14
Acceleration Techniques for Volume Rendering

Mark W. Jones

14.1 Introduction

Volume rendering offers an alternative method for the investigation of three dimensional data to that of surface tiling as described by Jones [1], Lorensen and Cline [2], Wilhelms and Van Gelder [3], Wyvill et. al. [4] and Payne and Toga [5]. Surface tiling can be regarded as giving one particular view of the data set, one which just presents all instances of one value – the threshold value. All other values within the data are ignored and do not contribute to the final image. This is acceptable when the data being visualised contains a surface that is readily understandable, as is the case when viewing objects contained within the data produced by CT scans. In certain circumstances this view alone is not enough to reveal the subtle variations in the data, and for such data sets volume rendering was developed [6, 7, 8, 9]. In this paper the underlying technique employed by volume rendering is given in Section 14.2 presented with the aid of a volume rendering model introduced by Levoy [6]. Section 14.3 examines various other volume rendering models and the differing representations they give of the same data. A more efficient method for sampling volume data is presented in Section 14.4 and acceleration techniques are covered in Section 14.5. In Section 14.6 a thorough comparison is made of many of the more popular acceleration techniques with the more efficient method of Section 14.4. It is shown that the new method offers a 30%-50% improvement in speed over current techniques, with very little image degradation and in most cases no visible degradation. A review of existing methods, systems and techniques is given in Section 14.7.

14.2 Volume Rendering

Many of the three dimensional data sets that need to be visualised contain an interesting range of values throughout the volume. By interesting, it is meant those parts of the volume to which the viewer's attention must be drawn in order for the viewer to gain insight to the physical phenomena the data represents. If the range of values is small, as for example the visualisation of the human skull from CT scans, then a surface tiling method will suffice. Most data sets do not fall into this category, but rather have a larger range of values or several different values which need to be represented in the visualisation. Such data sets need a method which can display the volume as a whole and visualise correctly those data values in which the viewer is interested.
The method known as volume rendering or direct volume rendering allows just that by rendering (visualising) the whole of the volume according to some definable criteria which describes those data values that are interesting, and how they should be dealt with.

14.2.1 Classification of data

In a typical data set every voxel contains some material property such as object density, probability, pressure or temperature. These data values have been measured or calculated in the *volume generation* stage. Visualisation of the data set will be based upon an interpretation of these values, which is defined by those values that are of interest, and how they will influence the overall visualisation. The data is converted from the values originally contained within the sampled volume into data in which each value indicates the importance of the corresponding voxel in the volume. These values are usually scaled linearly between the maximum and minimum representable by the number of bits available for each voxel. For example for an 8 bit representation, points with a value of zero are uninteresting, and points with a value of 255 are most interesting, with all other values scaled linearly between these. These values are used to compute opacity for each voxel by normalising them to lie between 0 and 1. An opacity of zero indicates the point is transparent, and therefore will not be seen, and a value of 1 is opaque, and will not only be seen, but will *obscure* points behind it. All other values are semi-transparent and will be seen with varying degrees of clarity. Each voxel is also given a colour, again based upon its value and/or position, which is combined according to opacity with all voxels behind it during the visualisation.
This process of converting the original data values into values of opacity is known as the classification process. It can be carried out with the use of a lookup table (LUT) using the very simple algorithm below.

for $k = 1$ **to** z
  for $j = 1$ **to** y
    for $i = 1$ **to** x
      $voxel'[i, j, k] = LUT[voxel[i, j, k]]$

The LUT will either have been defined by some expert who can decide between those values that are interesting and those that are not, or defined as the result of interaction. The user will adjust the values in the look up table according to the images produced in order to direct the visualisation of the data based upon what information needs to be gathered from the data and what view is desired of the volume.
Usually a more advanced classification process has to be carried out. In the case of data which has been generated by a medical imaging device, several stages of preprocessing may be carried out before the final grey level volume is created. This preprocessing consists of some or all of the following steps:

- *Filtering of original images:* The original images can be spatially filtered to remove noise using techniques such as median, modal and k–closest averaging filters. Other filters, could be used to detect and enhance edges [10, 11, 12].

- *Interpolation of images to create the 3D volume of grey level values:* Often the data is of a much higher resolution in the x and y image axes compared to the number of images (z axis). In some cases the resolution can differ by a factor of 20 [13]. If a regular volume is required which has equal dimensions in each axis, new slices must be interpolated between those already known [14, 15].

- *Segmentation of volume:* In medical applications it is desirable to display the different objects contained within the data separately, or to attach a different attribute, such as colour, to each object. In order to display, for example, the spinal column, from a set

of CT scans, the object in question has to be located and separated (segmented) from the surrounding objects. There is no automatic segmentation algorithm that works in all cases [16, 17] although interactive segmentation has proved to be successful [18, 19, 20, 21].

A typical segmentation process involves the user locating within a slice a seed point in the object they are interested in, from which a region is grown using thresholds. The thresholds define a range of values which encompass the object. The result of the region growing can be viewed in 3D, and parameters can be adjusted to correct the region interactively. Commonly regions that are incorrectly connected can have the *bridges* eroded, and regions that should be connected can be merged together using dilation. Currently segmentation is being used successfully to identify all parts of the human body (Section 14.7.8).

- *Opacity classification:* Often the straightforward mapping of value to opacity does not result in an accurate display of surfaces contained within the volume. In medical imaging, visualisation often has to display the interface between surfaces such as air/skin, skin/muscle and muscle/bone. These surfaces can be best visualised by setting the opacity according to a function of the gradient of the data [6]. Using this method the gradient is calculated using Equation 14.3 (page 256) and the opacity is assigned accordingly, such that voxels in the vicinity of the object interfaces (high gradients) will have high opacity.

The next step is to determine the representation of the volume. The image comprises of a regular grid of points, each of which can be assigned a certain colour. The image is calculated as a representation of what would be seen from a certain viewpoint, looking in a particular direction, under certain conditions. With the viewpoint and direction known a ray originating from each pixel in the image can be traced through the object domain.
The light and shade of each pixel is calculated as a function of the intensity along the ray originating from that pixel. The value of each pixel is governed by how the volume rendering model interprets the intensity profile and thus visualisation of the data is determined by the model.
The continuous intensity signal along the ray is reconstructed by taking samples at evenly distributed points on the ray and it is these samples that are evaluated by the model.

14.2.2 Levoy's rendering model

The rendering model popularised by Levoy [6] is that of assuming the volume to be back lit by a uniform white light. The light passes through the volume and is attenuated by the opacity it encounters on the path it tracks through the volume. The resulting light that reaches the image is the light that exits the volume. Each pixel is also given a colour which is determined by the *colour* of each sample along the ray. The colours are chosen in a colour classification process which is similar to the opacity classification process, except the colour is a triple of red, green and blue. The colour at each sample can also depend upon the lighting model chosen as will be described later.
Each voxel value in the volume is given by the function

$$f : \Re^3 \Rightarrow \Re \tag{14.1}$$

where $f(x)$ = measured value at x if $x \in \aleph^3$
otherwise $f(x)$ = value trilinearly interpolated from the 8 closest neighbours of x.

The quadruple $< r, g, b, \alpha >$ at each data point is calculated from the lookup table for the value at those points.
$< r, g, b, \alpha >$ = LUT($f(x)$)
where r, g, b, and α are the red, green, blue and opacity components respectively.
A function

$$g :< r, g, b, \alpha > \times \Re^3 \Rightarrow \Re \tag{14.2}$$

is defined such that
$g(\alpha, x)$ = the opacity at the point x trilinearly interpolated from its 8 closest neighbours.
$g(r, x)$ = the red at that point, and so on.
For each pixel **p** in the image, where **p** = $< i, j >$ i=1,...,I_x, j=1,...,I_y, where I_x and I_y are the dimensions of the image in the x and y axes, a ray $\mathbf{R_p}$ is traced into the object domain. If the ray intersects the volume of data, the length of the ray passing through the volume will be $l =\mid \mathbf{R_p} \mid$. If the ray is sampled at intervals of τ, there will be $K = \frac{l}{\tau}$ samples, each given by $\mathbf{R_p(n)}$ where $n = 1, ..., K$. $\mathbf{R_p(1)}$ is the ray entry point and $\mathbf{R_p(K)}$ is the last sample before the ray exits the volume. The colour and opacity at each sample is given by the function g, and for the nth red sample would be $g(r, \mathbf{R_p(n)})$. If attenuation of a back light with colour Background = $< B_r, B_g, B_b >$, is assumed as the rendering model, the resulting pixel colour triple $< p_r, p_g, p_b >$ is given by $< C_r, C_g, C_b >$ where for each pixel **p**
$C = Background$
for $n := k$ **downto** 1 **do**
$opacity = g(\alpha, R_p(n))$
$C_r = (1 - opacity) \times C_r + opacity \times g(r, R_p(n))$
$C_g = (1 - opacity) \times C_g + opacity \times g(g, R_p(n))$
$C_b = (1 - opacity) \times C_b + opacity \times g(b, R_p(n))$
endfor
$p =< C_r, C_g, C_b >$

This algorithm contains no view dependent visual cues which may aid the understanding of the visualisation. These can be added in the colour accumulation stage by calculating the spatial point of each sample and applying a shading operator to it before compositing that colour. The simplest shading operator is to depth cue the data. The process of depth cueing involves calculating the distance z, from the ray origin (pixel) to the sample point. The colour at the sample point is then attenuated according to some function based upon that distance. The simplest function is to multiply the colour intensities by $\left(1 - \frac{z}{Z}\right)$ where Z is the maximum distance in the scene.
It is also possible to apply a shading technique such as Phong's, by determining a normal to the data at the sample point which can be used in a similar way to a surface normal. The normal can be calculated by trilinear interpolation from the normals of its 8 closest voxel neighbours, whose normals are calculated using difference operators.
For voxel x, y, z in a data set where each voxel is a unit cube, its normal G is calculated using,

$$\begin{aligned} G &= (g_x, g_y, g_z), \\ g_x &= f(x+1, y, z) - f(x-1, y, z), \\ g_y &= f(x, y+1, z) - f(x, y-1, z), \\ g_z &= f(x, y, z+1) - f(x, y, z-1). \end{aligned} \tag{14.3}$$

The result of shading is a value between 0 and 1 which can be used to multiply the colour for a sample before being composited.

14.2.3 Results

Using this algorithm and suitably defined classification functions, images such as that of Figures 14.1 and 14.2 are created.

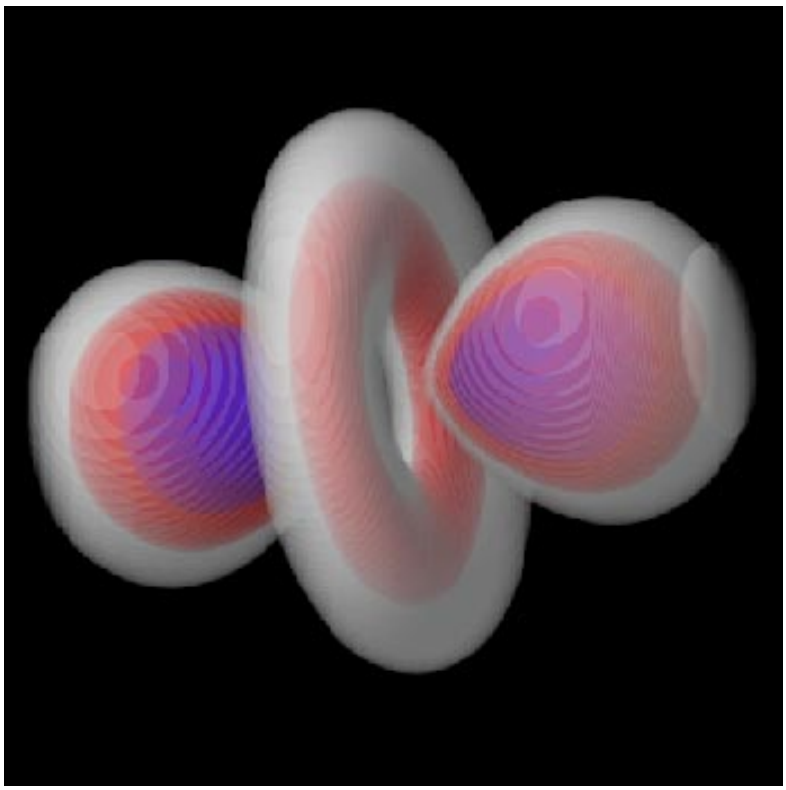

Figure 14.1: Volume rendering of AVS Hydrogen data set.

But what is the intended interpretation of these images? Firstly the classification functions must be examined. In the case of Figure 14.1, values below 50 have been set to be transparent (that is $\alpha = 0$). Values between 50 and 100 have been set to $\alpha = 0.15$ and the colours set to $< r = 1.0, g = 1.0, b = 1.0 >$, values between 100 and 200 set to $\alpha = 0.15$ and the colours set to $< r = 1.0, g = 0.0, b = 0.0 >$ and above 200, $\alpha = 0.8$, $< r = 0.0, g = 0.0, b = 1.0 >$. The image indicates that values above 200 (up to 255) occur in two distinct ellipsoidal sections, and values between 50 and 200 occur in two larger ellipsoids, and a torus around the centre. In this case the interpretation is correct. The lighting model has given the eye the required cues to determine the *shape* of the volumes contained within the data, and the colouration and opacity have allowed the three separate ranges to be distinguished.
In the case of the CT image (the data having values -1117 to +2248), values below -300 have been set to be transparent and values between -300 and 50 set to $< r = 1.0, g = 0.79, b = 0.6 > \alpha = 0.12$. These values contain the range for the skin in the CT images. The other range we are interested in are those values that produce the skull, 300-2248, and therefore these are set to $\alpha = 0.8$ and $< r = 1.0, g = 1.0, b = 1.0 >$. The interpretation is also correct in this case, the skin is mostly transparent, thus revealing the skull contained within. The lighting functions provide enough information to understand the surfaces – both exterior (skin) and interior (skull).

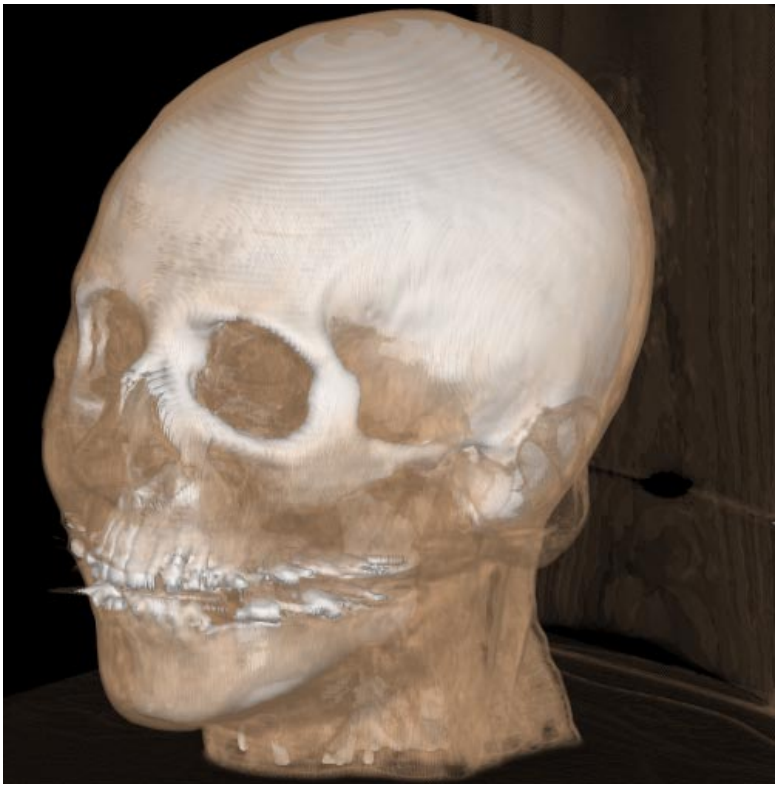

Figure 14.2: Volume rendering of CT head data set.

14.3 Volume Rendering Models

14.3.1 Maximum value images

For many applications specific viewing models other than the one mentioned in Section 14.2.2 are employed to present visualisations. The simplest is the widely used maximum value visualisation[1] [22]. For this approach the maximum sampled value along a pixel's ray is the value used for that pixel's colour. This method avoids the need for expensive shading calculations and composition operations, and therefore requires less computation than using a model such as Levoy's. The method is generally used to produce visualisations in which the data is rotated, so that the visualiser can identify areas of high value (hot spots). This type of visualisation results in images not unlike x-rays, and therefore is particularly useful in the medical profession where the users of such methods are familiar with x-rays.
The main problem with this method is the fact that images 180^o apart are identical. For animations of rotating objects this produces the effect of the object seeming to turn back and forth rather than rotate. Depth cueing can be employed to avoid this problem by attenuating the image according to the depth at which the maximum value occurs, and indeed, animations produced using depth cueing provide far more depth information than without.

14.3.2 X-Ray images

The maximum value images of Section 14.3.1 produce images that look like x-rays. X-ray images of the data can be produced by simulating how the rays pass through the volume,

[1] Personal communication with K. S. Sampathkumaran of Washington, USA. in which he stated that for medical applications maximum value projection gave a valuable insight to the hotspots within the data

calculating the absorption due to the density of the volume. If the density, ρ, along each ray from 0 to l is

$$\rho = f(t), 0 \leq t \leq l, \tag{14.4}$$

the density of material along the path of the ray is

$$\int_0^l f(t)dt. \tag{14.5}$$

This can be calculated quite simply using either Simpson's Rule (with three ordinates):

$$\int_0^l f(t)dt \approx \frac{1}{3}d\left(y_0 + 4y_1 + y_2\right), \tag{14.6}$$

or the Trapezium Rule:

$$\int_0^l f(t)dt \approx \frac{1}{2}d\left(y_0 + 2y_1 + \ldots + 2y_{n-2} + y_{n-1}\right), \tag{14.7}$$

where d is the distance between each sample. As $d \rightarrow 0$ the approximation becomes more accurate.
The image displayed is a function of the material density – for example, the inverse of the material density, and normalised to use the full range of brightness of the display. The result is an image which looks similar to an x-ray.

14.3.3 Standard model without normal shading

The main computational expense with the standard model is the shading calculation done at each sample position within the volume. If this could be removed, the amount of computation required can be dramatically reduced. One way would be to completely disregard shading operations and display the colour and opacities directly. This results in an image generated rapidly that contains no shading cues but does contain enough information to identify interesting areas within the data set.
Alternatively the expensive calculation of data normals can be disregarded, with the data being shaded using depth cueing. This gives the same look upon the data as the method without any shading at all, and in addition provides some feeling of the distance of the objects within the data. The drawback to both methods is the fact that valuable curvature information is not present in the images they produce.

14.3.4 Sabella's method

A major contribution to the subject of producing alternative images from 3D data sets is that of Sabella [7]. The volume data is treated as a varying density emitter, where the density is a function of the data itself. Sabella regards this to be an equivalent model to a particle system where the particles are sufficiently small. Rays originating from each pixel are traced into the volume data set as described in Section 14.2 and the volume is sampled as before. Sabella's viewing model calculates four parameters – the peak value occurring along the ray, and the distance where it occurs, the attenuated intensity, and the centre of gravity of the field along the ray. A combination of three of these four parameters are then mapped to the Hue, Saturation and Value (HSV) model.
The peak value and distance are self explanatory, whereas the attenuated intensity needs more explanation. It is taken to be the brightness, B, along the ray, where

$$B = \int_{t_1}^{t_2} e^{-\int_{t_1}^{t} \tau p^{\gamma}(\lambda) d\lambda} p^{\gamma}(t)\, dt \tag{14.8}$$

The variables and expressions of this equation are described in [7].
The $\int_{t_1}^{t} p^{\gamma}(\lambda)\, d\lambda$ represents the number of particles in the volume along the ray that spans between t_1 and t, where
t_1 and t_2 are the ray entry and exit points of the value data,
t is the ray parameter, $t_1 \leq t \leq t_2$, and
p is the density.
The term $e^{-\int_{t_1}^{t} \tau p^{\gamma}(\lambda) d\lambda}$ could be regarded as the transparency for simplicity, and is in fact proportional to transparency. The integral therefore calculates the contribution to the final intensity of particles along the whole ray, and the attenuation of the light due to the density of the particles.
It is shown that equation 14.8 is the continuous form of the discrete equation

$$\sum_{i=1}^{n} b_i \prod_{j=1}^{i-1} \theta_j \tag{14.9}$$

where b_i can be considered as the brightness at sample i, and θ_j is the transmittance (that is transparency) of sample j, $0 \leq \theta_j \leq 1$. The term $\prod_{j=1}^{i-1} \theta_j$ calculates the attenuation due to all the samples *in front* of sample i, and therefore the product $b_i \prod_{j=1}^{i-1} \theta_j$ is the contribution of sample i to the intensity. The sum of all contributions gives the brightness, B.
A practical method of computation can be derived from this equation:

$B = 0.0;$
$o = 1.0;$
for $i = 0$ **to** $length$ **do**
  $B = B + b_i \times o$
  $o = o \times \theta_i$
endfor

The peak value parameter is mapped to Hue, and the attenuated intensity is mapped to Value. Either the centroid, or distance is mapped to the saturation parameter. This has the effect of allowing colour to represent the maximum values, and the distance parameters to give depth information in the form of saturation, leading to what can most easily be described as fog. The attenuated intensity parameter gives an impression of the distribution of the data.

14.4 Efficiency Aspects of Volume Rendering

The volume rendering method as described in Section 14.2 is a costly process to compute due to the number of samples that have to be determined. In this section a new method which reduces this computation is presented. The method has been developed as a result of analysing where work is done in the volume rendering process. This section introduces the idea of choosing the distance at which samples are made along the ray such that computation can be reduced. The method requires a bare minimum of precalculation (a few tens of mathematical operations), and works without constraints for arbitrary view points. In Section 14.4.1 the calculations required by the volume rendering process are examined. In Section 14.4.2 the process of choosing the sampling distance to reduce the number of

calculations is introduced, and in Section 14.4.3 the effect this has on computational time and the images produced is investigated. Section 14.4.4 outlines some view artifacts this method introduces during animation loops, and suggests how these may be overcome. Section 14.4.5 offers conclusions for this method.

14.4.1 Computation involved during volume rendering

The algorithm has been sufficiently described elsewhere [6] for implementation, but in each case, although the spacing for the sampling is described, the value is left up to the reader. In the majority of cases a value of 1 unit will be chosen for simplicity, where each voxel in the data set has length 1 unit in each dimension. Figure 14.3 indicates the situation.

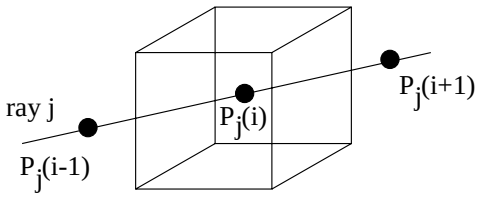

Figure 14.3: Ray passing through cube sampled at even intervals.

The spatial position of the i^{th} sample along the j^{th} ray is given by $p_j(i)$. The value which is sampled at that point is given by $S_{p_j(i)}$ where the sample is calculated by trilinear interpolation from the 8 neighbouring voxels that make up the cube that contains the point $p_j(i)$. The sampled value is usually the interpolation of the value and normal from each of the 8 neighbouring voxels [23] or the interpolation of the opacity and colour components, and normal from each of the 8 neighbouring voxels [6].
In order to calculate the normal of the sample $S_{p_j(i)}$ in Figure 14.3, the normals at each voxel $v_0, ..., v_7$ must be determined using an appropriate method, such as central differences. This involves the calculation of 8 central differences, each of which involves 3 subtractions, 6 voxel look ups (Equation 14.10) and one normalisation step.

$$\begin{aligned} G &= (g_x, g_y, g_z)\,, \\ g_x &= f\,(x+1, y, z) - f\,(x-1, y, z)\,, \\ g_y &= f\,(x, y+1, z) - f\,(x, y-1, z)\,, \\ g_z &= f\,(x, y, z+1) - f\,(x, y, z-1)\,. \end{aligned} \tag{14.10}$$

The normalisation step (Equation 14.11) involves 3 subtractions, 2 additions, 3 multiplications, 1 square root and 3 divisions.

$$n' \leftarrow \frac{n}{|\,n\,|} \tag{14.11}$$

To calculate the normals at each of the 8 cube vertices requires a total of 16 additions, 48 subtractions, 24 multiplications, 24 divisions, 8 square roots and 48 voxel look ups.
If the offsets in the cube for each axis are r_x, r_y and r_z (Figure 14.4), the values at $p_j(i)$ are calculated using trilinear interpolation as follows :

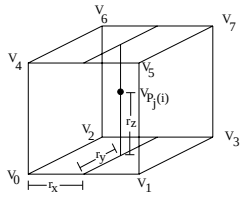

Figure 14.4: Trilinear interpolation within cube.

$$
\begin{aligned}
t_0 &= (v_1 - v_0)r_x + v_0 \\
t_1 &= (v_3 - v_2)r_x + v_2 \\
t_2 &= (t_1 - t_0)r_y + t_0 \\
t_3 &= (v_5 - v_4)r_x + v_4 \\
t_4 &= (v_7 - v_6)r_x + v_6 \\
t_5 &= (t_4 - t_3)r_y + t_3 \\
v_{p_j(i)} &= (t_5 - t_2)r_z + t_2
\end{aligned}
$$

which involves 7 subtractions, 7 additions, and 7 multiplications for each value that must be interpolated and a total of 8 voxel look ups. Furthermore, if a value such as opacity is required, a further classification table look up is necessary. If the value being interpolated is the normal, each v_i is now a triple and the calculations are therefore trebled. Also each step must be normalised, so a further 7 normalisation calculations are required. A summary of the total required computational operations is given in Table 14.1.

operation	normal	rgbα	value
+	51	28	7
–	90	28	7
$\div$	45		
$\times$	66	28	7
$\sqrt{}$	15		
voxel l. u.	48	8	8
table l. u.		32	

Table 14.1: Calculations required for trilinear sampling process.

As can be seen from the table, normal interpolation is the most expensive operation, particularly with its 15 roots, and 45 division operations. When this is put into context with the operation of the algorithm, it is realised that for a large image (500×500) and a large data set ($256 \times 256 \times 256$) this normal interpolation can be done up to a maximum of 100 million times. (Number of rays $\times$ the maximum ray length). It is this figure that lead to the investigation of ways to reduce the calculation.
One immediate solution is to precompute the normals at each of the voxels in a preprocessing step. The problem with this is the extra storage required – for a data set of $256 \times 256 \times 256$ an additional 200MBytes of storage is needed (12 Bytes for a normal per voxel). As a result of the usual problems of memory size, disk size and swapping it is far better for them to be calculated on the fly. In reality far less normals are computed during image computation since not all voxels contribute to the final image when adaptive termination is used. In an experiment with the CThead data of 7 million voxels, only 2 million voxel normals were computed.

14.4.2 Choosing the stepping distance

By choosing the stepping distance to be 1 unit, the samples may occur anywhere within the data set, that is to say, there are no means by which the computational cost can be reduced by taking advantage of where sample positions occur. It can be seen that since these samples can occur anywhere within the cube, trilinear interpolation for the values must be used. If the samples were to occur within the face of each cube, then only bilinear interpolation would be required. This reduces the need to know the normals at all 8 neighbouring voxels to just needing to know them at the 4 face neighbouring voxels. The stepping distance can be chosen so this situation occurs by calculating how far along the ray must be travelled to cross between successive cube faces parallel to a particular axis. By starting the ray on that face boundary, it is ensured that by using the correct step distance, each sample will be in a face, and therefore only bilinear interpolation is required.
The stepping distance is calculated by determining which axis the ray travels fastest in – that is for a given length of ray which axis it will travel furthest in. The rays are then adjusted to start in the closest face perpendicular to that axis, and then each sample will occur in a face perpendicular to the axis. Face interpolation functions are used depending upon in which face interpolation has to take place (perpendicular to the x, y or z axis). The ray is adjusted simply by scaling the stepping distance of the *fast voxel traversal* algorithm of Amanatides and Woo [24] so that the quickest axis has a stepping distance of 1 (2 division operations), and the fractional part of the sample point position is altered so that it is zero in the fastest axis, and contains the fractional placement in the other two axis (2 divisions, 2 multiplications, 2 subtractions and 2 additions). The fast voxel traversal algorithm then continues as normal.
The effect this has is two-fold. Firstly it is no longer necessary to calculate the normal from 8 neighbours, but rather from 4, halving the number of central difference calculations. Secondly, bilinear interpolation is used :

$$
\begin{aligned}
t_0 &= (v_1 - v_0)r_i + v_0 \\
t_1 &= (v_3 - v_2)r_i + v_2 \\
v_{p_j(i)} &= (t_1 - t_0)r_j + t_0
\end{aligned}
$$

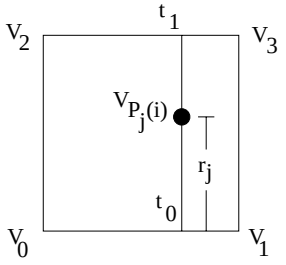

Figure 14.5: Bilinear interpolation within cube face.

Where r_i and r_j are the offsets along the two axes (Figure 14.5). This involves 3 subtractions, 3 additions and 3 multiplication operations for each value that must be interpolated, and a total of 4 voxel look ups. The total computational operations required are summarised in Table 14.2, with the old values in brackets.
It can be observed that about 50% of calculation can be saved using this method.

operation	normal	rgbα	value
+	23 (51)	12 (28)	3 (7)
-	42 (90)	12 (28)	3 (7)
$\div$	21 (45)		
$\times$	30 (66)	12 (28)	3 (7)
$\sqrt{}$	7 (15)		
voxel l. u.	24 (48)	4 (8)	8
table l. u.		16 (32)	

Table 14.2: Calculations required for bilinear sampling process.

14.4.3 Testing

The effect this has on computational time was tested by implementing the standard volume rendering algorithm using trilinear interpolation and then modifying it to calculate the required interval size, therefore allowing bilinear interpolation to be used. The algorithm was implemented on a DEC 3000 model 400 Alpha workstation and tested on the UNC Chapel Hill CThead and superoxide dismutase electron density map (SOD) data sets, and the AVS Hydrogen data set. Since the new method increases the interval size, a certain amount of time will be saved due to the fact less samples will be taken along the ray. In order to eliminate this effect the standard algorithm was modified so that it calculated the interval size, and used trilinear interpolation rather than bilinear interpolation (called Jump in tables). Taking this time into account allows the true saving to be revealed. One final point is that the interval size can vary between 1 when the viewing angle is axis aligned, and $\sqrt{3}$ when the viewing angle is at 45^o to each axis. Therefore testing is carried out at various angles to give differing interval sizes. The results appear in Tables 14.3 and 14.4.

	Standard Ray Termination				Adaptive Termination			
Angle	Standard	Jump	Bilinear	Gain	Standard	Jump	Bilinear	Gain
45,45,45	113.16	69.18	39.10	26.6%	73.18	54.30	40.10	19.4%
0,90,0	111.53	113.78	65.73	41.1%	64.20	65.53	50.50	23.4%
45,0,45	114.46	84.21	47.11	32.4%	73.73	61.93	35.77	35.5%

Table 14.3: Computation times for Hydrogen data set.

	Standard Ray Termination				Adaptive Termination			
Angle	Standard	Jump	Bilinear	Gain	Standard	Jump	Bilinear	Gain
45,45,45	365.85	276.57	141.71	36.9%	96.88	84.40	54.61	35.3%
0,90,0	401.67	448.60	244.36	39.2%	103.23	112.50	74.18	28.1%
45,0,45	344.49	351.67	183.01	46.9%	98.68	100.98	64.68	34.5%

Table 14.4: Computation times for CThead data set.

The normal calculations dominate the volume rendering process, and by reducing the number required and simplifying the interpolation process, substantial savings can be obtained as indicated by the tables. In fact the computation time has been reduced by about 30%–40%. This reduction is to be expected since the amount of computation done during normal calculation is reduced by 50% and also the fact that normal calculation requires a large

proportion of the running time when compared to the constant calculations that are made, such as calculating ray entry and exit points to the volume.
It is interesting to note (from the Jump column) that by increasing the step size, computational time also increases in some cases. This is because the step size is not significantly greater than 1, and some extra calculations involved when determining the number of samples outweigh any saving.
The method gives a better gain for the normal ray terminating method because a higher percentage of the computation time is spent calculating normals, for which the method offers a 50% speedup.

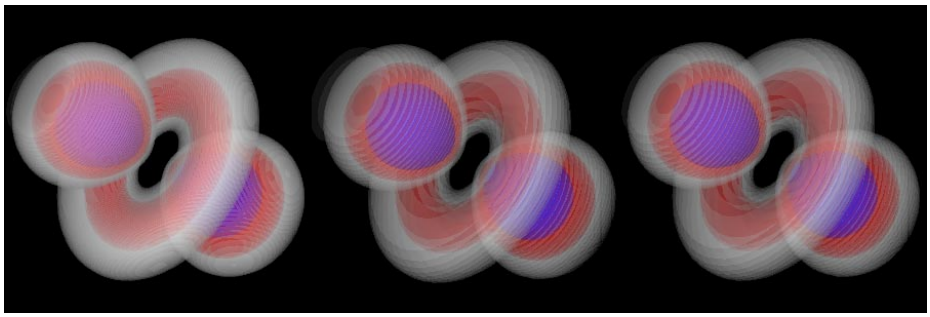

Figure 14.6: (a) Trilinear interpolation, (b) Increased interval size, (c) Bilinear interpolation.

Examining the image produced shows very little degradation in quality. This is to be expected since the samples are still being taken at evenly spaced intervals and are taken in such a way that all values along a ray are used in the sampling process. There is no difference between images 14.6 (b) and 14.6 (c), since both have the same interval size. There is a difference between images 14.6 (a) and 14.6 (b) which have been produced with an interval size of 1 and 1.732 units respectively. This difference is caused by samples being taken with larger intervals using the same compositing formula:
$o_i^{out} = (1 - o_i)o_i^{in} + o_i$ for $i = 1, \ldots, n$
where n is the number of samples, o_i is the opacity at sample i, and o_i^{out} and o_i^{in} are the opacities coming into and going out of i. The interval length does not come into this formula, and therefore in a volume of constant value, n samples with an interval length of m will give the same opacity for n samples with an interval length of 1.
If $m > 1$ the volume sampled with the larger interval size will seem more transparent than the volume traversed with an interval size of 1. This is quite acceptable since the opacity values are a subjective scheme to allow the presentation of the data. For static images there is no problem.
Figure 14.7 shows the resulting images for the CT head test data set.

14.4.4 Animation

As mentioned in the previous section, increasing the interval size, increases the transparency of the volume. This does not cause problems for static images, but for animation, where the viewpoint from which the data is seen from varies, the interval size is variable and during a rotation, the transparency may be perceived to rise and fall. A practical solution is to raise each value in the opacity classification table to some power proportional to the step size. Figure 14.8 gives two images, with differing interval lengths, produced using the modified classification table. As can be seen, the transparency of the images corresponds quite closely.

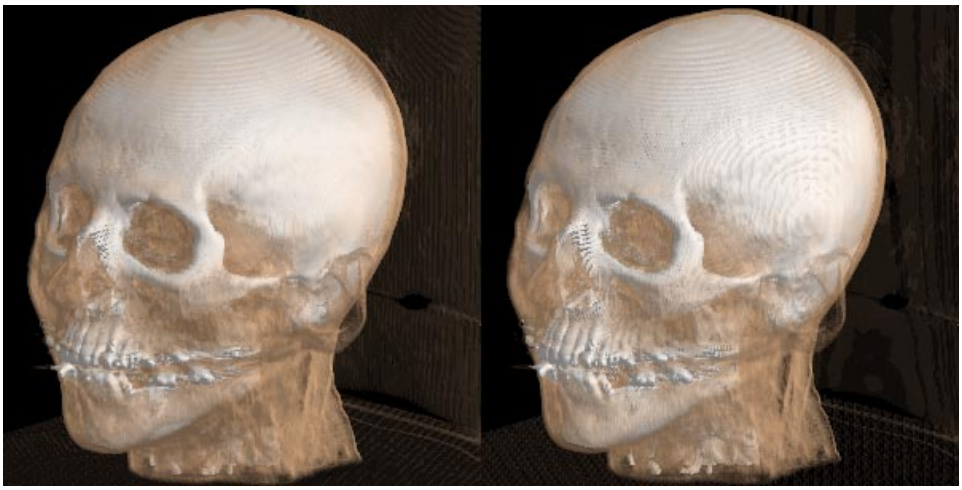

Figure 14.7: (a) Trilinear interpolation, (b) Bilinear interpolation.

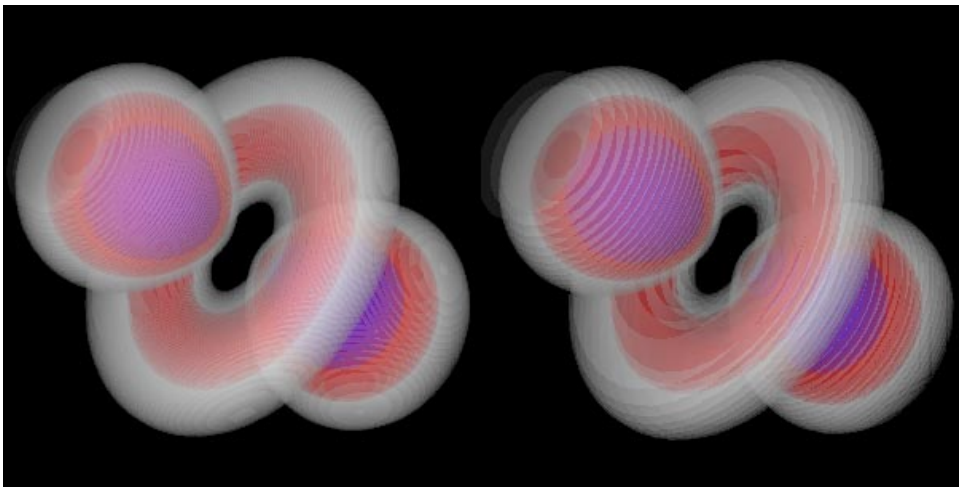

Figure 14.8: (a) Trilinear interpolation, (b) Bilinear interpolation.

14.4.5 Discussion

By analysing exactly how many operations are carried out during volume rendering, it has been possible to concentrate on a method that reduces these operations by about 50%. This substantial reduction has been achieved through the use of a variable interval size which is dependent upon viewpoint, and is calculated in order to make each sample fall within a cube face. This allows bilinear, rather than trilinear, interpolation to be used, with the associated reduction in computational complexity. This saving has been analysed both qualitively and also through the effect it has on the execution times of the volume rendering process. Since normal calculation is the main burden of volume rendering, a reduction by 50% of the operations required should result in a large reduction in computation time. This reduction is observed to be around 30% and reasons for this were given in Section 14.4.3. The images produced by the method are shown to be good representations of the volume, and problems that arise during animation where the viewpoint changes have been investigated. A practical solution to avoid these artifacts has been given, and testing shows it to be acceptable. This method compares well with other acceleration techniques (Section 14.5) since it gives a similar reduction in time without the large trade off in image quality that other methods suffer. This method involves very little computation to determine the stepping distance, and suffers from no constraints on the viewing direction and view point.

14.5 Acceleration Techniques

The volume rendering process has been identified as being very costly in terms of computation, and many methods exist that accelerate the computation time. This section reviews several acceleration techniques.

14.5.1 Adaptive rendering

Most of the expense involved in volume rendering is the fact that so many samples are taken along each ray during the compositing process. Since the number of rays is dependent upon image size, the larger the image, the more expensive it becomes to render the data set. Adaptive rendering attempts to reduce the workload by concentrating computation in areas where it is most needed. This technique can be applied quite simply to volume rendering by assuming the resultant image will contain large areas of coherency (colour and intensity). The principle is to ray trace the volume at a low image resolution and by treating four neighbouring pixels which have been ray traced as corners of a square the remaining pixels can be coloured using bilinear interpolation. This coarse image can be regarded as the first image in a sequence of images that progressively become more and more refined. The refinement process takes place by adaptively concentrating on areas with greatest change. If the pixels at the corners of the square vary by more than a given tolerance ϵ, the square is divided into four smaller squares and the new unknown corner points are computed. The process of computing the interior pixels using bilinear interpolation is repeated, and a new image can be displayed. This process is carried out until the corners of the square vary by less than ϵ or the size of the square is one pixel. At this point the image is correct to a tolerance of ϵ. The effect of this method is that computation is concentrated in areas where features change sharply, for example along the edge of an object. By relaxing the tolerance, less pixels are truly sampled, but more image defects become apparent, and so there is a trade off between image quality and speed. This method was also presented in [25, 20]. The image

in the middle of Figure 14.9 is a volume rendering of the hydrogen data using the adaptive rendering method with squares of an initial size of 16, and a tolerance of $\epsilon = 0.0275$ (or a difference of 7 intensity values out of 256). The image on the right shows the pixels that were actually ray traced, the rest being interpolated from these. The number of rays traced is 34335, as opposed to 160000 for the image produced by the standard method on the left. Whereas the standard method took 129.4s, the method using adaptive rendering took 53.0s. All images are 400 $\times$ 400.

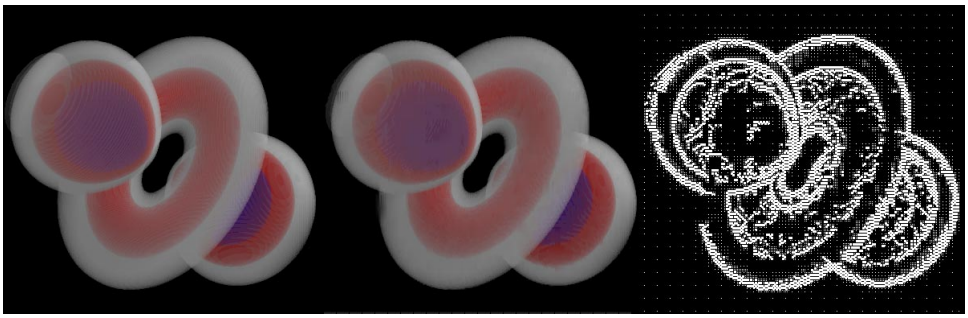

Figure 14.9: The adaptive rendering process.

14.5.2 Template-based volume viewing

In Section 14.4 it was shown that the process of trilinearly interpolating sample positions and function gradients along the ray is the most computationally intensive component of the volume rendering process, thus any method that avoids this will reduce the computation required. The template based method of Yagel and Kaufman [23] calculates a ray template which is a path of voxels through the volume. This template can be moved over the image construction plane in such a way that the volume is sampled uniformly without gaps (Figure 14.10). The ray template indicates which voxels are to be sampled, and these voxels are sampled without trilinear interpolation, with their gradients calculated using central differences. This is valid if it is assumed that the value in the volume does not vary over the cube represented by one voxel, which is not generally the case. The template is constructed so that it is either 6-way or 26-way connected. In this case 26-way connection is chosen because it results in less samples and allows the volume to be sampled uniformly without repetition. The image is formed by projecting and resampling the image construction plane to the desired dimensions.

14.5.3 Adaptive termination

The original volume rendering algorithm composites colour and opacities in a back to front manner. If at any time a fully opaque sample is encountered, the samples composited up to that point have no bearing on the final pixel colour since they are obscured. This is an undesirable situation since expensive interpolation and gradient operations are effectively ignored and do not make any contribution. In order to prevent this, the ray should be traced in a front to back manner so that in the event of an opaque sample, the ray can be terminated, and so no further samples need to be taken along the ray [26].
Using the definitions from Section 14.2.2 the front to back algorithm for each pixel **p** is

$C = Background$
$o = 1.0$

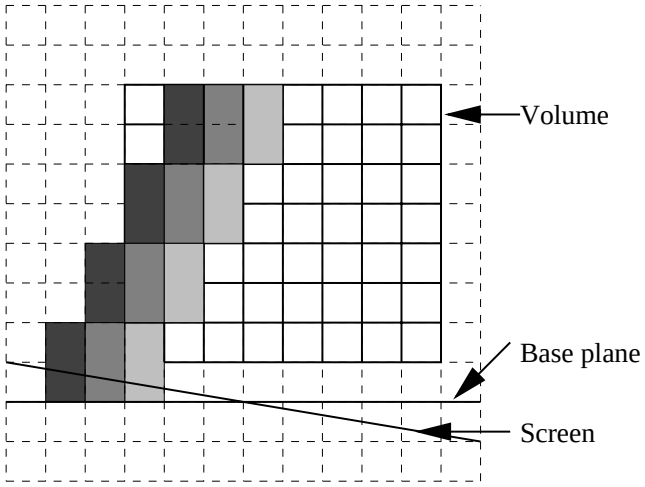

Figure 14.10: Using template to sample the volume.

$n = 1$
while $(n < K$ and $o \neq 0.0)$ **do**
$C_r = C_r + o \times g(\alpha, R_p(n)) \times g(r, R_p(n))$
$C_g = C_g + o \times g(\alpha, R_p(n)) \times g(g, R_p(n))$
$C_b = C_b + o \times g(\alpha, R_p(n)) \times g(b, R_p(n))$
$o = o \times (1 - g(\alpha, R_p(n)))$
$n = n + 1$
endwhile

The front to back compositing function is slightly more complicated, but also allows the use of adaptive termination to stop the rendering process when a useful image has been calculated. In the compositing function it is observed that as the accumulated transparency o approaches 0, the sample taken does not contribute significantly to the final pixel colour [26]. In fact the contribution is less than o, so for $o < \epsilon$ where ϵ is some small tolerance level, we can *adaptively terminate* the ray no matter how far it has travelled through the volume. Since the opacity along the ray reaches high values very quickly, when passing through slightly opaque solid matter, the ray is terminated long before it has passed through all of the volume, and thus large amounts of unnecessary computational effort can be saved. If the tolerance ϵ is relaxed rendering time is reduced although the image contains more artifacts.

14.6 Comparison of Methods

For the comparison of all the different methods, three test data sets were used – AVS Hydrogen, University of North Carolina (UNC) CThead, and UNC superoxide dismutase electron density map (SOD). Each test program was written in C on a DEC Alpha 3000/400 workstation using as much common code as possible to make timing comparisons fair. Each data set was rendered from a particular viewpoint under certain conditions for an image size of 400×400 to give the time. Such a large image was chosen in order to allow the differences between the methods to be more marked. The images were compared qualitatively (how they looked). The results for the CThead data are given in Table 14.5, results for the Hydrogen data are given in Table 14.6, and for the SOD data in Table 14.7.

Method	Adaptive Termination	Figure	Time (secs)
Standard	No	14.11(a)	731.37
Standard	Yes ($\epsilon = 0.1$)	14.11(b)	190.01
No shading	Yes ($\epsilon = 0.1$)	14.11(c)	81.09
Template	Yes ($\epsilon = 0.1$)	14.12(a)	10.10
Bilinear	Yes ($\epsilon = 0.1$)	14.12(b)	98.25
Jumps	Yes ($\epsilon = 0.1$)		140.93
Bilinear	No		292.64
Jumps	No		503.63
X-ray	N/A	14.12(c)	97.66
Maximum	N/A	14.13(a)	99.71
Depthmax	N/A	14.13(b)	106.23
Sabella	N/A	14.13(c)	133.61

Table 14.5: Results of different methods using CThead data.

For the CThead (Figures 14.11–14.13) there is no qualitative difference between any of the images using the standard viewing model. From these comparisons it would seem that the bilinear method with adaptive termination is the one that produces the best results quickly for this data set, being over 7 times faster than the method without any optimisations. An animation loop was created for the CThead (see accompanying video) using the adaptive termination technique, with and without the adjustment of the step size to enable bilinear interpolation. Using the normal method, the animation took 2hrs 6mins 12secs to produce, as opposed to 1hr 16mins 6secs with bilinear interpolation. There is no visual difference between the two which would suggest that the bilinear method can be used where high quality accurate images are required using the least computational time.

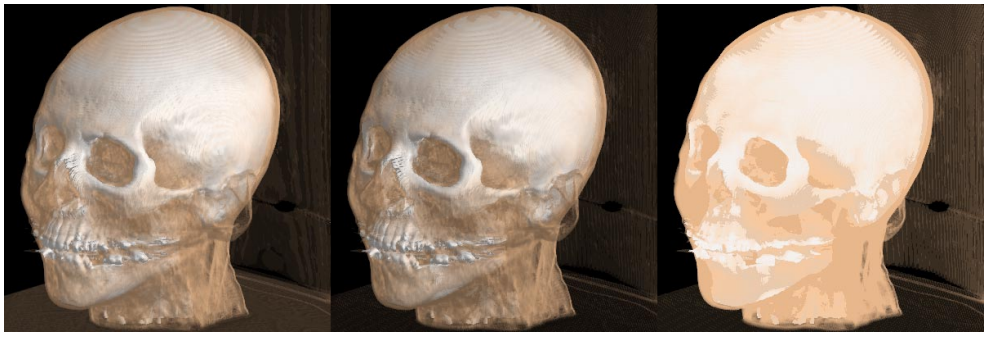

Figure 14.11: (a) Standard method (b) Adaptive Termination (c) No shading.

The template method is by far and away the fastest method – taking only 10 seconds for the test data, but produces a very coarse image. The image is adequate to gain a rough idea of the data and is ideal for quick visualisations, since it shows the main features. The problem with this method is that the time is not scalable, that is, it is constant for all image sizes, unlike the other methods, which are scalable. It was found that a 100×100 image created using the bilinear method, and scaled up took half the time of the template method to compute, and produced a comparable image, the only difference being the fact that the bilinear image was slightly smoother than the template image, and contained less sharp delineations of features which had provided good indicators in the case of the template method.
The x-ray, maximum, and maximum with depth cue methods produce alternative views of

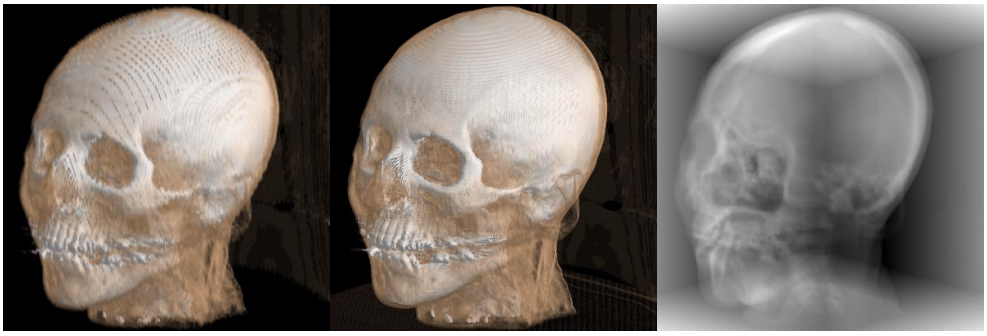

Figure 14.12: (a) Template method (b) Bilinear method (c) X-ray method.

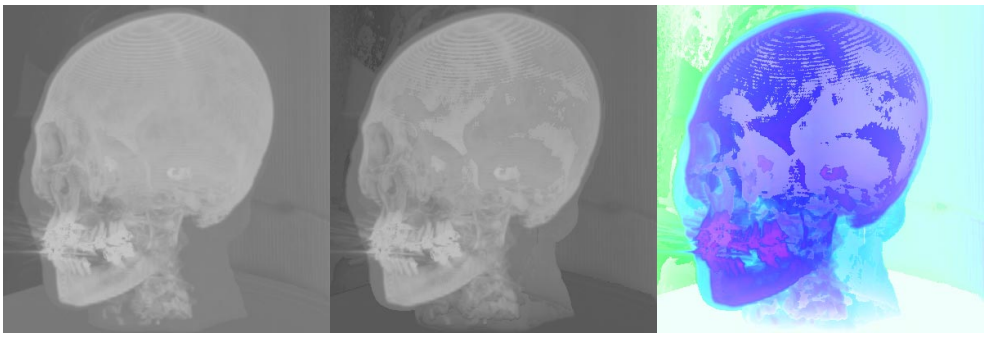

Figure 14.13: (a) Maximum method (b) Maximum with depth cue (c) Sabella's method.

the data in reasonably quick times. The maximum and x-ray methods produce similar images in much the same time. When using the maximum method to produce animations, the head looks as if it is swinging from side to side, rather than rotating, due to the fact that images 180^o apart are identical. This situation was somewhat improved by using the depth cueing method, and produces a better animation.

Finally the Sabella image of the skull is interesting, but it can only be used to show how the image differs from the standard methods. The image it produces in the case of the CThead serves no real purpose since the method was not intended for use on such data sets.

For the Hydrogen data set (Figures 14.14–14.16) the image produced by the Sabella method shows the data distribution using colour. Fogginess shows depth, and together they convey a lot of information, showing the structure of the data. The image looks good, but it needs effort to interpret the information.

The x-ray, maximum and maximum depth images all look very similar, and show the two *hot spots* and a general fuzziness around the toroidal section.

The only difference between the adaptive method and the standard method is that the adaptive image looks duller because brighter areas to the back of the volume are not contributing, due to the ray terminating. It was found that by making the error bound ϵ, tighter, this effect is reduced, and the time to produce the image increases. It was found that for $\epsilon = 0.05$ the image produced by the adaptive method looks identical to the image produced by the standard method and took 146.28s, a saving of 25%. The template method produces a coarse image which looks very fuzzy. It gives a rough idea of the data spread, but the image is weak and dull. This is due to the fact that less samples are taken along the ray, and therefore less light is contributed to the final intensity. The method without shading also produces the image quickly and gives a good impression of the data, although the lack of shape information from the shading detracts from the image. The bilinear method produces good clean images, but

Method	Adaptive Termination	Figure	Time (secs)
Standard	No	14.14(a)	202.32
Standard	Yes ($\epsilon = 0.1$)	14.14(b)	130.94
No shading	Yes ($\epsilon = 0.1$)	14.14(c)	30.67
Template	Yes ($\epsilon = 0.1$)	14.15(a)	1.95
Bilinear	Yes ($\epsilon = 0.1$)	14.15(b)	55.00
Jumps	Yes ($\epsilon = 0.1$)		104.45
Bilinear	No		69.88
Jumps	No		131.88
X-ray	N/A	14.15(c)	26.76
Maximum	N/A	14.16(a)	25.98
Depthmax	N/A	14.16(b)	29.87
Sabella	N/A	14.16(c)	36.62

Table 14.6: Results of different methods using hydrogen data.

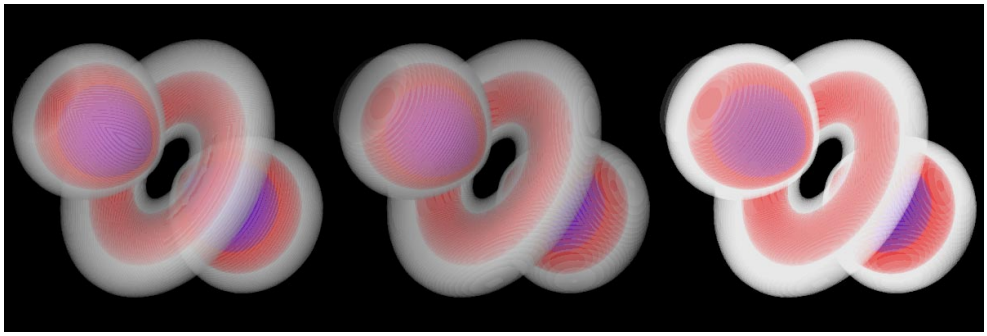

Figure 14.14: (a) Standard method (b) Adaptive Termination (c) No shading.

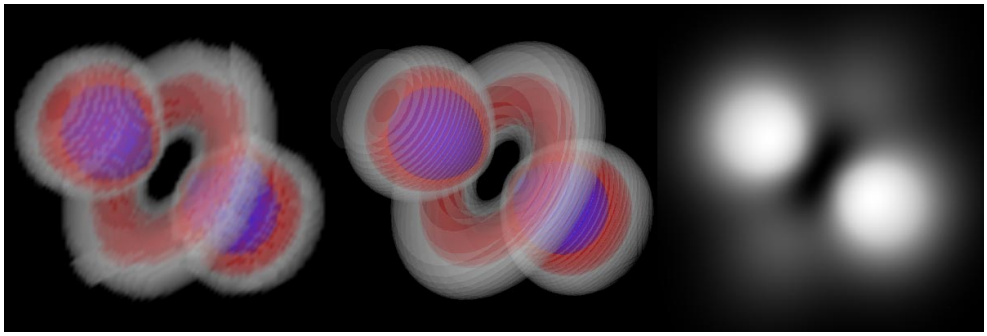

Figure 14.15: (a) Template method (b) Bilinear method (c) X-ray method.

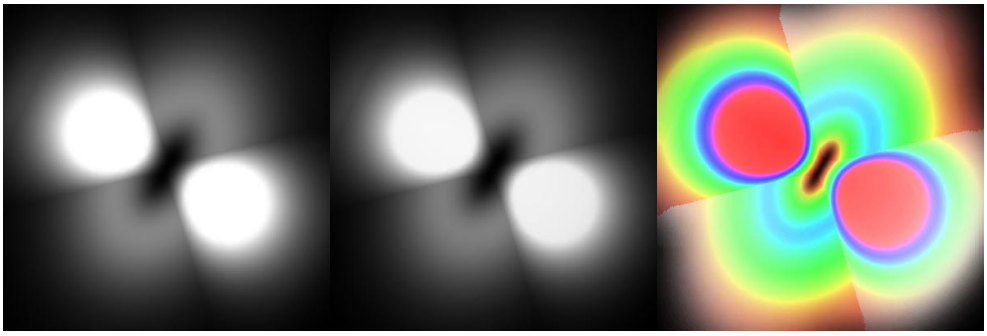

Figure 14.16: (a) Maximum method (b) Maximum with depth cue (c) Sabella's method.

they are slightly more transparent, and therefore dull, because less samples are taken along the ray. Nevertheless, the images are still very useful.

Method	Adaptive Termination	Figure	Time (secs)
Standard	No	14.17(a)	108.80
Standard	Yes ($\epsilon = 0.1$)	14.17(b)	99.95
No shading	Yes ($\epsilon = 0.1$)	14.17(c)	56.01
Template	Yes ($\epsilon = 0.1$)	14.18(a)	3.37
Bilinear	Yes ($\epsilon = 0.1$)	14.18(b)	51.68
Jumps	Yes ($\epsilon = 0.1$)		73.55
Bilinear	No		49.81
Jumps	No		72.98
X-ray	N/A	14.18(c)	43.79
Maximum	N/A	14.19(a)	42.44
Depthmax	N/A	14.19(b)	47.83
Sabella	N/A	14.19(c)	59.66

Table 14.7: Results of different methods using SOD data.

This problem also occurs with the SOD data (Figures 14.17–14.19), but the bilinear method results in an image that is even more weak and dull. Any impressions of curvature are lost as shading is not enforced over large coherent areas. In this case the method produces useful images, but they do not contain as much information as some of the other models and methods.

The image produced by the template method suffers even more from dullness and is very fuzzy. The method is still useful though, because it is so fast, and could be used to produce animations quickly to gain an overall impression of the data.

The image produced by the standard method with adaptive termination is duller than images produced without, and in this case the saving in time is not so significant, and in conclusion not worth the slight degradation in image quality. A good impression of the hot spots contained within the data can be gained through images produced without shading, also through images produced using the x-ray, maximum and maximum with depth cueing methods, which also bring out the blobby structure of the data quite well.

Finally the image produced using the Sabella model produces what is arguably the best image, with a clear indication of hot spots, with a good feel for the lumpy structure of the data, conveying the depth of the data well.

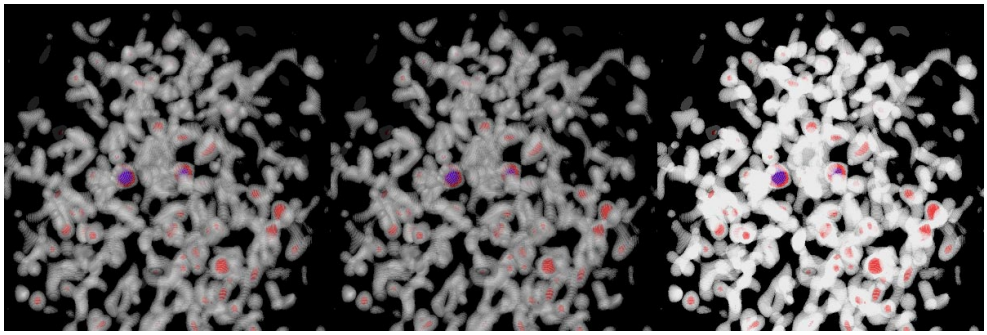

Figure 14.17: (a) Standard method (b) Adaptive Termination (c) No shading.

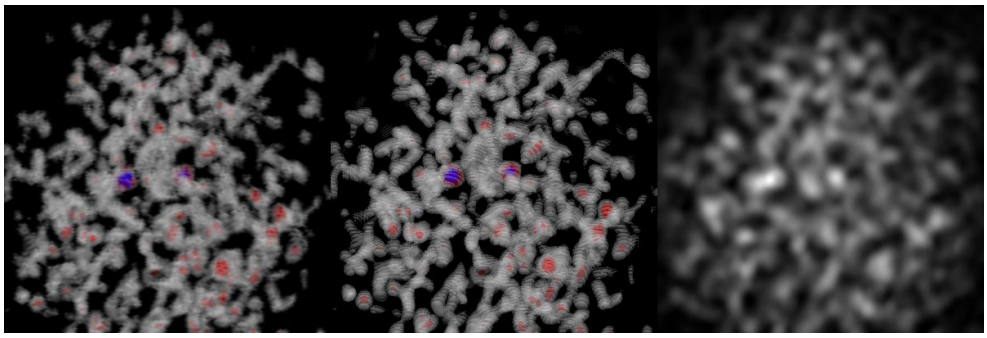

Figure 14.18: (a) Template method (b) Bilinear method (c) X-ray method.

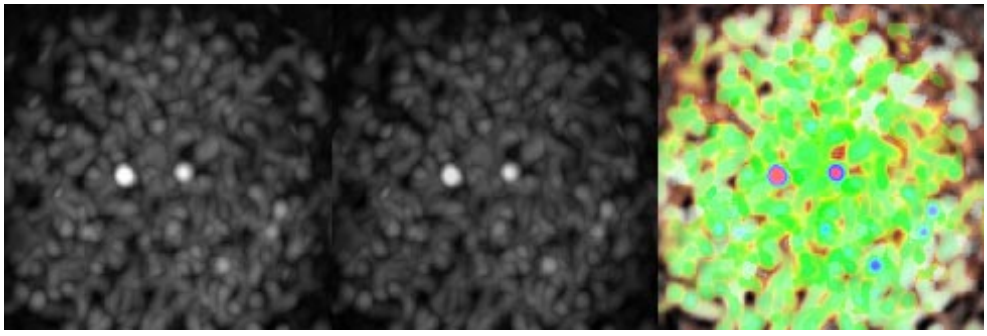

Figure 14.19: (a) Maximum method (b) Maximum with depth cue (c) Sabella's method.

14.7 A Review of Other Volume Rendering Techniques

The main viewing models available for the rendering of 3D data sets have been described and compared in Sections 14.2 and 14.3. The more important acceleration techniques have been reviewed in Sections 14.4 and 14.5, with comparative tests made about their speed and image quality in Section 14.6. This section is a review of the other techniques and systems that are available for the visualisation of volume data.

14.7.1 Splatting

One large area that has been focussed on is that of projecting the volume data set onto the image plane to produce the image. This differs from previously described techniques in that rather than a pixel by pixel traversal of the image, and a mapping from image space to object space, the volume is traversed in a cell by cell order with a mapping from object space to image space.
In footprint evaluation [27], Westover suggested projecting each sample within the data set onto the image plane, and *spreading* the energy of the sample according to its *footprint*. The footprint is obtained by applying the view transformation to a generic footprint which can be calculated by assuming that the reconstruction kernal is a sphere. The kernal's footprint is determined, and the energy spread is calculated according to some function, such as a Gaussian. The relative merits of footprints of various sizes (5×5 to 101×101) are described in the paper. The advantage of this method is the fact that each sample is projected onto the image and rays no longer have to be traced into the volume to integrate samples to find their contribution. This allows the application of parallel processing techniques since each processor can operate on a subset of the data rather than requiring access to the data as a whole.
This work was extended by Crawfis and Max [28] where the authors present an ideal reconstruction formula. Their goal was to produce a formula which produced splats with smooth densities, so that the structure of the individual splats was not visible. Their example is that of a cube of $n \times n \times n$ voxels, each emitting an intensity of 1, and having no opacity. The projection of such a cube should be as constant as possible. The function they calculate produces a variation of at most 1 part in 256. They also describe how to map textures onto the splats with the use of hardware texture mapping (Silicon Graphics Rendering Machines and Iris Explorer) to produce animations, and motion blurred images of 3D vector fields.
The work is also extended by Laur and Hanrahan [29], this time by approximating splats with Gouraud shaded polygons. The work they present is concerned with the production of real time rendering for interactive applications. Real time rendering is achieved by encoding the data as an octree, and projecting nodes of the octree using the splatting technique. Each node in the tree is associated with an error, the idea being that the node with greatest error is the node to be divided next in the refinement process. Using this method the user would be able to move a low resolution representation of the object in real time, which would be progressively refined whenever the user paused to consider the image.

14.7.2 Curvilinear grids

One aspect of volume rendering that does not receive so much attention is that of the display of curvilinear grids. The grids usually arise as the result of computational space being warped around an object of interest. A typical example is a computational fluid dynamics simulation of an aircraft wing, where the mesh has a much lower resolution away from the wing, than

in the vicinity of it. Distances between neighbouring points can vary by a factor of 10,000 over the whole data set. Conventional ray casting methods run into difficulty since rays can enter and exit through any cell face, and also re-enter cells (Figure 14.20). Max et al. [30] went some way to addressing this problem by dividing non-convex cells up into convex cells and non-convex cells which are guaranteed to have only one ray span intersection for a given viewpoint.

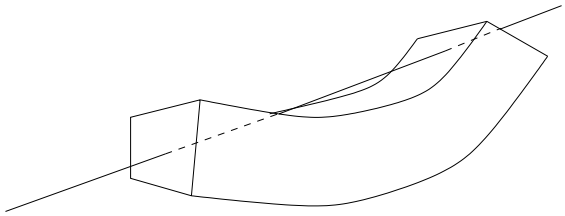

Figure 14.20: Ray can exit and re-enter curvilinear cells.

Alternatively, the large curvilinear grids could be resampled by a regular grid, and then ray casted. This method is problematical, since resampling would have to be carried out with a small step size to retain accuracy. This would result in a large increase in the volume of data. A larger step size would not result in such an increase in the amount of data, but has the problem that fine details in the data would be lost.
These problems can be avoided by using projection methods [8, 31, 32, 33]. Wilhelms and Gelder [31] show how accurate integration and composition can be performed on cells, in order to project the cells correctly onto the image plane. Wilhelms [33] also compares the projection and ray casting methods and shows that a speed up of 50–150 times is quite reasonable. A comparison of the different methods for projection was made by Williams [32], in which various methods are described for the projection of tetrahedral meshes. The order of traversal presents a problem, but once solved [34, 35], cells can be projected easily, usually in the form of hardware Gouraud shaded polygons.
It is also worth mentioning the work presented by Frühauf [36], which essentially solves many of the problems associated with the ray casting of curvilinear grids. In this paper the author shows how computational space can be unfolded into a regular grid. The same transformation can be performed on the light rays so that instead of sampling along straight rays in curvilinear space, samples are taken along curved rays in a regular computational space. This solves the problems of finding neighbours, and the ray/cell intersections. The ray direction is stored at each node as a vector, and some *particle tracking* technique, such as Runge-Kutta, is used to determine where the next sample occurs along the ray. Results show that very little error creeps into the ray paths, which therefore allows the production of accurate images.

14.7.3 Data compression

The data sets requiring visualisation are often very large. In fact they are usually larger by an order of magnitude, than data sets that can be handled easily. A typical data set can be around 16 MBytes. Techniques which reduce the volume of data can often have the advantage that images are also quicker to produce [37, 38].
Udupa and Odhner [37] extend their data model [39] to cater for volume rendering. In their method they assume (for a large reduction in data and image rendering times) that the opacity functions (of Section 14.2.1) have been defined using gradient operators such that voxels

away from object boundaries contribute very little, if at all, to the image. This allows them to define a data structure which encodes the data as a *shell* around an object boundary. The shell is then projected using a back-to-front or front-to-back method, and since only voxels within the shell rather than all the voxels in the scene domain are projected, large reductions in rendering time are achieved. They report shell rendering is nearly 1000 times faster than a straight forward implementation of a standard ray casting method.
Volume compression was the focus of the work of Ning and Hesselink [38], where data is replaced by indices to a codebook. Using vector quantization, the authors use nearest neighbour mapping of every k-dimensional input vector X to some vector X_i selected from a finite codebook of candidate vectors. The original data can be replaced by the index i, which allows the data to be compressed. They chose the voxel value, normal and gradient magnitude for a block of b^3 voxels as the vector to be compressed, and found that a compression ratio of 5:1 could be achieved for practical data sets. The rendering times of the data set increased by 5% due to the overhead of checking a table entry to uncompress the data. They also show how faster rendering can be achieved by rendering the codebook entries from the viewpoint and compositing the result along each ray. Since the codebook has only a few entries, rendering of the codebook entries takes negligible time. The rendering of the data is now reduced to a composition of the codebook renderings that occur along the ray which avoids the need to perform costly trilinear interpolation and shading calculations at each sample along the ray. A speed up of about 10:1 was achieved using this method of volume rendering, although there is loss of image accuracy since the codebook is only representative of the data, and not all possible ray entry and exit points for a given view can be catered for.

14.7.4 Spatial techniques

Spatial techniques seek to explore coherency within the data to accelerate rendering. An additional data structure is constructed using the data which can then be used to accelerate the rendering, usually by allowing rays to skip over portions of the volume that are uninteresting. The most popular spatial subdivision technique is that of the octree [40, 41], which was applied to volume rendering by Levoy [26]. In a preprocessing step an octree is constructed where each node is divided if it contains a voxel which has non-zero opacity. Each divided node branches into eight subtrees, one for each of the subvolumes constructed when the volume is divided in half in the x, y and z directions. The ray is traced through the octree, using information within the nodes to jump over large areas of non-contributing (zero opacity) voxels. Most of the computation during volume rendering is the trilinear interpolation of sample values along the ray from surrounding voxels. By knowing in advance that all values within a subvolume are going to produce an opacity of zero which would result in no contribution to the image, the ray can safely skip over that subvolume to the next sample point. This would avoid the costly sampling process for all samples along the ray within that subvolume, resulting in a reduction of computational cost and therefore time. Although a speed up was reported by Levoy [26], it was shown that this was not the case [1]. In addition to this problem, any change of opacity function would require the octree to be recomputed.
The fact that the octree does not speed up ray casting is also mentioned by Yagel and Shi [42]. The authors suggest that the additional cost of tracing the ray through the octree can be avoided by storing the octree information at the empty voxels as uniformity information. Using this process, empty voxels are assigned a value which represents which level of the octree they are in, which can then be used to cause the ray to leap forward to bring it to the first voxel beyond the uniform region.

Other methods are discussed by Yagel and Shi [42] in which either an additional volume, or the data volume itself is used to store proximity flags or values. The idea behind this is to place a shell of flags, or shells of values representing distance, around the object which will enable an efficient ray tracking algorithm to switch from an efficient space jumping rapid traversal algorithm, to an accurate ray traversal algorithm. The proximity flags would indicate a surface was near, or more efficiently, the shells of distance values would indicate how far a ray can jump without encountering an object. These methods suffer from the fact that 3D preprocessing is required that must be repeated for any change in the data.
Yagel and Shi [42] also show how a method retaining starting depths for rays can accelerate ray casting of successive images in a rotation by avoiding the need to calculate a traversal of empty voxels in front of the object. They show how to avoid some of the problems associated with rotated objects covering objects previously visible and report a speed up of 2 to 6 times. It should be noted that this speed up applies to ray traversal, and since in many applications it is the ray compositing step that is the most time consuming (see Section 14.4.1), a significant speed up in the volume rendering process may not be observed.
Spatial techniques are also used by Subramani and Fussell [43], in order to store interesting voxels in a $k-d$ tree. This tree is obtained by using a median cut space partitioning algorithm to efficiently divide space up so that either branch, at any one time, removes a similar number of voxels. Bounding volumes are also used at nodes to efficiently encapsulate the voxels within each subtree. This leads to an efficient subdivision of space – the $k-d$ tree, which can be efficiently ray traced. The authors report a reduction of over 90% in the amount of data that needs to be considered in order to produce the image, although they do not discuss rendering acceleration times.
One other acceleration method is the polygon assisted ray casting (PARC) technique of Avila et al. [44]. The object is simply approximated by polygons which can be projected onto two depth buffers – a front projection and a back projection. These two depth buffers will contain the start and end points for each ray of the image, and in this way the ray does not traverse parts of the volume that do not contribute to the final image. The problem with this method is that of how to choose the polygonal representation of the object. This can be achieved by using the faces of subvolumes of the data which contain a surface, and merely requires the decision of how fine the subvolumes should be. The method has the advantage that costly unnecessary ray sampling is avoided, and therefore produces images with less computation.

14.7.5 Shading techniques

The problem of interactive shading on less powerful graphics workstations was addressed by Fletcher and Robertson [45]. Interactive shading is desirable, since it allows researchers to move light sources around their data in real time, thus giving a better understanding as to the three dimensional nature of their data. Another benefit is the experimentation of light source positions during the production of animation sequences without the need of re-rendering each frame. Interactive shading is achievable by computing a restricted table of n normals (where usually $n = 256$). The normal calculated for each pixel in the rendered image, is then mapped to the normal in the table that has the closest matching direction to it. In recalculating the image with different shading parameters, only the intensities resulting from shading the normals within the table need to be computed. These values can then be mapped back onto the image to produce the new image. This results in a substantial reduction of computation – to a level most graphics workstations can handle. The main problem is the fact that only a restricted subset of normals can be reproduced accurately. One example image [45] is that

of a hemisphere which has been shaded with this technique, and has resulted in a hexagonal faceted image, each hexagon centred about a known normal. Their solution is to dither the normals using a process similar to the Floyd-Steinberg algorithm for dithering. They report images can be shaded at rates of 47 to 139 fps on a DEC 5000/200.

14.7.6 Volume seeds

Ma et al. [46] present a method in which the user is given interactive control of the final volume rendered image. The user can alter variables such as the opacity table and voxel colours in real-time through the use of a *cache*. The idea is that the volume is ray traced from the given viewing direction, and the value of each sample along the ray is stored in a three dimensional array. This array is simply traversed, and voxel values are looked up in the classification table in order to perform the compositing step. Since costly trilinear interpolation of the sample values is no longer required, real-time compositing of the image is achievable.
The user is also given control of the *volume seeds* technique. The user can place a seed anywhere in the volume with the effect that voxels close to the seed are enhanced, and those further away are made more transparent. The example given [46] is that of placing a seed in the throat to make the spine in the neck area easier to see. The effect is achieved by increasing the opacity of close voxels, and reducing the opacity of further away voxels using a simple function. Real-time control is achieved by combining the seed placement with the volume caching to give users full investigative control over their data.

14.7.7 Systems and environments

Techniques can be regarded as the precursors to fully fledged systems that provide environments for data visualisation. Commercial examples would be those of AVS, Data Explorer and the public domain Khorus packages. These all work on the data driven flow model and are based on the linking of modules to form networks (Figure 14.21).

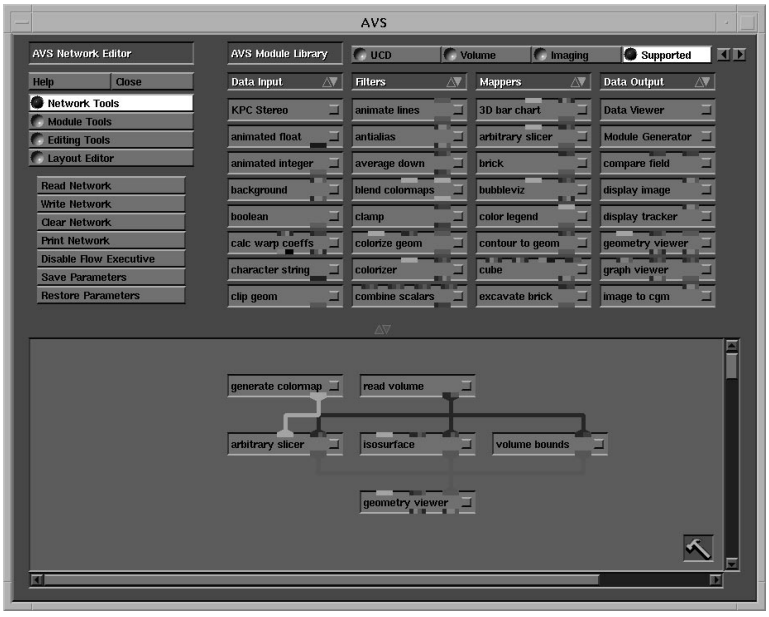

Figure 14.21: Network editor and module palette from AVS^{TM}

The commercial packages tend to be well known, and widely used, but very often it is the *homegrown* software that incorporates innovative features.

A simple but effective system is reported by Corrie and Mackerras [47]; the authors point out that one of the major benefits of their system is its flexibility in producing images over systems, such as AVS, which limit the user to a number of fixed rendering techniques. This flexibility is achieved by using a shading language, rather than a user interface. It is therefore flexible and extensible. The example visualisations and code shown in the paper [47] are for the standard shading model for volume rendering [6, 8], Sabella's model [7] and a propriety shader for the Dominion Radio Astrophysical observatory. The authors report that allowing a higher level shading language does result in a loss of efficiency when it comes to computing the images, with the example that their implementation of standard volume rendering takes about 1.7 times as long to render an image.

Montine [48] takes a low level approach to the task of providing a visualisation language and provides just the basic functions for visualisation. These functions are classified as to which object they operate on (volume – the original data, environments – lighting conditions and viewing parameters, and images). To visualise a set of data the user must provide the appropriate calls to all the functions. This requires low level knowledge from the user as to what the available functions are, but in this way provides the user with a powerful environment.

Smart particles are introduced by Pang and Smith [49] as a way of providing a framework for visualisation. Using existing techniques usually associated with 3D data sets, such as isosurfacing, particle tracking and volume rendering, the authors endeavour to show how an effective, but easy to use system can be made using the SPARTS (smart particles). The sparts are sprayed into the data set in much the same way as paint is sprayed from spray cans. Sparts can be of several forms such as surface seeking, volume penetrating, or flow tracking. Sparts can also operate on the data, and leave messages for other following sparts. The authors indicate that their system allows the user the flexibility of exploring their data using a simple and intuitive tool – the spray can. It is an effective tool since complex visualisations can be built up by using different combinations of sparts.

An example of a visualisation environment actually being used would be that of Yoshida et al. [20] in which the authors demonstrate the effectiveness of their system in support of neurosurgical planning (CliPSS – Clinical Planning Support System). They show the simple step by step process a user follows to produce data which can be used for surgical planning. Each slice is edited interactively to remove unnecessary data, such as the bed. The surgeon can use an extraction function to extract the volume of interest (VOI) by defining two threshold parameters and a seed. CliPSS then uses region growing to extract the VOI by grouping voxels together until a boundary is found. Boundaries are distinguished by moving a $7 \times 7 \times 7$ filter across the volume, calculating statistics locally from the voxels. The surgeon is given the option of rendering the VOI as a surface (isosurfacing) or as a volume (volume rendering). Surgical planning is supported by using a stereotactic frame which fits over the head of the patient. Using this frame a correlation between the computer model and patient can be made. The surgeon can plan entry points, direction and depth to probe for a target lesion on the computer. The surgical instruments can be attached to the stereotactic frame and positioned accurately over the lesion. The surgeon then knows the direction and depth to which he must probe. Using this tool operations are made much safer for the patient.

14.7.8 Anatomical atlas

Karl Heinz Höhne and his colleagues at the University Hospital Ependorf, Hamburg, are working on a project to segment and label the whole of the human anatomy [50, 51, 18, 16, 17]. Their main aim is to provide a hypermedia system for anatomical study, enabling students to contol their exploration of the body. They used their generalised voxel model [50] to store the available information about each voxel – its grey level value as obtained from CT or MRI, and its membership to parts of the body, or functional areas or a lesion. In addition to this model, and the graphical capability of the system, is a knowledge base which encodes the relationship between all the different regions (for example of the brain). The method by which the model is constructed, and the functionality of the system is described in [18] with additional information about producing the images of blood vessels in [51]. In order to isolate each entity within the brain, they first perform a semi-automatic segmentation to extract voxels belonging to the skull, brain or ventricular system. From this model an anatomist can examine the data slice by slice and identify and label regions. Each region is extracted by selecting it using region filling, thresholding and pixel selection. Basically the anatomist would point to an object, and define data thresholds which would identify the voxels that belong to the object as those that can be reached by a path of voxels with values between the threshold values. For troublesome regions pixels can be included or removed in a way similar to using a paint package. Once an object has been selected it is given an entry in the knowledge base indicating its name, and functionality – for example [50], "*gyrus calcarinus is part of the lobus occipitalis*".
With this model the user can explore the human head with ease. A typical enquiry could be to start with the whole head, and indicate a cut and depth. This cut would be performed and the user can point to the revealed anatomy and query the database to find out its nature. The database will provide a label to each object and show the user where the object occurs in the knowledge tree. Cut planes can be taken, but could ignore specified objects such as eyes. X-ray images can be simulated with the addition of being able to enquire what has contributed to the resulting intensity along the ray. The knowledge tree itself can be queried – for example the user could select an object from the tree, and ask for it to be displayed.
The resulting system is a powerful exploration tool, giving users a very natural feeling of dissection as they explore the brain. Höhne estimated that about one person year was spent on the segmentation of the brain during the construction of the brain atlas. The group is now extending its work to the whole of the human anatomy [17].

14.7.9 Image segmentation

It has already been mentioned in the previous sections that image segmentation is a necessary tool in order to extract the various parts of the voxel model, so that each object can be examined in isolation. The problem is that there is no one method that can automatically segment complex objects from voxel data. Most methods rely on interaction, in the form of the user selecting voxels within a slice that are within the volume of interest (VOI) and filling the volume in 3D within a certain range. The range is defined using thresholds and the region is defined simply as all those voxels that can be reached from the start voxel by a path of voxels that have values that lie between the thresholds. The thresholds can be defined by allowing the user to probe the data – displaying the values of voxels pointed to by the user within a slice. Examples of this method can be found in [18, 19, 20, 21].
Algorithms which perform automatic segmentation on images (for example [11]) are used to perform semi-automatic segmentation of voxel data. The regions produced using an

automatic method will often have false connections (or bridges) to other regions, or will be split from regions that they should be joined to. One simple method to solve the first case is to shrink the voxel model by n voxels (where n is small, typically $n = 1$), and then grow it by n voxels. This will have the effect of removing links of n voxels wide [52, 50]. Other methods use the image processing techniques of erode and dilate to erode away the bridges or dilate the region to join it to others [53, 50].
Yoo et al. [54] show the difference between syntactic and semantic classification. In syntactic classification, geometric clipping is used to cut away parts of the volume not required, and intensity values are mapped to opacity using mappings such as gradient value. They point out that these operate on the whole volume and cannot distinguish between features such as the different bones in a leg. Semantic classification isolates the different parts of the volume interactively. Firstly the regions are segmented automatically, and connected in the form of a branching tree. This tree is traversed interactively by the user in order to select the object of interest. They remark on the effectiveness of their system by giving the example that the extraction of the brain, which previously took 40 minutes, now takes about 10 mouse button clicks in a few seconds.

14.8 Conclusion

This chapter has been concerned with the production of images from volume data using the method of volume rendering. A thorough comparison of the more popular viewing models was given in Sections 14.2 and 14.3. The new work presented in Section 14.4 was concerned with the acceleration of image production, and was compared to the existing acceleration methods of Section 14.5. Section 14.6 presented a thorough comparison of these methods, in terms of computational requirements and image quality. A thorough review of many recent developments of volume rendering was given in Section 14.7.

Bibliography

[1] M. W. Jones. The visualisation of regular three dimensional data. *University of Wales Swansea, UK*, July 1995.

[2] W. E. Lorensen and H. E. Cline. Marching Cubes : A high resolution 3D surface construction algorithm. In *Proc. SIGGRAPH '87 (Anaheim, Calif., July 27-31, 1987)*, volume 21(4), pages 163–169. ACM SIGGRAPH, New York, July 1987.

[3] J. Wilhelms and A. Van Gelder. Octrees for faster isosurface generation. *ACM Transactions on Graphics*, 11(3):201–227, July 1992.

[4] G. Wyvill, C. McPheeters, and B. Wyvill. Data structures for soft objects. *The Visual Computer*, 2:227–234, 1986.

[5] B. A. Payne and A. W. Toga. Surface mapping brain function on 3D models. *IEEE Computer Graphics and Applications*, 10(5):33–41, September 1990.

[6] M. Levoy. Display of surfaces from volume data. *IEEE Computer Graphics and Applications*, 8(3):29–37, May 1988.

[7] P. Sabella. A rendering algorithm for visualizing 3D scalar fields. In *Proc. SIGGRAPH '88 (Atlanta, Georgia, August 1-5, 1988)*, volume 22(4), pages 51–57. ACM SIGGRAPH, New York, August 1988.

[8] C. Upson and M. Keeler. V-Buffer : Visible volume rendering. In *Proc. SIGGRAPH '88 (Atlanta, Georgia, August 1-5, 1988)*, volume 22(4), pages 59–64. ACM SIGGRAPH, New York, August 1988.

[9] L. Carpenter R. A. Drebin and P. Hanrahan. Volume rendering. In *Proc. SIGGRAPH '88 (Atlanta, Georgia, August 1-5, 1988)*, volume 22(4), pages 65–74. ACM SIGGRAPH, New York, August 1988.

[10] J. D. Foley, A. Dam, S. K. Feiner, and J. F. Hughes. *Computer Graphics, Principles and Practice, 2nd ed.* Addison Wesley, 1990.

[11] A. Low. *Introductory computer vision and image processing*. McGraw-Hill Book Company (UK) Limited, Maidenhead, Berkshire, 1991.

[12] P. Burger and D. Gillies. *Interactive Computer Graphics*. Addison-Wesley, 1989.

[13] M. W. Jones and M. Chen. A new approach to the construction of surfaces from contour data. In *Computer Graphics Forum, Proc. Eurographics '94 (Oslo, Norway, S eptember 12-16, 1994)*, volume 13(3), pages C–75–C–84. University Press, Cambridge, UK., September 1994.

[14] G. T. Herman, J Zheng, and C. A. Bucholtz. Shape-based interpolation. *IEEE Computer Graphics and Applications*, 12(3):69–79, May 1992.

[15] M. W. Vannier, J. L. Marsh, and J. O. Warren. Three dimensional computer graphics for craniofacial surgical planning and evaluation. In *Proc. SIGGRAPH '83 (Detroit, Michigan, July 25-29, 1983)*, volume 17(3), pages 263–273. ACM SIGGRAPH, New York, July 1983.

[16] A. Kaufman, K. H. Höhne, W. Krüger, L Rosenblum, and P Schröder. Research issues in volume visualization. *IEEE Computer Graphics and Applications*, 14(2):63–67, March 1994.

[17] A. Pommert, B. Pflesser, M. Riemer, T. Schiemann, R. Schubert, U. Tiede, and K. H. Höhne. Advances in medical volume visualization. Technical Report ISSN 1017–4656, Eurographics, 1994.

[18] K. H. Höhne, M. Bomans, M. Riemer, R. Schubert, U. Tiede, and W. Lierse. A volume-based anatomical atlas. *IEEE Computer Graphics and Applications*, 12(4):72–78, July 1992.

[19] T. R. Nelson and T. T. Elvins. Visualization of ultrasound data. *IEEE Computer Graphics and Applications*, 13(6):50–57, November 1993.

[20] R. Yoshida, A. Doi T. Miyazawa, and T. Otsuki. Clinical planning support system - CliPSS. *IEEE Computer Graphics and Applications*, 13(6):76–84, November 1993.

[21] D. R. Ney and E. K. Fishman. Editing tools for 3D medical imaging. *IEEE Computer Graphics and Applications*, 11(6):63–70, November 1991.

[22] R. S. Avila, L. M. Sobierajski, and A. Kaufman. Visualizing nerve cells. *IEEE Computer Graphics and Applications*, 14(5):11–13, September 1994.

[23] R. Yagel and A. Kaufman. Template-based volume viewing. In *Computer Graphics Forum, Proc. Eurographics '92 (Cambridge, UK, September 7-11, 1992)*, volume 11(3), pages C–153–C–167. University Press, Cambridge, UK., September 1992.

[24] J. Amanatides and A. Woo. A fast voxel traversal algorithm for ray tracing. *Proc. of Eurographics '87, Amsterdam, The Netherlands*, pages 3–9, August 1987.

[25] M. Levoy. Volume rendering by adaptive refinement. *The Visual Computer*, 6:2–7, 1990.

[26] M. Levoy. Efficient ray tracing of volume data. *ACM Transactions on Graphics*, 9(3):245–261, July 1990.

[27] L. Westover. Footprint evaluation for volume rendering. In *Proc. SIGGRAPH '90 (Dallas, Texas, August 6-August 10, 1990)*, volume 24(4), pages 367–376. ACM SIGGRAPH, New York, August 1990.

[28] R. A. Crawfis and N. Max. Texture splats for 3D scalar and vector field visualization. In *Proc. Visualization 93*, pages 261–266. IEEE CS Press, Los Alamitos, Calif., 1993.

[29] D. Laur and P Hanrahan. Hierarchical splatting: A progressive refinement algorithm for volume rendering. In *Proc. SIGGRAPH '91 (Las Vegas, Nevada, July 28-August 2, 1991)*, volume 25(4), pages 285–288. ACM SIGGRAPH, New York, July 1991.

[30] N. Max, R. Crawfis, and D. Williams. Visualization for climate modeling. *IEEE Computer Graphics and Applications*, 13(4):34–40, July 1993.

[31] J. Wilhelms and A. V. Gelder. A coherent projection approach for direct volume rendering. In *Proc. SIGGRAPH '91 (Las Vegas, Nevada, July 28-August 2, 1991)*, volume 25(4), pages 275–284. ACM SIGGRAPH, New York, July 1991.

[32] P. L. Williams. Interactive splatting of nonrectilinear volumes. In *Proc. Visualization 92*, pages 37–44. IEEE CS Press, Los Alamitos, Calif., 1992.

[33] J. Wilhelms. Pursuing interactive visulization of irregular grids. *The Visual Computer*, 9:450–458, 1993.

[34] P. L. Williams. Visibility ordering meshed polyhedra. *ACM Transactions on Graphics*, 11(2):103–126, April 1992.

[35] A. Van Gelder and J. Wilhelms. Rapid exploration of curvilinear grids using direct volume rendering (extended abstract. In *Proc. Visualization 93*, pages 70–77. IEEE CS Press, Los Alamitos, Calif., 1993.

[36] T. Frühauf. Raycasting of nonregularly structured volume data. In *Computer Graphics Forum, Proc. Eurographics '94 (Oslo, Norway, September 12-16, 1994)*, volume 13(3), pages C–293–C–303. University Press, Cambridge, UK., September 1994.

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

[38] P. Ning and L. Hesselink. Fast volume rendering of compressed data. In *Proc. Visualization 93*, pages 11–18. IEEE CS Press, Los Alamitos, Calif., 1993.

[39] J. K. Udupa and D. Odhner. Fast visualization, manipulation, and analysis of binary volumetric objects. *IEEE Computer Graphics and Applications*, 11(6):53–62, November 1991.

[40] H. Samet and R. E. Webber. Hierarchical data structures and algorithms for computer graphics. *IEEE Computer Graphics and Applications*, 8(3):48–68, May 1988.

[41] A. S. Glassner. Space subdivision for fast ray tracing. *IEEE Computer Graphics and Applications*, 4(10):15–22, October 1984.

[42] R. Yagel and Z. Shi. Accelerating volume animation by space-leaping. In *Proc. Visualization 93*, pages 62–69. IEEE CS Press, Los Alamitos, Calif., 1993.

[43] K. R. Subramani and D. S. Fussell. Applying space subdivision techniques to volume rendering. In *Proc. Visualization 90*, pages 150–159. IEEE CS Press, Los Alamitos, Calif., 1990.

[44] R. S. Avila, L. M. Sobierajski, and A. E. Kaufman. Towards a comprehensive volume visualisation system. In *Proc. Visualization 92*, pages 13–20. IEEE CS Press, Los Alamitos, Calif., 1992.

[45] P. A. Fletcher and P. K. Robertson. Interactive shading for surface and volume visualization on graphics workstations. In *Proc. Visualization 93*, pages 291–298. IEEE CS Press, Los Alamitos, Calif., 1993.

[46] K. L. Ma, M. F. Cohen, and J. S. Painter. Volume seeds: A volume exploration technique. *The Journal of Visualization and Computer Animation*, 2:135–140, 1991.

[47] B. Corrie and P. Mackerras. Data shaders. In *Proc. Visualization 93*, pages 275–282. IEEE CS Press, Los Alamitos, Calif., 1993.

[48] J. L. Montine. A procedural interface for volume rendering. In *Proc. Visualization 90*, pages 36–44. IEEE CS Press, Los Alamitos, Calif., 1990.

[49] A. Pang and K. Smith. Spray rendering : Visualization using smart particles. In *Proc. Visualization 93*, pages 283–290. IEEE CS Press, Los Alamitos, Calif., 1993.

[50] K. H. Höhne, M. Bomans, A. Pommert, M. Riemer, C. Schiers, U. Tiede, and G. Wiebecke. 3D visualization of tomographic volume data using the generalized voxel model. *The Visual Computer*, 6:28–36, 1990.

[51] A. Pommert, M. Bomans, and K. H. Höhne. Volume visualization in magnetic resonance angiography. *IEEE Computer Graphics and Applications*, 12(5):12–13, September 1992.

[52] K. D. Toennies, J. Udupa, G. T. Herman, I. L. Wornom III, and S. R. Buchman. Registration of 3D objects and surfaces. *IEEE Computer Graphics and Applications*, 10(3):52–62, May 1990.

[53] L. Caponetti and A. M. Fanelli. Computer-aided simulation for bone surgery. *IEEE Computer Graphics and Applications*, 13(6):86–92, November 1993.

[54] T. S. Yoo, U. Neumann, H. Fuchs, S. M. Pizer, J. Rhoades T. Cullip, and R. Whitaker. Direct visualization of volume data. *IEEE Computer Graphics and Applications*, 12(4):63–71, July 1992.