Clearance for diversity of agents' sizes in Navigation Meshes

Ramon Oliva[a,b], Nuria Pelechano[a]

[a] *Virvig, Universitat Politècnica de Catalunya*
[b] *EventLab, Universitat de Barcelona*

Abstract

There are two frequent artifacts in crowd simulation caused by navigation mesh design. The first appears when all agents attempt to traverse the navigation mesh and share the same way points through portals, thus increasing the probability of collisions with other agents or queues forming around portals. The second is caused by way points being assigned at locations where clearance is not guaranteed, which causes the agents to either walk too close to the static geometry, slide along walls or get stuck. To overcome this we use the full length of the portal and propose a novel method for dynamically calculating way points based on current trajectory, destination, and clearance, therefore guaranteeing that agents in a crowd will have different way points assigned. To achieve collision free paths we propose two novel techniques: the first provides the computation of paths with clearance for cells of any shape (even with concavities) and the second presents a new method for calculating portals with clearance, so that the dynamically assigned way points will always guarantee collision free paths relative to the static geometry. In this paper, we extend our previous work by describing a new version of the algorithm that is suitable for a larger number of navigation meshes, while further improving performance. Our results show how the combination of portals with exact clearance and dynamic way points improve local movement by reducing the number of collision between agents and the static geometry. We evaluate our algorithm with a variety of scenarios and compare our results with traditional way points to show that our technique also offers better use of the space by the agents.

Keywords: clearance, navigation meshes, dynamic way points

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. The algorithm implemented must also be efficient, as for a large scenario the paths for all characters need to be calculated within a small fraction of a second.

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 most 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 attempting to cross the portal from different directions. These perceptually unpleasant artifacts artificially reduce the flow rates through portals and the overall time for agents to reach their destination. An algorithm that can run in real time by assigning different way points to different characters can mitigate these issues.

Previous work is either bounded to a specific amount of clearance, only works with a specific type of navigation mesh (e.g. triangular meshes, medial axis), or calculates portal clearance on a per cell basis ignoring neighboring cells [1]. In contrast, our method is able to deal with an arbitrary amount of clearance and can work with any type of NavMesh. This applies even if cells are not strictly convex, or are too narrow.

Main Contributions. This paper 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. Finally, a general new technique to compute clearance over portals considering edges of neighboring cells. The algorithm is both straight forward and computationally efficient to allow the simulation of large 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* [2]) to find a path free of obstacles between two points in the scene.

The roadmap approach [3][4][5][6] 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 [5].

The navigation mesh approach [7][8][9][10][11][12] 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. NavMeshes have become more popular than roadmaps as the representation of the free space is more intuitive, clean, and provides a better description of location of the obstacles. We therefore focus on this environmental decomposition technique.

Local movement techniques aim to provide a mechanism for the autonomous characters to move from one location to the next in a path in a smooth and natural manner, while avoiding collisions with 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 [13][14][15][16][17][18]. The main problem of this approach is that characters tend to line up as they share the same attractor point over the portal. Some methods for achieving variety in characters' routes have been proposed. For example Pettre et. al. [19] presented a solution for roadmaps based on having a denser sampling of nodes, which allows for a better use of the free space at the expense of longer computational time. Other approaches using skeletons [20] allow for larger or smaller distances to the skeleton depending on crowd density. The problem with this later approach is that characters are spread as the density increases, but when densities are low they all tend to follow the same trajectories.

An improvement to traditional way points was introduced in [21] 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, this method does not properly address the problem of clearance, as it assumes that a cell is accessible by a character if the length of the portal that needs to be crossed is greater than or equal to 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 and guarantee 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 [22]. The main advantage of this approach is that every calculated path has the desired amount of clearance and as it is calculated offline, 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 either this size or smaller.

In [23], 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 [24], the Medial Axis of the set of obstacles is extracted to create a new data structure called the Explicit Corridor Map (ECM). The ECM computes 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 [25][26]. Straight skeletons have also been used to calculate roadmaps for path finding of multiple characters [20].

In [1] an algorithm to calculate paths with clearance for any type of NavMesh was introduced. However the algorithm calculated clearance on a per cell basis, ignoring the fact that in some navigation meshes with narrow cells, clearance may be defined by edges of neighboring cells. In this work, we extend the previous algorithm to make it suitable for a larger number of navigation meshes by introducing a recursive step, and we also present new techniques to improve efficiency for several steps of the algorithm.

3. Clearance Value of a Cell

Given a cell C, we define a cell cross as the pair ($\mathcal{P}_1$, $\mathcal{P}_2$) of 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 (see Figure 1). Note that it is not necessary to have strictly convex cells, as cells generated by NEOGEN [12] are allowed to have certain concavities depending on the convexity relaxation threshold chosen when creating the mesh.

The algorithm examines every portal endpoint and notch (i.e., a vertex such that its internal angle is greater than π) present in *stringLeft* and determines the closest edge in *stringRight*. The distance between the notch and the closest edge is the clearance value of this notch. Note that in this case, the endpoints of each string must be treated as if they were notches. If the closest edge to the notch is a portal edge, the algorithm recursively checks the distance between the notch and the edges lying in the adjacent cell through the portal. 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:

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

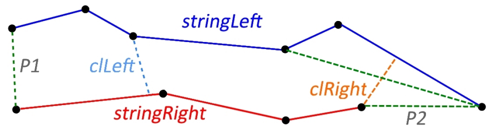

Figure 1: Clearance calculation for a given cell.

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 than or equal to 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 2).

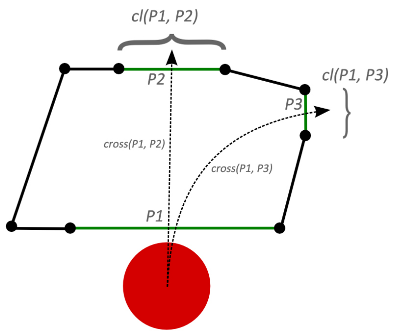

Figure 2: Example of different clearance depending on the crossing path through a cell.

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 (i.e. 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 shown in Figure 3.

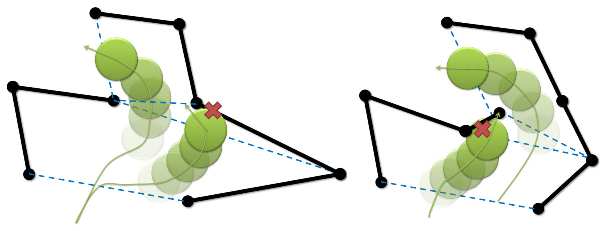

Figure 3: Examples of fixed way points that cause collisions. On the left, the way point is fixed at the center of a cell which causes a collision with the geometry. On the right the way point is assigned at a distance r of the portal endpoint also causing collision.

Let C_A be the cell where the character is currently located, 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 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 the portals that must be crossed to go from the current cell to the target cell in the path (usually portals in either C_A, C_B or their neigboring cells).
3. Limitation given by obstacle edges of the adjacent cells (or neighbors).

Cases 1 and 2 assume that the endpoints of portals are located over obstacles as occurs in most navigation meshes. In the case of grid based navigation meshes or when T-joints exist between portals, this would not be the case for certain portals and thus the algorithm should only consider those endpoints that are located over obstacles.

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 4. The resulting sub-segment $\mathcal{P}'$ has enough clearance only if the other edges in C_A and C_B are at a distance greater than or equal to r from $\mathcal{P}'$. If not , this sub-segment must be further refined to guarantee collision-free traversability.

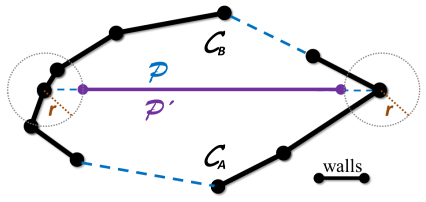

Figure 4: Example portal $\mathcal{P}$ that connects C_A and C_B. The shrunk portal $\mathcal{P}'$ is initialized by displacing the endpoints of the original portal $\mathcal{P}$ a distance r towards its center.

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 5 depicts this situation). C_A and C_B are thus two polygons with vertices 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. We refer to them as $\mathcal{P}_{AB}$ for the oriented edge that belongs to C_A, and $\mathcal{P}_{BA}$ for the oriented edge that belongs to C_B.

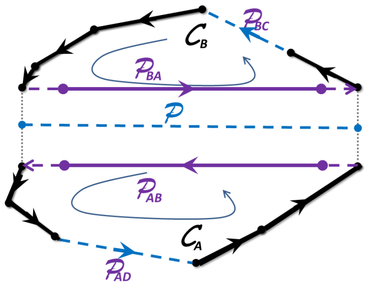

Figure 5: C_A and C_B separated by $\mathcal{P}$ are in fact two independent polygons with their vertices oriented in counter-clockwise order, so $\mathcal{P}$ is the overlapping of $\mathcal{P}_{AB}$ and $\mathcal{P}_{BA}$.

Case 2: Let 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 C_B with the next cell in the path C_C). In such a situation, it is possible that the endpoints of $\mathcal{P}'$ are determined by the endpoints of any exit portals in $\mathcal{P}_{E1}, ..., \mathcal{P}_{En}$, where $\mathcal{P}_{Ei}$ indicates a portal in the sequence of portals that needs to be crossed to go from C_B to the final cell in the path C_{Goal}. This occurs when one (or both) endpoint of a portal $\mathcal{P}_{Ei}$ 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 (of radius=agent's clearance) centered on the endpoints of the first exit portal $\mathcal{P}_{BC}$ intersects with $\mathcal{P}'$. If this intersection exists, we update $\mathcal{P}'$ accordingly and the process continues iteratively by checking the next exit portal. The algorithm stops when we find the first exit portal $\mathcal{P}_{Ei}$ that fails the test (none of its endpoints determines the endpoints of $\mathcal{P}'$) or when C_{Goal} is reached.

We take the polygons with oriented edges from Figure 5, and define $\mathcal{P}_{BA}[0]$ as the origin of the oriented portal $\mathcal{P}_{BA}$, and $\mathcal{P}_{BA}[1]$ as the end. As 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, so $\mathcal{P}_{BC}[0]$ 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 on $\mathcal{P}_{BC}[0]$ intersects $\mathcal{P}'_{BA}$ at a single point, then $\mathcal{P}'_{BA}[1]$ is set to be this intersection point.
- If the circumference centered on $\mathcal{P}_{BC}[0]$ intersects $\mathcal{P}'_{BA}$ at two points, then $\mathcal{P}'_{BA}[1]$ is set to be the intersection point that is furthest from $\mathcal{P}_{BA}[1]$.

Similarly, 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 6 shows the result of the algorithm using an example cell. Figure 7 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 (Figure 7 top) the way points assigned over the portals would continuously push the characters to collide with the static geometry.

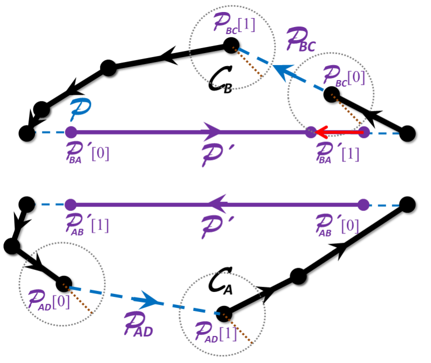

Figure 6: The endpoint $\mathcal{P}'_{BA}[1]$ of the entry portal is determined by the endpoint $\mathcal{P}_{BC}[0]$ of the exit portal, as the circumference centered on $\mathcal{P}_{BC}[0]$ intersects $\mathcal{P}'_{BA}$. The other end of $\mathcal{P}'_{BA}$ is not modified, as the circumference centered on $\mathcal{P}_{BC}[1]$ does not intersect $\mathcal{P}'_{BA}$.

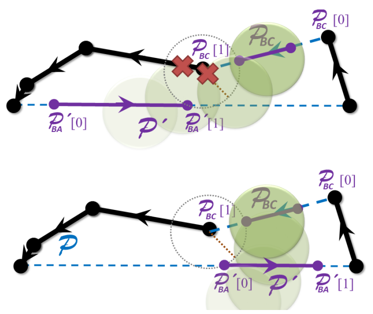

Figure 7: On the top we can see an example of what the character's trajectory would be if $\mathcal{P}'$ was not shrunk respecting the direction of the portals. The trajectory leads to a collision with the static geometry. On the bottom we can see the trajectory when clearance is calculated correctly.

Case 3: The final case to consider takes into account whether any obstacle edge limits the clearance of the portal. This can happen when an edge or portal of the current cell is at a distance smaller than the clearance value of the portal that we are shrinking. In the case of portals, the process must be repeated recursively.

Given a cell C_X, with 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 8).

The algorithm proceeds by displacing each edge $\vec{e}_{(i,i+1)}$ a distance of r units along 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 an intersection exists, the corresponding endpoint of $\mathcal{P}'_{BA}$ is updated depending on the direction of $\vec{e'}_{(i,i+1)}$ as follows:

1. if the edge is an obstacle edge:
   (a) If $v'_{(i)}$ is on the side of C_B and $v'_{(i+1)}$ is on the side of C_A, then $\mathcal{P}_{BA}[0]$ is set to be the intersection point.
   (b) If $v'_{(i)}$ is on the side of C_A and $v'_{(i+1)}$ is on the side of C_B, then $\mathcal{P}'_{BA}[1]$ is set to be the intersection point.
2. if the edge is a portal leading to C_D:
   (a) If $v'_{(i)}$ is on the side of C_B and $v'_{(i+1)}$ is on the side of C_A, then repeat the algorithm for the edges in C_D to update $\mathcal{P}_{BA}[0]$ if necessary.
   (b) If $v'_{(i)}$ is on the side of C_A and $v'_{(i+1)}$ is on the side of C_B, then repeat the algorithm for the edges in C_D to update $\mathcal{P}'_{BA}[1]$ if necessary.

Figure 8 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 9 shows the result of the algorithm.

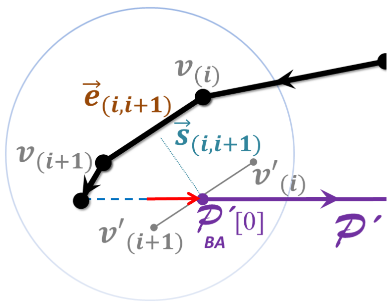

Figure 8: Close up of the top left of Figure 6 with the shrinking process due to displacing edges.

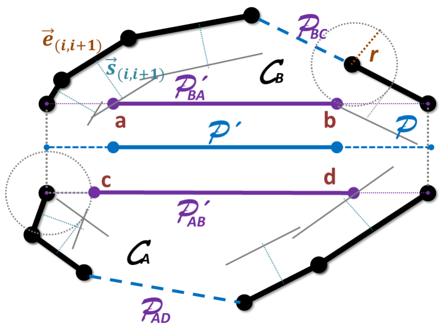

Figure 9: Final result $\mathcal{P}'$ after calculating the merging of the intermediate solutions $\mathcal{P}'_{AB}$ and $\mathcal{P}'_{BA}$. The resulting shrunk portal before merging illustrates the application of the three cases: Case 1 can be seen in c, Case 2 results in b and case 3 in a and d. $\mathcal{P}'$ 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. 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.

In general, Case 3 will always be the most restrictive and thus the key calculation, however there can be exceptions such as illustrated in Figure 10 where case 3 does not limit the clearance of the portal. Therefore all three cases are necessary, as if we simply use distance from endpoints we would fail to generate natural paths in certain scenarios.

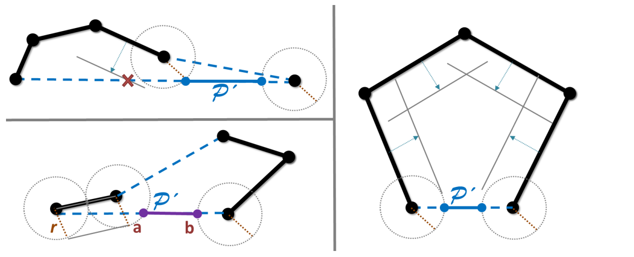

Figure 10: These examples show different situations where portal clearance is not defined simply by Case 3, and thus Cases 1 and 2 are necessary.

In Figure 11 we show an example where the recursive step would be necessary to compute exact clearance over the portal. Without recursivity the clearance on the left extreme of the portal would be given by the end point on the left hand side of the neighbouring portal, but with recursivity it is further reduced to the new intersection point a.

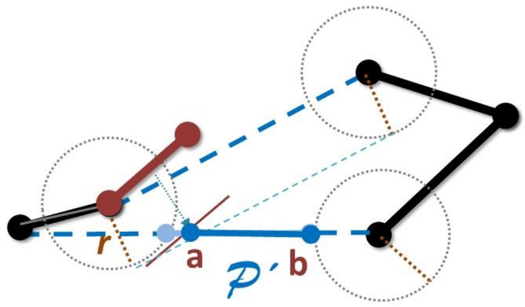

Figure 11: Example where the recursive step is necessary to compute clearance correctly.

Critical Radius:

All our calculations are required to perform in real time and we have already described an approach to speed up the system by storing information about clearance for agents of different radii. Other agents with radius similar to those already stored can then look up the information from a table instead of re-calculating. An additional technique implemented to speed up the process consists of pre-calculating a critical radius, ρ. The critical radius is defined as the maximum radius for which clearance depends exclusively on keeping a distance ρ from the portal endpoints. It is calculated by computing the minimum distance to an obstacle edge with its shrinking direction pointing towards the current portal. During run time, only agents of radius larger than ρ need to compute the portal clearance algorithm described in this section. Agents with radius r below ρ only need to keep a distance of r from the portal endpoints. As the critical radius is calculated off-line, this provides a speed up of 1.15 times faster on average during the real-time calculations. This speed up has been calculated over a variety of scenarios, most of them handmade to fully test the method. However in most of the scenarios obtained with NEOGEN, portal clearance is influenced exclusively by the portal endpoints, and thus the number of portals for which the full clearance algorithm needs to be executed will be minimal.

5. Dynamic Way Points

The method used to steer the character from one cell to another is a key aspect to create natural routes in navigation meshes. 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 directions and being forced to pass through the same point. A 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).

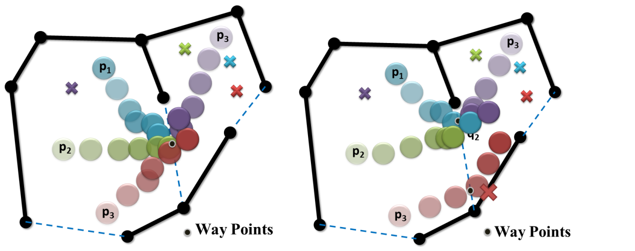

Figure 12: Typical lining up artifacts and bottlenecks when way points are set at either the center (left) or the closest endpoint of the portal (right).

Our dynamic way point 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 an 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. Each character's position approaching a portal will be different, so 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.

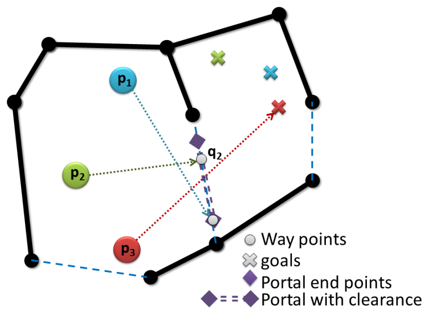

Figure 13: The attractor point of the red character is its own goal since it is visible from its current location. The green character has its orthogonal projection, $\mathbf{q}_2$, over the portal as its way point, whereas the blue character has the farthest away endpoint of the portal assigned as its 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 a 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. Local Movement

The local movement algorithm is based on a simple steering behavior with some extension to include physical forces as described in HiDAC [15]. Collision detection and repulsion forces between agents are calculated using the Bullet Physics Engine [27]. We have also used this library to perform calculations to speed up the detection of agents crossing portals. Agents move towards their next assigned dynamic way point while avoiding the static geometry and other moving obstacles. In order to keep track of the cell in which the character is located we have taken advantage of some of the features that the Bullet Physics Engine offers. By assigning a rigid body to the floor of each cell, we can efficiently compute the intersection between the character and the cells using Bullet's space partitioning.

This solves artifacts that usually appear when agents approach their assigned way point, and end up moving back and forth trying to reach the threshold distance to the target point. With our technique, a portal can be crossed at any point independently of the distance to their next assigned way point.

Note that with the method described above to detect when agents cross portals, we improve the local movement of both centered and dynamic way points. Traditional center way points require the agents to be a certain distance from the way point in order to assign the next portal. In many cases this leads to agents moving back and forth around portals as they attempt to reach a specific distance from an attractor. In our implementation this is not strictly necessary, as agents may cross portals despite not having reached their next way point. When this happens, they are immediately assigned to a new way point in the next portal without losing track of their current cell information. This avoids a common problem that arises in many simulations where the agents only update their current cell when they have reached their assigned way point, and thus agents may end up ”lost”.

7. Results

In order to evaluate the results obtained with our algorithm, we have carried out both qualitative and quantitative analysis. We have examined whether our clearance method combined with dynamic way points achieves a better use of space, and whether the performance of our algorithm is sufficient to work with large groups of agents in real time whilst computing paths with clearance and collision free way points.

Figure 14 (and the accompanying videos[1]) shows a comparison between using traditional way points (WP) at the center of portals and our method with dynamic way points (DWP) for two example scenarios. The first scenario is shaped as a donut and the second is 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. For each character, a random cell of the environment is selected as its destination cell. A path finding algorithm based on A* calculates the sequence of cells that the character needs to walk through to go from its current cell to the destination. Way points are assigned over portals connecting consecutive cells. Once a character reaches its destination cell, a new one is randomly assigned. Characters are considered to cross a portal as soon as the Bullet Physics Engine [27] detects that the character has arrived in the next cell of the path. Dynamic way points make better use of the space, use straight trajectories whenever possible and offer more natural looking trajectories for the characters, even when using a very simple rule based model 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 that they are more chaotic as characters bounce around portals trying to get close to the way point while avoiding each other.

Dynamic way points offer a better distribution of agents over portals which allows more agents to cross portals simultaneously. This increases flow rates through portals since it avoids artificial line formation. For example, in the donut scenario with 200 agents walking in the same direction, we observe 22% higher flow rates.

[1] `www.lsi.upc.edu/~npelechano/videos/C&G2014_Clearance.mov`

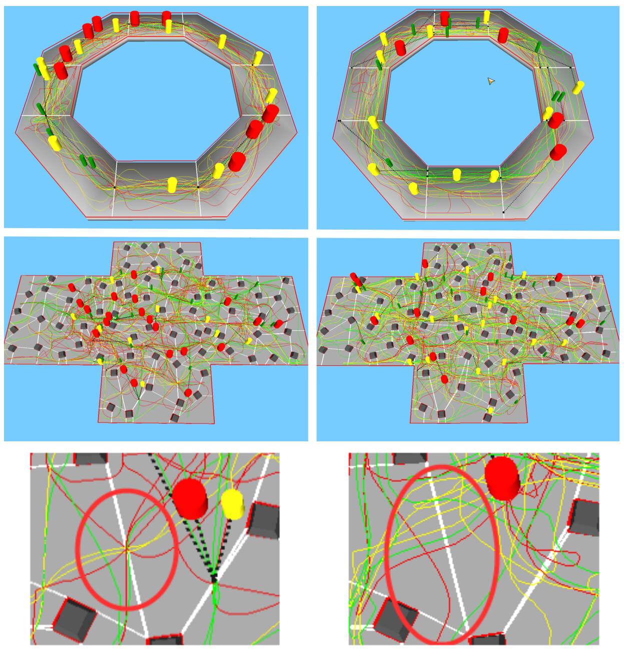
Figure 14: Comparison between having way points at the center of portals (on the left) and dynamic way points (on the right) for the donut scenario with 25 agents (top row), large cross scenario with 50 agents (middle row) and close up of the paths crossing a portal (bottom row).

7.1. Performance

The following results have been obtained in an Intel Core i7-3770 CPU @ 3.40GHz, 16GB of RAM, NVIDIA GeForce 680GTX . 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).

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 1.41 times the cost of the most efficient version (in this case *SimpleShrink(-)*) and 1.2 times for 2000 random queries. This means that the algorithm for calculating portals with exact clearance presented in this paper (*ExactShrink(+)*) is around

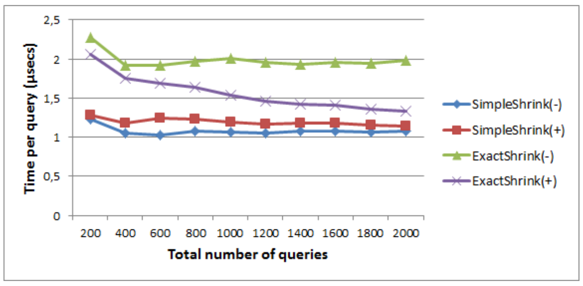

Figure 15: Comparison of the time taken per query (in microseconds) as the number of queries increases for the different shrinking techniques.

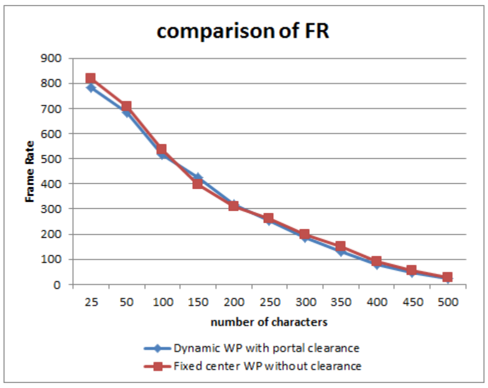

Figure 16: Average frame rates obtained in the large "'cross"' scenario as the number of characters increase for our method and a standard solution

20% more time consuming for 2000 queries than simpler implementations, but it also guarantees that every computed path will have enough clearance with the static geometry. 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. The time taken by the *ExactShrink(+)* algorithm converges towards the *SimpleShrink(+)* method.

It is also important to emphasize 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.

Including the recursive step when calculating clearance makes our method more robust without introducing a noticeable impact on the computational time.

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.

As 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 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). 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 within the overall simulation time. Both simulations can handle up to 500 characters in real time. Therefore we can claim that the computational cost of our technique is insignificant for the overall simulation time and that it provides results that are perceptually more convincing and make better use of the space, as shown in Figure 17 and the accompanying videos.

7.2. *Path finding*

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. 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 characters' size decreases, the final emerging trajectories of their color are wider, since their way points are assigned over larger shrunk portals.

7.3. *Comparison of dynamic collisions*

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, we have run several experiments to compare the average number of collisions for both fixed center way points and dynamic way points. We account for a collision between two rigid bodies at every tick of the physics engine (60x per second). Collisions are considered when an agent is in contact with the geometry (which also accounts for agents being stuck next to a wall due to a badly located way point)

As shown in Figure 18, for up to 100 agents the number of collisions between agents is almost zero, since at low densities there are not many chances of collisions and basic avoidance behavior can steer agents away from collisions. However once the densities start increasing we can observe how even when all the agents move in the same direction, collisions start appearing. As the graph shows, the number of collisions for fixed center WP is much higher than for DWP, since forcing all the agents to move towards the same point leads to chaotic behavior with loops in the agents' trajectories. This occurs for up to 175 agents for the donut scenario, since from this point onwards the density of agents in the environment is so high that bottlenecks are almost impossible to avoid.

In Figure 19 we can observe a comparison between the average number of collisions per clock tick as the number of agents increases for fixed centered versus dynamic way points. Our method to dynamically assign way points achieves a much

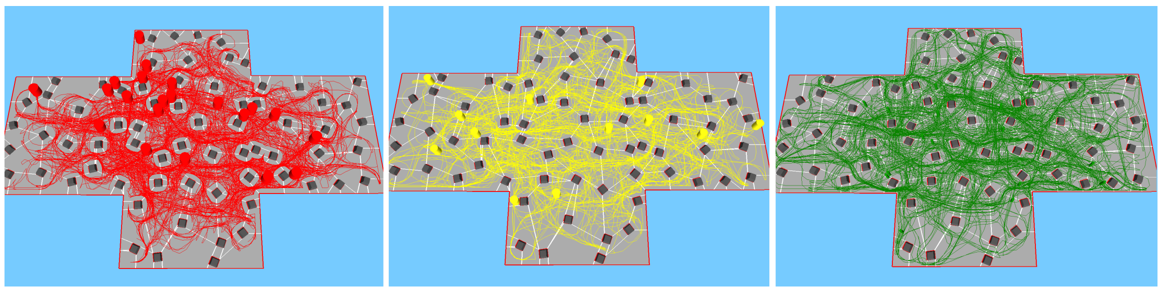

Figure 17: Trajectories followed by characters of different size. From left to right, the larger characters (red, $r = 2.0$) will not use the narrower portals and thus they can only walk through 97 of the 130 cells in the Navmesh, the medium characters can already get through most of the portals (yellow, $r = 1.5$) therefore being able to walk through 110 cells, and finally the smaller size characters (green, $r = 0.5$) can walk through all the portals having the largest shrunk portals (walkable cells=130).

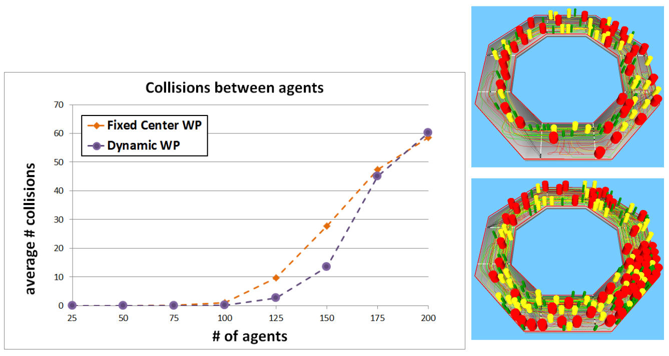

Figure 18: Comparing the average number of collisions per second between agents for the donut scenario as the number of agents increases. We compare dynamic way points against fixed center way points. On the top right we show the scenario with 100 agents and on the top bottom with 175 agents

lower number of collisions between agents which not only reduces artificial bottlenecks in the environment, but also results in smoother and more natural trajectories. As in the donut scenario, once the number of agents increases beyond 125, differences in the number of collisions start emerging between DWP and fixed center WP, until the total number is higher than 225. At this point, the high density of agents makes collisions inevitable, independent of the method used.

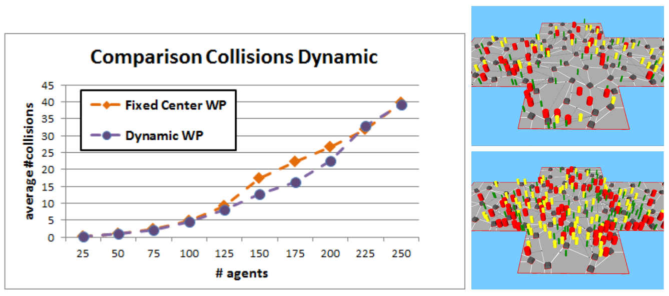

Figure 19: Comparing the average number of collisions per clock tick between agents for the cross scenario as the number of agents increases. We compare dynamic way points against fixed center way points. On the top right we have the cross scenario with 125 agents, and on the bottom right the same scenario with 225 agents.

While the graphs vary depending on the size of the scenario, length of portals and local navigation method, we observe that in all of our experiments, dynamic way points achieve better results than fixed center WP.

7.4. *Comparison of collisions against geometry*

The main advantage of having exact clearance calculations is that we guarantee that way points will only be assigned over portals where collision free paths exist. To evaluate this quantitatively, we have run several experiments using different scenarios and compared the following methods: (1) dynamic way points (DWP) over portals with exact clearance, (2) DWP over portals with simple clearance, and (3) fixed center way points. For the three methods, the local movement algorithm is the same, and the agents' goal cell is chosen randomly every time they reach their destination. For each case we have counted the number of collisions against the geometry that results from way points being badly assigned.

Obviously the results depend strongly on the quality of the portals created and the overall geometry. To show the potential of our method, we have designed scenarios with several examples of problematic portals (mostly ill-conditioned portals). Figure 20 shows the results of each of the methods in terms of paths followed by agents, and situations where they can easily get stuck trying to walk through a portal that does not guarantee clearance. As shown in Cases 2 and 3, agents may even get completely stuck against the geometry, whereas with our exact clearance method, agents are always steered towards way points that guarantee traversability. This holds even for maps with many ill-conditioned cells, such as the ones created manually for these experiments.

The quantitative results in terms of number of collisions against the geometry for this particular scenario are shown in Figure 21. The three methods use the same local movement algorithm, therefore the only difference comes from how and where way points are assigned. Our method outperforms previous work with regards to reducing the number of collisions against the geometry. We have performed comparisons for different crowd sizes. We have demonstrated that the differences become less significant as the crowd size increases. This occurs because there is a point where collisions are due to the high density of the crowd and not just the location of way points. In all cases, exact clearance provides the lowest number of collisions

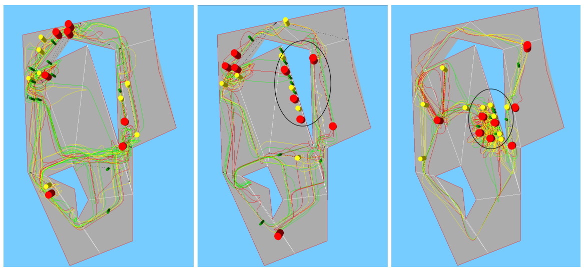
Figure 20: Comparing paths between the three methods. From left to right: (1) DWP over portals with exact clearance, (2) DWP over portals with simple clearance, and (3) fixed center way points. The areas where agents get stuck due to an ill-conditioned cell with a portal too close to the geometry (narrow cell) are circled.

against the geometry. If we compare fixed center against dynamic way points with simple clearance, fixed center performs better when it comes to avoiding collisions against the static geometry, since in most cases the center way point will be located at the furthest point from the geometry.

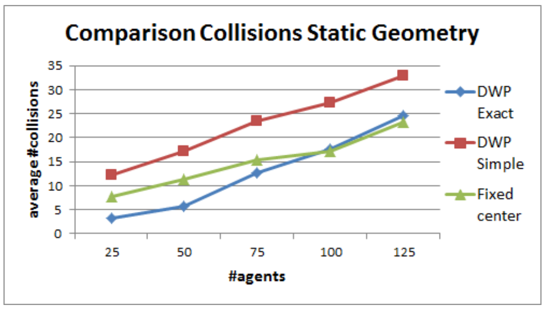

Figure 21: Average number of collisions against the geometry for each method tested (collisions counted at each clock tick, which corresponds to 60Hz).

Finally, Figure 22 shows the importance of using our exact clearance calculation when there are ill-conditioned cells. In this example we can see the portal calculated with our exact method against the simple method often used in video games. In both cases the segments over the portals that are traversable for each method are shown with a thin blue line. The character for which this clearance has been calculated is also circled in blue. In both examples, a red agent is trying to move from cell *A* to cell *B*. Our exact clearance algorithm provides the exact segment over the portal that can be crossed without collisions or errors. In the case of simple clearance, we can observe how the character is being steered towards a position that will lead to the wrong cell and to collisions against the geometry.

8. Conclusions

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 three 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,

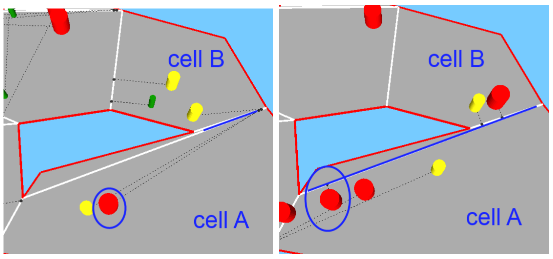

Figure 22: Clearance calculated with our exact algorithm (left) and with the simple clearance method (right).

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 based on the character position and hence, it mostly avoids two characters sharing the same attractor point.

Bullet Physics Engine [27] has been integrated in order to improve the overall quality of the simulation. Although its main purpose is to solve the collisions against moving and static geometry, we have used Bullet to efficiently detect when a portal crossing has been produced and avoided artifacts that arise in traditional methods as characters approach their target position.

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 [12] which is a NavMesh generator that provides an almost near-optimal number of cells with very few ill-conditioned cells. To show the potential of our method even for other kinds of NavMeshes, we have also manually generated navigation meshes with ill-conditioned cells.

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, avoid artificial line formation, reduce bottlenecks and collisions. In this paper we have also provided a quantitative evaluation of the improvements achieved with our exact clearance method by counting collisions against static and dynamic geometry. Results show how our method provides not only smoother paths with better usage of space, but also reduces the average number of collisions that are caused by way points not being correctly assigned. Compared to our previous work [1], we have made significants improvements in terms of generality as our new algorithm can handle a larger variety of navigation meshes, while improving performance with the introduction of the critical radius and a revised version of the code.

Acknowledgements

This work has been partially funded by the Spanish Ministry of Science and Innovation under Grant TIN2013-47137-C2-1-P.

References

[1] Oliva R, Pelechano N. A generalized exact arbitrary clearance technique for navigation meshes. In: Proceedings of Motion on Games. MIG '13; New York, NY, USA: ACM. ISBN 978-1-4503-2546-2; 2013, p. 103–10. doi:\bibinfo{doi}{10.1145/2522628.2522900}. URL `http://doi.acm.org/10.1145/2522628.2522900`.

[2] Hart PE, Nilsson NJ, Raphael B. A formal basis for the heuristic determination of minimum cost paths. Systems Science and Cybernetics, IEEE Transactions on 1968;4(2):100–7. doi:\bibinfo{doi}{10.1109/tssc.1968.300136}.

[3] Arikan O, Chenney S, Forsyth DA. Efficient multi-agent path planning. In: Proceedings of the Eurographic workshop on Computer animation and simulation. New York, NY, USA: Springer-Verlag New York, Inc. ISBN 3-211-83711-6; 2001, p. 151–62.

[4] Young T. Expanded geometry for points-of-visibility pathfinding. In: Game Programming Gems 2. Charles River Media; 2001, p. 317–23.

[5] Sud A, Gayle R, Andersen E, Guy S, Lin M, Manocha D. Real-time navigation of independent agents using adaptive roadmaps. In: Proceedings of the 2007 ACM symposium on Virtual reality software and technology. VRST '07; New York, NY, USA: ACM. ISBN 978-1-59593-863-3; 2007, p. 99–106.

[6] Rodriguez S, Amato NM. Roadmap-based level clearing of buildings. In: Proceedings of the 4th international conference on Motion in Games. MIG'11; Berlin, Heidelberg: Springer-Verlag. ISBN 978-3-642-25089-7; 2011, p. 340–52.

[7] Snook G. Simplified 3d movement and pathfinding using navigation meshes. In: Game Programming Gems. Charles River Media; 2000, p. 288–304.

[8] Tozour P. Ai game programming wisdom. In: Rabin S, editor. Building a Near-Optimal Navigation Mesh. Charles River Media; 2002, p. 171–85.

[9] Kallmann M. Path planning in triangulations. In: Proceedings of the IJCAI Workshop on Reasoning, Representation, and Learning in Computer Games. Edinburgh, Scotland; 2005,.

[10] Pettre J, Laumond JP, Thalmann D. A navigation graph for real-time crowd animation on multilayered and uneven terrain. In: Proceedings of the 1st International Workshop on Crowd Simulation. 2005, p. 81–90.

[11] van Toll W, Cook IV AF, Geraerts R. A navigation mesh for dynamic environments. Journal of Visualization and Computer Animation 2012;23(6):535–46.

[12] Oliva R, Pelechano N. Neogen: Near optimal generator of navigation meshes for 3d multi-layered environments. Computer And Graphics 2013;doi:\bibinfo{doi}{10.1016/j.cag.2013.03.004}. URL `http://www.sciencedirect.com/science/article/pii/S0097849313000435`.

[13] Reynolds CW. Flocks, herds and schools: A distributed behavioral model. SIGGRAPH Comput Graph 1987;21(4):25–34. doi:\bibinfo{doi}{10.1145/37402.37406}.

[14] Reynolds CW. Steering behaviors for autonomous characters. In: Proceedings of Game Developers Conference 1999. GDC '99; San Francisco, California: Miller Freeman Game Group; 1999, p. 763–82.

[15] Pelechano N, Allbeck JM, Badler NI. Controlling individual agents in high-density crowd simulation. In: Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation. SCA '07; Aire-la-Ville, Switzerland, Switzerland: Eurographics Association. ISBN 978-1-59593-624-0; 2007, p. 99–108.

[16] van den Berg J, Lin M, Manocha D. Reciprocal velocity obstacles for real-time multi-agent navigation. In: 2008 IEEE International Conference on Robotics and Automation. IEEE. ISBN 978-1-4244-1646-2; 2008, p. 1928–35. doi:\bibinfo{doi}{10.1109/robot.2008.4543489}. URL `http://gamma.cs.unc.edu/RVO/`.

[17] van den Berg J, Patil S, Sewall J, Manocha D, Lin M. Interactive navigation of multiple agents in crowded environments. In: Proceedings of the 2008 symposium on Interactive 3D graphics and games. I3D '08; New York, NY, USA: ACM. ISBN 978-1-59593-983-8; 2008, p. 139–47. doi:\bibinfo{doi}{10.1145/1342250.1342272}. URL `http://doi.acm.org/10.1145/1342250.1342272`.

[18] Snape J, van den Berg J, Guy SJ, Manocha D. The hybrid reciprocal velocity obstacle. Trans Rob 2011;27(4):696–706. doi:\bibinfo{doi}{10.1109/TRO.2011.2120810}. URL `http://dx.doi.org/10.1109/TRO.2011.2120810`.

[19] Pettre J, Thalmann D. Path planning for crowds: From shared goals to individual behaviors. In: Eurographics Short Presentations. 2005,.

[20] Haciomeroglu M, Laycock RG, Day AM. Distributing pedestrians in a virtual environment. vol. 0. Los Alamitos, CA, USA: IEEE Computer Society. ISBN 0-7695-3005-2; 2007, p. 152–9. doi:\bibinfo{doi}{http://doi.ieeecomputersociety.org/10.1109/CW.2007.9}.

[21] Curtis S, Snape J, Manocha D. Way portals: efficient multi-agent navigation with line-segment goals. In: Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games. I3D '12; New York, NY, USA: ACM. ISBN 978-1-4503-1194-6; 2012, p. 15–22.

[22] Mononen M. Recast navigation toolkit; 2009. http://code.google.com/p/recastnavigation/.

[23] Kallmann M. Shortest paths with arbitrary clearance from navigation meshes. In: Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation. SCA '10; Aire-la-Ville, Switzerland, Switzerland: Eurographics Association; 2010, p. 159–68.

[24] Geraerts R. 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; 2010, p. 1997–2004. doi:\bibinfo{doi}{http://dx.doi.org/10.1109/ROBOT.2010.5509263}.

[25] Karamouzas I, Geraerts R, Overmars M. Indicative routes for path planning and crowd simulation. In: Proceedings of the 4th International Conference on the Foundations of Digital Games. 2009, p. 113–20.

[26] Jaklin N, Cook IV AF, Geraerts R. Real-time path planning in heterogeneous environments. Computer Animation and Virtual Worlds 2013;5(24):285–95.

[27] Coumans E. Bullet physics library; 2013. http://bulletphysics.org/.