A Whole Surface Approach to Crowd Simulation on Arbitrary Topologies

Brian C. Ricks & Parris K. Egbert *Member, IEEE*

Abstract—Recent crowd simulation algorithms do path planning on complex surfaces by breaking 3D surfaces into a series of 2.5D planes. This allows for path planning on surfaces that can be mapped from 3D to 2D with distortion, such as multistory buildings. However, the 2.5D approach does not handle path planning on curved surfaces such as spheres, asteroids, or insect colonies. Additionally, the 2.5D approach does not handle the complexity of dynamic obstacle avoidance when agents can walk on walls or ceilings. We propose novel path planning and obstacle avoidance algorithms that work on surfaces as a whole instead of breaking them into a 2.5D series of planes. Our "whole surface" approach simulates crowds on both multistory structures and highly curved topologies without changing parameters. We validate our work on a suite of 30 different meshes, some with over 100,000 triangles, with crowds of 1,000 agents. Our algorithm always averaged more than 40 FPS with virtually no stalling.

Index Terms—Crowd Simulation, 3D Crowd Simulation, Path Planning, Obstacle Avoidance

1 Introduction

Animated films, special effect scenes, and games rely heavily on crowd simulation algorithms. Previous research has produced believable crowds in full 3D environments (such as flocks of birds) and 2D environments (such as crowds on flat surfaces). However, in between full 3D and 2D motion lies the important and emerging field of crowd simulation on arbitrary manifolds. Unlike full 3D environments, crowds on arbitrary surfaces are constrained to manifolds. Unlike traditional 2D crowds, the geometry of arbitrary surfaces adds new and largely unsolved path planning and dynamic obstacle avoidance problems.

Crowd simulation on arbitrary manifolds is applicable on any surface that cannot be approximated by a plane plus a height map. These surfaces include multistory structures such as skyscrapers, subways, and cruise ships; natural structures such as caves and arches; extraterrestrial surfaces such as asteroids, space stations, and spaceships; and surfaces where insects swarm, such as nests, colonies, floors, walls, or ceilings. Movies with these types of crowds include films with bizarre topologies such as *Inception*, films centering around multistory structures such as *Titanic*, and science fiction films set on asteroids such as *Armageddon* or in spacecraft such as *2001: A Space Odyssey*. Even the video game industry has seen the rise of characters on complex topologies as in the *Mario Galaxy* series.

Simulating crowds on arbitrary surfaces introduces new challenges to both path planning and obstacle avoidance. Recent academic and commercial middleware algorithms handle path planning by breaking surfaces into a series of planes. This is effective for surfaces that map to a set of planes without distortion. However, crowd simulation algorithms are needed for surfaces that do not map to a series of planes without distortion—including many of the films just mentioned. Also, the literature does not address the complexity of dynamic obstacle avoidance on arbitrary topologies.

Our goal is to resolve these issues and to increase the set of surfaces on which we can simulate crowds. To do this, we propose a "whole surface" approach to crowd simulation. Our contributions include both novel path planning and obstacle avoidance algorithms. Our proposed path planning approach improves discrete geodesic algorithms to produce quality motion at real-time rates (Sec. 3). Our approximate vision algorithm allows robust obstacle avoidance on arbitrary surfaces without collision false positives and false negatives (Sec. 4). As validation, we show that our algorithm correctly and quickly produces natural movement (Sec. 5). Combined, our whole surface approach accurately simulates crowds on a far larger set of surfaces than possible before.

2 Previous Work

We explain our work in the context of path planning, obstacle avoidance, and arbitrary surface crowd simulation research.

2.1 Global Path Planning

2D path planning has its roots in graph-based path planners, which run run quickly but often produce jagged results. Research relevant in improving jagged paths includes line of sight smoothing [1], [2] or more accurate algorithms such as fast marching methods [3].

• *B. Ricks and P. Egbert are with the Department of Computer Science, Brigham Young University, Provo, UT, 84606.*
E-mail: bricks@byu.edu, egbert@cs.byu.edu

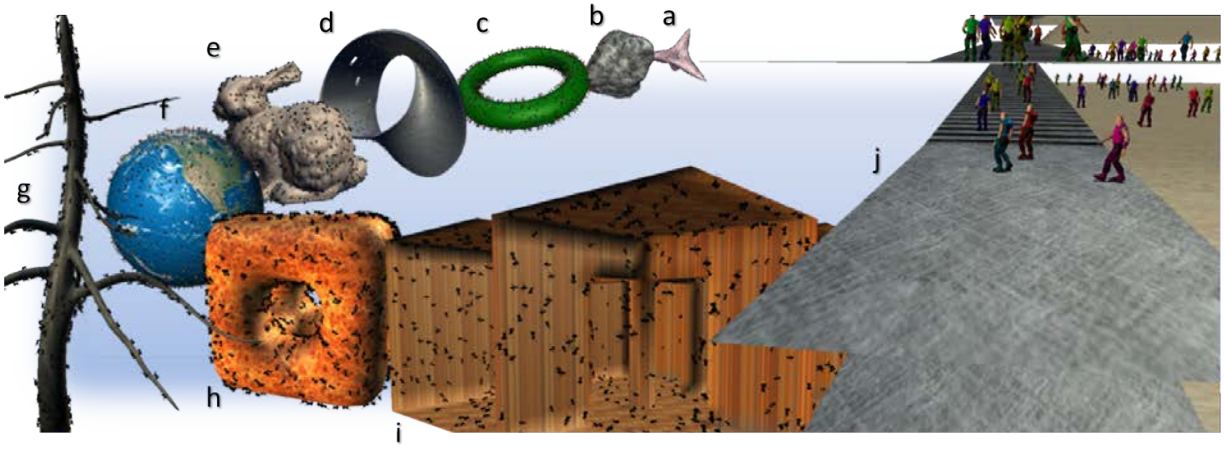

Fig. 1. Our test surfaces: (a) a tetrahedron, (b) the asteroid Golevka, (c) a torus, (d) a mobius strip with holes, (e) the Stanford Bunny, (f) a globe, (g) ants on a tree, (h) ants on a piece of food, (i) ants swarming inside a house, and (j) people in a multistory building. Bunny courtesy Stanford Computer Graphics Laboratory. Asteroids courtesy Scott Hudson, Washington State University, http://users.tricity.wsu.edu/~hudson/Research/Asteroids/models.html.

Advanced 2D path planning algorithms take into account clearance around static obstacles and have a sense of path width to avoid jams. These are often called navigation mesh approaches. Such works include Geraert's corridor maps [4], Kallmann's navigation queries on planar triangular meshes [5], Kamphuis et al.'s roadmaps [6], Pettré et al.'s navigation graphs [7], Lamarche and Donikian's constrained Delaunay triangulation [1], Hale's use of watershed partitioning [8], and Curtis et al.'s way portals [9]. These algorithms also work well on surfaces that have a distortion-free map from 3D to 2.5D. We explore why navigation meshes are insufficient for arbitrary surfaces in Sec. 3.2.

Other researchers have looked at path planning on 3D surfaces, but not in the context of crowd simulation applications. One of the earliest exact solutions came from Mitchell et al. [10] who could guarantee finding the shortest path on a surface (i.e. discrete geodesics) with a complexity of $O(n^2 log\ n)$. This time bound was improved by Chen and Han [11] to $O(n^2)$. Faster approaches include Martinez et al. [12] who iteratively refine an approximate shortest path. A quite powerful discrete geodesic method was proposed by Kimmel and Sethian [13] based on the fast marching method. Our framework is designed to improve results from any discrete geodesic algorithm for crowd simulation.

2.2 Local Obstacle Avoidance

Most 2D obstacle avoidance techniques trace their roots to Reynolds' work on flocking dynamics [14] that used instantaneous forces to avoid collisions. Helbing and Molnár [15] proposed a similar 2D instantaneous forces method called social forces. To address the jamming and stalling inherent with these methods, Fiorini and Shiller [16] introduced velocity obstacles. This method has numerous extensions, including van den Berg et al.'s reciprocal velocity obstacles (RVO) [17]. More recently, Guy et al. [18] changed RVO to optimize over agent effort.

Other work focused on the efficiency and believability of 2D crowd simulation. Courty and Musse [19] proposed GPU social forces. Narain et al. [20] simulated tens of thousands of agents using Euclidean and Lagrangian methods. Karamouzas et al. [21] achieved similar speed results with their collision prediction model. Singh et al. [22] created a crowd simulation algorithm using A* planning to find footstep patterns for agents. Representing each agent as five circles, their algorithm created realistically dense crowds. Pelechano et al.'s HiDAC algorithm [23] simulated large panic situations. Other work simulates crowds as a flow or continuum. Both Treulle et al. [24] and Jiang et al. [25] created large crowds this way. Others have worked on objectively evaluating crowd simulation [26], [27], [28]. Notable among these is the Singh et al.'s SteerBench project [29], [30]. This work is part of StreerSuite [31], an open-source platform for creating and evaluating steering algorithms.

This obstacle avoidance work has culminated in crowds that deftly avoid collisions in 2D and on surfaces that break down into a series of 2.5D planes (for surveys see [32] and [33]).

2.3 Crowds on Arbitrary Topologies

Some previous work has addressed the issue of crowds on non-planar surfaces, often by breaking surfaces into a series of 2.5D planes. Shao and Terzopoulos [34] created a model of New York City's original Pennsylvania station this way. Lamarche [35] proposed a method for character motion with features such as ducking under ceilings, but the algorithm approximated the 3D surface

with a 2D plane. Similarly, Jiang et al. [25], [36] broke down complex environments into blocks, each of which can be mapped to a regular grid. Deusdada et al. [37] did the same using an image-based approach. Authors van Toll et al. [38] expanded navigation meshes for multistory structures, but not arbitrary surfaces.

Rodriguez et al.'s work broke buildings into 2D subsections for pursuit-evasion applications [39]. Sturtevant and Geisberger [40] presented a grid method for path planning using multiple levels of abstraction. Stoyanov et al. [41] proposed a path planner for robotic applications on arbitrary 3D point cloud data. Unfortunately, this approach is too slow to generate large crowds at real-time speeds. Jund et al. [42] used a multiresolution grid to find dynamic obstacles on surfaces, but their approach only judged distances of the surface, which would result in collision false positives. Also, they did not deal with corridors around sharp edges and turns. Levine et al.'s work [43] provided an algorithm for character locomotion through complex environments with dynamic obstacles. The resulting agent movement was impressive, but their algorithm broke the agent-based paradigm of crowd simulation algorithms (as did the work of Singh et al. [22] discussed above). Similar to the approach we propose, Torchelsen et al. [44] used discrete geodesics to do crowd simulation on continuous 3D surfaces. While the results are pioneering, the meshes used do not have any edges or sharp curves. Additionally, it only allowed a limited number of destinations for agents, the algorithm used a very simple GPU-based obstacle avoidance method, and the algorithm could struggle from clear false positive artifacts.

In summary, current research effectively simulates crowds in 2D or on a series of planes. However, we have found no previous algorithms that can robustly handle crowds on both highly curved surfaces (such as asteroids and tree branches) and multistory buildings (such as apartment buildings or offices). The goal of our work is to expand the set of surfaces on which crowds can be simulated to include surfaces of any topology.

3 Path Planning

The first part of our surface as a whole approach is our path planning algorithm. The goal of a path planning algorithm is to find a path from an agent's initial position to the agent's destination. To do path planning, we define a function $Plan$ that takes the current agent's initial position, a_p, destination, a_d, and the surface mesh, $mesh$, and produces an ordered list of waypoints, W as follows:

$$Plan : a_p, a_d, mesh \rightarrow W \tag{1}$$

The goal of the path planning piece of a crowd simulation algorithm is to find an implementation of $Plan$ that produces natural motion for agents. The geometry of arbitrary surfaces is far more complex on arbitrary surfaces than it is in 2D. Thus, our main path planning contributions include an implementation of Eqn. 1 that creates corridors and does line of sight smoothing even on arbitrary surfaces.

Note that like previous crowd simulation work, we take two mesh inputs into our algorithm. The "surface mesh" represents where agents can move. The "render mesh" is used to render the scene. For example, consider simulating people in an office building. In this case the surface mesh would only contain triangles that belong to the floor where the agents can move. The render mesh would contain all the triangles to be rendered, including the floor, walls, and ceiling. On the other hand, if we were simulating ants in that same office building we would give the full office mesh as both the surface and render meshes. In this work we do not address the problem of automatically finding the walkable areas of a given mesh since there are already very effective algorithms for this task (e.g. [35]). All the figures in this paper show the surface mesh the crowd is using.

3.1 Stalling

Real crowds exhibit little stalling, i.e., people rarely jam for long periods of time. Instead, in real crowds people move naturally around each other en route to their destinations. Our experience has been that stalling is an effective quantitative measure of the realism of a crowd. Additionally, we have found that when we use a modern obstacle avoidance algorithm, crowd stalling is almost always a symptom of poor path planning.

To motivate our discussion of path planning for arbitrary surfaces, we begin with our five properties of effective path planning. In our experience, a path planner that has these properties will have almost no stalling and the crowd movement will be natural. These have been discussed less formally in previous work, but we explicitly list them here as a guide for our work. First, paths that produce natural motion will lead an agent from a_p to a_d. Second, each waypoint on the path should be on our surface. Let $\mathcal{P}(A)$ be the powerset of A. If S is the set of all points on our triangular mesh with $S \in \mathcal{P}(\mathbb{R}^3)$, then it follows that we want $W \in \mathcal{P}(S)$. Third, paths should not cross static obstacles. Fourth, we want path planners to work on a large set of surfaces without manually tuning parameters. If an implementation of $Plan$ in Eqn. 1 has these four attributes then the path should lead a single agent to its destination without stalling. In multiagent scenarios, particularly when there are corners or tight halls, agents will deviate from W to avoid other agents. Thus, we add a fifth property for stalling-free path planners: $Plan$ should produce a path with width.

Most 2D and 2.5D algorithms fail to meet the fourth or fifth properties when used on arbitrary surfaces. 2.5D crowd planning algorithms that work on multistory structures such as [25], [36] would fail on curved surfaces such as ants on a tree or astronauts on an asteroid (our fourth criterion). Algorithms that do focus on highly curved surfaces such as [44] would fail on the complexities of multistory structures, especially around corners

and up stairs (our fifth criteria). We propose a surface as a whole path planning algorithm that tries to keep all these five properties. Our resulting algorithm moves agents on arbitrary topologies with little or no stalling.

3.2 Discrete Geodesics

Our surfaces as a whole path planning approach uses discrete geodesics and adds key improvements designed for crowd simulation. However, one of our early thoughts was to use navigation meshes. By breaking surfaces into higher-level structures, navigation mesh algorithms seemed to have two advantages: they might run faster and they might lead to easy corridor creation. As noted in our previous work section, this approach has been used successfully in the specific domain of multistory buildings. However, the goal of our work is to expand the domain of crowd simulation surfaces to any topology and we discovered that in this domain navigation meshes are not always the right approach.

Consider the trivial case of an ant on a cube. Assume the ant is on the top face with a destination on the bottom. How can we apply navigation meshes to this surface? We could unfold the cube and then use navigation meshes. Unfortunately unfolding in ambiguous: One unfolding would lead the ant directly to the destination while another would lead the ant across every face before reaching the destination. Thus each time a path is planned the surface would need to be uniquely unfolded, thus defeating the goal of having a navigation mesh that can be used for all path planning. Additionally, when a surface is unfolded, it can appear that pieces of the surface are not connected when they really are. Another approach would be to leave the cube together and group triangles into regions that are not separated by obstacles. Unfortunately, since there are no static obstacles on a cube, the entire cube could be the same region. Thus, this approach would not provide any path planning information.

Thus, we use discrete geodesics for two reasons. First, discrete geodesics are complete since they always find a path to the destination if one exists (more in our results section). Navigation meshes, on the other hand, will only work if we strictly limit the domain of surface meshes. Second, discrete geodesics algorithms are fast enough for real-time crowd simulation (more in our results section). Additionally, by using our path improvements, we can improve discrete geodesics so they create very natural crowd movement.

3.3 Path Planning Improvement Overview

We use discrete geodesic algorithms for the reasons listed previously: they find paths on any surface and they run quickly. However, the linear paths found by discrete geodesic algorithms need to be improved to produce natural motion for large crowds. Alg. 1 and Fig. 2 summarize our three improvements. Fig. 2a shows an example of an agent traveling on a surface that is both highly curved and has a corner (e.g. the surface in Fig. 3). Fig. 2b shows a possible path generated by a discrete geodesic algorithm. This path is problematic since it lies directly on static obstacles (the edge of the hole) and has a jagged, unnatural turn. An agent following this path could stall because the path lies on an obstacle, look unnatural by following the jagged path, or jam if agents are the other way since the path lacks width. However, since we have used discrete geodesics, we are guaranteed to have found a path in one exists to our destination (see our results section). Our main contribution lies in creating corridors and doing path smoothing to solve these problems on arbitrary surfaces.

Fig. 2c shows our first improvement: away from edges. This improvement pushes waypoints away from static obstacles (see Sec. 3.4). Fig. 2d shows our second improvement: corridor expansion. This improvement gives paths width around static obstacles where jams are most likely to occur (see Sec. 3.5). These first two improvements run only when an agent requires a new path (i.e. the start of the simulation or when the agent reaches a destination). Fig. 2e shows our last improvement: line of sight. This improvement approximates line of sight on the surfaces to smooth out jagged paths (see Sec. 3.6). Quantitative validation of these improvements are given in our results section, Sec. 5.

3.4 Away From Edges

The paths created by discrete geodesic algorithms do not by themselves produce prefect paths for crowd simulation. One surface feature where this is very evident is around walls and corners. These features are frequently found in building surfaces (see Fig. 1j), science fiction surfaces (see Fig. 1d), or surfaces where insects swarm (see Fig. 3). In these cases, shortest paths tend to lie right along the edges of a surface. For example, consider the case of a building with an agent whose destination is on the next floor up. In this case the shortest path to the destination would lead from the agent to the corner of the stairs and go up the inside of the stairs to the next floor. Thus, an agent following this path would both want to go toward the very inside of the stairs (to follow the path) and at the same time want to avoid

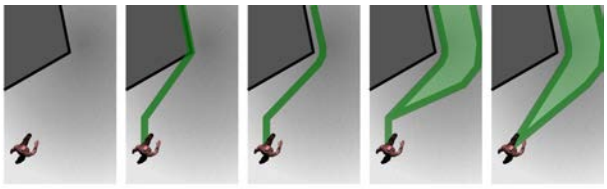

Fig. 2. Our path planning improvements. a) An agent going around a corner on a curved surface (e.g. Fig. 3). b) Initial path. c) Away from the edge (Sec. 3.4). d) Corridors (Sec. 3.5). e) Line of sight (Sec. 3.6).

Algorithm 1 Overview of our path finding approach

```
1: procedure ImprovePath(a_p, a_d, mesh)
2:   W ← DiscreteGeodesic(a_p, a_d, mesh)
3:   for all w ∈ W do
4:     if !EdgeVertice(w) then
5:       AddToList(W_improved, w)
6:     else                                   ▷ Improve the waypoint
7:       T ← TrianglesWindingFrom(w, w_{i+1})
8:       E ← EdgesAdjacent(w)
9:       for all e ∈ E do
10:        w_t ← w
11:        AddToList(W_temp, w_t)
12:      for all w_t ∈ W_temp do      ▷ Away From Edges, Sec. 3.4
13:        w_t ← MoveAway(w, w_t)
14:        if !OnEdge(w_t) then                ▷ Pruning
15:          if Contains(T, w_t) then          ▷ Winding
16:            AddToList(W_away, w_t)
17:      for all w_a ∈ W_Away do       ▷ Corridors, Sec. 3.5
18:        AddToList(W_improved, AddWidth(w_a))
19:  return W_improved
20: procedure NextWayPoint(W, a)          ▷ Smoothing, Sec. 3.6
21:   for all w ∈ W do
22:     if EdgeVertice(w) then
23:       return w                          ▷ Edge Waypoints
24:     if DivergentNormals(w, a) then
25:       return w                          ▷ Divergent Normals
```

the very inside stairs (to avoid the static obstacle). This type of poor path planning results in stalling since as the obstacle avoidance algorithm cannot move the agent toward the next waypoint without a collision.

To remove this unnatural behavior, our algorithm improves paths by pushing them away from static obstacles such . To push paths away from edges, our algorithm removes edge entries in W and replaces them with new way points(see Eqn. 1 and Alg. 1 Lines 7–16). We call this step the "away from edges" step of our algorithm. Conceptually, this is similar to using a Minkowski sum to shrink the surface around edges. The difference is that with our approach an agent in a crowded scenarios can use the area right next to obstacles if needed. Formally, our improvement implements a function $Away$ that takes a waypoint on an edge and replaces it with an improved list of waypoints.

$$Away : w_i \in W \rightarrow W_{improved} = \langle w_1, w_2, ..., w_n \rangle \qquad (2)$$

Finding the points for $W_{improved}$ on a surface of arbitrary topology is a three part process. (Fig. 4 shows these steps with Fig. 4a showing an example path that needs to be improved.) The away from edges step begins by identifying waypoints that lie of static obstacles. For each waypoint $w_i \in W$ that lies on a static obstacle, our approach finds all the surface edges the share an end point at w_i. For each edge our algorithm creates a set of proposed new waypoints at the same location as w_i, each of which is attached to one of these edges. Then we move each of the proposed waypoints along their corresponding edge away from the static obstacle. We move each waypoint approximately an agent's stride's width. As noted earlier, this paper does not address the research area of finding the walkable regions of a mesh, so we assume that all meshes have walkways wide enough for this path movement. An example of this step is shown in Fig. 4b.

Fig. 3. Ant agents moving on a soccer ball without the pentagons. Our algorithm handles this surface as easily as the crowds in Fig. 5 without changing parameters.

After this step proposed waypoints can lie on edges (again see Fig. 4b). In the second step of the away from edges improvement we scan over the proposed waypoints and prune any that are on static obstacles (see Fig. 4c). Now none of the remaining proposed waypoints lie on static edges, but notice that we have a sharp turn in the path. In the third step of the away from edges improvement our algorithm finds the triangles that share a vertex at w_i. Our algorithm then winds through these triangles starting with the triangle containing w_{i+1} and ending at the triangle containing w_{i-1}. Any waypoints contained in these triangles are kept while those not in the triangles are discarded. This last step smooths the away from edges path naturally as shown in Fig. 4d. These remaining waypoints are placed in the set $W_{improved}$ in Eqn. 2.

As we discuss in our results section, adding this away from edges improvement decreased stalling in our simulations by over an order of magnitude (see Sec. 5). However, there is still some stalling and we need addition improvements to prevent stalling around corners and edges in crowded scenes.

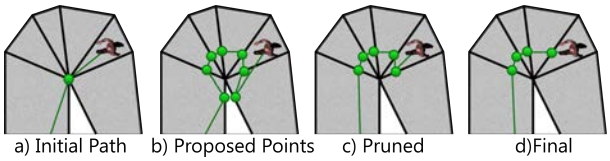

Fig. 4. Replacing waypoints in W with a list of new points $W_{improved}$. a) Find vertices on edges and b) add waypoints that are pushed away. c) Prune waypoints on edges and d) wind around to smooth the path.

Fig. 5. Lane formation with our algorithm. Our optimizations create believable crowds on this surface and the surface is Fig. 3 without changing parameters.

3.5 Corridors

The away from edges improvement works well in single-agent scenarios, but in multiagent scenarios paths need width to avoid stalling. The crowd simulation literature has proposed many methods of corridor creation in 2D. Unfortunately, the nature of arbitrary surfaces means many of these approaches do not work in 3D (see our navigation meshes discussion above). To give paths width on arbitrary surfaces, we propose a method of improved the away from edges step to create corridors. Combined, our away from edges and corridors algorithms form one of our key contributions to path planning on arbitrary surfaces.

To create a path with width, our corridors improvement starts with the proposed waypoints generated by Eqn. 2. Recall that these waypoints were found by identifying edges that ended at the static obstacle. In the corridors step we take the waypoints generated in the previous step and turn them into line segments. In the away from edges improvement, each proposed waypoint was attached to a mesh edge. In the corridors step, we create a line segment in the direction of these mesh lines with one end closer to the static obstacle and one end further away (see Alg. 1 Lines 17–18). In constructing these corridors we make them approximately four agent's width in length. This allows up to four lanes to develop within the corridor. As noted earlier, we assume that all meshes are wide enough (a more precise size could be used if the mesh were preprocessed by a navigable mesh algorithm). With this corridor improvement, we change $Away$ (Eqn. 2) to $Corridor : w_i \in W \rightarrow W_{improved} = \langle l_1, l_2, ..., l_n \rangle$.

The addition of these corridors creates realistic global crowd dynamics such as lane formation (see Fig. 5). This behavior is evident both in multistory structures and curved surface with edges or corners (e.g. Fig. 3 and Fig. 1d). We quantitatively measure the effectiveness of our corridors approach in Sec. 5.

3.6 Line of Sight

Thus far we have discussed discrete geodesic algorithms as a group. In reality, there are many discrete geodesic algorithms, each with their own relative advantages and disadvantages. Generally speaking, discrete geodesic algorithms can be characterized as either slower, perfectly accurate algorithms or faster, less accurate algorithms. Unfortunately, slower but more accurate algorithms run too slowly for real-time crowd simulation (see Fig. 12). Examples of fast enough algorithms include graph-based planners that find the shortest path across mesh vertices or mesh triangles (e.g. Dijkstras or A*). These slower algorithms cannot be used without improvements since the paths they produce are inherently jagged and lead to unnatural motion. This problem can be remedied by using a line of sight algorithm. For the line of sight improvement we approximate the line of sight for each agent on our surface. We then let the agent walk to the furthest waypoint it can see. Thus, agents ignore jagged turns in their path and head directly to the last point visible to them. Compared to previous 2D work that does line of sight smoothing (see Sec. 2), our contribution is that we quickly smooth paths on arbitrary surfaces. This is non-trivial since our path is not in 2D.

For this improvement, we assert that an agent's line of sight stops under two conditions (see Alg. 1 Lines 20–25). First, an agent cannot see waypoints around corners. This is simple to detect in our algorithm since we already search for static obstacles in our previous improvements. We simply flag waypoints that have been moved away from edges and turned into corridors for this purpose. Second, we stop an agent's line of sight when the normal of a waypoint diverges too much from the agent's current normal. This condition prevents agents from seeing waypoints that are blocked from sight by the curvature of the surface. For example, consider an ant that is going around a tree branch. In this case our normal test would stop the ant's line of sight where the branch curves away from the ant's vantage point. The divergence in the normals can be quickly calculated using a dot product. We assert that an agent cannot see a waypoint when the waypoint's normal and the agent's normal have a dot product less than .5. In our results section we show that this improvement puts the performance of graph-based planners on par with slower, more accurate planners.

4 Obstacle Avoidance

As previously mentioned, crowd simulation algorithms are composed mainly of two functions, a path planning function and an obstacle avoidance function. The last section detailed our path planning algorithm that improves discrete geodesic algorithms to remove stalling on arbitrary surfaces. In this section we detail our obstacle avoidance algorithm, which improves significantly upon [45].

4.1 2D Obstacle Avoidance Abstraction

At the heart of 2D crowd simulation algorithms is a function that takes each agent and finds the best heading and linear velocity for collision-free movement. If we call

this function $2DMove$, then this function can be written as:

$$2DMove : O \in (\mathbb{R}^2), a \rightarrow \theta', v' \quad (3)$$

Where O is a set of tuples in $\mathbb{R}^2$ that represent the heading and distance to static and dynamic obstacles and a is the location and heading of an agent. $2DMove$ uses the heading and distance to obstacles to choose a change in motion for collision-free movement. This change in motion is represented by the outputs of $2DMove$, with θ' and v' representing the agent's change in angle and linear velocity respectively.

The set of 2D agent-based crowd algorithms that implement the function $ObstacleAvoidance$ (or a very similar function) includes social forces [15], RVO [17], HiDAC [23], and anticipation [46]. Almost all 2D crowd simulation algorithms start their implementations of $2DMove$ by doing two things: they ignore distant obstacles and they change the absolute locations of obstacles into relative radial offset values about a. Both of these processes are important because they play important roles in producing realistic crowd movement on arbitrary surfaces.

Determining relevant obstacles can be done using a $2DRelevant$ function that quickly finds a subset of all the obstacles in a scene that are relevant to the current agent. Formally, $2DRelevant$ can be defined as:

$$2DRelevant : a, O \rightarrow O_{Relevant} \subseteq O \quad (4)$$

where a is the agent in question, O is the set of all obstacles, and the result $O_{Relevant} \subseteq O$ gives us obstacles that are relevant to $2DMove$. Without a $Relevant$ function, crowd simulation algorithms can be $O(n^2)$ since the angle and distance between each pair of agents must be calculated. Once the $2DRelevant$ function has found a small subset of obstacles, the $Offset$ function loops over each obstacle in $O_{Relevant}$ and finds the angle and distance to the obstacle. Off is a set containing a tuple in the power set of angles and distances.

$$2DOffset : a, o \in O_{Relevant} \rightarrow Off \in (\theta \times d) \quad (5)$$

Once all the offsets to all relevant obstacles are calculated, they are put in a set R. Using R, collision-free movement for agents can be found using some $2DMoveRelevant$ function as follows:

$$2DMoveRelevant : R \in (\theta \times \mathbb{R}), a \rightarrow \theta', v' \quad (6)$$

4.2 3D Obstacle Avoidance Abstraction

It is not surprising that previous 3D crowd simulation research (e.g. Torchelsen et al. [44]) has taken a similar approach to the 2D crowd simulation work. In other words, algorithms find relevant obstacles and then do obstacle avoidance with values centered about the current agent's location. If during this process the offsets to obstacles are converted from 3D offsets to 2D offsets then an algorithm can use a $2DMoveRelevant$ to do obstacle avoidance. By using $2DMoveRelevant$ an algorithm can leverage the very successful work in 2D obstacle avoidance without starting from scratch. Formally, we define the 3D obstacle avoidance problem as follows:

$$3DMove : a, O \in \mathbb{R}^3, mesh \rightarrow \theta', v' \quad (7)$$

Previous work follows this pattern and usually provides some sort of relevancy and offset function similar to the ones in 2D work as follows:

$$3DRelevant : a, O \rightarrow O_{Relevant} \subseteq O \quad (8)$$

$$3DOffset : a, o \in O_{Relevant} \rightarrow Off \in (\theta \times d) \quad (9)$$

Unlike 2D surfaces, finding the set of relevant obstacles and the radial offsets to obstacles' locations is a difficult problem on a 3D surface. Most previous approaches have not focused on the details of implementing $3DRelevant$ and the results have contained clear artifacts. There are two main approaches to implementing $3DOffset$: a surface distance approach and a Euclidean approach.

4.3 Artifacts in Current Approaches

One can implement $3DOffset$ using a surface distance algorithm such as A* or fast marching methods [3]. Once the path to an obstacle is calculated, the distance to the obstacles is the length of the path and the angle to the obstacle is the angle of the first segment of the path. Unfortunately, this fails to produce realistic crowd simulation since agents can collide even when their surface distance is extremely high. Fig. 6a depicts an agent (the orange agent on the floor) and the obstacle agent (the blue agent on the ceiling). The orange agent is at most a few meters from the blue agent, but the surface distance is the distance from the orange agent to the nearest wall, up the wall, and back across the ceiling. Even with an accurate surface distance algorithm, the orange agent does not think it as about to collide with the blue agent even though a collision is imminent. The angle output of a surface path implementation of $3DOffset$ will produce similar artifacts.

The other method for determining angles and distances is to use a Euclidean distance (e.g. Torchelsen et al. [44]). Given a vector v that is the vector between the

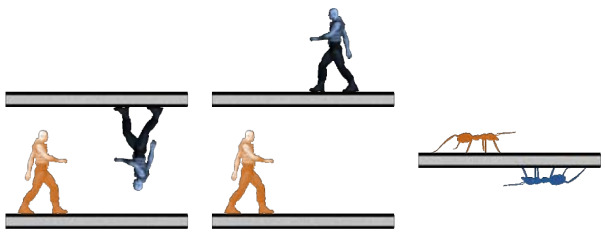

Fig. 6. Surface distance or Euclidean distance can produce false positives or negatives. a) A surface distance false negative. b&c) Euclidean distance false positives.

agent in question a and $obstacle$, Euclidean distance is the length of v. Euclidean angle is found by projecting v onto the plane with the same normal as a. Torchelsen et al. [44] accurately point out that Euclidean offset does not struggle with the false negative issues of a surface distance algorithm. However, instead of suffering from false negatives, a Euclidean implementation of $3DOffset$ suffers from the opposite problem: collision false positives. Consider Fig. 6b and Fig. 6c. In both cases a Euclidean distance approach will inaccurately report that the orange agent is about to collide with the blue agent. This is not correct and the agents will evade each other even though there is no imminent collision. The resulting unnatural motion is very obvious in multistory buildings, ants on a tree, or people in a subway station.

Neither the surface path nor the Euclidean implementation of $3DOffset$ addresses the last issue with crowds constrained to arbitrary surfaces in 3D space: the non-spherical shape of virtual characters. On arbitrary surfaces in 3D space, two agents can be on surfaces that are perpendicular to each other. If the crowd simulation algorithm only represents the collision surface of characters with a small sphere near each agent's feet, the heads of two agents may be on a collision course even if their collision spheres are not. Simply enlarging collision spheres alone will cause jams even when agents could easily pass each other.

Our approach to obstacle avoidance on arbitrary surfaces removes these problems. We remove false positives and false negatives by improving the $3DRelevant$ function. We remove problems with the non-spherical nature of agents by expanding the $3DOffset$.

4.4 Removing These Artifacts

Notice that in all false positive cases (see Fig. 6b and Fig. 6c), the obstacle agent that is incorrectly avoided is not visible to the orange agent. Thus, by improving $3DRelevant$ to discard agents that are not visible to the agent in question, a Euclidean implementation of $3DOffset$ would not have false positives. We have verified this point through extensive testing. Resolving collision detection issues in $3DRelevant$ instead of $3DOffset$ is one of the key contributions of this work.

A full synthetic vision algorithm could be used to determine which agents are mutually visible. This approach has been done using ray tracing by *Massive* [47] and on the GPU in Ondrej et al.'s 2D crowd simulation paper [46]. We do not embrace the ray tracing approach of *Massive* since our system is designed to run in real-time. Similarly, since we are placing agents on large meshes with up to 100,000 triangles, rendering the viewport of each agent to do synthetic vision on the GPU has proven impractical.

Fortunately, we can use geometry to make a quick and close approximation to agent visibility (see Fig. 7). Since each agent is constrained to a manifold, we know there is a surface at the feet of each agent. Leveraging this, we add two additional checks to our $3DRelevant$ function for each obstacle agent. We find the vector from the agent in question to the agent that is proposed as an obstacle. We call the vector from our agent's head to the other agent v_{head} and the vector from our agent's feet to the other agent v_{feet}. We then assume that the surface the obstacle agent is standing on can be locally approximated by a one to two meter radius circle at its feet and that the normal of this circle is the same as the obstacle agent's normal.

To do our approximate vision algorithm, we check two cases. First, we check to see if v_{head} and the agent's up vector have a positive dot product. This means that the agent has to look up to see the obstacle agent. If this is the case, then we check to see if v_{head} penetrates the circle about the obstacle agent's feet. If v_{head} penetrates this circle, then we remove this agent from the set $O_{Relevant}$. We make this assertion because we have very strong evidence that there is no way that the obstacle agent is currently visible. We further assert that if the other agent's head is a full body length above our agent, then they are not visible. Second, we then check to see if our agent's up vector and v_{feet} have a negative dot product. If so, our agent would have to look through the floor to see the other agent. Thus, we have strong evidence that the obstacle agent is not visible and we remove the other agent from $O_{Relevant}$. This approximation is not guaranteed to exactly match a full synthetic vision algorithm, but in practice there are no false positives or false negatives in the crowd movement.

The choice of a one to two meter disc is not arbitrary. Since two meters is approximately the height of our agents (the virtual model we use is quite tall) the disc has the same radius as the height of our agent. This is important because if it were larger than the height of our agent, then two agents approaching at a right angle (such as the corner of a cube) would incorrectly remove each other from the relevant obstacle set when they could actually see each other. On the other hand, stride length is generally at least a meter, which provides a logical lower bound on local flatness of the surface at an agent's feet. A radius smaller than a stride's width would mean the agent would not be able to plant its feet on the surface. We saw no visible difference in behavior in our agents with values between values of 1 and 2 meters. In cases where the height of the agents or their stride differs from these bounds, the size of the disc can be trivially changed.

This improvement to $3DRelevant$ does not handle the non-spherical nature of agents. In the 2D crowd simulation, the fact that agents are taller than they are wide is irrelevant, but in the 3D case this becomes important. There are two simple but error-prone ways of handling this problem. One solution is to represent the collision area of an agent with a small sphere about its feet. Unfortunately this leaves the torso and head of the agent exposed to collisions. Another approach would be to surround each agent by a sphere whose diameter

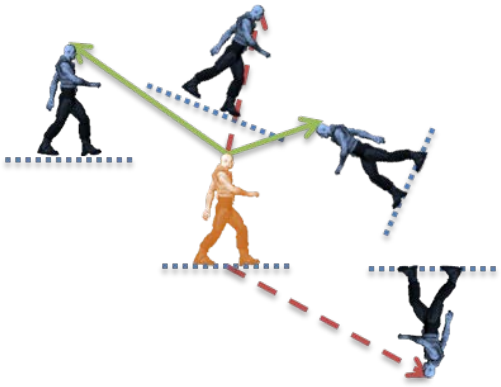

Fig. 7. Our improvement to the $Relevant$ function. The relevant agent (orange) checks if the vector to a possible obstacle agents penetrates an approximate surface at each agent's feet (dotted blue line). If the vector penetrates these surfaces (dashed red arrow) the obstacle agent is not considered relevant. If the vector does not penetrate (lighter green arrow) the agent is considered relevant.

Fig. 8. A surface with agents close together on the ceiling and floor. Without an accurate distance and angle function, agents will run into each other.

is the height of the agent. However, this would make agents passing each other incorrectly think they were in collision.

To resolve this problem, we define the collision surface of agents using multiple volumes instead of just one. This idea has been previously used in 2D crowd simulation to allow very dense 2D crowd simulations and to account for an agent's stride (see [22]). However, this has never been used in 3D crowd simulation to deal with an agent's height. Instead of one small sphere or one giant sphere, we define each character with multiple spheres in a way that approximates a cylinder and covers the agent completely. We choose spheres over other geometric primitives since spheres allow for simple collision detection and avoidance. Additionally, it is trivial to change the placement and number of the spheres to match the shape of agents (i.e. more for taller agents or fewer for shorter agents).

For the center of each of these spheres, we use a traditional implementation of $3DOffset$ and return the union of tuples. This is the reason we defined $3DOffset$ as returning a set in Eqn. 9. Formally, if o_{up} is the normal of the obstacle agent for which we want a more realistic representation, then our implementation of $3DOffset$ is as follows:

$$3DOffset_{Improved} = \bigcup_{i=0}^{2} Offset(a, o + i \cdot o_{up}) \qquad (10)$$

These improvements have a minuscule computational footprint, have a clear impact on the motion of the agents in a crowd, and can handle intentionally complex scenes such as that in Fig. 8. We give numerical results for the effectiveness of our obstacle avoidance technique in Sec. 5.

4.5 Density

Every crowd simulation has a density limitation, or a point at which agents will stall simply because there are too many agents in a choke point to proceed without collisions. Our algorithm does not propose a new obstacle avoidance algorithm but runs on top of almost any previous 2D crowd simulation algorithm. Thus, the density limitations of our implementation depend on the current underlying 2D crowd simulation. As expected, velocity obstacles handled dense situations better than social forces. Our algorithm could run with methods that allow for advanced jamming resolution such as pushing (e.g. Pelechano et al. [23]) or the use of complex jamming resolution rules (e.g. Singh et al. [31]).

5 Results

The goal of this work is to present our "surface as a whole" approach to crowd simulation. We have asserted that by using discrete geodesic algorithms, our path planning improvements, and our approximate vision algorithm we can simulate crowds across any topology. We validate these claims by giving evidence of our algorithm's completeness, speed, quality, and correctness. First, we show that our path planning approach always finds path in non-degenerate cases. Then we give overall speed results for our algorithm. Next, we give results about specific details of our path planning improvements. These results show that our improvements produce quality motion that can actually increase the frame rate of the simulation. Lastly, we give results about our approximate vision algorithm. These results show the correctness of our algorithm since our collision rates are practically zero.

5.1 Discrete Geodesic Speed Results

In Sec. 3.2 we asserted that a "surfaces as a whole" approach would be effective for crowd simulation on arbitrary topologies. We also asserted that discrete geodesic algorithms would always find a path if one existed and that they run at real-time rates on high triangle-count surfaces.

Our first claim is that our path planning algorithm will find a path if one exists. There are numerous proofs showing that graph-based search algorithms will find a

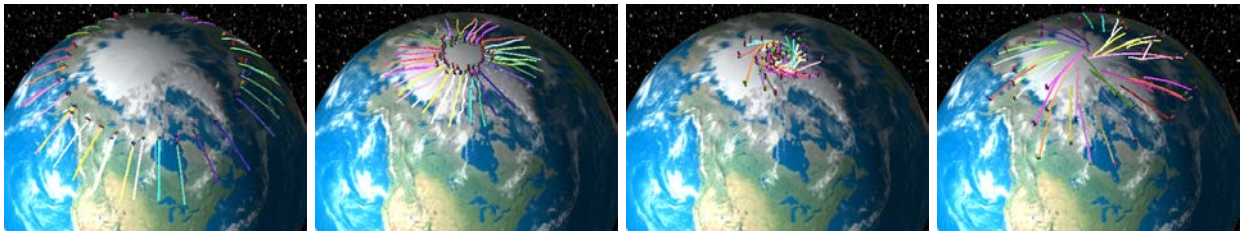

Fig. 9. Our whole surface approach to simulation produces spontaneous global effects seen in real crowds, such as the swirl pattern shown here.

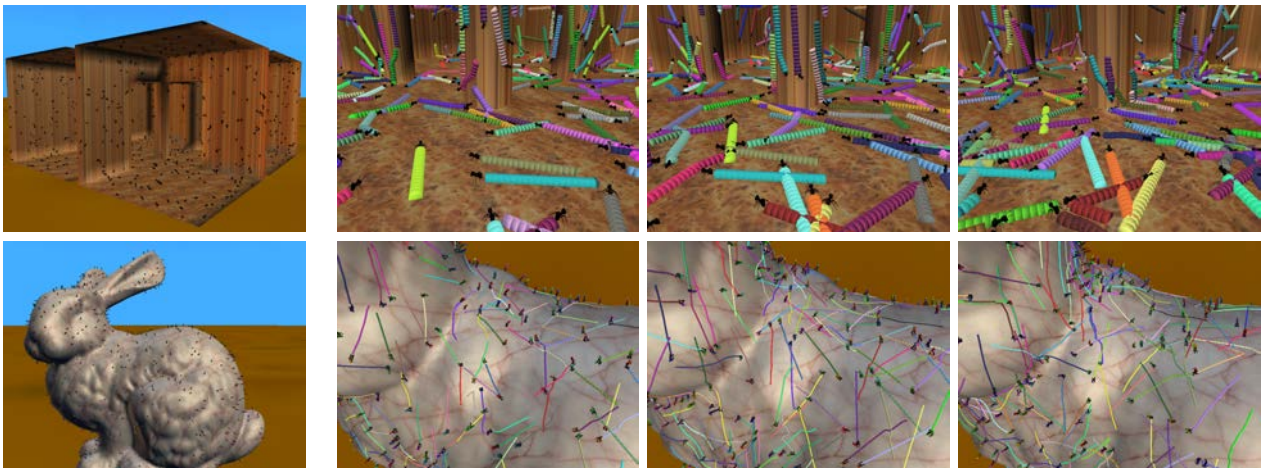

Fig. 10. Our agents move naturally and avoid collisions on complex surfaces even with perpendicular edges (ants in a room above) or when there is no distortion-free mapping from 3D to 2.5D (agents on the Stanford Bunny below).

path between two nodes if one exists (e.g. [48]). Thus, as long as an agent is connected by the surface to its destination, our graph-based planners will always find a path. We also used an iterative refinement discrete geodesic algorithm (detailed later). This algorithm starts off with a graph-based step, which means that it too will always find a path if one exists. Other discrete geodesic algorithms discussed in our previous work section have corresponding proofs of their completeness (e.g. [10]). Our experience verified what these proofs establish. On all our surfaces all our agents were able to find a path from their initial locations to their destinations. Our path planning improvements also produce feasible paths as long as the surface was not degenerate. As noted earlier, in this paper we do not address the problem of finding walkable surfaces. Thus, the width and curvature of our walkways were such that corridors would stay on our surface. Future work that combines work on finding walkable surface areas could label areas of the surface that are not wide enough for a full corridor expansion. In that case the algorithm could make sure the corridor does not expand wider than thin walkways.

To verify our speed claims about discrete geodesic algorithms, we generated crowds of 1,000 agents across an array of surfaces inspired by architecture, nature, science fiction, insect colonies, and topologically unique surfaces. These models range from a dozen triangles to over a hundred thousand, as listed in Table 1. Many of these surfaces are shown in figures (see column two of Table 1). The website http://sites.google.com/site/academicreviewdata/ contains mesh files and images for the surfaces shown in Table 1. This test suite contains more surface features and higher polygon counts than any other test suite we were able to find. This adds validity to our claim that our algorithm works on a larger set of surfaces than any previous work. Our tests were run on an Intel 17-2600 processor. Computation was done on the CPU (as opposed to previous focused on the GPU) using managed code written in C#.

The results in Table 1 give the frame rates using our improvements. These tests were run using a weighted A* algorithm for path planning and two different 2D obstacle avoidance algorithms: social forces and reciprocal velocity obstacles. The results are sorted by the number of triangles in each mesh. In general, the surface area of each mesh was proportional to the number of triangles. Thus, obstacle avoidance tended to be the dominant computational requirement for lower polygon-count surfaces while path planning tended to be the dominant requirement on higher polygon-count surfaces. Each trial had 1,000 agents. This table also lists how long it took our path planning algorithm to plan on the first frame. This is noteworthy since all agents must do path planning during this frame. Thus, this result gives a sense of how long it takes to compute paths for 1,000 agents.

TABLE 1
Results from our crowd simulation on 30 surfaces.

Name (Triangles)	*Example Fig.s*	FPS - 1,000 agents *S. Forces*	*RVO*	1^{st} *Path Plan (s)*
One Triangle (1)	-	514.25	330.83	0.07
Simple Cube (12)	-	155.01	60.56	0.29
Building Exterior (68)	-	172.32	66.67	0.29
Soccer Ball (80)	Fig 3	185.46	95.07	0.52
Bucky (108)	-	192.89	87.86	0.08
Building 1 (174)	Fig 1j	129.33	55.73	0.377
Building w/ Exit (226)	-	149.35	55.72	0.55
Building 2 (232)	-	310.29	206.05	0.35
Cube Holes (864)	-	202.34	92.31	0.18
Inverse Cube (1.2k)	-	189.61	96.39	0.55
Floor and Ceiling (1.6k)	Fig 8	160.3	73.97	0.46
Globe (1.7k)	Fig 1f	178.04	50.86	0.34
Golevka Asteroid (2k)	Fig 1b	232.21	153.84	0.35
Kleopatra Asteroid (2k)	-	430.89	274.08	0.32
Ants on Food (2.3k)	Fig 1h	198.48	88.36	0.33
Mobius w/ Holes (3.2k)	Fig 1d	170.19	78.87	0.6
Torus (3.2k)	Fig 1c	158.6	61.41	0.43
Mobius Strip (3.2k)	-	170.07	77.94	0.8
Reverse Torus (3.2k)	-	193.9	90.87	0.43
Room 2 (3.6k)	Fig 10	160.23	55.29	0.41
Room 1 (3.6k)	Fig 1i	180.1	57.51	0.42
Asteroid (6.4k)	-	211.19	100.63	0.52
Tree (14k)	Fig 1g	175.22	79.3	2.62
Medium Cube (19k)	-	143.72	80.82	3.62
Twisted Wire (40k)	-	55.51	49.49	31.13
Tetrahedron (59k)	Fig 1a	170.19	147.05	6.31
Stanford Bunny (70k)	Fig 1e	65.94	51.79	15.8
Abstract Art (102k)	-	128.81	120.38	9.13
Huge Cube (108k)	-	115.87	105.68	3.15
Olympic Rings (109k)	-	47.93	41.4	10.86

All of our models ran at 40 frames per second or faster, including the Stanford bunny with over 69,000 triangles and our abstract art, Olympic rings, and huge cube models, all of which had more than 100,000 triangles. Note also that the social forces results ran significantly faster than RVO, indicating that not all of our time was spent in path planning. Note also that the the first frame path planning time only exceeded ten seconds in three of our cases. Inspection of the path planning showed that the nature of these surfaces made it difficult for the A* heuristic to find the shortest path quickly. This is something our twisted wire surface was specifically designed to test, so we are not surprised by this result.

We believe these results validate our assertion that using discrete geodesics is both a complete and quick way to do path planning. The following results show that our path planning results produce quality motion.

5.2 Path Planning Results

In our path planning section, Sec. 3, we asserted that our away from edges, corridors, and line of sight improvements could produce natural crowd movement on surfaces of arbitrary topology. We validate these claims by showing that our improvements remove stalling. Then we show that our improvements lead to frequent destination arrivals even with less-accurate path planners. We then show that our improvements actually increase our frame rate since they reduce computationally-expensive jamming situations. Combined with our previous speed results, these results validate our claim that our approach creates quality motion on arbitrary surfaces with little or no stalling.

In Sec. 3.1, we discussed stalling as a quantitative way of measuring the quality of agent motion in a crowd simulation. To measure quality of our path planning improvements, we ran our simulation with and without our improvements. We then tracked the amount of stalling. Our results show that our path planning improvements clearly reduce stalling as shown in Fig. 11.

To generate the results in Fig. 11, we created crowds of 1,000 agents in a multistory building (see Fig. 1j). We chose this surface for our tests since the sharp turns on the surface made it the most prone to stalling. In our first trials agents had no improvements, next agents only had the line of sight improvement, then agents had the line of sight and the away from walls improvement, and last the agents had all improvements including the corridors improvement. For each crowd we considered an agent stalled if it had not moved more than a shoulder's width in five seconds. We ran our test for 80 seconds. Being stalled was almost always the result of being in a crowd jam or becoming lost by not getting around the corner of the stairs. For each crowd we averaged the percent of stalled agents for each frame. Without any improvements, over ten percent of the agents were stalled by the end of the simulation. With the line of sight optimization alone about five percent of agents were stalled. With the away from edges and line of sight improvement jamming fell to about half of a percent, but jamming was still noticeable. With all improvements including the corridors improvement, the percentage stayed at practically zero during the entire simulation. We tested our improvements cumulatively instead of by themselves since some of the improvements depend algorithmically on each other to work (e.g. corridors depends on away from edges). Both visually and nu-

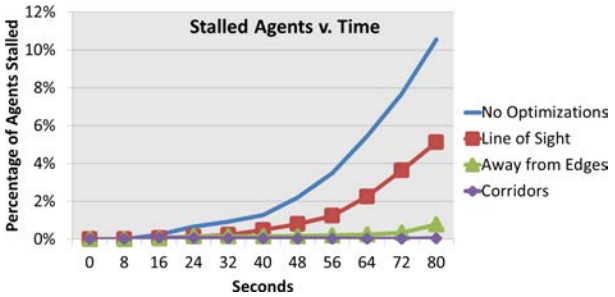

Fig. 11. Stalling versus time in 80s of simulation on an surface similar to Fig. 5. Each of our improvements reduced stalling, but the line of sight and away from edges improvements still showed clear stalling. All improvements combined (listed as corridors) reduced stalling to practically 0 throughout the simulation.

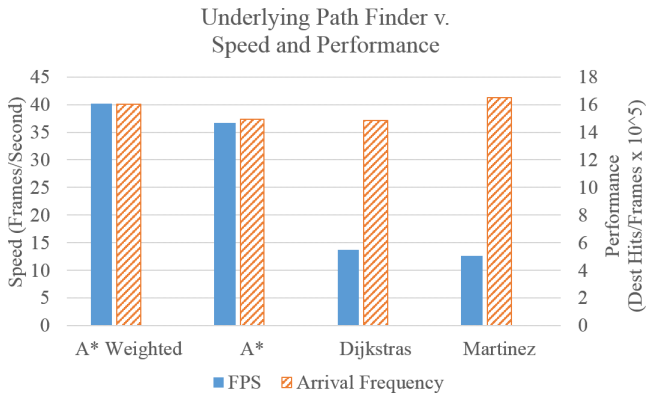

Fig. 12. How the underlying discrete geodesic algorithm affects speed and performance on our Olympic Rings environment (see Table 1). Speed is given in frames per second. Performance is measured by arrival frequency. This figure shows that our improvements let us increase speed without reducing performance.

merically the improvement was dramatic.

As mentioned in our comments on density, there were situations in which we could cause stalling even with all our improvements. One way to cause stalling was simply to increase the size of agents until there was no way for them to resolve collisions. For example, we could simply make the agents on the Mobius strip larger and larger until they could not pass each other. Also, sometimes after long simulation times jams formed as the result of rare and complicated agent patterns that social forces or RVO could not resolve. We are interested in using more complex jam-resolution algorithms in future work to see if these rare jams could be resolved. The only surface feature that seemed stall-prone with our approach were gorge-like structures. In gorge-like places two agents going opposite directions could jam since they were boxed in by the sides. The Stanford Bunny has such a feature around part of its neck. One piece of possible work in the area of navigable mesh generation could be to try and detect thin gorges and remove them from the mesh.

We further analyzed the speed and performance of our algorithm based on the underlying discrete geodesic algorithm. Fig. 12 shows how different discrete geodesic algorithms affect the speed and performance of our algorithm. Speed is measured in frames per second. Performance is measured in how frequently agents arrive at their destinations. Better path planning should give agents shorter paths to their destinations. This in turn should result in agents reaching destinations quicker, resulting in better performance as measured by arrival frequency. At the beginning of each simulation agents are given a random location with a random destination. When an agent reaches its destination it is assigned a new random destination and the $destHits$ count increments. At the end of each simulation we divide $destHits$ by the number of frames and agents. The resulting number represents the probability of an agent reaching it destinations on any given frame. Since this normalized value is too small to be seen on this graph, we scaled by 100,000 to get the arrival frequency reported in Fig. 12. Unlike normalized values in [26], this metric is only meaningful when comparing algorithms on the same surface. However, since our simulations run 1,000 agents the arrival frequency results are a robust indicator of our path planning algorithm's effectiveness.

We used four different discrete geodesic algorithms in Fig. 12: weighted A*, A*, Dijstra's, and Martinez et al.'s iterative refinement algorithm [12]. The first three algorithms only return paths that follow the vertices of triangles. The weighted A* algorithm is the fastest of the three but has the possibility of returning the longest paths. The iterative refinement algorithm returns shorter paths. It also returns the absolute shortest path if its initial guess is close to the true shortest path. We could have used any number of algorithms as the seed to the iterative refinement algorithm, but chose weighted A* for speed reasons. Results are shown from our Olympic Ring model (see Table 1) since it has the highest triangle count of any of our surfaces. Notice in this graph that there is a monotonic decrease in speed as we progress from less accurate to more accurate path finding algorithms. This is expected since more accurate algorithms require more computation. However, notice that the performance as measured in arrival frequency remains largely unchanged. The reason is that our line of sight algorithm compensates for the weaknesses of less accurate path planners. Thus, with our improvements, weighted A* runs three times faster than the iterative algorithm but produces results of equal quality.

Fig. 13 shows the frames per second for our algorithm moving 1,000 agents in a 15-story building (Fig. 1j). Results are shown for crowds with no improvements, line of sight only, away from edges and line of sight, and all our optimizations including corridors. Each bar shows the frames per second with the error bars showing

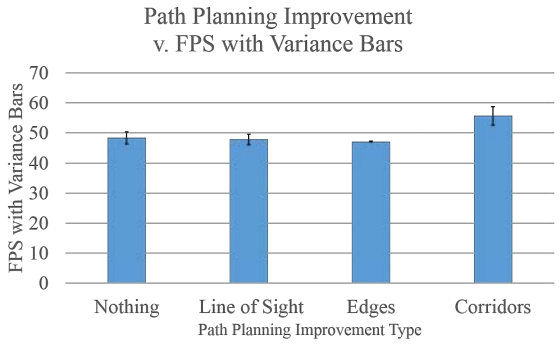

Fig. 13. Graph of how our improvements affect simulation speed running 1,000 agents in a 15-story building (see Fig. 1j). Error bars show the frames per second variance across multiple trials. Notice that of all our improvements combined (labeled corridors) ran fastest.

the variance of our trials. Our initial goal in producing these results was to see how much computation time each of our improvements cost. Notice that adding the line of sight and away from edges optimizations slightly reduced the frames per second of our algorithm. However, all the optimizations combined resulted in a net increase in frames per second. This is true even compared to no improvements at all. The reason is that when agents stall, the computation time increases as each agent has more and more agents against which it does obstacle avoidance. When agents flow by each other naturally then each agent only has to do collision avoidance calculations against a few other agents. Thus, our improvements combined resulted in a net reduction in computation time and net increase in frames per second. This speed improvement is statistically significant with a p value of .005.

5.3 Obstacle Avoidance Results

We also used manual inspection and statistical analysis to verify the effectiveness of our obstacle avoidance approach. In doing so, we looked to see that our algorithm did not suffer from the false negatives and false positives of previous work and that our approach did not suffer from collisions in general. To do this we ran our algorithm using three very different 2D local obstacle avoidance algorithms: a social forces model based on Helbing and Molnar's work [15], an anticipation model based on Ondrej et al. [46], and reciprocal velocity obstacles (RVO) [17]. We ran each of these on all 30 of our surfaces, which we detailed earlier. For each 2D obstacle avoidance algorithm we found the median collision percentage for the simulation. The only algorithm with a noticeable collision rate was our implementation of the anticipation model, which we found also had noticeable collision rates in our purely 2D crowd simulation framework. Thus, even though the timing results mirror those of reciprocal velocity obstacles, we do not include them in our table. Using social forces or RVO resulted in negligible median collision rates of .05% and .009% respectively. Visual inspect corroborated this quantitative result: We found that our agents did not have the clear collision false positives and false negatives of a surface distance or Euclidean distance approach. We believe these results validate our obstacle avoidance algorithm for agents on arbitrary 3D surfaces.

In addition to the statistical measures listed above, we noticed several features of our simulated crowds that closely resembled real crowds. For example, as shown in Fig. 5, agents spontaneously form lanes even on 3D surfaces. Interesting locations where such lanes would form include the inside of the torus (see Fig. 1c) and inner edge of the Mobius strip (see Fig. 1d). Similarly, as shown in Fig. 9, our agents produce the swirling effect noted in real crowds. Notice that we had the agents converge at one of the poles of the globe where the tessellation of the surface is the highest and the computation is the most difficult. Despite the high density of triangles, the agents moved naturally around each other and reached their destinations.

Combined, we believe these results validate our claim that a "surface as a whole" approach to crowd simulation produces natural movement across surfaces of any topology. Our algorithm is complete because it will always find a path if one exists, it runs at real-time speeds even on high triangle-count surfaces, it produces quality motion, and agents choose correct, collision-free movements.

6 Future Work

Our "surface as a whole" approach is an effective way of simulating crowds on arbitrary topologies and opens up future work in several areas. As mentioned earlier, algorithms such as corridors and navigation meshes (see Sec. 2) have proven effective for 2/2.5D crowd path planning. In Sec. 3 we explained why these approaches are not suitable for arbitrary surfaces, but we are still interested in studying the algorithmic changes required to take these algorithms from 2D to 3D. Additionally, the results of this work open up new areas in surface analysis for finding walkable areas on a larger set of surfaces than the research now addresses. We are also interested in objectively comparing our algorithm to real footage of crowds on 3D surfaces. This idea opens up interesting avenues in computer vision, including appropriate ways of tracking motion of people in multiple stories of buildings or insects crawling around on highly concave surfaces.

References

[1] F. Lamarche and S. Donikian, "Crowd of virtual humans: A new approach for real time navigation in complex and structured environments," *Computer Graphics Forum*, vol. 23, no. 3, pp. 509–518, 2004.

[2] S. Singh, M. Kapadia, B. Hewlett, G. Reinman, and P. Faloutsos, "A modular framework for adaptive agent-based steering," *Symposium on Interactive 3D Graphics and Games*, pp. PAGE–9, 2011.

[3] J. Sethian, "Fast marching methods," *SIAM review*, vol. 41, pp. 199–235, 1999.

[4] R. Geraerts and M. Overmars, "The corridor map method: A general framework for real-time high-quality path planning," *Proceedings of Computer Animation and Virtual Worlds*, vol. 18, no. 2, pp. 107–119, 2007.

[5] M. Kallmann, "Navigation queries from triangular meshes," *Proceedings of Motion in Games*, pp. 230–241, 2010.

[6] A. Kamphuis, M. Mooijekind, D. Nieuwenhuisen, and M. Overmars, "Automatic construction of roadmaps for path planning in games," *International Conference on Computer Games: Artificial Intelligence, Design and Education*, pp. 285–292, 2004.

[7] J. Pettré, P. Ciechomski, J. Maïm, B. Yersin, J. Laumond, and D. Thalmann, "Real-time navigating crowds: Scalable simulation and rendering," *Proceedings of Computer Animation and Virtual Worlds*, vol. 17, no. 3-4, pp. 445–455, 2006.

[8] D. Hale, "A growth-based approach to the automatic generation of navigation meshes," *Dissertation, The University of North Carolina Charlotte*, 2012.

[9] S. Curtis, J. Snape, and D. Manocha, "Way portals: Efficient multi-agent navigation with line-segment goals," *Proceedings of ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, pp. 15–22, 2012.

[10] J. Mitchell, D. Mount, and C. Papadimitriou, "The discrete geodesic problem," *SIAM Journal on Computing*, vol. 16, no. 4, pp. 647–668, 1987.

[11] J. Chen and Y. Han, "Shortest paths on a polyhedron," *Computational Geometry*, pp. 360–369, 1990.

[12] D. Martínez, L. Velho, and P. Carvalho, "Computing geodesics on triangular meshes," *Computers & Graphics*, vol. 29, no. 5, pp. 667–675, 2005.

[13] R. Kimmel and J. Sethian, "Computing geodesic paths on manifolds," *National Academy of Sciences of the United States of America*, vol. 95, no. 15, p. 8431, 1998.

[14] C. Reynolds, "Flocks, herds and schools: A distributed behavioral model," *Proceedings of ACM SIGGRAPH Computer Graphics*, vol. 21, no. 4, pp. 25–34, 1987.

[15] D. Helbing and P. Molnar, "Social force model for pedestrian dynamics," *Physical Review*, vol. 51, no. 5, pp. 4282–4286, 1995.

[16] P. Fiorini and Z. Shiller, "Motion planning in dynamic environments using velocity obstacles," *The International Journal of Robotics Research*, vol. 17, no. 7, p. 760, 1998.

[17] J. Van den Berg, M. Lin, and D. Manocha, "Reciprocal velocity obstacles for real-time multi-agent navigation," *Proceedings of Robotics and Automation*, pp. 1928–1935, 2008.

[18] S. Guy, J. Chhugani, S. Curtis, P. Dubey, M. Lin, and D. Manocha, "Pledestrians: A least-effort approach to crowd simulation," *Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pp. 119–128, 2010.

[19] N. Courty and S. Musse, "Fastcrowd: Real-time simulation and interaction with large crowds based on graphics hardware," *Short Paper in ACM SIGGRAPH/EuroGraphics Symposium on Computer Animation*, pp. 177–187, 2004.

[20] R. Narain, A. Golas, S. Curtis, and M. Lin, "Aggregate dynamics for dense crowd simulation," *ACM Transactions on Graphics (TOG)*, vol. 28, no. 5, p. 122, 2009.

[21] I. Karamouzas, P. Heil, P. van Beek, and M. Overmars, "A predictive collision avoidance model for pedestrian simulation," *Proceedings of Motion in Games*, pp. 41–52, 2009.

[22] S. Singh, M. Kapadia, G. Reinman, and P. Faloutsos, "Footstep navigation for dynamic crowds," *Proceedings of Computer Animation and Virtual Worlds*, vol. 22, no. 3–3, pp. 151–158, 2011.

[23] N. Pelechano, J. Allbeck, and N. Badler, "Controlling individual agents in high-density crowd simulation," *Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pp. 99–108, 2007.

[24] A. Treuille, S. Cooper, and Z. Popović, "Continuum crowds," *ACM Transactions on Graphics (TOG)*, vol. 25, no. 3, pp. 1160–1168, 2006.

[25] H. Jiang, W. Xu, T. Mao, C. Li, S. Xia, and Z. Wang, "Continuum crowd simulation in complex environments," *Computers & Graphics*, vol. 34, no. 5, pp. 537–544, 2010.

[26] M. Kapadia, M. Wang, S. Singh, G. Reinman, and P. Faloutsos, "Scenario space: characterizing coverage, quality, and failure of steering algorithms," *Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pp. 53–62, 2011.

[27] M. Kapadia, M. Wang, G. Reinman, and P. Faloutsos, "Improved benchmarking for steering algorithms," *Proceedings of Motion in Games*, pp. 266–277, 2011.

[28] N. Pelechano, C. Stocker, J. Allbeck, and N. Badler, "Being a part of the crowd: Towards validating VR crowds using presence," *Autonomous Agents and Multiagent Systems*, pp. 136–142, 2008.

[29] S. Singh, M. Kapadia, P. Faloutsos, and G. Reinman, "Steerbench: A benchmark suite for evaluating steering behaviors," *Proceedings of Computer Animation and Virtual Worlds*, vol. 20, no. 5-6, pp. 533–548, 2009.

[30] S. Singh, M. Naik, M. Kapadia, P. Faloutsos, and G. Reinman, "Watch out! a framework for evaluating steering behaviors," *Motion in Games*, pp. 200–209, 2008.

[31] S. Singh, M. Kapadia, P. Faloutsos, and G. Reinman, "An open framework for developing, evaluating, and sharing steering algorithms," *Proceedings of Motion in Games*, pp. 158–169, 2009.

[32] D. Thalmann and S. R. Musse, *Crowd simulation*. Wiley Online Library, 2007.

[33] N. Pelechano, J. Allbeck, and N. Badler, "Virtual crowds: Methods, simulation, and control," *Synthesis Lectures on Computer Graphics and Animation*, vol. 3, no. 1, pp. 1–176, 2008.

[34] W. Shao and D. Terzopoulos, "Autonomous pedestrians," *Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pp. 19–28, 2005.

[35] F. Lamarche, "Topoplan: A topological path planner for real time human navigation under floor and ceiling constraints," *Computer Graphics Forum*, vol. 28, no. 2, pp. 649–658, 2009.

[36] H. Jiang, W. Xu, T. Mao, C. Li, S. Xia, and Z. Wang, "A semantic environment model for crowd simulation in multilayered complex environment," *Proceedings of the 16th ACM Symposium on Virtual Reality Software and Technology*, pp. 191–198, 2009.

[37] L. Deusdado, A. Fernandes, and O. Belo, "Path planning for complex 3d multilevel environments," *Proceedings of the 24th Spring Conference on Computer Graphics*, pp. 187–194, 2008.

[38] W. van Toll, A. Cook, and R. Geraerts, "A navigation mesh for dynamic environments," *Proceedings of Computer Animation and Virtual Worlds*, vol. 23, no. 6, pp. 535–546, 2012.

[39] S. Rodriguez, J. Denny, A. Mahadevan, J. Vu, J. Burgos, T. Zourntos, and N. Amato, "Roadmap-based pursuit evation in 3D structures," *Proceedings of Computer Animation and Social Agents (CASA)*, 2011.

[40] N. Sturtevant, "A sparse grid representation for dynamic three-dimensional worlds," *Proceedings of the Seventh AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, pp. 73–78, 2011.

[41] T. Stoyanov, M. Magnusson, H. Andreasson, and A. Lilienthal, "Path planning in 3D environments using the normal distributions transform," *Proc. of IEEE/RSJ Int. Conf. on Intelligent Robots and Systems. IROS*, pp. 3263–3268, 2010.

[42] T. Jund, P. Kraemer, and D. Cazier, "A unified structure for crowd simulation," *Proceedings of Computer Animation and Virtual Worlds*, vol. 23, no. 3–4, pp. 311–320, 2012.

[43] S. Levine, Y. Lee, V. Koltun, and Z. Popović, "Space-time planning with parameterized locomotion controllers," *ACM Transactions on Graphics (TOG)*, vol. 30, no. 3, p. 23, 2011.

[44] R. Torchelsen, L. Scheidegger, G. Oliveira, R. Bastos, and J. Comba, "Real-time multi-agent path planning on arbitrary surfaces," *Proceedings of ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*, pp. 47–54, 2010.

[45] B. Ricks and P. Egbert, "Improved obstacle relevancy, distance, and angle for crowds constrained to arbitrary manifolds in 3D space," *Proceedings of Eurographics*, pp. 73–76, 2012.

[46] J. Ondřej, J. Pettré, A. Olivier, and S. Donikian, "A synthetic-vision based steering approach for crowd simulation," *ACM Transactions on Graphics (TOG)*, vol. 29, no. 4, pp. 123:1–123:9, 2010.

[47] Massive, "http://www.massivesoftware.com/," *URL*, November 2012.

[48] P. Hart, N. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *Systems Science and Cybernetics, IEEE Transactions on*, vol. 4, no. 2, pp. 100–107, 1968.

Brian C. Ricks is currently a computer science doctoral student at Brigham Young University. Brian Ricks' research interests include 3D crowd simulation, social dynamics in crowds, and articulated motion. He work has been published in the the Visual Computer journal, Intelligent Computer Graphics, the proceedings of Eurographics, and the proceedings of Computer Graphics International.

Parris K. Egbert is Computer Science Department Chair at Brigham Young University. His research includes real-time 3D computer graphics, global illumination, tools for computer animation, and the creation and navigation of virtual environments. His work has been published in SIGGRAPH, CVPR, Computational Intelligence, and TOG. Dr. Egbert is on the BYU's Center for Animation executive committee where student films have won 12 Student Emmy awards.