RIACS

Research Institute for Advanced Computer Science
NASA Ames Research Center

An Accuracy Test of a Cartesian Grid Method for Steady Flow in Complex Geometries

Marsha Berger
John Melton

RIACS Technical Report 95.02
February 1995

To appear in ***Proceedings 5th International Conference on Hyperbolic Problems , Stony brook, NY, June 1994***

An Accuracy Test of a Cartesian Grid Method for Steady Flow in Complex Geometries

Marsha Berger
John Melton

The Research Institute for Advanced Computer Science is operated by Universities Space Research Association, The American City Building, Suite 212, Columbia, MD 21044 (410) 730-2656

Work reported herein was supported by NASA via Contract NAS 2-13721 between NASA and the Universities Space Research Association (USRA). Work performed at the Research Institute for Advanced Computer Science (RIACS), NASA Ames Research Center, Moffett Field, CA 94035-1000

An Accuracy Test of a Cartesian Grid Method for Steady Flow in Complex Geometries*

Marsha Berger† John Melton‡

Abstract

We describe a method to solve the inviscid compressible Euler equations using non-body-fitted Cartesian grids. We briefly survey the geometric procedures needed for the "grid generation" in this approach and show some Cartesian grids for three dimensional configurations. The major source of error in these methods is at the irregular boundary cells. This has motivated us to study various improvements in the numerical scheme (a Godunov method) on a model problem with smooth flow in two space dimensions.

1 Introduction

We describe a Cartesian mesh method for solving the steady inviscid Euler equations for complicated geometries. This approach uses a regular Cartesian grid, with the solid boundaries carved out of the underlying grid in an essentially arbitrary way (see Figure 2.) This technique is sometimes called the "cookie cutter" approach. It leaves a border of irregular cells along the edge of a solid boundary, where special difference schemes are needed. However the rest of the grid is completely regular, allowing us to use highly efficient and accurate finite difference schemes over most of the domain. All the overhead for the geometry complexity is at the boundary. This is not the case for body-fitted or unstructured grids - the other popular ways to deal with complex geometry.

Until recently, Cartesian grid methods have received relatively less attention than the alternative approaches. Previous work includes [13], [7], [11] and [10], and of course TRANAIR [22] (using Cartesian grids for potential flow), a production workhorse at Boeing. In the time dependent case methods have been developed in [5], [6], [19], [20]. Lockheed-Ft.Worth [15] now uses a Cartesian grid method as one of their CFD tools, and a commercial package for "under the hood" automotive calculations is now available as well. Clearly, interest in these methods is growing.

The purpose of this talk is two-fold. We first give an overview of the types of geometric calculations used in generating a Cartesian grid, and present some results from these algorithms for 3D configurations. Secondly, we address the important question of accuracy. Just because we can generate a grid doesn't mean we can compute a good flow solution on it. We start with the simplest method on a 2D model problem, and measure the error in the computation. We then modify various pieces of the algorithm to try to isolate the sources of error, and improve the overall quality of the solution.

Since Cartesian grids are completely irregular at the boundaries of the domain, (where the grid intersects the body), it is not even clear what accuracy and convergence rate to expect. (There is beginning to be some literature on the question of accuracy on non-smooth grids [16], [21]). Cartesian mesh methods are usually applied to such complex problems that the sources of error may never be known in a quantitative way. We feel it is important to study *model problems* to gain an understanding of the behavior of these methods.

Our test problem was first used by [2] in a comparative study of the accuracy of quadrilateral vs. triangular body-fitted grids. We will compare our results with the corresponding body-fitted results in that work. Since for aerodynamic flows we are especially interested in the solution at the boundary, we will measure the accuracy

*To appear, Proc. 5th Intl. Conf. Hyp. Prob., Stonybrook, NY, 1994.

†Courant Institute, 251 Mercer Street, New York, NY 10012.

‡NASA Ames Research Center, Moffett Field, CA 94035.

along the boundary as well as over the entire flow field. A previous study of accuracy was also done in [8] for a slightly different method. We obtain similar results to theirs in this work.

In the next section we give an overview of the Cartesian mesh method. We list the quantities needed for the flow solver, and the geometric calculations needed to provide those quantities. In section 3 we describe our numerical algorithm. Section 4 presents the computational results for steady supersonic flow in a circular channel. We try several modifications of the basic algorithm to improve its convergence rate at the boundary. This work is still in progress, so instead of definitive conclusions we have a brief discussion of the current state of affairs in section 5.

2 A Cartesian Grid Mesh Generator

In this section we describe the geometric operations necessary to define the flowfield geometry to the flow solver. We present this specifically in the context of solving the compressible Euler equations in integral form,

$$\frac{d}{dt}\iiint_{\Omega} U\,dx\,dy\,dz = -\oint_{\partial\Omega} \mathbf{f}\cdot\mathbf{n}\,dS \tag{1}$$

where $U = (\rho, \rho u, \rho v, \rho w, \rho E)^T$ and

$$\mathbf{f}\cdot\mathbf{n} = \begin{pmatrix} \rho\mathbf{q}\cdot\mathbf{n} \\ \rho u\mathbf{q}\cdot\mathbf{n} + pn_x \\ \rho v\mathbf{q}\cdot\mathbf{n} + pn_y \\ \rho w\mathbf{q}\cdot\mathbf{n} + pn_z \\ (\rho E + p)\mathbf{q}\cdot\mathbf{n} \end{pmatrix},$$

where $\mathbf{q} = (u, v, w)$. These equations place no restrictions on the shape of an individual control volume. Over most of the domain the volumes will be the usual Cartesian cubical cell (although since mesh refinement will be used in computing the solution, the volumes of these cells will depend on the level of refinement). Only the body-intersecting cells will be irregular. For these cells special information will be required to accurately compute the flow and impose the solid wall boundary conditions $\mathbf{q}\cdot\mathbf{n} = 0$, where $\mathbf{n}$ is the unit normal to the surface.

This normal vector can be approximated in a cell using the divergence theorem: the sum of the surface areas multiplied by the normal vector over all sides of the volume sum to zero. Since the other sides of the cell are portions of a Cartesian cell, this computation is rather simple, and the area of the solid surface intersecting the cell does not need to be directly computed. A similar application of the divergence theorem simplifies the calculation of the volume centroid as well. In two dimensions, this amounts to approximating the boundary of the solid body by a piecewise linear segment in each cell. Note that in three dimensions, even if the body intersects each face in a straight line, it may still be impossible to find a plane that gives those straight line intersections.

For our numerical scheme (described completely in the next section in two space dimensions), the information needed by the flow solver is:

- area of each cell face (exterior to the solid body),
- face centroid for each face,
- volume centroid for each irregular cell,
- surface normal for the body surface in each cell at the boundary.

This information is found, for the most part, by intersecting Cartesian cells with the description of the body. For the computations here, the body is described by a list of triangles defining the surface. Other representations are possible however, and we have also investigated generating the Cartesian mesh directly from a CAD/CAM system using a NURBS description of the geometry [18]. The operations themselves are conceptually simple though unfortunately not simple to program. We illustrate them pictorially here. More details are found in [17].

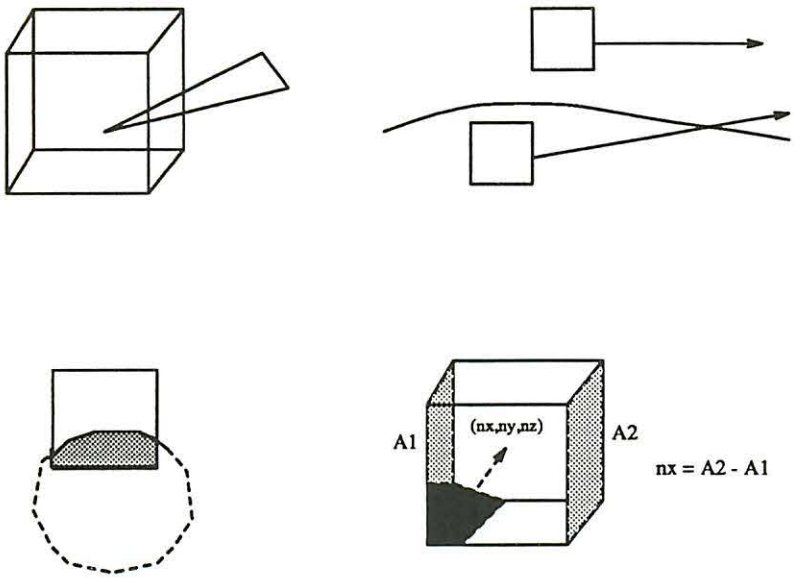

Figure 1: Illustration of some of the steps in Cartesian grid generation.

We first examine each cell to determine whether it is body-intersecting. This is illustrated in Figure 1. This step amounts to computing the intersections of lines and planes. If a cell is not body-intersecting, we determine whether it is inside or outside the flow field domain by casting a ray from the cell center out to the far-field and counting the number of intersections it makes with the surface. This is shown in Figure 1b. For cells that intersect the body, the face polygon is 'clipped' (see e.g. [12]) with the co-planar surface cross-section, using the Sutherland-Hodgman polygon clipping algorithm (illustrated in Figure 1c). The area and centroid of the resulting planar polygon is easily calculated. Applying the divergence theorem with the calculated areas of the 6 planar faces of each cell gives the integrated surface normal and area for the body surface in that cell. This is shown in Figure 1d. At present, the cell intersection information is computed for all cells at a pre-determined finest level of refinement. This is saved in what we call the "database grid". However, not all these cells need to be present in the flow solution. Using curvature criteria, also described in [17], the appropriate level of refinement to describe the body geometry is determined, before the flow solver is applied. (This is as opposed to the dynamic refinement based on the flow solution itself that occurs at various times during the flowfield solution process). It is likely that in future versions of the code, this will be done in a more efficient way.

In Figure 2 we show several views of an F16XL Cartesian grid. It has 277657 cells in all. Of these cells, 20%, or 56205 cells intersect the body. It took approximately 6 minutes on a C90 to generate the grid. We demonstrate that an actual flow solution for a complex configuration in three dimensions can be computed on such grids in Figure 3, which shows the Mach distribution for a supersonic transport test case. The flow solver in Figure 3 uses a Jameson-Schmidt-Turkel type scheme [14]. In the rest of this paper however we solve a two dimensional model problem with a Godunov type scheme to study the accuracy of flow computations on non-body-fitted grids.

3 A Cartesian Grid Flow Solver

In this section we describe a variation of a second order Godunov scheme for use on Cartesian non-body-fitted grids. The scheme is based on piecewise polynomial reconstruction in each cell. Such schemes typically give second order convergence for smooth flows on regular grids. For irregular grids however the usual truncation error analysis, based on cancellation of the leading error terms when differencing the fluxes, breaks down. We will study the convergence rate of our algorithm numerically in section 4. We first describe the basic method (in two space dimensions) applied over the regular portions of the domain. We then describe the modifications necessary for the irregular cells at the boundary of the domain. As much as possible the implementation

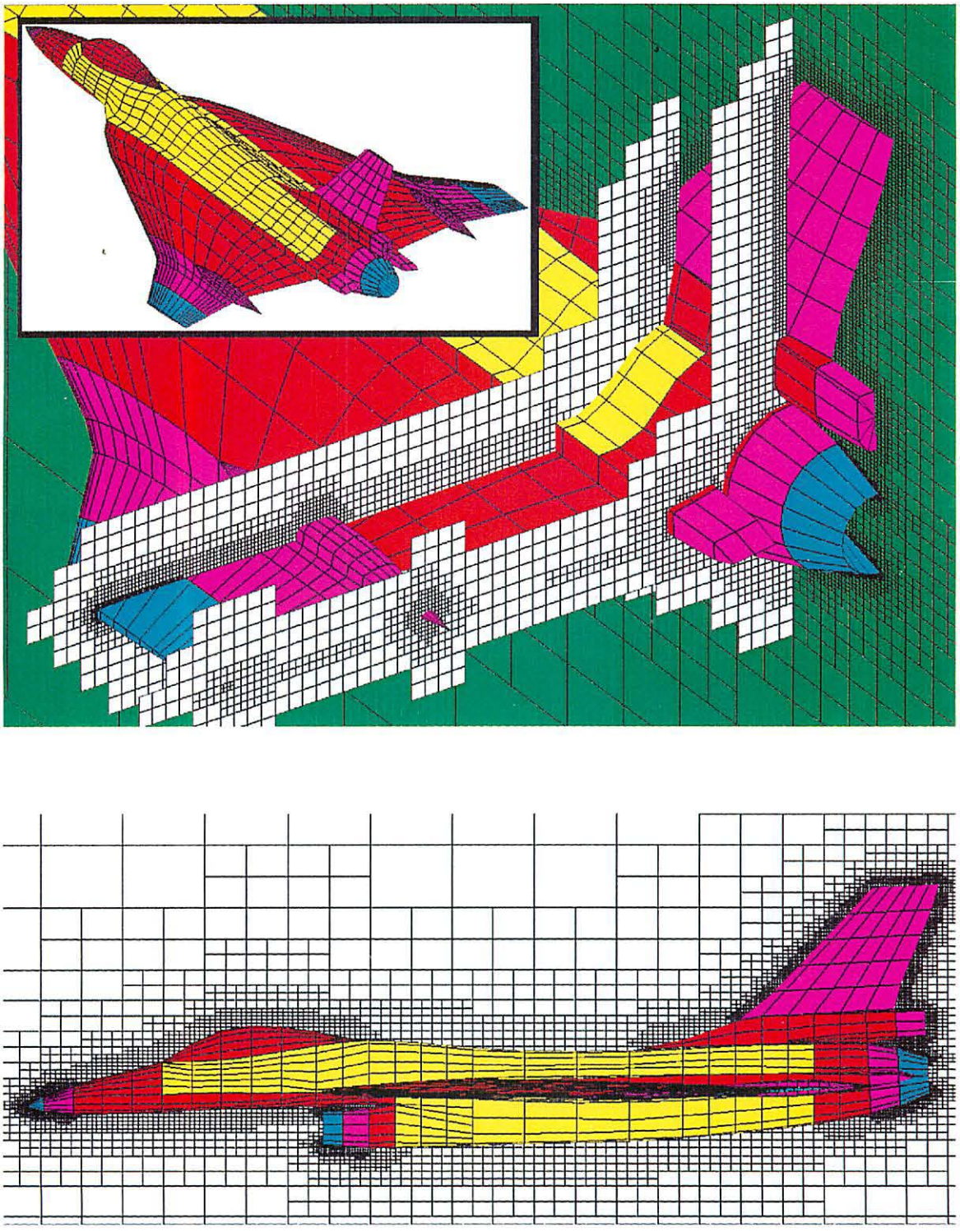

Figure 2: Cartesian Grid for an F16XL.

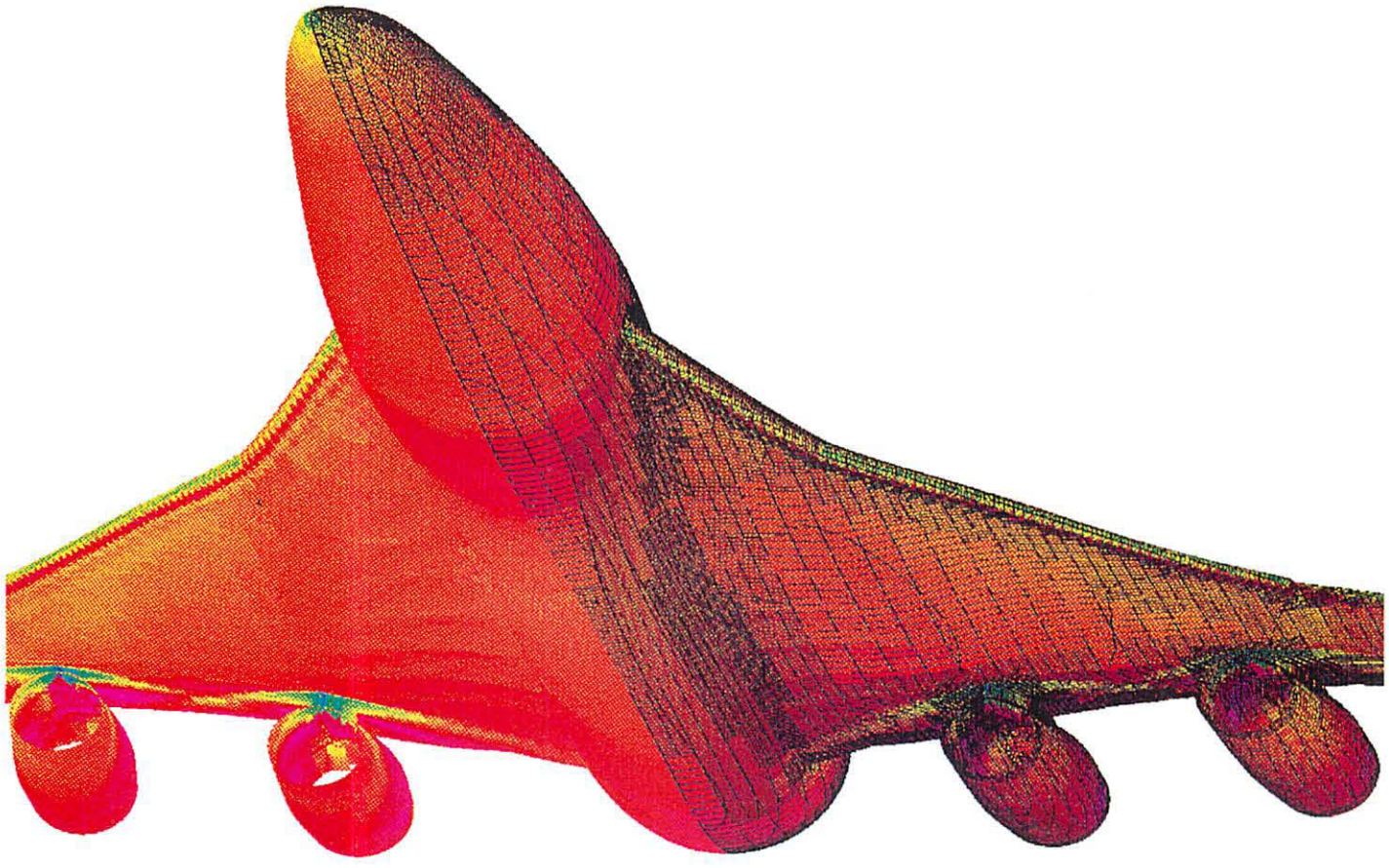

Figure 3: Mach distribution for supersonic transport.

proceeds in the same way: regular (vectorizable) calculations over the entire Cartesian grid (including cells in the "interior" of the body) followed by a "fix-up" pass over the irregular cells at the solid surface.

The basic scheme we use is a high resolution Godunov's method: for each cell we

- Reconstruct the solution in each cell using a piecewise linear function. We use a second order gradient based on central differencing in each direction.
- Predict a left and right state at the midpoint of each cell edge, using the gradient information.
- Solve a Riemann problem to determine a unique state at the midpoint of each edge. The Riemann solver we use is based on [9].
- Evaluate the integral of the fluxes in (1) using the midpoint rule.
- Advance the solution to steady state using a three stage Runge Kutta method with local time stepping.

These steps need to be modified for use at the irregular cells. For example, the midpoint of the cell edge for an irregular cell (centroid in 3D) needs to be calculated during the mesh generation phase described in Section 2. The actual length (in 2D) or face area (in 3D) needs to be calculated then as well. For steady state calculations, the volume of the cut cell is not needed. For the local time step for the irregular cells we take

$$\Delta t = \frac{cfl * A_{full}}{\max(u + c, v + c) * Length_{max}},$$

where $A_{full} = \Delta x \times \Delta y$ is the area of a regular (full) cell, and $Length_{max}$ is the length of the maximum side of a cut cell. For the border of cells *next* to the irregular cells, we use a one-sided second order derivative approximation instead of a central difference.

The most difficult modification concerns the reconstruction of the solution at the irregular cells themselves. Since the cell centroids generally do not line up in neighboring cut cells, we can no longer determine the

differences separately in each direction. We use a least squares procedure to determine the gradient in the following way. For a given cell u_0 we try to fit a linear polynomial so that

$$\bar{u}_0 + (x_j - x_0)u_{0x} + (y_j - y_0)u_{0y} = \bar{u}_j, \tag{2}$$

where the centroid of cell 0 is (x_0, y_0) with cell average $\bar{u}_0$, and the subscript j refers to the same quantities of a neighboring cell. For linear reconstruction, we use neighboring cells that share an edge with the given cell, (unless there are not enough, for example, in the triangular cell case, where we also include diagonal neighbors that share a node). This formulation is automatically conservative, since a linear function through $\bar{u}_0$ at the centroid automatically preserves the cell average. However, for quadratic reconstruction this is not the case, and considerably more work must be done. In our tests with quadratic reconstruction, (the results of which are described in Section 4), we set up a linear system for a polynomial p in cell 0

$$p(x,y) = c_0 + c_1(x-x_0) + c_2(y-y_0) + c_3[(x-x_0)^2 - s_{xx}] + c_4[(x-x_0)(y-y_0) - s_{xy}] + c_5[(y-y_0)^2 - s_{yy}]. \tag{3}$$

In (3), the unknowns are the coefficients c_1 through c_5, and the quantities $s_{xx} = \int_{cell_0}(x-x_0)^2$, s_{xy} and s_{yy} are pre-computed to maintain conservation,

$$\int_{cell_0} p(x,y) = \bar{u}_0 A_0. \tag{4}$$

This is necessary because, unlike the linear case, $p(x_0, y_0) \neq \bar{u}_0$. Given this definition of p, the reconstruction proceeds again in a least squares sense using

$$\int_{cell_j} p(x_j, y_j) = \bar{u}_j, . \tag{5}$$

where the neighboring cells j now include two rings of cells that share edges, starting from the given cell. To compute the integral of the quadratic polynomial, both for (4) and (5), we use a third order accurate approximation. Specifically, the integral over a triangle is approximately the average of the function values at the midpoints of the sides multiplied by the area of the triangle. In two dimensions this is somewhat less work than using the divergence theorem to reduce the computation to an integral around the boundary of the cell. More in depth discussions of high order polynomial reconstruction can be found in [4,3,1].

Despite all the care to accurately compute quadratic reconstructions with the correct cell average properties, we found virtually no difference in the results between using (3) and simply treating the cell averages as pointwise values. Since the former is much more expensive computationally, we would probably not do this again for steady state computations. However, in all the numerical examples, we compare the computed solution $\bar{u_j}$, which is an approximation to the integral average over cell j, with the "exact" integral average over the cell (computed to third order using the approximation mentioned above).

One final point concerns the use of limiters. Since our test problem is smooth, and our goal is to test the accuracy of our reconstruction procedures, we did not use a limiter. When it is necessary, we limit the reconstructed solution over the entire cell so that it does not exceed any neighboring solution value at the cell edge. We note that [2] reports better results using a face based limiter, where the reconstructed solution has a different limiter applied at each face (in 2D, each edge is limited separately).

4 Numerical Results

The two dimensional test problem for our numerical convergence study is found in [2]. We solve the inviscid compressible Euler equations for a supersonic vortex in a channel formed by concentric circular arcs. The boundaries of the channel form one quarter of a circle, with inner radius r_i and outer radius r_o. A smooth analytic solution exists for this problem, so the errors in the computation can be evaluated. The density is:

$$\rho(r) = \rho_i[1 + \frac{\gamma-1}{2}M_i^2\{1 - (\frac{r_i}{r})^2\}]^{\frac{1}{\gamma-1}}, \tag{6}$$

and the velocity varies inversely with the radius. We use the same geometry and test parameters as [2], $\rho_i = 1$, $r_i = 1$, $r_o = 1.384$, $M_i = 2.25$, and $p_i = 1/\gamma$. For all test cases, the program is run until the residuals have

Mesh Size	Second Order		First Order	
	Reg. Quad	Distorted	Reg. Quad	Distorted
31 × 6	.40	.97	16.04	15.77
61 × 11	.10	.43	7.39	7. 31
121 × 21	.03	.19	3.65	3.79
Rate	2.08	1.21	1.10	1.07

Table 1: 1st and 2nd order results of Aftosmis et al.

h	Linear				Constant			
	All cells	Ratio	Bndry. Cells	Ratio	All cells	Ratio	Bndry. Cells	Ratio
$.0533$	9.68(-2)		2.61 (-1)		5.28 (0)		7.65(0)	
$.026\bar{6}$	2.54(-2)	3.81	9.51 (-2)	2.74	2.75 (0)	1.92	4.34(0)	1.76
$.013\bar{3}$	6.23(-3)	4.07	3.71 (-2)	2.56	1.45 (0)	1.89	2.38(0)	1.82

Table 2: 1st and 2nd order Cartesian results for entire flowfield and boundary cells only.

converged four or five orders of magnitude. We measure the actual global error in the following tables. The relevant results of [2] for structured quadrilateral grids are given here as well for comparison purposes. In all cases the error in the density field is measured. Starting from the basic algorithm outlined in the previous section, we will experiment with the reconstruction algorithm, the flux evaluations, and the representation of the boundary itself to try to improve the accuracy of the results at the solid walls.

The first numerical study examines the errors using piecewise linear reconstruction of the solution. Table 1 summarizes the percent relative errors from [2], and includes their linear reconstruction (second order) and piecewise constant (first order) results. Also included are their results using randomly distorted quadrilateral meshes as an interesting aside. In their results, the l_1 and l_2 norms gave the same convergence rates and approximate error, so we only repeat the former. The quantity measured in [2] is

$$Error = \sum_k |\frac{q_{exact} - q_k}{q_{exact}}| A_k \tag{7}$$

where the sum is over all cells k. We will use the same definition in Table 2 to present the Cartesian results. Later we will also use the following measure of relative error in the l_1 norm,

$$\frac{||q_{exact} - q_{computed}||_1}{||q_{exact}||_1} = \frac{\sum_k |q_{exact} - q_k| A_k}{\sum_k |q_{exact}| A_k}, \tag{8}$$

where A_k is the area of the k^{th} cell. This is a direct discretization of the continuous L_1 norm. We also measure the error at the boundary only, defined as

$$\frac{||q_{exact} - q_{computed}||_1}{||q_{exact}||_1} = \frac{\sum_{k\in\partial} |q_{exact} - q_k| l_k}{\sum_{k\in\partial} |q_{exact}| l_k}. \tag{9}$$

In (9) the sum is taken only over the cut cells, and the cell error is multiplied by a length rather than the cell area. Here we take l_k to be the length of the boundary segment in that cell. Note however that a large cell could have a small boundary segment. We have also tried using $\sqrt{A_k}$ in the boundary norm with similar though less clean results. Although the use of a piecewise linear boundary representation in each cell is second order the convergence is not smooth, since the intersection of the boundary with the Cartesian mesh is not regular. We expect the error to be bounded by $const \times h^2$, but we do not expect an asymptotic expansion. Since the norm itself is not smooth, this increases the difficulty in measuring the accuracy of the approximation at the boundary cells.

In our Cartesian non-body fitted grid we use a regular grid spacing for our convergence study starting with $h = .053\bar{3}$ in each direction. There are 294 cells in the flow field on this coarsest grid, 206 full cells and 88

h	2nd Order Linear Recon	Ratio	Match Errors	Ratio	Quad at cut cells	Ratio	Quad at 2 rings	Ratio
$.053\bar{3}$	2.41 (-1)		2.31 (-1)		2.25 (-1)		1.83 (-1)	
$.026\bar{6}$	9.37 (-2)	2.57	9.17 (-2)	2.52	8.88 (-2)	2.53	6.45 (-2)	2.83
$.013\bar{3}$	4.01 (-2)	2.34	4.31 (-2)	2.13	3.98 (-2)	2.23	2.80 (-2)	2.30

Table 3: Various types of quadratic reconstruction at boundary cells.

cut cells. In the body-fitted case a grid with 31 x 6 nodes has 186 degrees of freedom. Thus the first entry in Table 2 corresponds to resolution slightly better than the first row of Table 1.

There are several things to note from Tables 1 and 2. The solution using regular quadrilaterals is slightly worse than the corresponding Cartesian solution, even with the flow alignment of the body- fitted grid. In fact, the extra accuracy of using regular Cartesian grids is apparent even in the first order results. The quality of the results for distorted quadrilaterals (where the nodes of the body-fitted grid are randomly perturbed a small amount) deteriorate quite a bit. In the Cartesian results, where one can view the perturbation as restricted to the boundary of the mesh, the boundary cells have similar accuracy, but the overall flowfield results do not suffer.

Table 2 also shows that the errors at the boundary are significantly worse than in the interior. Although the overall accuracy remains second order, often the errors at the boundaries are the most important in terms of measurable quantities such as lift and drag. The following experiments focus strictly on improving the errors at the boundary.

Next we try to improve the results from linear reconstruction by using quadratic reconstruction in four ways, shown in Table 3. First, we use a second order linear reconstruction instead of first order in the cut cells. To do this, we compute a quadratic in the cut cells, but only use the linear part of it to predict the edge states. This makes it more compatible with the interior scheme, which also uses second order gradient information. Secondly, we do quadratic reconstruction in the cut cells, but at edges of regular cells, we *match* the form of the error in the neighboring reconstructed edge state. This means that the cells bordering the cut cells will not suffer the loss of accuracy from the changing stencil, nor lose the local truncation error cancellation property. To use a 1D example, a Taylor series approximation to the edge state is

$$u_{edge} = u_{centroid} + \frac{h}{2}u_x + \frac{h^2}{8}u_{xx} + O(h^3)$$

If u_x is computed to second order, the leading term in the error for the edge state using linear reconstruction is

$$\frac{h^2}{8}u_{xx}. \tag{10}$$

To match the error at edges between cut cells and full cells, the cut cell edge state is approximated using the full quadratic polynomial, and then an error term of the form (10) is explicitly added in (either u_{xx} or u_{yy} depending on the edge orientation. Note however that the coefficients in 10 do not vary with cut cell size). This approach explicitly forces the first full cell to retain locally third order truncation error, since when differencing the edge states the second order errors cancel. This comes however with the penalty of reducing the locally third order accuracy of the cut cells.

Thus, in the third column we use the full quadratic reconstruction at all edges of the cut cells. In the fourth column we use quadratic reconstruction in both the cut cells and their neighbors. In all cases we only use the midpoint rule for a single flux evaluation at the cell edges. We measure only the percent *relative error* along the boundary using (9). The ratio between the errors in successive approximations is in the adjacent column. In all cases the convergence rate of the entire flowfield continues to be second order, as in Table 2.

Although in all cases the magnitude of the error decreases a bit over that of Table 2, the rate of convergence still falls below second order. Thus, for the next test we improve on the results in Table 3 by using 2 point Gaussian quadrature to do the flux evaluations, along with quadratic reconstruction at all edges of the cut cells. This result is summarized in the first two columns of Table 4, showing first the error over the entire flow

h	Quad+2pt.at cut cells				Quad+2pt.for whole FF			
	FF Error	Ratio	Bndry.Er.	Ratio	FF Error	Ratio	Bndry.Er.	Ratio
$.053\bar{3}$	9.45(-2)		1.44(-1)		6.62(-2)		1.19(-1)	
$.026\bar{6}$	2.59(-2)	3.65	4.31(-2)	3.34	1.09(-2)	6.07	2.73(-2)	4.35
$.013\bar{3}$	6.53(-3)	3.96	1.39(-2)	3.1	1.59(-3)	6.86	6.77(-3)	4.03

Table 4: Quadratic reconstruction with 2 pt. Gauss quadrature at cut cell edges (first 2 columns) and over the whole flow field (last 2 columns).

h	FF Error	Ratio	Bndry. Er.	Ratio	FF Error	Ratio	Bndry. Er.	Ratio
$.053\bar{3}$	4.94(-2)		1.24(-1)		6.59(-2)		1.19(-1)	
$.026\bar{6}$	1.34(-2)	3.68	3.84(-2)	3.23	1.08(-2)	6.10	2.73(-2)	4.35
$.013\bar{3}$	3.98(-3)	3.37	1.57(-2)	2.95	1.56(-3)	6.9	6.70(-3)	4.07

Table 5: Flowfield and boundary errors using multiple line segments per cell to represent the boundary. The first half of the Table uses quadratic reconstruction at the boundary; the second half uses it over the entire flowfield along with 2 pt. Gauss quadrature.

field, then the boundary errors. For comparison, we also use quadratic reconstruction with two point Gaussian quadrature over the entire flow field; this is summarized in the last two columns. There is improvement both in the actual level of the error as well as the convergence rate. (We would have hoped however that this rate would be closer to third order).

The final experiment in this series looks at the error coming from the boundary representation itself. A piecewise linear segment in each cell incurs an error of $O(h^3)$ in the area of the cell. A logical next step is to try a higher order representation of the boundary, although for 3D calculations this might be impractical unless working directly with a NURBS representation of the surface. We test the hypothesis that a large portion of the error comes from the boundary representation by using a subcell resolution of the boundary. We use multiple line segments in each cell taken from a grid that is four times finer in each direction. Everything else is as above: quadratic reconstruction with two point Gaussian quadrature at all edges of the cut cells. Other cells use linear reconstruction with a midpoint rule for the flux evaluations. The results are in Table 5. (For comparison, the last columns use quadratic reconstruction for the whole flowfield, along with the subcell boundary resolution).

Again, the actual error magnitude decreases, but it is not showing a second order rate. Hypotheses for this include the mismatch between the boundary scheme and the interior scheme, perhaps the boundary layer of error is a growing function of h, perhaps this mesh spacing is not in the asymptotic regime, or possibly a bug.

5 Discussion

This study is not complete, since we have not yet definitely isolated the major sources of error. However we have reduced the magnitude of the error by a factor of 5. Mesh refinement would have required the mesh spacing to be reduced by more than a factor of 2 in each direction, leading to more than 4 times the work. More investigation is needed to understand the accuracy tradeoffs in finding the most effective method for efficiently computing flows for complex configurations. This will be especially important in three dimension, when mesh refinement alone will not be an efficient way to improve accuracy (since a single cell division leads to 8 times the number of cells).

6 Acknowledgements

We thank Michael Aftosmis for many helpful conversations, and for suggesting the comparison with [2]. M. Berger was supported in part by AFOSR Grant 94-1-0132, and DOE Grant DE-FG02-88ER25053. Some of this work was performed at RIACS, whose support is gratefully acknowledged.

References

[1] Remi Abgrall. On essentially non-oscillatory schemes on unstructured meshes: analysis and implementation. *J. Comp. Phys.*, 114:45–58, September 1994.

[2] M. Aftosmis, D. Gaitonde, and T. Sean Tavares. On the accuracy, stability and monotonicity of various reconstruction algorithms for unstructured meshes. *AIAA-94-0415*, January 1994.

[3] Tim Barth. Recent developments in high order k-exact reconstruction on unstructured meshes. *AIAA-93-0668*, 1993.

[4] Tim Barth and Paul Frederickson. Higher order solution of the Euler equations on unstructured grids using quadratic reconstruction. *AIAA-90-0013*, 1990.

[5] M.J. Berger and R.J. LeVeque. An adaptive Cartesian mesh algorithm for the Euler equations in arbitrary geometries. *AIAA-89-1930*, June 1989. 9th Computational Fluid Dynamics Conf., Buffalo, NY.

[6] M.J. Berger and R.J. LeVeque. Cartesian meshes and adaptive mesh refinement for hyperbolic partial differential equations. In *Proc. Third Intl. Conf. Hyperbolic Problems*, 1990.

[7] D. K. Clarke, M. D. Salas, and H. Hassan. Euler calculations for multielement airfoils using Cartesian grids. *AIAA Journal*, 24, 1986.

[8] William J. Coirier and Kenneth G. Powell. An accuracy assessment of Cartesian-mesh approaches for the Euler equations. *AIAA-93-3335*, July 1993.

[9] Phil Colella and Harland Glaz. Efficient solution algorithms for the Riemann problem for real gases. *J. Comp. Phys.*, 59:264–289, June 1985.

[10] D. DeZeeuw and K. Powell. An adaptively refined Cartesian mesh solver for the Euler equations. *AIAA-91-1542*, June 1991.

[11] B. Epstein, A. Luntz, and A. Nachschon. Cartesian Euler method for arbitrary aircraft configurations. *AIAA Journal*, 30(3):679–687, March 1992.

[12] Foley, van Dam, Feiner, and Hughes. *Computer graphics, principles and practice.* Addison Wesley, 1990.

[13] R. Gaffney, H. Hassan, and M. Salas. Euler calculations for wings using Cartesian grids. *AIAA-87-0356*, January 1987.

[14] A. Jameson, W. Schmidt, and E. Turkel. Numerical solutions of the Euler equations by finite volume methods using Runge-Kutta time-stepping schemes. *AIAA-81-1259.*

[15] S. Karman Jr. Splitflow: a 3d unstructured Cartesian prismatic grid CFD code for complex geometries. Technical report, Lockheed Fort Worth Co., 1994. Preprint.

[16] H. O. Kreiss, T. A. Manteuffel, B. Swartz, B. Wendroff, and Jr. A. B. White. Supraconvergent schemes on irregular grids. *Math. Comp.*, 47, 1986.

[17] J. Melton, M. Berger, Michael Aftosmis, and Michael Wong. 3d applications of a Cartesian grid Euler method. *AIAA-95-0853*, January 1995.

[18] J. Melton, F. Enomoto, and M. Berger. 3D automatic Cartesian grid generation for Euler flows. *AIAA-93-3386*, July 1993.

[19] R. Pember, J. Bell, P. Colella, W. Crutchfield, and M. Welcome. An adaptive Cartesian grid method for unsteady compressible flow in complex geometries. Preprint UCRL-JC-115650, Lawrence Livermore National Laboratory, 1993.

[20] J.J. Quirk. An alternative to unstructured grids for computing gas dynamics flows around arbitrarily complex two dimensional bodies. Technical Report 92-7, ICASE, 1992.

[21] B. Wendroff and A.B. White. A supraconvergent scheme for nonlinear hyperbolic systems. *Comput. Math. Appl.*, 18, 1989.

[22] D. Young, R. Melvin, M. Bieterman, F. Johnson, S. Samant, and J. Bussoletti. A locally refined rectangular grid finite element method: application to computational fluid dynamics and computational physics. *J. Comp. Phys.*, January 1991.

RIACS
Mail Stop T041-5
NASA Ames Research Center
Moffett Field, CA 94035