SANDIA REPORT
SAND2004-1617
Unlimited Release
Printed April 2004

Visualization of Higher Order Finite Elements

David Thompson, Richard Crawford, Rahul Khardekar, and Philippe P. Pébay

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation,
a Lockheed Martin Company, for the United States Department of Energy's
National Nuclear Security Administration under Contract DE-AC04-94AL85000.

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Available to DOE and DOE contractors from

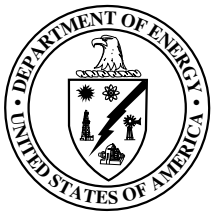
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865)576-8401
Facsimile: (865)576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: http://www.doe.gov/bridge

Available to the public from

U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone: (800)553-6847
Facsimile: (703)605-6900
E-Mail: orders@ntis.fedworld.gov
Online order: http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online



Contents

List of Symbols

Symbols are listed with roman letters followed by greek letters, followed by symbols with each group sorted so that uppercase letters precede lowercase. Note that we frequently take a lowercase letter to be a member of a set denoted by its corresponding uppercase letter. Maps denoted by a greek letter often have the corresponding uppercase roman letter as their ranges.

$A_{i,j,k}$	the coefficient of a term in a trivariate polynomial, called a *degree of freedom*
P	a set of points
R	the domain of some finite element (generically, as opposed to $\mathbf{\Omega}_{st}^{(t)}, \mathbf{\Omega}_{st}^{(q)}, \mathbf{\Omega}_{st}^{(T)},$ or $\mathbf{\Omega}_{st}^{(h)}$)
F	the range of some finite element's solution map, Φ
X	the range of some finite element's geometry map, Ξ. $X \subseteq \mathbf{\Omega}$
f	a value in some finite element's approximation of a solution. $f \in F$.
$\boldsymbol{r} = (\ r\ s\ t\ )$	a point in some finite element's parametric coordinates. $\boldsymbol{r} \in R$.
$\boldsymbol{x} = (\ x\ y\ z\ )$	a point in model coordinates. $\boldsymbol{x} \in X$.
Φ	the solution map, from R to the solution of some differential equation, F
Ξ	the geometry map, from R to model space, X
$\mathbf{\Omega}$	the problem domain in *model* space ($\bigcup_i X_i$)
$\bar{\mathbf{\Omega}}_{st}^{(t)}$	the domain of a standard triangle in *parameter* space
$\bar{\mathbf{\Omega}}_{st}^{(q)}$	the domain of a standard quadrilateral in *parameter* space
$\bar{\mathbf{\Omega}}_{st}^{(T)}$	the domain of a standard tetrahedron in *parameter* space
$\bar{\mathbf{\Omega}}_{st}^{(h)}$	the domain of a standard hexahedron in *parameter* space
ϕ	basis polynomials defined by Szabo as integrals of Legendre polynomials
σ	a simplex, usually a tetrahedron
τ	a 2-simplex (triangle)

1 Introduction, Background, and Motivation

Finite element meshes are used to approximate the solution to some differential equation when no exact solution exists. A finite element mesh consists of many small (but finite, not infinitesimal or differential) regions of space that partition the problem domain, $\mathbf{\Omega}$. Each region, or *element*, or *cell* has an associated polynomial map, Φ, that converts the coordinates of any point, $\boldsymbol{x} = (\ x\ y\ z\ )$, in the element into another value, $f(\boldsymbol{x})$, that is an approximate solution to the differential equation, as in Figure 1(a). This representation works quite well for axis-aligned regions of space, but when there are curved boundaries on the problem domain, $\mathbf{\Omega}$, it becomes algorithmically much more difficult to define Φ in terms of $\boldsymbol{x}$. Rather, we define an archetypal element in a new coordinate space, $\boldsymbol{r} = (\ r\ s\ t\ )$, which has a simple, axis-aligned boundary (see Figure 1(b)) and place two maps onto our archetypal element:

- the map from the new coordinate space (*parametric space*), R, to the solution space, F, $\Phi : R \to F$ and
- the map from the new coordinate space, R, to the original (*model space*) coordinates of $\mathbf{\Omega}$ $\Xi : R \to X \subseteq \mathbf{\Omega}$.

In other words, $f = \Phi \circ \Xi^{-1}$; this expression is meaningful, because Ξ is indeed an homeomorphism. For $\boldsymbol{r} \in R$, the approximate solution to our differential equation over a single element takes the following form

$$\Phi(\boldsymbol{r}) = A_{0,0,0} + A_{1,0,0}r + A_{0,1,0}s + A_{0,0,1}t + A_{1,1,1}rst + \cdots$$

Each coefficient, $A_{i,j,k}$, is called a degree of freedom. A degree of freedom (DOF) is a generalization of a nodal value; nodal values define a polynomial map, but they are interpolated by the polynomial map. Degrees of freedom are not interpolated, but they do define the polynomial map. For instance,

EXAMPLE 1.1. Let the equation

$$\Phi(r) = P_0 r + (1 - r)P_1 \tag{1}$$

be written as

$$\Phi(r) = A_0 + A_1 r, \qquad \text{s.t.} \qquad A_0 = P_1, A_1 = P_0 - P_1. \tag{2}$$

Both versions are the same, but (1) is written so that P_0 and P_1 are interpolated on $[-1, 1]$, while (2) only interpolates A_0 on $[-1, 1]$.

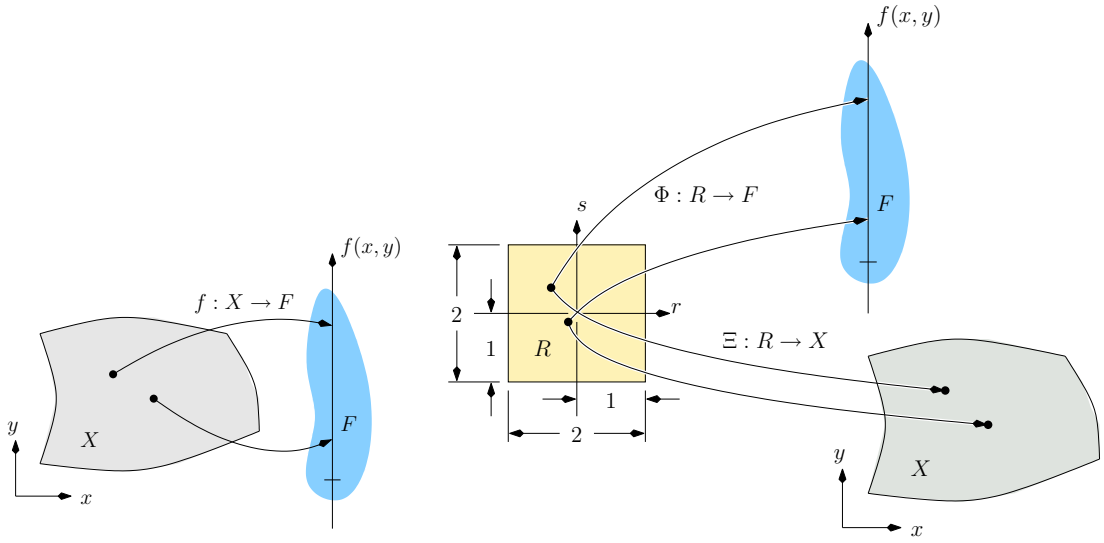
Just as (1) is a polynomial mapping our archetypal element to field values, there is a map for the geometry:

$$\Xi(\boldsymbol{r}) = \boldsymbol{A}_{0,0,0} + \boldsymbol{A}_{1,0,0}r + \boldsymbol{A}_{0,1,0}s + \boldsymbol{A}_{0,0,1}t + \boldsymbol{A}_{1,1,1}rst + \cdots$$

The degree of Φ and Ξ may be different, and if there are several Φ defined over $\mathbf{\Omega}$, then their degrees may differ. When different fields are interpolated with polynomials of differing degree, this is called *mixed-order* interpolation. In practice, the degree of any of these maps is allowed to vary from element to element.

1.1 Grouping Modal Shapes Into Nodes

The coefficients that define a map (Φ or Ξ) from a finite element's parameter space to model space may be grouped by the region of the cell where they have some effect. For instance, some coefficients correspond to a term that disappears on every boundary except a single face of the element; this coefficient is thus called a *face mode*. An example on a parametric element with domain $[-1, 1]^3$ would be the coefficient of $(s^2-1)(t^2-1)(r+1)$, which disappears on every face except $r = 1$. Other terms vanish everywhere except the interior of the cell – *volume modes* – such as $(r^2-1)(s^2-1)(t^2-1)$. There are also *corner modes* that disappear at every vertex of the cell save one, and *edge modes* that disappear at every edge save one. Notice that these modes do *not* disappear everywhere except their corresponding vertex, edge, or face; rather, they disappear at every other boundary element of the same or lower dimensionality. So, corner modes are not zero along edges, only at other corners. Edge modes vanish at other edges and all corner nodes. Face modes are zero on all but one face and on



(a) A single map from the problem domain to F can be difficult to specify.

(b) Instead, we generate 2 parametric maps from an archetypal element.

Figure 1: Two ways to define a solution field on a finite element.

all edges and corners. Volumetric modes are zero on all boundary facets – faces, edges, and corners. Figure 2 shows these groupings. Each set of coefficients associated with a particular boundary are called a *node*. The number of nodes per finite element depends only on the shape of the element: hexahedra have 8 corner nodes, 12 edge nodes, 6 face nodes, and a volume node. Tetrahedra have 4 corner nodes, 6 edge nodes, 4 face nodes, and a volume node. Since the number of coefficients for each type of node (corner, edge, face, or volume) varies with the polynomial degree of the map and the product space, a *node* may have several *mode* shapes – except corner nodes, which have a single coefficient. Figure 3 shows a concrete example of a polynomial interpolant broken down into nodes and their associated modes. Figures 4 and 5 show edge mode shapes and face mode shapes, respectively. We only consider volumetric interpolants with degree 6, so there is only a single volumetric mode shape and it is shown in Figure 2. The exact shape functions we employ are the hexahedral and tetrahedral hierarchic shape functions described by Szabo and Babuska [1991], which are scaled differently than those in Figure 2 but otherwise similar.

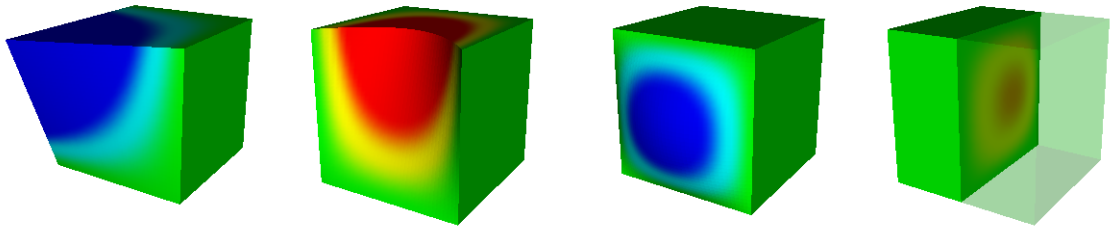


Figure 2: Coefficients of a map may be grouped by the locations where they have an effect.From left to right: a point mode, an edge mode, a face mode, and a volumetric mode.

$$
\begin{aligned}
\Phi(\boldsymbol{r}) \quad = \quad & \left.\begin{array}{l}
A_{c0}(1+r)(1+s)(1+t)+ \\
A_{c1}(1-r)(1+s)(1+t)+ \\
A_{c2}(1-r)(1-s)(1+t)+ \\
A_{c3}(1+r)(1-s)(1+t)+ \\
A_{c4}(1+r)(1+s)(1-t)+ \\
A_{c5}(1-r)(1+s)(1-t)+ \\
A_{c6}(1-r)(1-s)(1-t)+ \\
A_{c7}(1+r)(1-s)(1-t)+
\end{array}\right\} \text{Corner nodes, 1 mode per node} \\
& \left.\begin{array}{l}
\left.\begin{array}{l}
A_{e4,0}(r^2-1)(1-s)(1-t)+ \\
A_{e4,1}r(r^2-1)(1-s)(1-t)+
\end{array}\right\} \text{Edge node 0, 2 modes} \\
\left.\begin{array}{l}
A_{e5,0}(r^2-1)(1+s)(1-t)+ \\
A_{e5,1}r(r^2-1)(1+s)(1-t)
\end{array}\right\} \text{Edge node 1, 2 modes} \\
\left.\begin{array}{l}
A_{e0,0}(r^2-1)(1+s)(1+t)+ \\
A_{e0,1}r(r^2-1)(1+s)(1+t)+
\end{array}\right\} \text{Edge node 2, 2 modes} \\
\left.\begin{array}{l}
A_{e1,0}(r^2-1)(1-s)(1+t)+ \\
A_{e1,1}r^2(r^2-1)(1-s)(1+t)+
\end{array}\right\} \text{Edge node 3, 2 modes} \\
\left.\begin{array}{l}
A_{e2,0}(1-r)(s^2-1)(1-t)+ \\
A_{e2,1}(1-r)s(s^2-1)(1-t)+
\end{array}\right\} \text{Edge node 4, 2 modes} \\
\left.\begin{array}{l}
A_{e3,0}(1-r)(s^2-1)(1+t)+ \\
A_{e3,1}(1-r)s(s^2-1)(1+t)+
\end{array}\right\} \text{Edge node 5, 2 modes} \\
\left.\begin{array}{l}
A_{e4,0}(1+r)(s^2-1)(1+t)+ \\
A_{e4,1}(1+r)s(s^2-1)(1+t)+
\end{array}\right\} \text{Edge node 6, 2 modes} \\
\left.\begin{array}{l}
A_{e5,0}(1+r)(s^2-1)(1-t)+ \\
A_{e5,1}(1+r)s(s^2-1)(1-t)
\end{array}\right\} \text{Edge node 7, 2 modes} \\
\left.\begin{array}{l}
A_{e0,0}(1-r)(1-s)(t^2-1)+ \\
A_{e0,1}(1-r)(1-s)t(t^2-1)+
\end{array}\right\} \text{Edge node 8, 2 modes} \\
\left.\begin{array}{l}
A_{e1,0}(1+r)(1-s)(t^2-1)+ \\
A_{e1,1}(1+r)(1-s)t(t^2-1)+
\end{array}\right\} \text{Edge node 9, 2 modes} \\
\left.\begin{array}{l}
A_{e2,0}(1+r)(1+s)(t^2-1)+ \\
A_{e2,1}(1+r)(1+s)t(t^2-1)+
\end{array}\right\} \text{Edge node 10, 2 modes} \\
\left.\begin{array}{l}
A_{e3,0}(1-r)(1+s)(t^2-1)+ \\
A_{e3,1}(1-r)(1+s)t(t^2-1)+
\end{array}\right\} \text{Edge node 11, 2 modes}
\end{array}\right\} \text{Edge nodes} \\
& \left.\begin{array}{l}
\left.\begin{array}{l}
A_{f0,0}(1+r)(s^2-1)(t^2-1)+ \\
A_{f0,1}r^2(1+r)(s^2-1)(t^2-1)+
\end{array}\right\} \text{Face node 0, 2 modes} \\
\left.\begin{array}{l}
A_{f1,0}(1-r)(s^2-1)(t^2-1)+ \\
A_{f1,1}r^2(1-r)(s^2-1)(t^2-1)+
\end{array}\right\} \text{Face node 1, 2 modes} \\
\left.\begin{array}{l}
A_{f2,0}(r^2-1)(1+s)(t^2-1)+ \\
A_{f2,1}(r^2-1)s^2(1+s)(t^2-1)+
\end{array}\right\} \text{Face node 2, 2 modes} \\
\left.\begin{array}{l}
A_{f3,0}(r^2-1)(1-s)(t^2-1)+ \\
A_{f3,1}(r^2-1)s^2(1-s)(t^2-1)+
\end{array}\right\} \text{Face node 3, 2 modes} \\
\left.\begin{array}{l}
A_{f4,0}(r^2-1)(s^2-1)(1+t)+ \\
A_{f4,1}(r^2-1)(s^2-1)t^2(1+t)+
\end{array}\right\} \text{Face node 4, 2 modes} \\
\left.\begin{array}{l}
A_{f5,0}(r^2-1)(s^2-1)(1-t)+ \\
A_{f5,1}(r^2-1)(s^2-1)t^2(1-t)
\end{array}\right\} \text{Face node 5, 2 modes}
\end{array}\right\} \text{Face nodes} \\
& \left.\begin{array}{l}
A_{v,0}(r^2-1)(s^2-1)(t^2-1) \\
A_{v,1}r(r^2-1)s(s^2-1)t(t^2-1)
\end{array}\right\} \text{Volume node, 2 modes}
\end{aligned}
$$

Figure 3: A concrete example of a polynomial interpolant broken down into corner, edge, face, and volumetric nodes.Note that this particular interpolant isn't used by any finite element code – it's just an example.

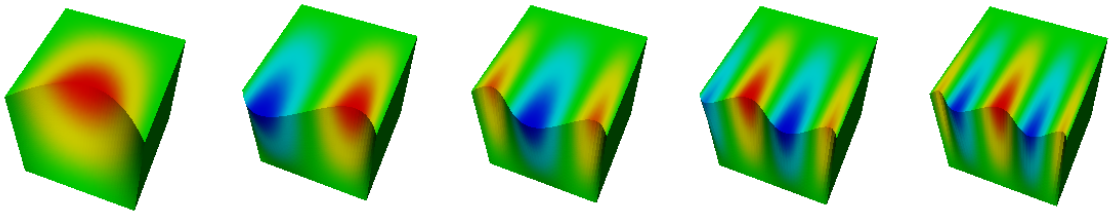


Figure 4: Each edge node may contain coefficients corresponding to different modes. These edge modes have degree 2 through 6 – total degree 4 through 8 when you include linear terms for the other 2 parametric coordinates.

1.2 Refining a Map

A solver algorithm may decide that the truncation error of a finite element's polynomial map is too large – that is, the polynomial degree of the map may not be high enough to closely approximate the exact solution to the differential equation over the entire finite element's domain, X. When this happens, a solver may

1. punt, and hope that the inaccuracy doesn't cause problems for anyone,
2. increase the degree of the polynomial map (called *polynomial refinement* or *p-refinement*), or
3. increase the number of finite elements in the region X (called *hierarchical refinement* or *h-refinement*, since the original h-refinement scheme involved splitting one element into several children that take its place).

Option 1 is obviously of limited value. Options 2 and 3 are often combined into an approach called *hp*-adaptive, where either option may be taken on a case by case basis. *hp*-adaptive algorithms can converge with many fewer DOF than either option alone *for certain classes of differential equations*.

1.3 Traditional Visualization Techniques

Once the solver has computed some finite element solution, analysts must interpret the meaning of all the $A_{i,j,k}$ that form the result in order to

- debug the mesh geometry, boundary conditions, solver parameters, or even the solver itself;
- compare a simulation to experimental results; and
- understand the behavior of the differential equations as it relates to the design goals.

Traditionally, these tasks are accomplished using a few standard visualization techniques:

- rendering the 2D boundary of a 3D mesh,
- coloring a 2D surface with a scalar value,
- deforming a mesh by a vector field,
- drawing hedgehog diagrams to show a vector field evaluated at some set of points,
- tracing streamlines and streaklines,
- computing a level set of Φ,
- volume rendering (ray tracing that assumes the solid model is translucent with volumetric effects),
- rendering the intersection of the mesh with a cutting surface,

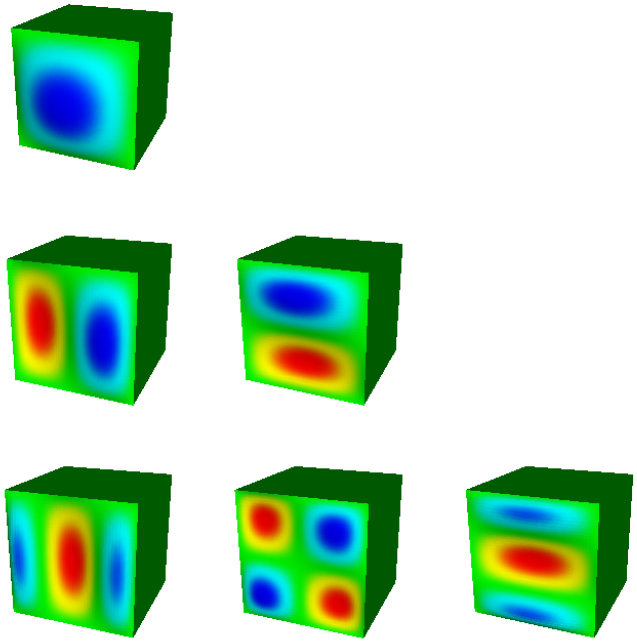


Figure 5: Each face node may contain coefficients corresponding to different modes. These face modes have degree 4 (top row) through 6 (bottom row) – total degree 5 through 7 when you include linear term for the other parametric coordinate.

- rendering a mesh clipped by a cutting surface.

Many of these techniques can be adapted from linear functions without much trouble. However, if care is not taken when an algorithm for linear meshes is applied to a higher order mesh, subtle issues may arise. Consider the first two techniques in the list above; many visualization packages currently render 20-node (quadratic) hexahedra as 8 linear hexahedra, as shown in Figure 6(a). However, a finer sampling of the geometric and solution maps (see Figure 6(b)) is required to show the features of the element's maps.

The next section describes how the mesh is stored in memory and why it is stored that way instead of some other way. After that, we'll move on to algorithms that visualize the mesh.

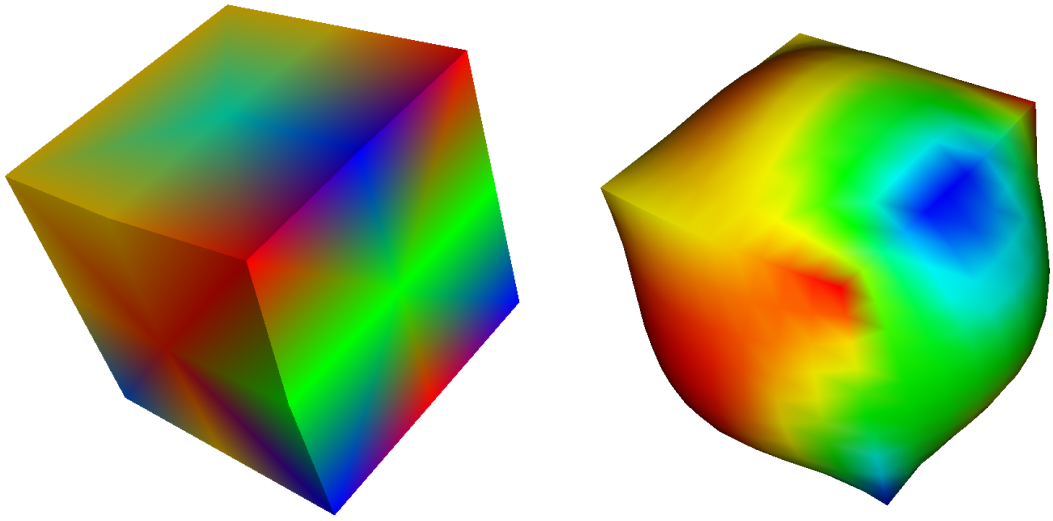
2 Mesh Representation

Currently, VTK has limited facilities for higher order finite elements; classes exist for several quadratic elements but facilities to render these objects in a way that guarantees the fields are properly presented do not exist. Also, there is no way to handle mixed-order interpolation and the mesh representation does not provide a way to handle $h-$ or $p-$refinements that will scale well with the number of elements or refinements. The goal of this section is first to discuss the issues involved, present our design decisions, and finally to describe the final design in detail.

2.1 What's In A Mesh?

We'll use VTK's current unstructured mesh class, vtkUnstructuredGrid, as a baseline to compare against our new mesh, which we'll call vtkShoeMesh. In a vtkUnstructuredGrid, space is reserved for

- point coordinates;
- cell connectivity;



(a) Typical rendering of a quadratic hexahedron as 8 linear hexahedra

(b) More accurate rendering of a quadratic hexahedron (10 samples per edge)

Figure 6: Two renderings of the same, 20-node hexahedral finite element

- cell type and offset into the connectivity array;
- field data over points, cells;
- arbitrary field data;
- and, optionally, reverse lookups (cells that reference a given point).

See Figure 7 for an illustration of how these fields reference each other. Storing meshes this way makes file I/O and cell insertion very fast. However, h-refinements can delete a large number of cells from a mesh when a region is being coarsened. Thus cell deletion speed will be an important benchmark.

For a given field defined on the mesh, there are DOF in addition to those defined by linearly interpolated cells. Our new mesh class must store these extra polynomial coefficients. Recall from §1.1 that these extra DOF for polynomial modes may be grouped into *nodes* (see Figure 2). We can think of these nodes as extra entries in each cell's connectivity array (see Figure 8). The number of DOF nodes and their physical significance (edge, face, or volume modes) are implicitly specified by the shape of the finite element. Thus, DOF nodes may be thought of as an extended sort of vtkPoints (for geometric data) or vtkPointData (for field data), with the exception that they *explicitly* specify a polynomial function over the cell rather than a single value at a point (which has an *implicit* linear interpolant over the cell). The other difference between DOF nodes and vtkPoints or vtkPointData is that DOF nodes may be defined by 0, 1, or many tuples while points and point data are always defined by a single tuple.[1]

By storing coefficients grouped into nodes, we may share the coefficients among cells with common edges and faces, just as points may be shared between linear cells. Being able to share edge and face coefficients is important because the number of values to be shared increases very quickly with the degree of the polynomial interpolant. In the case of a tensor product of degree d polynomials in r, s, and t, a hexahedral cell will have a total of d^3 coefficients, of which $6d^2 + 4d + 8$ are on the element boundary and could be shared. In general, the number of shareable coefficients will grow

[1] The ordering of these tuples in a DOF node is both important and confusing. See §2.5

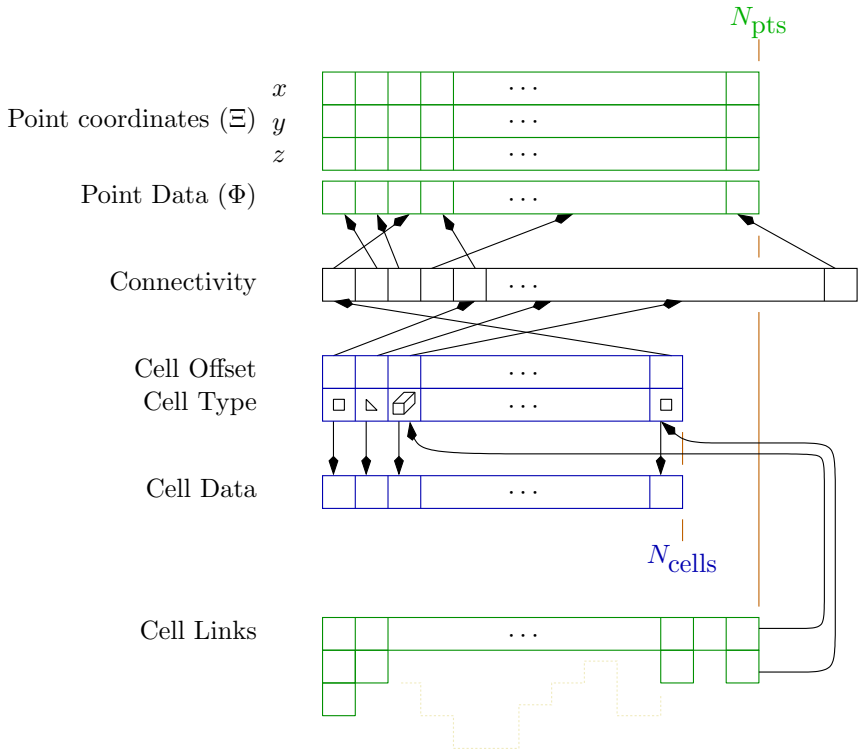


Figure 7: Data stored in a vtkUnstructuredGrid.

with the square of the degree of the polynomial interpolants and with the square of the number of cells (assuming a brick-like topology). This is a strong incentive to share coefficients between cells. While Sharing coefficients reduces the amount of storage required, it increases the amount of bookkeeping that must be performed, since we must keep track of the number of cells referencing a given DOF node's coefficients.

2.2 Reverse Lookup Of Connectivity Information

To reduce the cost of the accounting required when sharing coefficients, we could keep a reverse lookup table for DOF nodes analogous to the vtkCellLinks structure of the vtkUnstructuredGrid. This table would contain, for each DOF node, the list of cells that refer to that DOF node in their connectivity array. An added benefit of this reverse lookup table is that finding cells that share an edge or face would be much faster using the vtkShoeMesh compared to the vtkUnstructuredGrid, since this is now accomplished with a single lookup compared to lookups for all the vertices of the face or edge followed by the intersection of the sets of cells sharing the vertices of the edge or face. The question we must answer, is how will this storage scale with polynomial degree and the number of cells in the mesh?

For instance, do we really save space if we keep a reverse lookup table, or will it grow as fast as if we just store redundant coefficient data? Luckily, the reverse lookup table does not grow with the polynomial degree of cells, since modes are grouped into nodes. There is a linear storage cost per DOF node and per cell referencing each DOF node. Although cells mays have different numbers of DOF nodes, based on their types, the number of DOF nodes is $O(N)$ in the number of cells. We may just focus on the number of cells referencing a particular DOF node. For face nodes, there will be at most 2 cells referencing a node (in a manifold dataset, and generally not many more for non-manifold cases such as 2-D elements embedded into a 3D mesh). So face DOF nodes are approximately a constant storage cost, $O(1)$. Volume nodes may only be referenced by a single cell (in 3D, anyway). For edge nodes, the number of cells referencing an edge may be arbitrarily high.

This occurs when many cells share 2 vertices. This must, by definition, occur no more often than when cells share a single vertex. Because vtkUnstructuredGrid's CellLinks data structure records the number of cells sharing a single vertex, we know from practical experience that our reverse lookup table will be space-efficient. The only theoretical bound is a fully connected graph in which every cell references every DOF node, which is clearly unrealistic – especially for meshes of any size. A useful heuristic bound may be had by considering the included angle of faces attached to an edge shared by many cells. We can assume we are storing meshes of good quality since no meshing tool will ever produce a thin triangle or a sliver tetrahedron[2].

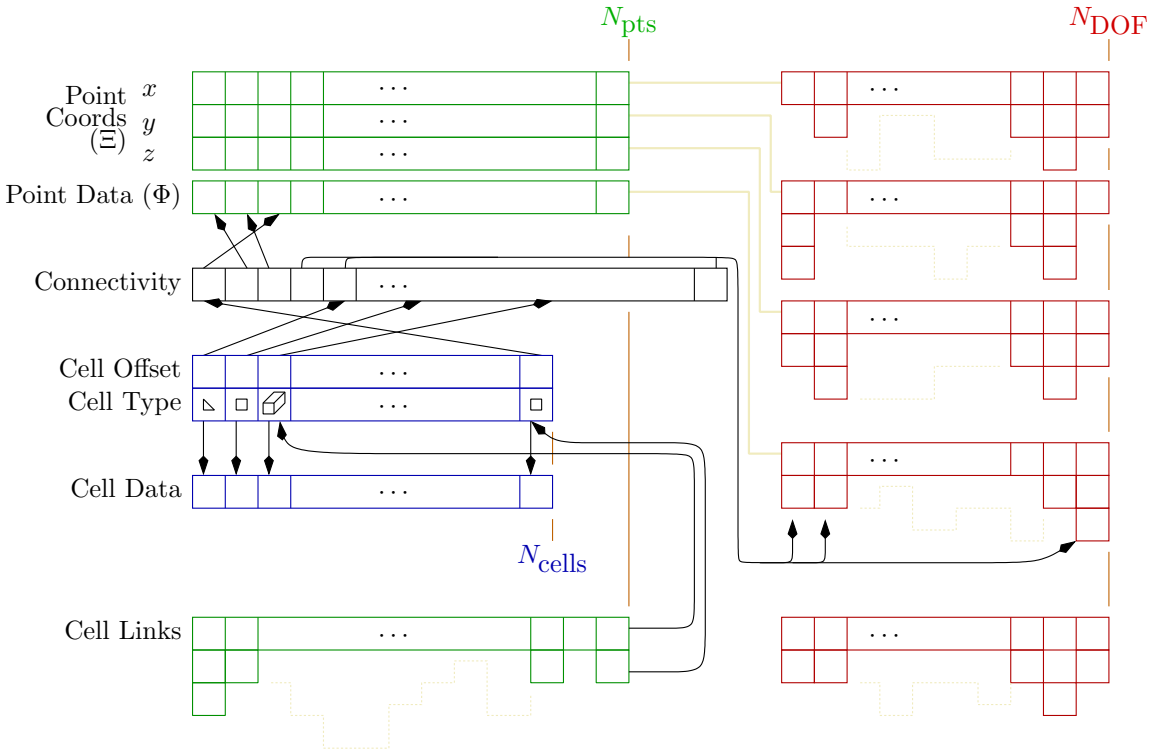


Figure 8: Data stored in a vtkShoeMesh.

Another possible problem with the lookup table is the computational complexity of maintaining it. Whenever a cell is inserted into or removed from the mesh, the table must be updated. Assuming the reverse lookup table is an unsorted linear array, this will require a search of the list for the cell – $O(n)$ in the number of entries for that DOF node – and a deletion operation – $O(n)$ in the number of entries for that DOF node. Again, for volume and face nodes, this cost will be constant but for edge nodes, it may be arbitrarily high.

Thus the lookup table appears to be better, at least for the average case, in both storage and computational efficiency.

2.3 Refinement and Coarsening Operations

The unstructured mesh class must also be able to accommodate h- and p-adaptation of mesh cells, in a simple and efficient way. Regarding h-adaptation, several types of methods can be considered. One approach consists in entirely remeshing the domain of interest from a discretization of its boundary, the latter being itself rediscretized with respect to the adaptation requirements. Such methods will be handled later, because they are not likely to be the most relevant for SHOE applications. We will also ignore, at least for now, specific techniques such as multigrid and non conforming methods.

[2]For those of you without a sense of humor, this is a joke.

Another class of h-adaptation methods is based on local modifications of the existing mesh, in order to follow size and, possibly, direction requirements. Such techniques are fast and it is, in addition, easy to keep track of the operations performed on the mesh.

Local h-refinement can be done in particular by the means of the following modifications of the mesh to be adapted:

- refinement around a vertex: all edges connected to a given vertex are split in two sub-edges, *e.g.*, by midpoint insertion. The elements sharing these edges are subdivided on the fly, in order to preserve mesh conformity (but the element subdivision strategy is not unique);
- element refinement: a given element is split by the means of a subdivision template, *e.g.*, a triangle is split in two sub-triangles by inserting the midpoint of its longest edge. Another approach, for simplicial meshes, is based on Delaunay insertion of vertices inside the elements to be refined.

p-refinement methods are very efficient for some classes of problems, in particular for those defined over domains with relatively simple geometries. However, it is rather delicate to implement them in the context of complex geometries, because the suitability of a boundary mesh element to make an order p finite element, with the desired p (p-compatibility), does not hold in general. A combination of h- and p-refinement (hp-refinement) may be the only way to achieve the required element order, and can be summarized as follows in the case of any given boundary mesh element K (for internal element, the method is immediate):

1. attempt to p-refine (either at once of by unit order increments) K;
2. in case of failure, h-refine the intersection of the mesh boundary and K;
3. start over, until p-refinement is achieved for all specified elements.

2.4 Cell Type, Definition, and Specification

SHOE groups cells by their common features. There are three "levels" of specification allowed: cell type (most general), cell definition, and cell specification (most specific). The most general way to group cells is to consider cells of the same *type*. The type of a cell is:

- the shape of its parametric space domain (e.g., hexahedral, tetrahedral, and so on),
- the family of polynomial interpolants used to approximate a function over the element. This is a set of polynomials that form the basis for all possible maps from the parametric space into model space. Examples include Lagrange and Legendre polynomials;
- the product space used to combine univariate polynomials into multivariate polynomial basis functions. This is a rule for deciding which terms (of the infinite number of polynomials to choose from) should be included in the basis of the map from parametric to model space. This rule is usually something like, "Take the tensor product of univariate polynomials of degree 1 to n", or "Take any product of univariate polynomials such that the total degree is less than n", and so on.

In fact, the cell type contains the standard description of a finite element as the ordered triple (K, P_K, Σ_K). It goes a long way towards specifying a cell, but there's still information needed. SHOE allows you to group cells by a *definition* that is more complete than just specifying type, but still not a complete specification of a cell. A cell definition contains cell type information plus

- The order of each scalar field (function) defined on the cells. This includes the geometric map (x, y, and z as functions of r, s, and t) which is always present and any other functions you wish to add (stress, strain, temperature, etc.). The order of any single field is always specified as a triple of integers; these integers may be interpreted differently by different product spaces. For example, the tensor product space uses each integer as the maximum degree of polynomial in each r, s, and t coordinate, respectively. The maximum total degree product space uses only the first integer as the total order.

- A list of constants describing the cell type and functions that can operate on cells defined by the cell type. The constants stored include the number of *corner* points that define the cell, the number of edges and faces on the boundary of the cell, and a text description of the cell definition (such as "Hexahedron, Legendre Interpolants, Tensor Product") This is stored as a pointer to a structure (named vtkCellOps). It is stored in the cell definition rather than the cell type so that development of cells that have operations specific to their interpolant order is possible. We feel that such a storage strategy would result in a large number of very similar structures that would waste memory and require significant maintenance, so we did not pursue this optimization.

Finally, a complete cell *specification* includes all of the above plus references to all the remaining cell-specific data:

- The offset of the cell in the connectivity array,
- A set of bits that specify the permutation of any higher order modal coefficients relative to their storage order (this is discussed in more detail in Section 2.5).

2.4.1 Abusing the system

Although cell definitions appear to require all fields use the same family of polynomial interpolants, there are ways around this. The most elegant solution would of course involve rewriting the mesh class. However, the existing code may be used if you are willing to define a new cell type whose product space uses the integers that specify a field's order to also specify a polynomial interpolant. For example, you might define a product space that used a tensor product when the first integer in a field order is negative and a maximum total degree product space when the first integer in the field order triple is positive. Since each field has its own field order, each could have its own interpolant.

The current mesh model also assumes that vector and tensor fields should have each component interpolated using the same interpolant and order. It is possible in VTK to create a set of scalar fields from a vector field. Thus, by decomposing a vector or tensor field into multiple scalar fields, you may apply different interpolant orders and (using the hack described above) even different polynomial families.

2.5 Node Permutations

Each finite element cell stores a 32-bit integer containing a specification of the coordinate transform to move the shape functions from their storage order into the cell's order.

2.5.1 Rectangular Domains

Let's consider the example shown in Figure 9, where we are trying to construct a cell (cell 2) that shares a face with an existing cell. The two cells, cell 1 and cell 2, both refer to the same DOF node for their shared face. On cell 1, the shared face corresponds to the cell's 3rd face (labelled C). On cell 2, it is the first face (labelled A). Let's say that the face node has 3 DOF: $\{ d, e, f \}$. Cell 1 uses these for interpolating a field F like so:

$$F = \ldots + dN_{2,1,2} + fN_{3,1,2} - eN_{2,1,3} + \ldots$$

with $N_{2,1,2} = \frac{1}{2}(1-s)\phi_2(r)\phi_2(t)$, $N_{3,1,2} = \frac{1}{2}(1-s)\phi_3(r)\phi_2(t)$, and so on. The shape functions are linear in s, indicating that the DOF correspond to a face mode in cell 1's $r-t$ plane (the figure shows $s=-1$). Note that e and f have been swapped and e has been negated. This implies that the storage order of the DOF is in reference to a different coordinate system than the cell.

We can generate a linear, 2×2, orthonormal transformation matrix to change from the DOF storage order into the cell's coordinates:

$$\begin{pmatrix} r_1 \\ t_1 \end{pmatrix} = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} r_{store} \\ t_{store} \end{pmatrix}$$

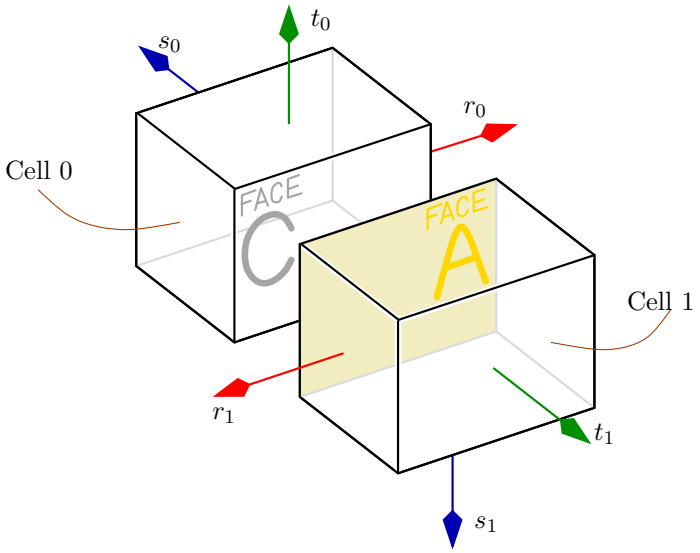


Figure 9: An example of the use of DOF node permutations stored with each cell.

where r_{store}, t_{store} are the storage coordinates and r_1, r_1 are the cell coordinates. Think of this transformation as a change of basis from one coordinate system (storage) to another (cell 1).

$N_{2,1,2}$ will be unchanged by the transformation because $\phi_n(x)$ are symmetric about the origin for even n (*i.e.*, $n = 2i$). Shape functions with odd n (*i.e.*, $n = 2i + 1$) are symmetric about $x = y$, so $\phi_{2i+1}(-x) = -\phi_{2i+1}(x)$. This property can be used to show that $eN_{3,1,2}(r_{store}) = -eN_{2,1,3}(t_1)$. Furthermore, given the transform, we can simply adjust the order and signs of the storage-order shape function coefficients before multiplying by cell-order shape functions to interpolate a value for F. This is a handy property because we can cache permuted coefficients from all DOF nodes for the entire cell and avoid the cost of applying transformations for each face and edge node each time F is evaluated.

In order to construct cell 2, we need to calculate the transformation between the storage coordinate frame of the shared face's DOF node and cell 2's coordinate frame. Unfortunately, there is no information on the coordinate frame stored with the DOF node – it is implicit. We're not lost, though, because we have a transformation from the storage coordinates to cell 1 and we can build a transformation from cell 1 to cell2.

The coordinate transform in the example above is one of 8 possible transforms for a quadrilateral face DOF node: there are 2 transforms associated with each corner vertex of the face. A hexahedron has 6 quadrilateral faces (3 bits each) and 12 edges (1 bit each), which means that all the permutations for a hexahedron (of any order) will fit into a 32-bit integer. Triangular faces have 6 possible permutations (still 3 bits), so wedge cells need 24 bits, pyramidal cells need 23 bits, tetrahedra need 18 bits. Octahedra need 36 bits, which is a bummer, but no one really uses those anyway, right? Each of the unique transformations for a hexahedron is assigned a 3 bit code. This code indicates which parametric coordinates to swap and negate. The codes are illustrated in Figure 10. These codes can also be concatenated very easily. Thus, using the example above, we can assign a code to face A on cell 2 relative to face C on cell 1. Then, knowing the code for face C on cell 1, we can modify the code for face A in cell 2 such that it transforms the stored DOF node coefficients directly, again by appropriate swaps and negations.

Each cell has a coordinate frame that is specified by the ordering of the corner vertices in the connectivity array. All of the permutation bits stored with the cell are used to convert from each DOF node's storage frame into that *cell*'s coordinate system. It is important to realize that given a DOF node for face C on cell 1, the permutation bits do *not* specify a transformation from the storage frame into a coordinate frame associated with face C, but rather cell 1's coordinate frame. *All of*

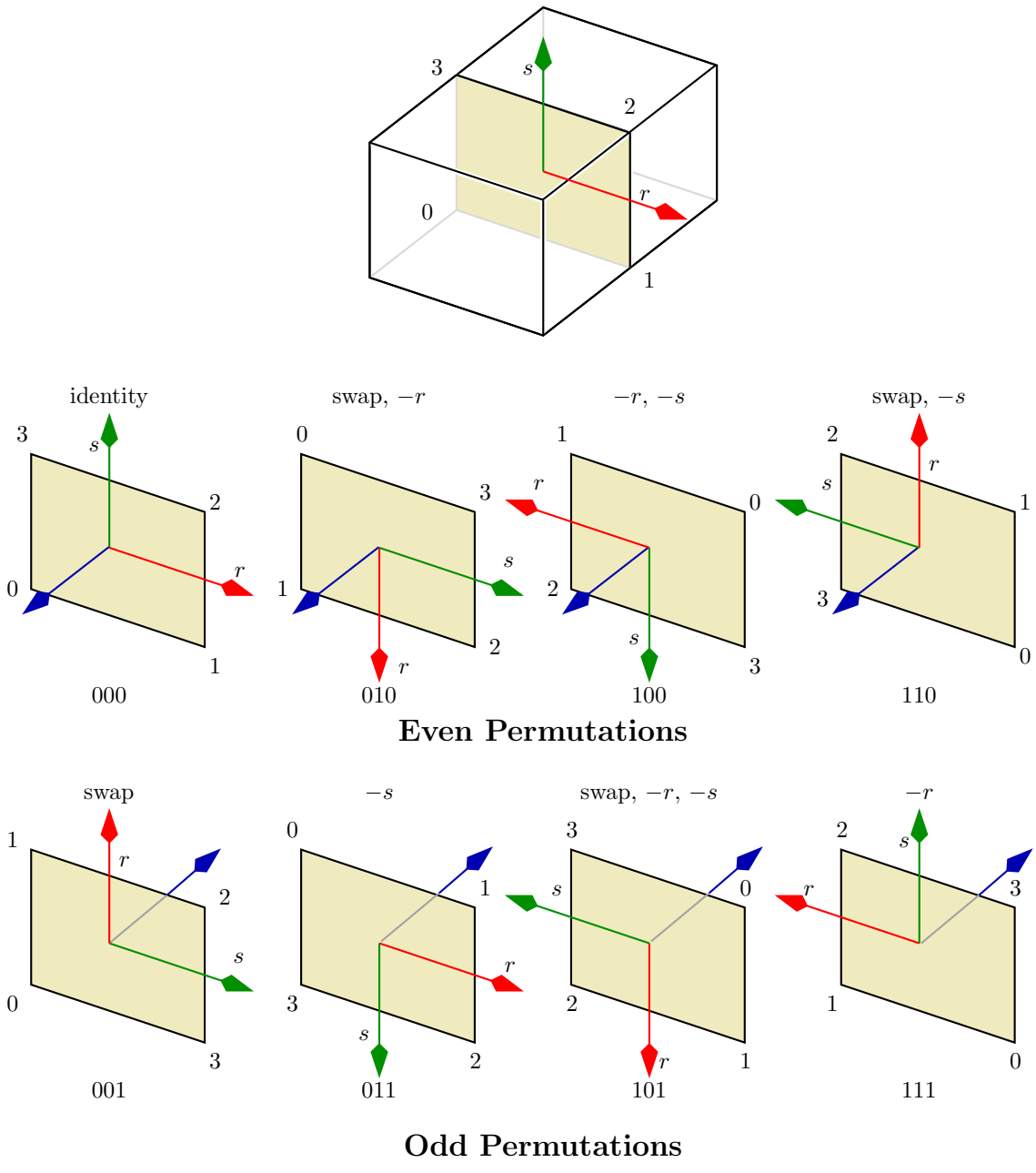


Figure 10: Codes for unique face node transformations for hexahedra.

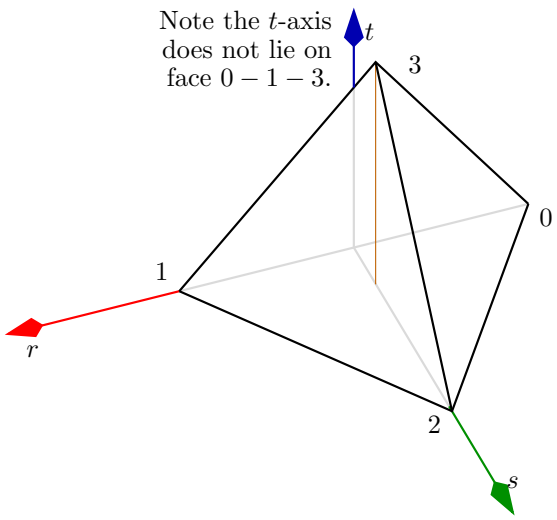


Figure 11: Coordinate system for a tetrahedron.

the shape function coefficients are transformed from an arbitrary storage coordinate frame into the cell's coordinate frame. Again, for hexahedral elements, the transformation amounts to swapping and negating selected coefficients.

2.5.2 Simplicial Domains

For simplices, and tetrahedra in particular, the transformations are not as simple, since three of the four faces on a tetrahedron are not parallel to any principle parametric coordinate plane. However, the shape functions for tetrahedra are written in terms of barycentric coordinates associated with the corner vertices. The values of the barycentric coordinates vary from 1 at the associated vertex to zero at the opposite face. Thus, the transformations for tetrahedral face nodes can be performed by swapping appropriate barycentric coordinates in the shape functions.

The tetrahedral coordinate system is shown in Figure 11. The three parametric coordinates have the following ranges:

$$\begin{aligned} r &\in [-1, 1] \\ s &\in \left[0, \frac{3}{\sqrt{3}}\right] \\ t &\in \left[0, 2\sqrt{\frac{2}{3}}\right] \end{aligned}$$

Consider the example in Figure 12.

3 Isosurfacing

Isosurfacing presents several interesting problems for higher order elements. For a trilinear interpolant $\Phi_{\text{trilinear}} \in L(X, \mathbb{R})$, it is easy to couch isocontouring as the tessellation of the submanifold of points in X that satisfy, for a given $a \in \mathbb{R}$,

$$\Phi_{\text{linear}}(\boldsymbol{x}) - a = 0$$

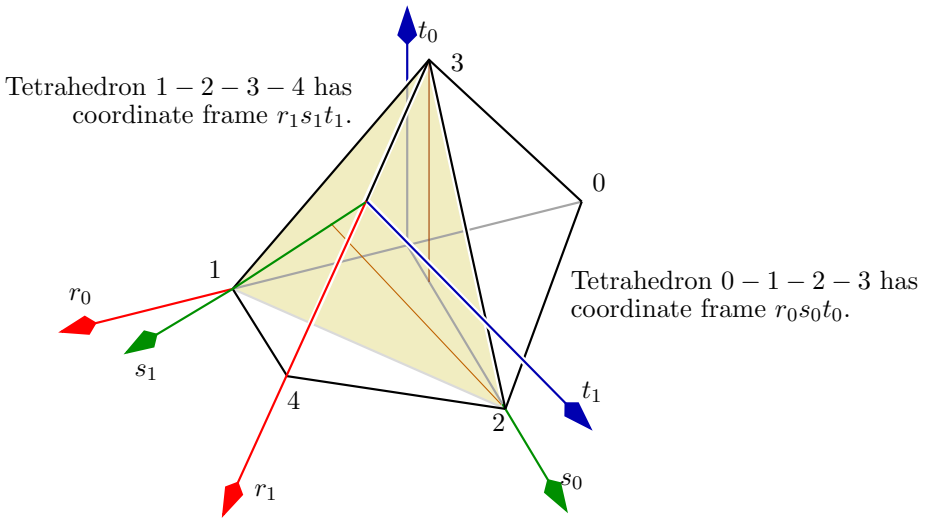


Figure 12: Example of tetrahedra sharing face node coefficients.

Figure 13: A change of basis can be used to find the DOF node permutation for a new cell given an existing cell sharing that node.

but higher order elements must use a parameter space map (shown in Figure 1(b)) which means we must solve, for a $\Phi_{\text{higher order}} \in \mathcal{F}(R, \mathbb{R})$ and a given $a \in \mathbb{R}$,

$$\Phi_{\text{higher order}}(\boldsymbol{r}) - a = 0 \tag{3}$$

and map the resulting locus of points from parameter space to model space using Ξ. This second mapping may require additional tessellation to adequately represent curved isosurfaces within some desired chord or screen-space error.

Additionally, and of much more concern, is the complexity of finding the locus of points that satisfy Equation 3 when Φ is nonlinear. We are basically asking for an approximation of the set

$$S_a = \{\Xi(\boldsymbol{r}) : \Phi(\boldsymbol{r}) - a = 0\}$$

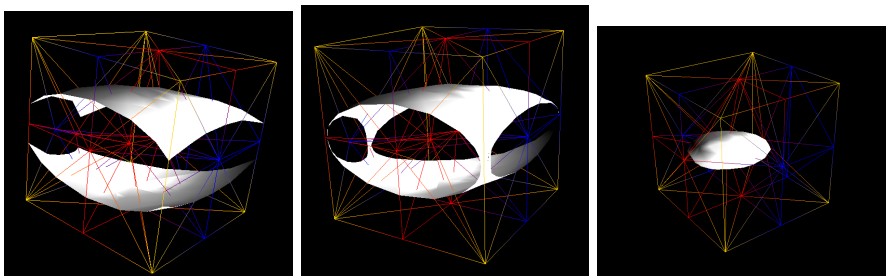
for a given $a \in \mathbb{R}$. We must accommodate the fact that Φ is not bijective and the range of Φ might be a disconnected set.

3.1 Trilinear vs. Higher Order Marching Cubes

The points outlined above mean that a careful examination of the linear isosurfacing algorithm is required. We have 2 options:

1. we may try to adapt the existing algorithm for trilinear interpolants to our higher order elements, or
2. we may branch out and try a different technique.

Very little literature exists in either case. For the first option, Max et al. [2000b,a] have performed some simple schemes for subdividing curved cells into regions that may be ordered for correct volume-rendering. In the second case, Bajaj and Xu [1997], Bajaj [1997] have pointed out a way to produce higher order surface patches that approximate the level set of an algebraic spline. Unfortunately, this requires both constraints on the form of the algebraic spline and a significant amount of processing.



(a) Cell edges should only intersect the isosurface once.

(b) Isosurfaces must intersect cell edges if they intersect a cell face.

(c) Isosurfaces may not be completely contained inside a cell.

Figure 14: Higher order cells break every assumption the linear algorithm makes.

Crossno and Angel [1997] and a variety of others [Stander and Hart, 1997, Desbrun and Gascuel, 1995] have used particle-based methods, but – as Stander and Hart [1997] notes – connecting the points into a surface triangulation is not simple. Thus our work will initially seek to modify the trilinear algorithm so that it may be applied to our higher order elements.

3.2 Problems with the Trilinear Algorithm

The trilinear isosurfacing algorithm considers each hexahedron or tetrahedron individually and assigns each point a color (black or white) based on whether the scalar field at that point is above or below 0, respectively. For a given cell, the topology of the submanifold of $\mathbb{R}^3$ satisfying $\Phi = 0$ is determined solely on the color of the vertices of the cell [3] and, for the hexahedron, the calculation of a quantity that classifies a possible interior saddle point.

In the trilinear case, interior saddle points do not correspond to the "creation" or "destruction" of isosurface components as a is varied; they only change the way the intersections of the isosurface with the cell edges are connected to form the approximation to the isosurface.

The following 3 statements are true for a trilinear interpolant and must be met in order for the isosurfacing algorithm to work.

1. Every edge of every cell in the mesh will intersect the isosurface exactly 0 or 1 times. In other words, no edge will intersect the isosurface more than once.

2. The isosurface may not intersect the face of a cell without intersecting at least 2 edges of the cell.

3. No isolated component of the isosurface may be completely contained in a single cell.

Figure 14 illustrates how none of these criteria are met by higher order interpolants. It depicts how a relatively simple ellipsoid similar to $\Phi(\boldsymbol{r}) = \boldsymbol{r} \cdot \boldsymbol{r}$ over a cube centered at the origin breaks all 3 rules.

3.3 The Modified Isosurfacing Algorithm

3.3.1 Overview

We propose adapting the isosurfacing algorithm by subdividing each higher order cell into regions where the properties of the previous section hold. These properties are not limited to linear functions; rather, they correspond to the presence of critical points. Consider our polynomial Φ, and its critical

[3] This is often implemented as a table lookup.

points $\boldsymbol{r}$ that satisfy $\boldsymbol{\nabla}\Phi(\boldsymbol{r}) = 0$. Where an extremum exists along a line, there are some function values which Φ will take on twice. For an extremum on a surface, some function values will be taken on by a closed curve around the extremum. And for an extremum in a volume, some function values will be taken on by a closed surface around the extremum. The trilinear algorithm assumes that all extrema will be taken on at boundaries, since that is the only place they may occur in a piecewise linear function. Thus our approach is to subdivide each nonlinear finite element wherever internal extrema exist, and then apply the linear algorithm to the resulting regions.

Approaches other than explicitly solving for critical points exist – for instance, subdividing a cell finely enough guarantees the correct topology. But the number of subdivisions required along each coordinate direction is equal to the order of the interpolant along that coordinate axis, which results in an extremely large number of cells and also truncates the range of the scalar field when the subdivision points do not coincide with critical points of the interpolant. Thus we opt to subdivide the cell into regions bounded by critical points. Rather than handle a large number of shapes for the resulting regions, we will handle only simplices, so each finite element must first be divided into a set of simplices even if it contains no interior extrema.

Calculating a critical point of a multivariate polynomial function is a relatively simple task for Newton's method. But calculating *all* the critical points of such a polynomial is not. Several techniques exist for finding critical values of $\boldsymbol{f}(\boldsymbol{r}) = \boldsymbol{\nabla}\Phi(\boldsymbol{r}) = 0$: homotopy, Gauss-Bonnet theory, and resultant techniques.

Resultant techniques [Manocha and Canny, 1993] build a matrix whose eigenvalues are related to the roots of the multivariate system $\boldsymbol{f}(\boldsymbol{x})$. Resultant techniques have only been brought to our attention recently and are part of the future work in this area.

Mann and Rockwood [2002] have applied the Gauss-Bonnet Theorem to the problem; the theorem, which is presented in more detail in §4.2.2, presents a surface integral related to $\boldsymbol{f}(\boldsymbol{r})$ that computes the sum of the indices of all the roots contains inside the surface. Using an octree approach to compute the value on smaller and smaller surfaces, a region containing each root is identified. The problem with this technique is that the index of a root may be positive or negative, and thus the sum of a region containing several roots may be zero.

3.3.2 Polynomial Homotopy Continuation for Critical Points

Homotopy methods [Morgan, 1987, Li et al., 1989, Verschelde et al., 1994] find all the roots of a system of polynomials, $\boldsymbol{f}(\boldsymbol{r}) = 0$ by constructing a new function, $H(\boldsymbol{r}, \lambda) = (1 - \lambda)\boldsymbol{g}(\boldsymbol{r}) + \lambda\boldsymbol{f}(\boldsymbol{r})$. By starting with $\lambda = 0$ and choosing a system $\boldsymbol{g}(\boldsymbol{r})$ whose roots are known we can track how the roots change as $\lambda \to 1$.

The starting polynomial system, $\boldsymbol{g}(\boldsymbol{r})$ must be chosen carefully so that its structure matches the polynomial $\boldsymbol{f}(\boldsymbol{r})$; we want $\boldsymbol{g}$ to contain no more roots than the maximum number $\boldsymbol{f}$ may contain. For dense polynomials, the bound on the number of roots, counting multiplicities and roots at infinity, is given by Bézout's Theorem:

$$D(\boldsymbol{f}) = \prod_{k \in K} deg(f_k)$$

where $deg(f_k)$ is the maximum, over all monomials in f_k, of the total degree of the monomial. For example, $deg(1 + 4r + 3st + 5r^2st) = 4$. For tensor product polynomials, we may end up with a dense polynomial which is likely to have nearly all of possible roots. However, the shape functions constructed by Szabo and Babuska [1991] may be sparse. Sparse systems in general have fewer roots and Bernstein, working with results by Kushnirenko, developed a tighter bound. The same result was also proven separately by Khovanskii and so the bound is generally called the BKK bound. We will not present the theory here as it is beyond the scope of this work. The important result is that the starting polynomial system, $\boldsymbol{g}$, must be chosen appropriately. In either case, starting systems are usually chosen with complex coefficients so that the paths that roots traverse as λ is changed do not intersect each other. Once a starting system is chose, predictor-corrector integrators are usually employed to trace roots.

Several software packages have been written implementing these methods: HOMPACK [Watson et al., 1987], PHCPACK [Verschelde, 1999], HOM4PS [Li et al., 1989], and PSS[Malajovich and

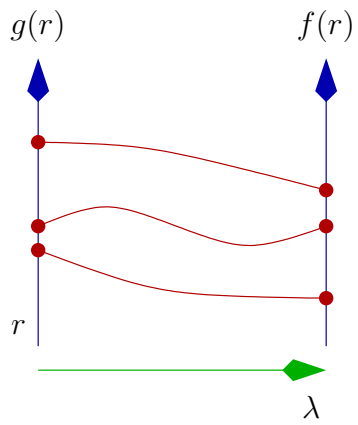


Figure 15: Homotopy techniques track roots as they move from known positions to the solution of a desired system.

Rojas, 2002]. We use PSS, available at http://lyric.labma.ufrj.br/~gregorio/software.html. For univariate polynomials, we use the GNU Scientific Library (GSL).

Computing the critical points of a cell can be compute intensive, especially as the order of the interpolant increases. For instance, an order 6 hexahedral cell with 3 critical points in its parameter space took approximately 5 minutes on a 1 GHz Intel P4 with 128 MB of RAM (no swapping took place). Luckily, this computation can be performed offline and the storage costs are not high compared to the coefficients that specify the interpolant. In fact, the very C++ class that stores the higher order coefficients (named vtkFunctionData) is also used to store critical points, since critical points are also indexed by the submanifold of the cell over which they occur. For instance, for a hexahedral cell, we must find the critical points of $\boldsymbol{f}(\boldsymbol{r})$ over the parameter-space of the entire cell (*i.e.*, $[0,1]^3 \subset \mathbb{R}^3$). Then, we must project $\boldsymbol{f}(\boldsymbol{r})$ to each face of the cell and edge of the cell and solve for critical points on each submanifold. Each of these regions has an associated DOF node and list of critical points. These points are used to further divide a simplicial decomposition of the cell into regions that satisfy the constraints of the linear isosurfacing algorithm.

3.3.3 Streaming Subdivision

Now that we have discussed how the critical points of $\Phi(\boldsymbol{r})$ will be calculated, we may decide how to subdivide the simplices composing the cell. Care must be taken during the subdivision of the initial simplices; new vertices that represent critical points must be connected so that edges between them represent the underlying structure of the interpolant properly. However, when properly computed, the resulting complex is guaranteed to generate output isosurface segments with the proper topology for any $\Phi(\boldsymbol{r})$ with a finite number of isolated critical points, $C_i \in C \in \mathbb{R}^3$. This is because we are in fact creating an abstract simplicial complex whose edges are homeomorphic to lines of steepest ascent/descent in the underlying space of the base complex.

Given an initial tessellation of an element's parametric domain, we apply an adaptive triangulation technique similar to Chung and Field [2000] which was in turn inspired by Velho [1996]. The main difference is that we currently use chord error at the parametric midpoint of each edge rather than the angle between normal vectors at each endpoint. The key design point of our implementation is that there are two tasks performed by an edge-subdivision based tessellation algorithm:

1. making a decision about whether an edge should be subdivided, and
2. applying a template to produce new elements based on which edges of an initial template require subdivision.

We split these two tasks into separate classes so that the same templates for subdivision could be applied to many different subdivision decision algorithms. The algorithms that decide whether edges require subdivision vary depending on

1. the interpolation algorithm used for the nonlinear function,
2. the criteria used in the decision (geometric distance, scalar field nonlinearity),
3. the purpose of the overall task requiring a tessellation.

Item 3 requires some further explanation; if the tessellation is being produced simply for display purposes, then a view-dependent subdivision may be performed. This greatly reduces the amount of work required, since no function evaluation need take place if both edge endpoints are safely outside the viewing frustum[4]. On the other hand, if the tessellation is produced as input to some further post-processing step, then no regions may be excluded from the calculation.

The templates that produce new simplices given some starting simplex, σ, and a set of edges of σ requiring subdivisions deserve some discussion. The problem of triangulating the new set of points (*i.e.*, the original vertices of σ and the mid-edge nodes being introduced by the subdivision) is not unique – there may be 0, 1, or many possible triangulations of the point set. With a streaming algorithm, we must guarantee that each simplex may be processed without any information on its neighbors. Since we want to maintain a compatible tessellation, this means that any simplices that are shared as a boundary between two higher-dimensional simplices must be tessellated identically when all the higher-dimensional simplices are divided, even where there are several distinct tessellations. For example, any triangle, τ, shared by two tetrahedra, σ_0 and σ_1, must be tessellated the same way when σ_0 and σ_1 are subdivided.

Ruprecht and Müller [1998] developed a method for deciding on a subdivision template so that adjacent simplices produced compatibly-tessellated boundaries. Their technique chooses triangulations that have the best aspect ratios. Unfortunately, they offer no solution when several tessellations exist and have identical aspect ratios. The next two sections review and extend Ruprecht and Müller [1998] to handle even these cases.

3.3.4 Unambiguous Cases

We use the same nomenclature as Ruprecht and Müller [1998], so this section is just a brief review of their results. When a tetrahedron, σ, is to be subdivided, we are given a list of edges of σ that will be divided. First, the vertices of σ are permuted into σ', a positive arrangement of σ that matches one of 12 cases (Ruprecht and Müller [1998] present 11 cases but we divide their case 3c into 3c and 3d so that σ' will always be a *positive* arrangement of σ). Cases are called out with

- the number of edges of a tetrahedron, σ, that should be subdivided, and
- a letter representing a unique configuration of those edges relative to each other.

Then, a collection of points, P, is created that includes σ' and the midpoints of edges in σ' that must be subdivided. This set of points, P, must be tessellated in a consistent manner so that simplices adjacent to σ will be compatible at the boundary they share with σ. Let's say we can produce such a tessellation. Call it σ''. For each of the 12 cases, there will be edges in σ'' that are constrained to be present and possibly some edges of σ'' that are not constrained. Ruprecht and Müller [1998] use geometry – the length of the edges of σ' – to decide how to connect points in P to form σ''. They choose edges for σ'' that produce tetrahedra with the best possible aspect ratio given P. Each case with unconstrained edges in σ'' will have several variants based on which edges of σ' are longer than the others (relative to which edges must be subdivided). Figure 16 shows all 6 variants for Case 3a.

Unfortunately, when edges of σ' are of equal length, their criterion gives no answer for how σ'' should be obtained. This leaves several possibilities for σ'' and it is ambiguous which one we should use so that σ'' remains compatible with its neighbors. We present a technique for these ambiguities in the next section.

[4] We assume that some safety margin is included so that edges which curve into the frustum are not excluded.

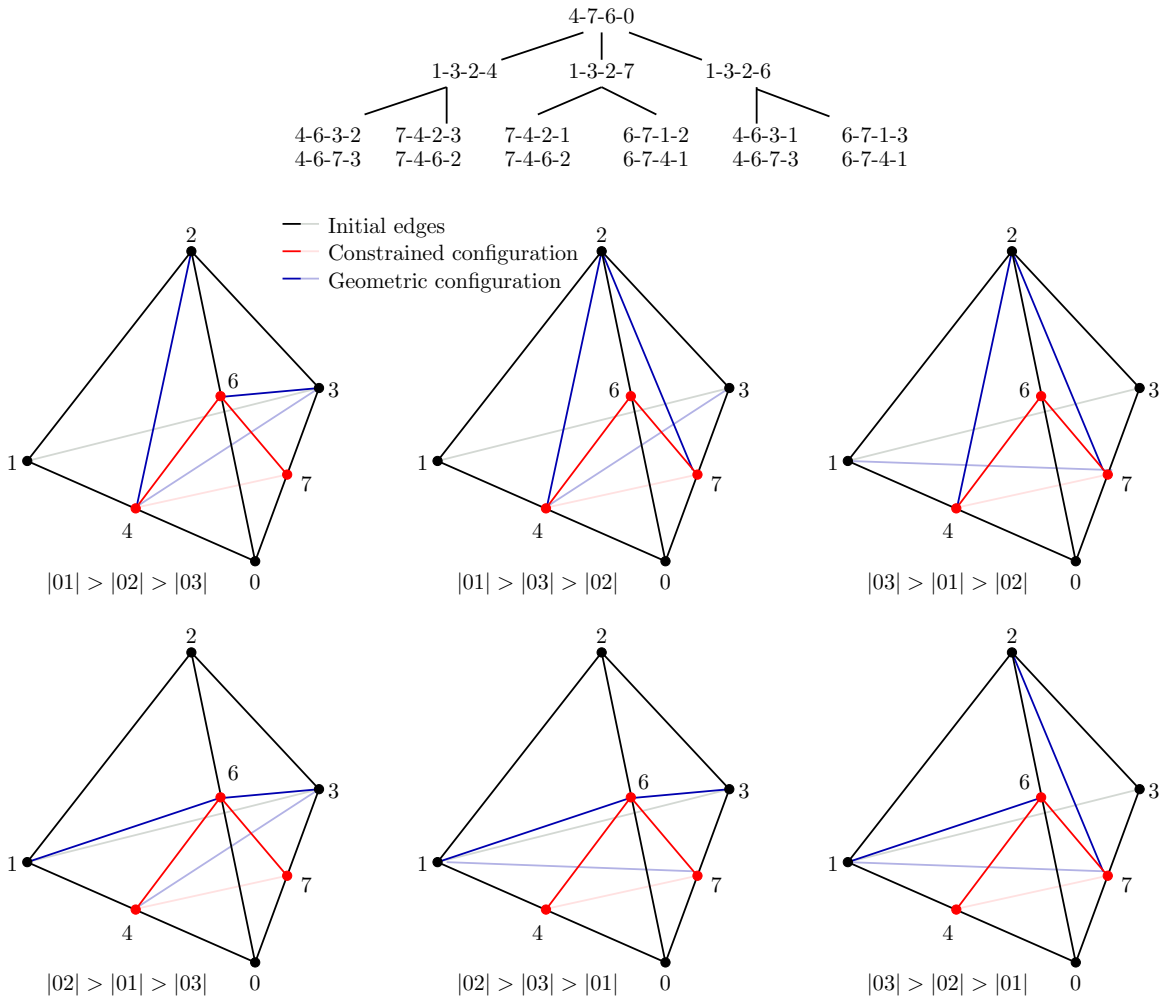
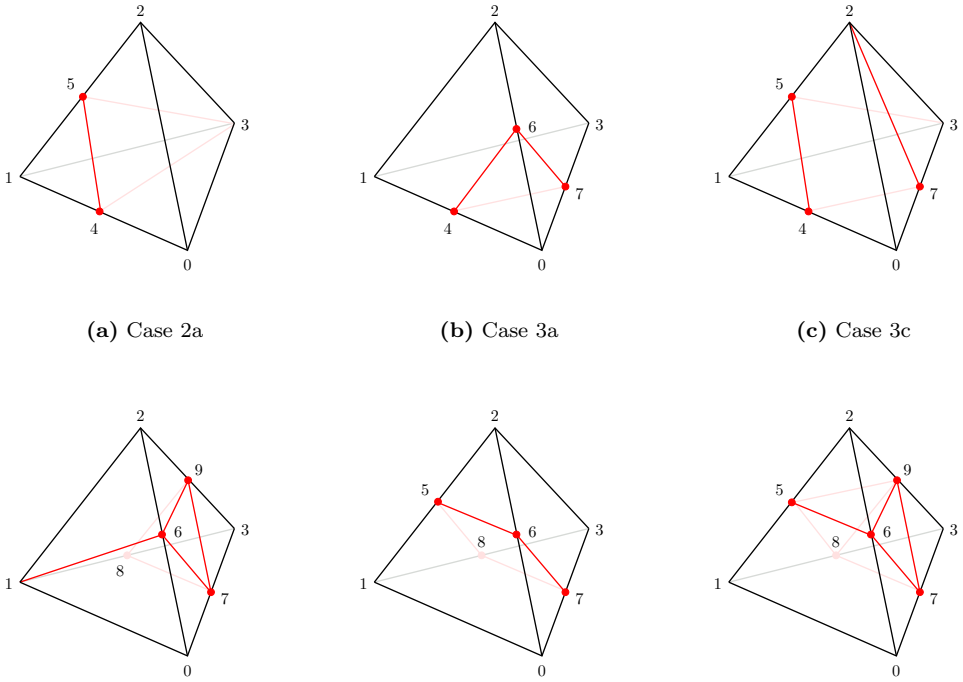


Figure 16: Subdivision of unambiguous case 3a.



(a) Case 2a **(b)** Case 3a **(c)** Case 3c

(d) Case 4a **(e)** Case 4b **(f)** Case 5

Figure 17: Potentially ambiguous configurations.

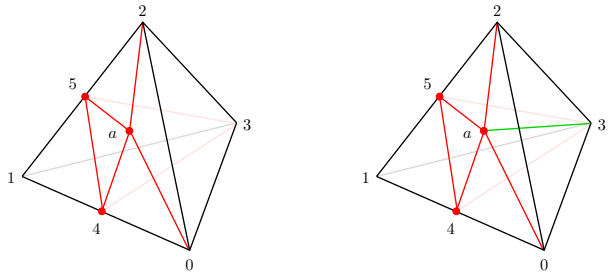
3.3.5 Ambiguous Cases

Ambiguous cases occur when a face can be split in two different ways, *i.e.*, whenever at least one face has exactly two edges of equal length that must be split. A simple enumeration shows that all such situations can be summarized by the means of reference configurations 2a, 3a, 3c, 4a, 4b or 5 (see Figure 17). In these cases, ambiguities are as follows:

Case 2a In this ambiguous case, when $|01| = |12|$ (see Figure 17(a)), the ambiguous face requires a point insertion to remove the ambiguity. By inserting a new vertex, we produce a subdivision of the triangle that is symmetric. (*cf.* Figure 18(a)).

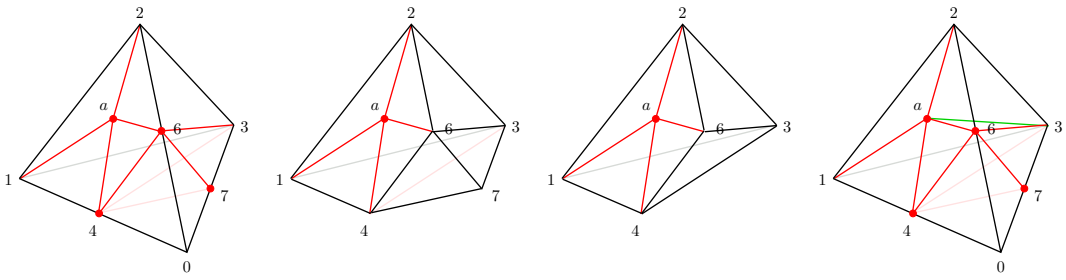
Case 3a Edges to be split are 01, 02 and 03, and therefore ambiguities arise if and only if at least two or three of them have the same lengths (see Figure 17(b)). If exactly two of these edges have same length, then the third one might be either shorter or longer. Hence, $1 + \binom{2}{3} \times 2 = 7$ ambiguous configurations may arise. But, up to a vertex permutation, those can be summarized with only three cases: $|01| = |12| > |03|$ (case 3cα, see Figure 19(a)), $|01| = |12| < |03|$ (case 3cβ, see Figure 20(a)), and $|01| = |12| = |03|$ (case 3cγ, see Figure 21(a)).

Case 3c Edges to be split are 01, 12 and 03, whence only faces 021 and 013 can have equivocal decompositions. One possibility is that exactly one face decomposition is ambiguous, which means that either $|01| = |12|$ and $|12| \neq |03|$, or $|12| = |03|$ and $|01| \neq |12|$. Depending on whether $\neq$ is indeed $<$ or $>$, these $2 \times 2 = 4$ configurations are represented, up to vertex permutation, by case 3cα ($|01| = |12| > |03|$, see Figure 22(a)) and 3cβ ($|01| = |12| < |03|$, see Figure 23(a)). The other possibility is that both face decompositions are ambiguous, *i.e.* $|01| = |12| = |03|$ (case 3cγ, see Figure 24(a)).



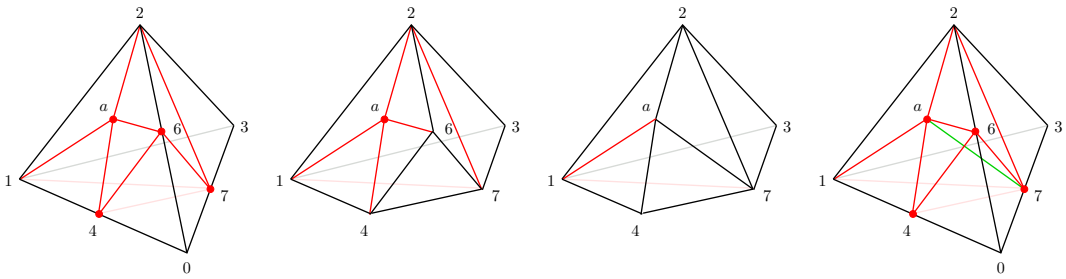
(a) **(b)**

Figure 18: Subdivision of ambiguous case 2a ($|01| = |12|$): (a) constrained configuration, and (b) complete subdivision.



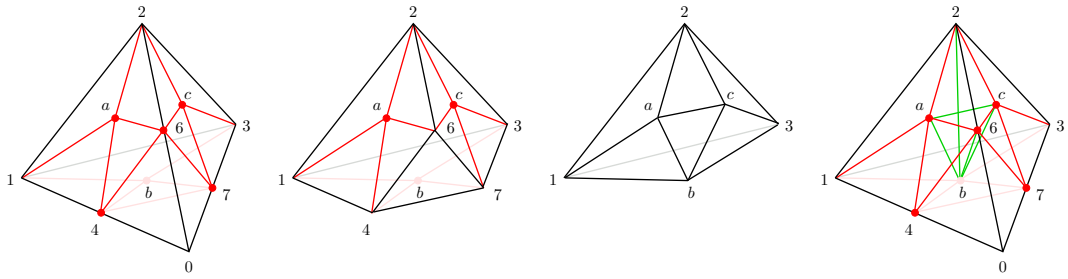
(a) **(b)** **(c)** **(d)**

Figure 19: Subdivision of ambiguous case 3aα ($|01| = |02| > |03|$): (a) constrained configuration, (b) after removal of 0467, (c) after removal of 4367, and (d) complete subdivision.



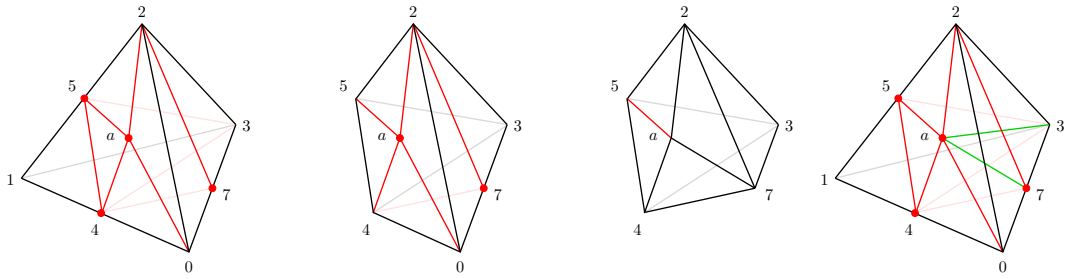
(a) **(b)** **(c)** **(d)**

Figure 20: Subdivision of ambiguous case 3aβ ($|01| = |02| < |03|$): (a) constrained configuration, (b) after removal of 0467, (c) after removal of $a267$, and (d) complete subdivision.



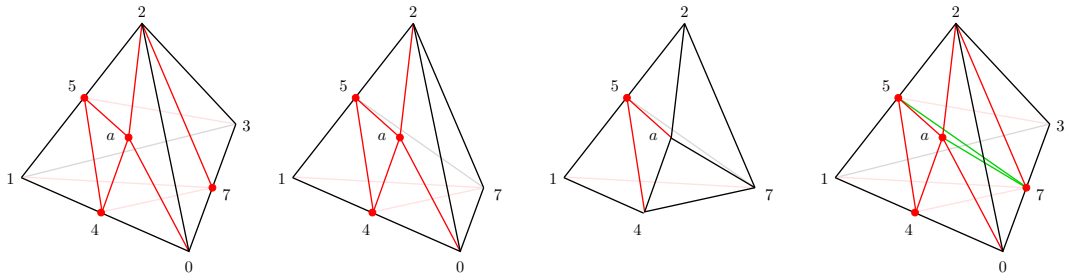
(a) **(b)** **(c)** **(d)**

Figure 21: Subdivision of ambiguous case 3aγ ($|01| = |02| = |03|$): (a) constrained configuration, (b) after removal of 0467, (c) after removal of 62c74a (with tetrahedra 26ac, and using edge b6, b6a4, b6ca, b67c and b647), and (d) complete subdivision.



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

Figure 22: Subdivision of ambiguous case 3cα ($|01| = |12| > |03|$): (a) constrained configuration, (b) after removal of 4153, (c) after removal of a047 and a207, and (d) complete subdivision.



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

Figure 23: Subdivision of ambiguous case 3cβ ($|01| = |12| < |03|$): (a) constrained configuration, (b) after removal of 7153 and 7523, (c) after removal of a047 and a207, and (d) complete subdivision.

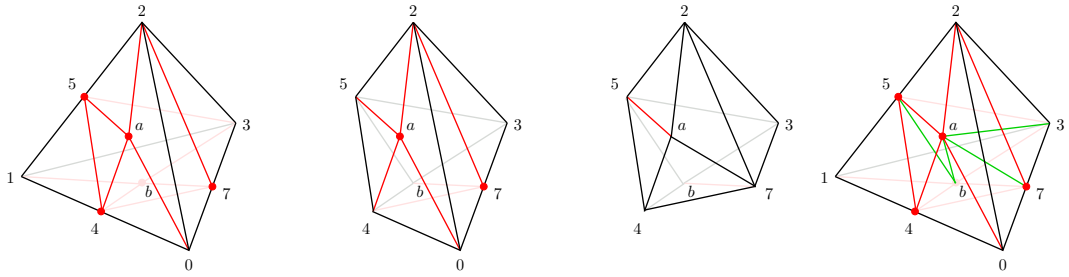


Figure 24: Subdivision of ambiguous case 3cγ ($|01| = |12| = |03|$): (a) constrained configuration, (b) after removal of 415b and b153, (c) after removal of a047 and a207, and (d) complete subdivision.

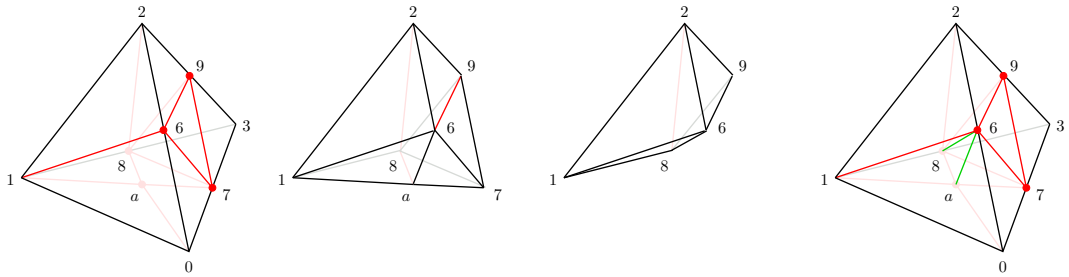


Figure 25: Subdivision of ambiguous case 4aα ($|03| = |13| > |23|$): (a) constrained configuration, (b) after removal of 7893, 670a and 601a, (c) after removal of 6978, 67a8 and 6a18, and (d) complete subdivision.

Case 4a Three edge midpoints belong to the same face, here represented by 032, that is therefore necessarily unambiguous. The remaining split edge, 13 is shared by faces 013 and 123; as each of these faces also has another edge midpoint, so both may be ambiguous. The remaining face has a unique edge midpoint and is thus unambiguous (*cf.* Figure 17(d)). What matters here are thus the lengths of the non-split edges compared to 13, since they decide the fate of the potentially ambiguous faces: one may have a single ambiguous face, which means that 13 is equal to only one of the non-split edges and either shorter or longer than the remaining one; or, one may have two ambiguous faces. In other words, there are $2 \times 2 + 1 = 5$ ambiguous configurations, that can be represented, up to a vertex permutation, by either 4aα ($|03| = |13| > |23|$, see Figure 25(a)), 4aβ ($|03| = |13| < |23|$, see Figure 26(a)), or 4aγ ($|03| = |13| = |23|$, see Figure 27(a)).

Case 4b This case is, by far, the most complex one: four edge midpoints distributed such that every face contains exactly two of them (*cf.* Figure 17(e)). Indeed, the distribution of midpoints along a "diametral" path around the tetrahedron allows each face to be ambiguous, and most of such ambiguities may be further refined in subcases, depending on the configuration of non-ambiguous faces. Again, grouping of topologically equivalent cases may be done, depending on the number, and respective positions, of ambiguous faces. First, it is interesting to notice that the fate of the faces is solely determined by $|02|$, $|12|$, $|03|$ and $|13|$, the two other edges playing no role here. It is also straightforward to see that one cannot have exactly three ambiguous faces, since having three ambiguous faces means that three distinct pairs of the split edges have the same

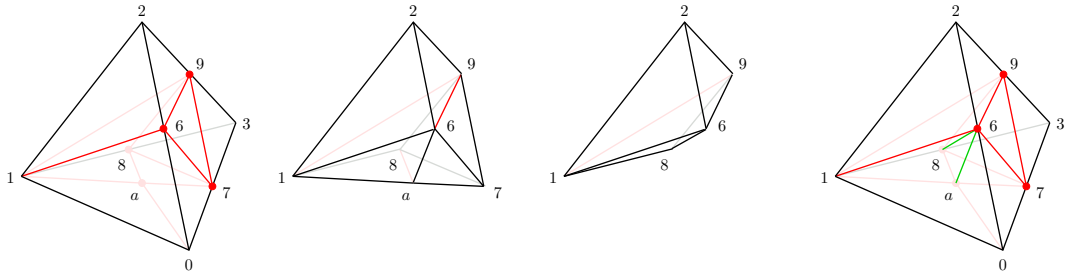


Figure 26: Subdivision of ambiguous case 4aβ ($|03| = |13| < |23|$): (a) constrained configuration, (b) after removal of 7893, 670a and 601a, (c) after removal of 6978, 67a8 and 6a18, and (d) complete subdivision.

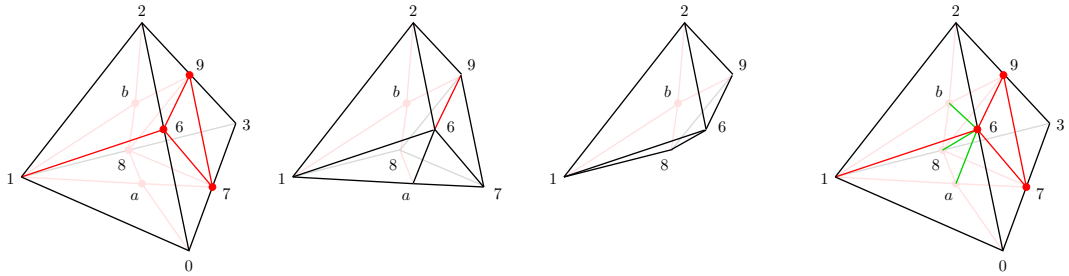
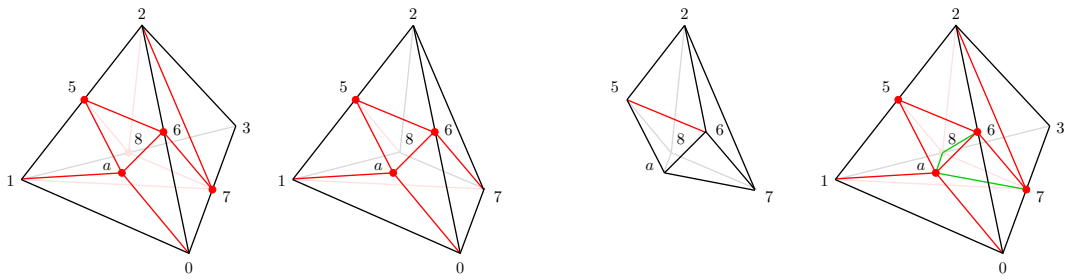
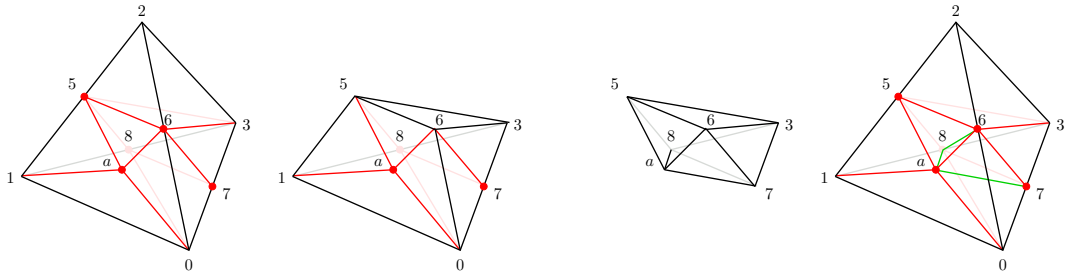


Figure 27: Subdivision of ambiguous case 4aγ ($|03| = |13| = |23|$): (a) constrained configuration, (b) after removal of 7893, 670a and 601a, (c) after removal of 6978, 67a8 and 6a18, and (d) complete subdivision.



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

Figure 28: Subdivision of ambiguous case 4bα ($|02| = |12| < |13| < |03|$): (a) constrained configuration, (b) after removal of 7823, (c) after removal of $a607$, $a158$, $a017$ and $a718$, and (d) complete subdivision using edge 68.

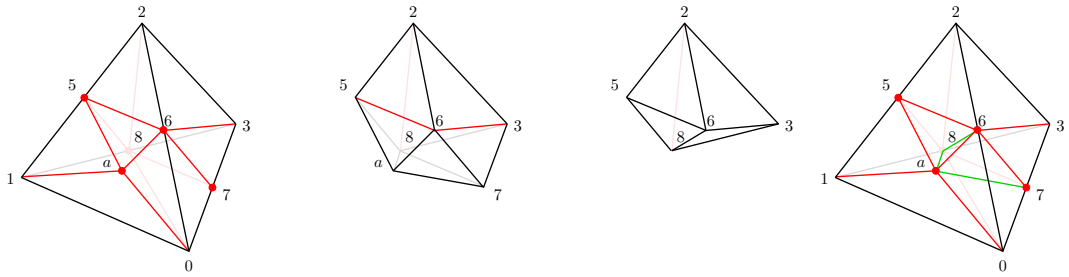


(a) **(b)** **(c)** **(d)**

Figure 29: Subdivision of ambiguous case 4bβ ($|02| = |12| > |13| > |03|$): (a) constrained configuration, (b) after removal of 6523, (c) after removal of $a607$, $a158$, $a018$ and $a708$, and (d) complete subdivision.

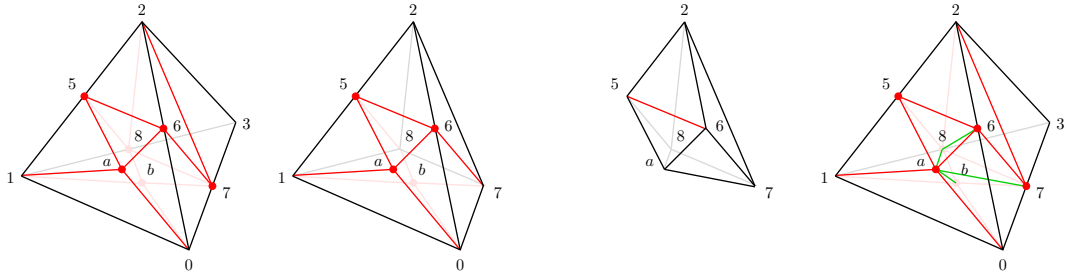
length, and thus by transitivity that all of these edges have the same length; in other words, four faces are ambiguous. Face 021 will be used as the ambiguous face to represent the class of all configurations with a unique ambiguity; in this case, $|02| = |12|$ and several possibilities are left to the other faces to be split, 032, 013 and 123 $|13|$, depending on the values of $|03|$ and $|13|$ relative to $|02| = |12|$. Each of these subcases can be deduced, up to vertex permutation, by either 4bα ($|02| = |12| < |13| < |03|$, see Figure 28(a)), or 4bβ ($|02| = |12| > |13| > |03|$, see Figure 29(a)), or 4bγ ($|03| < |02| = |12| < |13|$, see Figure 30(a)). If exactly two faces are ambiguous, then they either be opposed or adjacent. The former case has a unique topological representation, 4bδ ($|02| = |12| < |03| = |13|$, see Figure 31(a)), and all actual configurations can be deduced thereof, thanks to a vertex permutation. The latter case has a remaining degree of freedom, in the sense than three of the split edges have the same length, and the fourth one can be either shorter or longer; therefore, two configurations will represent all possible subcases: 4bϵ ($|02| = |12| = |03| < |13|$, see Figure 32(a)), and 4bζ ($|02| = |12| = |03| > |13|$, see Figure 33(a)). Finally, the unique case with four ambiguous faces is called 4bη ($|02| = |12| = |03| = |13|$, see Figure 33(a)).

Case 5 All edges but 01 must be split, and thus only faces 021 and 013 can lead to ambiguities, since each other face is univocally decomposed in 9 subfaces (see Figure 17(f)). Therefore, only 3 ambiguous configurations are possible, depending on whether one (two cases, represented up to a face permutation by configuration 5α: $|02| = |12|$ and $|03| > |13|$, see Figure 35(a)) or two (one



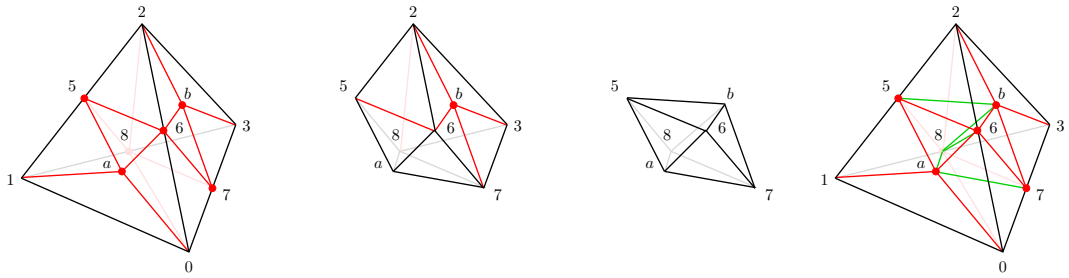
(a) **(b)** **(c)** **(d)**

Figure 30: Subdivision of ambiguous case 4bγ ($|03| < |02| = |12| < |13|$): (a) constrained configuration, (b) after removal of $a607$, $a158$, $a018$ and $a708$, (c) after removal of $67a8$, $6a58$ and 6378, and (d) complete subdivision.



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

Figure 31: Subdivision of ambiguous case 4bδ ($|02| = |12| < |03| = |13|$): (a) constrained configuration, (b) after removal of 7823, (c) after removal of $a607$, $a158$, $a01b$, $ab18$, $a0b7$ and $a7b8$, and (d) complete subdivision using edge 68.



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

Figure 32: Subdivision of ambiguous case 4bϵ ($|02| = |12| = |03| < |13|$): (a) constrained configuration, (b) after removal of $a607$, $a158$, $a018$ and $a708$, (c) after removal of $b625$, $b378$, $b238$ and $b285$, and (d) complete subdivision using edge 68.

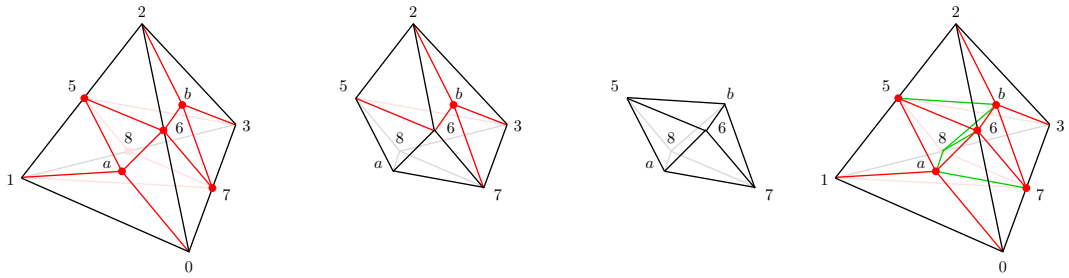


Figure 33: Subdivision of ambiguous case 4bζ ($|02| = |12| = |03| > |13|$): (a) constrained configuration, (b) after removal of $a607$, $a158$, $a018$ and $a708$, (c) after removal of $b625$, $b378$, $b235$ and $b385$, and (d) complete subdivision using edge 68.

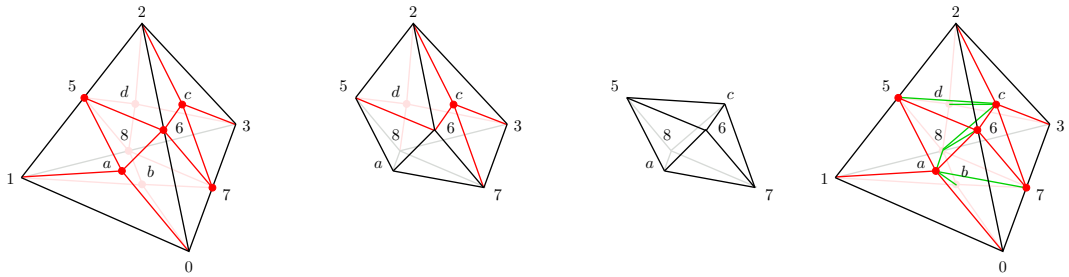


Figure 34: Subdivision of ambiguous case 4bη ($|02| = |12| = |03| > |13|$): (a) constrained configuration, (b) after removal of $a607$, $a158$, $a01b$, $ab18$, $a0b7$ and $a7b8$, (c) after removal of $c625$, $c378$, $c23d$, $c2d5$, $cd38$ and $c5d8$, and (d) complete subdivision using edge 68.

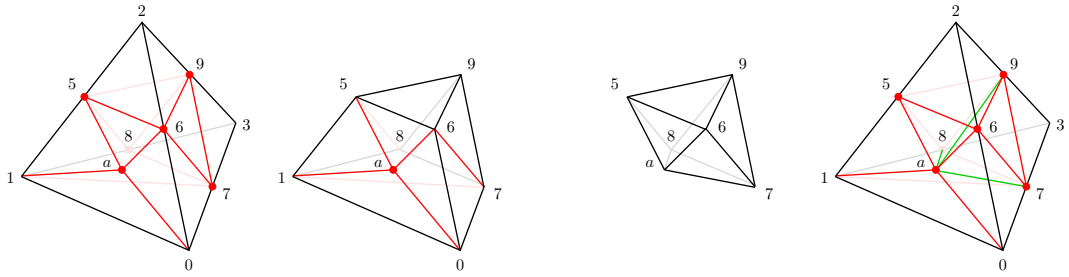


Figure 35: Subdivision of ambiguous case 5α ($|02| = |12|, |03| > |13|$): (a) constrained configuration, (b) after removal of 6529 and 7893, (c) after removal of $a607$, $a158$, $a017$ and $a718$, and (d) complete subdivision using edge $a9$.

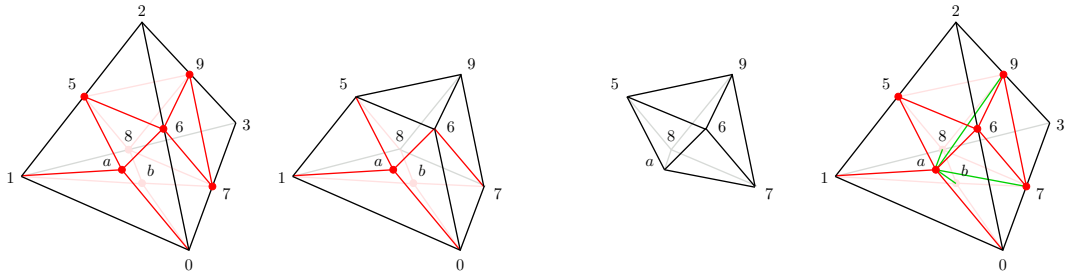


Figure 36: Subdivision of ambiguous case 5β ($|02| = |12|, |03| = |13|$): (a) constrained configuration, (b) after removal of 6529 and 7893, (c) after removal of $a607$, $a158$, $a01b$, $ab18$, $a0b7$ and $a7b8$, and (d) complete subdivision using edge $a9$.

configuration, 5β: $|02| = |12|$ and $|03| = |13|$, see Figure 36(a)) faces are ambiguous.

The non-ambiguous tetrahedral refinements of ambiguous cases, detailed in Table 3.3.5 and illustrated by Figures 18(b), 19(d), 20(d), 21(d), 22(d), 23(d), 24(d), 25(d), 26(d), 27(d), 28(d), 29(d), 30(d), 31(d), 32(d), 33(d), 34(d), 35(d), 36(d) are not unique in general. Those we propose seem to be the more natural ones, seemingly displaying decent elements qualities (at least when the initial tetrahedron so does).

4 Future Work

4.1 Correct Triangulation

As mentioned in §3.3.3 above, the initial triangulation of the critical points and cell corner nodes may not yield proper results. Consider the 2D example in Figure 37.

Case	Ambiguity	Tetrahedra
2a	$\|01\| = \|12\|$	$04a3$ $0a23$ 4153 $45a3$ $a523$
3aα	$\|01\| = \|02\| > \|03\|$	0467 4367 $a123$ $a263$ $a643$ $a413$
3aβ	$\|01\| = \|02\| < \|03\|$	0467 1327 $a127$ $a267$ $a647$ $a417$
3aγ	$\|01\| = \|02\| = \|03\|$	0467 $26ac$ $37cb$ $41ab$ $b6a4$ $b6ca$ $b67c$ $b647$ $2abc$ $1ab2$ $2b3c$ $321b$
3cα	$\|01\| = \|12\| > \|03\|$	4153 $a047$ $a207$ $a743$ $a273$ $a523$ $a453$
3cβ	$\|01\| = \|12\| < \|03\|$	7153 7523 $a047$ $a207$ $a527$ $a457$ 1547
3cγ	$\|01\| = \|12\| = \|03\|$	$415b$ $b153$ $a047$ $a207$ $a523$ $a273$ $a74b$ $a7b3$ $a45b$ $ab43$
4aα	$\|03\| = \|13\| > \|23\|$	7893 $670a$ $601a$ 6978 $67a8$ $6a18$ 1268 2689
4aβ	$\|03\| = \|13\| < \|23\|$	7893 $670a$ $601a$ 6978 $67a8$ $6a18$ 1269 1689
4aγ	$\|03\| = \|13\| = \|23\|$	7893 $670a$ $601a$ 6978 $67a8$ $6a18$ $612b$ $629b$ $698b$ $681b$
4bα	$\|02\| = \|12\| < \|13\| < \|03\|$	7823 $a607$ $a158$ $a017$ $a718$ $67a8$ $6a58$ 6278 6528
4bβ	$\|02\| = \|12\| > \|13\| > \|03\|$	6523 $a607$ $a158$ $a018$ $a708$ $67a8$ $6a58$ 6378 6538
4bγ	$\|03\| < \|02\| = \|12\| < \|13\|$	6238 $a607$ $a158$ $a018$ $a708$ $67a8$ $6a58$ 6378 6528
4bδ	$\|02\| = \|12\| < \|03\| = \|13\|$	7823 $a607$ $a158$ $a01b$ $ab18$ $a0b7$ $a7b8$ $67a8$ $6a58$ 6278 6528
4bϵ	$\|02\| = \|12\| = \|03\| < \|13\|$	$a607$ $a158$ $a018$ $a708$ $b625$ $b378$ $b238$ $b285$ $67a8$ $6a58$ $6b78$ $65b8$
4bζ	$\|02\| = \|12\| = \|03\| > \|13\|$	$a607$ $a158$ $a018$ $a708$ $b625$ $b378$ $b235$ $b385$ $67a8$ $6a58$ $6b78$ $65b8$
4bη	$\|02\| = \|12\| = \|03\| = \|13\|$	$a607$ $a158$ $a01b$ $ab18$ $a0b7$ $a7b8$ $c625$ $c378$ $c23d$ $c2d5$ $cd38$ $c5d8$ $67a8$ $6a58$ $65c8$ $6c78$
5α	$\|02\| = \|12\|, \|03\| > \|13\|$	6529 7893 $a607$ $a158$ $a017$ $a718$ $a859$ $a679$
5β	$\|02\| = \|12\|, \|03\| = \|13\|$	6529 7893 $a607$ $a158$ $a01b$ $ab18$ $a0b7$ $a7b8$ $a859$ $a679$

Table 1: Subdivisions of ambiguous cases

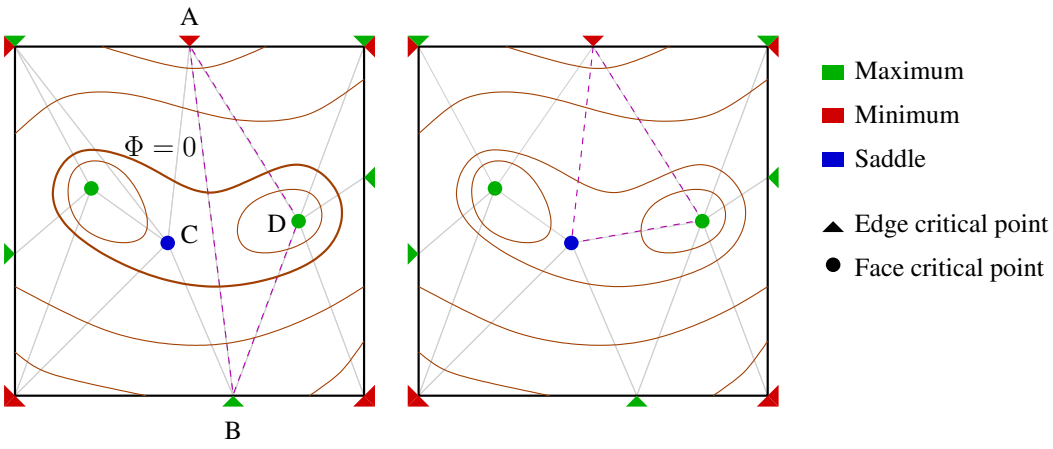


Figure 37: A random tessellation (as on the left) of a cell's critical and bounding points may yield simplices that fail the marching tetrahedra/triangles requirements: edge A-B intersects the isocontour $\Phi = 0$ twice. Edge-facet flips may be required to produce a correct tessellation, as shown to the right.

4.2 Future Work: Non-Isolated Critical Points

4.2.1 Motivation

Another aspect of our future work will be dealing with non-isolated critical points. Such cases occur in even relatively simple scalar fields, as illustrated by the following example:

EXAMPLE 4.1. Let $g : \mathbb{R}^2 \to \mathbb{R}$ be defined by $g(x, y) = x^2y^2$, a graphical representation of which is provided by Figure 38, left. Its partial derivatives, given by

$$\left(\forall (x, y) \in \mathbb{R}^2\right) \quad \frac{\partial g}{\partial x}(x, y) = 2xy^2 \quad \frac{\partial g}{\partial y}(x, y) = 2x^2y$$

are continuous and vanish when either $x = 0$ or $y = 0$; in other words, all points along the two coordinate axes, and only those points, are critical points of g. In particular, none of these critical points are isolated, as shown in Figure 39, left. In this case, all of them obviously correspond to an absolute (but not strict) *minimum* of g.

Now, considering $h : \mathbb{R}^2 \to \mathbb{R}$, defined by $h(x, y) = x^3y^3$ (see Figure 38, right), one then has

$$\left(\forall (x, y) \in \mathbb{R}^2\right) \quad \frac{\partial h}{\partial x}(x, y) = 3x^2y^3 \quad \frac{\partial h}{\partial y}(x, y) = 3x^3y^2$$

and, therefore, g and h have the same critical points, illustrated in Figure 39, right. However, in this case, none of them are *minima* – which means that an isosurface passing through them will not change in topology. So, unlike the first case, we do not need to refine our triangulation of the domain to get a good approximation to the isosurface.

The 3D scalar fields $(x, y, z) \mapsto x^2y^2z^2$ and $(x, y, z) \mapsto x^3y^3z^3$ clearly lead to a similar situation, with non-isolated critical points along the three coordinate axes. Two conclusions can be drawn from these simple examples: first, non-isolated critical points arise even from very simple 2D and 3D scalar fields; second, there are many different ways that a field may be degenerate and a more thorough investigation is required prior to being able to decide their effect on our algorithms.

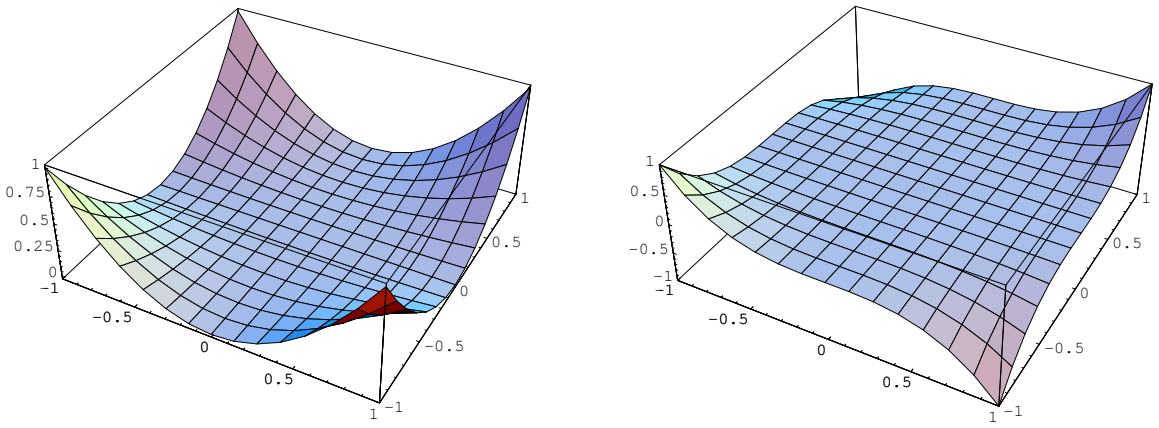


Figure 38: *Graphs of* $g : (x, y) \mapsto x^2y^2$ *(left) and* $h : (x, y) \mapsto x^3y^3$ *(right) over* $[-1, 1]^2$.

4.2.2 Gauss-Bonnet Approach

Even though our scalar field is piecewise polynomial, its degree is too high for the explicit determination of its critical points. This leaves us with numerical approximation. We have already proposed a solution (*cf.* §3.3.2) that detects isolated critical points, but which fails for non-isolated critical points, as exemplified in (*cf.* §3.3.2). In fact, this problem is very difficult, and lacks theoretical results from which algorithmic methods could be derived. To our knowledge, there is very little

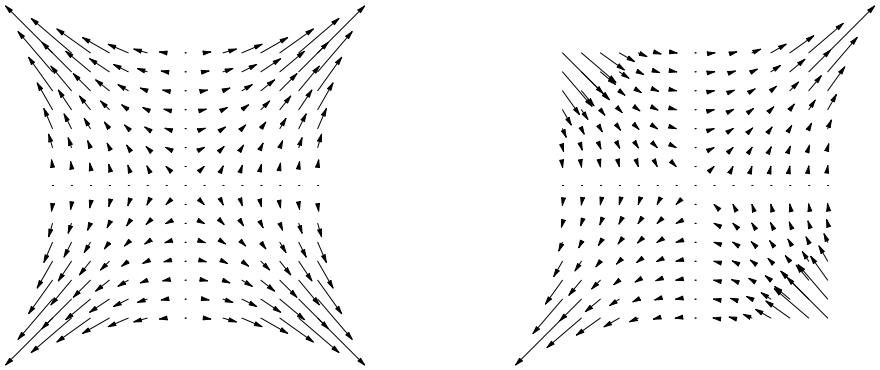


Figure 39: *Non-isolated critical points: vector fields* $\boldsymbol{\nabla} f$ *(left) and* $\boldsymbol{\nabla} g$ *(right) over* $[-1,1]^2$.

literature on the subject, but one *a priori* interesting approach is suggested in Mann and Rockwood [2002] for the detection of non-isolated critical points. Although, in this article the authors emphasize the framework of General Algebra (*i.e.*, a particular realization of a Clifford Algebra), because it provides an elegant way of dealing with geometric operations, the core of the method lies in the local form of the Gauss-Bonnet Theorem, presented here without the geometric algebra framework. We firstly recall a few useful definitions (S^d denotes the unit sphere in $\mathbb{R}^{d+1}$):

Proposition and Definition 4.2. Let X and Y each be a compact, connected and oriented manifold with the dimension d, and f be a $\mathcal{C}^\infty$ mapping from X to Y. There exists a unique integer, called the *degree of* f and denoted $\deg(f)$, such that, for all regular $y \in Y$,

$$\deg(f) = \sum_{x \in f^{-1}(y)} \operatorname{sgn}(J_x(f)),$$

where $J_x(f)$ denotes the jacobian of f in x.

Example 4.3. A few examples might make this definition clearer:

1. The degree of the identity function over X is 1, since each point of X has one and only one inverse image, itself, and the tangent map in each such point preserves orientation.

2. If a function is not surjective then, since some points of Y have no inverse image, the degree is necessarily 0.

Proposition and Definition 4.4. Let ξ be a $\mathcal{C}^\infty$ vector field over an open set $U \subset \mathbb{R}^d$, and $m \in U$ be an isolated zero of ξ. The *index of* ξ in m, denoted $\operatorname{ind}_m \xi$, is the degree of

$$\begin{array}{rccc} \gamma: & S^d & \longrightarrow & S^d \\ & \omega & \longmapsto & \dfrac{\xi(m+\varepsilon\omega)}{\|\xi(m+\varepsilon\omega)\|}, \end{array}$$

and is identical for all $\varepsilon > 0$ such that $\xi(\omega) \neq 0$ for all $\omega \in B(m,\varepsilon) - \{m\}$.

Example 4.5. 1. Consider the only zero of the identity function over $\mathbb{R}^d$, $\mathrm{Id}_{\mathbb{R}^d}$, $m = 0$ (see figure 40 for $d = 2$). Then, γ maps S^{d-1} onto itself as follows:

$$(\forall \omega \in S^{d-1}) \quad \gamma(\omega) = \frac{\omega}{\|\omega\|} = \omega$$

and, therefore, γ is the identity function over S^{d-1}. Hence, $\deg(\gamma) = \operatorname{ind}_0 \mathrm{Id}_{\mathbb{R}^d} = 1$.

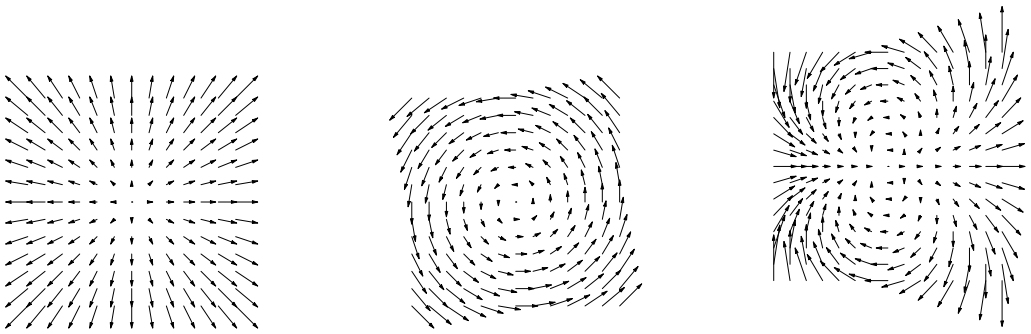


Figure 40: *Vector fields* $\mathrm{Id}_{\mathbb{R}^2}$ *(left),* $\xi_1 : (x, y) \mapsto (-y, x)$ *(center) and* $\xi_2 : (x, y) \mapsto (x^2 - y^2, 2xy)$ *(right) over* $\mathbb{R}^2$.

2. Let $\xi_1(x, y) = (-y, x)$ and $\xi_2(x, y) = (x^2 - y^2, 2xy)$ for all $(x, y) \in \mathbb{R}^2$ (see figure 40). Then, $(0, 0)$ is the only zero of both vector fields, and is therefore isolated. In order to determine the indices of ξ_1 and ξ_2 in $(0, 0)$, we have to examine the following functions:

$$\begin{array}{rccc} \gamma_1 : & S^1 & \longrightarrow & S^1 \\ & \omega & \longmapsto & \dfrac{\xi_1(\omega)}{\|\xi_1(\omega)\|} \end{array} \qquad \begin{array}{rccc} \gamma_2 : & S^1 & \longrightarrow & S^1 \\ & \omega & \longmapsto & \dfrac{\xi_2(\omega)}{\|\xi_2(\omega)\|}. \end{array}$$

   Now, if (x, y) belongs to S^1, so do both $\xi_1(x, y)$ and $\xi_2(x, y)$ (easy to check, by switching to polar coordinates). Hence, γ_1 and γ_2 are the respective restrictions of ξ_1 and ξ_2 to S^1. Therefore, γ_1 is a diffeomorphism with Jacobian equal to 1 in each point, thus $\deg(\gamma_1) = \mathrm{ind}_0\, \xi_1 = 1$. Regarding γ_2, the case is slightly more complicated since this function is not bijective, which can be easily seen by switching to polar coordinates:

$$(\forall \theta \in [0, 2\pi]) \quad \gamma_2(\cos\theta, \sin\theta) = (\cos 2\theta, \sin 2\theta)$$

   and therefore, conversely, each $w \in S^1$ has exactly two inverse images by γ_2. Finally, since the jacobian in each point is positive, $\deg(\gamma_2) = \mathrm{ind}_0\, \xi_2 = 2$.

Theorem 4.6 (Poincaré-Hopf Index Theorem). *Let X be an boundaryless orientable $\mathcal{C}^\infty$ manifold, and ξ a $\mathcal{C}^\infty$ vector field on X. If the zeros of ξ are isolated, then*

$$\chi(X) = \sum_{\{x \in X : \xi(x) = 0\}} \mathrm{ind}_x\, \xi = \mathrm{ind}_X\, \xi,$$

Where $\chi(X)$ is the Euler characteristic of X.

Proof. See, *e.g.*, Spivak [1999] □

Remark 4.7. Theorem 4.6 can be extended to the case of orientable $\mathcal{C}^\infty$ manifolds with boundaries, if the vector field is outward-pointing on the boundary. However, and to our great prejudice, Poincaré-Hopf Index Theorem cannot be generalized to the case of non-isolated critical points.

Theorem 4.8 (Gauss-Bonnet Formula). *Let n be a nonzero integer and d be an even number not greater than n. If X is a compact d-submanifold of $\mathbb{R}^n$, then*

$$\int_X K_d \delta = \chi(X)\, \mathrm{vol}(S^{n-1})$$

where K_d, δ and $\chi(X)$ respectively denote the Gauss curvature, the Euler characteristic of X and the Lebesgue measure on X, and $\mathrm{vol}(S^{n-1})$ is the volume of the unit ball in $\mathbb{R}^n$.

Proof. See, *e.g.*, Spivak [1999] □

EXAMPLE 4.9. Considering the gradients of f and g as defined in Example 4.1, one gets the vector fields defined over $\mathbb{R}^2$ by $\nabla f(x,y) = 2xy(y,x)$ and $\nabla g(x,y) = 3x^2y^2(y,x)$. In this context, combining Theorem 4.8 with Theorem 4.6 yields:

$$\int_{S^1} \mathrm{d}\ell(\gamma) = \int_{S^1} \frac{\nabla f(x,y)}{|\nabla f(x,y)|_2} \cdot \mathrm{d}x\mathrm{d}y = \int_0^{2\pi} \begin{vmatrix} \sin\theta & -\sin\theta \\ \cos\theta & \cos\theta \end{vmatrix} \mathrm{d}\theta = \int_0^{2\pi} \sin 2\theta \mathrm{d}\theta = 0$$

and

$$\int_{S^1} \mathrm{d}\ell(\gamma) = \int_{S^1} \frac{\nabla g(x,y)}{|\nabla g(x,y)|_2} \cdot \mathrm{d}x\mathrm{d}y = \int_0^{2\pi} \sin 2\theta \mathrm{d}\theta = 0$$

whence, in both cases, the field index within the unit ball is 0. In other words, critical point are *not* detected.

For this reason, Mann and Rockwood [2002] propose to examine separately the faces and edges of the cells, because any continuous curve of critical points entering a cell would necessarily pierce at least one of its edges or faces. However, several problemes arise, that have been partly addressed by these authors:

1. a set of non-isolated critical points, *e.g.*, a closed curve, could be entirely contained within the interior of a cell and would thus not pierce any of the the cell edges nor faces;
2. because of projection of the vector field unto lower-dimensional manifolds (faces and edges), "false positives" may appear, *i.e.* non-zero vectors whose projection is zero;
3. non-curve sets of non-isolated critical points, *e.g.*, a surface of such points, can intersect a cell face, forming a set of non-isolated critical points of the vector field projected onto this face. In this case, the Gauss-Bonnet would still not guarantee the detection of theses critical points, in particular if this surface does not intersect a cell edge.

4.3 Seed Cells

In order to accelerate the process of extracting isosurfaces, we may try to find a subset of the cells which, for any given isovalue, $\sigma \in \mathbb{R}$, intersect each connected component of $\Phi(\boldsymbol{r}) - \sigma = 0$ at least once. This technique has been considered by Wilhelms and van Gelder [July 1992], Cignoni et al. [June 1997], van Kreveld et al. [1997] for linear cells. We have implemented the sweep method introduced by van Kreveld et al. [1997] for higher order elements. Only one small change is required to the algorithm; if any cell contains critical points on its interior, it must be added to the list of seed cells.

Since our data structure contains a lookup table for converting a DOF node into a list of cells that reference it, finding the list of neighboring cells while traversing an isosurface component is simple. We are also storing the range of values taken on over a face by virtue of storing critical point information, so it is trivial to compute a list of neighboring cells that intersect an isosurface for further processing.

5 Conclusions

We have presented the design of a storage-efficient mesh representation for visualizing higher order finite elements. An implementation exists and the appendix describes a file format for storing these meshes on disk. We have discussed the mathematics of isosurfacing finite elements with nonlinear geometric and scalar field maps, including an analysis of several techniques for identifying critical points of the scalar field to be isosurfaced. Our prototype implementation, which uses a streaming tetrahedral refinement of the mesh, works with some datasets, but problems exist when there are non-isolated critical points or several isolated critical points in a single cell. Although there is clearly much work left, even naive tetrahedralizations of cell corner nodes and critical points produce results acceptable for visualizations.

Appendix: XML File Specification

SHOE meshes are stored in a file format based on Kitware's existing unstructured grid XML file format. The differences are

1. The `VTKFile` element's `type` attribute shall be `ShoeMesh`.
2. Rather than an `UnstructuredGrid` element, the `VTKFile` shall contain a `ShoeMesh` element.
3. The `Piece` element shall have a `NumberOfDofNodes` attribute, in addition to `NumberOfPoints` and `NumberOfCells`, whose value is the number of higher order nodes required to specify the mesh.
4. The `Piece` element shall contain two `FunctionData` elements, one with a `Type` attribute set to `Fields` and the other with a `Type` attribute set to `Geometry`.
   (a) The `FunctionData` element for `Geometry` shall contain exactly one `VaryingDataArray` element which specifies the higher order mode coefficients for corresponding linear values in the `Point` element of the current `Piece`.
   (b) The `FunctionData` element for `Fields` shall contain one `VaryingDataArray` element per point field specified in the current `Piece`'s `PointData` element.
   (c) Each `FunctionData` element may have attributes named `Scalars`, `Vectors`, and/or `Tensors`. The value of each of these attributes must be the name of a scalar, vector, or tensor field defined by the `Piece`'s `PointData` element. These attributes define which fields serve as the default scalar, vector, or tensor field in the event that multiple scalar, vector, or tensor fields are defined.
   (d) Each `FunctionData` element may contain an `Extrema` element used to store extremal values and critical points of the fields defined by the `FunctionData` element.
5. The `Piece` element shall *not* contain a `Cells` element as defined for `UnstructuredGrid` meshes. Instead, the `Piece` element shall contain a `Cells` element as described here:
   (a) The `Cells` element shall contain a `FreeRange` element with a `Name` attribute of `Connectivity`; a `Type` attribute set to `Int32` or `Int64`; an `EmptyEntry` set to `-1`, a `Size` attribute set to the number of *used* entries in the freelist; a `Capacity` attribute set to the total allocated length of the freelist; and a `format` attribute set to `ascii`. The `FreeRange` element shall contain two other elements: `Values` and `Holes`.
      i. The `Values` element shall contain a whitespace separated list of connectivity entries for all the cells. If an entry is unused (*i.e.*, it is a hole), then a value of `-1` shall be used. (This should be the value of the `EmptyEntry` attribute but we only support `-1` currently.)
      ii. The `Holes` element shall have a `MaxDeadSize` attribute whose value is the maximum size of any contiguous hole allowed in the freelist – any larger holes must be decomposed into multiple holes of smaller size. The `Holes` element shall also have an `RunLength` attribute whose value is the number of entries in the `Holes` element. The `Holes` element itself contains a whitespace-separated list of integers. The first entry specifies the number of holes of size 1. A list of offsets of these holes follow if the number is greater than zero. The next entry specifies the number of holes of size 2, again followed by the offsets of these holes. This continues until the number of holes of length `MaxDeadSize` and their offsets are specified.
   (b) The `Cells` element shall contain a `List` element with a `Name` attribute of `CellDefinitions`, a `Type` attribute of `Int32*13`, a `NumberOfEntries` attribute whose value specified the number of cell definition entries in the mesh, and a `format` attribute with a value of `ascii`. The `List` element shall contain a whitespace-separated list of integers, 7 plus 3 times the number of fields in the mesh for each cell definition. The integers correspond

to the cell shape, cell interpolant, cell product space, the number of elements referring to the definition, 3 integers specifying the order of the geometric interpolant, and 3 integers for each field specifying the order of the interpolant for that field.

(c) The `Cells` element shall contain a `FreeList` element with a `Name` attribute of `CellSpecs`, a `Type` attribute of `Int32,Int32,Int32`, a `NumberOfEntries` attribute specifying the number of cells in the mesh, a `NextEntry` attribute specifying the next entry of the `FreeList` to be used on further insertions, a `Capacity` attribute whose value is the total allocated length of the `FreeList`. The `FreeList` element shall contain `Values` and `Holes` elements.

i. The `Values` element shall have a `Size` attribute set to the length of the freelist. The element shall contain a whitespace-separated list of values, three per cell. The first is an offset into the list of cell definitions, the second is an offset into the connectivity list, and the final is a bit vector of permutations of all the DOF nodes the cell references.

ii. The `Holes` element shall have a `Size` attribute set to the number of holes in the `FreeList`. Unlike the `FreeRange`, only holes of size 1 are allowed. The holes element shall contain a whitespace-separated list of values that serve as offsets of holes in the freelist.

(d) The `Cells` element shall contain two `VaryingDataArray` elements. One such element shall have a `Name` attribute with a value of `CornerLinks`, a `Type` attribute with a value of `Int32`, a `NumberOfComponents` attribute with a value of 1, and a `format` with a value of `ascii`. The `VaryingDataArray` shall contain a `Values` element and an `Offsets` element.

i. The `Values` element shall have a `Size` attribute specifying the number of integers in the element itself. The element shall contain a whitespace-separated list of integers of the length given above. Each integer is a cell ID.

ii. The `Offsets` element shall contain a whitespace-separated list of offsets into the `Values` element above. There shall be `NumberOfPoints` values. Each offset specifies the list of cells that reference a given point.

(e) The second `VaryingDataArray` element contained in the `Cells` element shall have a `Name` attribute of `DofNodeLinks`. All other attributes in the `VaryingDataArray` element shall be the same as specified in the previous `VaryingDataArray` element. This `VaryingDataArray` element shall also contain `Values` and `Offsets` elements as specified above, the only difference being that the number of entries in the `Offsets` element shall be the number of DOF nodes.

An example of the file format is below. It describes 2 hexahedral elements with two point fields ("Density" and "Deflection"), each with an order 6 interpolant. The hexes share a single face, leaving the mesh with a total of 12 points and 33 DOF nodes. Some values particular to the mesh have been removed for readability, but the original file, `test_two_hexes.shoe` should be available with this report.

```
<?xml version="1.0"?>
<VTKFile type="ShoeMesh">
  <ShoeMesh Name="Two Flat Hexes"
   Description="Two hexahedral elements with zeroed out higher order geometry/fields">
    <Piece NumberOfPoints="12" NumberOfCells="2" NumberOfDofNodes="33">
      <FunctionData Type="Fields" Scalars="Density" Vectors="Deflection">
        <VaryingDataArray Name="Density" NumberOfComponents="1"
         Type="Float64" format="ascii">
          <Values Size="168">
            0 1 1 ... <!-- 168 total values, 1 per mode -->
          </Values>
```

```
    <Offsets>
        0   5  ... <!-- 33 total values, 1 per node -->
    </Offsets>
  </VaryingDataArray>
  <VaryingDataArray Name="Deflection" NumberOfComponents="3"
   Type="Float64" format="ascii">
    <Values Size="504">
      0 0 0  ... <!-- 504 more values, 3 per mode -->
    </Values>
    <Offsets>
        0   5  ... <!-- 33 total values, 1 per node -->
    </Offsets>
  </VaryingDataArray>
</FunctionData>
<PointData Scalars="Density" Vectors="Deflection">
  <DataArray Name="Density" NumberOfComponents="1"
   type="Float64" format="ascii">
    0.599082  0.240662  ... <!-- 12 total values, 1 per point -->
  </DataArray>
  <DataArray Name="Deflection" NumberOfComponents="3"
   type="Float64" format="ascii">
    0  -0.104829   0.094838 ... <!-- 36 total values, 3 per point -->
  </DataArray>
</PointData>
<CellData Scalars="Pressure">
  <DataArray Name="Pressure" NumberOfComponents="1"
   type="Float64" format="ascii">
    2.7182818284590451 3.1415926535897932 <!-- 1 per cell -->
  </DataArray>
</CellData>
<Points>
  <DataArray Name="Geometry" NumberOfComponents="3"
   type="Float64" format="ascii">
    -1 -1 -1 ... <!-- 36 total values, 3 per point -->
  </DataArray>
</Points>
<FunctionData Type="Geometry">
  <VaryingDataArray Name="Geometry" NumberOfComponents="3"
   Type="Float64" format="ascii">
    <Values Size="504">
      0 0 0  ... <!-- 504 more values, 3 per mode -->
    </Values>
    <Offsets>
        0   5   ... <!-- 33 values total, 1 per node -->
    </Offsets>
  </VaryingDataArray>
  <Extrema>
    <VaryingDataArray Name="Geometry Extrema" NumberOfComponents="7"
     NumberOfOffsetComponents="2" Type="Float64" format="ascii">
      <Values Size="35">
        1 1  0.      2   1 1 0.8
        ... <!-- 35 total values, 7 per critical point -->
      </Values>
      <Offsets>
```

```
            0 1    1 4   ... <!-- 66 total values, 2 per node -->
          </Offsets>
        </VaryingDataArray>
      </Extrema>
    </FunctionData>
    <Cells>
      <FreeRange Name="Connectivity" Type="Int32" EmptyEntry="-1"
       Size="54" Capacity="64" format="ascii">
        <Values>
          0  1  ... <!-- 54 total values, 27 per hexahedral cell -->
        </Values>
        <Holes MaxDeadSize="27" RunLength="27">
         0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
        </Holes>
      </FreeRange>
      <!-- Tuples are (Shape, Interpolant, Product Space, NumElements,
           GeometricOrder[3], FunctionOrder[][3])   -->
      <List Name="CellDefinitions" Type="Int32*13"
       NumberOfEntries="1" format="ascii">
        9 1 1   2   6 6 6     6 6 6   6 6 6
        <!-- cell shape, interpolant, product space + 3 order values per field -->
      </List>
      <!-- Tuples are (CellDef<offset into above>,
           Offset<connectivity>, NodePermutations)   -->
      <FreeList Name="CellSpecs" Type="Int32,Int32,Int32"
       NumberOfEntries="2" NextEntry="2" Capacity="8" format="ascii">
        <Values Size="6">0 0 0    0 27 0</Values>
        <Holes Size="0"></Holes>
      </FreeList>
      <VaryingDataArray Name="CornerLinks" Type="Int32"
       NumberOfComponents="1" format="ascii">
        <Values Size="16">
         0   0 1   0 1   0   0   0 1   0 1   0   1   1   1   1
        </Values>
        <Offsets>0 1 3 5 6 7 9 11 12 13 14 15</Offsets>
      </VaryingDataArray>
      <VaryingDataArray Name="DofNodeLinks" Type="Int32"
       NumberOfComponents="1" format="ascii">
        <Values Size="38">
          0   0 1   0   0   0   0 1   0 1   0   0   0 1   0   0   0
          0   0   0   0   0 1   0   1   1   1   1   1   1   1   1
          1   1   1   1   1   1   1
        </Values>
        <Offsets>
          0 1 3 4 5 6 8 10 11 12 14 15 16 17 18 19 20 21 23 24 25
          26 27 28 29 30 31 32 33 34 35 36 37
        </Offsets>
      </VaryingDataArray>
    </Cells>
  </Piece>
 </ShoeMesh>
</VTKFile>
```

References

C. Bajaj. *Introduction to Implicit Surfaces*. Morgan Kaufman, 1997. URL http://citeseer.ist.psu.edu/article/bajaj97implicit.html.

C. Bajaj and G. Xu. Spline approximations of real algebraic surfaces. *Journal of Symbolic Computation*, 23:315 – 333, 1997. Special Issue on Parametric Algebraic Curves and Applications.

A. J. Chung and A. J. Field. A simple recursive tessellator for adaptive surface triangulation. *Journal of Graphics Tools*, 5(3), 2000.

P. Cignoni, P. Marino, C. Montani, E. Puppo, and R. Scopigno. Speeding up isosurface extraction using interval trees. *IEEE Transactions on Visualization and Computer Graphics*, 3(2):158–170, June 1997.

Patricia Crossno and Edward Angel. Isosurface extraction using particle systems. In Roni Yagel and Hans Hagen, editors, *IEEE Visualization '97*, pages 495–498, 1997. URL http://citeseer.ist.psu.edu/35215.html.

Mathieu Desbrun and Marie-Paule Gascuel. Animating soft substances with implicit surfaces. *Computer Graphics (SIGGRAPH Proceedings)*, 29:287–290, 1995. URL http://citeseer.ist.psu.edu/desbrun95animating.html.

T. Y. Li, T. Sauer, and J. A. Yorke. The cheater's homotopy: an efficient procedure for solving systems of polynomial equations. *SIAM J. Numer. Anal.*, 26:1241–1251, 1989.

Gregorio Malajovich and J. Maurice Rojas. Random sparse polynomials. In *Proceedings of the FoCM 2000, special meeting in honor of Steve Smale's 70th birthday*, pages 251–266. World Scientific, 2002. URL http://lyric.labma.ufrj.br/~gregorio/papers.html#momentum.

S. Mann and A. Rockwood. Computing singularities of 3d vector fields with geometric algebra. In *Proceedings of IEEE Visualization*, pages 283–289, October 2002.

Dinesh Manocha and John Canny. Multipolynomial resultant algorithms. *Journal of Symbolic Computation*, 15(2):99–122, 1993.

Nelson Max, Peter Williams, and Claudio Silva. Approximate volume rendering for curvilinear and unstructured grids by hardware-assisted polyhedron projection. *International Journal of Imaging Systems and Technology*, 11(1):53–61, 2000a.

Nelson Max, Peter Williams, and Claudio Silva. Cell projection of meshes with non-planar faces. In *Proceedings of Dagstuhl 2000*. IBFI gem. GmbH, 2000b.

A. P. Morgan. *Solving polynomial systems using continuation for engineering and scientific problems*. Prentice-Hall, Englewood Cliffs, New Jersey, 1987.

Detlef Ruprecht and Heinrich Müller. A scheme for edge-based adaptive tetrahedron subdivision. In Hans-Christian Hege and Konrad Polthier, editors, *Mathematical Visualization*, pages 61–70. Springer Verlag, Heidelberg, 1998. URL http://citeseer.ist.psu.edu/223528.html.

M. Spivak. *Comprehensive Introduction to Differential Geometry*, volume 1-2. Publish or Perish, 1999.

Barton T. Stander and John C. Hart. Guaranteeing the topology of an implicit surface polygonization for interactive modeling. *Computer Graphics*, 31(Annual Conference Series):279–286, 1997. URL http://citeseer.ist.psu.edu/article/stander97guaranteeing.html.

Barna Szabo and Ivo Babuska. *Finite Element Analysis*. John Wiley & Sons, 1991.

Marc J. van Kreveld, René van Oostrum, Chandrajit L. Bajaj, Valerio Pascucci, and Daniel Shikore. Contour trees and small seed sets for isosurface traversal. In *Symposium on Computational Geometry*, pages 212–220, 1997.

Luiz Velho. Simple and efficient polygonization of implicit surfaces. *Journal of Graphics Tools*, 1 (2):5–24, 1996.

J. Verschelde. Algorithm 975: PHCPACK: A general-purpose solver for polynomial systems by homotopy continuation. *ACM Transactions on Mathematical Software*, 25(2):251–276, 1999.

J. Verschelde, P. Verlinden, and R. Cools. Homotopies exploiting newton polytopes for solving sparse polynomial systems. *SIAM J. Numer. Anal.*, 31:915–930, 1994.

Layne T. Watson, Stephen C. Billups, and Alexander P. Morgan. Algorithm 652: HOMPACK: A suite of codes for globally convergent homotopy algorithms. *ACM Transactions on Mathematical Software*, 13(3):281–310, 1987.

Jane Wilhelms and Allen van Gelder. Octrees for faster isosurface generation. *ACM Transactions on Graphics*, 11(3):201–227, July 1992.

DISTRIBUTION:

1 Timothy J. Baker
Mechanical & Aerospace Engineering Department
Engineering Quadrangle
Princeton University
Princeton, NJ 08544

2 Prof. Richard Crawford
Mechanical Engineering Department C2200
The University of Texas at Austin
Austin, TX 78703

1 Rahul Khardekar
1634 Oxford Street
Apt 305
Berkeley, CA 94709

1 Pr Pascal Frey
Laboratoire J-L Lions
Université Pierre et Marie Curie
Boîte courrier 187
75252 Paris cedex 05, France

1 Pr Jérôme Pousin
I.N.S.A. Lyon
Center for Mathematics
Bâtiment Léonard de Vinci
69621 Villeurbanne cedex, France

2 Will Schroeder
Kitware
28 Corporate Drive
Suite 204
Clifton Park, NY 12065

1 MS 1110
David M. Day, 9214

1 MS 0822
Lee Ann Fisk, 9227

1 MS 9012
Jerry A. Friesen, 8963

1 MS 9915
Curtis L. Janssen, 8961

1 MS 9915
Mike Koszykowski, 8961

1 MS 9051
Andy McIlroy, 8351

3 MS 9051
Philippe P. Pébay, 8351

1 MS 1110
Louis Romero, 9214

1 MS 9915
Mitchel W. Sukalski, 8961

1 MS 9012
Gary J. Templet, 8963

4 MS 9012
David C. Thompson, 8963

1 MS 0822
Brian N. Wylie, 9227

3 MS 9018
Central Technical Files, 8940-2

2 MS 0899
Technical Library, 4916

1 MS 0619
Review & Approval Desk, 4916