DATA STRUCTURES AND ALGORITHMS FOR MANIPULATION AND DISPLAY IN COMPUTER SIMULATED SURGERY

by
Simon Robert Arridge, B.A.

Dept. of Medical Physics and Bioengineering
University College London.

Submitted for
The degree of Doctor of Philosophy
of the University of London

December 1989
DATA STRUCTURES AND ALGORITHMS FOR MANIPULATION AND DISPLAY IN COMPUTER SIMULATED SURGERY

ABSTRACT

This thesis describes the development of a computer graphics facility, referred to as UCL3D, for the planning, simulation and evaluation of Maxillo-Facial surgery on a conventional super-minicomputer, with a colour graphics framestore.

The introduction defines the requirements as data acquisition, preprocessing, visualisation, dissection, manipulation, quantification, and registration. The principle innovations are in the visualisation, dissection and manipulation stages.

The first part of the thesis is concerned with basic definitions and a review of other work. The data acquisition is assumed to be from a medical imaging device that produces a 3D digital array of density values. The preprocessing stage involves interpolation, artefact removal, subregioning, and segmentation.

The computer representation of discrete objects from medical 3D data is discussed, and the need to have a volume based representation is justified. The implementation of UCL3D is in terms of octrees, although the principles could be applied to other representations. Both a binary and grey-value implementation have been developed, and for each case, a new structure variation is described.

Visualisation is discussed in terms of these representations and surface shading techniques appropriate to each are introduced. A new algorithm for deriving the quadtree projection of an octree in orthogonal directions is presented, and its advantages explained. The concept of a general volume mask, specified interactively, is introduced for the dissection problem. The space partitioning resulting from this mask is more general than previous methods.
Manipulation is considered as the combination of Boolean operations and translation and rotations, acting on combinations of dissected objects. The philosophy is to cut and merge medical objects to simulate the "osteotomies" encountered in surgery. Boolean expressions of objects may be visualised prior to being created. A description of the application of the system to several clinical cases is given and finally several areas for future work are suggested.
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.3</td>
<td>OPERATIONS ON MULTIPLE OBJECTS</td>
<td>157</td>
</tr>
<tr>
<td>6.4</td>
<td>RESULTS</td>
<td>158</td>
</tr>
<tr>
<td>6.4.1</td>
<td>Example Construction of an arbitrary object</td>
<td>158</td>
</tr>
<tr>
<td>6.4.2</td>
<td>Example Volume Mask from Region Masks</td>
<td>161</td>
</tr>
<tr>
<td>6.5</td>
<td>DISCUSSION AND CONCLUSIONS</td>
<td>161</td>
</tr>
<tr>
<td></td>
<td>CHAPTER SEVEN</td>
<td>163</td>
</tr>
<tr>
<td>7.1</td>
<td>INTRODUCTION</td>
<td>163</td>
</tr>
<tr>
<td>7.2</td>
<td>EXPLICIT BOOLEAN OPERATIONS</td>
<td>164</td>
</tr>
<tr>
<td>7.3</td>
<td>DISPLAY OF BOOLEAN EXPRESSIONS OF OBJECTS AND MASKS</td>
<td>167</td>
</tr>
<tr>
<td>7.3.1</td>
<td>Algorithms</td>
<td>167</td>
</tr>
<tr>
<td>7.3.2</td>
<td>Results</td>
<td>170</td>
</tr>
<tr>
<td>7.4</td>
<td>BILINEAR TRANSFORMATIONS</td>
<td>172</td>
</tr>
<tr>
<td>7.4.1</td>
<td>Surface descriptions</td>
<td>172</td>
</tr>
<tr>
<td>7.4.2</td>
<td>Volume Descriptions</td>
<td>172</td>
</tr>
<tr>
<td>7.4.3</td>
<td>Octree Descriptions</td>
<td>173</td>
</tr>
<tr>
<td>7.4.4</td>
<td>Transformation of Boolean Expressions of Octrees</td>
<td>175</td>
</tr>
<tr>
<td>7.4.4.1</td>
<td>Translation</td>
<td>176</td>
</tr>
<tr>
<td>7.4.4.2</td>
<td>Rotation</td>
<td>179</td>
</tr>
<tr>
<td>7.5</td>
<td>CONCLUSIONS</td>
<td>180</td>
</tr>
<tr>
<td></td>
<td>CHAPTER EIGHT</td>
<td>183</td>
</tr>
<tr>
<td>8.1</td>
<td>APPLICATION AREAS</td>
<td>183</td>
</tr>
<tr>
<td>8.2</td>
<td>OTHER SIMULATION SYSTEMS</td>
<td>184</td>
</tr>
<tr>
<td>8.3</td>
<td>INTERACTIVE CONSIDERATIONS</td>
<td>187</td>
</tr>
<tr>
<td>8.4</td>
<td>CLINICAL EXAMPLES</td>
<td>189</td>
</tr>
<tr>
<td>8.4.1</td>
<td>General Points</td>
<td>189</td>
</tr>
<tr>
<td>8.4.2</td>
<td>Case 1 : Crouzon Syndrome,</td>
<td>190</td>
</tr>
<tr>
<td>8.4.3</td>
<td>Case 2 : Treacher Collins’ Syndrome,</td>
<td>193</td>
</tr>
<tr>
<td>8.4.4</td>
<td>Case 3 : Orthognathic</td>
<td>199</td>
</tr>
<tr>
<td>8.5</td>
<td>DISCUSSION AND CONCLUSIONS</td>
<td>207</td>
</tr>
<tr>
<td></td>
<td>CHAPTER NINE</td>
<td>209</td>
</tr>
<tr>
<td>9.1</td>
<td>SUMMARY</td>
<td>209</td>
</tr>
<tr>
<td>9.2</td>
<td>FURTHER ASPECTS</td>
<td>212</td>
</tr>
<tr>
<td>9.2.1</td>
<td>Quantification</td>
<td>212</td>
</tr>
<tr>
<td>9.2.2</td>
<td>Soft tissue Prediction</td>
<td>213</td>
</tr>
<tr>
<td>9.2.3</td>
<td>Comparison of prediction to outcome</td>
<td>215</td>
</tr>
<tr>
<td>9.2.3.1</td>
<td>Registration</td>
<td>216</td>
</tr>
<tr>
<td>9.2.3.2</td>
<td>Evaluation of surface differences</td>
<td>218</td>
</tr>
</tbody>
</table>
TABLE OF FIGURES

Figure 1.1: Schematic diagram of laser surface scanning system. 23
Figure 1.2: A set of consecutive CT slices, from a patient with a cleft palate. 24
Figure 1.3: 16 CT slices, from the same set as figure 1.2, located in their 3D relationship. 25
Figure 1.4: A set of consecutive NMR slices of the head. 26
Figure 1.5: NMR slices of the brain, located into their 3D relationship. 27
Figure 1.6: Schematic of 2D and 3D display strategies. 32
Figure 1.7: System diagram of the UCL3D hardware. 38
Figure 1.8: System diagram of UCL3D software. 39
Figure 2.1: Overall scene relationships. 47
Figure 2.2: Images of a dry skull, illustrating the interpolation problem. 51
Figure 2.3: Principle of the 2D contour following algorithm. 59
Figure 3.1: A simple object in a 4 by 4 by 4 array. 70
Figure 3.2: Facetted images produced from the slices of figure 1.2. 69
Figure 3.3: The result of contour following at the bone threshold of a CT slice from figure 1.2. 73
Figure 3.4: Marginal index representation of the object in figure 3.1. 73
Figure 3.5: The symmetric recursive representation of the object in Figure 3.1. 77
Figure 3.6: Neighbour coding in the Masked Binary Octree representation. 79
Figure 3.7: The min-max greyscale structure incorporates the minimum and maximum of children in its fields. 81
Figure 3.8: The digital sphere at increasing levels of resolution (tree depths five to eight). 82
Figure 3.9: The chequered sphere at resolution eight. 82
Figure 3.10: The binary skull at increasing levels of resolution (tree depths five to eight). 83
Figure 3.11: Two different methods for finding a neighbour. 92
Figure 4.1: Relationship of object space to image space for general (perspective) projection. 96
Figure 4.2: Relationship of object space to image space for orthographic projection. 98
Figure 4.3: Image to Object space relationship for the Reynolds algorithm. 107
Figure 4.4: The order of visit of octree nodes is determined by the viewing direction. 111
Figure 4.5: The Octree to Quadtree relations for the FTB algorithm. 116
Figure 4.6: Octree/quadtree definitions. 120
Figure 4.7: Schematic of the node-tracing principle - rays are subdivided at Partial octree nodes. 122
Figure 4.8: Quadtree projection of an octree. 125
Figure 5.1: Example of a lighting model. 129
Figure 5.2: The orientations of pseudo-normal vectors in the surface context description. 131
Figure 5.3: Voxel neighbour directions. 132
Figure 5.4: Voxel contexts. 133
Figure 5.5: Object based shading of the skull derived from the slices of Figure 1.2. 136
Figure 5.6: NMR image of the head, displayed with and without partial volume shading. ................................................................. 140
Figure 5.7: Node tracing for the partial volume shading. ................................................................. 141
Figure 5.8: Comparison of possible shades resulting from different image-based methods. ................................................................. 143
Figure 5.9: Isometric view of non-interpolated depth buffer of the digital sphere. 144
Figure 5.10: Isometric view of the interpolated depth buffer of the digital sphere. 145
Figure 5.11: Shaded displays of the digital sphere using interpolated and non-interpolated image-based shading. 146
Figure 6.1: Regions R1 to R4 drawn on the shaded display of a sphere in multiple views. ................................................................. 159
Figure 6.2: The cylindrical volumes generated from the regions of figure 6.1. ................................................................. 160
Figure 6.3: The resulting objects from intersection and difference of the sphere and the volume mask delineated in figure 6.1. 160
Figure 6.4: NMR data with three different thresholds, and a volume mask applied to visualise the brain and MS lesions. 162
Figure 7.1: Boolean operations on octree nodes. ................................................................. 165
Figure 7.2: Result of merging a sphere with a demarcated piece after translation. 167
Figure 7.3: A slice dissected from the binary digital sphere, and a rotation specified. ................................................................. 181
Figure 7.4: The result of dissecting, rotating and merging an arbitrary slice from a sphere. 181
Figure 8.1: Stages in simulating surgery on Patient 1. ................................................................. 191
Figure 8.2: Marking the translation vector for Patient 1. ................................................................. 192
Figure 8.3: The original and predicted hard tissues. ................................................................. 192
Figure 8.4: Four views of Patient 2, and the first VOI. ................................................................. 195
Figure 8.5: Modifying the first VOI for Patient 2. ................................................................. 196
Figure 8.6: A second VOI is generated to further modify the mandible. ................................................................. 196
Figure 8.7: Display of the mandible sectioned from Patient 2. ................................................................. 197
Figure 8.8: Marking the rotation for Patient 2. ................................................................. 197
Figure 8.9: The rotation as specified. ................................................................. 198
Figure 8.10: Relationship of rotated mandible to the original. ................................................................. 198
Figure 8.11: The Prediction and original compared. ................................................................. 199
Figure 8.12: Two views of Patient 3, and the creation of the first VOI. ................................................................. 201
Figure 8.13: Specifying the rotation of the maxilla for Patient 3. ................................................................. 202
Figure 8.14: Three views of the rotated maxilla of Patient 3. ................................................................. 203
Figure 8.15: The prediction after maxilla movement, and the interference volume. ................................................................. 203
Figure 8.16: Marking the second VOI for removal of the mandible. ................................................................. 204
Figure 8.17: The mandible and the difference volume from the maxilla prediction. ................................................................. 204
Figure 8.18: Specifying the rotation of the mandible for Patient 3. ................................................................. 205
Figure 8.19: The combined prediction and the interference volume of the mandible. ................................................................. 205
Figure 8.20: Comparison of the final prediction and the maxilla prediction. ................................................................. 206
Figure 8.21: Comparison of the final prediction and the original for patient 3. ................................................................. 206
Figure 8.22: Transverse and sagittal slices reformatted from the predicted data volume. ................................................................. 207
Figure 9.1: Soft and hard tissues resulting from the predicted movement of the patient from figure 1.2 ................................................................. 214
Figure 9.2: Slices reformatted from the prediction of figure 9.1. ................................................................. 215
Figure 9.3: Facet model tiled from the contours of figure 9.2 after manual editing. .......................................................... 216
Figure 9.4: Three sets of data from the patient of figure 1.2 .......................................................... 217
Figure 9.5: The result of registering the three data sets of figure 9.4. .............................................. 218
Figure 9.6: Visualising surface differences. .......................................................... 219
Figure 9.7: Difference images of the three data sets from figure 9.5 .............................................. 219
Figure A.1: A sequence of nodes that would be handled differently by BTF, FTB and node-tracing algorithms. .......................................................... 227
Figure A.2: The skull of section 3.3.3, rendered with the interpolative and non-interpolative node-tracing algorithm. .......................................................... 230
Figure A.3: The average visits to octree nodes for the binary sphere. .......................................... 231
Figure A.4: The average visits to octree nodes for the Chequered Sphere. ........................................... 232
Figure A.5: The average visits to octree nodes for the binary skull. .................................................. 233
Figure A.6: Percentage octree nodes unvisited for the binary sphere. ............................................ 236
Figure A.7: Percentage octree nodes unvisited for the Chequered Sphere. ........................................... 237
Figure A.8: Percentage octree nodes unvisited for the binary skull. ............................................ 238
Figure A.9: Logarithm of the average number of octree node visits per pixels for the binary sphere. .......................................................... 241
Figure A.10: Logarithm of the average number of octree nodes visited per pixel, for the Chequered Sphere .......................................................... 242
Figure A.11: Logarithm of the average number of octree nodes visited per pixel, for the binary skull. .......................................................... 243
LIST OF TABLES

Table 1.1 : Sources of volume data in medical imaging ........ 24
Table 3.1 : The number of nodes in the binary octree representations of the three objects described in section 3.3.3. ......................... 84
Table 3.2 : The number of nodes in the greyscale representation of the skull at resolution level 8. ........................................ 85
Table 4.1 : View code masks ............................................. 113
Table 4.2 : Comparative times for the BTF algorithm on the four different octree representations. ............................................. 114
Table 4.3 : Relation of octree to quadtree children for orthogonal projections. 121
Table 5.1 : The 16 possible neighbour states for direction East. .... 134
Table 5.2 : The bit patterns of the 18 1-neighbours for the East face of a voxel. 135
Table 7.1 : Rules for the three simplest Boolean operations. ...... 165
Table 7.2 : Comparison of primitive and combined manipulation operations. 168
Table 7.3 : Space complexities of the four octrees A, B, A ∩ B, A - B. .. 170
Table 7.4 : Comparative times for operations on (Unmasked) octrees A,B. 171
Table 7.5 : Comparative times for operations on Masked octrees A,B. .... 171
Table 7.6 : Comparative times for operations on the (Unmasked) octrees shown in figure 6.3. ......................................................... 177
Table 7.7 : Space complexities of the octrees Sphere ∩ VOI, Sphere - VOI, shown in figure 6.3, before and after translation. ................. 178
Table A.1 : The average visits to node for the node-tracing algorithm as a function of level. The figures count all visits - direct and indirect. 234
Table A.2 : Algorithm statistics for the skull in binary representation, at levels of resolution from 3 to 8. .............................. 239
Table A.3 : Algorithm statistics for the Node-Tracer on the greyscale octree representation of the skull. The additional overhead of the interpolative shading is about 10%. Visit count is for all visits - direct and indirect. ..... 244
GLOSSARY OF NOTATION

Set Notation

- Intersection
- Union
- Proper subset of
- Proper superset of
- Subset of or equal to
- Superset of or equal to
- Empty set
- Is an element of
- Scalar (dot) product
- Vector product
- Gradient operator
- For all
- Closed interval
- Open interval
- Semi-open interval

Other Notation

Page number of definition

CT Computerised Tomography 23
NMR Nuclear Magnetic Resonance 23
SPECT Single Photon Emission Computed Tomography 23
PET Positron Emission Tomography 23
TOFPET Time of Flight Positron Emission Tomography 23
DSR Dynamic Spatial Reconstructor 29
\( \mathbb{R}^3 \) Euclidean Space in three dimensions 41
\( U_c \) The real world, continuous Universe 42
\( r \) A point in \( U_c \) 42
(\( r_1, r_2, r_3 \)) Real triple representing \( r \) 42
(\( \mathbf{\alpha}, \mathbf{\beta}, \mathbf{\gamma} \)) Basis vectors of \( U_c \) 42
\( \Phi_c(\mathbf{q}) \) Continuous characteristic function of an object 42
\( f(r) \) Continuous density function 42
(i,j,k) Integer triple of a voxel 43
\( \delta \) Dimension of voxel 43
V Voxel 43
\( \psi(i,j,k) \) Vector associated with a voxel 43
\( \mathbf{r} \) Vector to the point at the centre of Object-Space 44
\( <n\text{-adj}> \) is \( n \)-adjacent to 44
\( \Phi_d(i,j,k) \) Discrete characteristic function of an object 44
\( Q_c \) Continuous object 44
\( Q_d \) Discrete object 44
D Digitisation of an object 44
DDS Digital Discrete Scene 46
\( \mathbb{R}^2 \) Euclidean space in 2D 46
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s$</td>
<td>A point in 2D space</td>
<td>46</td>
</tr>
<tr>
<td>$(s_i, s_j)$</td>
<td>Real doublet associated with $s$</td>
<td>46</td>
</tr>
<tr>
<td>$(\Phi, \Theta)$</td>
<td>Basis functions of 2D screen</td>
<td>46</td>
</tr>
<tr>
<td>$(i,j)$</td>
<td>Integer doublet of a pixel</td>
<td>46</td>
</tr>
<tr>
<td>$P$</td>
<td>Pixel</td>
<td>46</td>
</tr>
<tr>
<td>$\eta_i$</td>
<td>Dimension of pixel</td>
<td>46</td>
</tr>
<tr>
<td>$p(i,j)$</td>
<td>Vector associated with a pixel</td>
<td>46</td>
</tr>
<tr>
<td>$\Theta_c(s)$</td>
<td>Continuous characteristic function of an image</td>
<td>48</td>
</tr>
<tr>
<td>$I_d$</td>
<td>Discrete image</td>
<td>48</td>
</tr>
<tr>
<td>$I_c$</td>
<td>Continuous image</td>
<td>48</td>
</tr>
<tr>
<td>$B$</td>
<td>Binarisation operator</td>
<td>49</td>
</tr>
<tr>
<td>$b(i,j,k)$</td>
<td>Binary DDS value</td>
<td>49</td>
</tr>
<tr>
<td>$S$</td>
<td>Segmentation Operator</td>
<td>53</td>
</tr>
<tr>
<td>$1$-voxel</td>
<td>Binary voxel with value 1</td>
<td>55</td>
</tr>
<tr>
<td>$0$-voxel</td>
<td>Binary voxel with value 0</td>
<td>55</td>
</tr>
<tr>
<td>$Q$</td>
<td>Set describing an object</td>
<td>55</td>
</tr>
<tr>
<td>$Q$</td>
<td>Set describing the background</td>
<td>55</td>
</tr>
<tr>
<td>$&lt;\text{n-connected}&gt;$</td>
<td>is $n$-connected to</td>
<td>56</td>
</tr>
<tr>
<td>$\partial$</td>
<td>Boundary of</td>
<td>56</td>
</tr>
<tr>
<td>$&lt;\text{P-adj}&gt;$</td>
<td>is P-adjacent to</td>
<td>57</td>
</tr>
<tr>
<td>$\rho$</td>
<td>Density threshold</td>
<td>58</td>
</tr>
<tr>
<td>$\text{ROI}$</td>
<td>Region of interest</td>
<td>67</td>
</tr>
<tr>
<td>$\Psi$</td>
<td>An octree node</td>
<td>76</td>
</tr>
<tr>
<td>$P,F,E$</td>
<td>Partial Full and Empty codes for a node</td>
<td>77</td>
</tr>
<tr>
<td>$J$</td>
<td>Octal code</td>
<td>85</td>
</tr>
<tr>
<td>$J_{m,p}$</td>
<td>Octal code fragment from levels m to p</td>
<td>85</td>
</tr>
<tr>
<td>$</td>
<td></td>
<td>$</td>
</tr>
<tr>
<td>$c_m$</td>
<td>Element of an octal code</td>
<td>86</td>
</tr>
<tr>
<td>$\vec{a}$</td>
<td>Vector to the centre of a node</td>
<td>86</td>
</tr>
<tr>
<td>$E_m$</td>
<td>Size of the edge of a node</td>
<td>86</td>
</tr>
<tr>
<td>$\Omega$</td>
<td>A Quadtree node</td>
<td>87</td>
</tr>
<tr>
<td>$K$</td>
<td>Quad code</td>
<td>87</td>
</tr>
<tr>
<td>$K_m$</td>
<td>Fragment of quad code</td>
<td>87</td>
</tr>
<tr>
<td>$\vec{g}$</td>
<td>Vector to the centre of a quadtree node</td>
<td>87</td>
</tr>
<tr>
<td>$\beta$</td>
<td>Bit operator</td>
<td>91</td>
</tr>
<tr>
<td>$\text{HSE}$</td>
<td>Hidden Surface Elimination</td>
<td>95</td>
</tr>
<tr>
<td>$\text{ISC}$</td>
<td>Continuous Image-Space</td>
<td>95</td>
</tr>
<tr>
<td>$(\vec{\mathcal{A}^\prime}, \vec{\mathcal{A}^\prime})$</td>
<td>Basis vectors of Image-Space</td>
<td>95</td>
</tr>
<tr>
<td>$X^\prime,Y^\prime,Z^\prime$</td>
<td>Dimensions of Image-Space</td>
<td>96</td>
</tr>
<tr>
<td>$r^\prime$</td>
<td>Vector in Image-Space</td>
<td>97</td>
</tr>
<tr>
<td>$L$</td>
<td>Vector from centre of Object-Space to centre of Image-Space</td>
<td>97</td>
</tr>
<tr>
<td>$U_d$</td>
<td>Discrete Universe</td>
<td>98</td>
</tr>
<tr>
<td>$T$</td>
<td>Depth Preserving Transform</td>
<td>99</td>
</tr>
<tr>
<td>$PT$</td>
<td>Projection Transform</td>
<td>99</td>
</tr>
<tr>
<td>$PT^{-1}$</td>
<td>Inverse Projection Transform</td>
<td>99</td>
</tr>
<tr>
<td>$\vec{g}$</td>
<td>Vector normal to the screen</td>
<td>99</td>
</tr>
<tr>
<td>$\lambda$</td>
<td>Step along a ray, or depth from screen</td>
<td>99</td>
</tr>
<tr>
<td>$\text{Sh}_c(Q_c)$</td>
<td>Silhouette of a continuous object</td>
<td>99</td>
</tr>
<tr>
<td>$Z_c(Q_c)$</td>
<td>Depth Buffer image of an object</td>
<td>99</td>
</tr>
<tr>
<td>ShD</td>
<td>Discrete Silhouette</td>
<td>99</td>
</tr>
<tr>
<td>------</td>
<td>----------------------</td>
<td>----</td>
</tr>
<tr>
<td>Zd</td>
<td>Discrete depth buffer</td>
<td>99</td>
</tr>
<tr>
<td>DyC</td>
<td>Display function</td>
<td>99</td>
</tr>
<tr>
<td>BTF</td>
<td>Back to Front</td>
<td>100</td>
</tr>
<tr>
<td>FTB</td>
<td>Front to Back</td>
<td>101</td>
</tr>
<tr>
<td>BSC</td>
<td>Back Surface Cull</td>
<td>104</td>
</tr>
<tr>
<td>\Lambda</td>
<td>Decision criterion to terminate a ray</td>
<td>108</td>
</tr>
<tr>
<td>CP</td>
<td>Convex Polygon representing the silhouette of a node</td>
<td>111</td>
</tr>
<tr>
<td>\vec{E}</td>
<td>Vector to eye</td>
<td>129</td>
</tr>
<tr>
<td>\vec{L}</td>
<td>Vector to light source</td>
<td>129</td>
</tr>
<tr>
<td>\vec{H}</td>
<td>Highlight direction</td>
<td>129</td>
</tr>
<tr>
<td>\is</td>
<td>Light source intensity</td>
<td>129</td>
</tr>
<tr>
<td>\ia</td>
<td>Ambient intensity</td>
<td>129</td>
</tr>
<tr>
<td>\is_{surf}</td>
<td>Surface shade intensity</td>
<td>129</td>
</tr>
<tr>
<td>\ka</td>
<td>Ambient shading weight</td>
<td>129</td>
</tr>
<tr>
<td>\kd</td>
<td>Diffuse shading weight</td>
<td>129</td>
</tr>
<tr>
<td>\ks</td>
<td>Specular shading weight</td>
<td>129</td>
</tr>
<tr>
<td>RTP</td>
<td>Radiotherapy Treatment Planning</td>
<td>183</td>
</tr>
</tbody>
</table>
This work would not have been possible without three people: Dr. Alf Linney, Principal Physicist, and Professor John Clifton, Head of Department, of University College Hospital Medical Physics, and Professor James Moss, of University College Hospital Dental School. These three campaigned tirelessly for the original funds to set up a Computer Graphics group at Medical Physics, and thus created the facilities for the author to work in. Dr. Linney was the main driving force, and allowed me freedom to pursue those avenues I did. The success of the project has, I hope, proved a vindication of such trust. He also read drafts of several papers, and chapters of this thesis and pointed out many directions I had overlooked. Professor Moss also showed continual interest in the work, and by tying it to a real application prevented me from straying too far into Outer Space.

I wish to thank Mr. Andrew Todd-Pokropek for agreeing to be my supervisor, for supporting my attendance at international meetings, and for reading and commenting on several complete drafts of this thesis. I am also very indebted to Dr. Paul Tofts, of the National Hospital for Nervous Diseases, Queen Square, who supplied me with NMR data, and was very enthusiastic and encouraging throughout the project; without him I would never have written up this thesis.

At an early stage in this work I was lucky enough to be able to visit several leading groups in Medical 3D Imaging in the USA. I am very grateful for the time taken by Dr. J.K. Udupa of the Medical Image Processing Group (MIPG) at the University of Pennsylvania, Philadelphia, to discuss their approach, and later Professor Gabor Herman who I met several times and corresponded with about my work. In particular I would like to thank Dr. Donald Meagher who I visited at Phoenix Data Systems in Albany New York in 1985, and who also discussed his approach at length. Many of my ideas arose from this discussion, and subsequently when he visited me at UCL in 1988. His enthusiasm was a great encouragement. I also wish to thank Dr. Karl Heinz Höhne, and Herr Andreas Pommert of the University of Hamburg, for several illuminating discussions.
The patient data was supplied with permission by Professor James Moss, and Mr. David James of UCH Dental School. I am very indebted to time given and interest shown by Mr. Sherif Gayed, Oral Surgery Registrar, of UCH Dental School, who discussed the cases in Chapter 8 in detail, and showed great enthusiasm for the potential of the system, as well as pointing out its shortcomings.

The menu styles of the UCL3D system, the graphics library format and the user interaction with the system were all inspired by the in-house General Imaging Package (GIP) developed in the Medical Physics department over many years for handling sets of 2D medical images - in particular by Andrew Todd-Pokropek, Paul Tofts, Dave Plummer and Sue Grindrod. Sue was responsible for porting this entire package to the UCL3D environment, for handling all the patient data acquisition protocols, and for managing the system. Without her UCL3D would be a set of programmes with nowhere to go. Above all I wish to thank Mr. David Plummer whose thorough professionalism in Software Engineering, all-encompassing expertise, and unswervable calmness and optimism were both a source of problem-solving and a model to inspire.

A Ph.D. thesis is never finished - it only asymptotically approaches the end and is eventually cut off and submitted. A number of people, under no obligation to do so, read closer approximations to the finished product and encouraged the next iteration. It is with deep gratitude that I include in this category Dr. Gabriele von Voigt of UCL Computer Science, Dr. Paul Otto of Canon Research Ltd., and Professor Steve Pizer and Dr. Mark Levoy of the University of North Carolina at Chapel Hill. Most of all I wish to thank Mr. David Delpy of Medical Physics who proof read the entire document as an interested layperson, corrected countless inconsistencies, and claimed to have enjoyed reading it.

Last but not least I acknowledge the support of my friends and colleagues during the blacker moods that thesis writing induces - in particular, Mark, Corinna, Tim, Johnny, and Olga. Especially I thank my wife Marzena for putting up with my nocturnal existence, for constant encouragement, and for believing that it would end.
DEDICATION

To my teacher Sensei Minoru Kanetsuka, who told me: "Don't shrink!".
CHAPTER ONE

INTRODUCTION AND OVERVIEW

1.1 INTRODUCTION

This thesis describes work undertaken as part of a project to design, implement, and evaluate a software package to use interactive 3D graphics for Maxillo-Facial surgical planning. The requirements for such a system need to be carefully defined. The use of 3D graphics in medical imaging is comparatively new (about ten years old), but recently its growth has been rapid. This may be attributed partly to the increasing performance/cost capability of modern computer technology, and partly to the increasing familiarity of clinical users with the possibilities available, and a commensurate demand for more sophisticated techniques. An exhaustive search of the literature would produce an overwhelming quantity of references. A number of comprehensive review articles are available however, covering both the technical aspects [UDUPA87, HÖHNE87a] and the clinical applications [HEMMY87].

Since the commencement of this work (1984) to the time of writing (1989), a number of 3D medical graphics "solutions" have become available. The term "solution" is used here, rather than "package", "workstation", or "software" because of the wide variety of implementations and capabilities offered. These vary from compact and efficient software packages designed to run in the hardware of the data acquisition device [UDUPA86b, VANNIER83a], to portable software packages designed to run on stand-alone dedicated computing platforms [DEV85, ROBB87], or to special purpose hardware [FISHMAN87a, GOLDWASSER87, LENZ86, MEAGHER84d, JACKEL85]. A brief overview of other solutions will be given in section 1.4 of this chapter. In order to have a solution, a problem must be formulated, and this is also the purpose of this chapter. Most other work has concentrated extensively on the visualisation aspects of the problem. A useful clinical tool must, however, also offer facilities for the manipulation and quantification of 3D data. Whereas there has been work in this area,
the solutions provided have often been restricted in their capabilities, this restriction often stemming from a limitation in the representation of the data employed. It is the belief of the author that the methodology described in this thesis and its implementation, referred to as the UCL3D package, offer some capabilities in advance of other systems.

The foregoing introduction has used some terminology that must be properly defined. The remainder of this chapter will therefore describe:
1) A summary of general 3D graphics methodologies,
2) A summary of the data acquisition methods in medical imaging,
3) A summary of other approaches to the use of 3D graphics in medical imaging,
4) A formulation of the specific requirements for the UCL3D package.

1.2 3D GRAPHICS METHODOLOGIES

Looking at the general field of 3D graphics, including CAD/CAM/CAE, Art, and Flight Simulation, four major classes of data representation may be delineated. These are:

1) Boundary based (B-R)
2) Constructive Solid Geometry (CSG)
3) Parametric
4) Spatial Enumerative (Voxel)

The boundary based methods describe only a set of surfaces, in terms of faces, edges and vertices. A variety of representations are available in detail, and inter equivalence has been shown for many [NEWMAN81, FOLEY82]. Faces may be polygons, or for example be constructed from B-splines (in 2D), or Bezier patches (in 3D) designed to maintain smoothness at boundaries in some specified way. The advantages of this approach lie in the existence of very well understood algorithms for highly realistic images, and the availability of special purpose hardware for many of these [FUCHS88]. The principle disadvantage is the lack of an internal representation - if a surface is cut or partially removed, there is no information about what is inside.
The CSG representation is one composed of solid primitives, such as spheres, cylinders, cones and rectangular prisms. These are connected in a Boolean tree (usually a binary tree), which leads to general complex volumetric objects, which are amenable to attribute labelling as in 2D design graphics [CGA83b]. This means that subtrees of the objects may be "pickable" and have their attributes modified, in a similar way to many 2D graphic design packages. Again, CSG is a well understood method, and some attempts have been made at specific hardware improvements for it [MORRIS88]. In addition the solid primitives can have physical attributes such as density and elastic moduli, that allow the use of powerful numerical techniques such as Finite Element Analysis. The disadvantages of this representation are due usually to the crudeness and limited number of the original solid primitives.

The parametric technique represents the database as functions, either everywhere in space or as surfaces or other manifolds in space. This technique is useful in Graphic Art and Engineering design and in mathematical or scientific applications such as visualization of electron density maps in molecules [LEVOY88], Fluid Mechanics [UPSON89], or Finite Elements, but is limited in other areas unless used in combination with one of the other techniques.

The Spatial Enumerative technique describes space as a tessellation, (i.e. a continuous repetition at some spatial frequency), of unit cells, each containing attributes that may be very simple (a single flag to state the presence or absence of an object) or quite complex (a density value and/or a surface description in terms of one of the other representations). The simplest tessellation of space is provided by the cube, but other possibilities may be derived by any simple bilinear transformation (scaling, shearing, rotation) of a cube. The unit cells are often called voxels, short for volume elements in analogy to 2D pixels. Samet and Webber [SAMET88a], mention tessellations that are not based on the cube, but these have limited application

Many representations may be thought of as mixtures of the above. For example the octree encoding scheme [MEAGHER82] which will be discussed in detail, may be thought of as a restricted CSG tree where the primitives are only cubes of different
sizes; or they may be a high level representation with the "leaf" level composed of parametric surfaces or Boundary representation faces edges or vertices [CARLBOM85, SAMET88a].

1.3 MEDICAL DATA ACQUISITION METHODS

The field of Medical Imaging, as with other High-Technology areas, is one where new methodologies, at first innovative, rapidly become routine, and even mandatory in the sense that users, once aware of a possibility come to view it as essential for their ability to proceed. Medical images were originally limited to photographs, X-ray radiographs, or histological slides from a microscope. Therefore they were essentially two-dimensional, and recorded in an analogue fashion. Today a variety of technologies can produce two and three-dimensional data sets in machine readable fashion. This section briefly reviews the sources of such data.

1.3.1 Surface Data Acquisition

The primary source of data considered in this thesis is volume data. However, for the specific surgical planning application that is discussed, surface acquisition methods are also important. Patient photographs are one possible source of this data, used in conjunction with radiographs, and registered together [FANIBUNDA83]. Others that have been used are stereophotogrammetry [BURKE83] and Moiré fringes [KANAZAWA78, DRERUP80]. A system for deriving high resolution surface data, based on optical range finding techniques was first described by Arridge et al [ARRIDGE85]. This system is incorporated into the UCL3D facility and is illustrated schematically in figure 1.1. The subsequent development of this system has not been the primary work of the author and will not be discussed in detail. However it will be referred to again in Chapter 9, where the need to register surface and volume data is discussed.
1.3.2 Volume Data Acquisition

There are four major modalities for three-dimensional medical volume data:

1) X-ray Computer Assisted Tomography (CT).
2) Nuclear Magnetic Resonance Imaging (NMR, referred to in the American literature as MRI).
4) Diagnostic Ultrasound.

A detailed discussion of data acquisition and reconstruction algorithms is out of place here, (see for example the Proceedings of the IEEE special issue on Medical Imaging [PROCIIEEE83]). They may be thought of formally in the same way: they each give a set of samples of a function of interest, at a certain spatial and density resolution. Table 1.1 summarises the main characteristics of each modality (see also [FLYNN83, GOLDWASSER87]).
<table>
<thead>
<tr>
<th>Modality</th>
<th>Parameter measured</th>
<th>Matrix size</th>
<th>No. of scans</th>
<th>Typical resolution (mm)</th>
<th>Quantisation (bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CT</td>
<td>Attenuation coefficient of X-rays.</td>
<td>256 x 256 -&gt; 512 x 512</td>
<td>50-100</td>
<td>(0.5 -1) x (0.5 -1) x (1.5 - 5)</td>
<td>10 - 14</td>
</tr>
<tr>
<td>NMR</td>
<td>T1/T2 relaxation density of nuclei: H, F, Na, P.</td>
<td>256 x 256</td>
<td>25-50</td>
<td>1 x 1 x (3 -10)</td>
<td>10 - 12</td>
</tr>
<tr>
<td>SPECT, PET, TOTPET</td>
<td>Concentration of administered isotope</td>
<td>64 x 64 -&gt; 128 x 128</td>
<td>10-64</td>
<td>(5 - 20) x (5 - 20) x (5 - 20)</td>
<td>6 - 10</td>
</tr>
<tr>
<td>Ultrasound</td>
<td>Acoustic impedance</td>
<td>256 x 256 -&gt; 512 x 512</td>
<td>20-30</td>
<td>(1 - 3) x (1 - 3) x (1 - 3)</td>
<td>Binary</td>
</tr>
</tbody>
</table>

Table 1.1: Sources of volume data in medical imaging

Figure 1.2: A set of consecutive CT slices, from a patient with a cleft palate.

The method of acquisition may sample data in two, three, or four dimensions (the fourth dimension being time), followed by a reconstruction in two or three dimensions. Thus CT data is usually acquired by a projection obeying the 2D Radon Transform, and reconstructed, in two dimensions, by numerically approximating the inverse Radon.
Transform. These reconstructed samples are often referred to as slices, and the third dimension is acquired by moving the scanner or object along one dimension, called the coaxial dimension. The sampling in this dimension may be variable, and is typically much less than the 2D pixel resolution. A plane perpendicular to this direction is called a transverse plane. Some CT systems have been designed for ultrafast scanning, particularly with cardiac applications in mind. These may acquire either 2D [BOYD83] or 3D [HARRIS83] projection data, repeatedly over short times. The reconstruction technique is still two dimensional however. Figure 1.2 shows nine consecutive CT slices of the head from a sequence of 50 obtained from a patient admitted for maxillo-facial surgery at UCH. The slices numbered 23 to 26 show that the patient has a cleft palate. The difference in resolution is easily seen by displaying each slice at its location in a box bounding the whole object, as in Figure 1.3 (16 slices only are shown). Here the data has been density windowed so that only the CT values corresponding to bone are displayed.

Figure 1.3: 16 CT slices, from the same set as figure 1.2, located in their 3D relationship.
NMR machines have a different reconstruction strategy since the dataset acquired is the Fourier transform of the parameter of interest, and is reconstructed by a 2D inverse Fourier Transform. Here however, the slice thickness is still greater than the pixel resolution, since the detector must average in the third dimension. Figure 1.4 shows a set of NMR slices of the brain of a patient suffering from Multiple Sclerosis, supplied by Dr. Paul Tofts of the National Hospital for Nervous Diseases at Queen Square. Figure 1.5 shows 12 of these slices located into their 3D relationship, where here the brain has been segmented from the surrounding tissue by a contour following algorithm, described in Chapter 2 (section 2.3.2.2.1). The numbers on each slice are the location in millimetres of the slice in the third dimension. The slice labelled 66 has a one dimensional profile drawn on it.

SPECT and PET data are also obtained by projection, although attenuation and scatter limit the applicability of the Radon Transform. Reconstruction techniques may therefore be iterative. The projections are in 3D, but the reconstruction is usually in 2D because of the mathematical difficulties arising from an incomplete dataset [ROGERS87].
Ultrasound data is obtained in a linear scanning mode, and the image is produced directly. For this reason it is usually used as a real time modality, and comparatively little work on three-dimensional graphics has been applied to this field [BRACCINI89].

Another source of data that is receiving some attention is microscopy, where volume data is taken either by sets of thin slices cut from a solid object [SCHULZE86], or directly using modern confocal scanning techniques [FRENKEL89].

The illustrations in this thesis will primarily be from CT data since this is routinely used for the patients undergoing facial surgery, which is the main application to be discussed. Some illustrations from NMR data will be used however, to demonstrate shading and segmentation methods.
1.4 OTHER APPROACHES TO 3D MEDICAL GRAPHICS

This section will give a brief overview of previous work in the field of 3D medical imaging. More detailed discussion will be developed within separate subject headings in subsequent chapters. For this reason, some terminology will be introduced here without a precise definition, which will be deferred to a later discussion.

1.4.1 Work of Other Groups

It will be convenient to refer to different approaches with reference to a number of key groups. Principle among these are:

1) MIPG: The Medical Imaging Processing Group at the University of Pennsylvania at Philadelphia. This was originally at the State University of New York at Buffalo (SUNY). The director is Professor Gabor T. Herman, and this group is credited with the pioneering work in voxel databases. Their main development was a commercial package 3D83, later superseded by 3D98, written in FORTRAN and designed to run on the Data General Eclipse S/100 processor within the General Electric GE9800 CT scanner [ARTZY81, HERMAN83a, UDUPA86b].

2) Vannier and Marsh: At the Mallinckrodt Institute, St. Louis Missouri. This group developed the commercial package Siemens 3DCT, also written in FORTRAN and designed to run on the PDP 11/34 processor of the Siemens Somatron CT scanner. They also used a McDonnell Douglas D-100M CAD/CAM system for a limited type of interactive surgical planning [VANNIER83a, TOTTY84].

3) CEMAX: (formally Contour Medical Systems) at Santa Clara California. A commercial company marketing the CEMAX 1000, a MC68000 based computing platform, for interactive surgical planning [DEV84, FELLINGHAM86].
4) *Meagher*: Originally at Rensselaer Polytechnic (Albany, New York), where he developed the original *volume based* approach to 3D medical graphics. He subsequently developed the special purpose hardware system INSIGHT at Phoenix Data Systems, also at Albany. He is currently president of Octree Corporation (San Jose, California) [MEAGHER82, 84c,d, 85].

5) *Goldwasser and Reynolds*: Originally collaborating with MIPG, they subsequently went on to form a company, Dynamic Digital Displays 3D Inc. in Philadelphia based on the **VOXEL PROCESSOR** special purpose hardware [GOLDWASSER85, 86, 87].

6) *UNC*: The University of North Carolina at Chapel Hill. Henry Fuchs is a world expert in VLSI technology and responsible for the VLSI polygon renderer **PIXEL-PLANES** [FUCHS88]. Steve Pizer is primarily interested in image processing and segmentation and syntactic descriptions [PIZER86].

7) *Hamburg*: The University of Hamburg under the directorship of K.H.Höhne. This group pioneered work on greyscale raytracing techniques with 3D gradient operators for shading [HÖHNE86, 87a]. The commercial system marketed by **KONTRON** (Munich) is based on this group’s work.

8) *PIXAR*: This commercial company based at San Rafael California grew from the Lucas Film graphic art company responsible for *Star Wars* (the film). The system is a special purpose hardware device using an approach called *volume rendering*. Published details of the technology are limited, but applications have been reported [FISHMAN86]

9) *Mayo*: The Mayo clinic, Minneapolis, Minnesota. This group developed fast volume based methods to inspect the very large quantities of data from the **Dynamic Spatial Reconstructor** (DSR) [ROBB83]. They developed some special purpose hardware [HEFFERNAN85b], and a software package designed to run on a **SUN** colour graphics workstation [ROBB88]. This package, called
ANALYZE, is available free of charge to academic collaborators, and is also marketed by CTI (Knoxville, Tennessee).

There are a number of other academic and commercial groups, some of which will be referred to in subsequent chapters.

1.4.2 Overview of methodologies

It is clear that different problems require different solutions. The CAD/CAM/CAE or Graphic Art applications require objects specified to arbitrary accuracy by a human designer to be portrayed in as realistic a fashion as possible. The limitation is usually only computation time tolerance, and this may be much less important if a single picture or set of pictures is required, which are then to be reproduced and used as illustrations or advertising. The first three classes of representation of section 1.2 (page 20) are the most commonly used in these applications.

In the medical field however, the problem is very different. One is presented with a set of samples of a function to a limited accuracy, and the most difficult problem is to interrogate and interpret the data in order to define the physical objects that are present, (an essential caveat being that one must not derive resolution higher than the sampling rate); the display of these objects is then comparatively simple. All of the representations have been used to some extent in medical imaging, although the most common are the Boundary representation and the Spatially Enumerative methods, because of their natural correspondence to the data.

Since the earliest volume data methods acquired 2D slices, it was natural to display them as such. When 3D volume data became available, the display mode employed was still sequential display of slices, either side by side as in figure 1.2, or sequentially in time, (often called a movie). One of the first processing developments was multiplanar reformatting, either in orthogonal (sagittal and coronal) directions, or general oblique directions [HERMAN77]. Later, techniques for producing curved reformatting planes were developed [ROTHMAN86].
In order to visualise complex 3D relationships, a 3D display mode is required. This may be either true 3D, or 2D representations of 3D, sometimes called *2.5D displays*. The author prefers to use this latter term to refer to the types of depth encoded images used in computer vision, often involving stereoscopic views [HERMAN86].

True 3D displays use mechanical devices such as the *varifocal mirror*, that display either sequences of 2D raster images [HARRIS86], or random 3D vectors [KENNEDY87]. Holographic techniques have also been suggested [GREGUSS77]. Some authors conjecture an "ideal system", where true 3D display is combined with interactive tools, that respond to the density of the imagined object [HERMAN86]. Such developments must await improvements in hardware that may take many years.

The rest of this thesis will use only 2D displays of 3D objects, such as are shown in chapters 4 to 9, and these will be referred to as 3D displays. This type of display may be divided into *surface rendering* and *volume rendering* approaches. A distinction will be made between the *data* representation and the *object* representation. Following the terminology of the MIPG (see for example [HERMAN83a]), the data representation will be referred to as a *scene*. To produce an object from a scene requires a *segmentation* operation. The resultant object may have a volume representation or a surface representation. These terms will be given a more formal definition in chapters 2 and 3. The B-R representation is a surface representation, and CSG and Spatial Enumerative approaches are volume representations; the parametric representation may be either.

Surface representations may be surface rendered, and volume representations may be either surface or volume rendered. Strictly, volume rendering will be used to refer to display processes operating on scene representations, rather than object representations; i.e. a volume rendering operation misses out the segmentation stage. A discussion of volume rendering is given in section 4.4.2.3. Figure 1.6 shows a schematic of the relationships between these concepts.

The development of different approaches has, historically, been largely dictated by the limited computer resources available. The first 3D medical images were derived using
the Boundary representation (B-R) method [KEPEL75, FUCHS77, CHRISTIANSON78, SUNGUROFF78, COOK83]. The procedure employed is to segment in each slice to produce a set of contours. The set of contours may be displayed as a 1D primitive, or connected in some way. The resultant "wire-frame" image can then be rendered in well understood ways. The polygonal faces are most usually triangles, and are very often referred to as "facets".

For various reasons the B-R method may be inadequate. Intuitively it is unsatisfactory to assign "hard" surfaces between sample points, and in particular there will be some arbitrariness as to what points are connected along edges, dependent on the algorithm...
employed. Nevertheless the B-R approach gives highly realistic pictures (see for example figure 3.2, in Chapter 3).

In some cases however it is definitely inappropriate to use the B-R approach. For example in the complex branching structure of the facial skeleton, or in the visualisation of vascular beds [PARKER86] where decisions about directions of branching are almost impossible to make automatically without a very sophisticated model-based approach. For these reasons, among others, the Spatial Enumerative (voxel) approaches were developed, originally by MIPG in what Herman called the Cuberille Environment (a binary digital array) [HERMAN77, 79]. Because this group’s original interest was in the display of surfaces in reasonable times on small computers, the first approaches in the Cuberille Environment were also surface based. These used an elegant and ingenious algorithm for extracting lists of faces only of those voxels that were on the surface, in the sense that they had no immediate neighbour in the direction of view [ARTZY81, HERMAN83b]. These programs allowed the display of much more complex structures, and had the advantage of being "unambiguous" in the sense that they only displayed data that represented the presence or absence of the object within a voxel. In fact there still existed some algorithm-dependent structure; in particular the method of defining the object which could depend on the threshold chosen or on the behaviour of some surface tracking or region growing algorithm. Also, the necessary interpolation step which compensates for the non-cubic nature of the original voxel data due to undersampling in the coaxial direction can be implemented in several ways. The MIPG methods are discussed in more detail in sections 2.3.2.1 (segmentation), 3.1.2 (representation) and 4.4.1.2.2 (display).

Thus this first approach in the voxel environment only stored surface elements, although beginning with the full 3D scene. For some applications however such surface techniques are not satisfactory. In particular if it is required to mask and cut away parts of volumes, either as a diagnostic aid to observe interior structures, or as a surgical simulation procedure, it is essential to have a volume representation. At first, the volume representations used were binary, again due to the restriction on computer memory. Various data structures have been proposed, ranging from direct use of a 3D
array [FREIDER85], to octree methods and variations [MEAGHER82], or types of run-length encoding [TRIVEDI85, REYNOLDS87].

By contrast, volume rendering techniques maintain all the original data, and do not attempt to make object classifications in advance. Instead, it is possible to perform ray-tracing, (or as Hohne [HÖHNE87] prefers to term it ray-casting), to model light traversing through the data. The first developments in this approach (by the Mayo group), were quite early, and consisted of reprojection of the data, (in this case CT), to give the impression of a radiograph [HARRIS79]. The advantages were that, firstly, the data could be viewed from any number of directions without any extra radiation dose to the patient, and secondly, that certain ranges of intensity could be "dissolved" away, to give images of, for example, soft tissue without any bone present.

Lately, due to the greater availability of processor speed and memory, these methods have been extended to incorporate greater graphic realism. These methods may be divided into those that perform a dynamic segmentation to detect a surface, and then surface render it, and those that actually integrate a property along the ray and leave the segmentation to the visual perception of the observer. In the former, a surface will correspond to the point where the ray encounters a voxel that satisfies a decision function, which may be simply a threshold [HÖHNE87], or a more sophisticated approach that attempts to detect the connectivity of the surface dynamically [TROUSSETT88]. In the latter, some "fuzziness" is included, so that sharp object boundaries are not required [LEVOY88]. Actually, it turns out that some classification of voxel types is still required, based on a priori knowledge, and the resulting displays are very sensitive to this classification.

As experience has grown of the types of displays that are possible, more emphasis has been laid on the need for an interactive capability. This is important both to allow the inspection of large quantities of data, (in effect to aid the observer in an intuitive segmentation), and to model the cutting away or opening out of the data, as if performing surgery. This latter manipulation capability is much harder to achieve and will be discussed at length in this thesis. Even the former capability, the need for real
time display, is difficult to achieve, and has lead to the development of special purpose hardware.

1.5 FORMULATION OF THE PROBLEM

The application to be considered here is interactive surgical planning - specifically as applied to maxillofacial surgery. The data may come from a variety of sources, and may be collected at different times. Some of the processes that the user will need to carry out will be:

1) Acquisition
2) Preprocessing
3) Segmentation
4) Visualisation
5) Manipulation
6) Quantification
7) Registration

1.5.1 Acquisition

The data acquisition stage involves one or more of the modalities described in section 1.3. For any given application the detailed data acquisition will require a protocol. This phase will not be discussed further. It will be assumed that the data is available in machine readable format, at a certain resolution. The volume data is in the form of a 3D matrix containing densities representing the estimate of some continuous function of interest, undersampled in the coaxial dimension. Surface data is also in machine readable format, although its resolution is more complex (i.e. usually spatially varying).
1.5.2 Preprocessing

A preprocessing stage may be required
i) to remove artifacts
ii) to interpolate the data in the axis of undersampling
iii) to filter the data if it is noisy
iv) to average the data, if there is too much or if only a coarse resolution is required.

1.5.3 Segmentation

Since the requirement is to manipulate objects as if they were a solid, a segmentation stage will be required. By segmentation will be meant the process of defining an object within the acquired data set. The notion of object itself will be precisely defined in Chapter 2.

1.5.4 Visualisation

Visualisation is the most developed graphics problem. Commencing with an Object in a consistent representation, the problem is to derive the image on a 2D screen that would be obtained if the object were a real one, with well-defined surfaces, and optical properties, being illuminated from some direction(s) and viewed by, for example, a television camera. The whole spectrum of graphics techniques can be (and has been) brought to bear on this problem: shading, depth cueing, parallax movement, shadowing, translucency, specular reflectivity, stereoscopic views etc. The essential question is to what degree sophisticated images enhance the information, at what cost?

1.5.5 Manipulation

This problem involves the interactive dissection of the defined objects, their independent transformation (translation, rotation, scaling, shearing), and the merging of separate parts. This was considered the primary goal, and the current superiority, of the UCL3D package. The methodology used is similar to CSG. That is, the dissection and merger is described by Boolean operations between objects. In UCL3D however, the
objects are octree representations of 3D arrays. In principle other representations could be employed.

1.5.6 Quantification

Ideally the user should be able to measure distances and angles between points, length, areas, and volumes of objects. Some of these will require connectivity knowledge of the object.

1.5.7 Registration

In 3D the requirement is to define a common coordinate system for the data describing the same object, but taken at different times, or with different methodologies. This problem is beginning to receive attention in the literature. Chapter 9 discusses some approaches that have been taken.

1.6 OVERVIEW AND THESIS ORGANIZATION

UCL3D is a software package designed to meet the requirements set out in the previous section. It is implemented on a 32 bit super-minicomputer (Norsk Data ND540) with 5MB RAM and 300MB hard disc, DMA interfaced to a 24 bit plane ("true colour") graphics framestore (GEMS33). The system is implemented as a small number of independent programs, running under a menu driven user environment. The software comprises about 30,000 lines of FORTRAN, and about 10,000 lines of a proprietary system level language similar to Pascal. In addition, about 300 lines of Assembler were incorporated, to improve the Gems graphics drivers. The system configuration is shown in figure 1.7. All of the 3D graphics displays used as illustrations in this thesis were produced using this system.

The software was implemented over a two year period from 1985 to 1987, as a DHSS funded project to investigate the application of Computer Graphics to Surgical Planning. The system was "frozen" in July 1987, to allow a fixed system to be evaluated in a separately funded clinical evaluation. Subsequent development by the author has taken
place on other systems, in particular a network of SUN workstations. Some of the results to be presented were obtained on this system.

The software uses both surface and volume representations of objects, and uses surface rendering for the 3D graphics displays. Objects can be dissected in arbitrarily complex ways, and multiple objects can be held in memory. These fragments can be translated and rotated and merged by Boolean operations. Quantitative measurements are possible. A full 2D display and manipulation program can operate either on original data slices, or on slices reformatted from dynamically created objects. Figure 1.8 shows the overall software strategy. The work of the author was primarily in the manipulation and visualisation areas of this system.

The thesis is organised as follows:

---

Figure 1.7: System diagram of the UCL3D hardware.
Chapter 2 covers notation and basic results for digital scenes, and describes the preprocessing and segmentation stages.

Chapter 3 describes in detail the different representations possible for 3D medical images, and outlines those used in UCL3D. In particular the octree representation used for the manipulation of 3D images is described.

Chapter 4 explains how visualisation is performed for data in the representations described in chapter 3 and introduces a new algorithm for obtaining the quadtree representation of the orthogonal projection of an octree.

Chapter 5 describes shading techniques derived for 3D objects, including an interpolative depth buffer shading that improves on previously reported methods.
Chapter 6 discusses the techniques developed for object dissection. The concept of a generalised volume mask is introduced here.

Chapter 7 explains the methods for transformation and merger, based on Boolean operations between octree descriptions of the objects of interest.

Chapter 8 shows the application of UCL3D to surgical planning for three specific cases evaluated using this system. The cases are chosen to demonstrate the flexibility of the UCL3D approach.

Chapter 9 summarises the original contributions of the author, and describes additional topics. These include the quantitative capabilities of UCL3D, registration, and assessment of predictive surface displacement. A number of suggestions for future work are made.
CHAPTER TWO

SCENE NOTATION AND SCENE PROCESSING

2.1 DIGITAL SCENES

2.1.1 Definitions

In order to discuss the processing of digital scenes it is necessary to define their relation to the continuous world that they are supposed to represent. In this chapter the notation necessary for describing 3D objects is introduced, and the segmentation process is described. The notation and concepts are derived in part from MIPG publications [e.g. SRIHARI81, HERMAN83a, UDUPA87]. Notation for describing imaging is introduced later as required. In general the same terminology can be used for the real, continuous world and for the discrete digital representation. The terms continuous and discrete will sometimes be left out, if the context makes it clear which is being discussed. The discussion of discrete volume data can be extended to other applications, such as Geophysics [WOLFE88], or wave-mechanics [LEVOY88]. As explained in Chapter 1, a distinction is made between scene representations and object representations. The latter is the subject of the next chapter.

An n-dimensional digital image is a representation of a continuous n-dimensional vector space. The discussion will be restricted to the case where n is two or three. The inclusion of time extends the concepts to four dimensions [HERMAN78, HEFFERNAN85b], and in principle, other dimensions (e.g. scale space, section 2.3.4) could be included. Udupa et al have discussed the general case [UDUPA82]. For the remainder, it is assumed that the continuous scene being modelled is a region within the three-dimensional Euclidean space \( \mathbb{R}^3 \). It is assumed that all the "objects" (the precise notion of an object will be given below) of interest are within a rectangular parallelepiped region of dimensions X by Y by Z, called the continuous Universe \( U_C \).
A point in Uc is described by a vector $\mathbf{r}$ that is a triple of real numbers ($r_1, r_2, r_3$), with respect to a Cartesian coordinate system ($\hat{e}_1, \hat{e}_2, \hat{e}_3$). Thus $\mathbf{r} = (r_1 \hat{e}_1, r_2 \hat{e}_2, r_3 \hat{e}_3)$. Points, lines, surfaces etc. defined in this coordinate system are said to be defined in Object-space.

It simplifies the discussion if the origin of Object-space is at the centre of the Universe. Then the Universe is bounded by six planar half-spaces:

$$
\begin{align*}
| 0 | & = X/2 & | 1 | & = X/2 \\
| 1 | & & | 0 | & \\
\end{align*}
$$

$$
\begin{align*}
| 0 | & = Y/2 & | 1 | & = Y/2 \\
| 1 | & & | 0 | & \\
\end{align*}
$$

$$
\begin{align*}
| 0 | & = Z/2 & | 1 | & = Z/2 \\
| 1 | & & | 0 | & \\
\end{align*}
$$

where the direction normals are outward from Uc; i.e. any point $\mathbf{r}$ in Uc satisfies $\mathbf{r} \cdot \hat{a}_i < d_i$ for each of the six direction normals $\hat{a}_i$ with associated constants $d_i$ of (2.1).

The notion of Object must be carefully defined. In space-planning and machine-vision applications an object is a physical reality with precise dimensions. In CAD applications an object is imaginary, but is still the representation of a (potentially) physical object. The object is binary in the sense that there exists a characteristic function $\Phi_c(\mathbf{r})$, everywhere in (continuous) object space, such that:

$$
\Phi_c(\mathbf{r}) = \begin{cases} 
1 & \text{if } \mathbf{r} \text{ is inside the object}, \\
0 & \text{if } \mathbf{r} \text{ is outside the object} 
\end{cases}
$$

(2.2).

In medical imaging applications, and also in geophysical and wave-function applications among others, there is instead a continuous density function $f(\mathbf{r})$, representing a physical property of interest. Two different approaches may be used to define such an object. The first is to carry out a segmentation on the density volume, that then allows a characteristic function to be defined. The nature of the segmentation operation will be discussed in section 2.3. Objects defined in this way will be referred to as hard objects. A second approach which is gaining more use at present is directly to visualise the density volume, and postpone the "segmentation" to the visual system of the observer's
brain. Objects defined in this way are referred to as soft objects. In the Surgical Simulation application, as explained, the requirement is not just visualisation, but also space-planning and CAD related techniques. Therefore objects will be defined in a hard way. It is a feature of the UCL3D system that the definition types can be mixed to a certain extent. This point will be returned to in chapters 6 and 7.

In terms of a computer representation of the 3D scene a digitisation step is carried out. For medical data acquired in a digital format this stage occurs in the procedures for data acquisition and image reconstruction. A limit is placed there on the spatial resolution. Thus a continuous scene is represented by a discrete lattice. This is described by partitioning \( \mathbb{U}_C \) with sets of planes orthogonal to the coordinate axes \( (\delta_x, \delta_y, \delta_z) \). We will assume that the partitioning planes are evenly spaced, but not necessarily at the same spacing in each dimension. In the UCL3D system, the partitioning planes may be at variable spacing, in which case the scene is resampled onto an evenly spaced lattice. The result is an \( L \times M \times N \) array of rectangular parallelepiped cells called voxels. A voxel is labelled by an integer triple \( (i,j,k) \), \( 0 \leq i < L \), \( 0 \leq j < M \), \( 0 \leq k < N \), also referred to as a digital point. The dimensions of a voxel are \( \delta_1 \times \delta_2 \times \delta_3 \), \( \delta_1 = X/L, \delta_2 = Y/M, \delta_3 = Z/N \). The digital point \( (i,j,k) \) is related to a voxel \( V \) in continuous space by associating with it a vector \( v(i,j,k) \) that lies at the centre of \( V \). Then \( V \) contains all points \( r \) such that:

\[
\begin{align*}
v_1 - \frac{1}{2} \delta_1 & \leq r_1 < v_1 + \frac{1}{2} \delta_1 \\
v_2 - \frac{1}{2} \delta_2 & \leq r_2 < v_2 + \frac{1}{2} \delta_2 \\
v_3 - \frac{1}{2} \delta_3 & \leq r_3 < v_3 + \frac{1}{2} \delta_3
\end{align*}
\] (2.3a)

In set-theoretic terms:

\[
V = \{ r \mid r_1 \in [v_1-1/2 \delta_1, v_1+1/2 \delta_1), r_2 \in [v_2-1/2 \delta_2, v_2+1/2 \delta_2), \\
r_3 \in [v_3-1/2 \delta_3, v_3+1/2 \delta_3) \}
\] (2.3b)

where \([...)\) refers to a semi-open interval. The vector \( v \) itself is defined as

\[
v(i,j,k) = \begin{vmatrix}
(i + 1/2)\delta_1 \\
(j + 1/2)\delta_2 \\
(k + 1/2)\delta_3 \\
\end{vmatrix} - \mathcal{L}
\] (2.4)
where \( \mathbf{r}_{c} = (X/2, Y/2, Z/2) \). This formulation is slightly cumbersome but it simplifies the digital representation of object space if all voxels are labelled with non-negative integer triples, whilst continuous object-space has its origin at the centre of the Universe. The vector is a *discrete vector* if it belongs to the set of vectors from the centre of one voxel to another. As before, where confusion does not arise, a digital point will also be called a voxel. The term *continuous voxel* will be used if a distinction is required.

*Connectivity* is important in many methods. Two voxels \( A \) and \( B \) are *2-adjacent* if they share a common face, *1-adjacent* if they share an edge and *0-adjacent* if they share a vertex. The notation \( A <n\text{-adj}> B \) describes this adjacency. It is a symmetric relation, \( (A <n\text{-adj}> B \Rightarrow B <n\text{-adj}> A) \). Also \( A <n\text{-adj}> B \Rightarrow A <m\text{-adj}> B \) if \( n > m \). Any voxel has six 2-adjacent neighbours, eighteen 1-adjacent, and twenty six 0-adjacent neighbours (the Universe is formally considered as being surrounded by a layer of 0-voxels). Sometimes the term face-adjacent or 6-adjacent is used for 2-adjacent, edge-adjacent or 18-adjacent for 1-adjacent, and vertex-adjacent or 26-adjacent for 0-adjacent. Srihari [SRIHARI81] calls 2-, 1-, 0- adjacencies 1-, 2-, 3- adjacencies respectively. The author prefers the notation used here, because it is naturally extended to other dimensions of space.

### 2.1.2 Digitisation

The notion of digitisation is of importance in space-planning and machine-vision applications. A *digital object* is a representation of a continuous object. Therefore it has a *discrete characteristic function* \( \Phi_{d}(i,j,k) \) such that

\[
\Phi_{d}(i,j,k) = \begin{cases} 1 & \text{if } (i,j,k) \text{ is inside the object,} \\ 0 & \text{if } (i,j,k) \text{ is outside the object} \end{cases}
\] (2.5).

If \( Q_{c} = \{t \mid \Phi_{c}(t) = 1\} \) is the continuous object, and \( Q_{d} = \{i,j,k \mid \Phi_{d}(i,j,k) = 1\} \) is the digital object then \( Q_{d} = D(Q_{c}) \) defines the *digitisation*, \( D \) of \( Q_{c} \). The type of digitisation depends on the mapping of \( \Phi_{c}(t) \) to \( \Phi_{d}(i,j,k) \). For example:

\[
D_{1} \quad \Phi_{d}(i,j,k) = \begin{cases} 1 & \text{if } V(i,j,k) \cap Q_{c} = V(i,j,k) \\ 0 & \text{otherwise} \end{cases}
\] (2.6)
(the digitisation requires every point in the associated voxel to be inside $Q_c$),
\[ D_2 \Phi_d(i,j,k) = 1 \text{ if } V(i,j,k) \cap Q_c \neq \emptyset \]
\[ \Phi_d(i,j,k) = 0 \text{ otherwise} \] (2.7)

(the digitisation requires any point in the associated voxel to be inside $Q_c$),
\[ D_3 \Phi_d(i,j,k) = 1 \text{ if } v(i,j,k) \text{ is contained in } Q_c \]
\[ \Phi_d(i,j,k) = 0 \text{ otherwise} \] (2.8)

(the digitisation requires the centre of the associated voxel to be inside $Q_c$),
\[ D_4 \Phi_d(i,j,k) = 1 \text{ if } V(i,j,k) \cap Q_c \text{ contains 50\% or more of points in } V(i,j,k) \]
\[ \Phi_d(i,j,k) = 0 \text{ otherwise} \] (2.9)

(the digitisation requires the average number of points in the associated voxel that are inside $Q_c$ to be greater than 50%).

$D_1$ is often used in space planning where it is essential that no interference between objects can take place. However it has the disadvantage that the digitization of the complement of $Q_c$ will be $D_2$. This will effect the validity of Boolean complement operations such as are described in Chapter 7, for the UCL3D package. $D_4$ is the digitisation implicit in many medical imaging applications where the term Partial volume, is used to estimate the actual percentage of an object in a voxel. Many applications [e.g. WENG87], suggest using $D_3$ as a computationally efficient way of approximating $D_4$. This acts like a 3D sampling function [GONZALEZ87]. A fuller discussion of digitisation is given by Srihari [SRIHARI81], and is also mentioned by Yau [YAU84]. Notice that the above discussion has still not defined how the object is segmented from the background. The characteristic function $\Phi_d$ is simply a 3D bitmap (in discrete space), that labels each digital point as belonging to the object or not, and $\Phi_c$ in continuous space is a function labelling each real point as inside or outside the object. The procedure for setting up the characteristic function is the segmentation stage.

2.1.3 Quantisation

As well as a digitisation stage, the computer representation of density volumes requires a quantisation stage. Again, in medical applications, the quantisation is usually carried out by the data acquisition stage. Typically the data is presented as a 12-bit
integer, with a conversion scale to a possibly real valued parameter. The value of a digital point \((i,j,k)\) is a functional of the density function within the associated voxel \(V(i,j,k)\). By analogy with the digitisation stage, some alternatives exist for the mapping of density to value. For example it may be the density value at the centre point of the voxel, or an average over the whole voxel. The second is the quantisation implicit in most data acquisition methodologies.

A discrete scene is the representation of a continuous scene in a discrete lattice. A digital discrete scene (DDS) is a discrete scene with a quantised value associated with each digital point. If the value of a digital point is multivalued the scene is called a grey scene. If the digital points represent only zero and one then the scene is a binary scene. For example if a characteristic function \(\Phi_b\) is derived for a DDS it may be represented as a binary scene. Figure 2.1 summarises the overall schema.

2.1.4 Screen definitions

The continuous screen is a region within two-dimensional Euclidean space \(\mathbb{R}^2\), bounded by a rectangle of dimension \(U\) by \(V\). A screen point is described by doublet \(s = (s_1, s_2)\) with respect to a 2D coordinate system with basis \((\mathbf{A}, \mathbf{A}') = (\mathbf{A}', \mathbf{A'})\). The screen is partitioned by lines parallel to the basis vectors into \(M\) by \(N\) pixels. A pixel is labelled with an integer doublet \((i,j)\) \(0 \leq i < M\), \(0 \leq j < N\) - a digital screen point. A pixel \(P(i,j)\) is strictly a continuous region of 2D space of dimensions \(\eta_1 = U/M\) x \(\eta_2 = V/N\) with a vector \(p(i,j)\) located at its centre. Then \(P\) contains all points \(s\) such that:

\[
\begin{align*}
    p_1 - 1/2 \eta_1 & \leq s_1 < p_1 + 1/2 \eta_1 \\
    p_2 - 1/2 \eta_2 & \leq s_2 < p_2 + 1/2 \eta_2
\end{align*}
\]

In set-theoretic terms:

\[
P = \{s \mid s_1 \in [p_1 - 1/2 \eta_1, p_1 + 1/2 \eta_1), s_2 \in [p_2 - 1/2 \eta_2, p_2 + 1/2 \eta_2)\}
\]

with \(p\) defined as

\[
p(i,j) = \frac{|i + 1/2| \eta_1 - |U/2|}{|j + 1/2| \eta_2 - |V/2|}
\]

(2.11)
Figure 2.1: Overall scene relationships.
The term pixel will sometimes be used for the digital point \((i,j)\), where confusion does not arise (cf voxel above).

In the same way as an object was defined as a binary function in 3D space, an *image* is defined as a region of 2D space. It has a 2D characteristic function \(\Theta_c(s)\) such that:

\[
\Theta_c(s) = \begin{cases} 
1 & \text{if } s \text{ is inside the image}, \\
0 & \text{if } s \text{ is outside the image}. 
\end{cases}
\] (2.12)

Then a *digital image* \(I_d\) is a digitisation of a continuous image \(I_c\), expressed by \(I_d = D(I_c)\), and \(I_d\) has a discrete characteristic function \(\Theta_d(i,j)\). The definitions of these concepts are similar to those for 3D. Notice that in a discrete image, 0-adjacency of pixels is also called 8-adjacency, and 1-adjacency of pixels is also called 4-adjacency.

### 2.2 SCENE PROCESSING

Some processes are carried out on scenes. These include sub-regioning, binarisation, interpolation, and segmentation. The first three operate on a scene to produce another scene. Segmentation operates on a scene to produce an object. This is such a large topic that it will be dealt with separately in section 2.3.

#### 2.2.1 Sub-regioning

A scene is formed as a subset of the voxels in the original scene. For example a *slice* may be taken by reformatting:

\[
\begin{align*}
\text{z-slice}_K &= \{ V(i,j,k) \mid 0 \leq i < L, 0 \leq j < M, k = K \} \\
\text{y-slice}_J &= \{ V(i,j,k) \mid 0 \leq i < L, j = J, 0 \leq k < N \} \\
\text{x-slice}_I &= \{ V(i,j,k) \mid i = I, 0 \leq j < M, 0 \leq k < N \}
\end{align*}
\] (2.13)

More flexibly, a two-dimensional digital image may be formed by any oblique slice through the data or even through a curved plane. The technique appropriate to octree
data has been described by Yau [YAU84]. The current implementation of UCL3D produces only orthogonal (i.e. coronal, sagittal, and transverse) reformatting.

Another meaning of sub-regioning is to choose a three dimensional Volume of Interest (VOI), within the data. For example, the MIPG packages allow a rectangular parallelepiped sub volume to be specified, which allows a simple form of spatial dissection.

Another operation that is sometimes required is to reduce the resolution of the scene, if, for example the system has a limited memory [GOLDWASSER86]. This can be achieved by averaging over a block of voxels.

2.2.2 Binarisation

This is the process of producing a binary DDS from a grey DDS. Any segmentation that effectively defines a hard object will have a characteristic function \( \Phi_o \) that can be represented in a binary DDS. However some 3D methodologies work entirely in binary DDS structure (MIPG, Vannier and Marsh), including the segmentation algorithms. In this sense binarisation means a mapping \( B \) on the voxel values that reduces them to binary. The simplest operation is thresholding at a value \( \rho \):

\[
\text{b}(i,j,k) = 1 \text{ if } \text{value}(i,j,k) \geq \rho
\]
\[
\text{b}(i,j,k) = 0 \text{ otherwise} \tag{2.14}
\]

In early work, which was primarily interested in bone tissues from CT data, such a mapping was sufficient. A simple extension is to allow a density window in the grey values

\[
\text{b}(i,j,k) = 1 \text{ if } \text{value}(i,j,k) \geq \rho_{\text{low}} \text{ and } \text{value}(i,j,k) \leq \rho_{\text{high}}
\]
\[
\text{b}(i,j,k) = 0 \text{ otherwise} \tag{2.15}
\]

This is used in the binary implementation of UCL3D, in common with in other systems [e.g. TROUSETT88].

Since the function \( b(i,j,k) \) is a binary function defined for each voxel, it is an example of a characteristic function \( \Phi_o \) defining an Object. The term binarisation is used to describe an operation mapping a scene to a scene. The term thresholding will be used
to describe the mapping of a scene to an object. This is an example of a segmentation operation, described in section 2.3.1 below.

2.2.3 Interpolation

This is the process by which a scene with different voxel dimensions is produced from the original scene. It is most usually done because data acquisition methodologies produce data that is more sparsely sampled in one direction than another. Often this is done just to reduce the "blockiness" of derived images. Many topological results utilised in the MIPG approach require cubic voxels [HERMAN79]. Usually nearest neighbour [VANNIER83, LENZ86], or linear [GOLDWASSER87, HERMAN88b] interpolation is carried out in the undersampled direction, to give the same sampling as in the other directions. Linear interpolation is that used in UCL3D.

A correct analysis of interpolation procedures necessitates consideration of the actual sampling process of the data acquisition. Ideally, data would come from a band-limited object that was sampled at or above the Nyquist frequency. Then the original space function $f(r)$ can really be recovered to any accuracy using, ideally a low-pass filter in the frequency domain [PARKER83]. Since this low-pass filter is a sinc function in the spatial domain, and therefore strictly infinite, some approximations to it are required. It is this approximation that gives rise to nearest neighbour or linear interpolation as mentioned. Some investigations have been made using higher order interpolants, such as cubic splines [PARKER83] and restoring splines [LEE83]. In practice the real function $f(r)$ is not of course band-limited, but its convolution with the point-spread function (PSF) of the imaging device will be. This suggests that a correct interpolation, to the resolution of the acquisition device is indeed possible if the PSF is known. Unfortunately the correct sampling in the coaxial direction would require overlapping slices that (for CT) impart an unnecessarily dangerous radiation dose to the patient and, secondly, take a long time. Experience on phantoms with such overlapping slices shows the improvements that may result [POMMERT89b]. An indication of the "interpolation problem" can be seen from figure 2.2, where a dry skull was scanned at maximum resolution (1.5mm) in the coaxial direction (pixel size 0.8mm square). The left hand images show the object segmented from the (interpolated) scene produced
from these slices, while the right hand images show the same object reconstructed from every alternate slice. The linear interpolation of the sharp edges produces an unrealistic "steppy" effect.

The interpolation schemes mentioned above operate on a grey DDS to produce a grey DDS. However, a different approach is to interpolate in a binary DDS to produce a binary DDS. Here an attempt is made to interpolate the shape of the object \(Q_c\). The idea is to assign a distance map to the segmented regions in 2D, where a pixel is labelled with its distance from the boundary. Then this distance map is interpolated, and new 2D regions created. A quantitative study of this technique where slices were deleted from the original scene, and recreated by interpolation, found a much greater accuracy over the grey scene methods [RAYA88]. Such evaluations are very dependent on the type of data, and further work is necessary in this area.

---

Figure 2.2: Images of a dry skull, illustrating the interpolation problem.
Another meaning to interpolation is to improve the resolution of the surface of objects. This has been done by producing an interpolated scene at finer resolution than the original sampling in all directions [HEMMY87, CLINE88], or by using the Partial Volume effect to "undo" the digitisation step (see Chapter 5). Of course these interpolation processes also suffer from the limitations of sampling theory [TIAN86].

2.2.4 Other Scene Operations

If the data is noisy it may need to be smoothed, for example with a median filter [VANNIER83]. In NMR applications some non-uniform correction may need to be made for anisotropy and inhomogeneity of the magnetic fields. This is also a 3D operation.

2.3 SEGMENTATION

As already mentioned, segmentation is required to define hard objects in the sense defined in section 2.1.1 above. Segmentation procedures take as input a digital scene and produce as output an object representation. The type of segmentation operation depends in detail on the object representation that is in use. The object representations considered will be classed as surface-based or volume-based. Surface-based representations define topological properties of an object. Volume-based representations define an object in terms of its characteristic function $\Phi_D$. The method of storing a representation in a computer is the subject of the next chapter.

The input scene to a segmentation operation may be either grey or binary. Three types of segmentation will be considered: thresholding, boundary detection, and region-growing. Each of these is possible in 2D images or 3D volume data. Boundary detection necessarily produces a surface representation of the object. Boundary detection and region-growing require knowledge of the connectivity of the DDS, which is not required by thresholding techniques. This is discussed in section 2.3.2 on 3D boundary detection.
Let us assume that the acquired data is a set of 2D slices, at a high resolution defined by the pixel size, arranged in parallel along the perpendicular dimension at a coarser resolution. It will be necessary to use some kind of interpolation to simulate uniform sampling in all directions. For each segmentation technique three variations are possible:

\[ S_1 : \text{Segment in 2D, interpolate, and create 3D structure} \]  \hspace{1cm} (2.16)
\[ S_2 : \text{Interpolate, segment in 2D, and create 3D structure} \]  \hspace{1cm} (2.17)
\[ S_3 : \text{Interpolate, create 3D structure, and segment in 3D} \]  \hspace{1cm} (2.18)

It seems likely that segmenting in higher dimensions will give a better result. See e.g. [HERMAN78] for an actual example.

2.3.1 Thresholding

If thresholding only is used then \( S_3 \) requires a greyscale volume representation, but the very great advantage is that the thresholding is then dynamic. In some applications dynamic thresholding is all that is required. For example UCL3D allows bone to be visualised from a volume data base derived from CT, simply by choosing a range of CT numbers for display that are relevant to bone. Then operations performed on the bone apply to the whole data set and implied changes to soft tissue are visualised simply by choosing the CT number range appropriately. In a binary or multivalued representation, the thresholds would have to be chosen when the structures were derived from the original data, either by method \( S_1 \) or \( S_2 \). If an unsatisfactory range was chosen then the data would have to be reprocessed.

A point to notice is that in traditional image processing applications, thresholding is achieved by deriving the intensity histogram, and choosing an intensity value that separates object and background, on the assumption that the histogram will be bimodal, or multimodal [GONZALEZ87]. In medical applications it is sometimes possible to choose a threshold based on knowledge of the absolute values represented by the data. For example CT produces data in Hounsfield Units that relates directly to the X-ray attenuation coefficient. Different tissues can be classified by their Hounsfield ranges. For example:
<table>
<thead>
<tr>
<th>Tissue</th>
<th>Hounsfield threshold</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cortical Bone</td>
<td>1500</td>
</tr>
<tr>
<td>Trabecular Bone</td>
<td>150</td>
</tr>
<tr>
<td>Soft Tissue</td>
<td>50</td>
</tr>
<tr>
<td>Water</td>
<td>0</td>
</tr>
<tr>
<td>Fat</td>
<td>-100</td>
</tr>
<tr>
<td>Air</td>
<td>-1000</td>
</tr>
</tbody>
</table>

Actually water and air are assigned the values given above, which is how the Hounsfield Unit is defined. Other modalities may not allow such an easy choice of threshold.

Using thresholding, the object is defined as the union of all voxels that satisfy the threshold criterion. Thus the discrete characteristic function is just the result of the binarisation operation defined in section 2.2.2:

\[ \Phi_d(i,j,k) = B[V(i,j,k)] \] (2.19)

Objects defined in this way have no implicit topological properties. These must be derived by separate operations.

### 2.3.2 Boundary detection

For many types of data, thresholding by itself would not be appropriate. For example, separating organs like liver, heart or kidney is not possible from CT scans by thresholding alone. Often the original data will have been obtained under conditions designed to make segmentation easy, but usually noise will still be a problem. Thus boundary detection is often preferred. Boundary detection may be carried out on either binary scenes, or grey scenes. The former encompasses the techniques employed in the MIPG approach, both in 2D and 3D, and to a lesser extent in 4D (time sequences of 3D data). The latter will sometimes need to incorporate the concepts of edge-detection.

This section summarises methods for deriving the surface description of an object. Some recent reviews have been published, (for example [UDUPA87]).
2.3.2.1 **Boundary Detection in a Binary Scene**

The methods in this section describe the work of the MIPG group. They operate in a scene that has been interpolated to a DDS with cubic voxels, and binarised. The term *Cuberille* is used for such a scene. Techniques have been applied both for 2D and 3D scenes.

2.3.2.1.1 **2D Boundary Detection**

These methods work either on the original slices before interpolation (i.e. method $S_1$ of section 2.3) or on slices sub-regioned from the interpolated scene (method $S_2$). In the former case an interpolation is required after the boundaries have been found [RAYA88]. In the latter case, the sub-regioning may be in any orientation (the CEMAX system makes use of this [FELLINGHAM86]). The output is a *directed contour* representation that may be displayed rapidly but only in a viewing plane orthogonal to the plane of the slices. This restriction on viewing direction is made by many systems, although not often stated, (for example PIXAR), either because the representation is 2D as described here, or to simplify the object-space to image space relationship, which will be discussed further in Chapter 4.

2.3.2.1.2 **3D Boundary Detection**

These methods, also due to MIPG, produce a 3D surface representation. They are topologically based and require precise topological definitions of an object. The discussion in this section follows the notation used by many MIPG publications, (for example [UDUPA87]).

A digital point $(i,j,k)$ in a binary DDS is called a *1-voxel* if it has value 1, and a *0-voxel* if it has value 0. Suppose $Q$ is the set of all 1-voxels in a binary DDS and $\tilde{Q}$ is the set of all 0-voxels (the *background*). In simple thresholding the set $Q$ is the object.

$A$ and $B$ are *n-connected* in a set $S$ if there is a sequence $V_0, V_1, \ldots, V_p$ such that:
Y is an \textit{n-component} of S if:

1) \( Y \subseteq S \),

2) \( A <n\text{-adj}> B \; \forall \; A, B \in Y \)

3) \( Y \) is not a proper subset of another set in \( S \) satisfying i) and ii)

The last condition is to ensure that \( Y \) is a \textit{maximal} subset of \( S \). An \textit{n-component} is then an equivalence class of the partition induced by imposing \( n \)-connectivity in \( S \) [SRIHARI81].

An \textit{n-object} is defined as an \textit{n-component} of \( Q \).

The \textit{boundary} between two sets \( Y \) and \( Z \) is the set of faces:

\[
\partial(Y,Z) = \{(y,z) \mid y \in Y, z \in Z, y <2\text{-adj}> z\}
\]

By contrast the \textit{border} is a layer of voxels containing the boundary. This is the set that might be removed during \textit{skeletonisation} (thinning), (see for example [LOBREGHT80]). An \textit{n-border} is a border wherein all voxels are \( n \)-adjacent to the background.

The connectivity of \( \bar{Q} \) must not be equal to the connectivity of \( Q \) [SRIHARI81, ARTZY81]. The first published method from the MIPG defined the object of interest as a 1-object \( Q_1 \), and the background as a 2-object \( Q_2 \). Then the required surface of interest was the boundary \( \partial(Q_1,\bar{Q}_2) \). Such a surface has the advantage of satisfying certain topological properties deemed "reasonable". For example a line connecting a point inside the surface to one outside the surface, intersects the surface an odd number of times.

The algorithm for finding the surface requires a "seed" face that is defined as on the surface. Then all other faces are tracked, essentially by traversing a binary tree representation of the directed graph defining adjacency of faces [ARTZY81]. The definition of adjacency of faces could be 1-adjacency or 0-adjacency (common edge,
common vertex). The method described by Artzy et al [ARTZY81] uses a common edge, and is termed *P-adjacency* by Herman and Webster [HERMAN83b]. It is defined in such a way that one could "walk" between 1-adjacent voxels by stepping from face to P-adjacent face. In addition P-adjacency of faces is defined and is different to that of voxels because it is not symmetric:

\[ f_1 \lessdot_{P-adj} f_2 \neq f_2 \lessdot_{P-adj} f_1 \]

This is because it is defined with respect to a direction of traversal. Each face only has two possible P-adjacent faces, either on the same voxel or on a 1-neighbour. Thus the graph of possible surfaces is a directed one and the algorithm to find it is equivalent to traversing a binary tree. This algorithm has the advantage of being formally provable by topological means [HERMAN83b].

A more sophisticated approach [UDUPA82b] allows other connectivity objects to be defined and can track in any finite number of dimensions. The algorithms proposed here find first the *k-border* of the object, then the n-components within that border, and then the boundary of these components. For this reason these methods are rather slower than those of [ARTZY81], but are more general. In particular, the technique could track a 4D border, for simulated data.

Recently the MIPG surface tracking algorithms was improved by a further refinement in the definition of connectivity [GORDON87]. The idea was to distinguish between the three different axes around which the faces of a voxel could be traversed. Then instead of tracking in all three directions, one becomes redundant, and in addition, some faces (about one third) do not have to be visited twice. Therefore the traversal is somewhat reduced in effort. The algorithm has not as yet been formally proved.

In all these approaches, depending on the connectivity assumed, the resultant objects are slightly different, but the qualitative results are very similar, and so are quantitative measures such as volume and surface area.

The advantage of boundary tracking methods is that only faces that actually border the object are determined. Some other methods determine the voxels that contain the surface. In this case the internal faces of these voxels are redundant. The disadvantages
of any surface based technique are that i) there is an overhead in surface tracking, ii) the object cannot be cut open without reverting to the original data and retracking. Also note that strictly, the surface only is derived, with the surface elements in a random order. Consequently it is not simple to "fill" the object or to determine if a point is inside. A slightly different approach is used by [UDUPA82a] which derives the surface in a sorted order so that region filling can readily be applied.

The choice of connectivity will be returned to in section 5.2.1.1 on surface shading.

2.3.2.2 Boundary detection in Greyscale scenes

A great amount has been written about edge detectors, but mostly in 2D. A useful review is given by Fu and Mui [FU81]. In UCL3D only 2D contour following algorithms have been implemented, but an advantage has been gained by operating on a full 3D data set, since 2D slices may be reformatted from the 3D data in any direction.

2.3.2.2.1 2D Grey Boundary Detection (Contour Following)

Method S, of section 2.3 is often used to produce a $^\text{a}_\text{B}-\text{R}$ description. First contours are derived in each slice, and then they are connected into a closed polyhedron by means of surface patches or tiles, also often called facets. The contours may be formed either by edge-detection or contour-following. The latter, which will in any case use some implicit edge operator to decide on the direction of search, produces a closed connected line (in topological terms, a manifold in two-dimensions). One of the first boundary detection algorithms was given by Liu [LIU77] and used the Roberts operator [ROSENFELD82] as its implicit edge operator. This, however, only produced pixel centres as boundary elements.

The UCL3D contour following algorithm operates on a greyscale array, following the boundary of a region at a given threshold $\rho$ (i.e. it gives the iso-contour at $\rho$), and uses intensity interpolation to give higher resolution. It uses 1-connectivity (i.e. edge- or 4-connectivity) of the pixel array. Thus four directions (left, right, up and down) are
possible for the contour at each pixel. At each stage the algorithm interpolates the pixel values in one of these directions, to give a point on the contour. Thus the output is a list of (x,y) pairs that do not necessarily lie on pixel centres. This algorithm was adapted for UCL3D by S. Grindrod from a public domain program supplied by Tony Redpath of the Western General Hospital, Edinburgh [REDPATH75]. An outline of the algorithm is given here:

1. Given a seed pixel by the user, use its value as the value of the isocontour threshold \( \rho \).
2. Create a binary map of the region. Initialise it to zero, and label all vertical pairs of pixels that straddle the threshold, by setting the flag in the map at the upper pixel of this pair.

---

Figure 2.3: Principle of the 2D contour following algorithm.
3. Search in the neighbourhood of the seed pixel to find a start pixel with the corresponding bit set in the map.
4. Initialise Current Direction to down.
5. Examine a 2 by 2 neighbourhood of the current pixel, in a clockwise sense, beginning with the current direction.
6. Interpolate the greyvalues to find the contour point between the current pixel and the next in the current direction.
7. Update the current pixel and current direction based on the states (above or below p) of the four pixels in the 2 by 2 neighbourhood.
8. If we have not returned to the start pixel goto 5, else return.

A schematic of the algorithm is given in Figure 2.3. As described, the algorithm gives one point for every pixel edge lying on the contour. These may be "thinned out" to give a smaller number of points by replacing points that are nearly collinear by the start and end of the line to which they are close. If the interpolation is not carried out, then the algorithm will output the same points as the 2D binary boundary tracking algorithm of section 2.3.2.1.1. One point of interest is that the algorithm only interpolates in one dimension. It will be seen that surfaces formed by connecting the contour points, will respond in the same way to shading algorithms, as an interpolating ray-tracing method that will be introduced in Chapter 4. A discussion of this point will be given in Chapter 5.

2.3.2.2 3D Grey Boundary Detection

This is more difficult, since for any point on the boundary, the next boundary element may be in a choice of directions. As in the 2D case the boundary may be found by combining a surface tracking algorithm like that of section 2.3.2.1.2 with an edge detector, or just by edge detection alone. In the latter case there is no guarantee that the surface will be topologically connected, which is often desirable. However these methods are often those employed in dynamic object segmentation where the object is "soft" as in volume rendering.

The method of [LIU77] was extended to 3D (in the same paper) and also to 4D [HERMAN78] in a completely straightforward way. These algorithms are rather similar in principle to the Cuberille surface tracking methods, but do not satisfy such elegant algorithm proving theories. They incorporate some error detection mechanisms by
keeping a running average of the edge-strength so far detected and rejecting parts of the contour that deviate too much from this. In this way they are related to region-growing techniques described below. The boundary elements are still only voxel edges, and hyper-voxel faces respectively. An alternative approach [RHODES79] used region growing in each slice followed by seeding 2-adjacent voxels in successive slices. The algorithm had to allow for "reentrant" surfaces, but claimed to be optimal in terms of the number of slice-fetch operations needed in a limited memory computer.

In principle, the intensity interpolating contour following algorithm given in 2.3.2.2.1 could be extended to 3D, for example by just interpolating across the face of the boundary elements detected by the Liu algorithm, or indeed by the Cuberille algorithm, if thresholding were incorporated dynamically. The difficulty then comes in how to represent what would in effect be a slightly warped 3D voxel boundary. The discussion of representations is the subject of the next chapter, but this specific problem does not seem to have been addressed.

If edge-detection alone is used, then the edge operator is a 3D one. A number of operators are possible. For example the 2D Roberts operator:

\[
\begin{bmatrix}
1 & 0 & 1 \\\-1 & 0 & 1
\end{bmatrix}
\]

extended to 3D and 4D is that used by [LIU77] and Herman and Liu [HERMAN78]. The theory of edge detection (in any dimension) notes that an edge is a place where the derivative of intensity is a maximum or minimum, and therefore that the second derivative is zero. Since differentiation will tend to amplify noise it is desirable also to smooth the image. Thus locally smoothing filters are employed such as the Marr-Hildreth operator [MARR80], which is the Laplacian of a Gaussian, or the Sobel filter. The extension of these to 3D has also been investigated [HÖHNE89b]. Kübler et al [KÜBLER87] implemented a small variance Difference of Gaussian (DOG) filter in 3D, which is a close approximation to the Marr-Hildreth operator. The Zucker-Hummel [ZUCKER82] operator is the extension of the Sobel operator, and can be implemented in a small 3D filter (often 3 by 3 by 3).
Edge operators are also derived by the attempt to fit a simple approximation to the image function. For example fitting a plane or second order polynomial in 2D results in the Prewitt operator as a 2 by 2 filter:

\[
\begin{bmatrix}
1 & 1 & 1 \\
1 & 1 & 1
\end{bmatrix}
\]

or 3 by 3 filter:

\[
\begin{bmatrix}
1 & 0 & 1 \\
1 & 0 & 1 \\
1 & 0 & 1
\end{bmatrix}
\]

respectively [DIZENZ086]. The generalisation to n dimensions was made by Morgenthaler and Rosenfeld [MORGENTHALER81]. These authors evaluated the improvement of a 3D approach over a 2D approach for a set of CT data, and also noted how to use an asymmetric neighbourhood to overcome the undersampled coaxial direction problem. The tangent plane in 3D so determined is just that that is required by shading algorithms (see section 5.2).

Fitting of a generalised stepfunction in 3D was how Zucker and Hummel developed their operator, intended as an extension of a 2D method due to Hueckel [HUEKEL71, HUEKEL73]. Since these edge operators can be implemented as small local filters they are often used in dynamic segmentation methods which are described in Chapter 4.

2.3.3 Region Growing

Region growing attempts to segment an image by defining a "seed" pixel or voxel and growing into regions that are connected and satisfy a condition that typically limits the deviation of any pixel or voxel from the average of the region already defined. Other techniques in the same category are region splitting and split and merge. A useful early review is given by Zucker [ZUCKER76].

Region growing is less widely used than the previous two techniques (thresholding and boundary detection). Region growing in a binary scene was used by [UDUPA82], and indicated by Heffernan and Robb [HEFFERNAN85b]. Some 3D connectivity procedures
were incorporated in the Insight graphics workstation [MEAGHER84d]. Crawford et al [CRAWFORD89] described an anisotropic, multi pass algorithm that used 2-connectivity in a 3D scene. Often manual intervention was required to correct mistakes. Use of region growing within the CEMAX system was described by Zonneweld et al [ZONNEWELD89], although no details were provided.

In common with boundary detection, it seems likely that use of 3D algorithms will improve over the application of 2D algorithms in separate slices. For example some attendant problems in 2D, such as partial volume effects, may be easier to control in 3D. This would appear to be a fruitful area for future research.

### 2.3.4 Other Segmentation Techniques

The above three techniques have been described not only because they are the most common, but also because their relatively efficient implementation allows them to be executed "on the fly" as a means of dynamically segmenting a grey DDS. This means that to some extent they can be applied in the volume rendering methods described in section 4.4.2.3. Some other techniques, that are lengthier, have been used for a segmentation stage that produces an object as a preprocessing step, and then concentrates on visualisation only. These are briefly described in this section.

**Feature Space Clustering** is a technique used in several image processing applications. It attempts to segment pixels (voxels) by their grouping in a multi-dimensional *Feature Space*, the coordinate axes of which, (e.g. density, edge strength, texture) are problem orientated. Thresholding can sometimes be thought of as one dimensional feature clustering, if the image (in 2D or 3D) is supposed to consist of just an object and a background, and the threshold is to be determined by for example finding the midpoint of a bimodal histogram. The presence of noise may lead to a corruption of the histogram, making such a decision impossible [PANDA78]. Since medical data has a more exact interpretation (i.e. the densities represent physical parameters), there is an absolute value for the threshold value, so the histogram is not required, and noise may be eliminated by other means (e.g. connectivity, or minimum object size). Yet, there
are still situations where the partial volume effect will lead to erroneous classification even if no noise is present. Trivedi et al \cite{TRIVEDI86c} applied the 2D method of Panda and Rosenfeld \cite{PANDA78} to 3D data from both NMR and CT data, where the requirement was to segment into three classes, and the intermediate class could not be distinguished by thresholding or surface tracking, from the boundary of the other two classes. Here the feature space properties were density and gradient, the gradient being simply the vector magnitude of the components along the principle $x,y,z$ axes of the first differential. It is interesting to note that such a two dimensional (density, gradient) classification crops up again in Levoy \cite{LEVOY88} on volume rendering. A useful review of clustering is included in the survey by Fu and Mui \cite{FU81}.

The description of edge detection in section 2.3.2.2 was mostly in terms of an edge operator that could be implemented as quite small convolution filters. However the original theory of Marr and Hildreth \cite{MARR80}, which was really concerned with human perception of edges, discussed the notion of a whole range of scales of objects, and characterised them by the output of their operator with different widths of Gaussian (typically up to about 30 pixels standard deviation). Such a filter becomes a very lengthy convolution. However, since the human observer is considered better at segmenting images than anything else, techniques that try to emulate such multiscale decisions have been attempted. These fall into the class of \emph{multiresolution segmentation}.

The stacking of 2D, 3D or 4D images, convolved by a Gaussian blurring function into lower resolution, results in an extra dimensional space called \emph{scale space}. Methods following the lines of maxima and minima as they blur into each other have been proposed as a segmentation method \cite{PIZER86}. The computational burden is very large, although certain iterative techniques can be used, exploiting the increasing band limitation with reduced resolution \cite{FOX88}.
2.4 CONCLUSIONS

This chapter has discussed the basic concepts of, and notation for, digital scenes, in two and three dimensions. The processing of such scenes has been discussed, and in particular the techniques to segment an object from a scene. It is advantageous to perform segmentation in as a high a dimensional space as possible, and to exploit the density and density gradients of the pixels (voxels). The ability to perform such segmentation depends on efficient random access to greyscale data in two and three dimensions. Thus the representation of the scene, and of the objects in the scene, is a critical factor. The next chapter examines various representation schemes that have been employed, and introduces those used by UCL3D.

It is also noted that recent visualisation techniques attempt to dynamically segment the object, or even not to segment, but to attempt to visualise the whole data. This is returned to in Chapter 4.
CHAPTER THREE

REPRESENTATION OF 3D MEDICAL OBJECTS

Chapter 1 introduced four types of representation for 3D data (Boundary, CSG, Parametric, and Spatial Enumerative), and made a more general classification into surface and volume representations. A second demarcation is between what may be termed random access or sequential access data. Thinking only of the display problem it is usually sufficient to provide an ordered list of those primitives (voxels or facets) that are to be displayed, with a mechanism for running through the lists in an order appropriate for the viewing direction [FREIDER85]. However other applications might want to interrogate the data in a random way - for example to point to an object and read the data value there, or to give quantitative distance values, or in surgical simulation to detect interference as parts of structures are moved with respect to each other. In such circumstances it is highly advantageous to have random access into the data. A three-dimensional array of voxels or an octree structure allows random access, whereas run length encoded structures do not.

A last classification that will be made here is to what type of data is stored. A B-R structure stores sufficient face edge and vertex information to fully describe a surface. In addition these primitives might have attributes such as colour and transparency, and possibly information to say whether they are part of a smooth surface, or a real abrupt edge. Thus it is possible, though quite complicated, to have several surfaces in one data base. For voxel type data on the other hand it is simple to store at each voxel information relevant to the original data at that point. Possible classifications would be Binary (i.e. just the presence or absence of an object), Multi-Valued which would store one of a number of objects, or Greyscale which would in effect maintain all the original data. Other possibilities would be to have even more attributes, such as local gradients, flags to indicate the presence of neighbours, or colour and transparency. Some discussion of different voxel structures, implemented on special purpose hardware is given by Meagher [MEAGHER84d].
This chapter summarises some data structures used for representation of 3D objects in medical imaging. To fix the ideas a specific example of an object will be given. The object described is shown in figure 3.1. It is a binary object in a 4 by 4 by 4 array.

3.1 BOUNDARY REPRESENTATIONS

3.1.1 B-R Approach

3.1.1.1 2D B-R (Contours)

Only pairs of points \( \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\} \), and a depth for each slice need to be stored. Each set of points in at one depth is a contour. In general several contours per slice are possible. If the ordering (clockwise or anticlockwise) of stored points is significant, the representation is called a directed contour. This allows internal and external surfaces to be differentiated. In the UCL3D package the non-directed contour representation is used to store the Regions of Interest (ROIs) for each slice. These are in a file structure associated with the original image files.
The 3D B-R is produced from the 2D B-R by a tiling algorithm. The output is a B-R description of the surface. Here it is the polygonal faces that are acting as the interpolating function, a rôle that has rather limited validity in terms of information theory. This is because the value of the characteristic function $\Phi_c$ at an intermediate point will depend on whether it is inside or outside the closed polyhedron, rather than on the values of the original density function in the neighbourhood.

The form of the tiling operation is far from trivial and the representations produced are not unique. Udupa [UDUPA87] refers to this as the "tiling problem". The problem may be seen as that of finding the optimal directed graph through the points produced by the contouring algorithm, with respect to a cost function. Several schemes are possible for the cost function. Keppel [KEPPEL75] chose to maximise the volume of the derived polyhedron. Fuchs et al [FUCHS77] minimised the surface area, and Cook et al [COOK83] minimised the angular step between successive facets. A simpler algorithm is one that chooses the edge that gives the shorter distance between points [CHRISTIANSON78]. This algorithm was adopted in UCL3D. Figure 3.2 shows the stages that produce a faceted description of the skin surface from the set of CT slices shown in figure 1.2. It is sufficiently fast to be run directly, (i.e. the 3D B-R is not stored in a file structure). The output of the algorithm is a vertex list of $(x,y,z)$ triples, and a list of triangular facets $(n_1, n_2, n_3)$ that point into the vertex list.

Such tiling algorithms are all concerned with the problem of connecting one contour in each slice. The problem of connecting branching and merging structures introduces a second degree of complication. This has been addressed to a much lesser degree. Christianson and Sedeberg [CHRISTIANSON78] suggested a scheme for fairly well behaved branching, but commented that manual intervention was usually required. Parker et al [PARKER86] attempted a heuristic method for connecting the complex branching structure of vascular beds. Some more methods are reviewed by Udupa [UDUPA87].
A B-R description might also be generated in an interpolated scene. This will produce a larger data base, but will more closely approximate the surface of the characteristic function. However this method does not seem to have been tried. A recent approach [CLINE88] suggests directly constructing tiles within voxels based on an estimate of whether the surface of the characteristic function intersects that voxel. This algorithm (Marching Cubes) first interpolates the DDS to a much finer lattice and then produces the B-R. The vertices located are always on the faces of the voxels in the scene, which ensures consistency. The disadvantage is large numbers of tiles which however can be input to a hardware renderer (e.g. PIXEL-PLANES).

To see the problems that would be encountered in applying a B-R description to the skeletal structure of the face, consider the output of the contour following algorithm of section 2.2 when applied to the bone threshold of one of the slices of figure 1.2. The resultant contours are shown in figure 3.3. Following the connections of such complex contours through successive slices becomes a very formidable problem.
3.1.2 Voxel Approach

3.1.2.1 2D Voxel Approach

In this representation, either pixel locations are stored in order around the contour, or the outside and inside edges of the raster conversion of the enclosed region [UDUPA82a]. An alternative proposed by Heffernan and Robb [HEFFERNAN85a], is a *chain code*, and a starting pixel. They also smoothed the contour by a Fourier expansion, so that variable resolution contours could be derived.
3.1.2.2 **3D Voxel boundary.**

This representation is stored as six lists, one for each orientation of the faces of a cubic voxel. Each member of the list stores the (x,y,z) coordinates of the centre of the face, or, equivalently, the (i,j,k) triple of the voxel to which it belongs.

Using the latter description, the object in figure 3.1 would be represented as:

- \( x^+ \) \((1,2,0), (1,3,0), (3,1,1), (2,2,1), (1,3,1), (0,3,2), (1,2,2), (1,2,3)\)
- \( x^- \) \((0,2,0), (0,3,0), (0,2,1), (0,3,1), (3,0,1), (0,3,2), (1,2,2), (1,2,3)\)
- \( y^+ \) \((0,3,0), (1,3,0), (0,3,1), (1,3,1), (2,2,1), (3,1,1), (0,3,2), (1,2,2), (1,2,3)\)
- \( y^- \) \((0,2,0), (1,2,0), (0,2,1), (1,2,1), (2,2,1), (3,1,1), (0,3,2), (1,2,2), (1,2,3)\)
- \( z^+ \) \((0,2,1), (1,3,1), (2,2,1), (3,1,1), (0,3,2), (1,2,3)\)
- \( z^- \) \((0,2,0), (0,3,0), (1,2,0), (1,3,0), (2,2,1), (3,1,1)\)

Notice that the object is a single 1-object, but is three separate 2-objects.

### 3.2 VOLUME REPRESENTATIONS

In this section, representation of a 3D digital array will be considered. A thorough review has been given by Srihari [SRIHARI81]. The representations considered are Digital arrays, Marginal Indexing, Run Length Encoded, Segment-Indexed Encoded (Segment End-Point), and Slice-Based. The discussion of octrees, which is the representation used in UCL3D, will be given in more detail in section 3.3.

#### 3.2.1 Digital arrays

Consider the basic structure of the form:

```plaintext
voxel_type voxel_array[XDIM][YDIM][ZDIM];
```

If `voxel_type` is density this will be called the 3D greyscale array representation. Then `voxel_array[i][j][k]` is an *estimate* (in the sense of section 2.1.3) of the value of the function at points \((x,y,z)\), where \((i,j,k)\) is the integer triplet of the voxel containing the point \((x,y,z)\). Other more compact representations may be derived from this.
If the object has been segmented from the data, then it may be represented as a union of voxels of value 1 (1-voxels), and the background as a union of voxels of value 0 (0-voxels). Then voxel_type is binary and the representation is called a 3D binary array. In practice the representation is often implemented by packing eight voxels to a byte [FRIEDER85].

If several objects have been segmented from the data, then the voxel_type may be made multivalued (for example enumerative in C or Pascal). Then some compression of the data is still achieved. This representation will be referred to as the Multivalued digital array.

The binary digital array representation of figure 3.1 is then:

```
\begin{array}{cccc}
\text{Slice 0} : & 0 & 0 & 0 \\
\text{Slice 1} : & 0 & 0 & 0 \\
\text{Slice 2} : & 0 & 0 & 0 \\
\text{Slice 3} : & 0 & 0 & 0 \\
\end{array}
```

3.2.2 Marginal Indexing

In implementing a 3D digital array, it is often convenient to use a pointer based representation rather than arrays of arrays. This is because, at the expense of a small increase in the storage requirement, a greater efficiency of access is achieved. The representation is dynamic and may be constructed by dynamic allocation:
1) Allocate an array of size ZDIM, of slice-pointers
2) For each slice
   2.1) If the slice is empty
       Assign NULL to the slice-pointer
   2.2) Else
       allocate an array of size YDIM, of row-pointers
3) For each row
   3.1) If the row is empty
       Assign NULL to the row-pointer
   3.2) Else
       allocate an array of size YDIM, of voxels

The structure is in effect a 3-level tree. In the language C, this allocation is very straightforward, and the array can still be randomly accessed as `voxel_array[i][j][k]`. The storage requirement has increased slightly (by the size of the pointers). However, now if any rows or planes of the data are empty, they can be represented as Null pointers. This representation is referred to by Srihari as *Marginal Indexing*. There is a choice as to the ordering of the dimensions (XYZ, XZY, YXZ, YZX etc.), which theoretically should be chosen to optimise the compression. Srihari also suggests that only one copy of a row be stored, if they are likely to be duplicated. This is unlikely in Medical data. Using the order ZYX, Figure 3.1 is represented in the form shown in figure 3.4.

---

Figure 3.4 : Marginal index representation of the object in figure 3.1
This representation has a close relationship to the Segment-Indexed type used by Reynolds, described in section 3.2.4.

### 3.2.3 Run Length Encoded (Start End Point)

Run length encoding of a binary digital scene is a list that stores the length of a run of 1-voxels and 0-voxels alternately. For example in a 4 by 4 by 4 array the object shown in figure 3.1 would be represented by:

\[(0 : 8), (1 : 2), (0 : 2), (1 : 2), (0 : 9), (1 : 4), (0 : 1), (1 : 2), (0 : 11), (0 : 1), (0 : 2), (1 : 1), (0 : 12), (1 : 1), (0 : 6)\]

Alternatively only the start and end of the 1-runs is stored.

\[(8 : 9), (12 : 13), (23 : 26), (28 : 29), (41 : 41), (44 : 44), (57 : 57)\]

The coding can be applied to grey scenes as well, in which case the value of the voxel must also be stored.

The run-length encoding ignores the organisation of slices and rows. The dimensions of the digital array must be known and the indices i,j,k run through in a consistent order. In the examples just given the indices were run through with i varying fastest, j second, and k (for slices) the slowest.

Use of run-length encoding is very efficient and in Chapter 4 display techniques for them will be mentioned. However, the coding is not random-access. If the value of voxel (i,j,k) is required, the list must be sequentially accessed. This is a considerable disadvantage in many quantitative operations.
3.2.4 Segment-Indexed Encoding

A method of using run-length encoding with a degree of random-access has used by MIPG [TUY84, TRIVEDI85, REYNOLDS87]. Here a row of a slice is encoded as a run length, but a marginal-index description is used to access the rows. The run-lengths are stored as start-end points referred to as line-segments, with a pointer to the next segment. This allows the storage to be dynamic. Reynolds et al [REYNOLDS87] used this coding both for the 3D objects and 2D images. The technique leads to efficient storage and very fast display algorithms (section 4.4.2.2). The representation of the object in figure 3.1 is:

Slice

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Null</td>
<td>Null</td>
<td>0:1</td>
<td>0:1</td>
</tr>
<tr>
<td>1</td>
<td>Null</td>
<td>3:3</td>
<td>0:2</td>
<td>0:1</td>
</tr>
<tr>
<td>2</td>
<td>Null</td>
<td>Null</td>
<td>1:1</td>
<td>0:0</td>
</tr>
<tr>
<td>3</td>
<td>Null</td>
<td>Null</td>
<td>1:1</td>
<td>Null</td>
</tr>
</tbody>
</table>

The method of Tuy and Tuy [TUY84] and TRIVEDI85] is very similar but uses a sequential list of run-length encoded rows instead of a linked list.

3.2.5 Slice-Based

It is possible to maintain only the slices, and to carry out interpolation "on the fly". This might be appropriate in a memory limited system. A method of applying ray tracing in such a representation was described by Rhodes et al [RHODES87] and discussed by [TUY84].

3.3 OCTREES AND QUADTREES

The octree structure for 3D arrays, and the quadtree structure for images, have been used extensively in UCL3D. In this section they are briefly introduced. In section 3.4 some notation and properties of octrees and quadtrees is described. The literature on such hierarchical representations is very large, and a full survey is outside the scope
of this thesis. The discussion will be limited to those properties that are of direct use in the application being considered. A recent two part review by Samet and Webber [SAMET88a, SAMET88b] is very comprehensive. These authors make the distinction between raster octrees for octrees derived from a digital array, and vector octrees for those derived from one of the other representations mentioned in chapter one (B-R, Parametric, or CSG). Since this discussion is concerned with the representation of digital scenes, the raster octree is the only one considered here.

A \textit{k-tree}, where \( k = 2^n \), is a representation of \( n \)-dimensional space. Thus a \textit{quadtree} is a symmetric, recursive, hierarchical description of an image, and was derived originally from the Warnock algorithm for hidden line elimination [EASTMAN70, KLINGER76]. The octree is its 3D extension, and was first described by Jackins and Tanomoto [JACKINS80]. Then the octree is a hierarchical representation with nodes that have a \textit{status} which is either homogeneous (also called a \textit{leaf} node) or inhomogeneous, (also called a Partial or \textit{internal} node). Inhomogeneous nodes are spatially divided by a binary cut along each axis of the Universe. Thus an octree is an eight-array tree. An octree description assumes that the Universe is represented in a 3D DDS of dimensions \( N \times N \times N \) with \( N = 2^n \) where \( n \) is an integer. The octree representation of this 3D DDS is a tree of depth \( n \), where level 0 is called the \textit{root} and level \( n \) the \textit{top} of the tree. A homogeneous node at level \( m \) represents a cube of voxels of edgelength \( 2^{(n-m)} \). In order to remain as far as possible independent of a particular implementation a node will be represented as an undefined structure \( \Psi \).

Srihari [SRIHARI81, SRIHARI84] has made various suggestions for modified tree structures including asymmetric octrees and bi-trees where each inhomogeneous node is divided only along one axis, either symmetrically or asymmetrically, chosen to maximise the compaction of the tree. The complexity of the algorithms for optimizing the decision about the position and orientation of the bisection planes is large, and these methods have had limited development by other workers [SAMET88a]. These will not be discussed further here, so that the term octree will refer to a \textit{symmetric recursive} description (Srihari’s term). Figure 3.5 shows this representation applied to the object in figure 3.1.
The information stored at a node characterises the type of tree. The colour of a node is typically a function of the values of its constituent voxels. Information may be stored at both homogeneous and inhomogeneous nodes. Octree structures that store information at inhomogeneous nodes are deemed pyramid trees. The information at a node may be binary, multivalue or greyscale.

### 3.3.1 Binary and Masked Binary Octrees

If the octree represents a binary digital scene then the colour is zero (Empty) or one (Full). The status (homogeneous or inhomogeneous) is also represented by one bit. Often a two-bit code is used to give the status and colour combined. In the literature Empty is also called White or Void, Full is also called Black, and Partial is also called Grey. Here the notation P,F,E will be use for Partial, Full, Empty respectively.

In the previous chapter the connectivity of a 3D scene was defined. In a binary scene the n-neighbourhood of a voxel is the status of those voxels that are n-adjacent to it. Similarly, the n-neighbourhood of an octree node is the status of nodes that are n-adjacent and at the same level. However the definition is slightly more complicated because there is no guarantee that the neighbouring nodes are at the same level. One definition would be that the neighbour status in direction $a$ of a node is the status of the adjacent node in direction $a$ at the same level, if it exists, or the status of the
smallest homogeneous node at a lower level (i.e. closer to the root), if no node is at the same level. Nodes on the border of the Universe are formally considered to have Empty neighbours. A more elaborate scheme for such multiresolution borders, is given by Chien and Aggarwal [CHIEN86].

Knowledge of the neighbourhood of an octree node is advantageous for several reasons that will become apparent in Chapter 4 on display techniques. Several different schemes were investigated by the author. If two bits per neighbour are used then the 2-neighbourhood requires 12 bits, the 1-neighbourhood requires 36 bits, and the 0-neighbourhood requires 52 bits. A simpler scheme that uses only one bit per neighbour was found useful, for limited applications. This coding is constructed in the following way:

For each node \( \Psi \), at level \( m \):
1) if \( \Psi \) is a voxel (i.e. \( m \) is the top level of the tree)
   Encode the \( n \)-neighbourhood of the voxel as one bit flags
2) else encode only the 2-neighbours as one bit flags as follows:
   2.1) For each 2-neighbour \( \Gamma \):
       2.1.1) if \( \Gamma \) is Full, set the flag
       2.1.2) else if \( \Gamma \) is Empty, unset the flag
       2.1.3) else recursively examine the four children of \( \Gamma \) that are closest to \( \Psi \)
            if they are all Full, set the flag
            if any are Empty, unset the flag

By recursively examine at step 2.1.3, the implication is that if one of the children is Partial, the same step is carried out on its children. The resultant effect is that \( \Psi \) will have its 2-neighbour flag set if the face in question is covered by a layer of voxels, and unset otherwise. Figure 3.6 shows an example. The neighbours in the \( z \) and \( y^* \) directions are Partial, but the neighbour flags are set Full. The 1-connected neighbour code for this node is:

\[
\begin{array}{ccccccc}
  x & x' & y & y' & \bar{z} & \bar{z}' \\
  E & E & E & F & F & F
\end{array}
\] (3.2)

This fairly complicated coding is useful for the Hidden Surface Elimination (HSE) problem discussed in Chapter 4, and also for object based shading, discussed in section 5.2.1.1. It is correctly preserved under Boolean intersection, and (at least for the
purposes mentioned) under Boolean Union. Unfortunately it is not sufficient for Boolean
Complement or Difference operations; (consider the code for the Complement of those
nodes in figure 3.6 - it is \[ E | E | E | F | F | E | E \], which is not the Complement of
(3.2)). Using two bits per flag, it is possible to construct a code that works under all
Boolean operations, however this turned out to be too elaborate to be useful, and was
not pursued further by the author.
The voxel neighbours, encoded in step 1), can use 0-, 1-, or 2-neighbours. In common with MIPG methods [CHEN84], the 1-neighbourhood, requiring 18 bits, was adopted. The neighbour flags, taken together, constitute an *n-mask*, and the type of structure that results will be referred to as the *Masked Binary* representation.

### 3.3.2 Greyscale and Min-max Greyscale

If the colour field of an octree represents the values of a greyscale digital scene then the octree is a *greyscale octree*, (or *grey octree*). It is immediately likely that the space saving potential of an octree may be lost in a grey octree because very many nodes will be the size of voxels. However there may still be large Empty nodes. The next section (3.3.3) will show the complexity of this representation for some example data.

If the octree is a *pyramid* type then the colour of a partial node is a function of the colour of the subtrees. The colour of an inhomogeneous node may be derived either from:

a) The colour of its eight children  
b) The colour of all voxels in that node.

For binary octrees Srihari suggested using a *mode-rule* so that the Partial node is assigned colour one if more than $8\alpha$ children are Full, where $(0 < \alpha < 1)$. He investigated the accuracy of the low-resolution trees for different $\alpha$. Another possibility is to assign colour one if greater than 50% of all voxels comprising that node are Full. Srihari’s criterion were largely to do with the low-resolution properties of octrees for transmission of data.

For greyscale trees the colour of a Partial node could be the mean of the colours of the eight children, or of all voxels in the node. In UCL3D a different structure has been investigated, where a Partial node stores the minimum and maximum of all children. Because this is derived in *post-order* traversal, it is also the minimum and maximum of all voxels. This leads to the Partial node having a different structure to a leafnode, but this is the case anyway for pointer-based structures. For treecode structures, the extra field will increase the size of the tree, but since only $1/7$ of the
Figure 3.7: The min-max greyscale structure incorporates the minimum and maximum of children in its fields.

tree consists of Partial nodes the increase is only ~15%. This structure has proved very useful in display, where a node that has no voxels within a given density range can be treated as Empty, for the purposes of the algorithm. Figure 3.7 is a schematic of this representation. This is termed a Min-Max greyscale octree.

3.3.3 Example Octree data

It is worthwhile examining the complexity of some typical data represented as octrees. The choice of data is difficult, since the complexity of an octree depends not only on its shape, but also its location. (Consider the change in structure of an octree that has one Full child and seven Empty children from the root, to the same object moved by one voxel in x, y, and z). In the illustrations and discussion of algorithms in this thesis both artificial and real medical objects will be used.

The digital sphere shown in figure 3.8 at increasing resolution, was constructed in the binary masked and unmasked structures. At the level 8 representation (i.e. a 256 x 256 x 256 array) the sphere had a radius of 64. The coarser resolutions were obtained by successive reduction of the level 8 tree. A grey version of the sphere was also constructed. This was done by estimating the actual volume of sphere that intersected the voxels at the top of the tree, and assigning a density based on this partial volume.
Figure 3.8: The digital sphere at increasing levels of resolution (tree depths five to eight).

Figure 3.9: The chequered sphere at resolution eight.
The complexity of the binary and grey spheres is therefore the same, but the latter has different colours for the surface voxels.

Because a sphere is a simple connected object it is not typical of most octree applications. Secondly consider the more complex object shown in figure 3.9. This object, which will be referred to as the "Chequered Sphere", is the Boolean intersection of the digital sphere and a 3D "chequerboard" consisting of a repetitive pattern of leaf nodes at level 3 of an octree.

The skull shown in figure 3.10 was constructed in all four data representations from the 256x256x256 digital array of CT data shown in figure 1.2 of Chapter 1. As in the sphere case, this tree was also reduced to give successively coarser representations. In the current implementation of UCL3D there are two bits for status and six for colour so that 64 colours are possible in a grey tree. Thus a compression of 64 : 1 was used to encode the 4096 density values from the original data. The binary tree represents the density values obtained by thresholding the top 44 of these 64 colours.

Figure 3.10 : The binary skull at increasing levels of resolution (tree depths five to eight).
Table 3.1 gives the number of Full, Partial and Empty nodes at each level for each object at its highest resolution (level 8), for the binary representations. Table 3.2 shows the same statistics for the grey octree representation of the skull. The interesting point to notice is that there are indeed homogeneous nodes above the top level of the grey tree. At level 2, about a third of the encoded space is represented by Empty nodes, with an increasing proportion at higher levels. Full nodes begin at level 5 (i.e. 8 by 8 by 8 voxel cubes). In fact the relative number of nodes in the grey tree is very similar to that in the binary tree. This suggests that the use of grey level octrees may not be as expensive as might at first be thought.
Table 3.2: The number of nodes in the greyscale representation of the skull at resolution level 8.

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Partial</td>
<td>1</td>
<td>8</td>
<td>44</td>
<td>236</td>
<td>1,366</td>
<td>8,739</td>
<td>58,942</td>
<td>349,348</td>
<td>-</td>
</tr>
<tr>
<td>Full</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>175</td>
<td>2,883</td>
<td>86,143</td>
<td>2,627,833</td>
</tr>
<tr>
<td>Empty</td>
<td>0</td>
<td>0</td>
<td>20</td>
<td>116</td>
<td>522</td>
<td>2,014</td>
<td>8,087</td>
<td>36,045</td>
<td>166,951</td>
</tr>
</tbody>
</table>

3.4 OCTREE IMPLEMENTATION

This section briefly describes the notation used for octree and quadtree operations. Some further notation will be developed in section 4.5.1 where figure 4.6 illustrates the main ideas. To simplify the description of algorithms a pseudo-code will be used. The definition of a term will be given in italics. A function that returns such a term will have the same name in bold. A node will be taken either as a record structure or as a pointer to a record structure. Associated parameters of a node, (for example, position, colour, value, octal code, size etc) will be given either as a member of a record (in normal type), or as the value of a function, without dictating which are actually in a node record. Further, functions will be allowed to return records and arrays, and records and arrays may be compared for equality, assigned to etc. Parameters to functions are always call-by-value. A similar notational convention was adopted by Jackins and Tanimoto [Jackins80]. For example colour(Ψ) and status(Ψ) return the colour and status of a node respectively.

3.4.1 Octree Notation and Definitions

The octal code or path of a node Ψ is string of octal digits representing the path from the root to that node. Let us define \( J = \text{octal\_code}(\Psi) \) then let \( J_{m,p} \) be the fragment of path \( J \) starting at level \( m \) and terminating at level \( p \), \( 1 \leq m \leq p \leq n \). If the fragment is of length 1, it will be called an element of the octal code. The notation \( \text{All}B \) will represent concatenation of the path fragments \( \text{A} \) and \( \text{B} \).
Let \( c_m = \text{element}(J,m) \) be the \( m \textsuperscript{th} \) element of code \( J \), \( 1 \leq m \leq n \) and \( m_\epsilon \in \{0..7\} \). Then \( J \) is the concatenation of its individual elements:

\[
J = J_{p} = \prod_{m=1}^{p} c_m = c_1||c_2||..||c_p \quad (3.3)
\]

If an inhomogeneous node \( \Psi \) at level \( m \) has octal code \( J(m) \), then \( J^{(m+1)} = J^{(m)}||k \) represents the path to the \( k \textsuperscript{th} \) child of \( \Psi : \Psi' = \Psi_.child[k] = \text{child}(\Psi,k) \), \( k \in \{0,7\} \). Each child so defined is a \textit{co-child} of the other seven.

Let \( a = \text{centre}(\Psi) \) be the cartesian coordinate of the centre of an octree node. Thus a node at level \( m \) describes the region of space containing points \( r \) such that:

\[
\begin{align*}
(a_1 - 2^{(n-m-1)})\delta_1 &< r_1 < (a_1 + 2^{(n-m-1)})\delta_1 \\
(a_2 - 2^{(n-m-1)})\delta_2 &< r_2 < (a_2 + 2^{(n-m-1)})\delta_2 \\
(a_3 - 2^{(n-m-1)})\delta_3 &< r_3 < (a_3 + 2^{(n-m-1)})\delta_3
\end{align*}
\]

(3.4)

with \((\delta_1, \delta_2, \delta_3)\) the voxel dimensions as defined in section 2.1.1. This region of space is the \textit{octant} represented by node \( \Psi \). It is a cube of \( E_m \) by \( E_m \) by \( E_m \) voxels, where \( E_m = 2^{(n-m)} \) is the \textit{edge length} of \( \Psi \).

Because of the finite resolution of the Universe, the vectors \( a \) are discrete vectors in the sense defined in section 2.1.1. At the top level of the tree, the vector will be the same as the voxel vectors \( v(i,j,k) \) defined by equation (2.4).

Also required are the coordinates of one corner of the octree node, termed the \textit{origin}. Let us define \( \text{origin}(\Psi) = \text{centre}(\Psi) - (2^{(n-m-1)},2^{(n-m-1)},2^{(n-m-1)}) \) as the Cartesian coordinate of the vertex defined as the left, bottom, front vertex of the octree node at level \( m \). This vertex has the smallest values of \( r \) that are consistent with a point inside the node.
3.4.2 Quadtree Notation and Definitions

Let us make a similar set of definitions for a quadtree:

Let the quadtree node be Ω. Let \( K = \text{quad_code_of}(Ω) \) be the quad code or path to a quadtree node, and \( K_m \) be the termination of this path at level m from the root. Let \( c_m = \text{element}(K,m) \) be the \( m^{\text{th}} \) element of \( K \); \( 1 \leq m \leq n \), \( c_m \in \{0..3\} \)

Let \( q = \text{centre}(Ω) \) be the two-dimensional coordinate of the centre of a quadtree node. Then a quadtree node at level m represents the region of the screen containing points \( s \) such that:

\[
(q_1 - 2^{(e-m-1)}\eta_i, q_1 - 2^{(e-m-1)}\eta_i) < s_1 < (q_1 + 2^{(e-m-1)}\eta_i, q_1 + 2^{(e-m-1)}\eta_i) \\
(q_2 - 2^{(e-m-1)}\eta_i, q_2 - 2^{(e-m-1)}\eta_i) < s_2 < (q_2 + 2^{(e-m-1)}\eta_i, q_2 + 2^{(e-m-1)}\eta_i)
\]

with \( (\eta_i, \eta_j) \) the pixel dimensions as defined in section 2.1.4. This is the quadrant represented by node Ω.

3.4.3 Encoding of octrees and quadtrees

There are a great many variations for the method of representing an octree or quadtree in a computer, that will typically depend on the amount of memory and speed requirements for any particular system. For example the most "unpacked" type of tree would explicitly record the status, node colour, local neighbourhood and coordinate position at each node and have full pointers to the memory location of each child. Or Empty nodes may be represented by null pointers. Usually the cartesian coordinates of a node are computed during execution of the algorithm, by keeping track of position in the tree traversal. Many operations e.g. Boolean Union, Intersection or Complement, or 90° rotation or translation by powers of two of the basic resolution, do not require spatial information, nor "random" access into the tree. In this case a packed tree form may be stored without the pointers, referred to by Oliver [OLIVER83a, OLIVER83b, OLIVER85] as treecode. Gargantini [GARGANTINI82a, 82b] has described a very compact structure, the linear octree (quadtree), where only the octal codes of the Full nodes are stored. Oliver has refered to the linear octree structure as leafcode.
Conversely an *explicit octree* is possible, wherein space for the largest possible octree is reserved. This is just an extension of the *explicit quadtree* structure described by Woodwark [WOODWARK82].

When considering the question of display however, it becomes necessary to access the sub-octants of a node in a priority order dependent on the viewing direction [DOCTOR81, MEAGHER82, GARGANTINI86a]. In the case of a "packed" octree structure therefore, pointers must be evaluated during tree traversal. Gargantini *et al* [GARGANTINI86a, 86b] shows the technique for doing this in a linear octree where the beginning and end of an octree list or sublist are passed to each call of the algorithm, and pointers determined by sequential search. Thus in a linear octree an Empty node is equivalent to a sublist of no elements, a Full node to a sublist of one element, and a Partial node to a list of two or more elements. Similarly, treecode is expanded as required, to include pointers, into a sixteen-ary tree (eight nodes, and eight pointers). Some other operations require random access, for example readout of a voxel value for some point in space (section 3.4.5), or reformatting a slice through the Universe [YAU84], which is most easily achieved in a pointer based structure.

The four types (explicit, pointer based, treecode, and linear), are approximately in this order for decreasing space requirement. Generally the last two must be expanded into the pointer based representation for many algorithms.

The *explicit* representation is useful in languages like FORTRAN with no record-structures or pointers. Here space is reserved for the worst case octree. This has the advantage of not requiring dynamic allocation of memory, and allows direct correspondence of octal code to node address. At level zero one partial node is allocated. At level 1, eight nodes are stored : (E P F P E E P E). At level 2, 64 nodes are stored, indexed by the octal code. For example, for the object of figure 3.1 we could assign an array :

```
leafnode_type level_2[64];
```

of which only 24 entries would be used; e.g. level_2[60] = E, level_2[61] = F etc.
The pointer based representation defines a node as a record structure, including an array of eight child pointers to descendant nodes. The precise nature of the record structure will depend on the memory/speed requirements of the system. Typically, variant structures are used, so that terminal nodes do not contain the child pointers. A variety of information can be stored at a node, at the expense of memory. This will be returned to in more detail in section 4.4.3.3.

The binary octree pointer based representation of the object in Figure 3.1 is that shown in Figure 3.5

The treecode representation is:

\[ P(E(EEEEE)F(EEEEEEF)E(EEEEFEE)) \]

The linear octree representation of figure 3.1 is:

(17), (2), (34), (61), (62), (65).

The most appropriate encoding method is a subject of much discussion. A good summary is provided in [SAMET88a]. In the author's implementation of UCL3D, octrees and quadtrees are stored as treecode for filing, and expanded to pointer based structures for interactive manipulation. The Boolean operations discussed in Chapter 7 could be implemented directly in treecode, but, because the visualisation and bilinear transformation operations are required interactively, the pointer based structure is used throughout. Explicit octrees were used in an early implementation, and are used in the octree creation stage, where the top level octree is first built, and then reduced to create the hierarchy. It would not be sufficient to store only Full nodes, so the linear octree is probably inappropriate. To save space in the pointer based octree, leaf nodes have no child pointers. In addition, at the level one below the top, the eight children of a Partial node are stored as members of a record, rather than via pointers.
3.4.4 Properties of octrees

Generally speaking, results for quadtrees are simply extended to octrees and higher dimensions. It is only historical that quadtrees have a larger literature. Thus references to basic results derived for quadtrees can be used for octrees and vice-versa. Many authors have pointed out that octree/quadtree algorithms often have a time complexity linear in the number of nodes in the tree [SAMET88a]. The power of this property comes when the number of nodes in the tree is examined as a function of the scene complexity. The basic result for a quadtree was proved by Hunter and Steiglitz [HUNTER79]. They found that the upper bound of the number of nodes is linear in the perimeter of the (2D) image and the resolution, apart from a small constant. Resolution here is just the number of levels in the tree (the depth).

By extension, the k-tree complexity theorem states that the number of nodes in a k-tree representing n-dimensional space (recall $k = 2^n$) is linear in the resolution plus the number of (n-1)-interfaces [SAMET88a]. The result follows from realising that nodes have to divide only near an interface. Meagher defines the scene complexity as the number of nodes in a tree [MEAGHER82]. A further corollary is that the performance of an algorithm in n-dimensions is often of the order of an (n-1)-dimensional algorithm operating on a digital array. Note that the complexity theorem gives a bound. The actual number of nodes may be considerably less.

It is easy to show, (see for example [WENG87]), that the number of Partial nodes in an octree representation is approximately $1/7^{th}$ of the total number of nodes, or about 14%. By extension the number of Partial nodes in a k-tree is approximately $1/(k-1)$. Thus the advantage of storing only leaf nodes is not necessarily very great.

3.4.5 Basic Operations on Octrees

Boolean and bilinear transformation operations are discussed in detail in Chapter 7. The former are linear in the sum of the nodes of the input trees. Translation is linear in the sum of the nodes in the input and output trees. Other simple operations are
reflection, rotation by 90 degrees, and scaling by a power of two. Rotation by an arbitrary angle is more complex to analyse, and will also be discussed in Chapter 7.

Sometimes in the algorithms to be discussed an octree or quadtree node will have to be condensed (sometimes called merging). This process converts an inhomogeneous k-tree node to homogeneous if its k children are all of the same status (and colour in a grey tree). The node will take on the status (and colour) of all its children, and they are deleted. Generally speaking, this is performed recursively at the end of an algorithm. This is an example of a post-order traversal algorithm, (also called top-down, in the author’s convention, or bottom-up if the root of the tree is the highest level as in some references), meaning that the children of a node are examined before the node itself.

The opposite procedure to condensing is spawning or splitting of an homogeneous node. This is often required in transformation algorithms. Here the homogeneous node is turned to inhomogeneous and k children are created, each with the status (and colour) of the parent. The children so created will sometimes be referred to as virtual children.

It was stated in section 3.4.3 above that a pointer based octree is random access. Suppose we want to examine the value of a voxel, or, equivalently the value of a point r contained in a continuous voxel V(i,j,k). This is the point location problem. It is solved by considering the rather simple relationship between the octal code of a node and the digital points it contains.

If a point r and digital point (i,j,k) are related by equations (2.3) and (2.4), then the octal code to r is defined as the octal code of the top level node representing the voxel V(i,j,k) that contains r. It is found by bitwise operations only on the integer triple (i,j,k). Let an integer A have a bit representation A_n\ldots A_1 A_0 and \( \beta_k(A) \) be a function returning bit k of A, then the m\textsuperscript{th} element of the octal code J is given by

\[
\begin{align*}
c_m(J) &= \beta_{(n-1-m)}(i) + 2\beta_{(n-1-m)}(j) + 4\beta_{(n-1-m)}(k) \\
&= \beta_k(i) + 2\beta_k(j) + 4\beta_k(k)
\end{align*}
\]

where n is the top level of the tree, and J is given by concatenation of the elements (Equation (3.3)). The concatenation may formed by an Exclusive Or (XOR) with each of the \( c_m \) shifted left by \( 3 \times (n-m-1) \) bits, so that J is stored in \( 3n \) bits. A pointer-
based tree may be searched along this path. A linear octree can be searched by a binary search. Treecode will require a sequential search. An explicit tree can be randomly accessed.

A related process is neighbour finding, which is a requisite in many algorithms. It is assumed that the required neighbour to a node $\Psi$ is of the same size or larger. One possibility is to compute the octal code of the neighbour and use the point location method to search from the root until the neighbour is found [KLINGER79]. The more usual technique which will be referred to as walk down and up, is to walk down the tree (towards the root) until a common ancestor $\Psi_c$ is found, then walk back, in a "reflection" of the path walked down [SAMET88a, GARGANTINI82b].

A method discovered by the author combines these two techniques. The key idea is to maintain a list of the nodes in the path from root to $\Psi$, called the ancestor array, which will contain $\Psi_c$ as the $p^{th}$ entry, where $p$ is the level of $\Psi_c$ referred to as the pivot level. Thus we only need to walk up from the known pivot, along the octal code of the required neighbour. Figure 3.11 shows the two different methods. This technique will be referred to as walk up from pivot. The following theorem holds:

![Diagram](image-url)

Figure 3.11: Two different methods for finding a neighbour.
Theorem 3.1: For the "walk-up-from-pivot" method of neighbour access, the expectation value for the number of node accesses to find a neighbour approaches two asymptotically from below with increasing depth of tree. For the "walk-down-and-up" method it approaches four, asymptotically from below.

Proof:

For the "walk-up-from-pivot" method, let there be M nodes at level m of the octree where $0 < M \leq 8^m$. For M/2 nodes the neighbour will be the immediate co-child, requiring one access. Of the remaining M/2 nodes, M/4 will require access via a Partial node at level m-2, requiring a maximum of two accesses, less if the neighbour is a higher homogeneous node. In general M/2^i will require at most i accesses. Thus the total number is bounded by $M(\frac{1}{2} + \frac{2}{4} + \ldots + \frac{m}{2^m})$. The sum of a series of the form $(x + 2x^2 + \ldots + nx^n)$ is given by:

$$S = (1-x)^2(x - (n+1)x^{n+1} + nx^{n+2})$$

Thus for $x = \frac{1}{2}$, the summation approaches two asymptotically from below, which gives the bound on the expectation value with increasing m.

For the "walk-down-and-up" method the number of accesses is at least twice that for the walk-up-from-pivot method in every case. Thus each term in the series is at least doubled implying an asymptotic value of four for the summation.

In addition the walk-down-and-up method will generally require an extra pointer, from child to parent, that is not required in the representation used here. Note that the octal code of the neighbour is required. If the origin of the current node is known, and the size, then the neighbouring octal code is computed by bitwise operations. The code can also be computed during a ray tracing operation that will be discussed in section 4.4.3.3.
3.5 CONCLUSIONS

From the above discussion it appears that the most general purpose data representation would be volumetric, greyscale, and random access, and this is the representation used in the UCL 3D package. Its possible disadvantages are the large memory size and long computing times. Several groups, especially those using volume rendering, adopt this approach. Usually the representation is just a 3D greyscale array, [HÖHNE87b, ROBB88, PIZER89], with perhaps special purpose hardware to maintain the whole volume in multi port memory and access by independent processors [FISHMAN86, GOLDWASSER85]. In the UCL3D system, designed for a small minicomputer, various types of octree database are used which allows the creation of multiple objects, such as the fragments from surgical operations, that occupy relatively small amount of memory. A great advantage of the octree approach is the efficient performance of Boolean operations which allows objects to be dissected and merged.
CHAPTER FOUR

DISPLAY TECHNIQUES 1 : HIDDEN SURFACE ELIMINATION

4.1 INTRODUCTION

There are two distinct considerations in obtaining a 2D display from a 3D dataset. The first is the correct determination of the visible surfaces, also called Hidden Surface Elimination (HSE). The second is the Surface Shading process which requires a model that approximates the physical reality that would occur if a real object with real optical properties were viewed under a given set of lighting conditions, orientation etc. This can include shadows, movement blur, parallax and stereoscopic views, many of which sophistication are inappropriate in the medical application. This chapter deals with Hidden Surface Elimination (HSE). Through most of the discussion it is assumed that an object has been defined in a "hard" way, as defined in 2.1.1, so that the location of the surface of the object is well defined. Surface shading is dealt with in the next chapter.

First it is necessary to introduce the notation and concepts of Image-Space, and the relationship of Image-Space to Object-Space.

4.2 IMAGE-SPACE AND ITS RELATION TO OBJECT-SPACE

In Chapter 2, Object-Space was defined as a region $U_c$ of continuous $\mathbb{R}^3$ space, with a coordinate system at its centre. Image-Space is another region $IS_c$ which, in general has truncated trapezoidal shape, and partially intersects $U_c$, with a coordinate system ($\mathbf{x}', \mathbf{y}', \mathbf{z}')$. $IS_c$ contains a screen, a two-dimensional surface, (usually a plane), which contains the image of objects in Object-Space.

Graphics systems such as Flight Simulators and CAD design packages allow the screen to be in an arbitrary position with respect to $U_c$, and include perspective projection,
Figure 4.1: Relationship of object space to image space for general (perspective) projection.

so that clipping needs to be carried out to decide what objects in $U_c$ lie inside the intersection $U_c \cap IS_c$. In medical imaging it is very common to derive images by orthographic projection so that $IS_c$ is also a rectangular parallelepiped $X'$ by $Y'$ by $Z'$. Herman and Liu justify this by the requirement that measurements on the image should relate quantitatively to those in Object-Space [HERMAN79]. Höhne et al prefer to use perspective projection, because their system uses a fixed object position and
multiple cutting planes, that are better realised in perspective views [HÖHNE87b]. Another place where perspective may be necessary is in Radiation Treatment Planning (RTP) where a "beam's eye view" is sometimes required [GOLDWASSER86]. Perspective was implemented in the B-R facility of UCL3D, but not for the voxel facilities.

Another point is that ISc is usually defined to contain Uc totally. Then clipping need not be carried out. The Image-Space origin will be at the centre of one face of ISc and the viewing direction towards the Object-Space origin. Then ISc is bounded by:

\[
\begin{align*}
\mathbf{r}' & = \begin{bmatrix} 0 \ 0 \ 1 \end{bmatrix} \\
\mathbf{r}' & = \begin{bmatrix} 1 \ 0 \ 0 \end{bmatrix} \\
\mathbf{r}' & = \begin{bmatrix} 0 \ 1 \ 0 \end{bmatrix} \\
\mathbf{r}' & = \begin{bmatrix} 1 \ 1 \ 1 \end{bmatrix}
\end{align*}
\]

Again, as in (2.1) the direction normals are outward from the defined space.

The relationship of Object-Space to Image-Space is most easily derived by the use of the homogeneous coordinate transformation [NEWMAN81, FOLEY84]. Using this procedure a vector \( \mathbf{r} \), in Object-Space is transformed to a vector \( \mathbf{r}' \) in Image-Space by a (in general) 4 by 4 matrix \( T \). For orthographic projections \( T \) is a 3 by 4 matrix, or, equivalently, a 3 by 3 matrix with a translation vector:

\[
\mathbf{r}' = \begin{bmatrix} t_{11} & t_{12} & t_{13} & x_e \\
                t_{21} & t_{22} & t_{23} & y_e \\
                t_{31} & t_{32} & t_{33} & z_e
\end{bmatrix} \mathbf{r} + \begin{bmatrix} x_e \\
                y_e \\
                z_e
\end{bmatrix}
\]  (4.2)

The vector \( \mathbf{r}_c = (x_c, y_c, z_c) \) is the translation vector (in Object-Space coordinates) from the centre of Uc, to the centre of ISc.
Figure 4.2 : Relationship of object space to image space for orthographic projection.

Figure 4.2 illustrates this relation. The origin of Object-Space is at the centre of \( U_c \) so that it transforms to the centre of Image-Space, if the viewpoint is orientated always towards the centre of \( U_c \). When considering the discrete representation it is more convenient to locate the origin of the discrete Universe, \( U_D \), at the point \((-X/2, -Y/2, -Z/2)\). The relationship between the two is via the vector
\[ I_c = \left( \frac{X}{2}, \frac{Y}{2}, \frac{Z}{2} \right) \]  

(4.3)

Image-Space contains a screen, defined in section 2.1.4. The transform T defined by (4.2) is called a depth preserving transform, because it conserves the dimensionality of the space it operates on. The projection transform PT maps a line in \( U_c \) onto the screen by reducing the dimensionality of the space it operates on by one. Therefore the inverse \( PT^{-1} \) mapping a point \( s \) onto a line, or ray given by

\[ r = s + \lambda \hat{g} \]  

(4.4)

where \( \hat{g} \) represents the direction normal of the screen. The parameter \( \lambda \) (0 ≤ \( \lambda \) < \( Z' \)) is called the step along the ray path, or the ray-step. If \( \hat{g} \) is along one of the principle directions ±\( \hat{A} \), ±\( \hat{Y} \), ±\( \hat{Z} \) of Object-Space then T is said to be an orthogonal projection.

Some Image-Space functions can now be defined:

A Silhouette image \( Sh_c(Q_c) \) of an object \( Q_c \) is a binary function such that

\[ Sh_c(s) = \begin{cases} 1 & \text{if } s = PT(r), r \in Q_c \\ 0 & \text{otherwise.} \end{cases} \]  

(4.5)

By comparison with (2.12), it can be seen that the silhouette function satisfies the criterion of being a characteristic function of an image.

The depth image or Z-buffer image \( Z_c(Q_c) \) of an object \( Q_c \) is a real function such that

\[ Z_c(s) = \begin{cases} \min(\hat{g}.T(r)) & \text{if } s = PT(r), r \in Q_c \\ \text{Not_Object} & \text{otherwise.} \end{cases} \]  

(4.6)

where Not_Object is some value used to indicate the background (usually a large positive number), T is the depth preserving transform, and PT the corresponding projection transform. Thus \( Z_c \) records the depth from the screen to the nearest point in the object. \( Sh_d \) and \( Z_d \) are the digitisation of \( Sh_c \) and \( Z_c \) respectively.

The displayed image (or just display) \( D_y_c \), of an object \( Q_c \) is the derived image that attempts to assign realistic shading to the image. Some shading techniques (discussed in section 5.2) derive \( D_y \) as an operation on \( Z \).
These definitions will be of use in Chapter 6, where the problem of creating an object model from a set of displays is examined.

4.3 GENERAL METHODS

Regardless of the database employed it is possible to delineate three basic HSE strategies.

1) Back to Front
2) Front to Back
3) Ray Tracing

The first two of these will be referred to as Object Driven and the third as Image Driven. The reason the distinction is made is the following:

Object Driven algorithms proceed through the space of the object, transforming object elements to Image-Space, and testing against image elements.

Image Driven algorithms proceed through the space of the image, transforming image elements into Object-Space, and testing for corresponding object elements.

The significance is that sometimes the object can be accessed sequentially in the former, whereas the latter need a random access. As contended in section 3.5, random access representations are the most appropriate for any but the simplest operations. The following discussion is meant to be rather general. A detailed review of different HSE strategies was given by Sutherland et al [SUTHERLAND74].

4.3.1 Back to Front algorithms

In general terms the Back to Front (BTF) algorithm is
1) Sort object elements by depth from the image screen ($\lambda$)
2) For all elements of the object, in decreasing value of $\lambda$
   3) Transform the object to Image-Space
   4) Determine shading
   5) Paint into the image.

In the Graphics literature this is known as the Painter's algorithm [e.g. NEWMAN81]. Generally, for example for B-R descriptions, all data elements need to be transformed and sorted (step 1), so that the correct order of painting is derived. This is often a considerable overhead. It is a distinct advantage of voxel based datastructures that this sorting step is not required. The data is said to be spatially Pre-Sorted.

Some methods store the data in a partially sorted description. For example Fuchs et al [FUCHS83] suggested a Binary Space Partition for B-R descriptions. For CSG models, octrees have been used to create a hierarchical ordering [FUJIMURA85, GLASSNER84], although these are usually applied in ray-tracing algorithms. Octrees have also been applied to B-R models [CARLBOM85].

4.3.2 Front to Back Algorithms

In general terms the Front to Back (FTB) algorithm is

1) Sort object elements by depth from the image screen ($\lambda$)
2) For all elements of the object, in increasing value of $\lambda$
   3) Transform the object to Image-Space
   4) For each pixel in the transformed object element
      If the pixel is currently "unlit"
      5) Determine shading
      6) Paint into the image.

If the available memory is large enough, a variation often used is the Z-buffer algorithm. Here the depth $\lambda$, initially a very large number, is maintained for each pixel. There is no advantage in performing step 1). Instead the steps 2) to 5) are performed,
but in the test, the pixels are painted if and only if the depth to be written would reduce the depth already present. This removes the sorting problem to a much less complex test. If for example only one convex object were in the data (e.g. a sphere) then no pixels would be over-written and the entire sort phase is saved [HERMAN79, HEFFERNAN85a]. Modern graphics workstations often have a frame buffer with bits reserved per pixel for z-buffer operations. For example 48bits per pixel allows 8 bits each for red, green, and blue, and 24bits for the z-buffer depth.

Once again however, the spatially pre-sorted voxel database allows the FTB to proceed efficiently, by eliminating the need for step 1. Further, for some datastructures, considerable advantage is made of area coherence in the image.

4.3.3 Ray Tracing Algorithms

The ideas of ray tracing are more recent than the previous two object based methods. In a generalised formulation ray-tracing is as follows:

1) For each pixel in the image
2) Derive the ray \( r = s + \lambda \hat{g} \)
3) set \( \lambda = 0 \)
4) while \( \lambda < Z' \)
5) if the ray intersects objects at \( r \)
6) determine shade, paint pixel; goto 1)
7) increment \( \lambda \)

This technique affords very sophisticated shading techniques. The ray, on successfully intersecting an object can be refracted, reflected, and split into partial rays [WHITTED80]. Transparency effects can be incorporated by attenuating the ray intensity as if through a coloured gel of a certain thickness. The literature on the subject is enormous, and the technique is the preferred one in all applications requiring high graphic realism. The disadvantage is the considerable computation overhead.
Models for reducing the algorithmic complexity include octrees [GLASSNER84, FUJIMOTO86].

When ray tracing is applied to medical volume data, it generally has a much simpler formulation (e.g. no refraction, or ray splitting), since the surfaces are not so precisely defined. The term *ray casting* is often used instead.

### 4.4 DISPLAY TECHNIQUES FOR MEDICAL DATA REPRESENTATIONS

This section discusses how the representations introduced in chapter 3 have been employed for the HSE problem in display of medical data, using one or more of the three methods: BTF, FTB, or raytracing. In particular, the application to octree structures is discussed in section 4.4.3. With respect to voxel databases, two philosophies seem to emerge. One is to treat the voxels as cubes, rendered from any viewpoint, (e.g. MIPG, and Goldwasser approaches), and the second is to organise the screen orthogonally to the data, rotating and resampling Object-Space if another view is required [GIBSON83, TIEDE87, HEFFERNAN85b, FARRELL85]. Both approaches were investigated by the author, but the latter allows faster display algorithms, including a new type of octree to quadtree raytracing discussed in section 4.4.3.3.

#### 4.4.1 Surface Representations

##### 4.4.1.1 2D Approaches (1D Primitives)

Line contours can be displayed rapidly by transforming to lines in Image-Space. No HSE is performed and no shading effects are produced. It is useful for real time rotation on a small computer. Examples may be seen in papers by Vannier and Marsh [VANNIER83], CEMAX [DEV85, WOOLSON86] and Arridge *et al* [ARRIDGE85].

The display of surfaces stored in the Directed Contour representation was developed by the MIPG group for very fast displays [UDUPA87, HERMAN88b]. Voxels are projected directly to pixels (they are not scan converted). The viewing directions are
limited to planes orthogonal to the planes of the directed contours, so that lines of pixels correspond to lines of voxels. General views have to be obtained by deriving contours in oblique planes reformatted from the original slices, [FELLINGHAM86] although this is time consuming.

The 2D approach is also that of [HEFFERNAN85a], although here the primitives are of variable length depending on the number of Fourier components used in the approximation of the contour, (see section 3.1.2.1).

4.4.1.2 3D Approaches (2D Primitives)

4.4.1.2.1 3D B-R Approach

Rendering of objects defined by a polygonal B-R is a very well understood problem [NEWMAN81, FOLEY82]. Methods were developed for the UCL3D package using a simple BTF algorithm preceded by a Back Surface Cull (BSC) (i.e. deletion from the display list of facets pointing away from the observer). Gouraud, Phong and specular shading were implemented, with arbitrary position of light source and view point. Example images were shown in figure 3.2. A Z-Buffer of the image was maintained in order to implement quantitative measurements. This will be returned to in Chapter 9.

4.4.1.2.2 3D Voxel Boundary Approach

The display of 3D voxel boundary representations was the purpose of the 3D83 package developed by Herman et al at MIPG, and incorporated on the GE9800 scanner and subsequently superseded by 3D98 [UDUPA86b].

Essentially the task is similar to the B-R problem. Recall (Section 3.1.2.2) that the description is in terms of six lists, one for each orientation of the face of a cube. The Back Surface Cull (BSC) is achieved by considering only those three lists for faces visible from a particular viewpoint. For restricted viewpoints only two, or even one
(orthogonal projections) list is required. The transform of each square face is a rhomboid that is actually scan-converted like a B-R facet. In the absence of perspective projection, only three rhomboid shapes are possible, which means that a lot of pre-computation can be done. Originally, each face was shaded according to its orientation to the light vector, so that the voxel edges were visible, followed by a low-pass smoothing filter. More sophisticated shading algorithms are described in section 5.2. Anti-aliasing was employed to reduce the "blockiness" of the images, by generating the image at a finer resolution than the display screen, and then averaging down.

The MIPG display process first described by Herman and Liu [HERMAN77] used the Z-buffer approach to solve HSE, and relied in part on certain mathematical properties of the faces of cubes in an ordered array, that are not valid for other shapes [HERMAN79]. Such properties led to theorems that produced an order of magnitude speedup over other existing B-R display algorithms, for the display of the (typically) 10,000 to 15,000 cube faces.

4.4.2 Volume Representations

The first example of the display of volume representations of medical data is often attributed to Meagher's work on octrees [MEAGHER82]. Before examining these methods, this section discusses the methods applicable to the other representations of section 3.2.

4.4.2.1 Digital Arrays

The first description, in terms of a binary array, was by Freider et al [FREIDER85], which used a BTF algorithm. The data was read out in a "slice by slice, row by row" fashion. This paper introduced a number of "tricks" for computational efficiency. Principle among these were:

1) Pre-computing the values $k^*t_{ij}$, where $t_{ij} \in T$ the transform matrix defined by (4.2), for integer $k$ up to the dimensions (L by M by N) of the discrete scene
(section 2.1.1). Then the transform of an integer triple \((i,j,k)\) could be derived by additions only.

2) Scaling the image so that a projected voxel occupied no more than one pixel, and could be shaded without any scan conversion.

3) Packing of a 2 by 2 by 2 voxel array into one byte (one bit per voxel) so that half resolution could be achieved, by considering the number of bits set in each byte (via a Look Up table). This may be compared with Srihari’s mode-rule described in section 3.3.2.

The extension of this to a greyscale array was used by the company (3D DDI), using 4 bits per voxel so that eight voxels packed into 32 bits [GOLDWASSER84]. This system was based around a multiprocessor architecture for rendering 3D digital arrays, and the data was stored in a recursive way, exactly as in an octree; therefore the algorithm used was termed recursive BTF. The images in figures 1.3 and 1.5 were produced by a BTF algorithm, simply reading on a slice by slice row by row basis, and displaying each voxel density after transformation.

The FTB algorithm, operating on a binary digital array was used by Gibson [GIBSON83], restricted to orthogonal directions.

Ray Tracing of digital arrays has been the subject of much recent literature. It does not seem to have been applied to binary digital arrays however. The application to greyscale digital arrays falls into the class of Volume Rendering algorithms, described below (section 4.4.2.3).

4.4.2.2 Marginal Indexing and Segment-Indexed (Segment Endpoint)

Apart from the description by Srihari [SRIHARI81], use of marginal indexing representations either in binary or greyscale form, does not seem to have been developed. The Segment Endpoint (SE) representation was used for BTF display by [TRIVEDI85] and for ray tracing by Tuy and Tuy [TUY84]. The closely related Segment Indexed representation was used to implement what is acknowledged as the fastest FTB algorithm for binary volume data. Reynolds made a trivial restriction on
the viewpoint that allowed the line segments in the object to transform into image segments of the screen [REYNOLDS87]. Thus the screen was also described by a 2D segment indexed model, called the *dynamic screen*. This structure was actually used to maintain an efficient representation of the characteristic function (2.12) of the image. The display itself was rendered directly.

If, for example, the object was encoded in the order ZYX (see section 3.2.2) then the viewpoint was restricted to that obtained by a rotation first around the x-axis, then around the z-axis. It is a property of the closure of the 3D rotation space group SU(2), that such a transform allows all viewpoints to be reached, only placing a restriction on the orientation of the screen. Then the line segments parallel to the x-axis also lie parallel to the x'-axis of the screen. Figure 4.3 illustrates this relation.

![Figure 4.3](image.png)

*Figure 4.3*: Image to Object space relationship for the Reynolds algorithm.

This algorithm was also incorporated into the VOXEL PROCESSOR system [GOLDWASSER86]. Although not commented on, it is presumably necessary to build the SE description from the greyscale array "on the fly", which may incorporate some overhead.
4.4.2.3 Volume Rendering Methods

These methods operate on greyscale digital arrays, and use ray-tracing. They do not perform any segmentation step to produce an explicit object representation. However, the methods may be divided into those that perform an implicit segmentation to find the hard surface $\partial \Phi$, and those that actually attempt to represent the complete volume. The author will use the term *dynamic segmentation* to refer to the former; in a recent letter to IEEE Computer Graphics and Applications (March 1989, Page 90), Reynolds suggested the term *Volume Compositing* for the latter.

In dynamic segmentation, rays are traced orthogonally into the array and a decision function $\Lambda$ used to determine where they stop. Using the notation of section 4.2, let $f(\xi')$ (the density function in Image-Space coordinates), be expressed as $f(s, \lambda)$. Then the depth buffer is derived by

$$Z(s) = \Lambda(f(s, \lambda))$$

(4.7)

The Hamburg group used various alternatives for $\Lambda$. For example:

$$\Lambda = \min \{ \lambda \mid \rho_{\text{low}} \leq f(s, \lambda) \leq \rho_{\text{high}} \}$$

(4.8)

(the depth is the smallest value of $\lambda$ such that the corresponding voxel falls into the density range $[\rho_{\text{low}}, \rho_{\text{high}}]$)

$$\Lambda = \min \{ \lambda \mid \frac{\partial f(s, \lambda)}{\partial \lambda} \geq G \}$$

(4.9)

(the depth is the smallest value of $\lambda$ such that the differential along the ray is greater than some threshold value $G$)

$$\Lambda = \min \{ \lambda \mid \nabla f(s, \lambda) \geq G \}$$

(4.10)

(the depth is the smallest value of $\lambda$ such that the gradient is greater than some threshold value $G$).

Other possibilities suggested were to segment the ray into disjoint sections satisfying one of these criteria, and select different segments interactively, thus peeling away layers of material [HÖHNE87a]. In their method, the greyscale array is fixed and any other view must be obtained by rotation and resampling the entire data using trilinear interpolation [TIEDE87].
All these techniques from the Hamburg group effectively form a 1D-segmentation operation based on properties along the ray. An attempt to improve on this by incorporating surface tracking from the candidate voxels was described by Troussset and Schmitt [TROUSSET88]. Their method imposed connectivity between voxels found in neighbouring rays, and a minimum volume for detected regions, (to eliminate noise). The implementation was limited to view points in the plane of the slice, and used only 2D-connectivity; reprojection, and reformatting were also possible. The results did not appear noticeably different from other methods, but some interesting research ideas could develop from this approach. One unfortunate consequence is that rays are no longer independent of each other, so a parallel processing implementation is not so forthcoming.

Instead of imposing a decision function, volume compositing techniques integrate some property along the ray. If the property is just $f(s,\lambda)$ then the method is called reprojection. Harris et al first used this technique but allowed a density mapping on $f$, that allowed ranges to be heightened or "dissolved out" [HARRIS79]. The more recent work of Levoy, from UNC, integrated a colour obtained by shading operations on the vector $Vf$, weighted by an opacity that itself was a function of both $f$ and $Vf$ [LEVOY88]. Thus the opacity function was really a feature space classifier (cf [TRIVEDI86], described in section 2.3.4), although this point was not made by Levoy. The results of this technique are very "realistic" (in the sense that "steppy" artifacts are eliminated) but their clinical value remains to be demonstrated.

The PIXAR approach also uses volume compositing, but here the different tissue types are manually segmented and then given attributes such as translucency, colour, and reflectivity on a rather arbitrary basis, so that the final picture "look nice" [FRENKEL89].

The issue of whether volume rendering techniques give a truer representation of the real world is rather a contentious one. However, it is certainly a fast-growing area, as may be seen from the number and diversity of papers at the recent Volume Visualisation Workshop held at the University of North Carolina, Chapel Hill in May 1989 [PROCvvw89].
4.4.2.4 Other Methods

A recent approach by Cline et al [CLINE88] is actually a surface description, obtained by trilinear interpolation on the original data to find voxels that contain the boundary of the characteristic function \( \partial \Phi_c \). These voxels are interpolated so that their projections are the same size as the required pixels (as in [FREIDER85]), but this means that arbitrary zooming is possible.

4.4.3 Octree Methods

As discussed in Chapter 3, the octree representation has been used extensively in UCL3D, largely for the ease of manipulation of multiple objects for the space planning aspects of surgical simulation. In section 4.4.3.3 a detailed description of a new ray-tracing algorithm, for orthogonal projections of octrees, is described. In this section, the more general viewpoint display problem is discussed for each of the octree structures described in section 3.3 - Binary, Masked Binary, Greyscale, Min-max Greyscale.

One point to note about octree algorithms is that they attempt to take advantage of a quadtree representation of the screen. The quadtree can hold either the image, or its characteristic function, in a similar way to the use of the dynamic screen in the Reynolds algorithm of section 4.4.2.2. In the latter case, the display will be rendered directly into a display buffer. For the description of the algorithms in this thesis, it is assumed that the quadtree holds the displayed image, and that if an octree node is to be displayed, there exists a function `shade_of_node` that will return the appropriate value to assign to the corresponding quadtree node.

4.4.3.1 Back To Front (BTF)

The basic BTF octree algorithm was described by Meagher [MEAGHER82]. It depends on the observation that for any viewpoint there exists an order of the eight octree children that allows them to be painted in so that none earlier in the sequence overpaint those later in the sequence. Furthermore the property is recursive, so that the
The order of visit of octree nodes is determined by the viewing direction.

Children can be run over in pre-order. For example, for the view of the object in figure 4.4, the nodes could be accessed in the order [2,0,3,6,1,4,7,5]. The *view-direction* will be labelled as that node visited *last* in the BTF traversal. Unlike in the Freider algorithm, the projected octree nodes will have variable size, and must be properly scan-converted. However, once again, the projected nodes are all the same shape. The BTF algorithm is therefore:

\[
\text{BTF}(\Psi) \quad \text{--- Back to Front display of octree node } \Psi.
\]

1) If \(\Psi.\text{status} = \text{Empty}\) then
   return
2) Else if \(\Psi.\text{status} = \text{Full}\) then
   display_node(\(\Psi\))
3) Else \quad \text{--- The node is Partial}
   for i in visit order of octree children do
      BTF(\(\Psi.\text{child[i]}\))
   endfor

Later in this section and in chapter 6, extensive use is made of this visit order property.

The display of an octree node is a six-sided convex polygon for completely general views, which reduces to a four sided polygon for some restricted views. The *Convex Polygon* (CP) of an octree node is thus treated formally as the intersection of six
(possibly four) infinite half planes (in Image-Space) described by inequalities of the form:

\[ a.x + b.y + c \leq 0 \]  \hspace{1cm} (4.11)

where \( a \) and \( b \) are the same for all nodes and \( c \) is a function only of node position and level. Thus \( c \) may be derived from three \( 2^n \) look up tables for each of the six half spaces, where these tables need only be calculated once for each image view position.

Consider now the form of the BTF algorithm in the Masked Binary representation. The 2-neighbour flags are used to determine whether some or all of the display faces of the node are masked, hence the name - Masked Binary. The key idea here is to construct a view code that represents which three (possibly two) faces are visible. The code has bits set where neighbour flags would be set on the faces visible from that direction. Eight different codes are possible for views including three faces, twelve for views including two faces, and six to include the orthogonal projections. Table 4.1 shows the eight codes for three faces. The others are derived quite easily. For any given view direction the view mask is ANDed with the complement of the neighbour mask. The result will be Null (EIEIEIEIEIE) if all neighbours are set.

The Result code is also a 12-bit code that will have at most three (possibly two) elements set. If none are set then the node is invisible. If all three (possibly two) are set then the node is displayable as a CP in the normal way. If one or two (one) are set then a part of the CP is visible. Consider the node that was shown in Figure 3.6, to illustrate the principles of the Masked Binary representation. It has neighbour code:

\[ E \ E \ E \ F \ F \ F \]

The complement of this neighbour code is:

\[ F \ F \ F \ E \ E \ E \]

If the view direction is the one shown in Figure 3.6, it is labelled by the last octant visited in BTF (5). From Table 4.1 the view code for view direction 5 is:

\[ E \ F \ F \ E \ E \ F \]

(i.e. faces \( x^+, y^+, z^+ \) are potentially visible). From

\[ \text{result code} = \text{view code AND } \neg(\text{neighbour code}) \]

gives

\[ \text{result code} = E \ F \ F \ E \ E \ E \]
Let the neighbour mask be encoded as follows:
\[ lx'lx'ly'ly'lz'lz' \]

Then the view codes are:

<table>
<thead>
<tr>
<th>Octant</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>IF IE IF IE IF IE</td>
</tr>
<tr>
<td>1</td>
<td>IE IF IE IF IE IF</td>
</tr>
<tr>
<td>2</td>
<td>IF IE IE IF IF IE</td>
</tr>
<tr>
<td>3</td>
<td>IE IF IE IF IF IE</td>
</tr>
<tr>
<td>4</td>
<td>IF IE IF IE IE IF</td>
</tr>
<tr>
<td>5</td>
<td>IE IF IF IE IE IF</td>
</tr>
<tr>
<td>6</td>
<td>IF IE IE IF IE IF</td>
</tr>
<tr>
<td>7</td>
<td>IE IF IE IF IE IF</td>
</tr>
</tbody>
</table>

Table 4.1: View code masks

This result_code is passed to display_node which determines that only \( x^+ \) and \( y' \) are to be rendered:

\[ \text{Masked_BTF}(\Psi) \quad \text{--- Back to Front Display of the Masked Binary Octree node } \Psi. \]

1) Result_code := \( \neg(\Psi.\text{code}) \) AND view_code
2) If \( \Psi.\text{status} = \text{Empty} \) or Result_code = Null then return
3) Else if \( \Psi.\text{status} = \text{Full} \) then display_masked_node(\( \Psi \), Result_code)
4) Else --- The node is Partial for i in visit order of octree children do BTF(\( \Psi.\text{child}[i] \)) endfor

The extension to the grey octree representation is also easy. Here step 2) of algorithm BTF is just:

2) Else if \( \Psi.\text{status} = \text{Full} \) and \( \Psi.\text{colour} \) is in threshold range selected
For the min-max grey octree representation the same amendment applies, and in addition step 3) is:

3) Else if $\Psi.\text{max} > \rho_{\text{Low}}$ and $\Psi.\text{min} < \rho_{\text{High}}$

The efficiency of these algorithms is compared in table 4.2. Here the skull of section 3.3.3 was displayed in the same view with all four representations. The grey scale representation is much slower, but is reduced in complexity by the use of the min-max structure.

4.4.3.2 Front-to-Back (FTB)

The FTB octree display algorithm was described by Meagher [MEAGHER82] and by Amans et al [AMANS86]. A simpler algorithm for orthogonal view was described by Doctor and Toborg [DOCTOR81]. The FTB algorithm makes use of the following observations which are not often stated explicitly, and are presented here without proof:

THEOREM 4.1: If a quadtree with root $\Omega_{\text{root}}$ completely covers the projected octree node $\mathbf{CP}(\Psi_{\text{root}})$, then the projection of any subnode $\Psi'$ contained in $\Psi_{\text{root}}$ at level $m$, is completely covered by at most four quadtree nodes $\Omega'(0) \rightarrow \Omega'(3)$ that are contained in $\Omega_{\text{root}}$ and are also at level $m$.

THEOREM 4.2: The projection of any child node $\Psi'.\text{Child}[i]$ at level $m+1$ is completely covered by at most four quadtree nodes $\Omega^*(0) \rightarrow \Omega^*(3)$ at level $m+1$ where

<table>
<thead>
<tr>
<th>Representation</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binary Masked</td>
<td>78</td>
</tr>
<tr>
<td>Binary Unmasked</td>
<td>257</td>
</tr>
<tr>
<td>Grey</td>
<td>585</td>
</tr>
<tr>
<td>Pyramid Grey</td>
<td>440</td>
</tr>
</tbody>
</table>

Table 4.2: Comparative times for the BTF algorithm on the four different octree representations.
the set \( A^* = \{ \Omega^*_k \mid k = 0 \text{ to } 3 \} \) is a subset of the set \( A^* = \{ \Omega^*_i, \text{child}[i] \mid k = 0 \text{ to } 3, i = 0 \text{ to } 3 \} \).

Note that since the cardinality of \( A^* \leq 4 \), the cardinality of \( A^* \leq 16 \), but that Theorem 4.2 is always true. Figure 4.5 shows the relation. The CP of the octree root is completely covered by the quadtree root. At level 2, the CP of root.child[3].child[6] is completely covered by quadtree nodes \{12,13,30,31\}, all at level 2.

The significance of these theorems is that they allow the FTB algorithm also to be recursive, and as long as a quadtree of the image is correctly maintained, it becomes efficient to test large Partial nodes against the quadtree. We will see later that these theorems are characteristic of octree theorems and consequent algorithms, namely that algorithms such as translation, rotation, and projection achieve their recursive nature by considering sets of nodes in two or more trees, the target of which completely contains the source.

In outline, the FTB algorithm is:

\[ \text{FTB}(\Psi, A) \quad \text{--- Front to Back display of octree node } \Psi, \text{ whose projection is covered by the (maximum four) quadtree nodes in the set } A. \]

1) If \( \Psi.\text{status} = \text{Empty} \) then return
2) Else if \( \Psi.\text{status} = \text{Full} \)
   2.1) For quadnodes \( \Omega \) in \( A \) do
       paint_oct_into_quad(\( \Psi, \Omega \))
   endfor
3) Else --- The node is Partial
   4) For i in visit order of octree children do
   5) Set \( A^* \) to Empty
   6) For k in 0:3 do
   7) If \( \text{quadrant}((A[k]) \cap \text{bounding_rectangle}(\text{CP}(\Psi)) = \emptyset \) then
      next k at 6)
   8) If \( A[k] \) is Empty
      8.1) create the children of \( A[k] \)
   9) For m in 0:3 do
10) If \( \text{status}(A[k].\text{child}[m]) \neq \text{Full} \) and \( \text{quadrant}(A[k].\text{child}[m]) \cap \text{bounding_rectangle}(\text{CP}(\Psi)) \neq \emptyset \) then
       put \( A[k].\text{child}[m] \) into the list \( A^* \).
12) If \( A^* \) is not empty
    FTB(\( \Psi.\text{child}[i], A^* \))
    endfor
13) Condense the set \( A \)
Figure 4.5: The Octree to Quadtree relations for the FTB algorithm.
procedure paint_oct_into_quad($\Psi, \Omega$)

1) If($\Omega$.status = Full) then
   return
2) Else if($\Omega$.status = Partial) then
   2.1) for i in 0 : 3 do
         paint_oct_into_quad($\Psi, \Omega$.child[i])
         endfor
   2.2) condense($\Omega$)
3) Else test $\Omega$ for intersection with CP($\Psi$)
   3.1) case inside :
         $\Omega$.status := Full
         $\Omega$.colour := shade_of_node($\Psi$).
   3.2) case outside :
         return
   3.3) case intersects :
         $\Omega$.status := Partial
         Perform 2)

A complete description of this algorithm was given by Amans et al [AMANS86]. The use of Masked Binary structures could afford some speed up since it allows a completely masked node to be considered Empty as in BTF. The extension to Grey and Min-Max Grey structures is also similar to the BTF case. Some preliminary timings of this algorithm did not show a great improvement over BTF, although it is likely that it could be made so, since its complexity is certainly less. This would be a suitable line of future development.

4.4.3.3 Ray-Tracing

As mentioned in section 4.3.3, octrees have been employed in ray tracing algorithms, although in a rather different way, as a method of space subdivision to reduce object-ray intersections in vector octrees [GLASSNER84, FUJIMOTO86]. Then a ray is tested against the contents of a node only if

   i) It intersects the node
   ii) The node is non-empty.

The adaptation of such techniques to raster octrees is possible, but does not produce greater efficiency than FTB methods. The situation is different when considering orthogonal projections however, and this case is treated separately in the next section.
4.5 ORTHOGONAL OCTREE ALGORITHMS

For orthogonal projections it is efficient to maintain the coherence of the 3D scene by projecting to a screen represented as a 2D quadtree. If a FTB octree to quadtree method is used then some saving is achieved over BTF if a Partial node projects onto a fully "lit" area of the screen which implies that none of the descendant nodes need be accessed. In practice the test for whether a node is fully hidden may be as expensive as repainting.

If an N by N image were raytraced into an N by N by N digital array, the complexity is between $O(N^2)$ and $O(N^3)$. Reynolds et al [REYNOLDS87] state that it is of the order of the volume of the complement of the objects being imaged. If the object were a completely full array of voxels the order would be $N^2$. If it were a completely empty array the order would be $N^3$. But if the image were represented as a quadtree and the object as an octree the complexity would be $O(1)$ for both cases. This trivial example suggests that some efficiency may be gained if the "rays" traced were of the size of quadtree nodes, and the progression of the ray was in step sizes determined by an octree representation of the 3D array. In this section an algorithm for performing this process is presented and compared with FTB and BTF. Ray tracing, as the name implies, physically models rays of light, sometimes many per pixel, as they reflect/refract and selectively absorb from multiple object surfaces until meeting a light source or becoming lost. Therefore the new algorithm will be referred to as Node-tracing to distinguish it from the more sophisticated methods used in "true" ray tracing.

4.5.1 Further Notation and Definitions

If an octree node $\Psi$ projects exactly onto a quadtree node $\Omega$, then for each child node $\Omega$.child[i] (i = 0 to 3) there correspond two octree children $\Psi$.child[front(i)] and $\Psi$.child[back(i)], where front(i) is the front child and back(i) is the back child. The set {front(i),back(i)} (i = 0 to 3) is a permutation of the set {0...7}.
Thus at level \( p \) a quadtree node corresponds to the projection of the \( 2^p \) octree nodes whose octal codes form the set \( \{ \|, \| \alpha(c_m(K(\Omega))) \} \), \( (m = 1 \) to \( p) \), where \( \alpha \) is either front or back. To be more explicit this set contains the nodes whose octal codes are:

\[
J = \text{front}(c_1(K(\Omega))) \| \text{front}(c_2(K(\Omega))) \| \ldots \| \text{front}(c_p(K(\Omega))) \| \text{front}(c_p(K(\Omega)))
\]

\[
J = \text{front}(c_1(K(\Omega))) \| \text{front}(c_2(K(\Omega))) \| \ldots \| \text{front}(c_p(K(\Omega))) \| \text{back}(c_p(K(\Omega)))
\]

\[
J = \text{front}(c_1(K(\Omega))) \| \text{front}(c_2(K(\Omega))) \| \ldots \| \text{back}(c_p(K(\Omega))) \| \text{front}(c_p(K(\Omega)))
\]

\[
J = \text{back}(c_1(K(\Omega))) \| \text{back}(c_2(K(\Omega))) \| \ldots \| \text{back}(c_p(K(\Omega))) \| \text{front}(c_p(K(\Omega)))
\]

\[
J = \text{back}(c_1(K(\Omega))) \| \text{back}(c_2(K(\Omega))) \| \ldots \| \text{back}(c_p(K(\Omega))) \| \text{back}(c_p(K(\Omega)))
\]

(4.12)

It is precisely these nodes that are traced by the new algorithm presented here, and the relationship between a quadtree node and the above octree node set is the key idea in the description of the algorithm.

For any octree node the vector \( \mathbf{a} = \text{centre}(\Psi) \) is transformed under \( T \) to the vector \( \mathbf{q} = \text{centre}(\Omega) \), where \( \Omega \) is the corresponding quadtree node. Let \( \text{proj}(k) \) be the child of a quadtree node onto which octree child \( k \) projects. Thus \( \text{proj}(\text{front}(c_m)) = \text{proj}(\text{back}(c_m)) = c_m \) and the quad code \( K \) of the quadtree node formed from the projection of an octree node \( \Psi \) with octal code \( J \) is given by \( K = \Xi_p[\text{proj}(c_m(J))] \). Conversely the inverse projection of \( \Omega \) gives just the octree node set defined in (4.12) above.

Let \( \text{offset}_{m,k} = \text{centre}(J_m \| f(c_k)) - \text{centre}(J_m) \) be the vector step in the octree involved in recursing to the front most children, i.e. \( \text{offset}_{m,k} \) is the vector step in the "ray" of a projected quadtree node, on recursion to child \( k \).

Let \( t_m = E_m g \) define the vector step across a node at level \( m \).

Recall that for any point \( r \) in \( U_c \) there is a path (octal code) to the voxel containing \( r \), given by \( J = \text{octal_path_to}(r) \), as derived in section 3.4.5.
Using this notation figure 4.6 shows a node at depth 3 of a 4-level octree, so that the world has edgelength $N = 16$. The node outlined in grey is labelled $\Psi^{(3)}$, to indicate that it is at level 3. Its immediate parent is $\Psi^{(2)}$, and its grandparent is $\Psi^{(1)}$. Similarly the projected quadtree node is $\Omega^{(3)}$, with parent $\Omega^{(2)}$, and grandparent $\Omega^{(1)}$. 

Figure 4.6: Octree/quadtree definitions.
Then:

\[ J^{(1)} = \text{octal}_c(\Psi^{(1)}) = 7 \quad \text{quad}_c(\Omega^{(1)}) = 3 = \text{Proj}(J^{(1)}) \]
\[ J^{(2)} = \text{octal}_c(\Psi^{(2)}) = 73 \quad \text{quad}_c(\Omega^{(2)}) = 31 = \text{Proj}(J^{(2)}) \]
\[ J^{(3)} = \text{octal}_c(\Psi^{(3)}) = 734 \quad \text{quad}_c(\Omega^{(3)}) = 312 = \text{Proj}(J^{(3)}) \]

\[ \text{centre}(\Psi^{(1)}) = (4,4,4) \quad \text{origin}(\Psi^{(1)}) = (0,0,0) \quad \text{centre}(\Omega^{(1)}) = (4,4) \]
\[ \text{centre}(\Psi^{(2)}) = (6,6,2) \quad \text{origin}(\Psi^{(2)}) = (4,4,0) \quad \text{centre}(\Omega^{(2)}) = (6,2) \]
\[ \text{centre}(\Psi^{(3)}) = (5,5,3) \quad \text{origin}(\Psi^{(3)}) = (4,4,2) \quad \text{centre}(\Omega^{(3)}) = (5,3) \]

Also:

\[ E_1 = 8, \quad E_2 = 4, \quad E_3 = 2 \]

and in the projection direction shown (-\( \hat{z} \)):

\[ \mathbf{t}_1 = (-8,0,0), \quad \mathbf{t}_2 = (-4,0,0), \quad \mathbf{t}_3 = (-2,0,0), \quad \mathbf{g} = (-1,0,0) \]

For quadtree projection in the six possible directions one possibility is shown in Table 4.3. The exact choice depends on the orientation of the quadtree required.

### 4.5.2 Algorithms

Oliver [OLIVER85] has extended the Front-to-Back algorithm to trace back through the octree when two co-children are both Empty. The algorithm presented here is a more explicit tracer. In words the algorithm is described as follows:

<table>
<thead>
<tr>
<th>j</th>
<th>( \hat{z} )</th>
<th>(-\hat{z})</th>
<th>( \hat{\xi} )</th>
<th>(-\hat{\xi})</th>
<th>( \hat{\eta} )</th>
<th>(-\hat{\eta})</th>
<th>( \hat{\xi} )</th>
<th>(-\hat{\xi})</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>4</td>
<td>6</td>
<td>7</td>
<td>7</td>
<td>5</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>6</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>4</td>
<td>5</td>
<td>7</td>
</tr>
</tbody>
</table>

Table 4.3: Relation of octree to quadtree children for orthogonal projections.
Node-Trace(octree node, quadtree node)

1: If the octree node is Full and displayable then
   paint the quadtree node appropriately
2: else if the octree node is inhomogeneous
   recurse in quadtree and octree
3: else it is Empty so step to the next node at this recursion
   level in the direction of the ray, and loop until the
   ray is lost or 1 or 2 occurs

A schematic of the idea is shown in figure 4.7.

---

Figure 4.7: Schematic of the node-tracing principle - rays are subdivided at
Partial octree nodes.
This type of algorithm presents some interesting difficulties which are now described. Essentially the problems arise when access is required to a neighbouring node that is not a co-child of the same parent. The technique to find a neighbour was given in section 3.4.5. When the neighbour is found, its level, less than or equal to the current recursion level, and greater than the pivot level, determines the size of step in $\lambda$ of the quad node "ray". In stepping past the current octree node the ray may encounter a homogeneous node at a lower level. If it is Empty it should step completely through it but continue at the same level of recursion beyond it. In Oliver’s method, the lower Empty node would be subdivided to create virtual octants. As soon as a Partial node is encountered the algorithm recurses in the quadtree and the octree. The new level of recursion will therefore be starting further into the tree than $\lambda = 0$, as would happen in normal raytracing of a 3D array. It is guaranteed that no voxels will be in front of this node at the higher level. On return from this level of recursion, no further nodes are required as the child processes will have traced either to a surface or outside the Universe.

In some cases a given octree node will be accessed more than once - namely a high level homogeneous node, preceded by inhomogeneous nodes. However it may be seen by inspection that the number of accesses will be at most the number of pixels in the quadtree representation at that level, i.e. the $2/3$ power of the number of voxels in that octree node.

```
procedure Node_Trace(oct_node, quad_node, ray_position, recursion_level, ancestor_array)
1) Make local copies of the passed parameters :
   current_oct_node := oct_node
   current_ray_position := ray_position
   current_level := recursion_level
   current_ancestor_array := ancestor_array
2) While the current_ray_position is within the Universe
3)   If the current_oct_node is full then determine its shade, and paint the
      quad_node accordingly
4)   Else if the current_oct_node is Partial then
5)      Set the quad_node to Partial
      --- Start four child rays, commencing at the front face of each of
      --- the four front children of the current_oct_node
```
for i in visit order of quadtree do
    Node_Trace(i\textsuperscript{th} front current\_oct\_node child, i\textsuperscript{th} quad\_node child, current\_ray\_position + offset to i\textsuperscript{th} current\_oct\_node child, recursion\_level + 1, current\_ancestor\_array)
endfor

Else the current\_oct\_node is Empty. Call New\_Node to get a new current\_oct\_node, current\_ray\_position, current\_level ( \leq \text{recursion\_level}) and current\_ancestor\_array :
    New\_Node(current\_ray\_position, length of current\_oct\_node, current\_level, recursion\_level)

endwhile

procedure New\_Node(current\_ray\_position, ray\_step, current\_level, recursion\_level)

--- When the current\_oct\_node is Empty, find the node next encountered by the current ray. If there is a node at the same recursion level, then return it.
--- Otherwise find the larger node in the path below the recursion level and return its level in current\_level

1) For the current\_ray\_position, and the required ray\_step, find the pivot\_level (i.e. the level where the paths to the current\_oct\_node, and the next oct\_node diverge.
2) Increment the current\_ray\_position by the ray\_step
3) Find the octal path of a node containing the current\_ray\_position after incrementing.
4) From the current\_ancestor\_array, the required octal\_code and the pivot\_level, find the next current\_node, and current\_level :
    Find\_Node\_and\_Level(current\_ancestor\_array, required octal\_code, pivot\_level, recursion\_level)

procedure Find\_Node\_and\_Level(current\_ancestor\_array, octal\_code, pivot\_level, recursion\_level)

1) Set k to pivot\_level
2) Set temp\_node to the k\textsuperscript{th} entry in the current\_ancestor\_array
3) While k < recursion\_level
4) If temp\_node is homogeneous then return it as current\_oct\_node, k as current\_level.
5) Else k := k + 1
   5.1) j := k\textsuperscript{th} element of octal\_code
   5.2) temp\_node := j\textsuperscript{th} child of temp\_node
   5.3) k\textsuperscript{th} entry in current\_ancestor\_array := temp\_node
   Endwhile
6) temp\_node is still inhomogeneous, so return it as current\_oct\_node, and recursion\_level as current\_level.
The pivot_level is easily found when the ray_position and ray_step are known. If the position of the ray is \( r = s + \lambda \hat{g} \) then after the step it is \( r' = r + l_{\text{node level}} \), where \( l_{\text{node level}} = E_{\text{node level}} g \). Thus \( r' = s + \lambda' \hat{g} \) where \( \lambda' = \lambda + E_{\text{node level}} \). If \( h \) is the highest bit in the expression \text{Exclusive}_\text{Or}(\lambda, \lambda')\) the pivot-level is given by \( (n-1-h) \). Equivalently \( h \) is the highest unchanged bit in moving from \( \lambda \) to \( \lambda' \). Thus the pivot-level may be obtained by one XOR and an entry in a \( 2^n \) look up table.

---

Figure 4.8 : Quadtree projection of an octree.
Figure 4.8 illustrates the progress of the algorithm through a simple octree of depth 2.

To clarify the ideas, let us follow the algorithm through the first few steps:

At the root, level = 0, the oct_node is Partial, so the quadtree root is set to Partial (step 5) and four child rays are started (step 6), by recursing the algorithm.

In level 1, the i = 0 ray encounters child 1 of the octree root, which is Partial. The quadtree root child 0 is set partial and recursion proceeds to level 2.

In level 2, the i = 0 ray encounters child 1,1 of the octree root, which is Empty (step 7). Thus the ray is incremented by the size of the current node (E), and New_Node is called. This finds the pivot_level as 1, increments the ray by E and finds the octal code J = 10 of the next node. Using the first entry in the ancestor array, the next node is found as the 0th child of root child 1. Since this exists, it is returned as the current node, and the current level is the recursion level, which is 2.

The ray is still in the Universe (step 2) so the while loop is executed again. The current_oct_node is Ψ_root.child[1].child[0] which is empty. On the second call to New_Node the pivot_level is 0, and the octal code required is 01. Descending from the root (entry 0 in the ancestor array) we find that Ψ_root.child 0 is empty. Thus the current_level is 1, while the recursion level is still zero. Since Ψ_root.child[0] is empty (step 7) is executed again, but now the ray_step (E) carries the ray outside the Universe. The ray terminates, and the quadtree node Ω_root.child[0].child[0] is Empty. Then the algorithm proceeds to quadtree node Ω_root.child[0].child[1].

The remaining stages of the algorithm can be summarised as follows:
quadtree Action steps visits to depth recorded level quadnode at quadnode
Level 1 \(j = 0\) recurse
  Level 2 \(j = 0\) \(E_2,E_2,E_1\) 4 Empty
  Level 2 \(j = 1\) \(E_2,E_2,E_1\) 4 Empty
  Level 2 \(j = 2\) \(E_2,E_2,E_1\) 4 Empty
  Level 2 \(j = 3\) none 1 0
Level 1 \(j = 1\) recurse
  Level 2 \(j = 0\) \(E_2,E_2\) 4 2
  Level 2 \(j = 1\) \(E_2,E_2\) 4 2
  Level 2 \(j = 2\) \(E_2\) 3 1
  Level 2 \(j = 3\) \(E_2,E_2\) 4 2
Level 1 \(j = 2\) \(E_1,E_1\) 2 Empty
Level 1 \(j = 3\) recurse
  Level 2 \(j = 0\) \(E_1\) 2 2
  Level 2 \(j = 1\) \(E_1,E_2\) 3 3
  Level 2 \(j = 2\) \(E_1\) 2 2
  Level 2 \(j = 3\) \(E_1,E_2,E_2\) 3 Empty

The visit figure refers to the number of octree nodes that have to be accessed to generate a given quadtree node. This point is discussed in more detail in appendix A. Another point that is worth noticing is that the quadtree of the image is produced entirely in pre order - i.e. no condensation of quadtree nodes will be required, which is not the case for FTB or BTF.

4.6 CONCLUSIONS

This chapter has described in some detail display algorithms appropriate for medical data. Algorithms can be divided into those that work on objects as if they had a physical reality and use (fairly) conventional graphic methods, and those that attempt to provide an interactive tool for inspecting large quantities of data. There is always a trade off between the preprocessing and segmentation time, and the run time of the algorithm. Also, many techniques achieve speed at the expense of flexibility.

In the surgical simulation application, it is necessary to treat objects as "hard" so that their spatial relationships are unambiguous. Octrees have been used because of the easy handling of multiple objects in spatial relationship (a space planning problem). Thus this chapter has gone into detail on various octree display algorithms. A new method
was described that produces a quadtree image by raytracing. Since it is image driven, it allows some volume rendering aspects to be incorporated. In particular, interpolation along the rays is possible, leading to an improved shading, at very little extra cost. This is the subject of the next chapter.
CHAPTER FIVE

DISPLAY TECHNIQUES 2 : SHADING

5.1 INTRODUCTION

Common to all shading approaches is the need for a lighting model, and inherent in any model is the need to estimate the surface normal \( \mathbf{n}(r) \) of the objects being viewed. In conventional Computer Graphics, the surface normal is well-defined at every point \( r \) on the surface and the problem is to simulate a realistic physical lighting effect. Typically the model is something like that shown in figure 5.1. Here normalised vectors are set up for \( \mathbf{E} \), the direction of view, \( \mathbf{L} \), the direction of the light source, \( \mathbf{H} \), the direction of highlight, and \( \mathbf{D} \), the bisector of \( \mathbf{L} \) and \( \mathbf{E} \). The light source is given a strength \( I_s \), and an ambient lighting \( I_a \) is assumed. Then various models can be applied to determine the shade \( I_{surf} \) at \( r \). Let us write the general form:

\[
I_{surf} = k_A I_A + I_s W(\lambda)(k_D L \cdot \mathbf{n} + k_s |\mathbf{E} \cdot \mathbf{H}|^\alpha)
\]  

(5.1)

where \( k_A \), \( k_D \) and \( k_s \) are weights for the ambient, diffuse and specular coefficients respectively, \( \alpha \) is the specular exponent which determines how sharply peaked the
highlights are, and $W(\lambda)$ is a weighting for the distance from the observer. This general expression is usually called the Phong model [BUI-TOUNG75], although the original formulation was not precisely as in (5.1). Other models can be derived as restrictions on this general model. If for example $k_s = 0$, $W(\lambda) = 1$, then simple cosine shading results. If $k_D = 0$, $\alpha = 0$, and $W(\lambda) = (\lambda_{\text{max}} - \lambda)/(\lambda_{\text{max}} - \lambda_{\text{min}})$, then no surface orientation is incorporated, and the shading is called depth shading.

More complicated models can be developed, for example by applying (5.1) separately to red, green and blue light sources, summing over different light sources at different positions, and incorporating surface texture properties into the specular reflection term. Further details can be found in many textbooks [NEWMAN81, FOLEY82].

5.2 SHADING TECHNIQUES FOR MEDICAL DATA

For medical data, the problem of interest is not really the lighting model, but the techniques for deriving the surface normals. The surface normal of the object is defined as the normal of the tangent plane to the characteristic function $\Phi_c$. But in a discrete representation the form of $\Phi_\text{d}$ is not sufficient to derive this plane. This section will discuss the models for approximating the surface normals at $\partial \Phi$, which are termed pseudo-surface normals.

In a similar way to HSE techniques, shading techniques may be classified as either object-based or image-based. In object based techniques the surface normals are estimated directly from the voxel neighbourhood. Image based techniques attempt to derive a surface normal vector by examining the gradient of the depth image $Z_\text{d}$ defined by (4.6) in section 4.2. The advantages of object based methods for binary representations are that firstly the vectors may be derived from a code that can be stored at the voxel, and secondly, because the domain of vectors is limited, vector and scalar products may be precomputed and stored in look up tables for fast implementation. Additionally, the effective surface normal is invariant with respect to the view direction, which is not the case for image based techniques, so that no shading artifacts are derived during rotation of the object. However a disadvantage is that the local neighbourhood must be known. This is satisfactory for fixed binary objects, but
not for greyscale data, where the object is derived dynamically, nor for cases where the data is modified (as is the case in surgical simulation).

A detailed study of shading in binary voxel databases, using both image-based and object-based methods has been given by Chen et al [CHEN85].

5.2.1 **Object Based Shading**

5.2.1.1 **Object-Based Binary Methods**

In binary representations, the estimation of the surface normal may be likened to the difficulty of measuring the slope of a line described only by binary valued pixels. The pseudo-normal vectors are derived from the local neighbourhood of the voxel to be

---

**SURFACE CONTEXTS:**

<table>
<thead>
<tr>
<th>Case</th>
<th>3</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Case</th>
<th>4</th>
<th>3</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Case</th>
<th>5</th>
<th>4</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 5.2 : The orientations of pseudo-normal vectors in the surface context description.
displayed. Chen et al [CHEN85] showed that for each face being shaded there were 81 combinations of relevant neighbours, producing only 25 separate pseudo-normal vectors - 150 possibilities for all six faces together. Recalling that the MIPG approach relies on knowledge of the 1-adjacency of faces, the result just mentioned may be seen by considering that for each face being displayed there are three orientations of the 1-neighbouring face in each of four directions (Figure 5.2). This implies 81 contexts for each face in the display lists, which could be stored using 7 bits per face. The 25 vectors arise because of redundancy of some of the contexts.

Rather than consider the context of faces, an approach investigated by the author was to record the context of each voxel. For 1-connectivity this requires an 18-bit code for each voxel. The idea was to develop an object-based contextual shading that would be preserved under Boolean operations. The importance of this stems from the central

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{voxel_directions.png}
\caption{Voxel neighbour directions.}
\end{figure}
method of UCL3D - that of simulating cutting and merging operations in 3D by creating 3D volume masks and displaying Boolean expressions of objects and masks. This will be developed further in chapters 6 and 7. The following discussion shows how to implement the contextual based shading for a voxel based model. To extend the description to an octree model, hierarchical neighbourhoods must be considered as described in section 3.3.1.

Consider the 18 1-neighbours of the voxel shown in Figure 5.3. The previous chapter used the labels \( x^+, x^-, y^+, y^-, z^+, z^- \) for the six faces. For ease of notation these 2-neighbours

---

**Voxel Contexts:**

- **Case:** 3  
  **Multiplicity:** 1

- **Case:** 2  
  **Multiplicity:** 1

- **Case:** 1  
  **Multiplicity:** 2

- **Case:** 4  
  **Multiplicity:** 1

- **Case:** 3  
  **Multiplicity:** 1

- **Case:** 2  
  **Multiplicity:** 2

- **Case:** 5  
  **Multiplicity:** 2

- **Case:** 4  
  **Multiplicity:** 2

- **Case:** 3  
  **Multiplicity:** 4

*Figure 5.4: Voxel contexts.*
are now relabelled as, E, W, N, S, U, D respectively (for East, West, North, South, Up, and Down). Then the 12 remaining 1-neighbours are labelled NE, NW, SE, SW, UE, UW, DE, DW, UN, US, DN, DS. Each face will be influenced by the presence or absence of eight of the 18 1-neighbours. To be specific:

<table>
<thead>
<tr>
<th>Face</th>
<th>Relevant Neighbours</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>DE, D, UE, U, SE, S, NE, N</td>
</tr>
<tr>
<td>W</td>
<td>DW, D, UW, U, SW, S, NW, N</td>
</tr>
<tr>
<td>N</td>
<td>NE, E, NW, W, UN, U, DN, D</td>
</tr>
<tr>
<td>S</td>
<td>SE, E, SW, W, US, U, DS, D</td>
</tr>
<tr>
<td>U</td>
<td>UN, N, US, S, UE, E, UW, W</td>
</tr>
<tr>
<td>D</td>
<td>DN, N, DS, S, DE, E, DW, W</td>
</tr>
</tbody>
</table>

Thus for each face we can construct an 18-bit mask, with eight bits set, which when ANDed with the 18-bit neighbour code will give one of 256 18-bit result codes. When all six faces are considered there will be six masks and 1536 possible result codes. These provide a look up for the 150 possible pseudo-normal codes.

Table 5.1: The 16 possible neighbour states for direction East.
For example, consider the East (x') face. The four neighbours DE, D, UE, U give a four-bit look up into one of five vectors in the XZ plane as shown in figure 5.4. The neighbours SE, S, NE, N give a four-bit look up into one of five vectors in the XY plane. The result of all cross-products is 25 vectors for this face. In Table 5.1, the 16 states of the DE, D, UE, U neighbours are given together with the vector case to which they give rise. F stands for a Full neighbour, E for an Empty neighbour, and x for irrelevant. The fact that certain neighbours do not always influence the normal vector gives rise to a multiplicity of certain cases. The sum of these multiplicities is 16.

The exact masking codes for each face will depend on the precise ordering of the bits representing neighbours. In the UCL3D implementation the neighbour ordering is set up as:

```
W E S N D U UW UE US UN DW DE DS DN SW SE NW NE
```

<table>
<thead>
<tr>
<th>Mask</th>
<th>0 0 1 1 1 0 1 0 0 0 1 0 0 1 0 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case</td>
<td>Z Y</td>
</tr>
<tr>
<td>1</td>
<td>0 x 0 1</td>
</tr>
<tr>
<td>2</td>
<td>0 1 0 x 0</td>
</tr>
<tr>
<td>3</td>
<td>0 0 1 x</td>
</tr>
<tr>
<td>4</td>
<td>1 0 x 1</td>
</tr>
<tr>
<td>5</td>
<td>x 0 x 1</td>
</tr>
</tbody>
</table>

Table 5.2: The bit patterns of the 18 1-neighbours for the East face of a voxel.
Table 5.2 shows the complete bit patterns that arise for the East face using this ordering. The remaining five faces can be derived by permutations. Then the neighbour masks are given in octal by:

E 172105      S 631054
W 174212      U 747400
N 630423      D 740360

The problem of looking up the 1536 result codes is a standard search problem. If the result codes are sequentially ordered they may be found by a binary search requiring at worst 11 steps. If stored in random-access then a $2^{18}$ look up table is required. A hashing algorithm could be employed as an intermediate solution. In the UCL3D package the random-access method was used. Figure 5.5 shows the shading obtained from this coding.

Figure 5.5 : Object based shading of the skull derived from the slices of Figure 1.2.
It is now apparent that since the neighbour code is preserved under Boolean operations, so is the validity of the look up. Thus the pseudo-normal to the complement of a voxel is found from ANDing the relevant view mask with the logical NOT of the neighbour code. Similarly the AND, OR, XOR of two or more objects, will give rise to a shade dependant on the result code from the AND, OR, or XOR of the neighbour codes.

One disadvantage of this method is the time to generate the neighbour codes for each voxel, even though they do not need to be recomputed. Another is the large amount of storage per voxel which loses the compression effect of only 1-bit per voxel. A further disadvantage is the limited domain of estimated surface normal vectors available in object-based methods. The number of vector directions chosen depends on the connectivity used in the volume description. For 2-, 1-, and 0-connectivity the number of discrete vectors is respectively 9, 25, and 625. In the MIPG approach objects are 1-connected, so 1-connectivity choice is that used by Chen et al [CHEN85] and produces good images although still exhibiting a degree of "jaggedness". The jagged artifacts depend to a certain extent on the lighting direction, and are discussed in section 5.3.

5.2.1.2 Object-Based Greyscale Methods

In object-based methods applied to greyscale representations, the idea is to derive the surface normal from the local neighbourhood of densities. The problem is then the same as finding the first order approximation to the surface by edge detection, as described in section 2.3.2.2.2. The shading that results depends on the size and type of edge filter chosen. Often the Zucker-Hummel operator [ZUCKER82], is used which corresponds to a 0-connectivity description of the data [POMMERT89a, ROBB89].

Although it is not an edge detector, several authors [e.g. MEAGHER84d] use the discrete approximation of the gradient operator \( \Delta(r) = \frac{-\nabla f(r)}{||\nabla f(r)||} \):
\[ \hat{n}_i = \frac{-[f(i+1,j,k) - f(i,j,k)]}{G} \]
\[ \hat{n}_j = \frac{-[f(i,j+1,k) - f(i,j,k)]}{G} \]
\[ \hat{n}_k = \frac{-[f(i,j,k+1) - f(i,j,k)]}{G} \]

\[ G = \left( \hat{n}_i^2 + \hat{n}_j^2 + \hat{n}_k^2 \right)^{1/2}. \]

Levoy used this as the surface normal in a full specular lighting model with multiple colours for every voxel traced in the volume compositing technique [LEVOY88] (see section 4.4.2.3). Several simpler 1D gradients, along and perpendicular to the ray were compared with Equation (5.3) by Höhne et al [HÖHNE86]. Note that although used in a ray tracing context, the shading is still object-based because it is the object-space coordinate system that is used in Equation (5.3).

Another type of shading to be classed as object-based greyscale is shaded density display [GOLDWASSER86]. Here the actual voxel density is assigned to the pixel, attenuated by distance. Obviously only the nearest voxels designated "displayable" can be visualised, so density windowing is used to "turn off" ranges of density. A variation where the density range was split into several intervals, each one assigned a separate colour, was used by Farrell et al [FARRELL85].

5.2.2 Image-Based Methods

5.2.2.1 Image-Based Binary Methods

Image based techniques are based on the idea that the tangent plane to the object can be estimated from the tangent plane to the depth image \( Z_D \). Then a 2D gradient operator can be passed through the image to give a pseudo-normal vector at each pixel. The idea seems to have been developed independently by Gordon and Reynolds [GORDON85] and Jackel [JACKEL85], although it is usually attributed to the former. The results show an improvement over object-based techniques, at least for binary data [CHEN85]. Unfortunately the orientation of these vectors is not relative to the surface and they change with changing viewpoint, but this is only noticeable in real-time systems or movies of rotating objects.
A number of extensions to the simple (3 by 3) gradient operator have been developed to deal with some of the artifacts of this technique. The most noticeable artifact is due to the inherent "steppiness" of voxel surfaces - a flat surface, at a shallow angle from perpendicular to the view direction, will be represented as a set of steps, all with the same pseudo-normals, with one pixel wide "jumps". The Gordon and Reynolds filter simply averaged across these jumps. Jackel proposed fitting of straight line sections in both the rows and columns of $Z_d$, although the details were left out [JACKEL85]. Bright and Laflin used a similar technique and also one that fitted between the midpoint of each step [BRIGHT86].

Another problem arises because the depth image alone does not distinguish between breaks in the surface and a very steep change in depth due to an actual surface nearly perpendicular to the screen. Gordon and Reynolds solved this problem by making their filter calculate both the forward and backward gradient and weighting against the higher value [GORDON85], with adjustable parameters that controlled what was felt to be the maximum jump between pixels in $Z_d$ that could be attributed to a real surface rather than a break. The method produced good images from selected viewpoints, but depended very much on the adjustment of the weighting factors. The authors suggested that a system be designed to allow the weighting parameters to be adjusted interactively.

The UCL3D implementation for binary representations does not differ significantly from that of Gordon and Reynolds. The extension developed by the author was to grey-scale representations.

5.2.2.2 Image-Based Greyscale Methods

The idea here is to incorporate the partial volume effect to interpolate the depth image $Z_d$. It is most efficient to implement this in a ray-tracing procedure, so that once the surface detection criterion $A = \min \{ \lambda | f(s, \lambda) \geq \rho \}$ is satisfied, the depth is interpolated to the previous voxel:
This depth then gives the same interpolated point as would be obtained for the contour
following algorithm of section 2.3.2.2.1, at least for surfaces orientated less than 45°
from the perpendicular to the image screen. In addition the pseudo surface normal
derived from the Gordon and Reynolds filter applied to this interpolated depth image
is the same as would be obtained if facets were constructed between each contour point
and then averaged at vertex points, as occurs in Gouraud or Phong shading. (Strictly
this is only true in 1D; in 2D it would depend on the tiling scheme between contours).
The name Partial Volume Shading will be used to refer to this technique. It is an
implicit assumption in such techniques that the underlying objects have a "hard"
boundary. Then the density recorded in a voxel that partially intersects this boundary
is assumed to be proportional to the volume of object that partially occupies that voxel.

Figure 5.6: NMR image of the head, displayed with and without partial
volume shading.
An illustration of the shading that results is seen in Figure 5.6 which displays the skin surface of the head, derived from the set of NMR slices shown in figure 1.4. The upper two images show the Partial Volume shading, the lower the ordinary Gordon and Reynolds shading. The improvement is certainly visually appealing, although a proper judgement would need to estimate the correctness in a quantitative way. This is discussed in the following section.

Partial Volume Shading is incorporated into the node-tracing algorithm of section 4.4.3.3 in a straightforward way. When the quadnode ray terminates, the previous octnode $\Psi_{\text{last}}$ is found and the colour of the quadtree node is given by (5.5). $\Psi_{\text{last}}$ is maintained, so that no extra accesses are required. If $\Psi_{\text{last}}$ is a Partial node (this can only happen in a min-max greyscale tree), then the quadtree node has to be split, and (5.5) evaluated separately for the four back children of $\Psi_{\text{last}}$; sometimes this may occur recursively. Figure 5.7 shows how the node-tracing algorithms is adapted for this interpolation procedure.

---

Quadtree ray splits and each child is interpolated backwards.

Figure 5.7: Node tracing for the partial volume shading.
Notice that this type of shading could not be carried out in a BTF or FTB algorithms, since these cannot tell explicitly which voxels belong to the surface.

5.3 COMPARISON OF IMAGE-BASED TECHNIQUES

In this section the correctness of different image-based methods of surface normal estimation is discussed in terms of the quantisation of the resulting vectors. What is examined is the extent to which smooth surfaces produce discontinuous shading jumps, due to the poor sampling of possible vector directions.

Consider a z-buffer image derived by projecting a voxel onto a pixel and recording its distance. In a binary data structure the depth buffer will be discretised to integer values between zero and N where N is the size of the voxel array (assumed cubic). Then the set of vectors obtainable depends on the size of 2D filter applied across the z-buffer, and on the maximum depth change between voxels that is accepted as being part of one surface, rather than the overlap of disjoint surfaces. Let us say the filter is M by M and the maximum distance accepted is B. It is obvious that only a discrete set of vectors can be derived. For example, considering changes in one dimension only, the vectors belong to the set (before normalisation):

\[
\begin{bmatrix}
1 & M & 1 & M & 1 & M & 1 & M & 1 \\
0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\
1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
1 & M & -1 & 1 & M & -1 & 1 & M & -1 \\
1 & M & -1 & 0 & 1 & 0 & 1 & 0 & 1 \\
1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
\end{bmatrix}
\]

By taking the scalar product of a lighting vector with these sets of vectors we can see the degree to which the derived shades are smoothly varying. In Figure 5.8, curve a shows the derived shades for a normal 3 by 3 filter and a maximum depth cut off of 10, as suggested by the original work of Gordon and Reynolds [GORDON85]. The
shade curve is quite poorly sampled when the lighting vector moves towards being perpendicular to the surface normal. In curve b, the extreme case where B is 255 corresponding to no distinction about overlapping surfaces being made in a 256 deep array of voxels. Here the sampling is very smooth near the parallel direction but still highly undersampled as the vector moves perpendicular to the surface. Figure 5.8c shows the result of taking the opposite extreme where a filter 255 by 255 is applied to the image and a single vector estimated with a depth cutoff of 10. This time the undersampling is in the parallel direction. Thus even extending the size of the filter has limited validity in an image based approach, when the image is restricted in resolution to the original voxel size [JACKEL85].

When considering the use of greyscale data to improve the resolution of the depth buffer by using the partial-volume effect, the domain of possible vectors is effectively increased. Figure 5.8d shows the range of shades derived by using only a 3 by 3 filter and a depth cutoff of 5, but using 64 greylevels (compressed from the range of
densities in the original data) to interpolate the depth values at each pixel. Now we find a very smooth sampling of the shading curve at all orientations of the lighting vector.

The principle is illustrated on the binary and greyscale digital spheres introduced in section 3.3.3. Figure 5.9 shows the depth image that results from the non interpolated raytracing method, and figure 5.10 shows that obtained from the interpolated raytracer. Figure 5.11 shows the resulting shaded images. This is a useful test since the real object is known exactly, and the true surface normals should sample continuously all directions in space. The artifacts when interpolation is not present are clearly visible,
and are especially marked when the lighting is oblique from the view direction. By contrast, the images when interpolation is used are very smooth under all lighting conditions.

5.4 CONCLUSIONS

This chapter has discussed the shading problem in medical 3D graphics and described the advantages of using a greyscale random access volumetric data base.

Figure 5.10: Isometric view of the interpolated depth buffer of the digital sphere.
The implementation of an object-based shading scheme was discussed, which has some attractive ideas. However experience with users has shown that image-based methods are preferable due to the more realistic looking images.

The discussion has been to a large extent qualitative. The validity of such methods as Partial Volume interpolation needs to be evaluated quantitatively. If correct, then there is a potential improvement in metric quantification, which would have useful implications for such areas as stereotactic surgery and prosthetic manufacture.

As far as possible, implementation details have been left out since the methods should be applicable to any data structure falling into the classification described. In UCL3D the Partial Volume shading technique was incorporated into the node-tracing algorithm of section 4.4.3.3. Appendix A analyses this algorithm in detail, and some comparisons made there indicate that this shading technique causes an increase in time and space complexity only of the order of 10% over the binary version.

![Figure 5.11](image_url)

Figure 5.11: Shaded displays of the digital sphere using interpolated and non-interpolated image-based shading.
CHAPTER SIX

DISSECTION (SECTIONING)

6.1 INTRODUCTION

This chapter describes the techniques developed to allow arbitrary sectioning or sub-regioning of volume data. The discussion will be in terms of algorithms and examples on binary and greyscale volume data encoded as octrees as described in previous chapters. Further aspects of manipulation will be covered in Chapter 7. The application of the techniques to real cases will be presented in Chapter 8.

Object Dissection means the interactive process required in a surgical simulation where the need is to visualise cutting of the object and thus to dissect it into one or more sub-objects. Some approaches have been used before. These may be classified as object-modification methods, and object-masking methods.

6.1.1 Object Modification

In these methods the data object itself is reconstructed with the required pieces removed. Obviously this requires the overhead of reconstructing the object. Using the display of an object in a given view, a closed contour is drawn on the screen. This contour is projected as a cylindrical volume (CV) into Image-Space to a depth specified by the user. The CV is transformed to Object-Space, and all parts of the object falling within this cylinder are deleted from the set of voxels comprising the object. Effectively a modified scene is produced, where the deleted voxels have their density function set to zero. A new object is then constructed from this modified scene.

The technique was first implemented in the 3D MIPG package, by operating in the Cuberille representation, using either the 2D or 3D boundary methods [BREWSTER84, CHEN84a]. A number of possibilities were tried. In 2D, either the slices, or the
Directed Contour representation of their boundaries were modified. In the former, the 3D boundary tracking algorithm could be applied to the modified slices. The latter case was faster but more limited because rotations out of the plane of the slices could not be accommodated. In 3D, a method using a full binary array was tried, where here the projected cylindrical volume "scooped out" all voxels within it, between minimum and maximum depths from the screen [CHEN84a]. Trivedi suggested using the depth-buffer shading as a visual cue for choosing these depth limits [TRIVEDI86].

The CEMAX approach was very similar, but allowed a more interactive editing session based in a windows environment [WEISBURN86]. In principle the technique could equally well be applied to B-R representations, although it does not appear to have been done.

6.1.2 Object Masking

The alternative is to delay the clipping of the object to the display procedures. These algorithms can only operate on volumetric data bases. Alternatives previously suggested are:

1) Clipping by half spaces.
2) Clipping by a cylindrical volume

6.1.2.1 Clipping by half spaces

Clipping by half spaces is achieved by testing each voxel to be displayed against a half space. This is efficient but grows linearly in the number of half spaces. As usual if the half spaces are orthogonal to the object space axes the technique is simpler. This method was used by several groups [MEAGHER84a, GOLDWASSER84, HÖHNE87a, ROBB88, CLINE88]. A major disadvantage is that only convex volumes may be delineated in this way.
6.1.2.2 Clipping by a cylindrical volume

If the method of 6.1.1 is used to describe a cylindrical volume, it would be possible not to display voxels within that volume, during traversal of a voxel display algorithm. This method was suggested by Freider et al [FREIDER85] although the voxels in the cylindrical volume were then deleted from the input file. However this is still somewhat limited in that a cylindrical volume is a restricted volume, even if concave, since it is (by definition) of identical cross section in any plane perpendicular to its axis.

6.1.3 The Generalised Volume Mask

For the UCL3D system the author implemented an unrestricted volume mask that did not affect the object in question. A Volume of Interest (VOI) of arbitrary complexity may be constructed and a Boolean expression of the mask and object may be viewed. Thus the VOI is like a 3D bitmask - a straightforward extension of 2D Regions of Interest (ROIs see section 3.1.1.1). To achieve efficiency the VOI is encoded as an octree. Thus subsequent display and manipulations algorithms operate on Boolean expressions representing, for example, the object inside or outside the VOI.

The VOI may also be useful in the segmentation stage. In 2D it is quite common to manually define a Region of Interest (ROI) on an image, and use it as a mask, within which a segmentation operation is performed.

The procedures for generating and displaying VOIs are now described

6.2 GENERATION OF ARBITRARY VOLUME MASKS

This section presents a method for constructing an arbitrary 3D volume mask, by interactively drawing 2D ROIs in multiple views and forming the Boolean intersection of their inverse projections. It is simple to construct any concave simply connected object in this way. With an extension it is also possible to include objects with interior holes. The MIPG suggested the use of volume masks, but only implemented 3D primitives like triangular prisms, and ellipsoids [BREWSTER 84, TRIVEDI86]. Also
implemented in UCL3D is a method of generating a VOI by combining a set of 2D ROIs generated in the original 2D slices of the data.

The advantage of a volume mask is that it may be stored in a compact form and subsequent operations such as viewing, translation and rotation can be performed on a Boolean expression of masks and data, rather than creating an intermediate dataset.

In UCL3D, where an octree encoding scheme is used for the dataset and VOIs, a high degree of compression is achievable for the usually simply connected masks. The masks are stored as binary trees and the data as greyscale trees (see section 3.3). In principle other 3D structures, or hybrid structures would be possible. Where relevant the notation of earlier chapters will be used.

Let us use the trivial octrees

U : The octree with a single Full node, representing the Universe
∅ : The octree with a single Empty node, representing empty space

In addition let us define set operations on octrees, in particular:

The Intersection of two trees : A ∩ B
The Union of two trees : A ∪ B
The Complement (Negation) of a tree : ¬A
The Difference of two trees : A ∩ ¬B ( = A - B)

6.2.1 Volume Mask Construction Algorithms

Two distinct forms of VOI generation will be specified. In the Surgical Simulation application, the user most often wishes to view a dataset and interactively demarcate sections to be modified. Thus it is useful to draw the required dissection and observe it in multiple views, editing as required. For other applications it is desirable to build a VOI from the set of ROIs in 2D planes.
6.2.1.1 Generation from multiple projections

The idea of constructing the 3D data set of an object from its projections in multiple views is not new. Chien and Aggarwal [CHIEN86] considered the reconstruction of objects from projections in three orthogonal directions, using a structure called the pseudo-octree. This technique is efficient and simple but leads to many restrictions on the type of object to be constructed. Since the work described in this chapter was carried out, Potmesil [POTMESIL87] gave a method for using an arbitrary number of projections, that is close to that employed here. The principle difference is that instead of having available a number of images and information concerning the viewing conditions used to obtain them, an object is considered as being viewed in successive directions and edited until the required state is achieved.

Potmesil considered the problem of determining the volume description of an object given a set of images obtained by a robot camera free to move around the object. Thus the real object $Q_c$ was viewed in a set of images $\{I_1, I_2, \ldots, I_n\}$, each of which had an associated viewing parameter set $\{VP_1, VP_2, \ldots, VP_n\}$, which included the position, orientation, and geometric characteristics of the camera. Associated with the object was a (discrete) object model $M_D$, described by an octree, that, ideally, represented the discrete object $Q_d$. Using the definitions of section 4.2, each image $I_i$ was associated with a Silhouette $S_{i}$ of the model, under a projection transform $PT_i$. The set of transforms $\{PT_i\}$ was derived from the set $\{VP_i\}$, and $M^*_D$ was an estimate of the model octree consistent with:

$$S_{i}(M_D) = S_{i}(M^*_D)$$  \hspace{1cm} (6.1)

Since the transformations $PT_i$ project 3D object space to 2D image space (i.e. they are one-dimensionally degenerate) the approximations $M^*_D$ are not unique. Instead a whole subspace of the form $M^*_D = M_D \cup \lambda G$ all satisfy equation (6.1), where $G$ is the set of all rays emanating from a point in the silhouette. In Potmesil’s method the silhouettes $\{S_{i}\}$ were obtained from the images $\{I_i\}$ by thresholding and binarisation.

The method of constructing the model $M_D$ was to construct, for each projection $P_i$ a Conic Volume $CV_i$ that was its inverse projection

$$CV_i = PT_i^{-1}(S_i)$$  \hspace{1cm} (6.2)
where \( CV_i \subseteq M^*_d \) (the equality is achieved if the image field of view totally contains the object). Recall that the silhouettes are 2D characteristic functions as defined in Equation (2.12) of section 2.1.4 and Equation 4.5 of section 4.2, and the \( CV \) is the set of all points \( r = s + \lambda \hat{g} \) (Equation 4.4) such that \( s \in S_h, r \in U_c \). Then the object was estimated by the intersection set of the inverse projections:

\[
M_d = CV_1 \cap CV_2 \cap ... \cap CV_n \quad (6.3)
\]

The algorithm also took into account the limited field of view for each image to allow flexible positioning of the camera. This involved a more complex relation than Equation (6.3), which is not required in the subsequent development in this section. This was due to the partial intersection of Image space \( IS_c \) with the Universe \( U_c \) as explained in section 4.2.

The method of generating VOIs described here may be considered a subclass of this general problem. The process starts with the object octree \( Q_d \) and displays it in a number of views to obtain a set of displayed images \( \{D_y\} \), where

\[
D_{yi} = \text{Display}(Q_d) \quad (6.4)
\]

For each view a number of arbitrarily complex regions are drawn on these images. Let us denote the set of these regions \( \{R_i\} \) where each \( R_i \) may have separately connected areas. Then the inverse projections are formed:

\[
CV_i = PT_i^{-1} (R_i) \quad (6.5)
\]

and the volume of interest defined as

\[
\text{VOI} = CV_1 \cap CV_2 \cap ... \cap CV_n \quad (6.6)
\]

The primary simplifications are:

1) Since the image space \( IS_c \) is defined as a superset of the object space universe \( U_c \), then equation (6.6) is strictly correct. The graphically generated displays \( \{D_{yi}\} \) correspond to the digitised images \( \{I_i\} \) obtained photographically.

2) Orthographic projections are used throughout so that the conic volumes \( \{CV_i\} \) are actually cylindrical volumes.
3) The display set \( \{D_y_i\} \) is obtained from the octree model and the regions \( \{R_i\} \) are interpreted as characteristic functions. They correspond to the projection silhouettes \( \{S_{h_j}\} \), since the regions are drawn using the set \( \{D_y_i\} \) as visual cues.

4) The transformations \( \{P_{T_i}\} \) are known exactly from the selected viewing conditions for each display.

However the author extended the algorithm in the following way:

Since the desired result is a sectioning of the original object \( Q_o \), the specification of the region set \( \{R_i\} \) is determined by the form of the final results:

\[
Q_o \cap \text{VOI} \quad \text{and} \quad Q_o - \text{VOI} \quad \text{(6.8)}
\]

Thus an iterative method of forming the VOI is allowed by considering at any stage the partial intersection set:

\[
(\text{VOI})_k = CV_1 \cap CV_2 \cap ... \cap CV_k \quad \text{(6.9)}
\]

and creating each subsequent display as one of

\[
\begin{align*}
D_{y_{k+1}} &= \text{Display} (Q_o) \text{ under transform } P_{T_{k+1}} \quad \text{(6.10a)} \\
D_{y_{k+1}} &= \text{Display}_\text{Boolean} (Q_o \cap \text{VOI}_k) \text{ under transform } P_{T_{k+1}} \quad \text{(6.10b)} \\
D_{y_{k+1}} &= \text{Display}_\text{Boolean} (Q_o - \text{VOI}_k) \text{ under transform } P_{T_{k+1}} \quad \text{(6.10c)}
\end{align*}
\]

Thus some flexibility is allowed in creating the VOI. The two extremes in approach would be

1) Generate all the displays \( \{D_y_i\} \) from equation (6.10a) and form the VOI from equations (6.5) and (6.6).

2) Generate each display iteratively by using equations (6.10), (6.5) and (6.9).

Note that, obviously, the user is allowed to use a subset of the image set \( \{D_y_i\} \) if desired. The basic philosophy is to use as many images obtained by equation (6.10) as are useful to provide the visual cues for creating the VOI.
6.2.1.2 Generation of Cylindrical Volumes

As also pointed out by Potmesil, the actual octree volumes $C_{V_j}$ are intermediate and of no interest. Instead the VOI is built by editing the volume described by equation (6.9) at each stage. Two procedures that are closely related are required: Create_Octree to build a tree at a spatial position, and Edit_Octree to make transformations to an existing tree. Both procedures return a pointer to an octree node. Formally, $VOI_0$, the zeroth iteration, is considered as a single node representing the whole Universe ($U_o$): $VOI_0 = U$. At the moment the representation of the region $R_i$ is left unspecified.

The algorithms may be implemented in two ways, in analogy with the two classes of display algorithm discussed in section 4.3

1) Inverse project the regions into the object space (Equation 6.5) and test intersection with existing nodes.
2) Forward project octree nodes into image space (Equation 6.10a) and test intersection with the region $R_i$.

The second approach is the one used in the current implementation of UCL3D, in common with other authors [POTMESIL87, CHEN88]. The argument given for this choice is that the complexity of the intersection test is two dimensional in the second case, and three dimensional in the first case. However, note that the overall complexity is the same as for an image display algorithm. In the second case it is the same as the standard FTB algorithm, while in the first case it is the same as a raytracing algorithm. Section 4.4.3.3 presented a raytracing algorithm that operated on quadtree nodes, appropriate for rendering an octree in orthogonal projections. It might be possible to adapt this algorithm to the octree creation procedure, which reduces the complexity of the type 1) approach to two dimensions. However, for the rest of this chapter the type 2) approach will be discussed.
Chapter 4 introduced the Convex Polygon which is the projection of an octree node under \( PT : CP = PT(\Psi) \). The algorithms are described as follows:

**Create_Octree** - create node at position \( r \) and level \( m \), for region \( R \) and forward projection transform \( PT \).

1) Find the projection \( CP \) of the node at \( r \) and level \( m \), under \( PT \)
2) Test the intersection of \( CP \) with the region \( R \)
3) If \( CP \) is inside \( R \) then create a Full node centred at \( r \), level \( m \).
4) Else if \( CP \) is outside \( R \) then create an Empty node centred at \( r \), level \( m \).
5) Else (provided we are not at the top level of the octree)
5.1) Create a Partial node centred at \( r \) and level \( m \)
5.2) For each of the children \( i = 0 \) to \( 8 \):
    Find the position \( r' \) of child \( i \)
    Create_Octree for position \( r' \), level \( m+1 \)
6) Else (the resolution of the octree has been reached) decide whether the intersection test is counted as inside or outside and perform 3) or 4).

**Edit_Octree** - Validate the existing node at position \( r \) and level \( m \), for region \( R \) and forward projection transform \( PT \). Modify the node if required.

1) If octree node is Empty then return with no further action
2) Find projection \( CP \) of the node at \( r \) and level \( m \), under \( PT \)
3) Test the intersection of \( CP \) with the region \( R \)
4) If \( CP \) is inside \( R \) then change node status to Full.
5) Else if \( CP \) is outside \( R \) then change node status to Empty.
6) Else (provided we are not at the top level of the octree)
6.1) If the node was previously Full then
    Change its status to Partial
    For each of the children \( i = 0 \) to \( 8 \):
    Find the position \( r' \) of child \( i \)
    Create_Octree for position \( r' \), level \( m+1 \)
6.2) Else (the node was previously Partial)
    For each of the children \( i = 0 \) to \( 8 \):
    Find the position \( r' \) of child \( i \)
    Edit_Octree for child \( i \), position \( r' \), level \( m+1 \)
7) Else (the resolution of the octree has been reached) decide whether the intersection test is counted as inside or outside and perform 4) or 5).

These algorithms require the test of a quadtree node against the projected octree node (CP). This is very closely related to the FTB display algorithm presented in section
4.4.3.2. In that case, if the CP was completely covered by Full quadtree nodes, then it was not displayed, if it was completely covered by Empty nodes, these nodes were recursively split and filled. In this case, if the CP is completely covered by Full nodes, the octree node is set Full; if completely covered by Empty nodes it is set Empty. The quadtree of the image is not modified.

6.2.1.3 Generation from parallel Region Masks

The method of generating the volume mask from the individual slices is quite straightforward and is included here for completeness. The method is summarised as follows:

1) For each slice
2) Choose a threshold $\rho$, and apply the contour following algorithm of section 2.3.2.2.1.
3) If the contour is satisfactory, next slice at 1)
4) Else manually edit the contour, or choose another $\rho$ at 2).
5) Build the (binary) volume representation, interpolating between slices, and setting all pixels in each slice to zero if they are outside the ROI defined by the contour for that slice.

In the interpolation step, nearest neighbour interpolation is used, to avoid interpolating between a voxel outside an ROI in one slice, and inside on the next slice. An alternative would be to use the shape-based method of [RAYA88], described in section 2.2.3.

6.2.2 Representation of Volume Masks

Chapter 3 presented various alternative octree representations: Binary, Masked Binary, Greyscale, and Min-Max Greyscale. The current implementation of UCL3D considers the VOI as a binary structure as mentioned in section 6.2. In principle it might be interesting to assign grey values to voxels on the surfaces of the VOI, based on the partial volume of their occupancy. It might be a possible development for the problem of generating octree models of real objects. Chien and Aggarwal [CHIEN86]
incorporate an explicit coding for surface voxels, and Potmesil [POTMESIL87] suggests using the pixel values in the set of images \( \{I_i\} \) to determine shape-from-shading factors, and incorporate them into the octree nodes. However this would not seem to be advantageous to the dissection problem.

### 6.3 OPERATIONS ON MULTIPLE OBJECTS

The method for creating an arbitrary volume mask has now been developed. The purpose of this mask is to allow operations on the original data, inside or outside the volume represented by this mask. Thus algorithms are required that operate on Boolean expressions of octrees. Examples are:

- **Boolean** \((\cap, \cup, \neg)\)
- **Display\_Boolean** \((\cap, \cup, \neg)\)
- **Translate\_Boolean** \((\cap, \cup, \neg)\)
- **Rotate\_Boolean** \((\cap, \cup, \neg)\)
- **Segment\_Boolean** \((\cap, \cup, \neg)\)

These are algorithms operating on pairs of trees, related by one of the Boolean operators Intersect, Union, Difference. Usually one of the operand trees is a binary VOI, and the other either a binary or grey data tree. In UCL3D these algorithms have been coded explicitly. Not all possibilities were implemented, only those required for Surgical simulation. In addition, one algorithm was developed for three operands: **Boolean**\((C \cup (A - B))\).

The form of these algorithms will be the subject of the next chapter. They are mentioned here to explain their use in this chapter. The **Display\_Boolean** algorithms were already used in the formation of the VOI masks themselves (equation 6.10). In the next section a comparison will be made between the **Display\_Boolean** and **Boolean** operations.
6.4 RESULTS

6.4.1 Example Construction of an arbitrary object

In order to demonstrate the features of the process an artificial object was created. A full analysis of the complexities of the objects will be deferred to Chapter 7, where the algorithms are discussed in more detail. Here only a summary of the steps taken is given. The timings are for the NORSK implementation, described in Chapter 1.

1) The sphere was displayed in one view:
   Display time :  25 s

2) A single concave ROI is drawn (R1), and projected to create \( VOI_1 = CV_1 \):
   create_VOI time :  130 s

3) The expression sphere \( \cap VOI_1 \) is displayed in a different view:
   Display_Boolean time :  45 s

4) Three convex ROIs are drawn (R2-R4) and projected back to edit \( VOI_2 = VOI_1 \cap CV_2 \):
   Edit_VOI time :  97 s

5) The expression sphere \( \cap VOI_2 \) is displayed in a different view:
   Display_Boolean time :  87 s

6) The expression sphere - \( VOI_2 \) is displayed in the same view as 5):
   Display_Boolean time :  98 s

7) The object sphere \( \cap VOI_2 \) is formed explicitly
   Boolean time :  35 s

8) The object sphere - \( VOI_2 \) is formed explicitly
   Boolean time :  50 s

9) The object sphere \( \cap VOI_2 \) is displayed from a different view
   Display time :  30 s

10) The object sphere - \( VOI_2 \) is displayed from the same view as 9)
    Display time :  58 s

The time for display varies with the viewing direction and the complexity of the objects. For the orthogonal views the algorithm of section 4.4.3.3 was used, whereas
Figure 6.1: Regions R1 to R4 drawn on the shaded display of a sphere in multiple views.

for the general views the BTF algorithm was used. The sphere used is the binary version introduced in Chapter 3, where its complexity was discussed. The complexity of the \( \{CV\} \) set and the VOI are discussed in the next chapter. The \( \{CV\} \) set and resultant VOI were not neighbour coded, so that the algorithms are essentially operating on Unmasked structures, apart from step 1).

Figure 6.1 shows the screen image at stages in this editing process. The left hand image shows the sphere in anterior view with R1 drawn on it (steps 1 and 2). The right hand image shows the left view of the expression sphere \( \cap \text{VOI}_j \), with R2-R4 drawn on it (steps 3 and 4). In figure 6.2, the cylindrical volumes of each region are shown explicitly, with their intersection in figure 6.2c, and difference in 6.2d. Normally the intermediate octrees \( \{CV_i\} \) are not generated, but here they have been used to evaluate the improvements to be gained using the algorithms of Chapter 7. Figure 6.3 shows two general views of the objects formed from the intersection and difference of the sphere with the complete volume mask (steps 7 to 10).
Figure 6.2: The cylindrical volumes generated from the regions of figure 6.1.

Figure 6.3: The resulting objects from intersection and difference of the sphere and the volume mask delineated in figure 6.1.
It was originally intended to represent binary objects in their Masked Binary
collection because of the faster display algorithms this affords. However this
eventually proved inefficient for two reasons. Firstly, the implementation of Boolean
operations other than Intersection and Union becomes very complex. Secondly, the time
to encode the neighbours is far too lengthy in the VOI creation process. The next
chapter gives some further discussion of this point, with example results.

6.4.2 Example Volume Mask from Region Masks

As an example of the use of the 2D method of generating a VOI, the brain was
segmented from the NMR slices of figure 1.4 and a volume representation was built.
This itself, is a binary octree representing the brain. When applied to the greyscale
octree built from all the slice data, it acts as a mask that can identify the grey data of
the brain within the complete volume. Figure 6.4 shows the combined display of three
objects. In dark blue is the result of a low threshold on the skin surface of the
complete data. In light blue is the result of a higher threshold on the expression
Head \cap VOI. In white is the result of a very high threshold on the same expression.
This last represents the Multiple Sclerosis lesions within the brain [ARRIDGE89].

6.5 DISCUSSION AND CONCLUSIONS

Here the simulated "cutting" of an object by masking inside or outside a Volume
of Interest (VOI) has been discussed. The advantage is that the original data is left
unaffected, whilst VOIs are represented compactly in an octree format.

Early 3D systems worked on binary representations. They allowed several objects to
be built from a given data set, using different segmentation procedures. Indeed, volume
modification techniques often built a new object from the modifications with respect
to the original data. A binary volume mask created in the way described in this chapter
may be used on several objects derived from the same data set. This was the original
design concept of UCL3D. For example, a binary octree representing hard tissue was
created, and one representing soft tissues. The dissection stage was carried out on visual
Figure 6.4: NMR data with three different thresholds, and a volume mask applied to visualise the brain and MS lesions.

displays of the hard tissue data, and the resultant mask used to also dissect the soft tissue structure. The technique was applicable on either Masked or Unmasked binary structures, although the latter was lengthy and not completely general.

The author subsequently extended the system to include greyscale representations. Then the visualisation of soft tissue ranges, masked by a VOI created using hard tissue displays, is achieved simply by using a different greyscale range in the Display_Boolean process. It is not advantageous to use a masked representation for the binary VOI. Obviously the use of greyscale structures saves the expense of a whole octree generation process.

The principle innovation is to employ algorithms that operate on a Boolean expression of octrees. The details of these algorithms will be included in Chapter 7, and the application in Surgical Simulation will be shown in Chapter 8.
CHAPTER SEVEN

MANIPULATION

7.1 INTRODUCTION

The application of 3D medical graphics to craniofacial surgical planning was one of the earliest ones, and its importance is well established [VANNIER84a, HEMMY83]. It was quickly realised that this application needed a degree of manipulation similar to CAD/CAM design packages. Indeed one of the first systems [VANNIER83], was actually developed in conjunction with a commercial CAD/CAM company (McDonnell Douglas). This was, however, limited to simple operations on 1D primitives such as mirroring to correct unilateral facial asymmetries.

This chapter will deal with the techniques for implementing more sophisticated techniques. So far this thesis has described how to display an object, how to demarcate an arbitrary subregion (a VOI) of an object and how to segment a new object within such a VOI. Now it is necessary to independently transform the "dissected" objects and merge them with each other. The former utilises standard octree transform algorithms (since an octree representation is that under consideration). The latter is implemented by Boolean operations.

Here transform means the process of coordinate transform. When considering the display of a data structure, the process of transforming the structure can be considered formally as either:

1) Transformation of object space relative to a fixed image space.

2) Transformation of image space relative to a fixed object space.

It is usually quickest to use the latter (because, for instance, only the visible data elements need be transformed), although the Hamburg group, for example, uses an object-space transform to ensure the viewing direction is always orthogonal to the object-space coordinate axes [TIEDE87].
When the scene to view consists of multiple objects, which can be independently transformed, a means of performing the first method is required. This is essential in the type of application which is the goal of UCL3D. The previous chapter discussed the display of Boolean expressions of objects. This is developed further here, and in a similar spirit, transforms are developed for Boolean expressions of objects.

7.2 **EXPLICIT BOOLEAN OPERATIONS**

Explicit Boolean operations are required actually to create merged and split objects. The implementation of Boolean algorithms amounts to the evaluation of a simple rule set at each node. The rules for the three simple Boolean operations are given in Table 7.1 where P/E is written because the intersection or difference of two Partial nodes might give an Empty node, (but not a Full node), and P/F indicates that the Union of two Partial nodes might give a Full node (but not an Empty one). These rules are illustrated in Figure 7.1.

Consider the Intersection of two nodes. In the intersection of two octrees in Binary representation, (see the review by Samet and Webber for example) [SAMET88a, SAMET88b], two trees are traversed until one or both is a leaf node. If one is Full and the other Partial, the subtree rooted at the partial node is copied to the result tree. Because two Partial nodes may generate an Empty node, some **condensation** of octree nodes is required in the intersection algorithm.

Implementation of Boolean algorithms in the Masked Binary representation is complex apart from Intersection and Union, and this representation was eventually abandoned. Implementations in other octree representations is straightforward, however. When **Intersecting** or **Unioning** two grey octrees, one is given priority, for the case where two nodes are Full, so that the colour of the resultant is the colour of the higher priority node. Also, some meaning must be attached to the colour of the complement of an Empty grey node, although this operation has little practical use in the medical field. For the **Difference** operation, the tree that is not being complemented is given priority (i.e A is higher priority in the expression A - B). Intersection is useful for

---

1Strictly, the operation of Union described here is therefore A ∪ (B - A)
finding interference volumes if bone pieces are moved into each other.

Since the application of these algorithms is concerned with masking an object inside or outside a VOI, the algorithms have to be able to operate on a grey tree and a binary
tree. The binary tree is formally treated as a grey tree with a uniform colour (usually arbitrary), and given lower priority.

For min-max greyscale octrees, the Partial nodes have to be reset by a post-order traversal, after the Boolean operation.

It is possible to construct operations for any list of objects. One explicit algorithm employed in UCL3D is to merge three objects using the relation $A \cup (B - C)$. The algorithm to do this is as follows, for the ordinary Binary representation:

```
Union_Diff($\Psi_A, \Psi_B, \Psi_C$)
1) If $\Psi_A.status = $ Empty and ($\Psi_B.status = $ Empty or $\Psi_C.status = $ Full) then
   allocate an Empty node $\Psi$
2) Else if $\Psi_A.status = $ Full or ($\Psi_B.status = $ Full and $\Psi_C.status = $ Empty) then
   allocate a Full node $\Psi$
3) Else
   One or more nodes are Partial
   3.1) If $\Psi_A.status = $ Empty
       Problem reduces to Difference
       $\Psi = \text{Difference}(\Psi_B, \Psi_C)$
   3.2) Else
       A must be Partial
       3.2.1) If ($\Psi_B.status = $ Empty or $\Psi_C.status = $ Full)
               copy the subtree rooted at $\Psi_A$ to the output tree
       3.2.2) Else if ($\Psi_C.status = $ Empty)
               $\Psi = \text{Union}(\Psi_A, \Psi_B)$
       3.2.3) Else if ($\Psi_B.status = $ Full)
               $\Psi = \text{Union}(\Psi_A, \bar{\Psi}_C)$
4) Else
   all nodes are Partial
   4.1) recursively pass the children of $\Psi_A, \Psi_B, \Psi_C$ to
       Union_Diff
   4.2) Condense($\Psi$)
5) return $\Psi$
```

Extension of this algorithm to other octree representations requires similar considerations to the earlier cases. An example is shown in Figure 7.2, from four different view points. Here B was the digital sphere, C the VOI used in Chapter 6, and A the resultant of an operation Translate_Boolean(B $\cap$ C) (to be discussed in section 7.4 below). The effect is to remove a complex piece from inside the sphere, translate it and merge with the part of the sphere that remains, in only two operations: Translate_Intersection(Sphere, VOI) and Union_Diff(Translated_Intersection, Sphere, VOI). Broken down into steps, the process would require four operations: 1) Intersect(Sphere, VOI); 2) Translate(Sphere_Intersection); 3) Difference(Sphere, VOI);
Figure 7.2: Result of merging a sphere with a demarcated piece after translation.

4) Union(Sphere_Difference, Translated_Intersection). The comparative times are given in Table 7.2 for the Norsk implementation. The Boolean operations are clearly more efficient in combined form. Unfortunately the Translate_Boolean operation is much worse than its component operations. An improved algorithm was developed in C in the SUN implementation, although it is still not optimal. A discussion of this point is given in section 7.4.4.

7.3 DISPLAY OF BOOLEAN EXPRESSIONS OF OBJECTS AND MASKS

7.3.1 Algorithms

The implementation of equation (6.10) in the previous chapter requires algorithms to display the Boolean expressions: A, ¬A, A ∩ B, and A - B, where A is the object octree and B the VOI octree and the Difference operation A - B is written for the Boolean expression A ∩ ¬B. The most general method would be to parse a Boolean
<table>
<thead>
<tr>
<th>Operation</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time for A = Intersection(Sphere, VOI)</td>
<td>35s</td>
</tr>
<tr>
<td>Time for B = Difference(Sphere - VOI)</td>
<td>50s</td>
</tr>
<tr>
<td>Time for C = Translate(A)</td>
<td>280s</td>
</tr>
<tr>
<td>Time for Union(C, B)</td>
<td>51s</td>
</tr>
<tr>
<td>Time for Translate_Intersection(Sphere, VOI)</td>
<td>690s</td>
</tr>
<tr>
<td>Time for Union_Diff(C, Sphere, VOI)</td>
<td>63s</td>
</tr>
</tbody>
</table>

Times for the different manipulation operations operating on a VOI and a sphere. The direct method (Translate_Intersection followed by Union_Diff) is to be compared with the times for the three other Boolean operations and the translate.

Table 7.2: Comparison of primitive and combined manipulation operations.

expression of octrees at each recursion level. However, here it is simplest to construct these algorithms explicitly. Assume that Display_Octree is a standard octree display algorithm, such as described in Chapter 4. Then for the ordinary binary octree representation the algorithms are:

**Display_Complement**

--- Display the complement of the tree rooted at Ψ.

1) If Ψ.status = Full then
   return
2) else if Ψ.status = Empty then
   display_node(Ψ)
3) else
   -- The node is Partial
   Recursively pass the children of Ψ to Display_Complement

**Display_Intersection**

--- Display the intersection of two trees rooted at Ψ_A and Ψ_B.

1) If either node is Empty then
   return.
2) If both nodes are Full
   display_node
   -- Both nodes occupy the same position
3) Else if one node is Full (Ψ_A say), and the other is
   Partial
   recursively pass the children of Ψ_B to Display_Octree
4) Else --- both nodes are Partial
    recursively pass the children of $\Psi_A$ and $\Psi_B$
    to $\text{Display} \_\text{intersection}$

$\text{Display} \_\text{Difference}$ --- Display the Difference of two trees rooted at $\Psi_A$
and $\Psi_B$.

1) If $\Psi_A$\_status = Empty or $\Psi_B$\_status = Full then
   return
2) else if $\Psi_A$\_status = Full and $\Psi_B$\_status = Empty then
   display_node --- Both nodes occupy the same position
3) else
   3.1) if $\Psi_A$ is Full then
      recursively pass the children of B
      to $\text{Display} \_\text{Complement}$
   3.2) else if $\Psi_B$ is Empty then
      recursively pass the children of A
      to $\text{Display} \_\text{Octree}$
   3.3) else --- both nodes are Partial
      recursively pass the children of A and B
      to $\text{Display} \_\text{Difference}$

$\text{Display} \_\text{Union}$ --- Display the Union of two trees rooted at $\Psi_A$
and $\Psi_B$

1) If both nodes are Empty then
   return
2) Else if either node is Full then
   display_node --- Both nodes occupy the same position
3) Else if one node is Empty ($\Psi_A$ say), and the other is Partial then :
   recursively pass the children of A
   to $\text{Display} \_\text{Octree}$
4) Else --- both nodes are Partial
   recursively pass the children of $\Psi_A$ and $\Psi_B$
   to $\text{Display} \_\text{Union}$

More complex expressions could be derived in similar ways to the direct Boolean operations.
7.3.2 Results

Table 7.4 compares the times for the operations to create an explicit octree $C = A \cap B$ or $C = A - B$, followed by its display, against the time to directly display the Boolean expression. The trees A and B are actually the cylindrical volumes demarcated in figure 6.2a and 6.2b. The number of nodes in these trees is expressed in Table 7.3. The display algorithm here is the BTF, and the timings are for the Norsk implementation.

The increased time for the co-display algorithm is much less than the time for the explicit formation of the Boolean expression.

<table>
<thead>
<tr>
<th>Nodes in A</th>
<th>Level 1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P$</td>
<td>8</td>
<td>48</td>
<td>216</td>
<td>928</td>
<td>3,616</td>
<td>12,096</td>
<td>27,520</td>
<td>-</td>
</tr>
<tr>
<td>$F$</td>
<td>24</td>
<td>480</td>
<td>1,472</td>
<td>7,680</td>
<td>35,968</td>
<td>106,496</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$E$</td>
<td>- 16</td>
<td>144</td>
<td>320</td>
<td>2,366</td>
<td>9,152</td>
<td>33,280</td>
<td>113,664</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Nodes in B</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P$</td>
</tr>
<tr>
<td>$F$</td>
</tr>
<tr>
<td>$E$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Nodes in Intersection(A, B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P$</td>
</tr>
<tr>
<td>$F$</td>
</tr>
<tr>
<td>$E$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Nodes in Difference(A, B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P$</td>
</tr>
<tr>
<td>$F$</td>
</tr>
<tr>
<td>$E$</td>
</tr>
</tbody>
</table>

Table 7.3: Space complexities of the four octrees A, B, A $\cap$ B, A - B.
Create is the time (in seconds) to perform the Boolean operation (Intersect or Difference). Display is the display time of the trees A,B and the resultant of the preceding Boolean. Disp_Boolean is the display time of the algorithm that co-traverses both trees.

Table 7.4 : Comparative times for operations on (Unmasked) octrees A,B.

<table>
<thead>
<tr>
<th></th>
<th>Create</th>
<th>Display</th>
<th>Disp_Boolean</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>-</td>
<td>110</td>
<td>-</td>
</tr>
<tr>
<td>B</td>
<td>-</td>
<td>150</td>
<td>-</td>
</tr>
<tr>
<td>Int(A,B)</td>
<td>58</td>
<td>106</td>
<td>106</td>
</tr>
<tr>
<td>Dif(A,B)</td>
<td>65</td>
<td>103</td>
<td>121</td>
</tr>
</tbody>
</table>

By contrast table 7.5 shows the times for displaying the intersection in the Masked Binary representation. The linking process results in a reduction of the number of Partial neighbours, thus allowing increased efficiency in the evaluation of invisible nodes. A complete investigation would actually inspect the nature of the neighbourhood codes. This is a possible future investigation, but perhaps not very interesting, in view of the fact that most 3D medical imaging packages prefer to work on greyscale data.

Table 7.5 : Comparative times for operations on Masked octrees A,B.

<table>
<thead>
<tr>
<th></th>
<th>Create</th>
<th>Display(1)</th>
<th>Link</th>
<th>Display(2)</th>
<th>Disp_Boolean</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>-</td>
<td>38</td>
<td>543</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>B</td>
<td>-</td>
<td>68</td>
<td>698</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Int</td>
<td>58</td>
<td>52</td>
<td>308</td>
<td>50</td>
<td>52</td>
</tr>
</tbody>
</table>

Create is the time (in seconds) to perform the Boolean operation (Intersect or Difference). Display(1) is the display time of the trees A,B and the resultant of the preceding Boolean. Link is the time to encode the neighbours. Display(2) is the display time of the resultant trees after linking. Disp_Boolean is the display time of the algorithm that co-traverses both trees.
7.4 **BILINEAR TRANSFORMATIONS**

First consider how a bilinear coordinate transformation is implemented in the various representations described in Chapter 3.

7.4.1 **Surface descriptions**

B-R descriptions, both in 2D and 3D, contain explicit coordinate information about points (vertices); edges and faces are described relative to these (e.g. in a directed graph). Therefore for a general homogeneous coordinate transformation such as Equation (4.2), one only needs to replace each point by its transformed point. This operation is therefore linear in the number of points in the representation. The voxel boundary representations used by MIPG, are also lists of coordinates for each face. A translation is therefore achieved by translating the coordinates of a face. This was incorporated into the CEMAX system, which was based on Directed Contours [WEISBURN86]. A general rotation however requires *resampling* of the original volume data, and derivation of new boundaries orthogonal to the new coordinate axes.

7.4.2 **Volume Descriptions**

Spatial enumerative techniques require resampling for all transformations. This is because coordinate information is not stored in the data representation. It is implicit in the order of data representation. A binary digital array is transformed with reference to the digitisation implicit in its creation (section 2.1.2). For example if D₁ was used (equation 2.6), then voxels that partially intersected new voxels would be discarded (the volume of the object would tend to decrease). If D₂ was used then new voxels would be created by partial intersections (the volume of the object would increase). For this reason Weng and Ahuja [WENG87] suggested D₃, as a volume preserving digitisation. The transform of a greyscale array strictly requires a low-pass filter in Fourier space, which, as is well known, is computationally inefficient [PARKER83]. Often nearest neighbour, or better, trilinear interpolation is used [TIEDE87]. For computational simplicity, only nearest-neighbour resampling has so far been implemented in UCL3D.
Because the representation is with respect to a finite Universe, parts of the data that transform to outside the new Universe are discarded. Similarly, parts of the new Universe that were previously outside the Universe are considered empty.

In the marginal-indexing representation, translation is quite simple to achieve by reallocating the pointers in the slices and rows, followed by a linear resampling in the columns. Slices or rows that translate outside the Universe are reset to null pointers. Rotation around the axis of the columns is achieved by a 2D rotation of the slice and row pointers. A general rotation will require resampling. The segment-indexed (Reynolds) representation can be transformed similarly. In this case, for the column translation, the start and endpoint values are incremented or decremented by the translation step in that direction.

Runlength encoded representations, because they do not store the slice or row locations, have to be recoded from the transformed digital array.

### 7.4.3 Octree Descriptions

Chapter 4 showed that the hierarchical nature of octree representations allows simple recursive procedures to implement an FTB algorithm. The same principle applies to translation and rotations. Suppose an object represented by an octree $A$ is transformed by a depth-preserving transform $T$, to an object $B$

$$B = T(A) \quad (7.1)$$

$A$ is called the *source* tree, and $B$ the *target*. The octree $A$ has a root node $\text{Root}_A$ which contains the whole Universe $U_A$ (discrete or continuous). The transformed object has a root node $\text{Root}_B$ which contains the whole transformed Universe $U_B$. The general method for forming $B$ is:
1) Commencing with $\Psi_B = \text{Root}_B$, level = 0, 
2) Find the image $\Psi'_B$ of $\Psi_B$ under $T^{-1}$, the inverse of $T$ 
3) Examine the set of nodes $S=\{\Psi'_A\}$ in $A$, at the same level that partially intersect $\Psi'_B$ 
4) If all nodes in $S$ are full then $\Psi'_B$ is full, 
5) Else if all nodes in $S$ are Empty then $\Psi'_B$ is empty 
6) Else recursively test the children of $\Psi'_B$ against members of the set $S'$, comprised of the children of the set $S$. Each child of $\Psi'_B$ is tested against a set $S' \subset S'$, where the cardinality of $S' \leq$ cardinality of $S$.

The key idea is similar to theorems 4.1 and 4.2 of Chapter 4. Specifically, consider first translation only:

**THEOREM 7.1** A translated node $\Psi'_B$ at level $m$ will be completely contained within a set of eight nodes $S=\{\Psi'_A\}$ in $A$ also at the level $m$, that are arranged in a $2 \times 2 \times 2$ array, (i.e. they are of the same size as a node of level $m-1$).

**THEOREM 7.2** The translation of any child $\Psi'_A\.child[i]$, at level $m+1$, will be completely contained within a set of eight nodes $S'=\{\Psi'_A^*\}$, where $S' \subset S'$, the set of at most 64 nodes formed from the children of the partial nodes in $S$.

The proof of these theorems, by simple geometric arguments is given by [MEAGHER82], and by Samet and Webber [SAMET88a]. Notice that the set $S'$ only contains children of Partial nodes in $S$; no virtual children are required. The translation algorithm that follows was given by Jackins and Tanimoto [JACKINS80], and also by Meagher [MEAGHER82]. Samet and Webber [SAMET88a] give the execution time as bounded by sum of the sizes of $A$ and $B$.

Let us examine the Jackins and Tanimoto algorithm, so that the extension to the Boolean expression case can be given. The algorithm was given in detail in [JACKINS80]. The basic outline is as follows:
Translate($\Psi_T', S$)  
---Build the intersection of $\Psi_T'$ (the inverse translation of the target node) with $S$ the set of (maximum) eight nodes overlapping octant($\Psi_T'$).

1) Generate eight empty nodes as the children of $\Psi_T'$
2) For i in 0 :7 do
3)   Set $S^*$ to Empty
4)   For $\Psi_A$ in $S$ do
5)     If $\Psi_T'.child[i]$ is Full then
6)       next i at 2) --- child was filled in previous loop on 4)
7)       If octant($\Psi_T'.child[i]$) $\cap$ octant($\Psi_A$) = $\emptyset$ then
8)         next $\Psi_A$ at 4) --- No spatial intersection of nodes
9)       7.1) If $\Psi_A.status = $ Full then
10)          $\Psi_T'.child[i] :=$ Full --- target contained in source
11)          next i at 2)
12) 7.2) Else put $\Psi_A$ into the list $S^*$
13) Else
14)     For k in 0 :7 do
15)       If $\Psi_A.child[k]$ is Empty then
16)         next k at 9)
17)       11) Else if octant($\Psi_T'.child[i]$) $=$ octant($\Psi_A.child[k]$) then
18)           copy the tree rooted at $\Psi_A.child[k]$ to $\Psi_T'.child[i]$
19)       12) Else if octant($\Psi_T'.child[i]$) $\cap$ octant($\Psi_A.child[k]$) = $\emptyset$ then
20)           next k at 9)
21) Else put $\Psi_A.child[k]$ into the list $S^*$
22) Endfor --- End loop on k
23) Endfor --- End loop on set $S$
24) 14) If $S^*$ is not empty
25) Translatce($\Psi_T'.child[i], S^*$)
26) 15) Endfor --- End loop on i
27) 16) Condense($\Psi_T'$)

Notice that only Full nodes are passed in the set $S^*$

7.4.4 Transformation of Boolean Expressions of Octrees

As discussed in Chapter 6, a sectioned object need not be explicitly created. Instead it is regarded as a segmentation (for example, thresholding) operating on the Boolean intersection (or difference), of an object and a VOI mask. It therefore seems appropriate to perform the transformation algorithms also on Boolean expressions. The method is similar to the display algorithms given in the previous section. For example Transform_Intersection uses the relation expressed in equation Table 7.1, to give the status of nodes for testing in steps 3) to 6) of the transform algorithm given above.
7.4.4.1 Translation

As an example, Translate\_Intersection will be described. This can be adapted in a straightforward way from the Translate algorithm given above. The key idea is to pass two sets containing nodes, $S_A$ and $S_B$, that always strictly correspond in space. Then the tests at 6) and 12) are only on one of these sets.

**Translate\_Intersection($\Psi'_T$, $S_A$, $S_B$)**

--- *Build the intersection of $\Psi'_T$ (the inverse translation of the target node) with $S_A$ and $S_B$ the set of (maximum) eight nodes overlapping octant($\Psi'_T$).*

1) Generate eight empty nodes as the children of $\Psi'_T$
2) **For** i in 0 : 7 **do**
3) Set $S_A^*$ and $S_B^*$ to Empty
4) **For** $\Psi_A$ in $S_A$, $\Psi_B$ in $S_B$ **do**
5) \[\text{If } \Psi'_T'.child[i] \text{ is Full then}
   \text{next i at 2)} \]
6) \[\text{Else if } \text{octant}(\Psi'_T'.child[i]) \cap \text{octant}(\Psi_A) = \emptyset \text{ then}
   \text{next } \Psi_A, \Psi_B \text{ at 4)} \]
7) \[\text{If } \Psi_A.\text{status} = \text{Full and } \Psi_B.\text{status} = \text{Full then}
   \text{7.1) If } \text{octant}(\Psi'_T'.child[i]) \supseteq \text{octant}(\Psi_A)
   \text{then } \Psi'_T'.child[i] := \text{Full} \]
   \[\text{next i at 2)} \]
   \[\text{7.2) Else put } \Psi_A \text{ into list } S_A^*, \Psi_B \text{ into list } S_B^* \]
8) \[\text{Else if one of the source nodes is Partial (}$\Psi_A$ \text{ say ) and the other Full}
   \text{9) For k in 0:7 do} \]
   \[\text{--- examine children of } \Psi_A \]
   \[\text{10) If } \Psi_A.\text{child}[k] \text{ is Empty then}
   \text{next k at 9)} \]
   \[\text{Else if } \text{octant}(\Psi'_T'.child[i]) = \text{octant}(\Psi_A.\text{child}[k]) \text{ then}
   \text{copy the tree rooted at } \Psi_A.\text{child}[k] \text{ to } \Psi'_T'.\text{child}[i] \]
   \[\text{11) Else if } \text{octant}(\Psi'_T'.child[i]) \cap \text{octant}(\Psi_A.\text{child}[k]) = \emptyset \text{ then}
   \text{next k at 9)} \]
   \[\text{12) Else put } \Psi_A.\text{child}[k] \text{ into list } S_A^*, \Psi_B.\text{child}[k] \text{ into list } S_B^* \]
   \[\text{Endfor} \]
   \[\text{--- End loop on k} \]
14) \[\text{Else}
   \text{15) For k in 0:7 do} \]
   \[\text{--- examine children of } \Psi_A, \Psi_B \]
   \[\text{16) If } \Psi_A.\text{child}[k] \text{ is Empty or } \Psi_B.\text{child}[k] \text{ is Empty then}
   \text{next k at 15)} \]
   \[\text{17) Else if } \text{octant}(\Psi'_T'.child[i]) = \text{octant}(\Psi_A.\text{child}[k]) \text{ then}
   \Psi'_T'.child[i] = \text{Intersect}(\Psi_A.\text{child}[k], \Psi_B.\text{child}[k]) \]
   \[\text{18) Else if } \text{octant}(\Psi'_T'.child[i]) \cap \text{octant}(\Psi_A.\text{child}[k]) = \emptyset \text{ then}
   \text{next k at 15)} \]
   \[\text{19) Else put } \Psi_A.\text{child}[k] \text{ into list } S_A^*, \Psi_B.\text{child}[k] \text{ into list } S_B^* \]
   \[\text{Endfor} \]
   \[\text{--- End loop on k} \]
Endfor --- End loop on \( S_A, S_B \)

20) If \( S_A^\ast, S_B^\ast \) are not empty
   \( \text{Translate}_\text{Intersection}(\Psi^\ast, \text{child}[i], S_A^\ast, S_B^\ast) \)
21) Endfor --- End loop on \( i \)
21) Condense(\( \Psi^\ast \))

Notice that at step 13, when the node in tree B is Full, and the node in tree A Partial, the virtual child (as defined in Chapter 3), must be placed in the list for set \( S_B^\ast \), in order to ensure that the next iteration of the algorithm receives nodes that correspond in size and position. Actually, it is probably sufficient to pass \( \Psi_B \) if the size and position information is also passed, rather than being stored in the record structure of the node. The details will depend on the implementation.

Other Boolean operations can be derived keeping in mind the rules of table 7.1. Not every possibility was explicitly coded in UCL3D. The implementation in grey or mixed grey/binary representations follows the same arguments as in section 7.2. It seems too complicated to implement these in Mixed Binary representation.

Table 7.6 compares the times for the operations to explicitly create the Intersection or Difference, \( C = A \cap B \) or \( C = A - B \), followed by translation, against the operations \( \text{Translate}_\text{Boolean}(A \cap B) \) and \( \text{Translate}_\text{Boolean}(A - B) \)

<table>
<thead>
<tr>
<th></th>
<th>Create</th>
<th>Translate</th>
<th>Translate _Boolean</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{Int(Sphere,VOI)} ) (ND)</td>
<td>35</td>
<td>280</td>
<td>690</td>
</tr>
<tr>
<td>( \text{Int(Sphere,VOI)} ) (Sun)</td>
<td>53</td>
<td>91</td>
<td>199</td>
</tr>
<tr>
<td>( \text{Dif(Sphere,VOI)} ) (Sun)</td>
<td>55</td>
<td>144</td>
<td>282</td>
</tr>
</tbody>
</table>

Create is the time (in seconds) to perform the Boolean operation (Intersect or Difference). Translate is the time of translation (vector \( (20,30,40) \)) of the resultant of the preceding Boolean. Translate _Boolean is the time of the algorithm that co-traverses both trees. The algorithms were performed on a Sun 3/60 and a Norsk Data ND540. The co-traversal algorithm is more expensive than the separate algorithms in this case, but saves space.

Table 7.6 : Comparative times for operations on the (Unmasked) octrees shown in figure 6.3.
Nodes in Intersection(Sphere, VOI)

<table>
<thead>
<tr>
<th>Level</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>8</td>
<td>8</td>
<td>52</td>
<td>235</td>
<td>906</td>
<td>2,922</td>
<td>7,660</td>
<td>-</td>
</tr>
<tr>
<td>F</td>
<td>0</td>
<td>0</td>
<td>16</td>
<td>219</td>
<td>1,476</td>
<td>7,255</td>
<td>30,010</td>
<td>-</td>
</tr>
<tr>
<td>E</td>
<td>-</td>
<td>56</td>
<td>12</td>
<td>165</td>
<td>755</td>
<td>2,850</td>
<td>8,461</td>
<td>31,270</td>
</tr>
</tbody>
</table>

Nodes in Intersection(Sphere, VOI) after translation

<table>
<thead>
<tr>
<th>Level</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>8</td>
<td>21</td>
<td>65</td>
<td>238</td>
<td>880</td>
<td>2,854</td>
<td>7,660</td>
<td>-</td>
</tr>
<tr>
<td>F</td>
<td>0</td>
<td>18</td>
<td>196</td>
<td>1,572</td>
<td>6,935</td>
<td>30,010</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>43</td>
<td>103</td>
<td>264</td>
<td>828</td>
<td>2,614</td>
<td>8,237</td>
<td>31,270</td>
<td>-</td>
</tr>
</tbody>
</table>

Nodes in Difference(Sphere, VOI)

<table>
<thead>
<tr>
<th>Level</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>8</td>
<td>40</td>
<td>156</td>
<td>699</td>
<td>2,618</td>
<td>8,938</td>
<td>24,453</td>
<td>-</td>
</tr>
<tr>
<td>F</td>
<td>83</td>
<td>777</td>
<td>4,511</td>
<td>22,255</td>
<td>85,568</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>24</td>
<td>164</td>
<td>466</td>
<td>2,197</td>
<td>7,495</td>
<td>24,796</td>
<td>110,056</td>
<td>-</td>
</tr>
</tbody>
</table>

Nodes in Difference(Sphere, VOI) after translation

<table>
<thead>
<tr>
<th>Level</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>8</td>
<td>32</td>
<td>141</td>
<td>628</td>
<td>2,411</td>
<td>8,328</td>
<td>23,088</td>
<td>-</td>
</tr>
<tr>
<td>F</td>
<td>0</td>
<td>65</td>
<td>859</td>
<td>4,865</td>
<td>20,978</td>
<td>80,296</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>32</td>
<td>115</td>
<td>435</td>
<td>1,754</td>
<td>6,095</td>
<td>25,558</td>
<td>104,408</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 7.7: Space complexities of the octrees Sphere $\cap$ VOI, Sphere - VOI, shown in figure 6.3, before and after translation.

For these results A and B were the digital sphere and the VOI defined on it as described in Chapter 6 and shown in figure 6.3. The complexity of the trees before and after translation is given in Table 7.7.

Both these algorithms were timed on a SUN 3/60, using a C implementation. The Translate_Boolean algorithms is slightly worse than the sum of the other two algorithms. There are various possibilities why this is so, but in any case the difference is not very large, and the space complexity of the intermediate tree is saved. Table 7.6 also compares an earlier implementation on the Norsk Data ND540 machine. The performance is worse there because a much cruder algorithm was implemented.
7.4.4.2 Rotation

Rotation, or general rotation and translation, procedures are very important for fully flexible simulations, but are lengthy to perform. Several authors [UDUPA86, TRIVEDI86b] have stressed the importance whilst describing the large computational overhead.

For rotation of octrees, similar results to theorems 7.1 and 7.2 hold. In this case however the cardinality of \( S \) and \( S^* \) is 27, so that the algorithm is more expensive to perform. The general case, rotation and translation, is of the complexity of the rotation process. Weng and Ahuja [WENG87] give another algorithm for general transformation, that looks at the forward transform \( \Psi'' = T(\Psi', \lambda) \) of each full node in \( A \), and traverses \( B \), testing all potential nodes as inside, outside, or intersecting with \( \Psi' \). The algorithm claims to have time complexity linear in the number of nodes in \( A \).

The details of rotation algorithms are often left out in the literature. To implement them the author adapted the Jackins and Tanimoto translation algorithm presented above [JACKINS80]. The principle differences are:

1) The cardinality of the set \( S \) is 27, because a generally positioned cube can intersect at most 27 cubes of the same size in a lattice.

2) The bounding cube, orientated in the source tree, of a transformed target node is used as the overlapping object.

3) Because the bounding cube is a superset of the target node, subtrees of the source are never copied, as at step 11) in the J-T algorithm.

4) A decision must be made at the top level of the target tree, when the target voxels are still in partial intersection with source voxels, as to the target status.

The rotation algorithm may be described as:
\textbf{Rotate}(\Psi_T', S) \quad \text{--- Build the intersection of } \Psi_T' \text{ (the inverse rotation of the target node) with } S \text{ the set of (maximum) } 27 \text{ nodes overlapping } \text{bounding\_box}(\Psi_T').

1) If at top level of tree find average occupancy of set S and set T' as Full or Empty.
2) Generate eight empty nodes as the children of \Psi_T'
3) For i in 0:7 do
4) \quad \text{Set } S' \text{ to Empty}
5) \quad \text{For } \Psi_A \text{ in } S \text{ do}
6) \quad \quad \text{If } \Psi_T'.\text{child}[i] \text{ is Full then}
7) \quad \quad \quad \text{next } i \text{ at 3)} \quad \text{--- child was filled in previous loop on 5)}
8) \quad \quad \text{Else if } \text{bounding\_box}(\Psi_T'.\text{child}[i]) \cap \text{octant}(\Psi_A) = \emptyset \text{ then}
9) \quad \quad \quad \text{next } \Psi_A \text{ at 5)} \quad \text{--- No spatial intersection of nodes}
10) \quad \quad \text{If } \Psi_A.\text{status} = \text{Full then}
11) \quad \quad \quad \text{If } \text{bounding\_box}(\Psi_T'.\text{child}[i]) \supset \text{octant}(\Psi_A)
12) \quad \quad \quad \quad \Psi_T'.\text{child}[i] := \text{Full} \quad \text{--- target contained in source}
13) \quad \quad \quad \quad \text{next } i \text{ at 3)}
14) \quad \quad \text{Else put } \Psi_A \text{ into the list } S'
15) \quad \text{Else} \quad \text{--- } \Psi_A \text{ is Partial}
16) \quad \text{Endfor} \quad \text{--- End loop on } k
17) \quad \text{For } k \text{ in } 0:7 \text{ do}
18) \quad \quad \text{--- examine children of } \Psi_A
19) \quad \quad \text{If } \Psi_A.\text{child}[k] \text{ is Empty then}
20) \quad \quad \quad \text{next } k \text{ at 10)}
21) \quad \quad \text{Else if } \text{bounding\_box}(\Psi_T'.\text{child}[i]) \cap \text{octant}(\Psi_A.\text{child}[k]) = \emptyset \text{ then}
22) \quad \quad \quad \text{next } k \text{ at 10)}
23) \quad \quad \text{Else put } \Psi_A.\text{child}[k] \text{ into the list } S'
24) \quad \text{Endfor} \quad \text{--- End loop on } k
25) \quad \text{Endfor} \quad \text{--- End loop on set } S
26) \quad \text{If } S' \text{ is not empty}
27) \quad \text{Rotate}(\Psi_T'.\text{child}[i], S')
28) \quad \text{Endfor} \quad \text{--- End loop on } i
29) \quad \text{Condense}(\Psi_T')

Figure 7.3 shows the binary digital sphere with a piece dissected out from it and a rotation specified. The means of specifying the rotation will be given in the next chapter. Figure 7.4 shows the slice rotated and Unioned back with the sphere. The rotation algorithm took 15 minutes on the SUN implementation.

7.5 CONCLUSIONS

The need for a sophisticated surgical planning package includes the ability to dissect, transform and merge pieces of object. In the UCL3D system this is achieved using
Figure 7.3: A slice dissected from the binary digital sphere, and a rotation specified.

Figure 7.4: The result of dissecting, rotating and merging an arbitrary slice from a sphere.
octree descriptions of objects and volume masks, and standard octree algorithms for the manipulation. This volumetric approach combines the advantage of complete access to the data, with a space planning ability similar to CAD/CAM applications.

The next chapter will show how all the preceding techniques can be combined to implement a powerful surgical simulation capability, with application to some clinical examples.
CHAPTER EIGHT

APPLICATION TO SURGICAL PLANNING

UCL3D has been undergoing clinical trial since its completion in 1987. This trial is evaluating ten patients from each of three classes: Class II, Class III and Cleft Palate; (these clinical terms are discussed later in this chapter). A full description of its evaluation is the subject of a separate study and is outside the scope of this thesis. In addition various craniofacial deformities have been evaluated: Treacher Collins’ Syndrome, First Arch Syndrome, Apert’s Syndrome, and Crouzon’s Syndrome. In this chapter some examples from facial surgery planning will be given, to illustrate the application of the techniques described in the preceding chapters. The aim of this chapter is to demonstrate the features of UCL3D that are in advance of other systems.

8.1 APPLICATION AREAS

By far the majority of clinical applications of 3D systems are for visualising skeletal anatomy from CT data. The reason behind this is probably twofold: firstly, CT is the most widespread of tomographic imaging modalities; and secondly the high contrast of bone to soft tissue makes segmentation very simple. Within this application area several separate applications may be delineated:

1) Maxillo-facial surgery
2) Cranio-facial surgery
3) Orthopaedic applications
4) Spinal applications.

There are of course numerous other applications, many of which are research areas. Notable among these is the use of 3D information in Radiotherapy Treatment Planning (RTP), which was one of the first applications suggested [SUNGUROFF78, BLOCH83]. From the technical point of view the requirements are very similar to those of ray-tracing, and ideally a full greyscale model should be used. Rosenman has provided a
recent review [ROSENMAN89]. Soft tissue applications include heart and lung studies [AXEL83, HARRIS79, HOFFMAN85, 89], and the musculo-skeletal system [PATE86].

The first actual clinical use of 3D images in medicine was for a spinal application, in this case for the diagnosis of disk disease [HERMAN80]. The complex twisting arrangement of the vertebrae is very hard to visualise in 2D slices, and led to the development of curved reformatting [ROTHMAN86]. A quantitative study comparing reformatted to 3D images for spinal applications, using the CEMAX system was described by Zinreich et al [ZINREICH86]. A general evaluation of this application has been given by Pate et al [PATE87].

Visualisation of the acetabulum and femoral head of the hip joint is an areas that has received a lot of attention. Here the user may need to separate the femoral head to inspect the inside surface of the acetabulum [BURK86]. Another requirement is the visualisation of complex fractures, which is one application that has benefited from the volumetric rendering techniques of PIXAR, [FISHMAN87, LAFFERTY86]. the CEMAX system has also been used for these applications [SCOTT87]. A review is given by Fishman et al [FISHMAN89].

The cranio- and maxillo-facial surgery application will be the one discussed in the remainder of this chapter, because of the nature of the clinical cases explored by UCL3D. The discussion will be from the point of view of a computer scientist. Several references from the clinical point of view exist [HEMMY83, CUTTING86, TESSIER86, VANNIER83b].

8.2 OTHER SIMULATION SYSTEMS

In orthognathic surgical planning it is common to take patient photographs and radiographs, and to plan an operation just by cutting these up with scissors and moving them around [HENDERSON74]. Obviously the insight gained by doing this is very limited. The UCL3D system can be thought of as a three-dimensional version of this. It allows the user to cut up volumes of data, and reposition them an arbitrary number of times, making quantitative measurements where desired. There are many more
sophisticated approaches that await future development, and which will be discussed in Chapter 9. Even so, the fairly (conceptually) simple methods developed so far represent an enormous clinical improvement over previous methods.

The first discussion of surgical simulation was by Vannier and Marsh [VANNIER83a]. They used a three-fold approach:

1) Mirroring of "normal" side of the skull.
2) Cutting computer manufactured models
3) CAD/CAM techniques on surface-based representations, using the McDonnell Douglas frame store

This approach has limited use for more complex syndromes, that do not result in a "normal" side of the face. This was pointed out by Cutting et al [CUTTING86a], who identified two main requirements for a computer aid to orthognathic maxillofacial surgery. These were:

i) A simulation procedure
ii) A Cephalometric system

The cephalometric system in turn must consist of

a) A consistent set of measurements
b) A normative standard.

From a Computer Scientist’s point of view, Udupa [UDUPA86] assessed the clinical requirements for surgical simulation in terms of what was then available, and suggested four main requirements:

i) Pre-Operative examination
ii) Simulated Osteotomy
iii) Simulated reconstruction
iv) Post-Operative follow up.

The Pre-Operative examination, of course, is largely the visualisation stage, combined with the ability to measure distances, (both directly, and along a surface), angles, areas and volumes. The need to remove obscuring objects is also useful. This might be achieved by cut planes, or by deleting connected objects (see for example [ZONNEFELD89]).
It is the simulation of osteotomy, and reconstruction that is the most challenging, and identified by most authors as the most lacking in current systems. It is this capability that is readily possible in UCL3D. Most other approaches have involved going back to original slices, modifying regions of interest therein, and reconstructing a separate object. The problem of simultaneously displaying two or more objects still remains. The MIPG approach [BREWSTER84] and the similar CEMAX approach [WEISBURN86] were described in section (6.1.1) : Regions drawn on the shaded display are combined with a specified depth to describe a set of cutting planes. The intersection of the planes with the contours of the original slices modifies the borders tracked by the surface detection algorithm which can then construct a new object.

Many applications might require multiple osteotomies and trial moves. This was very difficult to achieve in those systems described. Intersection of objects is required for two reasons. Firstly for collision detection; Udupa describes the application where a Le Fort III cut, used to correct exorbitism, may be followed by Le Fort I to correct malocclusion [UDUPA86a] (the different types of procedure are discussed further in section 8.4.1). It is essential that two pieces of bone are not placed in the same space. Secondly, if some bone must be removed, it is possible to show how much by moving a piece into overlap and showing the intersection.

Interference detection is trivial to detect in a volume based method. Although suggested by early authors [e.g. CHEN84a], most systems do not provide this because they do not operate on a true volume model. In UCL3D, the implementation of several Boolean operations, as described in Chapter 7, allows for sophisticated clinical simulations.

The use of an octree representation is not essential of course. Trivedi [TRIVEDI86] identified Boolean operations as a possible method of simulation, and demonstrated simple primitive "masks" in a binary digital array representation. The segment-endpoint (SE) representation has also been suggested [TRIVEDI85]. However, no clinical implementations have been reported.
An alternative approach has been to construct physical models, using a Numerically Controlled (NC) milling machine. This was reported by Vannier and Marsh [VANNIER83b], where 2D contours were cut out of aluminium plate by an NC machine, or just by hand out of thick plastic, and is one of the features available with the CEMAX system [DEV84, DONLON88]. Such capabilities are very useful, and if the models can be cut up, then interference detection is readily available. However, actually measuring interference volumes would be difficult, added to which, several models would have to be made, if several different surgical plans were to be tried out and evaluated.

8.3 INTERACTIVE CONSIDERATIONS

Development of a reliable user interface that can be used by a clinician without supervision is at least as hard as the development of the underlying graphics techniques. The author is indebted in particular to S.Grindrod and D.Plummer for their experience and cooperation in such a development. The full discussion of this interface, which is menu-driven is too lengthy to be given here [UCL3D87]. In the following examples, the user interaction will be given in terms of the algorithms described in previous chapters. For example Boolean intersection is described by Boolean(A ∩ B), rather than the menu selection "Detect overlap between two objects". However, some explanation must be given of the method of inputting points and specifying transformation parameters.

A number of formats exist for specifying translations and rotations. For translations a vector must be specified. This may either be typed in as (x,y,z) movements in millimetres, or marked, by identifying the start and end points of the vector. To mark a 3D point two options provided are

\[ M_1 \quad \text{Mark a point on the surface of a displayed object.} \]

\[ M_2 \quad \text{Mark an arbitrary 3D point in space.} \]

The marking is managed by maintaining for each displayed object the viewing transform and display parameters. Then a surface point is found by projecting a pixel along its inverse transform line, and identifying the first displayable voxel encountered. An arbitrary point is marked by displaying the inverse transform line of a pixel, in the
other view windows that are active. The projection of a second line from a different view identifies a point in 3D space.

To specify a translation, three options are possible:

T₁ Type the displacement vector (in millimetres) (8.3)
T₂ Mark a start point and end point, using M₁ or M₂ (8.4)
T₃ Mark a start point, and type in the end point (8.5)

To specify a rotation is more complex. A number of points are required in the object before rotation, and their corresponding positions after rotation. Three points are required to exactly specify a rotation. Since these are subject to error in specification a strategy for matching is required. The current method is:

let \( a₁, a₂, a₃ \) be the original points and \( a₁', a₂', a₃' \) be their specified transforms.

i) \( a₁ \) is translated to \( a₁' \) (8.6)
ii) The vector \( (a₂ - a₁) \) is aligned to \( (a₂' - a₁') \) (8.7)
iii) The vector \( (a₂ - a₁) \wedge (a₃ - a₁) \) is aligned to \( (a₂' - a₁') \wedge (a₃' - a₁') \) (8.8)

To specify the three points, the user is able to use the following options:

R₁ Mark three points using M₁ or M₂. For each point:
   - R₁₁ Mark the end point using M₁ or M₂ (8.9)
   - R₁₂ Type in the end point (8.10)

R₂ Mark two points using M₁ or M₂. For each point:
   - R₂₁ Mark the end point using M₁ or M₂ (8.11)
   - R₂₂ Type in the end point (8.12)

When two points are specified (R₂), then the third point is generated as:

\[
\begin{align*}
   a₃ &= a₁ + (a₂ - a₁) \wedge (a₂' - a₁') \\
   a₃' &= a₁' + (a₂ - a₁) \wedge (a₂' - a₁')
\end{align*}
\]

This ensures that the vectors used in (8.8) are already aligned.

If the set \( (a₁, a₂, a₃) \) is specified, then the set \( (a₁', a₂', a₃') \) may be specified by typing the displacement of each point.

This scheme is used because there is often a bias to the certainty with which the first point is known. Other approaches are possible, including least-square minimisation.
8.4 CLINICAL EXAMPLES

Three examples will be shown. These are chosen to illustrate some of the variety of methods developed and their use. The discussion will indicate the sequence of operations performed by the user and the quantitative output provided. The cases are

1) Crouzon’s disease.
2) Treacher Collins’ Syndrome.
3) Orthognathic.

8.4.1 General Points

It is worth considering what the clinical requirements for maxillo-facial surgery are, and what one expects to achieve in a surgical plan. Very often the problem is to effect a practical result, such as the correction of malocclusion (failure of the upper and lower jaws to meet properly), simultaneously with providing an acceptable facial appearance for the patient. This means that several techniques may need to be tried out in plan before an acceptable one is chosen [HENDERSON74]. For example, clinically, movement of the maxilla can be known in advance, but the mandible must be moved to fit occlusion afterwards.

A number of classes of patient are distinguished. For example:

Class I - Normal relationship for all jaws, but local irregularities.
Class II - Maxilla prominent, leading to overbite. Retruded mandible. The difference between division I and division II is in the arrangement of the teeth.
Class III - Prominent mandible. Generally requires forward slide of the maxilla.

A number of standard procedures have developed. For example, Le Fort I, Le Fort II, Le Fort III, and Kufner, are cuts of the midface, separating the maxilla and differing in their height, and amount of nasal and orbital regions included. They are used for conditions of midface hypoplasia (retracted maxilla). Operations on the mandible include condylotomy, subcondylar osteotomy, sagittal splits and inverted L osteotomy, all of which are on the ramus, for cases of mandibular prognathism (prominent chin), or
retrusion, and body osteotomies, and genioplasties that are on the lower part (the body) of the mandible. Illustrations of the differences can be found in standard textbooks (e.g. [MOORE85]). They are mentioned here simply to give an impression of the variety of techniques available, and the choices that face the surgeon.

One point that arose from clinical experience was the essential requirement for rotation of osteotomy fragments. The original NORSK implementation of UCL3D did not include rotation, except in a very inefficient manner. The SUN implementation has a faster rotation algorithm (given in section 7.4.4.2). For the purposes of these examples, the user interaction was specified on the NORSK implementation, and then reproduced on the SUN, for speed of processing. All the displays were on the NORSK system however. The rotatory movements are not usually measured before the surgery, and so the system must output them quantitatively.

8.4.2 Case 1: Crouzon Syndrome.

Patient 1, aged 15, was diagnosed as Crouzon’s Syndrome (craniofacial dysotosis). This is characterised by bulging eyes (ocular proptosis) with shallow orbits. A drooping of the lower lip and short upper lip are usually observed. Class III malocclusions and open bite were present. Several related problems occur in hearing and movement. The surgical procedure adopted is the Kufner osteotomy. It is a cut taking in the lower orbits of the eyes, to separate the maxilla. This whole piece is then advanced forward along the occlusal plane to correct the open bite. For this patient the advancement was estimated as 1.5cm.

The patient was scanned on a GE9800 with 35 slices. The voxel size was 0.86 by 0.86 by 5mm. It was necessary to remove the headrest from the scans by 2D ROI editing. Some artefacts were also removed.

The simulation stages were as follows:
A - Display(Patient) in Anterior, Left, and Right views.
B - Mark region R₁ in the Anterior view and Create_Octree V₁ = VOI₁
C - Display_Boolean(Patient ∩ VOI₁) in Left view.
D - Draw region $R_2$ on this view, and \texttt{Edit-Octree(V1)}.

E - Specify the vector of translation by marking the anterior of the maxilla at one end, and the final position so as to cause closure of the open bite ($T_3$).

F - Form a new object (Translated Maxilla) = \texttt{Translate Boolean(Patient \cap V1)}

G - Form the final object Prediction = \texttt{Union_Diff(Translate_Maxilla, Patient, V1)}

Figure 8.1 shows stages A to C of the above. The Anterior, Left and Right views are displayed with ROI, drawn on the Anterior view. The expression (Patient \cap VOI) is shown in the bottom right, with ROI drawn on this view. Figure 8.2 shows the translation vector drawn by marking arbitrary points in space to close the open bite. The movement specified was reported as: (Forward 2mm Downward 1mm). Figure 8.3 shows the resultant of stage G in Left and Anterior views, contrasted with the same views of the original patient. The change in the orbits of the eyes can be clearly seen.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure8.1.png}
\caption{Stages in simulating surgery on Patient 1.}
\end{figure}
Figure 8.2: Marking the translation vector for Patient 1.

Figure 8.3: The original and predicted hard tissues.

(In figures 8.1 and 8.2 tracing paper has been used to accentuate the overlay colours).
8.4.3 Case 2: Treacher Collins’ Syndrome.

Patient 2 is an example of Treacher Collins’ Syndrome requiring rotation and advancement of the mandible and the use of a costochondral graft. The patient is a two-year old girl, who had previously had a tracheostomy operation to correct the breathing difficulty induced by Micrognathia (small mandible).

This syndrome is an example of Mandibulofacial dysostosis, which is genetically inherited and is often lethal. The facial appearance is very characteristic, with depressed cheekbones, receding chin and "fishlike" mouth. Vision is normal but the eyes are downward sloping. The ears are often deformed, especially the middle ear and cochlea, leading to hearing defects as in this patient. The cleft palate found in this patient is characteristic in about 30% of cases. The mandible has deficient ramus (upper piece) and has several other deformities. Dental malocclusion and open bite are often present. The condition often causes mental retardation, which is compounded by hearing loss [GORLIN76].

The diagnosis was:
1. Treacher Collins’ Syndrome.
2. Hearing Deficiency
3. Cleft Palate.
4. Micrognathia (small mandible).

The dataset was obtained on a GE9800 using 41 slices. Two operations were performed. The first was to repair the Cleft Palate and the second was for correction of the mandible as follows:
1. Mandibular advancement and rotation.
2. Bilateral Costochondral graft.
3. Advance and rotate mandible using modified acrylic splint for fixation with several wedges made posteriorly to adjust the bite.

This type of syndrome is basically a growth deficiency of the mandible, and in particular the mandibular condyle is missing. The mandible needs to be rotated through
a large angle and a costochondral graft fashioned and implanted on the mandible. This graft is constructed from bone removed from the rib. The operation causes the occlusal plane to be opened, with the expectation that it will be filled later by growth. The images of the patient show that the condyle deficiency is greater on the right hand side.

The second operation only was simulated, and only the mandible movement. There is currently no facility to design implants, although it is a planned future development. From the Computer Science standpoint, this case requires a 3D sectioning of the mandible, followed by a rotation and union. This is an example where a connectivity-based segmentation operation, such as 3D region-growing, would be beneficial. However the removal of the mandible can be achieved using volume masks, with the benefit that soft tissue can also be separated. (It is not clear how a region growing algorithm would separate the soft-tissues, which are connected). The VOI formation is interesting because it requires several stages of Boolean operations.

A - Display(Patient) in four views - Anterior, Left, Superior, Inferior (Figure 8.4).
B - Define the mask V1 = VOI1 by drawing region R1 in the Left view.
   Display_Boolean (VOI1 ∩ Patient) in Right, Posterior, and Inferior views (Figure 8.5).
C - Edit_Octree(V1) from Inferior and Posterior together.
D - We wish to remove more on the right than the left. Therefore form the resultant : Mand_1 := Boolean(V1 ∩ Patient).
E - Display(Mand_1) in Right, Left, Inferior, and Anterior (Figure 8.6).
G - Form a new VOI (V2) that encloses the material on the right condyle, that is not part of the mandible.
H - Form Mandible := Boolean(V1 - V2).
I - Display the mandible in four views - Left ,Inferior, Anterior, General (θ = 135, φ = 60) (Figure 8.7).
J - Form Remainder := Boolean(Patient - Mandible). This is the skull minus the mandible so far sectioned out.
K - Display( Remainder), and specify the rotation using two arbitrary points in space, rotated to a further two points (method R_2)( Figure 8.8 and 8.9). The system outputs the movements as :
First point (condyle) - Forward 2mm, Right 2mm, Down 41.1mm
Second point (anterior mandible) - Forward 9mm, Up 2mm.

L - Mandible was rotated and displayed. It was decided that the forward movement was too much and it was translated back 8mm. The new object is Rotated_Mandible (Figure 8.10).

M - The final prediction is formed from Boolean(Remainder ∪ Rotated_Mandible) (Figure 8.11).

Figure 8.4 shows stage A, the four views of the Patient, with region R, drawn on the Left view. Figure 8.5 shows stages B and C - the modification of V1 from Inferior and Posterior. Stage E is illustrated in Figure 8.6, where a new VOI (V2) is being constructed. Stage I is shown in Figure 8.7. The rotation specification (stage K) is shown in Figures 8.8 and 8.9. After rotation the repositioned mandible is compared by superposition on the Remainder object and the original Patient. This is illustrated in Figure 8.10. The final prediction is compared in Figure 8.11.
Figure 8.5: Modifying the first VOI for Patient 2.

Figure 8.6: A second VOI is generated to further modify the mandible.
Figure 8.7: Display of the mandible sectioned from Patient 2.

Figure 8.8: Marking the rotation for Patient 2.
Figure 8.9: The rotation as specified.

Figure 8.10: Relationship of rotated mandible to the original.
8.4.4 Case 3: Orthognathic

Patient 3 was a 25 year old female admitted to correct facial disproportion. She had the following diagnosis:

1. Class II division I, (skeletal class II dental base relationship, with compensation of the incisors).
2. Incompetent lip morphology due to short upper lip (too much incisor showing).
3. Increased nasolabial angle.
4. Upper 6 tipped and rotated.
5. Upper central line shifted to the left.

The required orthodontic treatment planned was:
1. Le Fort I intrusion of maxilla, 7mm upward at the anterior, 3mm upward at the posterior.
2. Bilateral sagittal split, subsigmoid of the mandible. Upper midline 2mm to the left.
3. Genioplasty to correct lower midline 4mm to the right.

These measurements were made clinically when the patient was seen. In the actual operation, the third step (Genioplasty) was not performed. Four Champey plates were used to secure the segments in place. The patient was scanned on a Phillips 350 Tomoscan with 29 slices at 4.6mm intervals. Artefacts in two slices were corrected manually.

In the actual operation the first two procedures only were performed. These two procedures were simulated sequentially using UCL3D. The clinician performed the following steps:

A - Display(Patient) in Anterior and Left views (Figure 8.12).

B - Draw region $R_1$ in the Anterior view, simulating a Le Fort I cut.

C - Display_Boolean(Patient $\cap$ VOI$_1$) in Left view and Edit_Octree(V1) from this view.

D - Form Maxilla := Boolean(Patient $\cap$ VOI$_1$), and

Remainder_Max := Boolean(Patient - VOI$_1$).

E - Display(Maxilla) in Anterior and Left views.

F - Specify the rotation. Here two points were identified as arbitrary positions in space. These were points on the anterior and posterior of the maxilla delineated by the volume mask. Since the required movement had been decided in the clinical assessment the movements were typed in ($R_{2b}$):

$a_1$ (anterior maxilla) upward by 7mm and left by 2mm.
$a_2$ (posterior maxilla) upward by 3mm and left by 2mm.

The actual movements were calculated as (7.46, 2.13) and (3.2, 2.13) respectively (Figure 8.13). The discrepancy arises because of the rigidity of the object. Here the first point was given greater bias as explained on page 188.

G - Rotated_Max := Rotate(Maxilla) was performed (about 15 minutes) and Display(Rotated_Max) in Anterior, Left, and Inferior views (Figure 8.14).
Figure 8.12: Two views of Patient 3, and the creation of the first VOI.

H - \[ \text{Predicted}_\text{Maxilla} := \text{Boolean}(\text{Remainder}_\text{Max} \cup \text{Rotated}_\text{Max}) \] and
Interse\$ion_\text{Max} := \text{Boolean}(\text{Remainder}_\text{Max} \cap \text{Rotated}_\text{Max}) \] were formed and displayed in Anterior and Left views (Figure 8.15). Intersection\_Max shows the shape of material that must be removed in order to put the maxilla into the specified relationship.

I - After moving the maxilla, the mandible was transformed. Draw region Mand\_R, in the Anterior view and \text{Create}_\text{Octree} to form V2 := VOI,

J - \text{Display}_\text{Boolean}(\text{Predicted}_\text{Max} \cap \text{VOI}) \] in Inferior view and \text{Edit}_\text{Octree}(V2) from this view (Figure 8.16).

K - Form Mandible := Boolean(\text{Predicted}_\text{Max} \cap V2), and
Remainder\_Mand := Boolean(\text{Predicted}_\text{Max} - V2).

L - Display(Mandible) in Anterior and Inferior views, and
Display(Remainder\_Mand) in Anterior and Left views (Figure 8.17).

M - Specify the rotation for the mandible. In this case, the required movement could not be known preoperatively until the maxilla was moved. Thus both
\(a_1'\) and \(a_2'\) were specified interactively (\(R_{2a}\)) (Figure 8.18). The system reports the movement:

- \(a_1'\) (anterior mandible) forward 9mm upward 6mm and left 2mm.
- \(a_2'\) (posterior mandible) forward 1mm upward 2mm and left 2mm.

**N** - Rotated_Mand := Rotate(Mandible) was performed (about 25 minutes).

**O** - Final_Prediction := Boolean(Remainder_Mand \(\cup\) Rotated_Mand) and Intersection_Mand := Boolean(Remainder_Mand \(\cap\) Rotated_Mand) were formed and displayed in Anterior and Left views (Figure 8.19).

Intersection_Mand shows the shape of material that must be removed in order to put the mandible into the specified relationship.

**P** - Display(Final_Prediction) was compared in Anterior and Left views with both Display(Predicted_Maxilla) (Figure 8.20) and Display(Patient) (Figure 8.21).

Notice the cut that has been introduced in the ramus of the mandible.

**Q** - Transverse and sagittal 2D slices were reformatted from Final_Prediction (Figure 8.22).

---

**Figure 8.13:** Specifying the rotation of the maxilla for Patient 3.
Figure 8.14: Three views of the rotated maxilla of Patient 3.

Figure 8.15: The prediction after maxilla movement, and the interference volume.
Figure 8.16: Marking the second VOI for removal of the mandible.

Figure 8.17: The mandible and the difference volume from the maxilla prediction.
Figure 8.18: Specifying the rotation of the mandible for Patient 3.

Figure 8.19: The combined prediction and the interference volume of the mandible.
Figure 8.20: Comparison of the final prediction and the maxilla prediction.

Figure 8.21: Comparison of the final prediction and the original for patient 3.
8.5 DISCUSSION AND CONCLUSIONS

This chapter has not been intended as a detailed discussion of surgical procedures. The purpose has been to illustrate how the algorithms developed in Chapters 6 and 7 have practical application for visualising the simulation of such procedures.

What is the value of such a simulation? Three main points have arisen:

1) Firstly, the more that is known clinically before the operation the more confident the surgeon will feel, lessening the possibility of mistakes during the operation. Commonly the surgeons estimate several hours reduction in actual operating time.

2) Secondly, if the prediction is accurate, then it is useful for the patient, to assess whether the (considerable) stress of undergoing operation is worth it.

3) Thirdly, it is important for research - the user has a quantitative measure of movements and spatial relationships.
The simulation is of course crude - it relies on describing living tissue as if it were rigid. It can be thought of just as a three-dimensional version of the photographs and scissors techniques routinely used for practical planning cases [HENDERSON74]. Even this capability has produced great interest in clinical users, and represents a significant advance over simple static viewing of three-dimensional data. Ideally, for the third point mentioned one could quantitatively assess the effect of hard tissue surgery on soft tissue. This point, which has only begun to be addressed, is discussed in the next chapter.
CHAPTER NINE

SUMMARY, ADDITIONAL ASPECTS, AND FUTURE WORK

Medical 3D graphics is a recent and rapidly expanding field. The work presented in this thesis cannot hope to emulate all the aspects of other research and commercial systems. However the author proposes that the methods that have been introduced here are a starting point for a different approach to the problem of interactive applications in the field.

UCL3D is an ongoing project, and has also involved several other workers. The purpose of this chapter is to summarise the original contributions of the author, and indicate some other areas that may be considered as "work in progress". In addition suggestions for future developments can be made.

9.1 SUMMARY

The task facing the author was to design a simulation package for maxillo-facial surgical planning, and implement it on a small minicomputer, so that it could be used in a stand-alone fashion by a "computer-illiterate" surgeon. In surveying the history of the field of Medical 3D graphics, it can be seen that the first interest was one of visualisation, subsequently there arose a demand for manipulation and quantification. The last of these has been indicated implicitly, throughout this thesis, and in particular in Chapter 8, where quantitative output was demonstrated as used in a clinical applications. It is discussed in more detail in section 9.2.1 below. The visualisation aspects were developed to a large extent using standard techniques. Some original methods arose, largely as a consequence of the chosen representation. The manipulation aspects are what the author considers the most original feature of this system.

One of the points that has been stressed by the author is the importance of the choice of representation. The first developments by the author used a B-R representation,
and concentrated on visualisation. However, such a representation was inadequate for
the more complex manipulation tasks that were required. The representation must be
*volume-based*, in order to allow "dissection" of the objects of interest. It must be
*random-access* to allow quantitative measurement and local information. It must be
relatively compact, in order to manipulate multiple objects. In addition it is attractive
to use a *greyscale* representation in order to allow flexibility of *segmentation*. No one
representation is ideal for all tasks, and indeed some authors have suggested the use
of multiple representations, updated simultaneously by a multiple processor system.
Such an implementation is attractive, but would require a novel architecture that itself
might prove restrictive on further developments. Actually, UCL3D does provide
interconversion between representations, as will be discussed in section 9.2.2 below.

At the time of beginning the system development, the most advanced medical
workstation was Meagher’s INSIGHT system, based on an octree representation
[MEAGHER85a, 85b], and this representation was the primary one investigated by the
author. Some of the algorithms developed represent an advance on Meagher’s work,
although, of course, not emulating the speed of the INSIGHT system, which used
special purpose hardware. It should be stressed, however, that a number of other
representations might be applicable, depending on the hardware system that is being
used.

UCL3D was first implemented in a binary octree representation. The display algorithms
were BTF and FTB, although the latter, which are more complicated to implement did
not appear significantly faster. In order to speed up the display algorithms the Masked
Binary octree representation was investigated. This appeared attractive for two reasons -
firstly it allowed a speed up in BTF display (Chapter 4), and secondly allowed the
inclusion of object-based shading techniques (Chapter 5). However it was rejected in
the end for three reasons - firstly, the time and space complexity introduced by the
need to detect and store the neighbour codes proved prohibitive; secondly, the validity
of the neighbour codes broke down under Boolean operations involving Complement;
and thirdly, the flexibility in segmentation and shading introduced by using a greyscale
representation outweighed possible disadvantages due to slower algorithms and greater
space requirements.
The greyscale representation that was developed, stores at internal nodes, information on the minimum and maximum of all voxels represented by that node. This increases the efficiency of display algorithms since subtrees do not have to be traversed if they contain no voxels of interest.

In order to improve the appearance of image-based shading of octree represented objects a novel type of ray-tracing algorithm was developed (section 4.4.3.3). This generates the quadtree of the orthogonal representation of an octree, and allows "ray operations", such as described by the Hamburg group [HÖHNE87a]. In particular, sub-voxel (linear) interpolation was used to increase the resolution of the depth buffer and generate much smoother shading using image-based techniques (Chapter 5). A detailed analysis of this algorithm was given in Appendix A.

Manipulation of 3D medical data is a much less developed problem than visualisation. Previously, the main manipulation feature of 3D systems has been the desire for "cut away" views, such as slicing off the top of the cranium to see inside the brain, and were thus again primarily for visualisation. The techniques to do this, such as half-space masking, are much simpler to implement than a complex dissection simulation. The author therefore considers this area the main contribution of his work. The techniques developed allow an object to be "edited" in a general way, and individual fragments to be independently moved and merged together. A combination of these techniques allows sophisticated surgical simulations.

One general philosophy that has developed has been to attempt operations, such as display and translation, on Boolean expressions of multiple objects. For visualisation, this appears to produce savings in time and space complexity. For transformations the savings in time complexity are not so apparent, although this may be improved with more efficient algorithms.

It must be stressed that the work described here is only a beginning. There are many features that have yet to be developed, or are developed but are far too slow. Some of these will be discussed in sections 9.2 and 9.3.
9.2 FURTHER ASPECTS

So far, this thesis has discussed the problems of data acquisition, representation, segmentation, display, dissection, and transformation. The system described allows a clinician to display hard and/or soft tissues on the screen, interactively simulate the "cutting" of these tissues, and rearrange them in an arbitrary fashion. In addition quantitative output is required. This has been implicit in the techniques described, but will be commented on in section 9.2.1. Two further aspects are required. The first is an ability to simulate the non-linear effects on the soft tissue of a rearrangement of the hard tissue. The second is a means of comparing predicted and actual outcomes of surgery.

9.2.1 Quantification

Quantification is an important aspect in clinical applications and again it is not trivial to include if an inappropriate representation is employed. One of the most important aspects is to be able to measure distances between points on the surface of objects of interest. In UCL3D this is easily achieved by moving a cursor to the point on a 2D display of the object and then tracing the ray from this 2D point until it intersects the object. If the Z-buffer of the image is maintained, then the point in question is read out from the z value in the buffer at that pixel. By contrast, in a B-R representation, if the Z-buffer is not maintained, then the object point has to be found by searching in the representation for a facet whose projection includes the pixel selected [TRIVEDI86b].

The ability to measure surface areas and volumes is not present in UCL3D however, since these require connectivity knowledge of the object. If the object is assumed to be detected just by thresholding, then it is simple to sum the volumes of all voxels satisfying the threshold criterion. If several disconnected objects are present then it is only possible to measure the volumes if they are separated, perhaps by region-growing. Again, the use of a volume mask to demarcate the objects of interest would be an advantage. This has not been explicitly implemented in UCL3D, although it would be simple to do so if required. Surface area estimation is difficult to implement in the
greyscale representation. In the binary representation it could be achieved by summing the voxel faces that do not have a 2-neighbour flag set.

By contrast both surface and volume measurement are easy to achieve in both the B-R representations, and in the voxel boundary representations. In the former it is a case of summing the areas of polygons for the surface, and the volume of polyhedra obtained by connecting the facets to an internal point for the volume. These will both require extensive floating-point operations. In voxel boundary methods, on the other hand, it is trivial to sum the faces stored in their six component lists for the surface area, and the differences of the separate sums of faces in the positive and negative directions, weighted by position, for the volume [UDUPA81].

What is not always discussed however, is the accuracy of such measurements with respect to the approximations made in the representation. In B-R representations the quantitative output will depend on the tiling scheme used to construct the representation. In voxel methods it will depend on the connectivity choice used, and on the interpolation methods used. One interesting topic would be the degree to which the first order interpolation used in the shading scheme of section 5.2.2.2 accurately determines the surface. Such analyses are only beginning to be addressed [HOFFMAN85, RAYA88].

9.2.2 Soft tissue Prediction

Linear transformation of voxel arrays is adequate for hard tissue prediction, but inadequate for soft tissues. For example, the patient shown in figure 1.2, underwent a similar surgical simulation as that of Patient 1 in Chapter 8. The result is shown in figure 9.1, where here the soft and hard tissues have been moved by the same amount and are shown in superposition. Clearly the soft tissue shows an unacceptable "step". Clinically this is a well known problem, and the "cut and paste" technique of Henderson [HENDERSON74] has been shown to be inadequate. Several studies have been made of the relative movements of well located hard and soft tissue landmarks and their relapse over time [SUCKIEL78, WILLMOT81]. For certain points the soft tissue movement is only 80% to 90% of the hard tissue movement, with a high degree
One solution attempted was to utilise a B-R description for the soft tissue surface. The B-R is generated by 2D contouring in slices reconstructed from the transformed data volume. The slices formed from the example in figure 9.1 are shown in figure 9.2. The volume gap introduced by the translation is clearly visible. When the contouring algorithm was applied to this set, many broken regions occurred as are also shown in figure 9.2. UCL3D allows manual editing of these regions to smooth over the step. The tiling algorithm of section 2.3.2 was applied to these edited contours and the result is shown on the left in figure 9.3, compared with the result on the right from the unmodified data. This was thought more acceptable by the clinician involved [MOSS88].

However, several more powerful methods could be developed relatively easily within the software environment of UCL3D. The 3D vector of translation is known, and slices
in a plane containing this vector could be employed to simulate the relative movement of soft to hard tissues along this vector. A more sophisticated approach could be the use of Finite Element methods to relate the stress induced by a hard tissue shift to a strain in the soft tissue using elastic properties of tissue. This could be an interesting and fruitful line of research.

9.2.3 Comparison of prediction to outcome

Suppose that an acceptable prediction of post-surgical appearance has been made. How can it be compared with the real post-operative outcome? A post-surgical dataset is required and a means of registering this dataset to the predicted dataset. An evaluation metric is also required.
9.2.3.1 Registration

Registration is necessary because the coordinate system of datasets taken by different modalities, or at different times, are not identical. There appear to be two approaches to the registration problem which, again, are either surface or volume based. In the surface based approaches, a number of points are located interactively and identified as corresponding in each image. Then a transform is derived to map one set of points to the other, either as a rigid body [TOENNIES89], or including warping [SCHIERS89]. In the volume approach the cross-correlation of the characteristic function $\Phi_0$ for both objects is maximised by equating the first three coefficients of their Fourier expansions [GAMBOA-ALDECO86]. A more complex warping scheme for 2D scans was used by Bajcsy et al [BAJCSY83] to relate individual scans to an "atlas" of normals.

A preliminary attempt at the landmark based approach has been made in UCL3D. If three non-coplanar points are known precisely in each of two coordinate systems then there exists a linear transformation that changes one coordinate system into the other.
For example figure 9.4 shows three data sets, in a B-R description, obtained by 1) CT (upper left), 2) laser-scanning pre-operative (upper right), 3) laser-scanning post-operatively (lower right). An attempt was made to identify three points manually in each dataset, that were unchanged by the operation. The transformation was then derived using exactly the method for deriving the transform for rotation of objects described in section 8.3 equations (8.9) and (8.10). The resultant transformed sets are shown in figure 9.5.

Unfortunately, precise identification of three points is not possible, due to both manual and digitisation error. Possibilities would be to affix absolute landmarks on the face that were identifiable in all datasets. Also more than three points could be used and a least-squares fit made.
9.2.3.2 Evaluation of surface differences

Evaluation is not trivial because a means of finding the distance between surfaces is required. One solution is to show the distance between surfaces as an intensity, with different colours for positive and negative displacements. Figure 9.6 is a schematic diagram. An example is shown in figure 9.7 for the three data sets of figure 9.5. The surfaces are shown in dark blue, with positive differences in red, and negative differences in green. The upper left image is differenced with the upper right, the upper right with the lower right, and the lower right with the upper left. In the lower left, profile views of the three data sets are shown superimposed (CT image in blue, laser between 1) and 2) was quite well achieved, but not so well between 1 and 3), as is to be expected since there is a physical change between 1) and 3). The displacement of the lower jaw is visible. On a fast machine this visual display could be used for the matching process, without needing to identify landmark points.
Figure 9.6: Visualising surface differences.

Figure 9.7: Difference images of the three data sets from figure 9.5. Top Left: CT image in blue, with difference from top right superimposed; Top Right: laser scan image (preoperative) in blue, with difference from bottom right superimposed; Bottom Right: laser scan image (postoperative) in blue, with difference from top left superimposed; Bottom left: profiles of the three datasets, superimposed.
9.3 FUTURE DEVELOPMENTS

Medical image processing encompasses topics of interest to Mathematicians, Computer Scientists, Physicists, and Clinicians. There are many research topics that can be developed, either as extensions of the author's own work, or in new areas, some of which have already been indicated in section 9.2. Each of the areas of interpolation, quantification, soft-tissue prediction, and registration require investigation. In addition there are still many topics in segmentation that are underdeveloped.

Regarding the work presented here, there are several Computer Science related topics that could be investigated. The octree representation is only one of a class of volume-based, greyscale representations. It would be useful to compare the implementations of the system as described here with different representations. In particular the Marginal-Indexing representation (section 3.2.2) seems to have been neglected in the literature. Several of the algorithms would benefit from further analysis. In particular the translation and rotation of Boolean expressions of objects could undoubtedly be improved. As mentioned, algorithms like Translate_Boolean, Union_Diff and Display_Boolean, presented in Chapter 7 have been explicitly coded because of the need to improve their speed. It is possible to conjecture a scheme where complex Boolean relations are "parsed" at each level of an octree, and reduced to primitive operations. Then the extent of user interaction could be reduced still further.

The author has been at pains to stress the advantage of a volumetric representation, but perhaps the regular tessellation of a 3D array is too restrictive. Certainly the data is acquired in this format, but if manipulation and predictive reformation is carried out on this data, then such a representation may not be appropriate for the result. Consider the problem of estimating relaxation of hard tissues following corrective surgery, and the commensurate changes in soft tissue. It would be possible to include stress-strain and elasticity measures into the data, and to apply a Finite Element analysis. Then the resultant structures would be warped from the original tessellation, and would be complicated to render and quantitatively analyse. In addition, it is likely that future workers will attempt to include a priori knowledge into operations such as segmentation and manipulation. This would allow the modelling of muscles and other soft tissues in
an object-orientated way. What representations would be appropriate for such models is also an interesting topic.

UCL3D as so far described is only meant to be a prototype. Its purpose was to determine what functions are necessary and useful for a graphics workstation that allows interactive surgical planning. By far the largest drawback to the current system is its slow speed. Displays take tens of seconds to minutes to appear, and translation or rotations much longer. Many of these defects lie in the short amount of time available for development. It is expected that careful analysis of algorithms could produce many orders of speed up. However the real improvements are likely to come with judicious special purpose hardware. Such hardware must be flexible enough to include fast access to the original data, probably simultaneous representations, and above all the ability to handle multiple objects.

Clinical end users are extremely demanding in their requirements. They are not interested in technical advances unless of immediate advantage to their application. Despite a large body of research and development in 3D Medical graphics in the last decade, the real clinical advantages are a matter of contention [TESSIER86]. Perhaps the most interesting questions lie in the man-machine interface, and in the quantitative evaluation of the efficacy of the solutions. Throughout the, relatively short, history of medical imaging, developments in algorithms and concepts have gone on ahead of commensurate advances in computer technology. When the techniques described in this thesis are possible in real-time, and accepted as useful, it is certain that many more demanding questions will be asked.
APPENDIX A

COMPARISON OF OCTREE TO QUADTREE ALGORITHMS

A.1 INTRODUCTION

The principle aim of this appendix is to analyse the image driven "Node-Tracing" algorithm of section 4.4.3.3. The algorithm is compared with object-driven octree algorithms (BTF and FTB). Some performance criteria are defined in terms of the "visits" to each node of the data, and these are evaluated for the various types of octree structure introduced in Chapter 3. The evaluation is made for worst-case and synthetic objects, and the application to medical data. The performance of the new algorithm is no worse in general than other algorithms. Its advantage is its "active" nature. Section 5.2.2.2 showed how to interpolate the greyscale data along the rays to provide a better estimate of surface location, which gave rise to the improved shading effects described in section 5.3.

Section A.2 restates the BTF and FTB algorithms in their simpler form for orthogonal projections. In Section A.3 some analyses of the new algorithm are made, and comparative performance criteria based on the number of octree nodes visited for a given quadtree node, and the number of "unvisited" octree nodes after a full pass of the algorithm are suggested. Section A.4 presents some results of performance using the two criteria mentioned.

Both binary and greyscale octrees are considered. Because the node-tracing algorithm is quadtree driven, an interesting result is that greyscale octrees take about the same time as binary octrees. An improvement is found if the min-max representation introduced in section 3.3.2 is used. Results are presented for both synthetic data, and real medical objects.
The algorithm is limited to orthogonal projections, although general projections are highly desirable. This presents a greater difficulty because the 3D coherence of octree nodes no longer directly corresponds to the coherence of the 2D quadtree. Correspondence is maintained for some other special viewing directions, such as the isometric position [YAMAGUCHI84], but these are limited. One possibility is to use an efficient algorithm for rotating and resampling the octree [WENG87]. Several other groups (e.g. [TIEDE87]) limit displays to orthogonal projection and use object-space rotation to obtain general views. However experience with UCL3D suggests that multiple orthogonal projections are the most important for the application to Surgical Simulation and Planning.

A.2 ORTHOGONAL OCTREE DISPLAY ALGORITHMS

Let us consider the octree BTF (Section 4.4.3.1) and FTB (section 4.4.3.2) algorithms for the special case of orthogonal projection where each octree node corresponds to a single quadtree node. The quadtree and octree are traversed simultaneously, with two octree nodes (front and back) visited for each octree node.

The BTF algorithm is as follows:

\[
\text{Orthogonal\_BTF}(\Psi, \Omega) \quad \text{Back to Front orthogonal display of octree node } \Psi \text{ into quadtree node } \Omega.
\]

1) If \( \Psi \text{.status} = \text{Empty} \) then
   return
2) Else if \( \Psi \text{.status} = \text{Full} \) then
   \( \Omega \text{.status} := \text{Full} \)
   \( \Omega \text{.colour} := \text{shade\_of\_node}(\Psi) \)
3) Else \( \Psi \text{.status} = \text{Partial} \)
   For i in visit order of the quadtree nodes do
      \text{Orthogonal\_BTF}(\Psi \text{.child}[\text{back}(i)], \, \Omega \text{.child}[i])
      \text{Orthogonal\_BTF}(\Psi \text{.child}[\text{front}(i)], \, \Omega \text{.child}[i])
   endfor

The FTB algorithm is described as :
Orthogonal FTB($\Psi, \Omega$)—Front to Back orthogonal display of octree node $\Psi$ into quadtree node $\Omega$.

1) If $\Psi$.status = Empty do
   return
2) Else if $\Psi$.status = Full and $\Omega$.status = Empty do
   $\Omega$.status := Full
   $\Omega$.colour := shade_of_node($\Psi$)
3) Else --- Either the octree node or quadtree node is Partial
4) $\Omega$.status := Partial
5) For i in visit order of the quadtree nodes do
   if $\Psi$.status = Partial then
      5.1) Orthogonal FTB($\Psi$.child[front(i)], $\Omega$.child[i])
   else
      5.2) Orthogonal FTB($\Psi$, $\Omega$.child[i])
6) if $\Omega$.status = Full then --- It has just been filled
   do next i at 5)
7) if $\Psi$.status = Partial then
   7.1) Orthogonal FTB($\Psi$.child[back(i)], $\Omega$.child[i])
   else
      7.2) Orthogonal FTB($\Psi$, $\Omega$.child[i])
      Orthogonal FTB($\Psi$.child[front(i)], $\Omega$.child[i])
endfor
8) Test whether all four quadtree children have been painted. If so label the node as "Masked"

There are several equivalent forms for this algorithm. It is written in this form to make explicit the method of analysis used in section A.3; namely that we note that a quadtree node is tested at least once for every invocation of the procedure, and twice if and only if both itself and the corresponding front octree node are Empty. The particular advantage of the FTB is realised if the condition that all four children of a quadtree node are Full is recognised. Unless all four children have the same colour, the quadtree cannot be reduced to a lower node. Instead a "Masked" status is used to indicate that, although a node is inhomogeneous, none of its children need be tested against any further octree nodes.

Ray tracing an orthogonal projection into a digital array is simply a case of stepping from voxel to voxel. The worst case would be an Empty world, where each of $N^2$ pixels required $N$ steps. In a notation to be described below, we say that for this worst case the average voxel visit is 1 and the average voxel per pixel visit is $N$. 
Clearly some savings are likely if lower nodes of a quadtree can be stepped through an octree representation of the array.

A.3 ANALYSIS

Analysis of octree algorithms is notoriously difficult because of the spatial instability. Weng and Ahuja [WENG87] suggest a "worst-case" tree where all leafnodes are at level m, and an average case, where the number of nodes at level m is a fixed percentage of the total possible number of nodes at that level. However the worst case for FTB or BTF is not necessarily the worst for the node-tracing. A simple theorems about the performance of the Node-Tracer can be stated:

Theorem A.1 : The number of octree nodes visited per pixel for a quadtree node at level m in an n-level tree is bounded by $2^{(3m-2n+1)}$.

Proof : 

A quadtree node at level m is created when the algorithm encounters a Partial node at level m-1 in the octree. The algorithm steps through the octree at level m in steps of $2^{(n-m)}$ or greater. Let us suppose that the quadtree node was created from an octree node at depth $D(m-1)$ at level m-1 where $D(m) = d.2^{(n-m)}$, $0 <= d < 2^m$. There thus remain at most $A(1) + A(2) + ... A(m)$ steps at level m. The total number of steps will therefore be $2^m - D(m-1)$ steps at level m. From theorem 3.1 the expectation value of the number of accesses is at worst $2^m$. Thus the maximum accesses at level m is $2^{(m+1)}$. A quadtree at level m covers $4^{(n-m)}$ pixels. Thus the number of visits per pixel is at worst $2^{(m+1)} / 4^{(n-m)} = 2^{(3m-2n+1)}$, as stated.

Corollary A.1 : An octree node at level m will be accessed at most $4^{(n-m)}$ times, i.e. the 2/3 power of the number of voxels in the node.

Proof :

This proof follows from inspection of the algorithm. Notice that the only quadtree rays that can propagate are homogeneous; Partial quadtree rays are immediately
recursed. Then for a Full octree node, all child quadtree rays testing the node will be terminated. But there will be at most one ray for every pixel covered by the octree node, which is $4^{(n-m)}$. For an Empty octree node, any rays will be tested once and then handed straight onto the next node. Again there are at worst $4^{(n-m)}$ of these. For Partial nodes, either the algorithm will recurse (one visit), or the node is being visited enroute for a lower level node. But such a visit is only required for the frontmost children, and at worst for the $4^{(n-m)}$ front voxels of that node, if they exist.

From Theorem A.1 it follows that time and space complexity is bounded by $O(H 2^{m+1})$ where $H$ is the number of quadtree nodes. Thus at worst the Node-Tracer is twice as bad as directly tracing a 3D array. However in practice the worst case is very difficult to construct. An m-worst case would consist of an m-level tree where all nodes at level m were Empty, and all nodes at lower levels were Partial. This would be contradictory to the normal reduction of homogeneous volumes to lower level Empty nodes. As we shall see in section A.4, this could occur in a greyscale octree, if the dynamic threshold chosen did not include any of the grey-values in the tree. In that section we will also see that it is then an advantage to use a min-max octree representation, so that inhomogeneous nodes can be tested as falling outside the dynamic density range selected.

A.3.1 Performance criteria.

Because the Node-Trace algorithm is recursive in 2D rather than in 3D, its time and space complexity is difficult to compare with FTB or BTF. Let us use a criterion in terms of the number of octree nodes visited per quadtree node. Also of interest is how many times a given octree node is visited, and in particular how many nodes are completely unvisited. In the following, inhomogeneous nodes are referred to as Partial.

Consider the way the different algorithms perform when confronted with a sequence of nodes like those in figure A.1. Here A is a Full node and B a Partial node, both co-children of the same parent (not shown), and C and D are Empty, and also co-children of one parent (different to that of A and B). If the order of increasing depth from the
screen is ABCD, then the three algorithms will operate in the following way for the Masked and Unmasked Binary representation of these nodes:

Node: A  B  children of B  C  D
Visited by:
BTF
Unmasked  Yes  Yes  Yes  Yes  Yes
Masked    Yes  Yes  No   No   No
FTB
Unmasked  Yes  No   No   Yes  No
Masked    Yes  No   No   Yes  No
Node-Trace
Unmasked  Yes  No   No   No   No
Masked    Yes  No   No   No   No

It is the effect of these relationships on actual data that is studied in the following.

A.3.1.1 Front-to-Back

In FTB, at each access of the quadtree node, two octree nodes are potentially displayable. If the quadtree node is Full, neither is tested, one visit is counted at the quadtree node, and one visit at the front octree node. Otherwise two visits are counted at the quadtree node, and one visit each for the front and back octree nodes. From the
recursive definition of the algorithm, each Partial node in FTB is visited only once at most. If the front child is Full the co-child is not visited. Thus if the front octree node changes the status of the quadtree node to Full, the rear node will not be visited, and only one visit is recorded at the quadtree node. If the octree node is at a lower level than the quadtree node, (i.e. it is larger), assume that the FTB algorithm has no knowledge of this, so that it treats it as composed of virtual octants at the current recursion level. Thus the number of visits at the quadtree node at level m is bounded by \(2^{(n-m)}\). On counting the visits to an octree node, the only nodes recording more than one visit will be a large homogeneous node preceded by Partial nodes. Then the number of visits will be at most the number of voxels in that node. But a particular implementation will probably ensure that that node will only be visited once. Thus in the actual analysis only one visit was recorded in this case.

A.3.1.2 Node-Trace

When comparing the Node-Trace algorithm, it is seen that a quadtree node will stop recording visits as soon as it becomes Full or Partial. Thus at first sight, the performance cannot fail to be better. However the cost in stepping to a node that is not a co-child must be counted. A separate record was made of the "direct" visits to an octree node, wherein the node is tested against the corresponding quadtree node and/or projected into it, and the "indirect" visits, only made during neighbour access in the procedure \texttt{new\_node}.

If a node is a co-child then no extra cost is incurred. The paths to two adjacent nodes will be coincident only down to the pivot level, which at worst will be as low as the root. If the pivot-level is one less than the recursion level then the two nodes share the same parent, and only one direct visit to the quadtree is counted. Otherwise count one direct visit and \((\text{recursion-level} - \text{pivot-level} - 1)\) indirect visits to the quadtree, and count one indirect visit in each octree node encountered in the path of the adjacent node, after the pivot level. Corollary A.1 states that Partial nodes can record multiple visits, up to a maximum of \(4^{(n-m)}\) (i.e.(number of voxels in node)\(^2\)). As in FTB an homogeneous node may record multiple visits, if it is visited by the rays from the children of a Partial node that precedes it. This case is dealt with by Oliver
by creating the virtual octants (section 3.4.5), that are the homogeneous children of an homogeneous node, whereas no such extra nodes are required in Node Tracing. Here, the maximum visits at such a node is 4^{n-m}. However in the node-tracing case the visits represent real work of the algorithm. Thus the possible improvement of the algorithm will depend on to what extent the extra "unvisited" nodes compensate against the cost of multiply visiting higher nodes.

A.3.1.3 Back-to-Front

To complete the comparison the performance of the BTF algorithm should be included. Clearly every octree node is visited once in the Unmasked Binary representation. In practice there is no point in converting the octree to a quadtree as no testing will be done to make use of area coherence. Instead each quadtree node would be rendered as found, and overwritten by the Painter's algorithm. In order to compare, the visit to each pixel was recorded as 1/4^{n-m} and afterwards the quadtree was recreated by summing the visits at each pixel contained in an homogeneous quad node.

A.4 RESULTS

Let us compare the three algorithms (BTF, FTB, Node-Trace) on 3D objects represented in each of four data structures (Binary, Masked binary, greyscale and min-max greyscale). The objects are the Sphere, Chequered Sphere and Skull, introduced in section 3.3.3. For the chequered sphere, the advantages of the Masked Binary representation will be lost below level 3, whereas the FTB and node-tracing algorithms would be expected to perform in a similar way to the sphere case. For the images in figure A.2 the top 44 of the 64 levels in the grey octree were thresholded, representing the hard tissue. This corresponded to the CT value ranges used to construct the corresponding binary tree (masked and unmasked). In figure A.2 the improvement in shading obtained from sub-voxel interpolation is apparent, as described in section 5.3.
A.4.1 Binary Results

Figures A.3 to A.5 show the average visits to Full, Empty and Partial octree nodes as a function of level, for the three objects in their binary representations. As expected there is a general reduction in visits in the order BTF unmasked, BTF masked, FTB and Node-Trace. The performance of FTB and Node-Trace is of course identical for masked and unmasked data. For the Node-Trace algorithm separate plots are made for the direct only visits, and for all visits. Some interesting observations can be made.

Firstly, as expected, all nodes are visited once for the unmasked BTF algorithm. In comparing figure A.3 for the digital sphere and figure A.4 for the Chequered Sphere, the FTB and node tracing algorithms appears similar in trend for Partial nodes, but slightly more heavily visited in the chequered case. The Node-Trace shows a noticeable increase in the chequered case at level 3, precisely the level of tessellation introduced by the checkerboard. At all levels, the expected reduction in visits for the different algorithms is observed.
Average Octree node visits

(•) Partial nodes; (x) Full nodes; (o) Empty nodes.
(- - -) Unmasked BTF algorithm; (---) Masked BTF algorithm; (•••) FTB;
(- - - -) Node-Tracer with only direct visits counted. The multiple visiting of high Partial
nodes is given in Table A.1

Figure A.3: The average visits to octree nodes for the binary sphere.
Average octree node visits

Legend - see figure A.3

Figure A.4 : The average visits to octree nodes for the Chequered Sphere.
Average Octree node visits (c)

Legend - see figure A.3

Figure A.5: The average visits to octree nodes for the binary skull.
When looking at the leaf nodes, it is found that in all cases that FTB improves on BTF and that the masked BTF improves on unmasked BTF. However for the Node-Tracer, as explained in section A.3.1, homogeneous nodes can record multiple visits, and this effect has been excluded from the FTB analysis. In fact, by inspection, the Full nodes are still visited at a comparable rate in FTB and node-tracing, whereas the Empty nodes are substantially more visited. This effect results from the fact that rays will terminate at the first Full node encountered in the Node-Tracer, whereas Empty nodes propagate them. When considering the "indirect" visits, it is found that low level Partial nodes do have substantial numbers of visits, but they are very much less than the worst case possibility derived in Corollary A.1. These indirect visits have not been shown because they do not fit into the scale used. Instead Table A.1 lists the average number of visits for Full, Partial, and Empty nodes for the three data cases. In figure 5.8c the results for the skull data are presented. Similar trends may be observed, though

<table>
<thead>
<tr>
<th>sphere</th>
<th>level</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Partial</td>
<td>49.00</td>
<td>47.500</td>
<td>20.286</td>
<td>7.662</td>
<td>3.214</td>
<td>1.389</td>
<td>0.543</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Full</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.113</td>
<td></td>
</tr>
<tr>
<td>Empty</td>
<td>-</td>
<td>7.643</td>
<td>-</td>
<td>5.769</td>
<td>1.799</td>
<td>1.009</td>
<td>0.669</td>
<td>0.549</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>chequered sphere</th>
<th>level</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Partial</td>
<td>48.50</td>
<td>47.000</td>
<td>13.786</td>
<td>6.338</td>
<td>3.073</td>
<td>1.589</td>
<td>0.817</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Full</td>
<td>-</td>
<td>-</td>
<td>0.500</td>
<td>0.444</td>
<td>0.660</td>
<td>0.230</td>
<td>0.221</td>
<td>0.262</td>
<td></td>
</tr>
<tr>
<td>Empty</td>
<td>-</td>
<td>7.643</td>
<td>17.750</td>
<td>5.769</td>
<td>1.799</td>
<td>1.009</td>
<td>0.669</td>
<td>0.549</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Skull</th>
<th>level</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Partial</td>
<td>157.25</td>
<td>102.48</td>
<td>30.26</td>
<td>1.09</td>
<td>3.524</td>
<td>1.045</td>
<td>0.364</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Full</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.108</td>
<td>0.083</td>
<td>0.077</td>
<td></td>
</tr>
<tr>
<td>Empty</td>
<td>-</td>
<td>16.67</td>
<td>17.308</td>
<td>4.055</td>
<td>1.946</td>
<td>0.904</td>
<td>0.387</td>
<td>0.209</td>
<td></td>
</tr>
</tbody>
</table>

Table A.1: The average visits to node for the node-tracing algorithm as a function of level. The figures count all visits – direct and indirect.
somewhat smoother, since there is less structure in the data, and so the "resonance" effect in the first two cases is not present.

Because the average node visit criterion is difficult to compare between algorithms, dependent as it is on exact implementation details, let us consider two other measures. Figures A.6 to A.8 show the percentage of nodes unvisited by each algorithm. Here the expected trends are exhibited strictly, for each data case. Even when both visit types are accounted for in the Node-Tracer, it is found that consistently less nodes are visited at all levels than for the FTB algorithm.

Lastly the average number of nodes visited for a pixel to be displayed was considered, at each level of the tree. The method of deriving this was as follows:

For level 0 to n :

For each leaf node $\Omega$ count $4^{(n \text{-level})}$ pixels with a total of $nvis(\Omega)$ visits where $nvis$ is the number of visits recorded during execution.

For each Partial node $\Omega$ add $(1/8) \times nvis(\Omega)$ visits to each child node $\Omega$.child[0-7]

This is a slightly fairer criterion, because for the case of homogeneous nodes behind Partial nodes, both the FTB and Node-Tracer will record visits to the quadtree nodes that are generated, whereas the octree node visits are reduced, perhaps artificially, in the FTB case. Figures A.9 to A.11 show the logarithm of the number of visits per pixel recorded as described. It is found that the FTB and Node-Tracing algorithms are very similar at low levels, with the Node-Tracer slightly better at the top level of the tree. When the indirect visits are included however the Node-Tracer is reduced in performance at all levels.

In order to evaluate the total effect, the total average visit per octree node, the total percentage of unvisited nodes and the total average visit per pixel were calculated. In addition, for the skull case, the analysis was performed for successively coarser resolutions of the octree, down to a level of 3. These results are presented in Table A.2.
(•) Partial nodes; (x) Full nodes; (o) Empty nodes.
(-----) Unmasked BTF algorithm; (---) Masked BTF algorithm; (···) FTB;
(-----) Node-Tracer counting direct visits; (—) Node-Tracer counting all visits.

Figure A.6: Percentage octree nodes unvisited for the binary sphere.
Legend - see figure A.6.

Figure A.7: Percentage octree nodes unvisited for the Chequered Sphere.
Legend - see figure A.6.

Figure A.8 : Percentage octree nodes unvisited for the binary skull.
<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Total average octree visits</th>
<th>Total average octree nodes unvisited</th>
<th>Total average visits per pixel</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BTFU</td>
<td>BTFM</td>
<td>FTB</td>
</tr>
<tr>
<td>level 3</td>
<td>1.00</td>
<td>1.00</td>
<td>0.94</td>
</tr>
<tr>
<td>level 4</td>
<td>1.00</td>
<td>0.98</td>
<td>0.79</td>
</tr>
<tr>
<td>level 5</td>
<td>1.00</td>
<td>0.95</td>
<td>0.58</td>
</tr>
<tr>
<td>level 6</td>
<td>1.00</td>
<td>0.88</td>
<td>0.39</td>
</tr>
<tr>
<td>level 7</td>
<td>1.00</td>
<td>0.81</td>
<td>0.26</td>
</tr>
<tr>
<td>level 8</td>
<td>1.00</td>
<td>0.75</td>
<td>0.21</td>
</tr>
<tr>
<td>level 3</td>
<td>0.00</td>
<td>0.00</td>
<td>0.06</td>
</tr>
<tr>
<td>level 4</td>
<td>0.00</td>
<td>0.02</td>
<td>0.21</td>
</tr>
<tr>
<td>level 5</td>
<td>0.00</td>
<td>0.05</td>
<td>0.42</td>
</tr>
<tr>
<td>level 6</td>
<td>0.00</td>
<td>0.12</td>
<td>0.61</td>
</tr>
<tr>
<td>level 7</td>
<td>0.00</td>
<td>0.19</td>
<td>0.74</td>
</tr>
<tr>
<td>level 8</td>
<td>0.00</td>
<td>0.25</td>
<td>0.79</td>
</tr>
<tr>
<td>level 3</td>
<td>0.00489</td>
<td>0.00488</td>
<td>0.00461</td>
</tr>
<tr>
<td>level 4</td>
<td>0.02399</td>
<td>0.02350</td>
<td>0.01888</td>
</tr>
<tr>
<td>level 5</td>
<td>0.11285</td>
<td>0.10772</td>
<td>0.06495</td>
</tr>
<tr>
<td>level 6</td>
<td>0.54553</td>
<td>0.47791</td>
<td>0.21139</td>
</tr>
<tr>
<td>level 7</td>
<td>2.44733</td>
<td>1.87321</td>
<td>0.62960</td>
</tr>
<tr>
<td>level 8</td>
<td>8.17665</td>
<td>6.11504</td>
<td>1.81234</td>
</tr>
</tbody>
</table>

BTFU - unmasked binary Back-to-Front
BTFM - masked binary Back-to-Front
FTB - Front-to-Back
Node-Trace - "direct" counts only the nodes tested against an octree node; "all" includes those nodes encountered in neighbour access.

Table A.2: Algorithm statistics for the skull in binary representation, at levels of resolution from 3 to 8.
A.4.2 Greyscale results

The detailed analysis has been given for the binary data forms. When dynamically thresholded, the grey octree would be expected to behave in a manner approaching the worst case of a binary octree, since there will be far fewer large homogeneous nodes (although see the example in section 3.3.3). However it is found that dramatic savings in performance can be achieved by the use of the min-max structure described in section 3.3.2. Effectively, low level Partial nodes can be regarded as Empty if the chosen threshold range is outside the minimum and maximum fields, which are set at build time by postorder traversal of all children. Table A.3 gives the total octree node visit, total unvisited percentage, and visit per pixel figures for data with and without the min-max representation. Most significantly it is found that less than 10% of the data is visited at all in the min-max representation, an improvement of a factor of eight over the simple greyscale representation.

The algorithm was extended to include interpolative depth shading, as described in section 5.2.2.2 and the improved shading leads to a noticeable reduction in block-like artifacts. The interesting result is that the extra interpolation contributes very little to the space and time complexity of the algorithm. These results are also included in Table A.3, together with the actual timings recorded.

A.5 DISCUSSION

Rather than just quoting timings, which are very implementation dependent, the index used here is visits/node which will be a property of the algorithm and the encoded object space.

In attempting to optimise, efficiency balances have to be made between run time operations and available memory. In the small memory system used in these studies, dramatic variations in timings with the size of different octree and quadtree representations have been found. Essentially, the bottleneck comes when virtual memory
Figure A.9: Logarithm of the average number of octree node visits per pixels for the binary sphere.

BTFU - Back-to-Front Unmasked; BTFM Back-to-Front Masked; FTB Front-to-Back; NTRP - Node Trace Projective (direct) visits; NTRA - Node Trace all visits;
BTFU - Back-to-Front Unmasked; BTFM Back-to-Front Masked; FTB Front-to-Back;
NTRP - Node Trace Projective (direct) visits; NTRA - Node Trace all visits;

Figure A.10: Logarithm of the average number of octree nodes visited per pixel, for
the Chequered Sphere
Log Average Visits per Pixel (c)

BTFU - Back-to-Front Unmasked; BTFM Back-to-Front Masked; FTB Front-to-Back; NTRP - Node Trace Projective (direct) visits; NTRA - Node Trace all visits;

Figure A.11: Logarithm of the average number of octree nodes visited per pixel, for the binary skull.
requirements start to page outside physical memory. A less significant bottleneck is the degree to which a separate cache memory fetch is efficiently used.

One of the specific advantages of the octree is that it is recursive by nature, which allows independent processors to work on different parts using the same algorithm without contention for shared memory. However the display process involves a projection of 3D space onto 2D space and may be thought of as decomposable in 2D rather than 3D.

The octree structure lends itself rather well to paging and/or cache memory fetch algorithms, because the hierarchical structure generally means that spatially local data is also encoded in physically local memory. This is certainly not true for example for a simple 3D array without in-pointers. It is particularly true of "octree-driven" algorithms (PTB, BTF or Boolean operations), and in a sense, the Node-Tracing algorithm presented here attempts to force an octree into an unnatural mode of access. Nevertheless it is certainly of interest to note the percentage of data not visited by the algorithm. In particular, for dealing with large amounts of greyscale data note that only a small percentage of the data is accessed at all.

In addition to implications in virtual-memory machines this has significance when parallel processor architectures are considered. Although outside the scope of this thesis, note that partitioning of data for 3D arrays may be either static [GOLDWASSER87] or dynamic, as for example in transputer arrays [MORRIS88]. For the latter strategy,

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Total octvis</th>
<th>Total unvis</th>
<th>vis/pix</th>
<th>Timing(secs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node-Trace-A (no min-max)</td>
<td>0.67</td>
<td>0.65</td>
<td>0.61</td>
<td>400.7</td>
</tr>
<tr>
<td>Node-Trace-A (min-max)</td>
<td>0.08</td>
<td>0.94</td>
<td>0.58</td>
<td>142.56</td>
</tr>
<tr>
<td>Node-Trace-B (min-max)</td>
<td>0.09</td>
<td>0.94</td>
<td>0.58</td>
<td>159.56</td>
</tr>
</tbody>
</table>

Table A.3: Algorithm statistics for the Node-Tracer on the greyscale octree representation of the skull. The additional overhead of the interpolative shading is about 10%. Visit count is for all visits - direct and indirect.
an ability to leave alone large parts of the data may lead to greatly increased efficiency.

In the results presented here, the FTB algorithm is generally higher in performance than any BTF algorithm. The Node-Tracing algorithm is not dramatically better, and in some cases worse. It is also likely to be more complex to implement, but when speed requirements are a priority, fine tuning of the algorithm can be expected to give improvements. The real significance of the algorithm comes from the ability to genuinely trace along a ray for the purpose of local neighbourhood operations.

The algorithms and analysis presented here apply only to orthogonal projections. The general view BTF and FTB algorithms were given in Chapter 4, but for FTB the more complex testing generally seems to offset the advantages. Certainly the author has found that FTB and BTF masked perform comparably in the general view projections, but certainly better than BTF unmasked. Extension of the Node-Tracer to general views has not been implemented. A trial implementation using a 3D Digital Differential Analyser was found to be quite slow. However, in dealing with large arrays of data, it may be that static views that allow sophisticated operations may be just as useful as fast rotation of simple objects. [TIEDE87] suggests that rotation and trilinear resampling of the data may be sufficient if another view is genuinely required. In addition, Yamaguchi et al [YAMAGUGHI84a] and Oliver [OLIVER86] note that certain other special views retain 2D coherence. It is obvious that a "pure" octree is only of use if large volumes are representable, i.e. if significant numbers of Full nodes occur at higher levels. A further area of study would be the use of hybrid tree structures, where the octree is used essentially as space-subdivision technique at low levels, and an alternative representation (e.g. a small 3D array) is used for the highest levels.

A.6 CONCLUSIONS

The principle purpose of this appendix was to analyse the Node-Tracing algorithm of section 4.4.3.3, in comparison with BTF and FTB algorithms.

The worst-case performance was derived and results presented for the performance on some artificial and realistic objects. Both binary and greyscale structures were discussed.
It seems that in some circumstances it may be advantageous to trace a quadtree into an octree directly. Apart from time savings, this also opens up additional techniques such as sub-voxel interpolation. In medical imaging it is a definite advantage to be able to use dynamic thresholding. The significant feature revealed in the results presented here is that, despite some at first sight unattractive overheads involved in octree representations, potentially very great savings may be available in the extent to which data is accessed. Thus use of min-max greyscale octrees may provide a powerful way of manipulating data in a compressed format suitable for a small minicomputer.
REFERENCES

[ABRAM86]

[AHUJA84]

[AGIN73]

[AMANS86]

[ARRIDGE85]

[ARRIDGE89]

[ARTZY81]

[ATKINSON86]

[AXEL83]

[BAJCSY83]

[BELL85]

[BHATIA84]

[BLOCH83]
[BOOKSTEIN79]
11, 123-137.

[BOYD83]
Boyd DP, Lipton MJ, 1983, Cardiac computed tomography. Proceedings of the
IEEE, 71(3), 298-307

[BRACCINI89]
Braccini G, Massimetti M, Salvetti O, 1989, 3D Sonographic reconstruction of
neonatal and infant hip, Computer Assisted Radiology 89 (eds. Lemke HU, Rhodes
ML, Jaffe CC, Felix R) Springer-Verlag (Berlin), 235-239.

[BREWSTER84]

[BRIGHT86]
5(2), 131-137.

[BUI-TUONG75]
of the ACM 18 311-317.

[BURK86]
Burk DL Jnr, Mears DC, Cooperstein LA, Herman GT, Udupa JK, 1986, Acetabular
fractures : three-dimensional computed tomographic imaging and interactive surgical
planning, J. Computer Assisted Tomography, 10, 1-10.

[BURKE83]
Burke PH, Banks P, Beard LFH, Tee JE, Hughes C 1983 Stereophotographic
measurement of change in facial soft tissue morphology following surgery. British

[CARLBOM85]
Carlbon I, Chakravarty I, Vanderschel D 1985 A Hierarchical Data Structure for
Representing the Spatial Decomposition of 3-D Objects. IEEE Computer Graphics
and Applications 24-31, 5(4).

[CGA83a]

[CGA83b]

[CHEN88]
Chen HH, Huang TS, 1988 A survey of construction and manipulation of octrees,

[CHEN84]
Chen L-S, Hung HM, Levkowitz H, Herman GT, Trivedi SS, Udupa JK, 1984,

[CHEN85]
Chen L-S, Herman GT, Reynolds RA, Udupa JK, 1985 Surface Shading in the

[CHIEN86]
Chien CH, Aggarwal JK, 1986 Volume/Surface Octrees for the Representation of
Three-Dimensional Objects. Computer Vision, Graphics, and Image Processing 36,
100-113.


[DONLON88]

[DRERUP80]

[DYER80]

[EASTMAN70]

[FANIBUNDA83]

[FARRELL85]

[FELLINGHAM86]

[FISHMAN87]

[FISHMAN89]

[FLYNN83]

[FOLEY82]

[FOX88]

[FREEMAN74]
Freeman H 1974 Computer processing of line drawing images Computer Surveys 6(1) 57-97.

[FREIDER85]
[FRENKEL89]

[FU81]
Fu KS, Mui JK, 1981 A survey on image segmentation Pattern Recognition 13 3-16.

[FUCHS77]

[FUCHS83]

[FUCHS88]

[FUJIMOTO86]

[FUJIMURA83]

[FUJIMURA85]

[GAMBOA-ALDECO86]

[GARGANTINI82a]

[GARGANTINI82b]

[GARGANTINI86a]

[GARGANTINI86b]

[GIBSON83]

[GLASSNER84]

[GOLDWASSER85]

[GOLDWASSER86a]
Goldwasser SM, Reynolds RA, Talton D, Walsh E, 1986 Real time display and manipulation of 3D CT PET and NMR data, Proceedings SPIE 671 139-149

[GOLDWASSER86b]
Goldwasser SM, Reynolds RA, Talton D, Walsh E, 1986 Real-time interactive facilities associated with a 3-D medical workstation, Proceedings SPIE 626 491-503

[GOLDWASSER87]

[GONZALEZ87]

[GORDON85]

[GORDON87]

[GORLIN76]

[GOURAUD71]

[GREGUSS77]

[HARRIS78]
Harris LD, Robb, RA, Yuen, TS, Ritman, El, 1978 Non-invasive numerical dissection and display of anatomic structure using computerized x-ray tomography, Proceedings SPIE, 152, 10-18

[HARRIS79]

[HARRIS84]
Harris LD, Camp JJ, 1984, Display and analysis of tomographic volumetric images utilizing a varifocal mirror, SPIE 507, 38.

[HARRIS86]

[HENDERSON74]
[HENDERSON85]

[HEFFERNAN85a]
Heffeman PB, Robb RA, 1985 *A new method for shaded surface display of biological and medical images*, IEEE Transactions on Medical Imaging 4, 26-38

[HEFFERNAN85b]

[HEMMY83]

[HEMMY86]
Hemmy, DC, David DJ, 1986, *Skeletal morphology of anterior encephaloceles defined through the use of three-dimensional reconstruction of computed tomography*, Paediatric Neuroscience, 12, 18-22.

[HEMMY87]

[HERMAN77]

[HERMAN78]

[HERMAN79]
Herman GT, Liu HK, 1979 *Three dimensional display of human organs from computed tomograms* Computer Graphics and Image Processing, 9, 1-21

[HERMAN80]
Herman, GT, Coin, CG, 1980 *The use of three dimensional computer display in the study of disk disease* J. Computer Assisted Tomography, 4, 564-567.

[HERMAN82]

[HERMAN83a]

[HERMAN83b]

[HERMAN86]

[HERMAN87]
Herman GT, Roberts D, Rabe B, 1987, *The reduction of pseudoforamina (false

[HERMAN88a]

[HERMAN88b]
Herman GT, 1988, *From 2D to 3D representations*, In: Mathematics and Computer Science in Medical Imaging (eds: Viergever MA, Todd-Pokropek A), NATO ASI Series F, 197-220.

[HERMAN88c]

[HÖHNE86]

[HÖHNE87a]

[HÖHNE87b]

[HÖHNE89a]

[HÖHNE89b]

[HOFFMAN85]
Hoffman EA, Ritman EL, 1985 *Shape and dimension of cardiac chambers : importance of CT section thickness and orientation*, Radiology 155, 739-744

[HOFFMAN89]

[HUEKEL71]

[HUEKEL73]

[HUNTER79]
[JACKEL85]
Jackel D, 1985 The graphics PARCUM system : a 3D memory based computer architecture for processing and display of solid models, Computer Graphics Forum 4(1), 21-32

[JACKINS80]

[KAJIKA79]

[KENNEDY87]

[KEPPEL75]

[KLINGER76]

[KLINGER79]
Klinger A, Rhodes ML, 1979, Organization and access of image data by areas, IEEE Pattern Analysis and Machine Intelligence, 1(1), 50-60.

[KO86]

[KRUGER84]

[KÜBLER87]

[KUNII85]

[KUO89]

[KURSOGLU86a]
Kursunoglu S, Pate D, Resnick D, Andre M, Sartoris DJ, 1986, Computed arthrotomography with multiplanar reformatations and three dimensional image analysis in the evaluation of the cruciate ligaments : preliminary investigations. J.Canadian Association of Radiologists, 37, 153-156.

[KURSOGLU86b]
Kursunoglu S, Kaplan P, Resnick D, Sartoris DJ, 1986 Three dimensional computed

[LAFFERTY86]

[LEE83]

[LENZ84]

[LENZ86]

[LEVOY88]

[LINNEY89]

[LIPTON89]

[LIU77]
Liu HK 1977 Two and Three dimensional boundary detection Computer Graphics and Image Processing 6 123-134

[LOBREHT80]

[MAO87]

[MARR80]

[McEWAN89]

[McVEY82]
[MEAGHER82]

[MEAGHER84a]
Meagher DJ, 1984 Interactive Solid Modeling. Proceedings of CADCON West. (February)

[MEAGHER84b]

[MEAGHER84c]

[MEAGHER84d]

[MEAGHER85a]

[MEAGHER85b]

[MOORE85]

[MORRIS88]

[MORGENTHALER81]

[MOSS87]

[MOSS88]

[NEWMAN81]

[NORMAN88]

[OLIVER83a]
Oliver MA, Wiseman NE, 1983 Operations on Quadtree Encoded Images. The

[OLIVER83b]

[OLIVER85]

[OLIVER86]

[PANDER78]

[PARKER86]

[PARKER83]

[PATE86]

[PATHRIA87]

[PIZER86]

[PIZER89]

[POMMERT89a]

[POMMERT89b]

[POTMESIL87]
Potmesil M, 1987 Generating Octree Models of 3D Objects from their Silhouettes

[PROCIEEE83]
Proceedings of the IEEE, 1983, Special Issue on Medical Imaging, Proceedings of the IEEE, 71(3).

[PROC3D89]

[PROCVVW89]
Proceedings of Volume Visualization Workshop (Chapel Hill, NC, May 18-19), (ed : Upson C), Dept. of Computer Science, University of North Carolina, Chapel Hill, NC.

[RABEY77]

[RAYA88]
Raya SP, Udupa JK, 1988, Shape-based interpolation of Multidimensional objects, MIPG technical report, MIPG129.

[REDPATH75]
Redpath AT, Vickery SL, Wright DW, 1975, A set of FORTRAN subroutines for optimizing radiotherapy plans, Comp. Prog. Biomed. 5, 158.

[REED84]

[REYNOLDS85]

[REYNOLDS87]

[RHODES79]

[RHODES87]

[RHODES89]

[RICKETTS72]

[ROBB83]
Robb RA, Hoffman EA, Sinak LJ, Harris LD, Ritman EL, 1983 High-speed three-

[ROBB87]

[ROBB88]

[ROBB89]

[ROGERS87]

[ROSENFELD76]

[ROSENMAN89]

[ROTHMAN86]

[SAMET84a]

[SAMET84b]

[SAMET88a]

[SAMET88b]

[SARTORIS86a]

[SARTORIS86b]
Sartoris DJ, Andre M, Resnick C, Resnick D, 1986 Trabecular bone density in the

[SARTORIS86c]

[SATO82]

[SCOTT87]

[SCHIERS89]

[SCHULTZ78]

[SCHMITZ89]

[SCHRZE86]

[SHIRAC71]

[SRIHARI81]

[SRIHARI84]

[STAUDHAMMER86]

[STIEHL85]

[SUCKIEL78]
[SUGUWARA80]

[SUNGUROFF78]

[SUTHERLAND74]

[TESSIER86]

[TIAN86]

[TIEDE87]

[TOENNIES89]

[TOTTY84]

[TRIVEDI85]

[TRIVEDI86a]

[TRIVEDI86b]

[TRIVEDI86c]

[TROUSSSET88]

[TUY84]

[UCL3D87]
Department.

[UDUPA81]

[UDUPA82a]

[UDUPA82b]

[UDUPA83]

[UDUPA86a]
Udupa JK, 1986 Computerised surgical planning: current capabilities and medical needs, Proceedings SPIE 626, 474-482

[UDUPA86b]

[UDUPA87]

[UPSON89]

[VANNIER83a]

[VANNIER83b]

[VANNIER83c]

[VANNIER84]

[VANNIER87]

[VANWIJK80]
Van Wijk, MC 1980 Moire Contourgraphs - An Accuracy Analysis 14th Congress
of the International Society of Photogrammetry, Hamburg 824-833.

[WEISBURNA86]

[WENG87]

[WHITTED80]

[WILLMOT81]
Willmot DR, 1981 Soft tissue profile changes following correction of class III malocclusions by mandibular surgery, Br J Orthodontics, 8, 175-181

[WILSON84]

[WOLFE88]

[WOOD85]

[WOODWARK82]

[WOLSON86]

[WU84]

[Wyvill86]

[YAMAGUCHI84a]

[YAMAGUCHI84b]

[YAU83]

[YAU84]
[ZINREICH86]

[ZONNEWELD89]

[ZUCKER76]

[ZUCKER82]
### INDEX

<table>
<thead>
<tr>
<th>Term</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-adjacent</td>
<td>44</td>
</tr>
<tr>
<td>0-voxel</td>
<td>55</td>
</tr>
<tr>
<td>1-adjacent</td>
<td>44</td>
</tr>
<tr>
<td>1-voxel</td>
<td>55</td>
</tr>
<tr>
<td>2.5D displays</td>
<td>31</td>
</tr>
<tr>
<td>2-adjacent</td>
<td>44</td>
</tr>
<tr>
<td>3D binary array</td>
<td>72</td>
</tr>
<tr>
<td>3D digital array</td>
<td>71</td>
</tr>
<tr>
<td>3D greyscale array</td>
<td>71</td>
</tr>
<tr>
<td>8-adjacency</td>
<td>48</td>
</tr>
<tr>
<td>access random</td>
<td>66, 74, 87, 100, 210</td>
</tr>
<tr>
<td>access sequential</td>
<td>66, 100</td>
</tr>
<tr>
<td>B-Rep</td>
<td>20</td>
</tr>
<tr>
<td>back to front</td>
<td>223</td>
</tr>
<tr>
<td>Background</td>
<td>55</td>
</tr>
<tr>
<td>Binarisation</td>
<td>48, 49</td>
</tr>
<tr>
<td>Binary</td>
<td>66</td>
</tr>
<tr>
<td>Binary scene</td>
<td>46</td>
</tr>
<tr>
<td>Border</td>
<td>56</td>
</tr>
<tr>
<td>Boundary</td>
<td>56</td>
</tr>
<tr>
<td>CEMAX</td>
<td>28</td>
</tr>
<tr>
<td>Chain code</td>
<td>70</td>
</tr>
<tr>
<td>Characteristic function</td>
<td>42</td>
</tr>
<tr>
<td>discrete</td>
<td>44</td>
</tr>
<tr>
<td>Child</td>
<td>86, 89</td>
</tr>
<tr>
<td>Clipping</td>
<td>96</td>
</tr>
<tr>
<td>Co-child</td>
<td>86</td>
</tr>
<tr>
<td>Computer Assisted Tomography</td>
<td></td>
</tr>
<tr>
<td>Concatenation</td>
<td>23</td>
</tr>
<tr>
<td>Condense</td>
<td>91</td>
</tr>
<tr>
<td>Connectivity</td>
<td>37, 44, 52, 77, 212</td>
</tr>
<tr>
<td>Continuous scene</td>
<td>41</td>
</tr>
<tr>
<td>Continuous screen</td>
<td>46</td>
</tr>
<tr>
<td>Continuous Universe</td>
<td>41</td>
</tr>
<tr>
<td>Contour</td>
<td>32, 58, 67</td>
</tr>
<tr>
<td>Contour-following</td>
<td>58</td>
</tr>
<tr>
<td>CSG</td>
<td>20</td>
</tr>
<tr>
<td>Cuberille</td>
<td>33, 55</td>
</tr>
<tr>
<td>Density</td>
<td>35</td>
</tr>
<tr>
<td>Density function</td>
<td>42</td>
</tr>
<tr>
<td>Density window</td>
<td>49</td>
</tr>
<tr>
<td>Depth image</td>
<td>99</td>
</tr>
<tr>
<td>Depth preserving</td>
<td>99</td>
</tr>
<tr>
<td>Difference of Gaussian</td>
<td>61</td>
</tr>
<tr>
<td>Digital discrete scene</td>
<td>46</td>
</tr>
<tr>
<td>Digital image</td>
<td>48</td>
</tr>
</tbody>
</table>
Digital point ................................................. 43
Digitisation .................................................. 43, 44
Directed contour ............................................ 55
Discrete scene ............................................. 46
Discrete Universe .......................................... 98
Discrete vector ............................................. 44
Display ........................................................ 99
  image driven .............................................. 100
  object driven .............................................. 100
Displayed image .......................................... 99
Dissection .................................................... 36
Dynamic allocation ....................................... 72
Dynamic Spatial Reconstructor ...................... 29
Edge-detection .............................................. 54, 58
Edge_length .................................................. 86
Facet .......................................................... 58
Feature Space Clustering ................................. 63
front to back ............................................... 223
Geophysics ................................................... 41
Grey octree .................................................... 80
Grey scene .................................................... 46
Greyscale ..................................................... 66
Greyscale octree ............................................ 80
Hidden Surface Elimination ......................... 78, 95
Highlight ..................................................... 129
Homogeneous coordinate transformation ........ 97
Hounsfield Units .......................................... 53
Image ........................................................ 48, 95
Image Driven ............................................... 100
Image-Space ............................................... 95
  origin ....................................................... 97
Integer triple ............................................... 43
Interpolation ................................................. 36, 48
K-tree .......................................................... 76
  complexity theorem for ............................... 90
Leafcode ...................................................... 87
Leafnode ...................................................... 76
Line-segments .............................................. 75
Linear octree ............................................... 89
Magnetic Resonance Imaging ....................... 23
Malocclusion .............................................. 189
mandibular .................................................... 190
retrusion ...................................................... 190
Mandibular prognathism ................................. 189
Manipulation .............................................. 19, 34, 209
Marginal Indexing ......................................... 73
Maximal ...................................................... 56
Mayo clinic ............................................... 29
Meagher ...................................................... 29
Merging ................................................................. 36
MIPG ............................................................... 28
Mode-rule ........................................................... 36
Movie ................................................................. 30
Multi-Valued ......................................................... 66
Multiplanar reformatting ......................................... 30
Multiresolution segmentation .................................... 64
Multivalued digital array ......................................... 72
N-component ......................................................... 56
N-connected ......................................................... 55
N-mask ............................................................... 80
N-neighbourhood .................................................. 77
N-object ............................................................. 56
node

colour ................................................................. 77
internal .............................................................. 76
leaf ................................................................. 76
Nuclear Magnetic Resonance ................................. 23
Object ............................................................... 31, 36, 42
digital ................................................................. 44
hard ............................................................... 42
soft ................................................................. 43
surface representation ............................................ 31
volume representation ........................................... 31
Object Driven ......................................................... 100
Object-space ........................................................ 42
origin .............................................................. 42
Octal code .......................................................... 85
Octal path ............................................................ 85
element ............................................................ 85
fragment ............................................................ 85
Octant ............................................................... 86
origin .............................................................. 86
octree
creation ............................................................. 89
explicit ............................................................ 88
greyscale .......................................................... 80
leafcode ........................................................... 87
linear .............................................................. 87, 89
Masked Binary ...................................................... 80
Min-Max ............................................................ 81
pointer based ....................................................... 89
pyramid ............................................................ 80
raster .............................................................. 76, 117
root ................................................................. 76
top of .............................................................. 76
treecode ........................................................... 87
vector .............................................................. 76, 117
Octrees ............................................................. 117
<table>
<thead>
<tr>
<th>Term</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Opacity</td>
<td>109</td>
</tr>
<tr>
<td>Orthogonal</td>
<td>99</td>
</tr>
<tr>
<td>Orthographic projection</td>
<td>96</td>
</tr>
<tr>
<td>Osteotomy</td>
<td>190</td>
</tr>
<tr>
<td>body</td>
<td></td>
</tr>
<tr>
<td>condylotomy</td>
<td>189</td>
</tr>
<tr>
<td>genioplasty</td>
<td>190</td>
</tr>
<tr>
<td>inverted L</td>
<td>189</td>
</tr>
<tr>
<td>Kufner</td>
<td>189, 190</td>
</tr>
<tr>
<td>Le Fort I</td>
<td>186, 189</td>
</tr>
<tr>
<td>Le Fort II</td>
<td>189</td>
</tr>
<tr>
<td>Le Fort III</td>
<td>186, 189</td>
</tr>
<tr>
<td>sagittal split</td>
<td>189</td>
</tr>
<tr>
<td>subcondylar</td>
<td>189</td>
</tr>
<tr>
<td>Partial volume</td>
<td>45, 64, 81, 139, 146</td>
</tr>
<tr>
<td>Pivot level</td>
<td>92</td>
</tr>
<tr>
<td>PIXAR</td>
<td>29</td>
</tr>
<tr>
<td>Pixel</td>
<td>46</td>
</tr>
<tr>
<td>PIXEL-PLANES</td>
<td>69</td>
</tr>
<tr>
<td>plane</td>
<td></td>
</tr>
<tr>
<td>coronal</td>
<td>30</td>
</tr>
<tr>
<td>sagittal</td>
<td>30</td>
</tr>
<tr>
<td>transverse</td>
<td>25</td>
</tr>
<tr>
<td>Point</td>
<td>42</td>
</tr>
<tr>
<td>Point location</td>
<td>91</td>
</tr>
<tr>
<td>Point Spread Function</td>
<td>50</td>
</tr>
<tr>
<td>Positron Emission Tomography</td>
<td>23</td>
</tr>
<tr>
<td>Post-order</td>
<td>80, 91</td>
</tr>
<tr>
<td>Projection transform</td>
<td>99</td>
</tr>
<tr>
<td>Proper subset</td>
<td>56</td>
</tr>
<tr>
<td>PSF</td>
<td>50</td>
</tr>
<tr>
<td>Pyramid</td>
<td>77, 80</td>
</tr>
<tr>
<td>Quad_code</td>
<td>87</td>
</tr>
<tr>
<td>Quadrant</td>
<td>87</td>
</tr>
<tr>
<td>Quadtree</td>
<td>76</td>
</tr>
<tr>
<td>explicit</td>
<td>88</td>
</tr>
<tr>
<td>linear</td>
<td>87</td>
</tr>
<tr>
<td>pointer based</td>
<td>89</td>
</tr>
<tr>
<td>Quantisation</td>
<td>45</td>
</tr>
<tr>
<td>Quantitation</td>
<td>19</td>
</tr>
<tr>
<td>Radiation Treatment Planning</td>
<td>97, 183</td>
</tr>
<tr>
<td>Ramus</td>
<td>189</td>
</tr>
<tr>
<td>Raster octrees</td>
<td>76, 117</td>
</tr>
<tr>
<td>Ray</td>
<td>99</td>
</tr>
<tr>
<td>Ray-casting</td>
<td>34</td>
</tr>
<tr>
<td>Ray-step</td>
<td>99</td>
</tr>
<tr>
<td>Ray-tracing</td>
<td>34</td>
</tr>
<tr>
<td>Recursive BTF</td>
<td>106</td>
</tr>
<tr>
<td>Region</td>
<td>41</td>
</tr>
</tbody>
</table>
Region splitting .................................................. 62
Region-growing .................................................. 212
Regions of Interest .............................................. 67
Registration ......................................................... 216
  volume-based .................................................. 220
Representation ................................................... 20, 41
  B-Rep ............................................................ 20
  CSG .............................................................. 20
  greyscale ....................................................... 210
  Parametric ...................................................... 20
  surface-based .................................................. 52
  Symmetric recursive ......................................... 76
  volume-based .................................................. 52, 210
  Voxel ............................................................. 20
Reprojection ..................................................... 34
Root ................................................................. 76
Scene ............................................................... 31
Scene complexity ............................................... 90
Screen ............................................................. 95
Screen point ...................................................... 46
Segmentation ..................................................... 31, 42, 48, 52, 210
Shading
  Cosine ............................................................ 130
  depth .............................................................. 130
  image-based ................................................... 130
  object-based .................................................. 78, 130, 210
  Partial Volume ................................................ 140
  surface .......................................................... 95
Silhouette image ................................................. 99
Single Photon Emission Computed Tomography .......................... 23
Skeletonisation .................................................. 56
Slice ................................................................. 25, 48
Spawn .............................................................. 91
Split ................................................................. 91
Split and merge .................................................. 62
Status ............................................................... 76
Step ................................................................. 99
Sub-regioning ................................................... 48
Surface normal .................................................. 129
Surface rendering .............................................. 31
Thresholding ..................................................... 49
  dynamic ........................................................ 53
Tile ................................................................. 58
Tiling ................................................................. 68, 213
Top-down ........................................................ 91
Topological ....................................................... 52
Transaxial dimension ......................................... 25
Transform ......................................................... 163
Transformation ................................................... 36
depth preserving ...................................................................................................... 99
Treecode ...................................................................................................................... 87, 89
Value .............................................................................................................................. 46
Vannier and Marsh ...................................................................................................... 28
Varifocal mirror .......................................................................................................... 31
Vector ............................................................................................................................ 42
Vector octrees ........................................................................................................... 76, 117
Virtual children ........................................................................................................... 91
virtual octants ................................................................................................................. 123
Visualisation .............................................................................................................. 19, 209
Volume Compositing ...................................................................................................... 108
Volume rendering ......................................................................................................... 31
Voxel .......................................................................................................................... 20, 21, 43
  continuous .................................................................................................................... 44
Walk down and up ...................................................................................................... 92
Walk up from pivot ...................................................................................................... 92
Wave mechanics ........................................................................................................... 41
Windowing
  density ......................................................................................................................... 25
Z-buffer image .............................................................................................................. 99