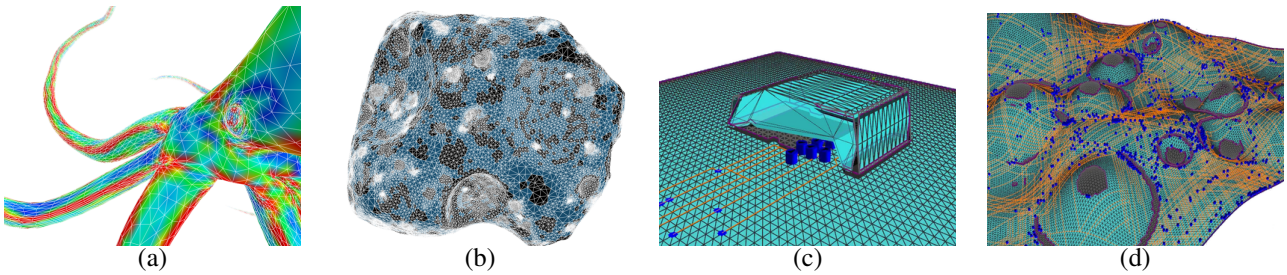
ACCLMesh: Curvature-Based Navigation Mesh Generation

Glen Berseth[1], Mubbasir Kapadia[2] and Petros Faloutsos[3]
[1]University of British Columbia [2]Rutgers University [3]York University



(a) (b) (c) (d)

Figure 1: *(a) Using curvature to characterize traversability on a 3D mesh. (b) The method to compute navigation meshes can handle complex 3D surfaces. (c) Evaluating height clearance with the navigation mesh allows agents to walk under a slanted overpass safely. (d) The approach can be integrated into standard navigation and animation systems to simulate thousands of agents on 3D surfaces in real-time.*

Abstract

We propose a method to robustly and efficiently compute a navigation mesh for arbitrary and dynamic 3D environments based on curvature. This method addresses a number of known limitations in state-of-the-art techniques to produce navigation meshes that are tightly coupled to the original geometry, incorporate geometric details that are crucial for movement decisions and robustly handle complex surfaces. We integrate the method into a standard navigation and collision-avoidance system to simulate thousands of agents on complex 3D surfaces in real-time.

CR Categories: Artificial Intelligence [I.2.11]: Distributed Artificial Intelligence—Multiagent Systems Computer Graphics [I.3.7]: Three-Dimensional Graphics and Realism—Animation;

Keywords: navigation mesh, crowd simulation, curvature

1 Introduction

Computer simulations involving autonomous virtual agents are becoming increasingly complex. Content construction often leads to large-scale, complex virtual worlds with a high level of geometric detail. A suitable representation of the traversable areas of these virtual environments is needed to facilitate efficient pathfinding queries while encoding environment geometry details that impact movement decisions.

Prior work has developed robust solutions for navigation mesh (*NavMesh*) generation for planar environments or restricted 3D environments that can be mapped to two dimensions. Voxel-based approaches [Memononen 2014] approximate the environment geometry and adaptively eliminate non-traversable geometry using ad-hoc heuristics (e.g., angular constraints relative to gravity). In an effort to produce meshes that are optimized for pathfinding operations, these approaches produce a *NavMesh* that is decoupled from the original mesh, lose important surface details that are important for movement decisions and don't generalize for unstructured, arbitrarily complex 3D surface meshes.

In this work, a method is proposed to compute a *NavMesh* that considers the terrain as a curved surface. Instead of using height maps or relative angle constraints on an approximation of the original geometry, discrete curvature is used to characterize the traversability of the geometric elements directly. The surface acceleration is calculated from the discrete curvature, this leads to the method being called *ACCLMesh*[1]. Given an agent's typical velocity, constraints are used to determine traversability on a surface, thus making the computed *NavMesh* a function of the agent's intrinsic movement capabilities. Since the computed *NavMesh* is generated directly from the original surface data, it offers a number of important advantages: (a) the output triangles are a subset of the original surface triangles and therefore do not intersect the original mesh; (b) there is no need to map the 3D environment to layered planar meshes, eliminating the need for *NavMesh* stitching; (c) the *NavMesh* incorporates important surface details that are essential for movement decisions; (d) since curvature calculations are local, the computational complexity of the approach scales linearly with the number of vertices in the mesh and facilitates dynamic *NavMesh* repair and (e) there exists a smooth function between geometrical changes in the mesh and the boundaries of the calculated traversable area.

ACCLMesh is compared to a current state-of-the-art solution *Recast* [Memononen 2014] on a number of challenging environment benchmarks including concave surfaces, large unstructured landscapes, asteroids, dynamic environments, and even the tentacles of an octopus. The *ACCLMesh* method is able to handle complex surfaces while encoding the mesh features that are essential for navigation. This framework is integrated with standard solutions for path planning and local collision-avoidance to simulate thousands of agents navigating on complex 3D terrain in real-time. The paper concludes with a discussion of other potential curvature calculation methods and the tradeoffs between our proposed method and current approaches which tradeoff accuracy and completeness for efficient path calculations.

[1]pronounced "accel-mesh"

2 Related Work

There is a growing body of research in discrete representations of environments that are amenable to efficient global navigation queries. These approaches generate a graph representing connectivity of free space in the environment which can be used by standard search algorithms [Pearl 1984] to find collision-free paths. We provide a broad overview below and refer the readers to [Kapadia and Badler 2013; Kallmann and Kapadia 2014] for additional details. Complementary to this is the problem of local movement of agents [Pelechano et al. 2008] that follow these global paths, while avoiding static as well as dynamic threats, which are not discussed here.

Grids are classical representations for path planning and are commonly used in virtual agent navigation [Shao and Terzopoulos 2005]. However, the computation time and solution quality greatly depends on the chosen resolution and can quickly become prohibitive for large environments. Roadmap approaches [Arikan et al. 2001; Sud et al. 2007] capture the connectivity of the free space but are unable to represent the geometry of the scene, and are not suitable for dynamically changing environments.

Navigation meshes [Kallmann and Kapadia 2014] partition the traversable space of the scene into convex regions, and offer an efficient decomposition of the environment. These include Explicit Corridor Maps [Geraerts 2010], local clearance triangulations on planar meshes [Kallmann 2010], portal graphs [Oliva and Pelechano 2011], waypoint graphs [Wardhana et al. 2013], and voxel-based approaches [Memononen 2014]. Extensions to these approaches facilitate efficient repairs in dynamic environments [van Toll et al. 2012; Kallmann 2014].

Navigation meshes for multi-layered and non-planar environments have also been developed in order to address 3D scenes [van Toll et al. 2011; Lamarche 2009; Jorgensen and Lamarche 2011; Oliva and Pelechano 2013; Memononen 2014]. These algorithms work well on surfaces that have a distortion-free map from 3D to 2.5D but are not suitable for arbitrary surfaces.

Researchers have also demonstrated the use of multiple environment representations [Kapadia et al. 2013b] and massive parallelization [Kapadia et al. 2013a; Garcia et al. 2014] to achieve computational speedup. Our work complements these approaches by proposing a novel navigation mesh representation for arbitrarily complex 3D environments.

Limitations. Previous work provides robust solutions that generate navigation meshes for specific environments types (e.g., 2D planar environments, or 3D environments which can be easily mapped to layered 2D environments). However, these approaches suffer from a number of limitations when dealing with arbitrarily complex 3D environments. Figures 2(a)-(c) illustrate navigation meshes that were generated using *Recast*. We observe that the *NavMesh* does not tightly fit the environment, and does not completely capture all traversable areas. In contrast, *ACCLMesh* is able to handle arbitrarily complex 3D surfaces, as seen in (d)-(f).

3 Traversability

In this section the metric that is used to determine the traversability on a surface is described. First, curvature is discussed and then how the acceleration on a surface is calculated given the curvature.

Curvature The measure of curvature used for this work is the *mean* curvature (κ^H) [Meyer et al. 2003]. This method formulates discrete mean curvature as the area gradient around each vertex. This formulation of curvature is appropriate because it works locally however, there are other methods to compute curvature. For example, it may be better to use a quadratic fitting method which can be more accurate when the geometry is course or noisy. Curvature is a reasonable metric for determining the difficulty of traversing an area.

Acceleration Curvature is normalized, and therefore has no notion of scale. Scale is incorporated principally by converting the curvature into an acceleration. Intuitively this conversion is accomplished by considering the instantaneous cyclic motion of the osculating circle, as follows:

$$a = \frac{v^2}{r} = \frac{v^2}{1/\kappa^H} = v^2\kappa, \tag{1}$$

where v is the speed along the curve. A non-unit speed, v, can be used to ensure that the resulting surface acceleration properly matches the agent's scale. For example, using this parameter the approach can calculate a proper *NavMesh* whether the agents represent humans, animals or insects.

Obstacles Areas of high acceleration can then be defined as obstacles. Areas of the surface with acceleration below an agent specific threshold a_{max} are considered traversable. The area of the surface that is considered non-traversable is an obstacle is defined as the region or regions of the surface with acceleration $> a_{max}$.

An additional benefit of acceleration over curvature, is that acceleration can be more intuitively considered as a measure of the effort it would take an agent to traverse an area or path. In this sense, *ACCLMesh* can potentially encode additional information about the environment. For example, the algorithm includes an optional step that can account for overhead clearance.

4 Computing the Navigation Mesh

This section details how acceleration, Equation 1, is used to compute a *NavMesh*. Specific subtleties and issues that arise in the process are also discussed.

Method Overview Given a mesh representing a terrain, an agent-specific threshold for acceptable acceleration, a_{max}, and an agent specific speed parameter, v, the following steps outlined below are used to compute a *NavMesh*. Figure 3 shows the main steps of the algorithm applied on an example mesh. The remainder of this section explains the steps of the algorithm in more detail.

Calculating Acceleration Boundaries One key feature of the proposed method is it can calculate the specific location of the acceleration boundary on a surface. This is achieved by essentially splitting edges of triangles as follows (Figure 3(b)).

Let there be some function f_{cut} that takes as input two points $\mathbf{x}_1$ and $\mathbf{x}_2$, with associated accelerations, and outputs a value in $(0, 1)$ that determines the location between the two points where the acceleration is equal to a_{max}. The cut function can be any kind of formula to best approximate the continuous curvature between two discrete points. For most of the examples in this work a linear blending approach $f_{acc}(\mathbf{x}_1, \mathbf{x}_2, a_{max})$ is used as f_{cut}:

$$f_{acc}(\mathbf{x}_1, \mathbf{x}_2, a_{max}) = \frac{a_{max}}{|(acc(\mathbf{x}_2) - acc(\mathbf{x}_1))|}. \tag{2}$$

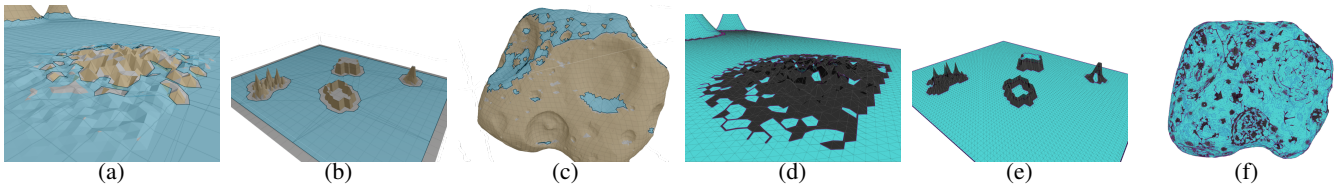


Figure 2: *Comparisons between* Recast *(a)-(c) and* ACCLMesh *(d)-(f). Compared to* Recast, ACCLMesh *does not intersect with the original geometry (d), can tightly fit any obstacle (e), and can efficiently handle complex surfaces with varied granularity (f).*

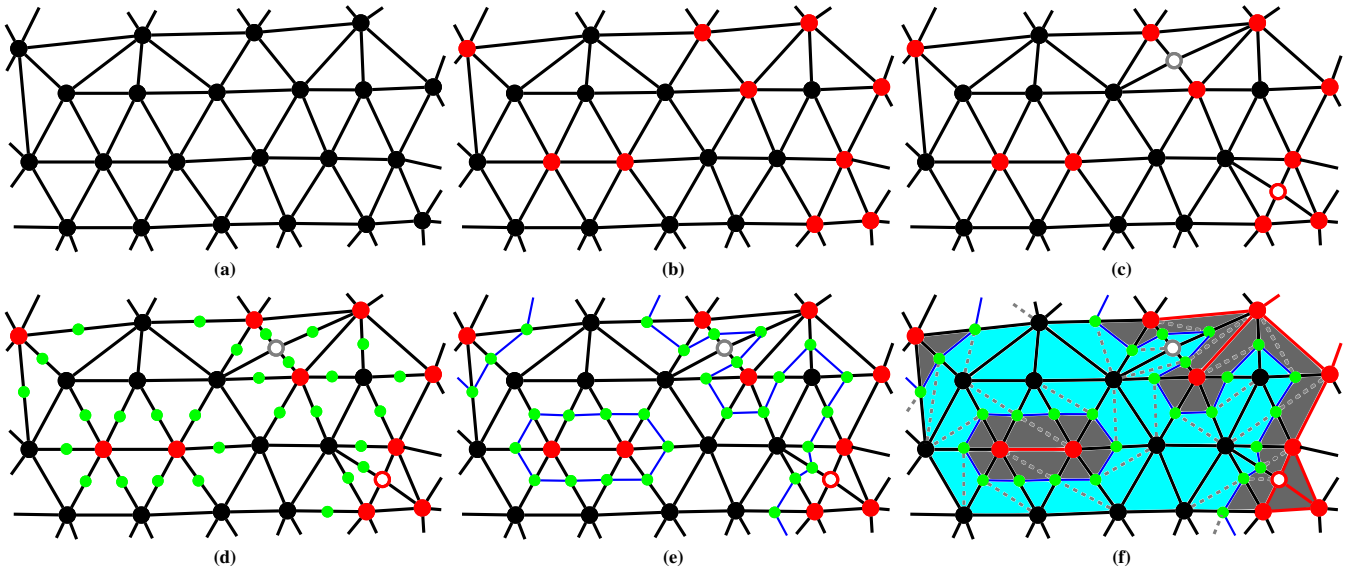


Figure 3: *Algorithm Overview. (a) Original mesh. (b) Red vertices have acceleration $a > a_{max}$. (c) Detection of biased edges, and biased edge splitting. (d) Computing cut points for edges with origin vertex $a < a_{max}$, and end point vertex $a > a_{max}$. Cut vertices highlighted in green. (e), (f) Re-triangulate and compute* NavMesh *in blue.*

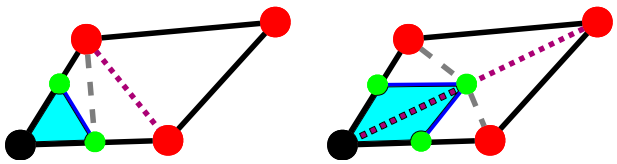


Figure 4: *An example of the biased edge issue. If the edge dividing the two triangles (dotted purple line) in the triangulation happens to connect two vertices with acceleration $> a_{max}$, then the result is the triangulation on the left. However, if one vertex of the dividing edge has acceleration $< a_{max}$, then the resulting triangulation with be the example on the right.*

If a very tight bound on the geometry is desired the cut function could return a number ϵ close to 1. This effect is demonstrated visually in Figure 7(e,f).

Biased Edges Simply removing edges that have vertices with acceleration $> a_{max}$ at both ends can lead to some undesired results. It assumes that the acceleration between those vertices is above a_{max}. However, if the four vertices of a pair of adjacent triangles is considered with 3 bad vertices between them, if the edge dividing the triangles happens to be flipped the resulting acceleration boundary would be different (see Figure 4).

In 2D a simple solution is to flip the edge. In 3D, edge flipping can result in different triangles with different angles at edges. Edge flipping would also deviate from the original geometry of the environment. Instead, in order to reduce the bias, the *problem edge* is split, making a new vertex $\mathbf{x}_{new}$ and the acceleration at $\mathbf{x}_{new}$ is set to the weighted sum of the neighbouring vertices, where each weight is the inverse relative distance of the neighbour. If d_{new} is the sum of distances between $\mathbf{x}_{new}$ and its neighbours $N(\mathbf{x}_{new})$, then the acceleration is calculated as:

$$a(\mathbf{x}_{new}) = \sum_{\mathbf{x} \in N(\mathbf{x}_{new})} \left(\frac{||\mathbf{x}_{new} - \mathbf{x}||}{d_{new}}\right) \cdot acc(\mathbf{x}). \qquad (3)$$

Two results of biased edge splitting can be seen in Figure 3(c) as new circles. The colour of $\mathbf{x}_{new}$ is red if the acceleration for the new vertex is above a_{max}, otherwise it is gray.

Height Clearance For 3D environments, the notion of clearance under objects is important. Although accounting for clearance can happen during other stages of an application, such as during simulation in character navigation, it is often convenient and more efficient to account for it with a *NavMesh*. The algorithm includes an optional height clearance stage that refines the *NavMesh* according to an agent specific height value h. In this case, areas of the environment are trimmed where a simulated agent, with height h, would intersect geometry, for example an overhead barrier or a low ceiling. We chose a distance calculation method as it offers a good balance between quality, performance and ease of implementation. We describe the method as follows:

1. For every vertex $\mathbf{x}$ in the mesh with $acc(\mathbf{x}) < a_{max}$
2. For every triangle t find the point p on t closest to $\mathbf{x}$
3. If this distance is $< h$
   (a) For each of the triangles around $\mathbf{x}$
      i. Check if p is indeed above the triangle by constructing a tetrahedra with p and the triangle
      ii. If the dihedral angles for that tetrahedra are all $< \pi/2$ then p is definitely over the triangle

This height clearance check is conservative and will remove extra portions of the surface triangles to be safe. If this check is positive an acceleration is assigned to $\mathbf{x}$ that is greater than a_{max} which results in this vertex being removed from the *NavMesh*.

Calculating The Navigation Mesh After all accelerations have been computed and cut vertices have been identified from the previous stages (Figure 3(d)), a re-triangulation needs to be performed. The additional cut vertices must be integrated into the mesh before triangles can be selected for the *NavMesh* based on the acceleration of each triangles vertices. This is done by adding an edge between the new cut vertices and another between one of the new cut vertices and the vertex this cut vertex is not connected to. The result of these steps can be seen in Figures 3(e) and 3(f). After all new triangles are triangulated the final step is to mark all triangles that have at least one vertex $\mathbf{x}$, where $acc(\mathbf{x}) < a_{max}$. These triangles form the resulting *NavMesh*.

Gravity is straightforward to include a step that accounts for gravity in this method. This can be done by removing any triangles from the *NavMesh* whose orientation deviates from the direction of gravity more than a user defined threshold.

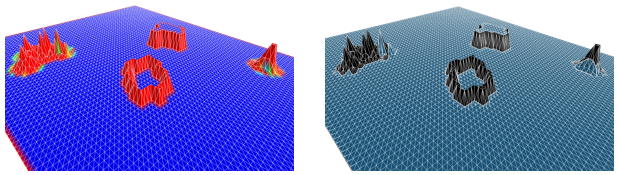
5 Features and Usage

In This section we discusses the features of the proposed method and how it is used in a practical setting. One example feature is, simulating crowds of agents on a *NavMesh* computed with the *ACCLMesh* method.

Pathfinding and Local-Collision Avoidance The *NavMesh* generated using our method can be integrated into standard pathfinding and collision-avoidance pipelines for multi-agent path planning and crowd simulation. An example is shown using A* [Hart et al. 1968] for pathfinding and a standard predictive collision avoidance algorithm (**ORCA**) [van den Berg et al. 2011]. **ORCA** is designed to work on planar environments and we discuss the main modifications we used to generalize it to 3D environments.

Path Planning. The output mesh can be transformed into a navigation graph $\Sigma = \langle \mathbf{V}, \mathbf{E} \rangle$, where $\mathbf{V}$ is the set of all vertices in the *NavMesh* with acceleration, $a < a_{max}$, and $\mathbf{E}$ is the set of viable transitions between adjacent vertices. Path planning is thus reduced to a discrete search that generates a sequence of edge traversals $\pi = \mathbf{Plan}(\Sigma, \mathbf{s}, \mathbf{g})$ from the start location $\mathbf{s}$ to the goal location $\mathbf{g}$. This can be accomplished using A* [Hart et al. 1968] or its variants. For optimal path computation, the geodesic distance should be used as a heuristic estimate. A method similar to SVG [Ying et al. 2013] that uses a global saddle vertex graph to approximiate optimial paths, could work well for realtime applications. A Euclidean distance measure worked well for the results shown in this work.

Obstacle Querying. Most collision-avoidance approaches need to query the presence of obstacles in an agent's vicinity when resolving collisions. Obstacles are closed polygons constructed from the list of vertices that make up the boundaries of high acceleration areas. These boundaries are highlighted in magenta in Figure 1(c) or the blue boundaries in Figure 3(e).



(a) (b)

Figure 5: *The acceleration (a) and computed* NavMesh *(b) result of using the* ACCLMesh *method on some example terrain.*

Movement on Mesh Surface. Agents are initialized on the mesh and associated with the triangle they are closest to. To check if an agent is in a triangle, a simple raycast down from the agent's head through the agent's feet is done. The agent-to-triangle association needs to be updated, with the agent's new position, at each frame. If the agent is still inside the same triangle, nothing needs to be done. If the agent is no longer in the same triangle, neighbouring triangles will be checked and the agent-to-triangle association will be updated with the new triangle the agent is closest to. The agent-to-triangle association is used to determine the movement of the agent along the *NavMesh* surface.

6 Results

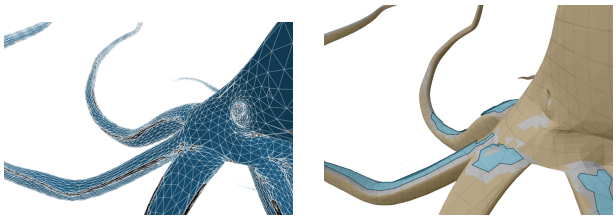
In this section, *ACCLMesh* is evaluated experimentally, and demonstrations of its main features are described.

Figures 5(a and b) and 2(b and e) compare the *NavMesh* produced by *ACCLMesh* against *Recast* on a mesh with terrain features often found in computer simulations. Figure 5(a) illustrates the acceleration calculations where blue is low acceleration and acceleration increases as the colours transition from blue to cyan to green to yellow and finally to red. A red colour corresponds to an acceleration value above a_{max}. Figure 5(b) shows the *NavMesh* that *ACCLMesh* method produces, and 2(b) shows the *NavMesh* from *Recast*. Our method produces a *NavMesh* that closely fits the original geometry. Mesh simplification methods could be used to reduce the number of triangles in the *NavMesh*, especially in the flat areas, at the cost of deviating from the original geometry. For example, vertex removal should work well as a mapping between the original vertices and the remaining vertices in the *NavMesh* can be maintained and local fixes could be done to fit the path to the original mesh.

Figures 1(a) and 6(a and b) show a similar comparison for a challenging terrain in the shape of an octopus. The long narrow geometry has low acceleration and could be considered easily traversable. Our curvature-based method comfortably eliminates the sharp edges around the arm's sides as seen in 6(a). In contrast, *Recast* fails to produce connected geometry and results in many thin safe *NavMesh* patches, some intersecting the original geometry, as seen in Figure 6(b).

Figure 2(c) shows that *Recast* generates a *NavMesh* for a small part at the top of the asteroid (in blue) but ignored the rest of the asteroid's surface (in brown). This is due to the naive assumption of gravity that *Recast* makes. As can be seen in Figure 1(b), *ACCLMesh* fits the geometry much better and does not suffer from the same naive assumption of gravity.

Using a curvature-based method to generate a *NavMesh* has a sub-



(a) (b)

Figure 6: *We compare the* ACCLMesh *result to* Recast. *ACCLMesh keeps more of the smooth geometry over the entire surface because it does not approximate the surface geometry. The acceleration on this mesh can be seen in Figure 1(a).*

tlety that is related to the resolution of the original data. In some cases *ACCLMesh* will not produce pleasing results when coarse geometries are present. An example of this type of geometry can be seen in Figure 7(a and c). The curvature-based approach does not handle this geometry well because there are technically no areas of low mean curvature κ^H along the ramp. The red region in Figure 7(a) shows that the entire ramp has acceleration that exceeds the threshold, resulting in a *NavMesh* that excludes the ramp, as seen in Figure 7(c). Figure 7(b and d) show the same geometry subdivided to produce new vertices in the middle of this ramp, which results in areas of low acceleration Figure 7(b). With the subdivided mesh *ACCLMesh* produces a proper *NavMesh* along the ramp Figure 7(d).

The tightness of the resulting *NavMesh* and its fit to the geometry can be controlled using the appropriate cut function, Section 4. For example, the cut function f_t can be used to output a number between 0 and 1 for any pair of points. In Figure 7(e) this number is always 0.9 and in Figure 7(f) it is 0.99. It is clear from the figure that this cut function can produce very tight boundaries around areas of high acceleration, and at the limit shrink them to zero. This effect could potentially be applied to *Recast* as well, by arbitrarily increasing the resolution of the voxelization. However, it would significantly increase *Recast*'s memory requirements and computation time and still produce only an approximation of the underlying geometry.

Computational Performance We compare the computational performance of *ACCLMesh* to Recast [Memononen 2014] over a set of terrains represented by triangle meshes. The results of this comparison can be seen in Table 1. The first mesh (ramp) is shown in Figure 7(d). The second environment is the same as the first but scaled to be $\sim$ 10 times larger. The scaling was done because *Recast* was giving very poor results on the small mesh. The third mesh represents a fairly large scale terrain which can be seen in Figure 1(d). The fourth mesh is the asteroid shown in Figure 2(f).

Table 1 shows that the proposed method can be orders of magnitude faster than *Recast*. However, we should note that *Recast* produces a mesh with very few triangles. It is fair to assume that potentially significant computation time is spent on keeping the number of triangles in the resulting mesh low. Nevertheless, the performance comparison shows that *ACCLMesh* is efficient and can therefore serve as the first stage of a pipeline that further processes the resulting *NavMesh* to satisfy the requirements of a particular application.

Crowd Simulation The resulting *NavMesh* can be used for multi-agent path planning and local-collision avoidance. Figure 1(d) illustrates a crowd of more than 1000 agents simulated on the complex 3D terrain using **ORCA** [van den Berg et al. 2011]. We also show an example where we use the clearance check technique to eliminate traversable areas that would lead to torso or head collisions with geometry that folds over on itself, such as the overpass in Figure 1(c).

Mesh	Ramp	S-Ramp	Terrain	Asteroid
# triangles	5128	5128	65536	213252
# vertices	2661	2661	33153	106629
B-box	$1.142e3m^3$	$1.130e6m^3$	$2.775e6m^3$	$4.741e6m^3$
ACCLMesh	16.1ms	16.1ms	204.8ms	989ms
Recast	34.8ms	392.2ms	913.9ms	4233.3ms

Table 1: *Performance comparison between* ACCLMesh *and* Recast *in milliseconds.*

7 Conclusion

This paper presents a curvature-based approach to analyze traversability on arbitrary surfaces and efficiently compute navigation meshes for complex 3D environments. In comparison to the current state-of-the-art, the approach produces navigation meshes that are tightly coupled to the original surface, thus avoiding mesh intersections and agent movement artifacts. The method incorporates the surface details necessary for navigation decisions, and can handle complex 3D surfaces where other approaches fall short. *ACCLMesh* is extended to include a height clearance checks. *ACCLMesh* can be easily integrated into standard navigation and collision-avoidance systems to simulate dense crowds on 3D surfaces at interactive rates.

There are several avenues of future exploration. Our work can be integrated with constraint-based approaches [Ninomiya et al. 2014] to compute paths that satisfy user-defined spatial constraints. Recent work in environment optimization [Berseth et al. 2015b; Berseth et al. 2014] could benefit from a more robust and computationally inexpensive *NavMesh* generation method. We are actively exploring how our approach can be integrated into other crowd simulation approaches [Singh et al. 2011a; Kapadia et al. 2012], and extended to handle more complex agent representations [Singh et al. 2011b; Berseth et al. 2015a].

References

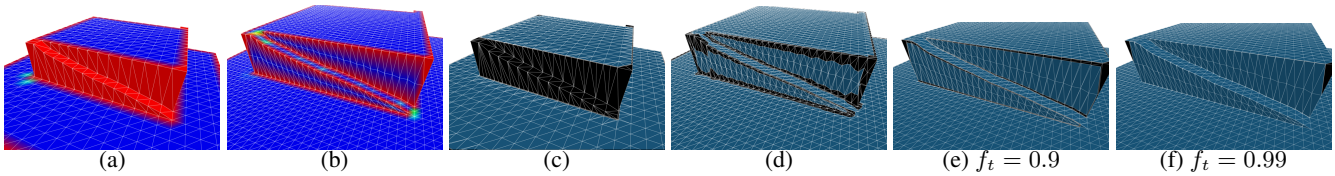
ARIKAN, O., CHENNEY, S., AND FORSYTH, D. A. 2001. Efficient multi-agent path planning. In *In Proceedings of the 2001 Eurographics Workshop on Animation and Simulation*, 151–162.

BERSETH, G., HAWORTH, M. B., KAPADIA, M., AND FALOUTSOS, P. 2014. Characterizing and optimizing game level difficulty. In *Proceedings of the Seventh International Conference on Motion in Games*, ACM, New York, NY, USA, MIG '14, 153–160.

BERSETH, G., KAPADIA, M., AND FALOUTSOS, P. 2015. Robust space-time footsteps for agent-based steering. *Computer Animation and Virtual Worlds*.

BERSETH, G., USMAN, M., HAWORTH, B., KAPADIA, M., AND FALOUTSOS, P. 2015. Environment optimization for crowd evacuation. *Computer Animation and Virtual Worlds 26*, 3-4, 377–386.

GARCIA, F., KAPADIA, M., AND BADLER, N. 2014. Gpu-based dynamic search on adaptive resolution grids. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, 1631–1638.



(a) (b) (c) (d) (e) $f_t = 0.9$ (f) $f_t = 0.99$

Figure 7: *A ramp on the side of a raised platform. The figures (a,c) show the acceleration calculations and* NavMesh *without subdivision. The Figures (b,d) show the corresponding calculations after subdivision. Tightening the areas of high acceleration with the cut function, f_t, for the ramp example, is shown in (e,f).*

GERAERTS, R. 2010. Planning short paths with clearance using explicit corridors. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, 1997–2004.

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

JORGENSEN, C.-J., AND LAMARCHE, F. 2011. From geometry to spatial reasoning: Automatic structuring of 3d virtual environments. In *International Conference on Motion in Games*, Springer-Verlag, 353–364.

KALLMANN, M., AND KAPADIA, M. 2014. Navigation meshes and real-time dynamic planning for virtual worlds. In *ACM SIGGRAPH 2014 Courses*, ACM, New York, NY, USA, SIGGRAPH '14, 3:1–3:81.

KALLMANN, M. 2010. Shortest paths with arbitrary clearance from navigation meshes. In *ACM SIGGRAPH/EG Symposium on Computer Animation*, 159–168.

KALLMANN, M. 2014. Dynamic and robust local clearance triangulations. *ACM Trans. Graph. 33*, 5 (Sept.), 161:1–161:17.

KAPADIA, M., AND BADLER, N. I. 2013. Navigation and steering for autonomous virtual humans. *Wiley Interdisciplinary Reviews: Cognitive Science 4*, 3, 263–272.

KAPADIA, M., SINGH, S., HEWLETT, W., REINMAN, G., AND FALOUTSOS, P. 2012. Parallelized egocentric fields for autonomous navigation. *The Visual Computer 28*, 12, 1209–1227.

KAPADIA, M., GARCIA, F., BOATRIGHT, C., AND BADLER, N. 2013. Dynamic search on the gpu. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, 3332–3337.

KAPADIA, M., BEACCO, A., GARCIA, F., REDDY, V., PELECHANO, N., AND BADLER, N. I. 2013. Multi-domain real-time planning in dynamic environments. In *Proceedings of the 12th ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, ACM, New York, NY, USA, SCA '13, 115–124.

LAMARCHE, F. 2009. Topoplan: a topological path planner for real time human navigation under floor and ceiling constraints. *Computer Graphics Forum 28*, 2, 649–658.

MEMONONEN, M. 2014. Recast: Navigation-mesh toolset for games.

MEYER, M., DESBRUN, M., SCHRDER, P., AND BARR, A. 2003. Discrete differential-geometry operators for triangulated 2-manifolds. In *Visualization and Mathematics III*, H.-C. Hege and K. Polthier, Eds., Mathematics and Visualization. Springer Berlin Heidelberg, 35–57.

NINOMIYA, K., KAPADIA, M., SHOULSON, A., GARCIA, F., AND BADLER, N. 2014. Planning approaches to constraint-aware navigation in dynamic environments. *Computer Animation and Virtual Worlds*, n/a–n/a.

OLIVA, R., AND PELECHANO, N. 2011. Automatic generation of suboptimal navmeshes. In *Motion in Games*. Springer, 328–339.

OLIVA, R., AND PELECHANO, N. 2013. Neogen: Near optimal generator of navigation meshes for 3d multi-layered environments. *Computers & Graphics 37*, 5, 403 – 412.

PEARL, J. 1984. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

PELECHANO, N., ALLBECK, J. M., AND BADLER, N. I. 2008. Virtual crowds: Methods, simulation, and control. *Synthesis Lectures on Computer Graphics and Animation 3*, 1, 1–176.

SHAO, W., AND TERZOPOULOS, D. 2005. Autonomous pedestrians. In *Proceedings of the ACM SIGGRAPH/EG Symposium on Computer Animation*, 19–28.

SINGH, S., KAPADIA, M., HEWLETT, B., REINMAN, G., AND FALOUTSOS, P. 2011. A modular framework for adaptive agent-based steering. In *Symposium on Interactive 3D Graphics and Games*, ACM, New York, NY, USA, I3D '11, 141–150 PAGE@9.

SINGH, S., KAPADIA, M., REINMAN, G., AND FALOUTSOS, P. 2011. Footstep navigation for dynamic crowds. *Computer Animation and Virtual Worlds 22*, 2-3, 151–158.

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

VAN DEN BERG, J., GUY, S. J., LIN, M., AND MANOCHA, D. 2011. Reciprocal n-body collision avoidance. In *Robotics Research*, vol. 70. 3–19.

VAN TOLL, W., COOK, A., AND GERAERTS, R. 2011. Navigation meshes for realistic multi-layered environments. In *IEEE/RSJ Intelligent Robots and Systems*, 3526–3532.

VAN TOLL, W. G., COOK, A. F., AND GERAERTS, R. 2012. A navigation mesh for dynamic environments. *Comput. Animat. Virtual Worlds 23*, 6 (Nov.), 535–546.

WARDHANA, N., JOHAN, H., AND SEAH, H. 2013. Enhanced waypoint graph for surface and volumetric path planning in virtual worlds. *The Visual Computer 29*, 10, 1051–1062.

YING, X., WANG, X., AND HE, Y. 2013. Saddle vertex graph (svg): A novel solution to the discrete geodesic problem. *ACM Trans. Graph. 32*, 6 (Nov.), 170:1–170:12.