Local Uniform Mesh Refinement (LUMR) is a powerful technique for solving hyperbolic partial differential equations. However, many problems contain regions where numerical dispersion is very large, such as steep fronts. In these regions, mesh refinement is not very efficient. A better approach in these regions is to locally transform the coordinate system to move with the front. We show how to combine these two approaches in a way which maintains the advantages of LUMR and the effectiveness of moving grids. Experiments with 2-D scalar problems are presented.

Local Uniform Mesh Refinement with Moving Grids

William D. Gropp

Research Report YALEU/DCS/RR-313

April 1984

The work presented in this paper was supported in part by the Office of Naval Research under contract N00014-82-K-0814 and by the National Science Foundation under grant MCS-8106181.

1. Introduction

Hyperbolic partial differential equations typically contain regions of greater activity such as shocks or wave fronts, and these features are dynamic. This behavior, combined with the difficulty of providing a global uniform grid fine enough to resolve the smallest important length scale, has stimulated much work on adaptive grid techniques for hyperbolic problems. There are two major approaches. The first is to use an irregular grid, where the choice of grid points depends on some function of the solution. The grid may be continuous in time. The second is to use locally uniform grids, and to replace these grids frequently. In this case the grids are not continuous in time.

This second method is called Local Uniform Mesh Refinement (LUMR) and is a powerful technique for the solution of partial differential equations. It consists of adaptively refining a uniform coarse grid by overlaying it with uniformly refined subgrids. The uniformity is important in insuring that the overhead of managing the refinement is kept small, and that the refinement does not defeat any vectorizability which may be present in the discretization. This method has been shown to be efficient for a variety of one and two dimensional time dependent problems [1, 4, 7]. We discuss how to combine LUMR and moving grids in a way which substantially improves the performance of each of these algorithms at very little additional cost.

In Section 2, we motivate this extension. In Section 3, we give a detailed description of the algorithm. In Section 4, we show computational experiments of this method, and compare it with uniform grid and regular LUMR calculations.

2. Motivation

In a numerical approximation to a hyperbolic PDE, the truncation error typically has leading form

$$T = C(t)k(h^p + k^p) \tag{2.1}$$

where $C(t)$ is independent of h and k. The usual approach in reducing the local error is to reduce h and k. However, since p is usually small ([8] argues that $p = 4$ is optimal for $u_t = u_x$), this is only moderately efficient. In particular, if the problem has n space dimensions, the work goes as (assuming $k/h = \lambda$ is constant) $h^{-(n+1)}$. Thus, to reduce the global error by a factor of 2 by using mesh refinement will require a decrease in h and k by $2^{1/p}$ and an increase in the work of $2^{(n+1)/p}$. Thus, the small p common in the solution of hyperbolic problems makes mesh refinement relatively inefficient at reducing the error in the computation.

What alternatives are there? It is also possible to reduce the local error by reducing $C(t)$ in (2.1). We may be able to do this by a change of variables to a moving coordinate system, as $C(t)$

depends on the problem being solved. For example, consider the problem

$$u_t + au_x = 0, \tag{2.2}$$

approximated with the leap-frog scheme. The principle truncation error is

$$T = \frac{1}{3}(k^3 u_{ttt} + ah^3 u_{xxx}) \tag{2.3}$$

We can use (2.2) to eliminate u_{ttt} from (2.3) to get

$$T = \frac{1}{3} a u_{xxx}(-a^2 k^3 + h^3) \tag{2.4}$$

so that $C(t) \to 0$ as $a \to 0$. It is easy to pick a coordinate transformation $(x,t) \to (\tilde{x},\tilde{t})$ which takes (2.2) to

$$u_{\tilde{t}} + 0 \cdot u_{\tilde{x}} = 0.$$

Moreover, leap-frog is exact for this problem.

The important thing to note in (2.4) is that reducing a by 2 reduces the truncation error by (roughly) 2. Thus, a significant savings in the effort required to reach a given error tolerance can be realized by finding an approximate transformation.

This is not a new approach; part of the success of the moving finite element method [9, 10] can be traced to this, as can the method of Davis and Flaherty [5]. What is new here is a way to reduce $C(t)$ in some cases at little additional cost in a mesh refinement code. We will not try to drive $C(t)$ to zero (though that is possible in exceptional cases), rather, we will apply uniform velocity transformations to each refined grid in an attempt to reduce $C(t)$ at low cost.

We also argue below that this is an effective way of reducing numerical dispersion in discontinuous or nearly discontinuous solutions. The best way to see this is to look at graphs of the numerical dispersion of a sample scheme on a model problem. We consider the leap-frog method applied to $u_t + au_x = 0$ for $0 \leq a \leq 1$. The model is $u_t + u_x = 0$ where a here is the result of shifting to a moving coordinate system. We let $k/h = \frac{1}{2}$. This gives the dispersion relation

$$\omega h = 2\sin^{-1}\left(\frac{1}{2} a \sin(h\xi)\right)$$

where the basic mode is $\exp(i(\omega t - \xi x))$. However, this is not quite what we want to compare. Because we are using a moving coordinate system, we actually have (in the original, stationary frame)

$$\omega h = 2\sin^{-1}\left(\frac{1}{2} a \sin(h\xi)\right) + (1-a)h\xi$$

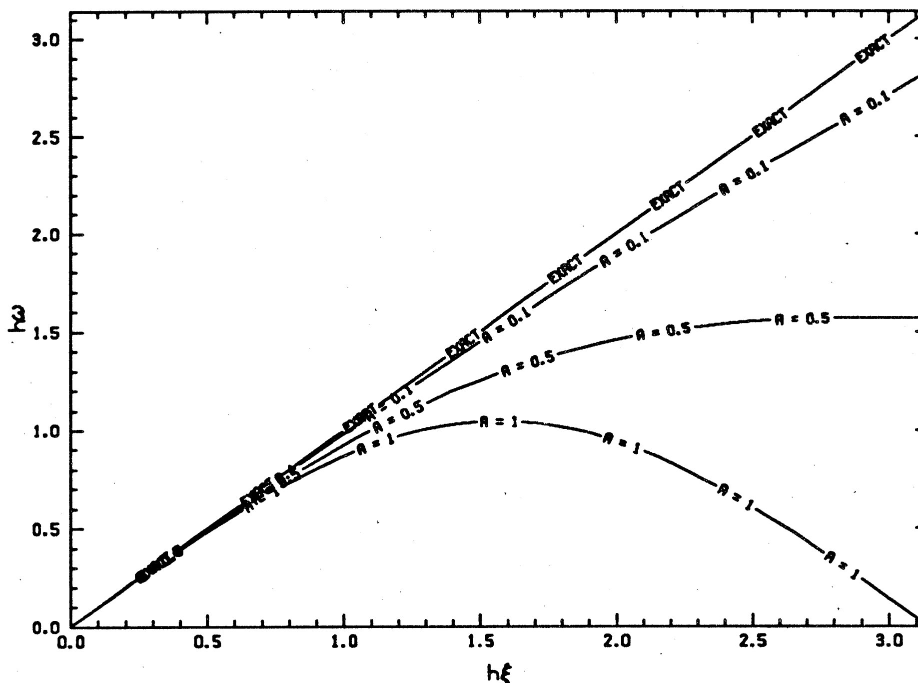


Figure 1: Dispersion relation for model problem for various values of a. Parasitic waves have been omitted.

Graphs of this for various values of a are shown in Figure 1.

The importance of this is that even for $a = \frac{1}{2}$, the numerical dispersion is significantly reduced. Thus, even an approximate moving transformation can be very beneficial. Also note that this reduction in numerical dispersion extends right to the highest frequencies, so this applies to discontinuous solutions as well.

3. Algorithm

In this section we discuss the algorithm for moving LUMR (MLUMR) in two space dimensions. We first discuss the grid structure, then the regridding strategy, and finally the algorithm for advancing the solution one time step. Our algorithm is more in the flavor of [7] than [1] in that we don't allow the grids to be arbitrarily oriented.

The algorithm is fully recursive, so multiple levels of refinement are easily accommodated.

3.1. Grid Structure

The mesh refinement algorithm produces a series of *levels* of grids, G_l. The initial, user specified, level (corresponding to the spatial domain of the problem) is denoted G_0. All levels are made up of rectangular pieces, called grids, denoted G_{li} for the ith grid in level l. Further, each of

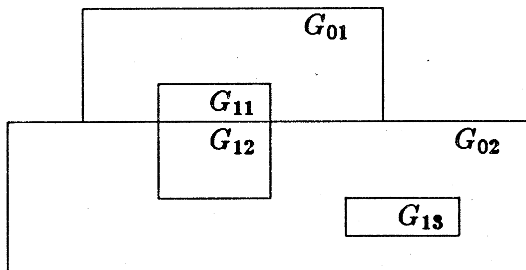


Figure 2: Sample grids. One level of refinement is present.

these pieces is lined up with the usual cartesian directions. Finally, grids must be strictly nested: if a grid G_{li} lies in $G_{l-1,j}$, then all of G_{li} lies in $G_{l-1,j}$ and none of G_{li} lies in any other grid at level $l-1$. The grid $G_{l-1,j}$ is called the *parent* of G_{li}, and G_{li} a *child* of $G_{l-1,j}$. A sample grid structure is shown in Figure 11.

Note that the grids G_{11} and G_{12} are two separate grids rather than a single grid lying partly in G_{01} and partly in G_{02}. This restriction significantly simplifies the data structures involved in managing the grids without imposing a restriction on the kinds of refinement.

Each grid G_{lr} contains a uniform mesh of points (x_i, y_j), $i = 1, \ldots, n$, $j = 1, \ldots, m$, and $x_i = x_0 + (i-1)h_l$, $y_j = y_0 + (j-1)h_l$, where (x_0, y_0) give the coordinates of the lower left corner of G_{lr}. The step sizes in space are h_l and the step sizes in time are k_l. If the boundaries of two grids abut, each contains the mesh points along the common boundary. Thus, points along a common boundary occur in two different places in memory; the algorithm will show how they are maintained with a single value. To simplify the programming, grids may only abut along rows.

It is important to note that the grids of one level overlay the grids of the coarser levels, rather than being patched into the coarser grids. This maintains the uniform nature of the grids, at a small cost in additional grid points. As the same point in the domain may lie in several grids, the algorithm must take steps to insure a single valued solution. Because the grids are regularly oriented, this can be done in an efficient, vectorizable fashion.

Each grid is also assigned a velocity $\vec{v}$. The entire grid will be moved uniformly at this velocity, with $\vec{v}$ changing as often as every time step. Because the grids move, they can run into each other and escape from their parent.

In the first case, there are two possibilities: one grid is overtaking the other, or two grids run into each other. In the case of one grid overtaking another, we allow the grids to overlap. Nothing else need be done, as this will still provide a good local match to the solution velocity. Alternately, we could force the grids to abut, and reset their velocities so the area-average of their respective velocities. In the case where two grids collide, this means that there is no local velocity

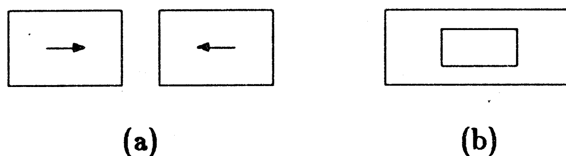


Figure 3: Colliding grids as in (a) are handled by changing to a stationary grid, with additional levels of refinement as necessary, as in (b).

transformation, and the error must be reduced by reducing the step sizes h and k. This can be handled naturally by MLUMR by regridding and generating an additional level of refinement as necessary (cf. Figure 3).

In the second case of grids escaping from their parent into another grid as in Figure 4, we force the grid to stop at the parent boundary, and regrid to create a grid in the adjacent parent. Because the grid can't move (relative to its parent), it may be necessary to refine it as well; again, this is handled in a natural fashion.

Another approach to this problem is to sacrifice strict nesting by parent and go to the looser *level nesting* discussed in [2]. This complicates the data structures. Because of the grid organization, only grids at the first level of refinement are likely to escape from their parent (a grid in the coarsest grid). The coarsest level will normally have very few components—only one in many cases. Grids at higher levels of refinement are likely to move with the same speed as their parent. Thus grids will rarely escape from their parents, and the additional cost of regridding when that happens is compensated for by the simplicity of handling the other grid-parent interactions.

3.2. Regridding

After some number of time steps on a level l, it is necessary to check to see if the refined grids at the finer levels need to be replaced with new refined grids. This is done in three steps. First, points in the grids at level l are *flagged* if it is necessary to refine about that point. This may be done by estimating the truncation error at that point or by some *ad-hoc* method based on user knowledge of the behavior of the solution. Second, these flagged points are surrounded by a buffer of marked points. This buffer is needed to isolate the interior of the refined grid from the coarse grid, which is necessary given our strategy of overlaying the refined grids. Further, this buffer allows us to take more steps between regriddings by giving a margin for the region of higher error to move around in. Finally, the marked and flagged points are scanned to determine the new

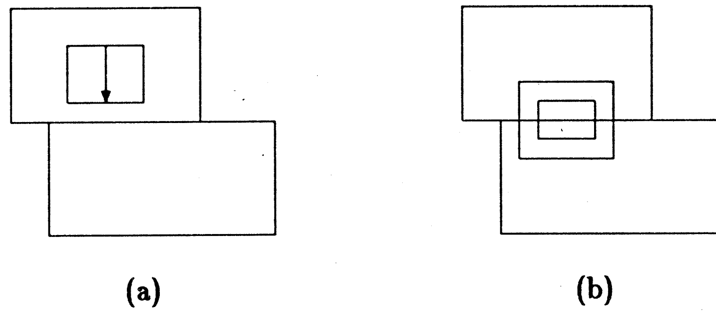


Figure 4: Grids leaving their parent as in (a) are handled by using stationary grids in the original parent and the new parent. An additional level of refinement, as in (b), may be necessary.

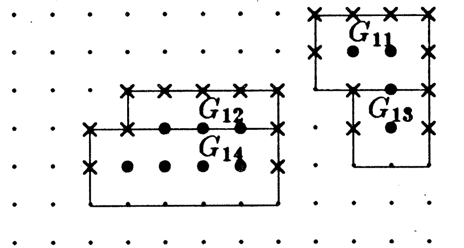


Figure 5: Regridding process. · denote mesh points, • denote flagged points, × denote buffer points. The resulting refined grids, G_{11} through G_{14} are outlined.

refined grids, and those grids are initialized with values from the previous refined grids (where they overlapped) and the parent grid.

The process of adding the buffers can be done by bit-wise logical operations of left and right shifts and or's. This helps keep the cost of the regridding down. The regridding process in shown in Figure 5.

3.3. Finding grid velocities

This is either the hardest or the easiest part, depending on what you plan to do. In the moving finite element (MFE) approach, the nodes move at a velocity which is automatically calculated to minimize a penalty function made up of the residual and some terms to keep the mesh well behaved. In our approach, we ask the user to specify the velocity of a grid, given the values on the grid and its location. For many problems this is quite reasonable; detecting near-discontinuities and computing an appropriate approximate velocity is often quite simple. In fluids problems, the appropriate velocity is often the fluid speed, and is immediately available. It should be emphasized

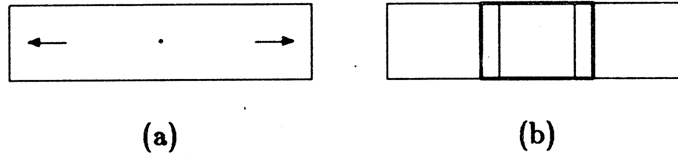


Figure 6: Grids over which the velocity varies greatly as in (a) are broken into several grids. In (b), the grid has been broken into three overlapping parts, with the center part (in thicker lines) remaining stationary.

that because we are not trying to get an exact value for the velocity, we can be much more cavalier about choosing the velocity. For a completely automatic method, one similar to that in [9] could be used.

3.4. Integrating a level

Each level consists of a number of possibly intersecting grids. Here we describe how a single level is integrated; in the next section, we put everything together.

The boundaries between the a refined grid and its parent are determined by using linear interpolation from the parent grid values. This has been shown to be stable [3] for stationary grids.

If the grids abut, then the boundary between them is handled in a special way. The points along the common boundary are duplicated in each grid. This is done to maintain the uniform structure of the grids. When integrating the top grid, the common boundary is also integrated, using data from both grids. Then the values computed on the common boundary are copied from the upper grid into the lower grid.

Overlapping grids require special care. In the case where the refinement is 1 (no refinement), overlapping grids can be handled efficiently by just injecting the values on the "refined" grids onto the parent grid, and then using the usual boundary procedure for fine/coarse grid interfaces to communicate the values back to the refined grids.

If the refinement is greater than 1, then the boundary values must be computed by interpolating from the overlapping "partner".

As an example, we will describe how one time step is taken on a coarse grid, where there is only one level of refinement and no regridding takes place during this time step. First, the coarse grid is integrated from time t to $t + k_0$. Next, the refined grids are integrated from time t to $t + k_1$,

```
procedure step( lev )
```

1. Integrate level lev
2. Check for time to regrid the *next* level. If so regrid the next level
3. Update grid velocities if enough steps have passed. Must check this *after* the regridding check, since we don't want to reset velocities and immediately regrid.
4. Integrate the finer levels, if any

   ```
   do i=1, refinement(lev)
         call step( lev + 1 )
   ```
5. Having integrated the finer level, we can update this level from the next finer level

Algorithm 1: Moving Mesh Refinement algorithm for advancing the solution one time step.

then from $t+k_1$ to $t+2k_1$, and so on until they reach $t+k_0$. The boundary values at the fine/coarse grid interfaces are taken from interpolation of the coarse grid values at times t and $t+k_0$. Finally, the values on the fine grid at time $t+k_0$ are injected into the coarse grid to update the solution there.

3.5. Integration

Now that we have the structure of the grids and the regridding algorithm, we can describe the algorithm for integrating a level one step in time. This algorithm is recursive.

The algorithm described in Algorithm 1 is fairly straightforward. Step 5 is the only one we have not talked about. This step insures that the solution on all the grids is single-valued, and, more importantly, that those areas of the coarse grid which are refined take their solution from the refined grids. If this were not done at every step, there would be a danger that the inaccurate solution on the coarse grid in this area could escape and contaminate the solution everywhere. This contamination is prevented by a combination of finite domain of dependence of the differential equation and the difference approximation, the buffers surrounding the areas of large error, and the injection of the better solution computed on the fine grid into the coarser grids.

A more subtle point is the order of the steps in the algorithm. When the routine to regrid is called, the parent level has been integrated one step beyond the step where the regridding is to take place. This lets us place grids where they will be needed, rather than where they had been needed. Also note that when the regridding routine is called, all grids at *finer* levels no longer have a legitimate parent path (since they now point to non-existent parents). This is acceptable, since

those grids will also be regrided before they will be used to update any parent grid, and they will not be integrated (their only use is to initialize the new grids)

Because handling overlapping grids is complicated and inefficient, particularly if every grid had a different velocity, we may want to force adjacent grids which have nearly the same velocities to have the same (area averaged) velocity. This can significantly improve the performance of the algorithm. We used this approach in our program.

4. Experiments

These experiments are all in two space dimensions and for scalar problems. Each contain discontinuities in either the solution or its derivatives. Each was run on a FPS-164 running SJE and using the E Release of the FORTRAN compiler at optimization level 3. Each problem is computed using a uniform coarse and a uniform fine grid, and both LUMR and MLUMR. In each of the mesh refinement calculations, the level zero grid (the coarsest grid) was the same as the coarse-only calculation. The width of the finest mesh in each calculation is denoted by h_f. No more than 1 level of refinement was used. Errors in the solutions were measured on the coarse grid. Three different measures were used. The ℓ_2-error is defined as $(n^{-1}\sum_{i=1}^{n} e_i^2)^{1/2}$, the ℓ_1-error is defined as $n^{-1}\sum_{i=1}^{n}|e_i|$, and the ℓ_∞-error is defined as $\max|e_i|$. The ℓ_2-error is the appropriate measure for smooth problems. The ℓ_1-error is more appropriate for non-smooth problems, and should be the primary basis for comparison in these tests. Since in these problems the largest error occurs at discontinuities, the the ℓ_2-error, which preferentially measures the larger errors, will be strongly affected by very small changes in the position of the discontinuities. The ℓ_1-error, on the other hand, weights all errors equally and is a better measure of the numerical dispersion away from the discontinuities. The ℓ_∞-error is easily contaminated by very small errors in positioning discontinuities and is useful only for continuous solutions.

As a first sample problem we take a variant of the revolving cone problem used in [6] and [2]. The problem is:

$$u_t - yu_x + xu_y = 0,$$

with the initial condition

$$u(x,y,0) = \begin{cases} 1 - 16r, & \text{if } r < \frac{1}{16}, \\ 0, & \text{otherwise,} \end{cases}$$

where

method	h_f	ℓ_2 error	ℓ_1 error	ℓ_∞ error	Time (seconds) STEP	USINTG	BNDY
coarse	1/20	6.65E-2	2.19E-2	5.98E-1	3.73	3.32	0.33
fine	1/80	1.38E-2	3.74E-3	1.22E-1	209.	204.	4.0
MR	1/80	1.40E-2	4.20E-3	1.22E-1	46.3	35.0	7.55
Moving MR	1/80	5.65E-3	1.39E-3	7.30E-2	35.4	25.8	6.20

Table 1: Results for first test problem. Errors are computed on the coarse grid. Times are for entire program (STEP), user integrator (USINTG), and boundary handling (BNDY).

$$r = \left(\left(x - \frac{1}{2}\right)^2 + \frac{3}{2}y^2\right).$$

The boundary conditions at the inflow boundaries are zero. The solution to this variable coefficient problem is given by rigid counter-clockwise rotation of the initial data about the origin with angular frequency $\omega = 1$. This is a good test of both regular and moving mesh refinement because finite difference approximations to this problem generate large, localized errors containing high frequency terms (due to the base and tip of the cone). For moving mesh refinement, there is the additional feature that the appropriate velocity varies substantially over the cone, so that a close match of the velocity of the solution and the fine grids is not possible.

In our test, we use Lax-Wendroff as the difference approximation, with inflow boundaries specified as $u = 0$ and outflow boundaries specified with first order extrapolation. A substantial effort was made to identify hotspots in the code, and those parts of the code were optimized. This was done to insure that these tests were realistic comparisons. For example, the execution time of the Lax-Wendroff scheme was improved by about 40%. The code is written to be efficient in all modes of operation (no refinement, unmoving refinement, or moving refinement), and is fully instrumented.

We can see from Table 1 that the MLUMR solution achieved significantly better results than LUMR at about 3/4 the cost. Both LUMR and MLUMR were much faster than a uniform grid. The graphs of the solutions in Figures 7–8 show that the error for both the uniform grid and the LUMR calculation is due primarily to numerical dispersion (the high frequency waves lag the solution, as is predicted by Figure 1), while the error in the MLUMR solution is due to a combination of dispersive error (caused by the variation of velocity over the cone) and dissipation in the scheme. We can see from Figure 2 that part of the efficiency advantage of MLUMR comes from needing less refinement (a consequence of the smaller dispersive error).

method	h_f	ℓ_1 error	ℓ_2 error	Time (seconds) STEP	USINTG	BNDY
coarse	1/40	1.84E-2	4.78E-2	.94	.88	.04
fine	1/80	1.18E-2	4.56E-2	7.22	7.05	.14
MR	1/80	1.17E-2	4.47E-2	5.00	3.61	.66
Moving MR	1/40	1.07E-2	3.99E-2	2.45	1.27	.49
Moving MR	1/80	6.88E-3	3.29E-2	4.76	2.97	.97

Table 2: Results for second test problem. Errors are computed on the coarse grid. Times are for entire program (STEP), user integrator (USINTG), and boundary handling (BNDY).

The second test problem is

$$u_t + uu_x + uu_y = 0$$

on the unit square, with $0 \leq t \leq 0.6$. Note that under the rotation $x' = (x+y)/\sqrt{2}$, $y' = (-x+y)/\sqrt{2}$, the equation becomes $u_t + \sqrt{2}uu_{x'} = 0$; the solutions of this equation are well known. This is a very difficult problem for any mesh refinement scheme because so much of the domain needs to be refined.

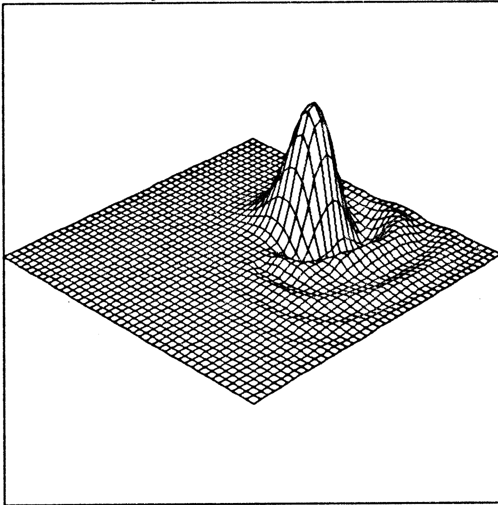
The initial data for our second test problem is

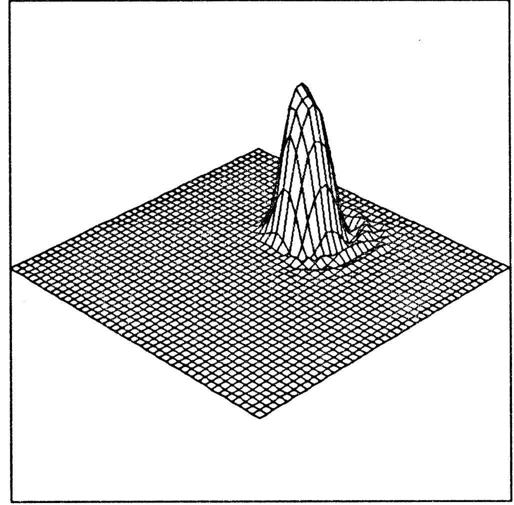
$$u(x,y,t=0) = \begin{cases} \frac{1}{2}, & \text{if } 0 \leq x \leq \frac{1}{2} \text{ and } 0 \leq y \leq \frac{1}{2}, \\ -\frac{1}{2}, & \text{if } \frac{1}{2} < x \leq 1 \text{ and } \frac{1}{2} < y \leq 1, \\ \frac{1}{4}, & \text{otherwise.} \end{cases}$$

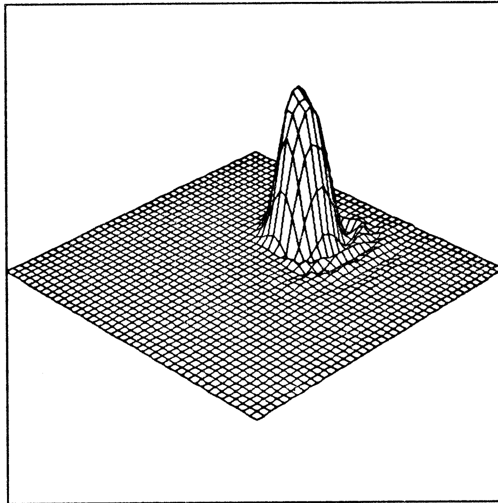
The solution of this problem consists of six shocks, two moving with speed 3/8, two with speed 1/8, and two stationary. At two points (the problem is symmetric about $x = y$), three shocks with different speeds come together. Further, since this is a nonlinear problem, even a perfect choice for the speed of the local grids will not completely eliminate the error. This is because characteristics converge on the shock, and thus there is no unique local velocity.

For this problem, the grid speeds are chosen nearly exactly; the speeds were set by the location and extent of the grid. If the grid covered several shocks, it was broken up into several overlapping pieces. Because the grid speeds were chosen nearly exactly, the errors in the computation come from only two places: the errors in fitting the moving grids to regions where multiple shocks interact, and the error in integrating a shock even in a co-moving reference frame. Thus, this example is complementary to the first example, where the error is due to range of velocities over the refined grid.

We can see from Table 2 that even without refinement, MLUMR gives a much better solution at a fraction of the cost of either LUMR or uniform grids. In this example, the ℓ_1-norm is the most representative, as an error of one mesh point at a shock can make the ℓ_∞-norm $O(1)$ and have significant impact on the ℓ_2-norm. From Figures 9–10 we see that the majority of the error in all computations is at the shocks, but that the "ringing" trailing the shocks is nearly absent in the MLUMR calculation. The highest error in the MLUMR calculation comes at the intersections of the shocks, where the MLUMR method can't take advantage of any local frame of reference.







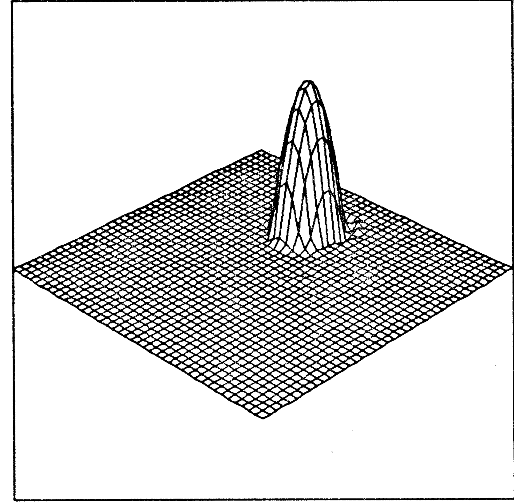
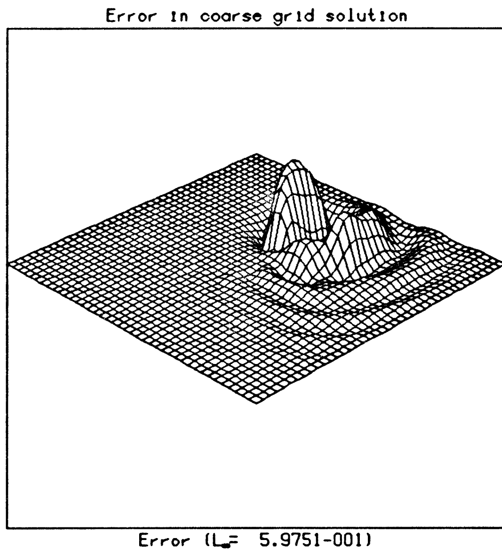
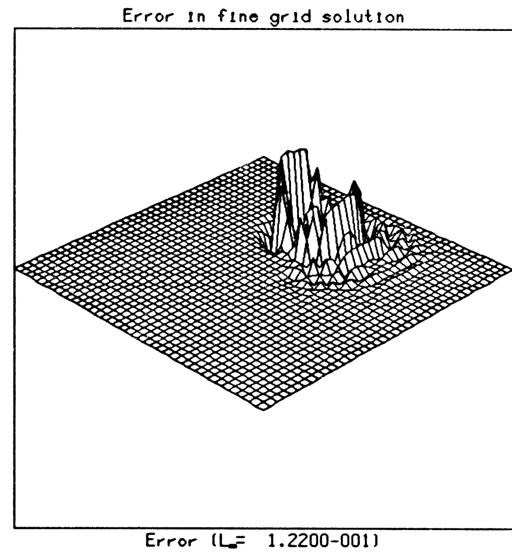
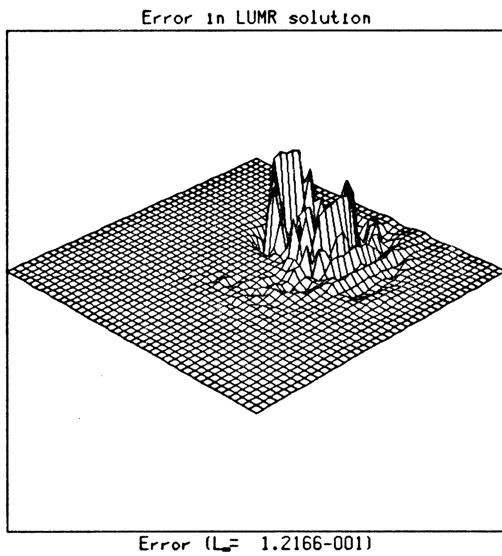


Figure 7: Solutions for revolving cone problem.







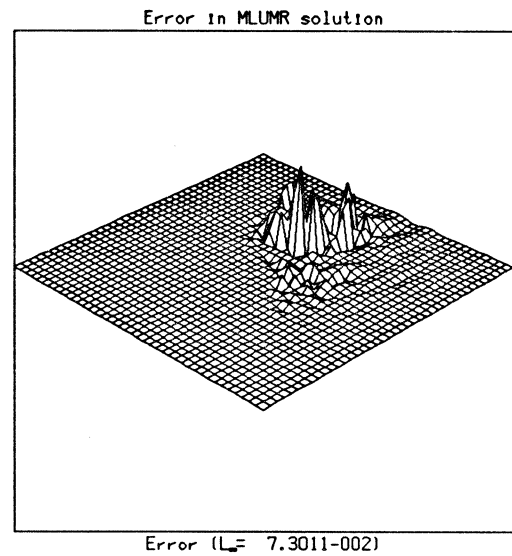
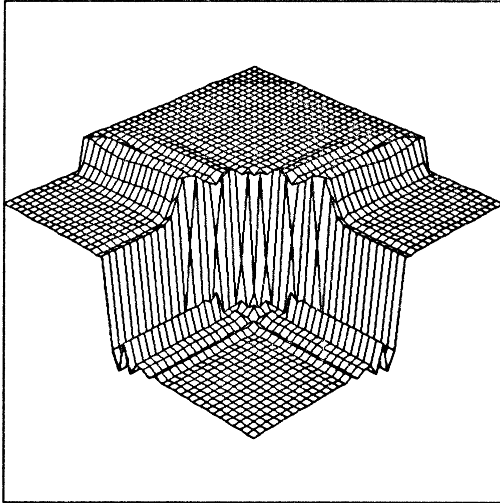
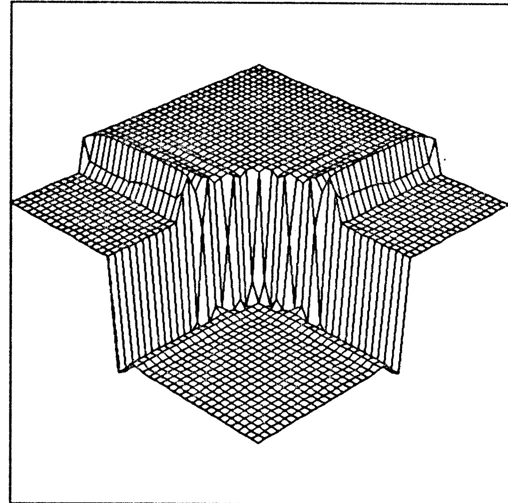
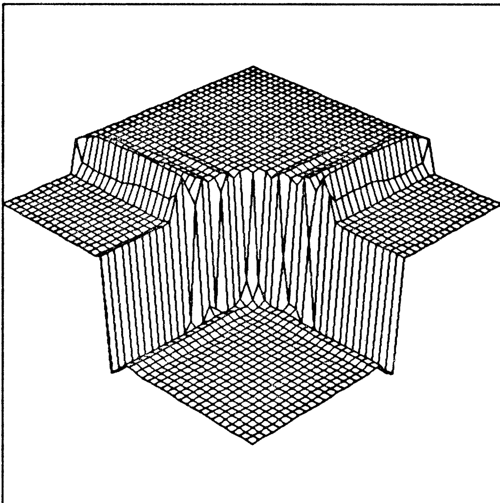


Figure 8: Errors in the solutions in Figure 7.







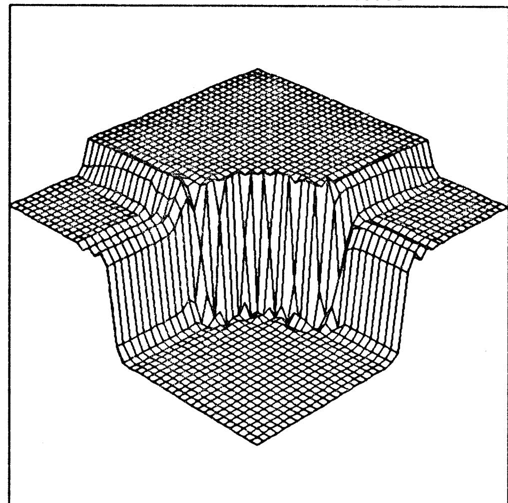
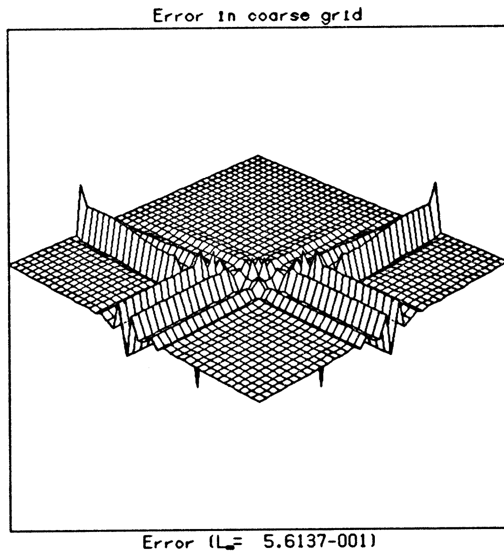
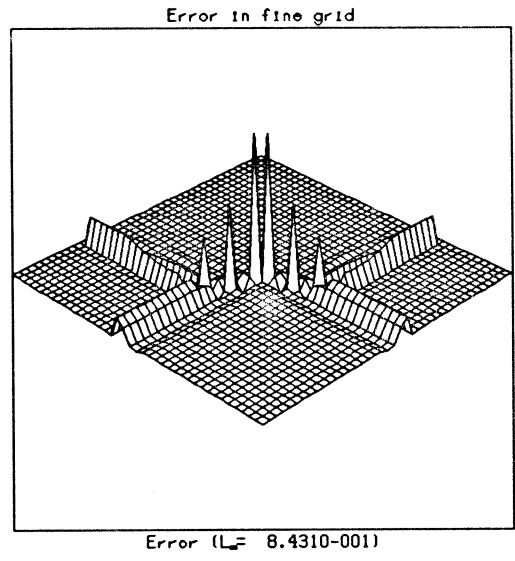
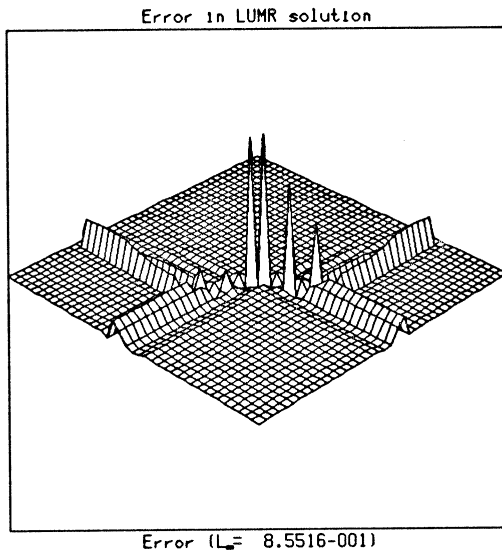


Figure 9: Solution for the second test problem.







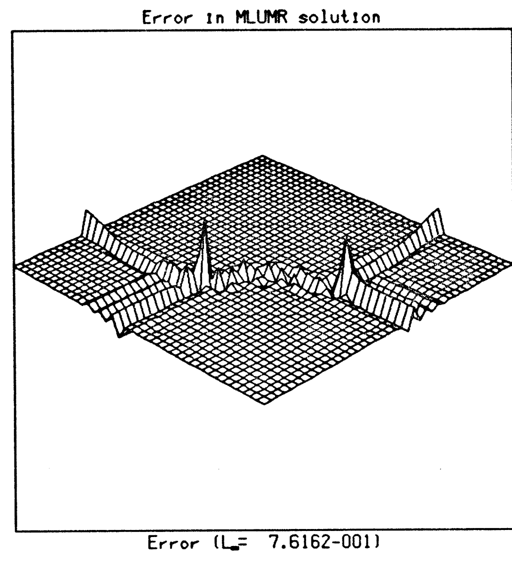
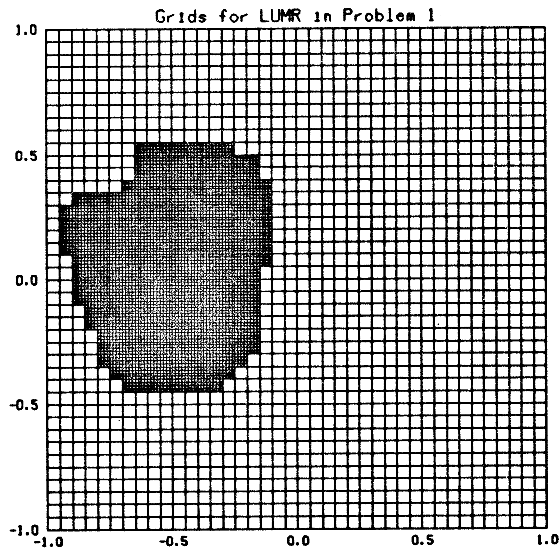
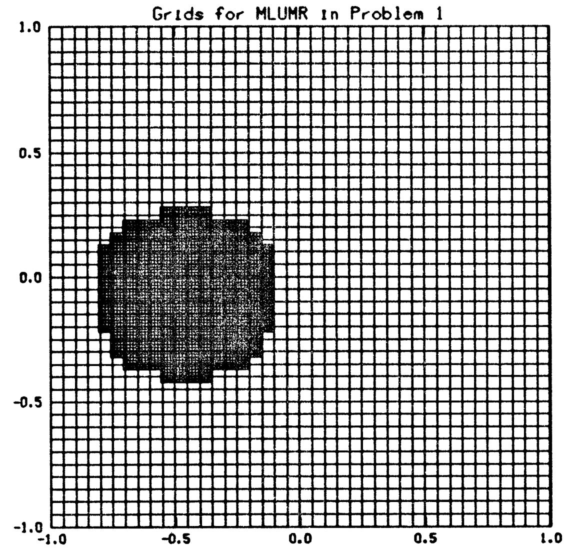
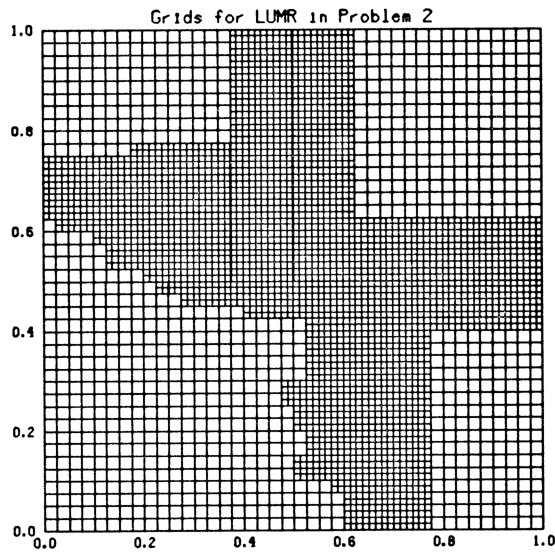


Figure 10: Errors in the solutions in Figure 9.







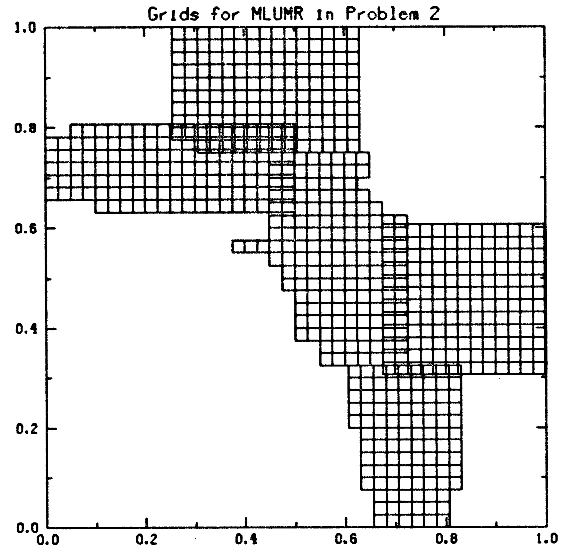


Figure 11: Grids for both test problems at the final time ($T = 3.375$ for problem 1, $T = 0.6$ for problem 2).

References

[1] Marsha J. Berger, *Adaptive mesh refinement for hyperbolic Partial Differential Equations,* Ph.D. Thesis, Stanford University, 1982.

[2] Marsha J. Berger and Joseph Oliger, *Adaptive mesh refinement for hyperbolic partial differential equations,* Technical Report Manuscript NA-83-02, Stanford University, March 1983.

[3] Marsha J. Berger, *Stability of interfaces with mesh refinement,* Technical Report 83-42, ICASE, August 1983.

[4] John H. Bolstad, *An adaptive finite difference method for hyperbolic systems in one space dimension,* Ph.D. Thesis, Stanford University, 1982.

[5] Stephen F. Davis and Joseph E. Flaherty, *An adaptive finite element method for initial-boundary value problems for partial differential equations,* SIAM Journal on Scientific and Statistical Computing, 3/1 March (1982), pp. 6–27.

[6] David Gottlieb and Steven A. Orszag, *Numerical Analysis of Spectral Methods: Theory and Applications,* SIAM, 1977.

[7] William D. Gropp, *A test of mesh refinement for 2-d scalar hyperbolic problems,* SIAM Journal on Scientific and Statistical Computing, 1/2 June (1980), pp. 191–197.

[8] H.-O. Kreiss and Joseph Oliger, *Comparison of accurate methods for the integration of hyperbolic problems,* Tellus, 24 (1972), pp. 199–215.

[9] Keith Miller and R. Miller, *Moving Finite Elements I,* SIAM Journal on Numerical Analysis, 18 (1981), pp. 1019–1032.

[10] Keith Miller, *Moving Finite Elements II,* SIAM Journal on Numerical Analysis, 18 (1981), pp. 1033–1057.