Visualization of 3D scalar fields.  
Application to air pollution concentration fields  
S. Ruiz, L. Cremades  
ITEMA, Universitat Politècnica de Catalunya,  
Ctra. Nacional 150, km 14.5, Aptdo. 508, 08220 Terrassa, Spain  
Dpt. de Proyectos de Ingeniería, U.P.C.,  
Diagonal, 647, Planta 10, 08028 Barcelona, Spain  

ABSTRACT  
Air pollution distributions usually form finite volumes inside the 3D domain of simulation models. We have developed an application to calculate and visualize isosurfaces whose main restraint is that they must be closed (or at least almost closed) into the closed domain boundaries.  

This application uses a 3D regular-spaced data mesh which defines the domain. This 3D mesh should be formed by a series of surfaces which should not be interlaced. These surfaces can have any shape.  

The procedure for generating an isosurface has two steps: 1) the contour of the isosurface is computed in every plane. Every contour has to be closed. If open contours are created, they will be closed through the domain boundaries. 2) This contour is compared with the contours of the surfaces at the upper planes in order to assign contours to isosurfaces. So, the isosurface is generated as a whole series of curves in the space. To visualize it, we use the primitive B-spline surface of the standard graphics PHIGS.  

This procedure allows us to get a good visualization of air pollution volumes in a relatively short computing time. Some tests and application examples are presented.  

KEYWORDS  
Visualization, isosurface, isocontour, scalar field, air pollution  

INTRODUCTION  
3D Finite Element Method (FEM) and 3D Finite Differences Method (FDM) used in simulation models are becoming more and more some of the most useful tools to
engineers and scientist for studying and analyzing complex structures. The increasing computation power of the new workstations is allowing an increase in accuracy and resolution of the results obtained by FEM and FDM. Generally, after some variable time of computation, these results consist of a huge amount of values representing a scalar field in 3D.

The problem arises when one interprets these values. So far, the post-processors complementing the FEM and FDM's results usually force the users to imagine the volumetric distribution of the values through some 2D cuts in several planes. Visualizing this scalar field in 3D can be very helpful to engineers and scientists. To visualize an isosurface of stress values, for instance, can help the designers to optimize a part.

An isosurface is a surface comprising a volume where a variable has a certain constant value. Its visualization in 3D allows to detect, in a glance, some complex relations that otherwise would be very difficult to look at. Associated to the isosurface, there is the concept of isocontour. An isosurface is formed by an infinite number of isocontours, in particular horizontal isocontours.

We have developed a program to calculate and visualize isosurfaces in 3D, especially those which are closed into the domain boundaries of the scalar field or that can be closed through the boundaries. As a first application, the program has been applied to the visualization of tridimensional isosurfaces of air pollution concentration fields over topographic maps. These fields usually form finite volumes inside the 3D domain of the simulation models, then they are appropriate to be visualized with this program.

BACKGROUND

From the sixties, several researchers have focused on how to visualize scenes of the physical space through a computer output device, usually a CRT screen. Since then, several techniques have been developed to obtain this. We can divide them between those used to create 3D isocontours (wire models) and those used to generate 3D isosurfaces.

Among the first ones, we can mention, foremost, the method for tracing the isocontours directly (Sabin [1]). In this method, the isocontours are calculated from analytical equations of surfaces. It produces very good results, but can only be applied to singular cases.

Secondly, there is the method of interpolation in a grid. The isocontours are approximated by a series of straight segments which are interpolated bilinearly from the values at the mesh nodes (Sutcliffe [2]). An isocontour passes through one of the sides of a cell when one vertex of that side has a greater value than the isocontour and the other vertex has a lower value. This method can give rise to some ambiguities if, for instance, the case shown in Figure 1a happens (cell nodes marked with "+" are nodes with a greater value than the chosen contour value, whereas nodes marked with "-" have a lower value). Both solutions shown in Figure 1 might be valid.
Figure 1. Method of interpolation in a grid cell: a) ambiguities, b) how to solve them.

Figure 2. Marching cubes' algorithm. Ambiguities derived from the case A. Three possible interpretations (Collins [3]).
However, to choose the correct one, the value at the centre of the cell should be interpolated. In this case, we have two different solutions depending on the central value (Figure 1b).

The third method for tracing isocontours consists of following the contour point after point, from two initial points (Sabin [1]). This algorithm has two main drawbacks: how to find the initial values and how to determine the ending condition in the case of a closed contour. It is more accurate than the previous one, but is also more complex.

As to the methods for generating isosurfaces, we can classify them in three large groups: methods of serial sections, methods based on cubic grids and methods of studying the volume itself (volume rendering).

The method of serial sections is useful when data are arranged as a stack of slices, such as, for instance, the data obtained from a CT scan. The first step is to identify the contours inside each slice. To do that, the methods above mentioned for creating isocontours can be used. Once the contours have been identified, those in adjacent slices are joined by a ribbon of triangles or higher-order surface patches. The problem with contour lines is that they do not contain sufficient information regarding the gradients associated with the 3D isosurface they describe (Collins [3]). So, the number of possible combinations to create the isosurface as a series of triangles can be very important. Nevertheless, some authors, such as Keppel [4] or Fuchs, Kedem and Uselton [5] have tried to find the optimum triangulation. An alternative approach taken by Sunguroff and Greenberg [6] represented sections of each contour by uniform B-splines. However, all of the above algorithms have problems in the case of bifurcations, i.e. where one contour in one slice divides into two or more contours in an adjacent slice. Nowadays, automatic methods are being researched to generate the complete isosurface in every case. Other applications, like TROTS (Veen and Peachey [7]) or ISOSRF (Wright and Humbrecht [8]), do not use any ribbon of patches to join the isocontours, but are based on removing-hidden-lines algorithms.

The methods based on cubic grids try to approximate the object volume as a set of voxels (Collins [3]). This technique, called "cuberille" technique, is usually applied in medical imaging, where accurate depiction of the object is more important than generating pleasing images.

Another algorithm based on data sets placed in a 3D regular grid is the "marching cubes". This is probably the algorithm which has been most widely implemented for the generation of isosurfaces within regular grids. In the case of a cubic grid each cube is considered in turn and the vertices of the cube are classified as being either "inside" or "outside" an isosurface based on the vertex value compared with a chosen isosurface value (Lorensen and Cline [9]). There are 256 possible ways in which a surface may intersect a cube, which are reduced to 15 if symmetry-related cases are ignored. The isosurface is finally approximated by a triangular mesh. The original algorithm has ambiguous cases, such as shown in Figure 2 (black nodes are outside the isosurface, whereas white nodes are intersection points of the isosurface): there are up to three possible solutions to visualize the isosurface.
Several authors, such as Wyvill, McPheeters and Wyvill [10] or Wilhelms and van Gelder [11], proposed some schemes to solve these ambiguities. In any case, other similar algorithms, the "marching tetrahedra", based upon tetrahedra instead of cubes, seem to work better. These algorithms create a tetrahedral grid from a cubic grid by dividing each cube into 5, 6 or 24 tetrahedra. This resolves the marching cubes ambiguity problem.

Finally, a number of techniques, collectively called volume rendering, have been developed to visualize the 3D structure of the interior of volumes, since it can be difficult to do by viewing individual slices or just isosurfaces. These techniques generally involve tracing an imaginary ray of light through the volume and applying a function, based on the data values encountered in the volume, to generate an intensity and opacity. These are then used to produce pixel values in the final 2D image (Collins [3]). These techniques produce very good results at the expense of important computation time.

To choose one of these methods to create isosurfaces is very dependent on the kind of application required and the graphic system (hardware and software) in which the application will run.

ALGORITHMS IMPLEMENTED

The goal of the application developed is to visualize 3D isosurfaces of air pollutant concentrations as fast as possible (interactively) from data sets arranged in a 3D Cartesian grid over a 3D topographic map. This grid should be regular in X and Y directions, but not necessarily regular in the Z direction. Taking into account these circumstances, we have chosen the method of serial sections to create the air pollution isosurfaces from isocontours, which in turn are created by following a modified method of interpolation in the rectangular grid. Isocontours are generated for each Z-dependent layer of the air pollution data set and afterwards the isosurfaces are created by joining iso-valued contours of the different layers. As we said before, the method of interpolation in a mesh is one of the least accurate methods to generate isocontours, but one of the fastest. To increase its accuracy, we can increase the resolution of the input data.

In the original algorithm all the grid cells were sequentially analyzed and then the segments were individually drawn. In our algorithm we try to find an isocontour by following the curve directly. So, once we have a point of the curve, we look for the next one and so we generate the curve point after point. Once a curve is completed, we look for other curves, but avoiding to analyse cell sides where a curve already passes through (only one isocontour segment can pass through a cell side). A curve then becomes a set of arranged cell sides, identified by three integer numbers \((i,j,k)\), where \(k = 1\) or \(2\). If \(k = 1\), it is a horizontal side between the points \((i,j)\) and \((i,j+1)\); while if \(k = 2\), it is a vertical side between \((i,j)\) and \((i+1,j)\). In this way, the algorithm works with integer numbers until the visualization step, in which the points where the "real" curves pass through the cell sides, are interpolated as real numbers. To visualize the isocontours, we make use of the non-uniform B-spline curve primitive of the graphic programming interface graPHIGS [12]. The graPHIGS is
Figure 3. Two ways to close a contour through the domain boundaries (S stands for starting point of the contour and E for ending point).

Figure 4. Presence of a bubble of 20-arbitrary units value inside a 10-a.u. isosurface (3D view and sections A and B). The value of the scalar field outside this isosurface is 20 a.u.
Visualization and Intelligent Design

based on the ISO PHIGS (Programmer's Hierarchical Interactive Graphics System) standard graphics.

This algorithm can be summarized as follows:

```
action: "searching isocontours in a layer"
last = first
while (not endsearch and not origin) do
    search origin from last
    if origin do
        last = origin
    endif
    while (not endcurves and origin) do
        search next
    endwhile
    generate curve
endwhile
end_of_action
```

The application only deals with closed isocontours. That means that if a contour is open, it will be closed through the domain boundaries. As shown in Figure 3, there are two ways to close a contour through the domain boundaries. We have chosen the one allowing to close the contour through the shorter boundary pathway, in order to consider that most of the surface enclosed by that contour is inside the domain (b in Figure 3). However, the program is open for modifying this criterion easily. To close the contour, the boundaries are followed point after point counter-clockwise from the initial point (S in Figure 3) of the open contour to its last point (E in Figure 3). For each isocontour of a layer, the algorithm keeps the following information:

- number of points of the contour
- layer
- whether it is closed with boundary points
- size of the smaller rectangle enclosing the contour
- number of points enclosed by the contour
- map of the inner points
- value of the scalar field enclosed by the contour

At the end of this first step of the process, we have a set of layers with a series of closed contours defined in each layer. The next step is to create surfaces comprising volumes in which the scalar field has a given value, in our case, an air pollution concentration. An isosurface is created by relating an isocontour at a layer with contours of the same value at the upper layers (the algorithm works from bottom to top layer). The first condition to relate two contours is that their values at the inner scalar field should be equal. This overcomes the problem of the presence of bubbles (Figure 4). The second condition imposed by the algorithm is the need of continuity between isocontours at adjacent layers. Two isocontours at adjacent layers belong to the same isosurface if the intersection of the projected areas over a horizontal plane exceeds 70%. It is easy to modify this value or the criterion.

With this algorithm, an isosurface is by definition an arranged set of curves with indices according to increasing order of domain Z layers, with only one curve per
layer. So, what happens in the case of bifurcations? The solution adopted is to consider a new isosurface (or more, depending on the number of bifurcations) from the bifurcation point up, in which the first contour is the contour where the bifurcation begins. In the case of unions, i.e. two or more isosurfaces converging on only one isosurface at the upper layer, the solution is analogous, but the last contour of the different isosurfaces corresponds to the first common contour. The volume resulting from this solution does not lack continuity.

This algorithm can be summarized as follows:

For each contour do
   if contour not related with contours at lower layers then
      create a new isosurface
   else
      if contour related with one only contour
         add contour to the isosurface
      else
         find the smaller surface among the surfaces related
         add contour to the smaller surface
         add contour to the other surfaces and close them
      endif
   endif
   if contour has more than one related contour at the upper layer then
      create new surfaces with contour
   endif
endfor

To visualize the isosurfaces, we use the non-uniform B-spline surface primitive of graPHIGS [12]. This primitive has an \( n \times m \) matrix of real numbers as input data, where \( n \) and \( m \) are the number of control points for \( u \) and \( v \) directions, respectively. In our application, the \( n \) direction corresponds with the path of each contour in a layer, whereas the \( v \) direction corresponds with the \( Z \) direction. Then, \( n \) is equal to the maximum number of points of the isosurface contours and \( m \) the number of contours defining the isosurface. To apply this primitive, every contour should have its points arranged in the same way and its number of points should be equal to \( n \) (the algorithm inserts additional points by interpolation if necessary). For generating a surface interpolated by B-splines, the minimum number of data points in a direction must be greater than the surface order. The greater is the order, the smoother is the surface, but the longer is the process and more data are required. After some trials, we chose the order 2, since it guarantees continuity \( C^1 \) and requires as few as three data per direction.

To visualize the topography, we also make use of this primitive from regularly spaced topographic data (terrain digital model). Some surface properties, such as derived from ambient and directional lighting effects, have been given to the surfaces by means of graPHIG's attribute subroutines.

Additional details about the program developed can be found in Ruiz [13].
Photo 1. Test with a spheric field: a) isocontours; b) isosurface.
TESTS AND EXAMPLES

We have carried out some tests to evaluate the graphic abilities of the program to visualize isocontours and isosurfaces of scalar fields. Here are presented some of the pictures obtained for two of these tests. Pictures have been taken directly from a colour screen, but because of technical and economical reasons, are published in black and white. However, we think that their quality is enough to give a good idea about the visualization. The domain in both tests is a rectangular mesh of 40x40x15 grid points and the topography is flat. The first test deals with a scalar field where the isosurfaces are usually closed (spheres), such as for instance the field generated by an electrical charge. Photo 1a shows the isocontours for 10 concentric "spheres" of scalar values ranging from 0.9 to 794 arbitrary units (a.u.) in 88-a.u. increments. The interior of each "sphere" encloses a volume in which all the points have a smaller value than the isosurface value represented by the "sphere". It should be noted that the isocontours intersect the domain boundaries. Photo 1b shows the 27-a.u. isosurface wholly closed into the domain.

The other test deals with open surfaces. The scalar field has their values distributed in such a way that the isosurfaces consist of two parallel planes. We can see from Photo 2 how the planes are closed through the domain boundaries, transforming the isosurfaces into truncated pyramids. Actually, the values at the boundaries do not correspond to the isocontour value; only points inside the domain have the same value than that of the isocontour or isosurface.

![Photo 2. Test with open isocontours.](image-url)
Photo 3. Air pollution concentration field from a power plant stack: a) isocontours of 10, 1000, 4000 and 8000 µg/m³ SO₂; b) isosurface of 10 µg/m³ SO₂.
Photo 4. Air pollution concentration field from a power plant stack: a) isocontours of 1000 $\mu$g/m$^3$ SO$_2$; b) isosurface of 1000 $\mu$g/m$^3$ SO$_2$. 
Photo 5. Ozone concentration field over the Barcelona area. Isocontours of 53.9, 171 and 257 µg/m³ O₃: a) at 14:00 LST; b) at 15:00 LST (for a simulated summer day)
Photo 6. Ozone concentration field over the Barcelona area. Isosurfaces of 257 $\mu$g/m$^3$ O$_3$: a) at 14:00 LST; b) at 15:00 LST (for a simulated summer day)
Next, we present two application examples of the program by using air pollution concentration field results from dispersion models. The domain has 40x40x15 grid points as before. However, the Z layers are not horizontal but terrain-following coordinates. The topography map corresponds to the Barcelona geographical area which extends 39x39 km² (grid spacing is 1 km). The topographic data are given in meters above sea level.

In the first example, the cloud visualized in Photos 3 and 4 corresponds to the simulation results of a power plant plume after the first hour of operation by using the MATHEW/ADPIC model (Cremades et al. [14]). Air pollution values are expressed as µg/m³ of SO₂. Isocontours of 10, 1000, 4000 and 8000 µg/m³ SO₂ are visualized together in Photo 3a, from which 1000-µg/m³ isocontours are separately represented in Photo 4a. Isosurfaces of 10 and 1000 µg/m³ SO₂ are shown in Photos 3b and 4b. We can see from Photo 4b how two 1000-µg/m³ isosurfaces have been formed from their isocontours.

The second example corresponds to some results of a simulation made by MARS model for the same topographic area as before (Flassak and Wortmann [15]). Air pollution values are expressed as µg/m³ of ozone. 2D images of the isocontours corresponding to three ozone values (53.9, 171 and 257 µg/m³) for two consecutive hours (14:00 and 15:00 LST) appear in Photo 5. The 257-µg/m³ isosurface value is visualized in 3D in Photo 6 for these hours. At noon and in the afternoon on the Mediterranean coasts, a strong sea breeze develops, especially in summer. The surface concentration patterns of ozone between 13:00 and 15:00 LST show that high ozone concentrations are formed above the sea and are transported by the sea breeze into the Llobregat valley (located at the left-hand side in the pictures; Barcelona city is at the centre close to the shore). As a consequence surface ozone concentrations exceed the levels in Barcelona. This agrees with the visualization results obtained by the program.

CONCLUSIONS

The algorithms implemented (interpolation in a grid for tracing isocontours and serial sections for isosurfaces) are suitable for visualizing volumes closed into the domain. In the case of open volumes, the program closes them through the domain boundaries. These algorithms allow us to get a reasonable response time to work interactively. Visualizing an isosurface requires about 45 s on an IBM RISC/6000 mod. 550 workstation. This performance might be improved by using non-standard graphic libraries.

The program has been applied to visualize 3D air pollution concentration volumes, but is ready to represent any other variable from scalar fields.

The code was developed in FORTRAN. The executing program requires 13 Mb of disk space.
REFERENCES