SEGMENTATION OF UNSTRUCTURED DATASETS

BY SMITHA BHAT

A thesis submitted to the

Graduate School—New Brunswick

Rutgers, The State University of New Jersey

in partial fulfillment of the requirements

for the degree of

Master of Science

Graduate Program in Electrical Engineering

Written under the direction of

Professor D. Silver

and approved by

_____

_____

_____

_____

New Brunswick, New Jersey

January, 1996

ABSTRACT OF THE THESIS

Segmentation of Unstructured Datasets

by Smitha Bhat

Thesis Director: Professor D. Silver

Datasets generated by computer simulations and experiments in Computational Fluid Dynamics tend to be extremely large and complex. It is difficult to visualize these datasets using standard techniques like Volume Rendering and Ray Casting.

Object Segmentation provides a technique to extract and quantify regions of interest within these massive datasets. This thesis explores basic algorithms to extract coherent amorphous regions from two-dimensional and three-dimensional scalar unstructured grids. The techniques are applied to datasets from Computational Fluid Dynamics and from Finite Element Analysis.

Acknowledgements

I would like to thank my research advisor, Professor Deborah Silver, for her guidance and support. I would also like to thank Professor Herbert Freeman and Professor Todd Cook for their participation on my commitee. I would also like to extend a special thanks to the researchers and staff at the VIZLAB and CAIP for their assistance. This work was supported by the NASA Ames Research Center (NAG2-829).

Table of Contents

List of Figures

Chapter 1

Basic Overview

Large scale experiments, numerical simulations and remote observations generate massive datasets which need to be transformed into pictures and other graphic representations to facilitate interpretation. This transformation to visual, "scientific visualization" images enables the identification and quantification of regions of interest within massive datasets. The techniques of scientific visualization are used to manipulate data in fields ranging from weather forecasting to molecular biology.

Scientific Visualization techniques are generally applicable to datasets which represent continuum, i.e., values exist between sampled data points. All continua being visualized are *types* of *tensor fields* which have simple transformation properties under coordinate transformations.

The *rank* of a tensor field is used to classify the physical field which then determines the visualization method which can be used. A **scalar** field is of rank *zero* and has a magnitude (i.e. one data value). Common scalar fields include density, pressure, electrostatic voltage, temperature and humidity. The magnitude varies with the location in space. A **vector** field is of rank *one* and has a magnitude as well as a direction. It is an n-tuple where n is the dimension of the space i.e. two in 2D, three in 3D etc. Common vector fields include velocity, vorticity, acceleration and magnetic field strength. A gradient vector field may be derived from a scalar field by differentiation with respect to each coordinate dimension of the space. An ***nxn*** **matrix** is used to represent a tensor field of rank *two*. An example of a second-order tensor is the stress tensor (solid mechanics). Second-order tensors may be symmetric (e.g., stress, strain) or asymmetric (e.g., vorticity of a flow). Higher order tensor fields also occur in fields like solid and differential geometry.

The format of datasets can also be broadly classified into two categories, namely **structured** and **unstructured** datasets. *Structured datasets* contain data that has been obtained by sampling uniformly within the volume of the experiment. Some of the datasets are available as cubic or rectilinear data. Others have curvilinear data, as in the case of those of Computational Fluid Dynamics (CFD), which have non-linear transformations applied to them so as to fill the volume of the fluid or to wrap around an object (e.g., aircraft wings). Connectivity between points is implicit and is defined by their position; a more detailed explanation is given in Chapter 2.

Unstructured datasets contain data that has been sampled at irregular intervals. Connectivity between the data points is provided by grouping adjacent points into cells. Cells may be tetrahedra, hexahedra, prisms, pyramids etc; and they may be linear (straight edges, planar faces) or higher order (e.g., cubic edges with 2 interior points on each edge). Tetrahedral cells are particularly useful as they allow more accurate boundary fitting, can be built automatically, and are simple to work with. Unstructured grids are common in Finite-Element Analysis (FEA), Finite-Volume Analysis (FVA) and CFD experiments.

An important part of scientific visualization is to *quantify* and *extract* objects or structures of interest. This is common to almost all disciplines, as the crucial part of understanding the original simulation, experiment or observation is the study of the evolution of "objects" present. For example, studying the progression of a storm, the motion and change of the "ozone hole" etc. Once regions of interest have been extracted, their evolution can be tracked. Extracting regions is similar to the concept of segmentation in computer vision. Most of the previous work in "object segmentation" has been confined to 2D and 3D "regular" datasets.

In this thesis, we will explore *basic* algorithms to extract coherent amorphous regions from 3D scalar unstructured datasets. A combination of techniques from computer vision, image processing, computer graphics and computational geometry are used. Techniques for the parallelization of these routines are also investigated. This thesis will provide a set of tools to extract regions of interest within unstructured datasets organized as tetrahedral cells. The dataset will be transformed into distinct objects

which can be tracked over several datasets in the time domain. Each object represents a group of tetrahedra that meet certain threshold criteria. In the degenerate case, structured datasets can also be handled. These routines can be general, to include all types of unstructured datasets.

The validity of the algorithms are verified using standard structured grids as comparisons showing that the results match those generated by the standard object segmentation tools. Various test cases will be used to measure the complexity of the algorithm and techniques to improve the performance will be suggested in the final chapter.

The practical applications of this thesis are in weather forecasting, fluid dynamics and mathematical simulations. Results will be provided for all these cases to emphasize the generality and extensibility of the algorithm.

The thesis is organized as follows. In Chapter 2, we cover the basis of dataset formats. Scientific Visualization techniques are presented in Chapter 3. 3D Segmentation techniques for continuous data are described in Chapter 4. Chapter 5 discusses previous work done on unstructured grids. Unstructured grid segmentation and the data structure used for it are presented in Chapter 6. In Chapter 7, the algorithm and implementation are presented. Examples and figures are discussed in Chapter 8. In Chapter 9, we present our conclusion and suggestions for future work.

Chapter 2

Datasets

Simulations of different scientific phenomena in various disciplines results in 2D and 3D datasets which need to be comprehended and interpreted. Algorithms and methods are devised to transform these large datasets into pictures and other graphic representations that facilitate perusal, interpretation and understanding. For any experiment or simulation, three domains need to be considered:

- the *physical* domain, P, where the equations defining the physical phenomenon under study are used to compute the value of the defining variables on a discretized grid;
- the *computational* domain, C, which is often a uniform coordinate system. The equations of the physical process in hand are transformed to this domain for the purpose of numerical computation;
- the graphical domain , G, where in order to visualize the physical domain, the physical coordinates need to be transformed into appropriate pixel coordinates and pixel values.

Grids need to be generated during the transition between the physical and the computational domain. In [24], meshes are classified into the following types (Fig. 2.1):

1. **Cartesian** ($P(i,j,k) \Longrightarrow C(i,j,k)$) : All cells in this grid type are identical axis aligned cubes ("voxels"). The physical coordinate system is mapped directly to the computational coordinate system.
2. **Regular** ($P(i,j,k) \Longrightarrow C(i*dx, j*dy, k*dz)$) : Regular grids are composed of identical rectangular axis aligned prisms.

3. **Rectilinear** ($P(i,j,k) \Longrightarrow C(x[i],y[j],z[k])$) : Rectilinear grids consist of axis aligned prisms. The distance between the points along an axis may not be equal, which results in dissimilar sized cells.

4. **Curvilinear** ($P(i,j,k) \Longrightarrow C(x[i,j,k],y[i,j,k],z[i,j,k])$) : In this type, grids are non-axis aligned. The computational space contain rectilinear grids which are transformed in physical space to fill a volume or wrap around regions of interest (e.g. aircraft wings in CFD applications).

5. **Block Structured** ($P(i,j,k) \Longrightarrow C(x_b[i,j,k],y_b[i,j,k],z_b[i,j,k])$) : This type consists of several structured grids (blocks) fitted together to fill the volume of interest. This type is useful as although structured grids are easy to use, they handle a limited range of topologies.

6. **Unstructured** ($P(i,j,k) \Longrightarrow C(x[i],y[i],z[i])$) : Unlike the previous types, there is no connectivity implied by this list of points. Edge/face/cell connectivity must be supplied separately. Cells may be tetrahedral, hexahedral, prisms, pyramids etc. and they may be linear (straight edges, planar faces) or higher order (e.g. cubic edges with two interior points on each edge).

7. **Hybrid** : This type consists of both structured and unstructured grids, putting each where their fitting and computational strengths are most beneficial.

A representation of all these grids is shown in Figure 2.1. The grid (i.e the computational domain) represents a sampled, much smaller version of the real world. The transition between the physical domain and the computational domain must be done correctly so as to account for what has not been modeled in the computational domain. Therefore, boundary conditions and initial conditions are specified to handle the spatial approximation of the rest of the world.

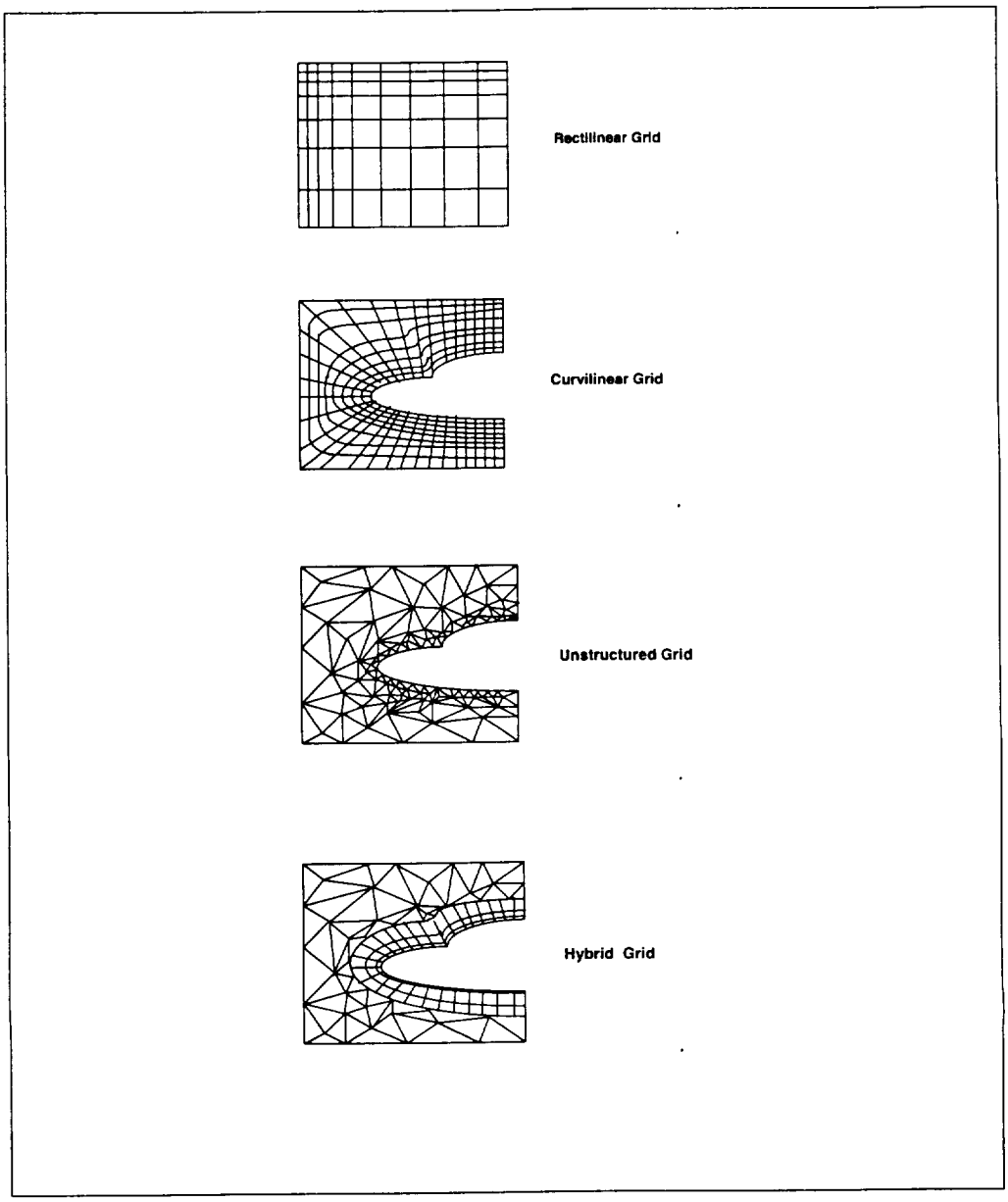


Figure 2.1: Mesh Types

Chapter 3

Scientific Visualization

Scientific Visualization is the application of computer graphics techniques to visualize scientific datasets. Scientific visualization methods have been extended to other fields like database visualization, computational geometry, information visualization etc. Simulations of different scientific phenomena in various disciplines result in 2D and 3D datasets which need to be comprehended and interpreted. Algorithms and methods are devised to transform these large datasets into pictures and other graphic representations that facilitate perusal, interpretation and understanding. This is the goal of scientific visualization.

Techniques for visualization of 3D scalar datasets may be broadly classified into two categories[3]: **direct volume rendering** and **surface fitting** algorithms.

Surface fitting algorithms are also called isosurfacing. An *isosurface* may be defined as a surface of constant threshold comprising of all those points $\mathbf{p}_i = (x_i, y_i, z_i)$ where the data value$(\mathbf{p}_i) - k = 0$ for some particular threshold value k. In surface fitting algorithms, a threshold is initially selected. Cells which have all corner values either above the threshold or below the threshold are eliminated. Primitives such as polygons or patches are then fitted onto the detected surface (constant-valued contour surface) and rendered. Common surface fitting algorithms include *marching cubes* [13], *dividing cubes* [2], *marching tetrahedra* [21] etc.

The *marching cubes* algorithm, a very popular isosurfacing technique [13], processes the volume cell by cell. Each cell is classified by comparing the eight values at the cell vertices with the specified threshold value. Cells which do not have the surface passing through them i.e. the values are not both above and below the threshold value are not considered. This classification results in a binary encoding which provides an index

into a precalculated table with the list of edges the surface passes through. Using the values at each edge vertex, the surface edge intersections are calculated using linear interpolation. The generated polygons are then rendered using standard tools.

The marching cubes method sometimes creates ambiguous conditions [16] by connecting the wrong set of points while generating triangles. One method of reducing these ambiguous conditions is to use the *marching tetrahedra* algorithm [21]. Here, each cell of the volume is divided into 5, 6 or 24 tetrahedra. The tetrahedra are then classified according to their projected profile with respect to a view point. Table edge intersection tests are done with the tetrahedra. As a tetrahedron has 6 edges, one or two triangles are sufficient to show the isosurface within it. The triangles are then rendered.

The *dividing cubes* [2] algorithm also processes the volume cell by cell, but without generating any intermediate surface primitives. When a cell is encountered with corner values both above and below the threshold value, it is projected to the screen space to find out if it projects into an area larger than a pixel. If it does not, the cell is rendered as a "surface point". Else, it is divided into subcells whose data values are calculated by interpolating the dividing cube's vertex values. The surface point for the subcells is considered to be at the center. The subcells are then rendered as a surface point onto the image plane where the computed intensity is assigned to the appropriate pixel.

Direct volume rendering algorithms include V-buffer [26], splatting [28], ray-tracing [11] etc. where in order to obtain the characteristics from the interior of a 3D dataset, volumetric primitives are used. Volumetric techniques can present a complete view of a 3D dataset without reducing its dimensionality. In these methods, the intermediate surface model is omitted, i.e. the elements can be directly mapped into screen space. The color and opacity of structures close to the view point are accumulated while those of structures further away are modulated. All the scene voxels contribute to the final image by means of color and opacity composition.

The *V-buffer* [26] is a cell-based method where all the pixels onto which a cell projects are processed before moving to the next cell. The cell are processed in a front-to-back order with respect to the view point. In each plane, the cells are processed beginning

from the closest one extending to those adjoining this cell based on their distance from the view point resulting in a concentric sweep about the initial cell. A bounding box for each cell is determined which is then clipped to each scan line to determine the pixels the cell projects on. The values at the four corners of the pixel are integrated from front to back and then averaged to calculate the current pixel color and opacity. Once the processing is completed, the next predetermined cell is handled.

Ray Tracing is a method where imaginary rays are sent from the view point through the pixels in the view plane, into the data volume. At each pixel, the ray is tracked through the volume computing the accumulated opacity and color. The ray is traced in the data volume until either the accumulated opacity reaches unity or the volume is exhausted. There are many different approaches to the ray tracing algorithm as given in [19, 26, 12]. Several methods also deal with acceleration techniques for the ray tracer.

Another technique for direct volume rendering is the *splatting* method [28]. The data volume is traversed in the front-to-back order. Each voxel is transformed into screen space and it's contribution to the image is calculated and composited based on lookup tables. The basic idea is that the energy of each voxel is spread around and therefore multiple pixels are affected by it. A *reconstruction kernel* is used to find the extent of the pixels. The projection of the kernel onto the image buffer is called a *footprint*. The image is completed when all the voxels have been "splatted" on the image buffer.

Chapter 4

Object Segmentation

Most of the available standard visualization tools provide techniques to render datasets but do not provide a mechanism to quantify the numerous *coherent* regions of interest within the dataset. A *coherent* region is a term given to a structure that persists for a *significant* amount of time, i.e. can be seen in datasets over a time series.

The topologies of coherent features are similar across various scientific domains. For weather data, this could include cloud patterns, storm and weather fronts across the globe etc. In fluid dynamics, coherent features include migration of enhancements and depletions of density (e.g bubbles or holes); finger like breakup or rollup of surfaces; stretching, entanglement and reconnection of vorticity or magnetic field lines (e.g. vortex structures associated with maxima events in Navier Stokes turbulence simulations).

In order to understand the results of an experiment, the intrinsic features of the data are looked for and extracted. These are then mapped into appropriate geometric models and displayed. This involves:

- Preprocessing the data
- Segmentation and feature extraction
- Surface generation and display

It may be necessary to filter or preprocess the data to reduce noise or subsample it to reduce the size of the dataset.

Many basic algorithms have been developed earlier [23] for feature identification and extraction. These algorithms help not only in visualizing and understanding large scientific datasets but also help in reducing visual clutter. Features or objects observed

in the dataset may be tracked in time providing insights into the evolving dynamics of the field. As quantification is also an essential step to understand the underlying physics, the volume, surface area etc. of objects may be measured which can then be used for comparison with objects from different datasets. We call these methods "object segmentation" and they are described below.

4.1 Segmentation

The concept of segmentation is typical in computer vision and 2D image processing. In computer vision, the goal of segmentation is to partition the entire image into quasi-disjoint regions; a region may correspond to a whole object or to a meaningful part of one.

In primitive region growing techniques [1], only aggregates of properties of local group of pixels are used to determine regions. A threshold value indicates whether a pixel is part of an object or not. The concept of neighborhood is not included in this definition. More sophisticated techniques incorporate the concept of connectivity; i.e., pixels are likely to be part of the same distinct region if they are connected and above a threshold value. Connectivity may be defined in the following way:

1. x_i in a region $\mathbf{R}$ is connected to x_j, iff there is a sequence of points, $\{x_i, .., x_j\}$ such that x_k and x_{k+1} are connected and all the nodes are in $\mathbf{R}$.

2. $\mathbf{R}$ is a connected region if the set of points $x \in \mathbf{R}$ has the property that every pair of nodes is connected.

3. $R_i \cap R_j = \phi, i \neq j$

The entire image is given by $\cap_{k=1}^{m} R_k$ where the set of regions, R_k, satisfying the last two properties is known as a partition. Each region is generally a unique, homogenous set defined as

$H(R_k) = true$ for all k and

$H(R_i \cap R_j) = false$ for $i \neq j$.

For a 2D uniformly sampled dataset, connectivity can be defined as 4-connected (pixels along the axes) or 8-connected(including diagonals), etc. In the case of a 3D regular dataset, pixels become voxels including a connectivity in the z direction. Objects could now be 6-connected (points along the axes), 18-connected(including the diagonals along the axes), or 26-connected (including the long diagonals of neighboring cubes).

Segmentation Techniques

Some of the segmentation techniques in computer vision include *blob coloring* and *region growing.*

The *blob coloring* algorithm assigns each "blob" like region a different color. In this 2D technique, an L-shaped template is used to scan the image from left to right and top to bottom, assigning a color to every non-zero node based on the color of the upper and left neighbors. Each connected region is distinguished with a different color and the total number of different connected regions is determined.

The *region growing* algorithm is a more complex algorithm in which there is a basic assumption of an object-background image. A histogram of gray levels is used to select the threshold level that divides the image pixels into either object or background. This process can be done recursively for each region to further segment the image. Now,

$H(R) = true$ if the points in R pass the histogram test, and

$H(R) = false$ otherwise.

If the points in R do not pass the histogram test, R should be split into subregions. Similarly, some regions may need to be merged.

4.2 Object Segmentation and Feature Extraction

Standard isosurface algorithms provide bounding surfaces and edges but do not provide any connectivity information or physical parameters such as volume, mass, centroid etc. of the different distinct regions. These kind of statistics are required for tracking

simulations over the time domain and for understanding the underlying model.

In [23], the above methods of segmentation were applied to scientific visualization datasets. Features in a dataset may be defined as *isovalued* clusters of points above a certain specified threshold. Features in a dataset can be identified by initially selecting points that meet the threshold criteria. This is done to segment the dataset for each threshold value. A *seed growing* method can be used to isolate features in the dataset [23, 15, 14, 25] with some modification similar to *region growing*. The maxima of the dataset are used as initial *seed* points. Points around the maxima are then grouped into regions based on neighboring information. As the threshold is reduced from the maxima of the dataset to the specified threshold, regions around the seed points grow and new seeds for further region growing arise. A tabulation of current active regions is maintained for each iteration. However, merging of different objects must be considered as there could be several seed points and a neighboring point could be a part of another region. To facilitate efficient determination of adjacency and membership of neighboring points as well as object intersection and merging, a spatial enumeration data structure (an octree) was used [8, 20].

The algorithm can be stated[22] as follows:

```
1. Eliminate all points below thresh_min.
2. Create first object using the global maximum.
3. For every threshold-value, from thresh_min to thresh_max
   do the following
3.1     For every existing object, do the following
3.1.1     For each point on the boundary of the object,
          do the following
3.1.1.1       For each neighboring point (the neighbor is searched
              at a distance of 1-grid point for a 6-point adjacency),
              do the following
              {
3.1.1.1.1       If the point is over the threshold and does
                not belong to current object, add the point
                to the current object
3.1.1.1.2       For all the remaining objects, do the following
                {
                  If this point belongs to the object, merge the
                  current object with the object
                }
3.1.1.1.3       Mark the point as boundary or non-boundary (if
                all the six neighbors of the point are in the
                object, the point is not a boundary)
3.1.1.1.4       Recursively repeat step 3.1.1.1 for the newly
                added point
              }
3.2           Look for new non-used regional maxima points (using
              octree for efficiency)
3.3           Create a new object with the point as a seed and
              repeat step 3.1
```

4. Use Marching Cubes algorithm to generate bounding surfaces (only boundary nodes need to be considered)

(For more details see [8])

Once the objects have been identified, they need to be quantified. The following parameters are computed for each of the distinct objects found:

- **Volume:** This is represented by the number of vertices in an object.
- **Relationship between scalar and vector of a vertex:**

$$S_i = \sqrt{\vec{V}_{xi}^2 + \vec{V}_{yi}^2 + \vec{V}_{zi}^2}$$

where S is the scalar value and V is the direction vector.

- **Mass: (0th order moment)** Mass is calculated as the sum of all the scalars in an object.

$$E = \sum_{i=0}^{n} S_i$$

where n is the total number of vertices with $S_i \geq threshold$.

- **Centroid: (1st order moment)**

$$\begin{aligned} C_x &= \frac{1}{E}\sum_{i=0}^{n} S_i * x_i \\ C_y &= \frac{1}{E}\sum_{i=0}^{n} S_i * y_i \\ C_z &= \frac{1}{E}\sum_{i=0}^{n} S_i * z_i \end{aligned}$$

- **Moment: (2nd order moment, or tensor matrix)**

$$\begin{pmatrix} I_{xx} & I_{xy} & I_{zx} \\ I_{xy} & I_{yy} & I_{yz} \\ I_{zx} & I_{yz} & I_{zz} \end{pmatrix}$$

$$
\begin{aligned}
I_{xx} &= \frac{1}{E}\sum_{i=0}^{n} S_i * (x_i - C_x)^2 \\
I_{yy} &= \frac{1}{E}\sum_{i=0}^{n} S_i * (y_i - C_y)^2 \\
I_{zz} &= \frac{1}{E}\sum_{i=0}^{n} S_i * (z_i - C_z)^2 \\
I_{yz} &= \frac{1}{E}\sum_{i=0}^{n} S_i * (y_i - C_y)(z_i - C_z) \\
I_{xy} &= \frac{1}{E}\sum_{i=0}^{n} S_i * (x_i - C_x)(y_i - C_y) \\
I_{zx} &= \frac{1}{E}\sum_{i=0}^{n} S_i * (z_i - C_z)(x_i - C_x)
\end{aligned}
$$

- **Vector sum:** The sum of all the vectors in an object is given by:

$$
\begin{aligned}
\vec{V}_x &= \sum_{i=0}^{n} \vec{V}_{xi} \\
\vec{V}_y &= \sum_{i=0}^{n} \vec{V}_{yi} \\
\vec{V}_z &= \sum_{i=0}^{n} \vec{V}_{zi}
\end{aligned}
$$

- **Unit of Vector-sum:** Unit vector of vector sum:

$$
\begin{aligned}
|V| &= \sqrt{\vec{V}_x^2 + \vec{V}_y^2 + \vec{V}_z^2} \\
\hat{V}_x &= \vec{V}_x/|V| \\
\hat{V}_y &= \vec{V}_y/|V| \\
\hat{V}_z &= \vec{V}_z/|V|
\end{aligned}
$$

- **Local maximum set:** The local maxima set is the set of vertices which are the "local maxima" for each of the distinct objects

The bounding surfaces are then calculated for each of the objects in a manner similar to marching cubes.

This type of segmentation has been used in CFD to search for the dominant vortex structures in isotropic 3D turbulence. In Fig. 4.1, the object segmentation technique is used to extract vortex structures in a turbulence dataset (see Appendix A).

4.3 Object Segmentation of Curvilinear Grids

A curvilinear grid is a 3-dimensional rectilinear grid in computational space which is "warped" or subjected to non-linear transformations in physical space so as to wrap around a region of interest. For a dataset with this type of a grid, the object segmentation technique must include the periodicity of the dataset. If an object happens to extend along a boundary, it would be split into two objects if periodicity is not enforced. The object segmentation algorithm described in Section 4.2 remains the same with only a change in the definition of the "neighboring" points. Points on the boundary of the dataset also have neighbors. These neighboring points are on the opposite face of the dataset. This means that the points on the opposite edge are treated as neighbors of the edge points in all three directions i.e., x,y,z. This is illustrated in Fig. 4.2. After the polygons have been generated as described in step 4 of given in Section 4.2, the polygons are remapped into physical space using a transformation matrix. The flow of the algorithm would now be as shown below:

```
Process inputs
    ↓
Feature isolation
    ↓
Generate bounding surfaces
    ↓
Transform triangles
    ↓
Display
```

For a "tokamak" dataset (see Appendix A), the following transformations were used to map a coordinate (x, y, z) to the physical space of the toroid (x', y', z') :

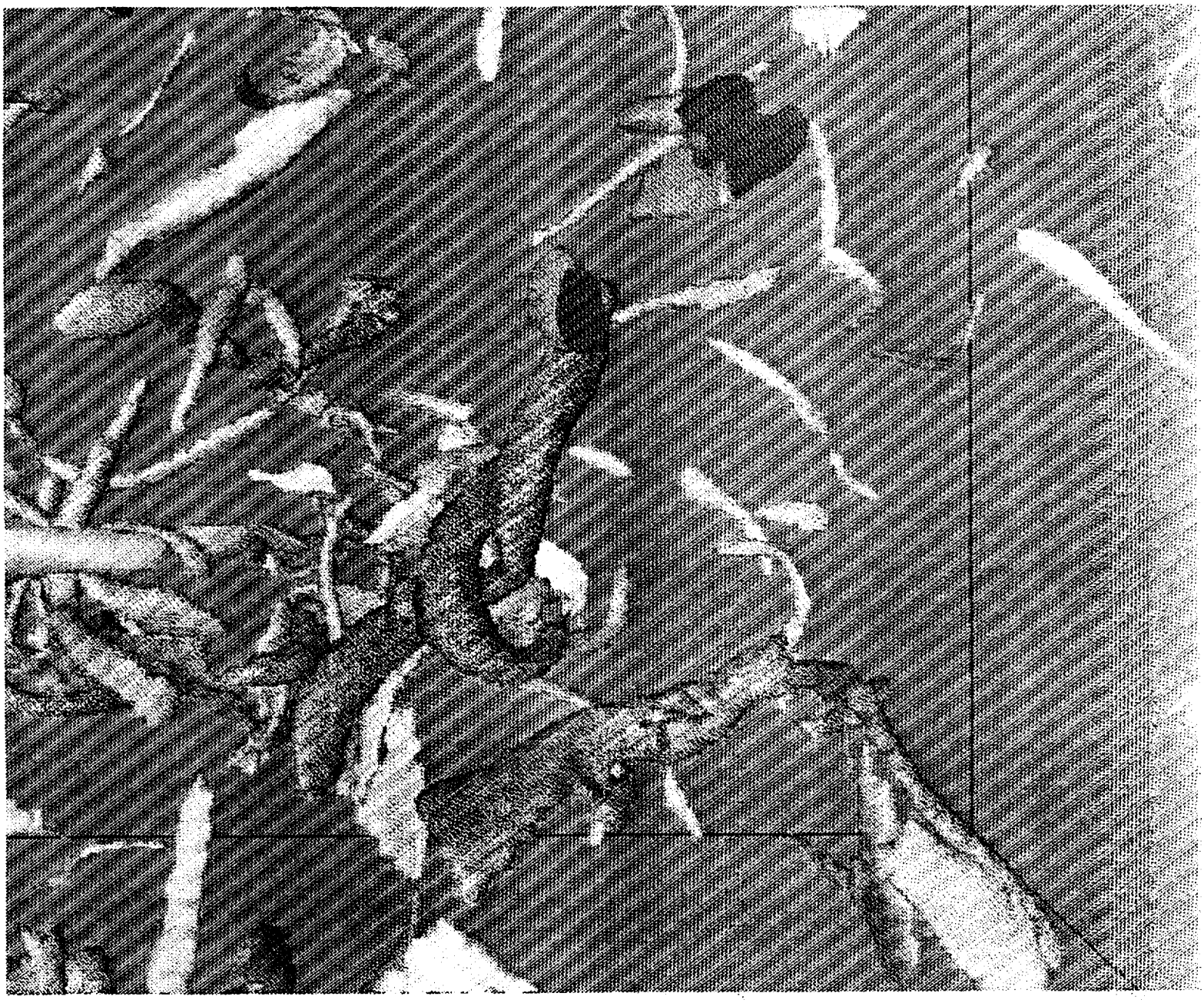


Figure 4.1: Object Segmentation of a Regular dataset

$$x' = (R_0 + r\cos\theta) * \cos\phi$$

$$y' = (R_0 + r\cos\theta) * \sin\phi$$

$$z' = r\sin\theta$$

where

$$R_0 = 600$$

$$r_0 = 300$$

$$r = (x + 0.5)/(nx - 1) * r_0$$

$$\theta = ((y/(ny - 1)) - 0.5) * 2\pi$$

$$\phi = ((z/(nz - 1)) - 0.5) * 2\pi$$

and

$$nx = 32$$

$$ny = 129$$

$$nz = 65$$

In Fig. 4.3, the feature extraction technique for a tokamak dataset is shown.

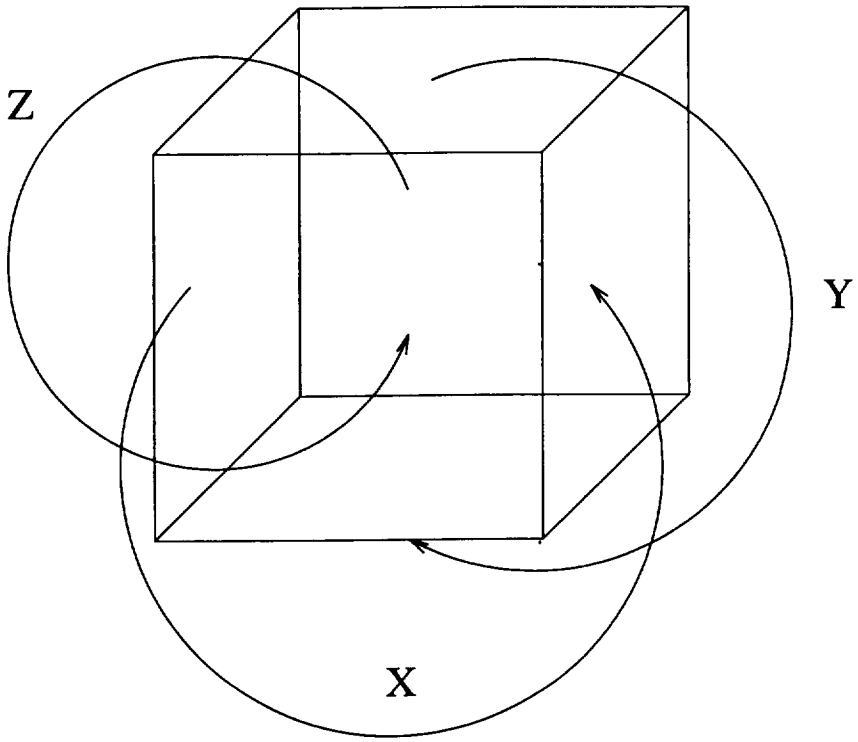


Figure 4.2: Illustration of Periodicity

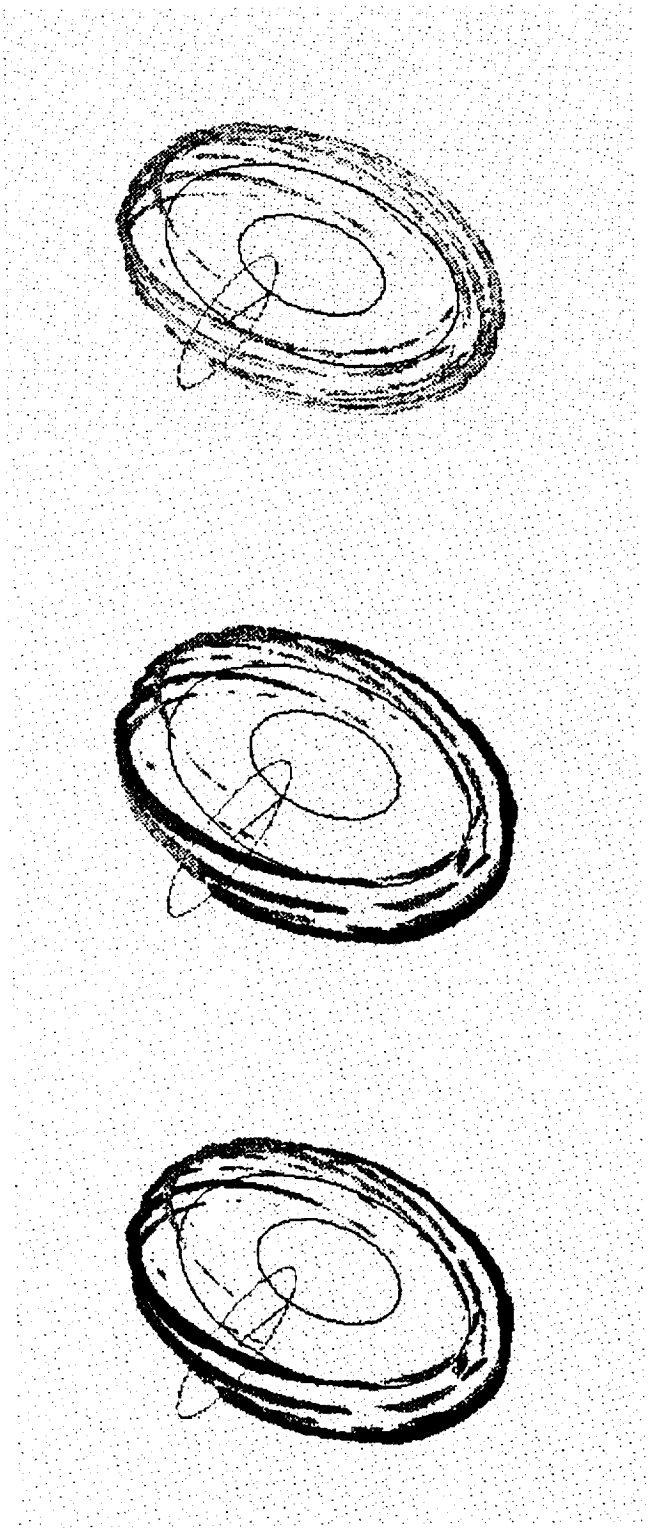


Figure 4.3: A curvilinear tokamak dataset (dimensions 32 X 129 X 65). The top figure is the isosurfaced dataset. The second is an object segmented figure without taking into account periodicity. Periodicity has been taken handled in the third image.

Figure 4.4: To illustrate periodicity, the same object has been isolated from the object segmented figures (periodicity has not been handled in the upper object, while periodicity has been considered in the lower object

Chapter 5

Related Efforts on Unstructured Grids

Unstructured grids are generally huge and complex datasets which are quite difficult to render. These grids are common since in many scientific simulations, measurements are made at discrete scattered locations. For example, temperature measurements taken at various locations of a mine, mineral concentration measurements taken at various depths at randomly located well sites etc. Furthermore, the resultant data of the finite element analysis (a method used frequently in the numerical solution of continuum problems) consists of coarse, unstructured volume elements with floating point values computed at the vertices.

The most obvious method to visualize an unstructured dataset is resampling; i.e., converting it into a rectilinear grid and using the standard visualization techniques mentioned earlier. The sampling could involve low-order or complex interpolation functions[5]. However, sampling causes a loss of data and accuracy. This is because an unstructured grid consists of cells which differ in size. For example, in flows around planes, as shown in Fig 2.1, cells located away from the surface of the object are thousands of times larger than those located at the surface. Resampling with the resolution of the larger cells would result in a great loss of precision, whereas resampling with the resolution of the smaller cells would result in a grid too huge to handle.

In recent years, various techniques have been implemented to represent and visualize data where the spatial x, y and z coordinates are not on a regular grid.

A technique called *span filtering*[7] has been used for displaying 3D isovalues of scalar fields within a solid finite element model. In this method, an initial traversal of the data elements with scalar values at their vertices yields a compact classification of the model by data values and ranges. Such a compressed data structure is useful in

providing a smaller list of elements which include a given isovalue. The initial traversal of data elements consists of the following steps. The range of data values is first divided into sub-ranges or "buckets". Elements are then grouped by the value of their lower bound bucket (i.e. the lower bound of their vertex values) and their span length (i.e. number of buckets spanned by the range of the values of the vertices). The groups are then grouped again by their lower bound bucket within a larger group grouped by the number of buckets this group spans. These classifications result in discrete ID lists which could then be scanned easily to obtain regions of interest.

Another method of visualizing datasets generated by the finite element analysis makes use of the fact that in finite element methods, the data is defined by the element's interpolation function[10]. In this method, the finite elements are reconstructed as *linear tetrahedral elements* (LTE) which have a simple interpolation function. LTEs are used since in order to calculate the data value at any point in a finite element grid, cartesian global coordinates have to be transformed into element local ones to locate the element within which the point lies. This involves non-linear simultaneous equations. In LTEs, the evaluation can be performed in global coordinates as the boundary surfaces are planar. Iso-valued surfaces are then calculated by either extracting triangular elements as iso-valued surfaces or by rendering the LTEs directly.

Unstructured grids can be volume rendered by using *ray casting*. However, this method is not well supported in workstations because of the extensive number crunching involved. The process can be accelerated by exploiting the coherency of the grid. In the "Double Z Item" buffer scheme[27], a unique identification number is assigned to each grid cell face. Two memory arrays of the same size are used. One memory array is the usual Z buffer and the other memory array is an *image buffer*. This is used to store id numbers of cell faces behind the ones that set the previous Z values on the previous pass through the data. For each layer of cell faces encountered, the identification values are decoded, data values are accessed and the shading contribution is composited into a separate final *image buffer*.

In [29], a projection method is used to volume render unstructured grids based on the *Projected Tetrahedra* algorithm given in [21]. As in any projection algorithm, the

cells of the dataset need to be ordered along the view direction. Visibility ordering of cells in a rectilinear grid is straightforward. However, it is not trivial for unstructured grids. The MPVO or *Meshed Polyhedra Visibility Ordering* algorithm[30] is a method for visibility ordering of cells of any acyclic convex set of meshed convex polyhedra. Here, an adjacency graph for the cells in the grid is constructed. For a specific view point, the visibility ordering is constructed by assigning a direction to each edge in the graph and then performing a topological sort on the graph. In order to handle non-convex grids, either modifications to the MPVO algorithm have been used or the grids have been preprocessed into convex grids. The tetrahedra in the dataset are then projected onto the view plane. Depending upon the view point and the orientation of the tetrahedra, 4 different projections are possible. Each projected tetrahedra is then decomposed into one to four triangles which are rendered. To speed up the original *Projected Tetrahedra* algorithm, methods have been provided[29] to approximate the opacity and color at the vertices of the projected tetrahedra.

A hybrid method of visualizing unstructured grids which are *sparse* (i.e. the mesh may consist of several smaller submeshes with large empty areas between them) is described in [9]. This kind of data is generally found in geological applications like reservoir simulations. A plane is considered perpendicular to a line in the image, incrementally. A "scan-plane buffer" is filled with temporary information about all the elements sliced by the plane. Each slice is broken into triangles and then into a set of line segments by linear interpolation. The visual parameters and a pointer pointing to the next expected segment are stored in the foremost scan-plane buffer location. All the segments are chained together by run length code. Before drawing the scan line, the final pixel value is determined by traversing the chain of segments in the scan-plane buffer, thus eliminating extensive neighbor searching operations.

In order to visualize spherical data, a representation called the "Sphere Quadtree" is described in [4]. Spherical data, generally obtained from earth and space sciences, has an inherent *spherical* distribution. Therefore, mapping of this kind of data onto a *flat* file system would introduce undesirable artifacts like "tears". The sphere quadtree helps in eliminating introduction of "tears" and also helps in representing the adjacency

relationship between the data components. It also helps in focussing quickly at the regions of interest in the data. The entire region (i.e. a sphere) is described as a collection of piecewise simple regions, namely the sphere is modelled with a convex polyhedron (an icosahedron whose sides comprise of 20 equilateral triangles) so that each face of the polyhedron is projected onto the surface of the sphere. The faces of the icosahedron are successively subdivided to approximate the sphere. There are further subdivisions depending upon tests administered to check how well the linear surface matches the data surface. As there is a natural correspondence between the nodes of the sphere quadtree and the trixel (area within a spherical triangle used to tile the sphere), adjacent and neighboring regions are easily determined.

For efficient visualization, datasets with vector data need different techniques suitable for particular applications. Vector visualization techniques may be classified based on two formulations namely the Eulerian formulation (where physical quantities are measured at discrete positions, e.g. at the nodes of the grid) and the Lagrangian formulation (where moving particles in the field are examined). Particle tracing can be directly on unstructured grids[6]. Here, numerical integration which is done to find successive positions of the moving particles can be done directly in the computational space of the individual cells of the grid.

All the above methods provide techniques to visualize unstructured grids. The following chapters will describe a method to segment unstructured grids to obtain regions of interest.

Chapter 6

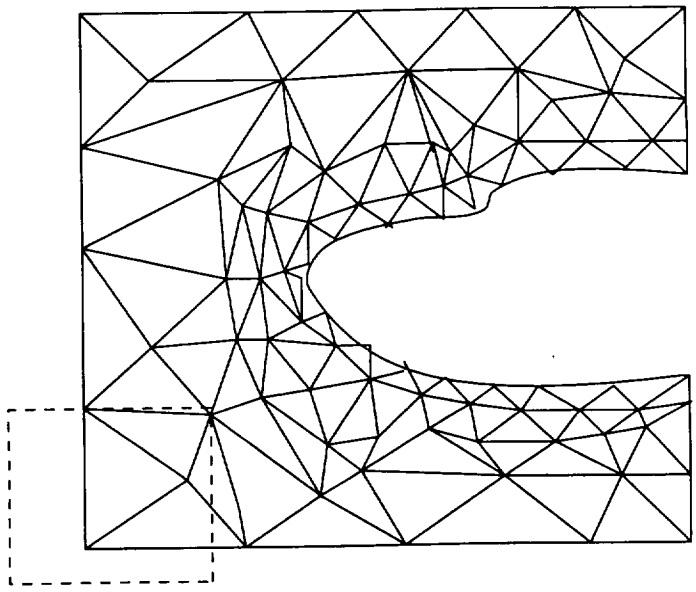
Unstructured Grids and Data Structures

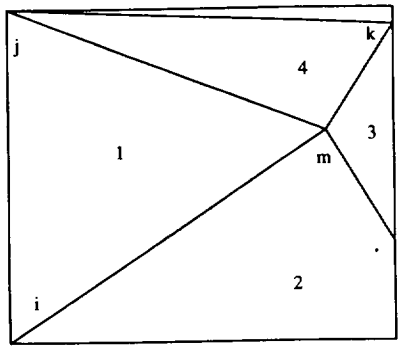
6.1 Object Segmentation of Unstructured Grids

As mentioned earlier, various techniques have been implemented to visualize unstructured datasets. In this thesis, the principles of *object segmentation* have been extended to unstructured datasets. Unstructured volumes have no implicit connectivity. The cells of the grid could be hexahedra, tetrahedra, pyramids, prisms etc. However, a volume of tetrahedral elements is popular as data in a tetrahedral cell is linearly distributed and every face of the cell is planar. The object segmentation technique has been used here on unstructured scalar volumes with tetrahedral cells. The tetrahedral model has three components:

- A nodal **data** component which gives the scalar value for each node
- A nodal **position** component which gives the x, y, z positions of each of the nodes
- A cell **topology** component which contains the identifiers of the four nodal data components

To extract features in the dataset, values that meet the threshold criteria are first selected. A seed growing technique is then used to locate the features in the dataset. i.e. regions are grown around seeds which are initially the extrema values in the dataset. The position of the neighboring values is not implicit as in the case of a regular dataset. For a given point, its neighboring points are the other three vertices of the tetrahedron it is part of. Therefore, it is necessary to keep track of all the tetrahedra this point is part of.





For a given point, m :

m(x, y, z) --> represents the position component of m

S(m) --> represents the data component of m

For triangles 1 and 4, the topology is described as follows:

T(1) = (i, j, m)

T(4) = (j, k, m) etc.

Figure 6.1: An Unstructured Grid

To avoid scanning the dataset each time for information about connectivity, a list of tetrahedra that have a given point in common is maintained as part of the data structure. All the points associated with the tetrahedra that form the region are flagged as "used". A new set of seed points is selected from the maxima of the remainder of the points. This algorithm is executed on the dataset iteratively until there are no more "active" points. At the end of each iteration, different regions or *objects* are merged if needed as there could be several seed points and any given point could be part of multiple regions. The final result of the algorithm is a set of distinct objects each represented as a list of tetrahedra. A *marching tetrahedra* algorithm is then applied directly to each object to obtain the bounding surfaces. These surfaces are represented as triangular polygons which can be directly rendered by standard graphics toolkits.

Once the objects have been identified, they need to be quantified. Parameters like *Volume*, *Mass*, *Centroid* etc. as described earlier are computed for each of the distinct objects found.

6.2 Data Structures for Object Segmentation

This section describes a set of compact data structures to store cell topology (points and their ordering) in an unstructured grid. In addition, the mathematical basis, representation and implementation will also be discussed.

6.2.1 Mathematical Basis

The cell structure is based on the topological construct called a Cell C_i. A tetrahedral cell C_i is an ordered sequence of points,

$$C_i = \{p_0, p_1, p_2, p_3\} \tag{6.1}$$

with $p \in P$ where P is the set of all points within the dataset. In the case of tetrahedral cells, the number of points defining a cell is four.

Two Cells C_i and C_j are *adjacent* to each other if at least one vertex in the two cells is the same, i.e., $p_i \in C_i$ and $p_i \in C_j$.

Another set, $U(p_i)$ represents the set of tetrahedral cells which have the point p_i as one of the vertices.

Hence,

$$U(p_i) = \{C_j : p_i \in C_j\} \tag{6.2}$$

A region R is defined as a *connected* region if the set of cells in R has the property that every pair of cells are adjacent. Each region is referred to as an *object.*

6.3 Representation

Memory requirement and simplicity have been the key considerations for the design of the data structures used to implement the segmentation algorithm. The dataset is read into a structure that is composed of four lists. The elements in each list can be addressed by an index into the respective arrays (Fig. 6.2) as described below.

6.3.1 The Data-List

This is an array of data values at each point of the dataset. Each point can have up to three data values, which can be accessed by an offset into this list. The list consists of nxm elements, where n is the number of points in the dataset and m is the number of data values.

6.3.2 The Position-List

This is the list of all points in the dataset.

This list contains the x, y, z coordinates of the point, a set of flags and a pointer to the *Cell-List* explained later. The flags are used for various purposes during the segmentation process and have the following functions:

- F_USED - When this flag is set, it indicates that the given point has been handled
- F_THR - When this flag is set, it indicates that the given point is above the threshold

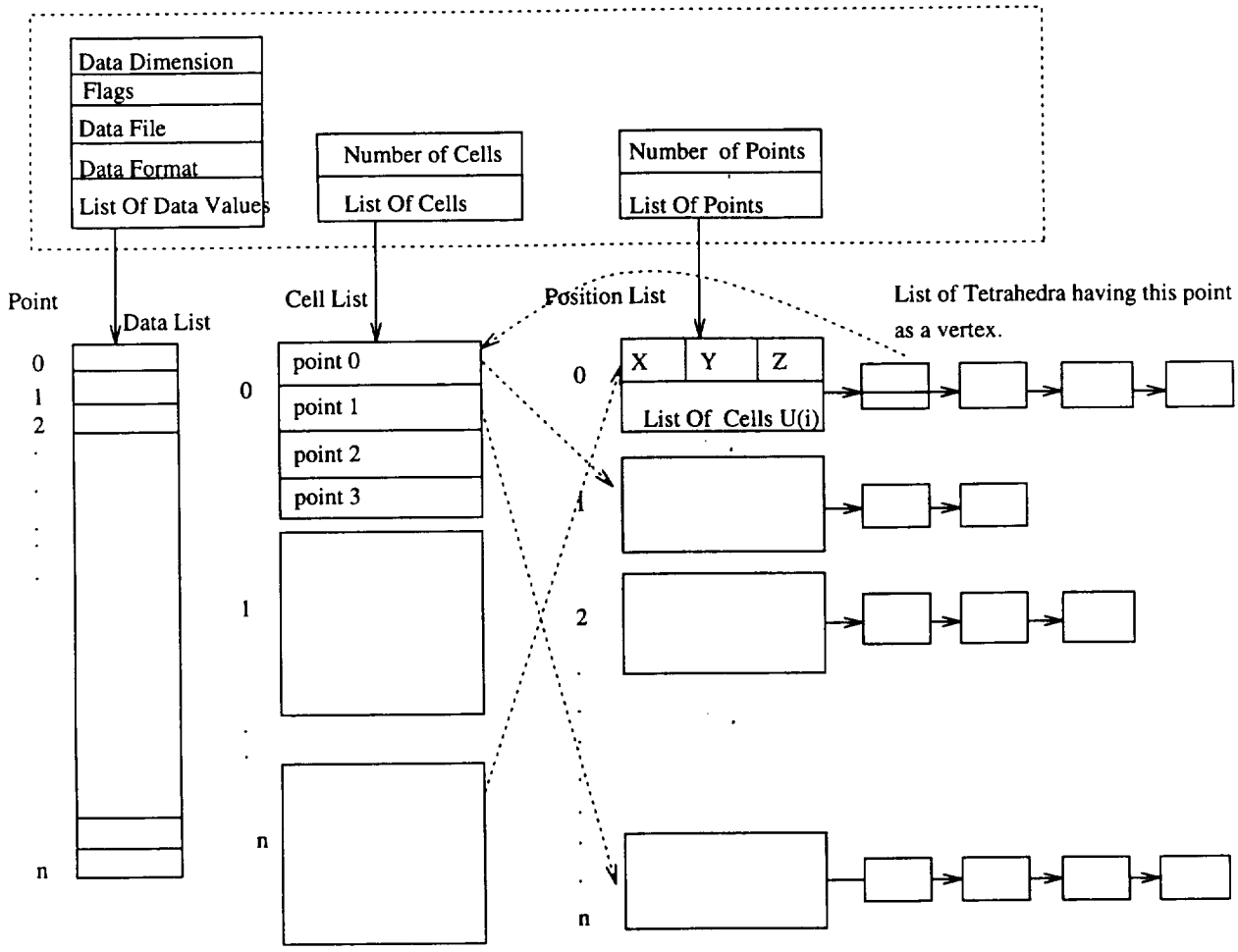


Figure 6.2: Elements in the data structure used

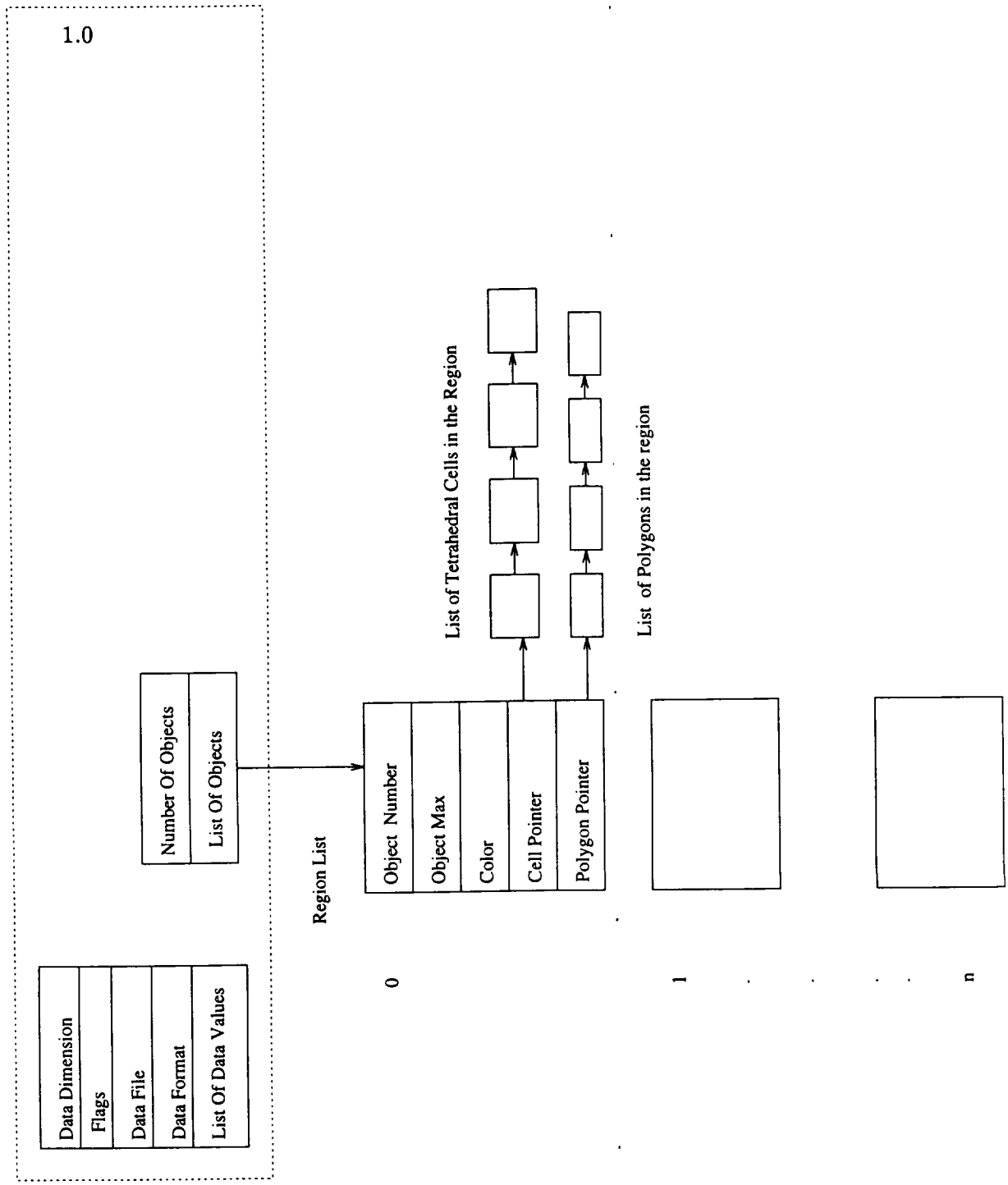


Figure 6.3: The Object/Region list

- F_UCD - This flag is used when the bounding surfaces for each object are generated

The Cell Pointer points to the list of tetrahedra which have this point in common which is the adjacency list. This list is generated by scanning the list of tetrahedral cells for every point in the dataset. This adjacency list is created once at startup.

6.3.3 The Cell-List

This is a list of all the tetrahedral cells in the dataset. Each cell is represented by pointers to the four vertices of the cell. Each pointer is an index into the Position-List described above. This list is also generated only once at startup from the input data file.

6.3.4 The Object-List

This is a list of objects that have been generated by the segmentation algorithm. The elements of the structure are shown in Fig 6.3. The Cell pointer points to a linked list of tetrahedral cells that comprise this object. This list can be used directly to calculate the mass, volume and other physical parameters for the objects. For visualization, a bounding surface can be generated and the list of triangles for the surface is pointed to by the Polygon Pointer. Each triangle consists of the x, y, z coordinates for each of the vertices.

Chapter 7

Algorithm and Implementation

This chapter provides the details of the algorithm to extract objects from unstructured datasets.

The intuitive overview of the segmentation algorithm is as follows. Initially, the adjacency list for the tetrahedral cells in the mesh is constructed. A seed point is then selected and all the cells having the seed as a vertex are determined and grouped together. For each vertex in this set, the adjacency list is scanned and cells are added to the *Object-List*, until no more points can be found. A bounding surface for each object is constructed using the marching tetrahedra algorithm.

The algorithm is divided into the following phases :

- The Input Phase
- The Preprocessing Phase
- The Segmentation Phase
- The Merging Phase
- The Post Processing Phase

7.1 The Input Phase

This is the setup phase for the algorithm. The dataset is scanned in and the *Position-List*, *Cell-List* and *Data-List* are populated.

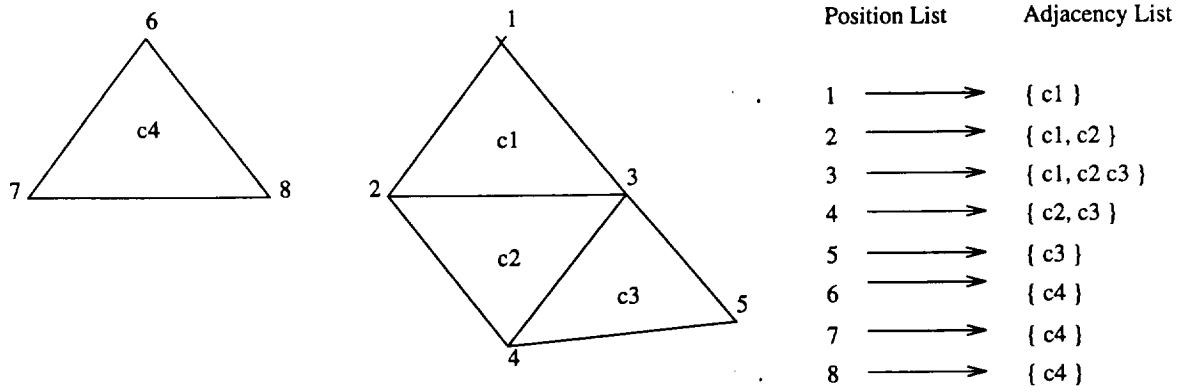


Figure 7.1: Sample Illustration of the Preprocessing Phase

7.2 The Preprocessing Phase

This is the first phase of the segmentation algorithm. The adjacency list is generated for the tetrahedra in the mesh.

All the points in the dataset are scanned and points above the threshold have their flag set to F-THR. All points below the threshold are thus eliminated. The threshold value is user determined and is dependent on the noise value.

```
For each data point P(i)
   If Data value D(i) ≥ Threshold, then mark point
   with the F-THRESHOLD flag.
For each point in P(i) over threshold
   For each Cell C(i) in the list C
      If Cell C(i) has P(i) as a vertex, add Cell C(i) to
      the adjacency list of P(i).
```

A sample illustration is shown in Fig. 7.1.

7.3 The Segmentation Phase

This phase will scan the adjacency list generated from the previous phase of the algorithm and generate a list of objects. The algorithm is described below.

```
While there are no points marked USED
  Determine Set of Points M(i) with highest data
  value and not marked USED.
    For each point in M(i) perform the following
    {
       Add an Entry O(i) to the Object-List
       Add all the Cells in the adjacency list of M(i) to O(i)
    }
  For each Object O(i) in the Object-List
    For each Cell in the Object-List
      {
       Determine the vertices from the Cell-List
       Determine the Cells connected to this vertex from the
       adjacency List of the Position-List.
       Add all these Cells to the Object-List for this Object O(i)
       Merge Objects using the Next phase of the algorithm
      }
```

An example is shown below (refer to Fig. 7.1).

Let point 4 have the maximum data value.

Pass 1:

$Object1 \longrightarrow c2, c3$

Pass 2:

$Object1 \longrightarrow c2, c3, c1$

Pass 3:

Get Next Maximum Value , point 8.

$Object2 \longrightarrow c4$

The *Object-List* now has 2 objects:

$Object1 \longrightarrow c2, c3, c1$

$Object2 \longrightarrow c4$

The example thus has 2 objects with the cells as marked to the right of the arrow.

7.4 The Merging Phase

This phase of the algorithm merges objects that have cells in common. Consider two objects comprising of the following cells:

$Object1 \longrightarrow c1, c2, c3$

$Object2 \longrightarrow c3, c7, c8$

Since Object1 and Object2 have cell "3" in common, the merged object would be:

$Object1 \longrightarrow c1, c2, c3, c7, c8$

The merge algorithm is executed whenever a new object is added to the *Object-List*. Let O_n be the object being created and C_n be the list of cells comprising this object.

```
For every Cell in C_n do
  If C(OBJ-NUM) = k, then
    Add the Cell-List C_n to O_k
    Rexecute the algorithm with O_k and C_k
  Set OBJ-NUM for all Cells = k
```

7.5 The Postprocessing Phase

This phase of the algorithm generates the bounding surfaces for the *Object-List*.

A surface is defined as a set of points $p_i = (x_i, y_i, j_i)$ such that $f(p_i) = 0$ for some function f. The bounding isosurface represents the boundary between the points above and below the user specified threshold value. A surface passes through a tetrahedral cell if and only if an intersection exists on an edge connecting oppositely signed cell vertices and no intersection exists otherwise. This implies that all tetrahedral cells

which do not have at least one point above threshold and one below can be eliminated from the isosurface calculations. A *marching tetrahedra* algorithm will generate three or four sided polygons which could then be rendered by some standard visualization package.

The algorithm is as follows:

```
For every cell in every object do
 {
  Determine edges affected [17] from Fig. 7.2
  Interpolate and determine points on the surface
  Generate the polygons from the points
 }
```

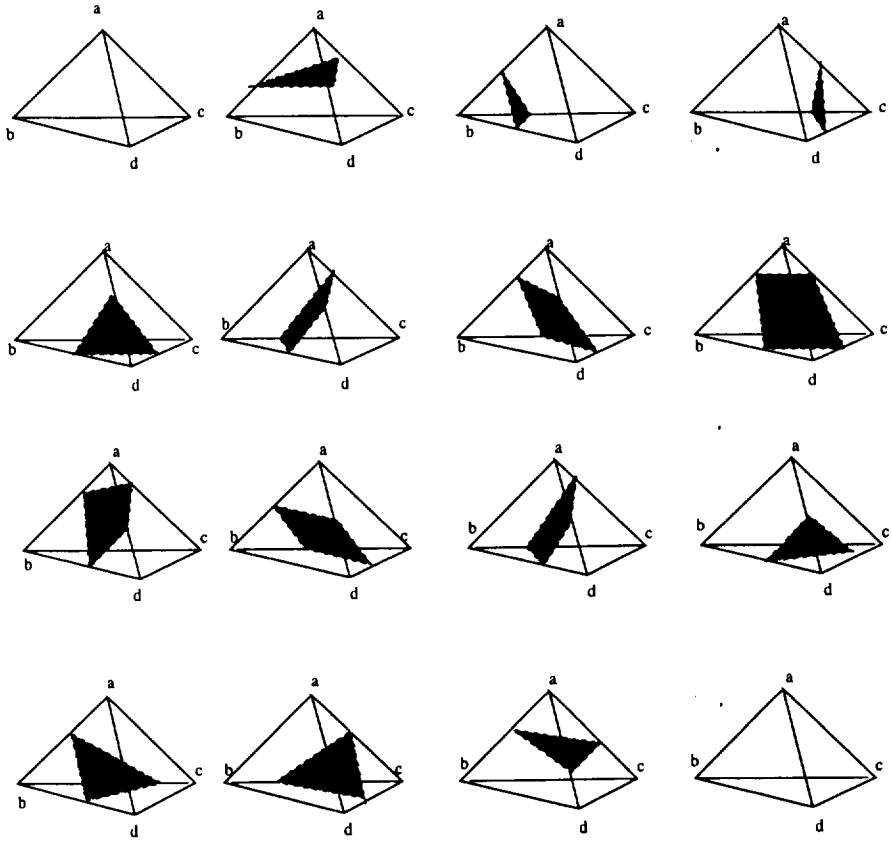


Figure 7.2: Cases to be considered when generating the bounding surface

Chapter 8

Examples

The segmentation technique was applied to two unstructured datasets.

The first dataset was provided by Lloyd Treinish. This dataset is the global atmospheric temperature described on a tetrahedral mesh. It has 311,040 tetrahedral elements and 74,095 points.

The second dataset was taken from the NASA web site provided by T. J. Barth and S. W. Linton. It has 595,536 tetrahedra and 112,551 points. This dataset consists of data for a multiple component wing computation of inviscid compressible flow (Mach = .2 and alpha = 0 deg).

The following figures describe some of the experimental results.

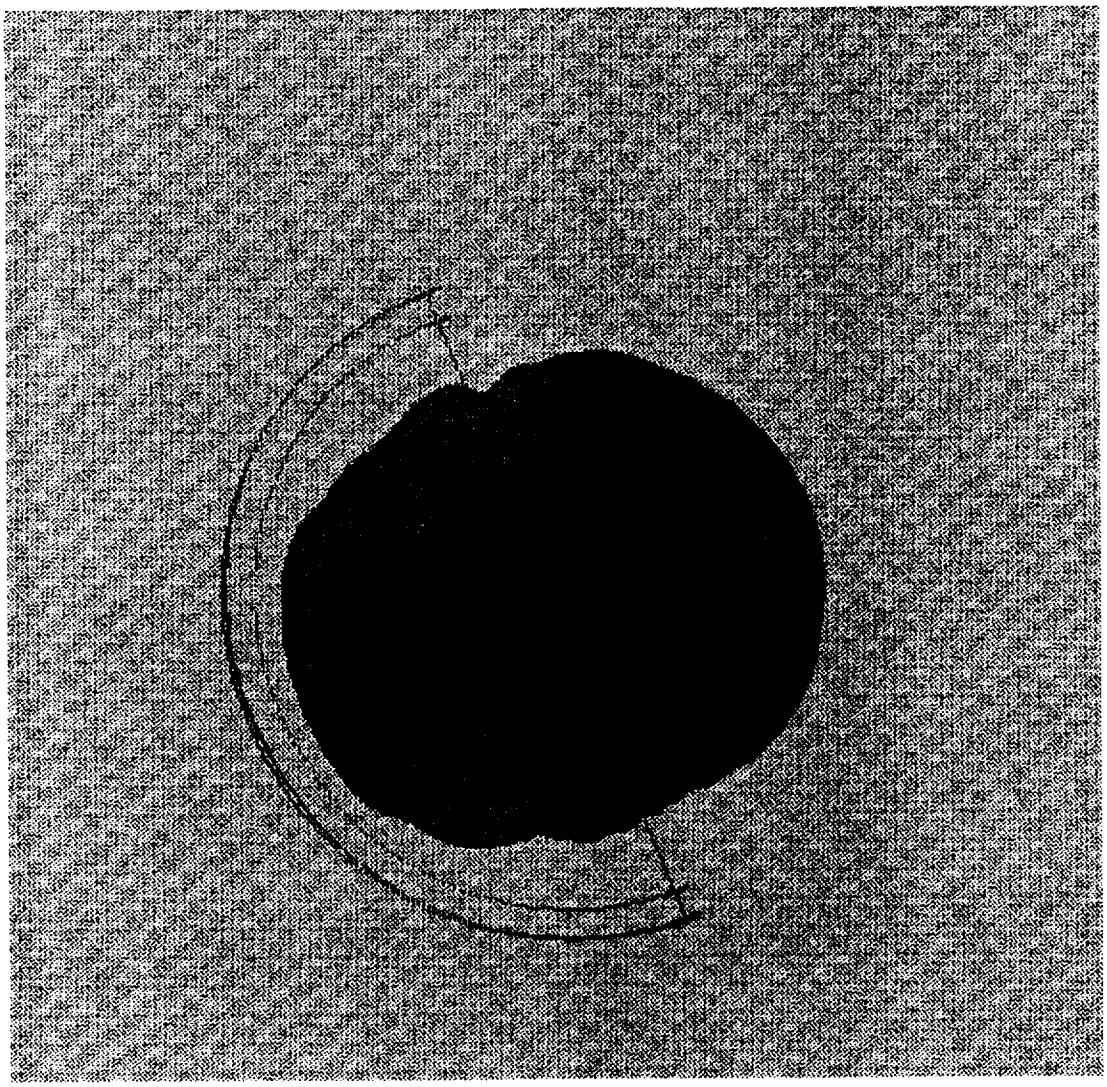


Figure 8.1: Segmentation of the atm dataset

```
Threshold(% of Max)      = 80%
Time for processing inputs = 11.3 sec
Time for segmenting        = 15.6 sec
Memory requirement         = 17.9 MB
```

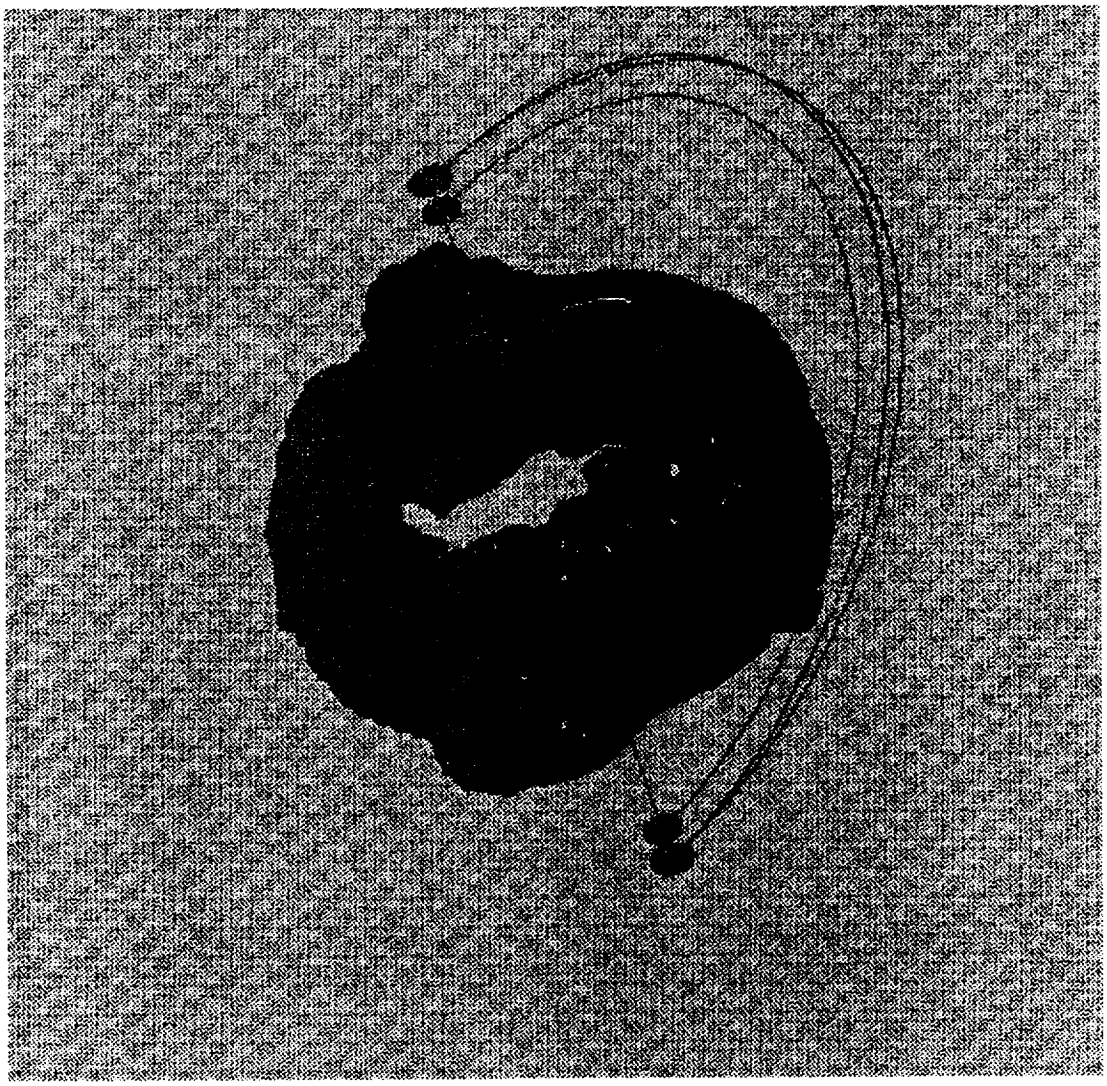


Figure 8.2: Segmentation of the atm dataset

```
Threshold(% of Max)          = 92%
Time for processing inputs = 4.6 sec
Time for segmenting          = 4.1 sec
Memory requirement           = 8 MB
```

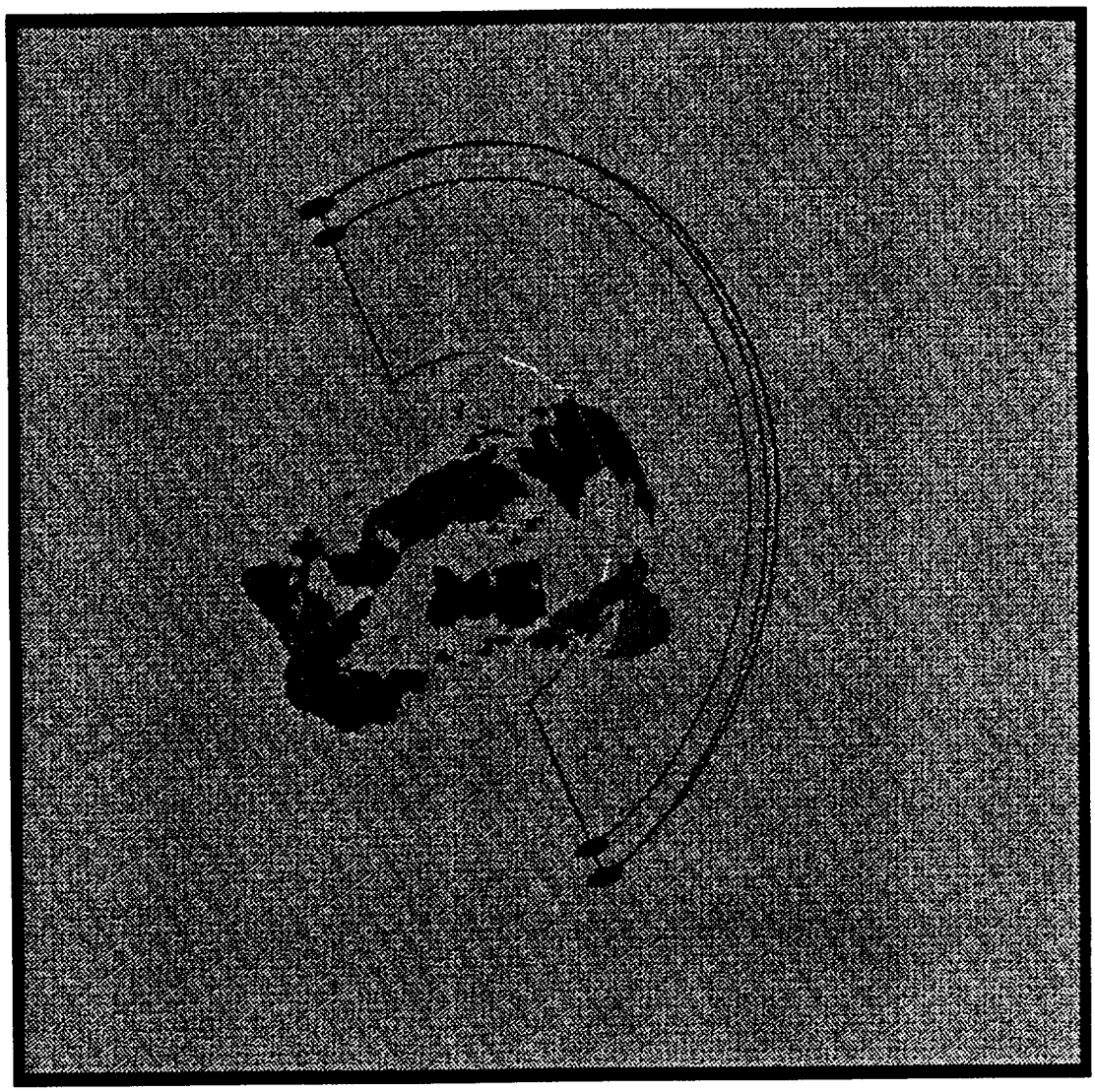


Figure 8.3: Segmentation of the atm dataset

```
Threshold(% of Max)          = 99%
Time for processing inputs = 3.7 sec
Time for segmenting          = 8.4 sec
Memory requirement           = 9.9 MB
```

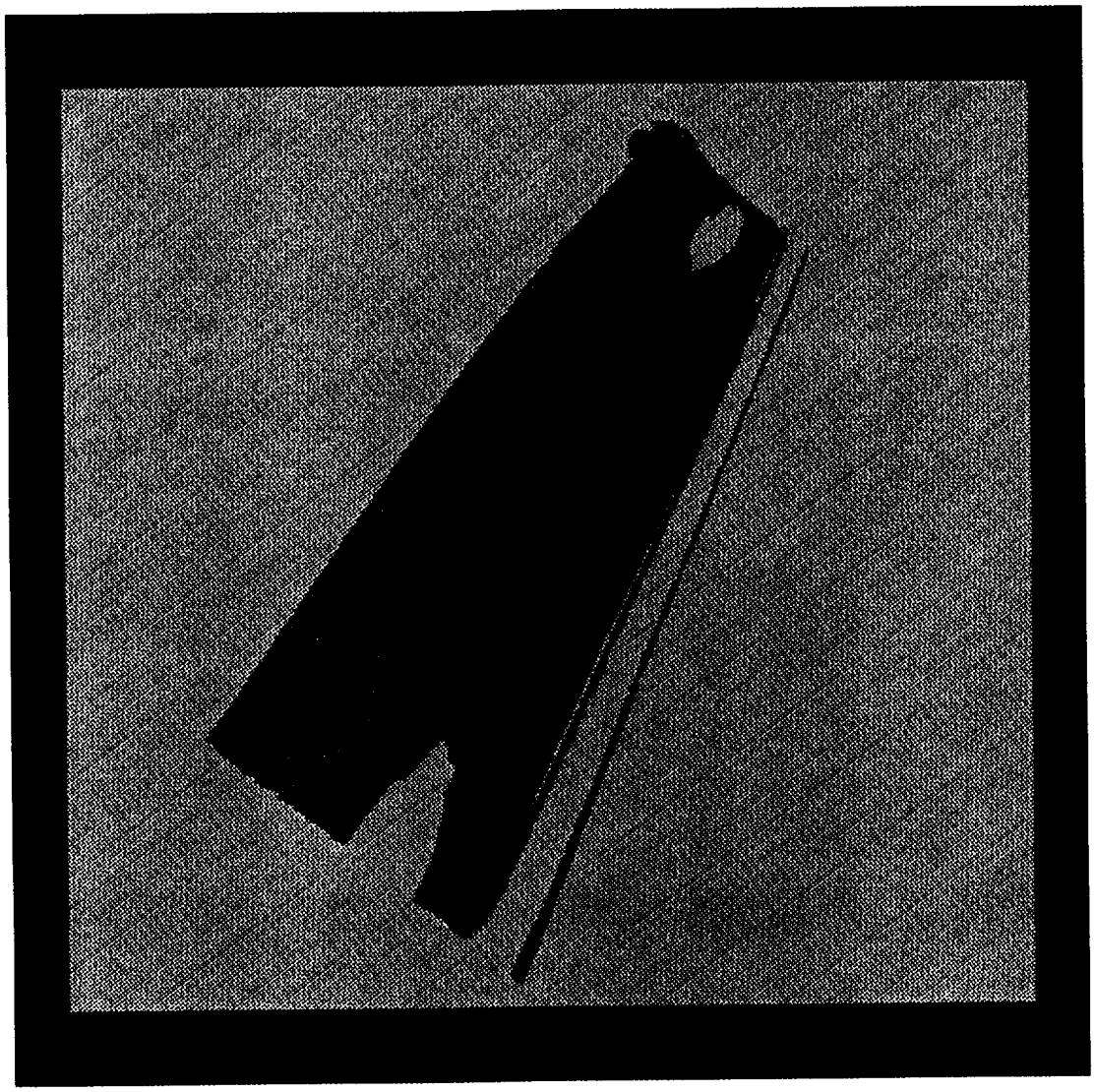


Figure 8.4: Segmentation of the wing dataset on density values

```
Threshold(% of Max)          = 93%
Time for processing inputs = 10.2 sec
Time for segmenting          = 40.2 sec
Memory requirement           = 20 MB
```

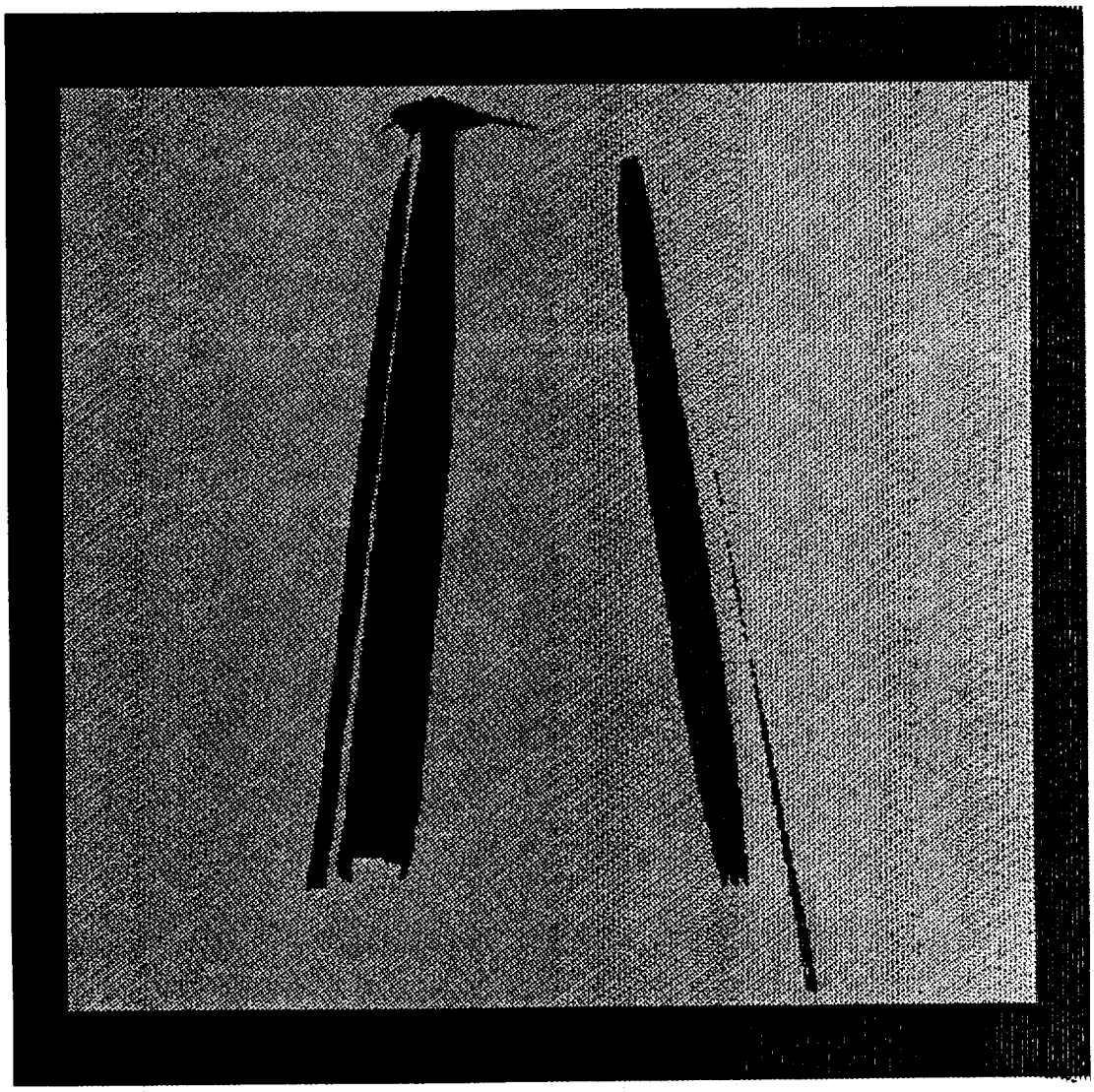


Figure 8.5: Segmentation of the wing dataset on density values

```
Threshold(% of Max)          = 92%
Time for processing inputs = 5.68 sec
Time for segmenting          = 36.05 sec
Memory requirement           = 19.8 MB
```

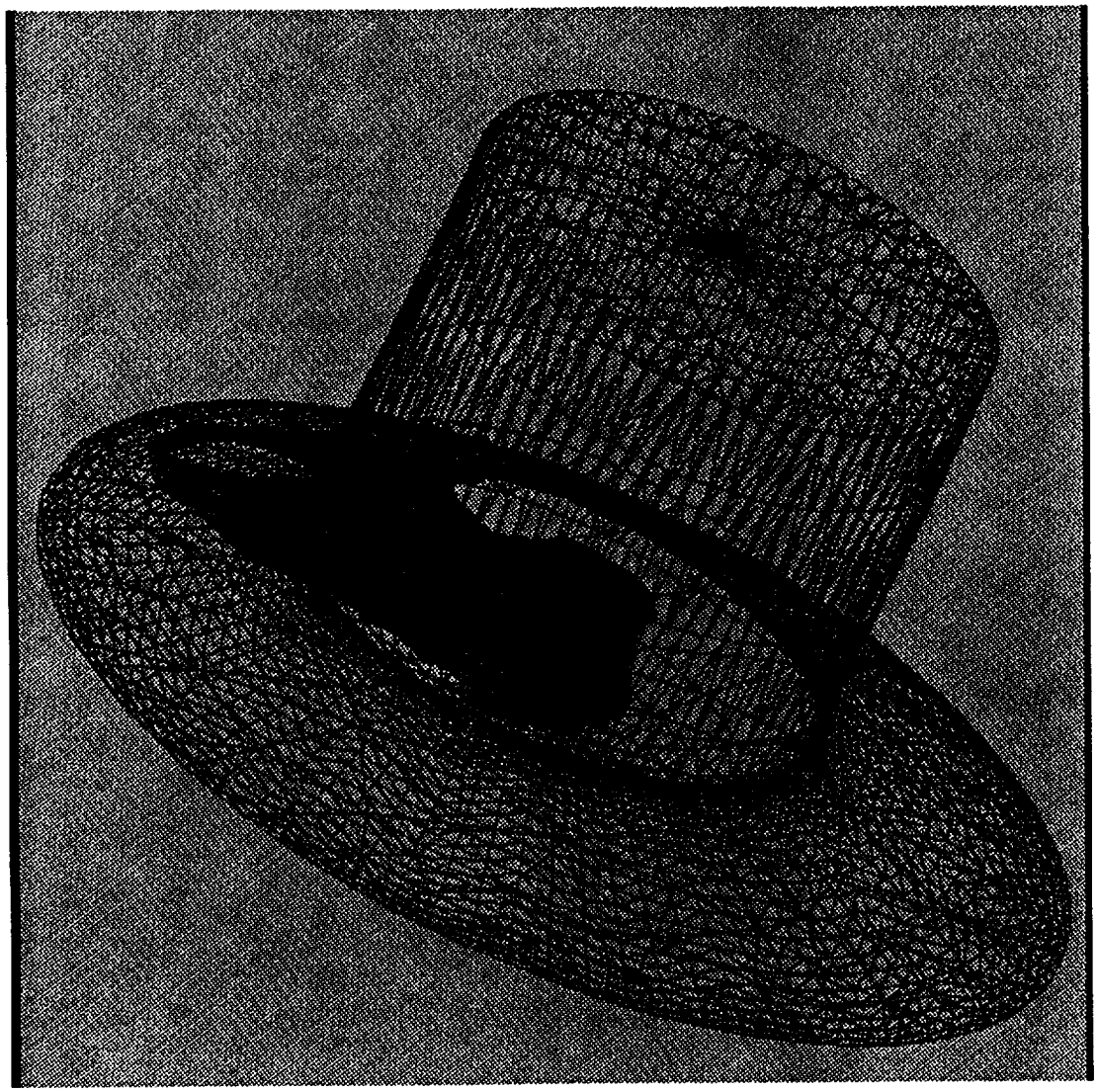


Figure 8.6: Segmentation of the wing dataset on mach number

```
Threshold(% of Max)          = 40%
Time for processing inputs = 11.5 sec
Time for segmenting          = 14.9 sec
Memory requirement           = 28.8 MB
```

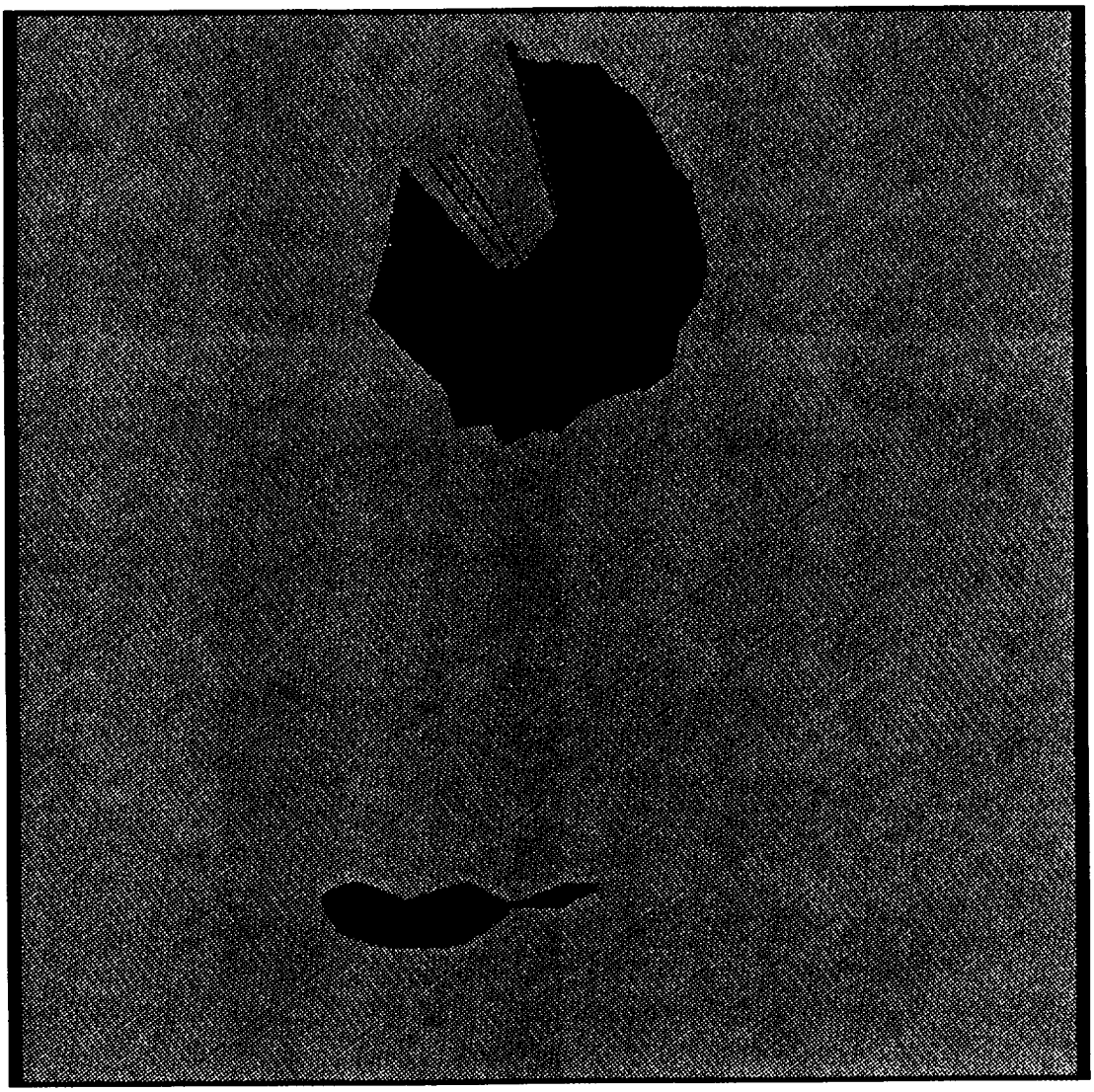


Figure 8.7: Segmentation of the first wing dataset on mach number

```
Threshold(% of Max)          = 42%
Time for processing inputs = 9 sec
Time for segmenting          = 12.2 sec
Memory requirement           = 26.8 MB
```

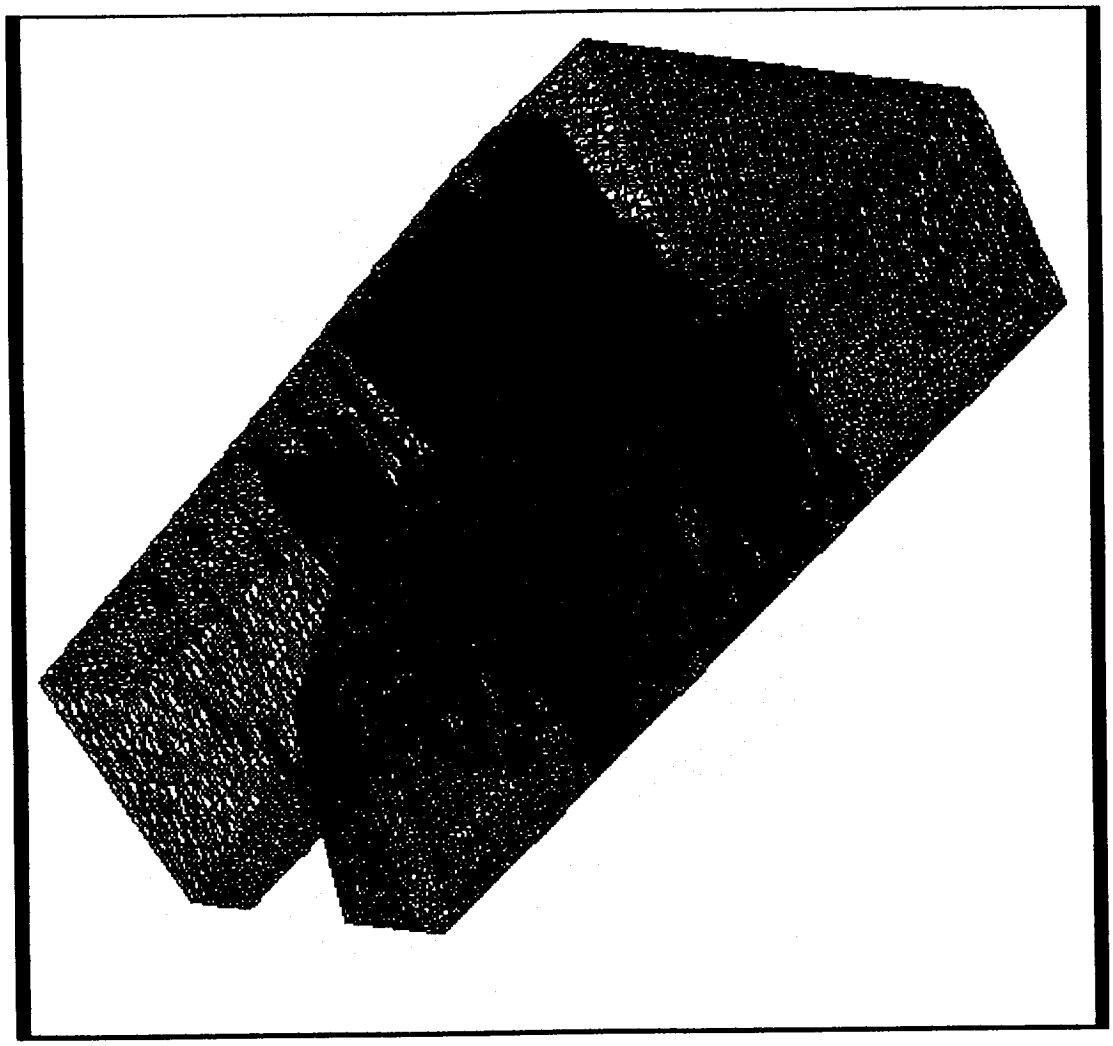


Figure 8.8: Segmentation of the second wing dataset on pressure

```
Threshold(% of Max)          = 10%
Time for processing inputs = 2.2 sec
Time for segmenting          = 12.1 sec
Memory requirement           = 6.5 MB
```

Chapter 9

Conclusion

In this thesis, we have implemented a segmentation algorithm for curvilinear and unstructured grids. Segmentation helps in focussing on coherent structures in the datasets. As mentioned earlier, the structures or "objects" thus obtained may be tracked in the time domain highlighting the topological relationship between evolving interacting variables. Data surgical operations may be performed on the extracted objects.

The work done in this thesis is preliminary. The algorithm should be parallelized to make it more efficient for large datasets. Tracking should also be incorporated. The concept of separation of objects should also be brought in. This is because, sometimes, it may be essential to consider one large connected object as several smaller objects. A lower resolution of the simulation might have rendered the object as being connected and such a large object may be difficult to comprehend when visualized.

Appendix A

Dataset Details

A.1 Object segmentation of a Curvilinear dataset(Figure. 4.3)

The dataset represents a 3D toroidal gyrokinetic particle simulation provided by Scott Parker of the Princeton Plasma Physics Laboratory, Princeton University.

"Tokamaks" are structures in magnetic fusion energy experiments([18]). An important part of these experiments is the understanding of the turbulent transport phenomenon which causes particles and energy to escape confining magnetic fields. This results in fusion output power comparable to the input power needed to maintain the tokamak plasma at very high temperatures(10keV). As practical experiments are difficult with the high temperatures involved, numerical simulations are very important for understanding the core plasma transport. The gyrokinetic equations are a reduced set of equations derived from the Vlasov-Maxwell equations for describing tokamak plasmas. The simulation results in large complex data consisting of 3D arrays evolving in time.

A.2 Object Segmentation of a Regular dataset(Figure. 4.1)

The dataset was produced by Dr. Shiyi Chen at the Los Alamos National Laboratory and has a spatial resolution of 256^3 grid points in a uniform mesh. The object segmentation technique was parallelized and run on a massive parallel computer(the CM5 with 512 processors) at the National Center for Supercomputing Applications(NCSA) and the SGI Onyx at Rutgers. A threshold value of 20% of the maximum was used to detect the objects observed.

References

[1] D. Ballard and C. Brown. *Computer Vision.* Prentice-Hall, Inc, Englewood Cliffs, New Jersey, 1982.

[2] H. E. Cline, W. E. Lorenson, S. Ludke, C. R. Crawford, and B. C. Teeter. Two Algorithms for Three-dimensional Reconstruction of Tomograms. *Medical Physics*, 15(3):320–327, May/June 1988.

[3] T. Todd Elvins. A Survey of Algorithms for Volume Visualization. *Computer Graphics*, 26(3):40–47, August 1992.

[4] G. Fekete and Lloyd Treinish. Sphere quadtrees: a new data structure to support the visualization of spherically distributed data. *SPIE Extracting Meaning from Complex Data: Processing, Display, Interaction*, 1259:242–253, 1990.

[5] Thomas A. Foley and David Lane. Visualization of Irregular Multivariate Data. *ACM Computer Graphics*, 1990.

[6] Thomas Fruhauf. Interactive Visualization of Vector Data in Unstructured Volumes. *Computer & Graphics*, 18(1):73–80, 1994.

[7] Richard S. Gallagher. Span Filtering: An Optimization Scheme For Volume Visualization of Large Finite Element Models. *ACM Computer Graphics*, 1991.

[8] M. Gao. Data Extraction and Abstraction in 3D Visualization, 1992. MS. Thesis, Dept. of Electrical and Computer Engineering, Rutgers University.

[9] Christopher Giertsen. Volume Visualization of Sparse Irregular Meshes. *IEEE Comp. Gr.*, 12(2):40–48, March 1992.

[10] K. Koyamada. Volume Visualization for the Unstructured Grid Data. *SPIE, Extracting Meaning from Complex Data: Processing, Display, Interaction*, 1259:14–25, 1990.

[11] M. Levoy. Display of Surfaces from Volume Data. *IEEE Computer Graphics and Applications*, 8(3):29–37, May 1988.

[12] Marc Levoy. Volume Rendering: A Hybrid Ray Tracer for Rendering Polygon and Volume Data. *IEEE Computer Graphics & Applications*, 10(3):33–40, March 1990.

[13] W. Lorensen and H. Cline. Marching Cubes: A High Resolution 3D Surface Construction Algorithm. *Computer Graphics*, 21(4):163–170, August 1987.

[14] K. Ma, M. Cohen, and J. Painter. Volume Seeds: A Volume Exploration Technique. *The Journal of Visualization and Computer Animation*, 4(2):135–140, 1991.

[15] J. Miller, D. Breen, W. Lorensen, R. O'Bara, and M. Wozny. Geometrically Deformed Models: A Method for Extracting Closed Geometric Models from Volumes. In *Computer Graphics*, number 4, pages 217–226, July 1991.

[16] Gregory M.Nielson and B.Hamann. The asymptotic decider: resolving the ambiguity in marching cubes. *Proceedings of Visualization '91*, pages 83–91, October 1991.

[17] Paul Ning and Jules Bloomenthal. An Evaluation of Implicit Surface Tilers. *IEEE Computer Graphics & Applications*, pages 33–40, November 1993.

[18] Scott E. Parker and Ravi Samtaney. CASE STUDY: Tokamak Plasma Turbulence Visualization. *Proceedings IEEE Visualization'94*, pages 337–340, October 1994.

[19] P. Sabella. A Rendering Algorithm for Visualizing 3d Scalar Fields. *Computer Graphics*, 22(4):51–58, August 1988.

[20] H. Samet. Neighbor finding in images represented by octrees. *Computer Vision, Graphics, and Image Processing*, 46:367–386, 1989.

[21] Peter Shirley and Allan Tuchman. A Polygonal Approximation to Direct Scalar Volume Rendering. *Computer Graphics*, 24(5):63–70, November 1990.

[22] D. Silver. Object-Oriented Visualization: Feature Based Algorithms for Visualization and Quantification. Tr174, CAIP Center, Rutgers University, February 1993.

[23] D. Silver and N. Zabusky. Quantifying Visualizations for Reduced Modeling in Nonlinear Science: Extracting Structures from Data Sets. *Journal of Visual Communication and Image Representation*, 4(1):46–61, March 1993.

[24] Don Speray and Steve Kennon. Volume Probes: Interactive Data Exploration on Arbitary Grids. *ACM Computer Graphics*, 24(5), November 1990.

[25] J. Udupa and D. Odhner. Shell Rendering. *IEEE Computer Graphics and Applications*, 13(6):58–67, 1993.

[26] C. Upson and M. Keeler. V-BUFFER: Visible Volume Rendering. *Computer Graphics*, 22(4):59–64, August 1988.

[27] Samuel P. Uselton. The Double Z Item Buffer for Volume Rendering Non-Regular Grids. *RNR Technical Report*, (RNR-91-027), September 1991.

[28] L. Westover. Footprint Evaluation for Volume Rendering. *Computer Graphics*, 24(4):367–376, August 1990.

[29] Peter L. Williams. Interactive Splatting of Nonrectilinear Grids. *Proceedings IEEE Visualization'92*, pages 1–7, 1992.

[30] Peter L. Williams. Visibility Ordering Meshed Polyhedra. *ACM Trans. On Graphics*, 11(2):103–126, April 1992.