CLaSSiC Project
Manuscript CLaSSiC-93-32

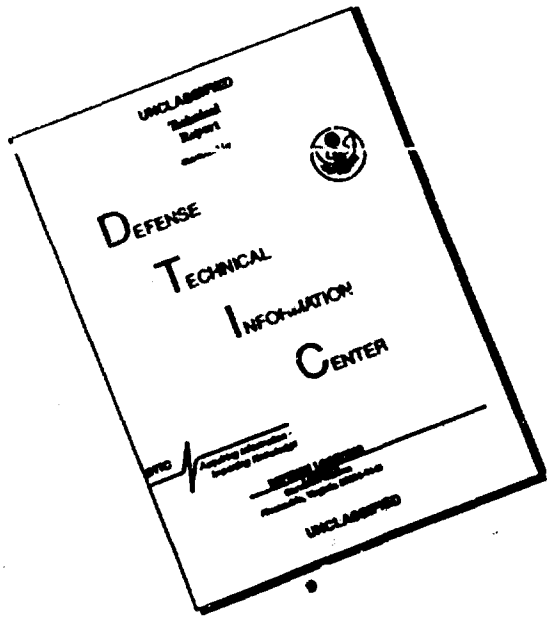
August 1993

3D Composite Grids Using Bézier Curves and Surfaces in Component Adaptive Methods

Ramana G. Venkata
Steven C. Suhr
Joseph Oliger
Joel H. Ferziger

Center for Large Scale Scientific Computation
Building 460, Room 313
Stanford University
Stanford, California 94305

DISCLAIMER NOTICE



THIS DOCUMENT IS BEST QUALITY AVAILABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

3D COMPOSITE GRIDS USING BÉZIER CURVES AND SURFACES IN COMPONENT ADAPTIVE METHODS*

RAMANA G. VENKATA†, STEVEN C. SUHR‡, JOSEPH OLIGER§, AND JOEL H. FERZIGER¶

Abstract. We discuss the use of component adaptive methods for the solution of time-dependent partial differential equations on complex geometries. After a brief exposition of the system and language requirements for an efficient use of these methods we describe a grid generation procedure M*E*S*H, built upon Bézier constructs, coupled with a graphical editor VOUS to serve as a visualization platform, and a programming language Vorpal designed to facilitate their implementation. We also discuss the interaction of the various components of our system and their interfaces.

Key words. component adaptive methods, composite grids, 3D grid generation, NURBS, interactive input/visualization, programming language

AMS subject classifications. 65M50, 65D17, 68N15

1. Introduction. We will discuss a software system which uses component adaptive methods to solve large-scale scientific problems on complex geometries, whose solutions are governed by time-dependent partial differential equations. These methods are designed to provide an efficient representation of the problem domain and its solution. We will describe the basic methods in section 2. In section 3 we will summarize the system and language needs identified in trying to implement these methods in an interactive scientific computing environment. In section 4 we will describe the grid generation module M*E*S*H, which was built using the Bézier curves, B-Splines and NURBS as the primary means of representation of the domain geometry into the system. In section 5 we will discuss an interactive visual editing system, VOUS, which can be used to input the definition of the domain and for program development. In section 6 a language Vorpal, designed to solve some of our other implementation difficulties, will be described.

2. An Overview of Component Adaptive Methods. Introducing the notation for our discussion, suppose that the problem we wish to solve is written as

$$u_t = L\,u + f \quad on \quad \Omega \times [0,T] \tag{1}$$

$$u(0) = u_0 \quad on \quad \Omega \tag{2}$$

$$B\,u = b \quad on \quad \partial\Omega \times [0,T] \tag{3}$$

where $\Omega \subset R^d$ is a bounded domain in the physical space and L is a partial differential operator on Ω. We assume this to be a well-posed initial boundary value problem. Let Ω_h be a discrete grid on Ω and $\partial\Omega_h$ a discretization of $\partial\Omega$. Let $v_h(t)$ be a grid function defined on Ω_h and $\partial\Omega_h$ at time $t = 0,\ k,\ 2k,\ \ldots$.

We will now discuss the use of difference methods on these grids. Without specifying a particular choice of method, let us assume that it is stable and p-th order accurate. The necessary form of stability is discussed in [14]. Without loss of generality, and avoiding complicated notation, we write

* This work has been supported by the Office of Naval Research under grants N00014-90-J-1344 and N00014-89-J-1815. Some of the funds for the support of this study have been allocated by the NASA-Ames Research Center, Moffett Field, California, under Interchange No. NAC 2-440 and by NASA via Contract NAS 2-13721 between NASA and the Universities Space Research Association (USRA).

† Department of Computer Science, Stanford University, Stanford, CA, 94305 (ramana@sccm.Stanford.EDU).

‡ Research Institute for Advanced Computer Science (RIACS), NASA Ames Research Center, Moffett Field, CA 94035-1000 (ssuhr@riacs.edu).

§ Department of Computer Science, Stanford University, Stanford, CA, 94305 (oliger@sccm.Stanford.EDU).

¶ Department of Mechanical Engineering, Stanford University, Stanford, CA, 94305 (ferziger@sccm.Stanford.EDU).

our method in explicit one-step form as

$$v_h(t+k) = L_h\, v_h(t) + k\, f_h(t) \quad on \quad \Omega_h \times [0,T]_k \tag{4}$$

$$v_h(0) = u_{0h} \quad on \quad \Omega_h \tag{5}$$

$$B_h\, v_h(t) = b_{h,k} \quad on \quad \partial\Omega_h \times [0,T]_k \tag{6}$$

where we use subscripts to denote projections onto the appropriate grids. If u_h is the projection of the solution of the system (1) - (3) onto Ω_h, since we have assumed our method to be p-th order accurate, then

$$u_h(t+k) = L_h\, u_h(t) + k\, f_h(t) + \tau_h(t) \quad on \quad \Omega_h \times [0,T]_k \tag{7}$$

$$\tau_h(t) = O(k\, h^p) \tag{8}$$

where τ_h is the local truncation error. We will use this same notation on the piecewise uniform grids we will discuss next.

2.1. Component Grids. We begin by forming a base *composite grid*

$$G_0 = \bigcup_j G_{0,j} \tag{9}$$

which will be characterized by a discretization parameter h_0. This is illustrated in Figure 1 where

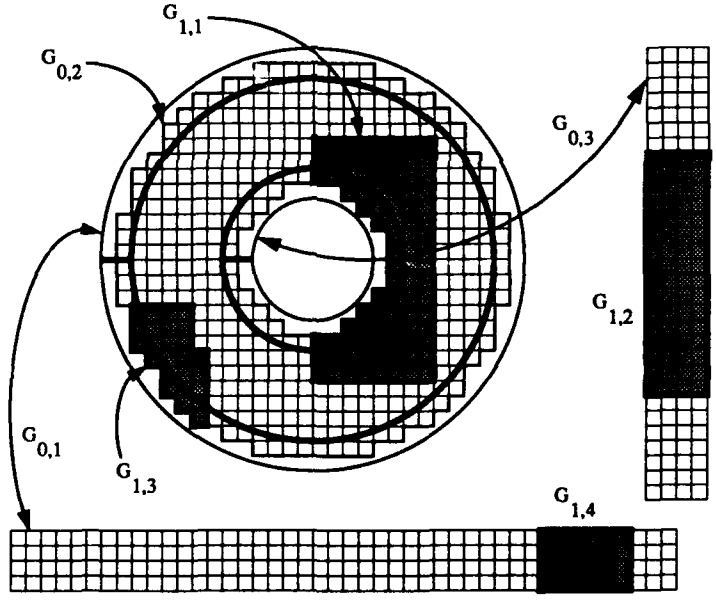


FIG. 1. *Adaptive Composite Grid Structure*

G_0 consists of the *component grids* $G_{0,1}$, $G_{0,2}$ and $G_{0,3}$ (ignore the shaded regions for the moment). $G_{0,2}$ is a stairstep grid with grid lines parallel to the coordinate axes. No coordinate transformation is needed to solve our equations on this grid. The curvilinear grids $G_{0,1}$ and $G_{0,3}$ are defined by

specifying their boundaries and cuts. Rectangular grids in computational space are then mapped onto these grids in physical space using coordinate transformations. The equations are also correspondingly mapped and solved in computational space. To reduce clutter in Figure 1, grids $G_{0,1}$ and $G_{0,3}$ are shown only in computational space.

Although the regions are discretized with equally spaced grid lines parallel to the axes in the computational coordinate system, the lines in the physical space are in general unevenly spaced. With its dual role of controlling the convergence of the method while maintaining stability, the discretization parameter h_0 characterizes the grid spacings in two ways:

(i) The distance between neighboring grid lines in ***physical space*** is at least h_0.

(ii) The distance between neighboring grid lines in each coordinate direction in the ***computational space*** is proportional to h_0.

The component grids are chosen so as to obtain a sufficiently accurate representation of $\partial\Omega$ by $\partial\Omega_h$. h_0 is an estimate of the step size required to obtain a sufficiently accurate approximation of the solution over at least some specified fraction of the domain.

The most general grids intended to be supported in our system are mapped grids which are stairstep grids in the computational coordinates. We have illustrated our component grid structure in R^2; the generalization of stairstep grids to R^3 is done in the obvious way. Curved grids are generalized to R^3 by mapping rectangular parallelepipeds in the computational space into the physical space.

2.2. Component Adaptive Grids. The component grids described in the last subsection were chosen to describe the domain and its boundary. In this subsection, we will describe the use of adaptive grids which are created and destroyed during the course of the computation in order to maintain sufficient accuracy through the entire domain.

During this process, we will create $L-1$ additional refinement levels of composite grids G_l, $l = 1, 2, \ldots, L-1$, on top of the base grid G_0. These will have spatial discretization parameters $h_l = h_0 / m^l$, where usually the refinement factor $m = 2, 3$ or 4. We have occasionally used L as large as 8 but usually $L = 2, 3$ or 4. As dictated by the nature of the problem and the numerical algorithm, we maintain an appropriate relationship between the spatial and temporal discretization parameters, h_l and k_l. In particular, for problems which are essentially hyperbolic in character, we usually use the same mesh ratio on all levels, i.e.

$$\lambda = k_l / h_l = constant \tag{10}$$

We require that these grids be level nested, i.e. the region G_l is fully contained *within* the region G_{l-1}. See the shaded refined grids $G_{1,1}$, $G_{1,2}$, $G_{1,3}$ and $G_{1,4}$ in Figure 1 (to reduce clutter, we show the refined grids $G_{1,2}$ and $G_{1,4}$ only in computational space). Since these grids have the same mesh ratio, they have the space-time structure illustrated in Figure 2. We take several time steps on the finer grid for each time step on the coarser grid.

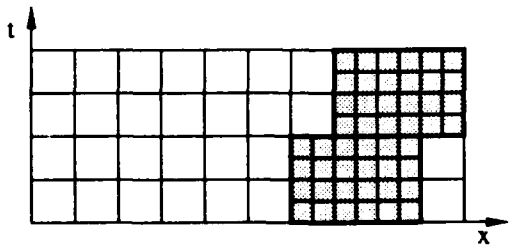


FIG. 2. *Space-time grid structure*

We generate the refinement grids in response to computed error estimates. If the solution is sufficiently smooth, we can use a variant of Richardson extrapolation, called ***step-doubling***, to

estimate the local truncation error τ_h. If v_h is the solution on an $h-k$ grid and v_{2h} is the solution on a $2h-2k$ grid, then it can be shown that

$$\tau_h = \frac{v_h - v_{2h}}{2(2^p - 1)} + O(k\,h^{p+1}) \tag{11}$$

Since τ_h can also be written as

$$\tau_h = k\,h^p\,r_h \tag{12}$$

where r_h is the projection of a smooth function (*which is bounded independent of* h) onto the h-grid, we can compute τ_h using (11) and then r_h using (12) at any given grid point. Since we know r_h, if we then require that h be chosen such that

$$|\,\tau_h\,| \leq \delta \tag{13}$$

for a suitable local error tolerance δ, we can use (12) and (10) to obtain the bound

$$h \leq (\delta / (\lambda\,|\,r_h\,|))^{1/(p+1)} \tag{14}$$

We can then find the largest h_l which satisfies this inequality. Alternatively, (13) may simply be used as an error criterion in the grid-modification procedure described below.

2.3. Adaptive Grid Generation and Integration. With this basic background, we will now explain the adaptive grid generation and integration processes. It is important to organize the data structure for the adaptive composite grid in terms of *connected components*, i.e. connected sets of component grids at each level. Since the grids are level-nested, this organization yields a tree based on containment. See Figure 3 for the tree representing the grid structure in Figure 1.

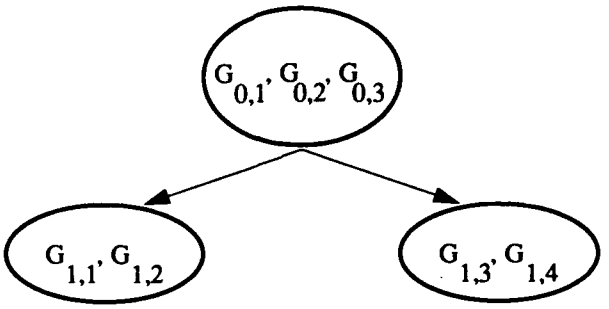


FIG. 3. *Tree structure of connected components*

Evaluation of h_0. Having begun with an estimated initial value of h_0, we perform a trial integration on this grid and estimate the error. If more than a given fraction, say 1/2, of the grid fails to meet the error criterion and needs to be refined, we refine the whole grid with h_0 reduced by a factor of m, repeating this process if necessary. (This overall refinement is also done during the course of the computation, if appropriate.) The fraction 1/2 is used because of the overhead associated with grid refinement. It has been ascertained experimentally that about 3/4 of a domain can be refined (with the associated overhead) at the same computational cost as refining the entire domain. Once h_0 is established or reevaluated, the other h_l are established.

Time integration. The solution is advanced in time as follows. The basic operation is to solve on a connected component. A time step of k_0 is first taken on G_0, and then each grid G_l, $l = 1, 2, \ldots, L-1$ is in turn advanced by k_l, connected component by connected component. Then G_{L-1} is brought up to $t+k_{L-2}$ and so on, until all of the grids are brought up to $t + k_0$. Whenever a refinement is brought up to the time level of the next coarser grid, the point values on the coarser grid under the refinement are replaced by averages of the solution on the refinement. Values of the solution on the interior edges of connected components at each level are obtained by interpolating in time the values already obtained on the next coarser grid. It is important that this interpolation be accurate to order p to maintain the optimal rate of convergence.

Grid modification. Errors are estimated at fixed intervals of time steps on each grid level, usually every 4 to 8 steps, by a process of trial integration. Points at which the error criterion is near violation are flagged, clusters of the flagged points are formed, and refinements overlaying the clusters are constructed. Once a new component grid is defined, the solution at previously refined points is retained, and the solution at all other points is obtained by interpolation accurate to order p from the parent grid. (At the initial time, in contrast, the solution at each refinement level is obtained from the initial data u_0.) The refined grids are constructed with buffer zones around the flagged points, sufficiently wide so that a phenomenon requiring refinement cannot move off the refined area before the next regridding. This is necessary to maintain accuracy and to prevent instability [4]. The grids are *moved* by constructing a new grid and deleting the old one, see Figure 2. Each refinement level is in turn refined as necessary in the same manner, until the finest level has been reconstructed.

Component Methods. The idea of component methods is quite simple within the current framework. We need not use the same solver or even solve the same equations on every grid. For instance, if we are solving a problem with isolated shocks and boundary layers, we can use a high order method where the solution is smooth and a special shock tracking or fitting method in the neighborhood of shocks, or solve inviscid equations outside of the boundary layers and viscous equations inside. We label the individual grids with the solver to be used therein, and provide a collection of appropriate solvers

3. System and Language Needs. The implementation and use of component adaptive methods poses several challenges. Since these are usually large computations that run for long periods of time on remote machines, it is convenient to be able to run them *interactively* so that the user can check on the progress of the calculations from a desk-top workstation, or ask for certain intermediate results to be downloaded to a visualization station. The user may wish to interrupt the simulation and modify the model or data, before continuing. So a certain amount of real-time control is desirable.

The geometrical definition of the base grid which covers the domain is a human-intensive task. It is essential to have interactive tools to aid the user in the specification, visualization and organization of the data.

As a secondary motivation for including interactivity in our system, we believe that in the future, programs will often be visualization-driven. In response to the request, "show the temperature field in region A (specified by drawing on the screen) at time t within an error tolerance of ε," a system would automatically perform the calculations needed to deliver the desired result. Our current work anticipates extensions in this direction.

Once the geometrical definitions of the bounding curves and surfaces of the domain are input to the system, we need a grid generator to compute the parametrization (mapping) of the various component grids and the corresponding transformation metrics. This requires an appropriate choice for the representation of the geometrical objects and access to a variety of grid generation algorithms to tackle widely different domains. Efficient algorithms for the coordinate inversion are also required.

The adaptive model requires a lot of software to manage the grid structure. It is too expensive to rewrite this code for every application. So we need a system which can be adapted to a diverse range of problems and which has an easily-used user interface. These grid systems require complicated data structures and extensive run-time storage management. We need to be able to refer to the structures abstractly in order to avoid error-prone repetition of lengthy expressions. For these reasons we have designed a grid generation module M*E*S*H [21], a graphical editing system VOUS [16], and a language Vorpal [20]. We will describe these tools and their uses in the present context. The interaction of the various components of the system is shown in Figure 4.

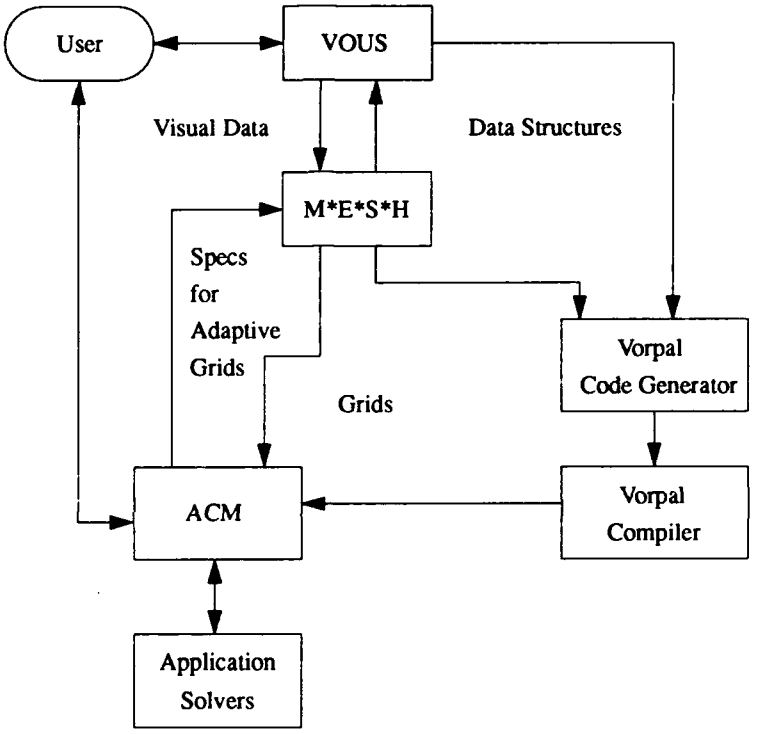


FIG. 4. *ACM software system*

M*E*S*H generates the required grids from the specifications of their bounding curves and surfaces, which are defined by the user in terms of Bezier constructs such as NURBS. The user interacts with the system through VOUS to define the problem domain, and by providing an application solver. The *Adaptive Component Method* (ACM), written in Vorpal and largely application-independent, manages the generation of refinements and performs error estimation and interpolation between grids. ACM makes use of M*E*S*H to generate refinement grids as needed, and it calls the user-defined application solvers to update the solution on connected components. Although we currently write most of our solvers in FORTRAN, it may be easier to do so in Vorpal in the future.

4. M*E*S*H.

4.1. Introduction. Numerical simulation of a flow is most convenient to implement if the flow domain is rectangular and flat. A simple orthogonal rectilinear grid can be laid on the domain and

the discrete difference equations solved subject to the boundary conditions, which are easily and accurately applied under such conditions. For non-rectilinear domains, the discretization should ensure proper application of the physical data at the boundary in order not to generate spurious solution components. The most general type of domain discretization that could be used is an *unstructured* grid, as in finite element methods. The mesh elements can be either quadrilateral or triangular (in 2D). The unstructured nature requires keeping track of a large amount of *topological adjacency* information, creating a substantial storage overhead. While this directionally-unbiased grid structure has certain advantages (applicability to a wide variety of domains, better algorithms for load distribution for problem-solving on parallel machines etc.), at this point, there aren't very many fast, accurate solvers available. Various researchers are working on the generation of triangular and tetrahedral meshes around complex shapes using algorithms based on Delaunay triangulation [2], [11].

To avoid the overhead problems and to be able to use the vast number of existing flow solvers that require a more structured formulation, we use a *boundary-conforming* curvilinear grid to overlay the domain. The gridlines are required to be nearly orthogonal so as to utilize the discretization formulations (for the equations) with minimal error. A rectilinear grid in *computational space* is then mapped to this curvilinear grid in *physical space* using a *coordinate transformation*. Various types of algebraic, elliptic and hyperbolic grid generation algorithms can be used to achieve this mapping.

However, it is often not possible to overlay the entire domain with a *single* grid since we would like to avoid problems such as

(i) arbitrarily high/low local gridline density dictated not by the physics, but by the geometry,

(ii) degeneracies in the transformation, when grid lines belonging to families which should be nearly orthogonal are nearly parallel instead, and

(iii) singularities in the Jacobian of the transformation.

In 2D, for example, singularities are caused by the mapping of a rectangular region in the computational space to a region with a different number of corners in the physical space, since a *continuous, non-singular* mapping cannot introduce new singularities (corners), nor smooth out existing ones.

These problems can be overcome by decomposing the domain into *multiple* regions, which are then individually discretized while retaining control over gridline-orthogonality and density, ensuring conformity to the physics of the problem. *Curvilinear parallelepiped*[1] grids are then formed in the physical space by mapping an orthogonal parallelepiped grid in the computational space onto each of these regions. The result of this *tessellation* of the physical domain is called a *composite grid*. The flow equations are also correspondingly mapped using the *metrics* of the transformation. They are then solved to the desired accuracy on the grids in the computational space, by using any of the fast dedicated solvers which take advantage of the grids' orthogonal, structured nature.

While one can use either *patched* (with zero overlap) or *overlapping* component grids in a composite grid, we chose the latter in our system due to its better functionality in terms of

(i) smoothness of the component grids,

(ii) topological flexibility, and

(iii) provision for moving domain boundaries.

During the global iteration, boundary information is exchanged by neighboring grids in the overlap zones. After an internal iterative cycle is completed, the updated information on the interior grid boundaries is interpolated from the interior of the proper overlapping grid and internal iterations performed again. The global iteration proceeds until the solution achieves convergence everywhere.

One needs to emphasize here that this *domain decomposition* is implemented due to the *geometrical complexity* of the original domain as opposed to other well-known reasons, such as achieving

[1] A curvilinear parallelepiped is defined as a polyhedral volume with six curvilinear quadrilateral faces; the faces don't need to be planar.

computational speed-up with parallel processors or for dividing the domain into zones based on the physical nature of the flow. Of course, our decomposition does not preclude sub-decompositions for those reasons.

Composite grids were used by Atta and Vadyak [1] to solve the full-potential equation using a composite-adaptive grid approach on overlapping grids. Rai *et al.* describe a composite grid method for the unsteady Euler equations using touching grids in [10] and for the compressible Navier-Stokes equations using overlapping grids in [18]. In incompressible flows, because of the form of the continuity equation, conservative exchange of information in the region of overlap becomes difficult [13]. Henshaw and Cheshire [8], [5], [6] describe composite grid methods and their generation on overlapping grids. Wijngaart [23] describes a composite grid method for incompressible flows in two dimensions. This work follows the last and relates to flows in three dimensions.

We will first describe the various constructs that are created in our composite grid system to convey the geometric and the boundary description of the flow problem into the system and to enable the grid generation. We will then discuss some related issues that arise from our use of the multiple grids.

4.2. Data Structures of the Composite Grid. Given a complex three-dimensional domain in physical space and the flow problem described therein, the *grid generation procedure* consists of the following steps:

(i) incorporating the geometric data, which constitutes the domain description, into the composite grid system,

(ii) specifying the boundary conditions applied to the equations that describe the flow problem, and

(iii) generating the set of discrete grids along with all the associated metrics as well as the boundary information.

We will now describe the various constructs employed in this procedure illustrating them in a simple, hypothetical 3D back-step flow domain in Figures 5 and 9.

4.2.1. Surfaces. The domain in physical space is divided into regions which are called *volumes*. These volumes are defined by their bounding *surfaces*, each of which is a curvilinear quadrilateral. In most instances, these surfaces can be defined by their bounding *curves*. A curve is any segment of the boundary contour of the domain that is parameterized by a single parameter. Thus, a curve is strictly a geometric feature. Once a surface is defined by specifying its bounding curves, an algebraic grid generation method, such as transfinite interpolation of these curves, results in a definition of the interior of the surface. Thus, knowing the parametric definition of each curve (which establishes the mapping between the curve parameter and the physical coordinates at every point along the curve), as well as the interpolation function from the edges into the interior of the surface, one can establish the relation between the parametric coordinates and the physical coordinates at any point on the surface, i.e. parameterize the surface.

While the algebraic grid generation is a fast and efficient algorithm to generate the interior parametrization from the boundary data, in some instances, one needs more control over the smoothness and distribution of the gridlines in the interior. In such cases, as in narrow regions, we use *elliptic* grid generation. A Laplacian or a Poisson (with suitably chosen control functions) system of partial differential equations with the curvilinear coordinates as the dependent variables is assembled and solved iteratively. The interior generated by the transfinite interpolation technique is used as the initial guess for this iterative solution.

A surface with important interior features may also be more directly defined as a tensor product surface (a surface formed by the sweep of a Bézier curve, with each of its control points moving along Bézier curves), a bicubic B-Spline surface (a surface composed of a collection of bicubic patches, with continuity conditions similar to those used in B-Splines) or by explicitly providing an external

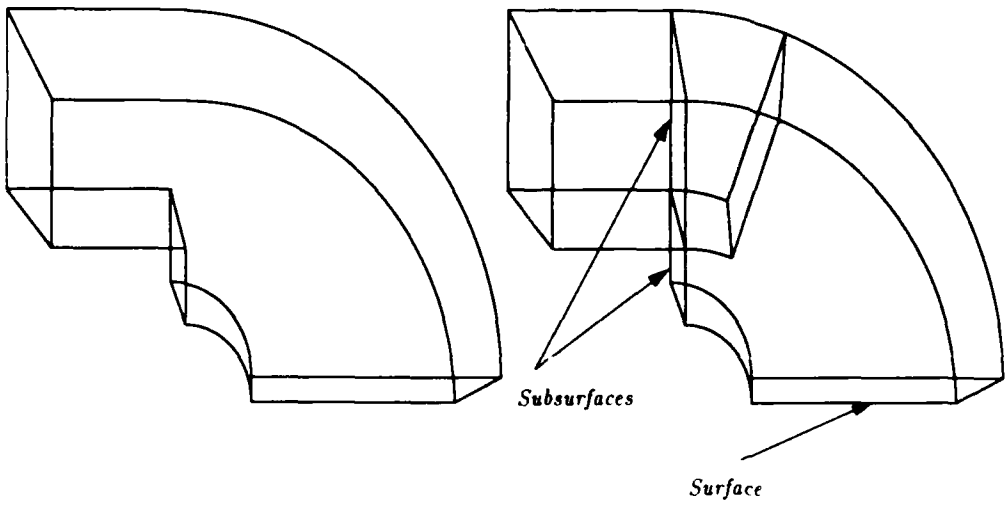


FIG. 5. *Physical Problem—Geometry and Boundary Data*

description. The surface construct contains *only* the geometric definition of the boundaries of the volume, and carries *no* data pertaining to the boundary conditions.

$$surface = (one.of) \begin{bmatrix} transfinite\ interpolation\ between\ curves \\ bicubic\ B\text{-}Spline \\ elliptic\ or\ external\ definition \end{bmatrix}$$

FIG. 6. *Description of a surface*

4.2.2. Curves. From the above description, it is evident that the types of constructs allowed for the definition of the curve largely determine the ease and range of applicability of our composite grid system to practical domains. We chose the Bézier family of curves, such as

(i) Bézier curves—for most normal curves,

(ii) B-Splines—for curves of a higher degree, and

(iii) NURBS (Non Uniform Rational B-Splines)—for conics and other rational curves.

as the primary means of representation of curves in our system. We will now briefly describe the properties of these curves and the reasons for our choice [7].

1. A Bézier curve is defined by the following[2]

de Casteljau algorithm:

Given: $b_0, b_1, \ldots, b_n \in E^3$ and $t \in \Re$, set

$$b_i^r(t) = (1-t)b_i^{r-1}(t) + tb_{i+1}^{r-1}(t) \begin{cases} r = 1, \ldots, n \\ i = 0, \ldots, n-r \end{cases}$$

and $b_i^0(t) = b_i$. Then $b_0^n(t)$ is the point with parameter value t on the *Bézier curve* b^n. The polygon **P** formed by $b_0, \ldots, b_n$ is called the *Bézier polygon* or the *control polygon* of the curve b_n and the

[2] E^n is the n-dimensional Euclidean (or point) space and $\Re^m$ is the m-dimensional linear (or vector) space

polygon vertices b_i are called the *Bézier points*. Since the Bézier curve is thus recursively composed of convex barycentric combinations (all weights are non-negative and sum to one), it has the following properties:

(i) convex hull property—the curve lies within the convex hull of the control points. Since the curve also has the *variation diminishing*[3] property, it is said to be *shape preserving*, i.e., wild wiggles not inherent in the data will not arise during curve fitting.

(ii) affine invariance—the curve is invariant under an affine transformation (involving just translation, rotation, stretching and shear). This implies that the Bézier curve resulting from a set of mapped points is the same as that resulting from a mapping of the original curve itself. This feature allows us to avoid redundant specification of curves in the system by using affine transformations of existing curves instead, if possible. By thus reducing the number of primitive objects to be explicitly defined, we can reduce the number of man-hours required to geometrically specify the domain, a human-intensive process at this point in time.

(iii) symmetry—replacing t by $(1 - t)$ has no effect on the curve.

(iv) pseudo-local control—when one of the control vertices is moved, though the whole curve changes, the effect is mostly local.

2. Modeling a curve of a complex shape by use of a single Bézier curve requires a representation of high degree, which is undesirable from a computational standpoint. In such cases, we use *spline* curves which are *piecewise polynomial* curves. In particular, a spline curve, composed of Bézier curve segments, which is specified by using a minimal information set is called a *B-Spline*. The information to be specified is minimized by utilizing the C^n continuity conditions to evaluate the junction points. The B-Spline possesses all of the advantages of the Bézier curve and, in addition, has more localized control along with the ability to represent more complex shapes.

3. However, a B-Spline cannot represent conics, which are a very popular design tool in industry. To represent these and other rational curves, we use NURBS, which make use of the idea that a conic section in E^2 can be defined as the projection of a parabola in E^3 into a plane. Since a NURB is defined in 3D as the projection through the origin of a 4D nonrational B-spline curve into the hyperplane $w = 1$, it has the ability to represent conics. It also has all the aforementioned properties of Bézier curves in addition to greater localized control.

4. The Bézier constructs appear to be a natural choice since these are the representations that are most often used in the CAD industry to design the domains on which we solve our flow problems. However, in addition to the above constructs, we also provide for non-Bézier parametric polynomial representations of curves, to be used where convenient.

4.2.3. Subsurfaces. We have described the means by which geometric domain data is input into the system. The physical connection to the flow problem is established via the concept of *subsurfaces*. A surface is composed of a tessellation of one or more subsurfaces, each of which has

$$subsurface = \left\{ \begin{array}{c} (physical\ role) \\ parametric\ interval\ (two\ of) \begin{bmatrix} (u, u_{min}, u_{max}) \\ (v, v_{min}, v_{max}) \\ (w, w_{min}, w_{max}) \end{bmatrix} \\ surface_{parent} \end{array} \right\}$$

FIG. 7. *Description of a subsurface*

exactly one *physical role* to play in the flow problem. For example, one part of a surface might be a

[3] asserts that a convex polygon generates a convex curve

physical boundary, while another part could be a *periodic boundary*. Those surfaces, or parts thereof, which are not part of the domain boundaries, but are artificially introduced during the division of the domain into volumes are initially given the role of an *auxiliary boundary*. For the boundary-role specification to be complete, each auxiliary boundary has to be further specified as an *interpolation boundary*, *periodic boundary*, *reentrant boundary* etc.

Thus, a subsurface is defined in terms of its parent surface, the parametric intervals it occupies within the parent surface and its physical role. Associated with each of these roles is a means of describing and enforcing that particular boundary role in terms of the variables of the flow problem. The specification of the surfaces and subsurfaces completes the description of the flow problem in

$$\textit{physical role} = (\textit{one.of}) \begin{bmatrix} \textit{physical boundary} \\ \textit{periodic boundary} \\ \textit{reentrant boundary} \\ \textit{interpolation boundary} \end{bmatrix}$$

FIG. 8. *Description of a physical role*

the physical space. We now proceed to the computational space and generate the grids along with all of the required flow information.

4.2.4. Faces. The domain in the discrete space is described in terms of *grids*. The grids are defined in terms of their bounding *faces*, each of which is a quadrilateral region, curvilinear in physical space and rectilinear in computational space. A grid face is constructed by a tessellation of subsurfaces with one continuous parameterization. The parameterization of the face is defined by linearly rescaling the parameterization of the component subsurfaces (which are defined with respect to their parent surfaces). The boundaries of the face are first parameterized in this fashion. Transfinite interpolation is then employed first to obtain a parameterization of the interior of each face from its boundaries and then of the interior of the grid from its bounding faces. This establishes the coordinate transformation from an orthogonal parallelepiped grid in computational coordinates (u, v, w) to the curvilinear parallelepiped grid in physical coordinates (x, y, z). The metrics of the transformation are also computed.

The subsurfaces that comprise a face *need not necessarily* be from the same surface; they need only be contiguous in physical space. Also, a subsurface can be a component of more than one face, as in the overlap region between two grids. Thus, a subsurface, which imparts the flow boundary information to a surface, also provides the geometric description of the face. *This duality aids in conducting information from the physical problem to the numerical problem.*

For the information exchange to be accurate, it is very important to insure that a pair of overlapping grids refer *precisely* to the same physical location in the overlap zone. One could conceive of a situation where two overlapping regions have differing geometric definitions of the overlap zone, as a result of which supposedly coincident locations do not actually coincide. This gives rise to inaccuracies during information transfer. However, in the present system, all of the overlapping components get identical geometric definitions of the overlap zone (which is subject to the accuracies of the individual parameterizations), since the same subsurface brings the geometric information to the relevant face of each component grid. It is important to keep this issue in mind during the inverse transformations also.

4.2.5. Subfaces. So far, we have only provided a means to transfer the geometrical description from physical space to computational space. We now introduce the concept of *subfaces* to facilitate the transfer of flow boundary information to computational space. The subfaces are contiguous parts

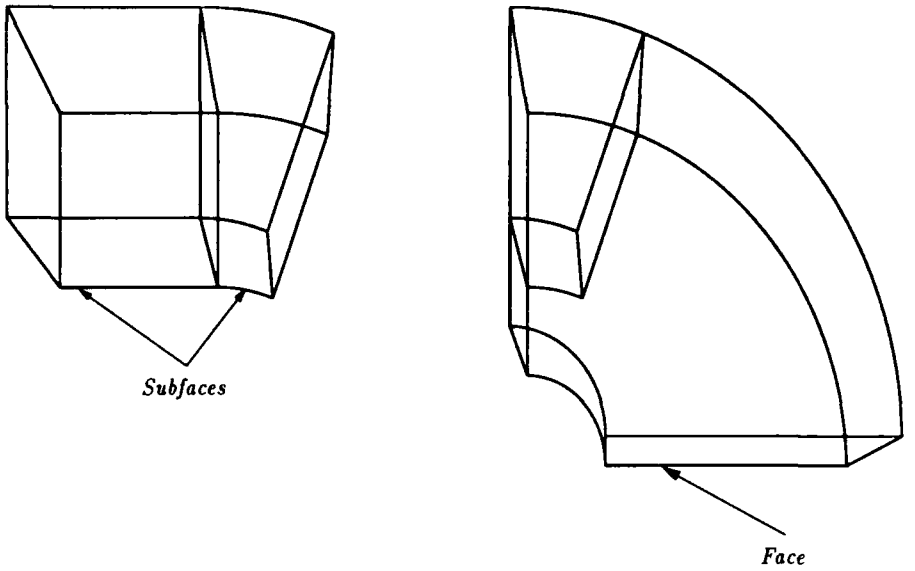


FIG. 9. *Numerical Problem—Geometry and Boundary Data*

of a face, each with exactly one role: ***physical boundary, periodic boundary, interpolation boundary*** etc.

Thus, if contiguous subsurfaces comprising a face (but from different surfaces) have the same boundary role, they would together form a single subface with that role and the associated means of enforcing that boundary condition, be it applying a physical boundary condition or getting the necessary information from a donor grid. Analogous to a subsurface, a subface is defined in terms of its parent face, the parametric interval it occupies in the parent face, and its role. Where the role is that of a physical boundary, we use a ***strongly functionally consistent*** [23] application of the boundary condition, i.e., we *do* apply the physical boundary condition at *all* points that lie on the domain boundaries. This is necessary to ensure that the solution converges both in the numerical as well as the physical sense, to the proper result. Where the role is that of a periodic boundary, an affine transformation linking the target and donor points is specified, so that the boundary condition at the target point can be obtained by evaluating the corresponding operator at the donor point. The process is the same in the case of an interpolation boundary, except that the communication is always between different grids.

For every grid, we also specify the number of grid cells in each coordinate direction.

$$subface = \left\{ \begin{array}{c} (numerical\ role) \\ parametric\ interval\ (two.of) \left[\begin{array}{c} (u, u_{min}, u_{max}) \\ (v, v_{min}, v_{max}) \\ (w, w_{min}, w_{max}) \end{array} \right] \\ face_{parent} \end{array} \right\}$$

FIG. 10. *Description of a subface*

4.3. Summary of grid generation constructs. To summarize:

(i) curves and surfaces are purely geometric features employed to input the geometric description of the domain into the system.

(ii) subsurfaces serve to provide the flow boundary information to the physical problem and the geometrical information to the numerical problem.

(iii) subfaces provide the flow boundary information to the numerical problem.

	physical problem	numerical problem
geometrical info	surfaces	subsurfaces
boundary info	subsurfaces	subfaces

TABLE 1
Functions of the constructs used in grid generation

$$
\textit{Composite grid} = \left\{ \begin{array}{l}
grid_1 \left\{ \begin{array}{l} \left[\begin{array}{l} face_{11} = \left\{ \begin{array}{l} \left[\begin{array}{c} subface_{111} \\ \cdots \\ subface_{11j} \end{array} \right] \\ \left[\begin{array}{c} subsurface_{111} \\ \cdots \\ subsurface_{11k} \end{array} \right] \end{array} \right. \\ face_{12} = \ldots\ldots\ldots\ldots \\ \cdots \\ face_{1m} = \ldots\ldots\ldots\ldots \end{array} \right] \\ (number\ of\ grid\ cells(u,v,w)) \end{array} \right. \\
grid_2 \left\{ \begin{array}{l} \left[\begin{array}{l} face_{21} = \left\{ \begin{array}{l} \left[\begin{array}{c} subface_{211} \\ \cdots \\ subface_{21l} \end{array} \right] \\ \left[\begin{array}{c} subsurface_{211} \\ \cdots \\ subsurface_{21n} \end{array} \right] \end{array} \right. \\ face_{22} = \ldots\ldots\ldots\ldots \\ \cdots \\ face_{2m} = \ldots\ldots\ldots\ldots \end{array} \right] \\ (number\ of\ grid\ cells(u,v,w)) \end{array} \right. \\
grid_t \ldots\ldots\ldots\ldots \\
surfaces = \left[\begin{array}{c} surface_1 \\ surface_2 \\ \ldots\ldots\ldots\ldots \\ surface_p \end{array} \right]
\end{array} \right.
$$

FIG. 11. *Hierarchical description of a composite grid*

4.4. Communication between Grids. For the exchange of information at the interpolation boundaries in the overlap zone of two overlapping component grids, the solver for a connected component can use a variant of the *Schwartz Alternating Procedure* [19]. This algorithm, applied to two overlapping domains Ω_1 and Ω_2, consists of

(i) starting from initial conditions as the boundary values on the edge of Ω_1 in the overlap region (thus decoupling the problem),

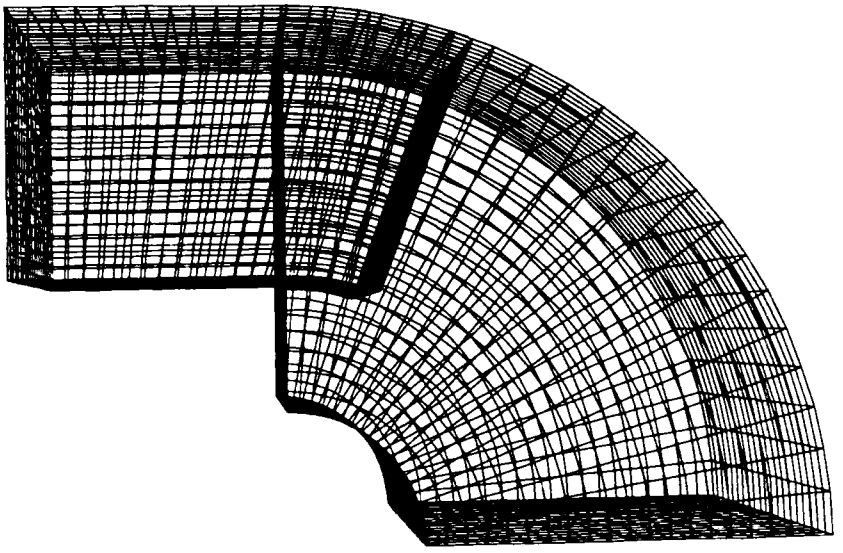


FIG. 12. *Composite grid—only gridlines on faces shown*

(ii) computing the solution in the interior of Ω_1,
(iii) obtaining boundary values on the edge of Ω_2 in the overlap region from the interior of Ω_1,
(iv) computing the solution in the interior of Ω_2,
(v) using this interior solution to determine the boundary values on the edge of Ω_1,
(vi) repeating the cycle till convergence.

We note that no special routines are needed to solve the interface equations, since boundary values are interpolated directly between grids. The convergence of this algorithm has been analyzed by many researchers. Oliger *et al.* [9] found that the strong dependence of the convergence of the algorithm on the amount of overlap can be reduced for elliptic problems using an overrelaxation technique for the boundary values.

The order in which the various component grids are traversed in a connected component depends upon the direction of information exchange between the grids, and thus on the physics. Either an explicit order is provided by the user, or a traversal algorithm based on a suitable criterion is supplied. The ordering also affects the suitability of the procedure for parallel processing. Oliger *et al.* introduce a Black-Red scheme in [9] that is suited for multi-processor machines.

4.5. Coordinate inversion. As mentioned before, the mapping is from computational space to physical space. However, the problem of obtaining the computational coordinates (u, v, w) given the physical coordinates (x, y, z) at a point also arises often during the course of the solution, e.g. in interpolation within the overlap zone. Since the original mapping is non-linear, an iterative technique is employed and the convergence depends on the initial guess. For most grids, the algorithm for determining the initial guess is as follows:

(i) A lookup table consisting of the physical and computational coordinate pairs of a set of *reference points* in each component grid is assembled. The choice of the reference points can be

individually specified for each grid, if need be. The default choice consists of the corners and the *midpoints* of the bounding contours of the grid. The logic behind this choice is the fact that most of the target points are close to the boundary contours, since they belong to the overlap zones which are adjacent to the boundaries.

(ii) Given the (x, y, z) coordinates of the target point, the two nearest neighbors among the reference points are ascertained. Their midpoint is taken as the initial guess.

(iii) If the grid is such that it is wrapped very near to itself, causing potential false choices for the initial guess, the user can specify a different algorithm, to localize the search area, where a quad-tree approach is used while walking along the boundary contours.

5. VOUS. A natural data structure to use for the description of domains, for our grid structures and programs is the *directed graph*. With this in mind, VOUS [16], an interactive, object-oriented, graphical editing system, was developed to provide a platform for the user to organize programs and data as graphs and execute functions on these objects.

The hierarchical nature of the composite grids lends itself to an easy and natural representation as a directed graph. The interfaces between the modules are defined in terms of LISP s-expressions [17] which are convenient and are standard representations for graphs. The domain can be specified as a hierarchical list where the first atom in each list specifies the object to be described in its sublist. Thus, we have a powerful means of representation of the entire domain for future manipulations. Subsequent addition or deletion of other constructs is also made convenient through this data organization.

In order to enable VOUS to serve as a graphical interface between the user and the rest of the system, its design has been extended to include drawing tools and several other features to aid in the specification of curves and surfaces and to ease the interaction between the user and the rest of the system.

(i) The user will input the geometric data for the curves, surfaces and other constructs into VOUS, which can provide a step-by-step guide to insure that all necessary parameters are input. The data can be interactively input and can also be read from previously constructed files. Once a construct is fully described, VOUS will interact with M*E*S*H to compute and graphically display the construct to the user, thus providing a platform for iterative definition of the geometry of the entire domain.

(ii) If the user needs to specify the intersection curve of two surfaces (say, as a bounding curve), VOUS can call the corresponding routine (e.g. Bézier intersection subroutines).

(iii) VOUS will provide the functionality for zooming in/out of regions of interest as well as perform affine transformations such as translation, rotation etc. of the constructs.

(iv) At every point in the process, the user will have access to the options available as well as the relevant help-files.

At the end of the problem specification, VOUS will organize the input data into a data structure that is read by M*E*S*H, which then computes the various component grids and assembles the composite grid required by ACM.

6. Vorpal. Vorpal is a programming language designed for the implementation of ACM and other scientific applications with similar requirements. It supports

(i) high-level data structures such as directed graphs,
(ii) unusual data abstractions such as curvilinear stairstep grids,
(iii) structured input and output streams,
(iv) modular program structure,
(v) parametric program instantiation, and
(vi) interactive execution.

We anticipate that the implementation of ACM and Vorpal will proceed in parallel, and that the final design of Vorpal will be influenced by our experience with the implementation of ACM.

The compiler for Vorpal will be a preprocessor which produces C++ code as its output. In a UNIX operating environment, where C and FORTRAN code are compatible, ACM and other Vorpal programs can have access to FORTRAN common blocks, and they can make use of subprograms written in FORTRAN and C as well as Vorpal.

Type constructors in Vorpal can be applied recursively and freely. The structure of input and output data can be defined in the same way as the structure of internal data. An object whose type is defined with value-like type constructors such as *array* or *set* will have a value which can be stored externally as well as internally. The value of such an object can be printed or read by a single operation, in a form which can be incorporated directly into a program source file as a data constructor. A LISP-like form and a binary form for external data will also be available, and explicit traversal of input or output streams will be allowed.

The initial implementation of a subset of Vorpal will focus on its modular structure and its support for a few predefined high-level type constructors. In addition to the integrated support for input and output, some other central features of this preliminary implementation will be the *automation of memory allocation* and the *integrated support of operations* on user-defined data types. The preliminary implementation of Vorpal will first be applied to simple model problems in an experimental version of ACM.

As Vorpal and ACM become more complete, Vorpal will include support for user-defined type constructors and for generation of Vorpal code in response to user-supplied data. For example, a user with a need for a particular kind of staggered grid should be able to provide a description of the staggering in the grid, from which a custom version of ACM will be constructed automatically. Finally, to increase the level of support for interactivity, and eventually to support concurrent execution, internal processes within a single UNIX process will be simulated. Active objects such as functions and processes can be referred to by variables and used in an object-oriented way.

7. Summary. We have described a software system, composed of a grid generation module M*E*S*H, which is built upon a variety of Bézier constructs, an interactive input/visualization module VOUS, and a language VORPAL with special features designed to be useful in scientific computing applications. We have also described the procedure for the generation of the component grids and their adaptive refinement during the course of the solution. To validate the composite grid generation and communication methodologies, we are currently solving some steady-state, incompressible flows in three-dimensional domains with complex geometries, using a Navier-Stokes solver that utilizes the pseudo-compressibility method [12] and an approximate factorization scheme.

8. Bibliography.

REFERENCES

[1] E. H. ATTA AND J. VADYAK, *A Grid Overlapping Scheme for Flowfield Computations About Multicomponent Configurations*, AIAA Journal, 21 (1983), pp. 1271–1277.

[2] T. BAKER, *Mesh Generation for the Computation of Flowfields Over Complex Aerodynamic Shapes*, Computers Math. Applic., 24 (1992), pp. 103–127.

[3] M. BERGER AND J. OLIGER, *Adaptive methods for hyperbolic partial differential equations* , J. Comp. Phys., 53 (1984), pp. 484–512.

[4] G. BROWNING, H. -O. KREISS, AND J. OLIGER, *Mesh refinement*, Math. Comp., 27 (1973), pp. 29–39.

[5] G. CHESSHIRE, *Composite Grid Construction and Applications* , Ph. D. Thesis, California Institute of Technology, Pasadena, CA, 1986

[6] G. CHESSHIRE AND W. D. HENSHAW, *Composite Overlapping Meshes for the Solution of Partial Differential Equations*, J. Comp. Phys., 90 (1990), pp. 1–64.

[7] G. FARIN, *Curves and Surfaces for Computer Aided Geometric Design—A Practical Guide*, Academic Press, Inc., San Diego, CA 92101, 1988.

[8] W. D. HENSHAW, *Part I: The Numerical Solution of Hyperbolic Systems of Conservation Laws; Part II: Composite Overlapping Grid Techniques*, Ph. D. Thesis, California Institute of Technology, Pasadena, CA, 1985

[9] J. OLIGER, W. SKAMAROCK, AND W. TANG, *Convergence Analysis and Acceleration of the Schwartz Alternating Method*, CLaSSiC Project Manuscript CLaSSiC-86-12, Stanford University, 1986.

[10] K. A. HESSENIUS AND M. M. RAI, *Applications of a Conservative Zonal Scheme to Transient and Geometrically Complex Problems*, Computers and Fluids, 14 (1986), pp. 43–58.

[11] Z. JOHAN, T. J. R. HUGHES, K. K. MATHUR, S. L. JOHNSSON, *A Data Parallel Finite Element Method for Computational Fluid Dynamics on the Connection Machine System*, Computer Methods in Applied Mechanics and Engineering, 99 (1992), pp. 113–134.

[12] D. KWAK, J. L. C. CHANG, S. P. SHANKS, AND S. R. CHAKRAVARTHY, *A Three-Dimensional Incompressible Navier-Stokes Flow Solver Using Primitive Variables*, AIAA Journal, 24 (1986), pp. 390–396.

[13] R. L. MEAKIN, *Application of Boundary Conforming Coordinate and Domain Decomposition Principles to Environmental Flows*, Ph. D. Thesis, Stanford University, CA, 1986.

[14] J. OLIGER, *Stability and error control for component adaptive grid methods*, Proc. of the IMA Workshop on Modeling, Mesh Generation, and Adaptive Numerical Methods for PDEs, Springer Verlag, NY, 1993.

[15] J. OLIGER, W. SKAMAROCK, AND R. L. STREET, *Adaptive grid refinement for numerical weather prediction*, J. Comp. Phys., 80 (1989), pp. 27–60.

[16] J. OLIGER, R. PICHUMANI, AND D. PONCELEÓN, *A Visual Object-Oriented Unification System*, CLaSSiC Project Manuscript CLaSSiC-89-23, Stanford University, CA, 1989.

[17] R. PICHUMANI, J. OLIGER, AND R. G. VENKATA, *Symbolic Expressions of Composite Grid Structures*, CLaSSiC Project Manuscript CLaSSiC-90-25, Stanford University, CA, 1990.

[18] M. M. RAI, *Navier-Stokes Simulations of Rotor/Stator Interaction Using Patched and Overlaid Grids*, Journal of Propulsion, 3 (1987), pp. 387–396.

[19] H. A. SCHWARTZ, *Über einige Abbildungsaufgaben*, Journal für die Reine und Angewandte Mathematik, 70 (1869), pp. 105–120.

[20] S. SUHR, *The Vorpal language for Scientific Computing, Applied to Adaptive Grid Generation*, Ph. D. Thesis (in preparation), Department of Computer Science, Stanford University, CA.

[21] R. G. VENKATA, J. OLIGER, AND J. H. FERZIGER, *Composite Grids for Flow Computations on Compler 3D Domains*, Proc. of the Fifth SIAM Conf. on Domain Decomp. Methods for Partial Differential Equations, (1991), pp. 605-613.

[22] R. G. VENKATA, *Three-dimensional Composite Grid Generation using Bezier Family of Curves and Surfaces*, Second SIAM Conf. on Geom. Design, (1991).

[23] R. F. WIJNGAART, *Composite-Grid Techniques and Adaptive Mesh Refinement in Computational Fluid Dynamics*, Ph. D. Thesis, Department of Mechanical Engineering, Stanford University, CA, 1989.