Semi-Automatic Navigation on 3D Triangle Meshes Using BVP Based Path-Planning

Leonardo Fischer, Luciana Nedel
Institute of Informatics
Federal University of Rio Grande do Sul – UFRGS
Porto Alegre, Brazil
{lgfischer, nedel}@inf.ufrgs.br

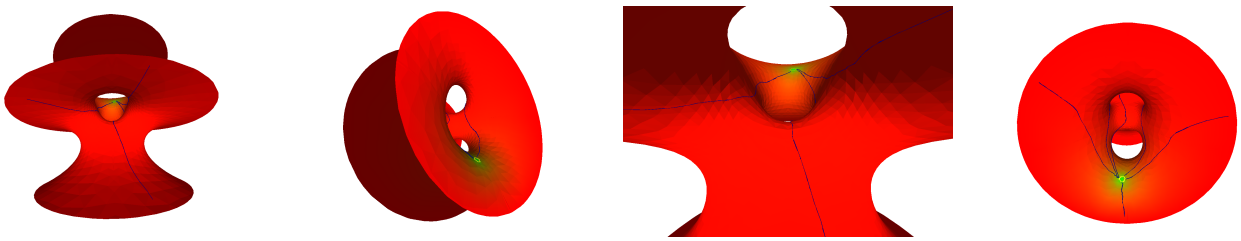


Fig. 1. Paths produced on the Costa Minimal Surface [1]. Note how lines starting in different points of the surface smoothly reach the target point (green).

***Abstract*—Efficient path-planning methods are being explored along the years to allow the movement of robots or virtual agents in planar environments. However, there is a lot of space to improve the quality of paths restricted to 3D surfaces, with holes and bends for instance. This work presents a new technique for path-planning on 3D surfaces called 3DS-BVP. This path planner is based on Boundary Value Problem (BVP), which generates potential fields whose gradient descent represents navigation routes from any point on the surface to a goal position. Resulting paths are smooth and free from local minima. The 3DS-BVP works on complex surfaces of arbitrary genus or curvature, represented by a triangle mesh, without the need of 2D parametrizations. Our results demonstrate that our technique can generate paths in arbitrary surfaces with similar quality as those generated by BVP-based methods in planar environments. Our approach can be applied in the development of new tools to automate the navigation on 3D surfaces, like the camera control in the exploratory visualization of 3D models.**

***Keywords*-3D path-planning; navigation; surfaces exploration.**

I. Introduction

Navigation on three-dimensional surfaces is a relevant problem for many application areas, such as: scientific visualization, where a user needs to inspect different objects, as organs in a medical application or engines in a CAD system; robotics, with the automatic definition of paths and motions for robots; and entertainment, more specifically on the video games domain, where the exploration of complex 3D worlds are much more challenging for the player than planar ones.

Navigation is a complex interactive task and is usually divided in two parts [2]: travel, and wayfinding. While the travel is the motor component of navigation, the low-level actions that a user makes to control the position and orientation of his/her viewpoint, the wayfinding is the cognitive component, and includes high-level thinking, planning, and decision making. It includes spatial understanding and planning tasks, such as determining a path from the current location to a goal location.

Path-planning algorithms are being explored for years. Several solutions were applied into robotics and virtual environments, with some of them focusing on a high performance path finding – normally including a pre-processing phase [3] –, and others on providing better paths. Although most of these solutions focus on the problem on the Euclidean plane, some of them are robust enough to handle systems with more than two degrees of freedom (as 3D path-planning [4] or path-planning for robotic arms with several joints [5]). However, path-planning methods restricted to arbitrary surfaces is not well explored in the literature.

Methods that focus specifically on 2D path-planning cannot be trivially modified to handle arbitrary surfaces. One possible approach is to use a sophisticated projection technique from the 3D surface to the Euclidean plane, and then modify the 2D path-planning algorithm to work on this projection, which is not a trivial task (due to the nature of this projection). Algorithms that handle 3D environments depend on their nature to be adapted to 3D surfaces, as in a given point of the surface the algorithm should behave as a 2D path planner. Graph based approaches are fast enough for real-time applications, but the generated paths are not as smooth as other approaches. In all these cases, the required work for porting is not negligible, and it is not clear how these algorithms will behave in this kind of environments.

In this work we developed a solution for the second part of the navigation problem on arbitrary surfaces, the wayfinding.

We present a new path-planning technique that handles the arbitrary 3D surface case, so called *3DS-BVP*, an acronym to *3D Surface Path Planner using Boundary Value Problems system*. The technique uses boundary value problem (BVP) systems to generate potential fields using a triangle mesh discretization. Using the gradient of the potential field, the agent can be guided through the environment. Briefly, the main contributions of this paper are:

- A numerical method that generates potential fields using a triangle mesh discretization;
- A path planner based on potential fields that draws smooth paths on 3D triangular surface meshes.

Our technique is based on a path-planning algorithm that is able to generate smooth paths with low probability of collision with obstacles, using potential fields.

The remainder of this papers is organized as follows. Section II presents the related work on path-planning for interactive applications and robotics. Sections III and IV describe the technique, while Section VI presents the results achieved and some discussion about it. Finally, in Section VII our conclusions and future works are discussed.

II. Related Work

Path-planning algorithms are being used to find a path to be followed from a given position to a goal one on a virtual environment. Many algorithms have been proposed to solve this problem, and the most part of them assume that the free space can be projected on a 2D surface.

Kallmann [6] used constrained Delaunay triangulations to discretize the free space of the environment in a triangle mesh, and a graph approach to search for free paths. Afterwards, he has also proposed a method that search for paths in an environment with specific clearance [7]. Despite the fact that these methods use triangle meshes as main data structure, they are developed only for planar environments. Our method do not make any difference between planar and 3D surfaces.

Techniques based on potential fields for navigation include the work of Rosell and Iniguez [8], Trevisan et al. [9], Treuille et al. [10], and Park [11]. These techniques use positions of obstacles and agents to compute a function. The result is a field from where the directions to a target position are derived. These techniques differ from each other in the function that is used to compute the field and how directions are derived, resulting in different behaviors for each technique. For example, the work of Trevisan et al. favors the exploratory behavior of an agent, while the work of Treuille et al. favors its use with crowds of autonomous agents. All these techniques were developed for 2D environments. Some of them can also be applied for 3D environments by adding one dimension to their equations, but this will significantly degrade its performance.

A path planner based on geodesic distances on triangular 3D meshes was recently proposed by Torchelsen et al. [12]. This work focuses on multi-agent systems, an uses a CPU/GPU architecture to handle the collision avoidance between the agents. The main advantage of this method is the high performance achieved. On the other side, the paths generated are close to the shortest ones, which can lead to a high probability of collision paths with static obstacles. Our potential field approach produces smooth paths that whenever is possible avoids getting very close to the obstacles.

Due to its performance and low memory requirements, graph-based approaches are the most common in the game industry. Popular game engines as Unreal Engine® and CryEngine® made use of it. In these methods, a graph represents the environment and the Dijkstra algorithm [13] (or one of its derivations) is used to find a path between two nodes. The difference between the approaches (as the ones proposed by Kavraki et al. [14], Barraquand et al. [15], Lavalle [16], and Kang et al. [17]) is the algorithm used to sample the graph from the environment and how it is updated. All these methods seem to be easily adaptable for 2D and 3D path-planning, but they are not being explored for arbitrary surface path-planning.

III. BVP Path Planner

The BVP Path Planner [9] is a 2D Path Planner that generates paths using the potential information computed from the numeric solution of

$$\nabla^2 p(\mathbf{r}) \;=\; \epsilon \mathbf{v} \cdot \nabla p(\mathbf{r}) \;, \tag{1}$$

with Dirichlet boundary conditions, where $\mathbf{v} \in \Re^2$ and $|\mathbf{v}| = \mathbf{1}$ corresponds to a vector that inserts a perturbation in the potential field; $\epsilon \in \Re$ corresponds to the intensity of the perturbation produced by $\mathbf{v}$; and $p(\mathbf{r})$ is the potential at position $\mathbf{r} \in \Re^2$, respectively. Both $\mathbf{v}$ and ϵ must be defined before computing this equation. The gradient descent on these potentials represents navigational routes from any point of the environment to the goal position. Trevisan et al. [9] shows that this equation does not produce local minima and generates smooth paths.

To solve numerically a BVP, we consider that the solution space is discretized in a regular grid ([9], [18]). Each cell (i, j) is associated to a squared region of the environment and stores a potential value $p(i, j)$. Using the Dirichlet boundary conditions, the cells associated to obstacles in the environment store a potential value of 1 (*high potential*) whereas cells containing the goal position store a potential value of 0 (*low potential*).

A high potential value prevents the agent from running into obstacles whereas a low value generates an attraction basin that pulls the agent. The relaxation methods employed to compute the potentials of free space cells is the Gauss-Seidel (GS). The GS method updates the potential of a cell c through:

$$p_c = \frac{p_b + p_t + p_r + p_l}{4} + \frac{\epsilon((p_r - p_l)v_x + (p_b - p_t)v_y)}{8} \tag{2}$$

where $\mathbf{v} = (v_x, v_y)$, and p_c, p_b, p_t, p_r and p_l are cells of a grid, as illustrated in Figure 2.

The GS method allows the use of partial results as an approximation of the potential field [19]. Since the exact solution is not necessary, we can control the accumulated error $e(t)$ at each iteration through a tolerance threshold e_{max}

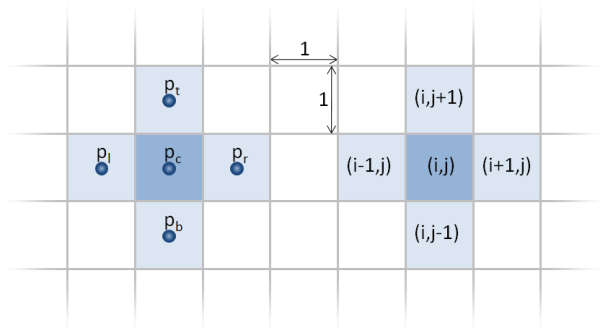


Fig. 2. Cells of a grid, when Equation 2 is evaluated for the cell p_c.

according to:

$$e(t) = \sum_{i=1}^{m} \sum_{j=1}^{n} |p(i,j)^t - p(i,j)^{t-1}| \geq e_{max}. \quad (3)$$

where $p(i,j)^t$ is the potential of the cell (i,j) at the iteration t, $p(i,j)^{t-1}$ is the potential of the same cell in the previous iteration and m and n are the grid dimensions.

After the potential computation, the agent moves following the direction of the gradient descent of this potential at its current position (i,j).

IV. Path-Planning on Triangle Meshes

In our proposal, the 3DS-BVP generates potential fields that produce smooth paths in triangle meshes. The method works accordingly to the steps: (1) discretize the environment in a set of cells; (2) calculate the potential field; (3) compute the gradient of the potential field.

A. Discretizing the environment

Triangle meshes are widely used in computer graphics industry to model objects and virtual environments, what means that our technique can be easily applied to existing 3D models and surfaces. Then, we assume that the environment is represented by a triangle mesh that is used as a triangular grid to compute the potential field.

The triangular grid is represented by a *Doubly-Connected Edge List* (*DCEL*) data structure [20] due to its simplicity, capability to find the neighbors of a vertex, face or edge in constant time, and ability to handle closed and open triangle meshes. For the scope of this work, an *open mesh* is a mesh of triangles that is topologically equivalent to a 2D plane. Meshes with *holes* in its surface are also considered *open meshes*. A *closed mesh* is *not* topologically equivalent to a plane, in such a way that it should be *broken* into two or more open meshes to build a mapping function from the mesh to a plane. *Closed meshes* also don't have holes in its surface.

As shown in Figure 3, an edge in the DCEL connects two vertices and is actually defined by a pair of half-edges, each one being a *twin* of the other. A face is then defined by a sequence of half-edges starting at the *boundary* half-edge of the face, usually in a counter-clockwise order. The half-edges

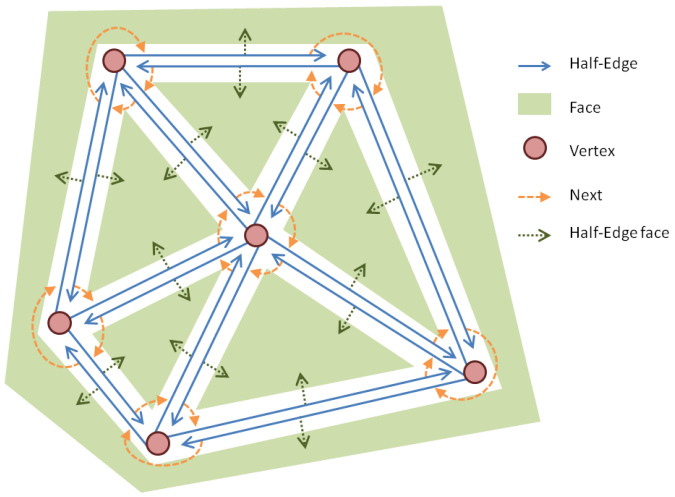


Fig. 3. DCEL data structure. Edges are represented by pairs of parallel blue arrows, vertices by red circles, and faces by green polygons.

are connected among themselves through the *next* and *previous* pointers.

The first step of the 3DS-BVP algorithm works as follows. To each vertex of the *DCEL* data structure, a potential value is assigned – in the same way of the BVP path planner cell in Section III. Then, each triangle of the grid is divided into three regions by connecting the medians of the edges of the triangle to its centroid (see Figure 4). Each triangle region is associated to the near vertex and, for a given vertex v of a triangle t, the function $Region(v,t)$ returns that associated region. Also, the algorithm assumes that each vertex v in the triangle mesh is associated with a set of triangles $Triangles(v)$, where each triangle in $Triangles(v)$ has one vertex equals to v (i.e. they share the same vertex). The cell associated to a given vertex v is then defined by function $Cell(v) = \{Region(v,t) | \forall t \in Triangles(v)\}$. The function $Vertex(c)$ returns the vertex associated to the cell c. Figure 5 illustrates these concepts.

We also define the concept of *neighbor cell* as two cells that have their associated vertices connected through a single pair of half-edges. This means that two cells c_1 and c_2 are neighbors if the function $Link(Vertex(c_1), Vertex(c_2))! = null$ satisfies. Assuming that the set C contains all the cells derived from the triangle mesh, the function $Neighbors(c_i) = \{c_j | \forall c_j \in C, i! = j, Link(c_i, c_j)! = null\}$ returns the set of neighbor cells for a given cell c_i.

B. Calculating the potential field

In order to execute the relaxation over the set of cells, the boundary conditions must be defined. These boundary conditions are set according to the positions of obstacles and the goal in the environment. We assume that obstacles and goal positions are constrained to the surface. As in the BVP Path Planner, each cell receives a tag and an initial potential value, as follows:

- cells associated with occupied areas of the environment and cells associated with *limiting vertices* (vertices in the

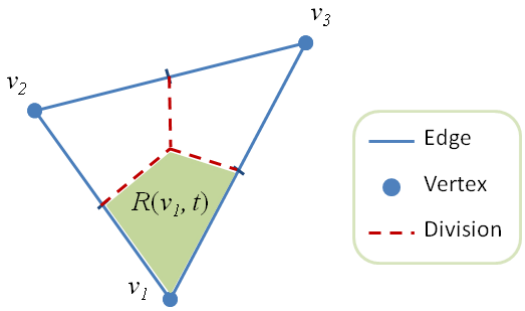


Fig. 4. Triangles are divided in order to build the cell mesh. In highlight, the area returned by the function $Region(v_1, t)$.

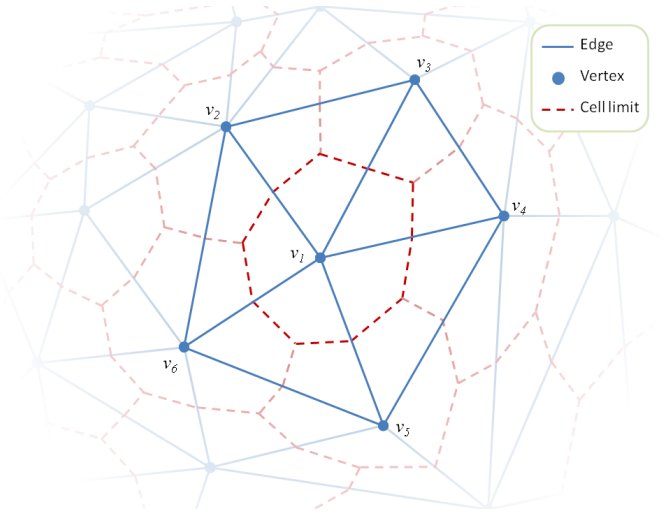


Fig. 5. A triangle mesh and its corresponding cell division. The five highlighted triangles ($\{v_1, v_2, v_3\}$, $\{v_1, v_3, v_4\}$, $\{v_1, v_4, v_5\}$, $\{v_1, v_5, v_6\}$ and $\{v_1, v_2, v_6\}$) share the same vertex v_1 and, therefore, they are in the set $Triangles(v_1)$. Applying the function $Region(v_1, t)$ for each triangle t in $Triangles(v_1)$ results in the cell associated to the vertex v_1, which is highlighted with the dotted line around v_1.

border of the mesh or in the limit of a hole in the mesh) are tagged as *occupied* and receive the *high potential* value;

- the cell associated with the goal position receives the *goal* tag and the *low potential* value;
- cells associated with *free* navigable areas of the environment are tagged as *free* and receive a mean value between the *low* and *high potential* values, as in the BVP Path Planner.

In order to update the potential value of the *free* cells, a set of functions need to be defined. Assuming that the function $Dist(v_1, v_2)$ returns the Euclidean distance between the vertices v_1 and v_2, $D_{min}(c_i)$ and $D_{max}(c_i)$ return the minimum and maximum distance, respectively, between the vertex of a cell c_i and the vertices in its neighborhood.

The influence of a cell c_j over the cell c_i, relative to the neighborhood of c_i is given by

$$I(c_i, c_j) = D_{min}(c_i) + D_{max}(c_i) - Dist(Vertex(c_i), Vertex(c_j)) \quad (4)$$

Equation 4 is an heuristical measure of how much the potential of one cell c_j has over its neighbor c_i. As $D_{min}(c_i)$ and $D_{max}(c_i)$ are computed from the same input set, it is guaranteed that $D_{min}(c_i) <= D_{max}(c_i)$. The closer the cell c_j is to c_i, closer the value of $I(c_i, c_j)$ is to D_{max}. The farther a cell c_j is from the cell c_i, closer the value of $I(c_i, c_j)$ is to D_{min}. Then, the result of $I(c_i, c_j)$ can be interpreted as how close a cell c_j is to c_i, in a scale between $D_{min}(c_i)$ and $D_{max}(c_i)$.

Based on Equation 4, the function

$$I_{total}(c_i) = \sum_{j=1}^{\#Neighbors(c_i)} I(c_i, c_j) \quad (5)$$

computes the sum of the influences that a cell receives from its neighbors. This function is then used in the equation

$$p(c_i) = \sum_{j=1}^{\#Neighbors(c_i)} p(c_j) \frac{I(c_i, c_j)}{I_{total}(c_i)}, \quad (6)$$

which is used to update the potential $p(c_i)$ of a *free* cell c_i.

The potential values of the *free* cells are updated using Equation 6 until the convergence sets in, as in the BVP Path Planner. A threshold error e_{max} is used to verify if the potential field has converged. Equation 3 is used over the whole set of cells C to compute the error at a given iteration.

C. Computing the gradient of the potential field

Calculating the potential field, an agent should be able to follow the gradient descent in order to reach the goal position. Equation 6 is able to mimics the results produced by the Laplace's Equation. We can then calculate the gradient of Equation 6, as we use to do with the Laplace's Equation.

One possible solution for the calculus of the gradient of the Laplace's Equation in an unstructured triangular mesh involves the use of the integral form of that equation, which can be obtained by integrating the equation over some volume Ω, and then applying the Gauss Divergence Theorem. The integral form of the Laplace's Equation and its equivalent after applying the Gauss Divergence Theorem is

$$\int_{\Omega} \nabla^2 p(r) d\Omega = \oint_{\partial\Omega} \nabla p(r) \cdot \hat{n} dA = 0. \quad (7)$$

The relation between the gradient of the function $p(r)$ over a volume and the integral on the surface of the volume of the function times the normal area vector is

$$\int_{\Omega} \nabla p(r) d\Omega = \oint_{\partial\Omega} p(r) \hat{n} dA. \quad (8)$$

The relation above, when computed in sufficiently small volumes can be used to compute an approximation of the gradient. This results in

$$\nabla p(r) \approx \frac{1}{\Omega} \oint_{\partial\Omega} p(r)\hat{n}dA. \quad (9)$$

Assume that we want to compute the approximation of the gradient in the triangle $t = \{v_0, v_1, v_2\}$. For this, we calculate the normal vector of each edge of each triangle and multiply each one by the length of the respective edge. The area of the triangle t is calculated by the function $Area(t)$. Function $p(r)$ is already computed in each vertex.

The approximation of Equation 9 on the triangle t yelds the equation

$$\nabla p(t) \approx \frac{1}{2Area(t)} \left[-p(v_0)\hat{n}_1 - p(v_1)\hat{n}_2 + p(v_2)\hat{n}_0 \right]. \quad (10)$$

that is then used to approximate the gradient of the potential field at the centroid of the triangle t.

To move towards the target position in the environment, an agent a must follow the gradient descent of the triangle where he is on. This triangle is kept as a pointer to the current face in the DCEL strucutre. When the agent walks out of the current face, the functions provided by the DCEL data structure can be used to check which one of the edges was crossed. The crossed edge will be represented by a half-edge h, and the function $Face(Twin(h))$ will be set as the current face of the agent. The function $UpdatePosition(a, g)$ is used to update the agent position and the current face based on the gradient g.

V. Potential Fields in 3D Surfaces

In a completely planar environment discretization, the path planner presented in Section IV behaves in a similar way to the BVP Path Planner (see Section III). The goal position must be checked with a *goal* tag while the obstacles are tagged as *occupied*. The limits of the environment are also tagged as *occupied* and all the remaining cells are tagged as *free*. All cells receive some potential value, according to what was specified in the Sub-section IV-B. Then, the relaxation step evaluates adequate values for *free* cells. And finally, the agent can use the gradient of the potential field to build a path to follow.

In this kind of environment, the path planner can take advantage of a simpler math, since all the vertex positions and normals can be manipulated in 2D. Naturally, a 3D discretization will require a third coordinate for the position of vertices and normals.

For 3D open meshes the path planner works with minimal modification, and the addition of a coordinate axis does not introduce significant changes in any of the methods presented in the previous section. During the relaxation, only the function $Dist(v_1, v_2)$ changes and must compute the Euclidean distance in 3D. In addition, the normals used to compute the gradient must also be stored in 3D. But the normals keep parallel to the plane defined by the triangle which they belong.

The most relevant modification in the algorithm to deal with 3D meshes is the handling of boundary conditions. Our algorithm requires that at least two different boundary conditions must be set: one *goal* cell and one *occupied* cell. We assume that the *goal* is always defined, so for 3D meshes we should take a special care with the *occupied* cells.

If the 3D mesh is *open* (as defined in Section IV-A), the *limiting vertices* will force the existence of *occupied* cells, as explained in Section IV-B. *Closed meshes* with obstacles on its surface will also force the existence of *occupied* cells. This will generate a gradient that prevents the agent from leaving the surface limits, collides with obstacles, and guides to the goal position.

However, if the mesh is *closed* and do not have any obstacle on its surface, then our algorithm will generate only *goal* cells. If this occurs, the relaxation step will stop only when all cells have their potential value equal to *low potential*. This will lead to a null gradient on the mesh (the result of Equation 10 will be $(0, 0)$), and the agent will have no clue about which direction it should follow.

To avoid the occurrence of null gradients, the cell containing the initial position of the agent is tagged as *occupied* and receives the *high potential* value. Although the cell does not contain any obstacle, this will force the potential field to have a gradient capable of guiding the agent from its initial position to the goal. Also, as the agent is leaving that position away, the obstacle added will not modify the existence of a path from the initial to the goal position.

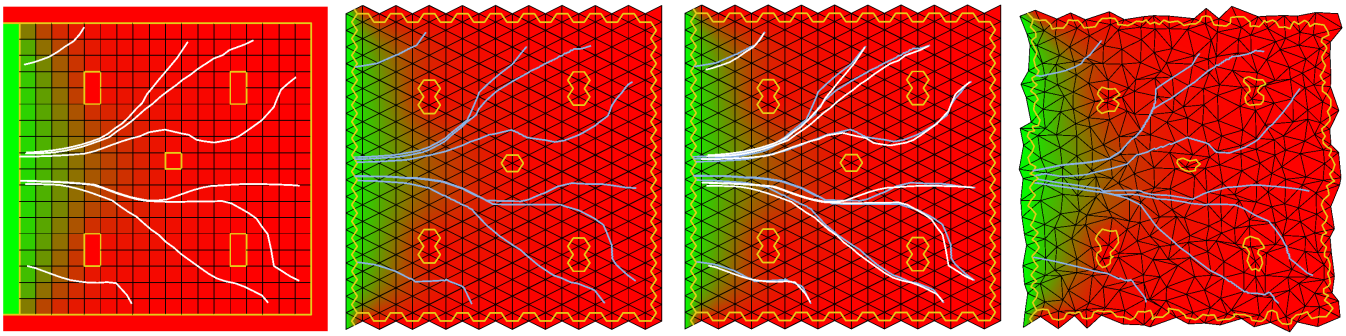
Meanwhile, this approach has a drawback. The path taken by the agent may change significantly according to the initial position of the agent inside the modified cell. Because the cell that corresponds to the initial position is modified to receive the *occupied* tag and *high potential* value, the gradient on the cell borders will be significantly different according to the triangle where the position is. The fact that the mesh is closed and do not have other boundary conditions allows the situation where many different paths can be taken to reach the same goal position.

VI. Results

In order to evaluate our work, we produced a set of tests. First, we present a comparison of our method in a planar triangular mesh, showing that our technique is able to produce smooth paths with low collision probability, in the same way that the BVP Path Planner does. Then, we present some of the results that we obtained with 3D arbitrary surfaces.

A. Comparing our method with the BVP Path Planner

We designed a set of test cases, where each test case is composed of a squared environment with some obstacles and goals in it. In the test cases 1 up to 4 the environment were discretized in a regular triangle grid. The test cases 5 up to 8 used the same environments as the first cases, but with addiction of noise to disturb each vertex position in the triangle



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

Fig. 6. Planar environment. The gradual color transition from green to red represents the value of the potential in each point of the environment, from 0 to 1, respectively. (a) The environment discretized in a regular grid, with the paths computed with the BVP Path Planner; (b) the environment discretized in a regular triangle mesh, with paths computed with the 3DS-BVP; (c) comparison between the paths generated with the BVP Path Planner and the 3DS-BVP; (d) a discretization of the environment using a triangle mesh with noise.

Test Case	Average Error	Std. Deviation	Max. Error	#Cells
1	0.00394	0.00606	0.06224	2,500
2	0.00303	0.00417	0.05802	10,000
3	0.01844	0.01283	0.08957	10,000
4	0.00604	0.00817	0.08911	10,000
5	0.00903	0.01042	0.15687	2,500
6	0.00469	0.00631	0.30755	10,000
7	0.01867	0.01717	0.49989	10,000
8	0.00792	0.01332	0.49913	10,000
Average	0.00897	0.00981	0.22030	

TABLE I
COMPARISON BETWEEN THE POTENTIAL FIELD GENERATED WITH OUR TECHNIQUE AND THE BVP PATH PLANNER. AS THE POTENTIAL FIELDS GENERATED WAS NORMALIZED, THE AVERAGE ERROR IN ALL CASES WAS ABOUT 0.8%, WITH A STANDARD DEVIATION OF ABOUT 0.9% OF THE RESULT PRODUCED BY THE BVP PATH PLANNER.

grid (better representing the triangles found in 3D surfaces). Figure 6 illustrates these test cases. The green region on the border represents the goal and the yellow lines delimit the obstacles. The white lines are the path generated by the BVP Path Planner, starting at random positions on the environment, while the blue lines are the path generated by our technique. Figure 6(a) illustrates our test cases using the regular grid, while Figure 6(b) and (d) used a triangle grid. Figure 6(c) combines the paths produced by the BVP Path Planner and the 3DS-BVP in a single image.

We can see that the paths produced by the 3DS-BVP in Figure 6 are quite similar to the paths produced by the BVP Path Planner. By adding noise, these paths loses some smoothness due to the presence of low quality triangles, but still mimics the results produced by the BVP Path Planner and by 3DS-BVP on regular triangle grid.

We have also compared the potential value in each cell of the potential field produced by the 3DS-BVP with the potential field generated by the BVP Path Planner. Table I presents a summary of the differences found between these values. We can see that, in average, the potential field generated by our method is almost the same potential field produced by the BVP Path Planner, with a difference of only 0.8% on average.

The highest difference was 49.989%, found in the test case 7. Although this is a considerable difference between the 3DS-BVP and the BVP Path Planner, it occurred in an environment that had an average difference of 1.867% and standard deviation of 1.717%. This difference occurred due to the existence of highly deformed triangles resulting from the noise. The test case 8 produced very similar potential values, and generated the paths illustrated in Figure 6(d).

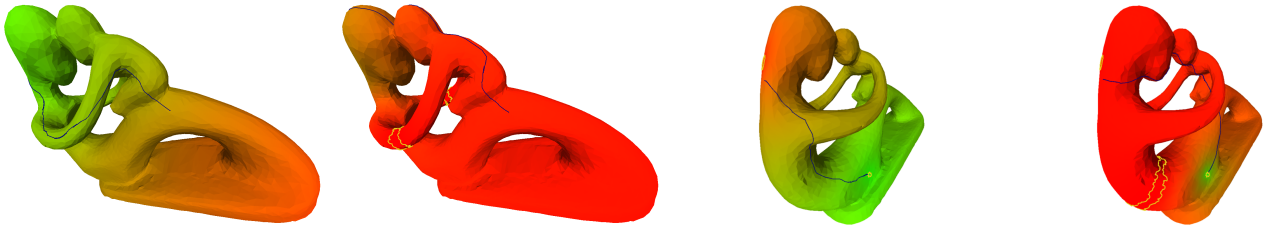
B. Path-planning evaluation in arbitrary meshes

We applied our algorithm on some 3D models to analyze how paths are generated on these surfaces. In Figure 1 we applied our algorithm on the Costa Minimal Surface [1], a complete minimal embedded surface with a genus with three punctures. We generated several paths on this surface, from several distinct initial positions to a predefined goal position. In all cases within this surface, the algorithm found a smooth path to reach the goal position.

In another experiment, we used a model of the *Fertility* statue (Figure 7) to generate paths on its surface. The *Fertility* has several genus, which also makes it a good example of the kind of environment that our technique deals with. In Figures 7(a) and (c), our planner has generated quality and smooth paths to reach the goal position. In Figures 7(b) and (d) we used the same initial and goal positions as (a) and (c), respectively, but we also defined some regions where the path could not crossover, simulating obstacles on the surface. The algorithm demonstrated to be able to find quality and smooth paths.

C. Performance evaluation

We measured the performance of our algorithm in several cases, including the surfaces presented in Section VI-B. We measured the time spent to compute the potential field, and the number of iterations needed to the convergence with a threshold error $e_{max} = 0.001$. Results are presented in Table II. The tests were executed in a Intel® Core i7 870, 2.93GHz, 4GB Ram and NVidia ® GeForce GTX 470.



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

Fig. 7. Several paths produced on a complex model (*Fertility* mesh), with several genus. Note that in (b) the initial and goal positions are the same as (a), but some obstacles resulted in a different path. The same occurs in (c) and (d).

Model	Faces	Vertices/Cells	Time (s)	Iterations
Car	1,292	665	0.2953	1,006
Costa	2,320	1,259	0.3948	854
Statue	10,000	4,994	41.2660	19,434

TABLE II
PERFORMANCE EVALUATION ON THREE TEST CASES.

Although the quality and smoothness of paths generated in the previous tests, our performance evaluation shows that there is still room for improvement. For smaller but complex models, like the Costa Minimal Surface, our algorithm is able to solve the potential field and produce a quality path for applications, like the motion of cameras in virtual environments. For more detailed models, like the Fertility, our algorithm still needs some improvements in its performance.

D. Limitations and degenerated cases

Analysing Equation 4 one can conclude that if, eventually, two vertices are in the same position, $D_{min}(c_i)$ will be equal to 0, and any vertex with a distance of $D_{max}(c_i)$ will have no influence over the vertex c_i. This may result in an interruption of the propagation of the potential from one region to another. Also, it is clear that if the mesh has an invalid triangle (with collinear vertices, for instance), the result of the Equation 10 is undefined.

We tested our method with many different meshes. In an experiment with several very long non-equilateral triangles, the paths lost its smoothness. This happens because the gradient of the adjacent triangles presented big differences between them.

VII. Conclusions and Future Work

We presented the 3DS-BVP, a technique based on potential fields for path-planning in arbitrary surfaces. As the main advantage of our technique, it generates smooth paths free from local minima on 3D surfaces, without the need of a 2D parametrization, or some other surface representation.

The potential field produced by the 3DS-BVP generates all the possible paths from a point in the environment to a goal position following the descent gradient. Paths generated on 3D surfaces shows to be quite similar to the quality of BVP Path Planner, with a potential field with a difference of only 0.8% to it on average. This is a good feature, because the 3DS-BVP uses as core a potential field with similar characteristics of the ones produced by the BVP Path Planner.

As its main drawback, its performance is not sufficiently good for real-time applications, as environments with several moving agents. We believe that this drawback can be minimized by improving our solution method and by developing GPU based methods to compute the potential field.

Some possibilities to improve the algorithm performance are being analyzed. One possible way is to improve the set of equations that we used, in order to obtain the potential field using less iterations. Also, the order that cells are evaluated reflects in the speed that the potential values from the obstacles and goals are propagated to the free cells. So, there should be an ideal order that makes the relaxation process faster. Another possible performance optimization is to implement our algorithm using GPU. Our method appears to been highly parallelizable, as the one shown in a previous work [21].

We intend to apply this algorithm mainly in virtual environments. New tools to control the virtual camera in CAD and modeling applications can be developed, in order to help the evaluation and visualization of 3D models. In video games like Prey® and Super Mario Galaxy® the player and its enemies walk on arbitrary surfaces to reach their objectives. Future video games using this kind of environment can also have benefits from our algorithm.

Acknowledgment

The authors would like to thank Renato Silveira for his ideas and help with the text, as well as CNPq-Brazil through projects 483947/2010-5, 580156/2008-7, 309092/2008-6 and Microsoft Brazil Interop Labs. for partially supporting this work.

References

[1] C. J. Costa, "Example of a complete minimal immersion in IR3 of genus one and three-embedded ends," *Bulletin of the Brazilian Mathematical Society*, vol. 15, no. 1-2, pp. 47–54, Mar. 1984.

[2] D. A. Bowman, E. Kruijff, J. J. LaViola, and I. Poupyrev, *3D User Interfaces: Theory and Practice*. Redwood City, CA, USA: Addison Wesley Longman Publishing Co., Inc., 2004.

[3] A. Calomeni and W. Celes, "Assisted and automatic navigation in black oil reservoir models based on probabilistic roadmaps," in *Proceedings of the 2006 symposium on Interactive 3D graphics and games - SI3D '06*, ser. I3D '06. New York, New York, USA: ACM Press, 2006, pp. 175–182.

[4] J. Carsten, D. Ferguson, and A. Stentz, "3D Field D: Improved Path Planning and Replanning in Three Dimensions," in *Proceedings of the 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '06)*, 2006, pp. 3381–3386.

[5] K. Belghith, F. Kabanza, L. Hartman, and R. Nkambou, "Anytime Dynamic Path-planning with Flexible Probabilistic Roadmaps." in *ICRA'06*, 2006, pp. 2372–2377.

[6] M. Kallmann, "Path Planning in Triangulations," in *Proceedings of the IJCAI Workshop on Reasoning, Representation, and Learning in Computer Games*, Edinburgh, Scotland, 2005.

[7] ——, "Shortest paths with arbitrary clearance from navigation meshes," in *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. Madrid, Spain: Eurographics Association, 2010, pp. 159–168.

[8] J. Rosell and P. Iniguez, "Path planning using Harmonic Functions and Probabilistic Cell Decomposition," in *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*. IEEE, 2005, pp. 1803–1808.

[9] M. Trevisan, M. A. P. Idiart, E. Prestes, and P. M. Engel, "Exploratory Navigation Based on Dynamical Boundary Value Problems," *Journal of Intelligent and Robotic Systems*, vol. 45, no. 2, pp. 101–114, May 2006.

[10] A. Treuille, S. Cooper, and Z. Popović, "Continuum crowds," *ACM SIGGRAPH 2006 Papers*, vol. 25, no. 3, pp. 1160–1168, Jul. 2006.

[11] M. J. Park, "Guiding flows for controlling crowds," *The Visual Computer*, vol. 26, no. 11, pp. 1383–1391, Jan. 2010.

[12] R. P. Torchelsen, L. F. Scheidegger, G. N. Oliveira, R. Bastos, and J. a. L. D. Comba, "Real-time multi-agent path planning on arbitrary surfaces," in *I3D '10: Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*. Washington, D.C.: ACM, 2010, pp. 47–54.

[13] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, Dec. 1959.

[14] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[15] J. Barraquand, L. Kavraki, J.-C. Latombe, R. Motwani, T.-Y. Li, and P. Raghavan, "A Random Sampling Scheme for Path Planning," *The International Journal of Robotics Research*, vol. 16, no. 6, pp. 759–774, 1997.

[16] S. M. Lavalle, "Rapidly-Exploring Random Trees: A New Tool for Path Planning," 1998. [Online]. Available: http://msl.cs.uiuc.edu/~lavalle/papers/Lav98c.pdf

[17] S.-J. Kang, Y. Kim, and C.-H. Kim, "Live path: adaptive agent navigation in the interactive virtual world," *The Visual Computer*, vol. 26, no. 6, pp. 467–476, Apr. 2010.

[18] R. Silveira, F. Dapper, E. Prestes, and L. Nedel, "Natural steering behaviors for virtual pedestrians," *The Visual Computer*, vol. 26, no. 9, pp. 1183–1199, Nov. 2009.

[19] E. Prestes, M. A. Idiart, P. M. Engel, and M. Trevisan, "Exploration technique using potential field calculated from relaxation methods," *Intelligent Robots and Systems*, vol. 4, pp. 2012–2017, 2001.

[20] M. D. Berg, O. Cheong, M. V. Kreveld, and M. Overmars, *Computational Geometry: Algorithms and Applications*, 3rd ed. Springer-Verlag, 2008.

[21] L. G. Fischer, R. Silveira, and L. Nedel, "Gpu accelerated path-planning for multi-agents in virtual environments," in *2009 VIII Brazilian Symposium on Games and Digital Entertainment*. IEEE, 2009, pp. 101–110.