Chapter 3 Methodology

3.1 Preface

In the previous chapter, the background of the research work involved in this thesis was provided. The process of segmentation and the level set and fast marching algorithms, were thoroughly explained. In this chapter, an overview of the research methodology is presented, followed by an explanation of the individual steps and problems involved.

3.2 Overview of Methodology

Figure 3.1 depicts the steps involved in the segmentation process presented in this research. The methodology begins with MR image acquisition. Images are acquired as 2-D slices that are in a proprietary format. Therefore, the slices have to be stored, and accessed using specific media. A third-party software is used to read these images. Following this, pre-processing is performed on the slices so that it can be read and viewed by the segmentation software. Steps are also carried out to prepare the data for segmentation. A separate set of pre-processing steps is performed for 2-D segmentation and 3-D segmentation.
The 2-D images can then be segmented using the 2-D version of the fast marching method. If 3-D segmentation was required, the 2-D images undergo further processing. Interpolation has to be performed first, to create a volume image. The volume image is then segmented using the 3-D version of the fast marching method. The result is displayed as a 3-D model, after isosurface rendering and the application of lighting models.
3.3 The methodology

3.3.1 MR Image Acquisition

MR images obtained through patient scans at the University Malaya Medical Centre, are provided for the ongoing research work by the Radiology Department of University Malaya. A Siemens Magnetom Vision MRI scanner, capable of imaging the whole body, is used in scanning. Scans are taken in the form of 2-D slices. Data sets of the femur, brain and heart are used in this research. Although femur segmentation is the thrust of this research, testing is also done on brain and heart images.

3.3.2 Storage and Access of Images

Data sets obtained from the scanner are in the ACR-NEMA (American College of Radiologists and the National Electrical Manufacturers Association) Version 2.0 format. This is a standard file format for storing medical images. It is also a proprietary format, capable of being read only by certain software. The data sets are stored in a 5¼ inch Magneto-Optical Disc (MOD). The stored data sets are read using a third-party software. Once these images have been read from the MOD, they are available as 2-D slices that can be processed for other purposes. However, they are still in a proprietary format and can therefore be read only by certain software.

Currently, the third-party software used to read the ACR images, is capable of
enhancing contrast and converting the data sets into the Tagged Image File Format (TIFF) format. This feature is necessary when ACR images are not clear. For this research, the ACR and TIFF file formats are used in presenting the segmentation algorithm and the results.

3.3.3 Pre-processing for 2-D Segmentation

Next, the images must be read into the segmentation software. The ACR images, are read in using a separate program specifically written for this purpose, in the past. The TIFF format can be read in directly, considering the fact that it is a popular format. Once the images have been read in, they are available as 2-D slices that can be viewed and processed further.

Many segmentation algorithms introduce some form of change into the image being segmented, either during or after segmentation. This change could be in the form of colour, highlighting of boundaries, or markings on the segmented region. It is required to clearly depict the results of the segmentation process. The 2-D segmentation algorithm employed in this research introduces colour into the image being segmented during the segmentation process. This leads to the next pre-processing step.

ACR data sets obtained from the scanner are read into the segmentation software as 2-D gray scale images. The images are stored in multidimensional arrays. ACR data usually falls within the range of [0, 2000]. This data range refers to the number of
gray levels in the image, that is the number of different shades of gray, between black and white, present in the image. If pixel values outside this range are applied to introduce other colours into the image, distortion caused by a shading effect is the result. Figure 3.2 depicts two ACR images of the femur. Figure 3.2(a) depicts the original image with gray level values in the range of \([0, 1466]\). In Figure 3.2(b), the intensity value of one of the pixels in the image has been changed to 3000. This has altered the entire image. Therefore, introducing colours other than those in the range of black to white is not possible.

<table>
<thead>
<tr>
<th>(a) Original ACR image</th>
<th>(b) ACR image with shading effect</th>
</tr>
</thead>
</table>

Figure 3.2 Shading effect caused by pixel values outside the data range

In the implementation of the fast marching algorithm, the segmented region must be depicted in a colour that clearly separates it from the rest of the image. If any colour within the data range is applied (between black to white), the segmented region will not be clearly distinguishable. In order to introduce other colours into the image, the 2-D \(m\)-by-\(n\) gray scale image has to be converted to a 3-D \(m\)-by-\(n\)-by-3 gray scale image. For a single pixel, three layers are needed to hold three values that correspond
to red, green and blue intensities. The conversion is done by building the \( m \times b y \times n \) image into an \( m \times b y \times n \times by \times 3 \) image, thus attaining the 3-D format. The steps involved are listed below:–

i. The original \( m \times b y \times n \times by \times 1 \) image \( I \), is read

ii. \( I \) is copied into the location of \( m \times b y \times n \times by \times 2 \)

iii. \( I \) is copied into the location of \( m \times b y \times n \times by \times 3 \)

Thus, a 3-D gray scale image with three layers is obtained. This image looks just like a 2-D image on display, although it contains a third dimension. Once a 3-D gray scale is obtained, any colour can be introduced into the image. While this conversion has to be performed for the sake of segmentation clarity, it comes at the expense of increased computational load.

The first problem that arose as a result of conversion to a 3-D gray scale, was display of the data. 2-D images can be displayed easily using the segmentation software's 2-D image display functions. These display functions are still capable of displaying a 3-D grayscale image. However, a limitation was encountered with 3-D images. The ACR data can be displayed as a 2-D image, but when it is converted to a 3-D grayscale, this is not possible due to its large data range.
To overcome this problem, the data was scaled into the $[0, 1]$ range, by normalization. The following are the steps involved:

i. The normalization factor, which is the largest element in the array, is obtained

ii. The entire array is normalized by dividing each array element with the normalization factor

iii. An array of 4-decimal pixel values is acquired

The pre-processing steps explained above were directly implemented for ACR images of the femur. ACR images of the brain and heart were obtained from the same MRI scanner, at a later point in time. Unfortunately, these images were distorted by intensity inhomogeneities. This type of artifact reduces the quality of an image by causing a shading effect to appear over it. Therefore, images are too dark to be segmented accurately. Another problem faced is that, a newer software version was used for the MRI scanner. As such, changes had been made in the way images were obtained and stored by the scanner. Due to these changes, ACR images of the heart and brain were further distorted by a dark shading effect when read into the segmentation software. These images were displayed even darker than the original images from the scanner, with image details being hardly visible.

Therefore, a third-party software was used to perform contrast enhancement to improve visualization of the ACR images. The contrast enhancement changes made to the images cannot be saved in the ACR format itself, hence these images have to be exported into the TIFF format. TIFF images can be read directly into the
segmentation software. TIFF data is stored in a different format compared to ACR data, and has a data range of only [0, 255]. Therefore, normalization is not necessary. Apart from this, when conversion to the TIFF format is done the images are automatically built into 3-D gray scale images. Therefore, TIFF images can be displayed directly. Figure 3.3 provides a summary of the pre-processing steps involves, before the image is loaded for segmentation. Boxes with the dotted lines represent processes that are carried out.

![Diagram of pre-processing steps for 2-D segmentation](image)

Figure 3.3 Summary of pre-processing steps for 2-D segmentation

### 3.3.4 The 2-D Segmentation Algorithm

Once the image has been built into a 3-D grayscale, it is loaded for display. Segmentation can then be performed. Figure 3.4 briefly depicts the implementation of
the fast marching algorithm. Each section will explain the flow of the algorithm in
detail.

```
Select seed point → Tag as Known
  ↓
Get 4-neighbours → Tag as Trial
  ↓
Reload Image
  ↓
Calculate Gradient
  ↓
Get minimum point
  ↓
Compute control function, F
  ↓
F > 0, Enter loop
```

Figure 3.4 Brief overview of 2-D segmentation algorithm

The fast marching method is a region-growing algorithm, in which a seed point must
be provided by the user. The beginning of the fast marching method is depicted in
Figure 3.5. The black sphere is the seed point placed by the user, representing the
known boundary value. The light gray spheres are grid points where the solution is
unknown.

Figure 3.5 The seed point
The method progresses by considering points in a thin zone (narrow band) around the existing front, and marching this thin zone forward [23]. In the implementation of the method, the seed point placed by the user represents the existing front, and the 4-neighbours of the grid point (Figure 3.7) are determined to form the thin zone around it.

The value of the existing seed point is frozen and new ones are brought into the narrow band zone, to allow the region growing process. The image has to be reloaded to view the changes taking place. The selection of which grid point in the narrow band to update, guides the region growing process [23]. This grid point is determined by calculation of the image gradient $T$, between the existing point and its 4-neighbours. The point with the minimum gradient is selected and used to calculate the control function, $F$. If $F$ was more than 0, the algorithm enters a loop to repeat the process.

Figure 3.6 depicts how the fast marching algorithm marches out [23]. In Figure 3.6(a), the algorithm starts marching outwards, from the known value, by first computing its 4-neighbourhood grid points. The 4-neighbours of the seed point are marked as dark gray spheres, and labeled A, B, C and D respectively (Figure 3.6(b)). In Figure 3.6(c), sphere A has been chosen as the minimum point. Subsequently, the 4-neighbours of A are determined, and A is frozen (Figure 3.6(d)).
In Figure 3.6(e), sphere D has been chosen as the minimum point among all the dark gray points in the narrow band. Therefore, the 4-neighbours of D are determined, and D is frozen (Figure 3.6(f)). The algorithm proceeds in this manner, being controlled by F. Management of the narrow band points, the image gradients and selection of the minimum gradient is done using the min-heap data structure.
3.3.4.1 Selection of the Seed Point

As soon as the image is displayed, the shape of the cursor changes from the usual white arrow, to a white and black cross. This indicates that the program is waiting for an input from the user. The user then selects the seed point on the region to be segmented, through a single click of the mouse. As soon as the seed point is placed, its coordinate and pixel value are read into the program, and it is tagged as Known, whereby the point is "frozen" and will not be used in computation again. From Section 2.7.3.3, in the fast marching method the speed function moves in only one direction, therefore a point will not be crossed twice. When a point is tagged as Known, the value of the corresponding pixel is changed such that it appears in the colour red on display (Figure 4.2).

3.3.4.2 Determining the 4-neighbours of a Pixel

The 4-neighbourhood of an object is defined based on the concept of connectivity (Section 2.3.3.1). Figure 3.7 describes the 4-connectivity(neighbourhood) of a point.

![Diagram of 4-neighbourhood of a point](image)

Figure 3.7 Four-neighbourhood of a point

In determining the 4-neighbours of a point, conditions at the boundaries of the square image, and its four corners must be considered. A point on the boundary of an image
will have only three neighbours, while a corner point will have only two (Figure 3.7).

Figure 3.8 pictures a grid point \( I(x,y) \) and its 4-neighbours. \( I \) represents the image, while \( x \) and \( y \) represent the x-coordinate and y-coordinate respectively. Therefore, \( I(x,y) \) represents the value of the pixel at coordinate \((x,y)\) in the image \( I \).

![Figure 3.8 A grid point and its 4-neighbours](image)

As soon as these points have been determined, the coordinates and pixel values of each point are stored in an array. These 4-neighbourhood points are then tagged as *Trial*, whereby they are highlighted in light blue on the image.

### 3.3.4.3 Reloading the Image

To see the changes that have taken place, the image has to be reloaded. After it is reloaded, an extremely minute pause of \( 1 \times 10^{-10} \) seconds is applied to see the reloaded image. The evolving of the red interface, bordered by a light blue boundary is visible (Figure 4.2).
3.3.4.4 Gradient Computation

In the next step, gradient computation is performed. The image gradient between the seed point and each of its 4-neighbours is computed. This is done based on the following set of equations. Firstly, the grid step needs to be determined. Based on Figure 3.9 the grid step along the x-axis and y-axis are given by [23]:

\[
\nabla x = h \quad \text{(Equation 3.1)}
\]

\[
\nabla y = k \quad \text{(Equation 3.2)}
\]

![Figure 3.9 Computational grid](image)

When the boundary marches out, it expands in the x and y direction of the plane. In obtaining the 4-neighbours, two neighbours are computed in the x-axis, while another two are computed in the y-axis. Therefore spatial derivatives have to be obtained for the x and y direction, to determine the gradient. The gradient is actually a change of pixel values across space, in the x-y plane. Given \( T \) as the gradient and a point \((x,y)\), these spatial derivatives are derived from difference operators, to calculate the \( T \) at each of the 4-neighbours. The difference operators are defined as follows:-
I. The forward difference operator uses information from grid points ahead of point \((x, y)\) in computation.

\[ \Rightarrow D^{+yT} \] uses information at \(y\) and \(y+k\) to compute the gradient value, thus information for the solution propagates from top to bottom.

\[ \Rightarrow D^{+xT} \] uses information at \(x\) and \(x+h\) to compute the gradient value, thus information for the solution propagates from right to left.

II. The backward difference operator uses information from grid points preceding the point \((x, y)\) in computation.

\[ \Rightarrow D^{-yT} \] uses information at \(y\) and \(y-k\) to compute the gradient value, thus information for the solution propagates from bottom to top.

\[ \Rightarrow D^{-xT} \] uses information at \(x\) and \(x-h\) to compute the gradient value, thus information for the solution propagates from left to right.

These operators allow computation of the change in pixel values across the image, in the four main directions surrounding a pixel. Therefore, it can be said that information is obtained "globally" in terms of the pixel, to trace the current status of that pixel, and make an informed decision about which point to choose.

(a) Computation of the difference operators for the \(y\) direction

(i) Deduction of \(D^{+yT}\)

The solution for the gradient, \(T\) at a new grid point begins with the Taylor series expansion, around the point \((x, y)\) [23]. The following equation is given for gradient
computation in the y direction, between point \((x, y)\) and its neighbouring point \((x, y + k)\) [23]:

\[
T(x, y + k) = T(x, y) + T_y(x, y)k + O(k^2)
\]

(Equation 3.3)

where the expression \(O(k^2)\) includes all terms of order \(k^2\) or higher.

Equation 3.3 is rearranged to produce the gradient derivative at point \((x, y)\) [23]:

\[
T_y = \frac{T(x, y + k) - T(x, y)}{k} + O(k)
\]

(Equation 3.4)

Equation 3.4 is known as a forward difference operator, since the grid point \((y + k)\) that is ahead of the current one, \(y\) has been used. Consecutively, the spatial derivative is defined as [23]:

\[
D^+yT = \frac{T(x, y + k) - T(x, y)}{k}
\]

(Equation 3.5)

Computation using Equation 3.5 will provide the gradient value between point \((x, y)\) and point \((x, y + k)\).

(ii) Deduction of \(D^yT\)

The following equation is used for gradient computation in the y direction, between point \((x, y)\) and point \((x, y - k)\) [23]:

\[
T(x, y - k) = T(x, y) + T_y(x, y)k + O(k^2)
\]

(Equation 3.6)

Equation 3.6 is rearranged to produce the backward difference operator [23]:

\[
T_y = \frac{T(x, y) - T(x, y - k)}{k} + O(k)
\]

(Equation 3.7)
The spatial derivative is then obtained [23]:

\[ D^{-y}T = \frac{T(x, y) - T(x, y-k)}{k} \]  \hspace{1cm} \text{(Equation 3.8)}

Computation using Equation 3.8 will provide the gradient value between point \((x, y)\) and point \((x, y-k)\).

\(b\) Computation of the difference operators for the \(x\) direction

(i) Deduction of \(D^{+x}T\)

Next, the equations for gradient computation in the \(x\) direction, between point \((x, y)\) and its neighbouring point \((x+h, y)\) are considered. A Taylor series expansion around the point \((x, y)\) produces the following equation [23]:

\[ T(x+h, y) = T(x, y) + T_{x}(x, y)k + O(k^2) \]  \hspace{1cm} \text{(Equation 3.9)}

Rearrangement of Equation 3.9 produces the backward difference operator [23]:

\[ T_{x} = \frac{T(x+h, y) - T(x, y)}{h} + O(k) \]  \hspace{1cm} \text{(Equation 3.10)}

The spatial derivative is then obtained [23]:

\[ D^{+x}T = \frac{T(x+h, y) - T(x, y)}{h} \]  \hspace{1cm} \text{(Equation 3.11)}

Computation using Equation 3.11 will provide the gradient value between point \((x,y)\) and point \((x+h, y)\).
(ii) Deduction of $D^x T$

Gradient computation in the $x$ direction, between point $(x, y)$ and point $(x - h, y)$ begins with the following equation [23]:

$$T(x - h, y) = T(x, y) + T_x(x, y)k + O(k^2)$$  \hspace{1cm} (Equation 3.12)

Equation 3.12 is subsequently rearranged to produce the backward difference operator [23]:

$$T_x = \frac{T(x, y) - T(x - h, y)}{h} + O(k)$$  \hspace{1cm} (Equation 3.13)

The final spatial derivative is then obtained [23]:

$$D^x T = \frac{T(x, y) - T(x - h, y)}{h}$$  \hspace{1cm} (Equation 3.14)

Computation using Equation 3.14 will provide the gradient value between point $(x, y)$ and point $(x - h, y)$.

Based on the equations above, four difference operators have been obtained to calculate the gradient, where each one corresponds to one of the 4-neighbours of the point $l(x, y)$.

i. $D^y T = \frac{T(x, y + k) - T(x, y)}{k}$

ii. $D^-y T = \frac{T(x, y) - T(x, y - k)}{k}$
iii. $D^{xy}T = \frac{T(x + h, y) - T(x, y)}{h}$

iv. $D^{-x}T = \frac{T(x, y) - T(x - h, y)}{h}$

From Equations 3.1 and 3.2, $k$ and $h$ represent the grid step of the underlying coordinate system. In the implementation of the fast marching method for this research, the grid step has been fixed as 1. Therefore the equations above are reduced to:

\[ D^{xy}T = T(x, y + 1) - T(x, y) \]  \hspace{1cm} (Equation 3.15)

\[ D^{-y}T = T(x, y) - T(x, y - 1) \]  \hspace{1cm} (Equation 3.16)

\[ D^{yx}T = T(x + 1, y) - T(x, y) \]  \hspace{1cm} (Equation 3.17)

\[ D^{-x}T = T(x, y) - T(x - 1, y) \]  \hspace{1cm} (Equation 3.18)

Hence, the gradient value between a point $(x, y)$ and each of its 4-neighbours, simply becomes the difference between the pixel $I(x, y)$ and each of the four neighbouring pixels.

Once the gradient values in all four directions have been computed, the coordinates and pixel values of the 4-neighbours are written into another array, together with the
corresponding gradient values. In other words, points that are tagged as Trial are written into another array, which is then built into a min-heap data structure. Section 2.3.2.4 explains heaps in detail. The Trial min-heap is built such that an element at position $k$ of the array, will have its left and right children at positions $2k + 1$ and $2k + 2$, respectively. If these positions were beyond the end of the array, then these children do not exist. Also, position $k$ will always contain a value that is equal to or smaller than its children. Therefore, the smallest element in the array, will always be at position $k = 1$ of the array.

3.3.4.5 Obtaining the Minimum Point

Once all the gradient values have been computed, the most suitable point needs to be chosen based on these gradient values. The choice of which point, will determine the direction of movement for the interface. Grid points are chosen in terms of the direction of the flow of information. Information in terms of the image, refers to pixel values. The information flows based on image gradient values [23,135].

The interface is required to march outwards. The direction from which the interface is marching is known as the upwind side, while the direction towards which the interface is marching is known as the downwind side. This is depicted in Figure 3.10. For the interface to march downwind, a difference scheme which reaches upwind should be used, in order to get information to construct the solution downwind [135]. Therefore, what is known as an upwind scheme is used to enable propagation downwind [23,135].
An upwind scheme allows propagation of information one way, from smaller gradient values to larger gradient values [23]. Therefore, the point with the minimum gradient value is always selected, thereby fulfilling the requirements of an upwind scheme [23,135]. A min-heap data structure is used for this very reason, to allow the minimum gradient value to be obtained easily and in the fastest possible time. Once the minimum gradient has been chosen, the coordinates and pixel value of the corresponding point are obtained.

Accordingly, the minimum point which is the first element in the Trial min-heap is removed. The heap is then swept through to adjust the values within, ensuring that the min-heap property is once-again fulfilled.
3.3.4.6 Computing the Control Function

The minimum gradient obtained from the previous section, is used to compute the control function, $F$. The fast marching explanation given in Section 2.7.3.3 mentions a speed function, which increases the propagation speed of the interface when it is far from an edge, and decreases the speed of the interface till it halts at the edge. For this research, a control function is employed instead of a speed function. Although the segmentation software is capable of extremely large computations at high speeds, it is not fast enough to incorporate speed in the segmentation algorithm.

$F$ is used to control the outward propagation of the interface, using gradient information. When the gradient is large, this indicates the presence of an edge in the image. $F$ is formulated such that it reduces to 0, when it nears the edge. $F$ is given by [156]:-

$$ F = e^{-c |\nabla I|} \quad \text{for } c > 0$$  \hspace{1cm} \text{(Equation 3.19)}

where $c$ represents a control parameter and $\nabla I$ represents the image gradient. Equation 3.19 is a variation of the Eikonal equation (Equation 2.11), in Section 2.7.3.3, given as:-

$$ |\nabla T| F = I $$

According to this equation, $F$ depends only on the local position of the interface at a given time. $F$ can be written as:-

$$ F = \frac{1}{|\nabla T|} $$  \hspace{1cm} \text{(Equation 3.20)}
The Eikonal equation states that $F$ is directly proportional to the gradient value. Therefore, when the gradient increases, $F$ will approach 0. However, in reality only an extremely large gradient value will ever reduce $F$ to 0. Therefore an exponent is incorporated in Equation 3.19, to allow values very close to zero for large image gradients. Theoretically, the image gradient should be sufficient to reduce $F$ to 0 and halt the interface. However, in reality once again, this theory will probably work only for the ideal image. Therefore, a control parameter is used to provide some form of control over the segmentation process, for the user.

A positive exponent value will increase $F$, while a negative exponent value will decrease $F$. The gradient could be positive or negative, depending on the pixel values. If a negative gradient was encountered, a positive exponent will produce a large $F$. Therefore, the gradient is placed within modulus brackets, to always yield a positive value. A negative sign is added to the control parameter instead. Thus, $F$ will always be a positive value approaching 0. This also fulfills the requirement of the fast marching method that the speed function must move in only one direction. Since $F$ is always more than 0 and halts the interface when it reaches 0, the interface moves outwards at all times.

3.3.4.7 In the Loop

If the value $F$ becomes zero, the algorithm will end, thus halting the interface. As long as $F$ is more than zero, the algorithm enters a loop, and continues in iteration. Referring to Figure 3.4, the algorithm selects a seed point again. This time, the seed
point is the minimum point selected as explained in Section 3.3.4.5. The seed point is tagged as *Known*.

The 4-neighbours of the seed point are computed and written into an array. The 4-neighbours that are obtained could include a *Known* (frozen) point from previous computations, therefore the array needs to be searched to remove any such point. The remaining eighbours are then tagged as *Trial*. After this, the image is reloaded to view the propagation of the interface.

The neighbours could also be points that have already been tagged as *Trial* in the previous computation. Therefore, the *Trial* min-heap must be searched to remove any repeating points. The removal is done from the heap instead of the neighbourhood array, because a point that has been retagged as *Trial* must be updated with the new gradient value. The old gradient value previously calculated at that point and stored in *Trial*, becomes irrelevant.

Once overlapping points are removed from *Trial*, gradient calculation is performed as explained in Section 3.3.4.4. Next, the coordinates and pixel values of the neighbours, together with the gradient values, are written into the *Trial* min-heap. These points have to be inserted one by one, ensuring each time that the min-heap property is once again adhered to. The next minimum point is then chosen, based on the minimum gradient value. This minimum point is deleted from the *Trial* min-heap, and the heap is swept through to adjust its entries, to adhere to the condition of a min-
heap. The minimum gradient value is then used to calculate the control function $F$.

If $F$ was more than 0, the algorithm continues into the loop again. The minimum point that was chosen becomes the next seed point. This point is tagged as $Known$, and so the algorithm proceeds. Thus, smaller gradient values which have been crossed by the interface become insignificant. The interface propagates by freezing these insignificant points, thus growing a red region.

### 3.3.5 Pre-processing for 3-D Segmentation

Figure 3.11 provides an overview of segmentation in 3-D. The methodology begins with reading in the 2-D images. As explained in Section 3.3.3, ACR images of the femur are read into the segmentation software using a specific program. ACR images of the brain and heart are distorted by intensity inhomogeneities, and are therefore too dark to perform accurate segmentation. Therefore they are read into a third-party software and converted to the TIFF format after contrast enhancement. Subsequently, they are read directly into the segmentation software .

As the images are being read into the segmentation software one by one, the slices are concatenated, or stacked up together in order. This produces a 3-D structure. In the next step, interpolation is performed through voxel calculation to obtain a gray level volume.
Figure 3.11 Pre-processing steps for 3-D segmentation

The 2-D data sets obtained from the scanner are taken at certain distances (in millimeters) from each other. In Figure 3.12, the distance between Slice 1 and Slice 2 is marked by D. The information between these two slices, spanning the distance of D, is missing. If the 3-D volume was to be built by simply stacking the slices up together and concatenating them, there would be discontinuities in the volume.
Therefore, after the 3-D structure is built, interpolation must be performed to fill in the missing information.

Figure 3.12 Voxel computation method

For this research, interpolation was done through voxel computation. A voxel can be defined through its size and intensity. While voxel size is irrelevant in the computation method used, voxel intensity is calculated using pixel values of adjacent slices. Referring to Figure 3.12, the shape of a voxel is that of a cube. Therefore, a voxel has eight corners. Four of those corners ($\alpha_1 - \alpha_4$) fall on Slice 1, while the other four ($\alpha_5 - \alpha_8$) fall on Slice 2. Hence, the intensity of a voxel can be computed as the mean of the pixel intensities, at the eight corresponding corner points that would form that voxel. This computation is given in the following equation:-
Voxel Intensity, \( v = \frac{\alpha_1 + \alpha_2 + \alpha_3 + \alpha_4 + \alpha_5 + \alpha_6 + \alpha_7 + \alpha_8}{8} \) \hspace{1cm} \text{(Equation 3.21)}

where \( \alpha \) is the pixel intensity

Once these calculations are completed, voxels are defined. A 3-D gray level volume is therefore created. Next, a slice from the 3-D volume is chosen by the user. This slice is displayed for the user to choose the region to segment. Consequently, 3-D segmentation can be performed.

### 3.3.6 The 3-D Segmentation Algorithm

The 3-D segmentation algorithm is pictured in Figure 3.13. This algorithm is basically an extension of the 2-D algorithm, with a few changes in implementation.

![Flowchart of the 3-D segmentation algorithm](image)

Figure 3.13 The 3-D segmentation algorithm
3.3.6.1 Selection of the Seed Voxel

The slice of the volume chosen by the user is displayed. The entire volume is not displayed, because the region the user wishes to segment may be within the volume, not visible from the outside. Instead the user is allowed to view the entire range of slices in the data set, and select the slice that clearly depicts the region to be segmented. The program then waits for an input from the user. The user selects the seed voxel on the region to be segmented, through a single click of the mouse.

As soon as the seed voxel is placed, its coordinate and value are read into the program, and it is tagged as Known. When a voxel is tagged as Known, the value of the corresponding voxel is changed to a value just outside the data range. For example, if the data range was [0, 1445], the Known value is set as 1446. If the data range was [>0, 1445], the value is set at 0. Selection of a Known value is done in this manner because if a value that is too far out is chosen, display of entire data set is affected by a shading effect. Precaution also needs to be taken not to select a value within the data set or the segmentation results will be distorted.

3.3.6.2 Determining the 6-neighbours of a Voxel

The 6-neighbourhood of a voxel \( I(x,y,z) \) is depicted in Figure 3.14. Apart from the 4-neighbours (Section 3.3.4.2) that are computed from the same slice, the other two neighbours are obtained from the slice before and after the current slice. \( I \) represents the volume image, while \( x, y \) and \( z \) represent the x-coordinate, y-coordinate and z-coordinate respectively. Therefore, \( I(x,y,z) \) represents the value of the voxel at
coordinate \((x,y,z)\) in the image \(I\).

![Diagram of a voxel and its 6-neighbours](image)

Figure 3.14 A voxel and its 6-neighbours

In determining the 6-neighbours of a voxel, conditions at the sides, the edges and the eight corners of the square volume, must be considered. As soon as these voxels have been determined, the coordinates and values of each voxel are stored in an array.

### 3.3.6.3 Reloading the Slice

To see the changes that have taken place, the slice currently being processed is reloaded. A pause is not necessary as in 2-D segmentation, because the process is taking place at a much slower rate. The evolving of the interface on that particular slice, is visible. The colour of the interface will depend on the Known value chosen by the program. The entire volume is not displayed at every time step, due to the large amount of time taken. Also, the spreading of the interface will not be visible if it is taking place inside the volume.
3.3.6.4 Gradient Computation

The image gradient between the seed voxel and each of its 6-neighbours is computed. When the boundary marches out, it expands in the x, y and z direction of the plane. In obtaining the 6-neighbours, two neighbours are computed in the x-axis, two in the y-axis, and another two in the z-axis. Therefore as explained in Section 3.3.4.4, spatial derivatives have to be obtained for all these axis directions, to determine the gradient. The forward and backward difference operators for the x- and y-axis have been defined in Section 3.3.4.4. The difference operators for the z-axis are defined as follows:

I. The forward difference operator uses information from the slice ahead of voxel \((x,y,z)\) in computation.

\(D^+T\) uses information at \(z\) and \(z+k\) to compute the gradient value, thus information for the solution propagates from back to front.

II. The backward difference operator uses information from the slice preceding the voxel \((x,y,z)\) in computation.

\(D^-T\) uses information at \(z\) and \(z-k\) to compute the gradient value, thus information for the solution propagates from front to back.

Computation of the difference operators for the x and y directions have been explained in Section 3.3.4.4. The only difference in 3-D segmentation is the incorporation of an extra dimension. The difference operators for the z direction are derived in the same way. Therefore, only the final set of equations is provided below.
Difference operators for the $x$ direction:

\[ D^+xT = T(x+1, y, z) - T(x, y, z) \]  \hspace{1cm} (Equation 3.22)

\[ D^-xT = T(x, y, z) - T(x-1, y, z) \]  \hspace{1cm} (Equation 3.23)

Difference operators for the $y$ direction:

\[ D^+yT = T(x, y+1, z) - T(x, y, z) \]  \hspace{1cm} (Equation 3.24)

\[ D^-yT = T(x, y, z) - T(x, y-1, z) \]  \hspace{1cm} (Equation 3.25)

Difference operators for the $z$ direction:

\[ D^+zT = T(x, y, z+1) - T(x, y, z) \]  \hspace{1cm} (Equation 3.26)

\[ D^-zT = T(x, y, z) - T(x, y, z-1) \]  \hspace{1cm} (Equation 3.27)

Hence, the gradient value between a voxel $I(x, y, z)$ and each of its 6-neighbours, becomes the difference between $I$ and each of those six neighbouring voxels.

Once the gradient values in all six directions have been computed, the coordinates and voxel values of the 6-neighbours are written into another array, together with the corresponding gradient values. This array is built into a min-heap data structure (Section 2.3.2.4 and Section 3.3.4.4).
3.3.6.5 Obtaining the Minimum Voxel

As explained in Section 3.3.4.5, the minimum gradient value must be chosen. The coordinates and value of the corresponding voxel are obtained. The minimum voxel is removed from the *Trial* min-heap. The heap is then swept through to adjust the values within, ensuring that the min-heap property is once-again fulfilled.

3.3.6.6 Computing the Control Function

The minimum gradient obtained from the previous section, is used to compute the control function, $F$. $F$ is applied in the same way, as explained in Section 3.3.4.6.

3.3.6.7 In the Loop

If $F$ is more than zero, the algorithm enters a loop, and continues in iteration. Based on Figure 3.12, the algorithm selects the previous minimum voxel (Section 3.3.6.5), as the new seed voxel. This voxel is tagged as *Known*. The algorithm continues as explained in Section 3.3.4.7. Extra heap operations are involved when the minimum voxel is removed and the heap is adjusted. Search and remove operations are also carried out to ensure that voxels are correctly updated and do not overlap. The spreading of the interface is only visible on the slices, and not across the third dimension. Once the segmentation algorithm is complete, the segmented 3-D data is converted into a binary volume. The value of *known* voxels are changed to 1, while all other values are changes to 0. The binary volume is then rendered for display.
3.3.7 Isosurface Rendering and Lighting Models

The isosurface algorithm is used to generate a surface for the volume image (Section 2.3.4). An isosurface is a three dimensional plot that connects points of constant value, known as isovalues, in 3D space to form a surface. The isovalue is the Known value selected in Section 3.3.6.1. A built in function of the segmentation software is applied for isosurface generation.

The isosurface is computed as a patch. A patch is a graphics object, composed of one or more polygons defined by the coordinates of its vertices. Each patch is bound by edges, which are line segments that connect the vertices. Patch properties such edge colour and face colour were defined for isosurface generation of the 3-D data. As the isosurface is generated, normals of the isosurface are computed. These normals, known as isonormals are important since they provide the visual appearance of lit surfaces [30]. This is also done using functions in the segmentation software.

Lighting where then applied. Lighting is a technique for adding realism to a graphical scene [30]. It does this by simulating the highlights and dark areas that occur on objects under natural lighting such as from the sun. Light properties such as colour, distance of the light source and its direction are manipulated to add realism to the 3-D model. Patch properties such as face lighting, edge lighting and light reflection can also be set. Once these functions have been executed, the segmentation volume is projected in a 3-D view for visualization.
3.4 *Summary*

The methodology involved for this research has been explained thoroughly in this chapter. Each step, beginning with MR image acquisition, storage and access of the images and pre-processing was explained. This was followed by a detailed description of the 2-D and 3-D segmentation algorithms. The chapter ended with an explanation of the final step in 3-D segmentation, isosurface rendering. In the next chapter, the results obtained from this methodology are provided. In the next chapter, the results obtained at every step of the methodology are presented and discussed.