Scientific Visualisation 3 1999
Assignment 1

Kevin Pulo
kev@zip.com.au
Student ID: 9719113

September 16, 1999

1 Question 1 — Visualising Lobster Density

1.1 Part (a)

1.1.1 2D Slices

2D slices are a simple method of extracting two dimensional data (suitable for displaying on a graphical workstation) from a three dimensional data set. The method is simple — a plane through the data set is specified (it may be orthogonal, that is, parallel to an axis of the data set, or arbitrarily oriented) and the data points at the intersection of this plane and data set are displayed on the plane. The plane may be displayed orthogonally on the screen or more commonly in its actual orientation in the 3D space.

Slices overcome in a simple fashion the inability to see an entire cube of data at once. They are often an extremely good way to get a first impression of a data set, and an animation which steps through all the slice planes is also very useful.

Slices are applicable to density data, such as the lobster's CT/MRI scan, as they correspond back to physical space — the density one would find taking a thin slice of the lobster. As such, it is easy for a human to interpret the visual image of the slice. However, while it is easy to understand what the slice is, caution must be exercised when attempting to understand how the different slices relate. Simply stepping through the slices is reasonable, but often requires a good deal of concentration to comprehend what is seen. Sometimes switching the slice to be perpendicular to its current position helps to show the context of the original slice.

The logical extension to this is to have more than one slice, each mutually perpendicular. This provides the desired context, and each slice supports the others in the user's comprehension of the data set. The major downside to this method is that it is often easy for the slices to obscure one another, which limits the amount of the slices that can be easily observed[1].

1.1.2 Isosurfaces

An isosurface is a surface connecting points of equal value. In the context of the CT/MRI lobster scan, the isosurface is a surface of constant density. This is appropriate because it allows the user to inspect the density of the lobster by examining the regions (surfaces) where it is constant. This is the same method by which elevation contours can show a hilly countryside.

[1] My AVS5 network, avs1a.net, attempts to avoid this problem by using a Thresholded Slicer, which has the ability to make different regions of the slice transparent.

There are several advantages to using isosurfaces. First, they are fully three dimensional, and so can provide a good deal of spatial context.

However, the disadvantages are numerous. As the surface is opaque (yet infinitesimally thin), internal structure is often lost, and thus is better suited to monotonic data, where all the points inside the (assumed closed) isosurface have values larger (or smaller) than that on the surface. An alternative is to cut away some of the surface to reveal internal structure, including the possibility of other internal isosurfaces.

Isosurfaces can be difficult or misleading to interpret. It is likely that the isosurface of the lobster will look like a lobster, especially towards the outside. The user needs to be careful to remember that this is a surface of constant density, not any real surface within the lobster.

Further, isosurfaces can be difficult to colour meaningfully. Colouring based on the density is mostly fruitless, as it results in a monochromatic colouring of the surface. However, some other variable may be coloured on the isosurface. The danger here is that the surface may then become too cluttered and hard to comprehend.

1.1.3 Direct Volume Rendering (DVR)

Direct Volume Rendering (DVR) is a more direct method of obtaining a 2D image of a 3D data set. Techniques such as slicing and isosurfaces (described above) are limited in that only some of the 3D data set directly contributes to the 2D output image. In DVR, all the data has the potential to contribute to the 2D output image.

DVR deals with voxels, volume elements, the 3D analogue of the 2D pixel. A variety of DVR methods exist, but all are based around the idea that voxels are assigned a colour and a transparency mask. This transparency means that obscured voxels may still contribute to the final image, and it is this mechanism that allows DVR to display an entire 3D data set, including its internals.

The most obvious advantage of DVR is that it has the potential to directly visualise the complete data set, in 3D, and with the internals at least partially exposed. Usually, where this level of sophistication in visualisation is required it outweighs the negative points.

DVR can be confusing to comprehend, the user must be aware that what they are seeing is a projection of the volume into the display window, and that there may be ambiguity as to the depth of some regions of the visualisation. The best way to resolve this is to provide interactivity — if the user is able to manipulate the volume being rendered and the viewpoint, they will be able to get a much better feel for the data set.

DVR images are also usually very fuzzy and blurry, this is because they are generated by considering many points in the data set and merging them. As such, they are not particularly suited where crisp images are required to study fine detail.

The lobster scan benefits from this technique by allowing the user to see inside the lobster (in terms of density). The DVR settings can be adjusted such that the regions of interest can be made more opaque, and thus will stand out more. The user can see how the more transparent regions surround these regions and their relationships to the more opaque regions.

1.2 Part (b) — Notes on viewing the AVS5 networks

1.2.1 ass1a.net — 2D Slices

This network uses the thresholded slicer to take arbitrarily oriented and positioned slices of the data. Further, the threshold is set to make the background data transparent, displaying just the part of the slice containing the lobster's data. Two slices are taken, one in the x-y plane of the lobster and the other perpendicular to this, running down the lobster's "spine". The transparency means that the background of either slice doesn't obscure the other, giving quite a good effect. Further, interactivity is achieved by selecting either slice plane in the top right Geometry Viewer and then using the right mouse button to drag this plane through the lobster. As this is done the display in both relevant Geometry Viewers is updated automatically, allowing the user to interactively examine the dual cross-sections of the lobster's density.

This file uses the thresholded slicer module, which appears to have some issues. Notably, its Resolution setting is not saved to the network file, and when loaded defaults to the lowest value, 12. For both thresholded slicers present in the network, this value should be increased to the maximum, 120.

1.2.2 ass1b.net — Isosurfaces

This network creates an isosurface of equal density in the lobster and displays it in the Geometry Viewer. The isosurface is coloured according to the rate of change of the density (in any direction, that is, the magnitude of the gradient of the density), red areas indicate slowly varying density (for example, the exoskeleton and an internal central region), and lighter yellow areas indicate faster varying density (typically softer tissues).

The value of the isosurface is wired to an animated float, and pressing the sleep button presented will step the isosurface through 20 steps from high density through to low density. This gives an excellent indication not only of how the density varies within the lobster, but also of how fast it changes within the lobster (due to the colouring of the isosurface).

1.2.3 ass1c.net — Direct Volume Rendering (DVR)

This network uses the Tracer and Display Tracker modules to perform a Direct Volume Rendering (DVR) of the lobster's density. Areas of high density are rendered as a semi-transparent light orange colour, and areas of lower density are rendered slightly less opaquely in a deeper red colour. As such, this technique is much more suited to the analysis of soft tissue and other low density regions. It clearly shows the outer shell as a fuzzy light orange region surrounding the reddish core of softer tissue.

The Display Tracker module is used to provide interactivity when examining the data set's rendering. However, when manipulating the view it is often best to turn off the Interpolate settings in both the Tracer and Display Tracker modules so that the display updates faster.

The view can be altered to penetrate the volume deeper, revealing more internal structure. This is achieved by modifying Tracer module's Alpha Scale control, which is actually wired to an Animated Float. Pressing the Sleep button of the Animated Float module will begin the stepping of the Alpha Scale, going inwards into the lobster in a similar fashion to ass1b.net.

2 Question 2 — Delaunay Triangulation

Delaunay triangulation is a method of constructing a surface mesh in a form suitable for computer graphics hardware. The input to the algorithm is a set of points in 2-Space (ie. a plane) and the height in 3-Space of the surface at that point. The output is connectivity information describing the surface as a series of tri-strips. In this way, it can be seen that Delaunay triangulation converts structured data (both regular and irregular) with implied connectivity to unstructured data with explicit connectivity.

Tri-strips are desirable from a computer graphics perspective because they are efficient in storage, (most) processing and importantly rendering. The essence is that after the first triangle has been specified, it only takes one vertex and two edges to extend the strip by another triangle. This leads to k triangles requiring $k + 2$ vertices and $2k + 1$ edges, both of which are more efficient than a triangular mesh of arbitrary connectivity. Further, for convenience the strips may be stored sequentially.

Delaunay triangulation works best when the surface has only small variations in the vertex density, that is, the vertices are evenly spaced in 3-Space. When this is the case, the algorithm will select triangles that are as close as possible to equilateral, resulting in an efficient and attractive surface. Further, the original data points are preserved, so no data is lost or approximated through interpolation.

However, the surface becomes more inefficient when it has large variations in the vertex density, that is, vertices which in some areas are closely bunched and in other areas are spread into vast plains. Here, the algorithm may potentially choose long, thin triangles in the regions of highly varying vertex density.

Delaunay triangulation cannot handle concave data sets at all. That is, where the parallel projection of the surface onto a plane produces a concavity, the Delaunay triangulation algorithm will blindly fill this cavity with triangles. The algorithm can, however, produce concave surfaces, providing the above condition is met[2].

The algorithm also suffers from being computationally slow to generate an optimal surface. This is usually avoided by allowing the surface to be "close enough", and applying heuristics to find semi-optimal surfaces.

In addition, data sets containing similar data points and features sometimes have dissimilar surfaces generated by Delaunay triangulation. A small change in a single vertex may alter the surrounding triangle or two, but this change may then have subsequent effects and repercussions throughout the rest of the surface. The two Delaunay meshes are still good approximations to the actual surfaces, but while the surfaces are similar the Delaunay meshes may not be (in terms of the triangles and their positions, sizes and relative layouts).

3 Question 3 — Marching Cubes

3.1 Overall Visualisation Purpose

Marching Cubes is an algorithm for the construction of surfaces. In the context of scientific visualisation, most commonly this is an isosurface, and Marching Cubes lends itself to this application very well. The surfaces are considered to be infinitely thin and reasonably well behaved — as shown in Section 3.4 Marching Cubes often has problems with surfaces that misbehave.

Marching Cubes is a high resolution method which derives from similar work on contouring. It requires node based volumetric data.

3.2 Algorithm Details

The algorithm works by examining each voxel of the dataset, and determines if each of the vertices is inside or outside the surface. This is recorded with the vertices, and in the case of isosurfaces involves a comparison of the vertex's data value with the user-selected isosurface value. In this way, the edges of the voxels along which the

[2] Consider, for example, a cube with one face "sagging" inwards.

isosurface intersects can be determined, and the position of the surface's intersection is approximated using trilinear interpolation.

The assumption is made that there is at most one surface intersection per edge, this gives rise to 256 possible voxel configurations (in terms of the vertex classifications — there are eight vertices, each of which can be in one of two states, $2^8 = 256$). By excluding cases which are specularly and rotationally isomorphic, this number reduces to 15 possible cases[3]. These cases describe the triangle mesh configuration of the voxel, and are thus used as the approximation to the surface.

The algorithm then calculates a normal vector for each vertex, by taking the central difference of the surrounding vertex values. This normal is used for shading purposes during the rendering stage.

The triangles and normals may then be output to any standard computer graphics rendering technique, where the image itself is formed.

3.3 Benefits

The main benefit of the Marching Cubes algorithm is that it produces high quality images, due to the algorithm placing high importance on preserving as much of the original data and structure.

The images produced have high levels of detail, and thus can be displayed well at high resolution. The main reason for this is that Marching Cubes generates sub-voxel sized facets, yielding detail on the scale of at least voxel dimensions.

The connectivity information between slices is preserved, that is, the top and bottom faces of voxels carry as much significance as the other faces, avoiding possible anti-symmetry. This is in contrast to techniques such as contouring slices followed by tessellating between slices, which exhibit this problem.

Importantly, the gradient information of the original data is preserved in the normal vectors of the vertices. While normal vectors could easily be calculated from the created triangle mesh and used for shading, the algorithm instead opts to use information from the dataset to generate these normals. The result is that the normals may not indeed be normal to the corresponding faces, but are more accurate when used for shading as they are based on the original data, not the triangle approximations. This makes more attractive images with better shading.

Finally, the algorithms divide and conquer approach is well suited to parallelisation, where this may be desired in an implementation.

3.4 Pitfalls

As Marching Cubes is an object-order algorithm, and not image-order, it can be inefficient, specifically, slow in computation and large in storage requirements. Each voxel may contribute up to four facets to the final geometry, and so even a modest sized dataset of 100 slices at 256 by 256 resolution may produce between 500,000 and 2,000,000 triangles[6]. This can have excessive processing and storage requirements, especially during the rendering stage.

Further, the as the facets generated are sub-voxel, many triangles end up being smaller than a single pixel after rendering. This is inefficient, and reveals the algorithms object-order nature. The Dividing Cubes algorithm [5, Reference 2] attempts to avoid this problem. It is similar to Marching Cubes except that it projects voxels that intersect the surface into the image plane. If these map to a single pixel or smaller, they are rendered as single points, otherwise they are subdivided into facets as with Marching Cubes.

[3]This includes the case where none of the vertices are inside the surface, but excludes the case where all the vertices are inside the surface. This is unusual, but is standard in all the literature on Marching Cubes.

Void data is not gracefully handled by the algorithm. If it is anticipated in the dataset it requires special cases to be added to the implementation if it is even treated at all. Also, not all datasets are well suited to the conceptual side of Marching Cubes. For example, amorphous data often doesn't contain thin surfaces suitable for isosurface visualisation, care must be exercised if it is attempted.

However, by far the largest problem with the Marching Cubes algorithm is that of ambiguity in the surface. This occurs when the wrong decision is made about whether a vertex is contained by the surface (such as when the assumption of one edge-surface intersection per voxel edge breaks down), or if the triangles chosen at adjacent voxels don't fit together properly. The result is a hole in the constructed surface, usually of the order of a voxel's face. One such example is illustrated in [2], which also refers to a similar algorithm by Wyvill *et al* in [2, References 2,3,4], which correctly handles these cases. In addition, Dividing Cubes [5, Reference 2], Marching Tetrahedra [5, Reference 14] and Asymptotic Decider [3], are all algorithms based on Marching Cubes which attempt to solve this problem.

4 Question 4 — Kernels

Kernels are a method of performing a convolution on an image. Kernels work by modifying each pixel's value in some way that depends on its surrounding neighbourhood of pixels. This can be used to increase or decrease the brightness of certain pixels, where these pixels are "selected" based on their local situation in the image, where they are and what their surrounding region is like. This is how kernels smooth images and remove noise — areas of high contrast have their pixels modified to slightly lower the high intensities and slightly increase the low intensities, and similarly isolated pixels and small regions of abnormally high intensity have their intensities lowered.

Kernels are usually represented as square matrices of odd side length. They may be either square or cross ("plus") shaped, as Equations 1 and 2 show, and are usually either $3x3$ or $5x5$ elements in dimension. The central matrix element is the pixel currently being processed, and the surrounding matrix elements correspond to the surrounding pixels. The numbers indicate the relative weights of the pixels in terms of their contribution to the new value for the current pixel. For example, Equations 1 and 2 show the common Gaussian kernel in both square and cross forms.

$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \tag{1}$$

$$\begin{bmatrix} & 1 & \\ 1 & 4 & 1 \\ & 1 & \end{bmatrix} \tag{2}$$

The Gaussian kernel is frequently used as a more sophisticated version of simple neighbourhood averaging, which uses the kernel in Equation 3. This method simply takes the average of the surrounding pixel values as the new value for the pixel, and the Gaussian kernel extends this idea by placing higher weights on closer pixels.

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{3}$$

Another method, which cannot be expressed as a matrix of pixel weights, is that of neighbourhood ranking. This method selects as its new pixel value the median of the surrounding pixel values.

The smoothing and removal of noise occurs because the kernel causes the pixels to become extended and their values are "smeared" over the area of the kernel. Thus areas of high contrast (such as sharp lines) have their edges blurred and their intensities lowered, causing the effect known as smoothing. Noise in the order of the size of the kernel is lessened or removed for exactly the same reasons — it becomes smeared over a greater area and so its overall affect on any local area in the image is lessened.

This introduces the first limitation in using kernels for noise removal — the size of the smallest interesting feature must be substantially larger than a pixel, and for better results should also be larger than the kernel size. The reason is simple — the kernel alters structures significantly when they are of comparable size and preserves larger structures. Indeed, structures smaller than the kernel, such as a single pixel, may be removed altogether or otherwise damaged by the kernel.

Some kernels, such as the simple and Gaussian neighbourhood averaging, suffer from the problem of edge shift, where edges in the image are shifted by a single pixel, due to the way the kernel gives those pixels an effect in the image at places where they should have none. The neighbourhood ranking method avoids this problem.

Smoothing and noise reduction kernels in general result in a loss of fine detail in the image (which is often desirable, since it may be noise) and more importantly a loss of image contrast. This can make small and low contrast features of the image harder to see. Related is a loss of greyscales and digitisation of the intensities which results from repeated applications of smoothing kernels.

The shape and type of the kernel can also influence the shapes of structures in the resulting images, especially with repeated applications. For example, both neighbourhood ranking and most cross kernels tend to make sharp corners more rounded.

References

[1] Lorenson and Cline, *Marching Cubes: A High Resolution 3D Surface Construction Algorithm*, Computer Graphics, Volume 21, Number 4, July 1987, pp 163 – 169.

[2] Durst, *Letters: Additional Reference to "Marching Cubes"*, Computer Graphics, Volume 22, Number 2, April 1988, pp 72 – 73.

[3] Nielson and Hamann, *The Asymptotic Decider: Resolving the Ambiguity in Marching Cubes*, Proceedings Visualization 1991, San Diego CA, October 22 – 25 1991, pp 83 – 91.

[4] O'Rourke, Joeseph, *comp.graphics.algorithms Frequently Asked Questions*, 1999, Question 5.10, URL: *http://exaflop.org/docs/cgafaq/cga5.html#What is the marching cubes algorithm?*

[5] Minty, Maccallum, Fisher, Hondroudakis, *Scientific Visualisation Course: Student Notes*, version 2.0.1, Section 4.3.1.2, The University of Edinburgh, 1996, URL: *http://www.epcc.ed.ac.uk/epcc-tec/documents/SciVis-course/SciVis.book_1.html*

[6] Lu, Tianyu, *Triangle Mesh Simplification*, Department of Electrical Engineering, University of Hawaii at Manoa.