INTERACTIVE REAL TIME FLOW SIMULATIONS

By
I. Sadrehaghighi, Graduate Research Assistant
and
S. N. Tiwari, Principal Investigator

Progress Report
For the period ended September 30, 1990

Prepared for
National Aeronautics and Space Administration
Langley Research Center
Hampton, Virginia 23665

Under
Research Grant NCC1-68
Dr. Robert E. Smith, Jr., Technical Monitor
ACD-Computer Applications Branch

October 1990
INTERACTIVE REAL TIME FLOW SIMULATIONS

By
I. Sadrehaghighi, Graduate Research Assistant
and
S. N. Tiwari, Principal Investigator

Progress Report
For the period ended September 30, 1990

Prepared for
National Aeronautics and Space Administration
Langley Research Center
Hampton, Virginia 23665

Under
Research Grant NCC1-68
Dr. Robert E. Smith, Jr., Technical Monitor
ACD-Computer Applications Branch

Submitted by the
Old Dominion University Research Foundation
P.O. Box 6369
Norfolk, Virginia 23508-0369

October 1990
ABSTRACT

INTERACTIVE REAL TIME FLOW SIMULATIONS

Ideen Sadrehaghighi
Surendra N. Tiwari

An interactive real time flow simulation technique is developed for an unsteady channel flow. A finite-volume algorithm in conjunction with a Runge-Kutta time stepping scheme has been developed for two-dimensional Euler equations. A global time step has been used to accelerate convergence of steady-state calculations. A raster image generation routine has been developed for high speed image transmission which allows user to have direct interaction with the solution development. In addition to theory and results, the hardware and software requirements are discussed.
ACKNOWLEDGMENTS

This is a progress report on the research project "Numerical Solutions of Three-Dimensional Navier-Stokes Equations for Closed-Bluff Bodies" for the period ended September 30, 1990. Specific efforts during this period were directed in the area of "Interactive Real Time Flow Simulations."

This work was supported by the NASA Langley Research Center through Cooperative Agreement NCC1-68. The cooperative agreement was monitored by Dr. Robert E. Smith, Jr., of the Analysis and Computation Division (Computer Applications Branch), NASA Langley Research Center, Mail Stop 125.
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ABSTRACT</td>
<td>iii</td>
</tr>
<tr>
<td></td>
<td>ACKNOWLEDGEMENTS</td>
<td>iv</td>
</tr>
<tr>
<td></td>
<td>TABLE OF CONTENTS</td>
<td>v</td>
</tr>
<tr>
<td></td>
<td>LIST OF FIGURES</td>
<td>vi</td>
</tr>
<tr>
<td></td>
<td>LIST OF SYMBOLS</td>
<td>vii</td>
</tr>
<tr>
<td>1.</td>
<td>INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>2.</td>
<td>GRID GENERATION</td>
<td>3</td>
</tr>
<tr>
<td>3.</td>
<td>EQUATIONS OF MOTION AND METHOD OF SOLUTION</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>3.1 Governing Equations</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>3.2 Finite-Volume Scheme</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>3.3 Time-Stepping Scheme</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>3.4 Bondary Conditions</td>
<td>8</td>
</tr>
<tr>
<td>4.</td>
<td>SOFTWARE AND HARDWARE CONSIDERATIONS</td>
<td>9</td>
</tr>
<tr>
<td>5.</td>
<td>RESULTS AND DISCUSSIONS</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>REFERENCES</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>APPENDIX A: RASTER IMAGE GENERATION ROUTINE: PLOTD</td>
<td>19</td>
</tr>
</tbody>
</table>
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Stretching transformation ($\eta = 0$)</td>
<td>4</td>
</tr>
<tr>
<td>2.2</td>
<td>Computational grid for a channel flow</td>
<td>4</td>
</tr>
<tr>
<td>4.1</td>
<td>CRAY and UltraNet Graphics Display Device</td>
<td>11</td>
</tr>
<tr>
<td>4.2</td>
<td>High performance multicomputer network</td>
<td>11</td>
</tr>
<tr>
<td>5.1</td>
<td>Pressure contours for different iterations ($\alpha = 0.8, \ M_\infty = 2.0$)</td>
<td>14</td>
</tr>
<tr>
<td>5.2</td>
<td>Mach number contours for different iterations ($\alpha = 0.8, \ M_\infty = 2.0$)</td>
<td>15</td>
</tr>
<tr>
<td>5.3</td>
<td>Pressure contours for different iterations ($\alpha = 15.0, \ M_\infty = 2.0$)</td>
<td>16</td>
</tr>
<tr>
<td>5.4</td>
<td>Mach number contours for different iterations ($\alpha = 15.0, \ M_\infty = 2.0$)</td>
<td>17</td>
</tr>
</tbody>
</table>
LIST OF SYMBOLS

Q = vector of conservative variables
F,G = flux vectors for coordinate directions
x , y = physical coordinates
h = height of channel
\bar{x} , \bar{y} = computational coordinates
u , v = velocity components in x and y directions
p = static pressure
M_\infty = free stream Mach number
\rho = density
E = total energy
e = energy per unit volume
S = cell area
\gamma = ratio of specific heats
\alpha = angle of attack
\beta = stretching parameter
\eta = clustering parameter
\delta = boundary-layer thickness
Chapter 1

INTRODUCTION

Since an interactive design process is one of the ultimate goals of Computational Fluid Dynamics (CFD), real time flow simulation with direct user interaction is an ideal approach in achieving this goal [1]*. This process requires a supercomputer with vast amount of memory, an extremely high bandwidth communication network and highly capable graphic workstations. Even with today’s rapid advances in the supercomputer developments, some of the components are not fast or large enough for a realistic three-dimensional problem. However, real time simulation of medium size two-dimensional flow problems (Euler equations) are possible today.

Accurate simulations are critical to the development and understanding of highly unsteady flow. For the reasons outlined above, a simple unsteady two-dimensional channel type flow has been studied here. The relative motion of the shocks and other strong gradients have been examined as the solution being computed. Once the solution has been examined, the user will inform the flow solver to take different action or to continue on the existing course. This requires large amount of data to be examined by the user, and small amount of information in form of instruction, to be send back to the mainframe computer. Consequently, this procedure requires a fast mainframe computer, an extremely fast network for data communication and a relatively fast workstation.

The computational process is initiated on the mainframe computer under

* The numbers in brackets indicate references.
interactive control by the user at a workstation. The equations of motion are integrated step by step on the mainframe computer. The choice of a variable to be visualized is instructed from the workstation and a separate rasterization program computes a raster image of the variable. The image is transmitted over the communication link to a raster display device to be viewed by the user. Images are continuously created, transmitted, and if desired, stored on a recording device. Having viewed the images on the raster display device, the user responds. If mainframe's computational and communication rates are sufficient, a real time interaction can be achieved. This, of course, depends on the size of the problem under investigation (i.e. number of grid points, equations of motion, and solution technique).
A suitable transformation for a channel type flow can be obtained using an analytical function. A simple case would be to use uniform spacing in x direction. For y direction, the spacing is obtained by [2],

\[
y = h \frac{\beta + 2\eta/((\beta + 1)/(\beta - 1))^{(\eta-\eta)/(\eta+\eta)} - \beta + 2\eta}{(2\eta + 1)(1 + [(\beta + 1)/(\beta - 1)]^{(\eta-\eta)/(\eta+\eta)})}
\] (2.1)

where \( h \) is the height of channel and \( \eta \) is the parameter which controls the grid clustering in y direction. Variable \( \bar{y} \) corresponds to normal coordinate in the computational domain. The stretching parameter, \( \beta \), is related approximately to the non-dimensional boundary layer thickness \( \delta/h \) by

\[
\beta = \left(1 - \frac{\delta}{h}\right)^{-\frac{1}{\eta}} , \quad 0 < \frac{\delta}{h} < 1
\] (2.2)

The amount of stretching for various values of \( \delta/h \) is illustrated in Fig. 2.1. For this transformation, if \( \eta = 0 \) the mesh will be refined near \( y = h \), whereas, if \( \eta = 0.5 \) the mesh will be refined equally near \( y = 0 \) and \( y = h \). For this study, values of \( \eta = 0.5 \) and \( \beta = 0.30 \) have been chosen and the resulting grid is shown in Fig. 2.2.
Fig. 2.1 Stretching transformation ($\eta = 0$)

Fig. 2.2 Computational grid for a channel flow
Chapter 3

EQUATIONS OF MOTION AND
METHOD OF SOLUTION

3.1 Governing Equations

The governing equations are time-dependent Euler equations [3]. The unsteady two-dimensional equations for a compressible perfect gas can be written in an integral form for a region $\Omega$ with boundary $\partial \Omega$ as

$$\frac{\partial}{\partial t} \int_{\Omega} Q \, dx \, dy + \int_{\partial \Omega} (F \, dy - G \, dx) = 0$$

(3.1)

where vectors $Q$, $F$, and $G$, are

$$Q = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ E \end{bmatrix}, \quad F = \begin{bmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ (E + p)u \end{bmatrix}, \quad G = \begin{bmatrix} \rho v \\ \rho uv \\ \rho v^2 + p \\ (E + p)v \end{bmatrix}$$

(3.2)

The total energy, $E$, is defined using the ideal gas relation as

$$E = \frac{p}{(\gamma - 1)} + \frac{1}{2} \rho (u^2 + v^2)$$

(3.3)

where $\gamma$ is the ratio of specific heats.
3.2 Finite-Volume Scheme

The governing equations in the integral form are applied to an arbitrary quadrilateral and the line integrals are approximated with the midpoint rule. The discretized result is

\[
\frac{d}{dt}(S_{ij}Q_{i,j}) + LQ_{i,j} = 0 \tag{3.4}
\]

where \( L \) is defined as the spatial discretization operator and \( S_{ij} \) is area of the cell. The components of \( Q_{i,j} \) represent the cell-averaged quantities and obtained by the mean values of the fluxes crossing the cell.

In order to overcome the instabilities associated with central differencing, a fourth order dissipative model is used. In this study, the dissipative model developed by Jameson, Schmidt, and Turkel [4] has been used. The finite-volume formulation, Eq. (3.4) is now expressed as

\[
\frac{d}{dt}(S_{i,j}Q_{i,j}) + Q_{i,j} - DQ_{i,j} = 0 \tag{3.5}
\]

where \( D \) represents the artificial dissipation operator. This operator can be written in the following way

\[
DQ_{i,j} = D_xQ_{i,j} + D_yQ_{i,j} \tag{3.6}
\]

where \( D_xQ_{i,j} \) and \( D_yQ_{i,j} \) are the contributions from each of the coordinate directions. In conservation form

\[
D_xQ_{i,j} = d_{i+1/2,j} - d_{i-1/2,j} \tag{3.7}
\]

\[
D_yQ_{i,j} = d_{i,j+1/2} - d_{i,j-1/2} \tag{3.8}
\]

where the terms on the right hand side have the form
\[ d_{i+1/2,j} = \frac{S_{i+1/2,j}}{\Delta t} [\varepsilon_{i+1/2,j}^{(2)} (Q_{i+1,j} - Q_{i,j}) - \varepsilon_{i+1/2,j}^{(4)} (Q_{i+2,j} - 3Q_{i+1,j} + 3Q_{i,j} - Q_{i-1,j})] \]  

(3.9)

The coefficients \( \varepsilon^{(2)} \) and \( \varepsilon^{(4)} \) are determined from the pressures gradients as

\[
\nu_{i,j} = \frac{|P_{i+1,j} - 2P_{i,j} + P_{i-1,j}|}{P_{i+1,j} + 2P_{i,j} + P_{i-1,j}} 
\]

(3.10)

\[
\varepsilon_{i+1/2,j}^{(2)} = \alpha^{(2)} \max(\nu_{i+1}, \nu_{i,j}) 
\]

(3.11)

\[
\varepsilon_{i+1/2,j}^{(4)} = \max(0, (\alpha^{(4)} - \varepsilon_{i+1/2,j}^{(2)})) 
\]

(3.12)

where typical values of the constants \( \alpha^{(2)} \) and \( \alpha^{(4)} \) are \( \frac{1}{4} \) and \( \frac{1}{256} \), respectively.

### 3.3 Time-Stepping Scheme

A modified four stage Runge-Kutta time stepping scheme was used to advance the solution in time. Due to its explicit character, this scheme is very simple and flexible. Hence, it is ideal for interactive computation purposes. At time level \( n \),

\[
Q^{(0)} = Q^{(n)} 
\]

(3.13)

\[
Q^{(1)} = Q^{(0)} - \alpha_1 \Delta t RQ^{(0)} 
\]

(3.14)

\[
Q^{(2)} = Q^{(0)} - \alpha_2 \Delta t RQ^{(1)} 
\]

(3.15)

\[
Q^{(3)} = Q^{(0)} - \alpha_3 \Delta t RQ^{(2)} 
\]

(3.16)

\[
Q^{(4)} = Q^{(0)} - \Delta t RQ^{(3)} 
\]

(3.17)

\[
Q^{(n+1)} = Q^{(4)} 
\]

(3.18)

where on the \((q+1)\)st stage

\[
RQ^{(q)} = \frac{1}{S}(LQ^{(q)} - DQ^{(0)}) 
\]

(3.19)
and

\[ \alpha_1 = \frac{1}{4}, \quad \alpha_2 = \frac{1}{3}, \quad \alpha_3 = \frac{1}{2} \]  \hfill (3.20)

### 3.4 Boundary Conditions

The boundary conditions are implemented by applying a slip velocity condition at the solid walls [5]. Also, the stagnation enthalpy is assumed to be constant along the solid surface and equal to that of the free-stream, i.e.,

\[ h_t = h + \frac{1}{2}(u^2 + v^2) = h_{\infty} \]  \hfill (3.21)

For solid wall, \((h_t)_{wall}\), the free-stream values \((u=1 \text{ and } v=0)\) are used, i.e.,

\[ (h_t)_{wall} = \gamma e_{F.S.} + \frac{1}{2} \]  \hfill (3.22)

Because of the supersonic nature of the flow, the outflow boundary is determined by extrapolating from interior points.
Chapter 4
SOFTWARE AND HARDWARE CONSIDERATIONS

The implemented software and hardware are discussed in [1]. Currently, the mainframe computer is a CRAY-2 which generally performs at 200-250 MFLOPS (Million Floating Point Operations Per Second). The mainframe is connected to UltraNet Graphics Display Device (UGDD), which is a network based frame buffer through a High-Speed Channel (HSC). The channel can support transfer rates up to 100 MBytes/sec. The frame buffer is a high-speed display system supporting high resolution monitors. The UGDD contains two memory arrays. While one array is displayed, the other is updated over the network. The user determines which array to display and controls the situation over the network. The UGDD supports a single user at a time and may be used to display color images at animation rates. The frame buffer can display RGB (Red, Green, Blue) images up to 1280 pixels horizontally by 1024 Scan-lines vertically. Figure 4.1 shows a pictorial representation of the hardware setup for a single hub (UltraNet 1000). Figure 4.2 shows the configuration using a CRAY supercomputer in conjunction with several mini-supercomputers and workstations.

There are several pieces of software required for a real time CFD simulation. They are: a grid generator, a flow solver, an image rasterizer, and an image manipulator. Each software component has been written and applied separately. However, for a true interactive process, these softwares must be integrated into one working
unit or be presented in parallel.

With the current CFD technology, it is possible to solve the Euler equations at 16 \( \frac{\mu \text{sec}}{\text{gridpoints} \times \text{iterations}} \) [6]. For a grid of 70 X 30, it will take one second for 62 iterations. This will result in 62 frames per second.

A rasterization program called PLOTD is developed to produce a raster image from the solution. A complete listing of the source code is provided in Appendix A. This software converts a specified variable defined on a surface to a raster RGB images (1024 X 768). The required CPU time for this step depends on the size of the grid and the complexity of the image. This step may take from few microseconds to a couple of seconds. Consequently, the image must not be very complex.
Fig. 4.1 CRAY and UltraNet Graphics Display Device

Fig. 4.2 High performance multicomputer network
Chapter 5

RESULTS AND DISCUSSION

The unsteady two-dimensional Euler equations are solved for a channel flow. In order to test the flow solver, the solution has been obtained by keeping all the flow parameters constant. The results are satisfactory and the shocks are captured very accurately. Figures 5.1 through 5.4 show pressure and Mach number contours at different iterations for different angles of attack. A more comprehensive test can be achieved by changing one of the flow conditions as the solution evolves. An ideal and relatively simple case would be rotating the angle of attack. The angle of attack can be expressed as

\[ \alpha = \alpha_{\text{min}} + (\alpha_{\text{max}} - \alpha_{\text{min}}) \sin(\omega \pi) \]  

where \( \alpha_{\text{min}} \) and \( \alpha_{\text{max}} \) are initial and final angles of attack, and \( \omega \) is the specified frequency. Initially, the solution starts with a uniform flow everywhere (impulse start). At this initial stage of the computation, none of the flow features has been altered. This, in fact, is the continuation of the case described earlier for testing the flow solver. After some iterations, once the flow features are established, the angle of attack is allowed to rotate using Eq. (5.1). It takes between 3-5 cycles for flow properties to reach a cyclic behavior.

Each frame takes at least 0.5 second of CPU time, which is relatively long time for a multiuser machine. One way to alleviate this is to take advantage of the parallel capability of the CRAY 2. This can be accomplished by allowing the flow
solution, rasterization and image transmission to be performed on different processors.
Fig. 5.1 Pressure contours for different iterations ($\alpha = 0.8$, $M_\infty = 2.0$)
Fig. 5.2 Mach number contours for different iterations ($\alpha = 0.8^\circ$, $M_\infty = 2.0$)
Fig. 5.3 Pressure contours for different iterations ($\alpha = 15.0^\circ$, $M_\infty = 2.0$)
after 763 iterations (converged)

after 305 iterations

after 175 iterations

after 95 iterations

after 45 iterations

Fig. 5.4 Mach number contours for different iterations ($\alpha = 15.0^\circ, M_\infty = 2.0$)
REFERENCES


Following subroutines are developed for a raster image generation. Main arguments are defined as:

\begin{itemize}
  \item \textbf{x,y} = grid coordinates
  \item \textbf{d} = flow variable to be contoured
  \item \textbf{n} = number of points in x direction
  \item \textbf{m} = number of points in y direction
  \item \textbf{is,ie} = desired start and end points for contouring in I-direction
  \item \textbf{js,je} = desired start and end points for contouring in J-direction
  \item \textbf{nnc} = number of contour levels
\end{itemize}
subroutine conplt(x,y,d,m,n,is,ie,js,je,a,b,nnc)

PARAMETER(NPMAX=10000,NPSMAX=100,NSMAX=100)

dimension x(m,n),y(m,n),d(m,n)

common /extrem/ xmin, xmax, ymin, ymax, dmin, dmax
real xmin, xmax, ymin, ymax, dmin, dmax

common /maping/ scalex, ofsetx, scaley, ofsety, scaled, ofsetd
real scalex, ofsetx, scaley, ofsety, scaled, ofsetd

dimension ix1(npsmax,3), iy1(npsmax,3)
dimension ix2(npsmax,3), iy2(npsmax,3)

logical ind(npsmax,2), flag(npsmax)

contour plot
written by: Eric L. Everton & Jamshid S. Abolhassani

nc=abs(nnc)

check for zero range
if (dmax.ne.dmin) then
compute delta
delf=(dmax-dmin)/float(nc)
else
zero range, issue error message and stop
write(*,*) 'maximum and minimum are equal'
stop
endif

set starting contour level
cl=dmin

set error flag to false
do 100 i=is,iel
flag(i)=.false.
100 continue

plot contours

150 continue

checks for contour level out of range
if(cl.lt.dmin) go to 150
if(cl.gt.dmax) return
convert contour level to look up table (lut) index
icolor=cl*scaled+offset
checks for lut index out of range
if(icolor.lt.int(a)) icolor=int(a)
if(icolor.gt.int(b)) icolor=int(b)
set current lut index (color)
call ufbcol(icolor)
c set do loop end controls
iend=iend-1
jend=jend-1
upper triangle do loop
do 200 j=js,jend
reset line plot flags
do 250 k=1,2
do 250 i=ls,iend
ind(i,k)=.false.
continue
vectorized do loop
do 300 i=ls,iend
initialize temporary variables
xl=x(i,j)
y1=y(i,j)
f1=d(i,j)
x2=x(i+1,j)
y2=y(i+1,j)
f2=d(i+1,j)
x3=x(i+1,j+1)
y3=y(i+1,j+1)
f3=d(i+1,j+1)
check to see if contour line intersects this triangle
if ((cl.le.max(f1,f2,f3)).and.(cl.ge.min(f1,f2,f3))) then
check to see if contour line intersects point 1
if (cl.eq.fl) then
  check to see if contour line intersects point 2
  if (cl.eq.f2) then
    check to see if contour line intersects point 3
    if (cl.eq.f3) then
      contour line intersects all three vertices
      ix1(i,1)=ofsetx+scalex*x1
      iy1(i,1)=ofsetx+scaley*y1
      ix2(i,1)=ofsetx+scalex*x2
      iy2(i,1)=ofsetx+scaley*y2
      ind(i,1)=.true.
      ix1(i,2)=ofsetx+scalex*x2
      iy1(i,2)=ofsetx+scaley*y2
      ix2(i,2)=ofsetx+scalex*x3
      iy2(i,2)=ofsetx+scaley*y3
      ind(i,2)=.true.
      ix1(i,3)=ofsetx+scalex*x3
      iy1(i,3)=ofsetx+scaley*y3
      ix2(i,3)=ofsetx+scalex*x1
      iy2(i,3)=ofsetx+scaley*y1
    else
      contour line intersects point 1 and point 2
      ix1(i,1)=ofsetx+scalex*x1
      iy1(i,1)=ofsetx+scaley*y1
      ix2(i,1)=ofsetx+scalex*x2
      iy2(i,1)=ofsetx+scaley*y2
      ind(i,1)=.true.
    endif
  else
    contour line did not intersect point 2
    check to see if contour line intersects point 3
    else if (cl.eq.f3) then
      contour line intersects point 1 and point 3
      ix1(i,1)=ofsetx+scalex*x1
      iy1(i,1)=ofsetx+scaley*y1
      ix2(i,1)=ofsetx+scalex*x3
      iy2(i,1)=ofsetx+scaley*y3
      ind(i,1)=.true.
    else

check to see if contour line intersects edge between point 2 and point 3

if ((cl-f2)*(cl-f3).le.0.) then
  contour line intersects point 1 and edge between point 2 and point 3
  
  ix1(i,1)=ofsetx+scalex*x1
  iy1(i,1)=ofsety+scaley*y1
  ix2(i,1)=ofsetx+scalex*(x2+(x3-x2)*(cl-f2)/(f3-f2))
  iy2(i,1)=ofsety+scaley*(y2+(y3-y2)*(cl-f2)/(f3-f2))
  ind(i,1)=.true.
endif
endif

contour line did not intersect point 1
check to see if contour line intersects point 2
else if (cl.eq.f2) then
  contour line intersects point 2
  
  ix1(i,1)=ofsetx+scalex*x2
  iy1(i,1)=ofsety+scaley*y2
  check to see if contour line intersects point 3
  if (cl.eq.f3) then
    contour line intersects point 2 and point 3
    
    ix2(i,1)=ofsetx+scalex*x3
    iy2(i,1)=ofsety+scaley*y3
    ind(i,1)=.true.
  else
    check to see if contour line intersects edge between point 1 and point 3
    if ((cl-f1)*(cl-f3).le.0.) then
      contour line intersects point 2 and edge between point 1 and point 3
      
      ix2(i,1)=ofsetx+scalex*(x1+(x3-x1)*(cl-f1)/(f3-f1))
      iy2(i,1)=ofsety+scaley*(y1+(y3-y1)*(cl-f1)/(f3-f1))
      ind(i,1)=.true.
    endif
  endif
endif
check to see if contour line intersects point 3

else if (cl.eq.f3) then

check to see if contour line intersects edge between point 1 and point 2

if ((cl-f1)*(cl-f2).le.0.) then

contour line intersects point 3 and edge between point 1 and point 2

ixl(i,1)=offsetx+scalex*x3
iy1(i,1)=offsety+scaley*y3
ix2(i,1)=offsetx+scalex*(x1+(x2-x1)*(cl-f1)/(f2-f1))
iy2(i,1)=offsety+scaley*(y1+(y2-y1)*(cl-f1)/(f2-f1))
ind(i,1)=.true.

endif

else

contour line does not intersect any of the vertices

check to see if contour line intersects edge between point 2 and point 3

if ((cl-f2)*(cl-f3).le.0.) then

contour line intersects edge between point 2 and point 3

ixl(i,1)=offsetx+scalex*(x2+(x3-x2)*(cl-f2)/(f3-f2))
iy1(i,1)=offsety+scaley*(y2+(y3-y2)*(cl-f2)/(f3-f2))

check to see if contour line intersects edge between point 1 and point 3

if ((cl-f1)*(cl-f3).le.0.) then

contour line intersects edges between point 2 and point 3 and between point 1 and point 3

ix2(i,1)=offsetx+scalex*(x1+(x3-x1)*(cl-f1)/(f3-f1))
iy2(i,1)=offsety+scaley*(y1+(y3-y1)*(cl-f1)/(f3-f1))
ind(i,1)=.true.

contour line does not intersect edge between point 1 and point 3

check to see if contour line intersects edge between point 1 and point 2

else if ((cl-f1)*(cl-f2).le.0.) then

contour line intersects edges
between point 2 and point 3 and
between point 1 and point 2

\[
\begin{align*}
ix2(1,1) &= \text{ofsetx} + \text{scalex} \times (x1 + (x2-x1) \times (cl-f1)/(f2-f1)) \\
iy2(1,1) &= \text{ofsety} + \text{scaley} \times (y1 + (y2-y1) \times (cl-f1)/(f2-f1))
\end{align*}
\]

\text{ind}(i,1) = .true.

must be a problem

else

set error flag

\text{flag}(i) = .true.

endif

contour line does not intersect edge
between point 2 and point 3
check to see if contour line intersects edge
between point 1 and point 3

else if (\((cl-f1) \times (cl-f3) \leq 0.0\)) then

contour line intersects edge
between point 1 and point 3

\[
\begin{align*}
ix1(1,1) &= \text{ofsetx} + \text{scalex} \times (x1 + (x3-x1) \times (cl-f1)/(f3-f1)) \\
iy1(1,1) &= \text{ofsety} + \text{scaley} \times (y1 + (y3-y1) \times (cl-f1)/(f3-f1))
\end{align*}
\]

check to see if contour line intersects edge
between point 1 and point 2

if (\((cl-f1) \times (cl-f2) \leq 0.0\)) then

contour line intersects edges
between point 1 and point 3 and
between point 1 and point 2

\[
\begin{align*}
ix2(1,1) &= \text{ofsetx} + \text{scalex} \times (x1 + (x2-x1) \times (cl-f1)/(f2-f1)) \\
iy2(1,1) &= \text{ofsety} + \text{scaley} \times (y1 + (y2-y1) \times (cl-f1)/(f2-f1))
\end{align*}
\]

\text{ind}(i,1) = .true.

must be a problem

else

set error flag

\text{flag}(i) = .true.

endif

contour line did not intersect edges
between point 2 and point 3 and
between point 1 and point 3
check to see if contour line intersects edge between point 1 and point 2

else if (((cl-f1)*(cl-f2)).le.0.) then

contour line only intersects edge between point 1 and point 2
must be a problem
set error flag

flag(i)=.true.
endif
endif
endif
endif

300 continue

do 350 i=is,iel

check to see if an error occurred

if (flag(i)) then

issue error message and stop

write(*,*) 'There is something fishy about conplt!' stop
endif

350 continue

do 360 i=is,iel

check to see if line is to be plotted

if (ind(i,1)) then

plot a single line

call ufblin(ixl(i,1),iyl(i,1),ix2(i,1),iy2(i,1))

check for more lines

if (ind(i,2)) then

plot two more lines

call ufblin(ixl(i,2),iyl(i,2),ix2(i,2),iy2(i,2))
call ufblin(ixl(i,3),iyl(i,3),ix2(i,3),iy2(i,3))

endif
endif

360 continue

200 continue

c lower triangle do loop
do 400 j=js,jel

c reset line plot flags
do 450 k=1,2
do 450 i=is,iel

ind(i,k)=.false.

450 continue

c vectorized do loop
do 500 i=is,iel

initialize temporary variables

xl=x(i,j)
y1=y(i,j)
f1=d(i,j)
x3=x(i+1,j+1)
y3=y(i+1,j+1)
f3=d(i+1,j+1)
x4=x(i,j+1)
y4=y(i,j+1)
f4=d(i,j+1)

check to see if contour line intersects this triangle
if ((cl.le.max(f1,f3,f4)).and.(cl.ge.min(f1,f3,f4))) then
  check to see if contour line intersects point 1
  if (cl.eq.f1) then
    check to see if contour line intersects point 3
    if (cl.eq.f3) then
      check to see if contour line intersects point 4
      if (cl.eq.f4) then
        contour line intersects all three vertices
        ix1(i,1)=ofsetx+scalex*xl
        iy1(i,1)=ofsety+scaley*yl
        ix2(i,1)=ofsetx+scalex*x3
      endif
    endif
  endif
endif
iy2(i,1)=ofsety+scaley*y3
ind(i,1)=.true.
ix1(i,2)=ofsetx+scalex*x3
iy1(i,2)=ofsety+scaley*y3
ix2(i,2)=ofsetx+scalex*x4
iy2(i,2)=ofsety+scaley*y4
ind(i,2)=.true.
ix1(i,3)=ofsetx+scalex*x4
iy1(i,3)=ofsety+scaley*y4
ix2(i,3)=ofsetx+scalex*x1
iy2(i,3)=ofsety+scaley*y1

else

contour line intersects point 1 and point 3

ix1(i,1)=ofsetx+scalex*x1
iy1(i,1)=ofsety+scaley*y1
ix2(i,1)=ofsetx+scalex*x3
iy2(i,1)=ofsety+scaley*y3
ind(i,1)=.true.
endif

contour line did not intersect point 3
check to see if contour line intersects point 4
else if (cl.eq.f4) then

contour line intersects point 1 and point 4

ix1(i,1)=ofsetx+scalex*x1
iy1(i,1)=ofsety+scaley*y1
ix2(i,1)=ofsetx+scalex*x4
iy2(i,1)=ofsety+scaley*y4
ind(i,1)=.true.
else

check to see if contour line intersects edge
between point 3 and point 4
if (((cl-f3)*(cl-f4)).le.0.) then

contour line intersects point 1 and
edge between point 3 and point 4

ix1(i,1)=ofsetx+scalex*x1
iy1(i,1)=ofsety+scaley*y1
ix2(i,1)=ofsetx+scalex*(x3+(x4-x3)*(cl-f3)/(f4-f3))
iy2(i,1)=ofsety+scaley*(y3+(y4-y3)*(cl-f3)/(f4-f3))
ind(i,1)=.true.
endif
endif
contour line did not intersect point 1
check to see if contour line intersects point 3

else if (cl.eq.f3) then
  contour line intersects point 3
  ix1(i,1)=ofsetx+scalex*x3
  iy1(i,1)=ofsety+scaley*y3
check to see if contour line intersects point 4

if (cl.eq.f4) then
  contour line intersects point 3 and point 4
  ix2(i,1)=ofsetx+scalex*x4
  iy2(i,1)=ofsety+scaley*y4
  ind(i,1)=.true.
else
  check to see if contour line intersects edge between point 1 and point 4
  if (((cl-f1)*(cl-f4)).le.0.) then
    contour line intersects point 3 and edge between point 1 and point 4
    ix2(i,1)=ofsetx+scalex*(x1+(x4-x1)*(cl-f1)/(f4-f1))
    iy2(i,1)=ofsety+scaley*(y1+(y4-y1)*(cl-f1)/(f4-f1))
    ind(i,1)=.true.
  endif
endif
check to see if contour line intersects point 4

else if (cl.eq.f4) then
  check to see if contour line intersects edge between point 1 and point 3
  if (((cl-f1)*(cl-f3)).le.0.) then
    contour line intersects point 4 and edge between point 1 and point 3
    ix1(i,1)=ofsetx+scalex*x4
    iy1(i,1)=ofsety+scaley*y4
    ix2(i,1)=ofsetx+scalex*(x1+(x3-x1)*(cl-f1)/(f3-f1))
    iy2(i,1)=ofsety+scaley*(y1+(y3-y1)*(cl-f1)/(f3-f1))
    ind(i,1)=.true.
endif

else

contour line does not intersect any of the vertices
check to see if contour line intersects edge
between point 3 and point 4

if ((cl-f3)*(cl-f4).le.0.) then

contour line intersects edge
between point 3 and point 4

ix1(i,1)=offsetx+scalex*(x3+(x4-x3)*(cl-f3)/(f4-f3))
iy1(i,1)=offsety+scaley*(y3+(y4-y3)*(cl-f3)/(f4-f3))

check to see if contour line intersects edge
between point 1 and point 4

if ((cl-f1)*(cl-f4).le.0.) then

contour line intersects edges
between point 3 and point 4 and
between point 1 and point 4

ix2(i,1)=offsetx+scalex*(x1+(x4-x1)*(cl-f1)/(f4-f1))
iy2(i,1)=offsety+scaley*(y1+(y4-y1)*(cl-f1)/(f4-f1))
ind(i,1)=.true.

contour line does not intersect edge
between point 1 and point 4
check to see if contour line intersects edge
between point 1 and point 3

delse if ((cl-f1)*(cl-f3).le.0.) then

contour line intersects edges
between point 3 and point 4 and
between point 1 and point 3

ix2(i,1)=offsetx+scalex*(x1+(x3-x1)*(cl-f1)/(f3-f1))
iy2(i,1)=offsety+scaley*(y1+(y3-y1)*(cl-f1)/(f3-f1))
ind(i,1)=.true.

must be a problem
else

set error flag
flag(i)=.true.
endif

contour line does not intersect edge

between point 3 and point 4
check to see if contour line intersects edge
between point 1 and point 4

else if (((cl-f1)*(cl-f4).le.0.) then

  contour line intersects edge
  between point 1 and point 4

  ix1(i,1)=ofsetx+scalex*(x1+(x4-x1)*(cl-f1)/(f4-f1))
  iy1(i,1)=ofsety+scaley*(y1+(y4-y1)*(cl-f1)/(f4-f1))

  check to see if contour line intersects edge
  between point 1 and point 3

  if (((cl-f1)*(cl-f3).le.0.) then

    contour line intersects edges
    between point 1 and point 4 and
    between point 1 and point 3

    ix2(i,1)=ofsetx+scalex*(x1+(x3-x1)*(cl-f1)/(f3-f1))
    iy2(i,1)=ofsety+scaley*(y1+(y3-y1)*(cl-f1)/(f3-f1))
    ind(i,1)=.true.

  end if

else

  set error flag

  flag(i)=.true.

endif

contour line did not intersect edges
between point 3 and point 4 and
between point 1 and point 4
check to see if contour line intersects edge
between point 1 and point 3

else if (((cl-f1)*(cl-f3).le.0.) then

  contour line only intersects edge
  between point 1 and point 2
  must be a problem
  set error flag

  flag(i)=.true.

endif

endif
denef
500 continue
C
   do 550 i=is,iel
C
      check to see if an error occurred
C
      if (flag(i)) then
C
         issue error message and stop
C
         write(*,*) 'There is something fishy about conplt!' stop
C
      endif
C
550 continue
C
   do 560 i=is,iel
C
      check to see if line is to be plotted
C
      if (ind(i,1)) then
C
         plot a single line
C
         call ufblin(ix1(i,1),iy1(i,1),ix2(i,1),iy2(i,1))
C
         check for more lines
C
         if (ind(i,2)) then
C
            plot two more lines
C
            call ufblin(ix1(i,2),iy1(i,2),ix2(i,2),iy2(i,2))
            call ufblin(ix1(i,3),iy1(i,3),ix2(i,3),iy2(i,3))
C
         endif
C
      endif
C
560 continue
C
400 continue
C
        increment contour level
C
        cl=cl+delf
C
        plot contours
C
        go to 150
C
end
C*********************************************************************
subroutine fminmax(f,fmin,fmax,ls,ie,js,je,m,n,iflag)
C
dimension f(m,n)
data eps/lel4/
check to reset minimums and maximums
if(iflag.eq.0) then
  fmin=eps
  fmax=-eps
end if
do 100 j=js,je
do 100 i=is,ie
  check to reset minimums and maximums
  fmax=amax1(f(i,j),fmax)
  fmin=amin1(f(i,j),fmin)
100 continue
return
end

C************************************************************************************************************
C subroutine getf(td, tu, tv, te, dd, m, n, ivar)
C dimension td(*), tu(*), tv(*), te(*), dd(*)
C common/fluid/gamma,gml,gpl,gmlg,gplg,ggml
C data cv/4390./

mn=m*n
crcv=l./cv
rggml=l./ggml
coe1=(gamma-1.0)*cv
coe2=gamma*(gamma-1.0)*cv

if(ivar.eq.1)then
  do 10 i=1,mn
     dd(i)=td(i)
 10 continue
  return
end if

if(ivar.eq.2)then
  do 20 i=1,mn
     rrho=l./td(i)
     ul=tl(i)*rrho
     vl=tv(i)*rrho
 20 continue
end if
dd(i)=(te(i)*rrho-0.5*(ul*ul+vl*vl))*rcv
continue
return
end if

if(ivar.eq.3)then
do 30  i=1,mn
  rrho=1./td(i)
  ul=tu(1)*rrho
  vl=tv(i)*rrho
  t1=(te(i)*rrho-0.5*(ul*ul+vl*vl))*rcv
  dd(i)=td(i)*gml*cv*t1
continue
return
end if

if(ivar.eq.4)then
  do 40  i=1,mn
    rrho=1./td(i)
    ul=tu(i)*rrho
    vl=tv(i)*rrho
    uv=ul*ul+vl*vl
    t1=(te(i)*rrho-0.5*uv)*rcv
    cl=abs(t1*gml*cv)
    dd(i)=sqrt(uv/cl)
  continue
end if

if(ivar.eq.5)then
  do 50  i=1,mn
    rrho=1./td(i)
    ul=tu(i)*rrho
    vl=tv(i)*rrho
    t1=(te(i)*rrho-0.5*(ul*ul+vl*vl))*rcv
    pi=td(i)*gml*cv*t1
    dd(i)=log(tl**rggml/pi)
  continue
end if

if(ivar.eq.6)then
  do 60  i=1,mn
    uv=tu(i)*tu(i)+tv(i)*tv(i)
    dd(i)=0.5*uv/td(i)
  continue
end if

if(ivar.eq.7)then
  do 70  i=1,mn
    rrho=1./td(i)
    ul=tu(i)*rrho
    vl=tv(i)*rrho
    t1=(te(i)*rrho-0.5*(ul*ul+vl*vl))*rcv
    pl=td(i)*gml*cv*t1
    dd(i)=(te(i)+pl)*rrho
  continue
end if
subroutine grldxy(x,y,m,n,is,ie,js,je,iflag)
C
dimension x(m,n),y(m,n)
common /maping/ scalex, ofsetx, scaley, ofsety, scaled, ofsetd
real scalex, ofsetx, scaley, ofsety, scaled, ofsetd
dimension ix(4),iy(4)
C
iflag=0 draw the boundaries only
iflag=1 draw constant-i lines only
iflag=2 draw constant-j lines only
iflag=3 draw the entire grid
C
set grid line look up table (lut) index (color)
call ufbcol(255)
c
compute do loop end and increment
ii=ie-1
jj=je-1
ii=1
jj=1
C
check to reset increments
if(iflag.eq.O.or.iflag.eq.1) jj=je-js
if(iflag.eq.O.or.iflag.eq.2) ii=ie-is
C
check to plot grid lines
if((iflag.ne.2).and.(ii.gt.O).and.(js.ne.je)) then
do 100 i=is,ie,ii
do 100 j=js,jj
       C
       convert world coordinates to screen coordinates
       ix(1)=scalex*x(i,j )+ofsetx
       ix(2)=scalex*x(i,j+1)+ofsetx
       iy(1)=scaley*y(i,j )+ofsety
       iy(2)=scaley*y(i,j+1)+ofsety
C
       plot grid line
       call ufblin(ix(1),iy(1),ix(2),iy(2))
100  continue
C
endif
check to plot grid lines

if((iflag.ne.1).and.(jj.gt.0).and.(is.ne.ie)) then
  do 200 j=js,je,jj
  do 200 i=is,iil
  convert world coordinates to screen coordinates
  ix(1)=scalex*x(i,j)+ofsetx
  ix(2)=scalex*x(i+1,j)+ofsetx
  iy(1)=scaley*y(i,j)+ofsety
  iy(2)=scaley*y(i+1,j)+ofsety
  plot grid line
  call ufblin(ix(1),iy(1),ix(2),iy(2))
  200 continue
endif
return
end

***************************************************************************

subroutine linmap(a1,a2,b1,b2,c1,c2)

compute scale factor and offset

if(a2.ne.a1) then
  c1=(b2-b1)/(a2-a1)
  c2=b1 - c1*a1 + .5
else
  c1=0.
  c2=(b1+b2)*0.5
end if
return
end

***************************************************************************

subroutine plotd(x,y,q,d,m,n,ivar,ls,Js,je,nc,sides,
*   id,idmin,idmax,lt,iflag)

dimension x(m,n),y(m,n),q(m,n,4),d(m,n)
common /extrem/ xmin, xmax, ymin, ymax, dmin, dmax
real xmin, xmax, ymin, ymax, dmin, dmax
common /maping/ scalex, ofsetx, scaley, ofsety, scaled, ofsetd
real scalex, ofsetx, scaley, ofsety, scaled, ofsetd
common/fluid/gamma,gml,gpl,gmlg,gplg,ggml
real idmin, idmax
save odmin, odmax

data gamma, odmin, odmax/1.4, +le30, -le30/

contour plot
written by: Eric L. Everton & Jamshid S. Abolhassani

gml = gamma - 1.
gpl = gamma + 1.
gmlg = gml / gamma
gplg = gpl / gamma
ggml = gamma * gml

ivar = 0 write out the grid
1 density
2 temperature
3 pressure
4 mach number
5 entropy
6 dynamic pressure
7 total enthalpy

------------------ scaling the grid & flow variables

if (it.eq.0) then
  compute x and y minimums and maximums
  call fminmax(x, xmin, xmax, is, ie, js, je, m, n, 0)
  call fminmax(y, ymin, ymax, is, ie, js, je, m, n, 0)

  compute aspect ratio and
  x and y ranges
  yoverx = 1024. / 1280.
  arange = xmax - xmin
  brange = ymax - ymin

  check for zero ranges
  if ((arange.eq.0.).or.(brange.eq.0.)) then
    a range is zero
    issue an error message and stop
    write(*,*) 'xmin - xmax or ymin - ymax eq 0'
    stop
  endif

  check aspect ratio with range ratio
  if (brange/arange.gt.yoverx) then
    adjust xmin and xmax
    recompute range
hafdif = .5 * (brange/yoverx - arange)
xmin = xmin - hafdif
xmax = xmax + hafdif
arange = xmax - xmin

else
adjust ymin and ymax
recompute range
hafdif = .5 * (arange*yoverx - brange)
ymin = ymin - hafdif
ymax = ymax + hafdif
brange = ymax - ymin
endif

add border
xmin = xmin - sides*arange
xmax = xmax + sides*arange
ymin = ymin - sides*brange
ymax = ymax + sides*brange

calculate scale and offset for x and y

call linmap(xmin, xmax, 0., 1279., scalex, offsetx)
call linmap(ymin, ymax, 0., 1023., scaley, offsety)

endif

generate d variable
call getf(q(1,1,1), q(1,1,2), q(1,1,3), q(1,1,4), d, m, n, ivar)

compute d minimum and maximum

call fminmax(d, cdmin, cdmax, is, ie, js, je, m, n, 0)
idflag = 1
if (cdmin.eq.cdmax) idflag = 0

calculate overall d minimum and maximum
if (cdmin.lt.odmin) odmin = cdmin
if (cdmax.gt.odmax) odmax = cdmax

check to use current d minimum and maximum
if (id.eq.0) then
set dmin and dmax to current

   dmin = cdmin
   dmax = cdmax
else
    set dmin and dmax to input
    dmin=idmin
dmax=idmax
endif
compute scale and offset for d
if(idflag.eq.1)
  call linmap(dmin,dmax,2.,255.,scaled,offsetd)
write current, overall & used d minimums and maximums
write(62,*) ' current dmin & dmax = ', cdmin, cdmax
write(62,*) ' overall dmin & dmax = ', odmin, odmax
write(62,*) ' used dmin & dmax = ', dmin, dmax
clear frame buffer
call ufbcle
write frame number
write(62,*)' Plotting frame number ',it
plot contours
if(idflag.eq.1)
  call conplt(x,y,d,m,n,is,ie,js,je,2.,255.,nc)
plot grid lines
call gridxy(x,y,m,n,is,ie,js,je,iflag)
send frame buffer to display
call ufbwri
return
end