Eighth Eurographics Workshop on Virtual Environments (2002)
S. Müller, W. Stürzlinger (Editors)

Interactive Visualization and Steering of CFD Simulations

O. Kreylos[†], A. M. Tesdall[†], B. Hamann[†], J. K. Hunter[‡] and K. I. Joy[†]

[†]Center for Image Processing and Integrated Computing (CIPIC), University of California, Davis, CA
[‡]Department of Mathematics, University of California, Davis, CA

Abstract

We describe a system that supports real-time interactive visualization of computational fluid dynamics (CFD) simulations. The system allows a user to place and manipulate visualization primitives, such as isolines and streamlines, during an ongoing simulation process. A user can interactively select and designate regions of the computational mesh for refinement as the simulation progresses, perform remeshing, and see the effects of the refinement on the simulation in real time. The system is being used for the study of two open problems in compressible fluid dynamics. We can interactively explore solutions as they are computed, identify flow field regions containing features of interest, and refine the grid in those regions in order to obtain a better result locally. The ability to visualize "live" data, and to make changes to the computational setup in real time, has helped us to understand the underlying fundamental CFD simulation issues of these problems in shorter times than would otherwise have been possible.

1. Introduction

In computational fluid dynamics (CFD) analyses, the tasks of simulation and visualization are usually done as independent steps – visualization following numerical simulation. In a typical procedure, the researcher first performs a calculation for some fixed amount of computational time, and then views the solution in a separate post-processing step, using a stand-alone visualization system. Based on the solution at this intermediate stage, a decision may be made to change simulation parameters, regrid, modify the modeled geometry, or simply continue the computation unchanged from that point. This process is repeated until the solution of a steady problem has converged, or until the solution of an unsteady problem is evolved to a given point in time.

We believe that it is more efficient, and more enlightening, to perform both steps – simulation and visualization – at the same time. A special-purpose visualization system that can be directly coupled with a running simulation, e. g, via a network connection, and that allows us to visualize "live" data as it is produced by the simulation, can be used to observe the simulation progressing towards its final result, instead of just visualizing that result. Such a system is not only a valuable tool for debugging a simulation under development, but it can also help in understanding the phenomenon being simulated, by showing how the simulation arrived at the final result. Even when a simulation code is already well-established, a specific simulation can still go awry, for example if inappropriate simulation parameters are specified. Instead of having to wait for a run to finish, and then finding out that the result is not as expected, it is often possible to detect that a simulation will yield suboptimal results early on and interrupt it, or sometimes even to improve its solution by changing parameters on-the-fly. In situations like this, a coupled simulation–visualization system can reduce the overall computation time significantly.

Our basic assumption is, that the simulation – be it steady or unsteady – computes transient solutions which converge towards the final solution over time, and that the time to compute one of those solutions is much shorter than the overall simulation time. In the two computational problems described in Section 4, computing a single solution takes on the order of a second, whereas a complete simulation can take days. Under this assumption, it is even possible to view real-time animations of simulation data while the simulation is still running. This does not mean, however, that a user has to observe a simulation over its whole running-time. Typically, it is sufficient to observe the transient solutions only until it is determined that the simulation parameters are set appropriately, and then come back occasionally to check if the simulation is still progressing as planned.

Having such a coupled system also allows cooperation in the other direction: It makes it possible to "steer" a simulation by changing parameters on-the-fly using the visual interface provided by the visualization system. How to steer a simulation, i. e., which parameters to change, depends on the specific problem; one example would be to change grid structures during computation to improve resolution in areas of interest.

An alternative approach to steering is based on using an algorithm that modifies grids or geometry automatically, as the solution progresses. Adaptive grid methods, for example, cluster grid points in regions where they are most needed, such as near shocks or in other areas where large gradients occur. The adaptive mesh refinement (AMR) approach, described in [1, 2], for example, uses a sequence of nested Cartesian grids that are refined in both space and time. Fine grids are recursively embedded in coarser grids until the necessary resolution is obtained. During a computation, refined grids are created or removed automatically, in response to estimated error, without user intervention. This approach has been used to compute two and three-dimensional solutions containing shocks (see, for example, [2]). Adaptive grid methods have also been developed for overset and unstructured grids, and are discussed in [10, 11, 14, 12].

Similarly, automatic methods can be used to dynamically modify geometry in response to a computed solution. For example, automatic CFD-based design methods have been developed for the optimization of aerodynamic shapes, including airfoils, wings, wing-bodies, and complete three-dimensional aircraft configurations. The approach in [8, 9, 16] uses a control theory-based adjoint formulation, and systematically modifies a configuration through manipulation of design variables that smoothly vary the shape of the aerodynamic surfaces. Examples of optimal three-dimensional designs based on this approach include complete business jet [15] and supersonic transport configurations. Reuther *et al.* [17] describe details and provide a concise summary of earlier aerodynamic shape optimization methods.

Even when using automatic or adaptive methods, interactive visualization of "live" simulation data is still beneficial; additionally, for some CFD problems, the complexity and expense of an automatic grid refinement scheme may not be necessary or desirable, but one may still need to locally refine the grid in order to resolve interesting features in the solution. For these problems, an interactive system that supports visualization of the solution as it is being computed is extremely useful. Such a system must allow a user to specify the region(s) where refinement is needed. In this paper, we apply the described system to the investigation of two open problems in compressible fluid mechanics that are amenable to interactive local grid refinement. The solutions contain intersecting shock waves, and we are interested in obtaining highly refined solutions in the neighborhood of the intersections. We interactively explore solutions as they develop, select regions for grid refinement, and continue the computations after remeshing.

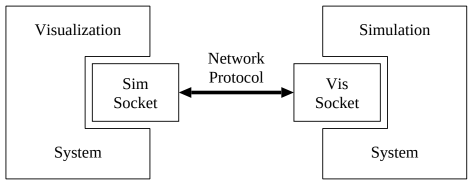

Figure 1: *Layout of integrated simulation–visualization system.*

2. Related Work

Computational steering has been an active area of research for over a decade; an excellent survey/taxonomy of existing systems can be found in [21]. The framework presented here differs from existing steering systems in that it is based on an interactive data exploration system, and as such focuses on interactivity in both visualization and steering. It is also aimed specifically towards time-varying simulations of two- or three-dimensional phenomena.

3. Overview of the Steering Framework

A steering framework is characterized by the fact that it closely couples simulation and visualization/interaction. To use a visualization program for steering, it must "understand," and be able to manipulate, the internal data representation used by the simulation. As simulations tend to use problem-specific data representations, the visualization component of a steering framework must be flexible enough to deal with those, without having to write a problem-specific visualization program. The impacts of this requirement are discussed in detail in Section 5.

From a system-architecture point of view, there are two ways of building a steering framework: One can build a monolithic program consisting of both parts, or one can construct two independent programs linked by a communication protocol. To support separate development of the two program components, and to provide maximum flexibility, we chose the latter approach. Building a distributed application has other benefits: It is possible to run the simulation and visualization programs on different machines connected via the Internet, and it is possible to connect to/disconnect from the ongoing simulation. In theory, even multiple visualization programs could be connected to the same simulation, allowing collaboration between users at multiple sites. The distributed system structure is shown in Fig. 1.

From a computational scientist's point of view, it is relatively simple to modify/extend a simulation program to support visualization and steering. The complexity of maintaining a remote connection to a visualization program and sending simulation data across that connection can be encapsulated in a single C++ class (`VisSocket`, see Fig. 1). We have found that adjustment and extension of a simulation code written in C/C++ can be done in a few hours. The visualization program has to accomodate the simulation program; the data exchanged between simulation and visualization programs are always in the format used by the simulation. In developing the steering framework, it was a goal to minimize the number of data conversions, and to perform all of them in the visualization program. This approach makes the visualization program more complex: Even when only considering grid-based simulations, a multitude of possible data representations exists, ranging from simple Cartesian to hierarchical AMR grids. Using the system architecture described in Section 5, it is possible to modularize the visualization program to accomodate all these representations.

The guiding principle behind developing the described visualization system was interactivity. Our previous research [23] shows that being able to interact with a visualization, as opposed to looking at a fixed image, aids in understanding the visualized phenomenon. Furthermore, since our system is coupled with a running simulation, changes in simulation data must be reflected in the visualization whenever they occur. In an interactive visualization system, a user can place visualization primitives (contour lines, streamlines etc.) anywhere inside the data domain. When a primitive is placed, it will follow all movements of the pointing device until it is released in its final position. The resulting animation of primitives provides an intuitive understanding of the visualized data's structure. When the visualization system is connected to a running simulation, the visualized data can be updated anytime – either on user request, or automatically whenever the simulation has completed a new transient solution. On a data update, all previously placed visualization primitives will reflect the change, resulting in a real-time animation of simulation data. A primitive can also be dragged while data is updated automatically; if the simulation can calculate transient solutions fast enough, a user can explore the data in both space and time. The visualization system's internal architecture and the provided visualization primitives are described in more detail in Section 5.

4. The Computational Problem

With the integrated simulation–visualization system, we have studied two problems in compressible fluid mechanics that involve flow at or near the speed of sound (transonic flow). These problems involve the reflection and focusing of weak shock waves, and are poorly understood theoretically due to the difficulty in analyzing multi-dimensional shock wave propagation. In particular, the weak shock reflection problem is a physically important example of a two-dimensional Riemann problem, which arise in the study of hyperbolic systems of conservation laws, including the Euler equations of gasdynamics. Other examples of transonic flow problems occur in aerodynamic applications, for example, in the flow around wings of commercial subsonic transport aircraft at cruise Mach number. Transonic flow is characterized by the occurrence of shock waves that are difficult to resolve numerically due to their weakness, and by the fact that the flow fields contain regions of both supersonic and subsonic flow. In the following, we briefly summarize the computational problems; see the references for detailed explanations.

In the first problem, we study the transition between Mach and regular reflection for weak shock waves reflecting off thin wedges (see Figs. 7 and 8 for examples of Mach and regular reflection, respectively). An analysis of the Euler equations of gas dynamics shows that regular reflection of a plane shock is impossible when the wedge angle $\theta < \theta_d$, where θ_d is the detachment angle, which is a function of the shock strength. One possible condition for transition from regular to Mach reflection, therefore, is that transition occurs at the detachment point, where $\theta = \theta_d$. A second plausible condition for transition is the sonic point, at which the point where reflection occurs is exactly sonic with respect to the flow behind the reflected shock. These two criteria are so close together that they cannot be distinguished from each other in experiments. Numerical studies to date have also been unsuccessful in resolving this problem, and the correct criterion for transition is unknown. See [3] for a detailed discussion of transition in shock reflection, and [4, 5] for further explanation.

An asymptotic problem that describes the reflection of weak shocks off thin wedges was formulated in [6], and is further discussed in [7, 13, 18]. The detachment and sonic points correspond to different values of a parameter *a* (see [7]) in the asymptotic problem that are very close in numerical value. We solved the asymptotic equations using the numerical method described in [19, 20]. We use a curvilinear grid that has a locally refined area of uniform grid very close to the reflection point, and is stretched exponentially away from the reflection point toward the outer numerical boundaries and the wall. This grid is shown schematically in Fig. 4(a). As we describe in Section 5, we refine the grid interactively during an ongoing simulation. Following each grid change, the solution is interpolated onto a new grid, and the simulation continues. In our numerical solutions, we use values of the parameter *a* close to the sonic and detachment values, and search for evidence of transition in the numerical solutions.

The second problem concerns the focusing of curved shock fronts. When a weak non-planar shock propagates down a shock tube, one of two configurations develops. If the shock is sufficiently weak it crosses over itself and forms a linear, fish-tail configuration, as shown in Fig. 9(a). For slightly stronger shocks, the non-linear configuration in Fig. 9(b) is observed; here, the acceleration of the central part of the shock prevents the shock from crossing over itself and forming a fish-tail configuration. In both configurations, an apparent triple point, consisting of three intersecting shocks, is observed both experimentally and in numerical solutions. It can be shown, however, that for sufficiently weak shocks, such intersections cannot exist. Apparent triple shock intersections also occur experimentally and in numerical solutions of the Mach reflection of weak shocks, which was investigated in [7, 20]. In [7, 20], numerical evidence was presented of a supersonic region near the triple point that contains additional expansion waves, and the resulting wave structure at the triple point is admissible by theory. An expansion fan has not been detected at the triple point in weak shock focusing, but the existence of solutions similar to those in [7, 20] would resolve the conflict between theory and experiment for the focusing problem. See [18] for further explanation of shock wave focusing.

To study the shock focusing problem we have applied the numerical scheme described in [7]. We use a Cartesian grid that has a locally refined area of uniform grid near the triple point, and is stretched exponentially toward the outer numerical boundaries and the wall, as shown schematically in Fig. 4(b)

5. Structure of Visualization System

Our goal was to create a visualization system that is close enough to the simulation system to appear to be a part of it, yet general enough to be able to be coupled with several different simulation systems without having to rewrite it every time. The major advantages of such a tight-knit system are twofold: First, such a system supports interactive visualization of "live" simulation data. Second, it allows us to steer the simulation by manipulating parameters, thus improving the quality of simulation results.

5.1. Interactive Visualization of "Live" Simulation Data

Our visualization system closely follows the system architecture of the one described in [23]. It can visualize grid structure, and data can be explored interactively by placing and moving visualization primitives in the simulation's domain. Since the example simulations described in Section 4 provide multi-valued data, with at least one scalar and one vector (velocity) for each grid vertex, the visualization primitives supported by our system are contour lines of any scalar value, and streamlines of the vector value. Both primitives can be colored by mapping any scalar value.

A typical use of our system is to first explore a single transient solution computed by the simulation by placing and dragging some visualization primitives; once primitives highlighting important features of the data have been placed, automatic data update is enabled to see how the placed primitives evolve over time. If new features develop in the data, or if existing features are no longer well-represented, automatic update is disabled, and the cycle is started over by adjusting the existing primitives or adding new ones. A benefit of the distributed system architecture described in Section 3 is that the visualization system can be connected to/disconnected from the simulation at will; therefore, it is not necessary for a user to continually supervise a long-running simulation. Typically, one only observes the first few computed solutions to check for validity, and then comes back once in a while to check if the simulation is still progressing as intended or to change parameters on-the-fly.

5.1.1. Visualization Primitives

Rendering contour lines (isolines) is probably the most commonly used visualization technique for scalar data. A contour line connects all points in a domain having a given function value c, and, for grid-based data, a contour line is typically computed using a version of the Marching-Cubes algorithm [24, 25, 26]. This algorithm is not suitable for interactive visualization. Therefore, we replace it with a "seeded isoline" method. We do not create an isoline by specifying a constant function value c, but by selecting a point inside the domain. The visualization system calculates the scalar value at the selected point, and we compute an isoline for that value by "growing" it outward from the selected point: First, a contour line fragment for the grid cell containing the selected point is created using the Marching-Cubes method; second, it is determined which edges of the cell are intersected by the contour line; and, third, these cells are visited in turn, creating the complete contour line, see Fig. 2. The main benefit of this method is that it can be interrupted at any time, enabling interactive response independent of data set size and allowing a user to "drag" a contour line to quickly and intuitively explore a scalar field.

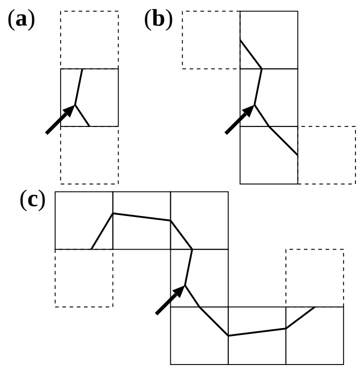

Figure 2: *Computing a seeded contour line. (**a**) Creating contour line fragment inside selected cell and determining its neighbours; (**b**) and (**c**) growing contour line by propagation through adjacent cells*

Streamlines are one means to visualize vector (flow-field) data. A streamline $p(t)$ is a solution to the initial value problem $p(t_0) = x_0$, $\frac{d}{dt}p(t) = f\big(p(t)\big)$. In a more physical sense, it describes the path of a massless particle released into a steady flow field $f(x)$ at point x_0. The standard method to compute streamlines is to iteratively solve the initial value problem using a Runge–Kutta method [27], see Fig. 3. Since this computation method is directly applicable to interactive visualization – the iteration can be interrupted at any time – streamlines are generated by selecting a starting point x_0 inside the domain, and then performing a fixed number of Runge–Kutta iteration steps. The generated sample points are finally connected by line segments to form a streamline.

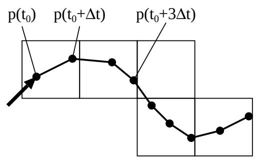

Figure 3: *Generating a streamline by iteratively solving an initial value problem starting at point* $x_0 = p(t_0)$

Currently, these two simple visualization primitives are the only ones supported. At this point, interactive placement of contour lines and streamlines is the most efficient way to visualize the two described phenomena. Once we gain more insight into the computational problem, tracking methods like the one presented in [22] might be applicable.

5.1.2. Multi-valued Data

In the case of multi-valued data sets, different data components can be selected for visualization-primitive generation. For example, a data set could contain one scalar quantity (density) and one vector quantity (momentum), as is common in many types of CFD simulations. In this case, contour lines could be generated for density, x- and y-components of momentum, or momentum magnitude. Primitives can be colored by any scalar component. Combining different components of possibly different fields for visualization, e. g., coloring contour lines of density with momentum magnitude, can aid in understanding data.

5.1.3. Simulation Data Update

The system described in [23] reads pre-computed data for visualization, while the system we describe here is directly connected to a simulation via a network interface. This allows a user to update the visualized data to the most recent transient solution in the simulation at any time. When the simulation data is updated, the already placed visualization primitives will follow the change. The visualization system can also automatically update the data whenever a new solution is available, thus creating a real-time animation. By following the changing visualization primitives over time, a user can more easily understand how a simulation progresses.

Since all described visualization primitives use the notion of a starting or seed point, we currently update all existing primitives by recalculating them, using the new data, starting from the same point. For streamlines, this is appropriate; for contour lines, however, keeping the same seed point might change the data value visualized by the contour line. This updating method is sufficient to visualize a simulation's evolution, but we intend to implement "tracking" of isovalues in a future program version.

5.2. Manipulating Simulation Parameters

In a tightly coupled simulation-visualization framework, data visualization is only one part of the collaboration. If visualization is used to "supervise" the simulation, it can also be used to "steer" the simulation. The goals of steering, i. e., interactive changing of simulation parameters, can be either to increase solution accuracy, to decrease run time, or to keep a simulation "on track." The details on how to steer a simulation depend on the simulated phenomenon, or, more specifically, on the parameters exposed to the visualization system and used for "tweaking." In computational steering, the process of retrofitting a stand-alone simulation code for interaction is often called "instrumentation."

In the two computational problems described above, we chose to instrument the simulation codes by allowing a user to change the grid during simulation, in order to adjust the region of finest refinement to the location where the most interesting features evolve. As described in Section 4, the grids for both problems contain a uniform refinement area, with grid cells growing exponentially in size when moving away from that region, see Fig. 4. To accomodate grid changes, the visualization system provides a "magnifying glass" that allows a user to interactively manipulate all parameters defining the grid. When these parameters are sent to an ongoing simulation, interpolation routines in the simulation code resample the current transient solution onto the new grid, and the simulation continues.

In principle, the described steering framework can be adapted to manipulate other simulation parameters as well, depending on the simulation being controlled. In the particular case of the two described computational problems, local grid refinement is by far the most important parameter that needs adjustment during a simulation.

5.3. Visualization System Architecture

The relative complexity of a coupled visualization system is due to the fact that it has to work closely to the simulation system. This implies that it must directly work on the data structures provided by the simulation; converting data to a canonical internal format is not an option. Since the visualization system also has to be interactive, the modules generating its primitives must also be tightly knit into the simulation's data formats. Thus, it seems that such a system has to be reimplemented from the ground up for each simulation it is to be coupled with. Upon closer inspection it turns out, however, that it is possible to isolate a small number of

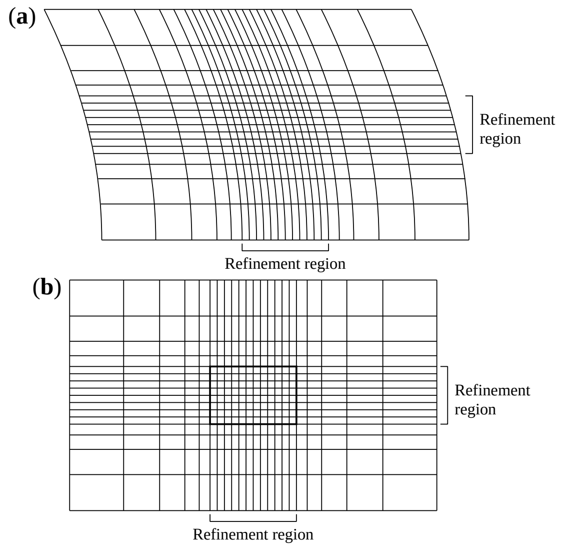

Figure 4: *Grid structures used in computational problems. In both grid(a) and grid(b), a local, uniform refinement region is surrounded by grid cells growing exponentially in size. (**a**) Curvilinear grid used in reflection problem; (**b**) Rectilinear grid used in focusing problem.*

basic interfaces that allow to isolate the structure of the simulation data from the rest of the visualization system without compromising efficiency.

Our visualization system consists of two main modules: The first, Data Set Storage, handles representation of simulation data in various formats; the second, Visualization, contains the functionality to generate and update visualization primitives, and to display them and interact with them in the user interface. To connect those two modules, the only two interfaces required for both provided visualization primitives, seeded isolines and streamlines, are the `Locator` and `IsoFragment` interfaces. The `Locator` interface allows navigating a data set and evaluating it at arbitrary points inside its domain. It is also used to communicate positions inside a data set between different system modules; in this respect, it plays a role analogous to the `iterator` interface found in the C++ Standard Template Library. The `IsoFragment` interface is specific to generating seeded isolines; it encapsulates how an isoline is generated iteratively, one line segment at a time, and how isolines propagate through grid cells. The visualization system's architecture is shown in Fig. 5.

6. Results

We have used the described simulation-visualization system to compute and explore solutions of the shock reflection problem described in Section 4 for values of the parameter a (see [7]) close to the detachment and sonic point values. In

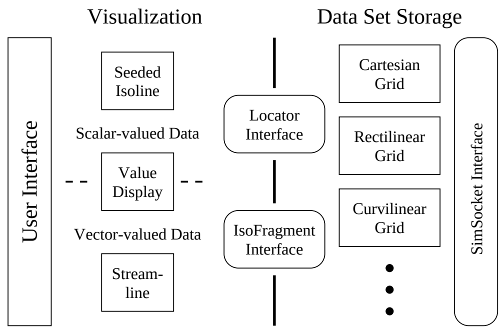

Figure 5: *Architecture of interactive visualization system.*

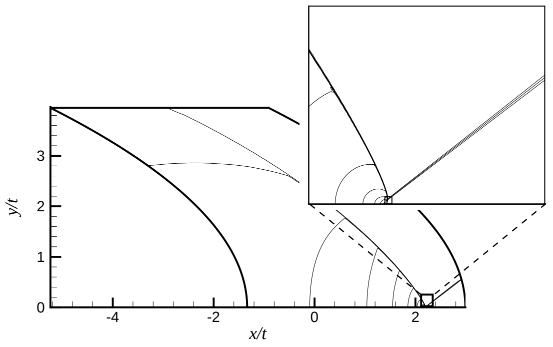

Figure 6: *A solution on the full numerical domain, illustrating the approximate size and location of the refined uniform grid area shown in Fig. 7, which is contained in the small rectangular box shown in the inset figure. The plots show contour lines of u-velocity for a Mach reflection near transition.*

our solutions, we locally refined the grid over a very small area close to the shock reflection point. In order to illustrate the size and location of the refined uniform grid, in Fig. 6 we plot contour lines of the x-velocity component u as a function of $(x/t, y/t)$ over the full numerical domain. The solution shown in Fig. 6 is for a value of a corresponding to Mach reflection. The refined grid area is too small to be visible in the main plot shown in Fig. 6. The inset figure shows an enlargement of the solution contained within the small rectangular box centered about the reflection point, as indicated. The solution shown in the figure inset also contains a small box centered at the reflection point, indicating the approximate location and size of the refined uniform grid.

During a computation, we interactively refined the grid until the solution near the reflection point was sufficiently well resolved. Fig. 7 shows a sequence of solutions computed on consecutively refined grids, for the same value of a as in Fig. 6. The regions shown contain the refined uniform grids, and correspond in area to the boxed region shown in the inset of Fig. 6. In this sequence, we have refined each grid by a factor of two in x/t and y/t in relation to the previous grid. The plots shown contour lines of u, and the contour lines are plotted at the same values of u in Fig. 7(a)–(d). The incident shock becomes more well defined as the grid is refined, and the Mach shock, which is extremely short for the value of a used here, identifies the solution as a Mach reflection.

In Fig. 8 we show u contour lines for a sequence of solutions computed on grids corresponding in resolution to those in Fig. 7, for a value of a corresponding to regular reflection near transition. The numbers of grid points in the refined regions shown in Fig. 7 and Fig. 8 are indicated in the figure captions. The shocks become sharper and thinner as the grid is refined in Fig. 8(a)–(d). A regular reflection is clearly visible. When computing solutions for values of a very close to the sonic and detachment values, however, we have not been able to discern the difference between Mach and regular reflection, due to the extreme shortness of the Mach shock at transition and the inherent numerical diffusion of our method.

In Fig. 9, we plot u contour lines, as a function of (x,y), for solutions at time $t = 1$ corresponding to the linear and non-linear configurations for shock focusing. In our computations, we maintained a fixed refined grid size, and moved the refined grid region interactively as necessary so that it always contained the triple shock intersection. Fig. 10 shows u contour lines for the non-linear focusing solution shown in Fig. 9(b), restricted to a small area near the reflection region. The number of points in the refined uniform grid, which is contained in the region shown in Fig. 10, is given in the figure caption. These solutions are preliminary, and further work, including more highly refined solutions, is required in order to resolve the solution near the triple point.

6.1. Visualization System Performance

We are routinely running the described simulation/visualization framework as a distributed application on two consumer-level desktop workstations (Intel Pentium/AMD Athlon 800 MHz, 128 MB, NVidia GeForce, Linux OS) connected by a 100 Mbit/s network. At the typical grid size of 1800×1500 cells, the simulation system needs between two and three seconds to calculate a new transient solution. The visualization system is always interactive: panning/zooming of a fixed visualization performs at above 30 frames/s, dragging of visualization primitives is performed at at least 10 frames/s. Transmitting a transient solution from the simulation system to the visualization system is limited by network bandwidth; in our environment and at typical grid size, the transmission time is between three and five seconds, during which time the visualization system is still responsive. If the simulation and visualization systems run on the same machine, transmission time becomes negligible. After a solution has been transmitted, it takes the visualization system about 0.1 to 0.5 seconds to update its internal data structures and re-calculate all placed primitives, during which time it does not respond to user interaction. Even in “movie mode,” where new transient solutions are transmitted as soon as the simulation system produces them, interactive exploration of the time-varying data is still possible.

7. Conclusions and Future Work

We have shown that integrating simulation codes and interactive visualization systems offers several benefits: Exploring data in real-time simplifies the process of understanding data; observing an ongoing simulation aids in debugging the simulation program and also understanding the problem being simulated; and interactively refining grids on-the-fly can improve simulation quality and/or reduce overall simulation time. We implemented such an integrated simulation–visualization system for the two described problems. Although these problems use different underlying data structures, the described architecture encapsulates those differences inside a single module. Using our system to compute highly refined solutions helped us to understand the inherent difficulties in the computational approach.

Future work includes adding more visualization primitives and more data structures to allow us to couple the visualization system with a wider range of simulation systems. For each simulation system, specific steering methods have to be developed and integrated into the existing framework. We intend to use the system to aid in further improving our computational method.

Acknowledgements

This work was supported by the National Science Foundation under contract ACI 9624034 (CAREER Award), through the Large Scientific and Software Data Set Visualization (LSSDSV) program under contract ACI 9982251, and through the National Partnership for Advanced Computational Infrastructure (NPACI); the National Institute of Mental Health and the National Science Foundation under contract NIMH 2 P20 MH60975–06A2, the Office of Naval Research under contract N00014–97–1–0222; the Army Research Office under contract ARO 36598–MA–RIP; the Lawrence Livermore National Laboratory under ASCI ASAP Level–2 Memorandum Agreement B347878 and under Memorandum Agreement B503159; and the Lawrence Berkeley National Laboratory. We also acknowledge the support of ALSTOM Schilling Robotics and SGI. We thank the members of the Visualization and Graphics Research Group at the Center for Image Processing and Integrated Computing (CIPIC) at the University of California, Davis.

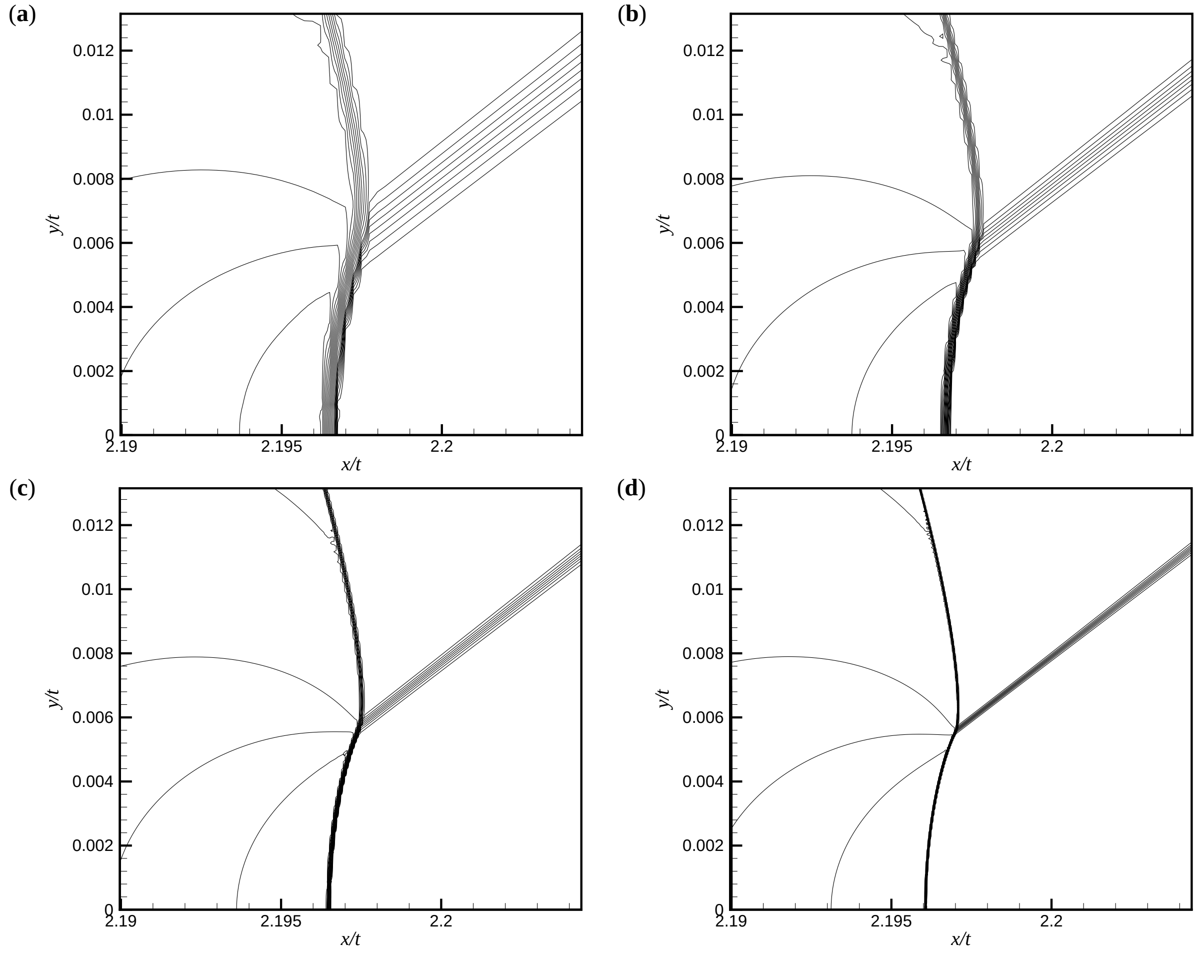

Figure 7: *A sequence of solutions in the refined uniform grid area, corresponding to the boxed area in the inset figure of Fig. 6, showing the effect of local grid refinement on the numerical solution. The plots show u-contours for solutions with the same value of a as in Fig. 6. The refined uniform grids, contained in the regions shown, have the following numbers of grid points: (a)* 56×62*; (b)* 112×124*; (c)* 224×248*; (d)* 448×496*. The total number of grid points used in the most refined computation in (d) is approximately* 4×10^6.

References

1. M. J. BERGER, AND P. COLELLA, Local adaptive mesh refinement for shock hydrodynamics, *J. Comput. Phys.*, **82** (1989), pp. 64–84.

2. J. BELL, M. J. BERGER, J. SALTZMAN, AND M. WELCOME, Three-dimensional adaptive mesh refinement for hyperbolic conservation laws, *SIAM J. Sci. Comput.*, **15** (1994), pp. 127–138.

3. M. BRIO AND J. K. HUNTER, Mach reflection for the two-dimensional Burgers equation, *Physica. D*, **60** (1992), pp. 194–207.

4. P. COLELLA, AND L. F. HENDERSON, The von Neumann paradox for the diffraction of weak shock waves, *J. Fluid. Mech.*, **213** (1990), pp. 71–94.

5. L. F. HENDERSON, Regions and boundaries for diffracting shock wave systems, *Z. Angew. Math. Mech.* **67** (1987), pp. 73–86.

6. J. K. HUNTER, Nonlinear geometrical optics, in *Multidimensional Hyperbolic Problems and Computations*, J. Glimm and A. Majda, eds., IMA Vol. **29**, Springer-Verlag, New York, 1991, pp. 179–197.

7. J. K. HUNTER, AND M. BRIO, Weak shock reflection, *J. Fluid. Mech.*, **410** (2000), pp. 235–261.

8. A. JAMESON, Aerodynamic design via control theory, *J. Sci. Comput.*, **3** (1988), pp. 233-60.

9. A. JAMESON, AND J. J. ALONSO, Automatic aerodynamic optimization on distributed memory architec-

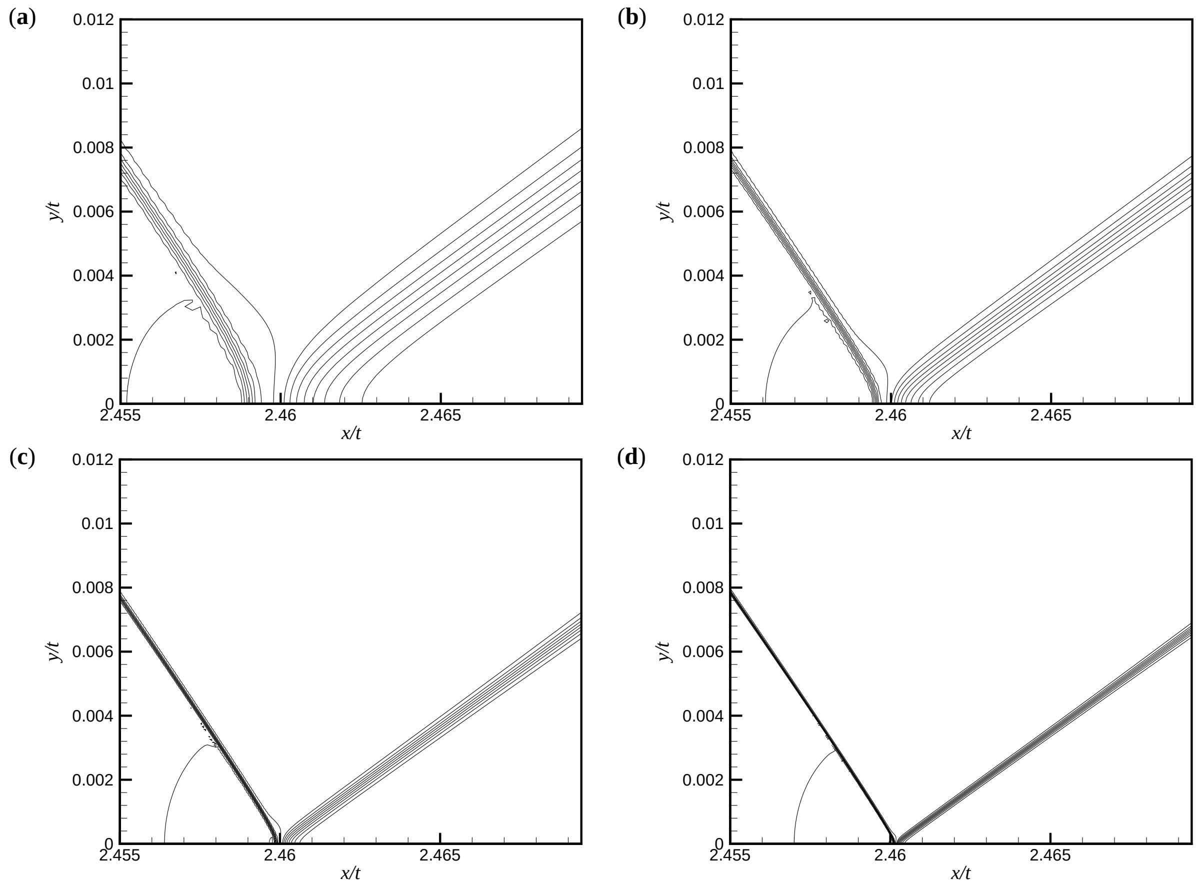

Figure 8: *A sequence of solutions for a regular reflection near transition, showing the effect of local grid refinement on the numerical solution. The plots show u contour lines in the refined grid area near the reflection point. The regions shown contain the refined uniform grids, which contain the same number of grid points as the plots shown in Fig. 7.*

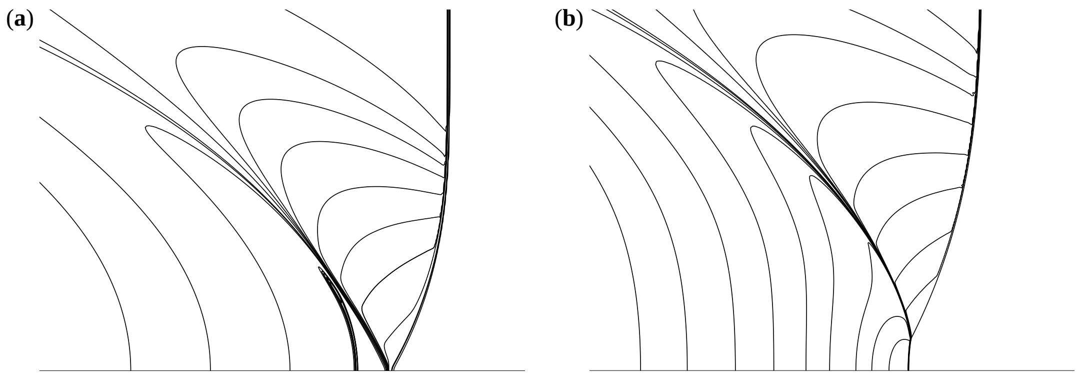

Figure 9: *Solutions for shock focusing, showing the (a) linear fish-tail, and (b) non-linear configurations at time $t = 1$. The plots show u contour lines.*

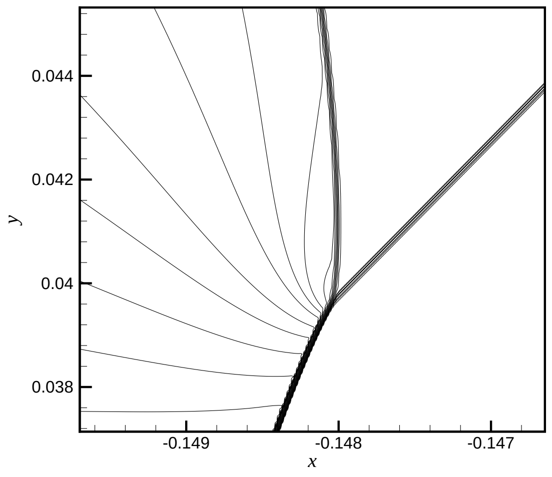

Figure 10: *The solution near the reflection point for non-linear focusing, showing contour lines of u, at t* $= 0.65$*. The region shown contains the refined uniform grid, which has* 320×480 *points.*

tures, AIAA Paper 96-0409, 34th Aerospace Sciences Meeting and Exhibit, Reno, Nevada, January 1996.

10. K. H. KAO, AND M. S. LIOU, Grid adaptation using chimera composite overlapping meshes, *AIAA J.*, **32** (1994), pp. 942–949.

11. K. MATSUNO, M. YAMAKAWA, N. SATOFUKA, Overset adaptive grid method with applications to compressible flows, *Computers and Fluids*, **27** (1998), pp. 599–610.

12. D. MAVRIPLIS, Mesh generation and adaptivity for complex geometries and flows, Handbook of Computational Fluid Mechanics, 417–459, Academic Press, San Diego, CA, 1996.

13. C. S. MORAWETZ, Potential theory for regular and Mach reflection of a shock at a wedge, *Comm. Pure Appl. Math.*, **XLVII** (1994), pp. 593–624.

14. J. PERAIRE, M. VAHDATI, K. MORGAN, AND O. C. ZIENKIEWICZ, Adaptive remeshing for compressible flow computations, *J. Comp. Phys.*, **72** (1987), pp. 449–466.

15. J. J. REUTHER, A. JAMESON, J. FARMER, L. MARTINELLI, AND D. SAUNDERS Aerodynamic shape optimization of complex aircraft configurations via an adjoint formulation, AIAA Paper 96-0094, 34th Aerospace Sciences Meeting and Exhibit, Reno, Nevada, January 1996.

16. J. J. REUTHER, A. JAMESON, J. J. ALONSO, M. J. RIMLINGER, AND D. SAUNDERS, Constrained multipoint aerodynamic shape optimization using an adjoint formulation and parallel computers, *Journal of Aircraft*, **36** (1999), pp. 51–60.

17. J. J. REUTHER, J. J. ALONSO, M. J. RIMLINGER, AND A. JAMESON, Aerodynamic shape optimization of supersonic aircraft configurations via an adjoint formulation on distributed memory parallel computers, *Computers and Fluids*, **28** (1999), pp. 675–700.

18. E. G. TABAK, AND R. R. ROSALES, Focusing of weak shock waves and the von Neumann paradox of oblique shock reflection, *Phys. Fluids*, **6** (1994), pp. 1874–1892.

19. A. M. TESDALL, Self-similar solutions for weak shock reflection, Ph.D. thesis, University of California, Davis, 2001.

20. A. M. TESDALL, AND J. K. HUNTER, Self-similar solutions for weak shock reflection, to be published in *Siam. J. Appl. Maths*

21. J. D. MULDER, J. J. VAN WIJK, AND R. VAN LIERE, A Survey of Computational Steering Environments, *Future Generation Computer Systems* **15(1)** (1999), pp. 119–129

22. F. REINDERS, F. H. POST, AND H. J. W. SPOELDER, Attribute-Based Feature Tracking, *Proceedings of the Joint EUROGRAPHICS and IEEE TVCG Symposium on Visualization, Vienna, Austria, May 1999, pp. 63–72*

23. O. KREYLOS, E. W. BETHEL, T. J. LIGOCKI, AND B. HAMANN, Virtual-Reality Based Interactive Exploration of Multiresolution Data, to appear in: Farin, G., Hagen, H. and Hamann, B., eds., *Hierarchical Approximation and Geometrical Methods for Scientific Visualization*, Springer-Verlag, Heidelberg, Germany

24. J. BLOOMENTHAL, Polygonization of Implicit Surfaces, *Computer Aided Geometric Design* **5(4)** (1988), pp. 341–356

25. W. E. LORENSEN AND H. E. CLINE, Marching Cubes: A High Resolution 3D Surface Construction Algorithm, *Proc. of SIGGRAPH '87* (1987), pp. 163–169

26. G. M. NIELSON AND B. HAMANN, The Asymptotic Decider: Resolving the Ambiguity in Marching Cubes, *Proc. of Visualization '91* (1991), IEEE Computer Society Press, Los Alamitos, CA, pp. 83–91

27. W. H. PRESS, S. A. TEUKOLSKY, W. T. VETTERLING AND B. P. FLANNERY, Numerical Recipes in C, 2nd ed. (1992), Cambridge University Press, Cambridge, MA