculate paths around arbitrary configurations of dynamic obstacles. The system described here assumes the dynamic obstacles have already been identified from sensory information and records. The system is similar to the Bugs algorithm or wall following, except that the full path around obstacles can be re-calculated and optimized repeatedly.

The problem of finding paths around dynamic obstacles is simplified by using a projection which makes the obstacle avoidance system suitable for real-time use. Furthermore, the system only deals with dynamic objects represented by oriented bounding boxes (OBBs). This restriction generally does not cause any problems because all dynamic obstacles can be contained within one or more tightly fitting OBBs. The wall edges from the area system are also considered as obstacles, such that the static obstacles are also taken into account when constructing a path around dynamic obstacles.

A robot may choose to ignore very small obstacles or obstacles that do not extend more than a small distance above the floor. The dynamic obstacle avoidance system, however, will always try to find a path around dynamic obstacles, and does not try to find a path that requires the robot to move over these obstacles. In some situations a pile of stacked obstacles may present a smooth top surface that may seem suitable for robot navigation. However, it is very hard to determine whether or not a stack of obstacles will collapse if the robot tries to move over the obstacles.

Fig. 6. Obstacles along the path from the camera position towards the door behind the table.

Figure 6 shows a room with a pile of boxes and a door behind a table. All the boxes, the table and the chairs are dynamic and can move through the room. A robot that wants to travel from the camera position towards the door will have to navigate around these obstacles. The walls and floor of the room are not dynamic and the area system only takes these static obstacles into account. As such, the optimized path retrieved from the area system is a straight line from the camera position to the door because none of the walls block the movement towards the door.

The dynamic obstacle avoidance system searches for any obstacles in proximity of the optimized path retrieved from the area system. If no obstacles are found, the robot can just follow the optimized path. If possible obstacles are found, the nearby walls from the area system are considered as obstacles as well. Each obstacle is represented by an oriented bounding box (OBB) which is projected onto a 2D horizontal navigation plane. With this projection a polygon is created that describes the contour of the projected OBB. The polygon of a projected obstacle is expanded and beveled to create a polygon that describes the 2D Minkowsky sum of the projected OBB of the obstacle, and the axis-aligned bounding box in which the robot resides. The polygon used for the obstacle avoidance is the outline of this Minkowsky sum.

### 3.1 Path Tree

A path tree is built using clock-wise and counter clock-wise edge walks along the expanded polygons. Each node in the tree is a line segment that describes part of a path around obstacles. The optimized path from the area system is followed until a polygon is hit. Whenever a polygon is hit, the tree branches to follow the edges of the polygon in both a clock-wise and counter clock-wise fashion. While following the edges of a polygon, a new node is added to tree for each edge of the polygon. While following an edge of a polygon another polygon may be hit. When this happens, a new node is added to the tree to follow the polygon that is hit. Before extending the tree with a node for a new edge of a polygon, the edge is tested to see if it faces the end point of the optimized path from the area system. If an edge faces this end point, the polygon is no longer followed and a new node is added to the tree in order to continue the path directly to this end point. Any polygons hit along this new path are again used to extend the tree. This process continues until all branches of the tree either reach the end point of the optimized path from the area system, or loop back

to a path already followed closer to the root of the tree. When following the edges of polygons in both a clock-wise and counter clock-wise fashion, a branch of the tree may loop back onto edges of polygons that have already been followed. Whenever that happens the branch is terminated.

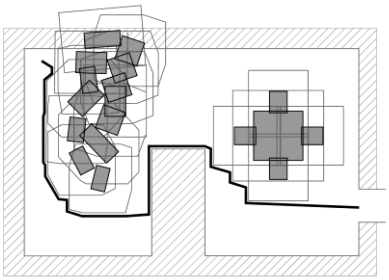

Fig. 7. Top down view of one of the paths around the obstacles.

Figure 7 shows a top down view of the room with obstacles that block a direct path from the camera position to the door. One of the branches of the tree is shown as a thick black line. The tree typically has many more branches, for instance, a branch that goes the opposite way around the table. These other branches are not shown for clarity.

Once the complete tree is built, the tree is traversed in a depth first manner in order to find all possible paths that may lead to the destination. Such a path is a sequence of nodes that represent line segments. A path that does not lead to the destination is discarded. A path which does lead to the destination is optimized by taking shortcuts where possible. Nodes and, as such, line segments are skipped if the new line segment that represents the shortcut does not intersect any other obstacles. Figure 8 shows the path from figure 8 that has been optimized in this way. The shortest path is chosen from all optimized paths that are found when traversing the tree. The path shown in figure 8 is the shortest path around the obstacles.

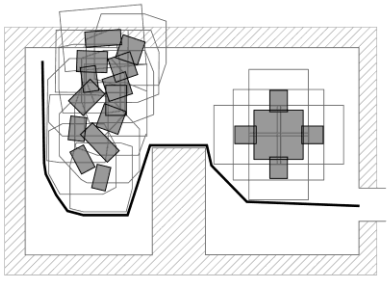

Fig. 8. Top down view of optimized path around the obstacles.

The start and/or end point of a path retrieved from the area system may be inside the expanded polygons of a dynamic obstacle. This can happen when the robot is standing on the edge of an obstacle near the ground, or if an obstacle covers the end point of the path from the area system. When this happens, the start and/or end point needs to be pushed out of any expanded polygons of obstacles in order to find valid paths around the obstacles. A point inside an expanded polygon is first pushed towards the closest edge of the expanded polygon. If that point is still inside other expanded polygons the point is pushed towards one of the intersection points of the expanded polygons. All intersections of the expanded polygons that contain the point are calculated and the first intersection that is outside any obstacles is chosen. If no such intersection is found, either the robot or the destination is completely surrounded by obstacles and the robot will first have to move obstacles out of the way to create a path.

The obstacle avoidance can be used to calculate paths around dynamic obstacles in real-time. The robot can recalculate paths around dynamic obstacles repeatedly such that paths will be updated in real-time while obstacles are moving. Furthermore, the system can cope with arbitrary configurations of dynamic obstacles, allowing a robot to find paths around or out of maze like configurations.

When the optimized path from the area system is recalculated repeatedly and the path around dynamic obstacles is also updated in real-time, there are cases where the area system and dynamic obstacle avoidance system may fight each other. The dynamic obstacle avoidance may change the path from the area system, which in return may cause the path from the area system to change as the robot moves along the modified path. In some situations this may lead to confusion where the systems are alternately directing the robot to go in a different direction. The solution to this is to follow the path from the area system until dynamic obstacles are found that block this path. Once dynamic obstacles are found the path from the obstacle avoidance is followed and the optimized path from the area system is no longer updated until the destination of this optimized path has been reached or there are no longer any obstacles blocking the path. In other words, the path from the area system is not updated while avoiding dynamic obstacles.

## 4. Results

The robots (artificial players) in the computer game Enemy Territory QUAKE Wars use the described system for real-time route and path finding. This computer game provides a variety of challenging environments for robot navigation. The environments are largely outdoors but there are also many smaller and larger buildings the robots can enter. Each environment in the game covers approximately a square mile. Large parts of the environments never change but there are also many dynamic obstacles such as other robots, human players and vehicles.

Table 1 shows statistics for several environments from the computer game Enemy Territory QUAKE Wars. For each environment table 1 shows the number of triangles that make up the static obstacles in the environment, the number of blocks the environment is broken up into, the number of triangles used to describe the boundary representation of configuration

space, the number of areas used for robot navigation, and the number of reachabilities between areas.

| Name | Obstacle triangles | Blocks | C-Space triangles | Areas | Reachabilities |
|---|---|---|---|---|---|
| Sewer | 112288 | 2768 | 156328 | 6089 | 30667 |
| Valley | 120133 | 4241 | 167638 | 9379 | 47595 |
| Volcano | 217553 | 2301 | 237680 | 7641 | 36818 |

Table 1. Statistics for several environments from Enemy Territory QUAKE Wars.

The above table shows that the number of triangles required to describe the boundary representation of configuration space is not significantly higher than the number of triangles that are considered static obstacles for robot navigation. This is partly because the robot lives inside a simple bounding volume. When a more complex bounding volume is used the number of configuration space triangles increases.

A small grid of 10 to 20 computers was used for the off-line compilation process to construct the boundary representation of configuration space. By using this small grid of computers the off-line compile time was reduced to just a couple of minutes.

In Enemy Territory QUAKE Wars 16 or more robots use the described system simultaneously. The combined CPU consumption of all robots is no more than 20% on a Intel Pentium 2 GHz processor.

## 5. Conclusion

The presented system is fully automated and can be used for path and route finding through arbitrary 3D polygonal environments with both static and dynamic obstacles. A polygonal representation with hundreds of thousands of polygons describing all static obstacles in the environment can be processed off-line within a few minutes on a small grid of today's computers. The off-line compilation process creates data structures that allow a robot to efficiently find routes and paths through the environment in real-time. Once dynamic obstacles are identified from sensory information, the system is also able to create near optimal paths in real-time around arbitrary configurations of dynamic obstacles.

The system has been successfully implemented and employed in the computer game Enemy Territory QUAKE Wars. This implementation shows that the system is resource-efficient. Multiple robots can use the system simultaneously for real-time path and route finding while only using a small percentage of all available compute power on a modest computer.

## 6. Future Work

For performance reasons the dynamic obstacle avoidance system uses a projection onto a 2D navigation plane. Dynamic obstacles well above a robot, or obstacles that do not extend more than a small distance above the floor can be ignored. However, imagine a thin pipe against a wall that fell over across a hallway such that it leans against the opposite wall at a

45 degrees angle. A robot may be able to navigate through the hallway by passing underneath the pipe close to the wall the pipe rest against. Unfortunately the dynamic obstacle avoidance system is unable to direct the robot through the hallway because it uses a projection of the pipe which covers the whole hallway. The pipe could be represented by multiple oriented bounding boxes where some are ignored if they are well above the robot. However, the dynamic obstacle avoidance could also be implemented as a wall following algorithm that follows the contours of 3D convex polytopes as opposed to polygons in the plane. This increases the complexity and the required compute power but will allow a robot to find paths around or through more complex configurations of dynamic obstacles.

## 7. References

G. Bauzil, M. Briot, P. Ribes (1981). A Navigation Sub-System Using Ultrasonic Sensors for the Mobile Robot HILARE, *1st In.. Conf. on Robot Vision and Sensory Controls*, pp. 47-58 and pp. 681-698, 1981, Stratford-upon-Avon, UK

J. Iijima, S. Yuta, Y. Kanayama (1983). Elementary Functions of a Self-Contained Robot YAMABICO 3.1, *Proc. of the 11th Int. Symp. on Industrial Robots*, pp. 211-218, 1983, Tokyo

G. Giralt (1984). Mobile Robots, *NATO ASI Series, Robotics and Artificial Intelligence*, Vol. F11, pp. 365-393, 1984, Springer-Verlag

H. P. Moravec, A. Elfes (1985). High Resolution Maps from Wide Angle Sonar, *IEEE Conference on Robotics and Automation*, pp. 116-121, 1985, Washington D.C.

Oussama Khatib (1985). Real-time obstacle avoidance for manipulators and mobile robots, *In Proceedings of the 1985 IEEE International Conference on Robotics and Automation*, pp. 500-505, March 1985

A. Elfes (1987). Sonar-based Real-World Mapping and Navigation, *IEEE Journal of Robotics and Automation*, pp. 249-265, Vol. RA-3, No 3, 1987

H. P. Moravec (1988). Sensor Fusion in Certainty Grids for Mobile Robots, AI Magazine, pp. 61-74, Summer 1988

J. Borenstein, Y. Koren (1989). Real-time Obstacle Avoidance for Fast Mobile Robots, *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 19, No. 5, pp. 1179-1187, Sept./Oct. 1989

Christopher I. Connolly, J.B. Burns, R. Weiss (1990). Path Planning Using Laplace's Equation, *Proceedings of the IEEE Conference on Robotics and Automation*, pp. 2102-2106, May 1990

V. Lumelsky, T. Skewis (1990). Incorporating range sensing in the robot navigation function, *IEEE Transactions on Systems Man and Cybernetics*, vol. 20, pp. 1058 - 1068, 1990

V. Lumelsky, Stepanov (1990). Path-planning strategies for a point mobile automaton amidst unknown obstacles of arbitrary shape, *Autonomous Robots Vehicles*, pp. 1058 - 1068, 1990, Springer, New York

J. Borenstein, Y. Koren (1991). The vector field histogram - fast obstacle avoidance for mobile robots, *IEEE Transaction on Robotics and Automation*, vol. 7, no. 3, pp. 278 - 288, 1991

Christopher I. Connolly, Roderic A. Grupen (1993). Applications of Harmonic Functions to Robotics, *Journal of Robotic Systems*, 10(7), pp. 931-946, 1993

Ishay Kamon, Elon Rimon, Ehud Rivlin (1998). Tangentbug: A Range-Sensor-Based Navigation Algorithm, *The International Journal of Robotics Research*, vol. 17, nr. 9, pp. 934-953, September 1998

I. Ulrich, J. Borenstein (1998). VHF+: Reliable obstacle avoidance for fast mobile robots, *Proc. of the IEEE Int. Conf. on Robotics and Automation*, pp. 1572 - 1577, 1998

I. Ulrich, J. Borenstein (2000). VHF*: Local obstacle avoidance with look-ahead verification, *Proc. of the IEEE Int. Conf. on Robotics and Automation*, San Francisco, pp. 2505 - 2511, 2000

Greg Snook (2000). Simplified 3D Movement and Pathfinding Using Navigation Meshes, *Game Programming Gems*, 2000, Charles River Media

Geraint Johnson (2003). Avoiding Dynamic Obstacles and Hazards, *AI Game Programming Wisdom 2*, 2003, Charles River Media

Maria Isabel Ribeiro (2005). Obstacle Avoidance, *The Robotics WEBook*, Institute for Systems and Robotics, Instituto Superior Técnico, October 2005, Lisboa, Portugal

H. Choset et al (2005). *Principles of Robot Motion: theory, algorihms and implementations*, Englewood Cliffs and New Jersey, 2005, MIT Press

J.M.P. van Waveren, Drs. dr. L.J.M. Rothkrantz (2006). Automated path and route finding through arbitrary complex 3D polygonal worlds, *Elsevier, Robotics and Autonomous Systems*, February 2006

Frederick Farnstrom (2006). Improving on Near-Optimality: More Techniques for Building Navigation Meshes, *AI Game Programming Wisdom 3*, 2006, Charles River Media

Colt McAnlis, James Stewart (2008). Intrinsic Detail in Navigation Mesh Generation, *AI Game Programming Wisdom 4*, 2008, Charles River Media

David Hamm (2008). Navigation Mesh Generation: An Empirical Approach, *AI Game Programming Wisdom 4*, 2008, Charles River Media

Ramon Axelrod (2008). Navigation Graph Generation, *AI Game Programming Wisdom 4*, 2008, Charles River Media

John W. Ratcliff (2008). Automatic Path Node Generation for Arbitrary 3D Environments, *AI Game Programming Wisdom 4*, 2008, Charles River Media