Correspondence

A Fast Path Planning by Path Graph Optimization

Joo Young Hwang, Jun Song Kim, Sang Seok Lim, and Kyu Ho Park

***Abstract*—In this paper, a fast path planning method by optimization of a path graph for both efficiency and accuracy is proposed. A conventional quadtree-based path planning approach is simple, robust, and efficient. However, it has two limitations. The first limitation is that many small cells are required to represent obstacles because the positions and shapes of the cells are not object-dependent and searching the shortest path in the path graph is slow. The second limitation is generation of nonoptimal paths due to large cells in a free space. The cost of traversing a cell is the same whether a path just clips a corner of the cell or actually passes through the entire width of the cell. The quadtree is very sensitive to obstacle placement and the error factor is proportional to the size of the cell.**

We propose a path graph optimization technique employing a compact mesh representation. A world space is triangulated into a base mesh and the base mesh is simplified to a compact mesh. The compact mesh representation is object-dependent; the positions of vertexes of the mesh are optimized according to the curvatures of the obstacles. The compact mesh represents the obstacles as accurately as the quadtree even though using much fewer vertexes than the quadtree. The compact mesh distributes vertexes in a free space in a balanced way by ensuring that the lengths of edges are below an edge length threshold. An optimized path graph is extracted from the compact mesh. An iterative vertex pushing method is proposed to include important obstacle boundary edges in the path graph. Dijkstra's shortest path searching algorithm is used to search the shortest path in the path graph.

Experimental results show that the path planning using the optimized path graph is an order of magnitude faster than the quadtree approach while the length of the path generated by the proposed method is almost the same as that of the path generated by the quadtree.

***Index Terms*—Mesh simplification, path planning.**

I. Introduction

Motion planning in the presence of obstacles is an important problem in robotics with applications in other areas such as simulation and computer aided design [1]. This problem is of particular interest to many fields of research, including VLSI design, global positioning systems (GPS) applications, and autonomous robot navigation. In recent years, a variety of approaches have been used to solve the Euclidean distance shortest path planning problem.

Skeleton, cell decomposition, and potential field approaches have been proposed for path planning. A recent survey of relevant literature can be found in [2]. In the skeleton approach, a free space, i.e., the set of feasible motions, is retracted, reduced to, or mapped onto a network of one-dimensional lines. This approach is also called the roadmap, or highway approach. The widely used skeletons are the visibility graph, Voronoi diagram, subgoal network, and the silhouette. Those approaches normally require extremely accurate sensor information to setup an initial graph and may suffer from a large convergence time as well [3]–[6]. The potential field approach [7], based on following the gradient of potential field lines, may fall into local minima which leads to nonoptimal solutions, or traps a robot. In cell decomposition approaches, the free space is partitioned into a set of simple cells, and the adjacency relationships among the cells are computed. The cells can be object-dependent or object-independent. In the object-dependent decomposition method, obstacle boundaries are used to generate the cell boundaries, and the union of free cells exactly defines the free space. The number of cells to represent the space is small, but the complexity of decomposition is high and the computations of containment, intersection, connectivity, and the adjacencies of the cells are very difficult. In comparison, object-independent decomposition methods such as the grid method and quadtree method are efficient in practical applications. However, since the shape and location of a cell are determined independently of the shape and placement of obstacles, the cells cannot tightly represent the boundaries of obstacles, resulting in many small cells near the obstacle boundaries.

The grid method guarantees finding the shortest path which is exact up to the resolution of the grid. However, the grid method has many small cells, so that the path planning stage is not efficient [8]–[11]. The quadtree method for a point robot is presented in [12]. A quadtree is generated by a recursive decomposition of a two-dimensional (2-D) environment into uniformly sized regions. It reduces the number of small cells in a free space.

In this paper, we focus on solving the two limitations of the quadtree method. The first limitation is that the quadtree representation requires very high resolution to model the obstacle surfaces accurately. Since many narrow passages in obstacle-free regions may disappear in a quadtree constructed from an input grid of low resolution, quadtree-based approaches require a very high resolution input grid, hence, resulting in a very complex path graph.

The number of cells to search in the quadtree can be reduced by using a multi-resolution representation. A cell in a free space is called "white" and a cell in an obstacle space is called "black." A cell which contains the mixture of white and black cells is called a "gray" cell. When a gray cell is selected as a part of an optimal path, it is refined into black and white cells. Using this adaptive traversing of the quadtree, the path planning can be accelerated. However, the multiresolution scheme cannot reduce the path planning time if the robot is going along the narrow passages near the obstacles. Another solution is to use object-dependent cell decomposition. However, it requires an input grid of much higher resolution than the quadtree and much larger computing time because the object-dependent cell decomposition algorithm involves complex computation.

The second limitation is that the quadtree method cannot guarantee finding the shortest path as mentioned in [12]–[17]. Since the position of a path graph node is always the center of the corresponding cell, the cost of traversing the cell is the same whether the path just clips a corner of the cell or actually passes through the entire width of the cell. Hence, the quadtree is very sensitive to obstacle placement and the error factor is proportional to the size of the cell. For example, Fig. 1(a) describes a situation which plans a path starting from S to G using the quadtree. Using Dijkstra's shortest path algorithm, path A is found while path B, which is the desired shortest path, cannot be found. Since the paths generated using the quadtree should pass the center of each cell, path C, which passes the region with large cells, has large cost.

This drawback of the quadtree method was addressed in [18] by introducing a new data structure, which is called a framed-quadtree. The border of a large cell in a free space is enclosed by an array of

Manuscript received April 4, 2001; revised January 23, 2003. This paper was recommended by Associate Editor W. A. Gruver.

The authors are with the Korea Advanced Institute of Science and Technology, Taejon, 305-701, Korea (e-mail: jyhwang@core.kaist.ac.kr; jskim@core.kaist.ac.kr; sslim@core.kaist.ac.kr; kpark@core.kaist.ac.kr).

Digital Object Identifier 10.1109/TSMCA.2003.812599

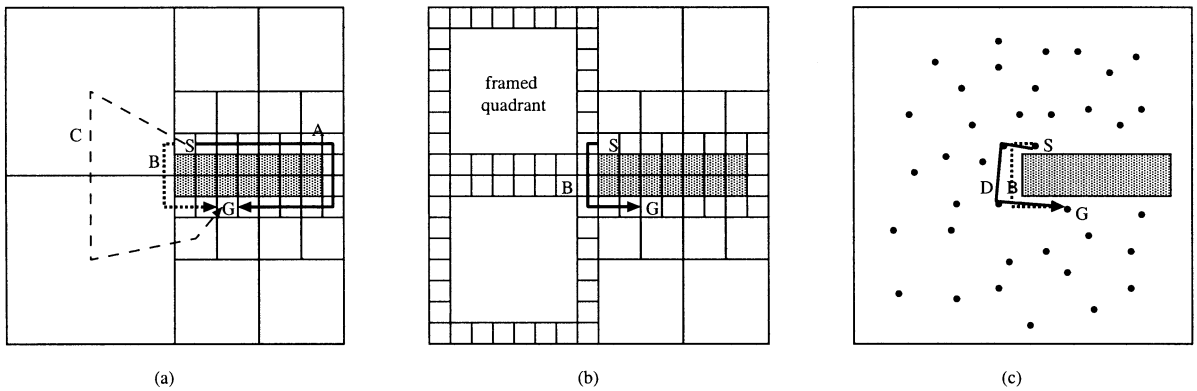


Fig. 1. (a) Quadtree. (b) Framed-quadtree algorithm. (c) Randomized roadmap. Conventional quadtree-based approaches cannot guarantee finding the shortest paths. Using Dijkstra's shortest path algorithm, the path C has larger cost than the path A, so the path A is found. The path B is the desired shortest path, which cannot be found in quadtree-based approach. The framed-quadtree method can find the path B by constructing a border cell array on the large cell. The randomized roadmap distributes nodes randomly in a free space in a balanced way.

square cells (called a "frame") which are of the smallest possible size allowed for a particular environment. In order to generate Euclidean shortest paths, they propagate a circular path planning wave through the workspace. This wave is used to compute an L2 distance transform for the border cells of obstacle-free framed-quadrants in a generalized breadth-first manner. The paths generated are optimal within the representation of the environment, which is better than both the uniform grid and the quadtree-based approaches. The randomized roadmap method distributes nodes in a free space in a balanced way and connects them using a local path planner. There is a trade-off: as the density of the nodes in a free space increases, the error factor is reduced while the path planning stage becomes slower.

We propose a path graph optimization approach to solve the two limitations of the quadtree method. We use a triangular cell as a basic primitive instead of a regular square cell of the quadtree. Mesh is a triangular decomposition of a world space, which is called triangulation. Initially the world space is triangulated from a uniform grid into a base mesh. Then the base mesh is simplified to a compact mesh. The shapes of the obstacles are approximated very tightly using much fewer triangles than the base mesh; hence, the obstacle boundaries are preserved in the compact mesh even after significant simplification. Additionally, the compact mesh has randomly distributed nodes in a free space to prevent overly long edges which may lead path planning to nonoptimal solutions. Thus, our approach is conceptually similar to that of the randomized roadmap's solution to the second limitation of the quadtree-based approach.

In the next step, a path graph is extracted from the compact mesh. The path graph consists of a subset of the vertexes and edges of the compact mesh which lie in a free space. The path graph is optimized for both efficiency and accuracy of path planning. The path planning stage is much faster than the quadtree-based approaches because the complexity of the optimized path graph is much lower than that of the quadtree. Moreover, the path graph can represent the world space with high fidelity using triangular cell primitives, and has distributed nodes in a free space in a balanced way to guarantee finding the shortest paths. Our experimental results show that the path planning using the optimized path graph is an order of magnitude faster than the conventional quadtree-based approach guaranteeing finding shortest paths.

Compared with quadtree and framed-quadtrees, which are based on regular square cells, our method can represent the obstacle boundaries with much fewer primitives. Even though much fewer primitives are used, the optimized path graph represents the world space as well as the quadtree does. Moreover, the nodes in a free space are distributed in a balanced way. Hence, path planning using the optimized path graph can find the shortest path much faster than those methods based on the quadtree representation. Compared with object-dependent cell methods, our method does not require an input grid of high resolution, nor does it require expensive computations such as containment, intersection, connectivity, and adjacency computations.

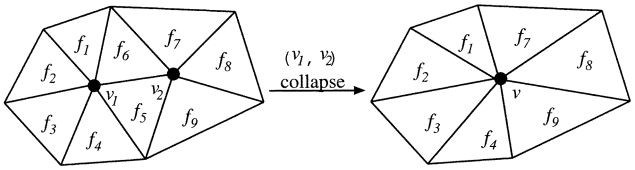


Fig. 2. Edge collapse operation simplifies the mesh.

To present the optimization of the path graph more effectively, we focus on the path planning of a point robot in a 2-D known environment in this paper. Our approach is easily extensible to three-dimensional (3-D), or higher dimensions and unknown environments, which will be discussed in Section V.

The rest of this paper is organized as follows. First, we introduce the mesh simplification algorithm in Section II. The path planning based on the path graph optimization is described in Section III. We present experimental results in Section IV. Finally, we conclude and discuss future works in Section V.

II. Mesh Simplification Backgrounds

Mesh simplification is a geometric optimization problem. A mesh is represented as $\langle V, E, F \rangle$ where V is the set of vertices, E is the set of edges, and F is the set of triangles. The mesh simplification is to simplify a highly detailed 3-D mesh into a compact mesh while minimizing the geometric error between the original mesh and the simplified mesh.

The problem has been one of the hottest issues of computational geometry and computer graphics where the simplified mesh is used for fast rendering. A recent survey is done by Heckbert [19]. The optimal mesh simplification problem is known to be NP-hard [20]. Global optimization is too computation intensive for the mesh simplification. Many heuristic algorithms based on local optimization are proposed [21]–[25].

In the heuristic mesh simplification algorithms, a mesh is simplified using an edge collapse operation as shown in Fig. 2, where an edge (v_1, v_2) is collapsed to a target vertex v. The neighbor triangles of the edge, f_5 and f_6, are removed and the edge is collapsed to a target vertex v. The edge collapse operation is iteratively performed as shown in (1), where M_0 is the original mesh, $ecol_i$ represents the ith edge collapse operation. When $ecol_i$ is performed, M_{i-1} is simplified to M_i. After

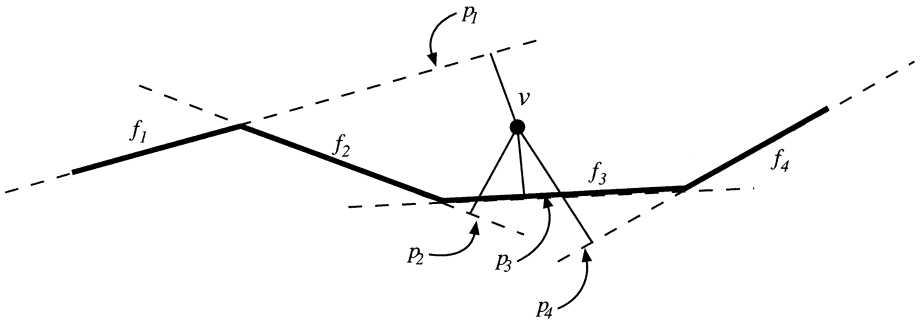


Fig. 3. Quadric error of a vertex is the sum of the squared distances to the planes of the vertex which are the planes containing the triangles of the original local geometry of the vertex. Let us assume that a vertex v has a original local geometry consisting of four triangles $f_1,\ f_2,\ f_3$, and f_4. p_i is the plane containing the triangle f_i. The quadric error of v is $d_1^2 + d_2^2 + d_3^2 + d_4^2$ where d_i is the distance from v to the plane p_i.

the subsequent n edge collapse operations, the compact mesh $\hat{M}$ is obtained

$$M = M_0 \xrightarrow{\text{ecol}_1} M_1 \xrightarrow{\text{ecol}_2} M_2 \xrightarrow{\text{ecol}_3} \cdots \xrightarrow{\text{ecol}_n} M_n = \hat{M}. \quad (1)$$

The order of the edge collapse operations is determined to minimize the geometric error between the original mesh and the compact mesh. The edges of M_i are sorted according to the cost of collapsing edges. The cost of collapsing an edge, $(v_1,\ v_2) \longrightarrow v$, is the geometric error between the local geometry and the original local geometry of the vertex v. The local geometry of a vertex consists of the neighbor triangles of the vertex which are the triangles meeting at the vertex. The "original" local geometry of a vertex is a part of the original mesh approximated by the local geometry of the vertex.[1] The original local geometry of each vertex of the original mesh is equal to the local geometry of the vertex. When an edge collapse $(v_1,\ v_2) \longrightarrow v$ occurs, the original local geometry of the target vertex v is the union of the original local geometries of v_1 and v_2.

For example, let us assume that the left mesh in Fig. 2 is the original mesh. The original local geometries of v_1 of v_2 are f_{1-6} and f_{5-9}, respectively. The local geometry of v_1 (v_2) is the same as the original local geometry of v_1 (v_2). The local geometry of the vertex v is $f_{1-4},\ f_{7-9}$. The original local geometry of the vertex v is f_{1-9}, which is the union of f_{1-6} and f_{5-9}. The cost of collapsing the edge $(v_1,\ v_2)$ is the geometric error between $f_{1-4},\ f_{7-9}$ and f_{1-9}.

Initially, the edges of the original mesh are sorted by the collapsing costs and stored in a priority queue. The top edge with the highest priority is removed from the queue and is collapsed. After a top edge is collapsed, the queue is updated accordingly. Collapsing the top edges continues until a termination condition is met.

To capture the geometric error between the original geometry and the local geometry of a vertex, there have been various approaches. A Hausdorff distance between two point sets, A and B, is defined as $\text{Max}(d(A,\ B),\ d(B,\ A))$, where $d(x,\ B)$ is the distance from a point x to the point set B and $d(A,\ B)$ is $\text{Max}_{x \in A} d(x,\ B)$. It is very expensive to compute the Hausdorff distance between two meshes exactly because many points on the meshes should be sampled. Practically, the vertexes, the center points of edges, and the center points of triangles of the two meshes are selected for computing the Hausdorff distance.

As an alternative to the expensive Hausdorff metric, simple and efficient geometric error metrics have been proposed [21]–[25]. Among those metrics, the quadric error metric proposed in [25] has been widely used because of its computing efficiency and high fidelity. Our geometric error is based on the quadric error metric and we describe the quadric metric in detail in the following.

[1]The geometric error of a vertex is defined as the geometric error between the original local geometry and the local geometry of the vertex.

The quadric error metric defines the geometric error of a vertex as the sum of the squared distances from the vertex to the planes containing the triangles of the original local geometry of the vertex, called as the planes of the vertex. An example is shown in Fig. 3 which is a 2–D view of the original local geometry of a vertex. The quadric error of a vertex can be written as (2). P_v is the planes of the vertex v and $p = [a\ b\ c\ d]^T \in P_v$ represents the plane defined by the equation $ax + by + cz + d = 0$, where $a^2 + b^2 + c^2 + d^2 = 1$

$$\Delta(v) = \sum_{p \in P_v} (p^T v)^2. \quad (2)$$

Equation (2) can be rewritten in quadratic form as follows:

$$\Delta(v) = \sum_{p \in P_v} (v^T p)(p^T v) \quad (3)$$

$$= \sum_{p \in P_v} v^T (pp^T) v \quad (4)$$

$$= v^T \left(\sum_{p \in P_v} K_p \right) v \quad (5)$$

$$= v^T Q_v v \quad (6)$$

where

$$K_p = pp^T = \begin{bmatrix} a^2, & ab & ac & ad \\ ab, & b^2 & bc & bd \\ ac, & bc & c^2 & cd \\ ad, & bd & cd & d^2 \end{bmatrix}. \quad (7)$$

Tracking the planes of a vertex while performing an edge collapse $(v_1,\ v_2) \longrightarrow v$ requires computing $P_{v_1} \cup P_{v_2}$, which incurs large memory and computation overheads for a complicated mesh. For efficiency, we compute the quadric of a vertex v by adding the quadrics of v_1 and v_2 instead of computing by summing K_p of all the planes of the vertex.

The simple addition method of computing the Q_v is computationally efficient and gives a conservative approximate error measure as shown in (8) and (9). If the planes of v_1 and v_2 are disjoint, the quadric of v is equal to the sum of the quadrics of v_1 and v_2. Otherwise, a single plane may be counted multiple times. The approximate error measure using $\hat{Q}_v$ is conservative: the quadric error computed using $\hat{Q}_v$ is not smaller than the quadric error computed using Q_v

$$\begin{aligned} Q_v &= \sum_{p \in P_{v_1} \cup P_{v_2}} K_p \\ &= \sum_{p \in P_{v_1}} K_p + \sum_{p \in P_{v_2}} K_p - \sum_{p \in P_{v_1} \cap P_{v_2}} K_p \\ &= Q_{v_1} + Q_{v_2} - Q_c \end{aligned} \quad (8)$$

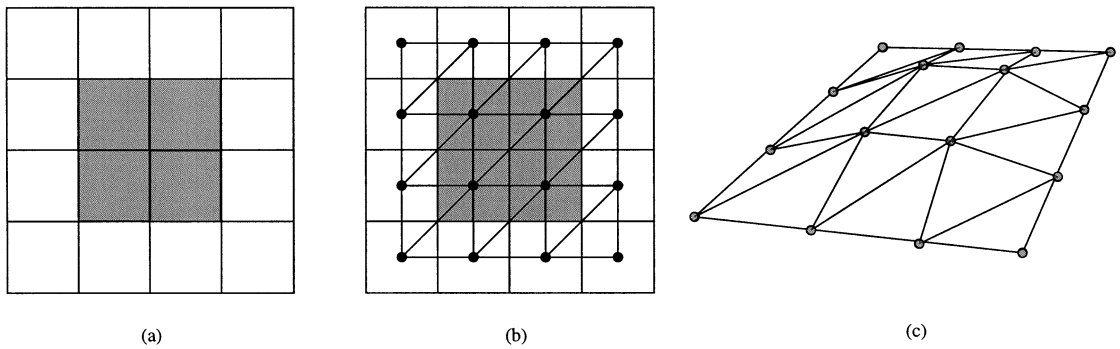


Fig. 4. Generation of the base mesh from grids. For illustration, the world space represented using a 4×4 grid is triangulated by 18 triangles. Generally, $2(N-1)^2$ triangles are created from $N \times N$ grid. (a) 4×4 input grid; shaded cell is a obstacle space and white cell is a free space. (b) A top view of the base mesh. (c) A perspective view of the base mesh. The free space vertexes are on the plane $z = 0$ and the obstacle space vertexes are on the plane $z = \alpha$, where $\alpha > 0$.

$$\hat{Q_v} = Q_{v_1} + Q_{v_2} = \begin{cases} Q_v, & \text{if } P(v_1) \cap P(v_2) = \phi \\ Q_v + Q_c, & \text{otherwise.} \end{cases} \quad (9)$$

To perform an edge collapse $(v_1, v_2) \longrightarrow v$, the position of the target vertex v should be determined. A simple scheme would be to select either v_1, v_2, or $(v_1 + v_2)/2$ depending on which one of these produces the lowest value of $\Delta(v)$. However, to minimize the geometric error introduced by the edge collapse, Garland *et al.* finds the position of v which can minimize the $\Delta(v)$ in [25] and we follow their method.

Since $\Delta(v)$ is quadratic, finding its minimum is a linear problem. The v is determined by solving $\partial\Delta/\partial x = \partial\Delta/\partial y = \partial\Delta/\partial z = 0$, which is equivalent to solving (10) for $\overline{v}$[2]

$$\begin{bmatrix} q_{11} & q_{12} & q_{13} & q_{14} \\ q_{21} & q_{22} & q_{23} & q_{24} \\ q_{31} & q_{32} & q_{33} & q_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix} \overline{v} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}. \quad (10)$$

Assuming that the matrix is invertible, we get $\overline{v}$ as (11)

$$\overline{v} = \begin{bmatrix} q_{11}, & q_{12} & q_{13} & q_{14} \\ q_{21}, & q_{22} & q_{23} & q_{24} \\ q_{31}, & q_{32} & q_{33} & q_{34} \\ 0, & 0 & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}. \quad (11)$$

If this matrix is not invertible, it is attempted to find the optimal vertex along the segment (v_1, v_2). If this also fails, $\overline{v}$ is chosen amongst the endpoints and the midpoint. The simplification algorithm based on the quadric metric is as follows:

1. Compute the Q matrices for all the vertices of the original mesh;
2. Compute the optimal collapse target $\overline{v}$ for collapsing each edge (v_1, v_2) of the original mesh. The quadric error of this target vertex, $\overline{v}^T(Q_{v_1} + Q_{v_2})\overline{v}$, is the cost of collapsing the edge;
3. Place all the edges in a heap keyed on cost with the minimum cost edge at the top;
4. Iteratively remove the edge with the minimum cost from the heap, collapse this edge (v_1, v_2), and update the costs of all edges involving v_1. This continues until the simplified mesh's geometric error is below the specified error threshold.

[2]You can verify this by taking partial derivatives of $v^T Q v = q_{11}x^2 + 2q_{12}xy + 2q_{13}xz + 2q_{14}x + q_{22}y^2 + 2q_{23}yz + 2q_{24}y + q_{33}z^2 + 2q_{34}z + q_{44}$.

III. Path Planning Based on Path Graph Optimization

Our contribution is to represent a world space as a 3–D mesh, simplify the mesh and extract a compact path graph from the simplified mesh. First, the world space is triangulated based on the input grid. The resulting mesh, which is called a base mesh, has many triangles. The base mesh is simplified without introducing much geometric error. Even after significant simplification, the simplified mesh can represent the world space with almost equal quality to that of the base mesh. Next, a path graph is extracted from the simplified mesh. The resulting path graph is much simpler than the base mesh while preserving the obstacle boundaries in the graph.

Our algorithm consists of five steps as follows, which will subsequently be explained in detail:

1) *generation of base mesh by triangulation of world space*;
2) *simplification of the base mesh*;
3) *generation of the path graph*;
4) *path planning using Dijkstra's shortest path algorithm*;
5) *path relaxation.*

A. Generation of the Base Mesh

A 2–D world space is converted to a 3–D base mesh to simplify the world space by the mesh simplification algorithm. The base mesh is set up based on the grid input as illustrated in Fig. 4. The center nodes of grid cells are used as the vertexes of the base mesh. Edges are created by connecting the vertexes according to the adjacency of the grid cells. The x and y coordinates of a vertex are the 2–D position of the vertex in the 2–D world space. The vertexes in a free space have zero z coordinate values while the vertexes in an obstacle space have nonzero z coordinate values. An example of base mesh generation is given in Fig. 4.

B. Modified Quadric Error Metric

To improve the path planning quality, balanced node distribution in a free space is required. However, using the conventional quadric error metric, the free space is overly simplified and has only a few vertexes. The base mesh has high curvatures on the regions near the obstacle boundaries and the other regions are all flat. Most of edges in a free space have zero collapsing costs. If the planes of v_1 and v_2 of an edge are the plane whose equation is $z = 0$, the target vertex v is placed on the plane $z = 0$, and as a result, the conventional quadric error of v is zero. Since collapsing those edges has zero cost, most of the edges in a

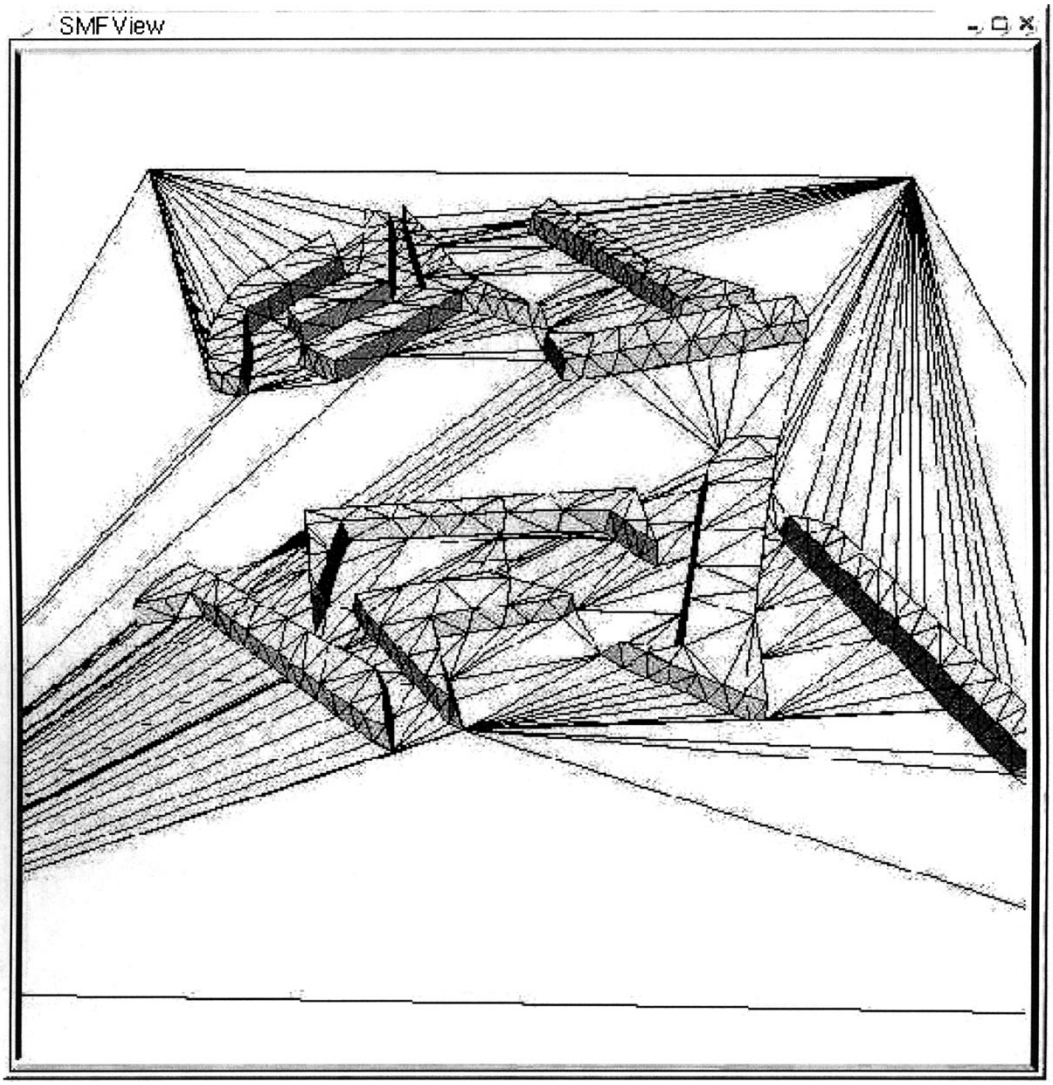


Fig. 5. Simplified mesh using conventional quadric metric. Obstacle boundaries are preserved well but free space is overly simplified.

free space are collapsed and only a few vertexes remain in a free space. For illustration, an experimental world space configuration shown in Fig. 8 is simplified using the conventional quadric error metric and the resulting simplified mesh is shown in Fig. 5, which is a 3–D rendered image of the mesh.

For the balanced distribution of nodes in a free space, we do not allow edge collapse which generates the edges of length longer than an edge length threshold. The free space is simplified as much as possible ensuring that the edges are not longer than the edge length threshold. We determine the edge length threshold empirically. In our experiment with various edge length thresholds, the edge length threshold of 10% of the world space size (51 grids for a 512×512 world space for example) is good for both efficiency and quality of path planning.

The termination condition of the simplification process is modified such that the simplification terminates when there is no more edge collapse which does not incur longer edges than the edge length threshold and the geometric error is below the error threshold.

C. Generation of Path Graph

The path graph consists of the vertexes and edges in a free space. Among the edges of the compact mesh, only the edges lying in a free space are selected as the arcs of the path graph. The obstacle boundaries should be included in the path graph to achieve a good quality path planning. The obstacle boundary edges are those edges which are in a free space, enclose an obstacle, and are right next to the obstacle. During the simplification process, the edges of the base mesh which lie in a free space representing the obstacle boundaries may be moved into an obstacle space. Those obstacle boundary edges should be adjusted to be in a free space and be included in the path graph. Otherwise, the quality of the path graph is poor and finding the shortest paths is not possible.

Optimal adjustment of the obstacle boundary edges is too expensive to be done on-line for path planning purposes. We propose an iterative vertex pushing scheme to adjust the obstacle boundary edges. The vertexes of the obstacle boundary edges crossing the obstacle space are pushed toward the free space by a small amount multiple times until all the boundary edges are placed in a free space. The direction of pushing a vertex is the vector sum of the normal vectors of the two obstacle boundary edges[3] meeting at the vertex.

There is a tradeoff between efficiency and quality in determining the pushing amount. When using small pushing amount, the quality of the obstacle boundary is high but the number of required iterations increases. When using large pushing amount, the narrow passages cannot be handled well. In our experiments, the pushing amount is the half of

[3]The normal vector of an edge is a vector which lies in the x–y plane and is normal to the vector which is constructed by the corner vertexes of the edge.

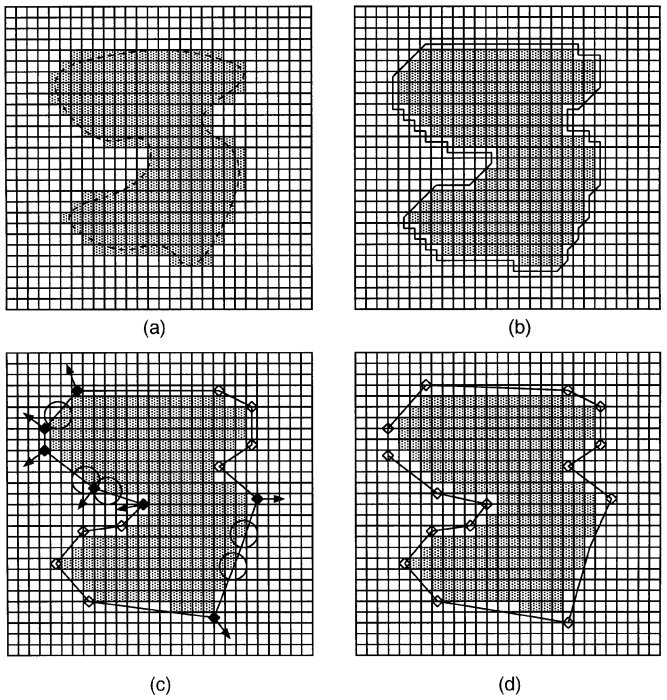


Fig. 6. Vertex pushing example. (a) Obstacle and the input grid for the obstacle. The cells in an obstacle space are shaded. (b) Obstacle boundary consists of 88 vertexes and 88 edges. (c) Obstacle boundary in the compact mesh consists of 15 vertexes and 15 edges. In the circled regions, the boundary edges intersect with the obstacle space. The vertexes to push are marked black and the pushing directions are shown for the vertexes. (d) After pushing the marked vertexes, all the obstacle boundary edges are in a free space.

the input grid size, which is small enough to handle the narrow passages well. Since the obstacle boundary edges of the compact mesh tightly represent the obstacle space, the number of required iterations is not large.

At each iteration, the vertexes of the edges crossing the obstacle space are pushed toward the free space by the half of the input grid size. To check if an edge crosses the obstacle space, digital discrete analyzer (DDA) [26] is used. DDA gives the cells of a uniform grid touched by a line segment on the uniform grid. If no obstacle boundary edge crosses the obstacle space, the vertex pushing is completed. An example is shown in Fig. 6.

The iterative vertex pushing method can recover most obstacle boundary edges of the compact mesh. For narrow passages, the vertex pushing method may fail when an edge crossing an obstacle is pushed but crosses another obstacle again. However, the problem occurs rarely because the narrow passages are represented accurately and occurs only for extremely narrow passages. Even so, it can be solved by using smaller vertex pushing amount and smaller geometric error threshold.

D. Comparison of Mesh and Quadtree Representations

The proposed mesh representation requires less number of vertexes than that required for the quadtree representation to model an obstacle with the same resolution.[4] To represent a boundary using the quadtree, cells near the boundary are divided until a desired resolution is reached. Since the shape and position of a cell is not object-dependent but prefixed, many small cells are required to represent the obstacle boundary. In contrast, in the mesh representation, the position of a vertex is optimized to represent the obstacle boundary tightly. An example is shown in Fig. 7. The mesh representation outperforms the quadtree representation for all possible boundary shapes whether flat or curvy because the positions of the vertexes are optimized adaptively to the curvatures. Even for long narrow curvy passages, the mesh representation requires fewer vertexes than the quadtree.

[4]The resolution of a representation is defined by the geometric error between the representation and the original obstacle boundary.

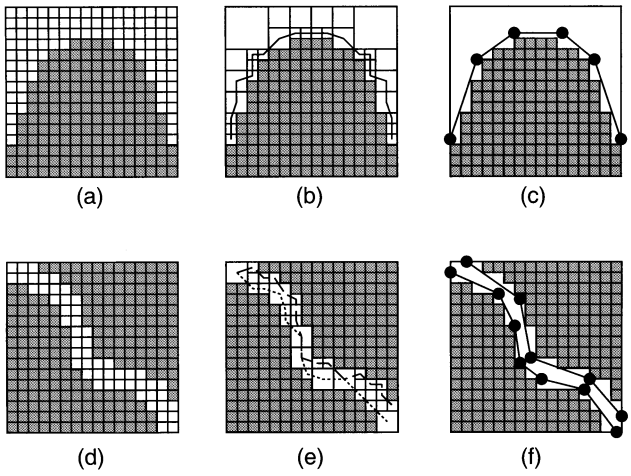


Fig. 7. Comparison of obstacle boundary modeling using the quadtree and the mesh. (a) Example obstacle. (b) Quadtree requires 24 vertexes to model the obstacle boundary. (c) Mesh requires six vertexes to model the obstacle boundary with the nearly equal resolution of (b). (d) Narrow curvy passage example. (e) Quadtree requires 31 vertexes. (f) Mesh requires 12 vertexes.

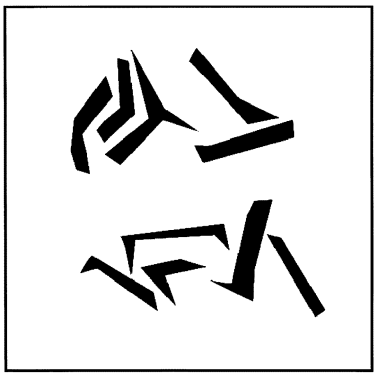


Fig. 8. World space configuration which we experimented. The obstacle space is black and the free space is white. The size is 512×512.

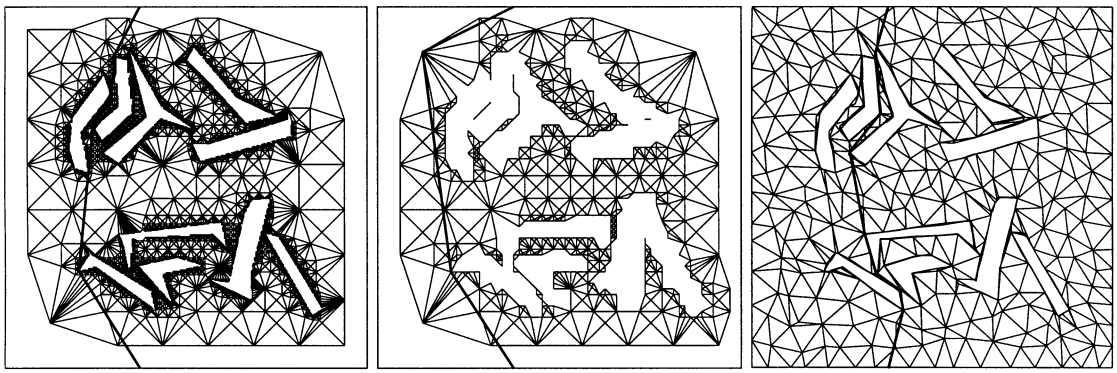
IV. Results

We implemented our algorithms and tested for synthesized world space configurations. Dijkstra's shortest path searching algorithm is used for searching the shortest path in a path graph, and a simple path relaxation algorithm is used to optimize the path. All experiments are performed on a SUN Ultra 20 workstation with 256 MB main memory. The experimental world space configuration of 512×512 size is shown in Fig. 8 where the obstacle space is black and the free space is white.

Initially the base mesh is constructed from the 512×512 grid and it took 50 seconds to simplify the base mesh and extract the path graph. After this preprocessing, the path planning can be done very quickly. For planning a path from (192, 0) to (192, 512),[5] the generated paths are shown as thick lines in Fig. 9, and the time complexity for solving the path planning problem is summarized in Table I.

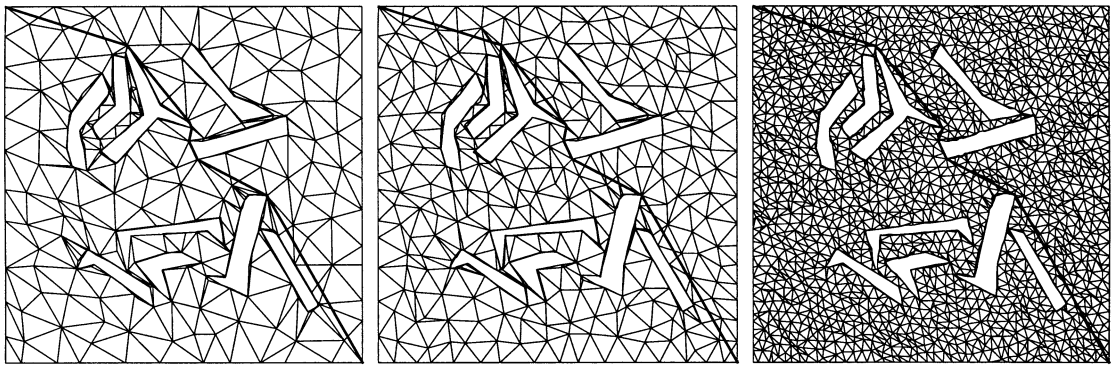
Even though the proposed method uses much simpler path graph than the grid and the quadtree methods, the proposed path planning

[5](0, 0) is the left top point of the world space and (512, 512) is the right bottom point of the world space.



(a) (b) (c)

Fig. 9. Path planning from (192, 0) to (192, 512) on a 512 × 512 world space. (a) Quadtree (512 × 512) $(\mathrm{nodes},\ \mathrm{arcs}) = (4225,\ 26638)$ path length = 554.506. (b) Quadtree (64 × 64) $(\mathrm{nodes},\ \mathrm{arcs}) = (440,\ 2458)$ path length = 619.65, (c) proposed $(\mathrm{nodes},\ \mathrm{arcs}) = (502,\ 2694)$ path length = 552.7. The generated path is the thick line segments. In qt (64 × 64) the narrow passages vanish. So a nonoptimal path is found. Using our optimized path graph which has complexity nearly equal to that of qt (64 × 64), narrow passages are preserved well, so the shortest path can be found.



(a) (b) (c)

Fig. 10. Path planning from (0, 0) to (512, 512). The edge length threshold l is controlled to generate different resolution path graphs. Even in a very low complexity path graph, the obstacle features remain. (a) $l \leq 70,\ (N,\ A) = (359,\ 1890)$ path length = 771.503, (b) $l \leq 50,\ (N,\ A) = (502,\ 2694)$ path length = 771.914, (c) $l \leq 20,\ (N,\ A) = (2106,\ 11760)$ path length = 772.603.

TABLE I
COMPARISON OF THE PATH PLANNING TIME COMPLEXITY. ALL ARE MEASURED IN SECONDS

	Uniform grid (512x512)	Quadtree (512x512)	Quadtree (64x64)	**Proposed** (512x512)
graph nodes	237,274	4,225	440	**502**
graph arcs	1,881,692	26,638	2,458	**2,694**
time	56.0	0.875	0.09	**0.09**
path length	556.0	554.506	619.65	**552.7**

method gives the paths whose lengths are nearly the same as those generated by the quadtree and the uniform grid methods. It is due to two advantages of the proposed method. The first advantage is considerable graph simplification without losing the important obstacle features. As shown in Fig. 9(a), the quadtree have many small cells to represent obstacle boundaries and the number of edges of the path graph is 26 638. As shown in Fig. 9(c), the proposed optimized path graph represents the obstacle boundaries as accurately as 512 × 512 quadtree using only 2694 edges. In contrast, the simplification of the quadtree by lowering the resolution of the quadtree cannot represent the obstacle features faithfully. Fig. 9(b) shows the 64 × 64 quadtree having 2458 edges. Even though the number of edges of the 64 × 64 quadtree is nearly the same as that of the proposed method, the narrow passages in an obstacle crowded environments disappear severely. As a result, the shortest path cannot be found and the path length of the generated nonoptimal path is 619.65 which is much larger than 552.7 of the proposed method.

The second advantage to our method is that the free space such that it does not exceed the user-specified edge length threshold during the simplification process. In contrast, the quadtree has cells which are too large in a free space, thus giving nonoptimal paths. The density of distribution of nodes in a free space is user-controllable as shown in Fig. 10, where the edge length threshold is 70, 50, and 20 grids for Fig. 10(a)–(c), respectively. There is a trade-off in choosing the edge length threshold. As the edge length threshold is smaller, the quality of the path is enhanced. However a too small threshold value is not good for efficiency. Experimentally we observe 10% of the world space size to be an appropriate choice for both efficiency and quality.

V. CONCLUSIONS AND FUTURE WORK

In this paper we propose a new path planning method to address the limitations of the quadtree method. The first limitation is that the quadtree requires many small cells to represent an obstacle boundary. The second limitation is that the free space cells may be so large as to result in suboptimal paths.

The main reason of the first limitation is that the quadtree representation is not object-dependent;the positions and shapes of the cells are not object-dependent. We propose a new compact mesh representation

for path planning. The vertexes and edges of the compact mesh are determined to optimally represent the curvatures of the obstacles with much fewer number of vertexes and edges than the quadtree method. A base mesh is generated by the triangulation of the world space and simplified by a modified quadric error metric to a compact mesh. The simplification algorithm optimizes the position of the vertexes to represent the obstacles tightly. The geometric error of the compact mesh and the base mesh is small while the compact mesh has much fewer number of vertexes than the base mesh.

To address the second limitation, the simplification algorithm spreads vertexes in a free space in a balanced way. The density of vertexes in a free space is controllable by specifying an edge length threshold. The path graph is extracted from the compact mesh. An iterative vertex pushing method is used to adjust the obstacle boundary edges crossing an obstacle space to be in a free space. Dijkstra's algorithm is used for searching the shortest path in the path graph. Experiments show that the proposed method generates paths as short as the uniform grid and the quadtree using a much simpler path graph.

In this paper we assume a 2–D static environment. However, the proposed method can be easily extended to 3–D dynamic environments. The quadric error metric supports 3–D or higher dimensions readily as described in [27]. The triangle in 2–D is equivalent to a tetrahedron in 3–D. So our algorithm can be extended to 3–D by changing only the base mesh generation step to 3–D tetrahedralization using a 3–D input grid data. For the extension to dynamic environments, we can exploit temporal coherency—only a small portion of the world space is dynamically changing. To find which portion of the world space is changing, we can partition the world space using a popular spatial hierarchy such as an octree. Then only updated portions of the mesh are refined to the resolution of the base mesh. New grid data is applied to the updated portions. By simplifying only the updated portions, the simplification of dynamic environments can be done efficiently.

REFERENCES

[1] J. C. Latombe, *Robot Motion Planning*. Norwell, MA: Kluwer, 1991.

[2] Y. K. Hwang and N. Ahuja, "Gross motion planning—A survey," *ACM Comput. Surv.*, vol. 24, pp. 219–291, 1992.

[3] T. Lozano-Perez and M. A. Wesley, "An algorithm for planning collision-free paths among polyhedral obstacles," *Commun. ACM*, vol. 22, pp. 560–570, Oct. 1979.

[4] J. T. Schwartz and M. Sharir, "On the piano movers problem-the case of a two-dimensional rigid polygonal body moving amidst polygonal barriers," *Commun. Pure Appl. Math.*, vol. 36, pp. 345–398, Oct. 1983.

[5] C. Alexopoulos and P. M. Griffin, "Path planning for a mobile robot," *IEEE Trans. Syst., Man, Cybern.*, vol. 22, pp. 318–322, Mar. 1992.

[6] J. Ilari and C. Torras, "Two dimensional path planning: A configuration space heuristic approach," *Int. J. Robot. Res.*, vol. 9, pp. 75–91, Feb. 1990.

[7] J. Barraquand, B. Langlois, and J. C. Latombe, "Numerical potential field techniques for robot path planning," *IEEE Trans. Syst., Man, Cybern.*, vol. 22, pp. 224–241, Jan./Feb. 1992.

[8] R. Jarvis and J. Byrne, "Robot navigation: Touching, seeing and knowing," in *Proc. 1st Australian Conf. Artificial Intelligence*, Nov. 1986.

[9] K. I. Trovato, "DifferentialA*: An adaptive search method illustrated with robot path planning for moving obstacles and goals, and an uncertain environment," *Int. J. Pattern Recognit.*, vol. 4, no. 2, pp. 245–268, 1990.

[10] G. Borgefors, "Distance transformations in arbitrary dimensions," *Comput. Vis., Graph. Image Process.*, vol. 27, pp. 321–345, 1984.

[11] T. E. Boult, "Updating distance maps when objects move," *Proc. SPIE*, vol. 852, pp. 232–238, 1987.

[12] S. Kambhampati and L. S. Davis, "Multiresolution path planning for mobile robots," *IEEE J. Robot. Automat.*, vol. RA-2, pp. 135–145, Sept. 1986.

[13] R. Chan, P. Tam, and D. Leung, "Robot navigation in unknown terrains via multi-resolution grid maps," in *Proc. Int. Conf. Industrial Electronics*, 1991, pp. 1138–1143.

[14] R. Mehrotra and D. Krause, "Obstacle-free path planning for mobile robots," in *Proc. 3rd Annu. Conf. Image Processing Applications*, 1989, pp. 431–435.

[15] H. Noborio, T. Naniwa, and S. Arimoto, "A fast path-planning algorithm by synchronizing modification and search of its path graph," in *Proc. Int. Workshop Artificial Intelligence Industrial Applications*, 1988, pp. 351–357.

[16] A. Zelinsky, "A mobile robot exploration algorithm," *IEEE Trans. Robot. Automat.*, vol. 8, pp. 707–717, Dec. 1992.

[17] N. Tomohide and S. Arimoto, "A quadtree-based path planning algorithm for a mobile robot," *J. Robot. Syst.*, vol. 7, no. 4, pp. 555–574, 1990.

[18] D. Z. Chen, R. J. Szczerba, and J. J. Uhran, Jr., "A framed-quadtree approach for determining Euclidean shortest paths in a 2-D environment," *IEEE Trans. Robot. Automat.*, vol. 5, pp. 668–681, Oct. 1997.

[19] P. S. Heckbert and M. Garland, "Survey of polygonal surface simplification algorithms," in *SIGGRAPH'97 Course Notes*, 1997.

[20] J. Cohen, A. Varshney, D. Manocha, G. Turk, H. Weber, P. Agarwal, F. Brooks, and W. Wright, "Simplification envelopes," in *Proc. SIGGRAPH'96*, Aug. 1996, pp. 119–128.

[21] W. J. Schroeder and M. S. Shephard, "Geometry-based fully automatic mesh generation and the Delaunay triangulation," *Int. J. Numer. Meth. Eng.*, vol. 26, pp. 2503–2515, 1988.

[22] G. Turk, "Re-tiling polygonal surfaces," in *Proc. SIGGRAPH'92*, July 1992, pp. 55–64.

[23] M. DeHaemer, Jr. and M. J. Zyda, "Simplification of objects rendered by polygonal approximations," *Comput. Graph.*, vol. 15, no. 2, pp. 175–184, 1991.

[24] J. Rossignac and P. Borrel, "Multi-resolution 3-D approximations for rendering complex scenes," in *Modeling in Computer Graphics: Methods and Applications*. New York: Springer-Verlag, 1993, pp. 455–465.

[25] M. Garland and P. S. Heckbert, "Surface simplification using quadric error metrics," in *Proc. SIGGRAPH'97*, 1997, pp. 209–216.

[26] K. Sung, "A dda octree traversal algorithm for ray tracing," in *Proc. Eurographics '91*, Sept. 1991.

[27] M. Garland and P. S. Heckbert, "Simplifying surfaces with color and texture using quadric error metrics," in *Proc. IEEE Visualization'98*, 1998, pp. 209–216.