# Clearance for diversity of agents' sizes in Navigation Meshes

Ramon Oliva<sup>a,b</sup>, Nuria Pelechano<sup>a</sup>

<sup>a</sup>Virvig, Universitat Politècnica de Catalunya <sup>b</sup>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 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-andPortal 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 <sup>27</sup> for all characters need to be calculated within a small fraction <sup>28</sup> 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 or un 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 eclearance, only works with a specific type of navigation mesh (e.g. triangular meshes, medial axis), or calculates portal cleartrast, 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 naviga<sup>55</sup> new technique to compute clearance over portals considering <sup>109</sup> sumes that a cell is accessible by a character if the length of the <sup>56</sup> edges of neighboring cells. The algorithm is both straight for-<sup>110</sup> portal that needs to be crossed is greater than or equal to the 57 ward and computationally efficient to allow the simulation of 111 diameter of the character, which is not always the case as we 58 large crowds.

#### 59 2. Related Work

Path planning of autonomous characters in virtual environ-60 61 ments is a central problem in the fields of robotics, videogames, 62 and crowd simulation. The most popular solutions are based on 63 a combination of global and local movement techniques.

The target of global navigation techniques is to provide a 65 representation of the free space of the scene that is usually ob-66 tained by either constructing a roadmap or a navigation mesh. 67 The main objective of both approaches is to generate a graph <sup>68</sup> that can be used by a search algorithm (usually  $A^*$  [2]) to find 69 a path free of obstacles between two points in the scene.

The roadmap approach [3][4][5][6] captures the connectiv-71 ity of the free space by using a network of standardized paths 72 (lines, curves). The main limitation of this representation is 73 that it does not describe the geometry of the scene, nor where 74 the obstacles are. Consequently, avoidance of dynamic obsta-75 cles is usually a hard task and not always possible, as exposed 76 in [5].

The navigation mesh approach [7][8][9][10][11][12] con-77 78 sists of the partition of the navigable space of the scene into con-79 vex regions, guaranteeing that a character can move between 80 two points of the same cell following a straight line, without 81 getting stuck in local minima. NavMeshes have become more <sup>82</sup> popular than roadmaps as the representation of the free space is 83 more intuitive, clean, and provides a better description of loca-<sup>84</sup> tion of the obstacles. We therefore focus on this environmental <sup>85</sup> decomposition technique.

Local movement techniques aim to provide a mechanism 86 87 for the autonomous characters to move from one location to the <sup>88</sup> next in a path in a smooth and natural manner, while avoiding 89 collisions with dynamic obstacles. These methods are generally <sup>90</sup> driven by setting way points within the portals of the NavMesh 91 that work as attractors to steer the agents in the right direc-<sup>92</sup> tion [13][14][15][16][17][18]. The main problem of this ap-<sup>93</sup> proach is that characters tend to line up as they share the same <sup>94</sup> attractor point over the portal. Some methods for achieving va-95 riety in characters' routes have been proposed. For example 96 Pettre et. al. [19] presented a solution for roadmaps based on 97 having a denser sampling of nodes, which allows for a better <sup>98</sup> use of the free space at the expense of longer computational 99 time. Other approaches using skeletons [20] allow for larger 100 or smaller distances to the skeleton depending on crowd den-<sup>101</sup> sity. The problem with this later approach is that characters are <sup>102</sup> spread as the density increases, but when densities are low they all tend to follow the same trajectories. 103

An improvement to traditional way points was introduced 104 <sup>105</sup> in [21] by using way portals where the whole length of the por-<sup>106</sup> tal can be used to attract the local movement of the agents, thus 107 resulting in more natural looking paths. However, this method

54 tion meshes consisting of cells of any shape. Finally, a general 108 does not properly address the problem of clearance, as it as-112 will show in this paper.

> In order to carry out path planning and guarantee that the re-114 sulting paths will have an arbitrary amount of clearance, a com-<sup>115</sup> mon solution consists of enlarging the obstacles by a specific 116 amount of clearance known as the Minkowski sum. An exam-<sup>117</sup> ple of an application using this method is Recast [22]. The main 118 advantage of this approach is that every calculated path has the 119 desired amount of clearance and as it is calculated offline, it 120 does not have an impact on the performance of the path finding 121 algorithm being used. However, its major drawback is that it is 122 bounded to a specific value of clearance, so all characters must 123 have either this size or smaller.

> In [23], Kallman introduced a new type of triangulation 124 125 called Local Clearance Triangulation (LCT) that allows paths 126 to be computed free of obstacles with arbitrary clearance. Such 127 triangulation is obtained by a process that iteratively refines the Constrained Delaunay Triangulation (CDT) resulting from the 129 starting set of obstacles. The resulting structure determines if 130 there exists a path free of obstacles for a given clearance value. 131 However, it introduces more cells in the partition of the scene, <sup>132</sup> thus dropping the performance of the path finding algorithm. 133 Another limitation of the method is that it only works for the de-134 scribed LCT but cannot be generalized to any navigation mesh.

> In [24], the Medial Axis of the set of obstacles is extracted 136 to create a new data structure called the Explicit Corridor Map 137 (ECM). The ECM computes the shortest path, the path that has 138 the largest amount of clearance, or any path in between. This 139 work has been further extended to calculate a finer set of at-140 tractors to obtain smoother paths [25][26]. Straight skeletons 141 have also been used to calculate roadmaps for path finding of 142 multiple characters [20].

> In [1] an algorithm to calculate paths with clearance for any 144 type of NavMesh was introduced. However the algorithm cal-145 culated clearance on a per cell basis, ignoring the fact that in 146 some navigation meshes with narrow cells, clearance may be 147 defined by edges of neighboring cells. In this work, we extend 148 the previous algorithm to make it suitable for a larger number of 149 navigation meshes by introducing a recursive step, and we also 150 present new techniques to improve efficiency for several steps 151 of the algorithm.

## 152 3. Clearance Value of a Cell

Given a cell C, we define a cell cross as the pair ( $\mathcal{P}_1, \mathcal{P}_2$ ) 153 <sup>154</sup> of C, where  $\mathcal{P}_1$  is the entry portal and  $\mathcal{P}_2$  is the exit portal. 155 We classify the obstacle edges of the cell into edges to the left 156 (stringLeft) and edges to the right (stringRight) in respect to the 157 path that crosses the cell from the entry portal to the exit portal 158 (see Figure 1). Note that it is not necessary to have strictly con-159 vex cells, as cells generated by NEOGEN [12] are allowed to 160 have certain concavities depending on the convexity relaxation <sup>161</sup> 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  $\pi$ ) 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 clostes est edge to the notch is a portal edge, the algorithm recursively exclusion the adjacent cell through the portal. The clearance value of the transformed the right string  $cl_L$  is the minimum of those distances. To compute the clearance value of the right string  $cl_R$ , we proceed in the transformed as follows:

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



Figure 1: Clearance calculation for a given cell.

It is only necessary to check the distance of the notches of 177 the string against the edges of the opposite string, as in the case 178 of a convex vertex, the distance to the opposite string must be 179 greater than or equal to the clearance value of the cell. This pro-180 cess is done off-line once the NavMesh of the virtual scenario 181 has been generated and, for each cell, we store in a table the 182 clearance value of every possible cell cross. This is because it 183 is possible to have a cell with three or more portals, where an 184 agent with a large radius can walk for example from portal  $\mathcal{P}_1$ 185 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.

#### 186 4. Finding Portals with Enough Clearance

<sup>187</sup> In order to avoid artifacts such as characters bouncing, slid-<sup>188</sup> ing or getting stuck on the edges of the geometry, we should <sup>189</sup> only assign way points that have enough clearance (i.e. that <sup>190</sup> they have the required distance from the static geometry for <sup>191</sup> the character to traverse the portal without collision). Note that



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.

fixing way points to the center of the cell does not always guar-antee collision free traversals, as shown in Figure 3.

Let  $C_A$  be the cell where the character is currently located, 195  $C_B$  be the next cell in the path and  $\mathcal{P}$  the portal that joins both 196 cells. We want to calculate the sub-segment  $\mathcal{P}'$  of  $\mathcal{P}$  such that 197 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 locos cated 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 rate 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

201

202

203

205

206

207

222 for each cell in counter-clock wise order (Figure 5 depicts this 263 be updated. Figure 6 shows the result of the algorithm using  $_{223}$  situation).  $C_A$  and  $C_B$  are thus two polygons with vertices given  $_{264}$  an example cell. Figure 7 illustrates the importance of respect- $_{224}$  in counter-clockwise order.  $\mathcal{P}$  can then be treated as two iden- $_{265}$  ing the ordering of the portals when calculating portals with 225 tical overlapping segments given in opposite order depending 266 clearance. Even though in both cases the characters can walk 226 on which cell they belong to. We refer to them as  $\mathcal{P}_{AB}$  for the 267 through the portals, in the first case (Figure 7 top) the way  $_{227}$  oriented edge that belongs to  $C_A$ , and  $\mathcal{P}_{BA}$  for the oriented edge  $_{268}$  points assigned over the portals would continuously push the <sup>228</sup> 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 229 230 path (i.e., a cell that is neither the starting cell of the path nor <sup>231</sup> the final one). In this case we have to cross the cell by crossing <sup>232</sup> two portals, an entry portal  $\mathcal{P}$  and an exit portal  $\mathcal{P}_{BC}$  (portal that <sup>233</sup> connects cell  $C_B$  with the next cell in the path  $C_C$ ). In such a sit-234 uation, it is possible that the endpoints of  $\mathcal{P}'$  are determined by <sup>235</sup> the endpoints of any exit portals in  $\mathcal{P}_{E1}, ..., \mathcal{P}_{En}$ , where  $\mathcal{P}_{Ei}$  indi-236 cates a portal in the sequence of portals that needs to be crossed <sup>237</sup> to go from  $C_B$  to the final cell in the path  $C_{Goal}$ . This occurs <sup>238</sup> when one (or both) endpoint of a portal  $\mathcal{P}_{Ei}$  is at a distance less 239 than or equal to the desired clearance value from the entry por-<sup>240</sup> tal  $\mathcal{P}$ . To handle this situation, we check if a circumference (of 241 radius=agent's clearance) centered on the endpoints of the first 242 exit portal  $\mathcal{P}_{BC}$  intersects with  $\mathcal{P}'$ . If this intersection exists, we 243 update  $\mathcal{P}'$  accordingly and the process continues iteratively by 244 checking the next exit portal. The algorithm stops when we find 245 the first exit portal  $\mathcal{P}_{Ei}$  that fails the test (none of its endpoints <sup>246</sup> determines the endpoints of  $\mathcal{P}'$ ) or when  $C_{Goal}$  is reached.

We take the polygons with oriented edges from Figure 5, <sup>248</sup> and define  $\mathcal{P}_{BA}[0]$  as the origin of the oriented portal  $\mathcal{P}_{BA}$ , and 249  $\mathcal{P}_{BA}[1]$  as the end. As the portals are also given in counter-<sup>250</sup> clockwise order, we can state that the origin of any portal can <sup>251</sup> only limit the clearance of the end of the portal for which we <sup>252</sup> are calculating clearance, so  $\mathcal{P}_{BC}[0]$  can only shorten  $\mathcal{P}'_{BA}[1]$ , <sup>253</sup> and  $\mathcal{P}_{BC}$  [1] can only shorten  $\mathcal{P}'_{BA}$  [0]. The algorithm to further <sup>275</sup> recursively.  $_{^{254}}$  shorten  $\mathcal{P'}_{BA}$  continues through the following two cases:

- If the circumference centered on  $\mathcal{P}_{BC}[0]$  intersects  $\mathcal{P}'_{BA}$ 255 at a single point, then  $\mathcal{P}'_{BA}[1]$  is set to be this intersection 256 point. 257
- If the circumference centered on  $\mathcal{P}_{BC}[0]$  intersects  $\mathcal{P}'_{BA}$ 258 at two points, then  $\mathcal{P}'_{BA}[1]$  is set to be the intersection 259 point that is furthest from  $\mathcal{P}_{BA}$  [1]. 260

26  $_{262}$  for intersections against  $\mathcal{P'}_{BA}$  to determine if  $\mathcal{P'}_{BA}[0]$  needs to

<sup>269</sup> 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 270 271 any obstacle edge limits the clearance of the portal. This can <sup>272</sup> happen when an edge or portal of the current cell is at a dis-<sup>273</sup> tance smaller than the clearance value of the portal that we are 274 shrinking. In the case of portals, the process must be repeated

Given a cell  $C_X$ , with a set of vertices in counter-clockwise 277 order  $\{v_0, v_1, ..., v_n\}$ , where each consecutive pair of vertices in <sup>278</sup> the sequence defines an oriented edge of the cell, i.e.  $\vec{e}_{(i,i+1)}$  is <sup>279</sup> the edge starting in vertex  $v_i$  and ending in vertex  $v_{i+1}$ , for i =<sup>280</sup> [0, *n* – 1]. We define the shrinking direction of an edge,  $\vec{s}_{(i,i+1)}$ , 281 as the unit vector perpendicular to the edge with its direction <sup>282</sup> pointing towards the interior of the cell (Figure 8).

The algorithm proceeds by displacing each edge  $\vec{e}_{(i,i+1)}$  a 283 <sup>284</sup> distance of *r* units along its shrinking direction,  $\vec{s}_{(i,i+1)}$ , if the Similarly, a circumference centered on  $\mathcal{P}_{BC}$  [1] is checked <sup>285</sup> shrinking direction points towards the portal (otherwise there is <sup>286</sup> no chance of intersection). After displacement we obtain v'(i)

<sup>288</sup> For each displaced edge  $\vec{e'}_{(i,i+1)}$ , we calculate its intersection <sup>314</sup> not need to be computed again.  $_{289}$  against  $\mathcal{P}'_{BA}$ , and if such an intersection exists, the correspond-  $_{315}$ <sup>291</sup>  $\vec{e'}_{(i,i+1)}$  as follows:

- 1. if the edge is an obstacle edge: 292
- 293 294
- 295 296
- 2. if the edge is a portal leading to  $C_D$ : 297
- (a) If  $v'_{(i)}$  is on the side of  $C_B$  and  $v'_{(i+1)}$  is on the side of 298  $C_A$ , then repeat the algorithm for the edges in  $C_D$  to 299 update  $\mathcal{P}_{BA}[0]$  if necessary. 300
- (b) If v'<sub>(i)</sub> is on the side of C<sub>A</sub> and v'<sub>(i+1)</sub> is on the side of C<sub>B</sub>, then repeat the algorithm for the edges in C<sub>D</sub> to 30 302 update  $\mathcal{P}'_{BA}[1]$  if necessary. 303

Figure 8 shows this process over the example scenario with 304 305 a magnified view of the area of interest. The same process is <sup>306</sup> performed for  $\mathcal{P}'_{AB}$  and finally,  $\mathcal{P}'$  is computed as the resulting <sup>307</sup> sub-segment of the intersection between  $\mathcal{P}'_{AB}$  and  $\mathcal{P}'_{BA}$ . Every <sup>308</sup> point in  $\mathcal{P}'$  is guaranteed to have enough clearance. Figure 9 309 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 310 311 the result of the transformation for a particular value of clear-<sup>312</sup> ance in a table. The next time that the portal needs to be shrunk,

and v'(i + 1) as the results of displacing vertices  $v_i$  and  $v_{i+1}$ . 313 the table is checked for that particular clearance value so it does

In general, Case 3 will always be the most restrictive and  $_{290}$  ing endpoint of  $\mathcal{P}'_{BA}$  is updated depending on the direction of  $_{316}$  thus the key calculation, however there can be exceptions such 317 as illustrated in Figure 10 where case 3 does not limit the clear-318 ance of the portal. Therefore all three cases are necessary, as if (a) If v'<sub>(i)</sub> is on the side of C<sub>B</sub> and v'<sub>(i+1)</sub> is on the side of C<sub>A</sub>, then P<sub>BA</sub> [0] is set to be the intersection point.
(b) If v'<sub>(i)</sub> is on the side of C<sub>A</sub> and v'<sub>(i+1)</sub> is on the side of C<sub>B</sub>, then P'<sub>BA</sub> [1] is set to be the intersection point.
f the adge is a perturbative C



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 <sup>322</sup> would be necessary to compute exact clearance over the portal. 323 Without recursivity the clearance on the left extreme of the por-324 tal would be given by the end point on the left hand side of the 325 neighbouring portal, but with recursivity it is further reduced to  $_{326}$  the new intersection point *a*.



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

#### **Critical Radius:** 327

All our calculations are required to perform in real time and 328 329 we have already described an approach to speed up the sys-330 tem by storing information about clearance for agents of dif-331 ferent radii. Other agents with radius similar to those already 332 stored can then look up the information from a table instead of 333 re-calculating. An additional technique implemented to speed <sup>334</sup> up the process consists of pre-calculating a critical radius,  $\rho$ . 335 The critical radius is defined as the maximum radius for which  $_{336}$  clearance depends exclusively on keeping a distance  $\rho$  from the <sup>337</sup> portal endpoints. It is calculated by computing the minimum 338 distance to an obstacle edge with its shrinking direction point-339 ing towards the current portal. During run time, only agents  $_{340}$  of radius larger than  $\rho$  need to compute the portal clearance  $_{341}$  algorithm described in this section. Agents with radius r be-  $_{383}$  $_{342}$  low  $\rho$  only need to keep a distance of r from the portal end- $_{384}$  ment algorithm used to steer them. This algorithm will natu-343 points. As the critical radius is calculated off-line, this provides 385 rally move characters away from each other to avoid collision. <sup>344</sup> a speed up of 1.15 times faster on average during the real-time <sup>386</sup> Each character's position approaching a portal will be different, 345 calculations. This speed up has been calculated over a variety 387 so their projection over the portal will also be different making 346 of scenarios, most of them handmade to fully test the method. 388 it virtually impossible for two different characters to share the <sup>347</sup> However in most of the scenarios obtained with NEOGEN, por-<sup>389</sup> same attractor point over the portal if the characters are at risk <sup>348</sup> tal clearance is influenced exclusively by the portal endpoints, <sup>390</sup> of colliding. 349 and thus the number of portals for which the full clearance al-350 gorithm needs to be executed will be minimal.

## 351 5. Dynamic Way Points

The method used to steer the character from one cell to 353 another is a key aspect to create natural routes in navigation <sup>354</sup> meshes. When way points are assigned at a fixed position, usu-355 ally the center of the portals, animation artifacts arise (Figure 12). The most common artifacts are line formation among char-357 acters that move in the same direction, and bottlenecks caused 358 by characters crossing cells in opposite directions and being forced to pass through the same point. A typical approach in 359 video games consists of setting the way points at a distance r360  $_{361}$  from the closest endpoint of the portal (where r is the radius <sup>362</sup> of the character). This solution provides slightly more natural 363 paths since paths are apparently shorter and at least two way <sup>364</sup> points are available for each portal, but it does not completely 365 solve the problem. Our work focuses on dynamically calculat-<sup>366</sup> ing 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 <sup>368</sup> of the character within the cell. First of all, we check if the 369 goal position of the character is visible from its current position <sup>370</sup> (i.e., the segment joining the current and the goal position of the <sup>371</sup> character only produces an intersection with portal edges). In <sup>372</sup> that case, the attractor point is simply the goal position. If the segment does intersect with at least one obstacle edge, we need 373 to compute a way point over the next portal in the path to steer 374 the character towards the next cell of the path. Our target is to avoid characters having the same attractor point, so we compute 376 the orthogonal projection point  $\mathbf{q}$  of the current position of the 377 character **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 **q** lies  $_{\texttt{380}}$  outside the limits of  $\mathcal{P}',$  then the furthest endpoint of  $\mathcal{P}'$  with <sup>381</sup> respect to the current position of the character is selected as a <sup>382</sup> temporal attractor, until **q** is valid.

The position of the characters is given by the local move-



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 392 invalid, the furthest endpoint of  $\mathcal{P}'$  is a better candidate as a <sup>393</sup> temporal attractor than the closest one. This is because when 394 the steering attractor is the closest endpoint, the character tends <sup>395</sup> to move too close to the walls, producing a bad quality route.

# 396 6. Local Movement

The local movement algorithm is based on a simple steer-<sup>398</sup> ing behavior with some extension to include physical forces as 399 described in HiDAC [15]. Collision detection and repulsion 400 forces between agents are calculated using the Bullet Physics 401 Engine [27]. We have also used this library to perform cal-402 culations to speed up the detection of agents crossing portals. 403 Agents move towards their next assigned dynamic way point <sup>404</sup> while avoiding the static geometry and other moving obstacles. 405 In order to keep track of the cell in which the character is lo-406 cated we have taken advantage of some of the features that the 407 Bullet Physics Engine offers. By assigning a rigid body to the 408 floor of each cell, we can efficiently compute the intersection 409 between the character and the cells using Bullet's space parti-410 tioning.

This solves artifacts that usually appear when agents ap-412 proach their assigned way point, and end up moving back and <sup>413</sup> forth trying to reach the threshold distance to the target point. 414 With our technique, a portal can be crossed at any point inde-<sup>415</sup> pendently of the distance to their next assigned way point.

Note that with the method described above to detect when 417 agents cross portals, we improve the local movement of both 418 centered and dynamic way points. Traditional center way points <sup>419</sup> require the agents to be a certain distance from the way point <sup>420</sup> in order to assign the next portal. In many cases this leads to 421 agents moving back and forth around portals as they attempt to 422 reach a specific distance from an attractor. In our implemen-<sup>423</sup> tation this is not strictly necessary, as agents may cross portals 424 despite not having reached their next way point. When this hap-425 pens, they are immediately assigned to a new way point in the 426 next portal without losing track of their current cell information. 427 This avoids a common problem that arises in many simulations <sup>428</sup> where the agents only update their current cell when they have 429 reached their assigned way point, and thus agents may end up 430 "lost".

# 431 7. Results

432 In order to evaluate the results obtained with our algorithm, we have carried out both qualitative and quantitative analysis. 433 We have examined whether our clearance method combined 434 435 with dynamic way points achieves a better use of space, and whether the performance of our algorithm is sufficient to work 436 with large groups of agents in real time whilst computing paths 437 with clearance and collision free way points. 438

Figure 14 (and the accompanying videos<sup>1</sup>) shows a comparison between using traditional way points (WP) at the center 440 441 of portals and our method with dynamic way points (DWP) for 442 two example scenarios. The first scenario is shaped as a donut 443 and the second is shaped as a cross with static obstacles ran-444 domly located. The local movement algorithm is the same for 445 all scenarios, and it is based on a simple rule based model with 446 collision avoidance, steering towards attractors (way points) and 447 collision response. For each character, a random cell of the en-<sup>448</sup> vironment is selected as its destination cell. A path finding al-449 gorithm based on A\* calculates the sequence of cells that the 450 character needs to walk through to go from its current cell to 451 the destination. Way points are assigned over portals connect-452 ing consecutive cells. Once a character reaches its destination 453 cell, a new one is randomly assigned. Characters are consid-<sup>454</sup> ered to cross a portal as soon as the Bullet Physics Engine [27] 455 detects that the character has arrived in the next cell of the path. 456 Dynamic way points make better use of the space, use straight 457 trajectories whenever possible and offer more natural looking 481 458 trajectories for the characters, even when using a very simple rule based model for their local movement. When way points 460 are fixed at the center of portals, we can observe that not only do 461 the paths not make use of the available space but also that they <sup>462</sup> are more chaotic as characters bounce around portals trying to get close to the way point while avoiding each other. 463

Dynamic way points offer a better distribution of agents 464 465 over portals which allows more agents to cross portals simul-466 taneously. This increases flow rates through portals since it 467 avoids artificial line formation. For example, in the donut sce-<sup>468</sup> nario with 200 agents walking in the same direction, we observe 469 22% higher flow rates.



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).

#### 470 7.1. Performance

475

476

477

480

482

483

484

The following results have been obtained in an Intel Core 471 472 17-3770 CPU @ 3.40GHz, 16GB of RAM, NVIDIA GeForce 473 680GTX . Figure 15 compares the time spent per query (mi-<sup>474</sup> croseconds) 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 <sup>486</sup> query, we randomly chose a cell of the NavMesh, a trajectory 487 to cross this cell (i.e. an entry portal and an exit portal) and a <sup>488</sup> clearance value (0.5, 1 or 1.5).

The results of this experiment highlight the efficiency of 490 our exact clearance method (*ExactShrink*(+)). The efficiency <sup>491</sup> of the algorithm increases with the number of queries as the 492 chance of producing a redundant query is higher, and even-493 tually, every query will be redundant. Results show that for 494 the case of 1000 random queries, the cost of *ExactShrink(+)* 495 is just 1.41 times the cost of the most efficient version (in this 496 case SimpleShrink(-)) and 1.2 times for 2000 random queries. 497 This means that the algorithm for calculating portals with ex-498 act clearance presented in this paper (ExactShrink(+)) is around

<sup>&</sup>lt;sup>1</sup>www.lsi.upc.edu/~npelechano/videos/C&G2014\_Clearance. mov



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

499 20% more time consuming for 2000 queries than simpler implementations, but it also guarantees that every computed path 500 will have enough clearance with the static geometry. As the number of queries increases, this percentage is further reduced. 502 For the given example, we get a probability of hit of 50% for 503 1000 queries, which means that one in two queries does not 504 need to be computed since it is already stored in the lookup table, and 90% probability of hit when it reaches 6000. The time 506  $_{507}$  taken by the *ExactShrink*(+) algorithm converges towards the *SimpleShrink(+)* method. 508

<sup>509</sup> It is also important to emphasize that this increment in time <sup>510</sup> does not have a big impact on the overall simulation since it is <sup>511</sup> insignificant compared to the cost of AI, rendering or physics.

Including the recursive step when calculating clearance makes <sup>513</sup> our method more robust without introducing a noticeable im-<sup>514</sup> pact on the computational time.

The memory requirements to store the lookup table are min-516 imal, since for each radius size we only need two 3D point coor-517 dinates for the corresponding shrunk portal. For example, in the 518 cross scenario with 208 portals, 3 character sizes and 12Bytes 519 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 li-521 brary, local movement algorithm, size of the scenario, size of 522 the crowd, and so on, we are not interested in how many charac-523 ters we can simulate in real time, but in comparing our method 524 525 for paths with clearance against the standard solution where 526 characters walk towards way points fixed at the center of por-527 tals without checking for any kind of clearance against the static geometry. Figure 16 shows a comparison of the average frame 528 rate achieved as the number of characters increases with and 529 without our technique, when all the other elements of the simu-530 lation stay the same. This graph compares the standard solution 531 (in red) against our technique (in blue). The results are practically the same (less than 5% smaller frame rate on average with 533 our method), meaning that the computational time required to 534 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 <sup>538</sup> claim that the computational cost of our technique is insignifi-539 cant for the overall simulation time and that it provides results 540 that are perceptually more convincing and make better use of <sup>541</sup> the space, as shown in Figure 17 and the accompanying videos.



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

# 542 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 charcharacters only traverse those cells with a clearance larger than their radius. Another nice outcome of the presented method is the we can observe in the image how as the characters' size decoreases, the final emerging trajectories of their color are wider, since their way points are assigned over larger shrunk portals.

# 552 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 average number of collisions for both fixed center way points and dynamic way points. We account for a collision between wo 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 renter 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 brack 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

<sup>581</sup> lower number of collisions between agents which not only re-<sup>582</sup> duces artificial bottlenecks in the environment, but also results <sup>583</sup> in smoother and more natural trajectories. As in the donut sce-<sup>584</sup> nario, once the number of agents increases beyond 125, differ-<sup>585</sup> ences in the number of collisions start emerging between DWP <sup>586</sup> and fixed center WP, until the total number is higher than 225. <sup>587</sup> At this point, the high density of agents makes collisions in-<sup>588</sup> evitable, 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.

<sup>589</sup> While the graphs vary depending on the size of the scenario, <sup>590</sup> length of portals and local navigation method, we observe that

<sup>591</sup> in all of our experiments, dynamic way points achieve better <sup>592</sup> results than fixed center WP.

#### 593 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 potenand the overall geometry. To show the potenmathematic 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 guarante clearance. As shown in Cases 2 and 3, agents may even get completely stuck against the geometry, whereas with our state 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 preagainst the geometry. We have performed comparisons for different crowd sizes. We have demonstrated that the differences 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 636 clearance calculation when there are ill-conditioned cells. In 637 this example we can see the portal calculated with our exact method against the simple method often used in video games. 639 640 In both cases the segments over the portals that are traversable 641 for each method are shown with a thin blue line. The character 642 for which this clearance has been calculated is also circled in 643 blue. In both examples, a red agent is trying to move from cell <sup>644</sup> A to cell B. Our exact clearance algorithm provides the exact 645 segment over the portal that can be crossed without collisions 646 or errors. In the case of simple clearance, we can observe how 647 the character is being steered towards a position that will lead 648 to the wrong cell and to collisions against the geometry.

## 649 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.

<sup>653</sup> Our method can be divided into the following three steps. <sup>654</sup> Firstly, during the construction of the NavMesh, the clearance <sup>655</sup> value of each cell is computed in order to obtain paths that guar-<sup>656</sup> antee 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 minimize the overall quality of the simulation. Although its main prove the overall quality of the simulation. Although its main minimize 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 for 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 qualtity 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

<sup>673</sup> portals or causing bottlenecks.
<sup>674</sup> We have tested our algorithm with NavMeshes of a vari<sup>675</sup> ety of scenarios created by NEOGEN [12] which is a NavMesh
<sup>676</sup> generator that provides an almost near-optimal number of cells
<sup>677</sup> with very few ill-conditioned cells. To show the potential of our
<sup>678</sup> method even for other kinds of NavMeshes, we have also man-

679 ually generated navigation meshes with ill-conditioned cells. For the qualitative evaluation of this work we have consid-681 ered that higher quality paths are those that tend to use most of 682 the available space, avoid artificial line formation, reduce bot-683 tlenecks and collisions. In this paper we have also provided a 684 quantitative evaluation of the improvements achieved with our 685 exact clearance method by counting collisions against static and 686 dynamic geometry. Results show how our method provides not 687 only smoother paths with better usage of space, but also re-688 duces the average number of collisions that are caused by way 689 points not being correctly assigned. Compared to our previ-690 ous work [1], we have made significants improvements in terms 691 of generality as our new algorithm can handle a larger variety 692 of navigation meshes, while improving performance with the 693 introduction of the critical radius and a revised version of the 694 code.

#### 695 Acknowledgements

This work has been partially funded by the Spanish Ministry 697 of Science and Innovation under Grant TIN2010-20590-C01-698 01.

#### 699 References

- 700 [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.
- 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.
- 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 IJ CAI Workshop on Reasoning, Representation, and Learning in Computer
   Games. Edinburgh, Scotland; 2005,.
- 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.
- 738 [12] Oliva R, Pelechano N. Neogen: Near optimal generator of navigation meshes for 3d multi-layered environments. Computer
  740 And Graphics 2013;doi:\bibinfo{doi}{10.1016/j.cag.2013.03.004}.
  741 URL http://www.sciencedirect.com/science/article/pii/
  742 S0097849313000435.
- 743 [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}.
- 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.
- 749 [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.
- 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/RV0/.
- 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:
- 764 //doi.acm.org/10.1145/1342250.1342272.
- <sup>765</sup> [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.
   <sup>767</sup> 1109/TRO.2011.2120810}. URL http://dx.doi.org/10.1109/TRO.
   <sup>768</sup> 2011.2120810.

- 769 [19] Pettre J, Thalmann D. Path planning for crowds: From shared goals to individual behaviors. In: Eurographics Short Presentations. 2005,.
- 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}.
- 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.
- 779 [22] Mononen M. Recast navigation toolkit; 2009.
   http://code.google.com/p/recastnavigation/.
- 781 [23] Kallmann M. Shortest paths with arbitrary clearance from navigation
   782 meshes. In: Proceedings of the 2010 ACM SIGGRAPH/Eurographics
   783 Symposium on Computer Animation. SCA '10; Aire-la-Ville, Switzer 784 land, Switzerland: Eurographics Association; 2010, p. 159–68.
- 785 [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.
- <sup>793</sup> [26] Jaklin N, Cook IV AF, Geraerts R. Real-time path planning in het <sup>794</sup> erogeneous environments. Computer Animation and Virtual Worlds
   <sup>795</sup> 2013;5(24):285–95.
- 796 [27] Coumans E. Bullet physics library; 2013. http://bulletphysics.org/.