Abstract

Finite element methods in computation physics often require the domain of interest to be discretized, or tiled in some manner with primitive shapes. When the domain itself has an irregular or unknown shape, finding a satisfactory tesselation can be a demanding task: the grid to be generated must conform to an arbitrary (polygonal) boundary, satisfy certain numerical requirements imposed by the finite element method, and in many cases must be *adaptive*—locally refine or coarsen on demand. In this paper a practical algorithm is described for generating grids which satisfy the desired conformal and numerical requirements of the finite element method in two dimensions. Unlike existing algorithms, this algorithm also admits local grid modifications with a bounded effect; a small upper bound on cost exists for maintaining grid requirements following a small local increase or decrease in grid density. The way the grid is constructed, and the guaranteed locality of changes make this algorithm highly amenable to parallelization. Experimental results for a sequential implementation are presented.

Locally-Adaptive Grid Generation Using Quadtrees

Clark Verbrugge[*] Prakash Panangaden[†] Riccardo Pucella[‡]
Department of Computer Science
McGill University
Montréal, Québec, Canada
{clump,prakash,pucella}@cs.mcgill.ca

1 Introduction

Until recently, algorithms in computational physics have largely focussed on using regular geometric figures as the basis of domain discretization. For physical situations where the geometry is irregular, and more importantly where the physics requires a dynamically adaptive grid, such regular tesselations are obviously inadequate. One can fairly easily develop data structures that express irregular grid structures and, with rather more effort, develop satisfactory algorithms that create irregular grids consistent with the requirements of the numerical aproximations being used. Fairly sophisticated grid-generation algorithms have been developed for use with finite element methods; see for example the recent book by P. L. George [11], the lecture notes by Weatherill [16] or the review article by Bern and Eppstein [6].

There is a basic problem with all these schemes if one attempts to use them in conjunction with an adaptive algorithm. In physical applications, one adapts the grid in response to some local criterion (typically the gradient of one of the physical variables exceeds a prescribed bound), and thus one wants to refine (or coarsen) a grid locally. With existing Delaunay-based algorithms, though, even such local changes can cause the entire grid to be scrapped and recomputed—an expensive procedure for large grids. It is of interest therefore to develop an algorithm that allows incremental recomputation of the grid while maintaining the geometrical exigencies of the finite element method. This is what we do in the present paper.

In 1988, Baker, Grosse and Rafferty [2] described an algorithm for grid generation which satisfies the specific numerical constraints of grid generation for the finite element method (no obtuse angles), while still conforming to an arbitrary polygonal boundary. Here, we develop an incremental algorithm. We build on some of the geometric insights of their algorithm but we are forced to deal with a variety of new problems. These are a significant explosion in the number and intricacy of the cases that need to be analyzed and the need to maintain "balance" conditions on the tree structures used with an eye to future parallel implementation.

[*]Research supported in part by NSERC and Centre de recherche informatique de Montréal.
[†]Research supported in part by NSERC and FCAR.
[‡]Research supported by FCAR.

A two-dimensional mesh in the context of the finite element method consists of a discretization of the domain into a non-overlapping set of triangles, connected edge to edge. Discretization into squares is also possible, but in order to have the mesh conform to the arbitrary polygonal boundaries (formed from straight line segments) found in unstructured (irregular) domains, triangles are to be preferred. A mesh, though, need not remain a static object throughout the entire computation. An *adaptive* finite element method accelerates convergence by dynamically changing the grid to place more grid points in the regions of greatest interest/variability, and fewer grid points in the more stable regions. The same accuracy as a much finer grid can then be obtained with fewer grid points; for instance, a moving shock wave can be adaptively tracked with an accuracy otherwise achievable only with a very fine grid covering the entire path of the shock wave. The mesh, then, should efficiently support adaptive reconfiguration.

1.1 Physical Background

The finite element method computes an approximate solution to a differential equation in the following way. One breaks the region into small subregions called "elements." The solution to the differential equation is approximated by some standard function, depending on a few parameters, across the element. Often one uses linear or constant functions. One matches the solution in neighbouring elements across their boundaries and obtains in this way a set of algebraic equations that partially constrain the approximate solution. The iteration to the solution proceeds by using the approximate solution in the differential equation and successively refining the approximation.

This method imposes some basic requirements on the mesh. In order to ensure that the linear interpolation across the boundary of each element leaves the system consistent (i.e. the system is not overconstrained by imposing matching at three points on a linear function), the finite elements must be connected edge-wise, with no vertices located along any edge except at its endpoints. It also imposes the condition that the finite elements completely and disjointly cover the domain of interest.

Unfortunately, not just any triangular mesh will do. For many fluid flow and heat conduction problems, the finite element method demands that the cosine of every angle in the triangulation be positive. Obtuse triangles, ones containing angles larger than $\pi/2$, can cause the generation of physically unrealistic results—such as an increase in temperature given a decrease in energy input—and are therefore undesireable in the mesh.

1.2 Computational Background

In practice one tries to construct a *Delaunay* triangulation. A Delaunay triangulation is a mesh where the circumscribing circle of each triangular element is free of any grid points in its interior. It also has the advantage that the minimum angle in the mesh is itself minimized over all possible triangulations of the same point set [1], and this tends to reduce the number of obtuse triangles. A Delaunay triangulation, however, despite its otherwise very nice properties does not *guarantee* that the mesh will not contain obtuse triangles.

If the mesh is to be dynamically adapted, it must be able to efficiently increase and decrease the density of grid points in a specified area. Since this problem is being investigated with respect to parallel computation, particularly distributed-memory machines, these operations should be both efficient and as *local* as possible, to minimize any inherent communication costs. In other words, any changes made to the grid due to adaptivity should have a localized effect, not requiring the reconstruction of the entire grid. This is actually a requirement only for a coarse-grained parallelization strategy, where the grid itself is distributed among a relatively small number the processors. However, given that the finite element method requires only local computation at each grid point, and has a great deal of dependency between adjacent elements, surface-to-volume ratio arguments suggest that a coarse-grained approach is the most workable.

Unfortunately, the more popular Delaunay algorithms, like Watson's [15], and Bowyer's [10] are not certain to modify a triangulation with only bounded effect. The "edge flipping" technique of Watson, while guaranteed to terminate, is not guaranteed to restrict its locus of activity, and can spread throughout the entire mesh. Bowyer's algorithm has a similar problem: inserting a single point can (in the worst case) require a complete retriangulation of the domain. A dynamic adaptive scheme cannot afford such expensive operations, particularly when the grid is large.

1.3 The Baker, Grosse and Rafferty Algorithm

The algorithm of Baker *et al* [2] works by overlaying a regular square grid of sufficient resolution over the domain. As long as each vertex of the domain coincides with a grid-point, and the grid is fine enough to ensure no more than one edge of the domain intersects a given square, it is possible to triangulate the domain. Each square can be triangulated separately with only acute triangles, including squares with an input edge intersecting them. Some extra effort is needed to deal with acute angles in the domain description—acute angles also constrain the minimum grid resolution. The complete details can be found, of course, in [2].

At first glance this algorithm does not seem to have any special advantages when considering adaptivity; a fixed resolution is certainly not amenable to local changes. However, as they mention (but do not develop) in their discussion of the algorithm, quadtrees can be used to allow for some local variation in grid size. This permits guaranteed local grid refinements, and a significant overall reduction in number of triangles in the grid too.

2 Outline of Our Algorithm

To summarize, the mesh and/or mesh generation algorithm should have the following properties:

1. The mesh should be a triangulation, and all vertices must be only at the corners of the triangles.
2. All triangles should be non-obtuse.
3. The density of grid points within a specified region should be dynamically adjustable.
4. Grid modifications should be as localized as possible.

The grid generation algorithm presented here, described in the next section, possesses the following features:

1. It generates a triangulation respecting arbitrary polygonal boundaries.
2. No triangle has an internal angle larger than $\pi/2$.
3. Starting from a "base" triangulation, the grid can be increased in density (within a specified region), and subsequently reduced as needed.
4. Modifications have a small and greatly-restricted non-local effect.

It should also be noted that since the grid generated has no obtuse angles, it is also automatically a Delaunay triangulation [1], and so it inherits the well-established numerical properties thereof.

The algorithm consists of two main stages. First, the quadtree structure itself is generated—a square large enough to contain the entire domain is recursively decomposed into four smaller squares, until a base level is reached. This base level will depend on the geometry of the input domain, the choice of input vertices, and the necessity of being able to generate acute triangles. In order to ensure this base level can be reached, a number of conditions need to be guaranteed; in section 3 we describe the desired conditions, and how they can be ensured during or by the end of the quadtree generation.

During this construction, the domain exterior is progressively identified and removed from further processing. Most algorithms ignore this facet of grid generation, a variety of means for mathematically distinguishing domain interior from exterior already being in existence (see for example [14]). However, we are required to be efficient, avoid numerical error, and keep computations within the quadtree as locally-contained as possible. We also cannot demand that all input vertices lie in "general position" (no 3 points on a line, no 4 on a circle), as is often assumed in computational geometry algorithms. Hence, in section 4 we develop an algorithm for filtering the domain exterior from the quadtree as we construct it, using only the information contained within an individual quad.

Once the quadtree has been constructed to a base level, the leaves of the quadtree are triangulated. The vertices forming the corners of each quad are added to the domain, and (acute) triangles are generated respecting the individual boundaries of each quad. Once each quad is consistently and completely triangulated with acute triangles, so will be the entire domain. Completing this process for each of the possible quad leaves forms the bulk of the effort in implementing the algorithm; in section 5 we describe the general approach, and provide detailed proofs and descriptions of each of the individual cases.

Once the triangulation has been constructed, it may be necessary to increase (and subsequently decrease) the density of the grid within a specified region, in accordance with the physical criteria discussed above. In section 6, we illustrate how incremental modifications can be made to the grid without necessitating a retriangulation of the entire domain.

In section 7, we prove some upper bounds on the size of the grid generated by our algorithm. Since our algorithm is highly dependent on the exact geometry of the input, experimental results can give a more accurate assessment of the algorithm's efficacy. Thus, in section 8 we provide experimental results for several different domains of varying complexity.

3 Generating the Quadtree

The quadtree can be generated in one of two fashions; either depth-first or breadth-first. The former involves generating the branches of the quadtree one at a time, making each branch as deep as needed before moving on to the next branch. The latter generates the quadtree level-by-level, building all branches at an equal rate. While each version attempts to minimize the consumption of different resources (space and time, respectively), for reasons that will become clear shortly, the breadth-first approach is preferred.

Part of the algorithm for generating the quadtree requires a definition:

Definition 1 *A* two-edge case *is a pair of edges meeting at the corner of a quad that form an acute interior angle to the domain.*

We begin with an initial square large enough to contain the entire input polygon. We will then recursively divide a given square s, at depth d in the quadtree, into four squares of depth $d+1$ if any of the following properties hold.

Vertex Condition An input vertex of the polygon is contained within s (including the boundaries), and is not coincident with one of the four corners of s.

Edge Condition More than one input edge of the domain properly intersects s, and the half-planes (domain side) determined by edges of at least one pair of such edges intersect within s, and such a pair is not a two-edge case.

Balance Condition Any square at depth d' with $d' > d+1$ shares a side with s, and s does not contain only two-edge cases, nor is s entirely external to the domain.

3.1 Vertex Condition

The vertex condition is self-explanatory—it merely ensures that each input vertex lies on the corner of a grid square, which is the primary operating assumption for this algorithm. Note that since the quadtree recursively divides itself in two with respect to both the x and y axes, in order for the algorithm to terminate the input vertices must have some finite base-2 representation. The maximum base-2 precision will then be a lower bound on the maximum depth of the quadtree.

3.2 Edge Condition

The edge condition is necessary to keep the number of quad configurations that must be triangulated small. It is not possible to demand that each quad be triangulated acutely when one might be intersected by an arbitrary number of input edges, each of which must be taken into account. Hence, quads are generally restricted to just a single input edge. However, even if there are multiple input edges intersecting a quad, when the domains to be triangulated (as indicated by the interior half-plane of each input edge) do not intersect, there can be no conflict if each such domain is triangulated separately.

There is one exception to the edge condition. If two edges meet at a corner of the quad and form an acute interior angle (to the domain), the *two-edge case*, it will still be possible to triangulate the acute region, and so this situation need not cause the quad to be deepened.

Note that while it is possible to triangulate a quad containing more than two edges, it is also possible to demand that no more than two edges properly intersect any quad. Quads containing more than two edges can be forced to be subdivided, and the algorithm will still terminate. This follows because in a simple polygon (or even one with holes) only pairs of lines can intersect, and between any non-intersecting pair of lines is a minimum distance. Once quads are smaller than that minimum distance, the two non-intersecting lines cannot be in the same quad. Thus, at a cost of slightly more triangles in complex regions, the task of generating the tree can be made significantly easier.[1]

3.3 Balance Condition

The final condition, the balance condition, is the one that makes triangulating the quadtree leaves possible. By ensuring each quad is adjacent to another quad of no more than one level deeper, one can be certain to find an acute triangulation of each quad that places no new points on the boundary of the quad. If the balance condition is not enforced, a quad s may exist that is adjacent to arbitrarily deeper quads. The corners of each such adjacent deeper quad will then lie on the boundary of s, and so will need to be considered when triangulating s—and an arbitrary multiplicity of boundary points makes it quite difficult to generate an acute triangulation that does not change the boundary of the quad. By enforcing the balance condition it can be guaranteed that s will have to consider no more than a single such side point, and moreover that this side point, if it exists, will be precisely midway along the side. Such regular conditions do allow acute triangulations to be independently generated.

Thus, each quad can be triangulated individually and it is still certain that the triangulations within quads sharing a side will match up. Note that the condition has some caveats; if a quad lies entirely external to the domain then it will not need to be triangulated, and so the balance condition does not have to apply, saving some memory and effort. For reasons of correctness, though, the balance condition *cannot* apply to the two-edge case. To do so would create an infinite recursion (see figure 1), as the balance requirements force the quad containing the two-edge case to be recursively subdivided, the result of which will be a smaller but identically unbalanced situation.

It is also the balance condition that makes a breadth-first approach more viable than depth-first. In order to ensure no neighbour of a quad is adjacent to a neighbour more than one level deeper, a depth-first approach would require multiple traversals—each time a branch is built, all its neighbours must be checked to ensure the balance condition is not violated. Retaining the balance condition during a level-by-level construction is somewhat simpler.

Once the above properties are not satisfied by any of the leaves of the quadtree, the tree construction terminates, and a case-by-case triangulation of the leaves ensues.

[1]Easier in that deciding when to subdivide a quad is simpler—otherwise, all edges would have to be checked each time to verify no two non-two edge cases have intersecting domain sides, which is an expensive operation.

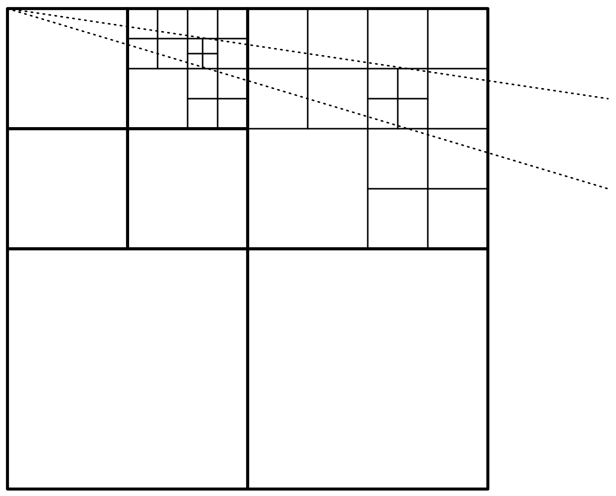

Figure 1: Enforcing the balance condition on the two-edge case causes infinite recursion.

4 Avoiding the Domain Exterior

Given the irregularity of the domains to which this algorithm is intended to apply, it is unlikely the domain will completely fill out the bounding square. In general, one can expect the bounding square to include some amount of domain exterior.

This can result in a significant waste of processing. No quads formed exterior to the domain are of interest, but without a method to distinguish interior from exterior, extra quads demanded by the balance condition cannot be pruned from construction. It is reasonably simple to eliminate these quads after the tree has been constructed by searching outside the domain from the vertices, but a more efficient method would be to prevent the construction of unneeded quads in the first place. Each time a leaf quad is subdivided into its four children, we would like to know which quads are interior or exterior, and in this way we can avoid processing quads that will never be used in the triangulation.

The following test can be used to determine which child quads of a parent quad are inside, outside or straddle the boundary of the region. We assume that every quad is provided with a list of (directed) input edges that lie within the quad boundary, and that the orientation of the domain with respect to the edges is known (that is, the domain is specified by either a clockwise or counterclockwise list of edges). Since this test is applied recursively, we can also assume that the parent quad has already been classified as interior, exterior or boundary.

We will need the following definitions:

Definition 2 *The anchor of a child quad Q in its parent quad P is defined to be the corner of the quad P contained in Q.*

Definition 3 *$\mathcal{B}(P)$ is the set of points forming the border of the quad P.*

Definition 4 *$\mathcal{S}_Q(P) \subseteq \mathcal{B}(P)$ is the set of points marking where an edge enters the region of P, but not the region of Q.*

Definition 5 *$\mathcal{E}_Q(P) \subseteq \mathcal{B}(P)$ is the set of points marking where an edge leaves the region of P, but not the region of Q.*

Given a subquad Q of P (i.e. P is the parent quad of Q), define the following order $\sqsubseteq_P^Q$ on $\mathcal{B}(P)$.

Definition 6 *$a \sqsubseteq_P^Q b \iff$ starting from the anchor of Q in P, and following the border of P in the clockwise direction, we encounter a before b.*

Definition 7 *Given a subset S of $\mathcal{B}(P)$, $Max_{\sqsubseteq_P^Q}(S)$ is the sup of S with respect to the $\sqsubseteq_P^Q$ order on $\mathcal{B}(P)$ restricted to S.*

Now, we present the actual test that allows us to determine whether a quad Q in P is internal or external, or contains some portion of the boundary. Obviously, if P is internal (resp. external), then all of its children quads will be internal (resp. external). So the only non-trivial case is when P contains the boundary. If Q also contains the boundary, then Q is neither internal nor external. If Q does not contain the boundary, we need the following fact and the following theorem.

Fact 1 *Given a point p not on a simple polygon C oriented clockwise, p is inside the region delimited by C iff $\exists$ a point b on C s.t. the directed tangent[2] of C at b (call it $\vec{bt}$) is "to the left" of the segment $\vec{bp}$ (*i.e., *$\vec{bp} \times \vec{bt} \geq 0$) and $\vec{bp}$ does not intersect C (except at b).*

Since C is a simple polygon, it's interior is connected—p is inside C iff a curve can be drawn from p to every other point inside C without intersecting the boundary of C. The clockwise orientation of C implies an interior point somewhere infinitesimally to the right of any tangent, and so p will be interior iff an uninterrupted line can be drawn from b to p keeping t to the left.

The proof of this fact is a direct translation of what it means for a point to be "inside" a region, and what it means for a curve to be oriented clockwise.

Theorem 1 *Given a simple polygon C oriented clockwise, and given Q a child quad of P. If P is a boundary quad (*i.e., *P contains the curve C), and Q is not a boundary quad, then Q is an internal quad iff $Max_{\sqsubseteq_P^Q}(\mathcal{S}_Q(P)) \sqsubseteq_P^Q Max_{\sqsubseteq_P^Q}(\mathcal{E}_Q(P))$.*

Proof: ($\Leftarrow$) We shall show that the anchor of Q in P is inside the curve C (thereby showing that the full quad Q is internal, since Q does not contain any piece of the boundary). First, with suitable rotation of the quad P, we can always place the child quad Q as the upper right child quad of P (the anchor of Q in P will then become the upper right corner of P). This rotation does not affect the internal/external quality of Q. Assume $Max_{\sqsubseteq_P^Q}(\mathcal{S}_Q(P)) \sqsubseteq_P^Q Max_{\sqsubseteq_P^Q}(\mathcal{E}_Q(P))$. Let e be $Max_{\sqsubseteq_P^Q}(\mathcal{E}_Q(P))$. Consider the following four cases:

1. e **is located on the upper boundary of the quad** P. Then e is a point on the boundary, with tangent "to the left" of the vector $\vec{ep}$, where p is the anchor of Q in P. The vector $\vec{ep}$ does not intersect the curve, by the maximality of e. Hence p is inside the region, and Q is internal.

[2]Note that tangents are not unique at vertices; any will do in this case.

2. e **is located on the left boundary of the quad** P. Then by the same argument as above, the upper left corner of P is inside the region. But since the curve does not intersect the upper boundary of P (by maximality of e), we have that the anchor of Q in P must be inside the region too, and Q is internal.

3. e **is located on the lower boundary of the quad** P. By the same argument, the lower left corner, the upper right corner and the anchor of Q in P are inside the region, and Q is internal.

4. e **is located on the right boundary of the quad** P. By the same argument, the lower right corner, the lower left corner, the upper left corner and the anchor of Q in P are inside the region, and Q is internal.

($\Rightarrow$) We shall prove the contrapositive. Again, rotate the quad P until the child quad Q becomes the upper right child quad of P. Assume $Max_{\sqsubseteq^Q_P}(\mathcal{E}_Q(\mathrm{P})) \sqsubseteq^Q_P Max_{\sqsubseteq^Q_P}(\mathcal{S}_Q(P))$. Now, reverse the orientation of the curve, so that the inside becomes the outside (and vice versa), and so that $Max_{\sqsubseteq^Q_P}(\mathcal{S}_Q(P))$ is a point where the curve leaves the quad C. By maximality of that point, we can apply the first part of the proof to get that the anchor of Q in P is inside the region, and hence that Q is internal **with respect to the curve with the reverse orientation.** Hence, Q is external with respect to the original curve. ∎

Note that if the boundary is specified counterclockwise, the same argument applies with clockwise and counterclockwise and left and right reversed.

5 Triangulating the Quadtree Leaves

It is not *a priori* clear that once the quadtree has been constructed according to the above criteria, every resulting leaf/square of the tree has a non-obtuse triangulation. Indeed, most of the subtlety of the algorithm is in the cases, and there are many of them. Below, all possible cases are illustrated, as well as arguments about each triangle's acuteness. The following two concepts will be required.

Definition 8 *If some triangle (a, b, c) is such that a vertex c lies on or between lines drawn perpendicular to ab from a and from b, and c also lies on or outside the circle with diameter $|ab|$ centered midway along ab, then (a, b, c) is not obtuse, and c is in* acute position *with respect to edge ab. The circle/disk defined by two points a and b will be specified as $\mathcal{D}\mathrm{isk}(a, b)$ (see figure 2).*

Note 1 *An obtuse triangle can always be decomposed into two right-angle triangles by drawing a line intersecting the obtuse vertex which is perpendicular to the opposing edge (see figure 3).*

Each one of our leaf-squares created by the above quadtree construction falls into one of the following mutually-exclusive and exhaustive categories:

1. Exterior; the quad is entirely outside the domain.

2. Interior; the quad is entirely inside the domain.

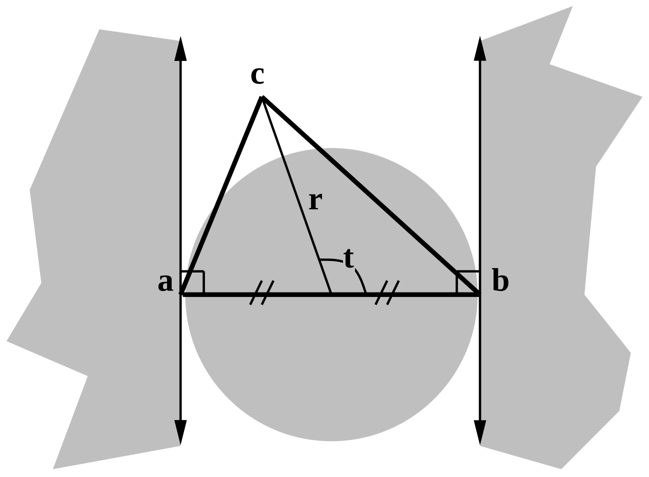

Figure 2: If c is not within the shaded region, (a, b, c) is acute.

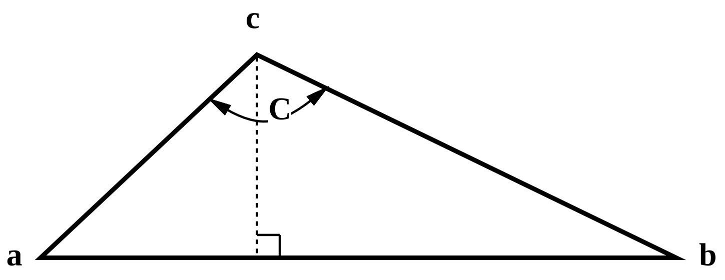

Figure 3: Splitting an obtuse triangle into 2 right-angle triangles.

3. Boundary; the quad contains some portion of the domain boundary.

The last case, boundary quads, includes a large number of subcases. It itself is then subdivided into cases based on the manner in which a quad can intersect the boundary:

1. One-Edge properly intersects the quad:
   (a) The edge intersects adjacent quad sides
   (b) The edge intersects opposing quad sides
2. Two-edges intersect the quad:
   (a) Domain side of edges do not intersect
   (b) The two-edge case

For each of the above cases, the balance condition forces the consideration of the possibility that each of the four quad leaf sides may or may not have a vertex at its midpoint. If a quad shares a side with another quad of depth one greater, then there will be a vertex midway along the same side corresponding to a corner of the smaller square. Each side that the square shares with another square that is at the same or higher depth will not have such a midpoint. Fortunately, this same property also ensures that there is no possibility of there being any other additional vertices on the sides of a quad. Thus, there are at most 16 possible configurations of midpoints for each subcase. As it will turn out, there are far fewer cases than this would indicate—symmetries, as well as not being concerned with the exterior domain allow for many combinations to be ignored.

Naturally, quads that are entirely exterior to the domain of interest do not need to be triangulated.

5.1 Interior Quads

When no input edges intersect, the task of triangulating the square is much simplified. Beyond the four corner vertices, there are only four possible other vertices (a midpoint on each side) to consider. Symmetries reduce the number of cases to a mere six (see figure 4). In each case, the triangles are trivially not obtuse.

5.2 Case 1a: An Input Edge Intersects Adjacent Sides

If an input edge enters from one side, s, of a square, and exits from one of the two sides that share a corner with s, it can be classified into one of four cases. The edge enters either above or below the midpoint on s, and exits similarly (though to keep things straight, instead of 'above' or 'below' the two halves of the exit side are called 'left' and 'right'). Of course there are symmetries; the edge can be assumed to enter from the left and exit on the bottom, and then any above-left configuration is a counter-clockwise rotation by $\pi/2$ of a below-right configuration. Thus, there are actually only 3 subcategories to consider: above-right, below-left and below-right.

A note about numbering within the diagrams: the three cases, below-left, below-right and above-right are indicated by the prefixes "BL," "BR" and "AR" respectively. This is followed by a number,

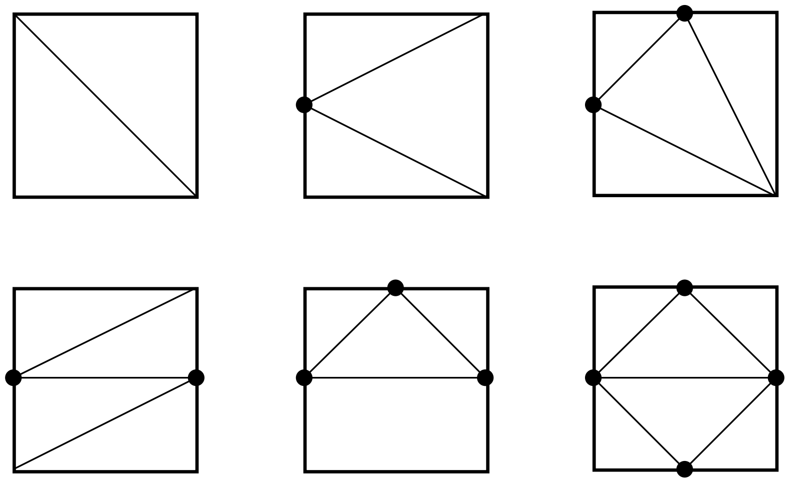
Figure 4: Possible triangulations of interior quads.

0 to 4, indicating the number of midpoints, then a decimal point and then a number indexing the possible combinations of midpoints. When the other "side" of the edge is being triangulated, the cases follow the same pattern, with the prefix being followed by a prime (*e.g.,* "BL′-2.1" instead of "BL-2.1").

Individual diagrams also have a consistent labelling scheme. Corners of the quad are labelled c_1 to c_4, going clockwise starting from the lower-left corner. The edge typically enters from the left at point e_1 and exits to the right at point e_2. Points added to the interior are labelled a, b, p or q, with the intention that a always lies at the center of the quad, b is usually the point corresponding to the third corner of a right-angle triangle (interior to the domain) made with e_1 and e_2, and p and q are individually placed. Midpoints along the sides of the quad are labelled m_1 to m_4, clockwise beginning with the left side (see figure 5). Finally, the horizontal line bisecting the quad is referred to as the *horizontal bisector,* and similarly the vertical line bisecting the quad is referred to as the *vertical bisector.* By *center* we mean the center of the quad.

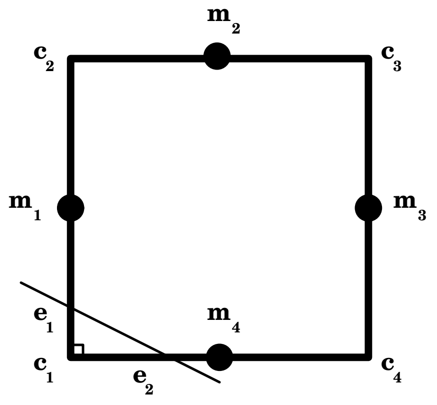

Figure 5: Labelling for individual quad diagrams.

5.2.1 Below Left Intersections

When the input edge intersects the square below the midpoint on the left side, and left of the midpoint on the bottom, there are either 16 cases or one case. If the domain of interest lies to one "side" of the directed input edge then there are 4 midpoints which may or may not be present, for a total of 16 possibilities. If the domain lies to the other "side," then there is only one trivial case (a below-left intersection cuts off a right-angle triangle corner, which can contain no midpoints). Here the 16 non-trivial cases are presented.

Note that for this case the input edge can be constrained such that $|c_1e_1| \leq |c_1e_2|$. If this is not true, the quad can be transformed by a vertical reflection followed by a counter-clockwise rotation by $\pi/2$, and then this constraint will hold.

The discussions that follow frequently make reference to the following simple results. Though sometimes intricate, none of these proofs require math more sophisticated than high school trigonometry.

Lemma 1 *Given a rectangle of size $w \times z$, let a be the upper-right corner, and let the origin, O, be the lower-left corner (see figure 6). If a line is drawn from a to some point e_2 on the bottom side, and a perpendicular to ae_2 is projected from e_2, intersecting the left side at e_1, then the vertical distance of e_1 from O is $y = x(w-x)/z$, and this value is maximal for $0 \leq x \leq w$ at $x = w/2$, whereupon $y = w^2/(4z)$.*
Proof: Note that $\theta = \angle e_2am$ and that $\theta = \angle e_1e_2O$. Thus, $\tan(\theta) = (w-x)/z$. We can compute the value of y then as $y = x\tan(\theta) = x(w-x)/z$.

To establish that this is maximal for $0 \leq x \leq w$, we simply take the derivative of the function for y with respect to x and note that y is 0 at $x = 0$ and $x = w$.■

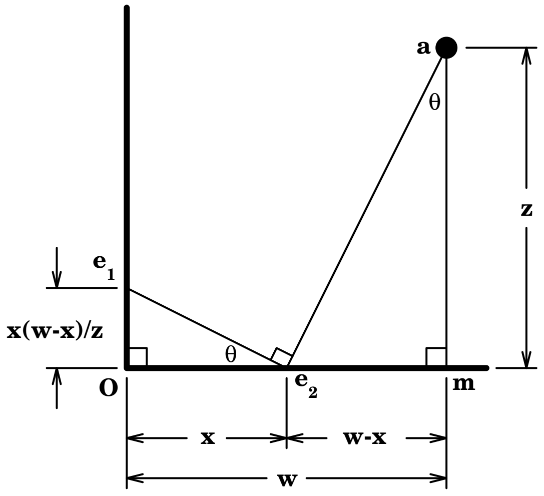

Figure 6: Height y is defined by $y = x(w-x)/z$

Corollary 1 *If $\angle ae_2e_1$ is not acute, then $y \leq x(w-x)/z$, and if angle $\angle ae_2e_1$ is not obtuse, $y \geq x(w-x)/z$.*
Proof: This follows trivially from lemma 1.

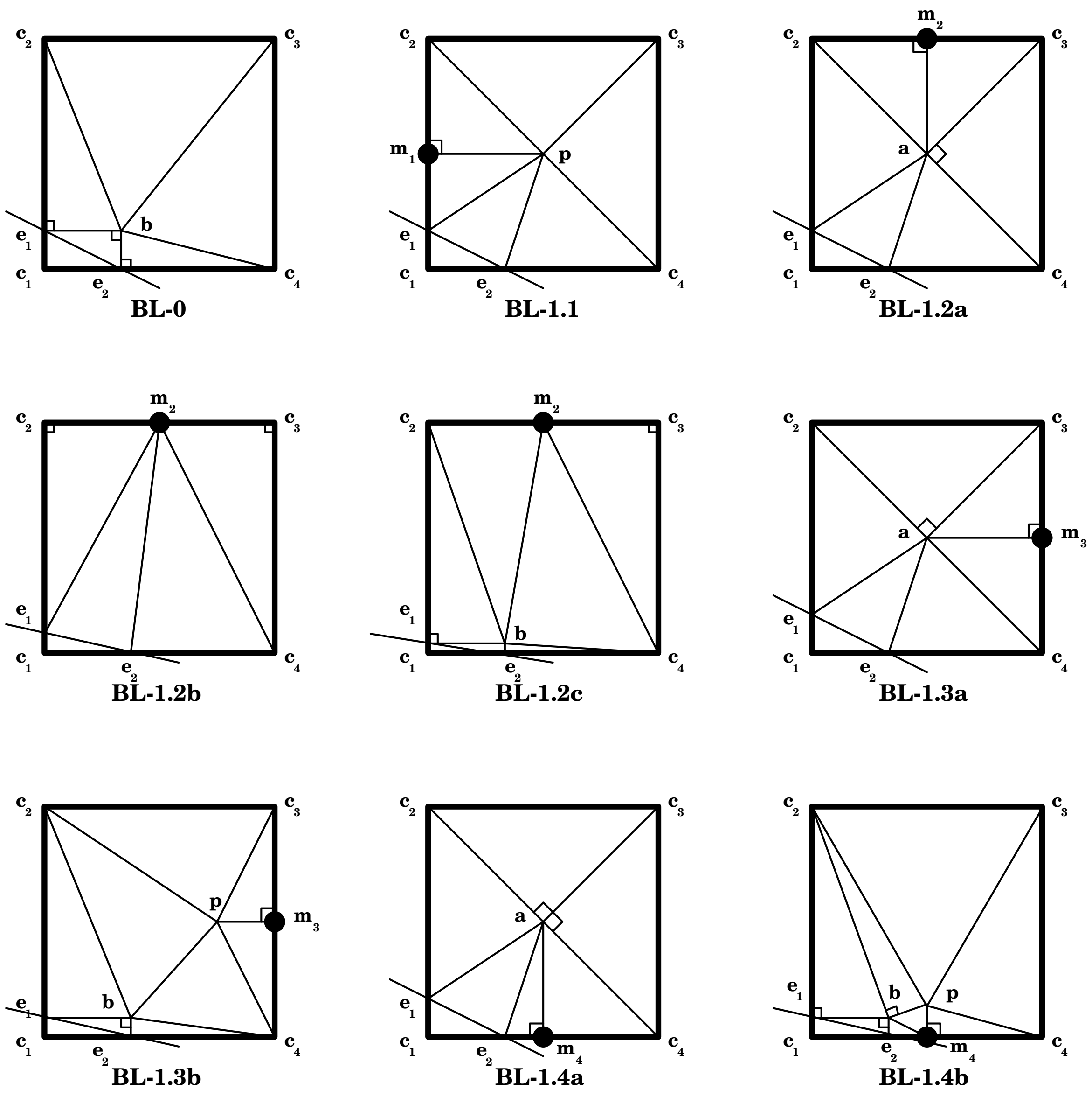

Figure 7: Subcases for adjacent below-left intersections with zero or one midpoint.

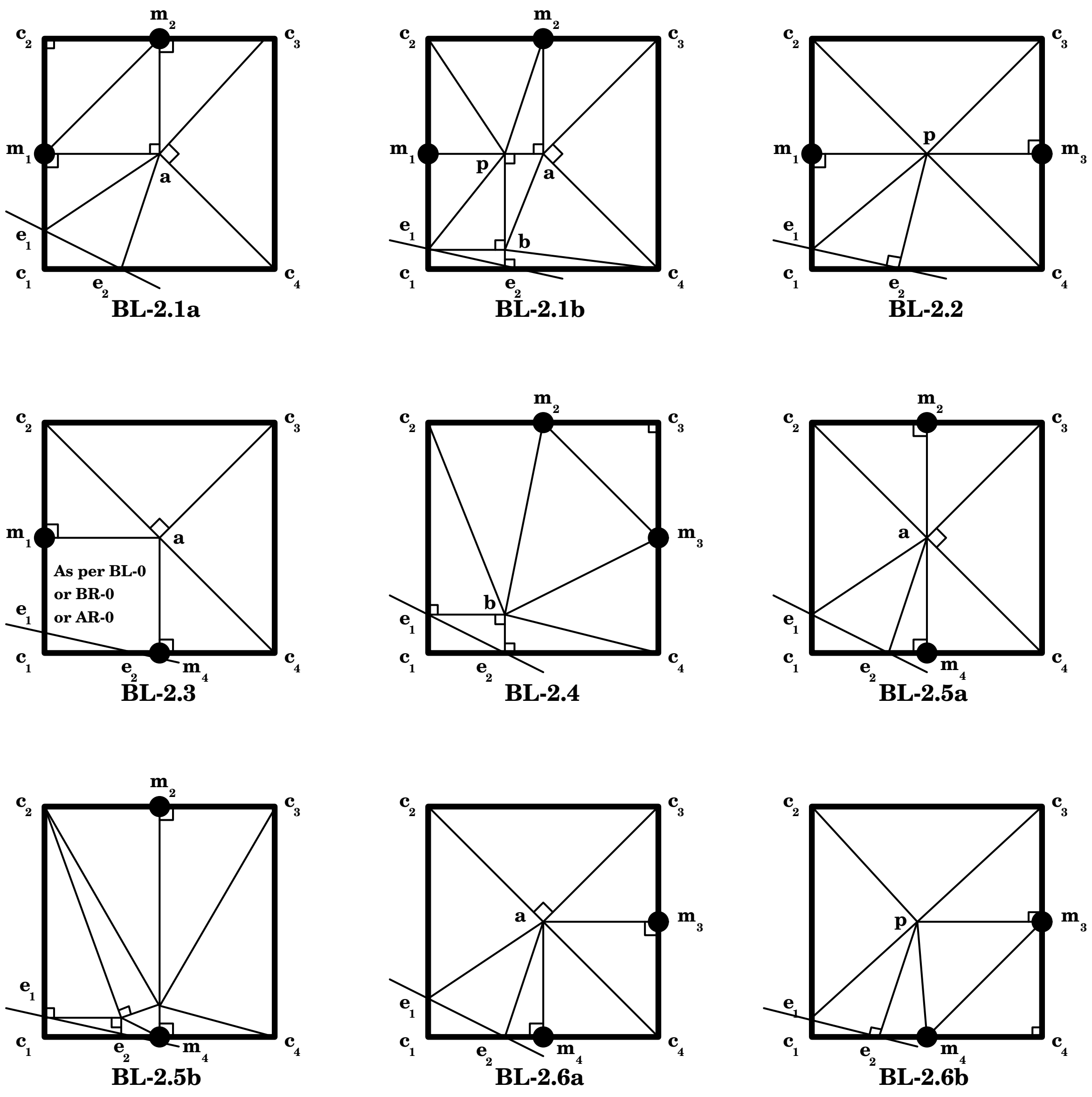

Figure 8: Subcases for adjacent below-left intersections with two midpoints.

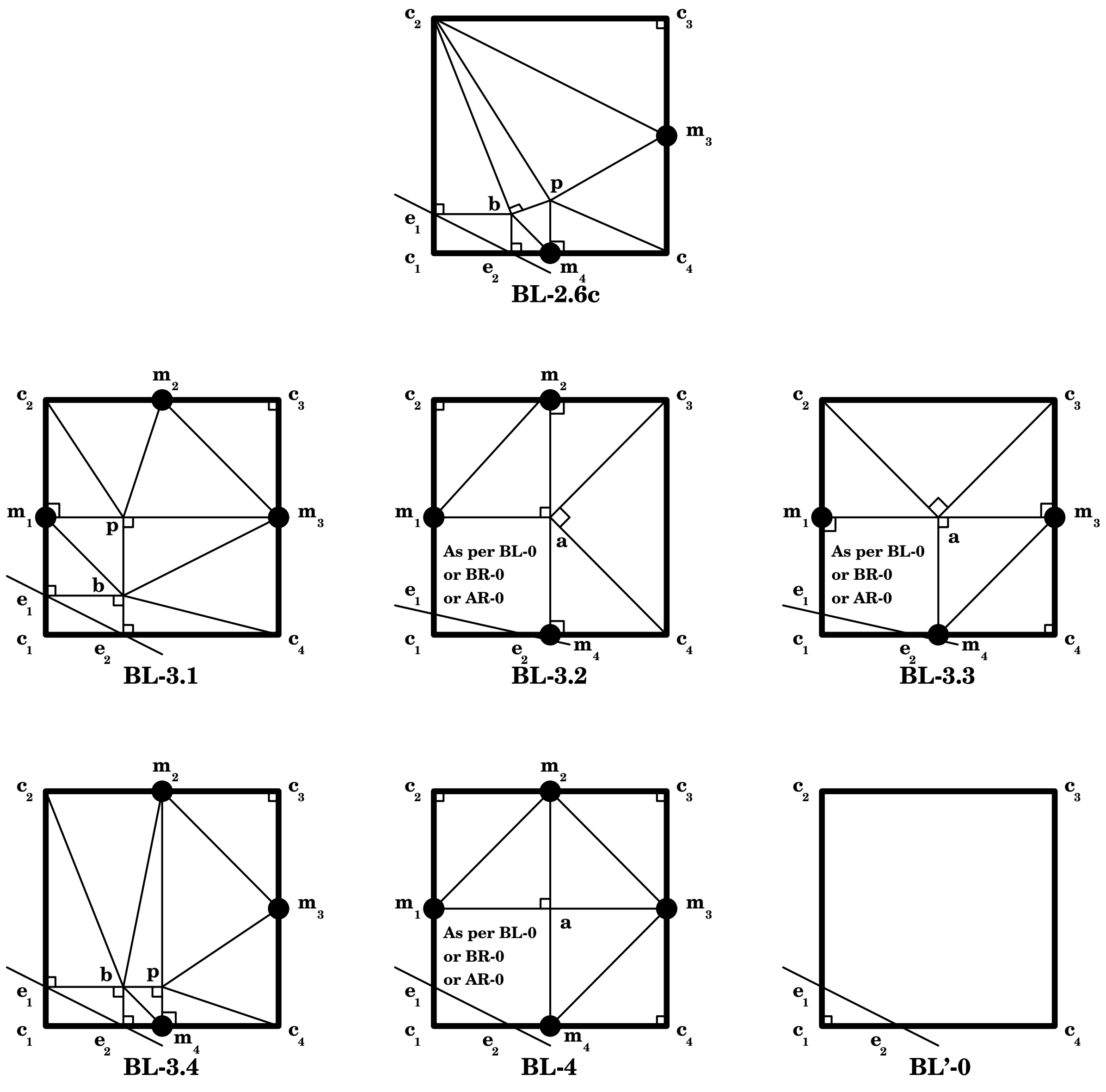

Figure 9: Remaining BL and all BL' subcases.

BL-0 See figure 7. Only two triangles are not right angle, (c_2, c_3, b) and (c_3, c_4, b). However, the construction of b forces b to lie in the lower left quarter of the quad. According to definition 2, then, b must be in acute position with respect to c_2c_3 and c_3c_4, and so both triangles must be acute.

BL-1.1 See figure 7. Point p is located as the point closest to the center along the horizontal bisector between m_1 and the center, such that $\angle e_1e_2p$ is no larger than $\pi/2$. Note that p will lie horizontally somewhere between e_2 and the center. Because $|e_1c_1|$ is no bigger than $|c_1e_2|$, a perpendicular to e_1e_2 extended from e_1 will always intersect left of the center and left of the intersection of a similar perpendicular extended from e_2, and so $\angle e_2e_1p$ will be acute as long as $|e_1e_2| > 0$. Since p is to the right of e_2, the angle $\angle e_1pe_2$ is contained in the $\pi/2$-angle formed between the horizontal bisector and a vertical passing through p, and hence is certainly not obtuse. The other triangles are simple to establish: because point p is on the horizontal bisector, p is necessarily in acute position to e_2c_4 and c_2c_3. Because p is left of the center, p is also in acute position to c_3c_4.

BL-1.2 See figure 7. If $\angle ae_2e_1$ is not obtuse, then this configuration can be triangulated as per subcase BL-1.2a. In such a case point a, being the center of the quad, is trivially in acute position relative to all of e_2c_4, c_3c_4 and e_1c_2. Angle $\angle e_2e_1a$ is acute for the same reasons as in BL-1.1, and angle $\angle e_1ae_2$ is acute because e_1 and e_2 lie in the lower-left quadrant.

Alternatively, if $\angle ae_2e_1$ is obtuse, then if $\angle m_2e_2e_1$ is acute and e_2 is horizontally at least 1/4 of the way along the bottom side, the quad can be triangulated as per BL-1.2b. Point m_2 is then guaranteed to be in acute position relative to e_2c_4. Because of the constraints on the vertical coordinate of e_1 imposed by corollary 1, the angle $\angle c_1e_1e_2$ varies between approximately 1.1 and $\pi/2$ radians within the allowed range of e_2. Angle $\angle m_2e_1c_2$ only varies between 0.46 and 0.52 radians, but in any case the two angles sum to more than $\pi/2$, leaving $\angle e_2e_1m_2$ less than $\pi/2$. Finally, because e_1 and e_2 lie in the lower-left quadrant, $\angle e_1m_2e_2$ is acute.

Finally, if $\angle m_2e_2e_1$ is also obtuse or e_2 is horizontally within the leftmost 1/4 of the bottom side, point b can be introduced and the quad triangulated as in BL-1.2c. Point b is then in acute position relative to edge m_2c_4: b is certainly within perpendiculars to m_2c_4 extended from m_2 and c_4, and (within its horizontal bounds) b necessarily lies below a tangent to $\mathcal{D}isk(m_2, c_4)$ at halfway along the bottom side ($\mathcal{D}isk(m_2, c_4)$ intersects the bottom side at the bottom right corner, and midway along the bottom side).

BL-1.3 See figure 7. If $\angle ae_2e_1$ is acute, the triangulation is similar to BL-1.2a, as shown in subcase BL-1.3a.

If $\angle ae_2e_1$ is obtuse, triangulation follows BL-1.3b. Here, point p is located along the horizontal bisector to the right of the center. Thus, p is in acute position relative to c_2c_3. In order to ensure that triangle (b, c_2, p) is acute, p is placed outside $\mathcal{D}isk(b, c_2)$, but inside the right angle formed by extending a perpendicular to bc_2 out from b. That this can always be done is established by the following lemma.

Lemma 2 *The perimeter of* $\mathcal{D}\mathrm{isk}(b, c_2)$ *intersects the horizontal bisector of the square (strictly) to the left of the right side.*
Proof: Because point b must be located in the region designated by corollary 1, the line segment c_2b reaches a maximum length in the degenerate case wherein b is coincident with the bottom midpoint. Thus, $|c_2b| \leq \sqrt{5}$ (assuming the square is 2×2, with origin at the lower-left corner). As well, the furthest to the right that the midpoint along c_2b can be located occurs in the same degenerate situation, where by similar triangles the center of the circle can be determined to be at coordinates $(1/2, 1)$. Hence, the intersection of the circle and the horizontal bisector of the square can be no further to the right than $\sqrt{5}/2 + 1/2$, which must be to the left of the right side of the 2×2 square.

Hence, by lemma 2 and the simple observation that a perpendicular to c_2b from b never intersects the horizontal bisector inside the square, it is certain p can be placed in acute position relative to both c_2c_3 and c_2b. The only triangle in doubt is (p, b, c_4). Corollary 1 implies that the height of b is bounded by $x(1 - x)$ (within our 2×2 square), and thereby b always lies below the diagonals c_2c_4 and c_1c_3. Thus, b must lie within perpendiculars to pc_4 from p and from c_4. Furthermore, $\mathcal{D}isk(p, c_4)$ does not intersect the curve $x(1 - x)$ within $0 < x < 1$, so b is in acute position to pc_4.

BL-1.4 See figure 7. Once again, if $\angle ae_2e_1$ is acute, the triangulation is similar to BL-1.2a, as shown in subcase BL-1.4a.

If $\angle ae_2e_1$ is obtuse, BL-1.4b is used instead. Point p here is located as the intersection of a perpendicular to c_2b at b and the vertical bisector; point p must be in acute position relative to c_3c_4. Since the location of b is constrained as per corollary 1, point p certainly can never be higher than the horizontal bisector, and so p is in acute position to c_2c_3. By construction, b must lie below p and above the bottom side, so angles $\angle pm_4b$ and $\angle bpm_4$ must be acute . As well, the placement of b implies that $\angle m_4bc_2$ must be no larger than π, and $\pi/2$ of that angle is "used up" by the right-angle $\angle pbc_2$. Hence $\angle m_4bp$ must be smaller than $\pi/2$.

BL-2.1 See figure 8. Again, if $\angle ae_2e_1$ is acute, the triangulation is similar to BL-1.2a, as shown in subcase BL-2.1a.

Otherwise subcase BL-2.1b applies; point p is located as the intersection of the horizontal bisector and a vertical extended up from b. Since e_2 is by definition left of the center, so will be p. The only triangle that is not a right-angle triangle is then (b, a, c_4). But since b is constrained as per corollary 1, b must be in acute position to ac_4: b must lie within perpendiculars to ac_4 extended out from a and c_4, and the center of $\mathcal{D}isk(a, c_4)$ maintains a distance of at least $1/\sqrt{2}$ (which is the same as its radius) away from any possible position of b.

BL-2.2 See figure 8. This situation is very similar to BL-1.1.

BL-2.3 See figure 8. This triangulation trivially follows.

BL-2.4 See figure 8. Since point b here is constrained to lie in the lower-left quadrant of the quad, b must be in acute position to m_3m_4, m_3c_4, and c_2m_3.

BL-2.5 See figure 8. Once more, if $\angle ae_2e_1$ is acute, the triangulation is similar to BL-1.2a, as shown in subcase BL-2.5a.

Alternatively, the triangulation is a trivial modification of the pattern shown in BL-1.4b, which is illustrated in BL-2.5b.

BL-2.6 See figure 8. Again, if $\angle e_1e_2a$ is acute, the triangulation is similar to BL-1.2a, as shown in subcase BL-2.6a.

Otherwise, assume a 2×2 square as in BL-2.6b; point e_1 is then constrained by corollary 1 to be such that if e_2 has x-coordinate x, then e_1 has y-coordinate no bigger than $x(1-x)$. In particular, if $0.41877 < x < 0.58123$, then $0.25 \geq y \geq 0.2434$. Because of this, $\mathcal{D}isk(c_2, e_1)$ can have radius no bigger than $1 - 0.1217$, and hence intersects the horizontal bisector strictly to the left of the point $(0.8783, 1)$. Let p be placed as the intersection of a perpendicular to e_1e_2 extended from e_2 and the horizontal bisector, and let v be the horizontal difference between p and the center: $v = 1 - p.x$. If we show that within the limited range allowed for x (and hence y) v must remain outside $\mathcal{D}isk(c_2, e_1)$, then triangle (e_1, c_2, p) is surely acute.

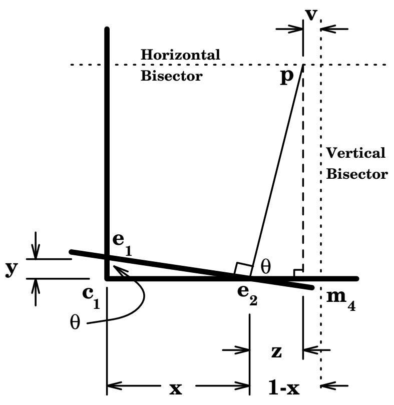

Figure 10: Subcase BL-2.6b detail.

Given x and y, we can calculate v as follows (see figure 10). Let θ be the angle $\angle c_1e_1e_2$; then $\tan(\theta) = x/y$. Let z be the length of the bottom side of a right-angled triangle formed from e_2, p and the intersection of a vertical through p and the bottom side, and note that $\theta = \angle(x+z, 0)e_2p$. From this we can conclude that $z = 1/\tan(\theta) = y/x$, and thus $v = 1 - x - z = 1 - x - y/x$. Distance v increases as y decreases, so for a y lower-bounded by 0.2343, v is maximal when $x = \sqrt{0.2343}$. Thus, v is maximal at about $v_{\max} \approx 0.013288$, which is well outside of $\mathcal{D}isk(c_2, e_1)$.

Also note that because p lies horizontally between e_2 and the center, p is in acute position to both c_2c_3 and e_2m_4, and m_4 is in acute position to pm_3.

Alternatively, if $y < 0.2343$ (which is certainly true if $x \leq 0.41877$ or $x \geq 0.58123$), then triangulation is as per BL-2.6c (see figure 9). Here p is located as the intersection of a perpendicular to c_2b

at b and the vertical bisector. Note that because of corollary 1, $\angle m_4bc_2 < \pi$, and so if $\angle pbc_2 = \pi/2$ then $\angle m_4bp < \pi/2$. Point b is therefore in acute position to pm_4. Point p will also be in acute position to m_3c_4 if p does not rise vertically above the horizontal bisector. However, in order for $\angle m_3pc_2$ to be acute p must satisfy the more stringent requirement that p lie outside $\mathcal{D}isk(c_2, m_3)$.

We can bound the height of p. Let $\theta = \angle e_1c_2b$, and let q be the point $(1, y)$ (the intersection of a horizontal passing through e_1 and the vertical bisector). Let z be the y-coordinate of p. Because $\theta = \angle qbp$ and $\tan(\theta) = x/(2-y)$, we can determine $z = y + (1-x)x/(2-y)$, and because z increases as y increases, we can bound the size of z by considering only a maximal y for a given x; that is, $y = x(1-x)$. By assumption, y is upper-bounded by 0.2343, and so z is upper-bounded by approximately 0.37. $\mathcal{D}isk(c_2, m_3)$ is fixed, and intersects the vertical bisector at y-coordinate $1.5 - \sqrt{5}/2 \approx 0.382$, hence p is outside $\mathcal{D}isk(c_2, m_3)$. Because p is of course lower-bounded by m_4, p must be in acute position to c_2m_3 and to m_3c_4.

BL-3.1 See figure 9. Point p is located as the intersection of a vertical extended up from e_2 and the horizontal bisector. Thus, point p is certain to be left of the center and in acute position relative to c_2m_2, and similarly m_2 will be in acute position to pm_3. Because b will be located left of the center, b will also be in acute position relative to m_3c_4. All the rest are right-angle.

BL-3.2 See figure 9. These results are trivial.

BL-3.3 See figure 9. These results are trivial.

BL-3.4 See figure 9. Here, point p is located as the intersection of a horizontal extended out from e_1 and the vertical bisector. Thus, p is in acute position to m_3c_4, and m_3 is in acute position relative to m_2p. Point b is constrained to lie in the lower left quadrant of the quad, and so point b is in acute position relative to c_2m_2.

BL-4 See figure 9. These results are trivial.

BL-0' See figure 9. These results are trivial.

5.2.2 Below Right Intersections

When the input edge enters below the midpoint on the left, and right of the bottom midpoint, the triangulation follows one of the patterns below. There are only 10 subcases in total here, though—when triangulated one side of the input edge there are only three midpoints that may or may not be present. When triangulating the other side, only a single midpoint could exist.

Note that within these cases, the point b within the quad diagrams has no particular significance (*i.e.*, it is not always the intersection of a vertical from e_2 and a horizontal from e_1).

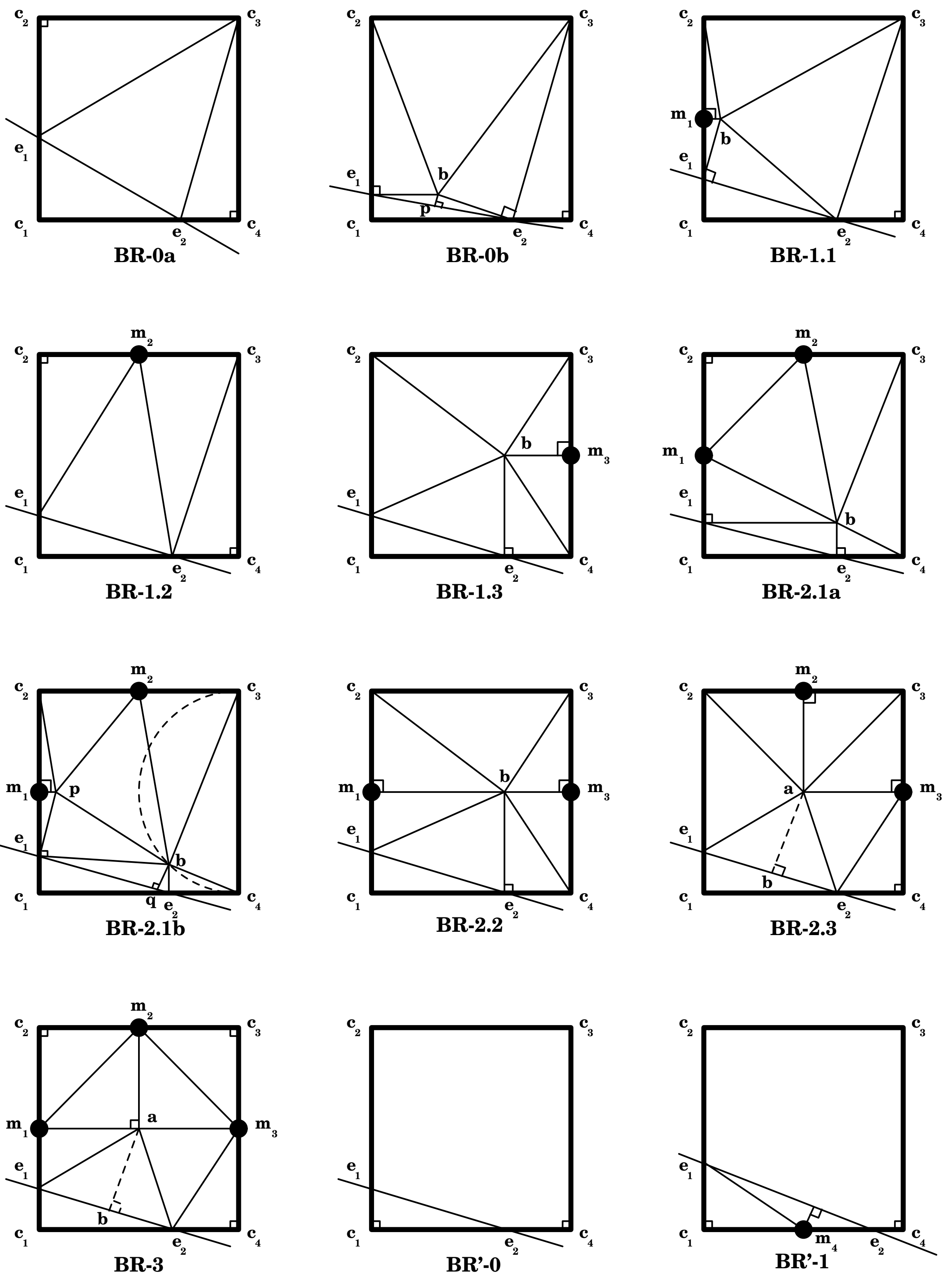

Figure 11: Subcases for adjacent below-right intersections.

BR-0 See figure 11. If $\angle c_3e_2e_1$ is acute, triangulation is performed as in BR-0a. Point e_1 must lie outside $\mathcal{D}isk(e_2, c_3)$, and so is in acute position to e_2c_3.

If this is not the case, point b is located as the intersection of a perpendicular to c_3e_2 extended from e_2 and a horizontal extended from e_1. Since e_1 is certainly below the horizontal bisector, b will be in acute position to c_2c_3. Point p is then located as per note 1.

BR-1.1 See figure 11. Point b is located as the intersection of a perpendicular to e_1e_2 extended out from e_1 and the horizontal bisector. Thus, point b will always be in acute position relative to edge c_2c_3. By lemma 1, $|m_1b|$ is maximal when e_1 is midway between c_1 and m_1; assuming a 2×2 square, $|m_1b| \le 1/(4x)$. Within the allowed range of x, this maximizes at $x = 1$, whereupon $|m_1b| = 1/4$. $\mathcal{D}isk(c_3, e_2)$ has radius no larger than $\sqrt{5}/2$, and its center has an x-coordinate no smaller than 1.5, so it intersects the horizontal bisector no closer to the left than about 0.382. Since $\angle e_2bc_3$ decreases as b moves to the left or e_2 moves to the right, and even when b is maximally to the right and e_2 maximally to the left $\angle e_2bc_3$ is acute, triangle (e_2, b, c_3) must be acute.

BR-1.2 See figure 11. Point e_2 is certainly in acute position to m_2c_3, and by the constraints on the positions of e_1 and e_2 (*i.e.,* being a below-right case) e_1 is in acute position to m_2e_2.

BR-1.3 See figure 11. Point b is located as the intersection of a vertical extended up from e_2, and the horizontal bisector. Since e_2 is certainly right of the vertical bisector and b is above e_1, b must be in acute position relative to e_1c_2, and since b is along the horizontal bisector, it is also in acute position to c_2c_3. Point e_1 must lie vertically between c_1 and m_3 so e_1 is in acute position to e_2b.

BR-2.1 See figure 11. Point b is located here as with the bottom left (BL) cases, as the intersection of a vertical extended from e_2 and a horizontal extended from e_1. If b lies outside $\mathcal{D}isk(c_3, c_4)$, then b is in acute position relative to c_3c_4, and triangulation is as per BR-2.1a. Because e_1 is never higher than the center, b is also in acute position relative to m_2c_3. Finally, because b must lie in the lower right quadrant, b must be in acute position to m_1m_2 as well.

If b would lie within $\mathcal{D}isk(c_3, c_4)$, then triangulation proceeds as in BR-2.1b. Let b lie on the intersection of a vertical up from e_2 and the perimeter of $\mathcal{D}isk(c_3, c_4)$, and position p as the intersection of a perpendicular to be_1 extended from e_1 and the horizontal bisector. Once again assuming a 2×2 square, we can express the equation describing $\mathcal{D}isk(c_3, c_4)$ as $y = 1 - \sqrt{-x^2 + 4x - 3}$.

Let x be the x-coordinate of e_2 and y be the y-coordinate of b. Let z be the y-coordinate of e_1, w be the x-coordinate of p, and let $\theta = \angle pe_1m_1$. Thus, it must also be that $\theta = \angle e_1b(0, y)$ and $\tan(\theta) = (z - y)/x$, and we can compute w from $w = (1 - z)\tan(\theta) = (1 - z)(z - y)/x$. In order to upper-bound w we note that if we assume a fixed x (and hence y), by lemma 1 w is maximized by placing e_1 precisely midway between y and m_1; $z = (y + 1)/2$. Thus, it must be that:

$$w \le \frac{(1 - \frac{y+1}{2})(\frac{y+1}{2} - y}{x} = \frac{(\frac{1-y}{2})^2}{x} = \frac{-x^2 + 4x - 3}{4x}$$

This function is maximal within our allowed range of x values when $x = \sqrt{3}$, at which point $w_{\max} = 1 - \sqrt{3}/2$.

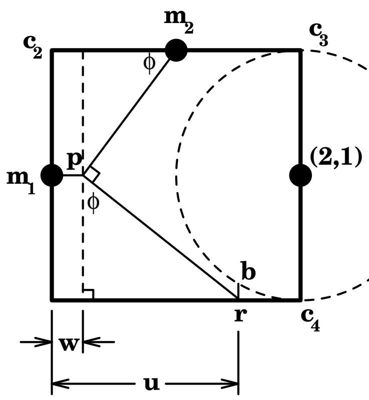

Figure 12: A necessarily acute angle in subcase BR-2.1b.

Now, consider a perpendicular to m_2p extended from p. It intersects the bottom side of the quad (the $y = 0$ line) at some point r (see figure 12). If we establish that pr never intersects $\mathcal{D}isk(c_3, c_4)$, then $\angle bpm_2$ must always be contained in $\angle rpm_2 = \pi/2$, and hence must be acute.

The position of r is dependent on p; as p moves to the right, r moves to the right. Thus, if pr does not intersect $\mathcal{D}isk(c_3, c_4)$ when w is maximal, neither does $p'r'$ for any $0 \leq p.x' < w$. We can calculate the x-coordinate of r by noting that if we let $\phi = \angle(w, 2)m_2p$, then $\tan(\phi) = 1/(1 - w)$, and $\phi = \angle(w, 0)pr$. Thus, if u is the x-coordinate of r, then $u = 1/(1 - w) + w$. As mentioned, u is maximal when w is maximal, so:

$$u_{\max} = \frac{1}{1 - w_{\max}} + w_{\max} = \frac{1}{1 - \sqrt{3}/2} + 1 - \sqrt{3}/2 = 1 + \frac{1}{2\sqrt{3}} \approx 1.288$$

The distance of the center of $\mathcal{D}isk(c_3, c_4)$ (which is $(2, 1)$) from the line pr can then be determined to be $\sqrt{7}/2$, which is certainly larger than the unit radius of $\mathcal{D}isk(c_3, c_4)$.

Point b can never be located outside of the lower-right quadrant of the square and p must lie along the horizontal bisector left of the center, and so angles $\angle pm_2b$ and $\angle m_2bp$ must both be acute. For the same reasons, p is in acute position to c_2m_2, b is in acute position to m_2c_3, and by construction b is in acute position to c_3c_4. The angle $\angle e_1be_2$ is obtuse by assumption, so once we locate point q as per note 1 the remaining triangles are all right-angled.

BR-2.2 See figure 11. This case is a trivial modification of BR-1.3.

BR-2.3 See figure 11. Point a, being the center of the quad, is trivially in acute position to e_2m_3 and e_1c_2. Angle $\angle ae_2e_1$ cannot be obtuse since e_2 is right of a (and hence $\angle c_4e_2a$ is obtuse), and $\angle e_2e_1a$ cannot be obtuse since both $\angle ae_1c_2$ and $\angle c_1e_1e_2$ must both be at least $\pi/2$. The final angle, $\angle e_1ae_2$ may or may not be obtuse—if it is, then a point b is introduced as per note 1.

BR-3 See figure 11. This case is virtually identical to BR-2.3.

BR'-0 See figure 11. This case is trivial.

BR'-1 See figure 11. This case is trivial.

5.2.3 Above Right Intersections

When the input edge enters above the midpoint on the left, and right of the bottom midpoint, there are only 8 possible subcases—each side of the input edge has two possible midpoints, each of which may or may not be present.

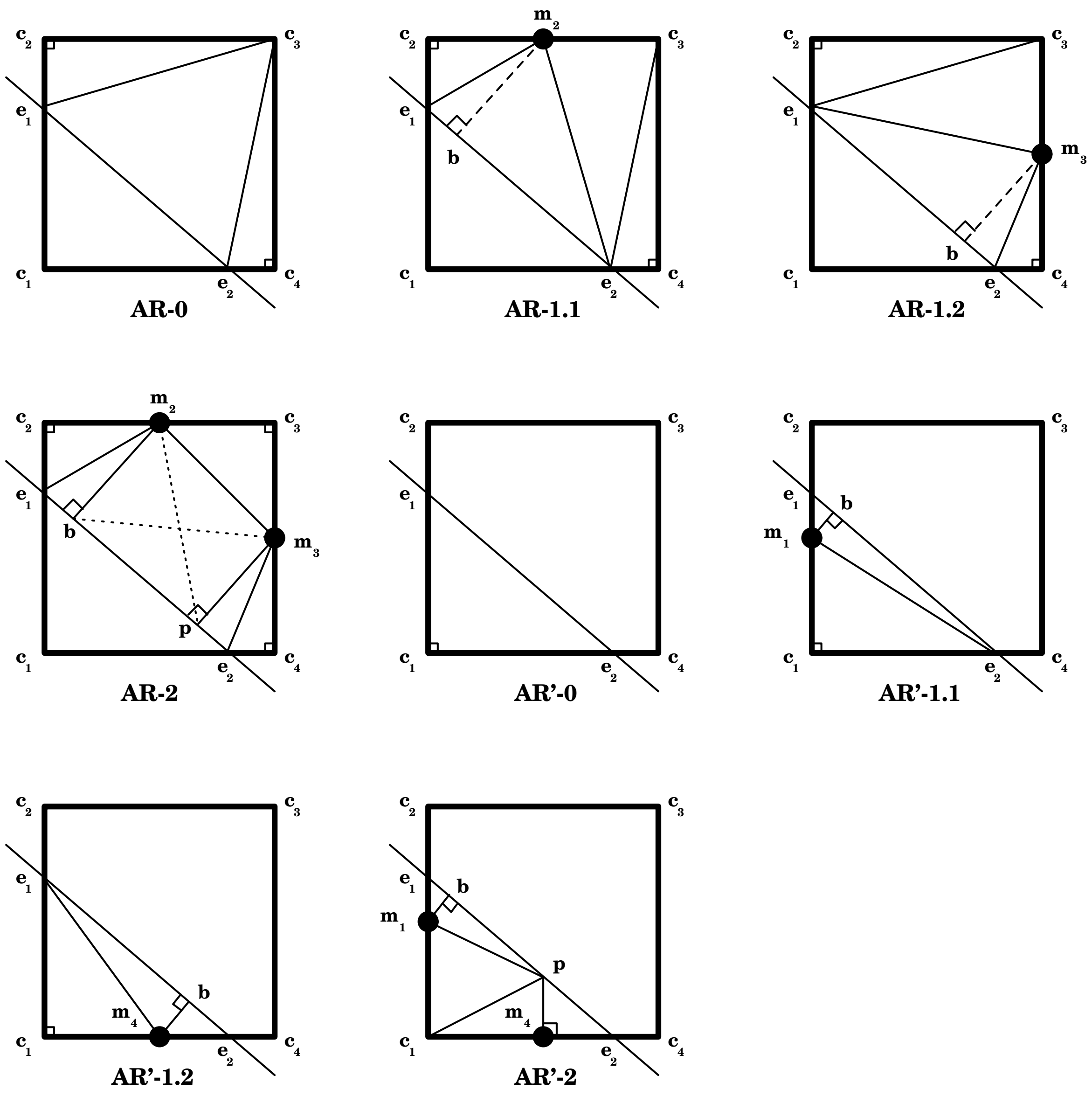

Figure 13: Subcases for adjacent above-right intersections.

AR-0 See figure 13. This case is trivial; the above right constraints on the input edge mean point c_3 must be in acute position to e_1e_2.

AR-1.1 See figure 13. Point e_2 must be in acute position relative to m_2c_3. Angles $\angle e_2e_1m_2$ and $\angle m_2e_2e_1$ must be acute by the constraints on e_1 and e_2. And if $\angle e_1m_2e_2$ is not acute, then point b is added according to note 1.

AR-1.2 See figure 13. This case is actually a vertical reflection followed by a counter-clockwise rotation of $\pi/2$ of case AR-1.1.

AR-2 See figure 13. Points b and p can always be located along e_1e_2 as the intersection of a line coincident with e_1e_2 and a perpendicular to e_1e_2 passing through m_2 and m_3 respectively. Because of the slope constraints on e_1e_2 implied by this being an above-right case, the projection of m_2m_3 onto e_1e_2 has both a non-zero minimal length and is forced to lie entirely along e_1e_2; in other words, b and p necessarily lie along the segment e_1e_2, and bm_2 and pm_3 do not cross. The result is a quadrilateral, (b, m_2, m_3, p) containing two $\pi/2$-angles; thus, at most one of the remaining two angles can be obtuse—and even so it cannot be too obtuse, since the slope of e_1e_2 is constrained. If the quadrilateral is then divided into two triangles so as to split the obtuse angle, two acute triangles must result.

AR'-0 See figure 13. This case is trivial.

AR'-1.1 See figure 13. Point b is placed according to note 1.

AR'-1.2 See figure 13. This is a symmetric variant of AR'-1.1.

AR'-2 See figure 13. Point p is located as the intersection of the vertical bisector and e_1e_2. Since the center of the quad can never be included in the domain below e_1e_2, p must lie vertically between the horizontal bisector and m_4—in other words, in acute position relative to m_1c_1. Point b is then positioned as per note 1.

5.3 Case 1b: Opposing Intersections

An edge entering from one side and exiting from the other can be assumed without loss of generality to enter from the left side and exit from the right. There are then four possibilities: either the edge enters above or below the midpoint on the left, and it exits similarly on the right.

However, symmetry reduces the four cases to two. An above-above situation is a vertical reflection of below-below, and above-below is a rotation of below-above. Hence, only below-above and below-below actually need to be considered.

5.3.1 Below Above Intersections

When the edge enters from below and exits above, both sides of the input edge can be triangulated identically—one side is simply a π-rotation of the other. Thus, there are a mere 4 subcases to deal with here, corresponding to the two possible midpoints.

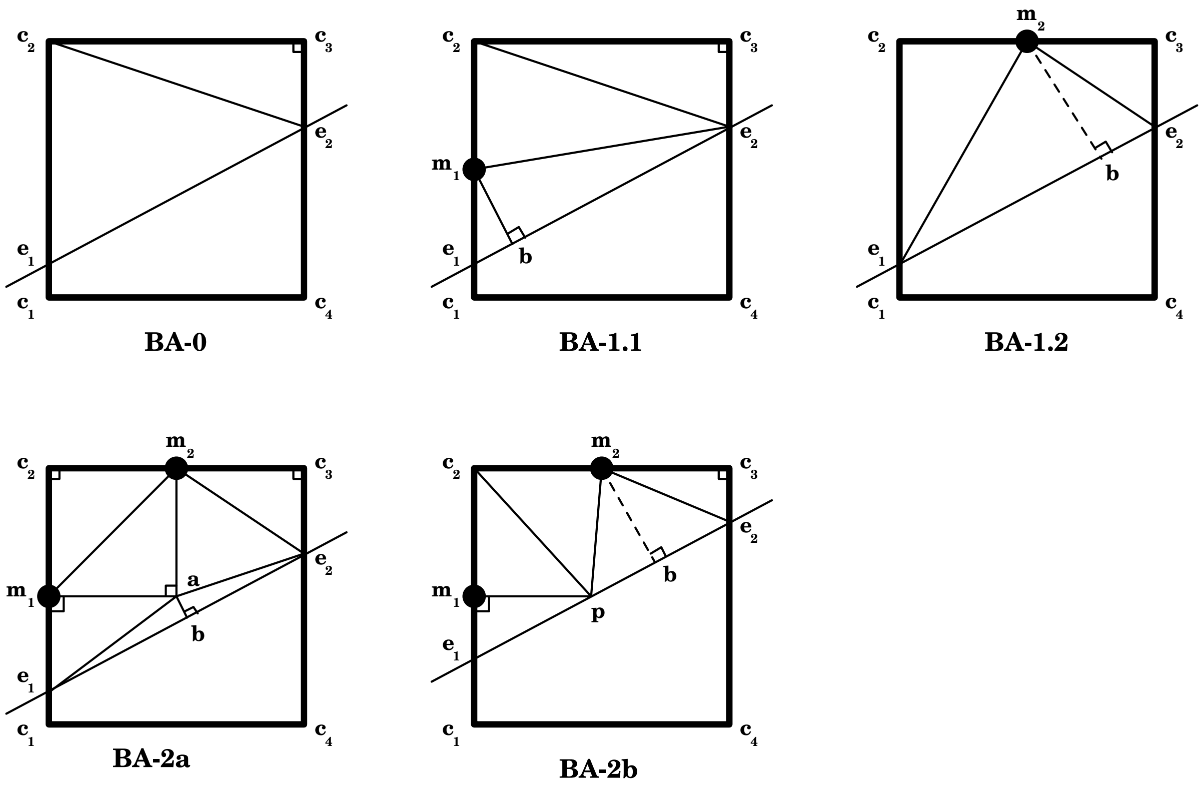

Figure 14: Subcases for opposing below-above intersections.

BA-0 See figure 14. This case is trivial; point e_2 must be located above the midpoint on the right, and so e_2 must be in acute position relative to c_2e_1.

BA-1.1 See figure 14. Again, point e_2 must be in acute position to c_2m_1. Point b is then located in accordance with note 1. Note that $\angle e_1m_1e_2$ must be obtuse, since e_2 is above m_1.

BA-1.2 See figure 14. Being a below-above case forces both $\angle e_2e_1m_2$ and $\angle m_2e_2e_1$ to be acute. The remaining angle, $\angle e_1m_2e_2$ may or may not be acute, but if it is not, point b is placed according to note 1.

BA-2 See figure 14. If the domain to be triangulated contains the center of the quad, then triangulation proceeds as in BA-2a. Point e_2 is in acute position to am_2, and point b is located to split the obtuse angle $\angle e_1ae_2$, as described in note 1.

If, however, the domain does not contain a, case BA-2b applies. Here, point p is located as the intersection of the horizontal bisector and e_1e_2. Since the center is not contained in the domain, p must lie to the left of point m_2, and so p is in acute position to c_2m_2. As with case BA-1.2, $\angle m_2e_2e_1$ cannot be obtuse, and since $\angle m_2pm_1$ is certainly obtuse, $\angle e_2pm_2$ is certainly not ($\angle m_1pe_1$, $\angle m_2pm_1$ and $\angle e_2pm_2$ must sum to π). Then, if $\angle pm_2e_2$ is obtuse, point b is situated as in note 1.

5.3.2 Below Below Intersections

When an edge enters below the midpoint and exits below as well, the edge may or may not have a positive slope. However, if the edge does not have a positive slope, a horizontal reflection converts it to a case that does. This results in 10 cases: 8 for the side of the input edge that may have up to three midpoints, and two for the side that can have but one midpoint.

BB-0 See figure 15. This case is trivial; since the slope of e_1e_2 is positive, e_2 must be in acute position to c_2e_1.

BB-1.1 See figure 15. Since point e_2 is bound to be below m_1, m_1 must be in acute position to c_3e_2, and for the same reason e_2 is in acute position to m_1e_1.

BB-1.2 See figure 15. Since this is a below-below case, e_1 and e_2 are below the horizontal bisector. Hence, m_2 is in acute position to e_1e_2.

BB-1.3 See figure 15. Point a, being the center, must be in acute position relative to c_2e_1. Since e_1 and e_2 must both lie below the center, $\angle e_1ae_2$ is certain to be at least $\pi/2$, which implies that $\angle e_1e_2a$ and $\angle ae_1e_2$ are both acute. By note 1, point b can be placed to split the obtuse angle.

BB-2.1 See figure 15. Because point e_2 must lie below the center, $\angle m_1m_2e_2$ cannot be larger than $\pi/2$, and because e_2 is along the right side, $\angle e_2m_1m_2$ must be acute as well, putting e_2 in acute position to m_1m_2. As with BB-1.1, e_2 is also in acute position relative to e_1m_1.

BB-2.2 See figure 15. Again, as with BB-1.1, point e_2 is in acute position to e_1m_1.

BB-2.3 See figure 15. This case is a trivial modification of BB-1.3.

BB-3 See figure 15. This case is a trivial modification of BB-2.2.

BB'-0 See figure 15. The slope of e_1e_2 is positive, and so e_1 must be in acute position to e_2c_4.

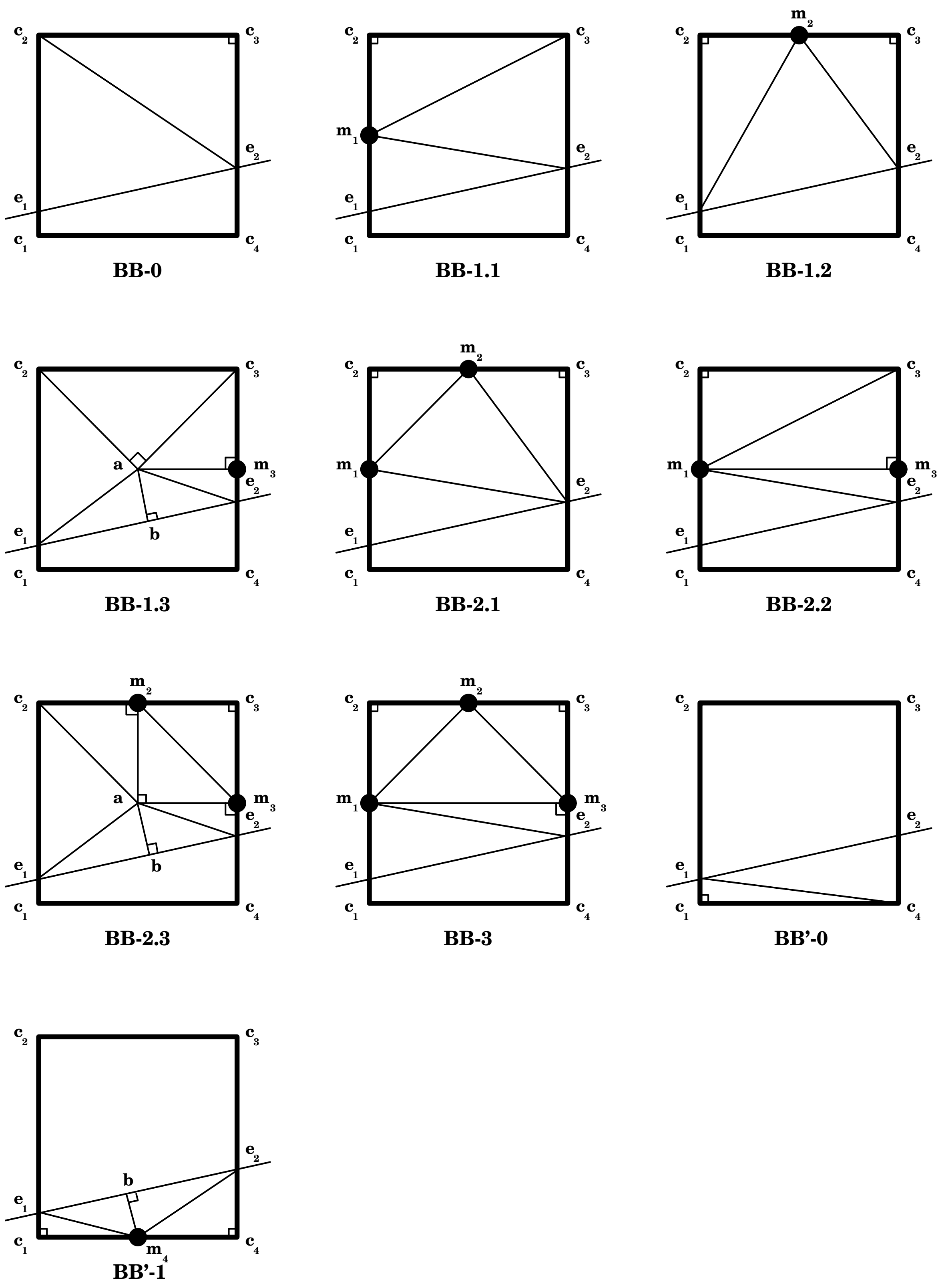

Figure 15: Subcases for opposing below-below intersections.

BB'-1 See figure 15. Since e_1 and e_2 are both bound to be below the horizontal bisector, $\angle e_2 m_4 e_1$ is necessarily obtuse. Thus, both $\angle m_4 e_1 e_2$ and $\angle e_1 e_2 m_4$ must be acute, and point b can be located as per note 1.

5.4 Case 2a: Two Input Edges With Non-Intersecting Domains

If two edges intersect the quad, and the domain sides of each edge do not intersect within the quad, then triangulation can proceed separately for each edge. Since only the domain sides will be triangulated, no conflict can result, and the independent triangulations of each edge can be (disjointly) merged.

5.5 Case 2b: Two Input Edges forming a Two-Edge Case

When two edges intersect a quad, their domain sides intersect within the quad, and they are not joined at a corner of the quad then the quad is recursively divided. This avoids considering many more cases, trying to triangulate the domain between two edges of more or less arbitrary orientation. Unfortunately, as can be seen in figure 1, recursively dividing such cases results in the quad actually containing the two edges joined at a corner having more than one "midpoint" along any or all of its sides.

Fortunately, a few observations and a simple scheme mentioned in Baker *et al*'s paper [2] make this problem tractable. Although each side of the quad may have an arbitrary number of points, since only the interior of the domain is being triangulated only some of these points will have to be considered. As well, the region being triangulated will have two unconstrained edges—any number of points can be added to the boundary of the domain (the two input edges), and still the triangulation of such a quad will not affect its neighbouring quads.

Such a situation can have two different configurations; either both input edges intersect the same side of the quad, or they intersect adjacent sides. Without loss of generality the vertex shared by the two input edges can be assumed to be the upper left corner of the quad, and then the input edges either both intersect the bottom side, the right side, or one intersects the bottom and one intersects the right. Once again symmetry reduces the cases—if both edges intersect the right side, then a simple rotation by $\pi/2$ transforms the case so both intersect the bottom side.

5.5.1 Two-Edge Case: Both on the Bottom Side

When both edges intersect the same bottom side of the quad, the resulting figure is an obtuse triangle, with the leftmost edge forming an obtuse angle with the bottom side (see figure 16). Such a configuration can be triangulated by adding lines parallel to c_2e_2 at each of the points along the bottom side (*e.g.,* line p_1b_1). Each such line intersects input edge c_2e_1, and at each such intersection point a line perpendicular to c_2e_2 is projected out to edge c_2e_2. This decomposes the original obtuse triangle into a collection of rectangles, which can be trivially triangulated, and right-angle triangles.

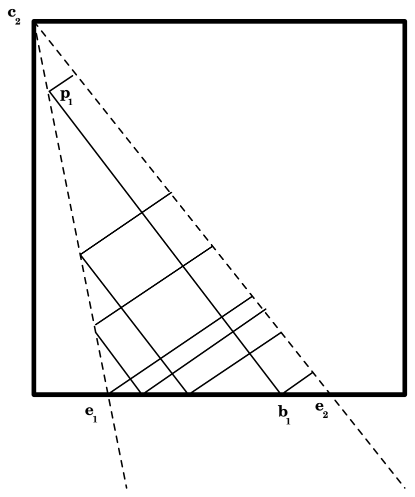

Figure 16: Triangulating the two-edge case when edges intersect the same side.

5.5.2 Two-Edge Case: One on the Bottom Side, One on the Right Side

If the input edges do not intersect the same side, the result can be partially triangulated as shown in figure 17. Horizontal lines are extended from each point along the right side until they intersect the leftmost edge, c_2e_1. At each such intersection point, and from each point along the bottom side, a vertical line is extended up to the highest horizontal. This decomposes some of the region into rectangles and right-angle triangles. The remaining region is similar enough to figure 16 to be triangulated in the same manner.

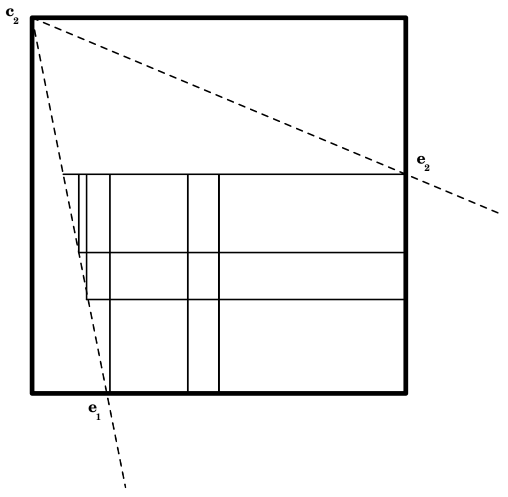

Figure 17: Triangulating the two-edge case when the edges intersect different sides.

6 Adaptivity

The above construction yields a complete triangulation of the domain computed from the leaves of the quadtree. This original structure constitutes a “base level,” a minimum depth for each branch of the quadtree, enforced in order to permit independent triangulations of each of the leaves. However, it is possible to grow the quadtree deeper than needed if greater resolution (*i.e.,* more grid points) is required in a given region.

A leaf which is “deepened” after reaching the base level of triangulation may cause other leaves to require deepening, in accordance with the criteria discussed above. Fortunately, as long as the quad to be deepened is not a two-edge case, the base level construction ensures that no quad deepened beyond the base level will need to be deepened for any other reason that the balance condition. Once a quad contains only a single edge, for example, recursively dividing it can never make that quad contain more edges, and once all input vertices lie at corners of quads, deepening the quads further cannot move them.

6.1 Adapting On a Budget

An efficient adaptive algorithm needs to have limited external effect when a given region is adapted. Within the quadtree context, this means deepening a quad beyond the base level should not propagate deepenings throughout the grid, or at least should limit the number. This algorithm, “dampens” the propagation of the balance condition, to the point where the following lemma can be stated.

Lemma 3 *If a non two-edge case leaf at depth d in the quadtree is deepened by one level (after the base level has been reached), then it will take at most $O(d)$ deepenings to restore the balance condition throughout the quadtree.*

Proof: The proof is inductive, relying on a recursive situation where only a single quad is “out of balance” with its neighbours. Note that it is not necessary to consider all possible arrangements of quads of varying depths—a quad at depth d just deepened to four quads of depth $d+1$ cannot propagate the imbalance to quads at a depth greater than $d-1$. In other words, the number of deepenings required to restore the balance condition will be maximized when quads neighbouring the unbalanced quad are at as shallow a depth as possible.

Since induction will be on the depth of the unbalanced quad, the base case, depth 0, is trivial—the root quad cannot be unbalanced with respect to its neighbours, since it does not have any.

Assume, then that there is a single quad, q at depth $d > 0$ that has just been deepened, and so is unbalanced with respect to its at least one of its neighbours. The discussion that follows demonstrates that after some constant number of deepenings, the quadtree will either again contain a single unbalanced quad with depth less than d, in which case the inductive argument holds, or will be completely balanced, or will contain two unbalanced quads of depth less than d. The latter situation will turn out to be closed—it either eventually produces a situation with just a single unbalanced quad, or it maintains exactly two.

Quad q must have a parent quad since $d > 0$. Call this parent quad p. Hence, q is an upper-left

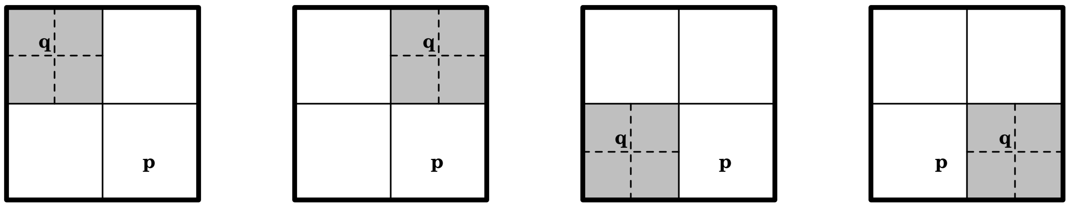

Figure 18: Possible positioning of q as a child.

child, an upper-right child, a lower-left child, or a lower-right child (see figure 18). Since these are all rotationally symmetrical, without loss of generality q can be assumed to be an upper-left child.

The neighbours of p, must all satisfy the balance condition, since q is assumed to be the only unbalanced quad. Coupled with the fact that p itself has been divided into q and three other children, it is necessary to conclude that p must be surrounded by quads of depth no less than p. Of course p must also be the child of some quad (or the same degenerate truth used in the base case will apply), in one of the four possible child positions. Two situations can then arise—either the deepening of q will affect (unbalance) one or two neighbours of p. All the legal quadtree decompositions for the first case are illustrated in figure 19[3] and for the second case in figure 20.

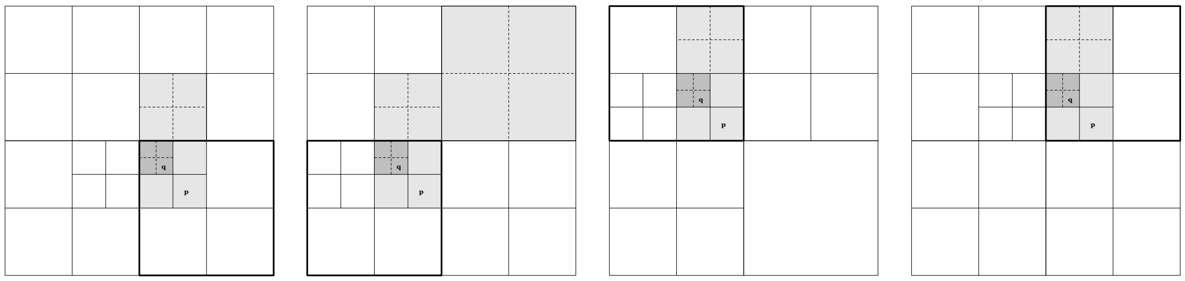

Figure 19: Legal quadtrees when p is a child and q unbalances only one neighbour.

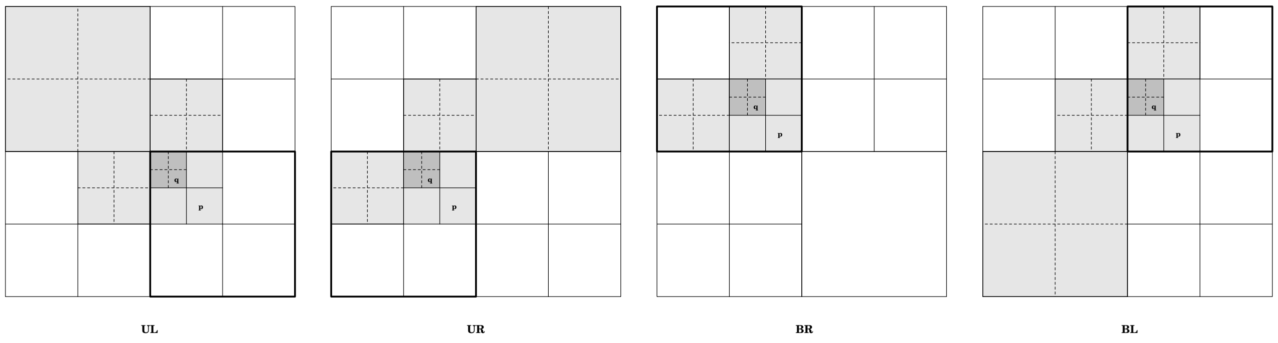

Figure 20: Legal quadtrees when p is a child and q unbalances two neighbours.

The first case in figure 19, when p is an upper-left child, is terminal—the quadtree is completely rebalanced. The inductive assumption applies to the other three cases, since they satisfy the initial

[3]The symmetric variation when the other quad adjacent to q is unbalanced is not shown.

requirement with a smaller depth—a single quad of depth $d-1$ or $d-2$ is left unbalanced in each situation, after causing either 1 or 2 quads to require deepening.

Unfortunately, it is not as simple when q unbalances two neighbouring quads. The first case in figure 20, again when p is assumed to be the upper-left child, results in a single unbalanced quad of depth $d-2$. The other three situations, when p is an upper-right, below-right and below-left child, require considering p's parent.

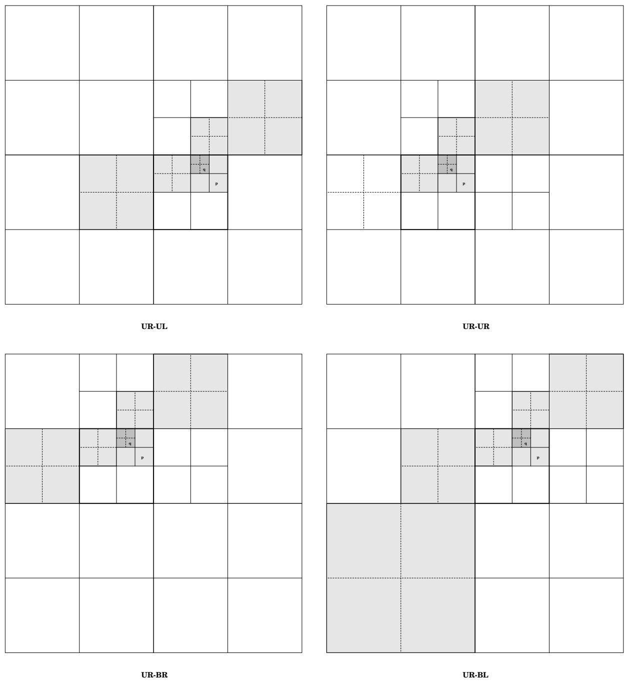

Figure 21: Further expansion of p as an upper-right case.

A further examination of the possibilities is illustrated in figure 21; p is assumed to be an upper-right child, and the four positions of p's parent are shown. The first and the second (UR-UL and UR-UR respectively) are eliminated by the inductive argument—a single unbalanced quad of depth $d-2$ is left after 4 further deepenings.

When p is assumed to be a below-right child, the four variations of parent it could have are shown in figure 22. Three of the cases, the first two and the last (BR-UL, BR-UR, BR-BL), result in a single unbalanced quad of either depth $d-2$ or $d-3$ after either 4 or 5 extra deepenings, and so the inductive argument can again be applied. The third case, (BR-BR) generates two quads at depth $d-2$.

The third possibility occurs when p is assumed to be a below-left child, and the four possibilities for its parent are shown in figure 23. Two of these (BL-UL and BL-BL) result in the familiar recursive situation, and two result in two unbalanced quads.

Thus, of all the possibilities, only five result in a non-terminal, non-recursive situation: UR-BR, UR-BL, BR-BR, BL-UR, and BL-BR. Two pairs of these, (UR-BL and BL-UR) and (UR-BR and BL-BR) can be transformed into each other by a rotation and a flip, and so only one of each pair

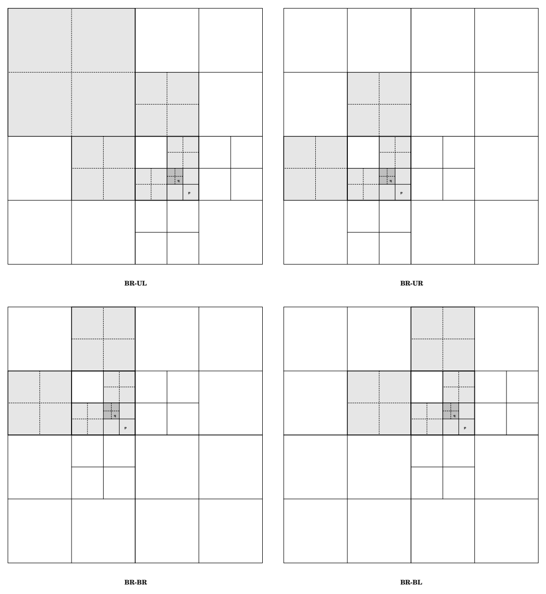

Figure 22: Further expansion of p as a below-right case.

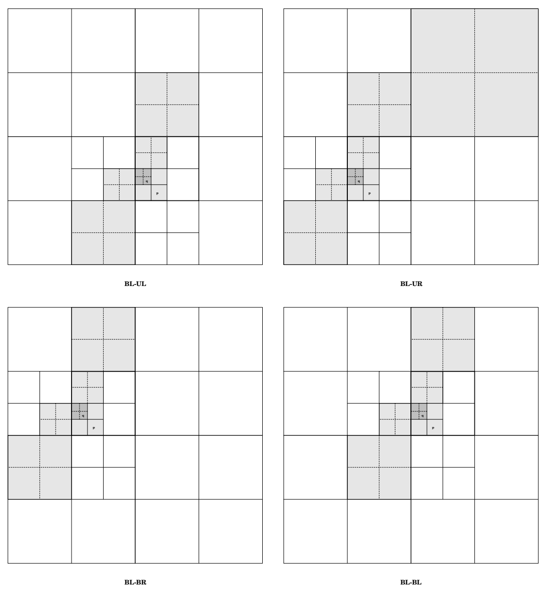

Figure 23: Further expansion of p as a below-left case.

(BL-UR and BL-BR) needs to be considered. The four further possibilities of (BL-UR) are shown in figure 24, and all of them result in a single unbalanced quad to which the inductive argument can be applied, or are terminal.

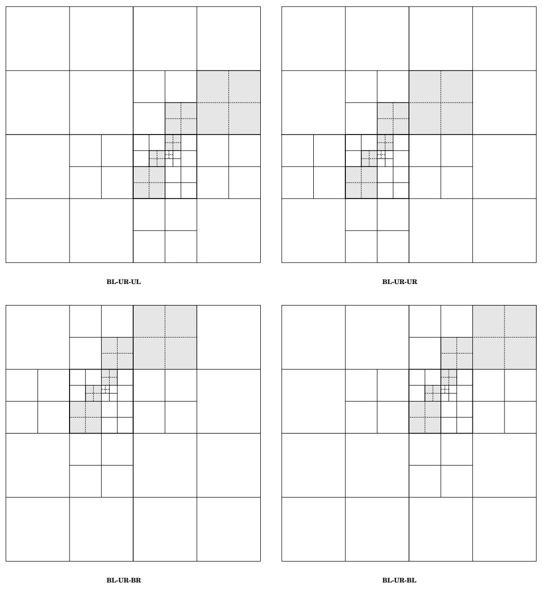

Figure 24: Further expansion of BL-UR.

This leaves only two possibilities, BR-BR and BL-BR, that might create more than a constant number of deepenings per level of depth. These two cases, however, can be characterized by the pattern shown in figure 25.

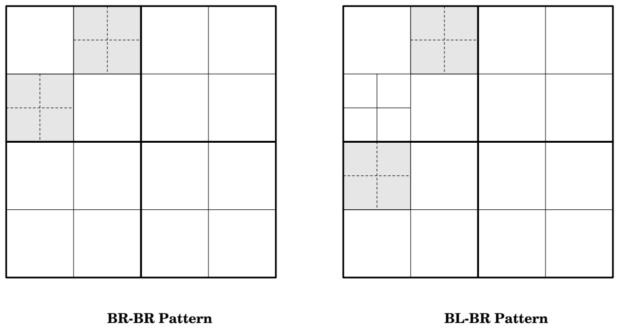

Figure 25: Patterns of growth when two quads are unbalanced.

In each case, four quads of depth d' containing two unbalanced quads of depth $d'+1$ are subdivided into quads of at least depth $d'+1$ around the perimeter of the depth $d'-1$ square (note that this last square may or may not form a quad). Internally, within the depth $d'-1$ square, all quads are balanced, and imbalance only occurs between the two indicated quads and quads external to the square. If these two patterns are then expanded one more level (figures 26 and 27), it becomes apparent that both patterns either repeat exactly at a smaller depth, or are reduced to a single unbalanced quad at a smaller depth.

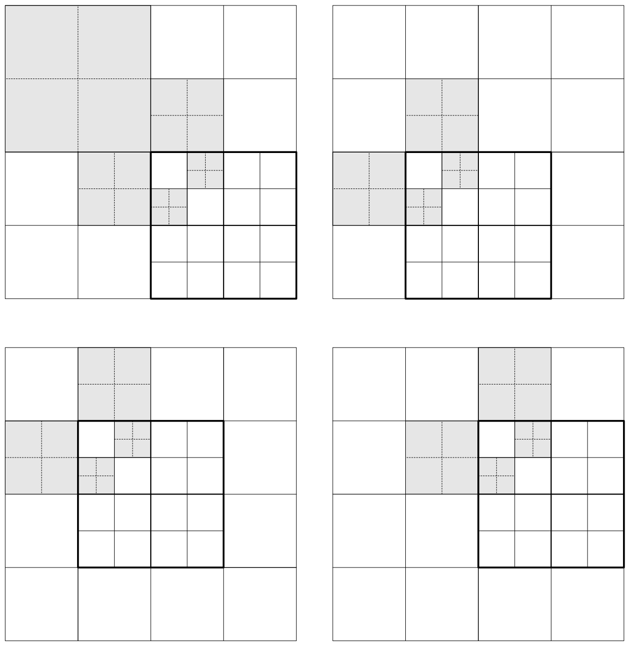
Figure 26: A schematic BR-BR, one step further.

Thus, the deepening of a single quad can produce at most 7 further deepenings before a single quad of a strictly lesser depth remains as the only unbalanced quad, or two unbalanced quads remain in the aforementioned pattern. Since the latter case either repeats or devolves into the former, the inductive argument holds.

Also note that the inclusion of arbitrary boundaries or two-edge cases does not change the maximum effects of propagation. Although deepening the latter can generate many more quads (depending on the angle), the balance condition does not apply to either of these types of quad, and so any unbalanced quads adjacent to an exterior or two-edge quad will not cause more deepenings in the direction of the exterior/two-edge quad. The maximum number of deepenings will always occur in the absence of these special quads. ∎

From this we can extract an "exponential dampening" theorem, demonstrating that adapting has a guaranteed small bound on non-local effects of a local refinement:

Theorem 2 *Let s be a non-two-edge case at depth d in the quadtree and let s' be a leaf quad at depth $d' < d$ affected by the deepening of s (d, d' both greater than or equal to the base level). Then s' is no more than $O(d - d')$ quads distant from s.*

Proof: From the proof for lemma 3, it is clear that at most a constant number of quads can require deepening per level in the chain of deepenings connecting s to s'. Thus, if the difference in depth between s' and s is $d - d'$, then the chain of affected quads between s and s' is of length at most $c(d - d')$ for some constant c. ∎

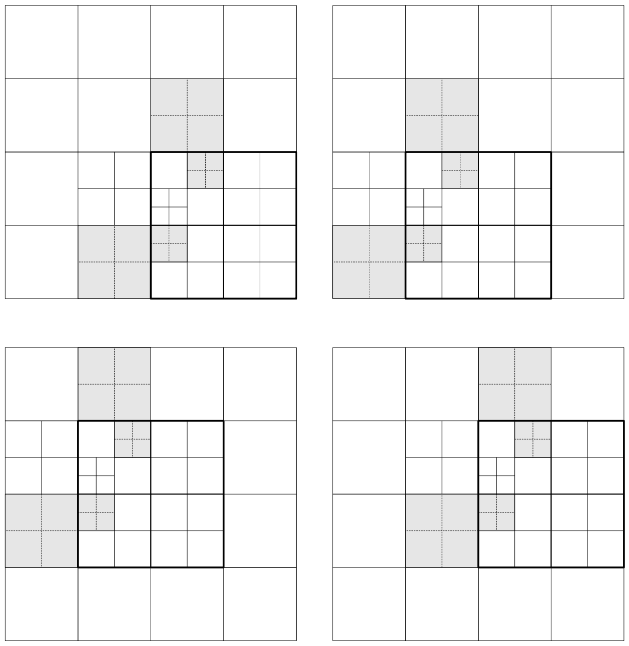

Figure 27: A schematic BL-BR, one step further.

6.2 Unadapting

Once a quad has been deepened beyond the base level, it can be subsequently “undeepened” to reduce resolution of the grid. Note that these operations are complementary, but not inversive—even if all four children of a given deepened leaf are subsequently undeepened, it may not restore the quadtree to its original condition.

In order to undeepen a quad deepened below the base level, the only condition that needs to be checked is the balance condition, which may propagate undeepenings in a manner similar to the deepenings. Since an undeepening merely reverses the effect of a deepening, and one can can never undeepen higher than the base level, undeepenings will have the same upper bound on cost as deepenings.

7 Runtime Analysis

The exact number of triangles/nodes generated by this algorithm is geometry-dependent. While deepening any single node/square can generate at most a depth-bounded number of further deepenings, the number of nodes that will have to be deepened to satisfy the vertex and edge conditions is not so easily determined. The vertex condition, for example, demands that the depth of a node is lower-bounded by the length of the binary expression of its input coordinates (with respect to the bounding square). The edge condition merely implies that there must exist a grid of sufficient resolution to ensure no two (non two-edge case) edges with a non-empty intersection of interior domains intersect the same grid square. This is sufficient for termination, but since the resolution required will depend on the distance between edges and their orientation with respect to the bounding square, it is not possible to give very tight *a priori* bounds. We can, however, give an upper bound, for which the following definition is needed:

Definition 9 *The interior angle between two edges, e_1 and e_2 connected at a vertex v* straddles *the x-axis (respectively, the y-axis) if there exists an infinite ray r, parallel to the x-axis (y-axis) starting at v with some continuous segment of r including v entirely interior to the domain.*

Definition 10 *The* smallest feature *of a polygon is the smallest distance between any two distinct vertices, or between any two non-intersecting lines.*

Note that the smallest feature is always defined for every polygon, and since it is defined only on non-intersecting objects, is always larger than 0.

Lemma 4 *Let b be the maximum length of the binary expression of any input vertex (with respect to the bounding square), f' be the smallest feature,*

$$f = \left\lfloor \log_2 \left(\frac{f'}{\sqrt{2}} \right) \right\rfloor$$

and let θ be the smallest angle interior to the domain that straddles neither the x nor the y-axis. If θ does not exist, or equivalently if $\theta \geq \pi/2$, then the smallest quad has sides of length no smaller than $\min(2^{-b}, 2^{f})$.

Proof: The node condition is satisfied by the 2^{-b}; every vertex must lie on the corner of a quad, since no vertex has binary expansion larger than b. This also helps with the edge condition with respect to intersecting edges. Since each vertex is required to lie on a quad corner, either the interior angle is larger than $\pi/2$, or the two edges straddle an axis—in either case, intersecting edges must lie in different quads. Once quads are small enough that no two non-intersecting features can lie within the same quad, the edge condition must be satisfied. The balance condition, of course, cannot cause a quad to be deeper than the maximum depth.■

Alternatively, if $\theta < \pi/2$, then some two-edge case will exist in the triangulated domain, and can result in much smaller quads being generated. Let e_1, e_2 be a pair of intersecting lines with interior angle θ, and let c be the corresponding cone. Let y_1, y_2 be the points of intersections of the arms of c and a line parallel to the y-axis at x-distance 1 away from v, and let x_1, x_2 be similarly defined for the x-axis. Since $\theta < \pi/2$ and the edges do not lie on nor straddle either axis, these intersections are uniquely defined in both cases.

Lemma 5 *Let d' be the smaller of the width of c at x_1, x_2, y_1 and at y_2, and let*

$$d = \left\lfloor \log_2 \left(\frac{d'}{\sqrt{2}} \right) \right\rfloor$$

Then the smallest square will have sides no smaller than $\min(2^{d-b}, 2^{d+f})$.

Proof: The node condition, and all edge conditions concerning non-intersecting features are satisfied according to theorem 4. However, even with such bounds it may happen that a two-edge case occurs in a quad of minimum size, in which case some of the surrounding quads may then contain two edges with intersecting domains. A minimum quad size of 2^d, though, is sufficient to ensure that the two edges exiting from a two-edge case do not lie within the same quad. Scaling this down to the minimum quad size for non-two-edge cases provides the given bounds. Note that two two-edge cases cannot interact; since the two cases do not intersect they are already separated by the minimum distance between features.■

8 Experimental Results

Expected behaviour of our algorithm is difficult to determine without some model of expected input. Nevertheless, our initial attempts at quite complex domains are very encouraging, with the number of nodes being essentially linear in the depth of the quadtree. Here we present the results from three very different domains: an unsymmetric wrench (figure 28), a hammer and sickle (figure 29), and an almost-unit square[4] (figure 30).

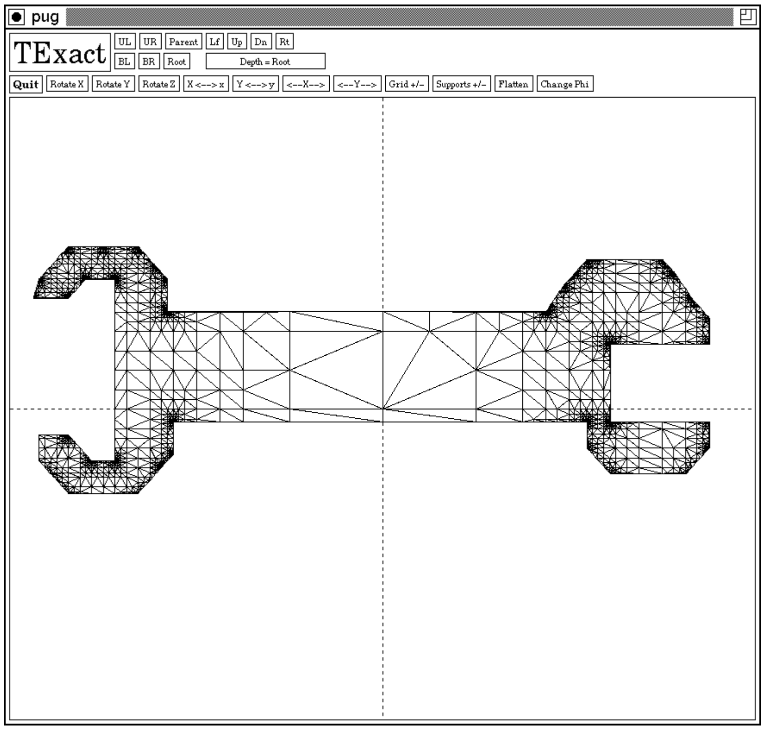

Figure 28: Triangulation of Wrench.

The wrench consists of 40 input nodes. The total number of nodes generated (and including the input) verses the maximum allowed binary expansion of input coordinates (*i.e.,* binary digits of input precision) is plotted in figure 31. For the sickle, consisting of 35 nodes, the results are in figure 32, and for the 4 node square in figure 33. Neither the wrench nor the square contain any 2-edge cases. The sickle contains just one 2-edge case (at the upper tip of the hammer).

[4]A true unit square would be adequately triangulated by just two triangles, and hence is uninteresting.

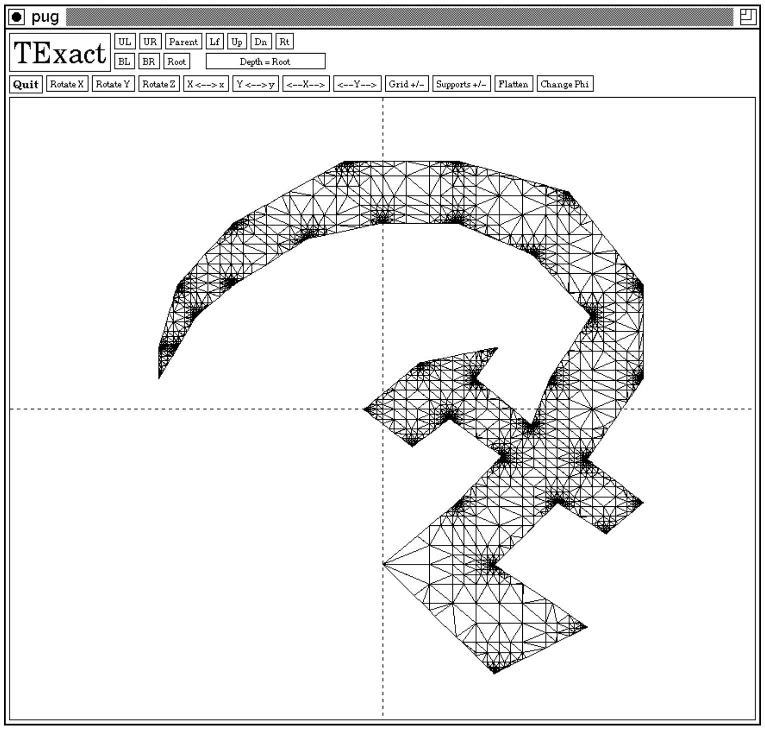

Figure 29: Triangulation of Hammer & Sickle.

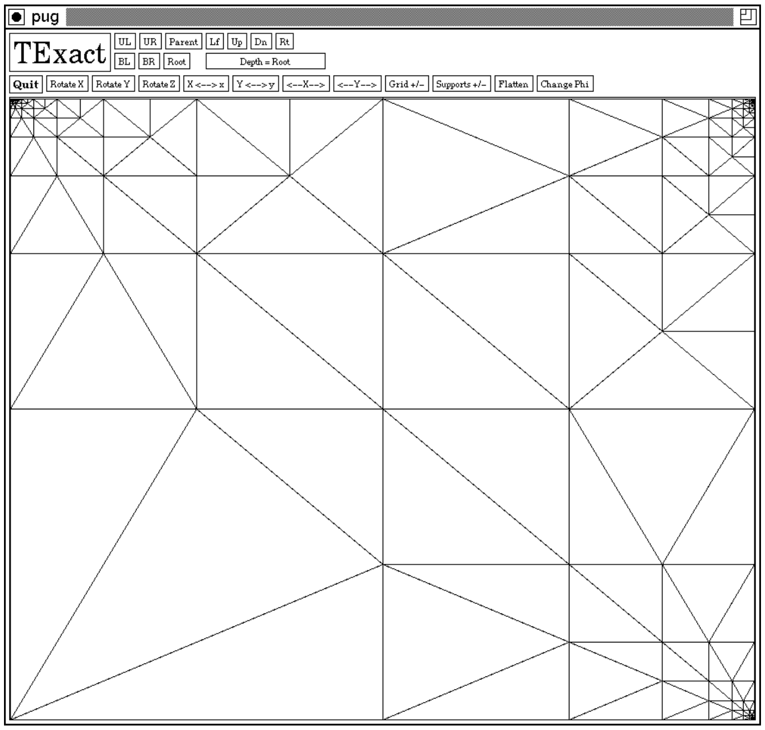

Figure 30: Triangulation of Square.

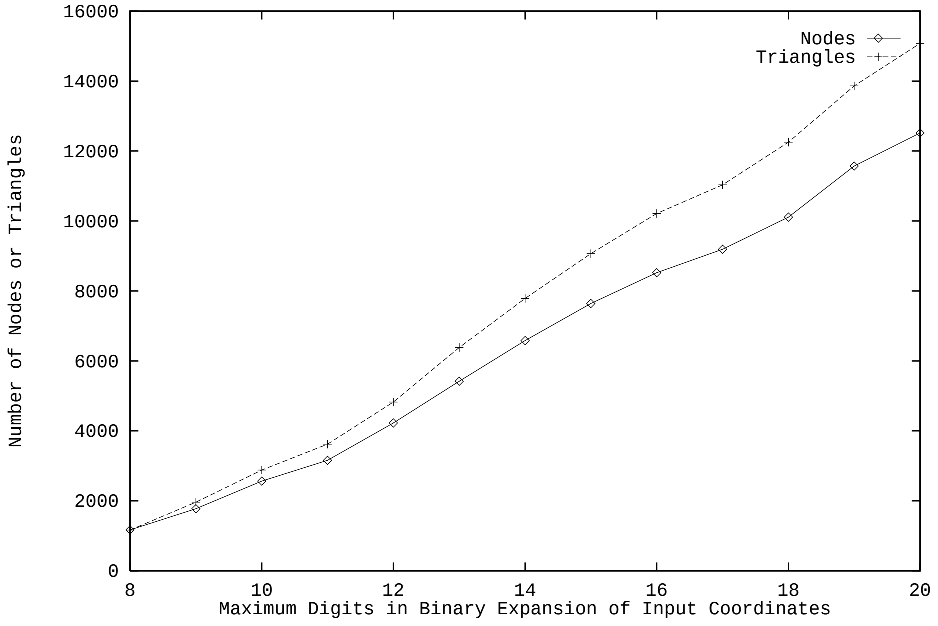

Figure 31: Vertices versus Precision for Wrench.

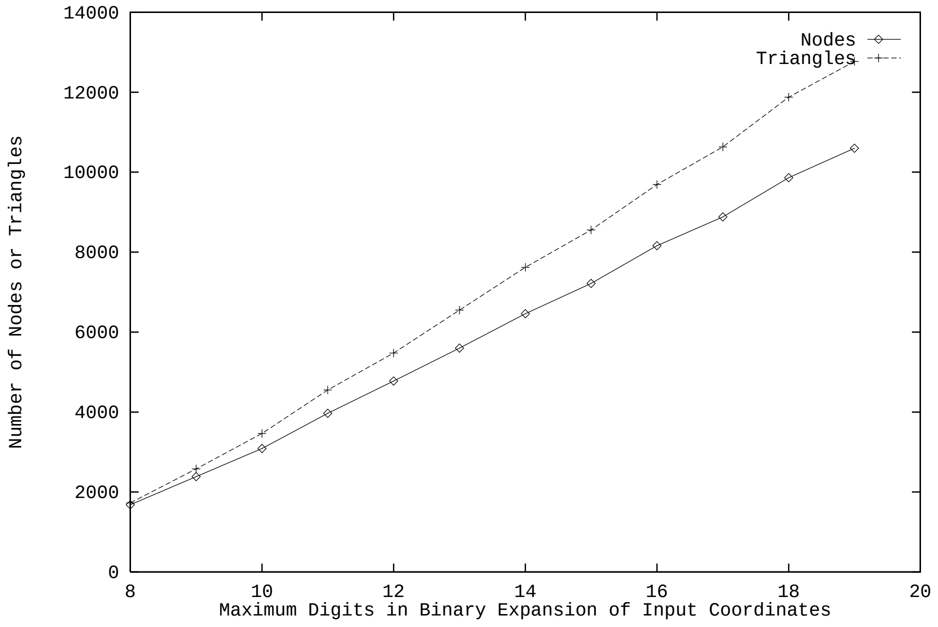

Figure 32: Vertices versus Precision for Hammer & Sickle.

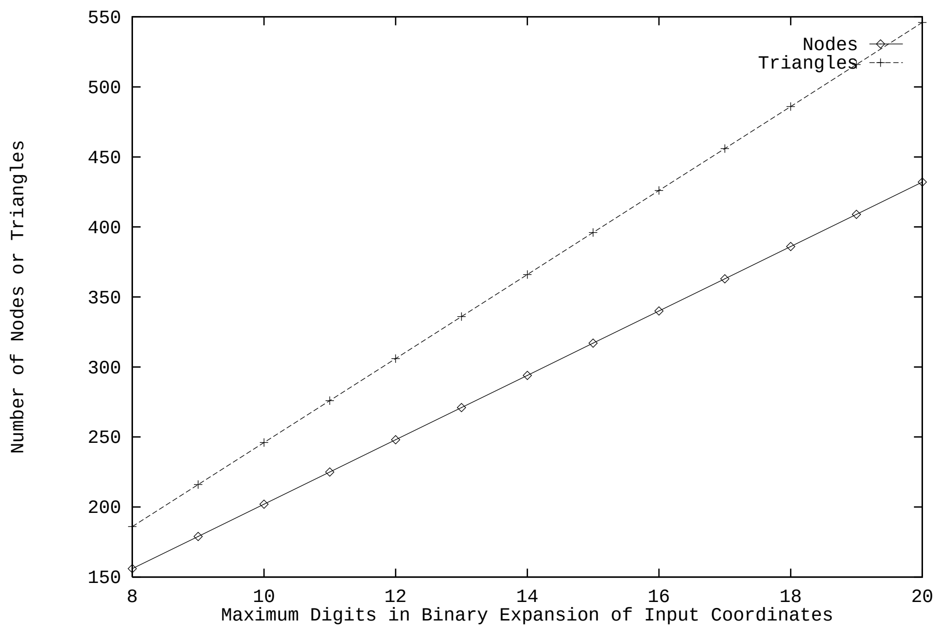

Figure 33: Vertices versus Precision for Square.

In each case, most or all of the coordinates have been chosen for their relatively long binary expansions, overconcentrating quadtree nodes (and hence triangles) around these points. Since there is usually some freedom in the specification of input vertices[5] it is straightforward to produce a much smaller and more balanced triangulation. Note that such coordinate selection does not change the domain—just the location along the domain of the selected vertices. The arch in figure 34, for instance, reaches its base level with a maximum quadtree depth of just 5 (*i.e.,* 5 digits of binary precision in input coordinates).

9 Related Work

Till recently the bulk of the work on generation of unstructured grids has been either heuristic, or has concentrated on generating grids satisfying the Delaunay criterion. A paper by Bern and Eppstein [6] provides an exhaustive survey of the field, particularly as it relates to the finite element method in fluid dynamics.

Techniques for generating bounded-size grids over arbitrary polygons (and without obtuse angles) have only begun to appear in the last few years; Baker, Grosse and Rafferty [2] being perhaps the very first to provide a provably correct algorithm. Their efforts were focussed on establishing the existence of an algorithm rather than on demonstrating its usefulness or feasibility in practice. They did not establish any bounds on the size of the triangulations that they generate nor did they

[5]Artificial domains are often selected with an eye to the perspicuity of their input coordinates, and actual domains are rarely so precise that some variation is not possible.

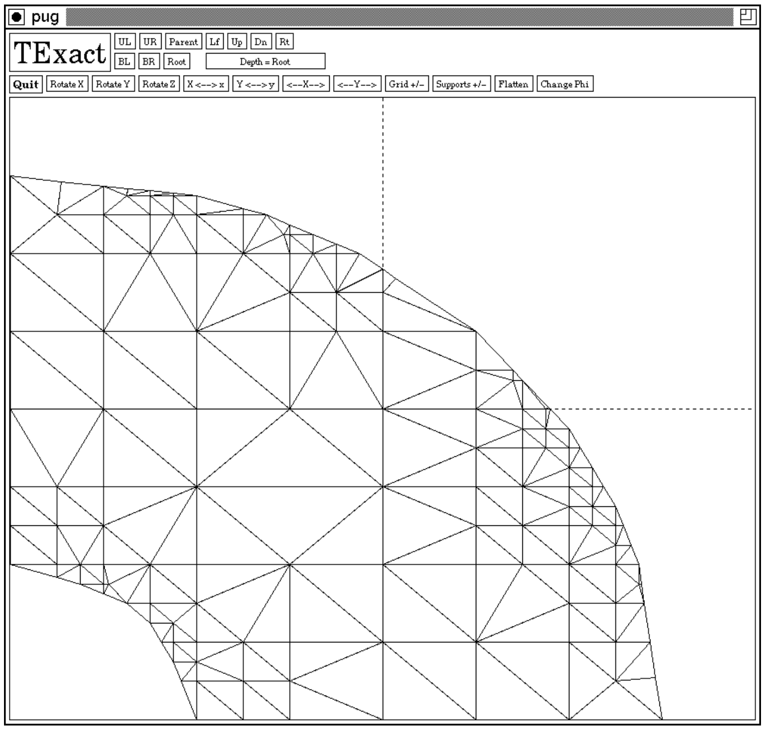

Figure 34: Triangulation of Arch.

implement their algorithm.

The algorithms that they develop are difficult to use because of the very high space requirements. The second algorithm that they give has the important advantage of avoiding small angles; something which our algorithm does not do, but it is extremely complicated and daunting to implement. Despite these criticisms this paper clearly opened up a new set of possibilities and demonstrated the existence of triangulations based on subtle ideas from modern computational geometry.

Further efforts have demonstrated $O(n)$ size[6] non-obtuse triangulations for *point sets,* but not so as to respect given polygonal boundaries [7],[8]. In 1991, Bern and Eppstein invented an algorithm to generate non-obtuse grids, but using $O(n^2)$ triangles. Indeed, until recently, the existence of $O(n)$ size non-obtuse triangulations of polygons was an open problem. This was finally resolved in 1994, when Bern, Mitchell and Ruppert [9] announced a divide-and-conquer style algorithm based on circle-packings that generates non-obtuse grids with a number of triangles linear in the input size.

The papers by Bern, Eppstein and Gilbert [7, 8] describe a family of related algorithms that cover a variety of cases. In three of them they work with given point sets rather than with given regions. This means that the boundary of the region is effectively the convex hull of the point set and is thus much more tractable than the regions we work with. Many applications in mechanical engineering require non-convex and even non-simply-connected regions. In one case they guarantee no small angles, but some of the angles could be obtuse, in the other two cases they avoid obtuse angles.

[6]For these problems, "n" is typically the number of input vertices.

The basic tradeoff is between the number of triangles and the minimum angle. The algorithm that does work for polygonal regions does not guarantee that there are no obtuse angles. They also give some higher dimensional algorithms with similar guarantees. In sum, while these are very interesting algorithms from the point of view of computational geometry they do not have all the requirements that one needs in practice.

In another paper by Bern and Eppstein [5] they develop an $O(n^2)$ triangulation (where n is the number of sides) which is guaranteed to contain no obtuse angles. The regions could have holes and certainly need not be convex so this is general enough for most applications. As usual this was not implemented and the algorithm appears quite tricky to implement in an efficient way because of the rather subtle decisions that are made. The main difficulty that we have with this was that it is not clear how one could modify this algorithm to make it adaptive.

Very recently Bern, Mitchell and Ruppert [9] announced a linear-size nonobtuse triangulation of polygons. The algorithm uses a very ingenious circle-packing scheme and is recursive in character. They do have an implementation which however leaves out one of the key steps: instead of computing the generalized Voronoi diagram they use heuristics for circle placement. While recursive, it is not clear to us how feasible this algorithm would be for an adaptive grid—the effects of replacing a given circle with some number of smaller ones does not have an easily determined effect. They do discuss parallelization, but only in the context of purely theoretical models; for instance, they show that their algorithm is in NC, using n^2 processors. Clearly this is not relevant to the feasibility of coarse-grained parallelization on realistic machines.

Several authors have also investigated the use of quadtrees in mesh generations, both heuristically (Yerry and Shephard [17]), and deterministically (Bern, Eppstein and Gilbert [8]). The former use quadtrees to essentially tile the domain with some (fixed) number of patterns, which can then be triangulated, while the latter extend an (other) algorithm presented in [2] to triangulate point sets with no obtuse angles.

10 Conclusions

The main contribution of the present paper is the development of an algorithm that efficiently constructs two-dimensional triangular grids that conform to polygonal boundaries, not necessarily simply connected, with the guarantee of nonobtuseness of the triangulation. Furthermore the quadtree structure permits one to incrementally refine or coarsen the grid while ensuring that the effects "fall off exponentially" as one goes away from the region of interest. This is important for minimizing the communication costs in parallel implementations, especially those intended for distributed-memory machines. We have implemented out algorithm and tested it experimentally on realistic examples.

The main advantage of this algorithm is its ability to adapt with reasonable locality, and its relative simplicity. It requires data structures no more sophisticated than quadtrees and lists, and algorithms no more complex than tree traversals. Further, and despite the large theoretical bounds on size, the algorithm has in practice been very fast and consistently produced triangulations comparable in size to other methods. The major factor dominating the size of the quadtree seems to be the vertex condition; input polygons having vertices with small binary expansions result in quite small triangulations. Given the intended application domain (grids for the finite element method in fluid dynamics), even grids containing vertices with long binary expansions can be quite useful—the physics of fluid movement suggests that placing many nodes around corners is very often desireable.

The heirarchical representation has a number of other advantages too. Not only is adaptivity localized, but it also suggests some obvious decomposition strategies for parallel construction, and for the (very expensive) solving of the system of equations. Partitioning a quadtree is a considerably more regular a problem than partitioning an arbitrary-shaped mesh of triangles. As well, the quadtree provides an efficient point location structure, which can be quite useful for attendant grid problems. User input, for example, is often desireable in order to inspect solution progress or to guide grid creation/management, but locating the point of a mouse click within a large unstructured grid can require other non-trivial data structures as well as the ones for grid creation and storage. The tree structure of a quadtree, coupled with the "semi-balanced" nature provided by the balance condition, can allow for efficient point location without any extra structure.

Our current work, jointly with colleagues in mechanical engineering, is focussed on integrating this grid-generation scheme with an adaptive solver based on the control-volume finite element method (CVFEM) due to Baliga and Patankar [3]. We are developing parallel implementations and are experimenting with a variety of platforms.

References

[1] Franz Aurenhammer, *Voronoi diagrams — a survey of a fundamental geometric data structure*, ACM Computing Surveys, 3 (1991), pp. 345–405.

[2] Brenda S. Baker, Eric Grosse, and Conor S. Rafferty, *Nonobtuse triangulation of polygons*, Discrete Comput. Geom., (1988), pp. 147–168.

[3] B.R. Baliga and S.V. Patankar, *Elliptic systems: Finite-element method II*, in Handbook of Numerical Heat Transfer, W.J. Minkowycz, E.M. Sparrow, G.E. Schneider, and R.H. Fletcher, eds., John Wiley & Sons Inc., 1988, pp. 421–461.

[4] Joshua E. Barnes and Piet Hut, *A heirarchical $O(n \log n)$ force calculation algorithm*, Nature, 4 (1986), pp. 446–449.

[5] Marshall Bern and David Eppstein, *Polynomial-size nonobtuse triangulation of polygons*, in Proceedings of the 7th ACM Symposium on Computational Geometry, 1991, pp. 342–350.

[6] Marshall Bern and David Eppstein, *Mesh generation and optimal triangulation*, in Computing in Euclidian Geometry, F.K. Huang, ed., World Scientific, 1992.

[7] Marshall Bern, David Eppstein, and John Gilbert, *Provably good mesh generation*, in Proceedings of the 32nd IEEE Symposium on the Foundations of Computer Science, 1990, pp. 231–241.

[8] Marshall Bern, David Eppstein, and John Gilbert, *Provably good mesh generation*, J. Comput. System Sci., (1994), pp. 384–409.

[9] Marshall Bern, Scott Mitchell, and Jim Ruppert, *Linear-size nonobtuse triangulation of polygons*, in Proceedings of the 10th ACM Conference on Computational Geometry, 1994, pp. 221–230.

[10] A. Bowyer, *Computing Dirichlet tesselations*, Comput. J., 2 (1981), pp. 162–166.

[11] P.L. George, *Automatic Mesh Generation: Application to Finite Element Methods*, John Wiley & Sons, 1991.

[12] Patrick Knupp and Stanly Steinberg, *Fundamentals of Grid Generation*, CRC Press, 1993.

[13] M. Pourazady and M. Radhakrishnan, *Optimization of a triangular mesh*, Comput. & Structures, 3 (1991), pp. 795–804.

[14] Franco P. Preparata and Michael Ian Shamos, *Computational Geometry: An Introduction*, Springer-Verlag, second edition, 1988.

[15] D.F. Watson, *Computing the n-dimensional Delaunay tesselation with applications to Voronoi polytopes*, Comput. J., 2 (1981), pp. 167–172.

[16] N. P. Weatherill, *Numerical grid generation*, Lecture Series 1990-06, von Karman Institute for Fluid Dynamics, 1990.

[17] M.A. Yerry and M.S. Shephard, *A modified quadtree approarch to finite element mesh generation*, IEEE Computing Applications, (1983), pp. 39–46.