A Generalized Exact Arbitrary Clearance Technique for Navigation Meshes

Ramon Oliva* Nuria Pelechano†
Universitat Politècnica de Catalunya

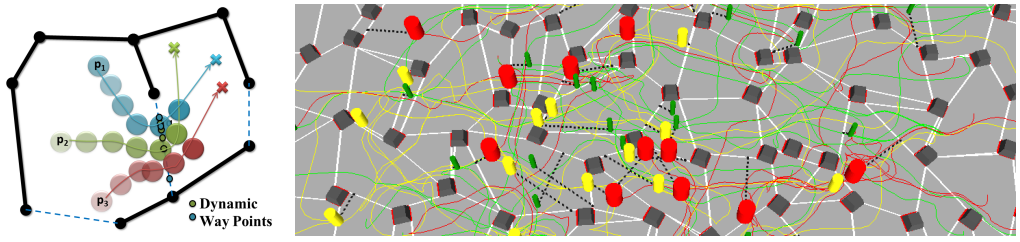

Figure 1: *On the left, we illustrate the trajectories followed by three characters as their way points over the portal with clearance are dynamically adjusted to their current positions and collision avoidance. On the right, paths followed by agents of different radii (from largest to smallest in red, yellow and green) in a NavMesh showing their dynamically assigned way points (in black).*

Abstract

There are two frequent artifacts in crowd simulation, the first one appears when all agents attempt to traverse the navigation mesh sharing the same way point over portals, increasing the probability of collision against other agents and lining up towards portals; the second one is caused by way points being assigned at locations where clearance is not guaranteed which causes the agents to walk too close to the static geometry, slide along walls or even get stuck. In this work we propose a novel method for dynamically calculating way points based on current trajectory, destination, and clearance while using the full length of the portal, thus guaranteeing that agents in a crowd will have different way points assigned. To guarantee collision free paths we propose two novel techniques: the first one provides the computation of paths with clearance for cells of any shape (even with concavities) and the second one presents a new method for calculating portals with clearance, so that the dynamically assigned way points will always guarantee collision free paths. We evaluate our results with a variety of scenarios, and compare our results against traditional way points at the center of portals to show that our technique offers a better use of the space by the agents, as well as a reduction in the number of collisions.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation I.2.9 [Artificial Intelligence]: Robotics—Workcell organization and planning

Keywords: clearance, navigation meshes, dynamic way points

Links: DL PDF

*e-mail:ramon.oliva.martinez@gmail.com
†e-mail:npelechano@lsi.upc.edu

1 Introduction

Applications such as video games require characters within a crowd to follow visually convincing paths in real time. Characters should move towards their destination along a realistic path, and at the same time maintain an appropriate amount of clearance with respect to the obstacles and avoid collisions with other agents as smoothly as possible.

Navigation meshes (NavMeshes) are commonly used to carry out navigation of autonomous characters. NavMeshes consist of a data structure that encodes the free space of the scene by splitting it into convex polygons, known as cells. A Cell-and-Portal Graph (CPG) is obtained where a node represents a cell of the partition and a portal is an edge of the graph that connects two adjacent cells. Then, given a start and a goal position, paths can be calculated through a variant of the classic A* algorithm. Finally, at every step of the simulation, a local movement algorithm is applied in order to guide the agent through the obtained path by computing intermediate goal positions (commonly known as way points) that connect the different nodes of the path.

When simulating a variety of characters, it is convenient to be able to calculate the shortest route for the characters based on their size. If we think of applications such as video games, this would allow a skinny character to escape from a large monster by running through a narrow passage. Efficiency is also a key aspect, as many characters may require a path computation in the same frame over a large scenario, so only a small fraction of a second is available.

Additionally, the method used to compute the way points is also critical in order to produce visually convincing routes. Most proposed solutions are based on computing a single point over the portal (usually at the center, or at the endpoints of the portal), so all agents share the same way point. This results in agents that tend to line up when approaching the portal from the same side, or form bottlenecks when several agents attempt to cross the portal coming from different directions. These artifacts reduce artificially the flow rates through portals and thus the overall time to reach their destination, and are also perceptually unpleasant. An algorithm that can run in real time assigning different way points to different characters can mitigate these artifacts.

Previous work is either bounded to a specific amount of clearance, or only work with a specific type of Navigation Mesh (e.g. triangular meshes, medial axis). On the contrary, our method is able to deal with an arbitrary amount of clearance and can work with any type of

NavMesh even if cells are not strictly convex. To show this, we have integrated our local movement algorithm into NEOGEN [Oliva and Pelechano 2013].

Main Contributions. This work presents a novel system to guarantee character trajectories with clearance that make the most of the available free space in the NavMesh. We present three contributions: Firstly, a novel technique to dynamically use the whole collision free space of portals to assign way points. Secondly, a novel method for calculating clearance in navigation meshes consisting of cells of any shape. And finally a new technique to compute clearance over portals. The algorithm is straight forward and computationally efficient to allow simulation of crowds.

2 Related Work

Path planning of autonomous characters in virtual environments is a central problem in the fields of robotics, videogames, and crowd simulation. The most popular solutions are based on a combination of global and local movement techniques. The target of global navigation techniques is to provide a representation of the free space of the scene that is usually obtained by either constructing a roadmap or a navigation mesh. The main objective of both approaches is to generate a graph that can be used by a search algorithm (usually A* [Hart et al. 1968]) to find a path free of obstacles between two points in the scene.

The roadmap approach [Arikan et al. 2001][Young 2001][Sud et al. 2007][Rodriguez and Amato 2011] captures the connectivity of the free space by using a network of standardized paths (lines, curves). The main limitation of this representation is that it does not describe the geometry of the scene, nor where the obstacles are. Consequently, avoidance of dynamic obstacles is usually a hard task and not always possible, as exposed in [Sud et al. 2007].

The navigation mesh approach [Snook 2000][Tozour 2002][Kallmann 2005][Pettre et al. 2005][van Toll et al. 2012][Oliva and Pelechano 2013] consists of the partition of the navigable space of the scene into convex regions, guaranteeing that a character can move between two points of the same cell following a straight line, without getting stuck in local minima. Currently NavMeshes have become more popular than roadmaps as the representation of the free space is more intuitive, clean, and provides a better description of the free space and the location of the obstacles. So we will focus on this global navigation technique.

The target of local movement techniques is to provide a mechanism for the autonomous characters to move from one location to the next one in the path in a smooth and natural manner, while avoiding collisions against dynamic obstacles. These methods are generally driven by setting way points within the portals of the NavMesh that work as attractors to steer the agents in the right direction [Reynolds 1987][Reynolds 1999][Pelechano et al. 2007][van den Berg et al. 2008a][van den Berg et al. 2008b][Snape et al. 2011].The main problem of this approach is that characters tend to line up as they share the same attractor point over the portal. An improvement to traditional way points was introduced in [Curtis et al. 2012] by using way portals where the whole length of the portal can be used to attract the local movement of the agents, thus resulting in more natural looking paths. However, the problem of clearance is not properly addressed on this paper, since they assume that a cell is accessible by a character if the length of the portal that needs to be crossed is greater or equal than the diameter of the character, which is not always the case as we will show in this paper.

In order to carry out Path planning guaranteeing that the resulting paths will have an arbitrary amount of clearance, a common solution consists of enlarging the obstacles by a specific amount of clearance known as the Minkowski sum. An example of an application using this method is Recast [Mononen 2009]. The main advantage of this approach is that every calculated path has the desired amount of clearance and it is calculated offline, so it does not have an impact on the performance of the path finding algorithm being used. However, its major drawback is that it is bounded to a specific value of clearance, so all characters must have the same size, or at most the specific clearance size.

In [Kallmann 2010], Kallman introduced a new type of triangulation called Local Clearance Triangulation (LCT) that allows paths to be computed free of obstacles with arbitrary clearance. Such triangulation is obtained by a process that iteratively refines the Constrained Delaunay Triangulation (CDT) resulting from the starting set of obstacles. The resulting structure determines if there exists a path free of obstacles for a given clearance value. However, it introduces more cells in the partition of the scene, thus dropping the performance of the path finding algorithm. Another limitation of the method is that it only works for the described LCT but cannot be generalized to any navigation mesh.

In [Geraerts 2010], the Medial Axis of the set of obstacles is extracted to create a new data structure called the Explicit Corridor Map (ECM). The ECM allows computing the shortest path, the path that has the largest amount of clearance, or any path in between. This work has been further extended to calculate a finer set of attractors to obtain smoother paths [Karamouzas et al. 2009][Jaklin et al. 2013]. Straight skeletons have also been used to calculate roadmaps for path finding of multiple characters [Haciomeroglu et al. 2007].

3 Clearance Value of a Cell

The clearance value of a cell depends on how we cross this cell. Given a cell $\mathcal{C}$ (see Figure 2), we define a cell cross as the pair ($\mathcal{P}_1$, $\mathcal{P}_2$) of $\mathcal{C}$, where $\mathcal{P}_1$ is the entry portal and $\mathcal{P}_2$ is the exit portal. We classify the obstacle edges of the cell into edges to the left (*stringLeft*) and edges to the right (*stringRight*) in respect to the path that crosses the cell from the entry portal to the exit portal. Notice that it is not necessary to have strictly convex cells, in fact, cells generated by NEOGEN [Oliva and Pelechano 2013] are allowed to have certain concavities depending on the convexity relaxation threshold chosen when creating the mesh.

The algorithm proceeds by iterating over every portal endpoint and notch (i.e., a vertex such that its internal angle is greater than π) present in *stringLeft* and the closest edge in *stringRight* is determined. The distance between the notch and the closest edge is the clearance value of this notch. Notice that in this case, the endpoints of each string must be treated as if they were notches. The clearance value of the left string cl_L is the minimum of those distances. To compute the clearance value of the right string cl_R, we proceed in the same way. Finally, the clearance value of the described path is computed as follows:

$$cl\left(\mathcal{P}_1, \mathcal{P}_2\right) = min\left(cl_L, cl_R\right) \tag{1}$$

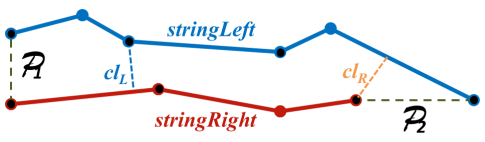

Figure 2: *Clearance calculation for a given cell.*

Note that it is only necessary to check the distance of the notches of the string against the edges of the opposite string, as in the case of a

convex vertex, the distance to the opposite string must be greater or equal than the clearance value of the cell. This process is done off-line once the NavMesh of the virtual scenario has been generated and, for each cell, we store in a table the clearance value of every possible cell cross. This is because it is possible to have a cell with three or more portals, where an agent with a large radius can walk for example from portal $\mathcal{P}_1$ to $\mathcal{P}_2$, but not from portal $\mathcal{P}_1$ to $\mathcal{P}_3$ (see Figure 3).

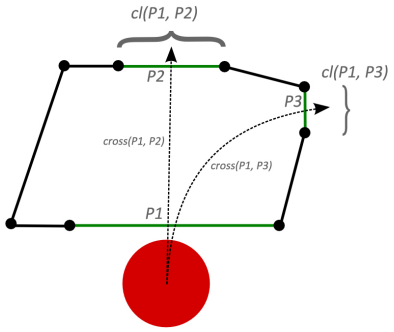

Figure 3: *Example of different clearance depending on the crossing path through a cell.*

4 Finding Portals with Enough Clearance

In order to avoid artifacts such as characters bouncing, sliding or getting stuck on the edges of the geometry, we should only assign way points that have enough clearance, meaning that they have the required distance from the static geometry for the character to traverse the portal without collision. Note that fixing way points to the center of the cell does not always guarantee collision free traversals, as we can see in Figure 4

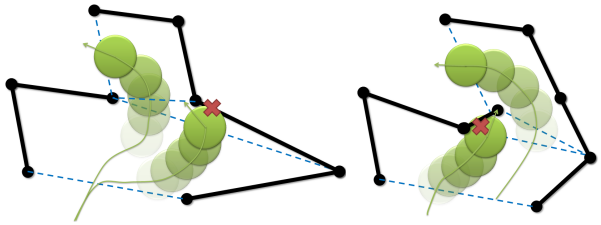

Figure 4: *Examples of fixed way points that cause collisions. On the left, the way point is fixed at the center of a cell which causes a collision with the geometry and on the right the way point is assigned at a distance r of the portal endpoint also causing collision.*

Let $\mathcal{C}_A$ be the cell were the character is currently located, $\mathcal{C}_B$ be the next cell in the path and $\mathcal{P}$ the portal that joins both cells. We want to calculate the sub-segment $\mathcal{P}'$ of $\mathcal{P}$ such that all points in $\mathcal{P}'$ have enough clearance. The idea is to shrink the portal $\mathcal{P}$ by displacing the obstacle edges of $\mathcal{C}_A$ and $\mathcal{C}_B$ a distance of r (where r is the radius of the character) towards the portal $\mathcal{P}$.

The algorithm for finding portals with enough clearance proceeds by reducing the size of the original portals based on the following three cases:

1. Limitations given by the endpoints of the current portal.
2. Limitations given by the endpoints of other portals in either $\mathcal{C}_A$ or $\mathcal{C}_B$.
3. Limitation given by edges of the adjacent cells.

case 1: The algorithm starts by displacing each endpoint of $\mathcal{P}$ a distance of r units towards the center of the portal as we can see in Figure 5. The resulting sub-segment $\mathcal{P}'$ has enough clearance only if the rest of the obstacles in $\mathcal{C}_A$ and $\mathcal{C}_B$ are at a distance greater than or equal to r from $\mathcal{P}'$. Otherwise, this sub-segment must be further refined to guarantee collision-free traversability.

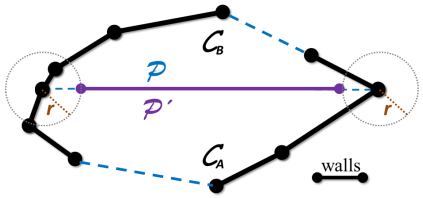

Figure 5: *Example portal $\mathcal{P}$ that connects $\mathcal{C}_A$ and $\mathcal{C}_B$. The shrunk portal $\mathcal{P}'$ is initialized by displacing the endpoints of the original portal $\mathcal{P}$ a distance r towards its center.*

In order to further shrink portal $\mathcal{P}'$ based on cases 2 and 3, we need to consider portals and edges as if they were defined for each cell in counter-clock wise order (Figure 6 depicts this situation). $\mathcal{C}_A$ and $\mathcal{C}_B$ are thus two oriented polygons where the vertices are given in counter-clockwise order. $\mathcal{P}$ can then be treated as two identical overlapping segments given in opposite order depending on which cell they belong to. Then we can refer to them as $\mathcal{P}_{AB}$ for the oriented edge that belongs to $cell_A$, and $\mathcal{P}_{BA}$ for the oriented edge that belongs to $cell_B$.

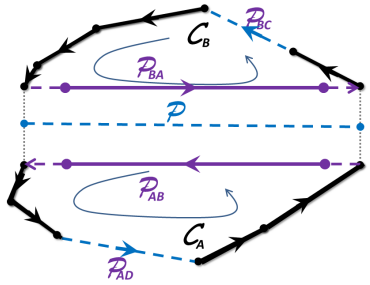

Figure 6: *$\mathcal{C}_A$ and $\mathcal{C}_B$ separated by $\mathcal{P}$ are in fact two independent polygons with their vertices oriented in counter-clockwise order, so $\mathcal{P}$ is the overlapping of $\mathcal{P}_{AB}$ and $\mathcal{P}_{BA}$.*

case 2: Let $\mathcal{C}_B$ be an intermediate cell on the character's path, i.e., a cell that is neither the starting cell of the path nor the final one. In this case, we have to cross the cell by crossing two portals, an entry portal $\mathcal{P}$ and an exit portal $\mathcal{P}_{BC}$ (portal that connects cell $\mathcal{C}_B$ with $\mathcal{C}_C$). In such a situation, it is possible that the endpoints of $\mathcal{P}'$ are determined by the endpoints of $\mathcal{P}_{BC}$. This occurs when one (or both) endpoint of $\mathcal{P}_{BC}$ is at a distance less than or equal to the desired clearance value from the entry portal $\mathcal{P}$. To handle this situation, we check if a circumference centered in the endpoints of $\mathcal{P}_{BC}$ intersects with $\mathcal{P}'$. If this intersection exists, we update $\mathcal{P}'$ accordingly.

Considering the polygons with oriented edges from Figure 6, let us define $\mathcal{P}_{BA}[0]$ as the origin of the oriented portal $\mathcal{P}_{BA}$, and $\mathcal{P}_{BA}[1]$ as the end. Since the portals are also given in counter-clockwise order, we can state that the origin of any portal can only limit the clearance of the end of the portal for which we are calculating clearance, $\mathcal{P}_{BC}[0]$ can only shorten $\mathcal{P}'_{BA}[1]$, and $\mathcal{P}_{BC}[1]$ can only shorten $\mathcal{P}'_{BA}[0]$. The algorithm to further shorten $\mathcal{P}'_{BA}$ continues through the following two cases:

- If the circumference centered in $\mathcal{P}_{BC}[0]$ intersects $\mathcal{P'}_{BA}$ at a single point, then $\mathcal{P'}_{BA}[1]$ is set to be this intersection point.
- If the circumference centered in $\mathcal{P}_{BC}[0]$ intersects $\mathcal{P'}_{BA}$ at two points, then $\mathcal{P'}_{BA}[1]$ is set to be the intersection point that is furthest from $\mathcal{P}_{BA}[1]$.

Symmetrically, a circumference centered on $\mathcal{P}_{BC}[1]$ is checked for intersections against $\mathcal{P'}_{BA}$ to determine if $\mathcal{P'}_{BA}[0]$ needs to be updated. Figure 7 shows the result of the algorithm over an example. Figure 8 illustrates the importance of respecting the ordering of the portals when calculating portals with clearance. Even though in both cases the characters can walk through the portals, in the first case the way points assigned over the portals would continuously push the characters to collide with the static geometry.

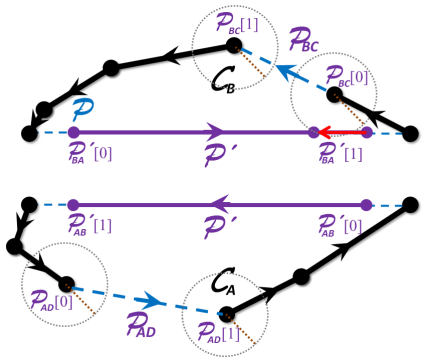

Figure 7: *The endpoint $\mathcal{P'}_{BA}[1]$ of the entry portal is determined by the endpoint $\mathcal{P}_{BC}[0]$ of the exit portal, since the circumference centered on $\mathcal{P}_{BC}[0]$ intersects $\mathcal{P'}_{BA}$. The other end of $\mathcal{P'}_{BA}$ is not modified, since the circumference centered in $\mathcal{P}_{BC}[1]$ does not intersect $\mathcal{P'}_{BA}$.*

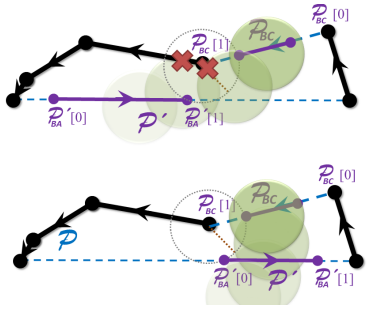

Figure 8: *On the top figure we can see an example of what the character's trajectory would be if $\mathcal{P}'$ was not shrunk respecting the direction of the portals. The trajectory leads to collision against the static geometry. On the bottom we can see the right trajectory when clearance is calculated correctly.*

case 3: Given a cell $\mathcal{C}_X$, and a set of vertices in counter-clockwise order $\{v_0, v_1, ..., v_n\}$, where each consecutive pair of vertices in the sequence defines an oriented edge of the cell, i.e: $\vec{e}_{(i,i+1)}$ is the edge starting in vertex v_i and ending in vertex v_{i+1}, for $i = [0, n-1]$. We define the shrinking direction of an edge, $\vec{s}_{(i,i+1)}$, as the unit vector perpendicular to the edge with its direction pointing towards the interior of the cell (Figure 9).

The algorithm proceeds by displacing each obstacle edge $\vec{e}_{(i,i+1)}$ a distance of r units along its its shrinking direction, $\vec{s}_{(i,i+1)}$ if the shrinking direction points towards the portal (otherwise there is no chance of intersection). After displacement we obtain $v'(i)$ and $v'(i+1)$ as the results of displacing vertices v_i and v_{i+1} . For each displaced edge $\vec{e'}_{(i,i+1)}$, we calculate its intersection against $\mathcal{P'}_{BA}$, and if such intersection exists, the corresponding endpoint of $\mathcal{P'}_{BA}$ is updated depending on the direction of $\vec{e'}_{(i,i+1)}$ as follows:

- If $v'_{(i)}$ is inside $\mathcal{C}_B$ and $v'_{(i+1)}$ is inside $\mathcal{C}_A$, then $\mathcal{P}_{BA}[0]$ is set to be the intersection point.
- If $v'_{(i)}$ is is inside $\mathcal{C}_A$ and $v'_{(i+1)}$ is inside $\mathcal{C}_B$, then $\mathcal{P'}_{BA}[1]$ is set to be the intersection point.

Figure 9 shows this process over the example scenario with a magnified view of the area of interest. The same process is performed for $\mathcal{P'}_{AB}$ and finally, $\mathcal{P}'$ is computed as the resulting sub-segment of the intersection between $\mathcal{P'}_{AB}$ and $\mathcal{P'}_{BA}$. Every point in $\mathcal{P}'$ is guaranteed to have enough clearance. Figure 10 shows the result of the algorithm.

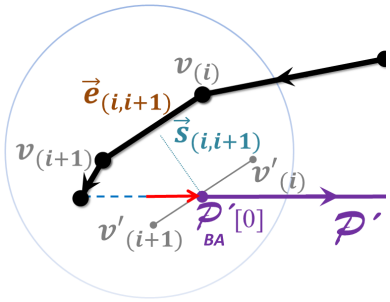

Figure 9: *Close up of the top left of Figure 7 with the shrinking process due to displacing edges.*

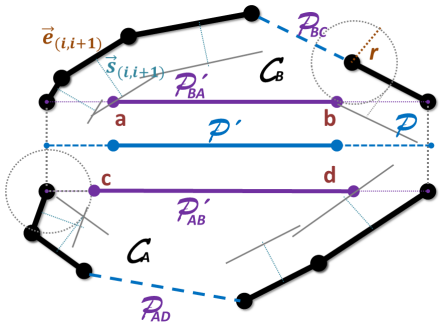

Figure 10: *Final result $\mathcal{P}'$ after calculating the merging of the intermediate solutions $\mathcal{P'}_{AB}$ and $\mathcal{P'}_{BA}$. The resulting shrunk portal before merging illustrates the application of the three cases: Case 1 can be seen in c, case 2 results in b and case 3 in a and d. $\mathcal{P}'$ is given in this example by the most limiting endpoints which are a and b.*

To accelerate the computation of the shrunk portal, we store the result of the transformation for a particular value of clearance in a table. So the next time that the portal needs to be shrunk, the table is checked for that particular clearance value so it does not need to be computed again.

Even though in the general scenario, case 3 will always be the most restrictive and thus the only necessary calculation, it is important to notice that there can be exceptions such as the ones illustrated in Figure 11 where case 3 may not limit in anyway the clearance of the portal. Therefore we want to emphasize the need for the three cases treated by our algorithm as well as illustrate why previous work that only considered distance to endpoints of the portals would fail to give natural paths in certain scenarios.

It is also important to notice that if the original navigation mesh has too many ill-conditioned cells, it may be necessary to iteratively check cases 2 and 3 with not only the adjacent cells, but also the cells that are connected to them. This can be easily incorporate by checking whether the other portals in the cell are at a distance smaller than r from $\mathcal{P}$ and if so, check against portal endpoints and edges in those cells with shrinking direction pointing towards $\mathcal{P}$. Since this is calculated only once and then stored in the lookup table, the impact on the final results would not be significant. The inclusion of this iterative step can be left to be decided by the user as a trade-off between portal clearance accuracy and performance.

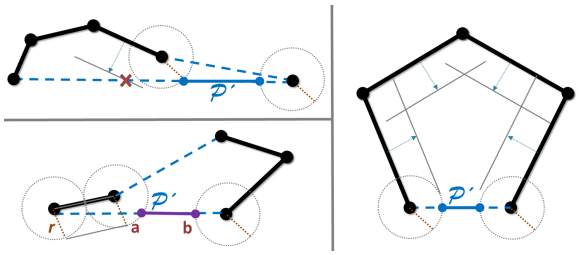

Figure 11: *These examples show different situations, where portal clearance is not defined simply by case3, and thus cases 1 and 2 are strictly necessary.*

5 Dynamic Way Points

In the case of navigation meshes, the method used to steer the character from one cell to another is a key aspect to create natural routes. When way points are assigned at a fixed position, usually the center of the portals, animation artifacts arise (Figure 12). The most common artifacts are line formation among characters that move in the same direction, and bottlenecks caused by characters crossing cells in opposite direction and being forced to pass through the same point. Another typical approach in video games consists of setting the way points at a distance r from the closest endpoint of the portal (where r is the radius of the character). This solution provides slightly more natural paths since paths are apparently shorter and at least two way points are available for each portal, but it does not completely solve the problem. Our work focuses on dynamically calculating way points over the shrunk portal (Figure 13).

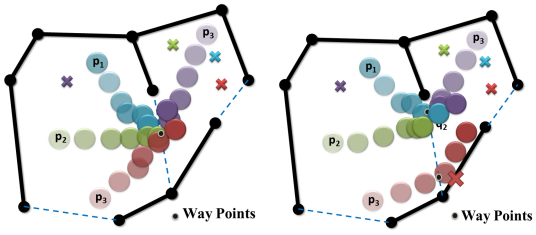

Figure 12: *Typical lining up artifacts and bottlenecks when way points are set at either the center (left) or the closest endpoint of the portal (right).*

Our dynamic way points assignation is based on the position of the character within the cell. First of all, we check if the goal position of the character is visible from its current position, i.e., the segment joining the current and the goal position of the character only produces intersection with portal edges. In that case, the attractor point is simply the goal position. If the segment does intersect with at least one obstacle edge, we need to compute a way point over the next portal in the path to steer the character towards the next cell of the path. Our target is to avoid characters having the same attractor point, so we compute the orthogonal projection point $\mathbf{q}$ of the current position of the character $\mathbf{p}$ over $\mathcal{P}'$, where $\mathcal{P}'$ is the shrunk portal after applying the algorithm described in section 4 over the portal $\mathcal{P}$. If $\mathbf{q}$ lies outside the limits of $\mathcal{P}'$, then the furthest endpoint of $\mathcal{P}'$ with respect to the current position of the character is selected as a temporal attractor, until $\mathbf{q}$ is valid.

The position of the characters is given by the local movement algorithm used to steer them. This algorithm will naturally move characters away from each other to avoid collision. Since character's position approaching a portal will be different, their projection over the portal will also be different, making it virtually impossible for two different characters to share the same attractor point over the portal if the characters are at risk of colliding.

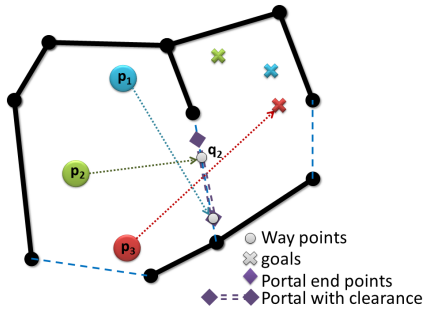

Figure 13: *The attractor point of the red character is its own goal since it is visible from its current location. The green character has its orthogonal projection, $\boldsymbol{q}_2$, over the portal as its way point, whereas the blue character has the farthest away endpoint of the portal assigned as its way point, since its current orthogonal projection lies outside the portal with clearance $\mathcal{P}'$.*

We have determined empirically that in the case of $\mathbf{q}$ being invalid, the furthest endpoint of $\mathcal{P}'$ is a better candidate as temporal attractor than the closest one. This is because when the steering attractor is the closest endpoint, the character tends to move too close to the walls, producing a bad quality route.

6 Results

In order to evaluate the results obtained with our algorithm, we have carried out both qualitative and quantitative analysis. The two things we are interested in are: first to examine whether our clearance method combined with dynamic way points achieves a better use of space, and second the performance of our algorithm to be able to work with large groups of agents in real time when computing paths with clearance and collision free way points.

Figure 14 (and the accompanying videos[1]) shows a comparison between using traditional way points at the center of portals and our method with dynamic way points for two example scenarios, the first one shaped as a donut and the second one shaped as a cross with static obstacles randomly located. The local movement algorithm is the same for all scenarios, and it is based on a simple rule based model with collision avoidance, steering towards attractors (way points) and collision response. Characters are considered to cross a portal as soon as the physics engine Bullet [Coumans 2013] detects that the character has arrived to the destination cell. Dynamic way points make better use of the space, with straight trajectories whenever possible and more natural looking trajectories for the characters, even when using a very simple rule based model

[1] http://www.lsi.upc.edu/ npelechano/videos/MIG2013.mp4

for their local movement. When way points are fixed at the center of portals, we can observe that not only do the paths not make use of the available space but also they are more chaotic as characters bounce around portals trying to get close to the way point while avoiding each other.

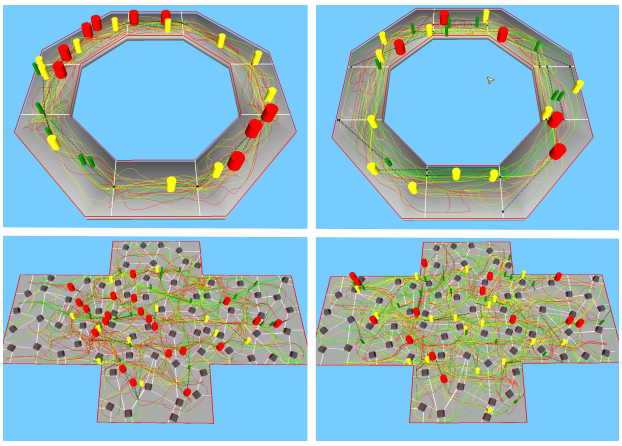

Figure 14: *Comparison between having way points at the center of portals (on the left) and dynamic way points (on the right) for the donut scenario with 25 agents (top row) and large cross scenario with 50 agents (bottom row).*

The following results have been obtained in an Intel Core 2 Quad Q6700 @ 2.66GHz 2.67GHz, 8GB of RAM . Figure 15 compares the time spent per query (microseconds) of different versions of the portal shrinking method:

- **SimpleShrink(-/+)**: A fast and simple method, commonly used on videogames and other virtual applications, that simply displaces the endpoints of the portal r units towards its center. The (+) version uses a lookup table to store previously computed shrunk portals, and the (-) calculates it at every simulation step.
- **ExactShrink(-/+)**: Our exact clearance solution described in section 4. The (+) version uses a lookup table to store previously computed shrunk portals while the (-) calculates it at every simulation step. Each test case consists of a set of queries where, for each query, we randomly chose a cell of the NavMesh, a trajectory to cross this cell (i.e. an entry portal and an exit portal) and a clearance value (0.5, 1 or 1.5).

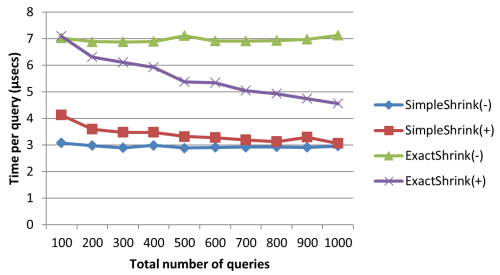

Figure 15: *Comparison for the time taken per query (in microseconds) as the number of queries increases for the different shrinking techniques.*

The results of this experiment highlight the efficiency of our exact clearance method (*ExactShrink(+)*). The efficiency of the algorithm increases with the number of queries as the chance of producing a redundant query is higher, and eventually, every query will be redundant. Results show that for the case of 1000 random queries, the cost of *ExactShrink(+)* is just 1.54 times the cost of the most efficient version, that in this case is *SimpleShrink(-)*. So the algorithm for calculating portals with exact clearance presented in this paper (*ExactShrink(+)*) is around 50% more time consuming for 1000 queries than simpler implementations, but it also guarantees that every computed path will have enough clearance with the static geometry. Notice that as the number of queries increases this percentage is further reduced. For the given example, we get a probability of hit of 50% for 1000 queries, which means that one in two queries does not need to be computed since it is already stored in the lookup table, and 90% probability of hit when it reaches 6000. So the time taken by the *ExactShrink(+)* algorithm converges towards the *SimpleShrink(+)* method.

and also that this increment in time does not have a big impact on the overall simulation since it is insignificant compared to the cost of AI, rendering or physics.

The memory requirements to store the lookup table are minimal, since for each radius size we only need two 3D point coordinates for the corresponding shrunk portal. For example, in the cross scenario with 208 portals, 3 character sizes and 12Bytes per 3D point, the total memory required is less than 15K.

Since there are many elements that affect the resulting frame rate of an application, such as: rendering engine, physics library, local movement algorithm, size of the scenario, size of the crowd, and so on, we are not interested in how many characters we can simulate in real time, but in comparing our method for paths with clearance against the standard solution where characters walk towards way points fixed at the center of portals without checking for any kind of clearance against the static geometry. Figure 16 (top) shows a comparison of the average frame rate achieved as the number of characters increases with and without our technique, when all the other elements of the simulation stay the same. This graph compares the standard solution (in red) against our technique (in blue). As we can see, the results are practically the same (less than 5% smaller frame rate on average with our method), meaning that the computational time required to calculate portals with clearance and dynamic way points is insignificant in the overall simulation time. Both simulation can handle up to 500 characters in real time. Obviously this result could be greatly improved with a different rendering engine, but it would not change the comparison provided in these results. At the bottom of this figure we show the decline in performance which in the worst case is around 0.12x, and on average is 0.05x. Therefore we can claim that the computational cost of our technique is insignificant for the overall simulation time, and provides results that are perceptually nicer and make better use of the space, as we can see in Figure 17 and the accompanying videos.

To show the results achieved by the path finding algorithm with clearance, we can observe in Figure 17 the different paths used by the characters depending on their size. As we can see, the larger characters only traverse those cells with a clearance larger than their radius. Another nice outcome of the presented method is the use of space made by the characters depending on their size. We can observe in the image how as the character's size decreases, their final emerging trajectories of their color are wider, since their way points are assigned over larger shrunk portals.

Finally we wanted to demonstrate quantitatively that having dynamic way points, not only provides better visual results independently of the local movement algorithm used, but also drastically reduces the number of collisions by spreading the crowd over the length of the portal. Figure 18 shows the results of a simulation where all the agents are moving in the same direction walking

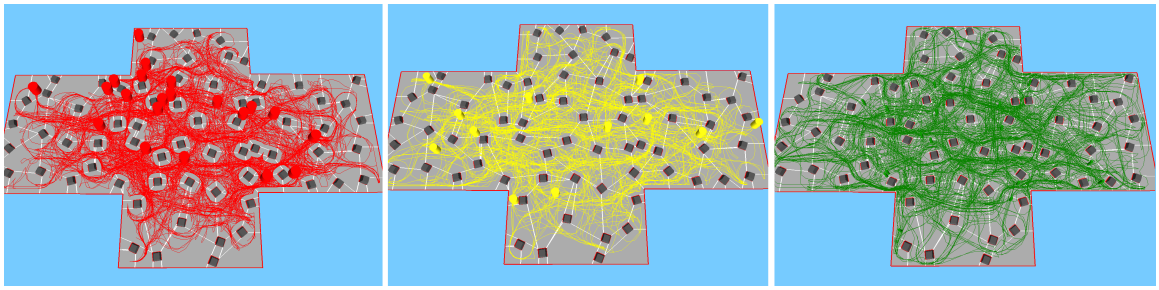

Figure 17: *Trajectories followed by characters of different size. From left to right, the larger characters (red, $r = 2.0$) will not use the narrower portals, the medium characters can already get through most ot the portals (yellow, $r = 1.5$), and finally the smaller size characters (green, $r = 0.5$) can walk through all the portals having the largest shrunk portals.*

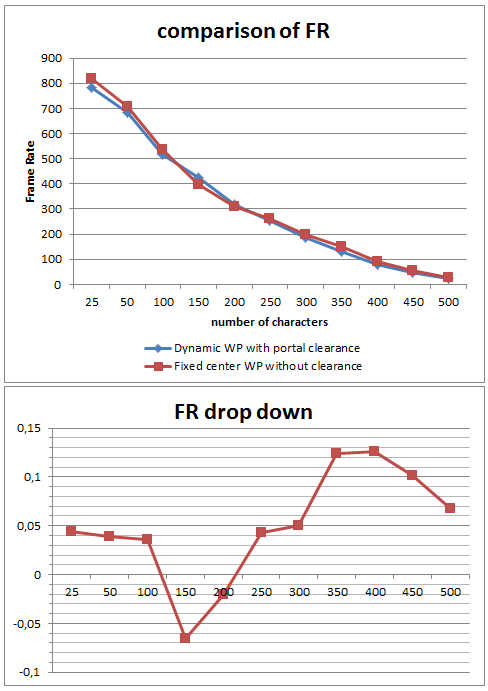

Figure 16: *Average frame rates obtained in large cross scenario as the number of characters increase for our method and a standard solution (top), and the performance drop down (bottom) calculated as how many times slower is the dynamic solution compared to the static one.*

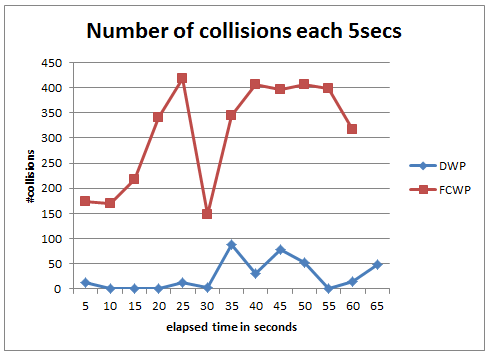

Figure 18: *Number of collisions between agents for the donut scenario when all the characters move in the same direction. We compare dynamic way points against fixed center way points.*

around the donut scenario. We have counted for one minute of simulation, the number of collisions that occur between agents every 5 seconds. As we can see in the graph, there is a drastic reduction in the number of collision when using Dynamic way points instead of Fixed center way points.

7 Conclusion

We have presented a general technique to compute paths free of obstacles with an arbitrary value of clearance that can be easily integrated in any existing Navigation mesh system.

Our method can be divided into the following steps: Firstly, during the construction of the NavMesh, the clearance value of each cell is computed in order to obtain paths that guarantee clearance when applying the A* algorithm. Secondly, the portals of the path are refined by shrinking them depending on the clearance required for each character and the surrounding geometry. Finally, way points over the shrunk portals are computed depending on the character position and hence, it mostly avoids two characters sharing the same attractor point.

Results show that our method is fast enough compared to simplest implementations, but produces paths of higher quality as it takes into account clearance for both path planning and way point calculations, and its dynamic assignation of way points along portals avoids characters lining up when crossing portals or causing bottlenecks.

We have tested our algorithm with NavMeshes of a variety of scenarios created by NEOGEN [Oliva and Pelechano 2013] which is a NavMesh generator that provides an almost near-optimal number of cells with very few ill-conditioned cells. If another NavMesh generator was used which created too many ill-conditioned cells, it may be possible that the portal clearance is not given exclusively by the connected cells, but also the immediately adjacent ones. In this case the algorithm could easily be adjusted by checking the distance to other portals in the cell and based on that consider also the visible edges of the adjacent cells in the algorithm described in section 3. In the future we would like to incorporate this possibility.

For the qualitative evaluation of this work we have considered that higher quality paths are those that tend to use most of the available space instead of lining up to have all agents crossing through the same way point, reduce bottlenecks and collisions against the geometry and between agents. As future work it would be interesting to consider some metrics to carry out a quantitative evaluation of clearance, path lengths and collisions.

Acknowledgements

This work has been partially funded by the Spanish Ministry of Science and Innovation under Grant TIN2010-20590-C01-01.

References

ARIKAN, O., CHENNEY, S., AND FORSYTH, D. A. 2001. Efficient multi-agent path planning. In *Proceedings of the Eurographic workshop on Computer animation and simulation*, Springer-Verlag New York, Inc., New York, NY, USA, 151–162.

COUMANS, E. 2013. Bullet physics library. http://bulletphysics.org/.

CURTIS, S., SNAPE, J., AND MANOCHA, D. 2012. Way portals: efficient multi-agent navigation with line-segment goals. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, ACM, New York, NY, USA, I3D '12, 15–22.

GERAERTS, R. 2010. Planning short paths with clearance using explicit corridors. In *IEEE International Conference on Robotics and Automation, ICRA 2010, Anchorage, Alaska, USA, 3-7 May 2010*, IEEE, 1997–2004.

HACIOMEROGLU, M., LAYCOCK, R. G., AND DAY, A. M. 2007. Distributing pedestrians in a virtual environment. IEEE Computer Society, Los Alamitos, CA, USA, vol. 0, 152–159.

HART, P. E., NILSSON, N. J., AND RAPHAEL, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on 4*, 2 (July), 100–107.

JAKLIN, N., COOK IV, A. F., AND GERAERTS, R. 2013. Real-time path planning in heterogeneous environments. *Computer Animation and Virtual Worlds 5*, 24, 285–295.

KALLMANN, M. 2005. Path planning in triangulations. In *Proceedings of the IJCAI Workshop on Reasoning, Representation, and Learning in Computer Games*.

KALLMANN, M. 2010. Shortest paths with arbitrary clearance from navigation meshes. In *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, SCA '10, 159–168.

KARAMOUZAS, I., GERAERTS, R., AND OVERMARS, M. 2009. Indicative routes for path planning and crowd simulation. In *Proceedings of the 4th International Conference on the Foundations of Digital Games*, 113–120.

MONONEN, M. 2009. Recast navigation toolkit. http://code.google.com/p/recastnavigation/.

OLIVA, R., AND PELECHANO, N. 2013. Neogen: Near optimal generator of navigation meshes for 3d multi-layered environments. *Computer And Graphics* (Apr.).

PELECHANO, N., ALLBECK, J. M., AND BADLER, N. I. 2007. Controlling individual agents in high-density crowd simulation. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, SCA '07, 99–108.

PETTRE, J., LAUMOND, J. P., AND THALMANN, D. 2005. A navigation graph for real-time crowd animation on multilayered and uneven terrain. In *Proceedings of the 1st International Workshop on Crowd Simulation*, 81–90.

REYNOLDS, C. W. 1987. Flocks, herds and schools: A distributed behavioral model. *SIGGRAPH Comput. Graph. 21*, 4 (Aug.), 25–34.

REYNOLDS, C. W. 1999. Steering behaviors for autonomous characters. In *Proceedings of Game Developers Conference 1999*, Miller Freeman Game Group, San Francisco, California, GDC '99, 763–782.

RODRIGUEZ, S., AND AMATO, N. M. 2011. Roadmap-based level clearing of buildings. In *Proceedings of the 4th international conference on Motion in Games*, Springer-Verlag, Berlin, Heidelberg, MIG'11, 340–352.

SNAPE, J., VAN DEN BERG, J., GUY, S. J., AND MANOCHA, D. 2011. The hybrid reciprocal velocity obstacle. *Trans. Rob. 27*, 4 (Aug.), 696–706.

SNOOK, G. 2000. Simplified 3d movement and pathfinding using navigation meshes. In *Game Programming Gems*. Charles River Media, 288–304.

SUD, A., GAYLE, R., ANDERSEN, E., GUY, S., LIN, M., AND MANOCHA, D. 2007. Real-time navigation of independent agents using adaptive roadmaps. In *Proceedings of the 2007 ACM symposium on Virtual reality software and technology*, ACM, New York, NY, USA, VRST '07, 99–106.

TOZOUR, P. 2002. Ai game programming wisdom. In *Building a Near-Optimal Navigation Mesh*, S. Rabin, Ed. Charles River Media, 171–185.

VAN DEN BERG, J., LIN, M., AND MANOCHA, D. 2008. Reciprocal velocity obstacles for real-time multi-agent navigation. In *2008 IEEE International Conference on Robotics and Automation*, IEEE, 1928–1935.

VAN DEN BERG, J., PATIL, S., SEWALL, J., MANOCHA, D., AND LIN, M. 2008. Interactive navigation of multiple agents in crowded environments. In *Proceedings of the 2008 symposium on Interactive 3D graphics and games*, ACM, New York, NY, USA, I3D '08, 139–147.

VAN TOLL, W., COOK IV, A. F., AND GERAERTS, R. 2012. A navigation mesh for dynamic environments. *Journal of Visualization and Computer Animation 23*, 6, 535–546.

YOUNG, T. 2001. Expanded geometry for points-of-visibility pathfinding. In *Game Programming Gems 2*. Charles River Media, 317–323.