LA-11812-MS

CAVEAT-GT: A General Topology Version of the CAVEAT Code

Los Alamos

Los Alamos National Laboratory is operated by the University of California for the United States Department of Energy under contract W-7405-ENG-36.

DISCLAIMER

An Affirmative Action/Equal Opportunity Employer

LA--11812-MS

DE90 013201

CAVEAT-GT: A General Topology Version of the CAVEAT Code

Michael C. Cline
John K. Dukowicz
Frank L. Addessio

Los Alamos National Laboratory
Los Alamos,New Mexico 87545

PAGES ii to iv

WERE INTENTIONALLY

LEFT BLANK

TABLE OF CONTENTS

CAVEAT-GT: A GENERAL TOPOLOGY VERSION OF THE CAVEAT CODE

by

Michael C. Cline, John K. Dukowicz, and Frank L. Addessio

ABSTRACT

We describe a numerical technique for solving two-dimensional, compressible, multimaterial problems using a general topology mesh. Multimaterial problems are characterized by the presence of interfaces whose shapes may become arbitrarily complex in the course of dynamic evolution. Computational methods based on more conventional fixed-connectivity quadrilateral meshes do not have adequate flexibility to follow convoluted interface shapes and frequently fail due to excessive mesh distortion. The present method is based on a mesh of arbitrary polygonal cells. Because this mesh is dual to a triangulation, its topology is unrestricted and it is able to accommodate arbitrary boundary shapes. Additionally, this mesh is able to quickly and smoothly change local mesh resolution, thus economizing on the number of mesh cells, and it is able to improve mesh isotropy because in a region of uniform mesh the cells tend to become regular hexagons. The underlying algorithms are based on those of the CAVEAT code. These consist of an explicit, finite-volume, cell-centered, arbitrary Lagrangian-Eulerian (ALE) technique, coupled with the Godunov method, which together are readily adaptable to a general topology mesh. Several special techniques have been developed for this extension to a more general mesh. They include an interface propagation scheme based on Huygens' construction, a "near-Lagrangian" mesh rezoning algorithm that minimizes advection while enhancing mesh regularity, an efficient global remapping algorithm that is capable of conservatively transferring quantities from one general mesh to another and various mesh restructuring algorithms, such as mesh reconnection, smoothing, and point addition and deletion.

---

1. INTRODUCTION

The CAVEAT code [1] was developed as an efficient and effective method for computing problems containing multimaterial interfaces and internal slip. One of its main features is that all variables are cell centered, a fact made possible by the use of the Godunov method. CAVEAT has been successfully used on a remarkable variety of problems. However, it still has inherent geometrical mesh limitations that cause the code to lose accuracy,

or even to break down due to unresolved mesh distortion. This is not the common "mesh tangling" of Lagrangian codes, but a mesh distortion associated with large and irregular changes of the original material interfaces, which define the boundaries of the mesh.

The two-dimensional CAVEAT code makes use of general quadrilateral cells [1], while the three-dimensional version uses hexahedral cells with "ruled" surfaces [2]. This choice implies a simple underlying topology of the mesh, and the corresponding "logical" (i,j) or (i,j,k) data structure common to most finite-difference or finite-volume codes. This simple data structure has many advantages and it is entirely adequate for many problems. A common topological configuration of this type of mesh is a rectangular block (or hexahedral block) in logical space, and the aspect ratio of this block as well as the total number of cells is chosen and fixed at the initial setup stage. This is typical of most codes, including CAVEAT. Now, for those problems in which the mesh boundary can evolve into complex and unpredictable shapes, such a choice, made at the initial time, will not be adequate and may quickly result in an unacceptable mesh, in spite of efforts to regularize or smooth the internal mesh. This difficulty is a symptom of the lack of flexibility in the topology of the mesh. Thus, the experimental development of a two-dimensional code, called CAVEAT-GT, was undertaken primarily as a project intended to investigate a solution of this problem by the use of a general topology mesh.

CAVEAT-GT was conceived as a code using the same hydrodynamics method as the CAVEAT code, but utilizing a mesh of arbitrary polyhedral cells (not necessarily convex). Since such a mesh is dual to a triangulation, its topology is unrestricted. Such a mesh readily adapts to arbitrary and dynamically changing boundary shapes. Further, it has the property of being able to smoothly and rapidly change local mesh resolution, such as in regions of large boundary curvature. Since the cells are expected to become locally regular (regular pentagons, hexagons, etc.) away from the perturbing influence of the boundaries, the resulting mesh is expected to have greater rotational isotropy as compared to a mesh of quadrilaterals.

There has been relatively little experience with similar, general topology methods. In this regard, one may mention Free-Lagrange methods [3], as well as finite-element methods. Generally, however, either the underlying hydrodynamics technique is different, or else a dynamically evolving mesh is not considered. Therefore, the development of CAVEAT-GT has been necessarily experimental. Nevertheless, the development of the code has been taken to the stage where nontrivial problems may be attempted successfully. The project is not completed, however. To be useful as a working code, additional features must be added, and further work on robustness and improved speed must be undertaken. This, we hope, will be possible in the future.

Since CAVEAT-GT shares the basic finite-volume Godunov method with CAVEAT, we will minimize discussion of the common aspects, which are available in the CAVEAT report [1], instead concentrating on the unique features and techniques developed for this code.

2. GENERAL DESCRIPTION

2.1. Mesh and Data Structure

2.1.1. Mesh. The general topology mesh is the primary feature of the CAVEAT-GT code. We will therefore describe it in some detail. The mesh, as well as the associated hydrodynamics method, is two-dimensional and can represent either plane or axisymmetric geometry.

The mesh is divided into a number of nonoverlapping blocks or regions. These regions are typically, but not necessarily, associated with different materials. The regions can adjoin and interact along their boundaries, in which case they form interfaces. The region boundaries are defined by straight line segments joining points (vertices) located along the boundary. Thus, the region boundaries may have arbitrary shape, constrained only by the requirement that there be no intersections. The regions may be multiply connected, so that a region may have several disjoint boundaries. Along interfaces, corresponding vertices, associated with different regions, are constrained to be coincident. Thus, along interfaces there is a double line of coincident vertices and straight line segments joining them. At certain points, called triple points, there may be more than two regions in contact. At such points there will be more than two coincident vertices. Symmetry lines, although not separating different materials, are treated as interfaces.

The boundary vertices, arranged in some order, define the shape and the interior of the region. The region interior contains a number of interior vertices. These interior vertices, together with the boundary vertices, form the vertices of a regular triangulation. The triangulation, which connects the vertices, is arbitrary in general, except that there can be no overlapping triangles. This triangulation is not the mesh used directly in the hydrodynamics calculation, but it is very useful in describing the topology (connectivity) of the mesh and the associated data structure. The straight-line segments defining the region boundaries are triangle sides in this triangulation.

Useful topological relationships for a simply connected region are

$$T = 2V - V_b - 2 \ ,$$

$$S = 3V - V_b - 3 \ ,$$

where S, T, and V are the total number of sides, triangles, and vertices, respectively, and V_b is the number of boundary vertices. That is, $V = V_i + V_b$, where V_i is the number of interior vertices. Thus, for a large mesh, unperturbed by boundary effects, there will be an average of two triangles and three sides per vertex, and each vertex will have an average of six neighbors.

For each boundary triangle side of a region we associate a fictitious boundary triangle, as illustrated in Fig. 1. These boundary triangles facilitate inter-region communication and storage of boundary quantities. They have no geometrical significance.

Associated with this triangulation is a dual mesh. This is the mesh actually used for hydrodynamics. Thus, associated with each triangle is a point called the cell vertex. Initially, when a new mesh is constructed, the cell vertex is placed at the triangle centroid. The algorithm is such that subsequently the cell vertex will always remain at the centroid of its associated triangle. Cell vertices of neighboring triangles are connected, as illustrated in Fig. 2, to form closed polygonal cells. Each cell is then associated with a triangle vertex or a cell point. These are the computational cells for the hydrodynamics method. Such cells need not be convex.

Computational cells that lie along a region boundary are somewhat different since the region boundary forms part of the cell boundary. Thus, boundary vertices are also cell points, and there are also cell vertices, associated with the fictitious boundary triangles, which are placed, by convention, midway along the associated boundary segments. This is illustrated in Fig. 3.

The above describes only the general properties and characteristics of our mesh. Such a mesh is capable of zoning arbitrarily shaped regions and dynamically changing its topology as the shape of the regions changes. The specific algorithms for defining and changing the mesh will be described subsequently.

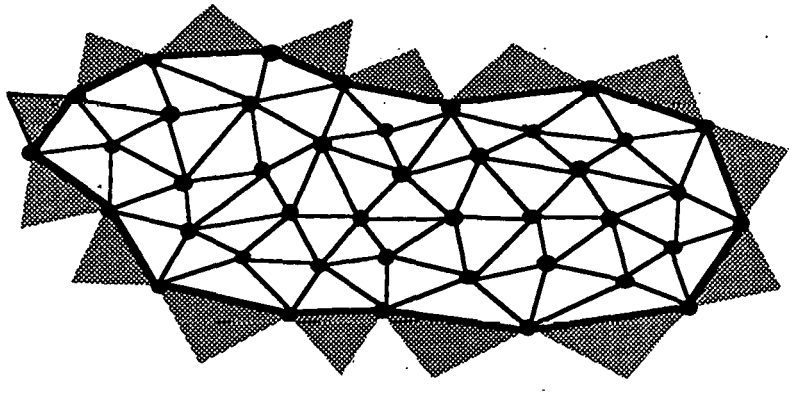


Fig. 1. A typical triangulation of a region, showing fictitious boundary triangles.

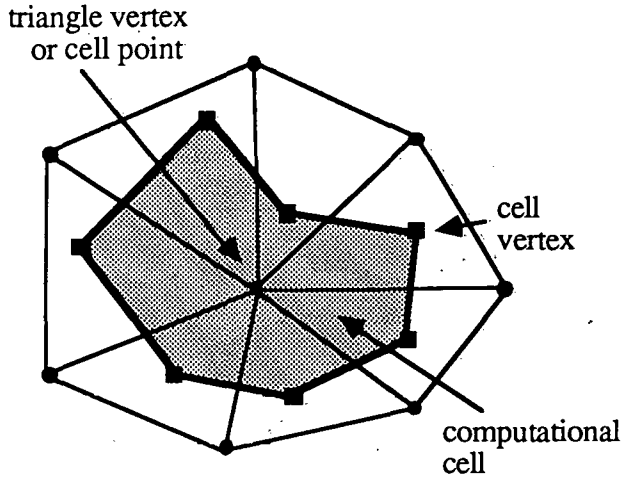


Fig. 2. An internal computational cell and the associated (dual) triangulation.

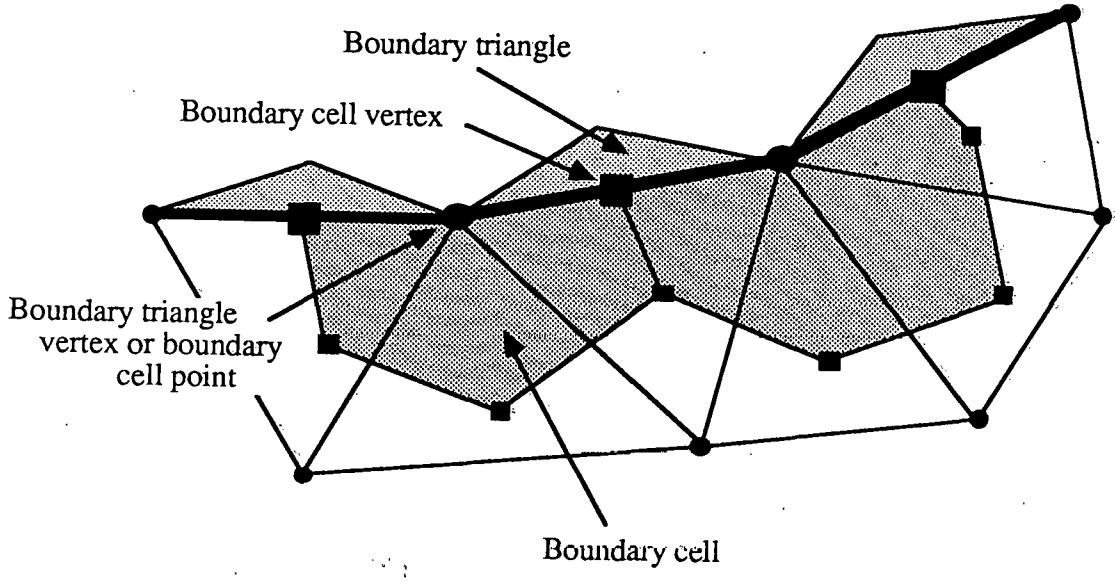


Fig. 3. Boundary cells and cell vertices.

2.1.2. Geometrical Properties. The CAVEAT-GT computational regions are composed of closed polygonal cells with an arbitrary number of sides. It is from this mesh that the code derives its flexibility. The mesh geometry in the vicinity of an interface between two regions is provided in Fig. 4. It may be seen from Fig. 4, that the boundary contours and points along interfaces are doubly defined. That is, there is a unique interface associated with each region. Cell quantities are defined at three locations; at cell centers, cell faces, and cell vertices. These locations are denoted using the index notation k, m, and n,

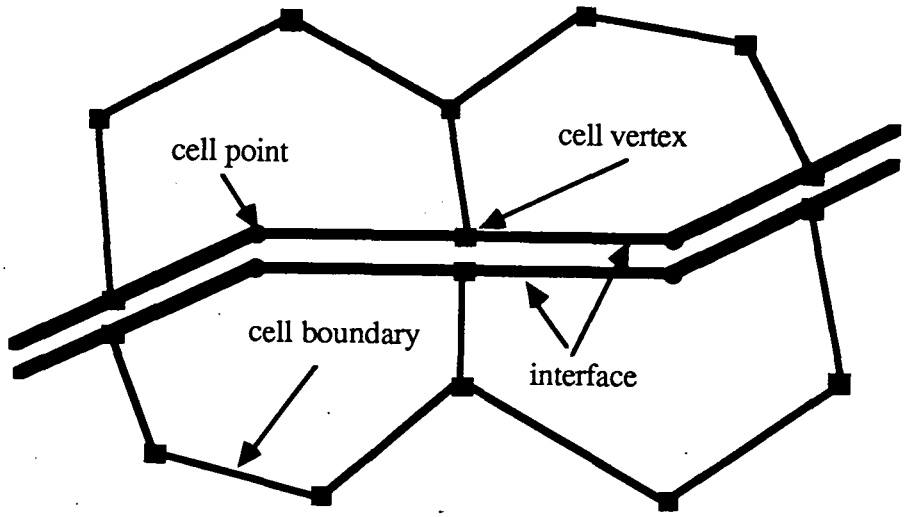


Fig. 4. Interface mesh geometry.

respectively. Primary-extensive quantities (mass, momentum, and total energy), as well as average-intensive quantities (density, velocity, internal energy, and pressure) are associated with cell centers. Gradients of these quantities also are associated with the cell centers for the higher order approach. Variables necessary to evaluate surface integrals (velocity and pressure) are assigned to the cell sides. Vertex quantities include the coordinate positions necessary to determine the mesh.

CAVEAT-GT may be used to analyze problems in either Cartesian (x,y) or cylindrically symmetric (r,z) coordinates. For cylindrical coordinates, all equations are expressed on a per radian basis. The two coordinate representations are modeled by introducing pseudo-Cartesian (x,y) coordinates through the use of a pseudo-radius:

$$R(x) = (1 - \beta) + \beta x ,$$

where $\beta = 0$ or 1 for Cartesian or cylindrical coordinates, respectively. In the cylindrical coordinate system, x is the radial coordinate and y is the axial coordinate. (In the code, $1 - \beta$ = CART and β = CYLN.)

During the course of each computational cycle, it is necessary to calculate the geometrical properties for each cell. These properties include the cell area, centroid, and a minimum cell distance. The cell areas are calculated by summing up the areas of the triangles that comprise each cell (Figs. 5 and 6):

$$A = \sum_n \frac{1}{2} [(\mathbf{x}_2 - \mathbf{x}_0) \times (\mathbf{x}_1 - \mathbf{x}_0) \cdot \mathbf{e}_z]_n . \tag{2.1}$$

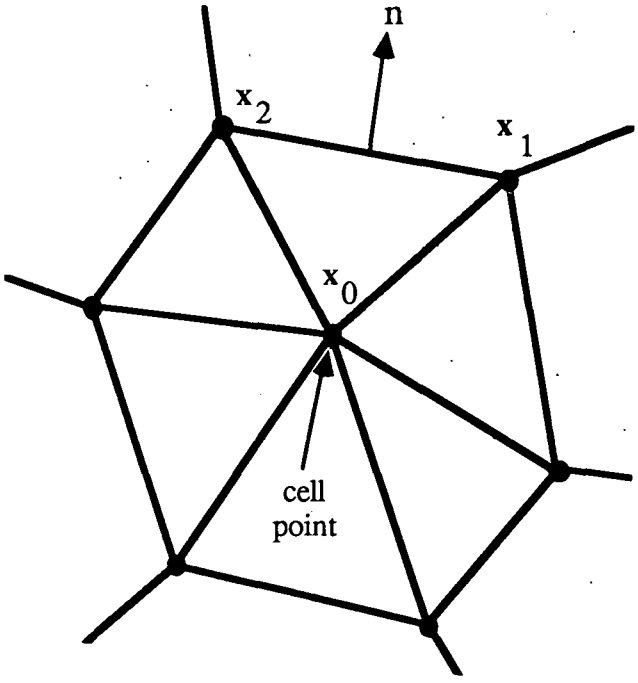


Fig. 5. Interior computational cell and the associated triangles.

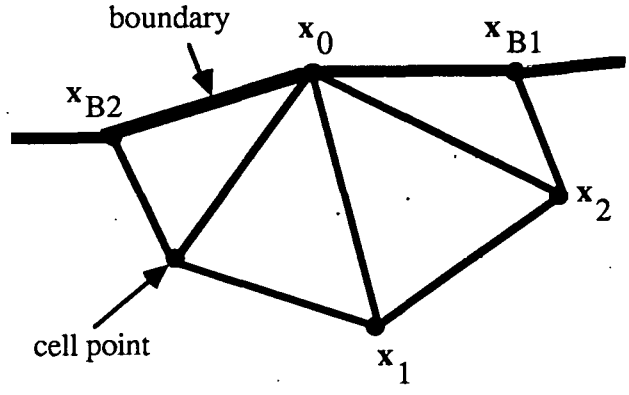


Fig. 6. Boundary computational cell and the associated triangles.

This is equivalent to summing the areas of all triangles whose vertices include the end points of each side and the origin. The number of triangles (n) associated with each cell is of course equal to the number of sides of the cell. For interior cells, the cell center ($\mathbf{x}_0$) cancels out (Fig. 5) when the sum is performed in Eq. (2.1)

$$A_I = \frac{1}{2} \sum_n [(\mathbf{x}_2 \times \mathbf{x}_1) \cdot \mathbf{e}_z]_n \,. \tag{2.2}$$

The cell center does not cancel for boundary cells (Fig. 6). Consequently, boundary cell areas are obtained by first calculating the sum provided in Eq. (2.2) and then adding in the terms involving the cell center location

$$A_B = \frac{1}{2} \sum_n \left\{ [(\mathbf{x}_2 - \mathbf{x}_1) \cdot \mathbf{e}_z] + [\mathbf{x}_0 \times \mathbf{x}_{B2} - \mathbf{x}_0 \times \mathbf{x}_{B1}] \cdot \mathbf{e}_z \right\}_n \,. \tag{2.3}$$

The volume of a computational cell is the sum of the volumes of the triangles that comprise the cell

$$V = \sum_k \langle R \rangle_k A_k \,, \tag{2.4}$$

where $<R>_k$ is the average pseudo-radius:

$$\langle R \rangle_k = (1 - \beta) + \beta \frac{1}{3} (\mathbf{x}_2 + \mathbf{x}_1)_k \ , \tag{2.5}$$

and for a boundary cell

$$\langle R \rangle_k = (1 - \beta) + \beta \frac{1}{3} (x_{B2} + x_{B1})_k \ . \tag{2.6}$$

In CAVEAT-GT, all cell-centered variables are associated with the cell centroid ($\mathbf{x}_c$). When cell side values are needed, for example, the variable, its gradient, and the distance from the side to the cell centroid are used. By definition

$$\mathbf{x}_c = \frac{1}{A} \int \int \mathbf{x} \, dA \ . \tag{2.7}$$

For convenience, consider only the x-component. To simplify Eq. (2.7), define the vector function $\mathbf{F} = \frac{1}{2} x^2 \mathbf{e}_x$. Then $\nabla \cdot \mathbf{F} = x$ and using the divergence theorem

$$\mathbf{x}_c = \frac{1}{2A} \oint x^2 \mathbf{e}_x \cdot \mathbf{n} \, dS \ , \tag{2.8}$$

where S is the contour around the cell area A and $\mathbf{n}$ is the unit outward normal to S. For each cell boundary segment ($\mathbf{x}_2 - \mathbf{x}_1$), a parametric representation is used

$$\mathbf{n} \, dS = [(y_2 - y_1)\mathbf{e}_x, (x_1 - x_2)\mathbf{e}_y] ds \ , \tag{2.9}$$

$$x = x_1 + s(x_2 - x_1) \ ,$$

where $0 \le s \le 1$. Substituting this parametric representation into Eq. (2.8) and performing the integration results in the equations (Fig. 5)

$$x_c = \frac{1}{6A} \sum_m [(y_2 - y_1)(x_2^2 + x_2 x_1 + x_1^2)]_m \ ,$$

and

$$y_c = - \frac{1}{6A} \sum_m [(x_2 - x_1)(y_2^2 + y_2 y_1 + y_1^2)]_m \ . \tag{2.10}$$

The area is obtained using Eq. (2.2). The sums are taken over all sides enclosing the cell. For a cell adjacent to a region boundary (Fig. 6), the two boundary segments $|\mathbf{x}_{B2} - \mathbf{x}_0|$ and $|\mathbf{x}_0 - \mathbf{x}_{B1}|$ must be included in the sum of Eq. (2.10).

CAVEAT-GT is an explicit code. That is, new time variables are obtained using source terms expressed as functions of old time variables. For this reason, there is a Lagrangian stability limit on the time-step size (Δt). The stability limit is a function of the minimum characteristic size of the computational cell (Δx_{min}). This characteristic size is obtained by fitting an ellipse to the cell which is done by first determining the moments of inertia of each cell about the cell's centroid

$$I_x = \int\int y^2\, dA - A\, y_c^2 ,$$

$$I_y = \int\int x^2\, dA - A\, x_c^2 , \tag{2.11}$$

$$I_{xy} = \int\int xy\, dA - A x_c y_c ,$$

where the area integrals are the cell moments of inertia about the origin (I_{xo}, I_{yo}, I_{xyo}). For example,

$$I_{xo} = \int\int y^2\, dA .$$

Defining $\mathbf{F} = \frac{1}{3} y^3 \mathbf{e}_y$, then $\nabla \cdot \mathbf{F} = y^2$. Using the divergence theorem, substituting in the parametric representation Eq. (2.9), and integrating one obtains

$$I_{xo} = -\frac{1}{12} \sum_m \{(x_2 - x_1)[y_2^3 + y_2^2 y_1 + y_2 y_1^2 + y_1^3]\}_m . \tag{2.12}$$

Similarly,

$$I_{yo} = \frac{1}{12} \sum_m \{(y_2 - y_1)(x_2^3 + x_2^2 x_1 + x_2 x_1^2 + x_1^3]\}_m ,$$

$$I_{xyo} = \frac{1}{24} \sum_m \{(y_2 - y_1)[x_2^2(3y_2 + y_1) + 2x_2 x_1 (y_2 + y_1) + x_1^2 (y_2 + 3y_1)]\}_m , \tag{2.13}$$

where the sum again is taken over all of the sides which comprise the cell. Therefore, the smaller of the two principal moments of inertia of the cell is

$$I_{min_c} = \frac{1}{2}(I_x + I_y) - \sqrt{\frac{1}{4}(I_x - I_y)^2 + I_{xy}^2} \quad . \tag{2.14}$$

For an ellipse, whose major and minor axes are 2a and 2b respectively, the area and minimum moment of inertia about its centroid are

$$A_e = \pi ab \ ,$$

and

$$I_{min_e} = \frac{\pi}{4} ab^3 \ . \tag{2.15}$$

Therefore, the minor axis is

$$b = 4\sqrt{\frac{I_{min_e}}{A_e}} \ . \tag{2.16}$$

Because the ellipse approximates the shape of the cell, the minimum cell distance is therefore

$$\Delta x_{min} \simeq 4\sqrt{\frac{I_{min_c}}{A}} \ ,$$

where the area (A) is obtained from Eq. (2.2).

2.1.3. "Z" Data Structure. Within the code, certain data are associated with triangles, triangle sides, and vertices. For example, cell vertices are associated with triangles, cell faces with triangle sides, and cells with triangle vertices. Vertices, sides, and triangles are sequentially numbered and the associated data are stored in arrays according to this numbering. The numbering is essentially random, except that boundary sides and triangles are numbered last and therefore data associated with them will appear at the end of the corresponding arrays.

The relationship of the vertices, sides, and triangles among themselves and their neighbors is described by the data structure. The data structure allows one to reconstruct the mesh by specifying which vertices are connected, by which sides, and in which order. In CAVEAT-GT the data structure is based primarily on triangle sides.

Consider two neighboring triangles, associated with side **m**, as illustrated in Fig. 7. We pick the two sides connected to side **m** that form the two arms of the letter "Z" (as opposed to the letter "S"). We label the vertex associated with the lower corner of the "Z" as **k1**, the side connected to this vertex as **m1**, and the triangle located between sides **m** and **m1** as **n1**. Similarly, we label the corresponding quantities associated with the upper corner as **k2**, **m2**, and **n2**. Thus, for each side **m** we store and associate the six quantities **k1**, **k2**, **m1**, **m2**, **n1**, and **n2**. The choice of orientation of the "Z" (rightside up or upside down) is arbitrary.

The information contained in the "Z's" is sufficient to reconstruct the mesh. It contains not only information on connectivity but also on the ordering (or sequence) of the sides or vertices around a given vertex. Additional information would be redundant.

This data structure is not unique. It does have the advantage of being compact and economical since it stores precisely six quantities per triangle side. Other schemes, such as those based on vertices, have the disadvantage of having to store an unspecified number of quantities per vertex (the number of vertex neighbors is unspecified, although the average is six).

The boundary "Z's" differ from the interior ones because there are no real neighboring triangles outside the mesh associated with a boundary side **m**. We have adopted the following convention: Triangle **n1** or **n2** will be the fictitious boundary triangle associated with the boundary side **m**. If the boundary side **m** is part of an interface, then the corresponding coincident boundary side of the adjoining region is stored for **m1** or **m2**. If the boundary side **m** is not on an interface, then there is no coincident side, and a zero is stored for **m1** or **m2**. This convention is the means for logically connecting different regions together.

Although redundant, we have found it convenient and useful to also store, for each triangle, the three triangle sides in counterclockwise order. This facilitates intercommunication between triangles and sides. It has also been found useful to define boundary and interface based data structures. A full description is given in Sec. 3.1.

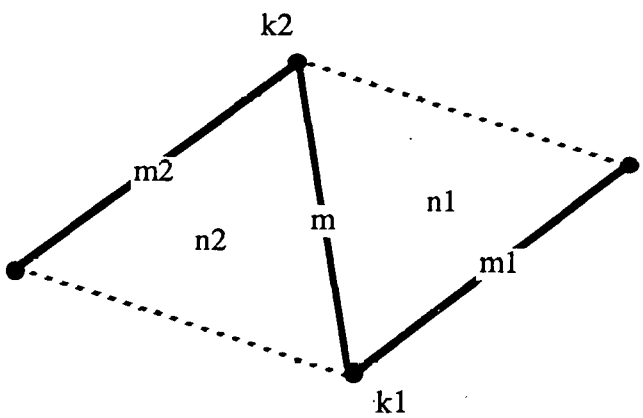


Fig. 7. "Z" data structure associated with side m and the triangulation.

2.2. Hydrodynamics

2.2.1. Finite-Volume Method. CAVEAT-GT, like CAVEAT, uses the finite-volume or control-volume formulation of the conservation equations. In this formulation, the conservation equations are integrated over an arbitrary moving control volume $V(t)$. The general conservation equations in this formulation become

Mass Conservation

$$\frac{d}{dt}\int_{V(t)} \rho dV + \int_{S(t)} \rho(\mathbf{u} - \mathbf{u}_v)\cdot\mathbf{n}dS = 0 ,$$

Momentum Conservation

$$\frac{d}{dt}\int_{V(t)} \rho\mathbf{u}dV + \int_{S(t)} \rho\mathbf{u}(\mathbf{u} - \mathbf{u}_v)\cdot\mathbf{n}dS = -\int_{S(t)} p\mathbf{n}dS + \int_{S(t)} \mathbf{n}\cdot\Pi dS + \int_{V(t)} \rho\mathbf{F}dV , \qquad (2.17)$$

Energy Conservation

$$\frac{d}{dt}\int_{V(t)} \rho E dV + \int_{S(t)} \rho E(\mathbf{u} - \mathbf{u}_v)\cdot\mathbf{n}dS = -\int_{S(t)} p\mathbf{u}\cdot\mathbf{n}dS + \int_{S(t)} \mathbf{n}\cdot\Pi\cdot\mathbf{u}dS$$

$$-\int_{S(t)} \mathbf{h}\cdot\mathbf{n}dS + \int_{V(t)} \rho\mathbf{F}\cdot\mathbf{u}dV + \int_{V(t)} Q dV .$$

In these equations, $S(t)$ is the surface of the control volume $V(t)$, which moves with the local surface velocity $\mathbf{u}_v$, and $\mathbf{n}$ is the unit normal vector directed outward from the surface. The operator d/dt indicates time rate of change following the motion of the control volume $V(t)$. Note that the control volume $V(t)$ may be arbitrary in shape and so this method is directly applicable to the multi-sided cells of CAVEAT-GT. Continuing the definitions, ρ is the mass density, $\mathbf{u}$ is the material velocity, $E = e + \frac{1}{2}\mathbf{u}\cdot\mathbf{u}$ is the specific total energy, and e is the specific internal energy. The pressure p is given as a function of ρ and e by an equation of state. In general, there will exist a deviatoric stress tensor Π, a specific body force $\mathbf{F}$, a heat flux vector $\mathbf{h}$, and an energy release rate per unit volume Q.

The above equations may be supplemented by the equation for conservation of volume:

$$\frac{dV(t)}{dt} = \int_{S(t)} \mathbf{u}_v\cdot\mathbf{n}dS , \qquad (2.18)$$

which also may be represented in kinematic form as

$$\frac{d\mathbf{x}_v}{dt} = \mathbf{u}_v \,, \tag{2.19}$$

where $\mathbf{x}_v$ is the coordinate defining the surface of the control volume.

CAVEAT-GT uses an Arbitrary Lagrangian-Eulerian (ALE) method to solve these equations. In this method the calculation is divided into a Lagrangian and a remapping phase. In the Lagrangian phase we set the control volume velocity equal to the material velocity ($\mathbf{u}_v = \mathbf{u}$) to obtain

$$\frac{d}{dt}\int_{V_L} \rho dV = 0 \,,$$

$$\frac{d}{dt}\int_{V_L} \rho \mathbf{u} dV = -\int_{S_L} p\mathbf{n} dS + \int_{S_L} \mathbf{n}\cdot\Pi dS + \int_{V_L} \rho \mathbf{F} dV \,, \tag{2.20}$$

$$\frac{d}{dt}\int_{V_L} \rho E dV = -\int_{S_L} p\mathbf{u}\cdot\mathbf{n} dS + \int_{S_L} \mathbf{n}\cdot\Pi\cdot\mathbf{u} dS - \int_{S_L} \mathbf{h}\cdot\mathbf{n} dS + \int_{V_L} \rho\mathbf{F}\cdot\mathbf{u} dV + \int_{V_L} Q dV \,,$$

where V_L is the Lagrangian control volume, S_L is its surface, and d/dt is the Lagrangian (material) time derivative. Following the Lagrangian phase we may wish to transfer quantities to a different mesh (i.e., we wish to transfer from V_L to a new mesh volume $\hat{V}_k$). This is performed in the remapping phase. The equations that accomplish this are

$$\frac{d}{dt}\int_{\hat{V}_k} q dV - \int_{\hat{S}_k} q\mathbf{v}_R\cdot\mathbf{n} dS = 0 \,, \tag{2.21}$$

where q represents the quantities ρ, $\rho\mathbf{u}$, and ρE, and $\mathbf{v}_R$ is a mesh velocity relative to the material velocity. It is clear, however, that we can alternatively obtain the final control volume quantities directly

$$Q_k = \int_{\hat{V}_k} q dV \,. \tag{2.22}$$

The use of Eq. (2.21) is known as advection or continuous remapping, and this is used when mesh changes are gradual, while the use of Eq. (2.22) is known as integral or global remapping, and this is useful when mesh changes are large or when the mesh topology changes.

In CAVEAT-GT the Lagrangian mesh is never explicitly calculated (i.e., using Eq. (2.19)), except for the normal motion of the interfaces and boundaries, which is done using a Huygens-like construction in subroutine INTFACE, and described in Sec. 2.3. Instead, a "near Lagrangian" mesh is constructed in the rezone phase (in subroutines REZTANG and REZLAGR, described in Sec. 2.4) and an advection algorithm, based on Eq. (2.21), is used for remapping quantities Q_k. The "near Lagrangian" algorithm is such that the cells of the new mesh attempt to have the same mass as the corresponding cells of the old mesh, as would be the case of Lagrangian cells, but they are less distorted than Lagrangian cells would be. Thus, this algorithm minimizes the magnitude of the relative velocity v_R in Eq. (2.21) and hence reduces advection errors. Nevertheless, even this "near Lagrangian" mesh can become distorted in time and must be replaced by a new mesh. Also, when the mesh topology is changed, for example when points are added or deleted to maintain resolution, an entirely new mesh must be constructed. A new mesh is produced by subroutine REZONE, and described in Sec. 2.4.1. When an entirely different mesh is introduced, the remapping is carried out using the global algorithm of Eq. (2.22). This is done in subroutine REMAPPER, and explained in Sec. 2.5.2. In general, the advection algorithm is much cheaper than the global remapping algorithm, and so we attempt to run with the "near Lagrangian" mesh for as long as possible before resorting to mesh restructuring and the use of a global remapping.

2.2.2. Godunov Method. The Lagrangian phase equations, Eqs. (2.20), require the pressure and normal velocity at the control volume surface. We use a version of the Godunov method in which these quantities, denoted by p^* and w*, respectively, are obtained from the solution of a local Riemann problem at the surface. An approximate Riemann solver is used [4], which is based on the following approximate shock Hugoniot:

$$p^* - p_s = \rho_s[a_s + A_s|w^* - w_s|](w^* - w_s)\,, \tag{2.23}$$

where the subscript s refers to the state on the left or right of the surface, and a_s and A_s are two material dependent parameters, defined as follows:

$$a_s = \textit{the local isentropic speed of sound} = \left[\frac{\partial p}{\partial \rho} + \frac{p}{\rho^2}\frac{\partial p}{\partial e}\right]_s^{\frac{1}{2}}, \tag{2.24}$$

$$A_s = \textit{limit of a strong shock} = \left[\frac{\rho_2/\rho_1}{\rho_2/\rho_1 - 1}\right]_s. \tag{2.25}$$

Thus, the strong shock parameter A_s is given in terms of the density ratio across a shock, in the strong shock limit. Typically this parameter does not depart greatly from unity, because of all materials with $p = p(\rho)$ it can be shown that $A_s = 1$, while for polytropic (ideal gas) equations of state, $A_s = 1/2\ (\gamma + 1)$. Thus, $1 \le A_s \le 4/3$ for the physically realistic range of γ ($1 \le \gamma \le 5/3$). Further details may be obtained in [1] and [4].

2.2.3. Gradients. CAVEAT-GT employs a Godunov numerical method to obtain solutions to the fluid dynamic equations. Consequently, the primary-extensive quantities such as mass, momentum, and energy, as well as the primary-intensive properties derived from them are associated with the centers of the polygonal cells. The computational procedure requires intensive quantities at the cell-faces. Cell-face intensive quantities are required to obtain left and right states for the Riemann solution and for advection and remapping.

The accuracy of the computation depends on the assumed spatial variation of a fluid dynamic quantity $\phi(\mathbf{x})$ about the centroid of the cell $\mathbf{x}_c$,

$$\phi(\mathbf{x}) = \phi(\mathbf{x}_c) + \nabla\phi_c \cdot (\mathbf{x} - \mathbf{x}_c) + 0(\Delta\mathbf{x}^2) \ . \tag{2.26}$$

The computational procedure is considered "first order" if $\phi(\mathbf{x}) = \phi(\mathbf{x}_c)$, that is, if all variables are assumed constant within a cell. It is considered "second order" if $\nabla\phi_c$ exists, that is, a linear variation is used for the flow variables.

Three options are available in CAVEAT-GT for computing the cell-centered gradients $\nabla\phi_c$. All three options first require computing trial gradients about the cell vertices $\mathbf{x}_n$ (Figs. 8 and 9), which are obtained as follows:

$$\begin{aligned} <\nabla_n\phi> &= \frac{1}{A_n} \int_{A_n} \nabla\phi \, dA \ , \\ &= \frac{1}{A_n} \oint_{C_n} \phi\mathbf{n} \, d\ell \ , \end{aligned} \tag{2.27}$$

using the divergence theorem. For an interior-cell vertex (Fig. 8), A_n is the area of the triangle whose vertices are located at the cell centroids ($\mathbf{x}_k$). For an exterior vertex (Fig. 9), A_n is the quadrilateral constructed using the boundary cell centroids and the boundary points located at the intersections of the boundary segments. C_n is the contour of integration around the area A_n and $\mathbf{n}$ is the unit-outward normal to C_n. Values of ϕ are available at the cell centroids (ϕ_c) from the solutions to the conservation equations and at the boundary points from the appropriate boundary conditions. A piecewise linear variation of ϕ is

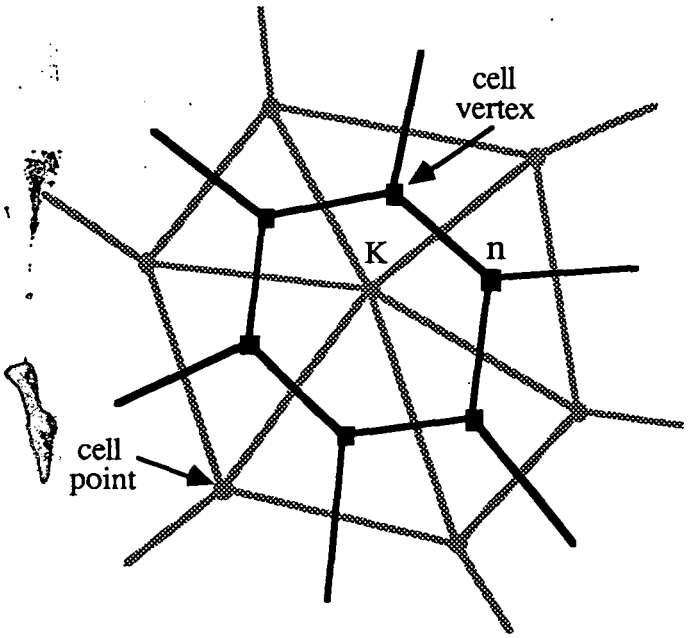


Fig. 8. Gradient calculation for an interior computational cell.

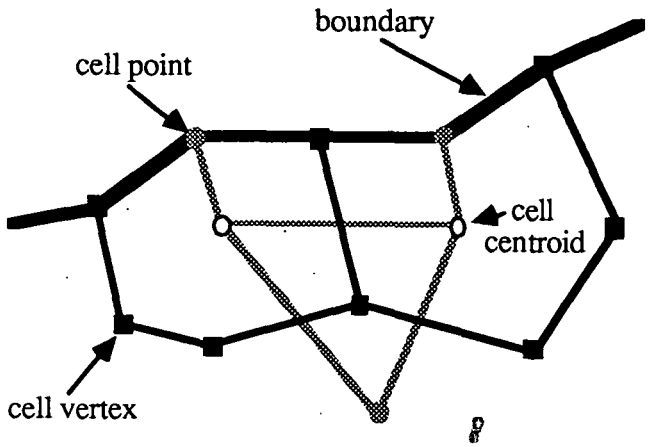


Fig. 9. Gradient calculation for a boundary computational cell.

assumed along the contour (C_n) consistent with the assumption that $\phi(\mathbf{x})$ is a linear function.

Cell-centered gradients may be computed directly from the trial gradients of Eq. (2.27) by simply taking an area average of the trial gradients at the cell vertices surrounding the cell centroid ($\mathbf{x}_k$), $\nabla_k\phi = <\nabla_k\phi>$, where

$$<\nabla_k\phi> = \frac{\sum_n A_n <\nabla_n\phi>}{\sum_n A_n} . \qquad (2.28)$$

This gradient is available in CAVEAT-GT as an option (**igrad** = 1). It uses no limiting to preserve local monotonicity. Consequently, in the vicinity of steep gradients, severe overshoots or undershoots may result when compared to neighboring data. For this reason, this option should be used with caution (negative densities may result, for example). However, its use can provide valuable insight into the diffusiveness of the other options.

To preserve monotonicity, the value of the gradient must be limited. When limiting is employed, the accuracy of the computation is reduced and additional dissipation is introduced into the calculation. Two methods are available in CAVEAT-GT for limiting the gradients. They include a van Leer (**igrad** = 2) and a monotone limiter (**igrad** = 3).

In the multidimensional extension to van Leer's one-dimensional limiter, a limiting coefficient α_k ($0 \le \alpha_k \le 1$) is obtained for each cell. Cell-centered gradients may then be obtained

$$\nabla_k\phi = \alpha_k <\nabla_k\phi> , \tag{2.29}$$

where $<\nabla_k\phi>$ are the nonlimited cell-centered gradients available from Eq. (2.28). The value of α_k is determined by enforcing local monotonicity. That is, the value of $\phi(\mathbf{x})$ within the cell is ensured to be bounded by the average values of ϕ in the neighboring cells. Therefore

$$\alpha_k = min(1., \alpha_{k_{max}}, \alpha_{k_{min}}) ,$$

where

$$\alpha_{k_{max}} = max\{0., [\bar{\phi}_{k_{max}} - min(\phi_k, \bar{\phi}_{k_{max}})]/[\phi_{k_{max}} - \phi_k]\} ,$$

$$\alpha_{k_{min}} = max\{0., [\bar{\phi}_{k_{min}} - max(\phi_k, \bar{\phi}_{k_{max}})]/[\phi_{k_{min}} - \phi_k]\} .$$

$\bar{\phi}_{k_{max}}, \bar{\phi}_{k_{min}}$ are the maximum and minimum values of $\phi(\mathbf{x}_c)$ in the neighboring cells. $\phi_{k_{max}}, \phi_{k_{min}}$ are the maximum and minimum values of ϕ in the cell obtained using Eq. (2.26) with $\nabla\phi_c = <\nabla_k\phi>$.

The van Leer limiting procedure permits the steepest possible gradients without excessive oscillations. However, this limiter is not guaranteed to be monotone because it can result in a "sawtooth" distribution of ϕ at cell boundaries. Consequently, a more conservative and therefore a more diffusive limiter is available in CAVEAT-GT. This last limiting procedure is referred to as the monotone limiter (**igrad** = 3):

$$\nabla_k\phi = \begin{cases} (\nabla_k\phi)_{min} , & (\nabla_k\phi)_{min} > 0 , \\ (\nabla_k\phi)_{max} , & (\nabla_k\phi)_{max} < 0 , \\ 0 , & otherwise , \end{cases}$$

where

$$(\nabla_k\phi)_{max} = max\{ <\nabla_n\phi> , \ n : \ cell\ vertices\} ,$$

$$(\nabla_k\phi)_{min} = min\{ <\nabla_n\phi> , \ n : \ cell\ vertices\} ,$$

and $<\nabla_n\psi>$ are the trial gradients obtained from Eq. (2.27).

2.2.4. Timestepping. Time differencing in CAVEAT-GT is explicit. Thus, in the Lagrangian phase, for any conserved cell quantity Q_k, such as volume, momentum, or total energy, we have

$$Q_k^{n+1} = Q_k^n + \Delta t^n \dot{Q}_k^n , \tag{2.30}$$

where n is the current cycle index, $\Delta t^n = t^{n+1} - t^n$, is the current time-step, and $\dot{Q}_k{}^n$ is the Lagrangian phase rate of change of $\dot{Q}_k$ (i.e., the right-hand-side of Eqs. (2.20), calculated in subroutine LAGRATES).

The time-step Δt^n is determined in subroutine TIMESTEP. Currently, there are four criteria for limiting the time-step incorporated into this subroutine. The first (subroutine argument LIMIT = 1) is based on the explicit CFL limit for a Lagrangian method. A minimum cell dimension d_{min} is determined in subroutine CELLGEOM, and described in Sec. 2.1.2. The time-step is then

$$\Delta t^n = min_k (d_{min}/a_k) ,$$

where a_k is the sound speed in cell k, and the minimum is taken over all cells k.

There are two limits on the time-step imposed by the Huygens construction used to advance interfaces and boundaries, described in Sec. 2.3. The first (LIMIT = 2) is called into effect when the construction creates a "loop" (i.e., a topological anomaly) in the boundary or interface shape. This is called the "bowtie" limit. The second restriction (LIMIT = 3) is involved if the interface segment is required to rotate by more than 90° relative to its original orientation (a physical impossibility). In both cases, the time-step is halved and the construction is repeated for all segments from their original position. Finally, there is an explicit accuracy limit (LIMIT = 4) that limits the Lagrangian volume change to 50%.

The time-step also is constrained to lie between the limits DTMIN and DTMAX, specified at the problem setup time.

2.3. Interfaces

2.3.1. Interface Propagation. Referring to Fig. 3, region boundaries are composed of straight line segments joining two cell points. Midway along a segment is a cell vertex separating the two cells. The above is true also for the coincident segment in case the boundary is part of an interface. Thus, for each such segment there will be two subsegments that separate two different boundary cells across the interface. The same is true for those boundary segments that represent free surfaces, except that the boundary represents an interface with a vacuum region. Each such subsegment defines a Riemann problem that is solved to obtain a normal velocity $\mathbf{w} = (\mathbf{u} \cdot \mathbf{n})\mathbf{n}$ associated with the subsegment. This

normal velocity is assumed to be located midway along the subsegment, or a quarter-length along the segment from each of the cell points. Boundary segments that represent symmetry boundaries, or specified velocity or pressure boundaries, also may be formulated in terms of Riemann problems, as described in Sec. 2.3.3, to determine the corresponding normal velocities on the subsegments.

The two normal velocities and their location along the segment define a linear distribution of normal velocity along that segment. We assume that the segment propagates with the local normal velocity. This is analogous to the propagation of a wavefront with a specified local wavefront velocity. The propagation of a wavefront is properly described by an Eikonal equation, whose solution may be obtained by the Huygens construction. In the present case, the Huygens construction predicts that the straight line segment will remain linear, as illustrated in Fig. 10.

Assuming that normal velocities w_1 and w_2 are located at points 1 and 2 along the original location of the segment, we draw circles of radius $w_1\Delta t$ and $w_2\Delta t$, centered at points 1 and 2, respectively, indicating the progress of "wavefronts" originating at points 1 and 2 during the time-step Δt. The new location of the segment will be along the envelope of all the circles centered at intermediate points, or, more simply, along the line tangent to both circles at points 3 and 4, respectively. These two points, defining the location of the new line, are given by

$$\mathbf{r}_3 = \mathbf{r}_1 + \frac{w_1}{w_2 - w_1}\left[-\beta^2\mathbf{r}_{21} + \beta\sqrt{1-\beta^2}\,\mathbf{k}\times\mathbf{r}_{21}\right],$$

$$\mathbf{r}_4 = \mathbf{r}_2 + \frac{w_2}{w_2 - w_1}\left[-\beta^2\mathbf{r}_{21} + \beta\sqrt{1-\beta^2}\,\mathbf{k}\times\mathbf{r}_{21}\right], \tag{2.31}$$

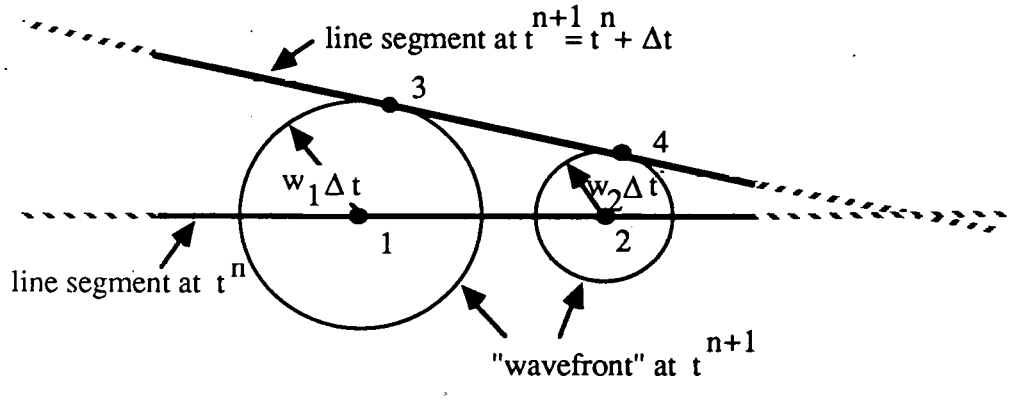


Fig. 10. Huygens construction for a boundary line segment.

and

$$\beta = (w_2 - w_1)\,\Delta t / |r_{21}| ,$$

where $\mathbf{r}_i = (x_i, y_i)$ are the coordinates of point i, $\mathbf{r}_{21} = \mathbf{r}_2 - \mathbf{r}_1$, and $\mathbf{k}$ is the unit vector in the coordinate direction orthogonal to the (x,y) plane.

Note that, as previously mentioned in Sec. 2.2.4, the time-step must be limited such that $\beta \leq 1$. This limits the new line to rotating by an angle of less than 90° or, from another point of view, it prevents the larger circle from engulfing the smaller one (a physical impossibility for a wavefront).

Given the collection of new boundary lines, we define the new segments to be the segments delimited by the intersection points of neighboring lines. Thus, given two neighboring lines with defining points 3, 4 and 3†, 4†, respectively, the intersection point $\mathbf{r}^*$ is given by the solution of the linear system

$$\mathbf{k} \cdot [(\mathbf{r}^* - \mathbf{r}_3) \times \mathbf{r}_{43}] = 0 ,$$

$$\mathbf{k} \cdot [(\mathbf{r}^* - \mathbf{r}\dagger_3) \times \mathbf{r}\dagger_{43}] = 0 , \tag{2.32}$$

which is

$$\mathbf{r}^* = \mathbf{r}_3 + \mathbf{k} \cdot [\mathbf{r}\dagger_{43} \times (\mathbf{r}\dagger_3 - \mathbf{r}_3)]\,\mathbf{r}_{43}/D , \tag{2.33}$$

where

$$D = \mathbf{k} \cdot (\mathbf{r}\dagger_{43} \times \mathbf{r}_{43}) .$$

This intersection point exists provided $D \neq 0$, that is, provided the two lines are not parallel.

Assuming all intersection points exist, they define the new shape of the boundary, unless a "loop" or a boundary intersection is created. This is illustrated in Fig. 11. Such a topological anomaly can be created if the time-step is too large. These loops are detected in subroutine BOWTIES, and if they exist then the time-step is reduced, as described in Sec. 2.2.4, until all are eliminated.

2.3.2. Resolving Singularities. The algorithm described in the previous section fails when two consecutive segments become parallel or collinear. In such a case an intersection point does not exist, and mathematically, the linear system of equations to be solved for the intersection point becomes singular. In order for the algorithm to be meaningful we

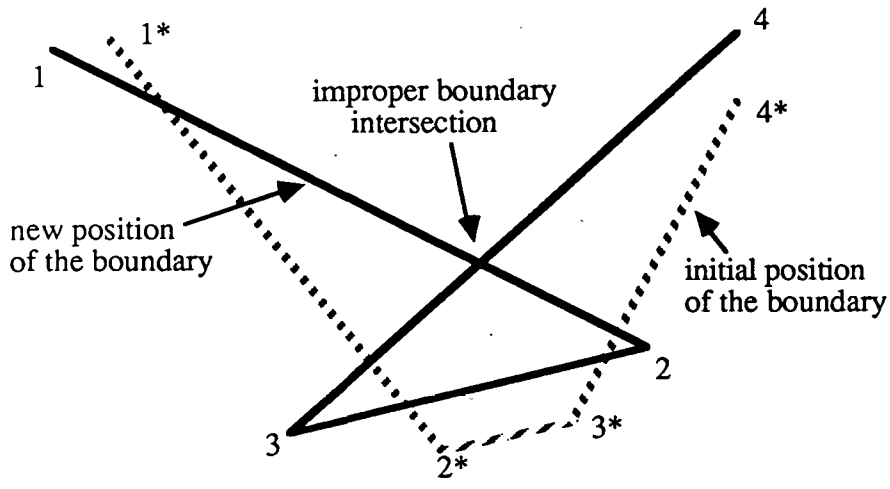


Fig. 11. An anomalous "loop" or "bowtie" created as a result of too large a time-step during the Huygens construction of the boundary shape.

must have a method for resolving this singularity. In practice, this situation will always arise along symmetry boundaries, where all segments are collinear, and occasionally during the dynamic evolution of the boundary when two consecutive segments become nearly parallel or collinear causing the intersection point to shoot off dramatically in either direction.

The procedure to resolve singularities is based on using a weighted average of two trial solutions. One solution is the side intersection points, $\mathbf{r}^*$, which was discussed in the previous section, while the second is the average of the new Riemann velocity points, $\mathbf{r}_a$ that are adjacent to the new intersection point as shown in Fig. 12. The new boundary point location is given by

$$\mathbf{r}_c^* = \begin{cases} \mathbf{r}^* , & \text{if } w \geq 1 , \\ w\mathbf{r}^* + (1-w)\mathbf{r}_a , & \text{if } w < 1 , \end{cases} \tag{2.34}$$

where

$$w = \frac{d_a^2}{2d_i^2} ,$$

d_a is the distance between the two new Riemann velocity points, and d_i is the distance from the new intersection point to the old cell point.

This algorithm is implemented in subroutine INTFACE.

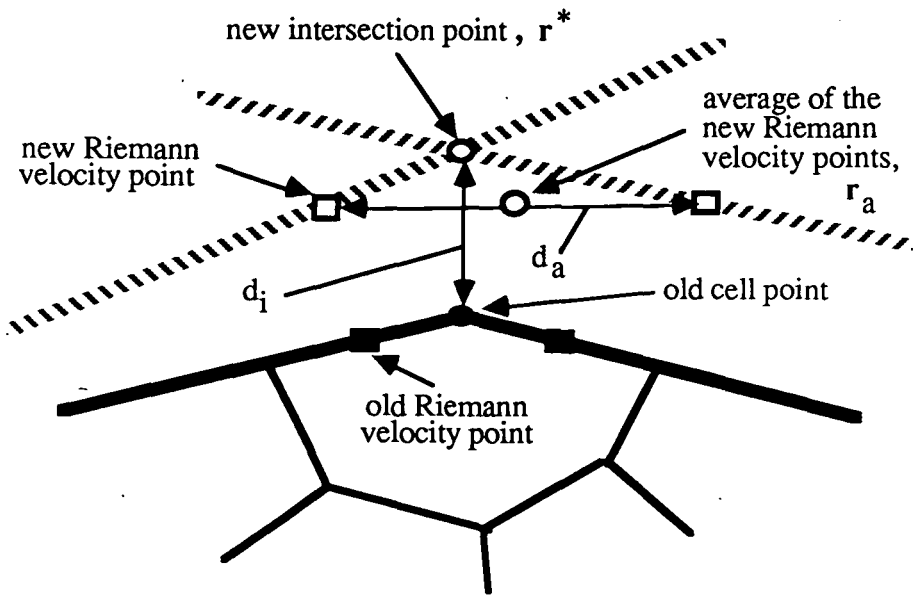


Fig. 12. Resolving boundary singularities.

2.3.3. Triple Points and Fixed Points. Usually there are certain points where a boundary is not smooth, i.e., where the boundary shape has a kink or a corner. The algorithm described above would inevitably smooth such kinks, which is undesirable. We therefore flag these points as special and obtain their coordinates by the Huygens construction, which preserves the kink. There are three types of such points. One type is classified as a triple point, and there are two types that are classified as fixed points.

Triple points occur, in general, when three regions adjoin at a point. The Huygens construction is indeterminate in such a case since there are three possible intersections. Since there is not enough physical information to resolve this inconsistency, we currently use a density weighted average of the three intersection points. The above is true even if one of the three regions represents a vacuum. However, if one of the regions is a symmetry half plane, then there is no inconsistency since the problem reduces to the intersection of a single pair of straight lines.

There are two types of fixed points. One type occurs because of the presence of symmetry boundaries. A fixed point of this type occurs where a region boundary (not an interface) intersects a symmetry line, or when two symmetry lines intersect (typically at right angles). The second type is a specific kink or corner in the boundary shape, present at the initial time, which is expected to persist during the solution. Such points are specified and flagged at the problem setup stage (Sec. 2.7).

The coordinates of triple and fixed points are calculated in subroutine TRIPLPT and they are flagged in subroutine BSTRUCT, except for the last type of fixed point, which is flagged in the setup code.

2.3.4. Boundary Conditions. Applying boundary conditions in a Godunov code is straightforward. Boundaries are treated in the same manner as interfaces, or even cell boundaries, by solving local Riemann problems. We identify six types of boundary conditions:

1) vacuum,
2) symmetry (reflecting),
3) specified pressure,
4) specified normal velocity,
5) specified inflow, and
6) outflow.

The case of the vacuum boundary is really a special case of the specified pressure boundary, with the pressure specified to be zero.

The symmetry or reflecting boundary condition uses the solution of a symmetric, but otherwise standard, Riemann problem in which the left and right hand states are identical except for the normal velocities, which are mirror images.

The specified pressure and velocity boundary conditions use a special Riemann solution obtained from the Hugoniot given by Eq. (2.23). If either p^* or w^* is specified, then this equation may be solved for the other quantity. Further details are available in Ref. [1].

The specified inflow boundary condition uses the specified velocity boundary condition to first move the inflow boundary with the inflow velocity, i.e., the specified inflow velocity is used in the Riemann solution. Next, the inflow boundary is moved back to its original location. Finally, the changes to the flow quantities necessitated by this non-Lagrangian motion of the inflow boundary are accounted for by local advection using the specified inflow density and internal energy. If the inflow velocity is supersonic, then the pressure is set equal to the specified inflow pressure.

The outflow boundary condition uses the specified pressure boundary condition to move the outflow boundary by using the specified outflow pressure in the Riemann solution. Next, the procedure follows that of the inflow boundary condition where now the local density and internal energy are used in the advection. If the outflow velocity is supersonic, then the local pressure is used in the Riemann solution.

The boundary conditions are computed in subroutine BOUNDARY.

2.4. Rezoning

2.4.1. Interior Algorithms. Rezoning refers to the process of creating a new mesh. In CAVEAT-GT we use two different rezoning algorithms to construct a mesh in the interior

of regions. The "near Lagrangian" algorithm constructs a mesh whose cells have nearly the same mass as the cells of the Lagrangian mesh, but with reduced distortion. This algorithm is useful to minimize advection errors when the mesh topology does not change. On the other hand, the general rezoning algorithm attempts to construct an entirely new mesh, regardless of topology, which is reasonably smooth and regular. Both algorithms manipulate the triangulation, rather than the computational mesh.

2.4.1.1. "Near Lagrangian" Algorithm. We start with a vector identity, which may be written as

$$\nabla^2 \mathbf{u} = \nabla D - \nabla \times \omega ,$$

where $D = \nabla \cdot \mathbf{u}$ is the divergence, and $\omega = \nabla \times \mathbf{u}$ is the vorticity, if the vector field $\mathbf{u}$ is considered to be a velocity field. Now, divergence is related to rate of change of Lagrangian cell volumes by

$$D = 1/v \, dv/dt , \tag{2.35}$$

where v is a specific Lagrangian volume, and vorticity, particularly nonuniform vorticity, is related to shear and mesh distortion. We now define a mesh velocity $\mathbf{u}_m$ by

$$\nabla^2 \mathbf{u}_m = \nabla D . \tag{2.36}$$

Taking the divergence and curl of this equation, we obtain

$$\nabla^2 D_m = \nabla^2 D ,$$

$$\nabla^2 \omega_m = 0 ,$$

where D_m and ω_m are the divergence and vorticity of the mesh velocity, respectively. We conclude, given that D_m and D satisfy the same boundary conditions, that $D_m = D$. Further, ω_m is a smooth function in the interior. Thus, Eq. (2.36) is suitable as the mesh generating equation with the property of preserving Lagrangian volumes, as suggested by Eq. (2.35). In practice, we modify Eq. (2.36) by defining

$$\delta \mathbf{r}_k = \mathbf{r}\dagger_k - \mathbf{r}_k = \Delta t \mathbf{u}_{m,k} ,$$

$$\delta V_k / V_k = (V\dagger_k - V_k)/V_k = \Delta t < D_k > ,$$

where $\mathbf{r}_k$ represents the position of cell point k, V_k is the cell volume, $<D_k>$ is the average cell divergence, and $\mathbf{r}\dagger_k$ and $V\dagger_k$ are the new position and volume, respectively, as a result of rezoning. Thus, the equation solved is Eq. (2.36) in the form

$$\nabla^2 \delta \mathbf{r}_k = \nabla (\delta V_k / V_k) , \tag{2.37}$$

where we have canceled a common factor of Δt.

In a Lagrangian computation the cell mass does not change ($M_k = \mathrm{M}_k{}^o$, where $\mathrm{M}_k{}^o$ is the original cell mass, i.e., at the time of the last global remapping). We would like the rezoning to have a similar property; therefore, assuming that local density is nearly uniform, we set

$$V\dagger_k = M_k^o V_k^l / M_k ,$$

where $V_k{}^l$ is the cell volume as a result of a Lagrangian time-step. Rewriting this, we obtain

$$\delta V_k / V_k = M_k^o (1 + \Delta t (dV/dt)_l / V_k) M_k - 1 , \tag{2.38}$$

which is the expression used for the effective divergence in Eq. (2.37).

Equation (2.37) is a linear Poisson equation. This is in contrast to the nonlinear equations obtained with most other rezoning methods. Because $\delta \mathbf{r}_k$ is known on the boundary, boundary conditions are simple, specified (Dirichlet) conditions. The discretization on the triangular mesh follows straightforward finite element practice, using linear elements [5], by minimizing the functional

$$\mathrm{I} = \int_\Omega [\nabla \delta \mathbf{r}_k \cdot \nabla \delta \mathbf{r}_k + 2 \delta \mathbf{r}_k \cdot \nabla (\delta V_k / V_k)] \, d\Lambda \tag{2.39}$$

with respect to $\delta \mathbf{r}_k$. Using this procedure we obtain two uncoupled matrix equations for δx_k and δy_k, respectively. The matrix so obtained is common to both equations and is symmetric and positive definite. It is therefore suitable for solution by a conjugate gradient-type method. We use the diagonally-scaled conjugate residual method [6] independently in each region, because the regions are decoupled by the boundary conditions.

Given the triangle vertex positions $\delta \mathbf{r}_k$, we linearly interpolate to find the change in location of cell vertices within each triangle. This implies that a cell vertex will always remain at the centroid of its respective triangle, assuming it started at the centroid.

The solution of Eq. (2.37) may produce negative area (inverted) triangles. However, even though such a situation may produce an acceptable computational mesh, inverted triangles will destroy the discretization of Eq. (2.37) on the following cycle. We therefore detect inverted triangles and trigger a general rezone if they are present.

The above rezoning procedure is implemented in subroutine REZLAGR, and the conjugate residual algorithm is implemented in subroutine REZCJRS.

2.4.1.2. General Rezoning Algorithm. The general rezoning algorithm is intended to produce a smooth mesh within specified boundaries. Because we work with a triangulation we cannot directly use established techniques such as the Brackbill-Saltzman [7] rezoning method, which are based on quadrilateral meshes. The use of a triangular mesh, however, gives us an extra degree of freedom in that we are able to change mesh connections, as well as the position of mesh vertices.

We follow the general ideas of the variational method [7] by specifying a composite functional

$$I = I_\theta + \alpha I_\delta , \tag{2.40}$$

where I_θ and I_δ are separate measures of mesh distortion and α is a relative weight that also performs a scaling function.

The functional I_θ measures the departure of the triangle angles from 60°. Referring to Fig. 13, I_θ is defined by

$$I_\theta = \sum_n (|\mathbf{r}_{12}|^2 + |\mathbf{r}_{13}|^2 + |\mathbf{r}_{23}|^2)/A_k ,$$

$$= \sum_n (\cot\theta_1 + \cot\theta_2 + \cot\theta_3) , \tag{2.41}$$

where the summation is over all triangles n, and A_k is twice the area of triangle k: $A_k = \mathbf{k} \cdot \mathbf{r}_{12} \times \mathbf{r}_{13}$. The functional is obviously insensitive to the area of each triangle and is minimized when each angle is 60°. This functional is closely related to the smoothness functional of Brackbill-Saltzman [7].

The functional I_δ measures the departure of the length of the sides of a triangle from equilateral. It is defined by

$$I_\delta = \sum_m |\mathbf{r}_{12}|^2 ,$$

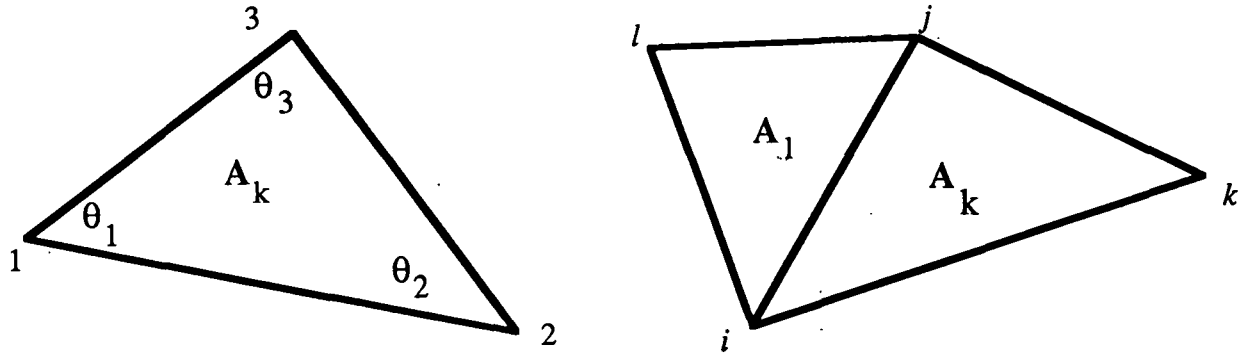


Fig. 13. Notation for the general rezoner.

where the sum is over all triangle sides. This functional measures the "elastic potential" of a system of springs connecting the triangle vertices, and is minimized when the lengths of all triangle sides are equal.

We can combine both functionals in a single summation over triangles as follows

$$I = \sum_n (|\mathbf{r}_{12}|^2 + |\mathbf{r}_{13}|^2 + |\mathbf{r}_{23}|^2)(1/A_k + \alpha/2) .$$

This suggests that the proper scaling may be obtained by

$$\alpha = 2N_t / \sum_n A_k$$

where N_t is the number of triangles. We usually wish independent control over the relative magnitude of the two terms in the functional and so we actually set

$$\alpha = \beta N_t / \sum_n A_k , \tag{2.42}$$

where β is an arbitrary dimensionless factor whose nominal value is 2.

The complete functional (Eq. 2.40) is minimized with respect to $\mathbf{r}_i$, the triangle vertex coordinates. However, the boundary vertices are specified and so the only free parameters are the interior vertices. Because boundary vertices are specified, the different regions are uncoupled. Thus it makes sense to define a different α_r for each region r, where the summation in Eq. (2.42) is now over all triangles in a given region r.

Differentiating the functional with respect to the coordinates of an interior point $\mathbf{r}_i$, and collecting terms associated with a particular side connecting points i and j, we can write the equation associated with point i as

$$\sum_j [C_{ij}(\mathbf{r}_j - \mathbf{r}_i) + D_{ij}\,\mathbf{k} \times (\mathbf{r}_j - \mathbf{r}_i)] = 0 \ ,$$

where

$$C_{ij} = -(1/A_l + 1/A_k + \alpha_r) \ ,$$

$$D_{ij} = (\mathbf{r}_{li} \cdot \mathbf{r}_{ji} + \mathbf{r}_{ij} \cdot \mathbf{r}_{lj} + \mathbf{r}_{il} \cdot \mathbf{r}_{jl})/A_l - (\mathbf{r}_{ki} \cdot \mathbf{r}_{ji} + \mathbf{r}_{ij} \cdot \mathbf{r}_{kj} + \mathbf{r}_{ik} \cdot \mathbf{r}_{jk})/A_k \ , \tag{2.43}$$

and where the subscripts l and k refer to triangles on either side of side ij as illustrated in Fig. 13, and the summation is over all points j which are connected to point i. When written in the above form, Eq. (2.43) represents a matrix system for the vertex coordinates $\mathbf{r}_i = \{x_i, y_i\}$ whose coefficients are 2×2 blocks of the form

$$\begin{bmatrix} C_{ij} & -D_{ij} \\ D_{ij} & C_{ij} \end{bmatrix} .$$

The resulting matrix is symmetric.

Notice that the coefficients C_{ij}, D_{ij} are functions of the coordinates, and therefore, in contrast to the equations of the previous section, Eq. (2.43) represents a coupled nonlinear system for the coordinates x_i and y_i. Consequently, the solution procedure is much more complicated and difficult. Further, Eq. (2.43) is written for a particular mesh topology, and so, as the solution changes, not only the matrix coefficients but also the mesh topology may change. We therefore view Eq. (2.43) as providing information on the direction of the desired changes in $\mathbf{r}_i$ and not necessarily on their magnitude, except in the vicinity of the fixed point or the converged solution.

We summarize the solution procedure as follows:

(1) freeze the coefficients of Eq. (2.43) and take two steps of the same diagonally-scaled conjugate residual algorithm as used in connection with the "near Lagrangian" rezoning algorithm,

(2) retain the direction but limit the magnitude of the step taken so that the maximum triangle side length or triangle area change is less than 20%,

(3) restructure the mesh to obtain the Voronoi connectivity as described in Sec. 2.4.3,

(4) if the mesh does not require reconnection and the maximum step size is below a preset criterion then terminate, otherwise recompute coefficients and start from step (1).

We have found that usually convergence is rapid. However, the algorithm can send points outside the boundary of the mesh, and indeed converge in such a situation. The solution of this problem seems to be to restructure the mesh on the boundary by adding points or reducing point spacing so that interior points are prevented from penetrating. The job of ensuring a satisfactory point distribution on the boundary is handled by subroutine REZBNDY.

The smoothness and regularity of the resulting mesh is sometimes not very good. The use of functional I_θ alone sometimes produces large variations in cell size. Results may be improved by varying the parameter β in Eq. (2.42), but this is an ad hoc procedure. The capability to add or subtract interior vertices clearly improves the situation.

This general rezoning algorithm is implemented in subroutine REZONE.

2.4.2. Boundary Algorithms. Advancement of the interfaces and boundaries that enclose each subdomain (region) in the calculation is provided by a geometric construction (Sec. 2.3). This construction uses the velocities normal to each segment (i.e., the Riemann velocity) to advance the boundary, but not to locate the cell points along the boundary segments. It remains to position the cell points tangentially along the boundary. This is accomplished through the use of either a "near Lagrangian" placement or a global rezone along the interface contour in addition to a configurational rezone to eliminate excessive curvatures if prescribed.

The "near Lagrangian" placement of the boundary points is intended to move the points in a manner that preserves the mass between the points in an effort to minimize advection across the cell sides that intersect the boundary. The approach used for this purpose is equivalent to the method that keeps the interface propagation algorithm well defined along symmetry boundaries (Sec. 2.3). However, because the interfaces are now known, the problem is one-dimensional in the arc length (s). The boundary points are placed at locations s_k, that minimize the variational functional

$$I_L = \sum_m \omega_m \left(\Delta s_m^2 \right)^2 , \tag{2.44}$$

where $\Delta s_m = s_k - s_{k-1}$. The weight ω_m is defined as $(\bar{\rho}/\bar{\rho}_0 \Delta s_0)_m$, where $\bar{\rho}_m = \frac{1}{2}(\bar{\rho}_k + \bar{\rho}_{k-1})$, $\bar{\rho}_0$ and Δs_0 are the initial average density and boundary segment length obtained following the last global rezone, and the sum is taken over the boundary segments which lie between two "fixed" points. Minimizing the functional in Eq. (2.44),

$$\frac{\partial I_L}{\partial s_k} = 2\{\omega_m(s_k - s_{k-1}) - \omega_{m+1}(s_{k+1} - s_k)\} = 0 , \tag{2.45}$$

results in a tri-diagonal system for s_k. This system of equations is solved using the Thomas algorithm [14]. Solutions (s_k) to Eq. (2.45) are bounded by the values of s_{k-1} and s_{k+1} from the previous time-step to ensure that a Courant-like condition is not violated.

An advantage of the general topology feature of CAVEAT-GT is the ability to add cells in regions demanding better resolution and eliminating cells where they no longer are required. Along the boundary, this amounts to placing boundary points where they are most needed. This is accomplished by calculating the point distribution parameter N, which is the solution to the ordinary differential equation

$$\frac{dN}{ds} = f(s, \kappa, \nabla\phi, \ldots) , \tag{2.46}$$

where f is the point distribution density function. This function can be chosen to equally distribute the boundary points along the interfaces in the absence of any distinguishing features. However, the point density function also should force boundary points to migrate into regions that require increased resolution. Such regions may be characterized by large values of the boundary curvature (κ) or the gradient ($\nabla\phi$) of a prescribed variable such as pressure (i.e., adaptivity). The point density function has been defined at each boundary point as

$$f \equiv \frac{1 + \alpha_\kappa (\Delta s\, \kappa)_k + \alpha_a \left| \frac{\Delta s \nabla\phi}{\phi} \right|_k}{1 + \alpha_{\kappa_{max}} (\Delta s\, \kappa)_k + \alpha_{a_{max}} \left| \frac{\Delta s \nabla\phi}{\phi} \right|_k} \frac{1}{\Delta s_{max}} , \tag{2.47}$$

where κ is the boundary curvature, defined by inscribing a circle through the k^{th} boundary point and its two neighbors, Δs_{max} is a user supplied maximum spacing for each region of the problem, and α_κ and α_a are user supplied constants. Finally, $\alpha_{\kappa_{max}}$ and $\alpha_{a_{max}}$ are defined as

$$\alpha_{\kappa_{max}} \equiv \frac{\Delta s_{min}}{\Delta s_{max}} \alpha_\kappa ,$$

and (2.48)

$$\alpha_{a_{max}} \equiv \frac{\Delta s_{min}}{\Delta s_{max}} \alpha_a ,$$

where Δs_{min} is a minimum spacing set internal to the code as a fraction of Δs_{max}. The above formulation equally spaces the boundary points at intervals of Δs_{max} in the absence of curvature or gradients, i.e.,

$$\lim_{\substack{\kappa \to 0 \\ \nabla\phi \to 0}} \frac{dN}{ds} = \frac{1}{\Delta s_{max}} . \tag{2.49}$$

Furthermore, in regions of large curvature the boundary points are more closely spaced to provide better resolution, i.e.,

$$\lim_{\kappa \to \infty} \frac{dN}{ds} = \frac{1}{\Delta s_{min}} . \tag{2.50}$$

A result identical to Eq. (2.50) applies to regions of large gradients for the variable ϕ.

To ensure that the boundary points are placed at identical locations along the contours defining interfaces, it is necessary that the point density functions are identical along the doubly defined contours bounding each region. Consequently, at multiply defined points along the boundaries, the maximum value of f is used.

Finally, the values of f should be smooth along the boundary contours. This ensures a smooth variation for the locations of the boundary points around the interfaces. Without smoothing, a large variation for the point spacing would result where regions of large curvature intersected lines of symmetry (i.e., across "fixed" points), for example. Smoothing is accomplished by solving the diffusion equation

$$\frac{df}{dt} = c^2 \frac{d^2 f}{ds^2} , \tag{2.51}$$

where $c^2\,\Delta t = 10\,\Delta s_{max}{}^2$. Equation (2.51) is differenced using an implicit, centered differencing technique. A tri-diagonal system results that is solved using the Thomas algorithm [14].

Having defined the point density function, Eq. (2.45) now is integrated along contours between "fixed" points. A linear distribution for f is assumed between boundary points. The resulting values of N are then scaled to ensure the final value of $N(s=L)$ is an integer; that is, the positions of "fixed" boundary points are not altered. In addition, the value of $N(s=L)$ is limited to guarantee that no more than one point is added on each boundary segment, nor fewer than every other point is eliminated ($N^n - int(N^n/2) \le N^{n+1} \le 2N^n - 1$).

Equation (2.45) is solved for $N(s)$ around the boundary contours every time-step. It is then tested to determine if the existing boundary point distribution is sufficient to resolve the existing boundary configuration. If point addition or deletion or gross point migration is unnecessary, boundary point locations resulting from the "near Lagrangian" description are used. If the "near Lagrangian" positions are inadequate to resolve the boundary, the boundary points are located according to Eq. (2.45) and a global rezoning of the mesh is required. When a global remap is performed there is no need to limit the relative motion of the boundary points.

In an effort to control excessive boundary oscillations derived from either physical or numerical effects, the option to smooth the boundary contour has been included. The configurational rezone of points along the boundary is accomplished by computing an approximate radius of curvature at each point. The radius of curvature (R) is obtained by constructing a circle through each point and its nearest neighbors (Fig. 14). If the radius of curvature is less than a minimum radius of curvature (R_{min}), the boundary point (x_2) is displaced a fraction (a_n) of the distance between its current location and the midpoint between its two neighboring points ($x_0 = \frac{1}{2}|x_2 - x_1|$):

$$x_2' = x_2 + a_n |x_o - x_2| \,. \tag{2.52}$$

Both R_{min} and a_n are user specified.

The algorithms to rezone the boundaries and interfaces are implemented in subroutine REZBNDY.

2.4.3. Mesh Restructuring

2.4.3.1. Interior Mesh Restructuring. When cell points are moved, as in the general rezoning algorithm, a mesh reconnection or restructuring can be used to produce a more regular mesh. A mesh restructuring produces a new triangulation, or alternatively, cell points change neighbors. It is therefore a mesh topology change. This provides additional freedom for the rezoning algorithm in its attempt to improve the mesh. We have chosen to use the connectivity associated with the Voronoi mesh [12]. The corresponding triangula-

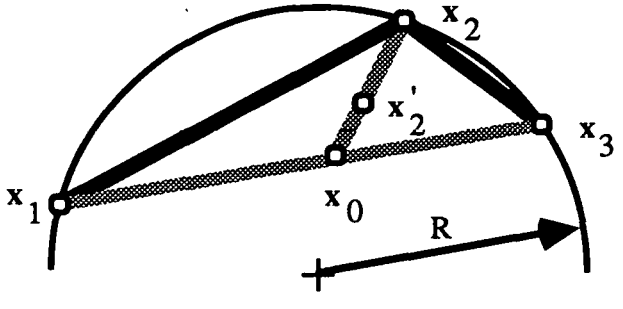


Fig. 14. Configuration rezone.

tion is called the Delaunay triangulation [13]. The Voronoi mesh is a unique subdivision of space into convex polygonal cells (in two dimensions) associated with an arbitrary distribution of cell points. The associated dual Delaunay triangulation has the following property: the diagonal of every quadrilateral formed by two neighboring triangles connects the two vertices whose angles sum to more than 180°. We use this property in an iterative procedure to generate the required connectivity quickly and very efficiently.

Consider Fig. 15 which represents the two triangles that are associated with side m as the diagonal. We form the quantity

$$K_m = \mathbf{k} \cdot [(\mathbf{r}_{24} \cdot \mathbf{r}_{14})(\mathbf{r}_{13} \times \mathbf{r}_{23}) + (\mathbf{r}_{13} \cdot \mathbf{r}_{23})(\mathbf{r}_{24} \times \mathbf{r}_{14})] , \tag{2.53}$$

which can be rewritten as

$$K_m = |\mathbf{r}_{24}||\mathbf{r}_{14}||\mathbf{r}_{13}||\mathbf{r}_{23}| \sin(\theta_3 + \theta_4) . \tag{2.54}$$

Thus,

$$\theta_3 + \theta_4 > 180° \text{ iff } K_m < 0 . \tag{2.55}$$

Hence, if $K_m < 0$ we wish to switch diagonals. When diagonals are switched, the "Z" data structures associated with sides m, $m1$, $m2$, $m3$, and $m4$ are restructured.

We anticipate that in many cases no restructuring will be necessary. We therefore organize the algorithm into two parts. First, sweep over all interior sides in a fast vectorizable loop and evaluate criterion Eq. (2.55) to determine if any sides need to be switched. We also note the first and last side in the loop that needs to be switched. Since switching a side may affect the status of other sides, therefore, once any side is switched all the affected sides need to be rechecked. Hence, once we find a side that needs to be switched then all subsequent sides are checked and switched as needed. Thus, if there is at least one side

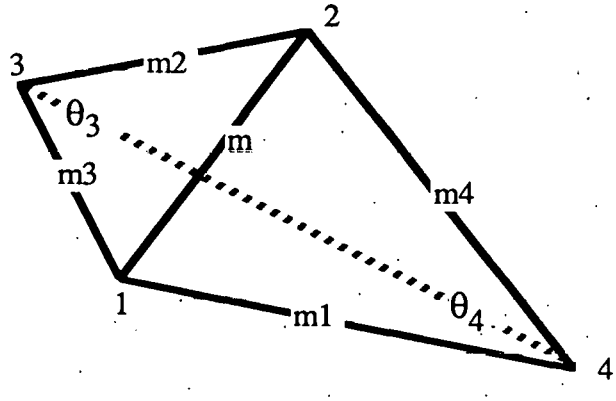


Fig. 15. Mesh restructuring by the switching of diagonals.

that requires to be switched we repeat the sweep, and switch sides as required, except that we start the sweep either in the forward direction from the first side needing to be switched, or in the reversed direction from the last side needing to be switched, the choice being determined by which case requires the fewest sides to be considered. The procedure is repeated until no more sides need to be switched. We have found in practice that convergence is very rapid, and the method always works, provided that the starting triangulation is regular (i.e., provided there are no side intersections).

When the parameter K_m is equal to zero then both diagonals are acceptable. This is a degenerate case that occurs in rectangular meshes, for example. In order to eliminate a lot of random flipping in such a case, we modify the criterion Eq. (2.55) to $K_m < -\varepsilon$, where ε is a very small number.

This restructuring algorithm is implemented in subroutine MESH.

2.4.3.2. Boundary Point Addition/Deletion. Boundary point addition/deletion consists of adding or deleting cell points on the boundary (external and interface) segments of the triangular mesh. Once the triangular mesh has been modified, the local data structure for the modified part of the mesh is updated. The number of points added is limited by the size of the data structure arrays. The deletion process is limited by the "fixed points" that are the boundary segment end points. The method for determining the location of the added/deleted cell points is discussed in Sec. 2.4.2.

Boundary point addition consists of adding a cell point on a boundary side midway between two existing points as shown in Fig. 16. Because no interior points are added, only one new boundary and one new interior side are produced. The new interior side is stored as the last interior side that causes the boundary sides to be shifted by one. The new boundary side is then stored as the last side. If the original boundary side is on an interface, then a point also is added on the opposite side of the interface.

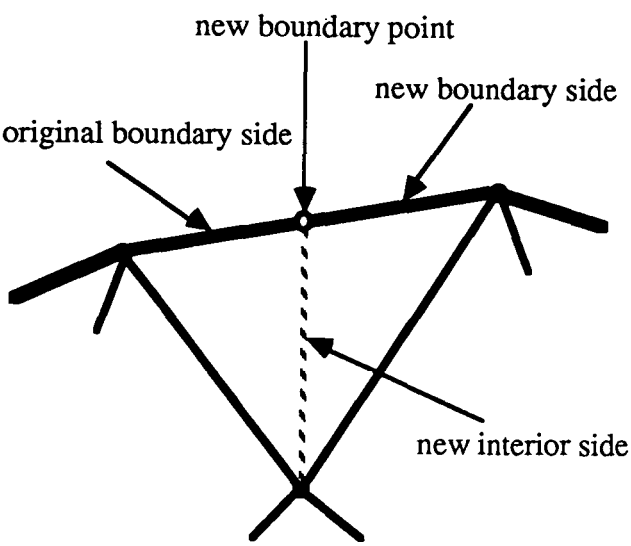


Fig. 16. Boundary point addition.

The deletion of a boundary point is a more difficult problem because there can be any number of connections with internal points. The case with no internal connections is shown in Fig. 17. In this case both adjacent boundary sides are removed. The one internal connection case is shown in Fig. 18. In this case, as well as the following cases, one boundary and one interior side are deleted. The remaining boundary side is reconnected as shown in Fig. 18. The two interior connection case is shown in Fig. 19. Here again one boundary and one interior side are deleted, while now the remaining boundary side and an interior side are reconnected. The three interior connection case is shown in Fig. 20. Here again one boundary and one interior side are subtracted, while now the remaining boundary side and two interior sides are reconnected.

For cases with more than three interior connections, the number of connections is systematically reduced to three connections. This is accomplished by reconnecting one interior side as shown in Fig. 21. This process is continued until the number of interior connections is three and the point then is deleted. One could continue this interior side reconnection until the number of interior connections is zero and then delete the point. This would simplify the coding in that the one, two, and three interior connection cases would not be needed. However, the no interior connection case requires approximately the same amount of computational effort as the other cases. This is because two sides are removed as in the other cases and a former interior side becomes a boundary side, which requires shifting the data in the same manner as deleting an interior side does. Therefore, reducing the number of interior connections one at a time until there are no interior connections and

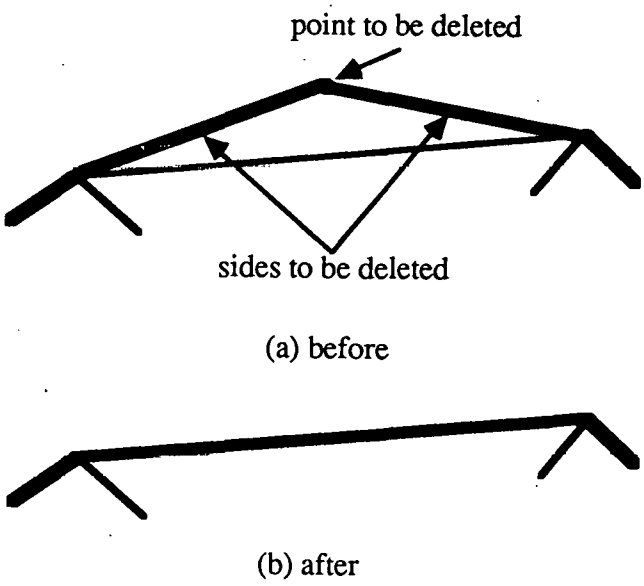


Fig. 17. Boundary point deletion with no internal connections.

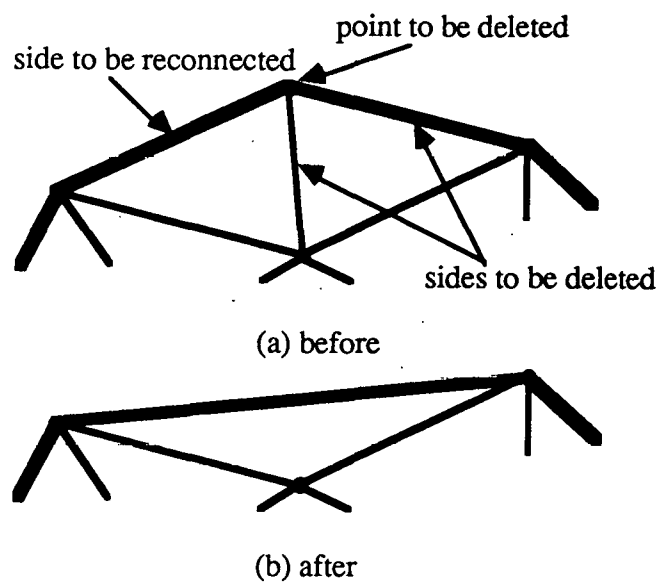


Fig. 18. Boundary point deletion with one internal connection.

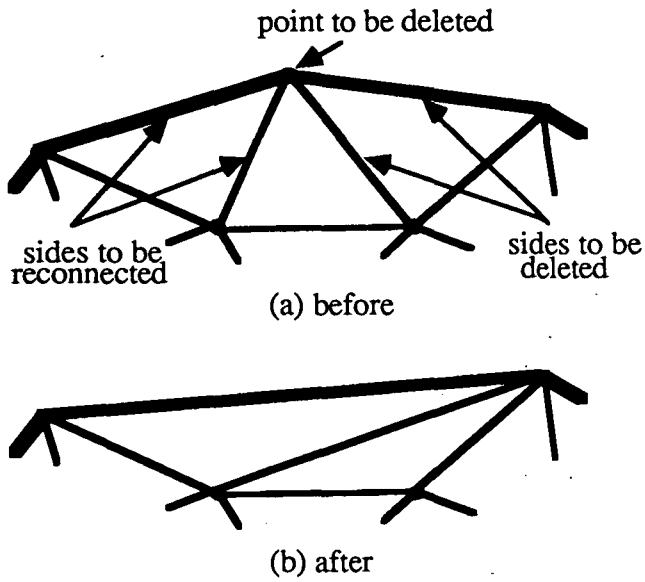


Fig. 19. Boundary point deletion with two internal connections.

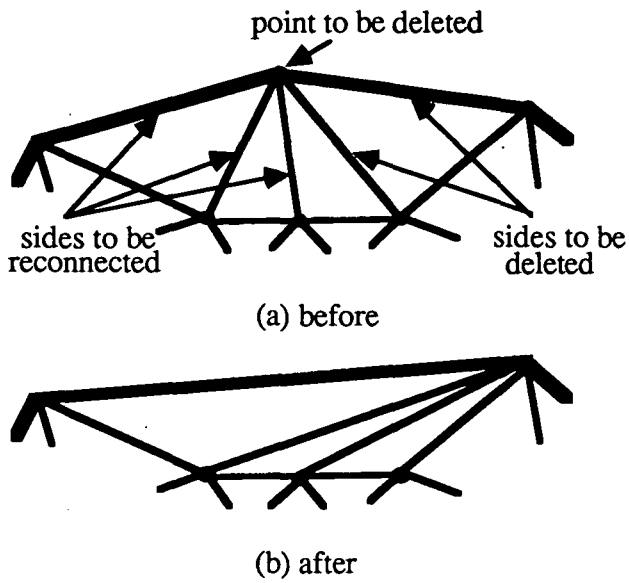


Fig. 20. Boundary point deletion with three internal connections.

then deleting the point would be computationally slower than using the special cases. Therefore, it was decided to directly delete the cases more likely to occur, and reduce and then delete the cases that rarely occur in order to improve the code efficiency.

2.4.3.3. Interior Point Addition/Deletion. Interior point addition/deletion consists of adding or deleting cell points to the interior parts of the triangular mesh. Once the triangular mesh has been modified, the local data structure for the modified part of the mesh is updated. The number of points added is limited by the size of the data structure arrays. The deletion process is limited to cell points with exactly five side connections. This five connection case was selected because of the observation that whenever a group of small triangles formed in the mesh, they were typically centered around a cell point with five connections.

Interior point addition consists of adding a cell point on an interior side midway between two existing points (Fig. 22) when the length of that side becomes larger than the average side length for that region by a user specified amount. This produces three new interior sides. These new interior sides are stored as the last interior sides, which causes the boundary sides to be shifted by three.

Interior point deletion occurs only for cell points with five side connections as discussed above, and when one of the connected sides' length becomes smaller than the average side length for that region by a user specified amount. As shown in Fig. 23, the cell point deletion causes 3 interior sides to be deleted and two sides reconnected. Because the three sides are deleted some of the interior and all boundary sides are shifted by three.

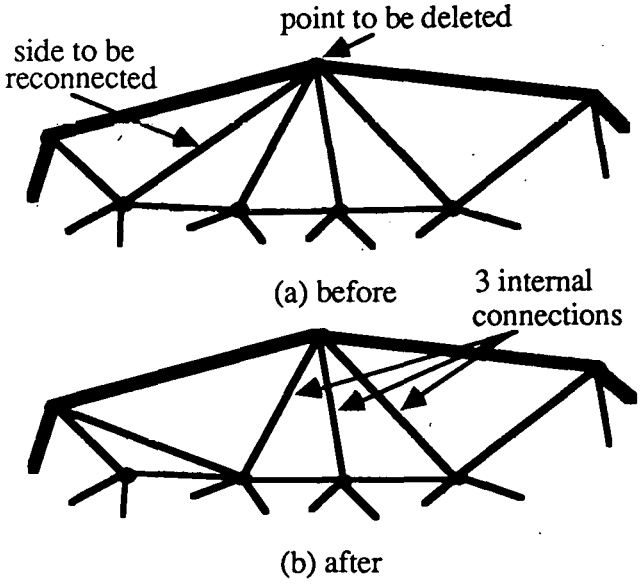


Fig. 21. Boundary point deletion with four or more internal connections.

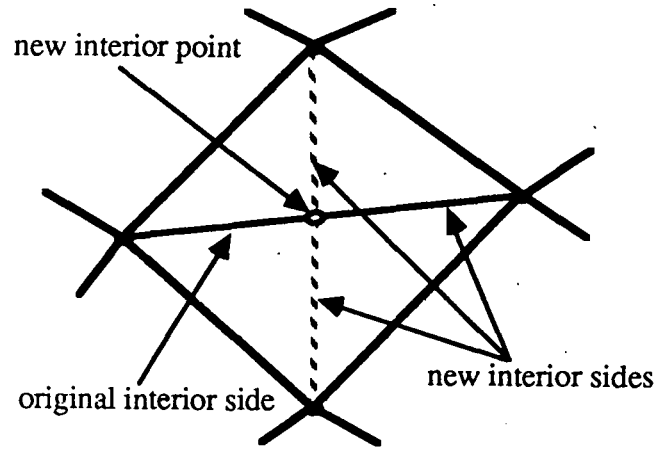


Fig. 22. Interior point addition.

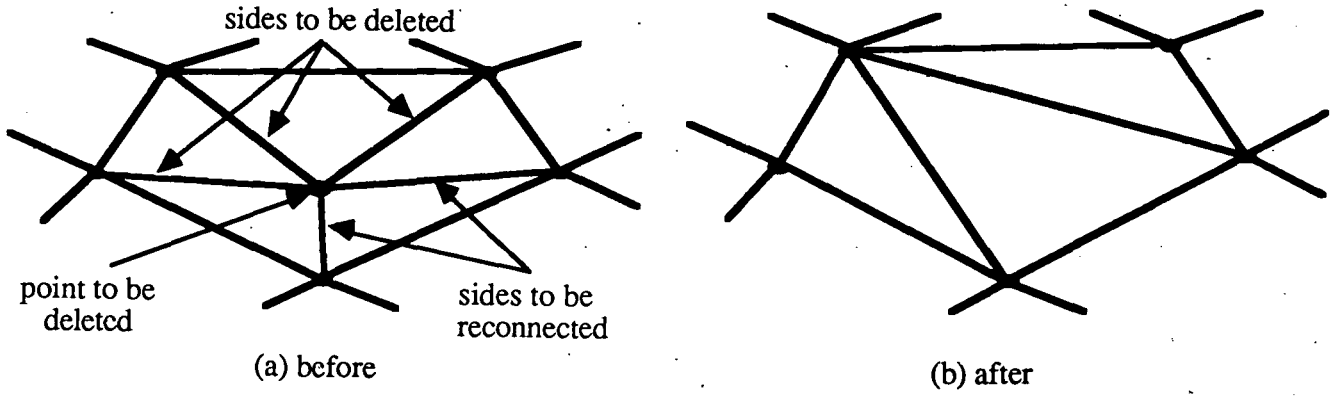


Fig. 23. Interior point deletion with five connections.

Both the boundary and interior cell point addition/deletion are implemented in subroutine NEWMESH. Once all cell point and side data structure has been updated, all triangle data structure is recalculated in subroutine NEWTRI.

2.5. Remapping

In order to accurately resolve material interfaces and to prevent nonphysical mixing and shear impedance, CAVEAT-GT follows the boundaries in a Lagrangian fashion. The normal material velocities are used to advance region boundaries. However, in the interior there will be, in general, a difference between the material velocity and the velocity used to move the mesh. In other words, the new mesh will differ from the Lagrangian

mesh. It is necessary, therefore, to remap the variables onto the new mesh for each time-step. If the difference between the material and mesh velocities is small enough to maintain the stability of the numerical algorithm, then the variables may be advected to the new mesh. Otherwise, a general remapping scheme must be employed. CAVEAT-GT contains both an advection and a general remapping algorithm.

2.5.1. Advection. Consider the intensive-variable $\rho\phi$ (i.e., momentum). The remapping of this variable by advection from its position following a pure Lagrangian motion to its position on the new mesh is determined from the equation

$$\frac{d}{dt}\int \rho\phi\, dV = -\oint \rho\phi(\mathbf{u}_f - \mathbf{u}_g)\cdot\mathbf{n}\, dS\ .$$

In discrete form, for each cell,

$$(\rho\phi)^* = (\rho\phi) - \frac{\Delta t}{V}\sum_m [<\rho\phi>(\mathbf{u}_f - \mathbf{u}_g)\cdot\mathbf{n}<R>\ell]_m\ , \tag{2.56}$$

where $\rho\phi$ is the appropriate intensive quantity (i.e., mass, momentum, and energy) following a pure Lagrangian motion from the original grid (x^n) to the new grid (x^{n+1}). The variable $(\rho\phi)^*$ is the intensive quantity at the new mesh position. The variables $\mathbf{u}_f$ and $\mathbf{u}_g$ are the fluid and grid velocities, respectively. Finally, $\mathbf{n}$ is the unit outward normal to the cell side ℓ, and $<R>$ is the pseudo-radius.

The normal fluid or material velocity ($\mathbf{u}_f\cdot\mathbf{n}$), in Eq. (2.56), is simply the cell side normal velocity obtained from the Riemann solution (w^*). The quantity $\Delta t\ \mathbf{u}_g\cdot\mathbf{n}<R>\ell$ is the volume swept out by the motion of the grid (Fig. 24). It may be obtained as the sum of two triangular areas

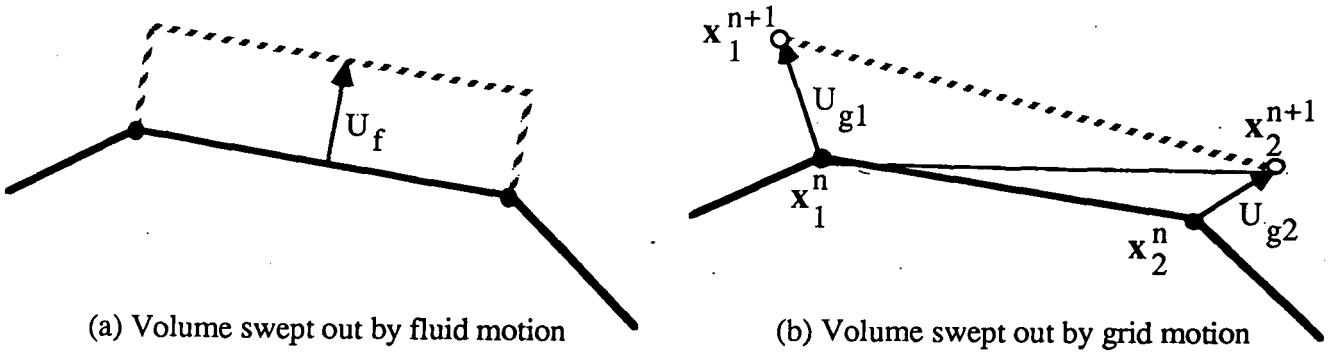
$$\Delta t\ \mathbf{u}_g\cdot\mathbf{n}<R>\ell = \frac{1}{2}\{(x_1^n - x_2^n)(y_2^{n+1} - y_2^n) - (y_1^n - y_2^n)(x_2^{n+1} - x_2^n)\}<R_L>$$

$$+\frac{1}{2}\{(x_2^{n+1} - x_1^{n+1})(y_1^n - y_1^{n+1}) - (y_2^{n+1} - y_1^{n+1})(x_1^n - x_1^{n+1})\}<R_u>\ , \tag{2.57}$$

where $<R_L> = (1-\beta) + \beta\frac{1}{3}(x_1^n + x_2^n + x_2^{n+1})$, with a similar expression for $<R_u>$.

The difference between the volumes swept out by the material velocity and the grid velocity is the advection volume

$$V_f = \Delta t\,(\mathbf{u}_f - \mathbf{u}_g)\cdot\mathbf{n}<R>\ell\ . \tag{2.58}$$



(a) Volume swept out by fluid motion

(b) Volume swept out by grid motion

Fig. 24. Advection volumes.

The quantity $<\rho\phi>$ in Eq. (2.56) is the average value of the intensive variable $(\rho\phi)$ fluxed across the cell boundary. Its value is determined from the upstream side of the cell side ℓ (i.e., donor cell). The upstream side is determined by the sign of the advection volume (V_f). If the advection volume is positive, then the k^{th} cell loses mass. That is, the relative material motion is out of the k^{th} cell and $<\rho\phi>$ is determined by the k^{th} cell. If the advection volume is negative, then the k^{th} cell gains mass; the relative motion is into the cell and $<\rho\phi>$ is determined by the neighboring cell.

Once the side of the cell boundary (ℓ) has been determined then $<\rho\phi>$ may be determined from a Taylor expansion about the cell centroid

$$<\rho\phi> = (\rho\phi)_o + \nabla_o(\rho\phi)\cdot(\mathbf{x}_m - \mathbf{x}_o)\,, \tag{2.59}$$

where the quantity $(\rho\phi)_o$ is a cell centered quantity (i.e., evaluated at the cell centroid) and $\mathbf{x}_m$ is the center of the cell boundary segment (ℓ). A spatially first order approach (IORDER=1) is obtained with zero values for the gradient in Eq. (2.59) [i.e., $<\rho\phi> = (\rho\phi)_o$]. This method is very diffusive. An improved advection scheme is obtained when nonzero gradients are used in Eq. (2.59) (IORDER=2 or 3). The methods available for determining the gradients are discussed in Sec. 2.2.3. The least diffuse technique is obtained when no limiting is used to obtain the gradient (IGRAD=1). However, this option tends to produce numerical oscillations.

The above advection algorithm is implemented in subroutine ADVECT.

2.5.2. General Remapping. General remapping refers to the process of transferring conserved information from one arbitrary mesh to another. This procedure is related to the process of advection, except that there is no time-step limitation and there is no restriction on the topology of the meshes. In general, we use this type of remapping when the mesh topology changes, either because of the addition or deletion of points or because of a mesh reconnection.

The techniques of this type of remapping for quadrilateral meshes have been well developed [8,9,10,11]. For the purposes of CAVEAT-GT, however, we have had to develop new techniques that allow for general topology meshes. In general, the method seeks to compute

$$Q_k = \int_{V_k} q(\mathbf{r})\, dV \,, \tag{2.60}$$

where $q(\mathbf{r})$ is the known distribution on the old mesh of a generic conserved quantity such as ρ, $\rho\mathbf{u}$, or ρE, and V_k is the volume of cell k of the new mesh. The problem, then, is to compute such integrals in the presence of arbitrary overlapping of the cells of the two meshes. This problem is simplified by converting the volume integrals to surface integrals:

$$Q_k = \int_{S_k} \mathbf{F}\cdot\mathbf{n}\, dS \,, \tag{2.61}$$

where $\nabla \cdot \mathbf{F} = q_l(\mathbf{r})$ in each old cell l, and S_k is the surface of new cell k. We generally restrict $q_l(\mathbf{r})$ to be at most linear within a cell:

$$q_l(\mathbf{r}) = q_l + \mathbf{G}_l\cdot(\mathbf{r} - \mathbf{r}_l) \,, \tag{2.62}$$

where $\mathbf{r}_l$ is the centroid and q_l is the average value of q in cell l, and $\mathbf{G}_l$ is the gradient of q in that cell, obtained as described in Sec. 2.2.3. Incorporating the case of cylindrical symmetry in a quasi-Cartesian coordinate system, we can write in general

$$\nabla\cdot\mathbf{F} = A + \mathbf{B}\cdot\mathbf{r} + \mathbf{r}\cdot\mathbf{CD}\cdot\mathbf{r} \,,$$

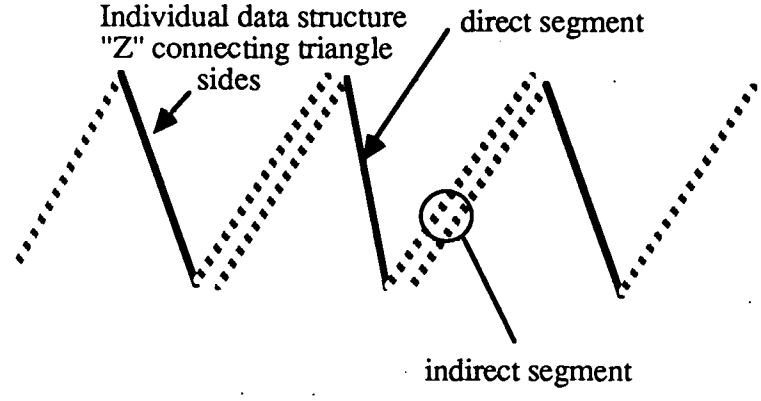
where A, $\mathbf{B}$, $\mathbf{CD}$ are cell dependent quantities. The flux function $\mathbf{F}$ is not unique and many choices are possible, but the following is the simplest and most natural [11]:

$$\mathbf{F} = (1/2\, A + 1/3\, \mathbf{B}\cdot\mathbf{r} + 1/4\, \mathbf{r}\cdot\mathbf{CD}\cdot\mathbf{r})\mathbf{r} \,.$$

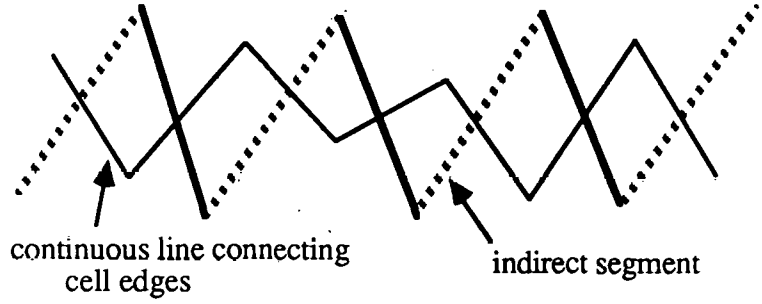
In a two-dimensional mesh the integration of Eq. (2.61) includes a sweep over the cell edges of both the old and the new mesh. The sweep over the edges of the old mesh is necessary because in general the flux function $\mathbf{F}$ will be discontinuous at the cell edges and it is necessary to subtract out the contributions of these discontinuities. One great advantage of quadrilateral meshes is that the cell edges form mesh lines that extend right across the mesh. It is therefore possible, starting from one boundary of the mesh, to trace the mesh lines of one mesh continuously through the other mesh, thus eliminating the need for a

search to locate the points of one mesh within the cells of the other. Such a search is obviously very costly if the meshes are not regular. For the general topology mesh of CAVEAT-GT it is not obvious that a corresponding procedure exists. It is fortunate that one can indeed define continuous and unique lines through a triangulation that, taken together, cover the mesh with no gaps (in this case, they cover the mesh twice over). These lines are either continuous closed curves or they begin and terminate at the boundaries.

Consider the "Z's" described in Sec. 2.1.3. First, we note that to each of the triangle sides that compose the arms and body of the "Z" there corresponds a cell edge. We designate the body of the "Z" as the "direct" segment and the arms as the "indirect" segments. Putting neighboring "Z's" arm-to-arm, as illustrated in Fig. 25, creates a continuous path through the mesh. Because of the one-to-one relationship between triangle sides and cell edges, this also creates a continuous path along cell edges. Notice that the direct segments exactly alternate with the indirect segments. Combining all such paths that intersect a given triangle, as in Fig. 26, we see that after tracing all such lines, each triangle side (cell



(a) Connecting "Z"s to form a continuous path



(b) Corresponding diagram illustrating continuous path connecting cell edges

Fig. 25. Constructing a continuous path along the cell edges of the CAVEAT-GT general topology mesh.

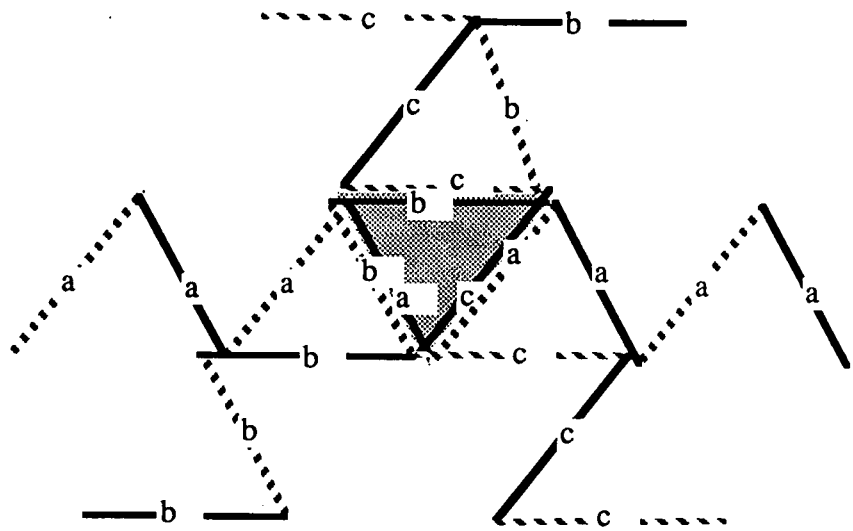


Fig. 26. Tracing through the mesh: For a typical triangle, all sides are traced exactly twice, once by a direct segment and once by an indirect segment.

edge) in the mesh is traced exactly twice, once by a direct segment and once by an indirect segment. We choose to update the integral (Eq. 2.61) only when tracing the direct segment. Thus, while sweeping over cell edges of a mesh we go over each edge twice but we update only once.

We start by tracing all the lines that originate (and therefore terminate) at the boundaries. Since each edge is traced twice, we set a flag on each boundary edge when it has been traced, both as a direct segment and as an indirect segment. When both flags on each boundary segment have been set, then we know that all lines starting and terminating at boundaries have been traced. For a topologically regular mesh, such as a mesh of quadrilaterals or hexagons, we know that this would be exhaustive. For the more general meshes of CAVEAT-GT we have found that in the great majority of cases this also exhausts all such lines. However, it is possible in such a mesh, as we have found very infrequently, that there are continuous closed loops in the interior of the mesh that do not intersect the boundary, and therefore would not be traced starting from boundary sides. To detect these loops we increment a counter whenever a side is traced. If this count does not equal twice the total number of sides after all the boundary-intersecting lines have been traced, then at least one internal loop must exist. We then locate a starting side on such a loop and proceed to trace along the loop until the trace closes on itself. We proceed in this way until all sides have been counted twice.

The process of tracing a cell side of one mesh through a cell of the other mesh is very similar to the corresponding procedure in a quadrilateral mesh [10]. All intersections of the straight line corresponding to the cell side being traced with the straight lines corresponding to the extensions of all the sides of the cell being traversed are checked to find the

"legal" intersections. These are the intersection points that lie in the interior of both sides. For a convex cell there will be at most two such intersection points. If there is only one, then the line terminates or originates within the cell, and if two, then the line crosses the cell. The side being crossed determines the new cell that is being entered by the traversing line. In CAVEAT-GT, we have made an improvement that reduces by approximately a factor of two the number of intersections that need to be checked. Because we know the tangential vector to the traversing line in the direction being traced (t) as well as the vector pointing in the outward normal direction to the side being checked (n) then we need only check that t points to the outside of the cell (i.e., $\mathbf{t} \cdot \mathbf{n} > 0$) to determine if an intersection needs to be found. This is substantially cheaper than computing and checking the actual intersection.

One feature of the CAVEAT-GT remapper is that we have arranged that both meshes exactly coincide along the boundaries and interfaces of the old mesh. This means that there will be no artificial losses or gains due to an inaccurate representation of the old boundary by the straight line segments of the new boundary. However, this also means that potentially a cell side of the new mesh along a region boundary could be composed of a number of straight line segments. This would considerably complicate the logic of finding the exiting intersection for such a cell. Fortunately, we are able to circumvent this problem by checking interior sides only for intersections, since if no such intersection is found then this means that the traversing line must terminate at the boundary in that cell.

Previously [8-11], it was always assumed that mesh cells were convex. This limitation is too restrictive for the meshes anticipated with CAVEAT-GT. We therefore found it necessary to generalize the method to handle concave cells. Since concave cells are relatively infrequent we do not wish to perform the extra work involved if it is not necessary. We therefore check both meshes and set a flag if the mesh contains concave cells. If the mesh does contain concave cells, then intersections in a concave cell are checked by going around the sides of the traversed cell both in the clockwise and anticlockwise direction. Aside from the entering intersection, this will find two possible exiting intersections. In most cases these intersections will coincide. However, when there are multiple re-entries, the intersections will differ and the one closer to the entering point is selected. This is illustrated in Fig. 27.

A known difficulty with this algorithm is the problem of coincidences between segments of the old and new mesh [9-11]. Coincidences require special treatment and they must be detected reliably on both the old and the new mesh. We sidestep this problem by destroying all potential coincidences by randomly displacing the cell vertex points of the new mesh. Since each vertex point is associated with a triangle, we determine the radius of the inscribed circle for scaling the displacement, and then randomly perturb the coordi-

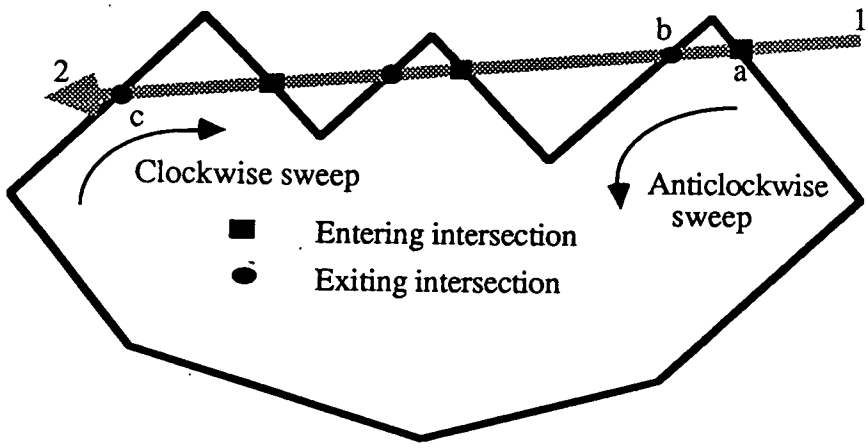


Fig. 27. Tracing a line through a concave cell: Entering at point (a), point (b) is found on the anticlockwise sweep and point (c) on the clockwise sweep. Point (b) is selected.

nates of the vertex point within a square whose dimensions are a very small fraction of the inscribed circle radius.

This algorithm is implemented in subroutine REMAPPER. Line segments are traced through cells of the old and new mesh in subroutines REMOLDSG and REMNEWSG, respectively, and the line integral contributions are evaluated in subroutine REMLNINT.

2.6. Equation of State

The equation of state package in the CAVEAT-GT code is the same one that is in the CAVEAT code. It consists of seven analytic and one tabular equations of state. The analytic equations of state are Linear, Quadratic (Osborne), Gamma Law, Stiffened Gas, Mie-Gruneisen (HOM), Becker-Kistiakowsky-Wilson (BKW), and Jones-Wilkins-Lee (JWL). The tabular equation of state is the Los Alamos National Laboratory (LANL) SESAME tables. For a further discussion of these equations of state, see the CAVEAT report [1].

2.7. Setup Code

The setup code was designed to be an interface between the user and the CAVEAT-GT code. The typical user is assumed to have a good knowledge of the problem he wishes to solve, but not a detailed knowledge of the CAVEAT-GT code. The setup code leads the user through each phase of setup in an interactive fashion. Upon completion of the setup, an input file for the CAVEAT-GT code is written. The first and most important part of the setup is specifying the geometry, boundary conditions, and generating a suitable initial triangular mesh.

The geometry is input by first making a simple sketch as shown in Fig. 28. In Fig. 28, the geometry is specified by the smallest number of straight or curved line segments (sides) that adequately describe the geometry. All sides and vertices (points where two

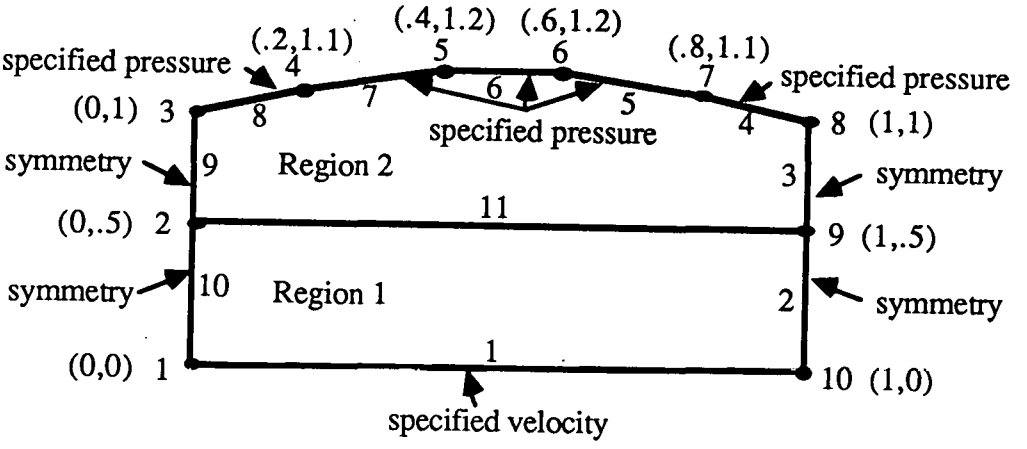


Fig. 28. Input geometry.

adjacent sides intersect) are numbered consecutively beginning with the number one. The ordering of the sides and vertices is arbitrary. The desired boundary condition on each side and the x and y location of each vertex is specified and each separate region is numbered. Extra regions can be specified to force a particular mesh point distribution and later removed. Regions containing holes must be initially treated as two half regions and not as a single region with a "cut." This is a requirement of the setup code only. Once the mesh has been generated, this extra region can be removed. The maximum number of regions that can meet at a vertex is three. After the above information has been organized in the sketch, the user can run the setup code that will request this information in a straightforward manner. The user can vary the maximum mesh size and triangle area smoothness until a satisfactory mesh is obtained.

Namelists CNTRL, PLOT, and GRID contain the parameters for controlling the calculation itself as well as input and output. These Namelists are written into the CAVEAT-GT code input file using all default values for each parameter. The definition of all parameters in these Namelists can be listed by the setup code if desired. The default values can be changed in the CAVEAT-GT code input file using a text editor.

The initial conditions then are specified for each region. While these conditions may vary from region to region they are constant inside each region. If specified pressure or velocity boundary conditions are employed, the values to be specified are requested.

All input required for a given problem is generated by the setup code except for desired changes to the Namelists CNTRL, PLOT, and GRID discussed above and desired changes to the equation of state (EOS) information. The EOS information is input using the same Namelist format as in the CAVEAT code. A default Namelist for a gamma law gas ($\gamma = 1.4$) is provided. If the user wishes a different EOS then he must prepare the Namelist EOSIN using the CAVEAT code report [1] and then replace the default EOSIN.

Note that because there is no burn model in CAVEAT-GT, the array BURNC in Namelist EOSIN is not used. Note also that the CAVEAT-GT codes use the SI system of units and not those used in CAVEAT.

In conclusion, the CAVEAT-GT code setup requires running the setup code and then making any desired changes to the parameters in the Namelists CNTRL, PLOT, and GRID as well as the EOS parameters.

2.8. Graphical Output

The graphical output consists of mesh plots (both triangular mesh and computational cells); velocity vector plots; contour plots of pressure, density, and energy; plots of boundary values of pressure, density, and energy vs x or y; and special mesh plots providing the local data structure (Sec. 3.5). These special mesh plots are used for debugging purposes. All plots are generated using the DISSPLA library of subroutines.

3. COMPUTER PROGRAM

3.1. Data Structure, Storage, and Masking

The "Z" data structure, discussed in Sec. 2.1.3, and other integer mesh parameters necessary to efficiently describe the mesh, require a large amount of storage. In order to decrease this storage, these integer quantities have been packed into arrays using the concept of masking. In this procedure, one first decides how many bits of a 64 bit word are required to store all expected values of a particular integer parameter. Next, one allocates this number of bits in a 64 bit word for storing this integer parameter by masking off the rest of the 64 bits. One can then move the mask to allow other integers to be stored in the same 64 bit word and thus greatly reduce the total amount of storage. A discussion of the parameters, the masks, and how to save/get these quantities is provided in the sections below. In what follows, the k integers denote the cell points or vertices of the triangular mesh, the m integers denote the sides of the triangular mesh, and the n integers denote the triangles themselves. The storage partitions for all data structure arrays that use masking are shown in Fig. 29.

3.1.1. Interior. The arrays **ns1** and **ns2**, shown in Fig. 29, store the "Z" data discussed in Sec. 2.1.3 and, therefore, are based on triangle sides. The integers **k1**, **m1**, and **n1** are stored in **ns1** while **k2**, **m2**, and **n2** are stored in **ns2**. The k integers use the rightmost 20 bits, the m integers use the next 20 bits to the left and the n integers use the next 20 bits to the left of the m integers. The integers in the **ns2** array are stored in the same manner. The leftmost 4 bits in the **ns1** array are used to store the boundary condition index. There are **nsd** total sides and **nsdi** interior sides or "Z's." The **nts** array is used to store the three side numbers of each triangle. Side **m1** uses the rightmost 20 bits, side **m2**

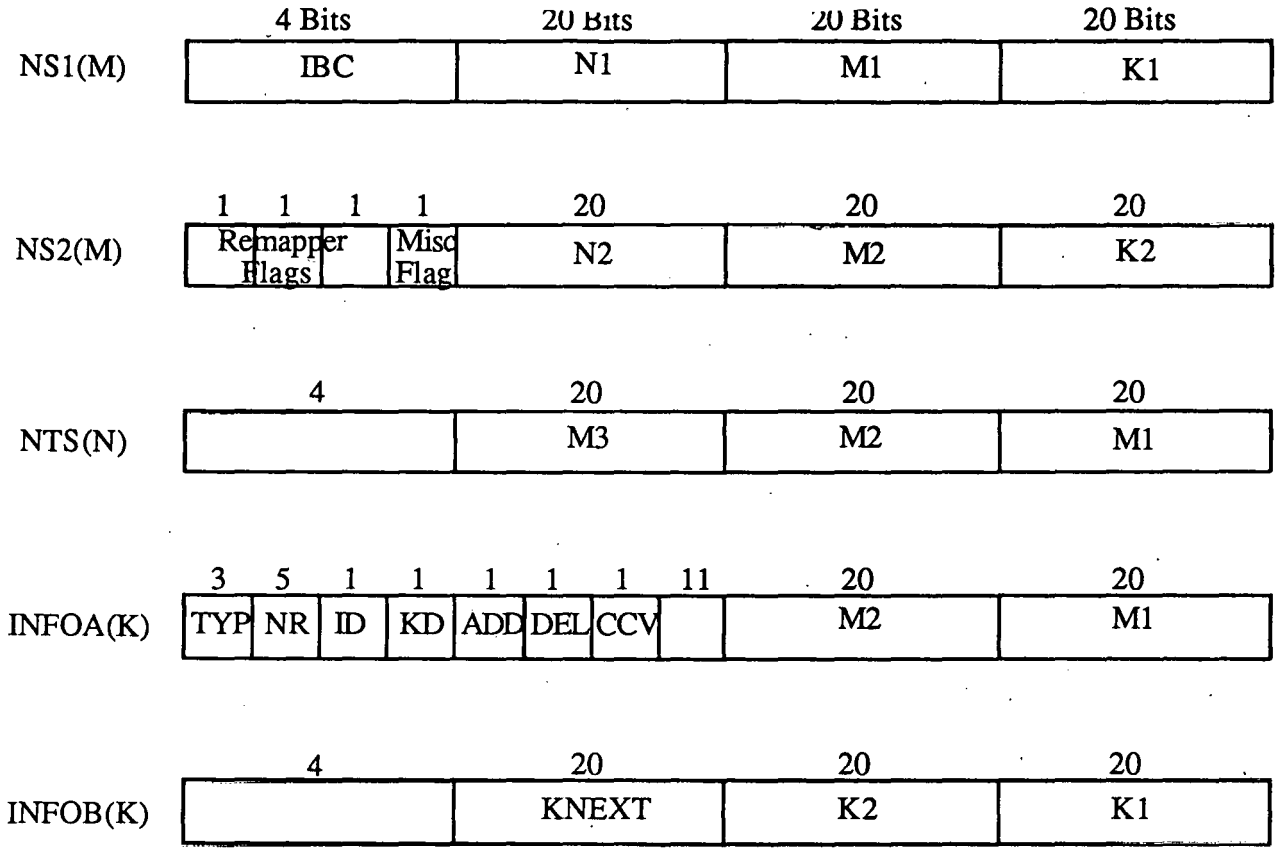


Fig. 29. Masking locations of the data structure arrays.

uses the next 20 bits to the left, and **m3** uses the next 20 bits to the left of **m2**. There are **ntr** total triangles and **ntri** interior triangles. The location of the cell points, which are the triangular mesh vertices (**k1** and **k2**, discussed above), is stored in the **xc** (x value) and **yc** (y value) arrays. The fluid cell vertex points, which are the triangle centroids, are stored in the **xv** and **yv** arrays. The locations of the cell and vertex points after the Lagrangian phase of a time cycle are stored in the **xcl**, **ycl**, **xvl**, and **yvl** arrays, respectively. The centroids of the fluid cells (indexed by the **k1** and **k2** values discussed above) are stored in arrays **xcd** and **ycd**. These values are the same as **xc** and **yc** if the cells are regular polygons, and then only in the interior. For the boundary cells, the **xc** and **yc** values always lie on the boundary. There are **np** cell points.

The data structure parameters may have different names in different subroutines and, therefore, listing them by name would not be very useful. However, the mask that specifies the storage location of each parameter does not change and, therefore, can be used to identify the parameters. The masks are scalar variables whose names start with MASK and are stored in common block MASKS. Corresponding to each mask is a shift count that tells how many bit positions the mask must be shifted so that the right-most bit of the parameter resides in the lowest order bit of a computer word. These shift counts are not stored as variables, but instead are coded directly in all masking statements.

Following is a list of the data structure parameter masks, the bit positions used for storage and a brief description of the parameter. For this discussion the bits are numbered from the right beginning with one.

MASK	BITS	DESCRIPTION
MASK1	1-20	Used to obtain **k1** from **ns1**, **k2** from **ns2**, **m1** from **nts**, **m1** from **infoa**, and **k1** from **infob**.
MASK2	21-40	Used to obtain **m1** from **ns1**, **n2** from **nts**, **m2** from **infoa**, and **k2** from **infob**.
MASK3	41-60	Used to obtain **n1** from **ns1**, **n2** from **ns2**, **m3** from **nts**, and **knext** from **infob**.
MASKBC	61-64	Used to obtain the side boundary condition index from **ns1**.
MASKNR	57-61	Used to obtain the cell point region number from **infoa**.
MASKTYP	62-64	Used to obtain the cell point type as shown in Fig. 28.

To obtain a parameter

An example of this procedure for the case where a shift is not required is the following:

```
K1 = NS1(M).AND.MASK1.
```

This gives one of the k values for side m. An example of this procedure for the case where a shift is required is the following:

```
M1 = SHIFTR(NS1(M).AND.MASK2,20).
```

To store a parameter

To store the above two parameters requires the following:

```
NS1(M) = K1.OR.(NS1(M).AND..NOT.MASK1)
```

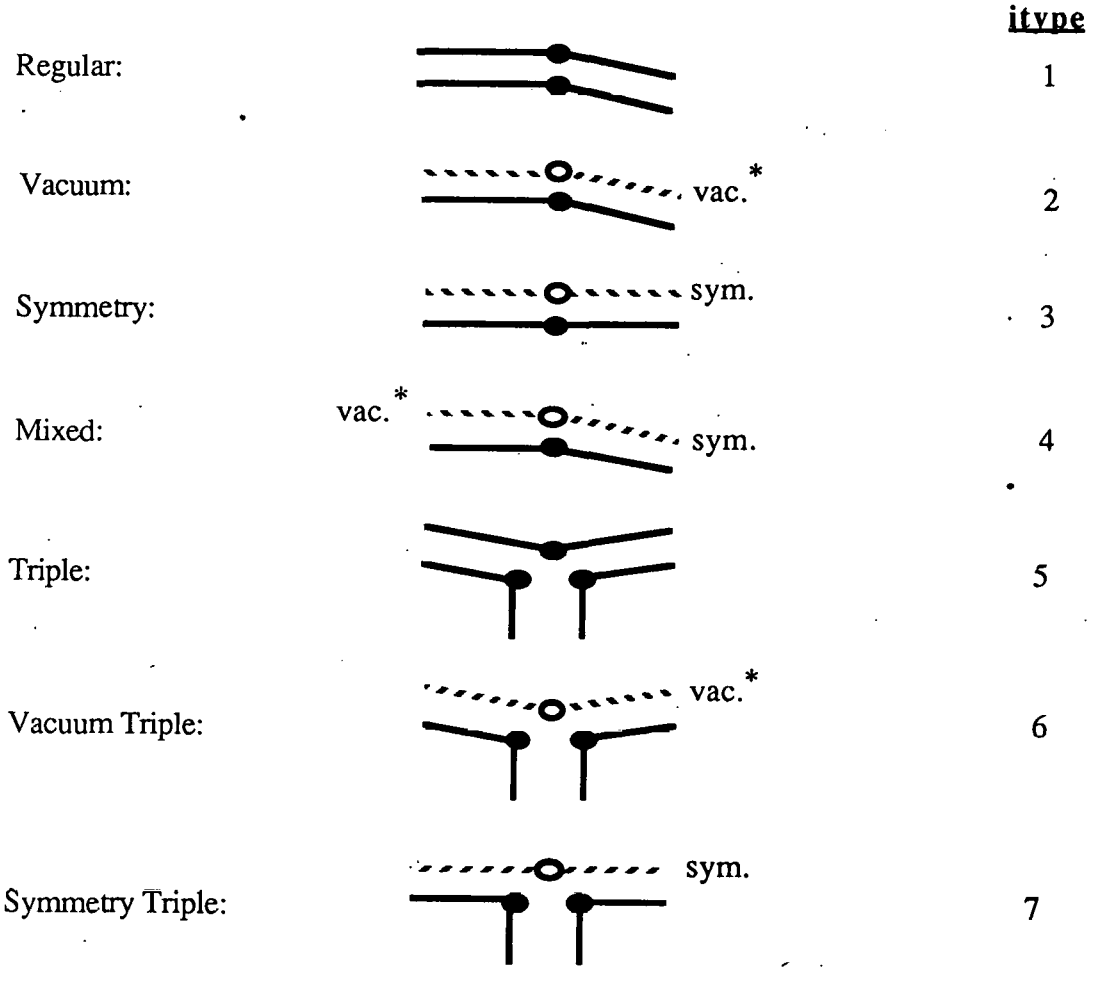
and

```
NS1(M) = SHIFTL(M1,20).OR.(NS1(M).AND..NOT.MASK2).
```

3.1.2. External/Interface Boundaries. On boundary sides ($m >$ **nsdi**), the middle side and both cell points of each "Z" lie on the boundary. One side is in the mesh interior while the other side is external to the mesh. If the boundary is an external boundary then the m value for the external side is zero. If the boundary is an interface boundary then the

m value is equal to the m value of the opposite side. The n value always is equal to the boundary triangle number (discussed below). There are no Z's defined external to the mesh. However, there are boundary triangles ($n >$ **ntri**) that are external to the mesh. One of the m values in the **nts** array is equal to the side on the boundary while the remaining two m values are zero for external and interface boundaries.

Two additional arrays called **infoa** and **infob**, shown in Fig. 29, also are used in defining the boundary data structure. These arrays are defined for cell points (k values). The **m1** and **m2** values in **infoa** are the two adjacent boundary sides to cell point k. The **kd** and **id** integers are flags that are discussed in the next section. The **nr** integer is the region number and the **itype** integer denotes the seven possible types of interface/boundary points, shown in Fig. 30. The **k1** and **k2** integers in **infob** are the other two cell points



* also specified velocity or pressure

Fig. 30. Types of interface/boundary points.

(**itype**=5), shown in Fig. 30, that occupy the same location in physical space as point k. If only two points are present (**itype**=1, 6, or 7), then one of the k's will be zero. If only one point is present (**itype**=2, 3, or 4), then both k's are zero. The **knext** integer is the k value of the next point on the boundary in the direction of increasing k. (If k is not on the boundary then **knext** equals zero.) **knext** is used in subroutine BSTRUCT to generate the **kl** array that is a linked-list of cell points around the boundaries beginning with the first region and continuing for all regions. Points that begin and therefore end a region appear twice in the **kl** array. The **ldbl (nr,i)** array tells which element of the **kl** array begins (**i**=1) and ends (**i**=2) region **nr**.

3.1.3. Flags. The term flag refers to single bit integers that can take on values of either 0 or 1. All but three of these flags are associated with cell points and are stored in the **infoa** arrays. The remaining three are associated with sides of the triangular mesh and are stored in **ns2**.

The following is a list of the flag masks, the bit position used for storage and a brief description of the flag use. For this discussion the bits are numbered from the right beginning with one. All flags are stored in **infoa** except as noted.

FLAG MASK	BIT	FUNCTION
maskadd	54	When nonzero, this flag denotes that a cell point has been added.
maskdel	53	When nonzero, this flag denotes that a cell point has been deleted.
maskccv	52	When nonzero, this flag denotes that a fluid cell is concave. This information is required by the remapper because the new mesh may enter and exit a concave cell in the old mesh more than once.
maskid	56	When nonzero, this flag denotes that a specific cell point operation has been completed.
maskkd	55	Same as **maskid**.
maskrev	51	When nonzero, this flag denotes that the ramp as opposed to the table part of the SESAME equation of state package is being used to calculate the pressure. For more information see the CAVEAT report [1].

mask61	61(ns2)	When nonzero, this flag denotes that a triangle side has been checked for intersection with a contourline in subroutine PLTCONTR.
mask62	62(ns2)	When nonzero, this flag denotes that a specific triangular mesh side operation has been completed in the general remapper subroutines.
mask63	63(ns2)	Same as **mask62**.
mask64	64(ns2)	Same as **mask62**.

3.1.4. Regions/Materials. The region number of each cell point is stored in the **infoa** array using the mask **masknr** discussed above. The material number for this region is obtained from the **nrmn** array in common block EOS. These material numbers must begin with one and increase by one for each different material. These numbers are not the SESAME equation of state material numbers. Several regions may have the same material and, therefore, the same material number. The ordering of the regions and materials is arbitrary. The maximum allowable number of different regions is 31 while the maximum allowable number of materials is 30.

3.2. Code Structure

The CAVEAT-GT code package consists of the following nine files: the setup code source file, the run code source file, the compilation controller file, along with three update and three input files for the three example cases. Information pertinent to understanding the structure and use of the CAVEAT-GT code is contained in this section while the example cases are discussed in Sec. 4. Included here are the naming conventions of the codes, files, subroutines, and variables. Variable definitions and subroutine functions also are described. The organization of the CAVEAT-GT code is provided. Finally, working array assignments are detailed.

3.2.1. File Naming Conventions. Early in the CAVEAT-GT development effort, it was recognized that three distinct codes were necessary to provide a user convenient and efficient method (Fig. 31). The three codes include a setup code (CAVGTS), the main code (CAVGTR), and a post-processor (CAVGTP). The setup code (CAVGTS) is a user interactive code that generates the initial mesh, flow variables, equation of state parameters, and data structure (Sec. 2.7). Output from the setup code is available as input to the main code (CAVGTR). The main code contains the hydrodynamics algorithms. Because of the greater geometrical complexity, graphical output may be expensive. Furthermore, the need to manipulate data provided by the main code suggests the need for a post-processor (CAVGTP). Output from the main code could be used directly as input to the post-

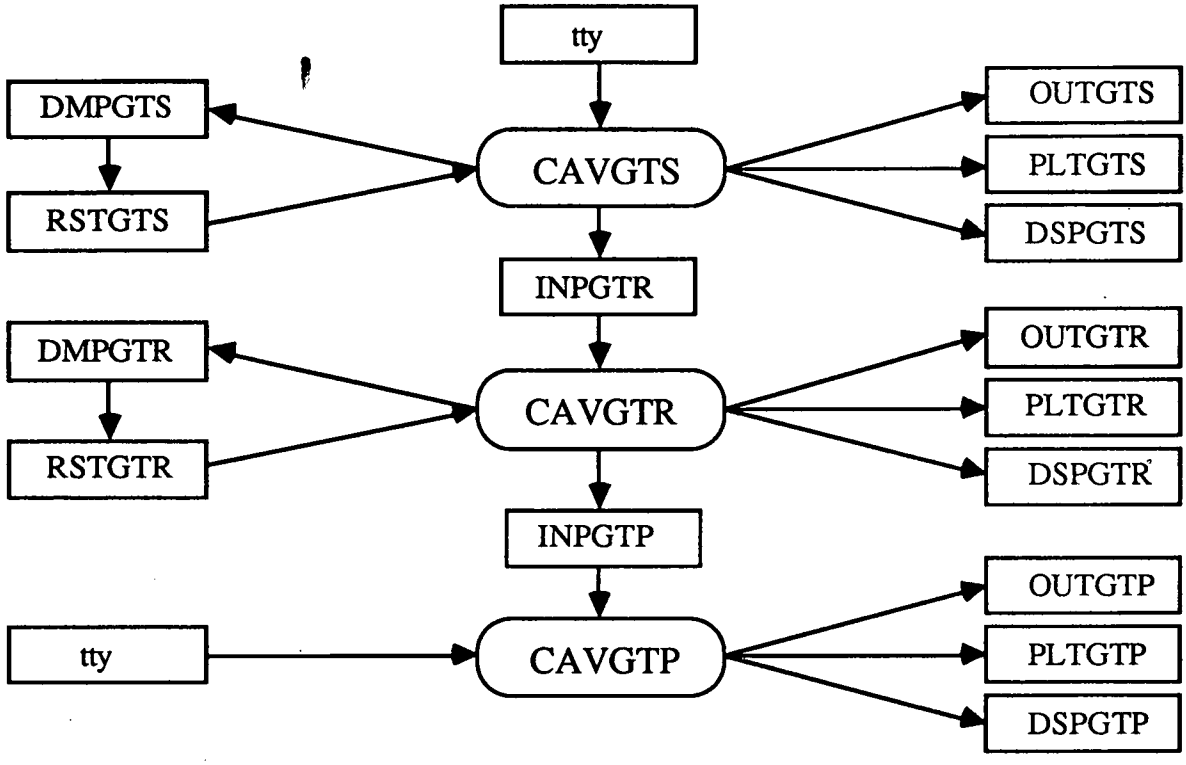


Fig. 31. File naming convention.

processor. The post-processor should also be a user interactive code. Its fundamental function would be to alter data from the main code and provide the desired graphics. At the present time, the post-processor code has not been written [Sec. 5.3].

A file-naming convention was conceived in an effort to easily identify the file type and its code affiliation. Files associated with the setup, main, and post-processor codes have a GTS, GTR, and GTP designation, respectively. Input, output, dump, restart, plot, and DISSPLA message files begin with INP, OUT, DMP, RST, PLT, and DSP, respectively. The executable files begin with CAV. Finally, the source files are named CGTSSRC, CGTRSRC, and CGTPSRC, respectively. A summary of the above convention is provided in Fig. 31. It should be observed that the dump files (DMPGTx) must be renamed (RSTGTx) to restart each of the codes. Output from setup code is used as input to the main code (INPGTR). Output from the main code could be used as input to the post-processor (INPGTP). Currently, OUTGTS contains the z-data structure information for the problem, and OUTGTR contains messages that also are sent to the user terminal (tty).

3.2.2. Flow Diagram. Flow diagrams for the subroutine calling sequence of the CAVEAT-GT computer program are provided in Figs. 32-35. The diagrams are intended to illustrate where in the computational cycle each subroutine is used and not to provide a detailed diagram of the logic inherent to a cycle. Descriptions of the function of each subroutine are found in Sec. 3.2.3.

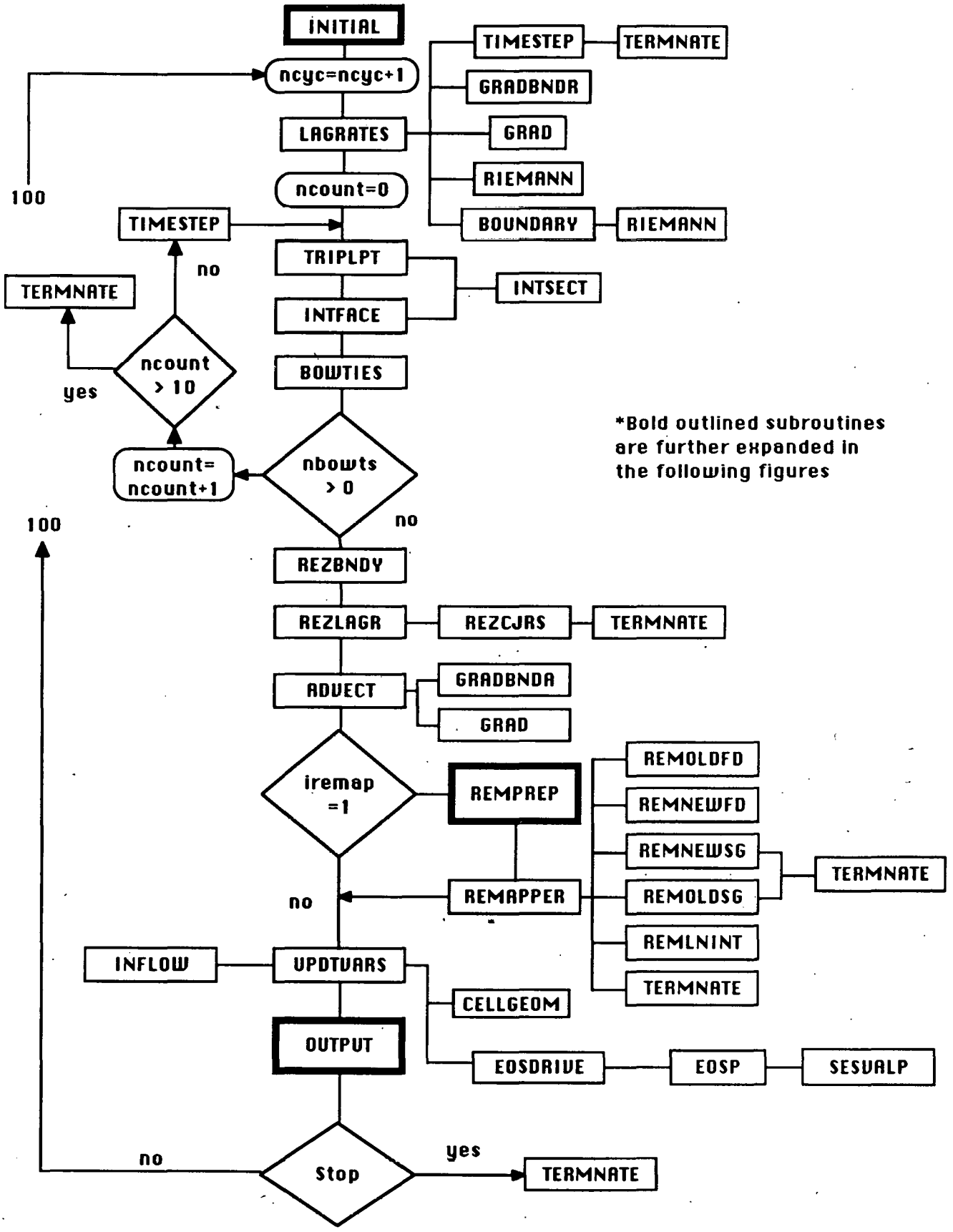


Fig. 32. CAVEAT-GT flow diagram.

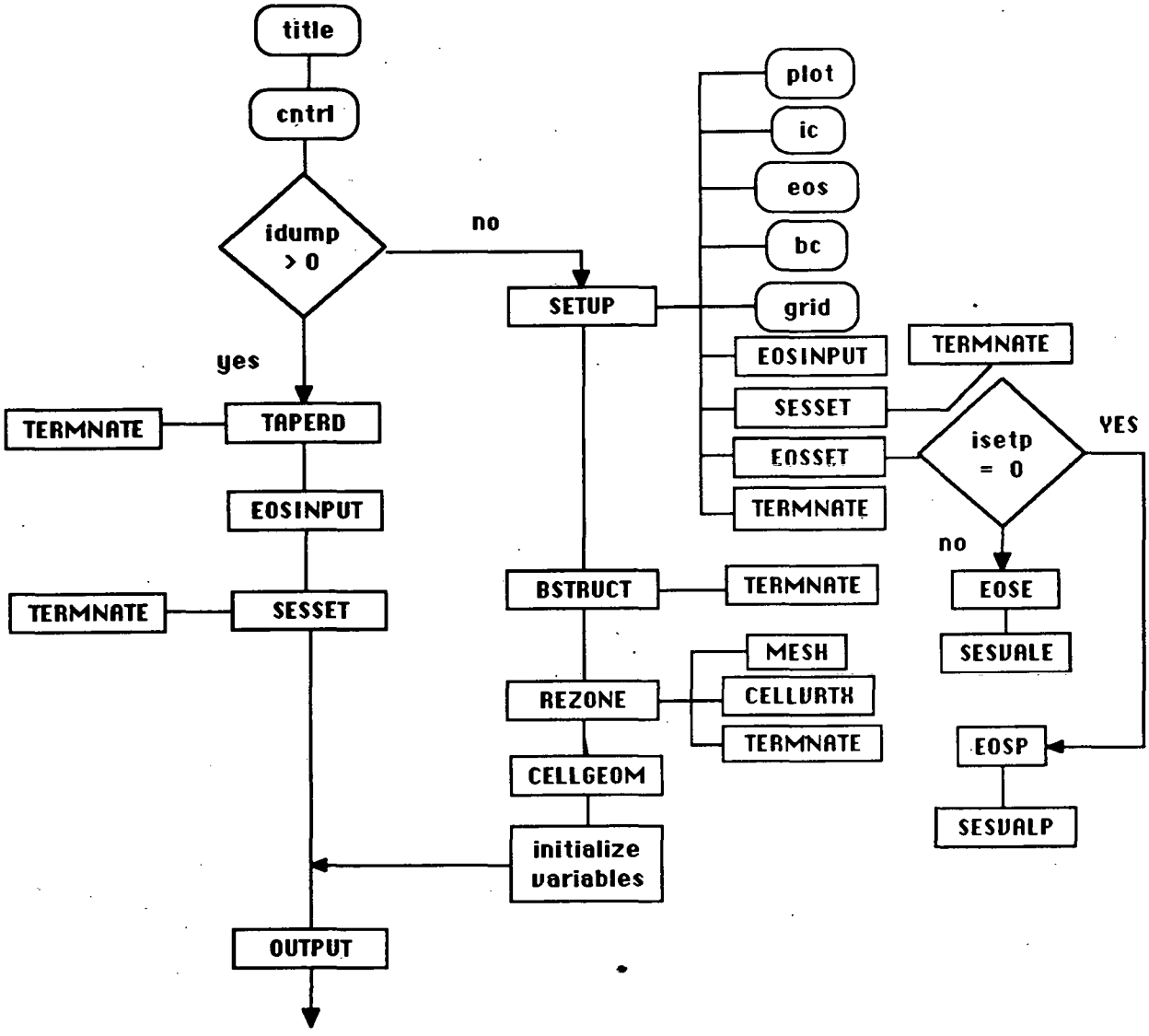


Fig. 33. Subroutine INITIAL.

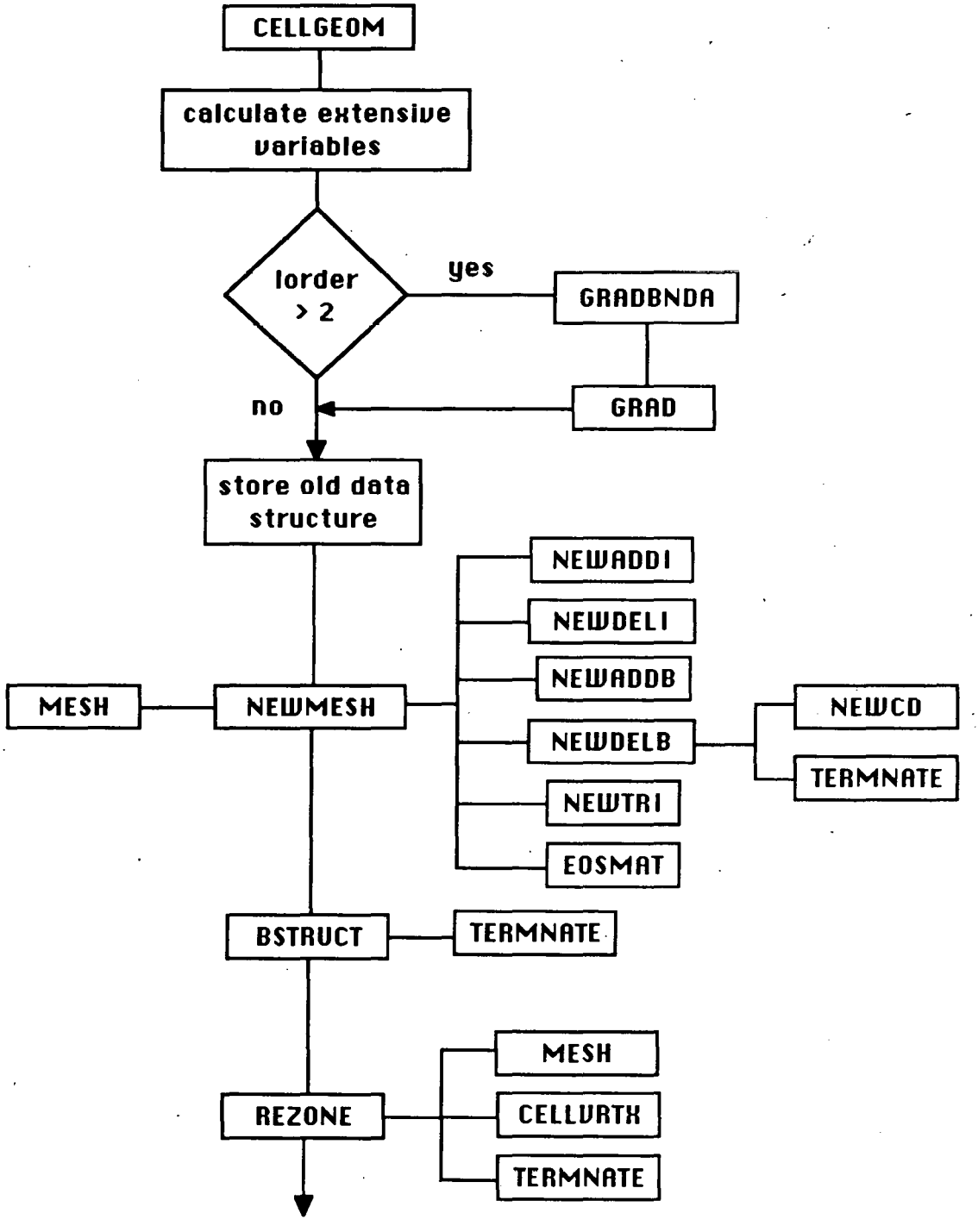


Fig. 34. Subroutine REMPREP.

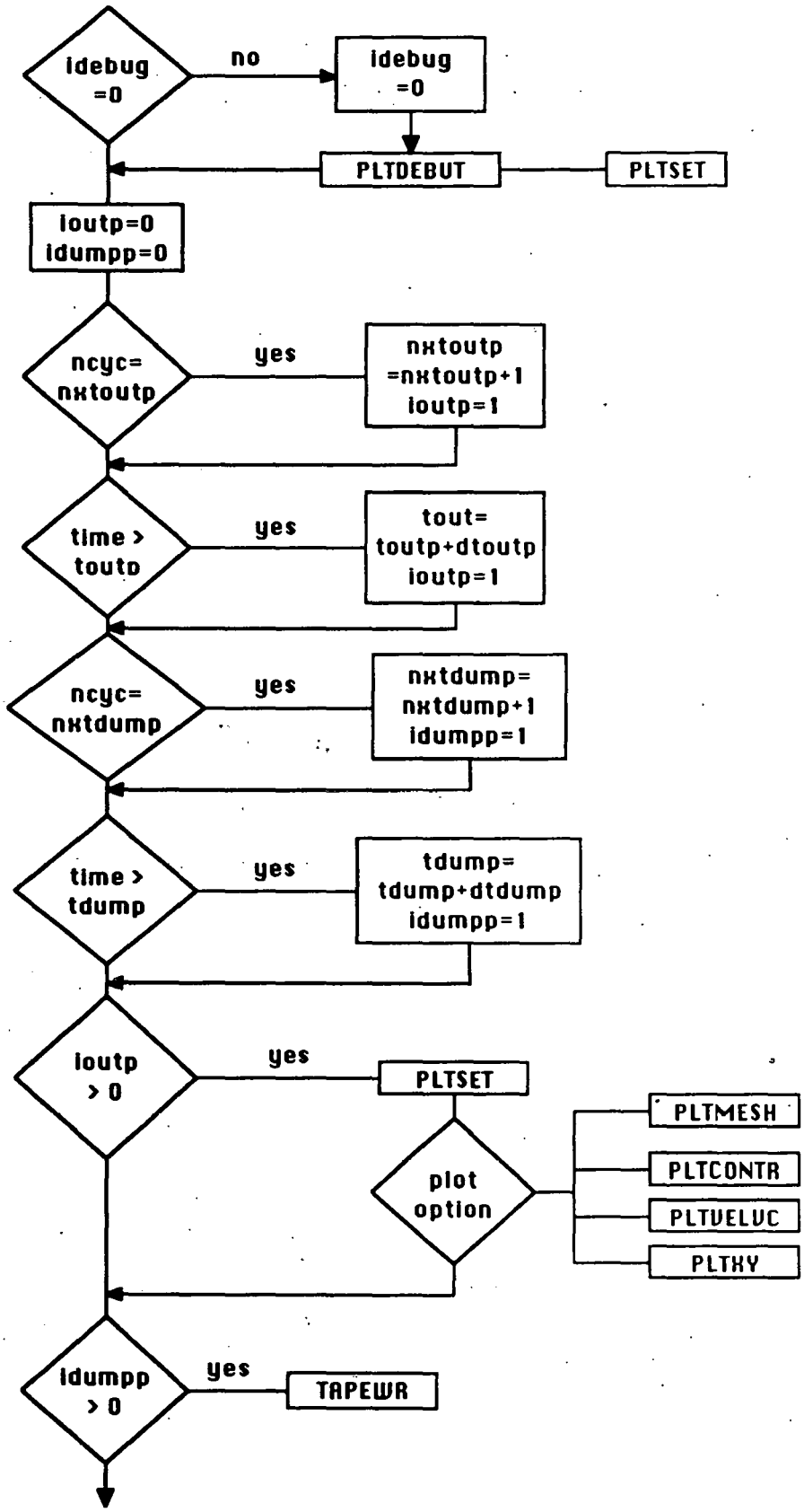


Fig. 35. Subroutine OUTPUT.

The fundamental architecture of a CAVEAT-GT cycle is provided in Fig. 32. The cycle is initiated by obtaining cell side variables (u^*,p^*) by solving a Riemann problem. These variables are used to obtain the surface integrals necessary to compute the time rates of change of volume, momentum, and energy (LAGRATES). Fixed points and boundaries and interfaces are then advanced (TRIPLPT and INTFACE). Boundary vertices are displaced along the boundary segments in an effort to provide optimal resolution and boundary contours with excessive curvature are smoothed (REZBNDY). The interior mesh is rezoned in a near Lagrangian fashion (REZLAGR) and the flow variables are advected to this new mesh (ADVECT). If it has been determined that a global remap is necessary, the data structure is modified and a new mesh is constructed (REMPREP). The flow variables then are remapped onto this mesh (REMAPPER). The variables are updated (UPDTVARS) and output is provided (OUTPUT).

The calling sequences of the subroutines INITIAL, REMPREP, and OUTPUT are provided in Figs. 33, 34, and 35, respectively.

3.2.3. Subroutine Description. Subroutine names in CAVEAT-GT are selected to reflect their function. Furthermore, the CAVEAT-GT subroutines are grouped into three basic sections. Subroutines are ordered alphabetically within each section. The first section of subroutines represents the main body of the code. Problem setup, output, and the algorithms necessary to advance the problem one time-step are contained in this section. An attempt has been made to use the first three or four characters of a subroutine name as an indication of its function. For example:

GRADxxx	,	are subroutines used in the calculation of the cell gradients.
INTxxx	,	are interface related subroutines.
NEWxxx	,	are subroutines used to generate a new data structure when a global remap is required.
REMxxx	,	are subroutines used by the global remapping algorithm.
REZxxx	,	are rezoning or mesh generation subroutines.
PLTxxx	.	are plotting subroutines.
EOSxxx	,	are equations of state subroutines.
SESxxx	,	are equations of state subroutines involving the SESAME library.

A brief description concerning the function of each of the subroutines in this first section, in alphabetical order, is provided.

ADVECT	updates the cell mass, momentum, and energy based on the Lagrangian cell-centered rates of change. The conserved quantities then are modified to reflect a stability limited (i.e., Courant condition) remapping (advection) of the extensive variables from positions that would result from a pure Lagrangian motion to the near Lagrangian mesh (Sec. 2.5.1).

BOUNDARY	determines solutions to the Riemann problem along boundaries. The appropriate boundary conditions along each segment are included.
BOWTIES	detects when the existing time-step size is too large such that boundary segments intersect or cross each other (Sec. 2.3.1).
BSTRUCT	provides boundary data structures. Information containing variables and bits necessary for the specification of boundary segment or vertex conditions, nearest neighbor definition, and boundary vertex ordering also are set.
CELLGEOM	computes the mesh geometric properties. Included are the cell area, volume, centroids, and the moments of inertia necessary for calculation of a minimum cell characteristic length (Sec. 2.1.2).
CELLVRTX	calculates the cell vertex locations. For interior vertices, these locations are the centroids of the triangles determined by the three neighboring cell centers. On boundaries they are the averages of the boundary cell centers lying on either side of the vertex position.
GRAD	computes the cell-centered gradients for the "second order" computations (Sec. 2.2.3).
GRADBNDA	loads the boundary value arrays necessary for the calculation of cell gradients of mass, momentum, and total energy densities along region boundaries. The values are used by the global remap and advection subroutines.
GRADBNDR	loads the boundary value arrays necessary for the calculation of cell gradients of the primary-intensive quantities (density, velocity, ...) along region boundaries. The values are required for the solution to the Riemann problem.
INFLOW	applies the inflow and outflow boundary conditions.
INITIAL	initializes the calculation. It drives the subroutines that read input files (SETUP) and dumps (TAPERD) as well as subroutines that set up the data structure (BSTRUCT) and compute the mesh geometry (CELLGEOM). Primary-extensive quantities also are initialized in this subroutine.
INTFACE	computes the new interface/boundary positions (Sec. 2.3). It relies on a Huygens construction based on the velocities obtained from the Riemann problem. An equidistribution term is used to remove the singularity encountered along lines of symmetry.
INTSECT	computes the intersection of the projected locations of two neighboring boundary segments. The projected segment locations are obtained from a Huygens construction using the two Riemann velocities associated with each segment.
LAGRATES	determines a provisional Courant-limited time-step (through a call to TIMESTEP), calculates and stores solutions to the Riemann problem (u^* and p^*), and computes the Lagrangian cell-centered rates of change for momentum, energy, and volume.
MESH	sweeps through the dual triangulation defined by the cell-centered positions and determines if a reconnection of the mesh is necessary. If the reconnection criterion is satisfied, then the reconnection is made.
NEWADDB	controls the addition of boundary cell points to the data structure and the modification of the information containing variables.
NEWADDI	controls the addition of interior cell points to the data structure and the modification of the information containing variables.
NEWCD	changes the diagonal of a quadrilateral formed by two adjacent triangles in order to simplify deleting a cell point.

NEWDELB	controls the deletion of boundary cell points from the data structure and the modification of the information containing variables.
NEWDELI	controls the deletion of interior cell points from the data structure and the modification of the information containing variables.
NEWMESH	is the driving subroutine that controls the addition and deletion of boundary cell points from the date structure and modifies the information containing variables appropriately.
NEWTRI	modifies the triangle information containing variables to reflect the results of adding or deleting boundary cell points from the problem.
OUTPUT	determines at which cycle numbers or computational time output is demanded. It also drives the subroutines that provide printed and graphical output as well as dumps.
REMAPPER	is the driving subroutine for the global remap algorithm (Sec. 2.5.2). Based on conservation principles, variables are transferred (or remapped) from one mesh to another.
REMLNINT	evaluates the line integrals over the cell sides necessary for the global remap algorithm (Sec. 2.5.2).
REMNEWFD	locates the new mesh cell into which an old mesh segment is directed and the new mesh side the segment crosses.
REMNEWSG	traces a new mesh line segment through the old mesh. It determines which of the old cells it enters and exits and which sides it crosses.
REMOLDFD	locates the old mesh cell into which a new mesh segment is directed and the old mesh side the segment crosses.
REMOLDSG	traces an old mesh line segment through the new mesh. It determines which of the new cells it enters and exits and which sides it crosses.
REMPREP	is the set up subroutine for the global remap algorithm (Sec. 2.5.2). It saves variables associated with the old mesh and calls subroutines that compute the mesh geometry, modify the data structure, and define the new mesh.
REZBNDY	repositions, as well as adds and deletes, boundary cell points in an effort to accurately resolve the boundary contour as it evolves during the calculation (Sec. 2.4.2).
REZCJRS	solves a system of linear equations using the diagonally scaled conjugate residual technique. This subroutine is called by REZLAGR.
REZLAGR	computes a "near Lagrangian" motion of the mesh (Sec. 2.4.1). That is, a new mesh that approximately preserves the original cell volumes is determined.
REZONE	constructs a new mesh based on the specified boundary vertex locations. The algorithm attempts to construct a regular mesh based on the uniformity of angles and sides of the dual triangulation (Sec. 2.4.1).
RIEMANN	obtains the solution to the Riemann problem.
SETUP	reads from the input file (INPGTR) and initializes the equation of state variables.
TAPERD	reads a dump from the restart file (RSTGRT).
TAPEWR	writes dumps to the dump file (DMPGTR).
TERMNATE	is called to terminate the problem under both normal (i.e., end of problem) and abnormal (i.e., code failure) conditions.

TIMESTEP	computes the time-step size based on a Courant stability condition or halves the time-step size if it has been determined that a boundary construction will fail for the existing time-step size. The time-step size also is halved if cell volume changes are too large.
TRIPLPT	computes the new locations of boundary triple points and "fixed" points. A Huygens-like construction using the Riemann velocities of the sides that form the triple point is used to advance these boundary vertices to their new positions.
UPDTVARS	updates the mesh geometry, the cell intensive quantities, and the equation of state variables following a successful time-step.
VORPTS	computes the cell vertex positions for Voronoi cells. (This subroutine currently is not used.)

The above subroutines advance the fluid state and mesh each time cycle. The following subroutines provide graphics output. The plot subroutines use the DISSPLA graphics package. A list of these subroutines, as well as a brief description of their function, is included.

PLTCONTR	provides contour plots of the flow variables. Currently, contour plots of pressure, density, and internal energy may be obtained.
PLTDEBUG	generates mesh plots when the debug option (IDEBUG=1) has been enabled (Sec. 3.5) while under the dynamic debugging tool (DDT). Four plots are provided by this subroutine. The first provides the entire mesh and indicates the window in which the debug plots are provided as set by the variables XWIND1, YWIND1 and XWIND2, YWIND2. The next three plots provide the labeled mesh sides, vertices, and triangles.
PLTMESH	provides a plot of the CAVEAT-GT mesh.
PLTSET	is a utility subroutine that calculates minimum and maximum values required by the plotting package.
PLTVELVC	provides a plot of the velocity vectors.
PLTXY	provides a two-dimensional plot of a flow variable versus the x- or y-coordinate along the problem boundary. Currently, pressure, density, or internal energy plots are available.

The equation of state subroutines follow the plotting subroutines. Given the density and either the pressure or internal energy, these subroutines provide the temperature and either the internal energy or pressure. A diverse set of analytic and tabular equations of state is available. The user specifies the desired state equation for each region through the input file (INPGTR). A detailed discussion of the available models is provided in Sec. 2.6 or Ref. [1]. A brief description of each equation of state subroutine is provided.

EOSBKW	calculates the state variables and their derivatives using the analytic BKW equation of state.
EOSDRIVE	is the driver subroutine for the equation of state calculation.
EOSE	is used to obtain the state variables when density and pressure are provided (i.e., internal energy and temperature are calculated). Depending on the user

	specified equation of state, this subroutine either calculates the state variables or accesses those subroutines that provide the necessary computations. Eight equations of state are available.
EOSHOM	calculates the state variables and their derivatives using the analytic HOM equation of state.
EOSINPUT	initializes the equation of state variables and reads the equation of state specifications (namelist EOSIN) from the input file (INPGTR).
EOSJWL	calculates the state variables and their derivatives using the analytic JWL equation of state.
EOSMAT	loads the equation of state arrays that allow vectorization in the computations of the state variables.
EOSP	is used to obtain the state variables when density and internal energy are provided (i.e., pressure and temperature are calculated). Depending on the user specified equation of state, this subroutine either calculates the state variables or accesses those subroutines that provide the necessary computations. Eight equations of state are available.
EOSSET	is the driver subroutine for the initialization of the state variables.
SESSET	initializes the equation of state variables when the SESAME tabular data are accessed.
SESVALE	is called to obtain the state variables and their derivatives when density and pressure are provided (i.e., internal energy is calculated) using the SESAME tabular data.
SESVALP	is called to obtain the state variables and their derivatives when density and internal energy are provided (i.e., pressure is calculated) using the SESAME tabular data.

3.2.4. Arrays and Variables. A brief description of the important variables used in the CAVEAT-GT code is provided in this section. The variables are listed alphabetically. For arrays, the appropriate dimensions are included in parentheses following the variable name. Array dimensions specifying the number of cells (nv), triangles (nt), sides (ns), and work storage (nwk) are provided by parameter statements. Currently, the number of regions (ir) is fixed at a value of 30. An asterisk following the variable indicates that it is specified by input. Variables followed by a double asterisk are temporary parameters in the work storage array (Sec. 3.2.5). The common block that contains the variable is provided in brackets following the variable description.

aamax	is the normalizing weight factor (refer to $\alpha_{a_{max}}$ in Sec. 2.4.2) applied to the adaptive term in the interface rezoning algorithm. [REMAP]
adsmx(ir)*	is an array specifying the ratio of side length to the region maximum side length above which an interior cell point is added on the side. [REMAP]
akmax	is the normalizing weight factor (refer to $\alpha_{\kappa_{max}}$ in Sec. 2.4.2) applied to the curvature term in the interface rezoning algorithm. [REMAP]
alpa*	is the weight factor (refer to α_a in Sec. 2.4.2) applied to the adaptive term in the interface rezoning algorithm. Currently, this term is absent from the code. [REMAP]

alpe* is the weight factor controlling the spacing on the boundary (1.0 denotes equal spacing and 0.0 denotes the previous cycle spacing). [REMAP]

alpk* is the weight factor (refer to α_κ in Sec. 2.4.2) applied to the curvature term in the interface rezoning algorithm. [REMAP]

alpn* is the weight factor (refer to α_n in Sec. 2.4.2) used in the configurational rezoning algorithm. This parameter should be limited $0 \leq \alpha_n \leq 1$. [REMAP]

ama(ir) is the material strong shock parameter for the region ir. [FLUX1]

area(nv) is the area of a cell. It also is used as temporary storage. [FLUX1]

atnr(ir) is the sum of the area of the triangles in each region (Sec. 2.4.1.2). [MESH1]

bmass(nv)** is the product of density times the boundary segment length required by the near Lagrangian rezone along the boundaries (Sec. 2.4.2).

cang* is an argument used by the DISSPLA graphics package that controls the number of labels on a contour curve. [PLOT]

cart* is set equal to 0 or 1 for a problem in cylindrical or Cartesian coordinates, respectively. [FLUX2]

curv(nv)** is the curvature of the boundary contour at a boundary point.

curvmax* is the maximum curvature (refer to R_{min} = 1/**curvmax** in Sec. 2.4.2) allowed by the configurational rezone algorithm. [REMAP]

cyln* is set equal to 1 or 0 for a problem in cylindrical or Cartesian coordinates, respectively. [FLUX2]

dendt(nv)** is the cell-centered Lagrangian rate of change of total energy.

dens(nv)* is the density of a cell. [FLUX1]

dmxdt(nv)** is the cell-centered Lagrangian rate of change of the x-component of momentum.

dmydt(nv)** is the cell-centered Lagrangian rate of change of the y-component of momentum.

dsmax is the maximum spacing for a boundary segment used by the interfacial rezoning algorithm (refer to ΔS_{max} in Sec. 2.4.2). [REMAP]

dsmin is the minimum spacing for a boundary segment used by the interfacial rezoning algorithm (refer to ΔS_{min} in Sec. 2.4.2). [REMAP]

dsmx(ir)* is the maximum distance allowed for a boundary segment (refer to ΔS_{max} in Sec. 2.4.2) in region ir. [REMAP]

dsmxmn is the average **dsmx** over all the regions and is used for scaling. [REMAP]

dt is the computational time-step size. [FLUX2]

dtdump* is the computational time increment for providing dumps to file DMPGTR. [FLUX2]

dtmax* is the maximum time-step size allowed by the calculation. [FLUX2]

dtmin* is the minimum time-step size allowed by the calculation. If the predicted time-step size falls below this value, the calculation will terminate. [FLUX2]

dtoutp* is the computational time increment for providing output. [FLUX2]

dvmdt(nv)** is the cell-centered Lagrangian rate of change of volume.

dvn(ns)** is the product of the cell-side normal velocity resulting from the Riemann solution and the cell-side length.

dx2,dy2	are the averages of **xmin** and **xmax**, and **ymin** and **ymax**, respectively. These variables are used by the graphics subroutines. [PLOT]
dxmn(nv)**	is the minimum characteristic length for the computational cell.
e(nv)*	is the specific internal energy of a cell. [FLUX1]
ebnd(nv)**	is used to store the total energy at the region boundaries.
eostab(20000)	is a working storage array used by the SESAME equation of state subroutines. [BLANKD]
eps*	is a generic small parameter (epsilon), specified through input. [FLUX2]
es(100,ir)	is a real array containing the equation of state parameters for each material region. [EOS]
eschng*	an array used to input the equation-of-state parameters into the **es** array (see the CAVEAT report [1]).
esif*	is the inflow internal energy used in the specified inflow boundary condition. [FLUX2]
esnew*	an array used to input the equation-of-state parameters in the **es** array (see the CAVEAT report [1])
forthpi	is equal to $\pi/4$. [FLUX2]
fv(ns)**	is the advection flow volume or the difference between the material and mesh velocities times the time-step size and cell side length (Sec. 2.5.1).
gascn(25,ir)*	is a real array containing parameters for the BKW and JWL equation of state for each material region. [EOS]
gex(nv),gey(nv)**	are the cell-centered gradients for pressure or total energy.
gradqx(nt),gradqy(nt)**	are trial cell-vertex gradients (Sec. 2.2.3).
grx(nv),gry(nv)**	are the cell-centered gradients for density or mass.
gux(nv),guy(nv)**	are the cell-centered gradients for the x-component of velocity or momentum.
gvx(nv),gvy(nv)**	are the cell-centered gradients for the y-component of velocity or momentum.
halfpi	is equal to $\pi/2$. [FLUX2]
idebug	is an integer that when nonzero enables the debugging graphics option (Sec. 3.5). [PLOT]
idump*	is an integer that if nonzero equals the dump number to be read in for a restart.
igrad*	specifies the gradient limiting technique used by the higher order computational method (Sec. 2.2.3). A value of 1, 2, or 3 results in no limiting, van Leer limiting, or monotone limiting, respectively. [FLUX2]
ihuyg	is an integer used to denote the failure of the Huygens construction used to propagate the boundary (Sec. 2.3.1). [TERMN]
ilogo*	is an input flag. A value of 1 provides the Los Alamos logo on all plots, and of 0 eliminates the logo. [PLOT]
imesh	is a counter that accumulates the number of calls made to subroutine MESH within an iteration from subroutine REZONE. [TERMN]
incdump*	is the cycle increment for providing dumps to file DMPGTR. [FLUX2]
incoutp*	is the cycle increment for providing output. [FLUX2]

infoa(nv),infob(nv) are information containing arrays associated with the boundary cell centers. They contain references to nearest sides and vertices on boundaries, as well as bits indicating, for example, region number and boundary conditions (Sec. 3). [MESH1]

infoaold(nv)** is an information containing array associated with cell centers of the old mesh prior to a general remap.

inout is a flag used in subroutine REMAPPER to indicate whether or not a cell segment is exiting a cell. [REMAP]

iorder* controls the approximate order of accuracy of the calculation. Allowed values are:

0; a first-order Riemann solution, advection, and global remap.

1; a second-order Riemann solution, first-order advection and global remap.

2; a first-order Riemann solution, second-order advection and global remap.

3; a second-order Riemann solution, second-order advection and global remap. [FLUX2]

ipdens* is an input flag. A nonzero value provides plots of density contours to the graphics file. [PLOT]

ipener* is an input flag. A nonzero value provides plots of specific internal energy contours to the graphics file. [PLOT]

ipmesh* is an input flag. A nonzero value provides mesh plots to the graphics file. [PLOT]

ippres* is an input flag. A nonzero value provides plots of pressure contours to the graphics file. [PLOT]

ipvc* is an input flag. A nonzero value includes the mesh with the velocity vector plots. [PLOT]

ipvelv* is an input flag. A nonzero value provides velocity vector plots to the graphics file. [PLOT]

ipxy* is an input flag. A nonzero value provides two-dimensional plots of pressure, density, and specific internal energy versus x (**ipxy** = 1) or y (**ipxy** = 2). [PLOT]

iremap is a flag that indicates if a global remap of the mesh is necessary. [REMAP]

irezone* is a flag that indicates if the boundary of the mesh is to be rezoned. [REMAP]

isegend is a flag used in subroutine REMAPPER that indicates when a segment has been traced to its end. [REMAP]

isetp* is an input flag used to indicate whether the initial pressure (**isetp** = 0) or specific internal energy (**isetp** = 1) is calculated by the equation of state subroutines during initialization. [EOS]

iskipf* is an integer used to specify the number of frames skipped on the graphics file following each time-step. This allows, for example, similar plots to be positioned on horizontal lines when microfiche copies of the plot file are produced. [PLOT]

iswitch is a variable used in subroutine MESH that accumulates the number of reconnections. [MESH2]

iter is an integer used as an iteration counter. [TERMN]

ivec* is an argument used by the DISSPLA graphics package that specifies the size and type of an arrowhead used on velocity vector plots. [PLOT]

k0	is an integer denoting a fatal error number. [TERMN]
kfirst	is the smallest cell number that is a boundary cell. [MESH2]
kl(nv)	is a vector containing the boundary cell point numbers as one proceeds in order from a logically fixed point around the region boundaries. Its contents change dynamically as cells are added/deleted. [MESH1]
klast	is the largest cell number that is a boundary cell. [MESH2]
klo(nv),klold(nv)**	are vectors containing the boundary cell numbers, as one proceeds in order around each region, prior to a general remap.
kmat(nv)	is an integer array containing the cell numbers (k) ordered by material number. [EOS]
kmatpnt(ir)	is an integer array containing pointers that provide the starting location of cells in each material group. [EOS]
ktabs	is the location of the first word in the storage block for the SESAME equation of state tables. [EOS]
lcfrst	is a dummy variable used by the dump subroutines to locate the beginning of the storage arrays. [FIRST]
lclast	is a dummy variable used by the dump subroutines to locate the end of the storage arrays. [LAST]
ldbl(ir,i)	contains the location in the **kl** array of the first (i=1) and last (i=2) boundary cell point for region **ir**. [FLUX1]
ldblold(ir,i)	contains the location in the **kl** array of the first (i=1) and last (i=2) boundary cell point for region **ir** in the old mesh prior to a global remap. [FLUX1]
ls(ns)**	is an index that provides the segment number and orientation for the Riemann velocities along boundaries and interfaces.
mask1	is a mask used to extract information in the first 20 bits of a word (Sec. 3.1). It is used to obtain cell-center numbers from the **ns1**, **ns2**, and **infob** arrays, and cell-side numbers from the **infoa** array. [MASKS]
mask2	is a mask used to extract information from bits 21 through 40 of a word (Sec. 3.1). It is used to obtain cell-side numbers from the **ns1**, **ns2**, and **infoa** arrays, and cell-center numbers from the **infob** array. [MASKS]
mask3	is a mask used to extract information from bits 41 through 60 of a word (Sec. 3.1). It is used to obtain cell-vertex numbers from the **ns1** and **ns2** arrays, and cell-center numbers from the **infob** array. [MASKS]
mask61	is a mask used to extract bit 61 from a variable. [MASKS]
mask62	is a mask used to extract bit 62 from a variable. [MASKS]
mask63	is a mask used to extract bit 63 from a variable. [MASKS]
mask64	is a mask used to extract bit 64 from a variable. [MASKS]
maskadd	is a mask used to extract or set bit 54 from the variable **infoa(k)** (Sec. 3.1). The result indicates if the k^{th} cell has been added to the mesh. [MASKS]
maskbc	is a mask used to extract bits 61 through 64 from the variables **ns1(m)** and **ns2(m)** (Sec. 3.1). The results are the boundary condition type number applied to the m^{th} cell side. [MASKS]
maskccv	is a mask used to extract bit 52 from the variable **infoa(k)**. The result denotes if the cell is convex or concave. [MASKS]

maskdel	is a mask used to extract or set bit 53 from the variable **infoa(k)** (Sec. 3.1). The result indicates if the k^{th} cell has been deleted from the mesh. [MASKS]
maskid •	is a mask used to extract bit 56 from the variable **infoa(k)** (Sec. 3.1). [MASKS]
maskkd	is a mask used to extract bit 55 from the variable **infoa(k)** (Sec. 3.1). [MASKS]
maskrev	is a mask used to extract bit 54 from a word. It is required by the SESAME equation of state subroutine EOSP.
masknr	is a mask used to extract bits 57 through 61 from the variable **infoa(k)** (Sec. 3.1). The result is the region number of the k^{th} cell. [MASKS]
masktyp	is a mask used to extract bits 62 through 64 from the variable **infoa(k)** (Sec. 3.1). The result is the type of interface on which the k^{th} cell lies. [MASKS]
mass(nv)	is the mass of a cell. It also is used as temporary storage. [FLUX1]
mnew(nt)**	stores the new mesh boundary side associated with an old mesh boundary triangle.
mnses	is an integer specifying the number of materials that are described using the SESAME equation of state tables. [EOS]
mntot	is an integer specifying the total number of materials. [EOS]
mold(nt)**	stores the old mesh boundary side associated with a new mesh boundary triangle.
nbowts	is an integer specifying the number of bowties or loops in the boundary (Sec. 2.3.1). [TERMN]
ncelmat(ir)	is an integer array containing the number of cells in each material region. [EOS]
ncyc*	is the beginning or current computational cycle number. [FLUX2]
ncycstop*	is the cycle number at which the calculation is terminated. [FLUX2]
ndump	is the number of the dump read from the restart file (RSTGRT) and compared with **idump**. [FLUX2]
nl	is the total number of cells on the boundaries (i.e., the active length of the variable **kl**). Its value changes dynamically as cells are added/deleted. [MESH2]
nln	is the total number of cells on the boundary (i.e., the active length of the variable **kl**) following the rezoning of the mesh. [REMAP]
np*	is the total number of cells. Its value changes dynamically as cells are added/deleted. [MESH2]
npn	is the total number of cells following the rezoning of the mesh. [REMAP]
nrd*	is the total number of regions in the problem. [MESH2]
nrez	accumulates the number of times matrix coefficients have been recomputed in subroutine REZONE. [TERMN]
nrmn(ir)*	is an integer array containing the material number of each region. [EOS]
ns1(ns),ns2(ns)	are information containing arrays associated with the cell sides. They contain information pertinent to the Z-data structure (Sec. 2.1.3) such as neighboring cell centers, sides, and vertices, as well as boundary condition information. [MESH1]
ns1old(ns),ns2old(ns)**	are information containing arrays associated with cell vertices of the old mesh prior to a general remap.

nsd*	is the total number of cell sides (interior plus boundary). Its value changes dynamically as cells are added/deleted. [MESH2]
nsdi*	is the number of interior cell sides. Its value changes dynamically as cells are added/deleted. [MESH2]
nsdiold	is the number of interior cell sides prior to the change in the data structure resulting from rezoning the mesh. [REMAP]
nsdold	is the total number of cell sides (interior plus boundary) prior to the change in the data structure resulting from rezoning the mesh. [REMAP]
nsold(nt)**	is the location of the new mesh boundary cell vertex relative to the old mesh boundary index **klold**.
ntabs	is an integer specifying the storage requirements for the SESAME equation of state tables. [EOS]
ntnr(ir)	is the number of triangles in each region (Sec. 2.4.1.2). [MESH1]
ntr*	is the total number of triangles (interior plus boundary). Its value changes dynamically as cells are added/deleted. [MESH2]
ntri*	is the number of interior triangles. Its value changes dynamically as cells are added/deleted. [MESH2]
ntriold	is the number of interior triangles prior to the change in the data structure resulting from rezoning the mesh. [REMAP]
nts(nt)	contains information associated with triangles (Sec. 3). It contains the three side numbers forming the triangle that connects the three neighboring cell centers. [MESH1]
nxtdump*	is the next cycle number at which a dump to file DMPGTR is to be made. On input, this variable specifies the cycle number of the first dump. [FLUX2]
nxtoutp*	is the next cycle number at which output is provided to the terminal (tty), print file (OUTGTR), and graphics file (PLTGTR). On input, this variable specifies the cycle at which output is initially supplied. [FLUX2]
oldmass(nv)	is the mass of a cell following the last global remap. [FLUX1]
pbnd(nv)**	is used to store pressure at the region boundaries.
pc(nv)*	is the pressure of a cell. It also is used as temporary storage. [FLUX1]
pi	is equal to $\pi = 3.1415\ldots$. [FLUX2]
pn(ns)**	is the cell-side pressure resulting from the Riemann solution.
psif*, psof*	are the inflow and outflow pressure, respectively, for the specified inflow and outflow boundary conditions. [FLUX2]
pspec*	is the pressure applied to specified pressure boundaries. [FLUX2]
ptdens(nv)**	is the region boundary point density function (Sec. 2.4.2).
r(nv)**	is the pseudo-radius (Sec. 2.1.2).
rbnd(nv)**	is used to store the density or mass at the region boundaries.
rhoe(nv)**	is the product of the density and total energy at the cell center.
rhosif*	is the inflow density used in the specified inflow boundary condition. [FLUX2]
rhou(nv)**	is the product of the density and x-component of velocity at the cell center.
rhov(nv)**	is the product of the density and y-component of velocity at the cell center.

s(nv)**	is the arclength at a boundary point as one proceeds around the region following the relocation of boundary cell points.
sdsmx(ir)*	is an array specifying the ratio of side length to the region maximum side length below which an interior cell point is deleted on one end of that side. [REMAP]
sixth	is equal to 1/6. [FLUX2]
sloc(nv),lold(nv)**	identifies the location of cell points along the boundary as a function of the location in the old (before remap) **kl** array. This is necessary for a global remap.
so(nv)**	is the arclength at a boundary point as one proceeds around the region prior to relocating boundary cell points.
solidcn(25,ir)*	is a real array containing parameters for the HOM equation of state for each material region. [EOS]
ss(nv)	is the local isentropic speed of sound for a cell. It also is used as temporary storage. [FLUX1]
switch	is a logical variable used in subroutine MESH to indicate when reconnections of the dual triangulation are necessary. [MESH2]
tarea(nt)**	are the areas of the triangles constructed by connecting cell centers.
tau	is the smoothing coefficient $c^2\Delta t$ (Sec. 2.4.2) used in the interfacial rezoning algorithm. [REMAP]
tdump*	is the computational time beyond which the next dump is made to file DMPGTR. On input, this variable specifies the computational time of the first dump. [FLUX2]
third	is equal to 1/3. [FLUX2]
thknss*	is an argument used by the DISSPLA graphics package that controls the thickness of contours specifying external boundaries and interfaces in the mesh plots. [PLOT]
time*	is the beginning or current computation time. [FLUX2]
title(10)*	is a character string containing the problem title. [PLOT]
tmass(nv)	is the inverse of the density times the length of a boundary cell side ($\rho_0\Delta s_0$). It is required by the near Lagrangian interfacial, rezoning algorithm (Sec. 2.3.4 and 2.4.2). [FLUX1]
tote(nv)	is the total energy of a cell. It also is used as temporary storage. [FLUX1]
toutp*	is the computational time at which output is provided to the terminal (tty), print file (OUTGTR), and graphics file (PLTGTR). On input, this variable specifies the first computational time at which output is provided. [FLUX2]
tstop*	is the computational time beyond which the calculation is terminated. [FLUX2]
ubnd(nv)**	is used to store the x-component of velocity or momentum at the region boundaries.
uc(nv)*	is the x-component of velocity of a cell. [FLUX1]
umom(nv)	is the x-component of momentum of a cell. It also is used as temporary storage. [FLUX1]
unsif*	is the normal inflow velocity used in the specified inflow boundary condition. [FLUX2]

unspec*	is the normal velocity applied to specified velocity boundaries. It is positive in the direction of the unit outward normal to the boundary. [FLUX2]
uvtx(nv),vvtx(nv)**	are the cell-centered x- and y-component of velocities, respectively, that result from the near Lagrangian rezoning of the mesh (Sec. 2.4.1).
vbnd(nv)**	is used to store the y-component of velocity or momentum at the region boundaries.
vc(nc)*	is the y-component of velocity of a cell. [FLUX1]
vmom(nv)	is the y-component of momentum of a cell. It also is used as temporary storage. [FLUX1]
vol(nv)	is the volume of a cell. [FLUX1]
vsf*	is a parameter used to specify the length of the maximum velocity vector relative to the average cell dimension. [PLOT]
work(10000)	is a working storage array used by the contour plot subroutine. [BLANKD]
works(nwk)	is working storage. Temporary storage is provided by this array (Sec. 3.2.5). [WORKC]
ws1(nv),ws2(nv)**	are the Riemann velocities along boundaries and interfaces.
xa,ya	are the plot frame length and height, respectively, in plotting space. [PLOT]
xaray(nv),yaray(nv)	are temporary storage arrays used for plotting abscissa and ordinate values. [PLOT]
xc(nv),yc(nv)*	are the cell point positions. [MESH1]
xcd(nv),ycd(nv)	are the cell centroid positions. [MESH1]
xcl(nv),ycl(nv)	are the "near Lagrangian" cell center positions. They also are used as dummy variables in subroutine INTFACE. [MESH1]
xcn(nv),ycn(nv)**	are temporary positions of the cell centers.
xmax,ymax,xmin,ymin	are the minimum and maximum plot frame coordinates in physical space. [PLOT]
xnmo(nv)**	results from integrating the point density function around the region boundary (refer to N in Sec. 2.4.2).
xv(nt),yv(nt)	are the cell vertex positions. [MESH1]
xvl(nt),yvl(nt)**	are the "near Lagrangian" cell vertex positions.
xwind1, xwind2	are the minimum and maximum abscissa values, respectively, of the debugging graphics window (Sec. 3.5). [PLOT]
xxc(nv),yyc(nv),xyc(nv)**	are the area moments of inertia of the computational cells (Sec. 2.1.2).
ywind1, ywind2	are the minimum and maximum ordinate values, respectively of the debugging graphics window (Sec. 3.5). [PLOT]

3.2.5. Work Arrays. The CAVEAT-GT code employs a work array (**works**) to store temporary arrays as they are required in the calculational procedure. The size of this array is 33*nv, where nv is the number of cells required by the problem. The work array is used in blocks. The size of each block being nv. Consequently, there are 33 blocks of available storage for the use as work arrays. Each block of storage may be accessed using the

parameter $\mathbf{nwk_x}$, where x = 1, 2, ..., 33. In order to use a block of work storage for a temporary array **phi** for example, one simply must equivalence **phi(1)** to **works** ($\mathbf{nwk_x}$).

Because many of the temporary arrays are calculated in one subroutine and used in another, it is useful to know when each block of the work array is being utilized. This prevents the accidental overwriting of a temporary array prior to its use. The allocation of each block ($\mathbf{nwk_x}$) of the work array (**works**) is provided in Table I for one cycle of a calculation. Temporary arrays based on the number of cells (k) in the problem require one block of storage of size **nv**. Information based on the number of triangles (n) and sides (m) require 2 and 3 blocks of storage, respectively. Consequently, some of the temporary arrays are observed to use multiple blocks in Table I.

In an effort to obtain additional storage, some of the permanent arrays also are used as temporary storage in the code. In subroutine INTFACE, the arrays **xcl** and **ycl** are used as temporary variables. The arrays **pc**, **ss**, and **area** also are used as temporary storage. Their temporary values (**rhou**, **rhov**, and **rhoe**) are calculated in REMPREP and used later in REMLINIT. Finally, **mass**, **umom**, **vmom**, and **tote** are used as temporary storage in the subroutine REZONE.

3.3. Setup Procedure

The setup code CAVGTS generates all required input for the run code CAVGTR and writes this input to a file called INPGTR that is read by CAVGTR. Any parameters that are not specified during the CAVGTS run are set to their default values. All parameters except the initial mesh data structure are written in namelist format. The value of any parameter can be changed using a standard text editor. Therefore, it is not necessary to run CAVGTS every time CAVGTR is used. The definition and default value of each parameter is given by the CAVGTS code and, therefore, is not listed here.

3.3.1. Mesh. The initial triangular mesh is generated using the TRIGEN package of subroutines from the PLTMG code [14].

The main program of CAVGTS requests the required data following the procedure discussed in Sec. 2.7. Then TRIGEN uses these data to generate a triangular mesh that is plotted by subroutine PLTMESH. The user then can vary the maximum triangle side length and adjacent triangle area smoothness to optimize the mesh. Next, subroutine GZDS is called to convert the TRIGEN data structure to the required "Z" data structure (Sec. 2.1.3). All internal interfaces are doubly defined and all boundary (external and interface) sides are moved to the end of the side arrays. Finally, all triangle data is generated. Then subroutine GFILE compacts the "Z" data using masking and writes these data into the INPGTR file. All additional mesh parameters are in Namelist $GRID and are written into the INPGTR file using default values.

TABLE I

WORK ARRAY ALLOCATION*

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1				r		coupl	sloc								sloc		ℓold	
				grx		diag1	so			grx	da,dxc		grx				grx	da,dxc
				gry		diag2	s			gry	da,dxc		gry				gry	da,dxc
		rbnd		gux		diag3	xnmo,ei,curv	resx	rbnd	gux	da,dxc	rbnd	gux				gux	da,dxc
5				guy			fi	resy		guy	dv,dyc		guy				guy	dv,dyc
		ubnd		gvx		scf		uvtx	ubnd	gvx	dv,dyc	ubnd	gvx				gvx	dv,dyc
				gvy		kltmp		uvty		gvy	dv,dyc		gvy				gvy	dv,dyc
				ℓs	ℓs	ℓs	xcn								xcn	resx		
				ℓs	ℓs	ℓs	ycn								ycn	resy		
10				ℓs	ℓs	ℓs			rhou		dxx			ns1old			ns1old	dxx
				ws1	ws1	ws1			rhov		dxx			ns1old			ns1old	dxx
				ws2	ws2	ws2	bmass	coef	rhoe		dxx			ns1old			ns1old	dxx
		vbnd		gex			ptdens	coef	vbnd	gex	dyy	vbnd	gex				gex	dyy
				gey				coef		gey	dyy		gey				gey	dyy
15		pdnd						diag	ebnd		dyy	ebnd		infoaold			infoold	dyy
									fv		dxy			ns2old			ns2old	dxy
									fv		dxy			ns2old			ns2old	dxy
									fv		dxy			ns2old			ns2old	dxy
							klo										klold	
20				pn				pp,xvℓ									xvℓ	xvℓ
				pn				ap,xvℓ									xvℓ	xvℓ
				pn				yvℓ									yvℓ	yvℓ
								yvℓ									yvℓ	yvℓ
			tarea,qcmin,cymin	dvmdt				dvmdt		tarea,qcmin,qymin	xxc		tarea,qcmin,qymin			coef1	mold	xcc
25			tarea,qcmax,cymax	dmxdt					dmxdt	tarea,qcmax,qymax	yyc		tarea,qcmax,qymax			coef1	mold	yyc
			sarea	dmydt					dmydt	sarea	xyc		sarea			coef1	nsold	xyc
			qmin,qxmin	dendt					dendt	qmin,qxmin			qmin,qxmin			coef2	nsold	
	dxmn		qmax,qxmax	dvn					dvn	qmax,qxmax	dxmn		qmax,qxmax			coef2	mnew	dxmn
			gradqy	dvn					dvn	gradqy			gradqy			coef2	mnew	
30			gradqy	dvn					dvn	gradqy			gradqy			diag		
			gradqx							gradqx			gradqx			arx,qx		
			gradqx							gradqx			gradqx			ary,qy		

*The first column refers to a block of storage in the work array (works). Columns 2-19 refer to the following subroutines in the computational cycle.

2. TIMESTEP
3. GRADBNDR
4. GRAD
5. LAGRATES, BOUNDARY
6. TRIPLPT, INTSECT
7. INTFACE
8. REZBNDY
9. REZLAGR, REZCJRS
10. ADVECT
11. GRAD, GRADBNDA
12. CELL GEOM
13. GRADBNDA
14. GRAD
15. REMPREP
16. NEWMESH
17. REZONE
18. REMAPPER
19. UPDTVARS, CELLGEOM

The CAVGTS code also writes a dump file called DMPGTS. This file contains all of the geometry data. By switching the file name to RSTGTS and using the CAVGTS code restart option the user can avoid reading in these data again when he wishes to change the spatial resolution of the mesh or any nonmesh parameters. In addition, many geometry parameters can be changed in the RSTGTS file. This allows the user to correct mistakes or change the geometry. A description of the contents of this file is contained in comment statements in the main program.

If desired, the "Z" data structure can be written in the OUTGTS file for high speed printing. The PLTGTS file contains a plot of the triangular mesh while the DSPGTS file contains any messages generated by the DISSPLA graphics routines.

3.3.2. Hydrodynamics. The hydro data consists of the initial conditions for all flow variables as well as the parameters that control the time-step, output, and numerical method. The initial conditions consist of the velocity, density, pressure, and internal energy for each flow region. These values are assumed to be constant in each region. The initial conditions are written into the INPGTR file in Namelist $IC.

The parameters that control the output and numerical method are written into the INPGTR file in Namelist $CNTRL. The definition and default value for each parameter are given by the CAVGTS code and, therefore, are not listed here.

3.3.3. Equation of State. The equation of state (EOS) parameters are written into the INPGTR file in two namelists, $EOS and $EOSIN. The $EOS namelist contains the array **nrmn**, which gives the material numbers as a function of region number. All other EOS parameters are contained in the $EOSIN namelist. This namelist is the same as the one in the CAVEAT code [1] and, therefore, is not discussed in detail here. The setup code writes a default $EOSIN into the INPGTR file assuming a gamma law gas ($\gamma = 1.4$). If the user wishes to employ another EOS, then he should generate the appropriate $EOSIN namelist following the instructions in the CAVEAT report. This new $EOSIN namelist would then replace the default namelist.

3.4. Updating, Compiling, and Execution

To compile the setup code use the LANL utility XEQ to compile the source code CGTSSRC by typing

```
XEQ  CGTSSRC ,
```

which generates an executable code called CAVGTS. (The XEQ utility treats all statements at the beginning of the source code that have a * in column one as though they were typed in at a terminal.) To execute this code type

```
CAVGTS
```

and follow the instructions. The CAVGTS code then generates the input file INPGTR for the run code.

The procedures to update, compile, and execute the run code are very similar to the corresponding procedures in CAVEAT [1]. Updating and compiling is done using the LANL command language interpreter called CTL. This utility allows one to invoke a controller (a sequence of commands) that automates the procedures required to update, compile, and print listings. The controller file, called PRCVGT, is an improved version of the CAVEAT controller PRCV.

Updating and compiling is simply a matter of invoking PRCVGT under CTL by typing

- PRCVGT

and responding to the appropriate prompts. A response of "abort" to any of the questions will cause the controller to terminate. The controller will normally ask for the name of the source file (CGTRSRC) in order to create an OLDLIB file. However, if this is a repeat session and an OLDLIB already exists, it will ask if one wants to use the existing OLDLIB (or else change to a new source file). PRCVGT then asks for the name of the update file and stores it in UPDECK. If an UPDECK file already exists, PRCVGT asks if one wants to use it or change to a different update file. If no update or modification of the code is desired, then generate a dummy UPDECK file containing the single line

*ID DUMMY .

PRCVGT then compiles the updated code and produces an executable code called CAVGTR. Following compilation, PRCVGT asks if a listing is desired and if so, it asks for a title of the listing. To execute the code type

CAVGTR .

The following is a sample of a typical interactive session:

```
ctl
 -----      ctl ver 03/01/83    latest news 03/01/83
*get ctllibc
 ----- 090  did not find default procedure library
/-prcvgt
*select ttyecho=nocommands
type abort to exit this procedure
no oldlib-- enter historian source file name
? cgtrsrc
input update file name
? updkst
CF000 - CFT VERSION - 1.14i  Edition 003  BUILT: 03/07/89   AT: 14:26:01
CF001 - COMPILE TIME =   15.1033 SECONDS
CF002 -    13143 LINES,     8589 STATEMENTS
controllee name is cavgtr   ,load length= 01370400
do you wish to produce and print an update listing? y or n
? n
/end
```

3.5. Debugging Graphics

Subroutine PLTDEBUG is included to generate a small window of the triangular mesh with all sides, cell points, and triangles labeled. This routine can only be accessed using the LANL routine DDT or some other dynamic debugging routine.

To activate PLTDEBUG, one sets a breakpoint at the beginning of subroutine OUTPUT on the desired time cycle (note that OUTPUT is called on every time cycle). Once the code has stopped at the beginning of OUTPUT, use the "set" or similar command to change the values of the variables **idebug**, **xwind1**, **xwind2**, **ywind1**, and **ywind2**. **idebug** is set equal to 1 while **xwind1** and **xwind2** denote the left and right x-values of the window, respectively. Similarly YWIND1 and YWIND2 denote the lower and upper y-value of the window, respectively. The code will then make the following four plots: the entire triangular mesh showing the window and three plots of the mesh in the window with the sides, cell points, and finally the triangles labeled. The mesh locations plotted are from the **xcd** and **ycd** arrays. These values are used in place of **xc** and **yc** so that a gap between regions is present, which allows easier viewing of the labels. These plots are produced regardless of whether the standard output plots are scheduled to be generated. After these four plots are generated, **idebug** is automatically set to 0 and no more debug plots are produced until once again the user stops the code at subroutine OUTPUT and resets **idebug** = 1.

3.6. Dumps and Restarts

The dump file (DMPGTR) contains all of the computational variables necessary to uniquely define the state of the calculation at a specified time cycle. This gives the user the ability to restart the calculation in a consistent fashion simply by using the dump file of a previous calculation as a restart file.

The initial time at which a dump is written to file DMPGTR is specified by the input variable **tdump**. Time increments at which dumps are taken thereafter are provided by **dtdump**. Alternatively, the user may specify dumps at input specified cycle numbers. The first dump cycle being **nxtdump** and cycle increments thereafter by **incdump**.

When a dump is written to the file DMPGTR, a message is written to both the terminal (tty) and the output file (OUTGTR). The message provides the dump number as well as the cycle number and computational time at which the dump was provided.

To restart the problem from a specified dump, one must first rename the dump file RSTGTR. Then use the existing input file (INPGTR) with the user specified dump number (**idump**) set to the dump from which a restart is desired. If the appropriate dump is found on RSTGTR, a message is provided to the terminal and the output file indicating dump cycle number and computational time of the restart. If the correct dump is not located on the restart file, a message indicating the failed attempt is provided to the terminal. This last

message includes the user specified dump number and the last dump number encountered on the restart file. The code then will terminate.

4. EXAMPLE CALCULATIONS

Presented here are the following three calculations: the shock tube problem presented in [15], as well as the blast wave and Taylor anvil problems discussed in [16] and [17]. Due to several modifications to the CAVEAT-GT code, there are small differences between these results and those in [16] and [17].

4.1. Shock Tube

This example is the shock tube problem as specified by Sod [15]. In this problem high-pressure gamma-law gas is located in the left end of a tube and low-pressure gamma-law gas in the right. Initially the high- and low-pressure gas are separated by a diaphragm that is broken at a time $t = 0$. For $t > 0$, a shock wave travels to the right followed by the contact surface or interface and an expansion wave travels to the left. The initial conditions are $\rho_L = p_L = 1, \rho_R = 0.125, p_R = 0.1, \gamma = 1.4, \Delta x = 0.01$, and a total length $L = 1$.

Because this is a one-dimensional problem in the x-direction, there is only one triangle in the y-direction and 200 (adjacent triangles for a square) in the x-direction. As a result there are two fluid or computational cells in the y-direction and 100 in the x-direction. The two cells in the y-direction are both boundary cells, that is, there are no interior cells.

The pressure and density, as a function of x, are shown in Fig. 36 for a time $t = 0.143$. These results are for the "second-order" scheme with van Leer limiting. The pressure results agree well with the exact solution. However, while the density results are excellent for the shock and expansion waves, the results for the contact are not as good. This is due to the fact that as the shock and expansion waves are formed at early time, there is not sufficient mesh resolution, causing numerical error. This error shows up at the contact surface for early time and then persists without change for later time. This type of error is also present in the CAVEAT code [1]. Note that the computed results curve in Fig. 36 is actually two curves, one for the top row of fluid cells and one for the bottom.

This problem uses an update to set the minimum ordinate value to zero and plot the exact solution on the p versus x and ρ versus x plots. Therefore, this problem will execute correctly without the update. The update file is UPDKST and the input file is INPGTRST. The cpu time for this problem is 17s on a Cray X-MP computer.

4.2. Blast Wave

This example is the blast wave problem presented in Ref. [16]. The problem domain is a 2×2 square, occupied by a gamma-law gas ($\gamma = 5/3$), with an "obstacle" located in the lower left corner as shown in Fig. 37. The mesh is relatively coarse, containing 593 fluid or

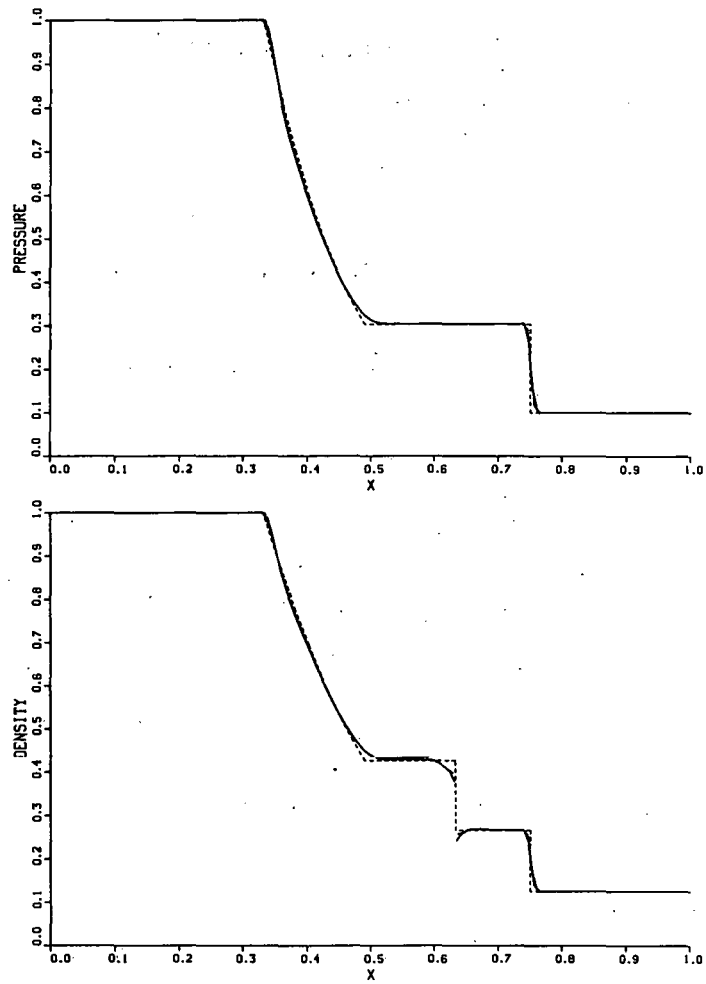


Fig. 36. Pressure (top) and density for the shock tube example problem (dashed line-exact solution).

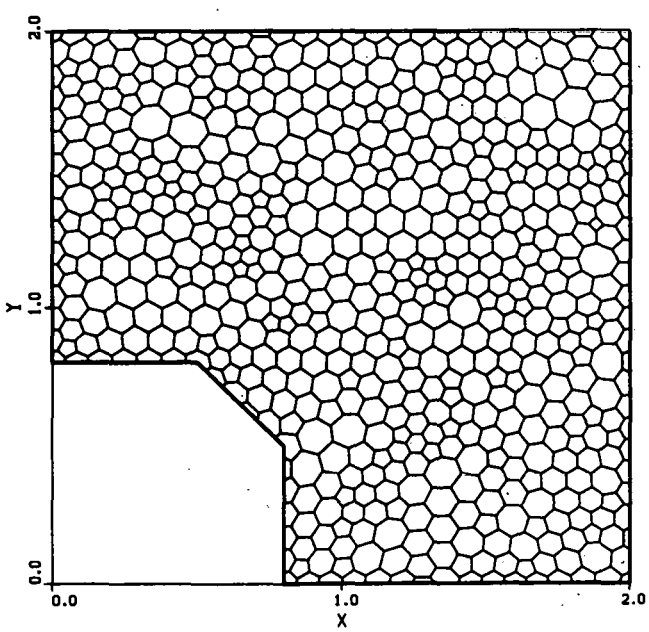


Fig. 37. Initial mesh geometry for the blast wave example problem.

computational cells, with a characteristic cell dimension of approximately 0.08. Initially, the fluid is uniform and quiescent with density and specific internal energy of 1 and 1.5×10^{-10} (dimensionless units), respectively. At $t = 0$, a source of energy is applied to the cell located at approximately $x = y = 0.95$. This cell instantaneously acquires an internal energy of 10.

As the calculation evolves, a cylindrical shock wave emanates from the energy source. The evolution of the pressure profiles for the "first-order" calculation is shown in Fig. 38. It may be observed that the wave is not propagated preferentially in any one direction on the internal mesh. This demonstrates the improved isotropy anticipated for this method, as compared to the related CAVEAT code [1], based on a quadrilateral mesh, that typically shows differences in shock propagation in directions that are skew to the mesh directions. For more results see Ref. [16].

This problem uses an update to add the initial energy to initiate the blast wave. Therefore, this problem needs the update to execute correctly. the update file is UPDKBW and the input file is INPGTRBW. The cpu time for this problem is 105s on a Cray X-MP computer. Like the results in Ref. [16], this problem did not employ interior cell point addition/deletion, therefore **adsmx** was set equal to a large number (1000.0) and **sdsmx** was set equal to zero.

4.3. Taylor Anvil

This example is the Taylor anvil or impact problem presented in Ref. [16]. This problem consists of a plate with dimensions 0.4 by 2.0, a density of 8.9, and traveling at a uniform velocity of 0.196 in the vertical (downward) direction. At $t = 0$ the plate encounters a rigid wall. The phenomena that one observes are that a shock wave propagates vertically from the point of impact and is then overtaken by a rarefaction wave as the plate "splatters" against the rigid wall. The initial mesh, shown in Fig. 39a is fairly coarse with 123 fluid or computational cells. The cell characteristic dimension is 0.1. The left and bottom boundaries are symmetry boundaries. The right boundary is a free surface ($p = 0$), while the top boundary is a specified velocity boundary ($v = -0.196$). The material obeys the Chaplygin equation of state,

$$p = k^2\left(\frac{1}{\rho_0} - \frac{1}{\rho}\right)$$

where $k = 3.49$ and $\rho_0 = 8.9$. The initial velocity corresponds to a Mach number of 0.5, based on the undisturbed sound speed. (Note this equation of state is not included in the CAVEAT-GT code and is added for this problem by updating the code. This equation of state was used to allow the computed solution to be compared to an exact solution in

Ref. [18].) The evolution of the mesh for a "second order," van Leer limited calculation is shown in Fig. 39. The ability of the technique to smoothly add and delete computational cells along the boundaries, as well as in the interior, can be observed. For more results see Ref. [16].

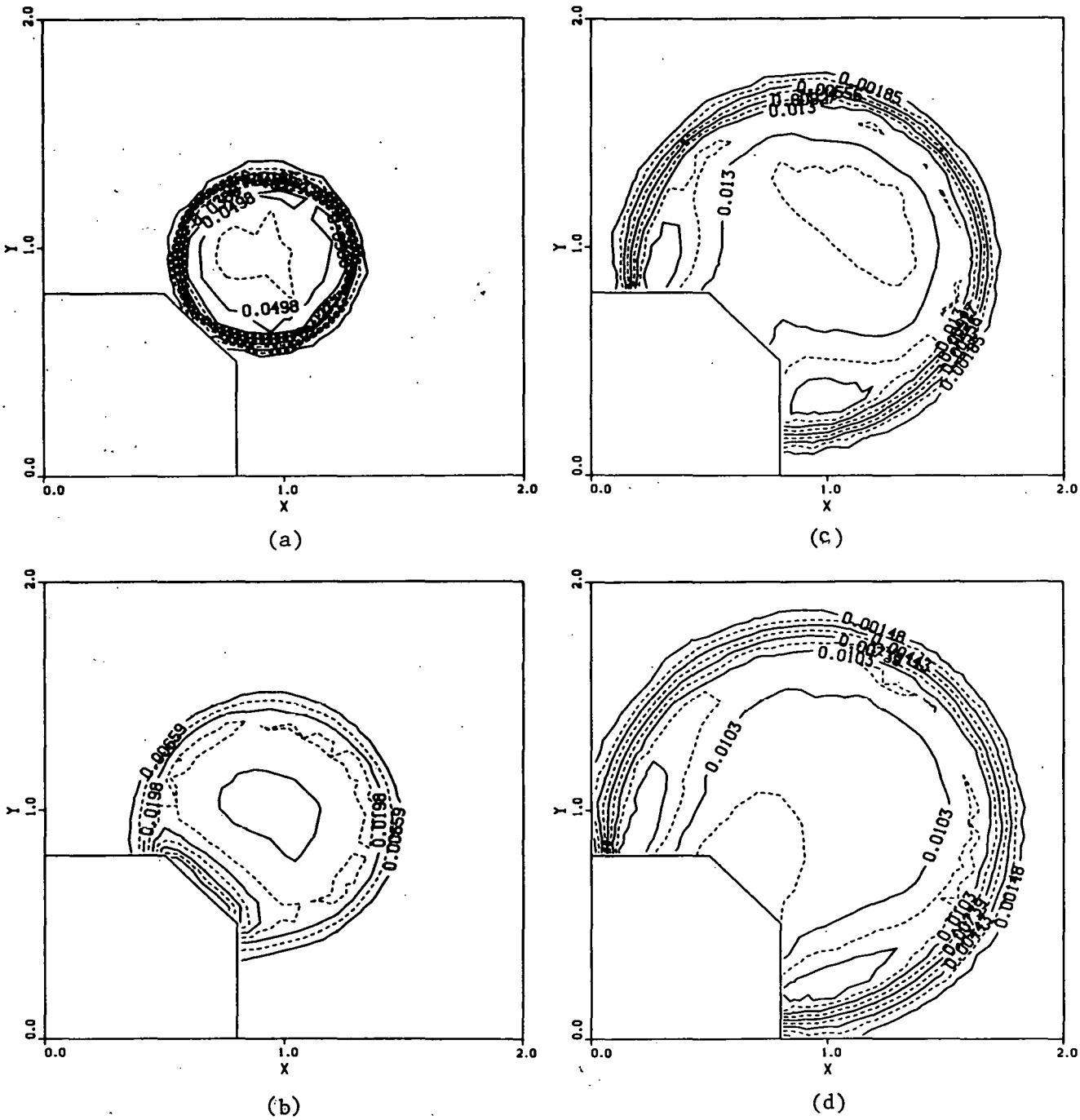


Fig. 38. Pressure contours for the blast wave example problem: (a) t=0.5; (b) t=1.0; (c) t=2.0; (d) = 2.5.

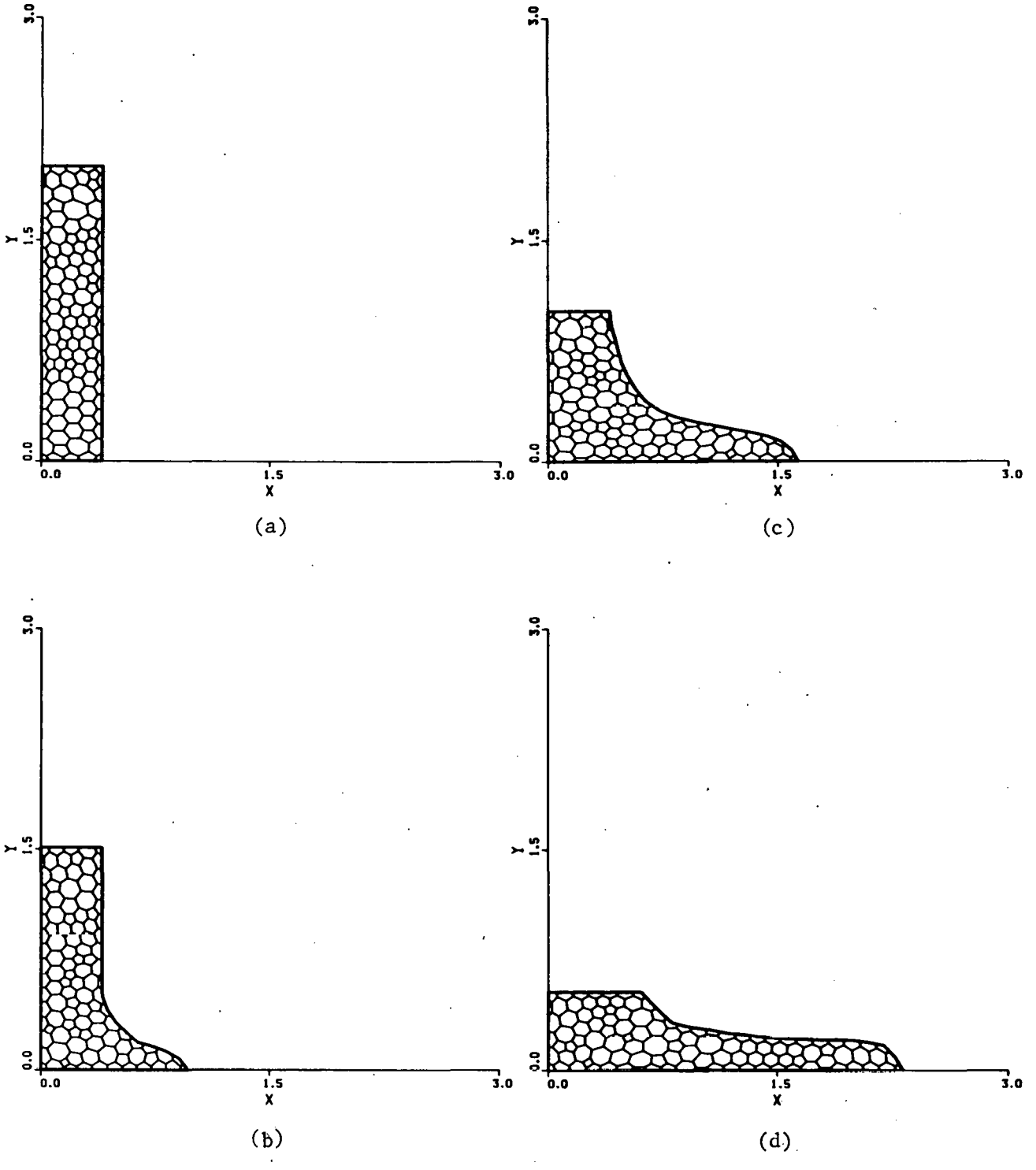


Fig. 39. Evolution of the mesh for the Taylor anvil example problem: (a) initial mesh; (b) t=2.5; (c) t=5.0; (d) t=7.5.

This problem uses an update to fix the plot limits so the plots are not rescaled during the run and to add the Chaplygin equation of state. Therefore, this problem will execute without the update, but will use the linear equation of state instead of the Chaplygin equation of state. The update file is UPDKTA and the input file is INPGTRTA. The cpu time for this problem is 25s on a Cray X-MP computer.

5. FUTURE WORK

5.1. Physical Models

With the exception of the available equations of state, CAVEAT-GT currently lacks many of the physical models found in CAVEAT [1]. Because of the similarity of the two codes, no difficulties are envisioned in adapting the physical models contained in the CAVEAT code to the general topology version. This will simply require modifying the models so that they are compatible with cells having arbitrary numbers of sides. Here we have in mind the high explosive (HE), molecular and turbulence viscosity, and strength models.

There are three high explosive models available in CAVEAT. They include a programmed-burn, the Chapman-Jouget (CJ) volume, and the Forest Fire burn models. These models simulate the energy released from high explosive materials in the presence of a propagating shock wave. The programmed-burn model is the most empirical of the three. It would be the first model implemented into CAVEAT-GT. By specifying a detonation speed based on data, the arrival time of the detonation front at a computational cell may be inferred. In CAVEAT, the effects of obstacles (i.e., shadowing) are included. As the wave sweeps over each computational zone, energy is deposited into the cell. This energy deposition combined with the equation of state for the material results in a pressure pulse in the cell. There is, however, no feedback between the material motion caused by the energy release and the wave propagation in this model. The BKW and JWL equations of state typically are used with this model. These equations of state already have been implemented into CAVEAT-GT. The CJ Volume Burn and Forest Fire burn models are reactive burn models.

Including a molecular viscosity in CAVEAT-GT would require the addition of the appropriate terms in the momentum and energy equations. Turbulence may be simulated using the simplest representation for the turbulent shear in the form of a constant eddy viscosity. A more sophisticated representation, using transport equations for the turbulent kinetic energy and dissipation length scale (TKE model), also has been implemented in the CAVEAT code. No difficulty is anticipated in transferring the turbulent models available in CAVEAT into the general topology version of the code.

Material strength effects have been combined with the Godunov numerical method in CAVEAT. An elastic-plastic model using a bilinear stress-strain response curve has been implemented, with both kinematic and isotropic hardening effects included. The modifications require solving differential equations for the stress field, as well as the back stresses and yield surface radius at the cell centers. The Riemann solution then is modified to include the cell-side normal and shear stresses. The implementation of this model into CAVEAT-GT should be straightforward.

Implementation of additional physical models such as heat conduction and diffusional radiation heat transfer has been under consideration for the CAVEAT code, and therefore might also be included in CAVEAT-GT.

5.2. Vectorization

Because the current code is more of a development than a production code, no special procedures to allow vectorization have been implemented. From the timing routines included in this code, it is seen that around half of the computer time is spent in the two rezone procedures. Therefore, several modifications to improve the vectorization of subroutines REZLAGR, REZONE, REZCJRS, and MESH have been proposed, but as yet they have not been implemented.

5.3. Post-Processor Code

At the present time, the post-processor code has not been written. Therefore, the main code CAVGTR does not currently write the INPGTP file as shown in Fig. 31. Because of the general topology of the CAVEAT-GT code, some of the graphic routines (for example, contour plots) are time consuming. Therefore, it would be desirable if the main code could write results to a disk file instead of stopping to generate graphic output. The post-processor code could then read this disk file and generate the graphical output after the main code has completed the calculation. This would greatly speed up the main code for cases where large amounts of graphical output are desired. It would also allow the user to generate additional graphical output without rerunning the main code.

ACKNOWLEDGMENTS

The authors wish to extend their appreciation to Hans Ruppel, who provided the encouragement and support for this endeavor. Gratitude also is extended to Adrienne Rosen for providing assistance in the preparation of this manuscript.

REFERENCES

[1] F. L. Addessio, D. E. Carroll, J. K. Dukowicz, F. H. Harlow, J. N. Johnson, B. A. Kashiwa, M. E. Maltrud, and H. E. Ruppel, "CAVEAT: A Computer Code for Fluid Dynamics Problems with Large Distortion and Internal Slip," Los Alamos National Laboratory report LA-10613-MS (1986).

[2] J. R. Baumgardner, Los Alamos National Laboratory, private communication, 1988.

[3] "The Free-Lagrange Method," Proceedings, 1st Int. Conf. on Free-Lagrange Methods, Hilton Head Island, South Carolina, 1985, M. J. Fritts, W. P. Crowley, and H. Trease, Eds. (Springer-Verlag, Berlin, 1985).

[4] J. K. Dukowicz, *J. Comput. Phys.* **61**, 119-137 (1985).

[5] A. M. Winslow, *J. Comput. Phys.* **2**, 149-172 (1967).

[6] R. Chandra, "Conjugate Gradient Methods for Partial Differential Equations," Yale University Thesis, University Microfilms, Ann Arbor, Michigan (1978).

[7] J. U. Brackbill and J. S. Saltzman, *J. Comput. Phys.* **46**, 342-368 (1982).

[8] J. K. Dukowicz, *J. Comput. Phys.* **54**, 411-424 (1984).

[9] J. D. Ramshaw, *J. Comput. Phys.* **59**, 193-199 (1985).

[10] J. K. Dukowicz and J. W. Kodis, *SIAM J. Sci. Stat. Comp.* **8**, 305-321 (1987).

[11] J. D. Ramshaw, *J. Comput. Phys.* **67**, 214-221 (1986).

[12] G. Voronoi, *J. Reine Angew. Math.* **134**, 198 (1908).

[13] B. Delaunay, Bull. Acad. Sci. USSR (VII), *Classe Sci. Mat. Nat.*, p. 793 (1934).

[14] R. E. Bank, "PLTMG Users' Guide," Edition 4.0, University of California at San Diego, La Jolla, CA (1985).

[15] G. A. Sod, *J. Comput. Phys.* **27**, 1-31 (1978).

[16] J. K. Dukowicz, M. C. Cline, and F. L. Addessio, *J. Comput. Phys.* **82**, 29-63 (1989).

[17] F. L. Addessio, M. C. Cline, and J. K. Dukowicz, "A General Topology, Godunov Method," Proceedings, Particle Methods in Fluid Dynamics and Plasma Physics, *Computer Physics Communications* **48**, 65-73 (1988).

[18] R. R. Karp, Los Alamos Scientific Laboratory report LA-8371 (1980).

*U.S. GOVERNMENT PRINTING OFFICE: 1990-0-773-034/20048

This report has been reproduced directly from
the best available copy.

Available to DOE and DOE contractors from
the Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831
prices available from
(615) 576-8401, FTS 626-8401

Available to the public from
the National Technical Information Service
U.S. Department of Commerce
5285 Port Royal Rd.
Springfield, VA 22161

Microfiche A01

Page Range	NTIS Price Code	Page Range	NTIS Price Code	Page Range	NTIS Price Code	Page Range	NTIS Price Code
001–025	A02	151–175	A08	301–325	A14	451–475	A20
026–050	A03	176–200	A09	326–350	A15	476–500	A21
051–075	A04	201–225	A10	351–375	A16	501–525	A22
076–100	A05	226–250	A11	376–400	A17	526–550	A23
101–125	A06	251–275	A12	401–425	A18	551–575	A24
126–150	A07	276–300	A13	426–450	A19	576–600	A25
						601-up*	A99

*Contact NTIS for a price quote.

Los Alamos National Laboratory
Los Alamos,New Mexico 87545