PREPRINT 2006:21

Optimal Search in Finite Element Triangulations Using Binary Trees

ERIK D. SVENSSON

Department of Mathematical Sciences
Division of Mathematics
CHALMERS UNIVERSITY OF TECHNOLOGY
GÖTEBORG UNIVERSITY
Göteborg Sweden 2006

Preprint 2006:21

Optimal Search in Finite Element Triangulations Using Binary Trees

Erik D. Svensson

CHALMERS | GÖTEBORG UNIVERSITY

Department of Mathematical Sciences
Division of Mathematics
Chalmers University of Technology and Göteborg University
SE-412 96 Göteborg, Sweden
Göteborg, August 2006

Preprint 2006:21
ISSN 1652-9715

Matematiska vetenskaper
Göteborg 2006

OPTIMAL SEARCH IN FINITE ELEMENT TRIANGULATIONS USING BINARY TREES

ERIK D. SVENSSON

ABSTRACT. We propose a simple algorithm that, given the set S of all n-simplices, $n = 2, 3$, in a finite element triangulation and a query point $p \in \mathbf{R}^n$ will find one n-simplex or $\mathbf{R}^n \setminus S$ containing p in $O(\log N)$ search time, where N is the number of n-simplices in the triangulation. The algorithm requires $O(N \log N)$ preprocessing time and $O(N)$ storage. We apply the algorithm on two finite element triangulations and demonstrate that the search time is of the same order as the time to evaluate the barycentric coordinates of one n-simplex, which we regard a relevant time scale in many finite element applications.

1. INTRODUCTION

Given the set S of all n-simplices, $n = 2, 3$, in a finite element triangulation and a query point $p \in \mathbf{R}^n$ we pose the following search problem: *Does any n-simplex in S contain p?* This problem relates to two fundamental problems in computational geometry: the *planar subdivision search* problem, that is, given a planar subdivision in $\mathbf{R}^2$ with a number of line segments, determine which region in subdivision contains p; or the *post-office* problem, that is, given a set of points, find the point that is closest to p, see [6] and references there in. There are many solutions and suggestions how to solve the subdivision search problem, for example [5, 4, 2, 7, 6]. With N denoting the number of n-simplices in S we characterize, cf. [4], a solution or algorithm to the posed search problem by: (1) *preprocessing time* -the time to construct search structures, (2) *space* -the storage used by the method, (3) *search time* -the time required to locate the region or point in S. It is possible to solve the search problem in an optimal way,

Date: April 18, 2006.

2000 *Mathematics Subject Classification.* 68U05.

Key words and phrases. computational geometry, point location, finite element, post-processing.

that is, with $O(N)$ preprocessing time, $O(N)$ space and $O(\log N)$ search time [4, 6]. However, these methods are often considered too complicated [2] and although the search time scales linearly the constant in the linear dependence, the *query constant*, may be large [6].

In this work we propose a simple algorithm to solve the posed search problem that is characterized as optimal in search time and space and requires $O(N \log N)$ preprocessing time.

2. Preliminaries

We first introduce a few concepts used in finite element practice and theory, see for example [1].

Let A_i for $i = 1, \dots, n+1$ be scalars not all equal to zero. A *hyperplane* π is subspace of $\mathbf{R}^n$ such that

$$\pi = \Big\{ x \in \mathbf{R}^n : \sum_{i=1}^{n} A_i x_i + A_{n+1} = 0 \Big\} \tag{2.1}$$

An *n-simplex* in $\mathbf{R}^n$ is the convex hull T of $n+1$ points $a_1, \dots, a_{n+1}$, called vertices, not all contained in a hyperplane, that is, for $n = 0, \dots, 3$: a point, a line segment, a triangle, or a tetrahedron. For $0 \le m \le n$, an *m-face* of the n-simplex T is an m-simplex whose vertices are also vertices of T.

Let $\Omega \subset \mathbf{R}^n$ be a polyhedral domain. A *triangulation* $\mathcal{T}$ is a partition of Ω into n-simplices T such that no vertex of any simplex lies in the interior of any m-face, for $1 < m < n$. A family of triangulations $\{\mathcal{T}_h\}_{h>0}$ is said to be shape-regular if there is a $\gamma > 0$ such that $h_T/\rho_T \le \gamma$ for all $T \in \bigcup_h \mathcal{T}_h$, where $h_T = \max_{T \in \mathcal{T}_h} \operatorname{diam}(T)$ and $\rho_T = \sup\{\operatorname{diam}(S) : S \text{ is a ball contained in } T\}$.

An *n-rectangle* is a set of the form

$$R = \prod_{i=1}^{n} [a_i, b_i] = \{x = (x_1, \dots, x_n) : \, a_i \le x_i \le b_i, \, 1 \le i \le n\}. \tag{2.2}$$

Again, let S be the set of all n-simplices, $n = 2, 3$, in a finite element triangulation $\mathcal{T}$ and set $\operatorname{card}(S) := N$, that is, we use the cardinal number to count the number of n-simplices in S.

In the complexity analysis we use a parameter N to measure the size of the search problem. We may interchangeably take N as the number of m-simplices, $0 \le m \le n$, in the triangulation. In $\mathbf{R}^2$ this is solely due to the Euler relations whereas in $\mathbf{R}^3$ we will have to impose additional constraints on the triangulation.

Consider a triangulation $\mathcal{T}$, where we now assume that Ω and $\partial\Omega$ are simply connected which will only influence the Euler relations that we will use. Let N_i for $m = 0, \dots, n$ be the number of m-simplices in the the triangulation and let N_m^∂ for $m = 0, \dots, n-1$ be the number of m-simplices on the boundary of the triangulation.

We first consider $n = 2$. By counting the edges and triangles in the triangulation we get the identity $2N_1 - N_1^\partial = 3N_2$ and since $0 \leq N_1^\partial \leq N_1$ we may estimate

$$\frac{3}{2}N_2 \leq N_1 \leq 3N_2,$$

which shows that the number of edges and triangles are interchangeable. Inserting this into the Euler relation for triangulations in $\mathbf{R}^2$, see for example [3],

$$N_0 - N_1 + N_2 = 1,$$

we get

$$1 + \frac{1}{2}N_2 \leq N_0 \leq 1 + 2N_2,$$

which shows that the number of vertices and triangles are interchangeable.

Consider next $n = 3$. By counting the faces and tetrahedra in the triangulation we get the identity

$$2N_2 - N_2^\partial = 4N_3 \tag{2.3}$$

and since $0 \leq N_2^\partial \leq N_2$ we may estimate

$$2N_3 \leq N_2 \leq 4N_3,$$

which shows that the number of faces and tetrahedra are interchangeable.

By counting the edges and tetrahedra in the triangulation we obtain $\sum_{i=1}^{N_1} a_i = 6N_3$, where $a_i = \text{card}(\{T \in \mathcal{T} : E_i \cap T = E_i\})$ is the number of tetrahedra neighboring the edge E_i. Hence

$$\bar{a}N_1 = 6N_3, \tag{2.4}$$

where $\bar{a} = N_1^{-1}\sum_{i=1}^{N_1} a_i$ is the average of $\{a_i\}_{i=1}^{N_1}$, which shows that the number of edges and tetrahedra are interchangeable.

Also by counting the edges and faces on the boundary we get the identity $2N_1^\partial = 3N_2^\partial$ which together with the Euler relation on the boundary $N_0^\partial -$

$N_1^\partial + N_2^\partial = 2$ implies that $N_2^\partial = 2(N_0^\partial - 2)$ and with (2.3) we get the identity

$$2N_2 = 4N_3 + 2(N_0^\partial - 2). \tag{2.5}$$

Inserting (2.4) and (2.5) into the Euler relation for triangulations in $\mathbf{R}^3$

$$N_0 - N_1 + N_2 - N_3 = 1,$$

we get

$$N_0 + N_0^\partial = \Big(\frac{6}{\bar{a}} - 1\Big)N_3,$$

and since $0 \leq N_0^\partial \leq N_0$ we may estimate

$$\frac{1}{2}\Big(\frac{6}{\bar{a}} - 1\Big)N_3 + \frac{3}{2} \leq N_0 \leq \Big(\frac{6}{\bar{a}} - 1\Big)N_3 + 3,$$

which shows that the number of vertices and tetrahedra are interchangeable.

Notice that we will have to impose $\bar{a} < 6$ in order to have a use full estimate. This is often true in practice since the triangulation is generated with a shape-regularity constraint. We also remark that we may try to use the uniform estimate

$$N_1 \min_{i=1,\dots,N_1} a_i \leq \sum_{i=1}^{N_1} a_i \leq N_1 \max_{i=1,\dots,N_1} a_i,$$

in the analysis above but in practice this will often be useless since instead of imposing $\bar{a} < 6$ we will have to impose $\max_{i=1,\dots,N_1} a_i < 6$ which is not likely to be true in practice –there are always a few edges where the condition fails. The mean value $\bar{a}$ is a milder condition better suited in this situation.

In conclusion, we just showed that provided $\bar{a} < 6$, we could use either $N = N_i$, for $i = 0, \dots, n$ in the complexity analysis. We will use this fact without further notice throughout this work.

Finally, we will use the notion *binary tree* denoting a data structure devised for fast data searching [8]. The binary tree contains a number of items called *nodes* of the tree. Each node contains data and zero or two links connecting to other nodes in the tree. The first node in the tree is called the *root*. A connected set of nodes of the tree is called a *subtree* and a node that has no connections to other nodes is called a *leaf*. See the illustrations in Figure 2.1. The *height* of the tree is equal to the maximal number of nodes connecting the root and any leaf.

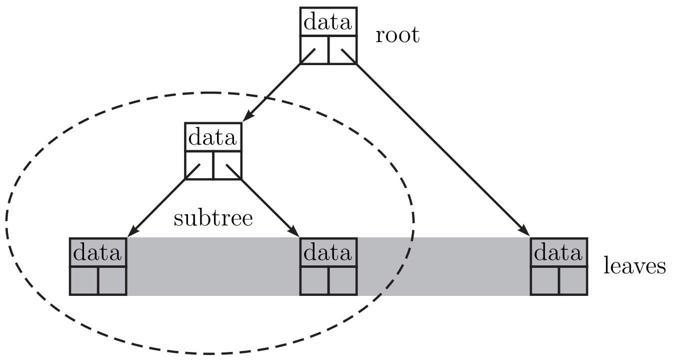

Figure 2.1: Binary tree.

3. Binary search in triangulations

We devise a binary tree that will be used to find the n-simplex containing the query point p, or no n-simplex if p is not in Ω. Each node in the tree will contain numbers A_i for $i = 1, \ldots, n+1$ representing a hyperplane π, and the subtrees *negSubtree* and *posSubtree*, also binary trees that are parts of the entire tree. The hyperplanes will partition $\mathbf{R}^n$ into negative and positive sides that will be used to sort the n-simplices in the triangulation at preprocessing. As a result of this sorting, every leaf will contain a set of n-simplices S_l where ideally S_l is such that $\operatorname{card}(S_l) = 1$ or at least close to 1.

Given a query point p we use the search Algorithm 1 to find 0 or 1 n-simplex in the triangulation containing the point, 0 meaning that p is outside the Ω. In the algorithm we use a generic algorithm `inSimplex(`T, p`)` to test whether T contains p and we refer to Algorithms 4 or 5 in the Appendix for details.

In the sections below we describe two algorithms for constructing the binary tree. Both algorithms however suffer from different deficiencies and it is only after combining them in a new one we obtain an algorithm that will be useful in practice.

3.1. Partitioning along x_i-hyperplanes.

Let R be the smallest n-rectangle containing Ω. For a_i and b_i as in (2.2), defining the n-rectangle, set $a = (a_1, \ldots, a_n)$ and

$$dx = (dx_1, \ldots, dx_n) = (b_1 - a_1, \ldots, b_n - a_n).$$

Algorithm 1: `findSimplex`(point p, binary tree *binaryTree*)

```
Input: point $p$, binary tree $binaryTree$
Output: $n$-simplex T or 0 (no $n$-simplex)

if no subtrees then                                  /* at a leaf */
    forall $T \in S_l$ do                         /* linear search */
        if inSimplex($T, p$) then        /* see Algorithm 4 or 5 */
            return $T$
    return 0                          /* no $n$-simplex was found  */
else                                           /* choose a subtree */
    if $\sum_i^n A_i p_i + A_{n+1} < 0$ then
        return findSimplex($p, negSubtree$)
    else
        return findSimplex($p, posSubtree$)
```

Find the largest side of R, and set $i = \text{argmax}_{i=1,\dots,n}(dx_i)$ and let π denote the hyperplane with $A_i = 1$ and $A_{n+1} = -a_i - dx_i/2$ ($A_j = 0$ for $j \neq i$ and $j < n+1$). Partition R along the hyper plane π into to n-rectangles R_- and R_+. Sort the n-simplices $T \in S$, where we recall that S is the set of all n-simplices in the triangulation, now also contained in R. Add T to S_- if $T \cap R_- \neq \emptyset$ and add T to S_+ if $T \cap R_+ \neq \emptyset$. Repeat this procedure recursively for the pairs (R_-, S_-) and (R_+, S_+) until $\text{card}\,(S) < 2$ or $\text{card}\,(S) = \text{card}\,(S_-)$ or $\text{card}\,(S) = \text{card}\,(S_+)$. We summarize this procedure in Algorithm 2.

The height of the tree is $\sim \log N$ and each recursive step in the preprocessing requires sorting $\sim N$ n-simplices. Hence, the preprocessing time for the binary tree is $O(N \log N)$.

The search time will require $O(\log N)$ operations, but the query constant will be rather large since at the leafs a linear search is preformed. The number of simplices in S_l will be roughly bounded by the number of n-simplices neighboring a node in the triangulation, in practice this is ~ 10 for $n = 2$ and ~ 40 for $n = 3$. This will slow down the search and due to this Algorithm 2 is not a good choice in practice.

3.2. **Partitioning along $(n-1)$-faces.** Recall that S is the set of all n-simplices in $\mathcal{T}$ and that N_{n-1} is the number of $(n-1)$-faces in the triangulation. Let π_i for $i = 1, \dots, N_{n-1}$ be the hyperplanes defined by the $(n-1)$-faces in the triangulation. Denote the halfspaces on opposite sides

Algorithm 2: `binaryTreeRectangular`(S, a, dx)

```
Input: a set $S$ of $n$-simplices, $a$ and $dx$ defining an $n$-rectangle $R$
Output: binary tree data structure
Data: the binaryTreeRectangular contain numbers $A_i$ for
      $i = 1, \dots, n+1$ representing the hyperplane $\pi$, subtrees
      negSubtree and posSubtree, and a set $S_l$ of $n$-simplices.

$A_i = 0$ for $i = 1, \dots, n+1$                      /* initialization */
if $card(S) < 2$ then                                        /* if leaf */
    $S_l = S$
    return this binaryTreeRectangular
else
    $i = \text{argmax}_{i=1,\dots,n}(dx_i)$
    $dx_i = dx_i/2$
    $A_i = 1$
    $A_{n+1} = -a_i - dx_i$
    forall $T \in S$ do                               /* sort simplices */
        if $\sum_{i=1}^{n} A_i a_j + A_{n+1} < 0$ for one vertex $a_j \in T$ then
            add $T$ to $S_-$
        if $\sum_{i=1}^{n} A_i a_j + A_{n+1} > 0$ for one vertex $a_j \in T$ then
            add $T$ to $S_+$
    if $card(S) > card(S_-)$ and $card(S) > card(S_+)$ then        /* new
    subtrees */
        negSubtree = binaryTreeRectangular($S_-, a, dx$)
        $a_i = a_i + dx_i$
        posSubtree = binaryTreeRectangular($S_+, a, dx$)
    else                                                        /* leaf */
        $S_l = S$
        return this binaryTreeRectangular
```

of π_i by $\mathbf{R}^n_{i,-}$ and $\mathbf{R}^n_{i,+}$. Now sort the simplices $T \in S$ and add T to $S_{i,-}$ if $T \cap \mathbf{R}^n_{i,-} \neq \emptyset$ and add T to $S_{i,+}$ if $T \cap \mathbf{R}^n_{i,+} \neq \emptyset$. Choose one of these hyperplanes $\pi = \pi_i$ such that

$$i = \text{argmax}_{i=1,\dots,N_{n-1}} \begin{cases} \text{card}\,(S_{i,-})/\text{card}\,(S_{i,+}) & \text{if card}\,(S_{i,-}) < \text{card}\,(S_{i,+}), \\ \text{card}\,(S_{i,+})/\text{card}\,(S_{i,-}) & \text{otherwise,} \end{cases}$$

and set $S_- = S_{i,-}$ and $S_+ = S_{i,+}$. Repeat the procedure recursively for S_- and S_+ until card $(S) < 2$ or card $(S) =$ card (S_-) or card $(S) =$ card (S_+).

This procedure creates a binary tree and we summarize it in Algorithm 3.

The height of the tree is $\sim \log N$ and each recursive step in the preprocessing requires sorting $\sim N^2$ n-simplices. Hence, the preprocessing time for the binary tree is $O(N^2 \log N)$, which is far from optimal.

The search time will require $O(\log N)$ operations and the query constant will be rather good. Also, in this situation, a linear search is performed at the leafs. However, in this case the number of simplices in S_l will be small, mostly 1 and with small and rare variations. We have not made any attempts to give a rigorous upper bound for the number of simplices in S_l.

Due to the scaling of the preprocessing time this algorithm is not a good choice in practice, at least not for large triangulations.

3.3. `binaryTreeRectangular` **and** `binaryTreeFace` **combined.** We notice that the deficiencies in Algorithms 2 and 3 are complementary, small preprocessing time and large search time for Algorithm 2 but large preprocessing time and small search time for Algorithm 3. In other words it seems desirable to combine the algorithms in such way that only the favorable characteristics of the algorithms remain and cancel the deficiencies. The idea is to let Algorithm 3 continue where Algorithm 2 is terminated. We input $S = S_l$ from Algorithm 2 into Algorithm 3 and let it refine the tree further. In this way we will gain a binary tree with good query constant since the $\text{card}(S_l)$ after Algorithm 3 has terminated will be small, and since Algorithm 3 is only applied on small sets S from Algorithm 2 it will not have major impact on the total preprocessing time.

If we assume that there are $\sim N$ different leaves in the tree after Algorithm 2 has terminated and that each such leaf holds M n-simplices then the total preprocessing time will be the preprocessing time for Algorithm 2 plus the preprocessing time for Algorithm 3 applied on N sets each holding M n-simplices, that is, the total preprocessing time will scale like $O(N \log N + NM^2 \log M)$ which is close to $O(N \log N)$ for small M and large N.

Note that we may also try to apply Algorithm 3 on a smaller set $S_s \subset S$ (S outputted from Algorithm 2), chosen by some means, which will improve the preprocessing time at the expense of the search time. For example we may take S_s to be the n-simplex whose barycenter is closest to the center of mass of all barycenters of all n-simplices in S. Then $\text{card}\,(S_s) = 1$ and the total complexity will be $O(N \log N)$. This will alter $\text{card}\,(S_l)$ at the final leafs, when Algorithm 3 has terminated, and card (S_l) will be larger but still relatively small when compared to `binaryTreeRectangular`.

4. Numerical examples

We now consider two triangulations, one in $\mathbf{R}^2$, Figure 4.1, and the other in $\mathbf{R}^3$, Figure 4.2. We build the search structure proposed in Section 3.3 and measure: the preprocessing time and the average search time for 10^6 randomly chosen query points as function of number of nodes N in the triangulations as we perform 4 and 3 uniform refinements in the $\mathbf{R}^2$ and $\mathbf{R}^3$ triangulations, respectively.

The preprocessing time is normalized with the preprocessing time for the triangulations at start and the search time is normalized with the time it takes evaluating the barycentric coordinates for one n-simplex, see the Appendix where we account for the implementation used. The motivation for the normalization of the search time is to find a time scale appropriate for finite element applications. For example, it is often necessary to evaluate the barycentric coordinates when post-processing finite element data.

In Figures 4.3 and 4.4 we visualize the search process in the two dimensional triangulation. We search for a query point contained in the shaded triangle in Figure 4.3 and marked with the bullet $\bullet$ in Figure 4.4. We also plot the the hyperplanes π (lines) used to partition the triangles in the triangulation. After 12 levels in the binary tree the triangle containing the query point could be identified. There are 9 layers from Algorithm 2 and 3 layers from Algorithm 3.

Finally we plot the results from the measurements in Figures 4.5 and 4.6, where we also make a least square data fit to the appropriate scaling, $O(N \log N)$ for the preprocessing time and $O(\log N)$ for the search.

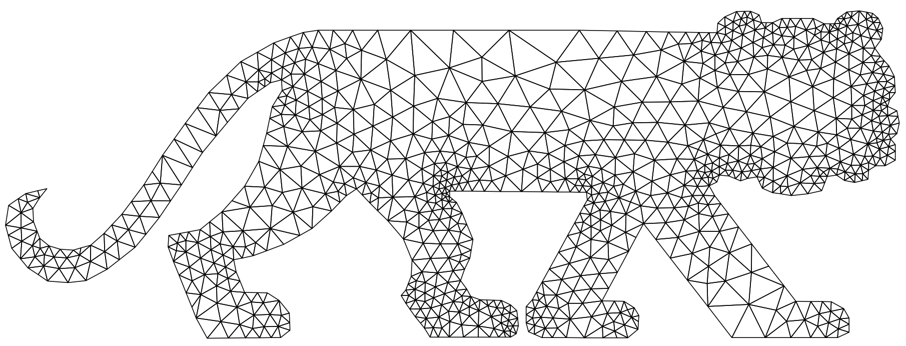

Figure 4.1: A two-dimensional triangulation with 940 nodes and 1572 triangles.

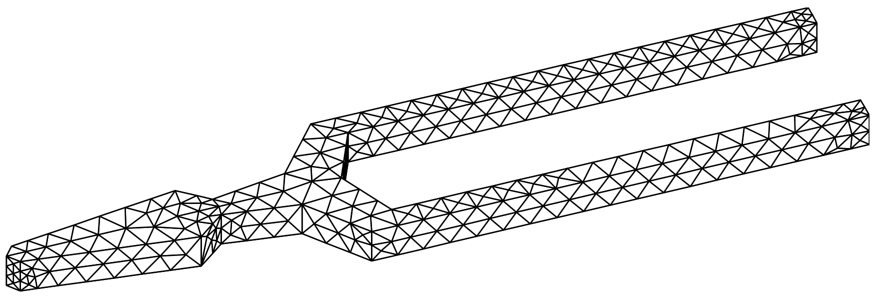

Figure 4.2: A three-dimensional triangulation with 578 nodes and 1567 tetrahedra.

Algorithm 3: `binaryTreeFace(`S`)`

```
Input: a set $S$ of $n$-simplices
Output: binary tree data structure
Data: the binaryTreeFace contain numbers $A_i$ for $i = 1, \dots, n+1$
      representing the hyperplane $\pi$, subtrees negSubtree and
      posSubtree, and a set $S_l$ of $n$-simplices.

$A_i = 0$ for $i = 1, \dots, n+1$                          /* initialize */
if $card(S) < 2$ then                                        /* if leaf */
    $S_l = S$
    return this binaryTreeFace
else
    r = 0.0      /* parameter do decide the best partition */
    /* Let $\pi_i$ with scalars $B_i$ be the hyper planes defined
    by the $N_{n-1}$ $(n-1)$-faces in $S$.                            */
    forall $\pi_i$ do
        forall $T \in S$ do                              /* sort simplices */
            if $\sum_{i=1}^{n} B_i a_j + B_{n+1} < 0$ for one vertex $a_j \in T$ then
                add $T$ to $S_{i,-}$
            if $\sum_{i=1}^{n} B_i a_j + B_{n+1} > 0$ for one vertex $a_j \in T$ then
                add $T$ to $S_{i,+}$
        if $card(S_{i,-}) \leq card(S_{i,+})$ and $r < card(S_{i,-})/card(S_{i,+})$ then
            $r = \text{card}(S_{i,-})/\text{card}(S_{i,+})$
            $S_- = S_{i,-}$ and $S_+ = S_{i,+}$
            $A_i = B_i$ for $i = 1, \dots, n+1$
        else if $card(S_{i,+}) < card(S_{i,-})$ and $r < card(S_{i,+})/card(S_{i,-})$
        then
            $r = \text{card}(S_{i,+})/\text{card}(S_{i,-})$
            $S_- = S_{i,-}$ and $S_+ = S_{i,+}$
            $A_i = B_i$ for $i = 1, \dots, n+1$
    if $card(S) > card(S_-)$ and $card(S) > card(S_+)$ then         /* new
    subtrees */
        negSubtree = binaryTreeFace($S_-$)
        posSubtree = binaryTreeFace($S_+$)
    else                                                          /* leaf */
        $S_l = S$
        return this binaryTreeFace
```

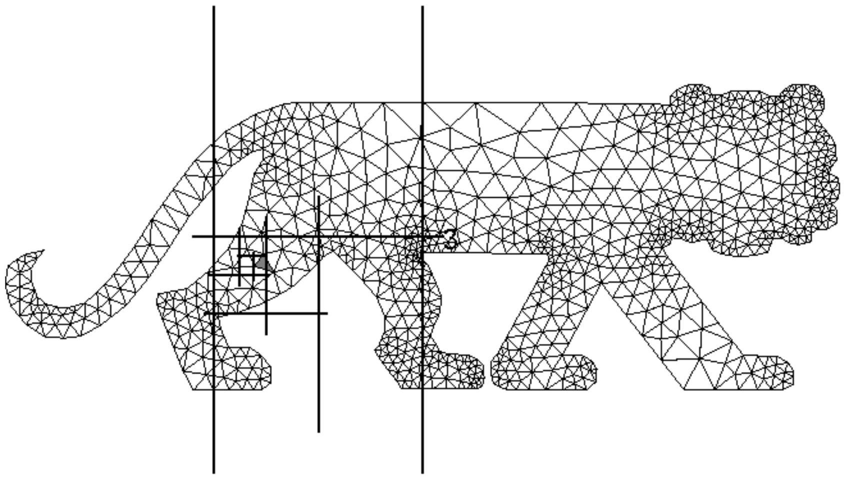

Figure 4.3: Search process using the binary tree with Algorithm 2. We are searching for a query point contained in the shaded triangle in the rear leg of the tiger. The horizontal and vertical lines are the hyperplanes π used to partition the triangles in the triangulation.

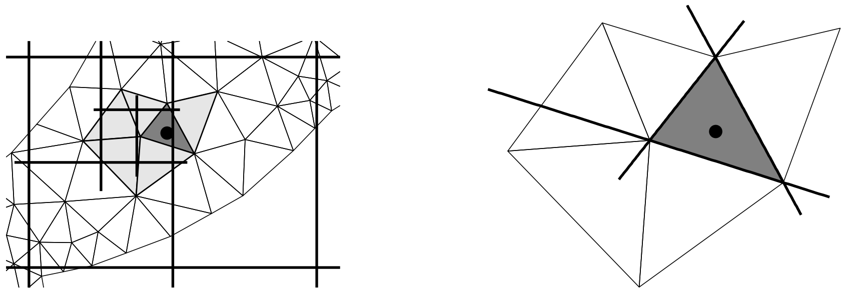

Figure 4.4: The query point is marked with the bullet $\bullet$. **(left)** Search in the tree with Algorithm 2, zoom in. The set of shaded triangles is the set S_l in the leaf from Algorithm 2. **(right)** Search in the tree with Algorithm 3. The algorithm terminates with one triangle in the final leaf.

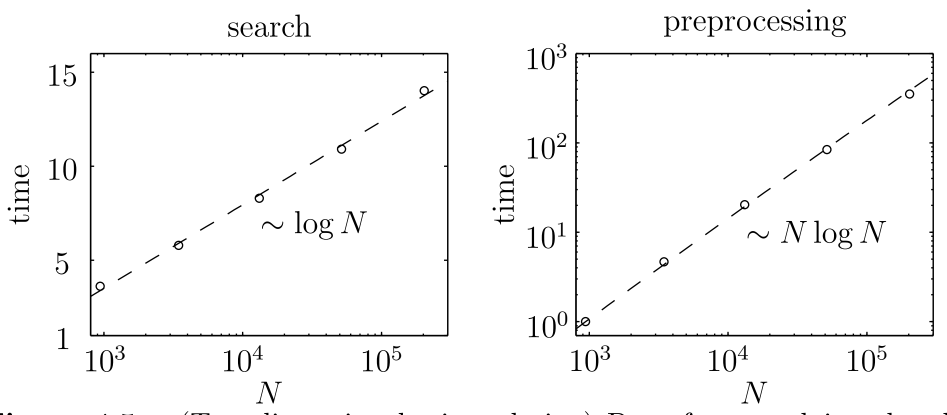

Figure 4.5: (Two-dimensional triangulation) Data from applying the algorithm in Section 3.3 to the triangulation in Figure 4.1. The dashed lines are least square fits. **(left)** Average search time for 10^6 randomly chosen query points. The time is normalized with the time to evaluate the barycentric coordinates for one triangle, which is a characteristic time scale in finite element post-processing. **(right)** Preprocessing time normalized with the preprocessing time of the triangulation at start.

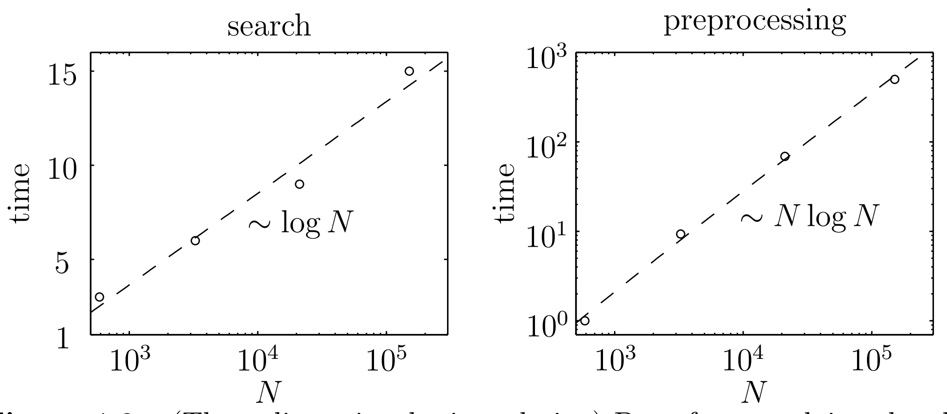

Figure 4.6: (Three-dimensional triangulation) Data from applying the algorithm in Section 3.3 to the triangulation in Figure 4.2. The dashed lines are least square fits. **(left)** Average search time for 10^6 randomly chosen query points. The time is normalized with the time to evaluate the barycentric coordinates for one tetrahedron, which is a characteristic time scale in finite element post-processing. **(right)** Preprocessing time normalized with the preprocessing time of the triangulation at start.

Appendix A. Various simple algorithms for n-simplices

In this appendix we give account for various simple algorithms or mere implementations of mathematical notions that we have used throughout this work on n-simplices with vertices in $a_i = (x_i, y_i)$ for $i = 1, \ldots, 3$ (triangles) or $a_i = (x_i, y_i, z_i)$ for $i = 1, \ldots, 4$ (tetrahedra) as in Figure A.1. We represents barycentric coordinates λ with the $(n+1) \times (n+1)$ matrices $\mathcal{M}$. For $x \in \mathbf{R}^n$ we then get the barycentric coordinate as $\lambda = \mathcal{M}x$.

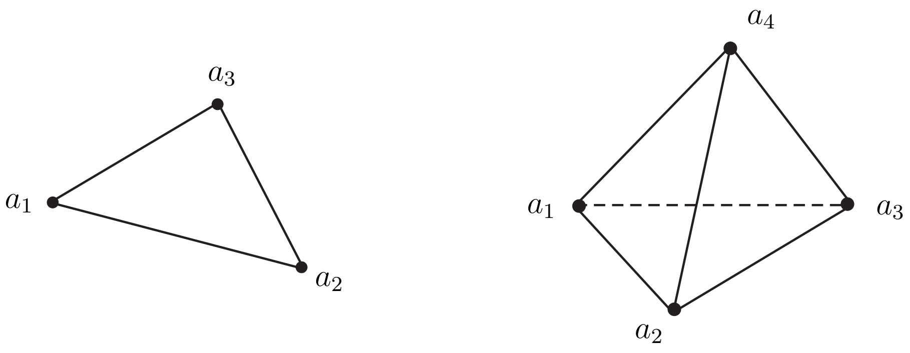

Figure A.1: n-simplices. **(left)** A triangle with vertices a_1, a_2 and a_3. **(right)** A tetrahedron with vertices a_1, a_2, a_3 and a_4.

A.1. **Triangles, $n = 2$.**

A.1.1. *Volume.* We compute the signed volume as the vector product $V(a_1, a_2, a_3) = ((a_2 - a_1) \times (a_3 - a_1))/2$ which in terms of the vertices is

$$V(a_1, a_2, a_3) = \frac{1}{2}\big((x_2 - x_1)(y_3 - y_1) - (x_3 - x_1)(y_2 - y_1)\big),$$

and the volume is $|V(a_1, a_2, a_3)|$.

A.1.2. *Barycentric coordinates.* The matrix $\mathcal{M}$ is the inverse to

$$\begin{pmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ 1 & 1 & 1 \end{pmatrix},$$

cf. [1], and we get

$$\mathcal{M} = V(T)^{-1}\begin{pmatrix} y_2 - y_3 & x_3 - x_2 & x_2y_3 - x_3y_2 \\ y_3 - y_1 & x_1 - x_3 & x_3y_1 - x_1y_3 \\ y_1 - y_2 & x_2 - x_1 & x_1y_2 - x_2y_1 \end{pmatrix},$$

A.1.3. *Point in a triangle.* In order to test whether a point p is contained in a triangle T we test if p and a_3 are on the same side of the line trough a_1 and a_2, and likewise for the other two vertices, cf. [6, Code 1.6, p. 29].

Algorithm 4: `inTriangle`(T, p)

```
Input: triangle $T$, point $p$
Output: true ($p \in T$) or false ($p \notin T$)
$v = V(a_1, a_2, a_3)$
if $v * V(a_1, a_2, p) < 0.0$ then
    return false
if $v * V(a_3, a_1, p) < 0.0$ then
    return false
if $v * V(a_2, a_3, p) < 0.0$ then
    return false
return true
```

A.2. **Tetrahedra,** $n = 3$**.**

A.2.1. *Volume.* We compute the signed volume as the vector triple product $V(a_1, a_2, a_3, a_4) = (a_2 - a_1) \cdot ((a_3 - a_1) \times (a_4 - a_1))/6$ which in terms of the vertices is

$$\begin{aligned} V(a_1, a_2, a_3, a_4) = \frac{1}{6}\big(& - (x_4 - x_1)(y_3 - y_1)(z_2 - z_1) \\ & + (x_3 - x_1)(y_4 - y_1)(z_2 - z_1) \\ & + (x_4 - x_1)(y_2 - y_1)(z_3 - z_1) \\ & - (x_2 - x_1)(y_4 - y_1)(z_3 - z_1) \\ & - (x_3 - x_1)(y_2 - y_1)(z_4 - z_1) \\ & + (x_2 - x_1)(y_3 - y_1)(z_4 - z_1)\big), \end{aligned}$$

and the volume is $|V(a_1, a_2, a_3, a_4)|$.

A.2.2. *Barycentric coordinates.* The matrix $\mathcal{M}$ is the inverse to

$$\begin{pmatrix} x_1 & x_2 & x_3 & x_4 \\ y_1 & y_2 & y_3 & y_4 \\ z_1 & z_2 & z_3 & z_4 \\ 1 & 1 & 1 & 1 \end{pmatrix},$$

cf. [1], and we get

$$\mathcal{M} = V(T)^{-1} \begin{pmatrix} | & | & | & | \\ \mathcal{M}_1 & \mathcal{M}_2 & \mathcal{M}_3 & \mathcal{M}_4 \\ | & | & | & | \end{pmatrix},$$

where

$$\mathcal{M}_1 = \begin{pmatrix} y_4(z_3 - z_2) + y_3(z_2 - z_4) + y_2(z_4 - z_3) \\ y_4(z_1 - z_3) + y_1(z_3 - z_4) + y_3(z_4 - z_1) \\ y_4(z_2 - z_1) + y_2(z_1 - z_4) + y_1(z_4 - z_2) \\ y_3(z_1 - z_2) + y_1(z_2 - z_3) + y_2(z_3 - z_1) \end{pmatrix},$$

$$\mathcal{M}_2 = \begin{pmatrix} x_4(z_2 - z_3) + x_2(z_3 - z_4) + x_3(z_4 - z_2) \\ x_4(z_3 - z_1) + x_3(z_1 - z_4) + x_1(z_4 - z_3) \\ x_4(z_1 - z_2) + x_1(z_2 - z_4) + x_2(z_4 - z_1) \\ x_3(z_2 - z_1) + x_2(z_1 - z_3) + x_1(z_3 - z_2) \end{pmatrix},$$

$$\mathcal{M}_3 = \begin{pmatrix} x_4(y_3 - y_2) + x_3(y_2 - y_4) + x_2(y_4 - y_3) \\ x_4(y_1 - y_3) + x_1(y_3 - y_4) + x_3(y_4 - y_1) \\ x_4(y_2 - y_1) + x_2(y_1 - y_4) + x_1(y_4 - y_2) \\ x_3(y_1 - y_2) + x_1(y_2 - y_3) + x_2(y_3 - y_1) \end{pmatrix},$$

$$\mathcal{M}_4 = \begin{pmatrix} x_4(y_2z_3 - y_3z_2) + x_3(y_4z_2 - y_2z_4) + x_2(y_3z_4 - y_4z_3) \\ x_4(y_3z_1 - y_1z_3) + x_3(y_1z_4 - y_4z_1) + x_1(y_4z_3 - y_3z_4) \\ x_4(y_1z_2 - y_2z_1) + x_2(y_4z_1 - y_1z_4) + x_1(y_2z_4 - y_4z_2) \\ x_3(y_2z_1 - y_1z_2) + x_2(y_1z_3 - y_3z_1) + x_1(y_3z_2 - y_2z_3) \end{pmatrix},$$

A.2.3. *Point in a tetrahedron.* In order to test whether a point p is contained in a tetrahedron T we test if p and a_4 are on the same side of the plane trough a_1, a_2 and a_3, and likewise for the other three vertices, cf. [6, Code 1.6, p. 29].

Algorithm 5: `inTetrahedron`(T, p)

```
Input: tetrahedron $T$, point $p$
Output: true ($p \in T$) or false ($p \notin T$)

$v = V(a_1, a_2, a_3, a_4)$
if $v * V(a_1, a_2, a_3, p) < 0.0$ then
    return false
if $v * V(a_1, a_4, a_2, p) < 0.0$ then
    return false
if $v * V(a_1, a_3, a_4, p) < 0.0$ then
    return false
if $v * V(a_2, a_4, a_3, p) < 0.0$ then
    return false
return true
```

References

[1] P. G. Ciarlet, *Basic error estimates for elliptic problems*, Handbook of Numerical Analysis, Vol. II, North-Holland, 1991.

[2] O. Devillers, S. Pion, and M. Teillaud, *Walking in a triangulation*, Internat. J. Found. Comput. Sci. **13** (2002), 181–199.

[3] A. Ern and J-L. Guermond, *Theory and Practice of Finite Elements*, Springer-Verlag, 2004.

[4] D. Kirkpatrick, *Optimal search in planar subdivisions*, SIAM J. Comput. **12** (1983), 28–35.

[5] R. J. Lipton and R. E. Tarjan, *Applications of a planar separator theorem*, SIAM J. Comput. **9** (1980), 615–627.

[6] J. O'Rourke, *Computational Geometry in C*, second ed., Cambridge University Press, 1998.

[7] N. Sarnak and R. E. Tarjan, *Planar point location using persistent search trees*, Comm. ACM **29** (1986), 669–679.

[8] H. Schildt, *C the Complete Reference*, third ed., McGraw-Hill, 1995.

Department of Mathematical Sciences, Chalmers University of Technology, SE-412 96 Göteborg, Sweden

E-mail address: `erik.svensson@math.chalmers.se`