An Automatic Adaptive Refinement and Derefinement Method for 3D Elliptic Problems

Feng Yu[1], Yarden Livnat[2] and Christopher R. Johnson[2]
[1]EDS-Unigraphics
10824 Hope Street, 2N-205
Cypress, CA 90630
and
[2]Department of Computer Science
University of Utah
Salt Lake City, UT 84112
Email: yuf@edsug.com, ylivnat@cs.utah.edu and crj@cs.utah.edu
WWW: http://www.cs.utah.edu/~sci/

November 13, 1996

Contents

1 Introduction 1

2 Finite Element Method 2

3 Error Bounds 5

4 Implementation 8

5 Examples 14

6 Discussion and Conclusion 15

Abstract

We present the theory and implementation for a new automatic adaptive h-refinement and -derefinement method for two- and three-dimensional elliptic problems. An exact lower error bound for derefinement is obtained theoretically in terms of the finite element solution, complementing the various known upper error bounds for refinement. These error bounds are used to determine where to insert and/or remove mesh elements. To implement the method, we utilize an adaptive two- and three-dimensional Delaunay tessellation that preserves the Delaunay properties while locally adding points to, or deleting points from, the previous Delaunay mesh. We provide computational examples for two- and three-dimensional elliptic problems on unstructured grids.

1 Introduction

Adaptive methods have been demonstrated to be an effective way to achieve efficient and accurate finite element solutions to partial differential equations [1]. A goal of adaptive methods is to obtain the solutions within a specified accuracy with minimal computational cost. The basic ingredient for applying adaptation is the knowledge of (exact) error bounds or error estimates. These error bounds are normally derived from the finite element solution via *a posteriori* processes. For computational efficiency, local error bounds are used whenever possible so that one is able to specify where to adapt; global error bounds are generally substituted in cases where no local error bounds are available. An optimal adaptation requires both refinement and derefinement. While refinement seeks to minimize error and thus obtain the desired accuracy, derefinement seeks to optimize the computational aspects of the problem by minimizing unnecessary degrees of freedom caused by *overestimating* the solution accuracy. The importance of the latter process becomes evident when one wishes to reach a given overall accuracy when constrained by limited resources or when the problem is time-dependent and it is not practical to continually add degrees of freedom in different locations as the solution varies over time.

There have been many papers that have derived various upper error bounds that are appropriate for refinement [2, 3, 4, 5, 6, 7, 8, 9, 10, 11]. However, few very plausible discussions about lower error bounds for derefinement appear in the literature [12, 13].

Due to its own complexity and its relevancy to the physical contents of problems, the implementation of the adaptive process stands as a relatively independent issue. For this reason, implementation of large-scale adaptive methods is often neglected in more theoretical papers on adaptation. The simplest adaptive implementation is to regenerate the whole finite element mesh for the new configuration every time the adaptation is performed. Obviously, this method can require a substantial duplication of effort, especially when the researcher needs to adapt only small portions of the previous, already generated mesh. A computationally more effecient implementation is to refine and/or derefine the mesh in a local manner, and iterate each adaptive step automatically until the desired accuracy is met. As the mesh becomes more and more deformed during adaptation, mesh construction directly affects the convergence speed of the finite element solution. This leads one to consider ways of incorporating the adaptation procedure with an optimal mesh generation scheme. Delaunay tessellation is such a scheme in the sense that it equally spaces the mesh elements over an arbitrary set of nonuniform points. The Delaunay method particularly suites large-scale application problems with complex geometry, since no additional complexity is added to deal with the unstructured nature of the mesh. The refinement and derefinement are usually accomplished by inserting or removing configuration points in specified regions of the previous mesh.

This paper will illustrate how these connected issues of adaptation and mesh generation by presenting an adaptive method using automatic refinement and derefinement for two- and three-dimemsional elliptic problems. In particular, we address "Poisson-like" problems, from theory through implementation. Theoretically, we utilize well known upper error bounds for refinement. However, we also derive a new lower error bound as a counterpart to the upper error bounds. This new lower error bound has the advantages of being local, of not requiring

an additional subproblem to be solved, and of being easy to implement. Implementationally, we use Delaunay tessellation to generate a two- or three-dimensional finite element mesh. To generate the Delaunay mesh, we use a Watson-type algorithm [14, 15], because it works for arbitrary dimension and is thus appropriate for both two- and three-dimensional cases. Furthermore, the Watson-type algorithm blends naturally with the local adaptive point insertion and local point insertion algorithm. Each adaptive iteration is automatically performed element-by-element by comparing the error bounds calculated from the current finite element solution with (arbitrarily) allowed tolerances. Points are added into the previous Delaunay mesh where the errors are too large and removed from the previous mesh where the errors are too small in such a way that the new mesh preserves the Delaunay property. Implementation for the general three-dimensional case is, as one would imagine, more complicated than it is for the two-dimensional case. This is especially true for the derefinement phase of the process because of the potential for geometic degeneracy within the mesh structure. We note that the theory and implementation are general and can be implemented into any two- or three-dimensional elliptic finite element problem as long as the error bounds are specificed.

The paper is organized as follows: In section 2, we give a brief mathematical formulation of the finite element method for elliptic partial differential equations. This provides the necessary framework for the subsequent derivation of the adaptative scheme. In section 3, we present some of the more popular upper error bounds and derive a new lower error bound for Poisson-like equations. We present the implementation of our adaptive method in section 4. In section 5, we present several examples of our results. Finally, we conclude with a discussions of future work in section 6.

2 Finite Element Method

We begin by formulating a general elliptic problem in three space: $(x, y, z) \in \Omega \subset R^3$. Consider the linear, self-adjoint and positive-definite partial differential equation for the smooth function $u(x, y, z)$:

$$\nabla \cdot (a \nabla u) + bu = f \quad in\ \Omega \tag{2.1}$$

with the general boundary conditions:

$$a \nabla u \cdot \hat{n} = g \quad on\ \Gamma_1 \equiv \partial\Omega_1, \tag{2.2}$$

$$u = u_0 \quad on\ \Gamma_2 \equiv \partial\Omega_2. \tag{2.3}$$

Here $a(x, y, z)$ is a symmetric and Euclidean tensor field parameter with six (three for the two-dimensional case) independent components, appropriate for anisotropic problems, and $b(x, y, z)$ is a scalar field parameter. $f(x, y, z)$ represents an internal source scalar field, and $g(x, y, z)$ an external vector field applied in the normal direction $\hat{n}$ of the Neumann boundary. $u_0(x, y, z)$ is the Dirichlet boundary value of u. Note that in our formulation, the boundary is either Neumann or Dirichlet: $\partial\Omega = \partial\Omega_1 \cup \partial\Omega_2$ and $\partial\Omega_1 \cap \partial\Omega_2 = \emptyset$

The weak form of equation (2.1) requires u to satisfy the following integral equation for any piecewise smooth function $v(x, y, z)$:

$$\int_\Omega (a\nabla u \cdot \nabla v + buv) d\Omega = \int_\Omega fv d\Omega + \int_{\partial\Omega_1} gv d\Gamma \tag{2.4}$$

subject to

$$u = u_0, \; v = 0 \quad on \; \Gamma_2. \tag{2.5}$$

To avoid a singular integration from (2.4), one usually requires u, v to be in the first Sobolev space $H^1(\Omega)$. The so-called Sobolev spaces are simply Hilbert spaces of function u such that the derivatives of u are L_2-integrable:

$$H^m(\Omega) = \{u \mid D^\alpha u \in L_2(\Omega), |\alpha| \leq m\} \tag{2.6}$$

where m, α are nonnegative integers, and

$$D^\alpha v = \frac{\partial^{|\alpha|} v}{\partial x_1^{\alpha_1} \ldots \partial x_m^{\alpha_m}} \qquad |\alpha| = \alpha_1 + \ldots + \alpha_m \tag{2.7}$$

In other words, we are able to define a series of norms for $u \in H^m(\Omega)$:

$$||u||_{H^m} = (\sum_{|\alpha|=m} \int_\Omega |D^\alpha u|^2 d\Omega)^{1/2}. \tag{2.8}$$

For comparison, recall the L_m norms associated with the Lebesgue spaces take the form:

$$||u||_{L_m} = (\int_\Omega |u|^m d\Omega)^{1/m}. \tag{2.9}$$

Notice $||u||_{H^0}$ is just $||u||_{L_2}$. Other norms may be defined. Of particular relevance to our derivation is the energy norm

$$||u||_E = (\int_\Omega (a\nabla u \cdot \nabla u + buu)\, d\Omega)^{1/2}. \tag{2.10}$$

Consider the finite element version of the above formulation. First, the three-dimensional domain must be bounded and is discretized into a set of non-overlapping tetrahedra (triangles for the two-dimensional case):

$$T_{h,p} = \{\Omega_k\}_{k=1}^n, \quad \bigcup_{k=1}^n \Omega_k = \Omega. \tag{2.11}$$

The discretization size of each element Ω_k is specified by h_k, while the degree of the interpolation is specified by an integer p_k. Such a restriction leads to the standard finite element subspace of polynomial functions, formally denoted by

$$\begin{aligned} V_{h,p} \;\; = \;\; & \{u \mid u \in H^1(\Omega), \quad u \in C^0(\Omega), \\ & u|_{\Omega_k} \in \text{polynomials of order at most } p_k, \forall\, \Omega_k \in T_{h,p}\}. \end{aligned} \tag{2.12}$$

The elliptic finite element problem is an approximation of equation (2.4) for $u_{h,p}, v_{h,p} \in V_{h,p}$ such that:

$$\begin{aligned} & \sum_{k=1}^{n} \int_{\Omega_k} (a \nabla u_{h,p} \cdot \nabla v_{h,p} + b u_{h,p} v_{h,p})\, d\Omega \\ = & \sum_{k=1}^{n} \int_{\Omega_k} f v_{h,p}\, d\Omega + \sum_{k=1}^{n} \int_{\partial\Omega_k \cap \Gamma_1} g v_{h,p}\, d\Gamma. \end{aligned} \tag{2.13}$$

The discretized equation (2.13) is a finite dimensional problem for any specified set of interpolation p_k. This is evident if one explicitly expands $u_{h,p}$ and $v_{h,p}$ in terms of the interpolation bases. Therefore, in principle, the finite element solution $u_{h,p}$ can always be obtained. As the discretization size $h = max\{h_k\} \to 0$, the approximation $u_{h,p}$ will converge to the exact solution u.

Linear elements are a common choice for application problems such that $p = max\{p_k\} = 1$. In this case, $u_h(x, y, z)$ is a linear function of its values at the nodes of tetrahedra (or other types of element). For each tetrahedron k, we have

$$u_h^k(x, y, z) = \sum_{i=1}^{4} N_i(x, y, z) u_h^{k,i} \tag{2.14}$$

where,

$$u_h^{k,i} = u_h^k(x_i, y_i, z_i). \tag{2.15}$$

The basis function,

$$N_i(x, y, z) = a_i + b_i x + c_i y + d_i z, \tag{2.16}$$

is linear in coordinates and satisfies the interpolation condition $N_i(x_j, y_j, z_j) = \delta_{ij}$. Substituting equation (2.14) into (2.13), equation (2.13) becomes

$$\begin{aligned} & \sum_{k=1}^{n} (\int_{\Omega_k} (a \nabla N_i \cdot \nabla N_j + b N_i N_j) d\Omega) u_h^{k,j} v_h^{k,i} \\ = & \sum_{k=1}^{n} (\int_{\Omega_k} f N_i d\Omega) v_h^{k,i} + \sum_{k=1}^{n} (\int_{\partial\Omega_k \cap \Gamma_1} g N_i d\Gamma) v_h^{k,i}. \end{aligned} \tag{2.17}$$

Since equation (2.16) holds for arbitrary $v_h^{k,i}$, we obtain a set of linear equations

$$\sum_{k=1}^{n} K_{ij}^k u_h^{k,j} = \sum_{k=1}^{n} I_i^k, \quad \text{with } u_h = u_0 \text{ on } \Gamma_2 \text{ nodes} \tag{2.18}$$

where the stiffness matrix $K_{ij}^k = \int_{\Omega_k} (a \nabla N_i \cdot \nabla N_j + b N_i N_j) d\Omega$ and the source/boundary term $I_i^k = \int_{\Omega_k} f N_i d\Omega + \int_{\partial\Omega_k \cap \Gamma_1} g N_i d\Gamma$ are number coefficients. Assuming *reasonable* source/boundary conditions, u_h can then be solved using a linear equation solver.

3 Error Bounds

The absolute error of a finite element solution,

$$e(x,y,z) = u(x,y,z) - u_{h,p}(x,y,z), \tag{3.1}$$

is usually, by definition, measured collectively using a particular norm over certain regions. A commonly-used measure is the energy norm of equation (3.1). The local error norm sums over an element:

$$||e||^2_{E(\Omega_k)} = \int_{\Omega_k} (a\nabla(u-u_{h,p})\cdot\nabla(u-u_{h,p}) + b(u-u_{h,p})(u-u_{h,p}))d\Omega, \tag{3.2}$$

and the global error norm covers the entire domain:

$$||e||^2_{E(\Omega)} = \sum_{k=1}^{n} ||e||^2_{E(\Omega_k)}. \tag{3.3}$$

Obviously, one is not able to evaluate equation (3.2) or equation (3.3) directly, as the exact solution u is not available. A central objective of adaptive methods is to obtain error bounds (or error estimates) in terms of the finite element solution $u_{h,p}$ such that the exact error norm in equation (3.2) or (3.3) is no larger or smaller than such estimated bounds.

There are many successful examples of the use of upper error bounds $||e||_E \leq \epsilon_{up}(u_{h,p})$ as functionals of $u_{h,p}$, particularly for elliptic problems. There are a number of estimates that can be used, each of which has relative advantages in performance with respect to precision, efficiency, and reliability. The approaches of Babuska et al. [2, 3], of Oden et al., and of other authors [4, 5, 6, 7] result in rather tight error bounds. However, the approach of Johnson and Hansbo [8, 9] has been shown to be more efficient since it avoids solving extra local problems without much loss of accuracy. The so-called Z^2-approach of Zienkiewicz and Zhu [10, 11] uses a different philosophy. Their method directly uses the difference of two finite element solutions $e_{h,p} = u_{h',p'} - u_{h,p}$, with $u_{h',p'}$, as the estimation of the true error to equation (3.1), to achieve a presumably more accurate solution. However, while this method may yield more precise adaptation in the sense that all estimations are local, the nature of the method makes it generally less reliable as the residual may be orthogonal to the current error estimation direction in the Z^2 estimator. It is also the case that the Z^2 approach does not handle singularities well.

A well-known global upper error bound, which is simple to implement, is reliable, and does not *over-pursue* accuracy, is [1]

$$||e||_{E(\Omega)} \leq ch^p||u||_{H^{p+1}(\Omega)} \tag{3.4}$$

where c is a constant independent of h and p. To a high magnitude of error, $u_{h,p}$ may be substituted for u. Another simple upper error bound with $p = 1$ is

$$||e||_{L_2(\Omega)} \leq ch^2||u||_{H^2(\Omega)}, \tag{3.5}$$

$$||\nabla e||_{L_m(\Omega)} \leq ch||D^2u||_{L_m(\Omega)}. \tag{3.6}$$

However, upper bounds are only half of the error estimation story. Without the complement lower error bounds, one could hardly achieve an optimal adaptation. For static problems, it is a question of using the given resources in the most efficient way. For time-dependent problems, regions that need refinement at one moment may turn out to be over-meshed at another moment. Thus the need for adaptive refinement *and* derefinement, especially for parabolic problems in which the spatial values are functions of time. However, even the elliptic problems in equations (2.4)-(2.5) could be implicitly time-parameterized when the sources and/or boundary conditions vary in terms of time *externally*. Currently there is a lack of successful lower error bound implementations for large-scale three-dimensional problems. Most of the estimates presented in the literature are global in nature and are very complicated to implement. Many require one to solve difficult local problems prior to the final derefinement step. While these lower bound estimates may be theoretically sound, they are, impractical, if not impossible, to implement.

However, there exists a lower error bound that is local, efficient, reliable and easy to implement for elliptic problems. For simplicity, we consider the case $b = 0$ in equation (2.4), i.e., the Poisson-like equations.

To begin, recall that our goal is to find a quantity whose integral, as a functional of the finite element solution $u_{h,p}$ and the source/boundary f, g, is of the same magnitude of the error norm. It is probable for such a quantity to be formed from certain combinations of the source term and the boundary-type term from the governing equation (2.4), in which the free function v may be specified at will. There are not many such combinations when one enforces the consistency to the quantity's physical dimension.

The challenge in finding such a quantity is twofold. First, it seems unlikely that there exists a direct combination that will lead to desired error bound, as surely some previous investigation would have uncovered this relationship. Secondly, finding such a quantity becomes more difficult if one wants to maintain the locality of the error bound. To see this, notice that the approximation in equation (2.13) has the same source and obeys the same boundary condition as the exact version in equation (2.4). The difference of the two solutions is globally controlled by

$$\int_{\Omega} a \nabla e \cdot \nabla v d\Omega = 0, \quad \forall v \in V_{h,p}, \tag{3.7}$$

which is a particular form of Cea's theorem [16, 1]. This relation plays a significant role in the derivation of global error bounds. However, locally, only the exact integral equation holds for any small region:

$$\int_{\Omega_k} a \nabla u \cdot \nabla v d\Omega = \int_{\Omega_k} f v d\Omega + \int_{\partial \Omega_k} a (\nabla u \cdot \hat{n}) v d\Gamma \tag{3.8}$$

in equivalence to its differential form in equation (2.1). One cannot exploit the local version of equation (2.13) because of the discontinuity of the internal finite element boundary terms. It is the essence of the finite element method that one makes the problem simpler locally by giving up some smoothness while simultaneously imposing constraints globally to match the exact solution.

To answer the above difficulties we seek an approximation of the error in a collective manner, e.g, in the form of an energy norm. For many applications it should be sufficient

to start with the gradient of a quantity instead of the quantity itself. Meanwhile, we notice that in case the contribution from the volume term is dominant over that from the boundary terms, using equation (3.8) would be enough to sufficiently approximate the error. In this way we observe that one may define the gradient as

$$\nabla\epsilon = ha^{-1}(f + \nabla\cdot(a\nabla u_{h,p}))\hat{r_0} \tag{3.9}$$

for the error approximation ϵ. Here a^{-1} is well-defined since the tensor is nondegenerate. $\hat{r}_0$ is a unit vector of arbitrary direction. The insertion of element size h matches the magnitude and dimension of the error. The claim is that the energy norm of ϵ is a local lower error bound of the finite element solution $u_{h,p}$:

$$c_{low}||\epsilon||_{E(\Omega_k)} \leq ||e||_{E(\Omega_k)}. \tag{3.10}$$

Before proceeding with the formal proof, let us list some useful inequalities we will use in the proof [1]:

$$\int_{\Omega_k} a\nabla u\cdot\nabla v d\Omega \leq ||u||_{E(\Omega_k)}||v||_{E(\Omega_k)}, \tag{3.11}$$

$$\int_{\Omega_k} uv d\Omega \leq ||u||_{L_2(\Omega_k)}||v||_{L_2(\Omega_k)}, \tag{3.12}$$

$$||u||_{E(\Omega_k)} \leq a_{max}||\nabla u||_{L_2(\Omega_k)}, \tag{3.13}$$

$$||\nabla u||_{L_2(\Omega_k)} \leq (1/a_{min})||u||_{E(\Omega_k)}, \tag{3.14}$$

$$||u||_{L_2(\partial\Omega_k)} \leq c(h^{-1}||u||^2_{L_2(\Omega_k)} + h||\nabla u||^2_{L_2(\Omega_k)})^{1/2}, \tag{3.15}$$

$$||\nabla u\cdot\hat{n}||_{L_2(\partial\Omega_k)} \leq c(h^{-1}||\nabla u||^2_{L_2(\Omega_k)} + h||\nabla^2 u||^2_{L_2(\Omega_k)})^{1/2}, \tag{3.16}$$

$$||\nabla^p u||_{L_2(\Omega_k)} \leq ch^{q-p}||\nabla^q u||_{L_2(\Omega_k)}, \quad 0\leq q\leq p\leq 2 \tag{3.17}$$

where u, v belong to the appropriate Sobolev spaces. Equations (3.11)-(3.12) represent the Cauchy-Schwarz inequalities. Equations (3.13)-(3.14) hold because the tensor a is symmetric and positive definite with $a_{max/min}$ = maximum/minimum value of the diagonalized a. The fact that the finite element boundary contribution does not exceed the magnitude of the volume contribution is stated by equations (3.15)-(3.16). Finally, equation (3.17) compares the norms of a function with different orders of differentiation.

Now we proceed to prove equation (3.10). Starting with the equation (3.9), we have

$$\begin{aligned}\|\epsilon\|^2_{E(\Omega_k)} &= \int_{\Omega_k} a\nabla\epsilon\cdot\nabla\epsilon\, d\Omega \\ &= \int_{\Omega_k} hf(\hat{r_0}\cdot\nabla\epsilon)\, d\Omega + \int_{\Omega_k} h\nabla\cdot(a\nabla u_{h,p})(\hat{r_0}\cdot\nabla\epsilon)\, d\Omega \end{aligned}\tag{3.18}$$

We can replace the source term by applying equation (3.8) with $v = h\hat{r_0}\cdot\nabla\epsilon$ and expand the second term by integration by parts. Equation (3.18) then becomes

$$\begin{aligned}||\epsilon||^2_{E(\Omega_k)} &= \int_{\Omega_k} ha\nabla u\cdot\nabla(\hat{r_0}\cdot\nabla\epsilon)d\Omega - \int_{\partial\Omega_k} h(a\nabla u\cdot\hat{n})(\hat{r_0}\cdot\nabla\epsilon)d\Gamma \\ &\quad + \int_{\partial\Omega_k} h(a\nabla u_{h,p}\cdot\hat{n})(\hat{r_0}\cdot\nabla\epsilon)d\Gamma - \int_{\Omega_k} ha\nabla u_{h,p}\cdot\nabla(\hat{r_0}\cdot\nabla\epsilon)d\Omega \\ &= \int_{\Omega_k} ha\nabla e\cdot\nabla(\hat{r_0}\cdot\nabla\epsilon)d\Omega - \int_{\partial\Omega_k} h(a\nabla e\cdot\hat{n})(\hat{r_0}\cdot\nabla\epsilon)d\Gamma. \end{aligned}\tag{3.19}$$

Expanding the first term using equations (3.11)-(3.17) and noting that the unit vector $\hat{r_0}$ is free to take arbitrary direction, we obtain

$$\begin{aligned}\int_{\Omega_k} ha\nabla e \cdot \nabla(\hat{r_0} \cdot \nabla\epsilon)\, d\Omega &\leq h\|e\|_{E(\Omega_k)}\|\nabla\epsilon\|_{E(\Omega_k)} \\ &\leq c_1\frac{a_{max}}{a_{min}}\|e\|_{E(\Omega_k)}\|\epsilon\|_{E(\Omega_k)} \end{aligned} \tag{3.20}$$

Similarly, the second term in equation (3.19) can be expanded to,

$$\begin{aligned}&\int_{\partial\Omega_k} h(a\nabla e \cdot \hat{n})(\hat{r_0} \cdot \nabla\epsilon)\, d\Gamma \\ &\leq c_2 h\|\nabla e \cdot \hat{n}\|_{L_2(\partial\Omega_k)}\|\nabla\epsilon \cdot \hat{n}\|_{L_2(\partial\Omega_k)} \\ &\leq c_3(\|\nabla e\|^2_{L_2(\Omega_k)} + h^2\|\nabla^2 e\|^2_{L_2(\Omega_k)})^{\frac{1}{2}}(\|\nabla\epsilon\|^2_{L_2(\Omega_k)} + h^2\|\nabla^2\epsilon\|^2_{L_2(\Omega_k)})^{\frac{1}{2}} \\ &\leq c_4\|\nabla e\|_{L_2(\Omega_k)}\|\nabla\epsilon\|_{L_2(\Omega_k)} \\ &\leq c_5\|e\|_{E(\Omega_k)}\|\epsilon\|_{E(\Omega_k)}\end{aligned} \tag{3.21}$$

where the $c's$ are all positive constants independent of the finite element parameters h, p. The $c's$ do, however depend on the problem parameter a_{max}/a_{min}. Substituting equations (3.21, 3.22) back into equation (3.19) we arrive at,

$$\|\epsilon\|^2_{E(\Omega_k)} \leq c_5\|e\|_{E(\Omega_k)}\|\epsilon\|_{E(\Omega_k)} \tag{3.22}$$

Finally, canceling one $||\epsilon||_{E(\Omega_k)}$ from both sides, we obtain equation (3.10). The global lower error bound is automatically

$$c_{low}(\sum_{k=1}^{n}||\epsilon||^2_{E(\Omega_k)})^{1/2} = c_{low}h(\sum_{k=1}^{n}\int_{\Omega_k}(f + \nabla\cdot(a\nabla u_{h,p}))^2(\hat{r}_0 a\hat{r}_0)d\Omega)^{1/2}. \tag{3.23}$$

It would be natural to extend our result to the $b \neq 0$ case, but we will leave this exercise for future work.

Given both upper and lower bounds, our adaptive method can be simply stated as the following: refine the region where the upper error bound is $\geq \delta_{up}$ and derefine the region where the lower error bound is $\leq \delta_{low}$. However, we should note that this is not quite as straightforward as it first appears. First, the theory does not determine the leading constants c_{up} and c_{low} in the error bounds; these constants could be approximated through test calculations of $u_{h,p}$-dependent parts of the error bounds for different h values, or by approximating some pseudo-exact solutions. Also, the adaptation is in terms of *normalized* tolerances δ_{up}/c_{up} and δ_{low}/c_{low}. Secondly, when no local error bound is available, we have to use the local contribution to a global error bound as a substitute.

4 Implementation

Unfortunately, throughout much of the development of adaptive methods, theory and implementation have been treated as largely orthogonal subjects. From the viewpoint of realistic

applications, a theory without an effective implementation is not very appealing. On the other hand, a program without the support of a rigorous theory, no matter how extraordinary its output would appear, may be unreliable and even misleading. Despite their mutual importance, interfaces that embody both sound adaptive theories and practical implementations are not common, and such interfaces for large-scale three-dimensional problems with complex geometries are especially rare. For optimal adaptations that apply refinement and derefinement, only a few examples of such theories exist at all, and most of those that do exist utilize uniform grids [12, 17, 13, 18].

The situation is concretely illustrated by various implementational issues. No matter what adaptive method one uses to solve the problem, one first needs to choose an appropriate finite element method. For problems over simple geometric domains with homogeneous, isotropic physics, this may not be an important point: regular, uniform grids would be the obvious choice. However, with problems dealing with a complex geometry as well as with complicated physics, a well designed unstructured mesh configuration may significantly reduce the total number of elements necessary to define the geometry and physics and thus will reduce the overall solution time. As such, we do not simply seek an algorithm that is best for initial mesh generation: that is more or less considered as a given starting configuration upon which the adaptation takes place. We are more interested in finding an algorithm that incorporates mesh generation, refinement, and derefinement all in an optimal way. We also consider the scope of adaptation to be carried out. Essentially, we want to minimize the amount of repeated work during the mesh adaptation; thus, we prefer to refine and derefine the mesh locally. Moreover, we always seek efficient, reliable, and easily implementable error bounds.

For unstructured problems, we utilize the Delaunay tessellation to create the finite element mesh, and then to apply the adaptive scheme. The general procedure is outlined as follows:

```
main()
{
  point_configuration_input();
  mesh_generation();
  parameter_setting\_up();
  finite_element\_solution();
  error_checking();
  while(errors not within specified tolerances)
  {
    refinement();
    derefinement();
    parameter_updating();
    finite_element_solution();
    error_checking();
  }
  solution_output();
}
```

Let us describe the program step by step. First, we input the set of coordinates of

all initial finite element nodes, along with the geometric constraints imposed by the external/internal boundaries (We also note that we are able to generate the original Delaunay meshes from surfaces of triangulated data as well.). In terms of pure adaptation, the coarse finite element configuration – connectivities of points should also be given, regardless how this initial mesh is generated. Let us discuss briefly Delaunay tessellation. Delaunay tesselation works by minimizing obtuse angles within the mesh. Because of this minimizaiton, a minimal number of elements are needed to describe even a complex geometry. Furthermore, Delaunay tessellation is well suited for arbitrary, nonuniform point configurations.

There are a number of Delaunay mesh generation algorithms. Some have been optimized for the two-dimensional case but cannot be extended to three-dimensions and/or are inappropriate for incorporating refinement and derefinement. We utilize the Watson-type [14, 15] algorithm – a natural n-dimensional algorithm for local adaptation. This algorithm relies primarily on the insertion of a new point into an existing Delaunay mesh. The algorithm then removes any elements too close to the point and then reforms the Delaunay mesh by adding new elements in the local area bounded by the removed elements, as follows:

```
point_insertion(point p)
{
  for (each tetrahedron/triangle t of mesh)
  {
    if (p within the circumsphere of t)
      {
         put all faces of t into list l;
         remove t from mesh;
      }
   }
   remove faces that appear twice on l from l;
   for (each remaining face f on l)
   {
      form tetrahedron/triangle t' in terms of p and f;
      add t' to mesh;
   }
}
```

The algorithm then generates the mesh by inserting every point sequentially into a simple reference mesh that is large enough to surround all the set of points and can be easily set up by hand. One example of such a mesh might be a box with five tetrahedra or a square with two triangles. The algorithm then removes all the elements related to the reference mesh. The entire process works as follows:

```
mesh_generation()
{
   reference_mesh_making();
   for (each boundary point p in configuration)
   {
```

```
      point_insertion(p);
   }
   reference_mesh_removal();
   for (each nonboundary point p in configuration)
   {
      point_insertion(p);
   }
}
```

To save costs, reference elements are removed after all boundary points are inserted; nonboundary points are put into the mesh within the natural boundary.

With the finite element geometry set-up, we assign the physical parameters a, f, g, u_0, etc. onto relevant elements. For this paper, we use linear basis functions, where the coefficients of the basis functions from equation (2.15) are of the form

$$a_i = \frac{(-1)^{i1}}{6V}[x_{i1}(y_{i2}z_{i3} - y_{i3}z_{i2}) + x_{i2}(y_{i3}z_{i1} - y_{i1}z_{i3}) + x_{i3}(y_{i1}z_{i2} - y_{i2}z_{i1})], \quad (4.1)$$

$$b_i = \frac{(-1)^{i}}{6V}(y_{i2}z_{i3} - y_{i3}z_{i2} + y_{i3}z_{i1} - y_{i1}z_{i3} + y_{i1}z_{i2} - y_{i2}z_{i1}), \quad (4.2)$$

$$c_i = \frac{(-1)^{i1}}{6V}(x_{i2}z_{i3} - x_{i3}z_{i2} + x_{i3}z_{i1} - x_{i1}z_{i3} + x_{i1}z_{i2} - x_{i2}z_{i1}), \quad (4.3)$$

$$d_i = \frac{(-1)^{i}}{6V}(x_{i2}y_{i3} - x_{i3}y_{i2} + x_{i3}y_{i1} - x_{i1}y_{i3} + x_{i1}y_{i2} - x_{i2}y_{i1}), \quad (4.4)$$

$$V = \frac{1}{6}(a_1 + a_2 + a_3 + a_4). \quad (4.5)$$

Here we use abbreviated indices $ij = i + j \bmod 4$ and $|V|$ = volume of the tetrahedron (the two-dimensional version of this formula can be simply derived from the above formula by letting $z = 0$). Equation (2.17) is now a simple finite dimensional linear matrix equation of the form $Ax = b$ for unknown u_h. We need only to select an effective solver and solve the system.

With the approximation solution available, we do the *posteriori* error checking. The upper and lower error bounds are chosen as the local norm of equations (3.4) $ch||u_h||_{H^2(\Omega_k)}$ and (3.10) respectively, while the specification of (normalized) tolerances is problem-dependent and varies in terms of the desired accuracy. We test each finite element and make lists of both to-be-refined and to-be-derefined elements. We will insert a point at the center of each element on to-be-refined list (we do not address the question of what is the best insertion position for a particular element/error), and remove a point if all the elements containing it as a node are on the to-be-derefined list. Note that we can choose to insert points along boundaries if the point distribution is not dense enough, however, we rarely remove boundary points so as to maintain the geometric constraints of the problem.

Knowing where to refine and derefine the mesh, we are in position to implement the point insertion and point removal functions. The main feature here is the conservation of the Delaunay structure after the two operations – to counter local reconstructions of the Delaunay mesh. For the former part, it is obvious that we may use the identical *point_insertion()*

function as introduced during mesh generation. At this point, let us emphasize that a basic assumption of Delaunay algorithms is that the Delaunay mesh is unique. This is true generically from the fundamental theory of Delaunay tessellation: the chance of violation is negligible. However, degeneracies, as pointed out by [14, 15], do occur, especially in the three-dimensional case. A simple way to handle the degeneracies is to perturb the position of the point to be off the circumsphere. In the extreme situation that the point is also on the same plain with a face of the element, we may want to move the point in finite scale.

Removing points, however, is another story. If there were no degeneracy, this would be an easy inverse process of point insertion. However, the inverse process in not unique and we must proceed differently for derefinement. The basic strategy is to remove the elements surrounding the point together with the original point. We then form the local Delaunay mesh in terms of the points on the surrounding boundary:

```
point_removal(point p)
{
   for (each element t of mesh)
   {
      if (a node of t is p)
      {
        put the face f opposite to p on t into face_list l_f;
        put every point on f into point_list l_p if not already there;
        remove t from mesh;
      }
   }
   local_mesh_forming(l_f,l_p);
}
```

The difficulty in implementing the derefinement occurs in the *local_mesh_forming()* step. It seems to be a non-issue since we know how to regenerate the local Delaunay mesh, with or without the assistance of an auxiliary local box/square mesh. However, the key point here is to have the overall Delaunay structure preserved. The task is thus much more complicated: generate the Delaunay mesh under the constraint that the (discretized) boundary surface of the mesh has exactly the set of faces in the list l_f. This is the compatibility condition of the local Delaunay mesh with the rest of the original mesh. Only when this condition is satisfied is the entire mesh is conforming and remains Delaunay. The part that makes the statement nontrivial is the non-uniqueness of the Delaunay mesh over a given set of points. Because of the degeneracies, there are several possible paths that lead to the same final mesh configuration. Different paths may result in different mesh boundaries, and merely breaking this ambiguity by randomly picking a path, e.g., via perturbing point positions, will usually not lead to the required boundary. Although the mesh surface may remain with the same geometry, it is discretized differently. For instance, a cube can be Delaunay-tessellated in a number of different ways (even the number of tetrahedra can be non-unique: 5 or 6), in which each square bounding face may be crossed in either way accordingly. The indefiniteness inside the mesh may not matter as long as the mesh is Delaunay, but it is the source of possible invalid surface discretizations that we must avoid. In two-dimensional case, it doesn't bother

us since different paths of triangulation lead to the same final boundary discretization (extra triangles outside the concave boundary may easily be removed). A simple example is that a square may be triangulated in two ways, but the square boundary is always the same. Unfortunately in three dimensions, surface discretization is not conserved in most of the cases. Furthermore, it is inappropriate to handle the problem through certain post processes such as modifying the surface or adding degeneracy-breaking points, since we are *derefining* the mesh and dealing with the fixed *internal* boundaries.

The manner we form the local mesh is constructive. The idea is to start with a face on the face_list l_f. We then form a tetrahedron with one of the points in the point_list l_p according to the Delaunay criterion. We then tessellate the local region recursively by tracing from each of the other faces of the tetrahedron just formed. Each trace may end up with either the correct or the wrong face on the surface due to degeneracy. If a trace leads to a wrong face, we reject it and try another possibility out of the degenerate set. The recursive process continues until all traces end correctly. Because the local region is simply connected, all faces on l_f will be covered, thus the surface-discretization-constrained Delaunay mesh is obtained. Note the number of recursive steps vary with the degree of degeneracy, but is always limited within the finite local region. The (high-level) algorithm reads as follows:

```
local_mesh_forming(list l_f, list l_p)
{
  let f be a face on l_f;
  face_tracing(f,l_f,l_p);
}

face_tracing(face f, list l_f, list l_p)
{
   flag = 0;
   for (each point p on l_p but not on f)
   {
      form tetrahedron t in terms of f and p
      if (t not already exist and circumsphere of t not contain
      any point on l_p but not on t)
    {
        flag = 1;
        add t to mesh;
        for (each face f' ≠ f on t)
       {
           if (f' not on l_f)
             {
                 if (f' on region surface or
                 face_tracing(f',l_f,l_p) = FALSE)
               {
                   flag = 0;
                   break;
               }
```

```
            }
        }
         if (flag =  1)
        {
            return TRUE;
        }
      }
    }
     remove tetrahedra added along this trace from mesh;
     return FALSE;
}
```

After each iteration of refinement and derefinement, we update the physical parameters for all new elements and solve the problem over the new mesh and then check the errors again. The process stops when errors are within the specified tolerances and the final mesh configuration and/or finite element solution may be output at the end of our implementation.

5 Examples

In this section we provide a selection of two- and three-dimensional examples of results obtained from our adaptive theory and implementation. It seems more interesting to illustrate them by plotting the changes of mesh structures due to adaptations, rather than give every detail of the process. To provide easy to view figures, we will not present the results from large-scale data sets although we have utilized our method for large-scale problems in computational medicine involving millions of tetrahedral elements.

[A note to the reviewers: We were unable to obtain acceptable gray-scale visualizations of the large-scale three-dimesional examples. If this paper is accepted, we will include at least 1-2 more large-scale 3D examples. Thanks.]

Example 1: 2D and 3D L-problems

We start off with this standard problem. The domain is a square without the upper left quarter. Let it be bounded by lines connecting points $(0,0),(2,0),(2,2),(1,2),(1,1),\ (0,1)$. We impose the Dirichlet boundary condition $u|_{x=0} = 1, u|_{x=2} = 0$; and Neumann condition $g = 0$ on other pieces of boundaries. The input is a regular point configuration so the mesh is uniform, Fig. 1.1. After adaptation, Fig 1.2, the mesh is contracted to point $(1,1)$. A similar L-shape problem in 3D is depicted in Fig. 1.3 and Fig. 1.4.

Example 2: 2D Bioelectric Field Problem

This problem is derived from a slice of our full three-dimensional thorax model [19, 20]. The problem is defined as the forward problem in electrocardiography in which one estimates

voltages on the surface of the body from measured heart voltages. The governing equation is

$$\nabla \cdot (\sigma \nabla V) = -I \quad in\ \Omega \tag{5.1}$$

$$V = V_0\ on\ \Gamma_E, \quad \sigma \nabla V \cdot \hat{n} = 0\ on\ \Gamma_T \tag{5.2}$$

where V is the electrostatic potential, I is the primary cardiac current source per unit volume, σ is the (inhomogeneous and anisotropic) conductivity tensor. The epicardial (heart) surface is Dirichlet and the torso (body) boundary is Neumann. The initial mesh over the complex geometry is shown by Fig. 2.1. After adaptation, the configuration is denser near the internal heart boundary where the change of potential is fast, Fig. 2.2.

Example 3: Simulation of a Heart Beat

We now extend the previous example to include time dependency. In particular, we employed a set of measurments taken on the surface of the heart during a open chest surgery. A slice was taken from the full three-dimensional model. These measurments were than applied as time-dependent Dirichlet boundary conditions on the epicardium. The algorithm performed several iterataions before a new set of boundary conditions were applied. Three snapshots of the progresive adaptation during the heart beat are depicted in Figs. 3.1-3.3. Fig 3.4 shows an augmented view of the mesh where the the z coordinate was assigned the value of the electric voltage. The refinement in areas of high gradient and the derefinement in areas of low gradient is evident.

6 Discussion and Conclusion

We have developed an adaptive method using automatic refinement and derefinement for elliptic problems in two and three dimensions from theory through implementation. To avoid unnecessary complications at this stage, we did not pursue the most general case for all types of partial differential equations, especially in the theoretical discussion. We note that the error bounds do not have to be associated with the Delaunay mesh, they can be used with any finite element mesh. In turn, our implementation is appropriate for any given error bounds. In this regard, we expect to obtain effective (upper or lower) bounds for problems, e.g., of parabolic type, in that solutions are dynamically time-dependent hence an optimal mesh depends primarily on both refinement and derefinement. On the implementation side, we could try the h-p method over Delaunay mesh hoping the performance/cost ratio would be better. Of course, seeking optimization of finite element solving schemes does not have to be through the *posteriori* adaptation only, though it plays a fundamental role. Other optimization methods,such as making a reasonably distributed initial mesh from certain physical considerations, will increase the speed of convergence as well. In conclusion, we emphasize that our adaptive method is appropriate for large, complex, inhomogeneous, anisotropic, time-dependent, and even moving finite element problems of elliptic type.

Acknowledgments

This work was supported in part by the National Science Foundation and the National Institutes of Health. The authors would like to thank K. Coles and J. Schmidt for their helpful comments and suggestions. Furthermore, we appreciate access to facilities that are part of the NSF STC for Computer Graphics and Scientific Visualization.

References

[1] P. G. Ciarlet and J.L. Lions. *Handbook of Numerical Analysis: Finite Element Methods, Vol.1-2.* North-Holland, 1991.

[2] I. Babuška and W. C. Rheinboldt. A posteriori error estimators for the finite element method. *Int. J. Numer. Methods Engrg.*, 12:1597, 1978.

[3] I. Babuška. Feedback, adaptivity and a posteriori estimates in finite elements: Aims, theory, and experience. In I. Babuška et al., editor, *Accuracy Estimates and Adaptive Refinements in Finite Element Computations*, page 3. Wiley, New York, 1986.

[4] R. Bank and A. Weiser. Some a posteriori error estimators for elliptic partial differential equations. *Math. Comp.*, 44:283, 1985.

[5] J. T. Oden, L. Demkowicz, W. Rochowicz, and T. A. Westermann. Towards a universal *h-p* finite element strategy, part 2, a posterior error estimation. *Comput. Methods Appl. Mech. Engrg.*, 77:113, 1989.

[6] M. Ainsworth and J. T. Oden. A procedure for a posteriori error estimation for *h-p* finite element methods. *Comput. Methods Appl. Mech. Engrg.*, 101:73, 1992.

[7] R. Verfürth. A posteriori error estimators for the stokes equation. *Numer. Math*, 55:309, 1989.

[8] C. Johnson. *Numercial solution of Partial Differential Equations by the Finite Element Method.* Cambridge University Press, Cambridge, 1990.

[9] C. Johnson and P. Hansbo. Adaptive finite element methods in computational mechanics. *Comput. Methods Appl. Mech. Engrg.*, 101:143, 1992.

[10] O. C. Zienkiewicz and J. Z. Zhu. A simple error estimator and adaptive procedure for pratical engineering analysis. *Int. J. Numer. Method Engrg.*, 24:337, 1987.

[11] M. Ainsworth, J. Z. Zhu, A. W. Craig, and O. C. Zienkiewicz. Analysis of zienkiewicz-zhu a posteriori error estimator in the finite element method. *Int. J. Numer. Method Engrg.*, 28:2161, 1989.

[12] P. K. Jimack. Adaptive error control in the finite element method for time-dependent problem. *Reseach Report, Univ. of Leeds*, 92.11, 1992.

[13] R. Lohner and S. Aita. Tgv tunnel-entry simulations using a finite element code with automatic remeshing. In *Proceedings of AIAA*, number 93-0890. AIAA, 1993.

[14] D. F. Watson. Computing the *n*-dimensional delaunay tessellation with application to voronoi polytopos. *Computer Journal*, 24:167, 1981.

[15] J. C. Cavendish, D. A. Field, and W. H. Frey. An approach to automatic three-dimensional finite element mesh generation. *Int. J. Numer. Method Engrg.*, 21:329, 1985.

[16] J. Cea. Approximation variationnelle des problems aux limites. *Ann. Inst. Fourier (Grenoble*, 14:345–444, 1964.

[17] J.D. Baum and R. Lohner. Three-dimensional store separation using a finite element solver and adaptive remeshing. In *Proceedings of AIAA*, number 91-0602. AIAA, 1991.

[18] P. Mehrotra, J. Saltz, and R. Voigt, editors. *Unstructured Scientific Computation on Scalable Multiprocessors*. MIT Press, Boston, 1992.

[19] C.R. Johnson, R.S. MacLeod, and P.R. Ershler. A computer model for the study of electrical current flow in the human thorax. *Computers in Biology and Medicine*, 22(3):305–323, 1992.

[20] C.R. Johnson, R.S. MacLeod, and M.A. Matheson. Computational medicine: Bioelectric field problems. *IEEE COMPUTER*, pages 59–67, Oct., 1993.

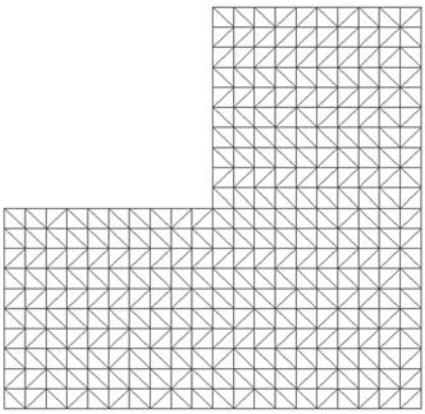

Figure 1: L-problem: original mesh

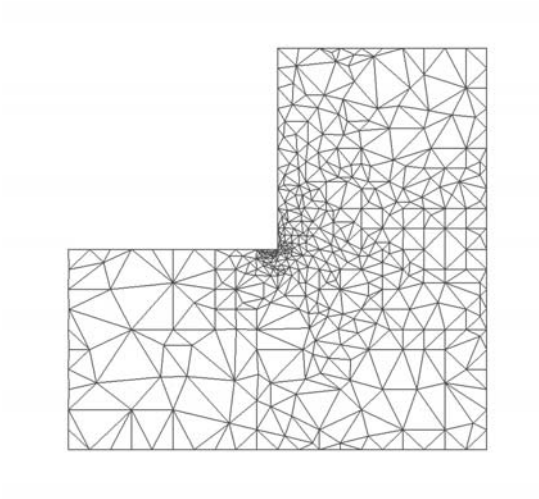

Figure 2: L-problem: adapted mesh

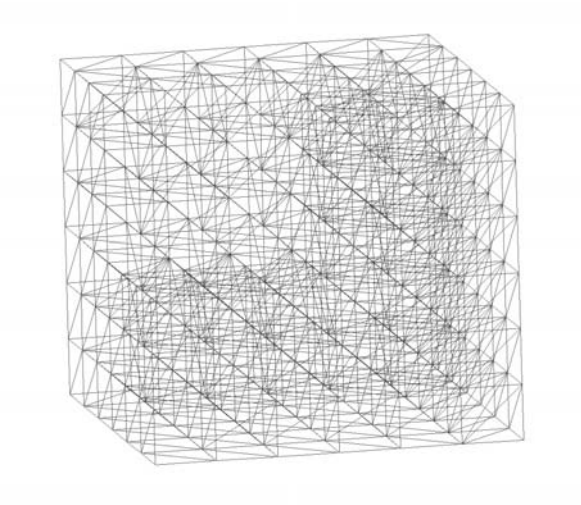

Figure 3: 3d L-problem: original mesh

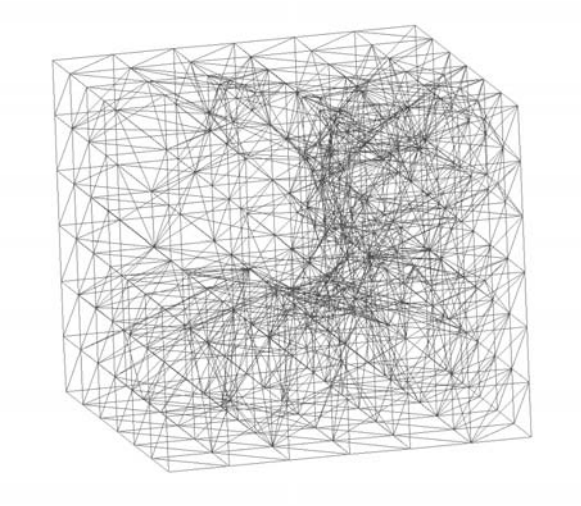

Figure 4: 3d L-problem: adapted mesh

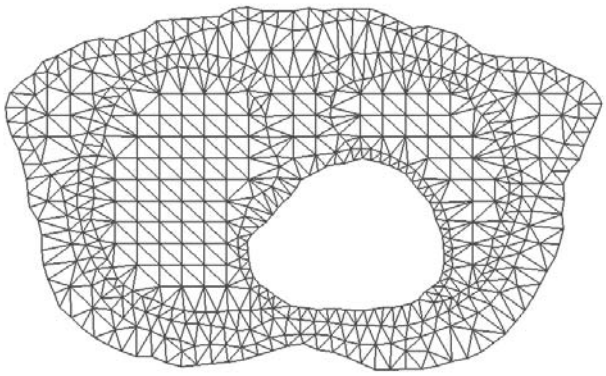

Figure 5: Torso model: original mesh

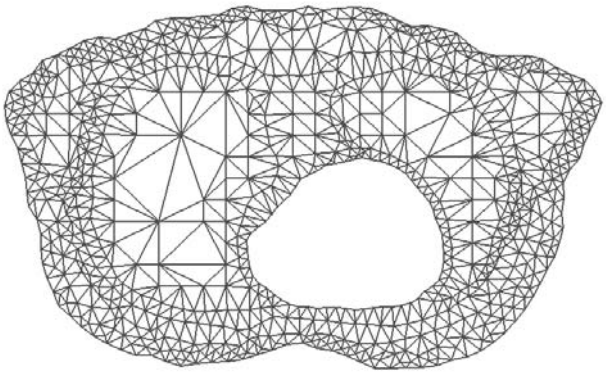

Figure 6: Torso model: adapted mesh

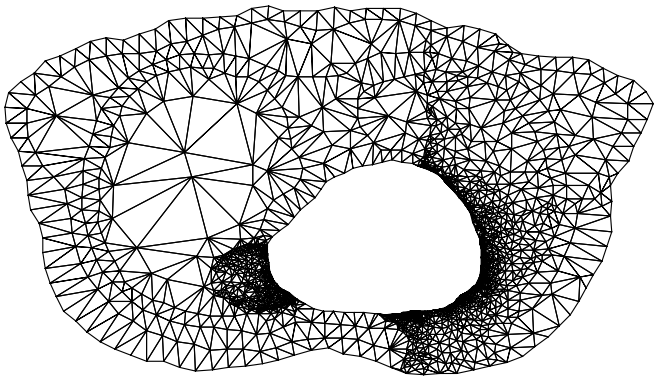

Figure 7: Heart Beat: time = 147

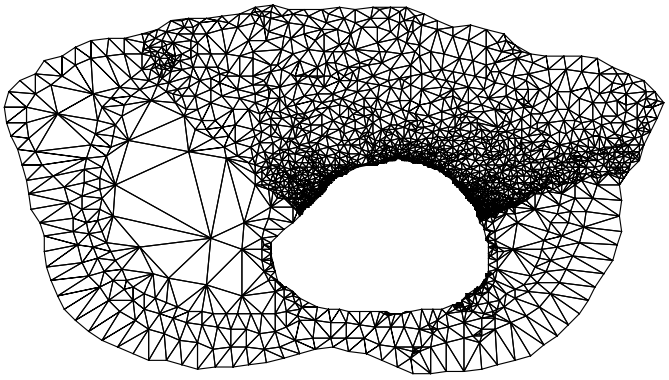

Figure 8: Heart Beat: time = 165ms

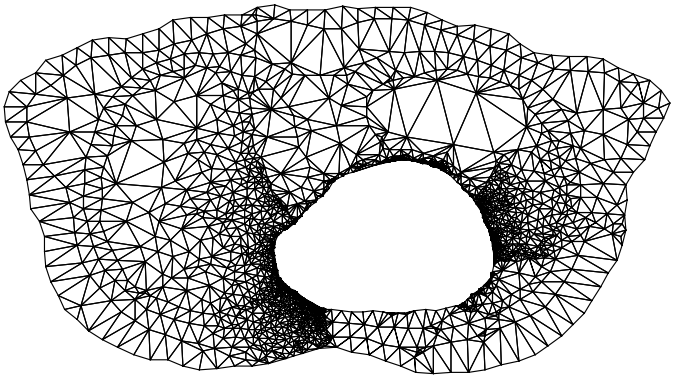

Figure 9: Heart Beat: time = 191ms

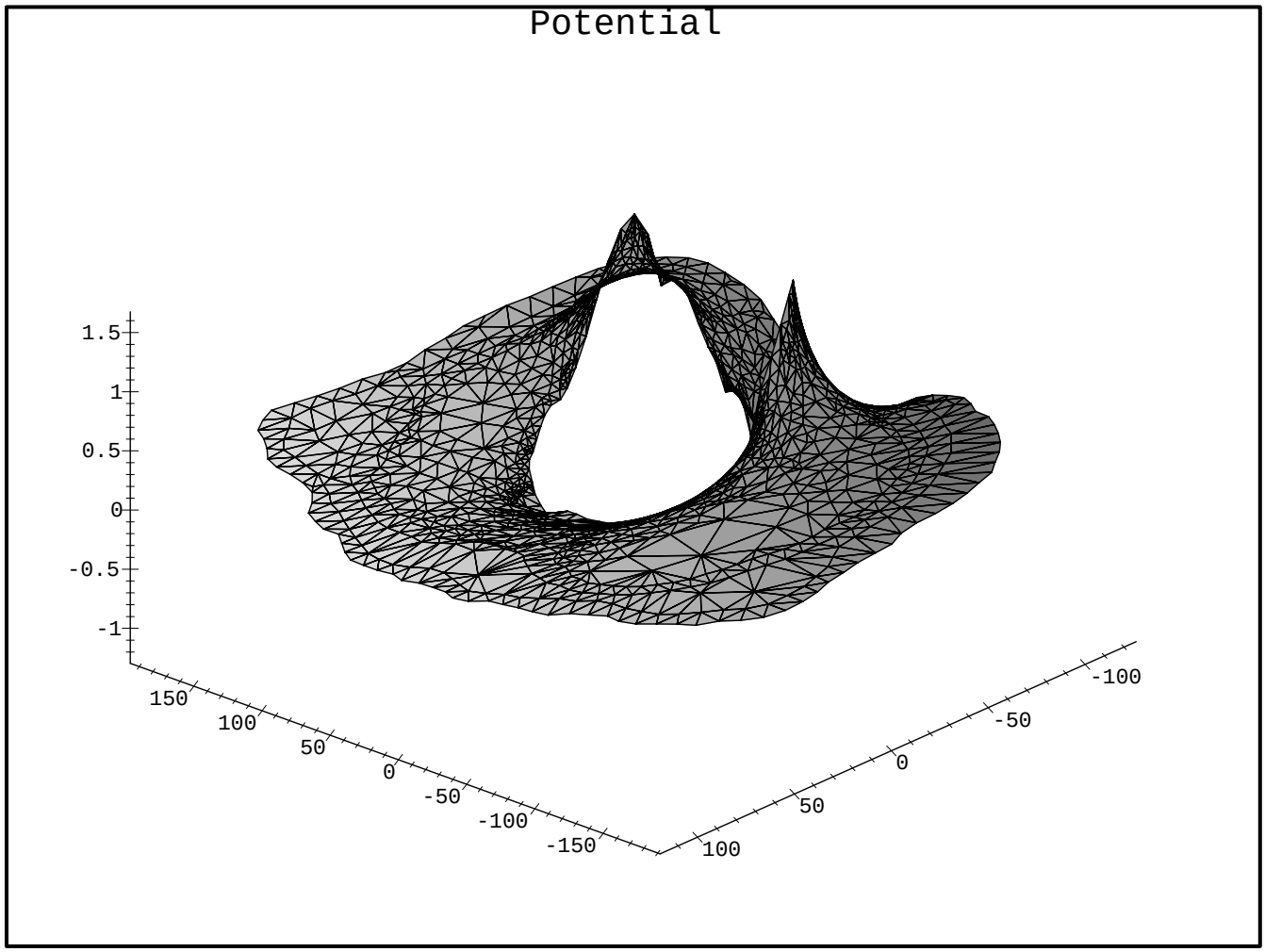

Figure 10: Heart Beat: augmented view