Exact and efficient interpolation using finite elements shape functions

Gustavo Silva, Rodolphe Le Riche, Jérôme Molimard, Alain Vautrin

Exact and efficient interpolation using finite elements shape functions

G.H.C. Silva, R. Le Riche, J. Molimard and A. Vautrin

École Nationale Supérieure des Mines de Saint-Étienne, France
158, cours Fauriel F-42023 Saint-Étienne cedex 2

January 3, 2007

Abstract

The increasing use of finite elements (FE) and optical full-field measurement methods have contributed to the growing need to compare mesh-based degrees of freedom with experimental fields of sparse data points. Applying generic interpolation algorithms (e.g. using linear, cubic and B-spline weight functions) creates a dependency of the interpolated field to non-physical parameters that may affect high-frequency information through implicit filtering and introduces fundamental assumptions to the shape of experimental data field. The alternative is to use the existing FE mesh and shape functions to determine mesh degrees of freedom at each experimental data point. This comparison technique makes no assumptions beyond those already made in the FE model. In this sense, interpolation using element shape functions is exact. However, this approach requires calculating the local FE coordinates of the experimental points (inverse mapping), which is non-trivial and not a standard part of FE software.

Basic implementations of FE interpolation can be time consuming and impractical for applications requiring the comparison of large fields to be repeated several times. This article analyzes a two-step process for FE interpolation. First the element containing a given data point is determined, then the interpolation inside the element using reference coordinates. An efficient strategy is proposed, which relies on cross-products, element bounding-boxes and a multi-dimensional storage array (virtual mesh). The strategy yields a linear computation cost with respects to the number of elements in the FE mesh and number of data points in the experimental field, contrary to a quadratic cost from standard approaches. A sample application is given using a plate with a hole.

Contents

Summary of notation

$e(p)$	An element containing p
$\epsilon(p)$	A neighborhood around point p
n_i^j	The i^{th} node of set j
$\mathbf{n}^{mesh}$	The set of nodes in a finite elements mesh
$\mathbf{n}^{\epsilon(p)}$	Set of nodes in a neighborhood of p
N_e	Number of elements in a finite element's mesh.
N_p	Number of points in in the data field.
$\overline{N}_{cl}$	Average number bounding-boxes that claim an arbitrary data point.
$\overline{N}_k$	Average number of finite elements in a virtual element.
$\overline{N}_{ve}$	Average number of virtual elements which share a reference to a finite element .
p	A data point
$\mathbf{p}$	A set of data points
STi	Search Technique i.
$T, \hat{T}$	Time measured in CPU seconds and number of operations, respectively.
T_b	Build time or pre-processing time.
T_c	Time complexity.
T_f	Failure time.
T_i	Search time for a data point belonging to the i^{th} element in a sequential list.
T_s	Success time.
T_t	Total search time.
$T_X\|_Y$	Time factor X for search technique Y.
$\vec{v}$	A vector
v_i	The i^{th} scalar component of vector $\vec{v}$
$\vec{x}(p)$	Coordinates of point p in the global coordinate system
$\vec{\xi}(p)$	Coordinates of point p in an element's reference coordinate system

1 Introduction

The increasing use of finite elements has given rise to a series of applications requiring the comparison of discrete data fields with finite elements results. These applications include mesh-interaction [4, 22], geological topography [10], visualization [16, 20], calibration of boundary conditions [17] and material identification [5, 7, 9, 11, 12, 14, 15]. In general, a finite elements solver provides information only at nodal or integration points. Available data points may not always coincide with mesh points, and before a comparison can be performed, it is necessary to map one field of points onto the other.

Three types of mapping techniques can be employed for this purpose: node placement, where FE nodes and experimental points are forced to coincide [7, 14], approximation and interpolation [1] [8, 13, 12]. Since the accuracy of the FE models is a function of node position, node placement techniques introduce an undesired coupling between the position of data points and the accuracy of the finite elements model. Generic interpolation and approximation algorithms (ex. using linear, cubic and B-spline weight functions) creates a dependency of the interpolated field to non-physical parameters that may eliminate high-frequency information through implicit filtering, and introduces fundamental assumptions to the shape of experimental data field. This article presents interpolation techniques using FE shape functions. This approach was selected over other mapping techniques because it makes no assumptions other than those already introduced in the finite elements model. The strategies discussed in this article consist of two parts: First, the element containing each data point (the owner element) is identified. Next, the coordinates of the data point are transformed into the finite element's reference coordinate system (hereafter this procedure is referred to as

[1] Unlike interpolation, approximation points do not necessarily match experimental or FE data points.

inverse mapping). The reference coordinates are used to compute the element's shape-function coefficients which perform the interpolation.

Many of the cited applications require the interpolation of anywhere from 1×10^3 to 1×10^6 data points to be repeated several thousand times [15]. Hence, it is very important that this mapping be carried out as quickly, accurately and reliably as possible. To address this demand, the article analyzes a series of techniques to reduce the number of operations required for interpolation of large data fields. These techniques consist of different search methods for the owner element (including the cross-product and bounding-box tests, and the virtual mesh), and inverse mapping methods (including analytical and iterative methods). We begin with a description of the different search algorithms, followed by a theoretical estimation in computational cost. Since the search time is dependent on the exact mesh configuration, the algorithms are bench-marked using meshes of 2D 4-nodes quadrilateral elements (QUAD4). Using the theoretical estimates, the algorithm with the lowest computational cost is determined. A C++ implementation of these algorithms is used to support the theoretical estimates on a plate with a hole problem.

2 Overview of interpolation techniques

This section presents a brief overview of the possible techniques for comparing discrete data fields with FE results.

2.1 General interpolation

The *griddata* function in the Matlab software package [13] provides four techniques to obtain data, $u(\mathbf{p})$, at arbitrary points, $\mathbf{p}$, from a cloud of sparse data, $u(\mathbf{n}^{mesh})$: triangle-based linear, cubic, nearest neighbor and biharmonic spline (B-spline) interpolation. All of these techniques except B-spline [21] generate interpolation meshes via Delaunay triangulation [3]. A triangle defines a zone of influence for interpolation coefficients W_i,

$$u(p) = \sum_{i=1}^{\text{no. nodes in } \epsilon(p)} W_i \left(\vec{x}(p), \vec{x}(\mathbf{n}^{\epsilon(p)}) \right) u(n_i^{\epsilon(p)}) \; . \tag{1}$$

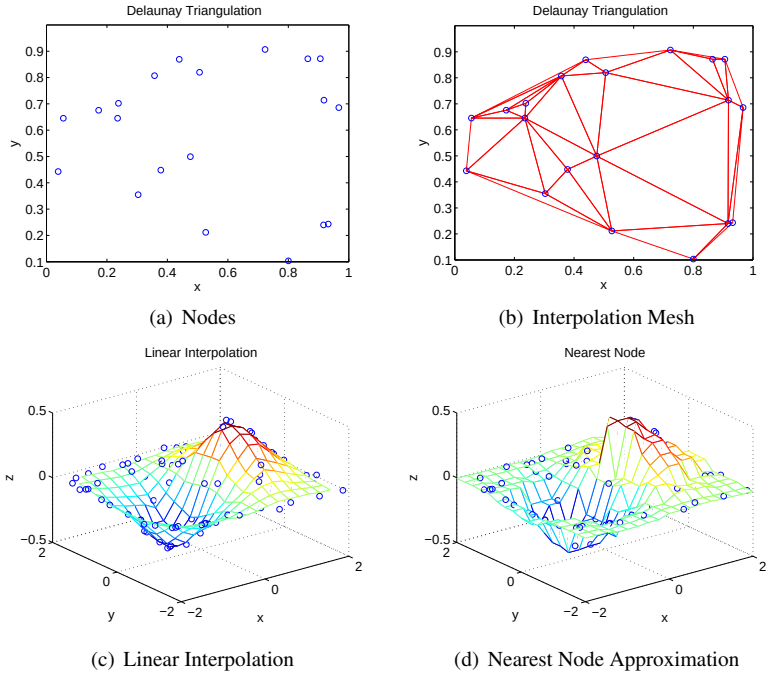
The nearest-node and linear interpolation algorithms are the fastest, but have discontinuous zeroth and first derivatives, respectively. The other two are continuous up to the second derivatives, but are significantly slower. In addition, *griddata* does not account for fields with an internal discontinuity, such as a plate with a hole. The algorithm fills the hole with elements, thus introducing boundary effects on the interpolated data. Kriging [8] is another popular interpolation approach popular in geo-statistics. It has the advantage of providing an estimation of the interpolation variance. However, it is computationally expensive (requiring the solution of a large linear system for each data point). Another drawback of general interpolation is that they affect the data by introducing non-physical parameters to the comparison (i.e., kernel width, polynomial degrees, variogram length and scales in kriging).

2.2 Methods based on finite elements shape functions

With FE shape functions, N_j, the value, $u(p)$, is estimated from node values, $u(\mathbf{n}^{e(p)})$, of the element, $e(p)$, containing the data point, p (hereafter referred to as the owner element),

$$u(p) = \sum_{j=1}^{\text{no. nodes in } e(p)} N_j \left(\vec{\xi}(p) \right) u(n_j^{e(p)}) \; . \tag{2}$$

Notice that in general shape functions are written in the reference system of the element containing point, p. Calculating the local coordinates of a point is not a trivial operation, requiring two steps. First, finding the owner element. Second, calculating the local coordinates of the point (an operation referred to as inverse mapping), which in general involves solving a multi-dimension non-linear system of equations. The comparison of the FE model and data fields is performed either at the coordinates of the FE nodes or data points. Continuity is guaranteed for the zeroth derivative, but is generally discontinuous for higher-order derivatives.



(a) Nodes (b) Interpolation Mesh

(c) Linear Interpolation (d) Nearest Node Approximation

Figure 1: Interpolation using the Matlab *griddata* function.

Projection of data points onto FE response space

E. Pagnacco and D. Lemosse [18] describe a technique where the data field is approximated at the nodal coordinates of the finite elements mesh. The method searches for node values, $u(\mathbf{n}^{mesh})$, that best fit available data, $u'(\mathbf{p})$. The approximation is defined as the projection of the data onto the finite elements model space. The values of $u(\mathbf{n}^{mesh})$ are determined by solving a least-squares problem,

$$\min_{u(\mathbf{n}^{mesh})} \sum_{i=1}^{\text{no. points in } \mathbf{p}} \left[u'(p_i) - \sum_{j=1}^{\text{no. nodes in } \epsilon(p_i)} N_j\left(\vec{\xi}(p_i)\right) u(n_j^{e(p)}) \right]^2 . \tag{3}$$

This technique requires a number of data points greater than the number of finite elements nodes, and that the data field encompass a representative part of the finite elements mesh. The computational cost involves determining the reference coordinates of each data point, and solving the least-squares problem. For applications where the comparison is performed multiple times over a non-changing finite elements mesh, the interpolation of the data points has to be performed only once. This projection method may eliminate (filter) high-frequency information in the measurement (e.g. experimental noise).

FE interpolation at data points

Instead of interpolating measured data points at FE node coordinates, this article selects the interpolation of data from nodes at the coordinates of the experimental points, leaving the experimental data unaltered. The strategy consists of simply evaluating equation (2) for each data point, $p \in \mathbf{p}$. A special effort is made to improve the speed of determining the owner element. This approach allows for the computation of degrees of freedom at arbitrary points, and has no restriction in the distribution or the number of data points. The FE solution is completely independent of the position and size of the experimental data field, and since the same mesh is used for solving the model and interpolating the data, the resulting interpolated field is an exact representation of the FE solution. Also, similar to the previous technique, the interpolation coefficients can be computed only once for a non-changing mesh, thus saving time on applications that repeat the interpolation several times. Different from the previous technique, this approach does not perform implicit filtering, thus allowing for the consideration of high-frequency information.

3 Determination of the owner element

Interpolation using shape functions is accomplished in three steps: the determination of the element containing the data point (owner element), the transformation of the data point's coordinates into the reference coordinate system (inverse mapping), and finally the application of the finite elements shape functions to determine the degrees of freedom at that point (equation (2)). The determination of the owner element is not required to be an elaborated step in the algorithm. The program could attempt sequentially inverse mapping a point for every element in the mesh. Using the reference coordinates the program can check if the point falls inside the domain of the element (section 4). However, since inverse mapping may be complex and numerically expensive, such an approach would be inefficient. Moreover, inverse mapping algorithms are not guaranteed to have a solution for points outside the element's domain, which affects the reliability of a two-step interpolation algorithm. Instead, a combination of simple tests is used to determine the owner element before performing the inverse mapping.

3.1 Element tests

This section describes the cross-product and bounding-box tests. The term "element test" refers to any technique to determine whether a point lies inside or outside of an element.

3.1.1 Cross-product test

The cross-product test (Figure 2(a)) consists in a series of cross and dot products, which determines if a data point lies inside the intersection of the "vertex cones" of an element. The approach is equivalent to techniques described by P. Burke [6]. If point p is inside the element, then for every node, $n \in \mathbf{n}^{e(p)}$, the vector, $\vec{np}$, will lie inside the cone created by the two adjacent node vectors, $\vec{ni}$ and $\vec{nj}$. This condition is checked using the vectors $\vec{s_1}$ and $\vec{s_2}$,

$$\vec{s_1} = \vec{ni} \times \vec{np} \tag{4}$$

$$\text{and } \vec{s_2} = \vec{np} \times \vec{nj} \; . \tag{5}$$

These vectors will point in the same direction for internal points and in opposite directions for external points. Hence, an internal point must satisfy the condition,

$$\vec{s_1} \cdot \vec{s_2} \geq 0 \; , \tag{6}$$

for all nodes in the element. If a point lies outside the element, there is at least one vertex, n, such that

$$\vec{s_1} \cdot \vec{s_2} < 0 \; . \tag{7}$$

The cross-product test requires that a spatially ordered list of nodes be available for each element (this is standard in finite elements mesh formats). This test is exact for elements with linear edges, and approximate otherwise.

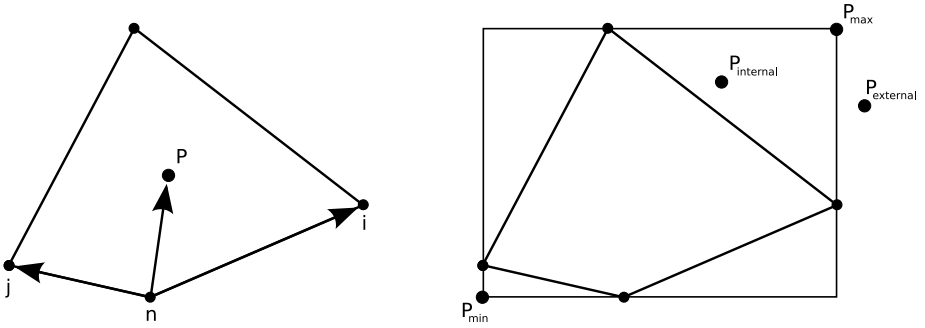
3.1.2 Bounding-box test

The bounding-box test creates an encompassing box around the finite element [19], and compares the coordinates of the data point with the bounding-box's two opposite corner points p_{min} and p_{max} (see Figure 2(b)). The coordinates of any point inside the bounding-box must satisfy the inequality,

$$x_i(p_{min}) \leq x_i(p_{internal}) \leq x_i(p_{max}) \; , \tag{8}$$

for all dimensions, i. At least one of the inequalities will be false for an external point. The complexity of the test includes two parts: computing the boundaries of the bounding-box (later stored in memory), and testing if a data point lies inside the bounding-box. Since the bounding-box only approximates the geometry of a finite element, it is possible for a point to be inside the bounding-box, but outside the finite element ($p_{internal}$ in Figure 2(b)). Thus, in order to identify the owner element, it is necessary for the bounding-box

test to be used together with an exact test such as the cross-product test. Section 3.3, discusses the advantages of scanning a list of finite elements using the bounding-box and cross-product tests together instead of a search algorithm using only the cross-product test. The algorithm first eliminates impossible owner elements with the computationally inexpensive bounding-box test before checking actual owner elements with the more expensive cross-product test.



(a) Cross-Product Test (b) Bounding-Box Test

Figure 2: (a) The cross-product test uses cross and dot products to check if a point lies inside all vertex cones of an element. (b) The bounding-box test uses a rectangular approximation of the element to quickly eliminate impossible owner elements (case of external points).

3.2 Searching the element list

The default implementation for finding an owner element is to sequentially scan through a list of elements. Figure 3(a) shows two data points in a finite elements mesh and their corresponding places in a sequential storage container. If a point lies near the end of the list, then element ownership tests must be performed for a large number of elements in the list. Guessing a good initial point in the list requires knowledge of how the mesh was created, which is not always possible. This section introduces the virtual mesh [19], which is intended to limit the number of elements scanned by element tests. A virtual mesh has two features: First it has a computationally efficient way of determining the sub-region of the mesh that a data point belongs to. The sub-regions are referred to as virtual elements. This is typically obtained through a regular paving of the space. Second, each virtual element, v, stores a list, $\mathbf{e}^v$, of finite elements, e, that possibly share a segment of area,

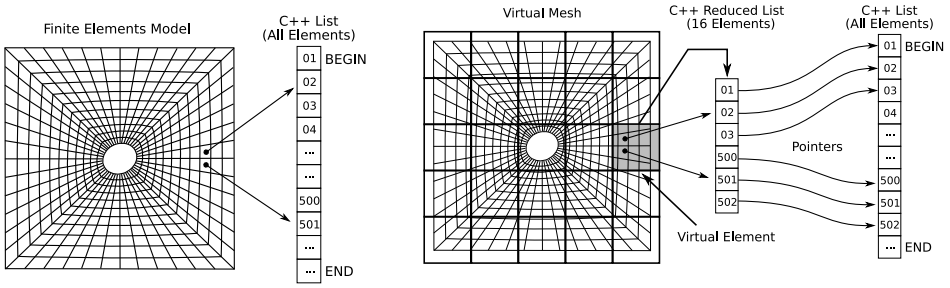
$$\mathbf{e}^v = \left\{ e \in \mathbf{E} \mid e \hat{\cap} v \neq \emptyset \right\} \ , \tag{9}$$

where $\hat{\cap}$ is the operator "possibly intersects". This operator is implemented here by determining the virtual element index range for the two opposite diagonal points, $\gamma_i(p_{min})$ and $\gamma_i(p_{max})$ (see Figure 2(b) and equation (10)). The list, $\mathbf{e}^v$, is typically much smaller than the complete list of finite elements, $\mathbf{E}$.

Figure 3(b) illustrates the simplest virtual mesh, made of regular rectangles in 2D. This virtual mesh is used for the experiments in this article. Retrieving the correct virtual element is very inexpensive. The i^{th} dimension index, γ_i, of the virtual element containing the data point p can is determined using only 3 operations,

$$\gamma_i = floor\left(\frac{x_i(p) - x_i}{dx_i}\right) \ , \tag{10}$$

where x_i is the origin of the virtual mesh grid and dx_i is the virtual mesh's grid-step in the i^{th} direction. Once the virtual element has been retrieved, its small list of finite elements, $\mathbf{e}^v$, is scanned using element tests.



(a) No Virtual Mesh

(b) With Virtual Mesh

Figure 3: Examples of a finite elements mesh and element containers. (a) With a sequential element list it is possible for interpolation points, which are very close in space to be far away in the container. (b) Virtual elements contain small lists of finite elements in their neighborhoods.

3.3 Comparison of owner element search algorithms

This section compares three algorithms to find the owner element of a data point. Each algorithm is defined by a combination of element tests and an element retrieval technique (either sequential or based on a virtual mesh).

- Search technique 1 (*ST1*) sequentially scans the full list of finite elements using the cross-product test. The process is restarted at the beginning of the list for each data point.
- Search technique 2 (*ST2*) is a sequential search of the full finite elements list using a combination of the cross-product and bounding-box tests. This algorithm is essentially the same as the previous technique, except that the cross-product test is performed only if the data point is inside the finite element's bounding box. If the element fails the bounding box test, the algorithm moves to the next element in the list. The idea is to eliminate impossible owner elements with the computationally inexpensive bounding-box test before using the more expensive cross-product test.
- Search technique 3 (*ST3*) uses the cross-product and bounding-box tests to search small lists of finite elements obtained with the virtual mesh. For each data point, the algorithm retrieves the appropriate virtual element, and scans the small list of finite elements using a test similar to ST2.

The efficiency of an element search algorithm is estimated by counting average number of operations. Operations are defined as basic math operations ($+ \ - \ \times \ \div$), comparison operations ($= \neq < > \leqslant \geqslant$), assignment operations and standard mathematical functions in the C library. A finite element program containing a sequential list of N_e elements as illustrated in Figure 3. The search time for a data point belonging to the i^{th} element in this list is

$$\hat{T}_i = (i-1) \times \hat{T}_f + \hat{T}_s \ , \tag{11}$$

where $\hat{T}_s$ (success time) and $\hat{T}_f$ (failure time) are the computation times (measured in number of operations) to determine that the data point does or does not belong to an element, respectively. Assuming for an average field, that a point can belong to any element in the list with equal probability, the time complexity, $\hat{T}_c$, namely the average number of operations to determine the owner element of one point is

$$\hat{T}_c(N_e) = \frac{1}{N_e} \sum_{i=1}^{N_e} \hat{T}_i \ . \tag{12}$$

The total search time, $\hat{T}_t$, is an estimate of the average number of operations required to identify the owner element for N_p arbitrary data points including a preprocessing time, $\hat{T}_b$ (required to initialize bounding boxes and the virtual mesh),

$$\hat{T}_t(N_e, N_p) = \hat{T}_b(N_e) + \hat{T}_c(N_e) \times N_p \ . \tag{13}$$

The objective of the analysis is to determine which combination of element tests results in the smallest total search time, $\hat{T}_t$. $\hat{T}_t$ was estimated for the mentioned search techniques applied to meshes of QUAD4 elements. These search times are functions of the number of elements, N_e, the number of interpolation points, N_p, as well as mesh-specific geometric parameters, $\overline{N}_{cl}$, $\overline{N}_k$ and $\overline{N}_{ve}$. $\overline{N}_{cl}$ is a measure of mesh distortion, specifically the average number of finite element bounding-boxes that will claim an arbitrary data point. $\overline{N}_{ve}$ is the average number of virtual elements possessing a reference to the same finite element. $\overline{N}_k$ is the average number of finite elements in a virtual element. An approximate relationship between $\overline{N}_k$ and $\overline{N}_{ve}$ is developed in Appendix C. It uses a coefficient α, which is the ratio of the average bounding-box area of a finite element, $\overline{A}_e$, to the area of a virtual element, $dx \times dy$,

$$\alpha = \frac{\overline{A}_e}{dx \times dy} \ . \tag{14}$$

The approximation considers the limiting behavior of $\overline{N}_k$ and $\overline{N}_{ve}$ as $\alpha \to 0$ and $\alpha \to \infty$, while accounting for and mesh to virtual mesh offsets (see Figure 14 in Appendix C),

$$\overline{N}_k \approx \frac{1}{\alpha} + 1 \tag{15}$$

$$\text{and } \overline{N}_{ve} \approx \alpha + 1 \ . \tag{16}$$

If α is very large (i.e. very small virtual elements) the majority of the virtual elements will contain the reference to only one finite element, leading to a decrease in the time complexity, $\hat{T}_c$, of the search algorithm. However, since there are more virtual elements, the preprocessing time, $\hat{T}_b$, will offset the advantage gained by the smaller $\hat{T}_c$. α^*, which represents the the best compromise between $\hat{T}_c$ and $\hat{T}_b$ is obtained by differentiating $\hat{T}_t|_{st3}$ with respects to α and solving for a zero (see Appendix C),

$$\alpha^* = \sqrt{\frac{1.25 N_p}{N_e}} \ . \tag{17}$$

Table 1 summarizes the search times of interest, the proofs of which can be found in Appendix A.

Time	Cross-Product Test	Bounding-Box Test	Virtual Mesh
$\hat{T}_b$	0	$20N_e$	$(5N_e + 7) + \left[20 + \overline{N}_{ve} + 12\right] N_e$
$\hat{T}_f$	37.5	2.5	Does Not Apply
$\hat{T}_s$	60	4	Does Not Apply

Search Technique	Computation Time
ST1	$\hat{T}_t\|_{st1} = 18.75 N_e N_p + 41.25 N_p$
ST2	$\hat{T}_t\|_{st2} = 1.25 N_e N_p + 20.75 \overline{N}_{cl} N_p + 110.5 N_p + 20 N_e$
ST3	$\hat{T}_t\|_{st3} = (\overline{N}_{ve} + 37) N_e + \left[1.25 \overline{N}_k + 20.75 \overline{N}_{cl} + 116.5\right] N_p + 7$

Table 1: Computation times

Figure 4 illustrates some of the basic features of the different $\hat{T}_t$ curves. All functions are linear with respect to N_p. The slope of *ST1* is clearly the largest, followed by *ST2* and *ST3*, which stays nearly flat. In contrast, the offset of *ST3* is the largest, followed by *ST2*, while *ST1* has no offset. These offsets are due to the preprocessing time, $\hat{T}_b$, required by the virtual mesh and the bounding box calculations. Next, pairs of element tests are compared while varying all parameters. Figure 5 is a comparison of *ST1* and *ST2*. The curves represent iso-lines where $\hat{T}_t|_{st2} = \hat{T}_t|_{st1}$. *ST2* is more efficient than *ST1* for $N_p > 12$ as long as $\overline{N}_{cl}$, the average number of elements that claim a data point, is lower than 80% of the number of elements, N_e. This condition is satisfied for all but very distorted meshes. The extra bounding-box build time $\hat{T}_b$ is compensated by the decreasing slope in the $\hat{T}_t$ vs. N_p curve. Figure 6 is a comparison of *ST2* and *ST3*. The figure shows that for any reasonable mesh, $N_e \geq 30$ and $N_p \geq 50$, it is advantageous to use a virtual mesh. In section 5 we validate these conclusions experimentally. It uses a C++ implementation of the different search techniques.

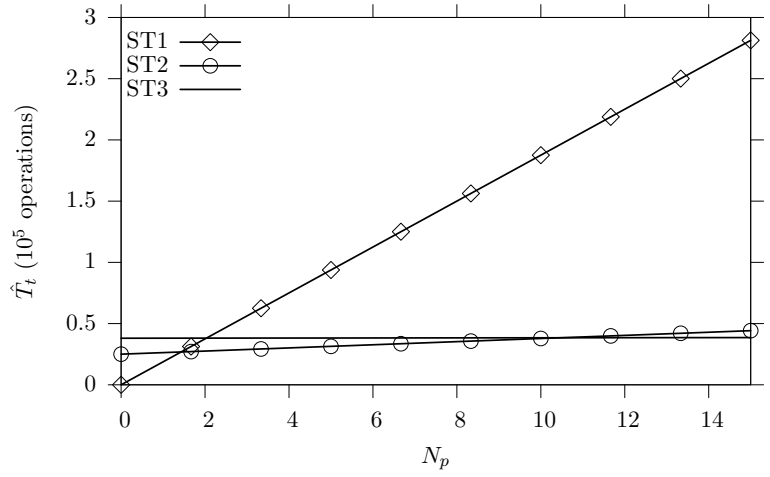


Figure 4: Number of operations vs. number of data points for methods ST1, ST2 and ST3 ($N_e = 10000$, $\overline{N}_{cl} = 10$ and $\alpha = 0.05$).

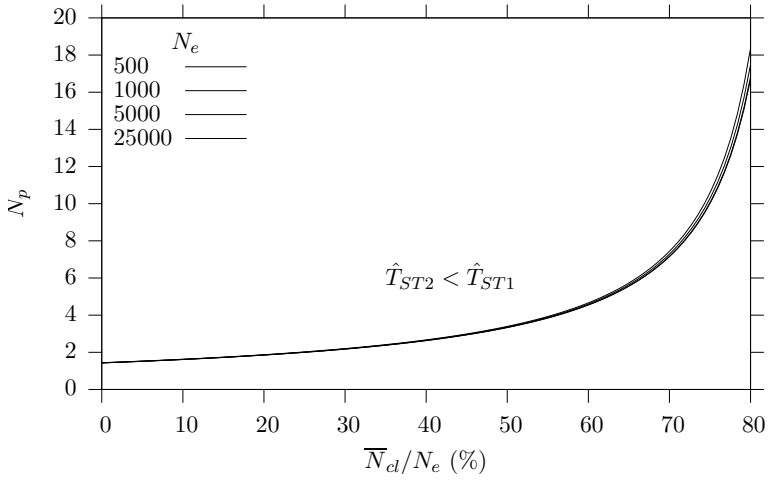


Figure 5: Comparison of *ST1* and *ST2*. The curves represent iso-lines where $\hat{T}_t|_{st2} = \hat{T}_t|_{st1}$. The regions above the curve are such that $\hat{T}_t|_{st2} < \hat{T}_t|_{st1}$. Notice that N_e does not affect the curves to a significant degree.

4 Inverse mapping

Finite elements shape coefficients N_j are generally known functions of an element's local coordinates $\vec{\xi}$ (equation (2)). Hence in general, to evaluate these coefficients it is necessary to compute $\vec{\xi}(p)$ from $\vec{x}(p)$. Equation (18) defines a non-linear mapping function from the reference coordinates, $\vec{\xi}$, to the global coordinates, $\vec{x}$,

$$x(p) = \sum_{j=1}^{\text{no. nodes in } e(p)} S_j\left(\vec{\xi}(p)\right) x(n_j^{e(p)}) \; . \tag{18}$$

The coefficients S_j are also known functions of $\vec{\xi}$, and are the same as N_j for iso-parametric elements. The opposite operation (i.e. calculating $\vec{\xi}(p)$ from $\vec{x}(p)$) is called inverse mapping.

Inverse mapping is a non-trivial operation, which in general needs to be calculated numerically, although for special cases it is possible to invert the shape functions analytically. Whichever technique is applicable, the accuracy of the inversion can be tested as illustrated in Figure 7. If the inversion is successful the difference, $\delta_\xi = \|\vec{\xi}^* - \vec{\xi}^T\|$, should approach machine precision. Since $\vec{\xi}^T$ is unknown, δ_ξ cannot be

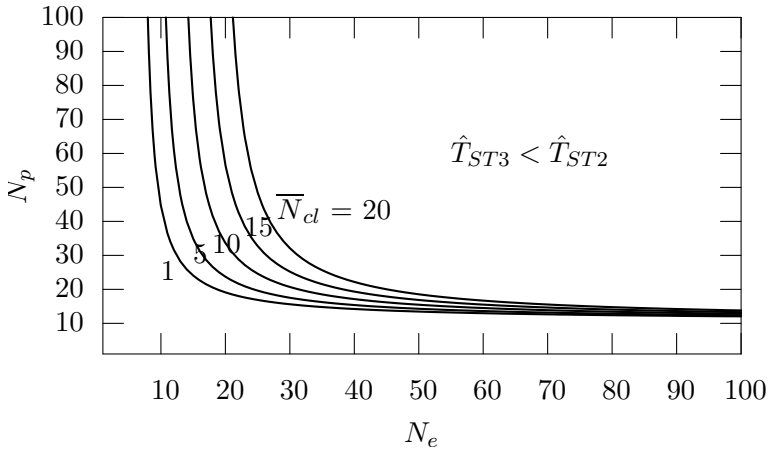


Figure 6: Comparison of *ST3* and *ST2*. The curves represent iso-lines where $\hat{T}_t|_{st3} = \hat{T}_t|_{st2}$. The curves use the optimum α and $\overline{N}_{cl} = 1, 5, 10, 15, 20$.

computed directly. Instead, by mapping $S : \vec{\xi}^* \rightarrow \vec{x}^*$, $\delta_x = \|\vec{x}^* - \vec{x}\|$ is used as a measure of the inverse mapping error. For shape functions where the analytical solution of the inverse is unknown, δ_x is typically taken as the cost function to be minimized, by varying $\vec{\xi}$. Section 5 implements both analytic and iterative inverse mapping strategies for a QUAD4 element. The iterative approach uses a Newton-Raphson optimizer to minimize δ_x. Sections 4.1 and 4.2 discusses analytic and iterative inversions, respectively.

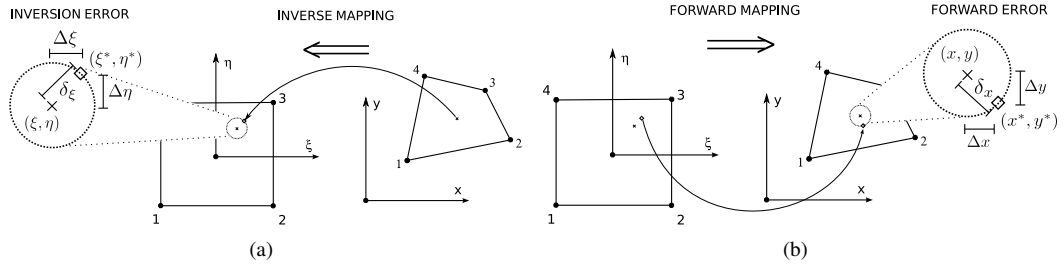


Figure 7: Illustration of how to check the inverse mapping accuracy. (a) The inverse mapping function $S^{-1} : \vec{x} \rightarrow \vec{\xi}^*$ may contain errors $\delta_\xi = \|\vec{\xi}^* - \vec{\xi}^T\|$ caused by convergence accuracy, where $\vec{\xi}^T$ are the target coordinates. (b) The inversion error δ_ξ cannot be computed directly, since $\vec{\xi}^T$ is unknown. Instead, the accuracy of the inversion can be determined by mapping $\vec{\xi}^*$ to the real coordinate system $S : \vec{\xi}^* \rightarrow \vec{x}^*$ and computing the error $\delta_x = \|\vec{x}^* - \vec{x}\|$. If the inversion is successful, δ_x should approach zero.

4.1 Analytic inverse mapping for QUAD4 elements

For a QUAD4 element equation (18) can be rewritten as

$$x = a_0 + a_1\xi + a_2\eta + a_3\xi\eta \tag{19}$$

$$\text{and } y = b_0 + b_1\xi + b_2\eta + b_3\xi\eta \ , \tag{20}$$

where the coefficients a_i and b_i are solved by evaluating the shape functions at node coordinates (see Appendix B)[2]. The reference coordinates are determined by solving for either ξ or η using one equation, x or y, then substituting the result into the remaining equation. For instance, solving for ξ using the x equation,

$$\xi = \frac{x_0 - a_2\eta}{a_1 + a_3\eta} \ , \tag{21}$$

[2]The (p) is omitted from $x(p), y(p), \xi(p)$ and $\eta(p)$ for convenience.

then substituting ξ into the y equation, we obtain

$$A\eta^2 + B\eta + C = 0 \ , \tag{22}$$

where the coefficients A, B, C are

$$\begin{array}{rcl} A & = & a_3 b_2 - a_2 b_3 \ , \\ B & = & (x_0 b_3 + a_1 b_2) - (y_0 a_3 + a_2 b_1) \ , \\ C & = & x_0 b_1 - y_0 a_1 \end{array} \tag{23}$$

and $x_0 = a_0 - x$, $y_0 = b_0 - y$. Thus, in general, inverse mapping requires the solution of a quadratic equation. However, if equation (21) results in division by zero (i.e. $a_1 + a_3\eta = 0$), an alternative inverse mapping formulation must be used. For geometrically admissible elements (i.e. elements wit no crossing edges and a non-zero area) two cases cover all possible situations. Details including proofs of these formulations can be found in Appendix B.

case 1, $a_1 \neq 0$, $a_3 \neq 0$: The value of η is obtained from $a_1 + a_3\eta = 0$,

$$\eta = \frac{-a_1}{a_3} \tag{24}$$

and ξ by substituting η into the y equation,

$$\xi = \frac{y_0 a_3 + a_1 b_2}{a_3 b_1 - a_1 b_3} \ . \tag{25}$$

case 2, $a_1 = 0$, $a_3 = 0$: The value of η is obtained by solving the x equation,

$$\eta = \frac{x_0}{a_2} \tag{26}$$

and ξ by substituting η into the y equation,

$$\xi = \frac{y_0 a_2 - x_0 b_2}{a_2 b_1 + x_0 b_3} \ . \tag{27}$$

4.2 Iterative inverse mapping

In applications where the shape functions are more complex than QUAD4 elements, an analytical inversion of the shape functions may become too tedious to be carried out. Figure 8 illustrates one such case. The element in question is a 20 node 3D brick element, which contains a set of three 2^{nd} order 3D shape functions. The alternative is to use an iterative technique to compute the reference coordinates of data point, p.

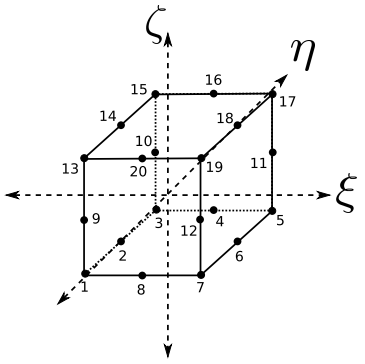
The iterative process begins with an initial guess of the reference coordinates, $\vec{\xi}_0$, which can be selected from node coordinates, integration points, element center, or any other point inside the element. Consider a formulation of the square of the forward error, f, where $\vec{\xi}^T(p)$ and $\vec{x}^T(p)$ are the target coordinates of p,

$$f(\vec{\xi}) = \delta_x^2(\vec{\xi}) = (x_j^T(p) - S_i(\vec{\xi}) x_j(n_i^{e(p)}))^2 \ . \tag{28}$$

Notice that both $x_j^T(p)$ and $x_j(n_i^{e(p)})$ are known quantities, and $\mathbf{S}_i$ are known functions of $\vec{\xi}$, the unknowns. Since the shape functions are continuous in the reference coordinate system, the function $f(\vec{\xi})$ must be continuous and zero (its minimum) at the solution point $\vec{\xi}^T$, $f(\vec{\xi}^T) = f_{min} = 0$. Hence, it is possible to minimize f to solve for the reference coordinates of p. Since the shape functions are polynomials, a gradient method may be used for this purpose. Using this technique the condition,

$$g_i = \frac{\partial f}{\partial \xi_i} = 0 \ , \tag{29}$$

must be satisfied at the solution point. Hence, the iteration step is $\Delta\vec{\xi} = -\frac{1}{2}\mathbf{H}^{-1}\vec{g}$, where $\mathbf{H}$ is the Hessian of f, $H_{ij} = \frac{\partial^2 f}{\partial \xi_i \partial \xi_j}$. The individual partial derivatives are computed from equation (28),



(a) Reference Element

Corners:

$S_i(\xi,\eta,\zeta) = \frac{1}{8}(1+\xi\xi_i)(1+\eta\eta_i)(1+\zeta\zeta_i)(\xi\xi_i+\eta\eta_i+\zeta\zeta_i-2)$

Sides:

$S_i(\xi,\eta,\zeta) = \frac{1}{4}(1+\xi^2)(1+\eta\eta_i)(1+\zeta\zeta_i)$

$$\vec{x}(p) = \sum_{i=1}^{20} S_i \vec{x}(n_i^{e(p)})$$

(b) Shape Functions

Figure 8: A 20-node 3D brick element (BRK20)

$$\frac{\partial f}{\partial \xi_i} = 2(x_k^T(p) - S_l(\vec{\xi}) x_l(n_k^{e(p)}) \frac{\partial S_l(\vec{\xi})}{\partial \xi_i} \quad ,$$

$$\frac{\partial^2 f}{\partial \xi_i \partial \xi_j} = 2\frac{\partial S_l(\vec{\xi})}{\partial \xi_i}\frac{\partial S_l(\vec{\xi})}{\partial \xi_j} + 2(x_k^T(p) - S_l(\vec{\xi}) x_l(n_k^{e(p)})) \frac{\partial^2 S_l(\vec{\xi})}{\partial \xi_i \partial \xi_j} \quad .$$

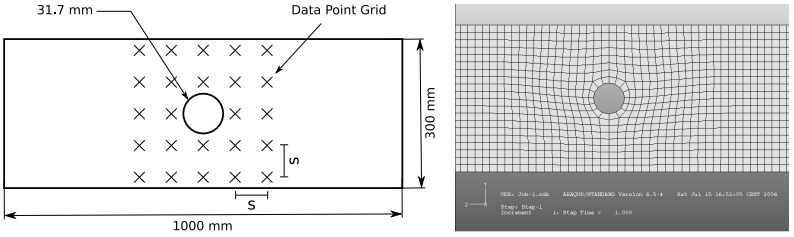
The iteration should continue until reaching the stopping criteria, $\delta_x \leq \delta_{allow}$. The stop criteria depends on the inversion accuracy required by an application.

5 Numerical experiments

The efficiency of owner element search algorithms were estimated in section 3.3 by counting operations. It assumed that other contributions to the algorithm's time complexity are negligible (e.g. memory allocation, different computation costs for integer and floating point arithmetic, etc). This section checks these assumptions by comparing the theoretical estimates with numerical experiments. The computation times are measured when interpolating data-point grids, with varying number of points, N_p, over meshes of different element numbers, N_e. The experiments are conducted with a C++ implementation of the interpolation algorithms, with an ABAQUS 6.4 finite elements solver [1]. The computer test-bed is a Toshiba satellite A60 with a Pentium 4 processor running on a GNU/Linux Debian 3.1 operating system [2]. The FE model is an open-hole plate specimen illustrated in Figure 9. The model's left side is fully constrained, and the right side subjected to a 1550 KPa surface traction. The data points, **p**, are located on a grid of step s (Figure 9) [3]. The total processing time, T_t, is measured by an embedded C++ timer, which measures the time in seconds from the beginning of the interpolation procedure until all data points in the grid have been interpolated. The procedure is repeated with different mesh sizes, N_e, and grid steps, s.

Figures 10(a) and 10(b) show the test results for the ST2 algorithm, which uses the bounding-box and cross-product tests to sequentially scan the entire list of finite elements. The execution time, T_t, increases linearly with N_e and N_p when N_p and N_e are held constant. Notice that the slopes of T_t increases with N_e and N_p. The experimental behavior is in agreement with the $\mathcal{O}(N_e \times N_p)$ time complexity $\hat{T}_t|_{st2}$ in section

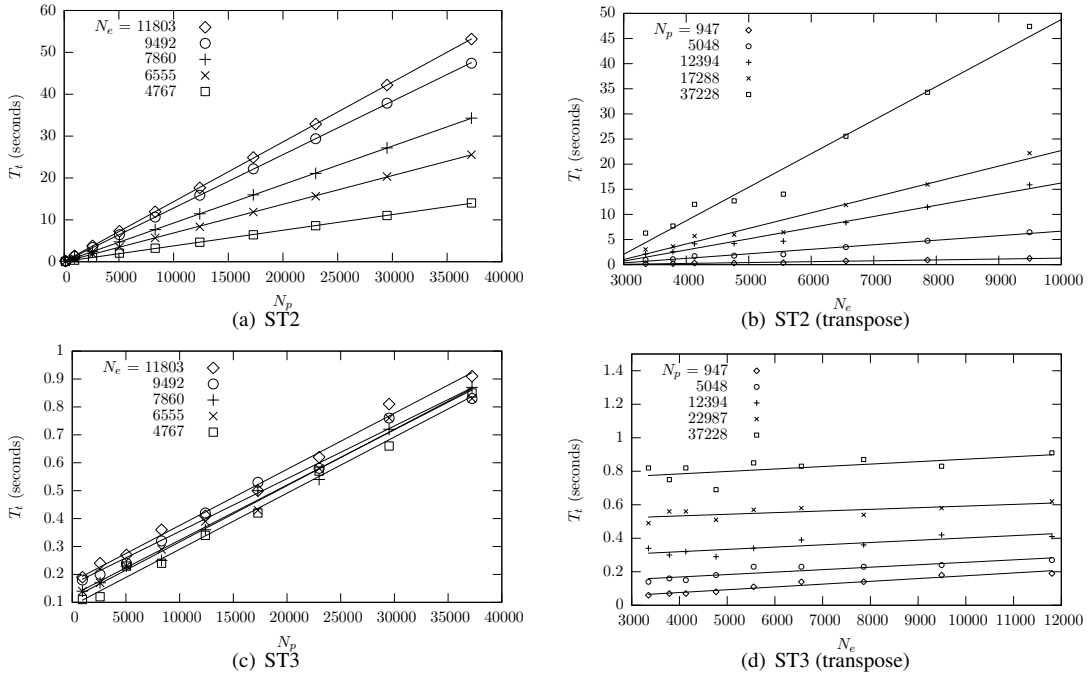
[3] The data point grids encompass only a square window on the center of the model. Since the numbering of the elements follows a rectangular pattern, the square data field can be considered average.



(a) (b)

Figure 9: (a)Test-Specimen Geometry: The model is an open-hole tensile test with the left side fully constrained and the right side subjected to a 1550 KPa surface traction. (b) Details of the finite elements mesh.

3.3 (Table 1). Figure 10(c) shows the results for the ST3 algorithm, which uses a combination of the cross-product and bounding-box tests and a virtual mesh container. Clearly, the interpolation algorithm using the virtual mesh is more efficient than the previous algorithm. For large number of points ($N_e \geq 5000$), the virtual mesh reduces computation times by a factor of 10 to 60 times. Notice that T_t varies linearly with N_p and N_e, but unlike the previous case, the slopes of these curves are independent of N_e and N_p. These results match the $\mathcal{O}(\mathcal{A}N_e + \mathcal{B}N_p)$ in the time complexity $\hat{T}_t|_{st3}$ ($\mathcal{A}$ and $\mathcal{B}$ are functions of mesh distortion, but independent of N_e and N_p, see Table 1).



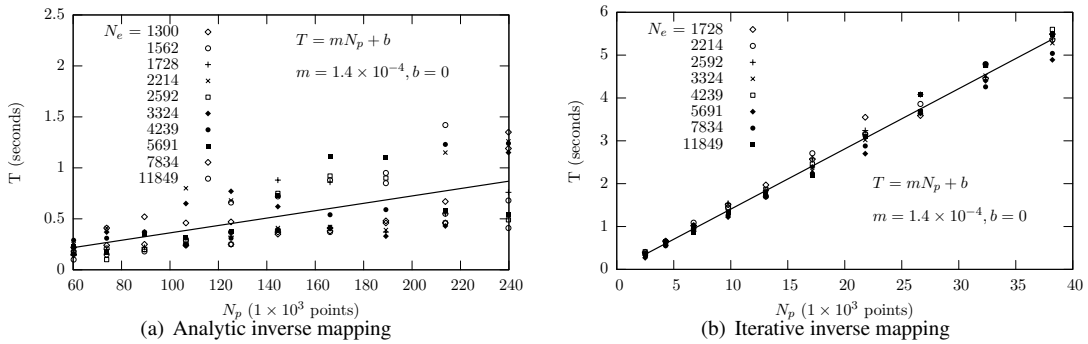
(a) ST2 (b) ST2 (transpose)

(c) ST3 (d) ST3 (transpose)

Figure 10: Element search times (measured numerically)

Next, Figures 11(a) and 11(b) show the recorded inverse mapping times for analytic and iterative inverse mapping algorithms. The iterative technique is a 2D Newton-Raphson minimization of δ_x. The optimizer has a stopping criterion of $|\delta_x| < 1e^{-5}$mm. Inverse mapping times for both techniques show a linear variation with N_p and no tendency with respects to N_e. The scatter on the graphs is due to a dependency of the inverse mapping cost on the position of the data points inside an element (see Appendix A). The figures show that inverse mapping is performed very quickly in comparison to the owner element search. For the analytic solution, nearly 250000 data points are interpolated in less than 1 CPU second. The iterative approach is significantly slower, but still is capable of interpolating 40000 points in about 5 seconds. If the field comparison is performed several times over a constant mesh, the index of the owner element

and reference coordinates of each data point can be saved after the first interpolation and reused in each subsequent comparison. This practice can permit further reduction in interpolation time.



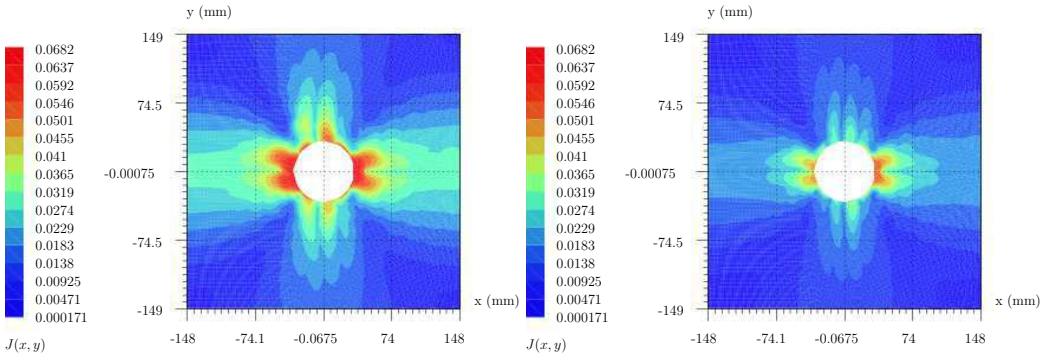
(a) Analytic inverse mapping (b) Iterative inverse mapping

Figure 11: Inverse mapping and interpolation times using (a) analytic and (b) iterative inversion algorithms.

This interpolation technique is now applied to the construction and visualization of error maps. The plate with a hole (Figure 9) is composed of an orthotropic material of properties $E_{11} = 100$ GPa, $E_{22} = 36$ GPa, $G_{12} = 25$ GPa, $\nu_{12} = 0.54$ GPa. A "reference model" is solved with a fine mesh of approximately 33000 elements, and its nodal displacements are taken as the experimental data field. The error maps are produced by solving several models with the same material properties, but varying mesh densities N_e, and interpolating displacements at each data point. Figure 12 shows two error maps of the normalized distance,

$$J(p) = \sqrt{\left(\frac{u(p)^{ref} - u(p)^{test}}{u^{ref}_{max} - u^{ref}_{min}}\right)^2 + \left(\frac{v(p)^{ref} - v(p)^{test}}{v^{ref}_{max} - v^{ref}_{min}}\right)^2}, \tag{30}$$

where $p \in \mathbf{n}^{ref.\ mesh}$. Notice that as the number of elements increases the mesh approximates better the curvature around the hole, which decreases the error J.



(a) Test mesh: $N_e = 700$ (b) Test mesh: $N_e = 1500$

Figure 12: Error maps $J(p)$ between a reference mesh of $N_e = 30000$ and test meshes of (a) $N_e = 700$ and (b) $N_e = 1500$.

6 Conclusions

This article proposes and analyzes algorithms for mapping mesh information to arbitrary points using finite element's shape functions. The algorithms consist of two parts: the identification of the owner element, and the mapping of interpolation points into the element's reference coordinate system. The interpolation is then a direct evaluation of the element's shape functions.

The strategy for determining the owner element for each data point is critical to the numerical efficiency of the procedure. The use of a virtual mesh has been proposed as an alternative to sequentially searching the entire list of finite elements. It was shown that this indexing technique permits the time complexity to increase linearly with increasing mesh size, N_e, and number of data points, N_p, contrary to sequential techniques, which increase quadratically with $N_e \times N_p$. In addition to the virtual mesh, an efficient test for determining if a point belongs to an element has been developed. It is a combination of a fast approximate test based on bounding-boxes and an exact test based on cross-products, which works for all linear elements. Inverse mapping has been discussed through the analytical example of bilinear quadrilateral elements, and numerically in the general cases.

Finally, the different interpolation strategies have been implemented in C++, and applied to a finite elements model of open-hole tensile test. As a typical example, the best strategy (virtual mesh, mixed test) interpolates over 40,000 data points in less than 1 second for a finite elements mesh of 10,000 elements on a Pentium 4, 2.8 GHz computer. The numerical tests have confirmed the theoretical analysis based on operations counting.

References

[1] *ABAQUS/CAE user's manual : version 6.4.* Pawtucket, RI : ABAQUS, 2003.

[2] *Debian GNU/Linux 3.1 Bible*. John Wiley & Sons, 2005.

[3] B. C. Barber, D. P. Dobkin, and H. Huhdanpaa. The quickhull algorithm for convex hulls. *ACM Trans. Math. Softw.*, 22(4):469–483, 1996.

[4] A. Beckert. Coupling fluid (CFD) and structural (FE) models using finite interpolation elements. *Aerospace Science and Technology*, 4(1):13 – 22, January 2000.

[5] A. K. Bledzki, A. Kessler, R. Rikards, and A. Chate. Determination of elastic constants of glass/epoxy unidirectional laminates by the vibration testing of plates. *Composites Science and Technology*, 59(13):2015–2024, October 1999.

[6] P. Bourke. Determining if a point lies on the interior of a polygon. Crawley, AU., November 1989. Available http://local.wasp.uwa.edu.au/ pbourke/geometry/insidepoly/index.html.

[7] L. Bruno, F. M. Furgiuele, L. Pagnotta, and A. Poggialini. A full-field approach for the elastic characterization of anisotropic materials. *Optics and Lasers in Engineering*, 37:417–431, April 2002.

[8] N. Cressie. *Statistics for spatial data*. Wiley-Interscience, New York, 1st edition, 15 January 1993.

[9] J. Cugnoni, T. Gmur, and A. Schorderet. Identification by modal analysis of composite structures modelled with FSDT and HSDT laminated shell finite elements. *Composites Part A-Applied Science and Manufacturing*, 35(7-8):977 – 987, 2004.

[10] Y. Fukushima, V. Cayol, and P. Durand. Finding realistic dike models from interferometric synthetic aperture radar data: The February 2000 eruption at Piton de la Fournaise. *Journal of Geophysical Research-Solid Earth*, 110(B3), 23 March 2005.

[11] K. Genovese, L. Lamberti, and C. Pappalettere. Mechanical characterization of hyperelastic materials with fringe projection and optimization techniques. *Optics and Lasers in Engineering*, 44:423–442, May 2006.

[12] J. Kajberg and G. Lindkvist. Characterisation of materials subjected to large strains by inverse modelling based on in-plane displacement fields. *International Journal of Solids and Structures*, 41(13):3439–3459, February 2004.

[13] Mathworks. *Matlab: The language of technical computing*. Mathworks, Natic, 7 edition, June 2004.

[14] M. H. H. Meuwissen, C. W. J. Oomens, F. P. T. Baaijens, R. Petterson, and J. D. Janssen. Determination of the elasto-plastic properties of aluminium using a mixed numerical-experimental method. *Journal of Materials Processing Technology*, 75(1-3):204 – 211, March 1998.

[15] J. Molimard, R. Le Riche, A. Vautrin, and J.R. Lee. Identification of the four orthotropic plate stiffnesses using a single open-hole tensile test. *Experimental Mechanics*, (45):404–411, 2005.

[16] G.P. Nikishkov. Generating contours on fem/bem higher-order surfaces using java 3d textures. *Advances in Engineering Software*, 34(8):469–476, August 2003.

[17] S. Padmanabhan, J. P. Hubner, A. V. Kumar, and P. G. Ifju. Load and boundary condition calibration using full-field strain measurement. *Experimental Mechanics*, 46(5):569 – 578, October 2006.

[18] E. Pagnacco and D. Lemosse. A coupled FE based inverse strategy from displacement field measurement subject to an unknown distribution of forces. Clermont-Ferrand, France, 10 July 2006. Photomechanics.

[19] A. Rassineux. *Maillage automatique tridimentionnel par une méthode frontale pour la méthode des éléments finis*. PhD thesis, Université Henri Poincaré, Vandoeuvre les Nancy, France, 1994. (in French).

[20] M. Rumpf. Recent numerical methods - a challenge for efficient visualization. *Future Generation Computer Systems*, (15):43–58, September 1999.

[21] D. T. Sandwell. Biharmonic spline interpolation of GEOS-3 and SEASAT altimeter data. *Geophysical Research Letters*, 12(2):139–142, February 1987.

[22] R. van Loon, P. D. Anderson, and F. N. van de Vosse. A fluid-structure interaction method with solid-rigid contact for heart valve dynamics. *Journal of Computational Physics*, 217(2):806 – 823, SEP 20 2006.

A Search time details for QUAD4 elements

Search technique 1 (cross-product test + sequential scan):

Algorithm D.1 contains the pseudocode for ST1, which sequentially scans a list of finite elements using the cross-product test. The time complexities for this test are estimated by counting operations in this code. To compute the average failure time, $\hat{T}_f|_{cp}$ (failure time for a cross-product test), notice that an external point may fall outside any of the vertex cones with equal probability (see table 2). Hence,

$$\hat{T}_f|_{cp} = \frac{\hat{T}_{f1} + \hat{T}_{f2} + \hat{T}_{f3} + \hat{T}_{f4}}{4} = \frac{15 + 30 + 45 + 60}{4} \,. \tag{31}$$

Failure Node	Number of Operations
1	$2 \times$ Cross-Product $+ 1 \times$ Comparison
2	$4 \times$ Cross-Product $+ 2 \times$ Comparison
3	$6 \times$ Cross-Product $+ 4 \times$ Comparison
4	$8 \times$ Cross-Product $+ 6 \times$ Comparison

Table 2: Cross-product test: Number of operations for failure at the n^{th} vertex cone. Note: Each Cross-Product requires 7 operations.

For an internal point all vertex cones must be tested, hence $\hat{T}_s = \hat{T}_{f4}$,

$$\hat{T}_f|_{cp} = \frac{75}{2} \tag{32}$$

$$\text{and } \hat{T}_s|_{cp} = 60 \,. \tag{33}$$

Applying equation 12 yields,

$$
\begin{aligned}
\hat{T}_c|_{ST1} &= \tfrac{1}{N_e}\sum_{i=1}^{N_e}\hat{T}_i\ , \\
&= \tfrac{1}{N_e}\sum_{i=1}^{N_e}(i-1)\hat{T}_f + \hat{T}_s\ , \\
&= \tfrac{\hat{T}_f}{N_e}\sum_{i=1}^{N_e} i + \tfrac{\hat{T}_f}{N_e}\sum_{i=1}^{N_e}(\hat{T}_s - \hat{T}_f)\ , \\
&= \hat{T}_f \times \tfrac{N_e+1}{2} + (\hat{T}_s - \hat{T}_f)\ , \\
&= \hat{T}_f \times \tfrac{N_e}{2} + (\hat{T}_s - \tfrac{\hat{T}_f}{2})\ .
\end{aligned}
$$

Substituting values for $\hat{T}_f|_{cp}$ and $\hat{T}_s|_{cp}$ yields,

$$\hat{T}_c|_{cp} = \frac{75}{4}N_e + \frac{165}{4}\ . \tag{34}$$

The total processing time for ST1 is

$$\hat{T}_t|_{ST1} = \frac{75}{4}N_eN_p + \frac{165}{4}N_p\ . \tag{35}$$

Search technique 2 (bounding-box and cross-product tests + sequential scan):

Algorithms D.2 and D.3 contains the pseudocode for ST2, which sequentially scans a list of finite elements using the bounding-box and cross-product tests. The algorithm begins by computing the bounding boxes for each element in the preprocessor, then storing them in memory. the average time required to build a bounding box $\hat{T}_b|_{bb}$ is

$$
\begin{aligned}
\hat{T}_b|_{bb} &= 4 \times \text{Assignment} + 3 \times 4 \times \text{Comparisons} + \tfrac{6+2}{2} \times \text{Assignment}\ , \\
\hat{T}_b|_{bb} &= 20\ .
\end{aligned} \tag{36}
$$

Each data point's coordinates are tested against the bounding-box's reference points, p_{min} and p_{max} (2 comparisons per dimension). Notice, that external points may fail a comparison at any dimension with equal probability (see Algorithm D.3), thus the average failure time $\hat{T}_f|_{bb}$ is

$$
\begin{aligned}
\hat{T}_f|_{bb} &= \tfrac{1+2+3+4}{4} \times \text{Comparison}\ , \\
\hat{T}_f|_{bb} &= \frac{5}{2}\ .
\end{aligned} \tag{37}
$$

For an internal point all comparisons must be performed, hence $\hat{T}_s|_{bb}$ is

$$\hat{T}_s|_{bb} = 4\ . \tag{38}$$

Next, we assume the first i finite elements in the list will be discarted by the bounding-box test. After i elements, a list of $\overline{N}_{cl}$ (the average number of bounding-boxes claiming a data point) elements will be searched using the cross-product test, hence,

$$\hat{T}_c|_{ST2} = \frac{1}{N_e}\sum_{i=1}^{N_e}(i-1)\hat{T}_f|_{bb} + \overline{T}_s|_{cl}\ , \tag{39}$$

$$\hat{T}_t|_{ST2} = \hat{T}_b|_{bb}N_e + \hat{T}_c|_{bb}N_p\ . \tag{40}$$

Where $\overline{T}_s|_{cl}$ is the average time to search $\overline{N}_{cl}$ finite elements using the cross-product test,

$$\overline{T}_s|_{cl} = \frac{1}{\overline{N}_{cl}}\sum_{k=1}^{\overline{N}_{cl}}(k-1)\hat{T}_f|_{cp} + k\hat{T}_f|_{cp} + \hat{T}_s|_{cp}\ . \tag{41}$$

Evaluating the expressions yields,

$$\overline{T}_s|_{cl} = \frac{\overline{N}_{cl} - 1}{2}(\hat{T}_s|_{bb} + \hat{T}_f|_{cp}) + \hat{T}_s|_{cp} - \hat{T}_f|_{cp} \ ,$$

$$\hat{T}_c|_{bb} = \frac{N_e - 1}{2}\hat{T}_f|_{bb} + \overline{T}_s|_{cl} - \hat{T}_f|_{bb} \ .$$

Substituting values for $\hat{T}_f|_{bb}, \hat{T}_s|_{bb}, \hat{T}_f|_{cp}, \hat{T}_s|_{cp}$ we obtain the time complexity for ST2,

$$\hat{T}_c|_{ST2} = \frac{5}{4}N_e + \frac{83}{4}\overline{N}_{cl} + \frac{442}{4} \ , \tag{42}$$

and the total processing time,

$$\hat{T}_t|_{ST2} = \frac{5}{4}N_eN_p + \frac{83}{4}\overline{N}_{cl}N_p + \frac{442}{4}N_p + 20N_e \ . \tag{43}$$

Search technique 3 (bounding-box and cross-product tests + virtual mesh):

Algorithm D.5 contains the pseudo code for initializing a VM using a bounding box. The total build time for ST3, $\hat{T}_b|_{ST3}$, is divided into the time to calculate virtual mesh's borders, $\hat{T}_{bg}|_{vm}$, initialize an element's bounding box, $\hat{T}_b|_{bb}$ and time to add the element's reference to a range of virtual elements, $\hat{T}_{add}|_{vm}$. Hence,

$$\hat{T}_b|_{ST3} = \hat{T}_{bg}|_{vm} + (\hat{T}_b|_{bb} + \hat{T}_{add}|_{vm})N_e \ . \tag{44}$$

Algorithm D.7 determines the boundaries of the virtual mesh, thus

$$\hat{T}_{bg} = 4 \text{ Assignment} + 4N_e \times \text{Comparison} + \left(\frac{1}{N_e}\sum_{i=1}^{N_e} 2i \times \text{Assignment}\right) + 4 \text{ Math Operations} \ ,$$

$$\hat{T}_{bg}|_{vm} = 5N_e + 7 \ . \tag{45}$$

Algorithm D.6 contains the pseudo code to add finite elements to the virtual mesh,

$$\begin{aligned} \hat{T}_{add}|_{vm} &= 2 \times \text{Get}(\gamma(p)) + \overline{N}_{ve} \times \text{Assignment} \ , \\ &= 2 \times (6) + \overline{N}_{ve} \ , \end{aligned}$$

$$\hat{T}_{add}|_{vm} = \overline{N}_{ve} + 12 \ . \tag{46}$$

Where $\overline{N}_{ve}$ is the average number of virtual elements containing a reference to the same finite element. This number is dependent on the mesh geometry and virtual mesh step (see section 3.2). Substituting values for equation (44),

$$\begin{aligned} \hat{T}_b|_{vm} &= \hat{T}_{bg}|_{vm} + \left[\hat{T}_b|_{bb} + \hat{T}_{add}|_{vm}\right] N_e \ , \\ &= (5N_e + 7) + \left[20 + \overline{N}_{ve} + 12\right] N_e \ , \end{aligned}$$

$$\hat{T}_b|_{vm} = (\overline{N}_{ve} + 37)N_e + 7 \ . \tag{47}$$

Algorithm D.4 is the pseudo-code for retrieving a virtual element. Notice that a virtual element is retrieved using only 2 operations per dimension,

$$\hat{T}_e|_{vm} = 6 \ . \tag{48}$$

The bounding-box and cross-product tests are used to search the lists $\mathbf{e}^v$. Thus the time to search a virtual element, $\hat{T}_{se}|_{ve}$, has the same computational complexity as ST2. Substituting the average number of elements in $\mathbf{e}^v$, $\overline{N}_k$, into equation (42) yields,

$$\hat{T}_{se}|_{ve} = \frac{5}{4}\overline{N}_k + \frac{83}{4}\overline{N}_{cl} + \frac{442}{4} \ . \tag{49}$$

In general, the time complexity and total search time for ST3 are

$$\hat{T}_c|_{ST3} = \hat{T}_e|_{vm} + \hat{T}_{se}|_{ve} \tag{50}$$

$$\text{and } \hat{T}_t|_{ST3} = \hat{T}_b|_{vm} + \hat{T}_c|_{ST3} \times N_p \tag{51}$$

respectively. Substituting yields,

$$\hat{T}_c|_{ST3} = 6 + \frac{5}{4}\overline{N}_k + \frac{83}{4}\overline{N}_{cl} + \frac{442}{4} \tag{52}$$

$$\text{and } \hat{T}_t|_{ST3} = (\overline{N}_{ve} + 37)N_e + \left[6 + \frac{5}{4}\overline{N}_k + \frac{83}{4}\overline{N}_{cl} + \frac{442}{4}\right] N_p + 7 \tag{53}$$

B Inverse mapping theorems

This section establishes a mathematical basis for the inversion technique presented in section 4.1. The presented proofs are limited to elements posessing a unique analytical inverse (i.e. elements with non-zero Jacobians). For a robust interpolation algorithm, it is recommended that the mesh generator possess an algorithm to minimize mesh distortion (usually available in mesh generating software). For all other elements, the analytical inverse mapping algorithm will propose a solution, which must be verified using forward mapping (presented in section 4). In the case where the analytical solution cannot be obtained analytically, an alternate technique should be used instead.

Definition 1. *Kinematically admissible element*: Let the shape functions F of a 2D bi-linear quadrilateral element be defined by equations 54, such that $(\xi, \eta)\epsilon[-1, 1]^2$. A kinematically admissible element is such that the nodal arrangement in the real coordinate system does not contain any crossing segments, and has a non-zero area.

Let the shape functions, $F\colon \xi(p) \rightarrow x(p)$, of a QUAD4 element be written as

$$\begin{array}{rcccl} x_0 & = & x - a_0 & = & a_1\xi + a_2\eta + a_3\xi\eta \\ \text{and } y_0 & = & y - b_0 & = & b_1\xi + b_2\eta + b_3\xi\eta\,, \end{array} \tag{54}$$

where the coefficients a_i and b_1 are

$$\begin{array}{rclcrcl} a_0 & = & \frac{1}{4}\left[(x_{n1} + x_{n2}) + (x_{n3} + x_{n4})\right]\,, & & b_0 & = & \frac{1}{4}\left[(y_{n1} + y_{n2}) + (y_{n3} + y_{n4})\right]\,, \\ a_1 & = & \frac{1}{4}\left[(x_{n2} - x_{n1}) + (x_{n3} - x_{n4})\right]\,, & & b_1 & = & \frac{1}{4}\left[(y_{n2} - y_{n1}) + (y_{n3} - y_{n4})\right]\,, \\ a_2 & = & \frac{1}{4}\left[(x_{n3} + x_{n4}) - (x_{n1} + x_{n2})\right]\,, & & b_2 & = & \frac{1}{4}\left[(y_{n3} + y_{n4}) - (y_{n1} + y_{n2})\right]\,, \\ a_3 & = & \frac{1}{4}\left[(x_{n1} - x_{n2}) + (x_{n3} - x_{n4})\right]\,, & & b_3 & = & \frac{1}{4}\left[(y_{n1} - y_{n2}) + (y_{n3} - y_{n4})\right]\,. \end{array}$$

Proposition 1. *If $a_1 + a_3\eta \neq 0$, then for all kinematically admissible elements, the inverse function, $F^{-1}\colon \vec{x}(p) \rightarrow \vec{\xi}(p)$, is a solution of the quadratic system,*

$$\xi = \tfrac{x_0 - a_2\eta}{a_1 + a_3\eta} \quad \textit{and} \quad A\eta^2 + B\eta + C = 0\,, \tag{55}$$

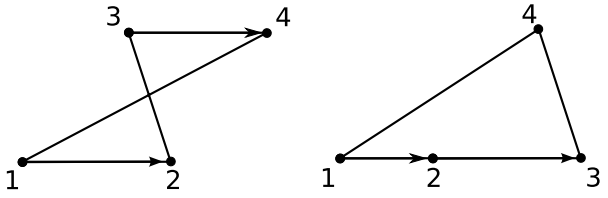
where the coefficients A, B and C are

$$\begin{array}{rcl} A & = & a_3b_2 - a_2b_3\,, \\ B & = & (x_0b_3 + a_1b_2) - (y_0a_3 + a_2b_1)\,, \\ \textit{and } C & = & x_0b_1 - y_0a_1\,. \end{array}$$

Proposition 2. *If $a_1 + a_3\eta = 0$, $a_1 \neq 0$, and $a_3 \neq 0$, then for all kinematically admissible elements, the function $F^{-1}\colon \vec{x}(p) \rightarrow \vec{\xi}(p)$ is*

$$\xi = \tfrac{y_0a_3 + a_1b_2}{a_3b_1 - a_1b_3} \quad \textit{and} \quad \eta = \tfrac{-a_1}{a_3}\,. \tag{56}$$

Proof. For all kinematically admissible elements such that $a_1 + a_3\eta = 0$, $a_1 \neq 0$ and $a_3 \neq 0$, we prove that $a_3b_1 - b_3a_1 \neq 0$, which guarantees 56 can be used without division by zero. Assuming without loss of generality that the nodes are numbered counter-clockwise. If $a_3b_1 - b_3a_1 = 0$ then $a_3b_1 = b_3a_1$,



(a) (b)

Figure 13: If any two node vectors $\vec{12}$, $\vec{23}$, $\vec{34}$, or $\vec{41}$ are parallel and point in the same direction, the nodal configuration is kinematically inadmissible.

$$[(y_{n2} - y_{n1}) + (y_{n3} - y_{n4})][(x_{n1} - x_{n2}) + (x_{n3} - x_{n4})] = \\ [(y_{n1} - y_{n2}) + (y_{n3} - y_{n4})][(x_{n2} - x_{n1}) + (x_{n3} - x_{n4})] \, .$$

Simplifying the expression yields,

$$(x_{n2} - x_{n1})(y_{n4} - y_{n3}) = (x_{n4} - x_{n3})(y_{n2} - y_{n1}) \, ,$$

$$\Rightarrow \frac{(y_{n4} - y_{n3})}{(x_{n4} - x_{n3})} = \frac{(y_{n2} - y_{n1})}{(x_{n2} - x_{n1})} \, .$$

Thus, the vectors $\vec{12}$ and $\vec{34}$ point to the same direction, which results in a kinematically inadmissible configuration (see 13(a)).

□

Proposition 3. *For all kinematically admissible elements, such that $a_1 + a_3\eta = 0$ and $a_3 = 0$, the inverse function $F^{-1}\colon \vec{x}(p) \to \vec{\xi}(p)$ is*

$$\xi = \tfrac{y_0 a_2 - b_2 x_0}{b_3 x_0 + a_2 b_1} \quad \textit{and} \quad \eta = \tfrac{x_0}{a_2} \, . \tag{57}$$

Proof. The proof follows directly from substitution into equation 54. It shows that if there is a division by zero in equation 57 the element is kinematically inadmissible.

part 1:

For all kinematically admissible elements, such that $a_1 + a_3\eta = 0$, and $a_3 = 0$, it is necessary that $a_2 \neq 0$. If $a_1 + a_3\eta = 0$, and $a_3 = 0$ then $a_1 = 0$, thus

$$(x_{n2} - x_{n1}) = (x_{n4} - x_{n3}) \text{ and } (x_{n1} - x_{n2}) = (x_{n4} - x_{n2}) \, ,$$

$$\Rightarrow x_{n3} = x_{n4} \text{ and } x_{n1} = x_{n2} \, .$$

If $a_2 = 0$ then

$$x_{n3} + x_{n4} = x_{n1} + x_{n2} \, ,$$

$$\Rightarrow x_{n1} = x_{n2} = x_{n3} = x_{n4} \, .$$

Hence, the resulting node coordinates are collinear, which represents a kinematically inadmissible element.

part 2:

Assume that $a_2 \neq 0$ and that $b_1 a_2 + b_3 x_0 = 0$. Using $x_0 = x - a_0$,

$$b_3 x = b_3 a_0 - b_1 a_2 \ ,$$

$$\begin{gathered}[(y_{n1} - y_{n2}) + (y_{n3} - y_{n4})]x = \\ \tfrac{1}{4}[(y_{n1} - y_{n2}) + (y_{n3} - y_{n4})][x_{n1} + x_{n2} + x_{n3} + x_{n4}] \\ -\tfrac{1}{4}[(y_{n2} - y_{n1}) + (y_{n3} - y_{n4})][(x_{n3} + x_{n4}) - (x_{n1} + x_{n2})] \ .\end{gathered}$$

case 1: $b_3 \neq 0$

Substituting $x_{n1} = x_{n2}$ and $x_{n3} = n_{n4}$ and simplifying yields,

$$\begin{gathered}[(y_{n1} - y_{n2}) + (y_{n3} - y_{n4})]x = [(y_{n1} - y_{n2}) + (y_{n3} - y_{n4})][x_{n1} + x_{n3}] \\ +[(y_{n2} - y_{n1}) + (y_{n3} - y_{n4})][x_{n3} - x_{n1}] \ .\end{gathered}$$

Solving for x,

$$x = \frac{(y_{n3} - y_{n4})x_{n1} + (y_{n1} - y_{n2})x_{n3}}{(y_{n3} - y_{n4}) + (y_{n1} - y_{n2})} \ .$$

Defining α and β, where $\alpha + \beta = 1$ yields,

$$x = \alpha x_{n1} + \beta x_{n3} \ .$$

For any internal point it is necessary that both $\alpha > 0$ and $\beta > 0$. However, if $y_{n3} - y_{n4} > 0$ it follows that $y_{n1} - y_{n2} < 0$ for an element with no intersecting edges. Hence, α and β always have opposite signs for a valid element, thus the point x is not an internal point.

case 2: $b_3 = 0$

It follows that

$$(y_{n3} - y_{n4}) + (y_{n1} - y_{n2}) = 0 \ ,$$

$$y_{n1} - y_{n2} = y_{n4} - y_{n3} \ ,$$

$$0 = (y_{n3} - y_{n4})x_{n1} + (y_{n1} - y_{n2})x_{n3} \ ,$$

$$0 = (y_{n2} - y_{n1})x_{n1} + (y_{n1} - y_{n2})x_{n3} \ ,$$

$$x_{n1} - x_{n3} = 0 \Rightarrow x_{n1} = x_{n3} \ ,$$

$$\Rightarrow x_{n1} = x_{n2} = x_{n3} = x_{n4} \ .$$

The result is a collinear nodal arrangement, which is kinematically inadmissible. □

C Selecting the virtual mesh grid size

The build time, $\hat{T}_b|_{vm}$, and time complexity, $\hat{T}_c|_{vm}$, are both directly proportional to the average number of finite elements in $\mathbf{e}^v$, N_k and the average number of virtual elements that share a reference to the same finite element, $\overline{N}_{ve}$, respectively (equations (47) and (52)). To determine the best compromise between search time and build time, we define the ratio α,

$$\alpha = \frac{\overline{A}_e}{dx \times dy} \ . \tag{58}$$

. Where $\overline{A}_e$ is the average area of the finite element's bounding-boxes and $dx \times dy$ the virtual mesh's grid step. Notice that as the grid step decreases towards zero (i.e. $\alpha \to \infty$), the numbers $\overline{N}_k \to 1$ and $\overline{N}_{ve} \to \infty$ (Figure 14(c)). Similarly, for an infinite mesh, as the grid step increases to infinity (i.e. $\alpha \to 0$), the numbers $\overline{N}_k \to \infty$ and $\overline{N}_{ve} \to 1$. The exact relationship between these numbers is highly dependent on the geometry of the finite elements mesh. An approximation of this relationship is obtained by neglecting mesh distortion (see Figure 14) and considering only their limiting behaviors,

$$\overline{N}_k \approx \frac{A_1}{\alpha} + A_2 = \frac{1}{\alpha} + 1 \tag{59}$$

$$\text{and } \overline{N}_{ve} \approx B_1\alpha + B_2 = \alpha + 1 \tag{60}$$

The proposed approximations satisfy the limiting behavior of α. In addition, for $\alpha = 1$, if we consider the possible offset of the virtual and real mesh, it is required that $\overline{N}_k$ and $\overline{N}_{ve}$ are bounded by $[1, 4]$, which is also satisfied by the choices of coefficients $A_i = 1, B_i = 1$ (see Figure C).

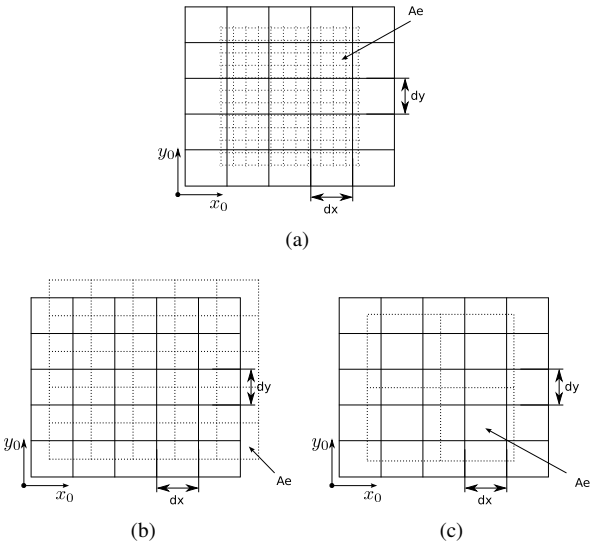


Figure 14: Different combinations of finite elements bounding-box grids (dotted lines) and virtual meshes.

Substituting equations (59) and (60) values into $T_t|_{ST3}$ yields an expression in the form of

$$\hat{T}_t|_{ST3} = \mathcal{A}N_p + \mathcal{B}\overline{N}_{cl}N_p + \mathcal{C}(\alpha)N_e + \mathcal{D}N_p\overline{N}_k(\alpha) + \mathcal{E} \ .$$

Differentiating the expression and solving for a zero, $\alpha*$, we obtain

$$\alpha^* = \sqrt{\frac{1.25N_p}{N_e}} \ . \tag{61}$$

Substituting this value into equation (53), results in the optimum processing time for the virtual mesh,

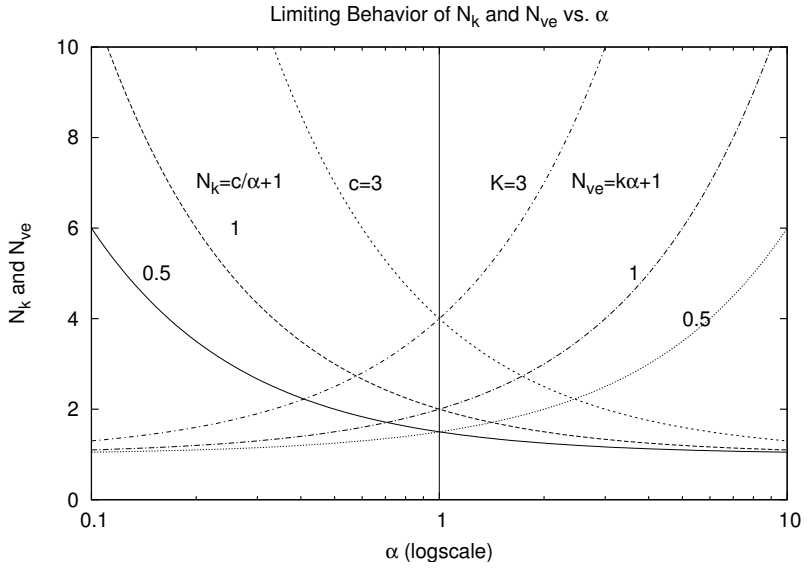


Figure 15: Plot of equations (59), and (60) in logscale. The plots include a variation of the coefficients to represent the limiting cases. Notice that the two equations are symmetric with respects to each other.

$$\hat{T}_t^*|_{ST3} = \left(\sqrt{\frac{1.25N_p}{N_e}} + 38\right) N_e + \left[6 + \frac{5}{4}\left(\sqrt{\frac{N_e}{1.25N_p}} + 1\right) + \frac{83}{4}\overline{N}_{cl} + \frac{442}{4}\right] N_p + 7\,. \qquad (62)$$

D Pseudo code for search algorithms

Algorithm D.1: CROSS PRODUCT TEST($Element\ E, Point\ P$)

comment: Scalar Cross-Product: Po(origin), P1(head), P2(head)

procedure SCP($Point\ Po, Point\ P1, Point\ P2$)
 {**return** $((P1.x - Po.x) * (P2.y - Po.y) - (P2.x - Po.x) * (P1.y - Po.y))$

procedure CROSSPRODUCTTEST($Element\ E, Point\ P$)
 if SCP($E.node[1], E.node[N_n], P$) * SCP($E.node[1], P, E.node[2]$) < 0
 then return $(false)$

 for $i \leftarrow 2$ **to** $N_n - 1$
 do {**if** SCP($E.node[i], E.node[i+1], P$) * SCP($E.node[i], P, E.node[i-1]$) < 0
 then return $(false)$

 if SCP($E.node[N_n], E.node[0], P$) * SCP($E.node[N_n], P, E.node[N_n - 1]$) < 0
 then return $(false)$

 return $(true)$

Algorithm D.2: INITIALIZEBOUNDINGBOX($Element\ E$)

comment: 4 Assignment Operations

$E.x_{min} = E.node[0].x$
$E.x_{max} = E.node[0].x$
$E.y_{min} = E.node[0].y$
$E.y_{max} = E.node[0].y$

comment: 3×4 Comparison + 6_{max} or 2_{min} Assignment Operations

for $i \leftarrow 1$ **to** 3
do
if $E.x_{min} > E.node[i].x$
then $E.x_{min} = E.node[i].x$
if $E.x_{max} < E.node[i].x$
then $E.x_{max} = E.node[i].x$
if $E.y_{min} > E.node[i].y$
then $E.y_{min} = E.node[i].y$
if $E.y_{max} < E.node[i].y$
then $E.y_{max} = E.node[i].y$

Algorithm D.3: ISINSIDETHEBOUNDINGBOX($Element\ E, Point\ P$)

if $P.x > E.x_{max}$
then return $(false)$
else if $P.y > E.y_{max}$
then return $(false)$
else if $P.x < E.x_{min}$
then return $(false)$
else if $P.y < E.y_{min}$
then return $(false)$
else return $(true)$

Algorithm D.4: GET VIRTUAL ELEMENT INDEX($Point\ P$)

comment: (1 Subtraction + 1 Division + 1 Cast Operation) $\times$ Dimension

procedure GET INDEX($Vector\ P, i, j$)
$i =$ FLOOR$((P.x - x_0)/dx)$
$j =$ FLOOR$((P.y - y_0)/dy)$

Algorithm D.5: INITIALIZE VIRTUAL MESH($Elements\ List\ V_e$)

procedure INITIALIZE VIRTUAL MESH($Elements\ List\ V_e$)
SETUP VIRTUAL MESH GEOMETRY(V_e)
for each $Element\ E \in V_e$
do INITIALIZEBOUNDINGBOX(E) $\#see\ D.2$
ADD FINITE ELEMENT TO VIRTUAL MESH(E)

Algorithm D.6: ADD FINITE ELEMENT TO VIRTUAL MESH($Element\ E$)

procedure ADD FINITE ELEMENT TO VIRTUAL MESH($Element\ E$)
- **comment:** Determining the VE index range for Element E
- GET INDEX($E.P_{min}, i_{min}, j_{min}$) $\#see\ D.4$
- GET INDEX($E.P_{max}, i_{max}, j_{max}$)
- **for** $i \leftarrow i_{min}$ **to** i_{max}
  - **do** **for** $j \leftarrow j_{min}$ **to** j_{max}
    - **do** $\{VirtualMesh[i][j] \leftarrow E$

Algorithm D.7: SETUP VIRTUAL MESH GEOMETRY($Elements\ List\ V_e$)

procedure SETUP VIRTUAL MESH GEOMETRY($Elements\ List\ V_e$)
- **comment:** Determining the boundary of the VM:
- $x_0 = x_f = x\ of\ first\ node\ of\ first\ element\ in\ V_e$
- $y_0 = y_f = y\ of\ first\ node\ of\ first\ element\ in\ V_e$
- **for each** $Element\ E \in Ve$ **and for each** $Node\ P \in E$
  - **do**
    - **if** $P.x < x_0$ **then** $x_0 = P.x$
    - **if** $P.x > x_f$ **then** $x_f = P.x$
    - **if** $P.y < y_0$ **then** $y_0 = P.y$
    - **if** $P.y > y_f$ **then** $y_f = P.y$
- **comment:** Determining the mesh steps
- $dx = (x_f - x_0)/(number\_of\_x\_divisions)$
- $dy = (y_f - y_0)/(number\_of\_y\_divisions)$