University of Delaware
Department of Electrical and Computer Engineering
Computer Architecture and Parallel Systems Laboratory

Self-Avoiding Walks over Two-Dimensional Adaptive Unstructured Grids

Gerd Heber
Rupak Biswas†
Guang R. Gao

CAPSL Technical Memo 21
April 20, 1998

†MRJ Technology Solutions, Mail Stop T27A-1, NASA Ames Research Center, Moffett Field, CA 94035

University of Delaware • 140 Evans Hall • Newark, Delaware 19716 • USA
http://www.capsl.udel.edu • ftp://ftp.capsl.udel.edu • capsladm@capsl.udel.edu

Abstract

In this report, we present a new approach to construct a "self-avoiding" walk through a triangular mesh. Unlike the popular approach of visiting triangles using *space-filling curves* which is based on a geometric embedding, our approach is combinatorial in the sense that it makes use of the mesh-connectivity only. We present an algorithm for constructing a self-avoiding walk which can be applied to any unstructured triangular mesh. The algorithm's complexity is $\mathbf{O}(n \cdot \log(n))$ where n is the number of triangles in the mesh. We show that for hierarchical adaptive meshes the algorithm can be easily parallelized, taking advantage of the regularity of the refinement rules. The proposed approach should be very useful in the runtime partitioning and load balancing of adaptive unstructured grids.

Contents

List of Figures

1 Introduction

Advances in adaptive software and methodology notwithstanding, parallel computational strategies will be an essential ingredient in solving complex, real-life problems. However, parallel computers are easily programmed with regular data structures; so the development of efficient parallel adaptive algorithms for unstructured grids poses a serious challenge. An efficient parallelization of these unstructured adaptive methods is rather difficult, primarily due to the load imbalance created by the dynamically-changing nonuniform and irregular grid. Nonetheless, it is generally believed that unstructured adaptive-grid techniques will constitute a significant fraction of future high-performance supercomputing. Mesh adaption and dynamic load balancing must be accomplished rapidly and efficiently, so as not to cause a significant overhead to the numerical simulation.

Serialization techniques play an important role in all parts of a Finite Element Method (FEM) [22] over adaptive unstructured grids. A numbering of the unknowns allows for a matrix-vector notation of the underlying algebraic equations. Special numbering techniques (Cuthill–McKee, frontal methods) have been developed to optimize memory usage and locality of the algorithms. On the other hand, in many cases, runtime support for decomposing dynamic adaptive grid hierarchies is based on a *linear representation* of the grid hierarchy [16] in the form of a *space-filling curve*. Salmon et al. [19, 20, 23] have demonstrated the successful application of techniques based on space-filling curves to N–body simulations. Other researchers [1, 7, 11, 13, 14, 15, 16, 17] have also shown how space-filling curves can be used for graph partitioning and similar graph-related problems.

The general idea of a space-filling curve is a sort of a serialization (or linearization) of visiting points in a higher-dimensional space. A standard method for the construction is to embed the object of study into a regular environment (where the standard space-filling curves live). This approach introduces a kind of an "artificial" structure in the sense that the entire construction depends on the embedding.[1] Furthermore, this type of a linear representation forgets about the *combinatorial* structure of the mesh which drives the formulation of operators between the finite element spaces. The questions arises whether one could reduce the different requirements for a serialization in an adaptive FEM over unstructured grids to a common denominator.

In this report, we present a new approach to construct a "self-avoiding" walk through a triangular mesh. Unlike the popular approach of visiting triangles using space-filling curves which is based on a geometric embedding, our approach is combinatorial in the sense that it makes use of the mesh-connectivity only. We present an algorithm for constructing a self-avoiding walk which can be applied to any unstructured triangular mesh. The term *unstructured* refers to information that can hardly be compressed, and which does not contain any symmetry

[1] It is similar to describing metric properties of a curved surface in terms of a parametrization in the enveloping (uncurved) three space instead of doing *inner geometry*, i.e., geometry based only on measurements on the surface itself. The only inner properties of a mesh are its decomposition into cells of different dimensions and a face relation describing how the cells are glued together (see Appendix A).

or redundancy. At this level, any labeling or numbering of the mesh components is as good (or as bad!) as any other scheme. However, the situation changes if the mesh adaption is done in a *hierarchical* manner where we, based on a set of simple rules for coarsening and refinement, pay particular attention to the triangulation of the initial mesh. In contrast with a general adaption scheme that produces just another unstructured mesh, this strategy exploits the structure of the adaption hierarchy to simplify the labeling process. We would like to modify (or adapt) the labeling only in regions where the mesh has been altered but not rebuild the entire indexing. This idea has been extensively developed in [8].

Self-avoiding walks can be used to improve the efficiency of the respective algorithms, in particular locality and load balancing. However, we need to search for some special classes of self-avoiding walks that will facilitate runtime load balancing with good locality in terms of memory access or communication. Since we are not interested in just any kind of walks, we have to first tackle the question of the existence of walks with the desired properties. In this report, we give an existence proof which is constructive in the sense that it also provides an algorithm for generating such walks. For the case of hierarchical adaptive refinement the proposed algorithm is easily amenable to parallelization. In the process of deriving the existence of a particular class of walks, a natural mathematical framework emerges which, we believe, can be easily adapted for other special classes of walks.

In Section 2, we define a certain class of walks whose existence for arbitrary unstructured meshes will be proved in Section 3. In Section 4, we discuss some related algorithmic aspects and present some sample results. The algorithm presented in Section 4 works for arbitrary unstructured meshes; however, a parallelization is non-trivial and the walk has to be completely rebuilt after mesh adaption. In Section 5, we briefly review mesh adaption in general and hierarchical refinement in particular. We find the missing regularity that enables us to prove the existence of constrained walks and to formulate a truly parallel algorithm for the construction of walks which is well-behaved with respect to mesh adaption. Section 6 concludes the report with some perspectives.

2 Definitions

Consider an arbitrary two-dimensional triangular mesh[2] $\mathfrak{M}$ and denote the underlying set of triangles by H. Let $\#H$ denote the cardinality of H.

Definition 2.1. *A mapping*

$$\omega : \{1, \ldots, \#H\} \longrightarrow H \tag{1}$$

[2]By a *mesh* we understand a simplicial complex coming from the simplicial decomposition of a connected 2D manifold (with or without a boundary). For technical details, we refer the reader to [10, 18, 21] (see also Appendix A).

is called a <u>self-avoiding walk</u> $\Longleftrightarrow_{\text{def}}$ ω *is a bijection and*

$$\forall i \in \{1, \ldots, \#H - 1\} \quad \omega(i) \text{ and } \omega(i+1) \text{ share an edge or a vertex.} \tag{2}$$

Definition 2.1 states that a walk visits each triangle exactly once and that *jumps* (over non-incident faces) are forbidden.

Remark. At first glance, condition (2) might appear rather weak. As simple examples show, the requirement that triangles following one another should share an edge, is too strong. Generally, self-avoiding walks do not exist under this assumption (cf. Fig. 1 for a trivial example).

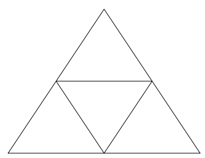

Figure 1: A simple counterexample (look at the dual graph!).

In the following, we consider a special class of self-avoiding walks.

Definition 2.2. *A self-avoiding walk is called* <u>proper</u> $\Longleftrightarrow_{\text{def}}$

$$\forall i \in \{2, \ldots, \#H - 1\} \quad \omega(i-1) \cap \omega(i) \neq \omega(i) \cap \omega(i+1). \tag{3}$$

Definition 2.2 states that for three triangles following one another in a self-avoiding walk, "jumping" twice over the same vertex is forbidden (cf. Fig. 2).

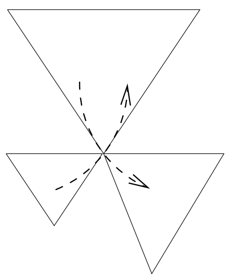

Figure 2: A forbidden jump.

If two triangles t_1, t_2 share an edge, we write $t_1 \mid t_2$. Let ω be a self-avoiding walk. If $\omega(i) \mid \omega(i+1)$, we will write $\omega(i) \vdash \omega(i+1)$ indicating that ω enters $\omega(i+1)$ from $\omega(i)$ over an edge. If $\omega(i) \nmid \omega(i+1)$, we write $\omega(i) \curvearrowright \omega(i+1)$ indicating that ω jumps into $\omega(i+1)$ from $\omega(i)$ over a vertex.

We conclude this section with two technical results. The respective proofs are not very difficult, but beyond the scope of this report.

Lemma 2.1. *Let t, t_1, t_2, t_3 be triangles in a mesh $\mathfrak{M}$ and*

$$\forall i \in \{1, 2, 3\} \quad t_i \mid t. \tag{4}$$

Then either

$$t_1 \cap t_2 \cap t_3 = \emptyset \tag{5}$$

or $\mathfrak{M}$ is tetrahedral (cf. Fig. 3). $\square$

(Note that if two triangles in a mesh share two vertices they *must* share the corresponding edge.)

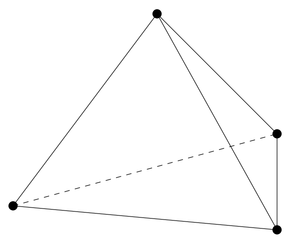

Figure 3: A tetrahedral mesh.

Lemma 2.2. *Let $\mathfrak{M}$ be a mesh. There exists a triangle τ in $\mathfrak{M}$ such that $\mathfrak{M} - \tau$, the complex obtained by removing τ from $\mathfrak{M}$, is still a mesh.* $\square$

3 Existence

Proposition 3.1. *There exists a proper self-avoiding walk for an arbitrary triangular mesh $\mathfrak{M}$.*

Proof. We prove proposition 3.1 by induction over the number of triangles.

Assume that there exist proper self-avoiding walks for meshes with n triangles. Let $\mathfrak{M}$ be a mesh with $n + 1$ triangles.

Let τ be a triangle in $\mathfrak{M}$ such that $\mathfrak{M} - \tau$, the mesh consisting of all the triangles in $\mathfrak{M}$ except τ, is a mesh (see lemma 2.2). This mesh has n triangles and hence, by our induction assumption, there exists a proper self-avoiding walk $\omega_{\mathrm{M}-\tau}$ for $\mathfrak{M} - \tau$. We show that $\omega_{\mathrm{M}-\tau}$ can be extended to a proper self-avoiding walk ω_{M} for $\mathfrak{M}$. We call this ω_{M} a *proper extension* of $\omega_{\mathrm{M}-\tau}$.

Let t be an triangle of $\mathfrak{M} - \tau$ sharing an edge with τ (i.e. $t \mid \tau$ holds). For an appropriate $i \in \{1, \ldots, n\}$, there holds

$$t = \omega_{\mathrm{M}-\tau}(i). \tag{6}$$

The discussion naturally can be split up into four cases, depending on whether $\omega_{\mathrm{M}-\tau}$ enters t through an edge or a vertex and leaves through an edge or a vertex, respectively.

We omit the subscript $\mathfrak{M}-\tau$ from ω in the following for the sake of clarity. If we do not know whether the transition from $\omega(i)$ to $\omega(i+1)$ goes over an edge or a vertex, we write $\omega(i) \rightarrow \omega(i+1)$.

In the figures below, parts of $\omega_{\mathrm{M}-\tau}$ are drawn as solid lines whereas the modifications leading to ω_{M} are drawn as dashed lines. Triangles are indexed by their position in the walk (e.g. $\omega(i)$ is denoted by i).

Case I: $\boxed{\omega(i-1) \vdash \omega(i) \vdash \omega(i+1)}$

Figure 4 illustrates the modifications necessary if $\omega_{\mathrm{M}-\tau}$ enters and leaves t through an edge.

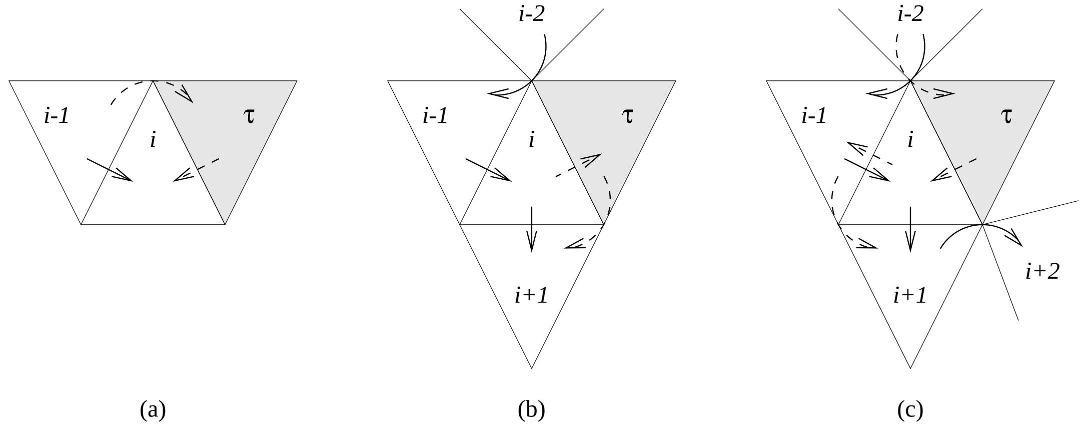

Figure 4: Walk entering and exiting the triangle adjacent to τ via an edge.

- If $\omega(i-1) \cap \tau \neq \omega(i-2) \cap \omega(i-1)$, then $\omega(i-1) \rightarrow \tau \vdash \omega(i)$ is a proper extension of ω (cf. Fig. 4(a)).
- If $\omega(i-1) \cap \tau = \omega(i-2) \cap \omega(i-1)$, then $\omega(i-2) \curvearrowright \omega(i-1)$.
  - If $\omega(i+1) \cap \tau \neq \omega(i+1) \cap \omega(i+2)$, then $\omega(i) \vdash \tau \rightarrow \omega(i+1)$ is a proper extension of ω (cf. Fig. 4(b)).
  - If $\omega(i+1) \cap \tau = \omega(i+1) \cap \omega(i+2)$, then $\omega(i+1) \curvearrowright \omega(i+2)$.
    * If $\omega(i-1) \cap \omega(i+1) \neq \omega(i+1) \cap \omega(i+2)$, then $\omega(i-2) \rightarrow \tau \vdash \omega(i) \vdash \omega(i-1) \rightarrow \omega(i+1)$ is a proper extension of ω (cf. Fig. 4(c)).

* Let $\omega(i-1) \cap \omega(i+1) = \omega(i+1) \cap \omega(i+2)$. According to lemma 2.1, either $\mathfrak{M}$ is tetrahedral or $\omega(i-1) \cap \omega(i+1) \cap \tau = \emptyset$. If $\mathfrak{M}$ is tetrahedral, there is nothing to prove. For the other case, we have $\emptyset = \omega(i-1) \cap \omega(i+1) \cap \tau = \omega(i+1) \cap \omega(i+2) \cap \tau = \omega(i+1) \cap \tau = \omega(i+1) \cap \omega(i+2)$. This is a contradiction with our assumption $\omega(i+1) \curvearrowright \omega(i+2)$ which means $\omega(i+1) \cap \omega(i+2) \neq \emptyset$. Therefore, if $\mathfrak{M}$ is not tetrahedral, the case $\omega(i-1) \cap \omega(i+1) = \omega(i+1) \cap \omega(i+2)$ is not possible.[3]

Case II: $\boxed{\omega(i-1) \vdash \omega(i) \curvearrowright \omega(i+1)}$

Figure 5 illustrates the modifications necessary if $\omega_{\mathrm{M}-\tau}$ enters t through an edge and leaves it through a vertex.

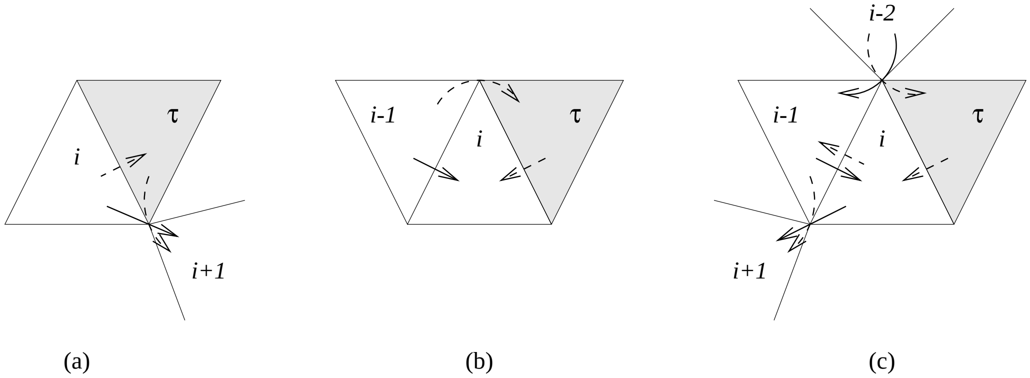

Figure 5: Walk entering the triangle adjacent to τ via an edge and iting via a vertex.

- If $\omega(i) \cap \omega(i+1) \subset \omega(i) \cap \tau$, then $\omega(i) \vdash \tau \rightarrow \omega(i+1)$ is a proper extension of ω (cf. Fig. 5(a)).
- Suppose $\omega(i) \cap \omega(i+1) \notin \omega(i) \cap \tau$. (Note that $\omega(i) \cap \omega(i+1)$ definitely consists of a single vertex!)
  - If $\omega(i-1) \cap \tau \neq \omega(i-2) \cap \omega(i-1)$, then $\omega(i-1) \rightarrow \tau \vdash \omega(i)$ is a proper extension of ω (cf. Fig. 5(b)).
  - If $\omega(i-1) \cap \tau = \omega(i-2) \cap \omega(i-1)$, then $\omega(i-2) \curvearrowright \omega(i-1)$. In that case, $\omega(i-2) \rightarrow \tau \vdash \omega(i) \vdash \omega(i-1) \rightarrow \omega(i+1)$ is a proper extension of ω (cf. Fig. 5(c)).

Case III: $\boxed{\omega(i-1) \curvearrowright \omega(i) \vdash \omega(i+1)}$

Figure 6 illustrates the modifications necessary if $\omega_{\mathrm{M}-\tau}$ enters t through a vertex and leaves it through an edge.

[3]Obviously, there are proper self-avoiding walks for tetrahedral meshes. Note that a tetrahedral mesh cannot be a proper submesh of a larger 2D mesh which is a manifold as a topological space [10, 18, 21].

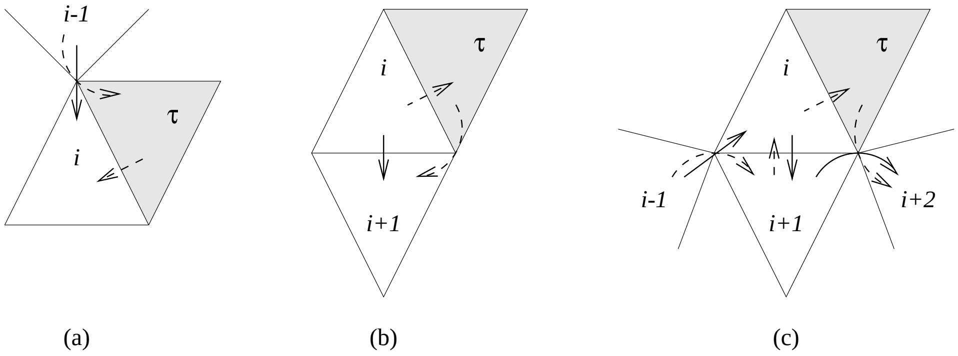

Figure 6: Walk entering the triangle adjacent to τ via a vertex and exiting via an edge.

- If $\omega(i-1) \cap \omega(i) \subset \omega(i) \cap \tau$, then $\omega(i-1) \rightarrow \tau \vdash \omega(i)$ is a proper extension of ω (cf. Fig. 6(a)).
- Suppose $\omega(i-1) \cap \omega(i) \notin \omega(i) \cap \tau$.
  - If $\omega(i+1) \cap \tau \neq \omega(i+1) \cap \omega(i+2)$, then $\omega(i) \vdash \tau \rightarrow \omega(i+1)$ is a proper extension of ω (cf. Fig. 6(b)).
  - If $\omega(i+1) \cap \tau = \omega(i+1) \cap \omega(i+2)$, then $\omega(i+1) \curvearrowright \omega(i+2)$. In that case, $\omega(i-1) \rightarrow \omega(i+1) \vdash \omega(i) \vdash \tau \rightarrow \omega(i+2)$ is a proper extension of ω (cf. Fig. 6(c)).

Case IV: $\boxed{\omega(i-1) \curvearrowright \omega(i) \curvearrowright \omega(i+1)}$

Figure 7 illustrates the modifications necessary if $\omega_{\mathrm{M}-\tau}$ enters and leaves t through a vertex.

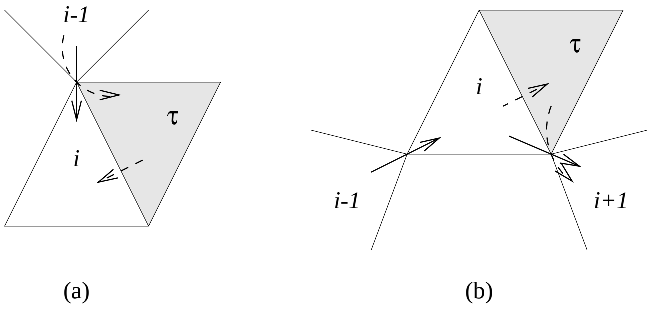

Figure 7: Walk entering and exiting the triangle adjacent to τ via a vertex.

- If $\omega(i-1) \cap \omega(i) \subset \omega(i) \cap \tau$, then $\omega(i-1) \rightarrow \tau \vdash \omega(i)$ is a proper extension of ω (cf. Fig. 7(a)).
- If $\omega(i-1) \cap \omega(i) \notin \omega(i) \cap \tau$, then $\omega(i) \vdash \tau \rightarrow \omega(i+1)$ is a proper extension of ω (cf. Fig. 7(b)).

Since we have been able to properly extend $\omega_{\mathrm{M}_{-\tau}}$ in all cases, our proposition is proved. $\square$

3.1 A Closer Look onto Proposition 3.1

Proposition 3.1 tackles the plain existence of proper self-avoiding walks only. It ensures the solvability of an "initial value problem". In this subsection, we briefly address the question of solvability of the corresponding "boundary value problem".

According to the terminology in algebraic topology [18, 21] we formulate the following definition.

Definition 3.1. *Let $\mathfrak{M}$ be a triangular mesh. A sequence of $n \leq \#\mathfrak{M}$ triangles $(t_i)_{1\leq i\leq n}$ is called a* <u>chain</u> $\Longleftrightarrow_{\text{def}}$

$$\forall i,j \in \{1,\ldots,n\} \quad i \neq j \Longrightarrow t_i \neq t_j \tag{7}$$
$$\forall i \in \{1,\ldots,n-1\} \quad t_i \cap t_{i+1} \textit{ consists of an edge} \tag{8}$$

A chain "connects" its initial and final triangles. Figure 8 shows an example of a chain.

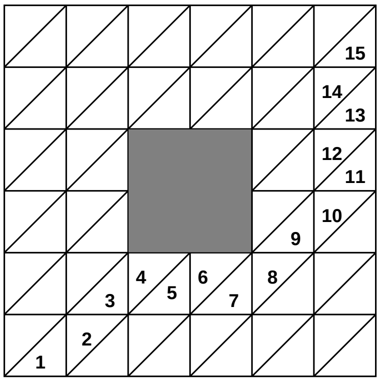

Figure 8: A chain.

Proposition 3.2. *For an arbitrary triangular mesh[4] there exists a chain between any two triangles $A, \Omega \in \mathfrak{M}$.*

Proof: (a.a.) Assume that there are triangles τ_1 and τ_2 in the mesh $\mathfrak{M}$ which cannot be connected by a chain. Denote by $C(\tau_i),\ i = 1,2$ the set of triangles that can be connected with τ_i by a chain. It follows from our assumption that there holds

$$C(\tau_1) \cap C(\tau_2) = \emptyset. \tag{9}$$

[4]Note that according to our definition a mesh is always *connected.*

For two triangles $t_1 \in C(\tau_1)$ and $t_2 \in C(\tau_2)$ either

$$t_1 \cap t_2 = \emptyset \tag{10}$$

or

$$t_1 \cap t_2 \text{ consists of exactly one vertex.} \tag{11}$$

If there holds

$$\forall t \in C(\tau_1) \forall t' \in C(\tau_2) \quad t \cap t' = \emptyset \tag{12}$$

we have a contradiction with the connectedness of our mesh.

If there are triangles $t_1 \in C(\tau_1)$ and $t_2 \in C(\tau_2)$ whose intersection consists of exactly one vertex v, then the underlying topological space cannot be at manifold at this point (cf. [18, 21]).

Consequently, our assumption that there are triangles in the mesh which cannot be connected by a chain must be wrong. This proves our proposition. □

Although proposition 3.2 guarantees the existence of chains between two arbitrary triangles of a mesh, it does not tell us anything about how to construct one. Using coloring techniques and list ranking, this can be easily accomplished.

Chains are "incomplete" proper self-avoiding walks. If we want to construct a proper self-avoiding walk with prescribed initial and final triangles we first construct a corresponding chain. With the help of the rules provided in the proof of proposition 3.1 we can extend it then to a proper self-avoiding walk.

Corollary 3.1. *Let $\mathfrak{M}$ a triangular mesh and $A, \Omega \in \mathfrak{M}$ two arbitrary triangles. There exists a proper self-avoiding walk ω with $\omega(1) = A$ and $\omega(\#\mathfrak{M}) = \Omega$.* □

4 The Basic Algorithm

The proof of proposition 3.1 provides an *algorithm* for the construction of proper self-avoiding walks: starting from an arbitrary triangle of $\mathfrak{M}$, choose new triangles sharing an edge with the current "subcomplex" and extend the existing proper self-avoiding walk over the new triangle. This algorithm can be represented recursively. Figure 9 shows the pseudo-code for the basic function, called **EXTEND_WALK**. The algorithm begins with the selection of an arbitrary triangle from the mesh. After removing this triangle from the mesh and inserting it (as the first triangle) into the walk, the function **EXTEND_WALK** is called with P being the *position* of the first triangle in the walk.[5]

[5] By a *position* in a walk, we mean a reference to a triangle in the walk rather than an index. We need a data structure with fast insertion; therefore, an array is not appropriate. In our C++ implementation, a position is given by `list<Triangle>::iterator`.

```
EXTEND_WALK(M, W, P)
mesh M, walk W, position of a triangle in walk P;
{
    while (triangle T remains in M sharing an edge with the triangle at P) {
        insert T into W according to the rules discussed in the proof of proposition 3.1;
        remove T from M;
        set new position Q to position of T in W;
        EXTEND_WALK(M, W, Q);
    }
}
```

Figure 9: The basic function.

The complexity of this algorithm is $\mathbf{O}(n \cdot \log(n))$, where n is the number of triangles in the mesh. This complexity is obtained as follows. Since we organize our triangles in a red-black tree [6], a search in this structure is of complexity $\mathbf{O}(\log(n))$.[6] The edge–triangle incidence relation can also be organized in a red-black tree. Since we have to perform this search for each triangle in our mesh, the overall complexity of our algorithm is $\mathbf{O}(n \cdot \log(n))$. Note that trading space for time (store the edge–triangle incidence relation with each edge [3]), this can be improved to $\mathbf{O}(n)$.

Note that the algorithm described above can be improved in many ways. For example, we completely ignored additional information like nodal degrees. Such information might be useful for the design of walks with special properties.

Remark: Given the existence of proper self-avoiding walks, another approach, as an acceleration technique, might be favorable. For example, self-avoiding walks have a long tradition in the application of Monte Carlo methods [2, 12] to study long-chain polymer molecules. A discussion of this approach is beyond the focus and the scope of this report.

4.1 Results

We implemented a sequential algorithm for the generation of a proper self-avoiding walk using the `set` and `map` containers from the C++ Standard Template Library [24] which is part of the ANSI–C++ standard [25]. Both containers are implemented with red-black trees.[7] The results presented in Tables 1 and 2 were obtained for some sample meshes on a 110 MHz microSPARC II using GNU's `g++` compiler (version 2.7.2.3) with the `-O` flag. Note that the times in the tables below include the setup time for the mesh and the edge-triangle incidence structure.

[6] This is the complexity of the fastest known search algorithms.
[7] One might also think about an implementation using hash tables.

triangles	*edges*	*time* (s)
2048	3136	0.30
4096	6240	0.57
8192	12416	1.31
16384	24768	2.71
32768	49408	5.54
65536	98688	11.82
131072	197120	24.07

Table 1: Runtimes for the basic algorithm using red-black trees for different mesh sizes.

The results in Table 1 indicate the dependence of the runtime on the number of triangles (or edges). According to our description of the algorithm the runtime may also depend on the triangle/edge ratio. The results in Table 2 explore this possibility.

triangles	*edges*	*ratio*	*time* (s)
65536	131073	0.500	12.85
65536	114690	0.571	12.27
65536	106500	0.615	12.21
65536	102408	0.640	12.15
65536	100368	0.653	12.13
65536	99360	0.660	12.02
65536	98880	0.663	11.99
65536	98688	0.664	11.82

Table 2: Runtimes for the basic algorithm using red-black trees for different triangle/edge ratios.

The numbers in Table 2 clearly show that the dependence of the execution time on the triangle/edge ratio is negligible and confirm our complexity estimate for the whole algorithm.

5 Optimization and Parallelization

A standard parallelization of the deterministic algorithm would be based on the observation that if the proper extensions of a self-avoiding walk for two triangles do not interfere with each other, then they can be done in *parallel*. This general strategy is followed, e.g., in parallel constrained Delaunay meshing [5]. Such an approach is feasible in dealing with single meshes, when no additional structural information is available, or with families of meshes without any hierarchical structure, although this procedure might be computationally expensive in connection with adaption.

For *hierarchical* meshes a completely different approach is possible: the general algorithm has to be applied only once to the initial mesh and this "coarse" self-avoiding walk can be *reused* after adaption. On the local level, we are going to exploit the regularity of the refinement rules. This leads us to the consideration of *constrained* self-avoiding walks. Local regularity allows us to prove the existence of solutions for the underlying "boundary value problem". (In that sense, the algorithm derived from the proof of proposition 3.1 guarantees the solvability of "initial value problems" only.)

5.1 Unstructured Mesh Adaption

Unstructured grids for solving computational problems have two major advantages over structured grids. First, unstructured meshes enable efficient and automatic grid generation for problems with complex geometries or those with dynamically moving boundaries. Second, the ability to dynamically adapt an unstructured grid is a powerful tool for efficiently solving computational problems with evolving physical features. By redistributing the available mesh points to capture phenomena of interest, such adaptive procedures provide a robust, reliable, and efficient alternative to conventional fixed-size numerical techniques which may fail to resolve fine-scale phenomena, incur excessive computational costs, and produce incorrect results.

Two types of mesh adaption strategies are commonly used with unstructured grids. In the first strategy, the grid is either globally or locally regenerated with a higher concentration of points in the high-error regions and with a lower concentration of points in the low-error regions. Although the resulting grids are usually well-formed with smooth transitions between regions of coarse and fine mesh resolution, such schemes are computationally intensive. This is a major drawback for time-dependent problems where the mesh must be adapted every few time steps. The second strategy involves local modifications of the existing grid by individually adding points to the mesh where the error indicator is high, and removing points from regions where the indicator is low. The advantage of this strategy is that relatively few mesh points need to be added (deleted) at each *refinement* (*coarsening*) step. This makes it particularly attractive for unsteady problems where the solution usually changes rapidly. However, complicated hierarchical data structures must be maintained to keep track of the evolving mesh and simplify the coarsening procedure.

In two-dimensional FE meshes, triangular elements are the most popular. A triangle can be refined in different ways: by bisecting one or more edges, or by adding additional points in the interior. However, the most most popular strategy of subdividing a triangle targeted for refinement is by bisecting all three of its edges to form four congruent triangles as shown in Figure 10. This type of subdivision is called *isotropic* (or 1:4) and the resulting triangles are referred to as being *red*. A problem is that the refined mesh will be *nonconforming* unless all the triangles are isotropically subdivided. To get a consistent triangulation without global refinement, a second type of subdivision is allowed. For example, a triangle may be subdivided into two smaller triangles by bisecting only one edge as shown in Figure 10. This type of subdivision is called *anisotropic* (or 1:2 in this case) and the resulting triangles are referred to as being *green*. The process of creating a consistent triangulation is defined as a *closure* operation. Note that several iterations may be necessary to achieve closure [3].

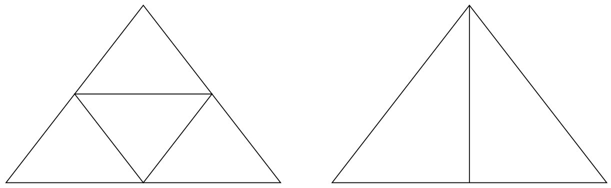

Figure 10: The left picture shows the isotropic subdivision of a triangle. The right one gives an example of anisotropic subdivision.

One major problem with the anisotropic refinement of triangular elements is that repeated subdivision can lead to poor *mesh quality*. Poor mesh quality is defined as a grid deficiency that leads to inaccurate numerical solutions. In order to establish dynamic unstructured mesh adaption as a powerful computational tool, a technique is needed to ensure that the mesh does not deteriorate after several levels of refinement. A common *a priori* strategy to control the quality of a refined mesh is to always subdivide the longer edges (if any) of every triangle that have an edge marked for subdivision. A popular *a posteriori* technique to improve the quality of a mesh is to perform *edge swapping*.

Since both these techniques have certain limitations, we follow a strategy similar to that described in [4] for tetrahedral meshes. *For the remainder of this report, we consider the case of hierarchical refinement and coarsening only.* It is assumed that a given initial mesh always has acceptable element quality. A couple of additional rules are then applied, primarily to assure that the quality of the adapted mesh does not deteriorate drastically with repeated refinement:

1. All triangles with exactly two bisected edges have their third edge also bisected. Thus, such triangles are isotropically refined.

2. A green triangle cannot be further subdivided. Instead, the previous subdivision is discarded and isotropic subdivision is applied to the (red) ancestor triangle.

Fortunately, these additional constraints also simplify the indexing scheme that is described in the next section.

5.2 An Indexing Technique for Hierarchical Meshes

It is the task of an *index scheme* to properly name or label the various objects (vertices, edges, triangles) of a mesh. We prefer the term index scheme instead of *numbering* to stress that the use of natural numbers as indices is not sufficient to meet the naming requirements of the FE objects on parallel architectures.

We give here a brief description of our indexing technique for the sake of completeness; for a detailed discussion, refer to [8]. Note that our technique is intended to be used for hierarchical meshes only.

Our index scheme is a combination of *coarse* and *local* schemes. The coarse scheme labels the objects of the coarse mesh in such a way that the incidence relations can be easily derived from the labels. The vertices are enumerated starting from 1. For the example in Figure 11,

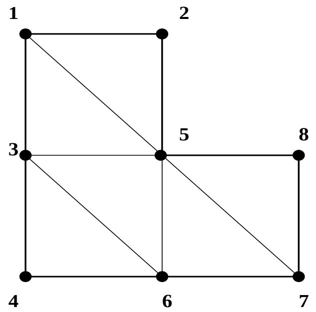

Figure 11: An L-shaped domain and its coarse triangulation.

the set of vertices for the coarse triangulation consists of the following numbers:

$$\text{vertices} = \{1,\ 2,\ 3,\ 4,\ 5,\ 6,\ 7,\ 8\}.$$

The edges of the coarse triangulation are indexed by ordered pairs of integers that correspond to the endpoints of the edges. The ordering is chosen so that the first index is less than the second one. For the example in Figure 11, the set of coarse edges consists of the following pairs:

$$\begin{aligned}\text{edges} = \{&(1,2),\ (1,3),\ (1,5),\ (2,5),\ (3,4),\ (3,5),\ (3,6),\\ &(4,6),\ (5,6),\ (5,7),\ (5,8),\ (6,7),\ (7,8)\}.\end{aligned}$$

The same principles are applied to index the coarse triangles. They are denoted by the triple consisting of their vertex numbers in ascending order. Thus, the set of coarse triangles reads:

$$\text{triangles} = \{(1,2,5),\ (1,3,5),\ (3,4,6),\ (3,5,6),\ (5,6,7),\ (5,7,8)\}.$$

Note that this index scheme can be applied to elements with curved boundaries as well.

The ideal model for local considerations is given by the two-dimensional standard simplex σ^2 (cf. Figure 12):

$$\sigma^2 := \{(x_1, x_2, x_3) \in \mathbb{R}^3 \mid x_1 + x_2 + x_3 = 1\}. \tag{13}$$

The local scheme exploits the regularity (and the finiteness) of the refinement rules to produce names for the objects at subsequent refinement levels [8]. We use (scaled) natural coordinates as indices in the local scheme. Again, this is done in a way such that incidence

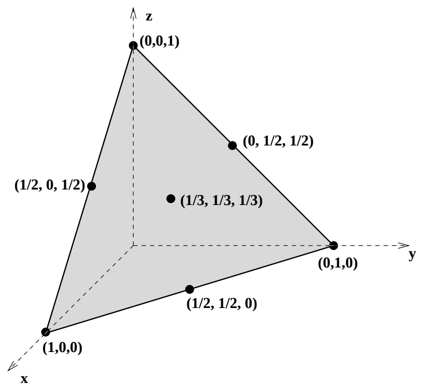

Figure 12: The two-dimensional standard simplex and some points on it.

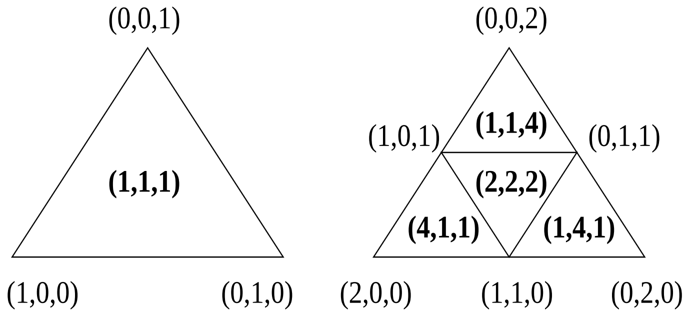

Figure 13: Examples of the local index scheme for triangular elements. The vertices and triangles are denoted by integer triples (triangles by bold face).

relations and the refinement level are *encoded* in the indices of the objects (cf. Figure 13). For example, the set of vertices at level k in the local model is given by:

$$V_k(\sigma^2) := \{(a, b, c) \in \mathbb{N}^3 \mid a + b + c = 2^k\}. \tag{14}$$

Obviously, there holds $v \in V_k(\sigma^2) \iff 2v \in V_{k+1}(\sigma^2)$ and therefore we have the following embedding $V_k(\sigma^2) \subset V_{k+1}(\sigma^2)$. This shows that we can easily move between refinement levels by rescaling.

Denote by $E_k(\sigma^2)$ the set of edges at level k. We choose the integer triple corresponding to the midpoint of an edge as its index. Hence $E_k(\sigma^2) \simeq V_{k+1}(\sigma^2) - V_k(\sigma^2)$ and

$$E^{(k)} = E^{(k)}(\sigma^2) := \bigsqcup_{i=0}^{k} E_i(\sigma^2) \Longrightarrow E^{(k)} \simeq V_{k+1}(\sigma^2) - V_0(\sigma^2). \tag{15}$$

Denote by $T_k(\sigma^2)$ the set of triangles on level k (red and green) and let

$$T^{(k)} = T^{(k)}(\sigma^2) := \bigsqcup_{i=0}^{k} T_i(\sigma^2). \tag{16}$$

We choose the integer triples corresponding to the barycenter of a triangle as its index.

Figure 13 shows the local indices for the vertices and the triangles on the first two refinement levels (for isotropic subdivision).

The coarse and local schemes are combined by taking the union of the Cartesian products of the coarse mesh objects with their corresponding local schemes. Ambiguities are resolved by using a *normal* form of the index.

The key features of such a scheme are:

- Each object is assigned a *global name* that is independent of any architectural considerations or implementation choices.
- Combinatorial information is translated into simple arithmetic.
- It is *well-behaved* under (adaptive) refinement. No artificial synchronization/serialization is introduced.
- It can be extended (with appropriate modifications) to three dimensions [8].

5.3 Constrained Self-Avoiding Walks

We call a subset $T \subset T^{(k)}(\sigma^2)$ a *local refinement* of σ^2 if it can be obtained by applying the refinement rules described in section 5.1.[8]

The *refinement level of a coarse triangle* is defined to be zero. For an arbitrary triangle, the refinement level is defined as the successor of the refinement level of the *parent triangle* (the triangle it was created from by subdivision). The *refinement level of a triangulation* is defined to be maximum of the refinement levels of its triangles [8]. We denote the level of a triangulation T by $l(T)$.

The scheme developed in the previous section allows us, for the case of hierarchical refinement, to decompose our considerations into global and local cases. Here "global" means "related to the initial (coarse) mesh". "Local" means "restricted to a triangle of the coarse mesh". Given a walk over and a local refinement of the coarse mesh one might ask whether it is possible to extend the walk over the triangles of the local refinement. In addition, we would like to decouple the considerations for different coarse triangles. This leads naturally to what we call *constrained* self-avoiding walks: a walk over the initial mesh leaves a kind of a trace or footprint on each coarse triangle which is then translated into a "boundary value problem" for the extension of the self-avoiding walk.

[8] Since we allow coarsening, the term "local refinement" is somewhat misleading. However, any local refinement obtained by a combination of coarsening and refinement can be obtained by refinement only, discarding the *history* of the adaption. This assumes that coarsening did not occur beyond the initial mesh.

In this section we prove the the existence of constrained proper self-avoiding walks. The regularity of the local picture allows us to do so. We start with the definition of a *constraint*.

Definition 5.1. *A pair* $(\alpha, \beta) \in V_1(\sigma^2) \times V_1(\sigma^2)$ *is called* <u>constraint</u> *on* $\sigma^2 \Longleftrightarrow_{\text{def}} \alpha \neq \beta$.

Remark: The previous definition states that we forbid constraints requiring that a walk enters and leaves a triangle through the same edge or the same vertex. (The latter would imply a forbidden jump in the "coarse" walk.) In the definition, we implicitly used the embedding $V_0(\sigma^2) \subset V_1(\sigma^2)$ and the isomorphism $E_0(\sigma^2) \simeq V_1(\sigma^2) - V_0(\sigma^2)$. Figure 14 shows a few examples of constraints.[9]

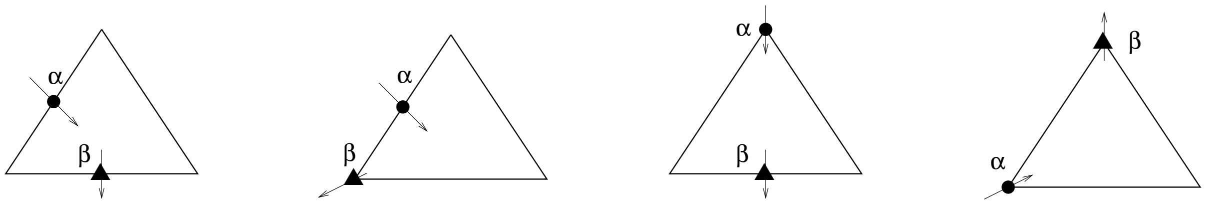

Figure 14: A few examples for constraints.

Definition 5.2. *Let* $c = (\alpha, \beta)$ *be a constraint on* σ^2 *and* $T \subset T^{(k)}(\sigma^2)$ *a local refinement of* σ^2. *A proper self-avoiding walk* $\omega : \{1, \dots, \#T\} \to T$ *is called* <u>compatible</u> *with* $c \Longleftrightarrow_{\text{def}}$

$$\alpha \subset \omega(1) \text{ and } \beta \subset \omega(\#T). \tag{17}$$

The following can be easily shown:

Lemma 5.1. *Let* $c = (\alpha, \beta)$ *be an arbitrary constraint on* σ^2. *Let* $R, G \in T_1(\sigma^2)$ *be a red and green triangle, respectively. There exist proper self-avoiding walks on* R *and* G *which are compatible with* c. $\square$

Remark: Lemma 5.1 is valid not only for constraints on σ^2 and $R, G \in T_1(\sigma^2)$, but also for constraints on red and green triangles at any level, since the proof of lemma 5.1 depends only on the fact whether a triangle is red or green.

Obviously, the formulation of constraints is not restricted to coarse triangles, i.e., triangles at level zero. A slightly more general definition than definition 5.1 goes as follows:

Definition 5.3. *Let* $k \in \mathbb{N}, k > 0$. *A pair* $(\alpha, \beta) \in V_k(\sigma^2) \times V_k(\sigma^2)$ *is called* <u>constraint</u> *on* $t \in T_{k-1}(\sigma^2) \Longleftrightarrow_{\text{def}} \alpha \neq \beta$.

Proposition 5.1. *Let* $c = (\alpha, \beta)$ *be an arbitrary constraint on* σ^2 *and* $T \subset T^{(k)}(\sigma^2)$ *be a local refinement of* σ^2. *There exists a proper self-avoiding walk* $\omega_{T,c}$ *on* T *which is compatible with* c.

[9] The *entry* constraints are denoted with bullets and the *exit* constraints with filled triangles.

Proof: We prove proposition 5.1 by induction over the level $l(T)$ of T. Lemma 5.1 is the starting point for our induction ($l(T) = 1$).

Assume now that $l(T) = n + 1$. Let $T' = T - (T \cap T_{n+1}(\sigma^2))$. T' is a local refinement of σ^2 and $l(T') = n$. This technique is known as *hierarchical coarsening* [9] (cf. Figure 15).

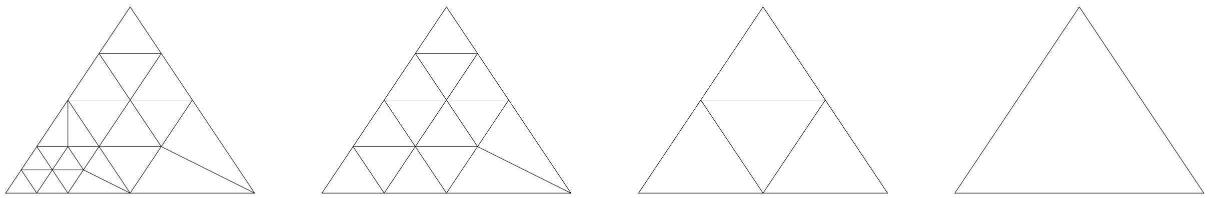

Figure 15: Four steps of hierarchical coarsening.

By our induction assumption, there exists a proper self-avoiding walk $\omega_{T',c}$ which is compatible with c. We show that there is a proper extension $\omega_{T,c}$ of $\omega_{T',c}$.

T can be obtained from T' by local refinement. Let τ be a triangle of T' which must be subdivided (red or green) in order to obtain T. $\omega_{T',c}$ *induces* a constraint on the children of τ or, in other words, the constraint on τ is *propagated* to τ's children by $\omega_{T',c}$. This situation is illustrated in Figure 16. Figure 16(a) shows a local refinement T and the given constraint (α, β). Figure 16(b) shows one step of hierarchical coarsening where we already have a proper self-avoiding walk $\omega_{T',c}$. This self-avoiding walk induces constraints on the higher-level triangles as shown in Figure 16(c)).

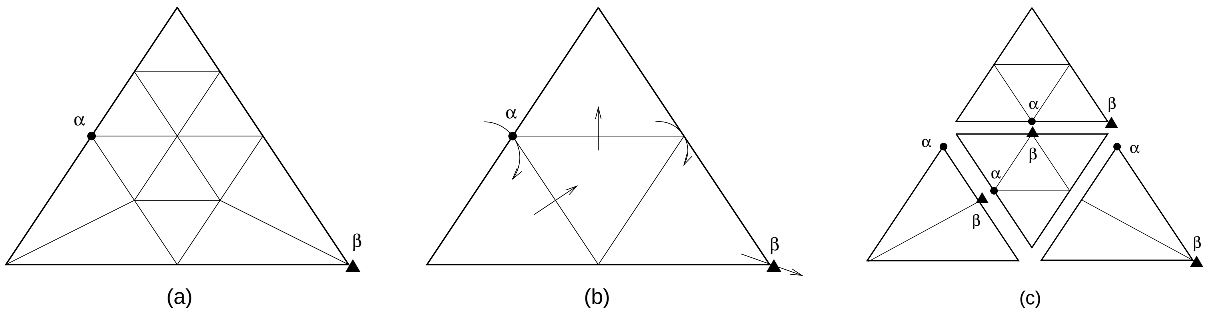

Figure 16: Two steps of constraint propagation.

Since τ will be refined red or green, we can apply lemma 5.1 (and the remark thereafter) to properly extend $\omega_{T',c}$ over the children of τ.

If we apply this procedure to all triangles $t \in T'$ which must be subdivided to get to T, we obtain $\omega_{T,c}$.

This proves our proposition. $\square$

Figure 17 shows an example for the correct extension of a proper self-avoiding walk to a higher level of refinement. Figure 17(a) shows the *mechanical* way dictated by the constraints.

This walk can often be improved in the sense that certain jumps over vertices can be replaced by jumps over edges since the triangles under consideration share an edge (cf. Figure 17(b)).

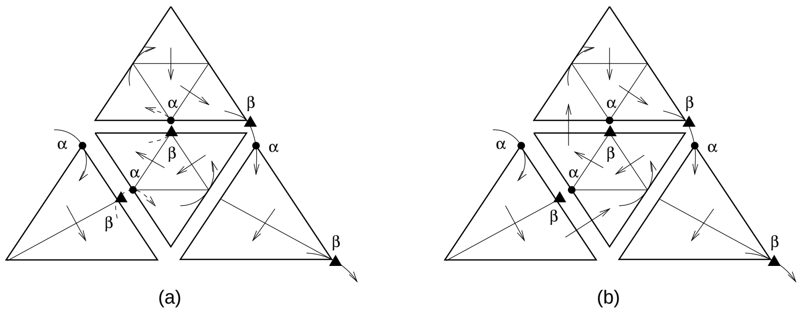

Figure 17: Extension over a higher level of refinement.

Given a constraint on σ^2 and a local refinement $T \subset T^{(k)}(\sigma^2)$, proposition 5.1 guarantees the existence of a compatible proper self-avoiding walk. The proof of proposition 5.1 also provides an algorithm for the construction of the solution of this "boundary value problem". The synthesis of propositions 3.1 and 5.1 provides a simple parallel algorithm which requires changing an existing proper self-avoiding walk only where the mesh has been adapted (coarsened/refined). If the mesh adaption affected only levels higher than k, the walk has to be modified only over triangles whose level is greater than k. Note that the local adaptions of the walk are decoupled from one another and can be done independently; thus, this is a truly parallel algorithm. This work will be the subject of a forthcoming report.

6 Conclusions and Perspectives

In this report, we have developed a theoretical basis for the study of self-avoiding walks over two-dimensional adaptive unstructured grids. We described an algorithm for the construction of proper self-avoiding walks over arbitrary unstructured meshes and determined its complexity to be $\mathbf{O}(n \cdot \log(n))$. This was confirmed by a sequential implementation whose results we report. We discussed the issues for a parallelization of this algorithm and suggested a significant improvement for hierarchical adaptive unstructured grids. For this situation, we have also been able to prove the existence of constrained proper self-avoiding walks.

Obviously, the results obtained thus far allow a straightforward generalization to three-dimensional meshes. However, since there is more *freedom* in three-space, there is some flexibility in the generalization of these results.

The application to *load balancing* is likewise rather straightforward. However, modifications to improve *locality* in the sense of a generalized *frontal method* requires a more detailed study of the relationships between (more specialized) constraints on walks and cache memory behavior.

A Basic Concepts of Combinatorial Topology

Definition A.1. *A* complex $\mathfrak{C}$ *consists of a set of elements, a mapping* $d : \mathfrak{C} \to \mathbb{N}$ *(dimension), and a binary relation* $<$ *(face) on the elements of* $\mathfrak{C}$ *satisfying the following conditions:*

K.1. *If* $A_1, A_2 \in \mathfrak{C}$ *and* $A_1 < A_2$*, then* $d(A_1) < d(A_2)$.

K.2. *The relation* $<$ *is transitive.*

K.3. *If* $A_1 < A_3$ *and* $d(A_3) - d(A_1) > 1$*, then there exists a* $A_2 \in \mathfrak{C}$ *such that* $A_1 < A_2$ *and* $A_2 < A_3$.

The elements of $\mathfrak{C}$ sometimes are called *cells*. $A_1, A_2 \in \mathfrak{C}$ are called *incident*, iff either $A_1 < A_2$ or $A_2 < A_1$.

A complex $\mathfrak{C}$ is called n*-dimensional*, iff it does not contain cells of dimension greater than n and each cell of smaller dimension is incident with at least one n-dimensional cell.

A *pseudomanifold* is a n-dimensional complex in which each $(n-1)$-dimensional cell is incident with at most two n-dimensional cells. A pseudomanifold is called *closed*, iff each $(n-1)$-dimensional cell is incident with exactly two n-dimensional cells.

The *d-dimensional standard simplex* σ^d is given by

$$\sigma^d := \{(x_1, \dots, x_{d+1}) \in \mathbb{R}^{d+1} \mid x_1 + \cdots + x_{d+1} = 1\}. \tag{18}$$

(Note that σ^d lives in $\mathbb{R}^{d+1}$.)

A d-dimensional cell A^d is called *simplicial*, iff its boundary complex is isomorphic to the boundary complex $\partial\sigma^d$ of the d-dimensional standard simplex σ^d.

Definition A.2. *A* simplicial complex $\mathfrak{S}$ *is a complex, whose cells are simplicial and any two cells have different boundary complexes.*

The first example in Figure 18 shows a complex which consists of two one-dimensional and two zero-dimensional cells. This complex is a closed one-dimensional pseudomanifold and is not simplicial. The second complex shown in Figure 18 is a closed simplicial pseudomanifold. Note that in both examples, the underlying topological space (S^1) is a manifold.

References

[1] M. Benantar, U. Dogrusoz, J.E. Flaherty, M.S. Krishnamoorthy: Triangle graphs, *Appl. Numer. Math.*, 17(2), 1995, pp. 85–96.

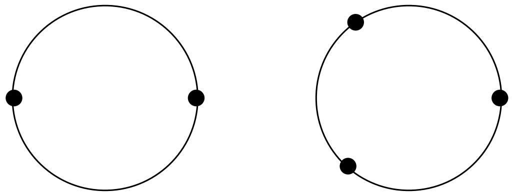

Figure 18: A non-simplicial and a simplicial complex.

[2] K. Binder, D.W. Heermann: *Monte Carlo Simulation in Statistical Physics*, Springer-Verlag, Berlin, 1997.

[3] R. Biswas, R.C. Strawn: A new procedure for dynamic adaption of three-dimensional unstructured grids, *Appl. Numer. Math.*, 13(6), 1994, pp. 437–452.

[4] R. Biswas, R.C. Strawn: Mesh quality control for multiply-refined tetrahedral grids, *Appl. Numer. Math.*, 20(4), 1996, pp. 337–348.

[5] L.P. Chew, N. Chrisochoides, F. Sukup: Parallel constrained Delaunay meshing, *ASME/ASCE/SES Summer Mtg: Special Symp. on Trends in Unstructured Mesh Generation*, 1997.

[6] C.H. Cormen, C.E. Leiserson, R.L. Rivest: *Introduction to Algorithms*, MIT Press, Cambridge, 1990.

[7] U. Dogrusoz, M.S. Krishnamoorthy: Hamiltonian Cycle problem for triangle graphs, *CS Tech. Rep.*, TR 95-7, Rensselaer Polytechnic Institute, Troy, 1995.

[8] J. Gerlach, G. Heber: Fundamentals of natural indexing for simplex finite elements in two and three dimensions, *RWCP Tech. Rep.*, TR 97-008, Tsukuba Research Center, Tsukuba-shi, Japan, 1997.

[9] J. Gerlach: Application of natural indexing to adaptive multilevel methods for linear triangular elements, *RWCP Tech. Rep.*, TR 97-010, Tsukuba Research Center, Tsukuba-shi, Japan, 1997.

[10] L.C. Glaser: *Geometrical Combinatorial Topology I & II*, Van Nordstrand Reinhold Company, New York, 1970.

[11] M. Griebel, G. Zumbusch: Hash-storage techniques for adaptive multilevel solvers and their domain decomposition parallelization, *AMS Contemporary Math. Series*, 218, 1998, pp. 279–286.

[12] J.M. Hammersley, D.C. Handscomb: *Monte Carlo Methods*, Methuen & Co Ltd., London, 1964.

[13] X. Liu, G. Schrack: Encoding and decoding the Hilbert order, *Software - Prac. and Exp.*, 26(12), 1996, pp. 1335–1346.

[14] C.-W. Ou, S. Ranka, G. Fox: Fast and parallel mapping algorithms for irregular problems, *J. of Supercomputing*, 10(2), 1995, pp. 119–140.

[15] C.-W. Ou, M. Gunwani, S. Ranka: Architecture-independent locality-improving transformations of computational graphs embedded in k-dimensions, *9th Intl. Conf. on Supercomputing*, 1995, pp. 289–298.

[16] M. Parashar, J.C. Browne: On partitioning dynamic adaptive grid hierarchies, *29th Hawaii Intl. Conf. on System Sciences*, 1996, pp. 604–613.

[17] J. R. Pilkington, S. B. Baden: Dynamic partitioning of non-uniform structured workloads with spacefilling curves, *IEEE Transactions on Parallel and Distributed Systems*, 7(3), March 1996, pp. 288–300.

[18] K. Reidemeister: *Topologie der Polyeder und kombinatorische Topologie der Komplexe*, Akademische Verlagsgesellschaft M.B.H, Leipzig, Germany, 1938.

[19] J. Salmon, M.S. Warren, G.S. Winckelmans: Fast parallel tree codes for gravitational and fluid dynamical N-body problems, *Intl. J. of Supercomputer Appl.*, 8(2), 1994, pp. 129–142.

[20] J. Salmon, M.S. Warren: Parallel, out-of-core methods for fast evaluation of long-range interactions, *8th SIAM Conf. on Parallel Processing for Scientific Computing*, 1997.

[21] H. Seifert, W. Threlfall: *A Textbook of Topology*, Academic Press, 1980.

[22] B. Szabo, I. Babuska: *Finite Element Analysis*, John Wiley, New York, 1991.

[23] M.S. Warren, J.K. Salmon: A parallel hashed oct-tree N-body algorithm, *Supercomputing*, 1993, pp. 12–21.

[24] `http://www.sgi.com/Technology/STL/`

[25] `http://nw.demon.co.uk/ocsltd/c++/`