



# University of HUDDERSFIELD

## University of Huddersfield Repository

Xu, Qian

Interactive Volume Deformation Based on Model Fitting Lattices

### Original Citation

Xu, Qian (2012) Interactive Volume Deformation Based on Model Fitting Lattices. Doctoral thesis, University of Huddersfield.

This version is available at <http://eprints.hud.ac.uk/id/eprint/17820/>

The University Repository is a digital collection of the research output of the University, available on Open Access. Copyright and Moral Rights for the items on this site are retained by the individual author and/or other copyright owners. Users may access full items free of charge; copies of full text items generally can be reproduced, displayed or performed and given to third parties in any format or medium for personal research or study, educational or not-for-profit purposes without prior permission or charge, provided:

- The authors, title and full bibliographic details is credited in any copy;
- A hyperlink and/or URL is included for the original metadata page; and
- The content is not changed in any way.

For more information, including our policy and submission procedure, please contact the Repository Team at: [E.mailbox@hud.ac.uk](mailto:E.mailbox@hud.ac.uk).

<http://eprints.hud.ac.uk/>

# **Interactive Volume Deformation Based on Model-Fitting Lattices**

**BSc. Qian Xu**

A thesis submitted to the University of Huddersfield in partial  
fulfilment of the requirements for the degree of Doctor of Philosophy

School of Computing and Engineering  
University of Huddersfield

Jun 2012

## **Acknowledgements**

I would like to thank the School of Computing and Engineering at the University of Huddersfield for providing this great opportunity of study and facilitating me throughout this project. I wish to thank my colleagues at the Computer Graphics, Imaging and Vision (CGIV) Research Group within the University of Huddersfield for their continuous and consistent help and support to the project and myself.

I would like to express my sincere gratitude to my director of studies, Dr Zhijie Xu, for his exceptional support and guidance throughout this project. He was willing to take a chance on my research from the beginning, and has always pushed me to fill in that one last detail to elevate the level of my thinking and my works.

A great deal of consideration and thanks must go to my family. My parents, Yonghan Xu and Yan Ping, continue to be my role models for living life with passion, creativity, and hard work. They have sacrificed many days without me, yet all of this would be for nothing without them.

## Abstract

Volume visualization, which is a relatively new branch in scientific visualization, not only displays surface features of a model, but enables an intuitive presentation of the internal information of the object. Its comprehensive visualization algorithms developed in the last decade have brought in challenges such as complex data processing, real-time operations, and application-specific system performances. These challenges were elaborated in the manner of research objectives in the thesis.

By devising a novel volume deformation pipeline, this thesis managed to explore volume-model-related operations applied for complicated applications through illustrating the feasibility of the designed system that was verified by experimental results. The contribution of the programme was demonstrated via testifying the effectivities of the four system design characteristics. Firstly, the clustering-based segmentation methods were adopted by the volumetric data processing module within the proposed volume deformation system for managing the complicated structures often existing in large volume data sets. Secondly, a novel mesh construction method was formulated in terms of optimizing the control lattices for the following deformation process. Thirdly, the volume deformation approach devised in the research has taken advantages of the parameterization process of the entire shape-change process. Finally, the GPU-based parallel process architecture was utilized to accelerate the calculation of Gaussian sampling in the lattice construction process; the progressive locations of the removed points in the simplification scheme; and the integration of kinetic energy for determining the deformation behaviours.

## **List of Publications**

Xu, Q., D. Gledhill, et al. (2011). "Volume deformation based on model-fitting surface extraction." In: Proceedings of the 17th International Conference on Automation and Computing: 175-180.

Xu, Q., M. Holder, et al. (2010). "Improved iso-surface extraction for hybrid rendering application." In: Proceedings of the 16th International Conference on Automation and Computing: 96-101.

Xu, Q., Z. Xu, et al. (2009). "Predicting specific gravity and viscosity of level-of-detail control in hybrid rendering application." In: Proceedings of Computing and Engineering Annual Researches' Conference 2009: 76-81.

Xu, Q. and Z. Xu (2009). "A hybrid rendering framework for the real-time manipulation of volume and surface models." In: Proceedings of the 15th International Conference on Automation and Computing: 160-165.

## List of Figures

Figure 1.1	An engine box represented via different modes. ....	2
Figure 1.2	A CT scan of a human brain .....	3
Figure 1.3	A diagram of the design system pipeline.....	9
Figure 2.1	Various applications of volumetric information.....	12
Figure 2.2	Illustrations of volume model formations.....	13
Figure 2.3	Diagrams of forward-mapping and backward-mapping methods .....	15
Figure 2.4	Shading results determined by different optical models.....	16
Figure 2.5	A diagram of a viewing ray for volume rendering integral .....	18
Figure 2.6	A diagram of tri-linear interpolation.....	20
Figure 2.7	Diagrams of colour accumulation and approximated rendering integral.....	22
Figure 2.8	The results of ray-casting.....	22
Figure 2.9	Diagrams of pre-, post- and pre-integrated classifications .....	24
Figure 2.10	The results of object-aligned slices.....	27
Figure 2.11	Results of view-aligned slices. ....	29
Figure 2.12	A diagram of the shear warp model with the parallel projection mode.....	30
Figure 2.13	A diagram of interpolating scan-lines.....	32
Figure 2.14	A diagram of shear warp algorithm with the perspective projection mode .....	32
Figure 2.15	A diagram of MIP .....	34
Figure 2.16	Results of MIP .....	35
Figure 2.17	Illustrations of MIP-based and LMIP-based imaging methods .....	36
Figure 2.18	Various shading effects .....	37

Figure 2.19	Results of first hit approach for the human head data set.....	38
Figure 2.20	A diagram of deferred shading approach.....	39
Figure 2.21	A diagram of deferred ambient occlusion approach.....	39
Figure 2.22	A result of volume scattering.....	40
Figure 2.23	A diagram of a shading composition design.....	41
Figure 3.1	A diagram of the clipping method via tagged volumes.....	44
Figure 3.2	A diagram of the depth-based volume clipping method.....	45
Figure 3.3	Illustrations of surface deformation models.....	45
Figure 3.4	Illustrations of the ray-deflector volume deformation method.....	47
Figure 3.5	Results of the spatial TF method.....	47
Figure 3.6	Illustrations of the warping-based volumetric representation method.....	49
Figure 3.7	A diagram of a physics-based deformation method for medical simulations...	49
Figure 3.8	A diagram of Chain-Mail algorithm.....	51
Figure 3.9	Results of Chain-Mail-based deformation.....	51
Figure 3.10	Diagrams of FEM-based approximate representation.....	52
Figure 3.11	Illustrations of FEM-based volume deformation.....	53
Figure 3.12	A diagram of Mass-spring system.....	53
Figure 3.13	Illustrations of a volume deformation based on Mass-spring system.....	54
Figure 3.14	Diagrams of CVG-based Boolean operations.....	55
Figure 3.15	Results of CVG-based Boolean operations.....	55
Figure 3.16	Illustrations of a dynamic CT baby data.....	56
Figure 3.17	Illustrations of a volumetric particle system.....	57
Figure 3.18	Illustrations of CFD simulations based on volume animation techniques.....	57

Figure 3.19	The comparison table of Chain-Mail, FEM and Mass-spring performances....	59
Figure 4.1	Display of a CT-scanned human head data. ....	64
Figure 4.2	The results of HVS. ....	67
Figure 4.3	The results of EMVS. ....	70
Figure 4.4	The results of KMVS. ....	72
Figure 4.5	The results of MSVS. ....	80
Figure 4.6	Diagrams of ATF design and the result of analysing a CT-scanned data.....	83
Figure 4.7	Energy $E_{x,y}$ in active contour algorithm versus the number of iteration .....	85
Figure 4.8	Illustrations of active contour algorithm.....	86
Figure 4.9	Illustrations of a domain of interest initialized in active surface algorithm .....	86
Figure 4.10	Results of active surface algorithm in volume segmentation .....	89
Figure 5.1	Results of constructed lattices based on the segmented information.....	91
Figure 5.2	Diagram of sampling process in MC algorithm.....	93
Figure 5.3	Diagram of the look-up table for surface-edge intersection .....	94
Figure 5.4	Results of extracted iso-surfaces. ....	94
Figure 5.5	Results of extracted iso-surfaces .....	96
Figure 5.6	Results of decreasing sampling rate .....	98
Figure 5.7	Illustrations of adaptive subdivision method.....	99
Figure 5.8	Illustrations of vertex decimation method .....	100
Figure 5.9	Illustrations of vertex merging method.....	101
Figure 5.10	Illustrations of vertex adaptive subdivision solution .....	104
Figure 5.11	Illustrations of deformed control lattices .....	104
Figure 6.1	Pipeline of deformation module .....	107



Figure 6.2	Random shape changes on the extracted lattice (in 2D and 3D) .....	109
Figure 6.3	Mesh-based mass-spring system.....	111
Figure 6.4	Diagrams of decomposing the applied force in mass-spring system.....	113
Figure 6.5	Results of lattice deformation with mass-spring system. ....	114
Figure 6.6	Results of lattice deformation with different stiffness coefficients .....	116
Figure 6.7	Diagram of mapping process deployed in the DOGME.....	117
Figure 6.8	Diagram of I-DOGME.....	118
Figure 6.9	Diagram of improved mapping process deployed in the I-DOGME.....	120
Figure 6.10	Results of volume deformation with different stiffness values .....	120
Figure 6.11	Results of clipped results after the mapping process.....	121
Figure 6.12	Diagram of relationships between control vertices and underlying voxels ....	121
Figure 6.13	Diagram of creating the octree-based lookup function.....	122
Figure 6.14	Diagram of using the octree-based lookup function.....	122
Figure 6.15	Diagram of the arrangement of 8 partitions.....	123
Figure 6.16	Diagram of an digital nested relationship to present the octree structure.....	123
Figure 6.17	Diagram of generating internal relationship for indexing operations.....	126
Figure 6.18	Results of volume deformation with different parameter settings.....	126
Figure 6.19	Results of fixed volume deformation .....	127
Figure 7.1	Diagram of hierarchical structure in CUDA programming model .....	133
Figure 7.2	Diagram of CUDA-based volume rendering pipeline .....	134
Figure 7.3	Illustrations of cubic sampling region .....	136
Figure 7.4	Diagram of CUDA-based lattice construction.....	139
Figure 7.5	Diagram of triangulizing polygons.....	143

Figure 7.6	Diagram of octree data structure and the displacement mapping design.....	145
Figure 7.7	Results of octree-based lookup function.....	146
Figure 8.1	Results of DVR with a 2D TF and the ATF design.....	152
Figure 8.2	Results of the volume data processing module.....	154
Figure 8.3	Results of extracting lattices from the isolated segments .....	156
Figure 8.4	Results of non-physics-based deformation .....	157
Figure 8.5	Results of the IVD method .....	158
Figure 8.6	Comparative results of different TF designs .....	159
Figure 8.7	Comparative results of different implementations.....	160
Figure 8.8	Comparative results of different deformation solutions .....	161
Figure 8.9	Results of deformed behaviours resulting from lattice simplification levels .	161
Figure 8.10	Comparative results of different lattice simplification levels.....	162
Figure 8.11	Various results of IVD method.....	163
Figure 8.12	Comparison table showing performances of IVD, ABS and CIVD .....	164

## List of Tables

Table 4.1	Mechanism of HVS .....	67
Table 4.2	Mechanism of EMVS .....	69
Table 4.3	Mechanism of KMVS.....	72
Table 4.4	Comparison between processing times.....	73
Table 4.5	Comparison among performances of processing different data .....	74
Table 4.6	Comparison among three kinds of clustered results. ....	75
Table 4.7	Comparison between the results of EMVS and KMVS .....	76
Table 4.8	Mechanism of MSVS .....	80
Table 4.9	Mechanism of active surface algorithm.....	88
Table 5.1	Comparisons between results of decimation and subdivision solutions.....	102
Table 5.2	Comparison between results of decimation and subdivision solutions .....	103
Table 6.1	Mechanism of vertex displacement calculation.....	117
Table 6.2	Mechanism for locating voxel's sequence number.....	119
Table 6.3	Mechanism for locating grid nodes .....	125
Table 8.1	Results of using KMVS and MSVS to process different data sets.....	153

## List of Abbreviations

1/2/3D	one-/two-/three-dimensional
ATF	Automatic Transfer Function
BRDF	Bidirectional Reflectance Distribution Function
CFD	Computational Fluid Dynamics
CSG	Constructive Solid Geometry
CT	Computer Tomography
CUDA	Compute Unified Device Architecture
CVG	Constructive Volume Geometry
DIP	Digital Image Processing
DOGME	Deformation of Geometric Models Editor
DVR	Direct Volume Rendering
EM	Expectation-maximization
EMVS	Expectation-maximization algorithm-based Volume Segmentation
FEM	Finite Element Method
FFD	Free-Form-Deformation
HVS	Hierarchical clustering-based Volume Segmentation
I-DOGME	Improved Deformation of Geometric Models Editor
IDVR	Indirect Volume Rendering

IIR	Indexable Inherent Relationship
IVD	Interactive Volume Deformation
KM	K-means
KMVS	K-means clustering-based Volume Segmentation
LUT	Lookup Table
MC	Marching-Cubes Algorithm
MIP	Maximum Intensity Projection
MRI	Magnetic Resonance Imaging
MS	Mean-shift
MSVS	Mean-shift clustering-based volume Segmentation
PC	Personal Computer
PDE	Partial Differential Equation
SIMD	Single Instruction, Multiple Data
SIMT	Single Instruction, Multiple Threads
TF	Transfer Function

# Table of Contents

Acknowledgements .....	I
Abstract .....	II
List of Publications.....	III
List of Figures .....	IV
List of Tables.....	IX
List of Abbreviation .....	X
Table of Contents .....	XII
Chapter 1 Introduction.....	1
1.1 Volume Data Acquisition.....	3
1.2 Volume Visualization and Deformation.....	5
1.3 Research Objectives .....	6
1.4 Contributions to Knowledge.....	9
1.5 Thesis Outlines .....	11
Chapter 2 Review on Volume Visualization Approaches .....	12
2.1 DVR Optical Model .....	15
2.2 Ray-casting Theory and Practice.....	16
2.2.1 Volume Rendering Integral Model.....	17
2.2.2 Tri-linear Interpolation.....	19
2.2.3 Alpha Blending Operation.....	20
2.2.4 Classification Process .....	23

2.3	Texture-mapping Approaches.....	27
2.4	The Shear-warp Model .....	30
2.4.1	Parallel Projection Algorithm .....	30
2.4.2	Perspective Projection Algorithm.....	32
2.5	The Maximum Intensity Projection .....	34
2.6	Volume Shading Techniques .....	36
2.6.1	Monte-Carlo Techniques for Iso-surface.....	37
2.6.2	Volume Scattering.....	40
2.7	Research Objectives .....	41
Chapter 3 Volume Model Manipulation Strategies.....		43
3.1	Volume Clipping.....	43
3.2	Volume Deformation .....	45
3.2.1	Non-physics-based Deformation.....	45
3.2.2	Physics-based Deformation .....	49
3.3	Constructive Volume Geometry.....	54
3.4	Volume Animations.....	56
3.5	Analysis on Current Challenges.....	58
3.6	Design Criteria.....	60
Chapter 4 Volumetric Data Processing .....		63
4.1	Applying Clustering Methods for Classifying Volume Data.....	64
4.1.1	Hierarchical Clustering-based Volume Segmentation (HVS).....	65
4.1.2	Expectation-Maximization Algorithm-based Volume Segmentation (EMVS) .....	67

4.1.3	K-means Clustering-based Volume Segmentation (KMVS) .....	70
4.1.4	Evaluations of Segmentation Approaches .....	73
4.2	Segmentation Improvement and Cluster Representation.....	78
4.2.1	Mean-Shift Clustering-based Volume Segmentation (MSVS).....	78
4.2.2	Design of Automatic Transfer Function (ATF) .....	82
4.3	Design of Boundary Extraction.....	83
4.3.1	Active Contour Algorithm.....	84
4.3.2	Region-based Active Surface Method .....	86
4.4	Summary .....	89
Chapter 5 Lattice Construction and Refinement .....		91
5.1	Marching Cubes Algorithm .....	92
5.1.1	Sampling and Vertex Extraction Process .....	92
5.1.2	Triangulation Process .....	93
5.1.3	Automatic Construction and Model-fitting Lattices .....	94
5.2	Lattice Refinement .....	95
5.2.1	Varying Sampling Rates .....	97
5.2.2	Adaptive Subdivision Scheme .....	98
5.2.3	Vertex Decimation Approach .....	99
5.2.4	Vertex Merging Method.....	100
5.2.5	Tests and Evaluations .....	101
5.3	Summary .....	105
Chapter 6 Volume Deformation .....		107



6.1	Constructing Deformable Solids .....	108
6.1.1	Lattice Deformation.....	109
6.1.2	Embedding Mass-spring Mechanism-based Framework.....	110
6.1.3	Implementing Resistance Force Mechanism .....	114
6.2	Displacement Mapping .....	117
6.2.1	Mapping Process Design.....	117
6.2.2	Design of Indexable Inherent Relationship (IIR) .....	118
6.3	Octree-based Lookup Function .....	122
6.3.1	Implementing Octree Data Structure .....	124
6.3.2	Constructing Octree-based Lookup Mechanism .....	124
6.3.3	Accomplishing Volumetric Deformation .....	125
6.4	Fixing Deformation .....	127
6.5	Summary .....	128
Chapter 7 System Integration and Acceleration .....		131
7.1	Preparations of CUDA-based Programming.....	132
7.2	CUDA-based Volume Visualization.....	134
7.2.1	Geometric Modelling Function .....	135
7.2.2	Kernel Sampling Function.....	136
7.2.3	Kernel TF .....	137
7.2.4	Kernel Accumulation Function .....	137
7.3	CUDA-based Lattices Construction .....	138
7.3.1	Kernel MC Function.....	139

7.3.2	Kernel Triangulation Function .....	140
7.3.3	Kernel Subdivision Function .....	143
7.4	CUDA-based Displacement Mapping.....	145
7.4.1	Kernel Octree-based Lookup Function .....	145
7.4.2	Kernel Mapping Function .....	147
7.5	Summary .....	149
7.5.1	SIMT Architecture .....	149
7.5.2	Synchronizing Kernel Functions .....	149
Chapter 8 Test and Evaluation .....		151
8.1	Efficiency Evaluation on Volume Segmentation .....	151
8.2	Effectiveness Test on Lattice Construction .....	155
8.3	Flexibility Assessment on Interactive Deformation.....	156
8.4	System Run-time Performance Evaluations .....	158
8.5	Summary .....	164
Chapter 9 Conclusions and Future Work .....		166
9.1	Conclusions .....	167
9.1.1	Efficient Volume Data Processing.....	167
9.1.2	Adaptive Lattice Manipulation .....	168
9.1.3	Flexible Deformation Control.....	169
9.1.4	GPU-accelerated System Integration .....	169
9.2	Future Work .....	170
References .....		172

## Chapter 1 Introduction

In contrast to the “pure” mathematical studies carried out by a small group of elite mathematicians in the 19th century, the so-called applied mathematics had been enjoying great success and public attention since the turn of the 20th century (Gluchoff, 2005). One of the main contributions to this phenomenon is the increased integration of the theoretical and abstract mathematical concepts with other scientific disciplines, such as physics, engineering, economics and even biology (Matsuura, Oharu et al., 2003; Dumas, Druez et al., 2009; Hernandez, Mateos et al., 2009; Britz, Strutwolf et al., 2011). The breakthroughs in those vastly diversified areas were supported by new mathematical tools and theories developed in the first half of the 20th century: statistics, topology and modern integral theory.

The invention and wider spread of modern computer technologies since the second half of the 20th century have further accelerated this trend. For example, stemming from computer graphics, volume rendering has been growing into an important research field in the last two decades (Engel, 2004). Most of the developments of volume visualization and application techniques in the 1980s and 90s had focused on exploring the theoretical and mathematic foundations of the visualization process. Since the mid-1990s, three leading research groups have proposed a series of improvements for PC-grade and efficient volume visualization techniques: the portable visualization clients by the Visualization and Interactive Systems group in the University of Stuttgart (Engel, Oellien et al., 2000); advanced volumetric modelling methods by the Visual and Interactive Computing group in the University of Swansea (Chen and Tucker, 2000), and versatile volume shading designs by the

Scientific Computing and Imaging Institute in the University of Utah (Kniss, Premoze et al., 2003).

Compared with conventional 3D modelling and visualization techniques, volume models allow direct or indirect access to their internal structures, instead of only showing their surface features. In Figure 1.1, all four snapshots are showing an engine box. Besides the vivid surfaces, wireframe-based model A does not show any internal information. In contrast, model B is a volume model, which can be processed into model C and D respectively to exhibit the interiors via two popular representation modes.

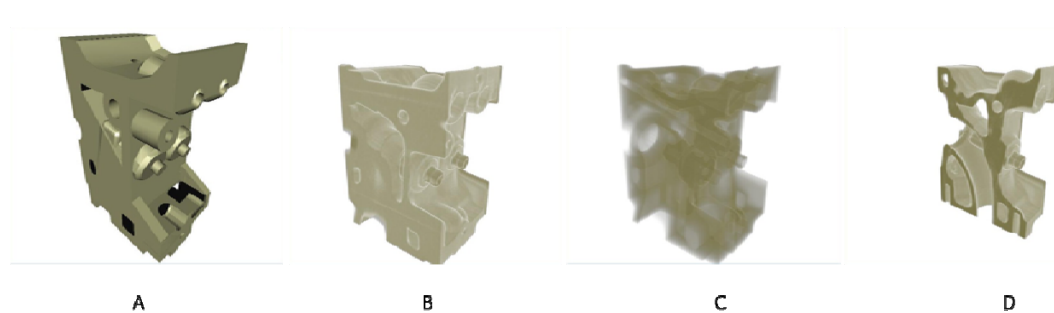


Figure 1.1 An engine box represented via different modes. Model A is a surface model. Model B presents a volume model; model C shows the internal structure through modifying the optical characters; which model D represents the interiors via the clipping operation.

This visualization technique aims to gain the understanding of multi-dimensional information and to display it as images. Similar to other modelling techniques, it also suffers from a series of challenging problems, such as occlusion within models, random data structures, noisy data, and artefacts generated from display mechanisms. The main objectives of recent researches were to develop solutions for exploring the “visibility” of various volumetric structures (Aykanat, Cambazoglu et al., 2007; Sakamoto, Kawamura et al., 2010). This thesis deepened the understandings by devising a novel solution of which users can freely customize the behaviours of

volume visualization process via interactive manipulation. In this thesis, Chapter 1 starts explaining the volume data source and carry on a brief introduction of two kinds of general volume data acquisition methods. After accomplishing the recapitulative content, the research objectives and the main contributions from the programme will be highlighted at the end of this chapter accompanied by the thesis outline.

## 1.1 Volume Data Acquisition

In the 1960s, an American theoretical physicist, Allan M. Cormack, published a mathematical model for calculating different rates of absorption of ionized radiation (i.e. X-ray) when crossing through different body tissues (Cormack, 1979). Based on this model, a British engineer, G. N. Hounsfield, invented the world's first Computerized Tomography (CT) scanner, an imaging device that allows 2D or 3D sectional or volumetric models to be reconstructed in order to represent internal information from the probed subject (Hounsfield, 1979). Due to their significant achievements, Cormack and Hounsfield won the Nobel Prize in 1979. Figure 1.2 shows the snapshots of a sliced human brain.

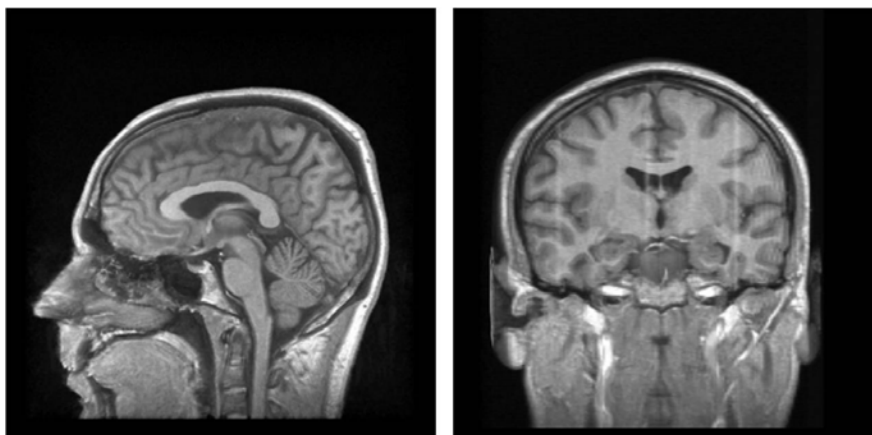


Figure 1.2 A CT scan of a human brain

The model proposed by Cormack was based on the Radon transformation in Integral

Geometry developed by Johan Radon in 1917 and translated into English in 1986 (Radon, 1986). The Radon transform is widely applicable to tomography, the creation of an image from the scattering data associated with cross-sectional scans of an object. If a function  $f(x,y)$  represents an unknown density, then the Radon transform represents the scattering data obtained as the output of a tomographic scanner. Hence the inverse of the Radon transform can be used to reconstruct the original density from the scattering data, and thus it forms the mathematical underpinning for tomographic reconstruction, also known as image reconstruction.

The Cormack model resolves the inverse Radon Transformation issue through convolution and inverse projection. Therefore, once the  $f(x,y)$  is determined, it is possible to reconstruct the sectional images of the measured tissue. Techniques like Magnetic Resonance Imaging (MRI) may use a less invasive measuring mechanism, but its foundation theory is similar to that for scanned image registering and reconstruction.

Current CT and MRI scanning processes were normally time-consuming and uncomfortable for patients, due to the rigid body postures which need to be maintained throughout the dedicated solutions (Olabarriaga and Smeulders, 2001; Fluck, Vetter et al., 2011). The latest advancements in digital processing have seen fast scanning technologies being developed and utilized with varying degrees of success (Zhou, Matsumoto et al., 2010; Cho, Cho et al., 2012). However, the scanned images often suffer from unsatisfactory qualities (e.g. noise-level) due to their inherent acquisition mechanisms that can cause great difficulties for image and 3D reconstruction (Qian, Joshi et al., 2011). One of the motives for this research is to investigate data filtering (pre-processing) techniques, prior to attempts being made to

improve the performance of the following volume visualization and deformation approaches.

## **1.2 Volume Visualization and Deformation**

After sampling the volume data in a regular sequence, the inherent information can be converted into a series of parameters. The following visualization work can be implemented directly (known as Direct Volume Rendering, or DVR), or accomplished indirectly by relying on an iso-surface and then “copying” the polygonal displaying strategy using surface modelling techniques (called Indirect Volume Rendering, or IDVR). In order to explore the various features inside the volume data, a special Lookup Table (LUT) is utilized for the purpose of data classifying, and to “individualize” them with associated values. These values are to be accumulated together on the image plane so that the inherent information can be represented via an understandable image.

As a process for manipulating volume models, volume deformation can be categorized as both non-physics-based and physics-based (Correa, Silver et al., 2010). The former techniques were well-known because of the ability to “freely” deform volumetric objects. In associated deformation applications, there was little or no regard to the consideration of physical realism (Forsberg, Mooser et al., 2008; Sugihara, Wyvill et al., 2010). In contrast, physics-based techniques are strictly governed by the result of utilizing physical equations to calculate the external and internal forces (Nealen, Müller et al., 2006; Bordemann, 2008). As the line between these non-physics-based and physics-based deformation approaches is becoming blurred in surface modelling techniques, more and more researchers are trying to use non-physics-based techniques to produce the physical effects in volume-based

applications (Correa, Silver et al., 2010). However, most of them relied heavily on man-made constraining operations and one-sided assumption-based works, which had undoubtedly caused many artefacts and imprecise outputs during the “real-life” simulations.

This thesis referred to physics-based volume deformation as a technique whereby deformation was driven explicitly by applying forces on the control lattices enclosing the volume model. In addition, both inertia and internal forces were considered. This deformation method aims to describe the deformation behaviours and exhibit the internal transformation precisely.

As an important criterion for evaluating deformation techniques, the performance of interactive operations always influences the development of volume deformation (Yuan, Zhang et al., 2010). The requirement for “on-the-fly” mechanisms exists at every stage of the deformation pipeline, such as the sampling, transforming and final displaying stages. In particular, the physics-based methods rely on complicated calculations in order to provide precise deformation results, and consequently physics-based manipulations of the volume model will lead to an inconceivably time-consuming data processing. As a result, the physics-based volume deformation requires hardware-based acceleration techniques to maintain a higher interactive rate during the simulation.

### **1.3 Research Objectives**

It is a common scenario that when deploying volume-based operations, the involved complex data sets are often of 10 times scale, as compared to the surface-model-only implementations, which leads to a heavy workload of sampling and rendering



operations for the host computer. The major development of the computing platform took place at the turn of the new Millennium empowered by the rapid consumer grade computer (see Appendix B) evolution (some reckoned as a revolution) (Mark, Steven et al., 2003). As a result of this, complex scientific simulation and visualization tasks have started moving from expensive workstations to Personal Computers (PCs). The current consumer grade computers can support many complicated graphical and non-graphical modelling at a near interactive rate through larger memory storage, broader data bus and faster data access. Volume visualization and its applications are among the first to benefit.

Accompanied by the increasing computing power supplied by the innovative hardware platforms, volume models and their derived manipulations are quickly coming out of the shadow of the surface models and becoming one of the main representation forms for special applications (Strengert, 2005; Tatarchuk, Shopf et al., 2008). Although there were still many challenges in volume visualization and application techniques, i.e., in the real-time interaction arena, many research designs have been focusing on improving the efficiency of the volume-based applications, enhancing the rendering quality, or developing the interactions with the volume models.

In addition to directly rendering, volume models can support other processing methods so that more intrinsic information can be acquired, e.g. clipping methods. This project created a novel volume deformation mechanism which enabled precise representations of the deformation behaviours, rapid interactive operations and versatile applications. The aim and objective of this project can be summarized as follows:

- To investigate the state-of-the-art of volume visualization and deformation techniques. The evaluation criteria were summarized based on the literature review and used to discuss the pros and cons of the implemented system. The system performances were categorized into three terms: “Flexibility”, “Efficiency”, and “Accuracy”, of which the evaluations were respectively estimated through processing different volume data, counting the real-time performance and tackling customization operations (Correa, Silver et al., 2010; Patete, Iacono et al., 2012).
- To improve the system performance. Besides the help offered by GPU-based acceleration designs, the solution also comprised the dedicated data processing strategy designed for solving the problem of large volume data size (Cates, Lefohn et al., 2004). By studying the classic Digital Image Processing (DIP) methods, this project finished a high-dimensional DIP solution for carrying out volumetric data classification and information extraction (Fang, 2001).
- To accomplish the physics-based deformation mechanism for manipulating volumetric features. First of all, a mathematic model was constructed for partitioning the resulting deformation behaviour into a set of changed characters within the volumetric space (Bachmann, Bouissou et al., 2009). Secondly, the assignment of these characters was based on an order which records the nested structures inside the volume model. And the control lattices met the demands: nonexistence of self-intersecting polygons, this “model-fitting” lattice conforming to models’ surface features, and the adaptive meshwork structure (Sauvage, Hahmann, et al., 2008). In addition, the data exchange between CPU and GPU were investigated for further efficiency gain (Lefohn, Kniss, et al., 2003).

## 1.4 Contributions to Knowledge

Besides the literature review, the main research efforts discussed in this dissertation were respectively reflected by four functional modules (as shown in Figure 1.3). As a result, the contributions to knowledge from this project can be summarized as:

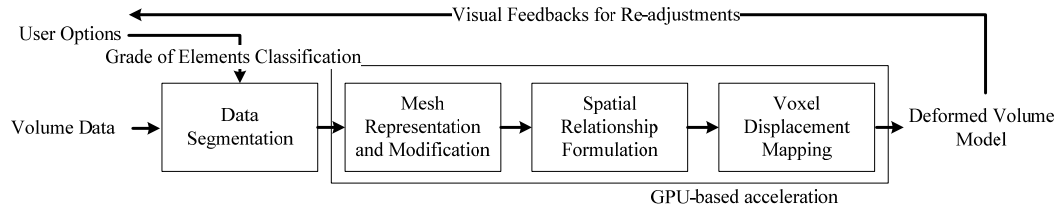


Figure 1.3 A diagram of the design system pipeline

- Volume data segmentation abstracted interesting information from the continuous volume data; meanwhile, filtered out the inherent noise and the trivial segments, and consequently avoided useless data computation and processing tasks. Instead of processing the resulting images, or relying on per-frame-based operations, the strategy of volume data segmentation created a one-off 3D data pre-processing before progressing visualization and deformation operations. It was published in the 17<sup>th</sup> ICAC conference paper. By rendering these segment data, the result directly exhibited the internal structure which can offer help in choosing the interesting data segment(s).
- In order to customize the extent of transformations, the spatial determination process relied on two key designs. One was a lattice-based “interface” between applied forces and underlying volume models. The outcome of the lattice construction process can influence the results of deformation operations. One was a tree-like framework for ascertaining whether the voxels belong to the deformed domain. Both them were explained in the same paper.

- The spatial displacement response within volume models was derived from similar surface-modelling-based deformation examples, which styled the external force calculation. And the deformation process was “translated” into a set of parameter computations through determining a special data structure to standardize the transformation operations. As a result of this, the parameter calculation and data accessing tasks can be accelerated by the GPU-based parallel computation design, so that the interactive rate of future operations can be optimized. The benefit from GPU-accelerated achievements was firstly mentioned in my paper published in 16<sup>th</sup> ICAC conference.
- Similar to the conventional displacement mapping techniques for mesh-based object deformation, the voxel displacement map was designed to record voxels’ offset distances. This project utilized the view-aligned proxy geometry for volume data storage, therefore, the voxel displacement map was correspondingly 3D-based. This 3D texture mapping avoided the complicated interpolation processes involving the combinations of 2D and 3D textures, and consequently improved the system efficiency. This part is based on the work document published in 15<sup>th</sup> ICAC conference.
- By harnessing the parallel processing capabilities of GPU, the hardware acceleration design in this research managed data through synchronizing each sub-tasking. Therefore, the visualization and deformation operations were partitioned into a set of subtasks synchronized in a parallel structure for improving the system efficiency. This design was explained in the 17<sup>th</sup> ICAC conference paper.

## 1.5 Thesis Outlines

A comprehensive literature survey regarding contemporary volume visualization methods is recorded in Chapter 2. Chapter 3 provides an in-depth discussion on the strategies of manipulating volume models, as well as the state-of-the-art in volume deformations. The actual research methodology, design approach, implementation strategy and evaluations are respectively covered into four chapters. Chapter 4 employs image segmentation methods and improved them for classifying volume data. The classification enables the display of internal structures of complicated volume data and the extraction of interesting data segment(s), which in turn, simplifies the workload of the following data processing activities. The content in Chapter 5 explains the lattices construction's design principles and working mechanisms for enclosing the manipulated volume data. As a vital part of this physics-based volume deformation system, the constructed lattices met various pre-defined requirements, such as the "model-fitting" lattice, flexible modification and rapid construction. Chapter 6 covers the implementation of the deformed volume models. A mathematical model has been established to subdivide the deformation behaviour into the displacements of voxels through a "deformation parameterization" operation. Chapter 7 reveals the details of the Compute Unified Device Architecture (CUDA)-based implementations of the system prototype. This programming model was used to separate the lattice construction and deformation parameterization processes into sub-computation tasks, and to synchronize them into a parallel computing architecture for the acceleration purpose. Chapter 8 uses the common evaluation criteria to assess this deformation system in terms of its intermediate results, real-time performances, rendering quality. Chapter 9 concludes the research with achievements and future work.

## Chapter 2 Review on Volume Visualization Approaches

In early 1980s, volume visualization started attracting discussions in scientific communities, due to its potential and powerful capabilities in revealing the internal structures of objects (Drebin, Carpenter et al., 1988). However, limited by the computational methods and platforms at the time, volume visualization and deformation techniques faced tough challenges in various types of practical applications (as shown in Figure 2.1), especially in real-time operations. In the last decade, various research and pilot projects had focused on improving the quality of the final rendered results; meanwhile, maintained adequate performances in real-time operations (Kruger and Westermann, 2003; Strengert, Magallon et al., 2005; Tatarchuk, Shopf et al., 2008; Fluck, Vetter et al., 2011).

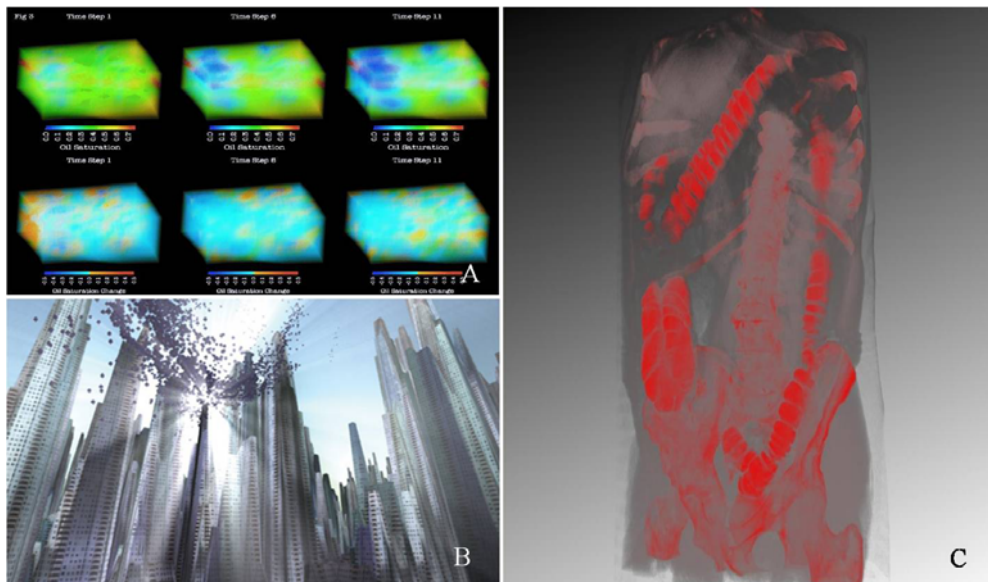


Figure 2.1 Various applications of volumetric information. Image A is for studying oil reservoirs in underground rocks (courtesy to Paul *et al.*). Image B illustrates a volumetric lighting method for gaming scenes (courtesy to Nvidia Corporation). Image C exhibits a torso model for the medical applications.

From this review of volume visualization terms, a comprehensive introduction of various visualization approaches can be accomplished with potential challenges in different applications. Based on this review, the research problems can be concluded and chosen as the objectives for developing visualization properties.

Volume visualization techniques represent the object via displaying its spatial characters in the form of images. Between the data acquisition and visualization stages, there exists a modelling process which fills a 3D space (as shown in Figure 2.2 (A)) with a set of 3D geometrical elements in order of the original data's inherent sequences. Each data can be assigned to a cubic element which is regarded as a volumetric pixel, and therefore named as "Voxel". All voxels (as shown in Figure 2.2 (B)) can be accessed in the form of partitions inside of the volumetric space (as shown in Figure 2.2 (C)). Each voxel can provide two types of parameters: one is 3D coordinates defined via its spatial location, and the other is a scalar value derived from the raw volume data. Depending on different voxel processing approaches, volume visualization can be categorized into direct (DVR) and indirect (IDVR) strategies.

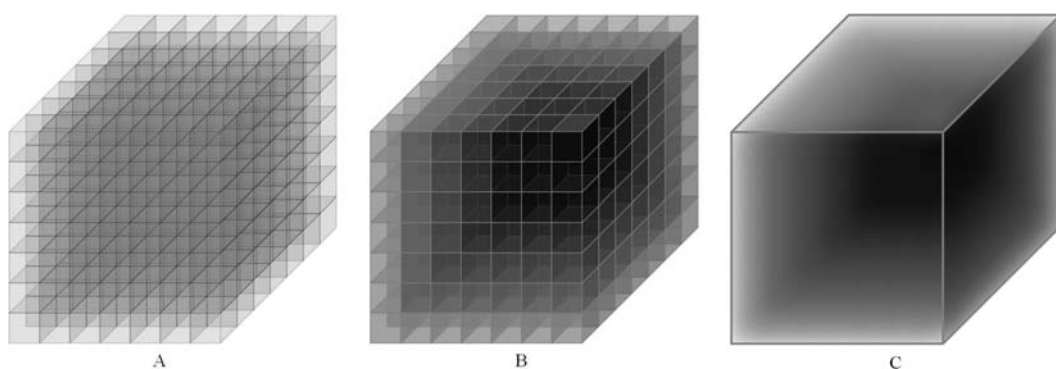


Figure 2.2 Illustrations of volume model formations

IDVR methods use vertices to replace voxels for indirectly representing the volume data. Marching-Cubes (MC) algorithm is a prevalent IDVR, which consists of following operations: extracting vertices from the volume data, grouping them

according to their scalar values, and implementing the Delaunay triangulation for each group of vertices (Lorenson and Cline, 1987) . This algorithm and output images will be covered in Chapter 5.

On the contrary, DVR solutions carry out a series of direct operations on voxels. These visualization algorithms require a pre-defined optical model for managing the conditions of volumetric light emission, light scattering, light absorption and ambient occlusion among voxels. These physical quantities are all based the voxels' optical properties which are notionally determined by their scalar values, and will be numerically represented by a series of RGBA quadruplets after the sampling process. This process of converting scalars into colours is yielded by the so called "Classification" phase.

The next step is sampling these values by casting a set of (parallel) rays through the volumetric space. Depending on the different directions of these rays, DVR methods can be divided into backward-mapping and forward-mapping ones (shown in Figure 2.3) (Engel, Hadwiger et al., 2004). In forward-mapping approaches, the voxels forward project themselves onto the image plane, to compose the final image via a sort of pixel distribution. Backward methods regard the viewing rays as sampling tools, which penetrate through the pixels in the image plane and detect the voxels' parameters for the image synthesis process. Because the system introduced in this thesis mainly relied on backward-mapping DVR in the visualization phase, the review of volume visualization techniques only focused on the forward-mapping methods. Consequently, without any additional explanation, the directions of DVR techniques discussed in the following context are all backward-oriented. As an important part of the image synthesis process, calculating the light propagation via the integrating light



interaction effect for each point within a volumetric space is based on the choice of optical models.

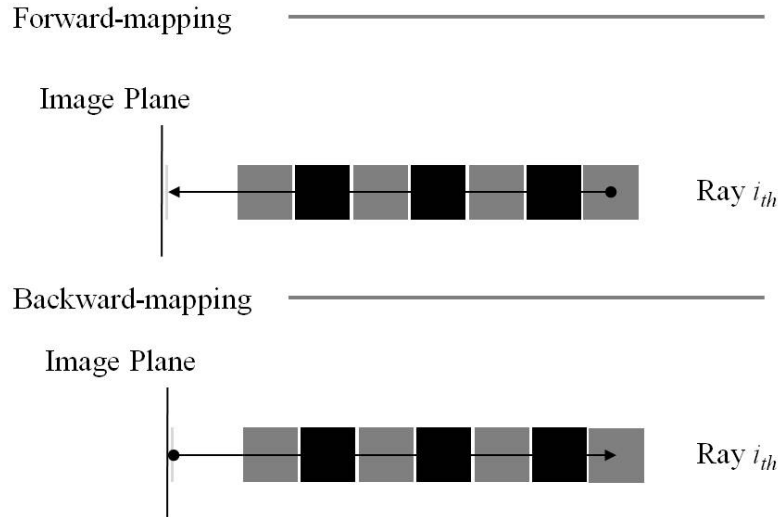


Figure 2.3 Diagrams of forward-mapping and backward-mapping methods

## 2.1 DVR Optical Model

An optical model serves as a paradigm that composes lighting-emitting points of uniform physical quantity. Based on different conditions of light propagation inside a volumetric space, the optical models can be classified as:

- Absorption only. This type of optical model determines the volumetric space to be a kind of black hole. In this region, all lights are completely absorbed, and consequently unable to support emitting or scattering conditions (as shown in Figure 2.4 (A)).
- Emission only. In comparison to the absorption only model, which prevents light emission and scattering, the emission-only optical model just focuses on the light emission condition, without any consideration of absorption or scattering (as shown in Figure 2.4 (B)).

- Absorption plus emission. As a synthesis of the above two optical models, this model simultaneously enables light emission and absorption. Because most DVR-based applications always overlook the discussion of scattering and indirect illumination, the absorption plus emission optical model is the most popular choice for volumetric light propagation (as shown in Figure 2.4 (C)).
- Scattering and shading. Using this optical model, the light scattering can be treated among the particles at voxel level. The scattering condition comprises projecting lights onto the surface of each voxel from a light source without any impeded objects, and being generated by the occlusion states among the voxels (as shown in Figure 2.4 (D)).
- Multiple scattering. As an extension of simulating light scattering in a volumetric space, the multiple-scattering optical model allows performances of the complicated mechanism of an incident light scattered by multiple voxels.

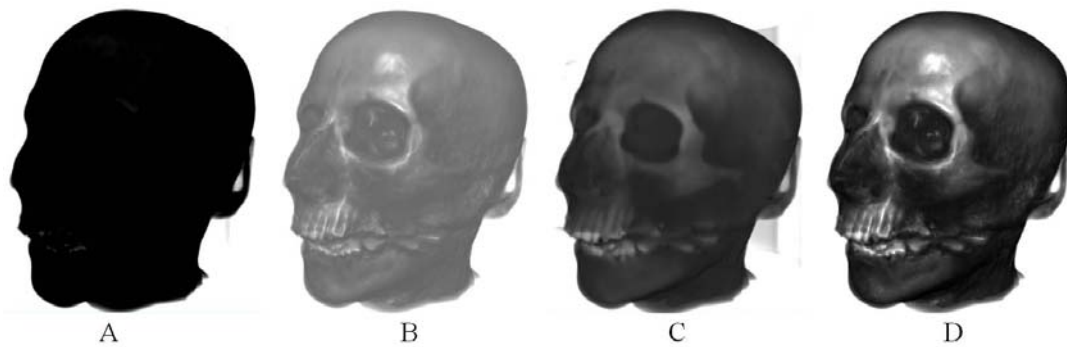


Figure 2.4 Shading results determined by different optical models

## 2.2 Ray-casting Theory and Practice

The ray-casting displays mechanism was summarized as sampling 3D information (voxels) and rendering in 2D formats (pixels), which belonged to the image-order rendering method (Ray, Pfister et al., 1999). Besides, the texture mapping approach,

which decomposes a volumetric space into the given type of slices (2D or 3D), was classified as an object-order rendering method (covered in section 2.1.3) (Weng, Lin et al., 2002).

As the most direct solution for evaluating the volume rendering integral along the rays from image space to object space, ray-casting was defined as the most basic DVR algorithm and a kind of backward-mapping approach (Levoy, 1988; Choi, Shin et al., 2000). After sampling the voxels' optical properties via casting (parallel) rays in the viewing direction through pixels into the volumetric space, ray-casting accumulates the resulting properties for each ray in the form of evaluating the volume rendering integral and rendering the results in the manner of pixels in the final display.

### **2.2.1 Volume Rendering Integral Model**

In every volume rendering method, the volume rendering integral was always evaluated in a certain direction. Generally, the viewing ray was chosen for evaluating the integral, even if it was unclearly defined. Because the sampling mode was not continuous in practice, the related optical properties were not continuous either. In order to approximate the evaluation of the volume rendering integral, the calculation was digitally replaced by a Riemann sum, which performs the accumulation of the properties along the viewing ray in terms of colour values (Levoy, 1988).

Each viewing ray will, of course, penetrate through a number of voxels in certain statuses including through a voxel's centre, through a voxel and on one of six tangent planes of it. The simplest condition is traversing through a voxel's centre so that its scalar value can be directly used as the sampled result of the viewing ray at this voxel. For the other two statuses, the sampled results all need to be calculated via the tri-linear interpolation in ray-casting-based applications (or 2D interpolation for

texture-based processing).

After the sampling process, the scalar value of voxel  $S(\vec{V}_i(dis))$  can be gained for the following calculations. The  $\vec{V}_i(dis)$  indicated a voxel which is sampled by the  $i_{th}$  viewing ray  $\vec{V}_i$  at a distance  $dis$  along this ray to a virtual viewpoint (as shown in Figure 2.5). This vector-based parameter comprised the information for indexing the scalar values stored in a kind of texture memory. When the most popular optical model (absorption plus emission) is employed, the indexed scalar value is represented via a colour value  $C_{emission}(S(\vec{V}_i(dis)))$  called emissive colour, and the absorption coefficient  $k$  is defined for describing the condition of light absorption via  $C_{absorb}(S(\vec{V}_i(dis)))$ . It can be written as (Engel, Hadwiger et al., 2004):

$$C_{absorb}(S(\vec{V}_i(dis))) = k \cdot S(\vec{V}_i(dis)) \quad (2.1)$$

Based on these two kinds of colour values, the volume rendering integral can carry out the resulting composition of colour values sampled along the rays. For example, in order to calculate the result of the viewing ray passing through a distance  $d$  (shown in Figure 2.5), the absorbed and emissive colours at different locations can be worked out respectively.

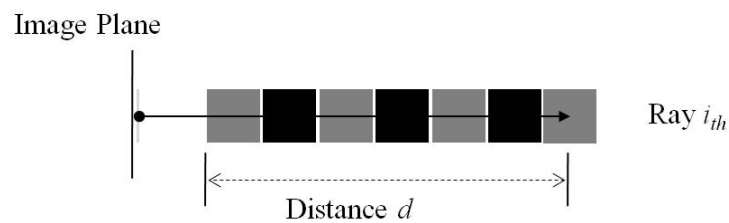


Figure 2.5 A diagram of a viewing ray for volume rendering integral

First of all, it is assumed that there are  $n$  voxels detected on this ray and no intervals between any two neighbouring voxels. Based on the equation 2.1, for the constant

absorption coefficient  $k$ , the light absorption on this ray can be written as:

$$C_{absorb} \left( S \left( j \cdot \frac{d}{n} \right) \right) = k \cdot S \left( j \cdot \frac{d}{n} \right) \quad j \in [0, n] \quad (2.2)$$

On the condition that  $k$  is dynamic, and depending on its position, the equation 2.2 will be changed into:

$$C_{absorb} \left( S \left( j \cdot \frac{d}{n} \right) \right) = k \left( j \cdot \frac{d}{n} \right) \cdot S \left( j \cdot \frac{d}{n} \right) \quad j \in [0, n] \quad (2.3)$$

In the same way, the corresponding light emission can be represented via  $C_{emission} \left( S \left( j \cdot \frac{d}{n} \right) \right)$  ( $j \in [0, n]$ ). After finding the related  $C_{absorb}$  and  $C_{emission}$ , the final result of this voxel on the image plane is  $C_{partition}$ , which can be written as:

$$C_{partition} = C_{emission} - C_{absorb} == S \left( j \cdot \frac{d}{n} \right) \cdot (1 - k \left( j \cdot \frac{d}{n} \right)) \quad (2.4)$$

### 2.2.2 Tri-linear Interpolation

Tri-linear interpolation was usually utilized to implement a multivariate interpolation for generating the sampling result in a voxel whose centre cannot be penetrated through by a viewing ray (Engel, Kraus et al., 2001). The tri-linear interpolation locates the resulting point with its scalar value through weighting eight neighbours' coordinates and their scalar values (Rajon and Bolch, 2003). For example, as shown in Figure 2.6, the Viewing ray needs to gain the scalar value of the point  $S_{interp}$ . After knowing about the scalar values of  $S_{interp}$ 's eight neighbours ( $S_{000} - S_{111}$ ), the first cycle of tri-linear interpolation locates the new generated points ( $S_{00} - S_{11}$ ). Based on these points, the second cycle of the interpolation locates the point  $S_{interp}$ , and returns its scalar value as the final output. Because of the capability of calculating

the interpolated values, the tri-linear interpolation is applied to “filter” ambiguous representations, and avoid visual artefacts caused by limited inputs with intervals between data, or inaccurate results because of discontinuous samplings.

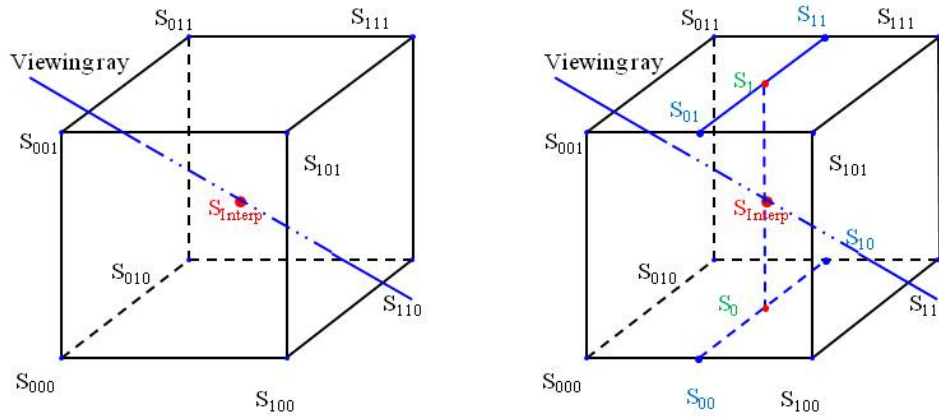


Figure 2.6 A diagram of tri-linear interpolation

### 2.2.3 Alpha Blending Operation

After obtaining  $C_{partition}$  in equation 2.4 and finishing the associated explanations of interpolation processes, an integral part of volume rendering is to carry on the composition operations in the alpha blending process, which accumulates colour values in the back-to-front or front-to-back order. The example shown in Figure 2.5 is a front-to-back approach, i.e., its composition starts at the voxel closest to the image plane and ends with a given voxel.

The composition process accumulates the sampling values in an opacity-weighted calculation, which is obtained by pre-multiplying the original value  $C_{partition}$  by its associated opacity property: alpha value. In this front-to-back method,  $C_{compos}$ , which represents each step of the composition, starts at  $C_{compos} = 0$  and accumulates the result of multiplying the  $j$ -th voxel's value  $C_{partition}(j)$  by the corresponding alpha value  $A_j$  (Wittenbrink, Malzbender et al., 1998):

$$C_{compos}(j, j + 1) = C_{partition}(j) + (1 - A_j)C_{partition}(j + 1) \quad j \in [0, n] \quad (2.5)$$

This alpha value is related to the voxel's location and the theory can be written as (Blinn, 1994):

$$A_j = A_{j-1} + (1 - A_{j-1})A_{original_j} \quad j \in [0, n], \quad (2.6)$$

which means that the alpha value  $A_j$  is determined by its previous one  $A_{j-1}$  and the alpha value  $A_{original_j}$  of the sampled value at this point is  $C_{partition}(j)$ . Then the calculated  $A_j$  is loaded in the equation 2.5 for the composition calculation.

By iterating the calculation in equation 2.5, the volume rendering integral  $C_{integral}(i)$  on the  $i_{th}$  ray is calculated via the Riemann-sum-based composition (Engel, Hadwiger et al., 2004):

$$C_{integral} = \sum(C_{compos}) = \sum(1 - A_j)C_{partition}(j) \quad (2.7)$$

Playing the role of controlling the supremum in the Riemann-sum-based volume rendering integral, alpha blending can determine the terminal of composing colour values in the front-to-back method. The composition can define an optimal indicator (known as early-ray-termination), which determines the progress of alpha blending. As shown in Figure 2.7, when the cumulated alpha value  $A_j$  (yielded in equation 2.6) is equal to 1.0, the result of the Riemann-sum-based composition will stop at the current position.

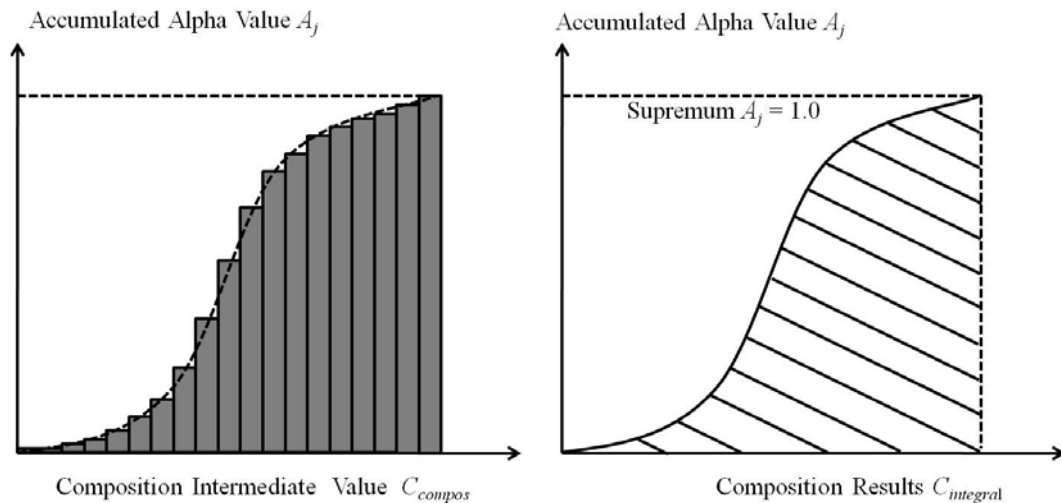


Figure 2.7 Diagrams of colour accumulation (left image) and approximated volume rendering integral (lined shadow area in right image)

$C_{integral}$  is the final result of the  $i_{th}$  ray after passing through the volume rendering integral, and represented by a few pixels which are stored in the frame buffer in the form of 2D texture. The resulting images are shown in Figure 2.8.

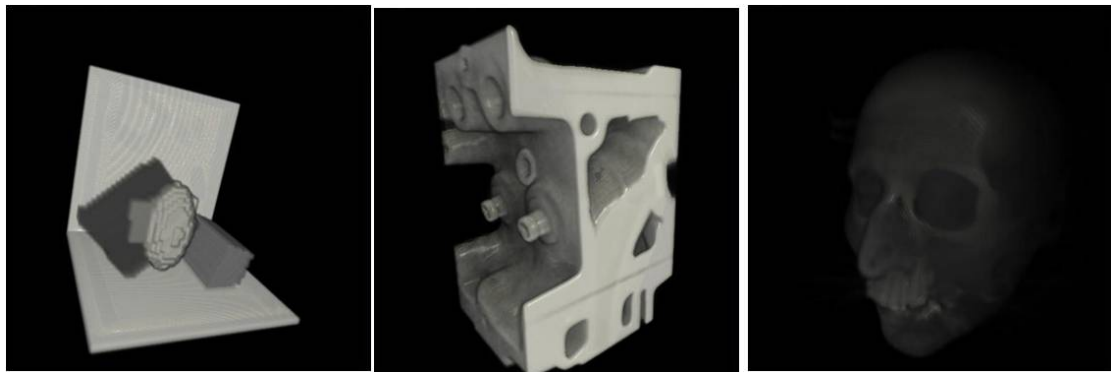


Figure 2.8 The results of ray-casting

However, as a number of image-order algorithms suffered from low efficiency caused by redundant computations, every ray-casting-based application struggled against the same challenges that the occasioned heavy sampling work on every ray and the iterative computations of opacity composition (Ray, Pfister et al., 1999). To overcome these drawbacks, the researchers mainly focused on developing an orientated rendering mode (e.g. digital boundary determinations to highlight or conceal given



regions) (Leu and Chen, 1999), increasing the efficiency of sampling mechanisms (e.g. early ray determinations to manage the progress of sampling every ray) (Hadwiger, Sigg et al., 2005), and improving display capability with the aid of applied hardware (e.g. high-quality performance of multiple volumes to enable realistic displays within complicated environments) (Tatarchuk, Shopf et al., 2008). In addition, there were a series of hybrid-based solutions that combines the image-order and object-order algorithms together to overcome their inherent disadvantages whilst, furthermore, maintaining their respective advantages for improving the performance of these solutions (Westermann and Sevenich, 2001).

#### **2.2.4 Classification Process**

Classification process relies on Transfer Function (TF) for assigning optical properties (colour, opacity, etc.) to the voxels by indexing their scalar value in a colour-based lookup-table (Engel, Hadwiger et al., 2004). Different combinations of the classification and the interpolation-based filtering processes will make the results of visualizing the same model totally different. The alternate order between classification and filtering processes respectively forms pre- and post- classification methods, and produces two kinds of results (as shown in Figure 2.9 (B and C)). Image D is an output of the pre-integrated classification design. Besides the visual artefacts in their results, all visualized features can be fully represented based on the complicated design of Nyquist frequencies in the TF design which always limited the real-time performance of visualization applications (Arens and Domik, 2010).

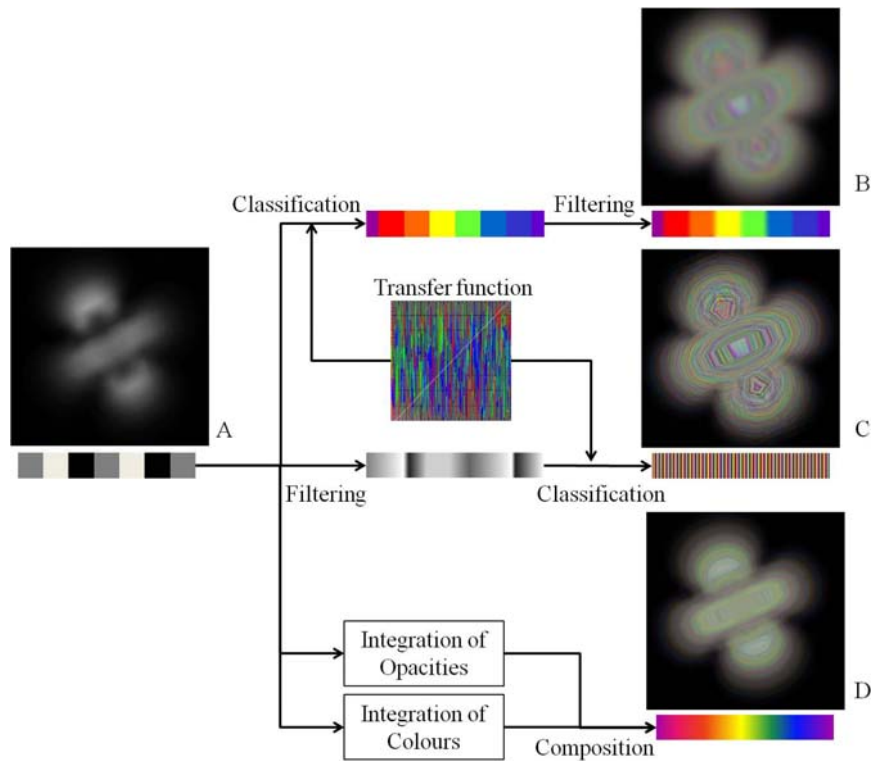


Figure 2.9 Diagrams of pre-, post- and pre-integrated classifications

For overcoming these limitations, an advanced solution was proposed to use the numerical integration to replace the complex Nyquist frequencies in the TF design (Engel, Kraus et al., 2001). Its idea is to “linearize” the sampled volume data into a whole segment which comprises a start point and ends with the last sampled data. The information of this segment needs to be integrated before the classification, so-called pre-integrated classification. In this solution, the length of the segment will be increased as the sampling work proceeds. The integration will keep a non-stop update on the colour and opacity of this segment. This integration requires two simple TFs for colouring the segment and voxels respectively.

As shown in Figure 2.9, the integration operation consists of two steps. The integration is for calculating the opacity of a segment. The other one is for integrating the sampled voxels’ colour values. The integrated opacity of this segment  $\alpha_{inter}$  can be written as (Engel, Kraus et al., 2001):

$$\alpha_{inter} = 1 - \exp \left( - \int_1^n \tau(S(\vec{V}(D))) dD \right) \quad (2.8)$$

where  $S(\vec{V}(D))$  represents a voxel's scalar values,  $\tau(s)$  is a simple TF for transforming voxels' scalar values  $s$ , and  $n$  is the number of voxels covered in this segment (the distances between voxels are assumed to be zero). The other integration is for the colour value of the segment  $C_{inter}$  can be written as:

$$C_{inter} = \int_1^n \left( T \left( S \left( \vec{V}(D) \right) \right) * \exp \left( - \int_1^n \tau \left( S \left( \vec{V}(D') \right) \right) dD' \right) dD \right) \quad (2.9)$$

with the other TF  $T(s)$  for colouring the segment. Besides avoiding the complex Nyquist frequencies in pre- and post- classification approaches, the pre-integrated classification can generate high quality visual results (as shown in Figure 2.9(D)).

TF had also attracted a great deal of attention on its multidimensional-based applications. In volume-based applications, as the simplest method, 1D TF directly maps the voxels' scalar values to associated colours and opacities. But, it cannot differentiate between volumetric subspaces whose voxels share the same scalar value but belong to different regions, e.g. the skull and teeth segments in a CT-scanned human head data. In order to solve the problem of inadequate representations in 1D TF, more parameters were utilized as the other dimensions in the TF (Arens and Domik, 2010). Besides the scalar value, a 2D TF can use the gradient magnitude as the second dimension for determining the differences between these domains (Levoy, 1988; Konig and Groller, 2001). The magnitude of gradient is used to represent the sampling statuses. For example, there won't be any change when sampling within a volumetric subspace, and the sudden change of gradient will happen when the current location of sampling is outside of the subspace (Kniss, Kindlmann et al., 2002).

Another 2D TF uses the curvature of volumetric subspaces as the second dimension. Generally, the shape of each volumetric sub-space contains a unique combination of the most and least curvatures (Hladuvka, Konig et al., 2000). Therefore, this 2D TF can differentiate complex volumetric information. Due to its capability of shape discrimination, this curvature-based TF usually serves as a shape-based analysis in the surgical simulations (de Vaal, Neville et al., 2011).

Besides the above TF design, 2D TF techniques also use the other available properties as the second dimension, such as distance-based method is based on the radiation radius of a pre-decided point (Tappenbeck, Preim et al., 2006), size-based TF uses the scale space for detecting the sizes of object domains (Correa and Ma, 2008), texture-based method relies on the texture analysis which detects the change of texture properties for mapping given specific opacities and colours to voxels (Caban and Rheingans, 2008). With voxel's scalar value, each of these available properties is used to form a multivariate control which enables the 2D TF methods to reveal more features than 1D methods (Kniss, Kindlmann et al., 2001; Kotava, Knoll et al., 2012).

Due to the increasing number of dimensions, multidimensional TF applications need to simplify the huge workload of manual assignments. Different from the complex trial and error tests in the multidimensional TF techniques, the automatic and semi-automatic methods all depend on the pre-defined criteria for driving the mapping operations (Pfister, Lorensen et al., 2001). Automatic TF designs are good at maintaining the interactive rate to real-time applications (Petersch, Hadwiger et al., 2005). Semi-automatic methods emphasise the importance of the user's intuitions and the flexibility of keeping a few manual modifications according to given conditions (Pinto and Freitas, 2008). The criteria can be categorized as two types: image-driven

and data-driven. Image-driven TF techniques usually focused on the quality of visual results and estimates the optimal design via a series of parameterized criteria (Pinto and Freitas, 2006; Park and Bajaj, 2007). Data-driven methods generally concentrate the capability of precise data displays, e.g. identifying the boundaries between volumetric sub-spaces (Kaul, 2010).

### 2.3 Texture-mapping Approaches

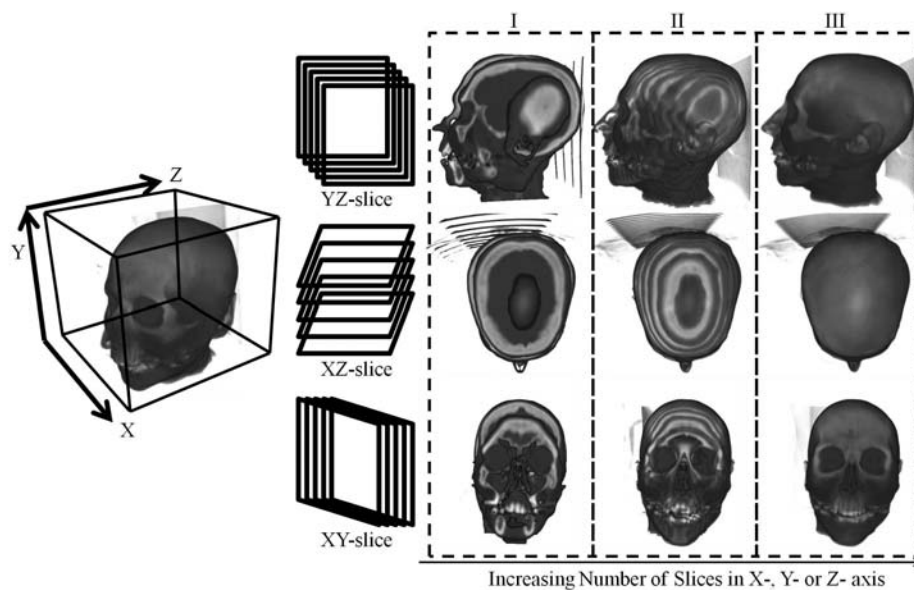


Figure 2.10 The results of object-aligned slices

Initially, texture-mapping plays a “skinning” role in surface modelling techniques. It serves to map visual features onto the surfaces of vertex-based frameworks, in order to represent the appearance of surface models. Mostly, these features are stored and processed in the form of 2D texture, so this displaying technique is named texture-mapping. In voxel-based environments, the volume data set is represented via a set of 2D textures, and anticipates the composition in the form of texture-based units. Unlike the ray-casting method (which belongs to the image-order method), texture-based volume visualization is an object-order method, and its displaying

quality is mainly dominated by the texture arrangement design (Engel, Hadwiger et al., 2004). In primitive texture-based volume rendering applications, three stacks of slices (textures), so-called object-aligned slices, respectively performed the view of a volume model in X-, Y- or Z-axial directions (as shown in Figure 2.10).

In associated applications, these slices were all pre-constructed, with the intention of determining one stack once for given conditions, such as the spatial attitudes of the volume object with a fixed viewpoint, along with orbit orientations of the viewpoint and complicated combinations of these two conditions. After finishing the slice-based representation phase, the following composition phase plays a “mapping” role, displaying the final results after implementing the integral calculations and the blending process slice by slice. Shear-warp model is a typical example of the object-aligned slices. Its mechanism of the composition process and operations will be explained in the section on shear-warp model below.

Because every slice is a potential candidate in 2D texture-based applications, producing the final display results generally requires the execution of the sampling, filtering (interpolation) and blending process three times for a proper representation. Besides the trebled workload, another manifestation of its disadvantages is the complicated pre-definition of the conditions for the slice exchange. Although this texture-based method manages to suffice for regular volume rendering applications, the above listed inconveniences restrict its performance in high-quality display applications.

For non-hardware accelerated techniques, texture-mapping-based volume rendering methods have always lost in most competitions with image-order methods (Weiskopf, Hopf, et al., 2001). By the 1980s, progressive graphics hardware techniques made a

great leap in terms of texture-based data manipulation, and consequently texture-mapping-based techniques attracted intensive attention. Benefiting from these advancements, texture-based applications obtained an increasing competitiveness in visualization, in comparison to other visualization methods. Meanwhile, an advanced texture-mapping method, so-called view-aligned slices, which constructs a one-off texture-based representation for replacing 2D textures, was proposed to save the predefinitions of the conditions with its developed texture representations (shown in Figure 2.11).

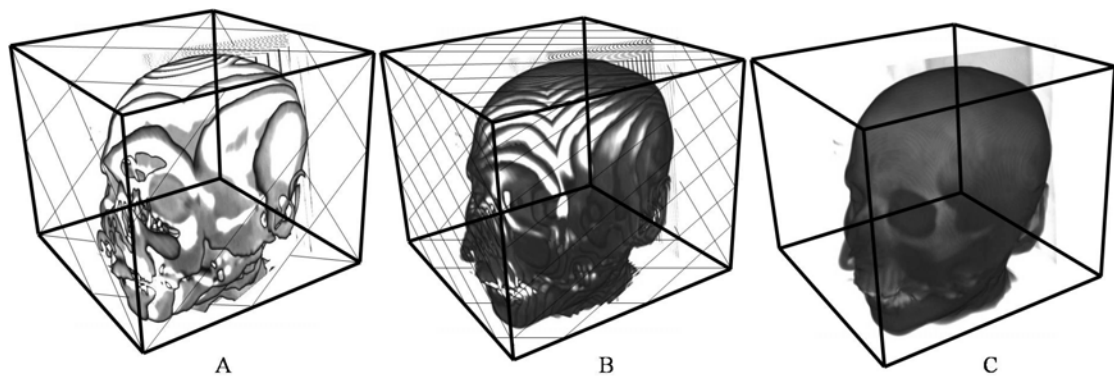


Figure 2.11 Results of view-aligned slices. From Image A to Image C, the sampling rate is gradually increased.

However, the display qualities of both 2D and 3D texture-based volume rendering are all limited by the frequency of the slicing volume data sets. For example, both column-II in Figure 2.10 and image B in Figure 2.11 show visibly discrete regions in the final results because of the incorrect initialization of the slices' properties. By increasing the number of slices, the resulting display can minify these interruptions so that a smooth look is produced for the naked eye (column III in Figure 2.10 and image C in Figure 2.11). Besides these usages, texture-based representation also plays an important part in implementing the shearing-warp model.

## 2.4 The Shear-warp Model

The shear-warp model makes the voxels project themselves, and consequently replaces casting rays into the volumes. The main objective of this forward-mapping approach is to simplify the complicated interpolations and compositions caused by transforming 3D properties into 2D results in an arbitrary kind of transformation (Levoy, 1994). Its basic idea is to shear and warp the volume model in the form of a fixed stack of slices, so that a 3D composition of the voxels' properties can be approximated via a 2D solution. Its potential customers are the applications which require a lower sampling accuracy and display quality than those of high-quality approaches. Depending on the different types of viewing mode, the shear-warp model is respectively optimized for parallel projection and for perspective projection.

### 2.4.1 Parallel Projection Algorithm

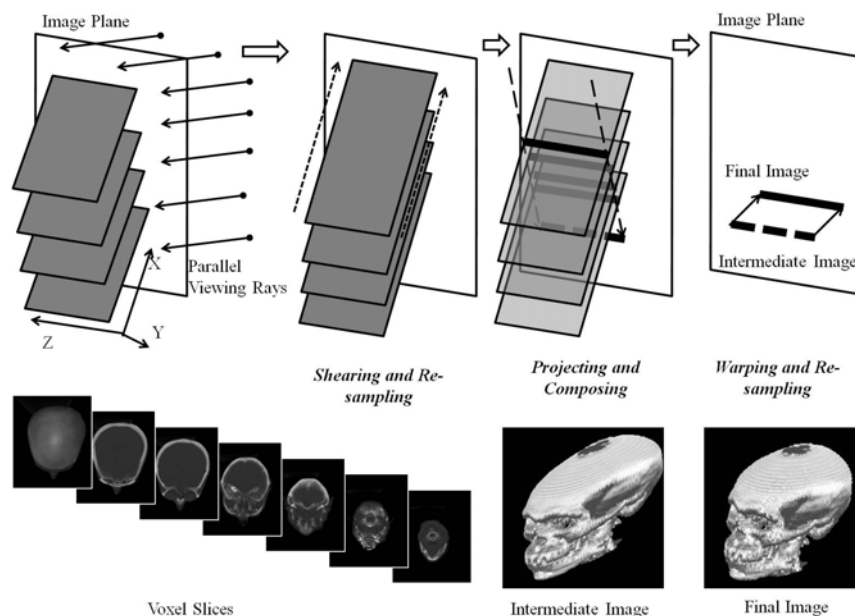


Figure 2.12 A diagram of the shear warp model with the parallel projection mode

As shown in Figure 2.12, the parallel viewing rays penetrate the image plane perpendicularly. After slicing the volume data, the shearing operation manipulates



these slices perpendicular to the viewing rays. Essentially, the sheared slices can parallel the image plane. In Figure 2.12, an intersection angle is forcedly drawn between slices and the image plane, in order to highlight the condition that the directions of the projecting intermediate image and the final image are not coplanar.

Because the shearing operation was along the Z-axial direction in this figure, the z-coordinate can be kept constant. In other words, the original locations of voxels on each slice all obtain the displacements in the (x, y)-plane. Therefore, the shearing operation can be expressed by:

$$M_{SO} = M_O \times M_S = \begin{bmatrix} x_1 & y_1 & z_1 \\ \vdots & \vdots & \vdots \\ x_n & y_n & z_n \end{bmatrix} \times M_S \quad (M_S = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ \Delta x & \Delta y & 1 \end{bmatrix}) \quad (2.10)$$

where  $M_O$  means the original coordinate matrix,  $M_{SO}$  represents the sheared result and  $M_S$  is a defined shearing matrix. The  $\Delta x$  and  $\Delta y$  respectively mean the X- and Y-axial displacement of voxels on the  $n_{th}$  voxel slice. Then, the sampling process will follow the calculated  $M_{SO}$  and the sampled results are composited along the Z-axis. Unlike the 3D composition of voxels' optical properties in the ray-casting method, the shear-warp model carries out the composition by taking each slice as a unit. Therefore, the results of viewing rays intersecting the slices can be approximated via a series of scan-lines, which consist of voxels with the same z-coordinate. In the same way, the tri-linear interpolation in ray-casting methods can be replaced by the bi-linear one. Therefore, the interpolated properties of voxels can be also treated as scan-line-based (as shown in Figure 2.13), and calculated for compositions.

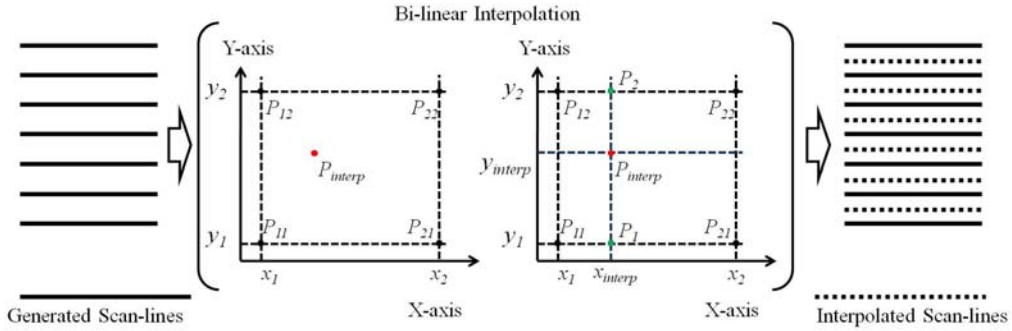


Figure 2.13 A diagram of interpolating scan-lines

The composition of sample results is then projected as an intermediate image. However, this image is also sheared (shown in Figure 2.12). Before being mapped onto the image plane, the warping operation is required to restore the sheared image. In order to implement a reverse calculation on the intermediate image, the warped result  $M_{WSO}$  can be written as (Levoy, 1994):

$$M_{WSO} = M_{SO} \times M_w \quad (M_w = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -\Delta x & -\Delta y & 1 \end{bmatrix}) \quad (2.11)$$

Through this warping operation, the stored image can be mapped onto the image plane.

### 2.4.2 Perspective Projection Algorithm

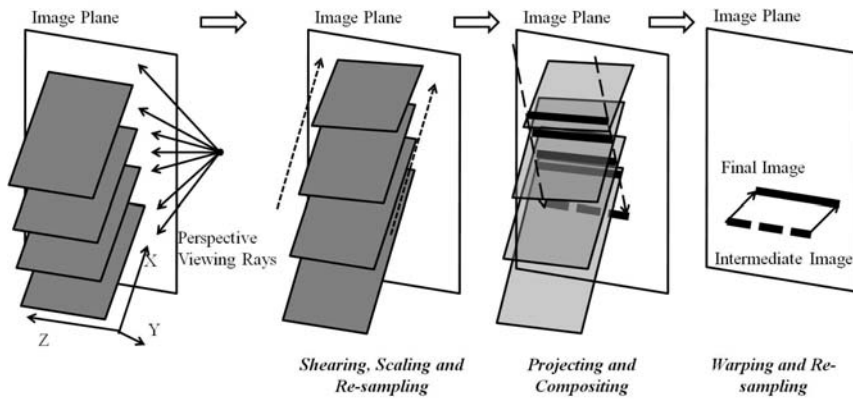


Figure 2.14 A diagram of shear warp algorithm with the perspective projection mode

In order to shear the volume object for the perspective projection condition, the slices need a combination of shearing and scaling operations to implement a similar projection transformation to that which exists in the viewing frustum in Figure 2.14.

This shearing operation is converted into (Levoy, 1994):

$$M_{SO} = M_O \times M_S = \begin{bmatrix} x_1 & y_1 & z_1 & 0 \\ \vdots & \vdots & \vdots & \vdots \\ x_n & y_n & z_n & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times M_S \quad (M_S = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \Delta x & \Delta y & 1 & u \\ 0 & 0 & 0 & 1 \end{bmatrix}) \quad (2.12)$$

In the viewing frustum,  $1/u$  means the distance between the camera and the origin of the viewing space. In this shear-warp model,  $u$  is designed to control scaling slices after shearing terms, and the scale is  $1/(1+u)$ . In the same way, the warping operation is (Levoy, 1994):

$$M_{WSO} = M_{SO} \times M_w \quad (M_w = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -\Delta x & -\Delta y & 1 & -\frac{1}{1+u} \\ 0 & 0 & 0 & 1 \end{bmatrix}) \quad (2.13)$$

Therefore, the result of the shear-warp model with perspective projection can be obtained. Because perspective and parallel projections all rely on the same composition, interpolation and sampling mechanism, both the final images of these two algorithms are the same.

As a complement to high-quality rendering methods, shear-warp model authentically simplified the interpolation and composition tasks required in the volumetric texture-mapping techniques through using one stack of slices for various conditions. Research of shear-warp model mainly focused on the improvement of TF to improve the efficiency of transformation properties, exploring the trade-off between online

shading management and performance penalties, and experimenting with the feasibility of the coexistence with other object-order methods (Wu, Bhatia et al., 2003).

## 2.5 The Maximum Intensity Projection

Unlike the compositing of optical properties in the ray-casting and shear-warp approaches, Maximum Intensity Projection (MIP), which is a backward-mapping DVR approach, keeps the maximum property value encountered along a viewing ray as the ray's final footprint projected on the image plane (Wallis, Miller et al., 1989). Its most popular applications were in the field of medical imaging, with its capability for computationally fast imaging, such as cancer screening equipment, CT scanners and diagnoses in nuclear medicine. As shown in Figure 2.15, the MIP's composition work can be considered as a texture-based method. However, its differences from the other texture-based techniques are the textures parallel the viewing ray.

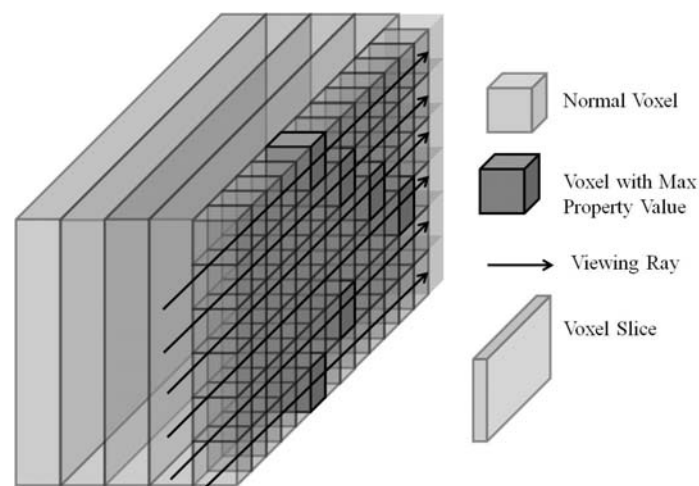


Figure 2.15 A diagram of MIP

By carrying out the maximum operator along one side of each slice, the final result of MIP can be considered as a set of projections of slices which are perpendicular to the

image plane. The projection of each slice can be written as (Wallis, Miller et al., 1989):

$$Max(x, y_n, z_m) = I_{MIP}(y_n, z_m) = \max(O_0, O_1, O_2, \dots, O_{n-1}) \quad (2.14)$$

where  $Max(x, y_n, z_m)$  denotes the property value encountered along the  $n_{th}$  viewing ray through the  $m_{th}$  slice;  $I_{MIP}(y_n, z_m)$  symbolizes the location of the associated footprint on the image plane, and  $O_x$  ( $x \in [0, n)$ ) is the optical value of the voxel at coordinate  $(x, y_n, z_m)$ . Figure 2.16 shows the result of MIP technique.

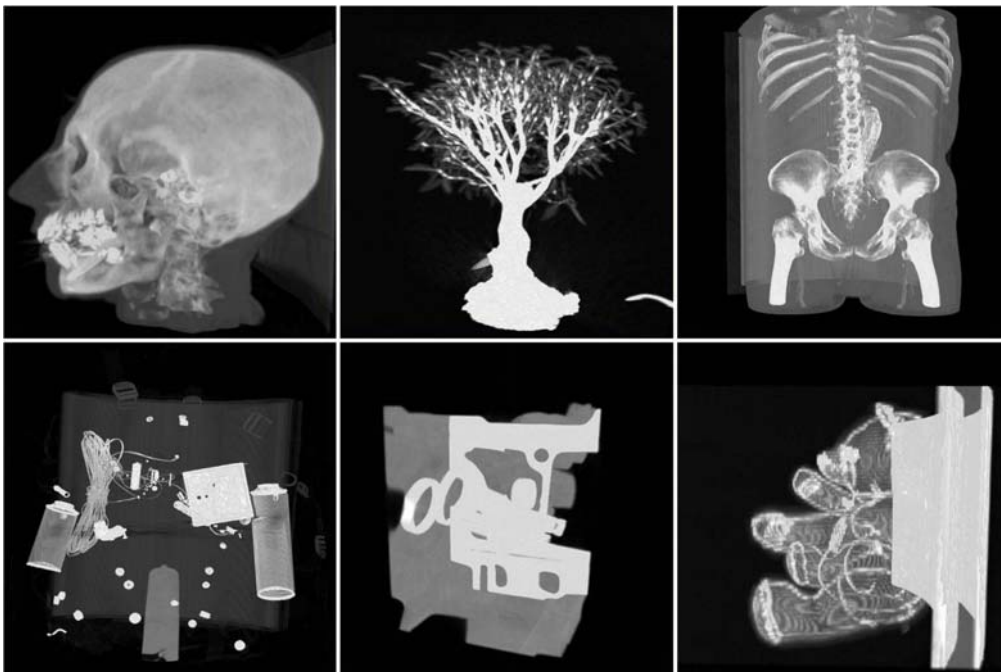


Figure 2.16 Results of MIP

The mechanism of MIP determines that each pixel value in the result is obtained by locating a maximum optical value. This result cannot support an adequate depiction of the spatial content of overlapping regions, because of the lack of “depth information”. As one general solution for creating MIP-based animations (e.g. rotation), an illusion is pre-constructed by determining the location of a set of viewpoints, and using the slice-based transformations in shear-warp model to obtain the associated image of

MIP with every viewpoint's location (Cai and Sakas, 1998; Fang, Wang et al., 2002).

In addition, shortening the sampling region is a direction of assigning the MIP with the ability of representing occlusions. Technically, the idea of the solution is to pre-define a threshold value to stop the iteration of maximum operator at the first time the intermediate result reaches this “threshold” (Sato, Shiraga et al., 1998; Han, Keyser et al., 2009). This solution is as named Local Maximum Intensity Projection (LMIP), and its improved results are shown in Figure 2.17.

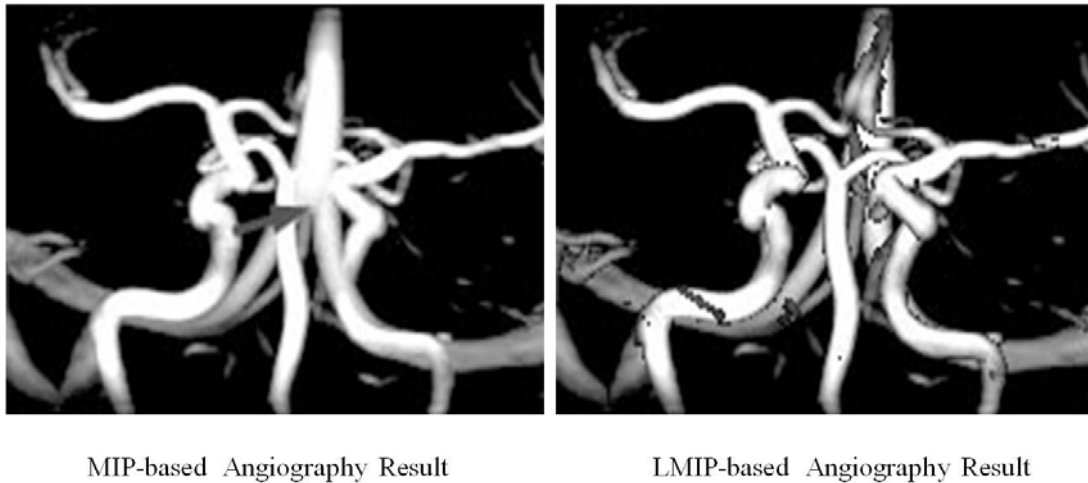


Figure 2.17 Illustrations of MIP-based and LMIP-based imaging methods (courtesy to Sato *et al.*)

## 2.6 Volume Shading Techniques

Playing an important part in rendering operations, volume shading methods can increase the authenticity of simulations by performing sophisticated displays with realistic lighting effects inside volumetric spaces (Hadwiger, Sigg et al., 2005; Rieder, Palmer et al., 2011) (shown in Figure 2.18). In order to perform the light interactions within a volumetric space, the most direct solution is to use dedicated optical models to determine light emission and absorption properties for each voxel, so that the resulting appearances can be uniformly translucent or opaque. In addition, a set of

vertices extracted from the voxels which share the same scalar value, form a surface to display the illumination effects in surface-modelling-based strategies (Hadwiger, P.Ljung et al., 2009).

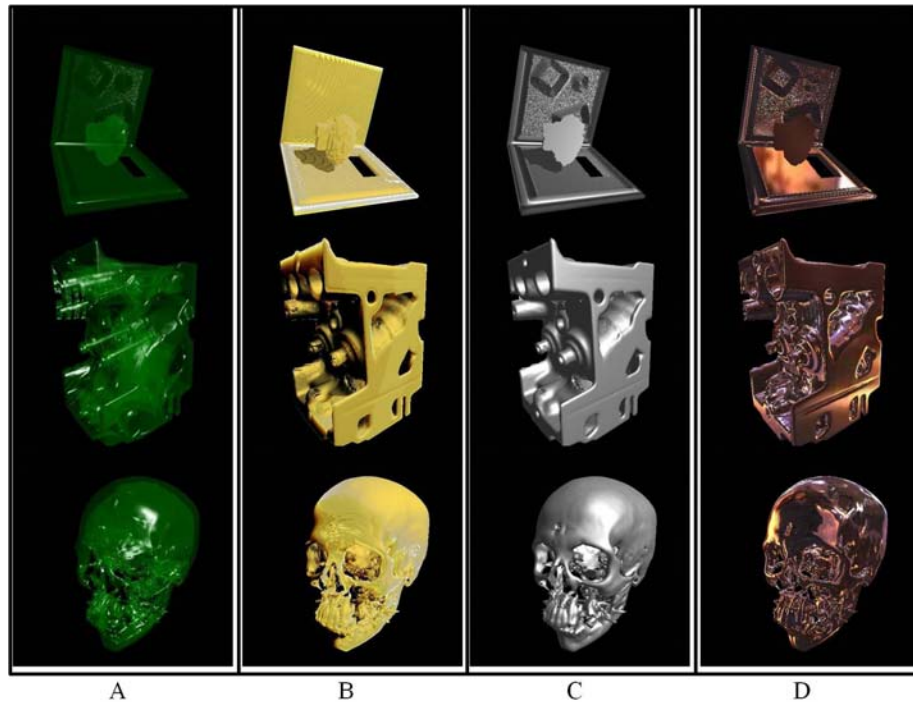


Figure 2.18 Various shading effects. Column A exhibits transparent results. Column B contains opaque contents. Column C comprises surface-based results. Column D represents the sphere-mapping effects column C.

### 2.6.1 Monte-Carlo Techniques for Iso-surface

The Monte-Carlo techniques for the iso-surface extracted from the volume data, can be divided into three steps: first hit approach, deferred shading approach and deferred ambient occlusion approach (Hadwiger, P.Ljung et al., 2009)

- First hit approach

Similar to the iso-surface extraction mechanism, the first hit approach iterates a sampling cycle successively to construct a specific surface and pre-specifies a scalar value as the threshold value for the sampling process simultaneously. Then, this

specific surface, so-called first iso-surface is constructed by gathering the voxels which are the first ones to obtain the given scalar value. Based on computing the normal vectors to determine the reflection characters on the first iso-surface in the form of a floating-point RGBA quadruplet, the outcome of first hit approach can be represented by a frequency histogram method in Figure 2.19 (A). The result of enabling gradient vectors in the outcome of first hit approach is illustrated in image B.

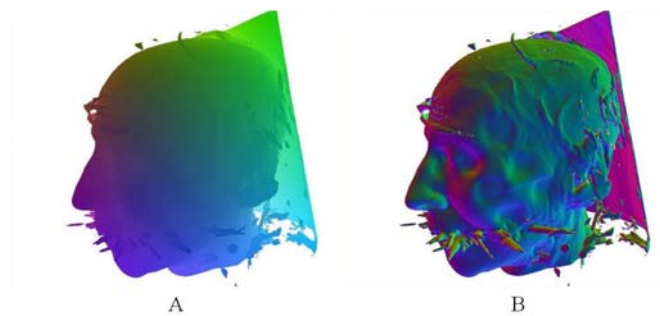


Figure 2.19 Results of first hit approach for the human head data set

- Deferred shading approach

After obtaining the results from the first hit approach, the shading of the first iso-surface can be computed by loading the normalized gradient and evaluating local illumination appearances (e.g. Phong shading with a point light) (Bennebroek, Ernst et al., 1997). This section just focuses on computing the reflectance behaviours, and the content of scattering terms will be explained in the following section. The reflection property of each element on this iso-surface is constant and precisely addressed by a shift invariant Bidirectional Reflectance Distribution Function (BRDF) (Nicodemus, 1965). The result of deferred shading approach is shown in Figure 2.20.



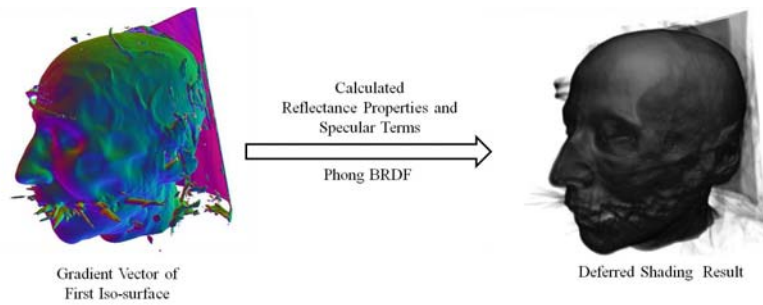


Figure 2.20 A diagram of deferred shading approach

- Deferred ambient occlusion approach

Working on the same first iso-surface, the deferred ambient occlusion approach plays the shading calculation on the local environment surrounding the surface, based on the intersection condition between this surface and a series of random rays. These rays all start from the surface and follow the reflection mechanism in a shift variant BRDF. Unlike the invariant one in the deferred shading approach, the anisotropic specular terms cannot be directly obtained through indexing a pre-filtered reflection map (Zhang, Zhu et al., 2011). The local environment needs to be split into a set of hemisphere domains around the start points of the rays on the first iso-surface to form a sort of local orientation for orienting the specular terms. These domains come from using Monte-Carlo integration to calculate the incident radiance over each vertex on the iso-surface (Hadwiger, P.Ljung et al., 2009). Figure 2.21 shows the result of the ambient occlusion approach.

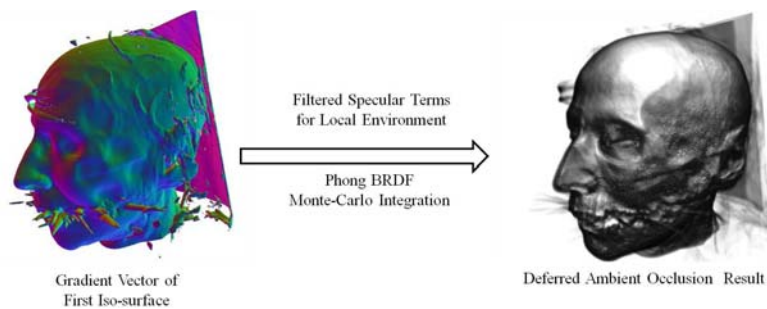


Figure 2.21 A diagram of deferred ambient occlusion approach

### 2.6.2 Volume Scattering

After finishing the explanation of Monte-Carlo techniques for rendering iso-surface, the rest contents will focus on volume scattering. Scattering is a physical process which makes the light deviate from its original path. In surface-modelling-based applications, the light interactions usually happened in an assumed vacuum and only takes place on the surface of models. For simulating direct or indirect illuminations, evaluation of the conditions of incident rays and specular rays will respectively rely on the single or multiple-scattering design (Stankevich, Shkuratov et al., 2003). Volume scattering usually took place in the form of the light deflections on the exterior and interior of volumetric objects. Similar to the deferred ambient occlusion approach, volume scattering also needs to define hemisphere domains as the incident radiances for tracing anisotropic scatterings inside the volume model according to the Monte-Carlo method (Nishita, Dobashi et al., 1996; Roy and Ahmed, 2011). The determination of these domains will influence the display quality of the final results. By taking advantage of multiple-scattering, the associated applications can differentiate the displays of homogeneous regions in the final result. For example, the eyehole region in the right image can reveal more features than the left one in Figure 2.22.

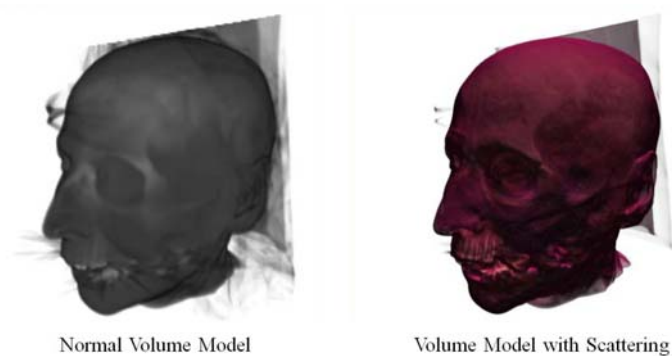


Figure 2.22 A result of volume scattering

As shown in Figure 2.23, there is a result of volume shading (for a human head data set) which was produced by combining the above mentioned shading approaches together and carrying out a composition of hybrid shading processes. Besides the realistic effects, this composition also brought challenges in terms of maintaining a applicable performance in real-time applications (Hadwiger, P.Ljung et al., 2009).

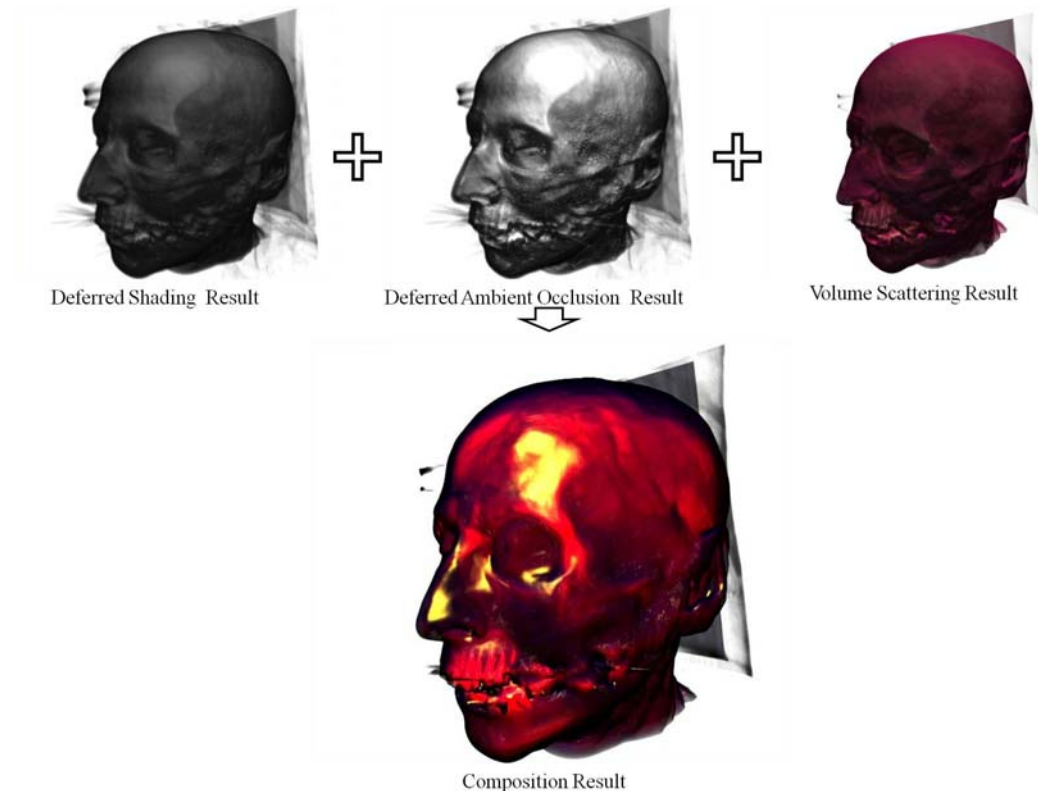


Figure 2.23 A diagram of a shading composition design

## 2.7 Research Objectives

Based on the above introduction of volume visualization techniques, the common research problems of large data size, artefacts, low efficiency and poor image qualities, were set as the research objectives of developing visualization properties:

- Large data handling. The limitations and derived problems caused by processing large data sets can affect the whole system, throughout the displaying,

computation and manipulating processes. Because researchers still insist on increasing data sizes to obtain more information or carry out higher level performances, the related solutions cannot fully overcome these derived problems. The most challenging task is to find the balance between performance and efficiency in volume-based applications.

- Reducing visualization artefacts. As a useful data filtering tool, data interpolation and (re-) sampling processes were usually utilized at a high frequency for maintaining the continuity of data sets in data processing stages in primitive solutions. However, these solutions also suffered from the limitations caused by the largely increased proportion of estimated data to initial data. The authenticity of estimated contents was difficult to prove, and indiscriminate estimation operations might cause redundant features.
- Improving interactive rate. The low interactive rate was usually caused by handling a huge workload of computations, such as processing large amounts of data, implementing accurate representations and performing complicated illumination distributions. Although much effort has been spent on data compression approaches, hardware acceleration techniques and algorithm simplifications, volume-based applications still struggled against the problem interactive rate in order to gain enough “room” to improve the system efficiency.
- Improving image qualities. In order to achieve the desired interactive rate, a few acceleration designs implemented rapid data processing by simplifying intermediate or final outputs. Artefacts and fuzzy features usually derived from decreased image qualities. According to actual demands of various applications, most of solutions proposed a proper balance to manage the trade-off between image resolutions and data processing times.

## Chapter 3 Volume Model Manipulation Strategies

In addition to developing visualization terms, constructing a novel data manipulation mechanism is another important research goal in this dissertation. By reviewing the state-of-the-art in the field of volume model manipulation, the pros and cons of each method will be analysed and evaluated. Afterwards, the problems and derived limitations inside deforming volume models will be concluded and treated as the corresponding research objectives of data manipulation.

### 3.1 Volume Clipping

As one of the simplest manipulation methods, the volume clipping technique can provide effective assistances in understanding volumetric contents, by enabling ichnography-like results according to the positions of “cutting points” (Weiskopf, Engel et al., 2003). In most of volume-based applications, it was considered as a complement to the TF, e.g. the cutting plan in medical imaging (Maruya, Nishimaki et al., 2010).

Most volume clipping methods are geometry-based methods, which determine the visibility of each voxel according to its position in the volume model. Depending on different ways of presenting clipping information, volume clipping techniques can be classified as clipping via tagged volumes, and depth-based clipping (Engel, Hadwiger et al., 2004).

The first method is based on pre-defining a clipping region (as shown in Figure 3.1). This method supports arbitrary clip geometries. However, it requires a series of accessorial operations for optimizing clipped results, e.g. using the tri-linear

interpolation or nearest-neighbour sampling for “smoothing” the clipping region for improving the display quality (Weiskopf, Engel et al., 2003). These operations generally cost extra processing time on interpolation computations.

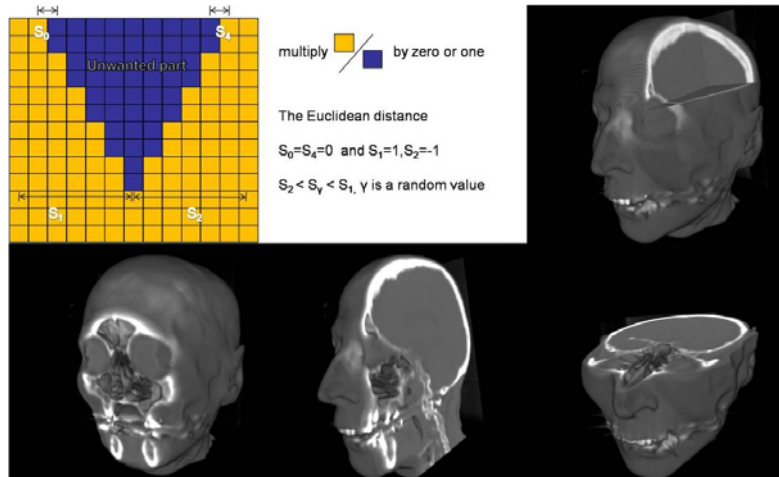


Figure 3.1 A diagram of the clipping method via tagged volumes

The second method relies on the stencil test (Westermann and Ertl, 1998). Technically, the depth-based clipping methods focus on partitioning the model into several fragments by using a stencil test to determine the distance of each portion from the image plane (shown in Figure 3.2). After labelling these fragments as visible and invisible, the volume data is respectively mapped to these partitions. As a result, these segments and included data sets can be correctly located, manipulated and visualized. The depth-based clipping method enables a flexible clipping geometry through taking advantage of its capability of managing the resolution of results at voxel level.

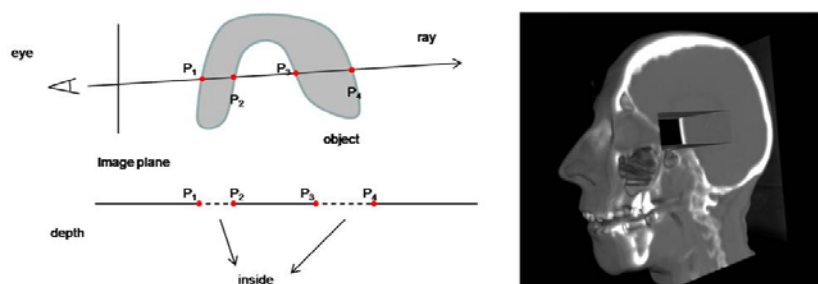


Figure 3.2 A diagram of the depth-based volume clipping method

## 3.2 Volume Deformation

Similar to the deformation approaches in surface-modelling-based applications (as shown in Figure 3.3), volume deformation techniques perform the shape changes on the volumetric objects so that the movements of the internal structures can react to the surface changes. For manipulating the volumetric structures, the deformation techniques need to build up a dedicated framework for connecting voxels and consequently synchronize the displacements of interior and exterior voxels properly according to specific applications. In this thesis, based on different types of the deformation behaviours, the volume deformation techniques were categorized into two types: non-physics-based and physics-based deformations.

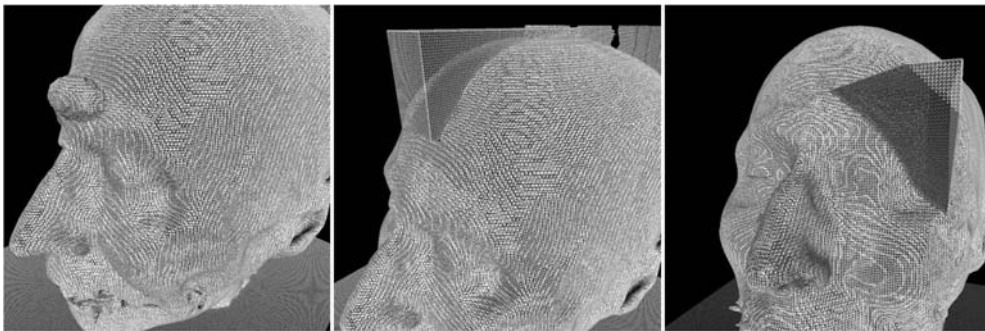


Figure 3.3 Illustrations of surface deformation models

### 3.2.1 Non-physics-based Deformation

Non-physics-based deformation means that the mechanism of generating deformation is not explicit, i.e. lacks the ability of performing physical characters in the deformed results. When applying these deformation techniques to manipulate the volume models, the most popular solution is to treat the volumetric information as a nesting-doll-like arrangement of surfaces or a stack of cross sections, and make a set

of linear deformation copies on these elements roughly. Basically, these deformation methods can be regarded as repeating a mesh deformation method in several times. This strategy can uniform the manipulations of all voxel so that the process of creating the connections between voxels can be overlapt. As a result, their associated applications can be implemented conveniently and output the deformed results rapidly. Due to these advantages, many researchers carried out different non-physics-based volume deformation approaches: ray-deflection volume deformation (Kurzion and Yagel, 1997), spatial TF method (Chen, Silver et al., 2003), and warping-based volumetric representation (Correa, Silver et al., 2010).

- Ray-deflection volume deformation

By employing the properties of voxels and specific visualization methods, this deformation manipulates the volume model in the manner of rendering. By changing the regular trajectories of sampling rays during the sampling phase, this deformation method can changed the resulting property values encountered on every ray. The deformation can present regional changes via four types of deflector (Kurzion and Yagel, 1997). The first one is a discontinuous deflector, which suffices for exhibiting interval-based clipping effects (in Figure 3.4 (A)). Image B shows the result of combining rotate and scale deflectors. Both deflectors are based on matrix multiplication or inversion to generate the deformed features on the corners of the enlarged mouth. Translate deflectors are used to assign certain displacements to voxels via a series of matrix algebra operations (as shown in image C).



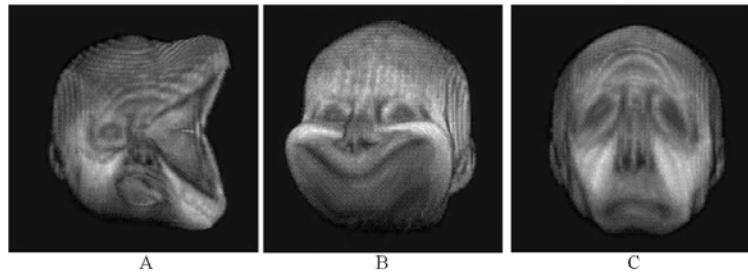


Figure 3.4 Illustrations of the ray-deflector volume deformation method (courtesy to Yail *et al.*)

- Spatial transfer function

The spatial TF serves as a “noise map”, which is constrained to anticipate into the regular conversion phase of translating scalar values into optical properties. This deformation method can perform the clipping result by defining a given parameter, which is used to indicate a (symmetrical) partition of “references” for the translating process (as shown in Figure 3.5 (A)). By accomplishing a many-to-many data mapping process, the spatial TF relies on a sweeping method for carrying out the stretching effects in image B (Chen, Silver *et al.*, 2003). However, the spatial TF cannot independently achieve resizing operations via a self-change of several parameters. Therefore, this method demands guidance from the control lattice construed for presenting shape changes, e.g. the results of a squeezing action in image C.

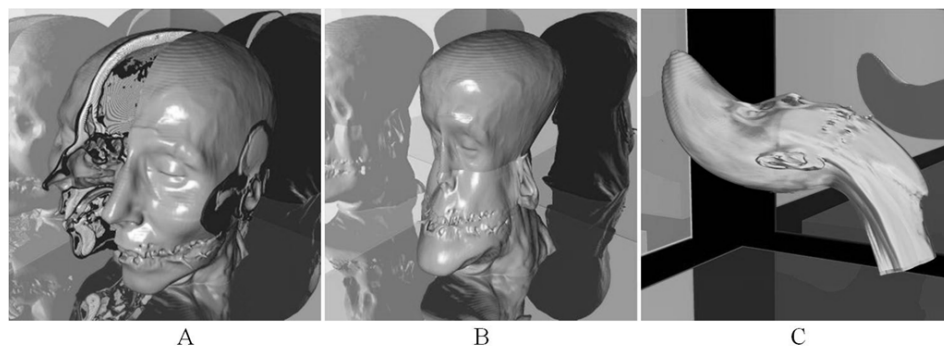


Figure 3.5 Results of the spatial TF method (courtesy to Chen *et al.*)

- Warping-based volumetric representation

These deformation methods transform the volume object by treating it as a homogeneous solid, which can perform uniform behaviours with respect to the given degrees of smoothness and partitioned organization. Technically, this deformation process is also based on the matrix manipulations, and all behaviours are determined by a warping-based matrix. However, because the lack of determining control lattices in the continuous volume data, the texture-based lattices were used to enclose the entire volume data including useless parts. As show in Figure 3.6, the idea of this deformation method didn't consider the connections between voxels (or groups of voxels). In this deformation, the "clone" operations were implemented through manipulating the texture-based lattices, and transforming their stored voxels (Correa, Silver et al., 2010). The shortcoming of this deformation process is the lack of differences between transforming soft tissues and shinbones to perform elastic behaviours shown in column A. The associated solution was realising a so-called sub-dividable deformation based on a series of constrained processes: offline data segmentation and accessorial restoring operations (as shown in column C and D) (Lewis, Cordner et al., 2000; Correa, Silver et al., 2010).

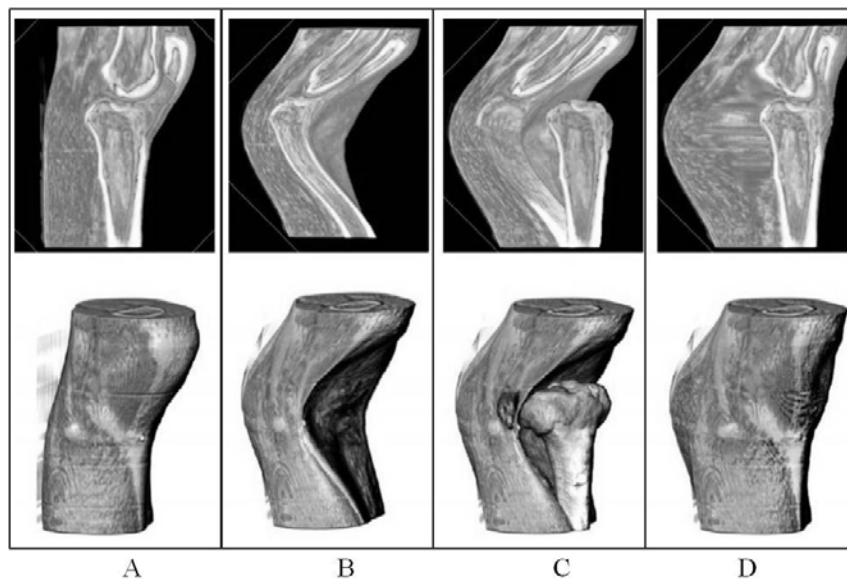


Figure 3.6 Illustrations of the warping-based volumetric representation method (courtesy to Carlos *et al.*)

### 3.2.2 Physics-based Deformation

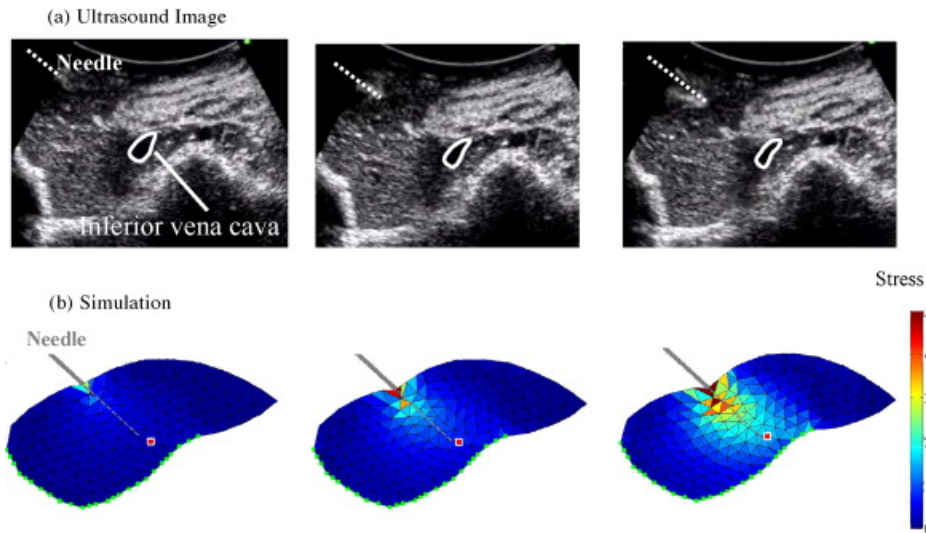


Figure 3.7 A diagram of a physics-based deformation method for medical simulations (courtesy to Kobayashi *et al.*)

As an interdisciplinary field, physics-based deformation techniques can involve Newtonian dynamics; continuum mechanics; numerical computation; differential geometry; vector calculus; approximation theory, and computer graphics. Because of their capability of presenting physical characters, such as gradient changes and realistic deformed features (as shown in Figure 3.7), these deformation methods were usually adapted for implementing Computational Fluid Dynamics (CFD) simulation (e.g. fluid-structure interactions) (Bathe and Zhang, 2009), representing irregular behaviours in soft tissues (e.g. operation training) (Kobayashi, Onishi et al., 2010), and transforming rigid bodies (e.g. bridge columns' responses to shaking events) (Baydaa, Ling et al., 2011).

For achieving Free-Form-Deformation (FFD), most of physics-based deformation applications required various control lattices for enveloping the models partially or

completely. These lattices were defined to orient the deforming forces and represent them on the underlying surfaces of objects to perform arbitrary and nonlinear deformation results. Physics-based volume deformation applications also relied on this principle and required the volumetric displacement mapping design for manipulating voxels. After the mapping process between control lattices and underlying voxels, the embedded volume needs to be converted into various deformable solids for determining the irregular grades of synchronizations of voxels' movements. These deformable solids are all constructed via choosing proper mathematical models as their stencils, and consequently enable a series of standardized connections between voxels. Based on these connections, the mathematical model can manage the spatial distribution of voxels within the volumetric space, so that the displacement for every voxel in the deformation area can be worked out. The review of physics-based deformation techniques was classified based on prevalent mathematical models, such as Mass-spring system (Provot, 1995), Finite Element Method (FEM) (Keeve, Girod et al., 1996), and Chain-Mail algorithm (Gibson, 1997).

- Chain-Mail algorithm

Chain-Mail algorithm is mainly used to simulate the spring and damping actions among large partitions by separating the object model into several blocks and computing their elastic changes during the deformation (Gibson, 1997). In volume-based applications, each block comprises a certain number of voxels (chain elements), which are surrounded by interspaces (chain regions) (as shown in Figure 3.8 (A)). The intersectional domains, which are assumed to exist between every two closest groups of voxels (chain elements), serve as the “air spring” regions and have three components along X-, Y- and Z- axial directions respectively (Li and Brodlie,

2003). According to predefined limitations of compressing and stretching the air spring region, the maximally compressed and maximally stretched conditions can be implemented respectively (as shown in Figure 3.8 (B and D)).

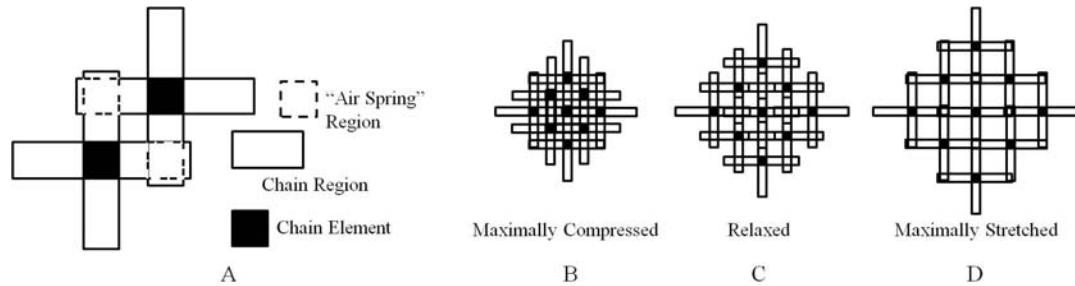


Figure 3.8 A diagram of Chain-Mail algorithm

During this deformation, applying a force on one chain element will cause a series of chain reactions in the form of a combination of magnified and minified “air spring” regions for simulating the results of the stretching (or compressing) operation. The closer the distance between the chain element and the centre of deformation area is, the clearer the changes of surrounding air spring regions are. Figure 3.9 shows different gradient distributions of deformable contents on a soft tissue in a virtual endoscopy application (Drager, 2005).

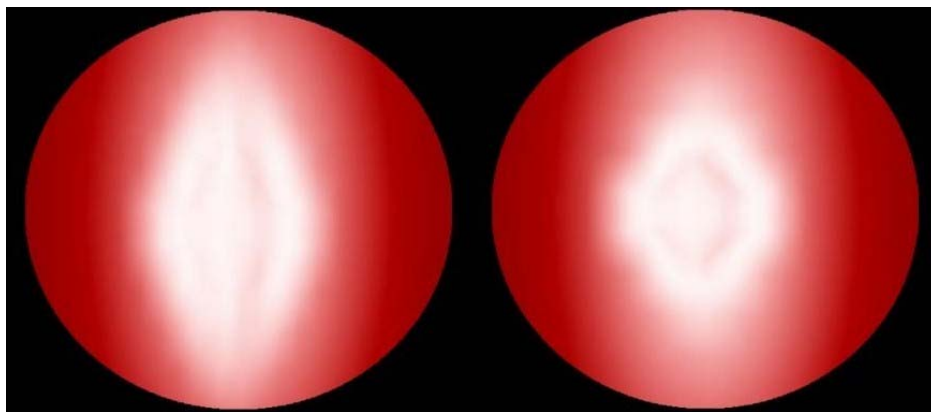


Figure 3.9 Results of Chain-Mail-based deformation (courtesy to Drager, C.)

- Finite Element Method

In order to perform perfectly continuous volume deformation, FEM utilizes the Partial

Differential Equation (PDE) to calculate the elastic behaviours' properties (Keeve, Girod et al., 1996). In volume-based applications, the FEM treated the volume as a continuously connected content with a random control lattice, and utilized simple equations to approximate the effect of running PDE on this lattice (Nienhuys and Stappen, 2000) (shown in Figure 3.10). This design usually focuses on the precise representation of the details of shape changes in training environments (e.g. surgical operation simulation (Nakao and Minato, 2010)), medical imaging (e.g. image-guided neurosurgery (Vigneron, Boman et al., 2008)), and engineering simulations (e.g. simulations of seismic pile-supported bridge structure (Baydaa, Ling et al., 2011)).

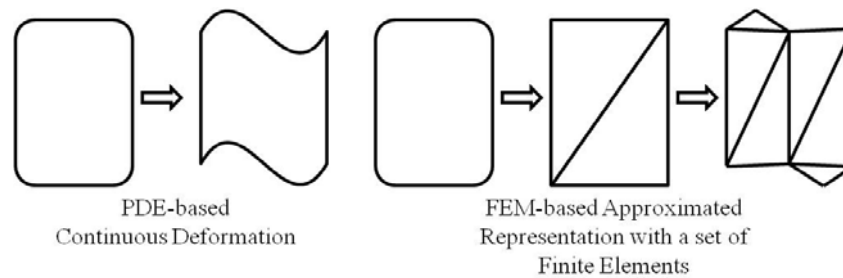


Figure 3.10 Diagrams of FEM-based approximate representation

For example, in order to perform the deformed features on the irregular grids intersected by the cut path (as shown in Figure 3.11 (a)), the FEM-based volume deformation subdivided the initial control lattices which surrounds the region of interesting segment (cut path), and approximated an intermediate representation in the form of divided grids (cut surface) (as shown in image b) (Nakao and Minato, 2010). Afterwards, the resulting elastic behaviours along the cut path can be approximated by the displacements of divided grids (as shown in image c and d). After filtering the subdivided lattice and mapping the vertices' properties to associated voxels, the preserved result and elastic performances of the cut region are respectively revealed in Figure 3.11 (A and B)

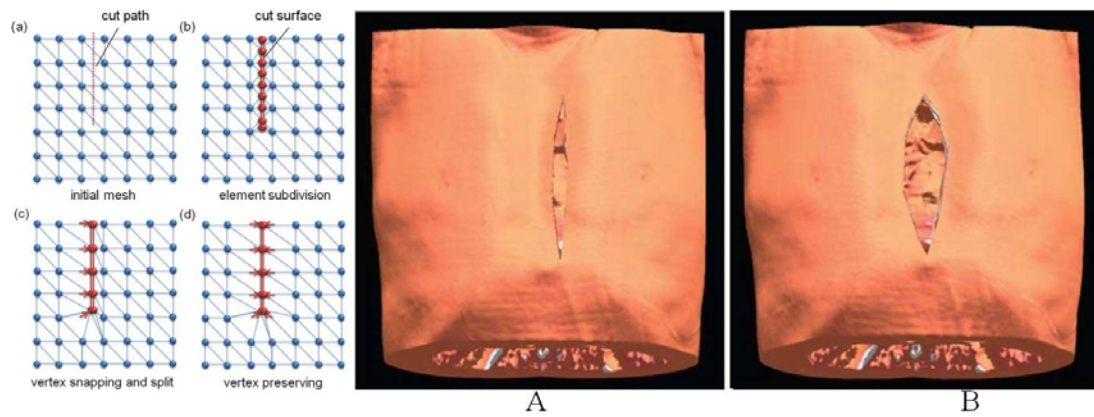


Figure 3.11 Illustrations of FEM-based volume deformation (courtesy to Nakao *et al.*)

### ● Mass-spring system

With the helps from the developing hardware accelerations (Tejada and Ertl, 2005; Courtecuisse, Jung et al., 2010) and optimization designs (Azar, Metaxas et al., 2002; Natsupakpong and Cavusoglu, 2010), the volume deformation techniques based on Mass-spring system had been improved representing deformations precisely and efficiently. Instead of approximating continuous deformation works and dynamic subdivision tasks in FEM-based volume deformations, the preparations for the construction of Mass-spring system involve partitioning the volumetric object into a set of voxel-based masses, and embedding a series of standardized spring-linking mechanisms (Natsupakpong and Cavusoglu, 2010) (shown in Figure 3.12).

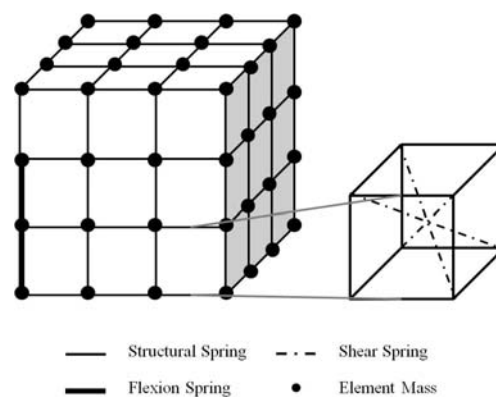


Figure 3.12 A diagram of Mass-spring system

Based on these spring-linking mechanisms, the deformed contents can be evaluated

by the movements of fixed masses (without subdivisions) (as shown in Figure 3.13 (A and B)). For example, in a breast survey application, the solution was proposed to convert the whole volume model into quite a few vertices. The devised Mass-spring system was built up based on these vertices, and consequently relied on a huge computation work and complicated transformations between vertices and voxels to produce the deformation and visualization results precisely (as shown in Figure 3.13 (C and D)) (Patete, Iacono et al., 2012).

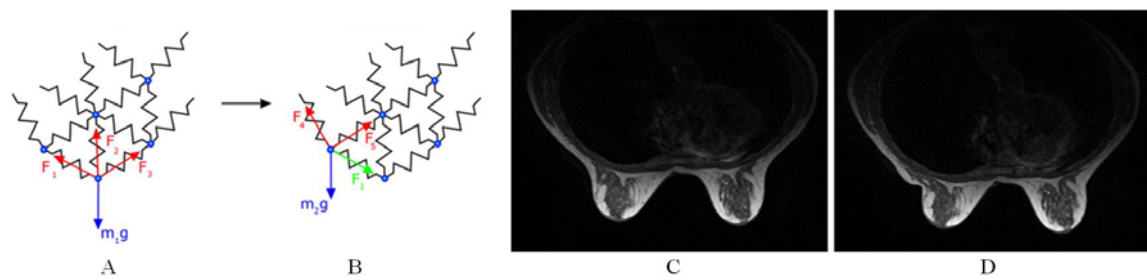


Figure 3.13 Illustrations of a volume deformation based on Mass-spring system (courtesy to Patete *et al.*)

### 3.3 Constructive Volume Geometry

For representing the complicated scenes consisting of multiple volume models, many researches utilized the surface modelling-based strategies as the solution. For example, a voxel-based version of so-called Constructive Solid Geometry (CSG), Constructive Volume Geometry (CVG) was devised to carries out voxel-based Boolean operations to represent the combination of multiple volume models (shown in Figure 3.14) (Chen and Tucker, 2000).



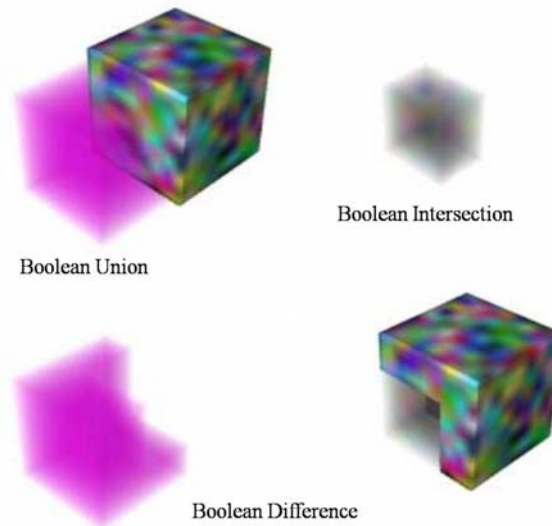


Figure 3.14 Diagrams of CVG-based Boolean operations (courtesy to Chen *et al.*)

This novel modelling method offers an algebraic framework for facilitating the combination operations on volumetric objects. CVG implemented a series of volumetric CSG models based on the union, intersection and difference operations which were built upon voxel-based manipulations with their scalar fields. For implementing various operations, there exist differently standardized classes which respectively indicate a set of computing methods for amending voxels' properties. As a result, both the opacity and colour properties of each voxel inside the crossed area were recomputed for performing the correct occlusions and the resulting colour overlays.

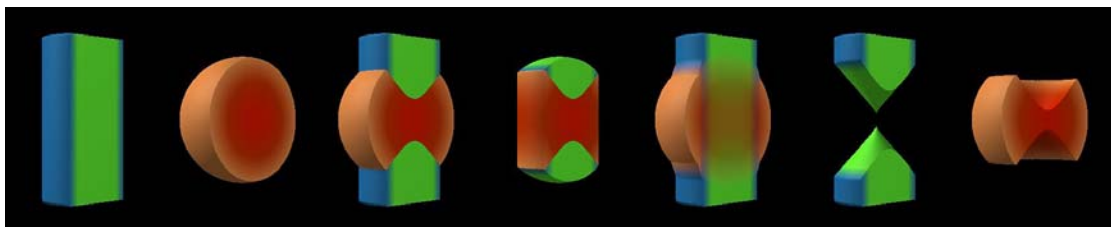


Figure 3.15 Results of CVG-based Boolean operations (courtesy to Chen *et al.*)

Based on this volumetric Boolean operation design, CVG-based operations could handle the complicated manipulations of multiple volumetric objects (as shown in

Figure 3.15). Constructing tree-like structures, managing memory requirements and designing hardware-based accelerations had been chosen as further solutions to the problems derived from the low processing efficiency caused by huge size of multiple volume data (Chen, Clayton et al., 2003).

### 3.4 Volume Animations

Like integrating polygon-based deformation methods with time lines to achieve animation efforts, some volume animation techniques also use time parameters to direct the execution sequence of volume deformation processes. In the past decade, research in volumetric animation has mainly focused on medical or hydro-mechanical applications (Binotto, Comba et al., 2003; Carmona and Froehlich, 2011). Figure 3.16 shows three snapshots of CT baby data.

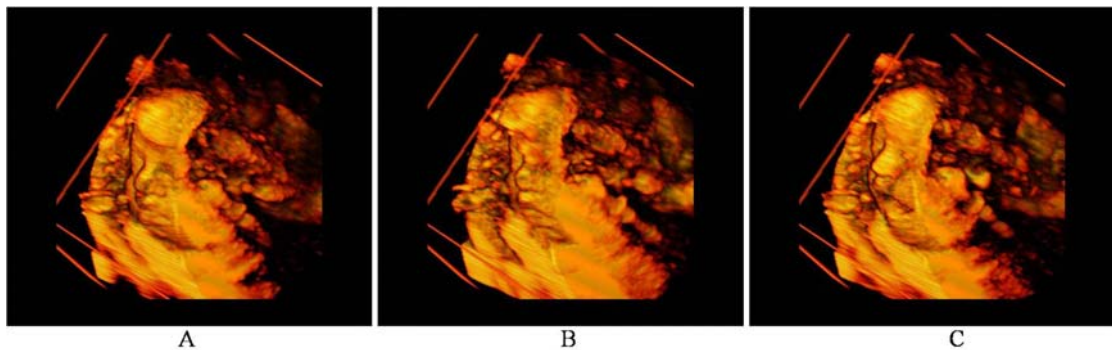


Figure 3.16 Illustrations of a dynamic CT baby data

Besides dedicated volume animation applications, the volume model was regarded as a special “agent” which delineates the polydirectional light interactions within volumetric spaces; e.g. the volumetric particle system consisting of a set of vertex-based elements (as shown in Figure 3.17). This particle system has shown its significant potential for performing visual effects in games and other entertainment industries. It was widely used to simulate smoke, explosions and fires (Green, 2005).

With the assistance of advanced hardware-based acceleration techniques, the volumetric particle system can enable high-quality particle simulation and maintain interactive displays with an increasing number of particles.

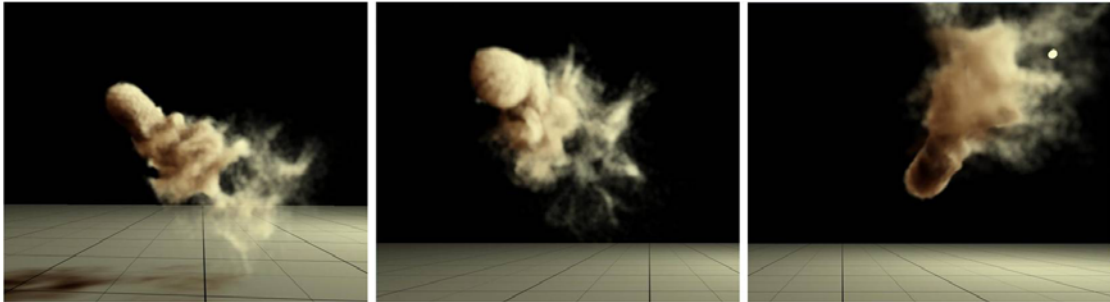


Figure 3.17 Illustrations of a volumetric particle system (courtesy to Green)

Since the 1990s, volume animation techniques have been used to load scalar CFD data, and improving photo-realistic images and animations with limited illumination effects for CFD simulations (as shown Figure 3.18 (A and B)) (Jaganathan, Tafreshi et al., 2008). Through benefiting from hardware-assisted acceleration supports, the volume animation techniques have been implemented and integrated with advanced illumination calculations to improve the authenticity and applicability of CFD simulations (as shown in Figure 3.18 (C and D)) (Corrigan, Camelli et al., 2011).

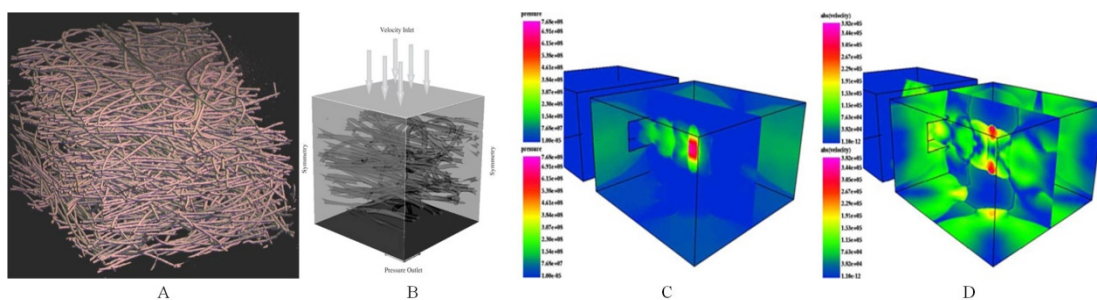


Figure 3.18 Illustrations of CFD simulations based on volume animation techniques (courtesy to Jaganathan et al. and Corrigan et al.)

### 3.5 Analysis on Current Challenges

By enabling non-physics-based deformation methods to accomplish complex deformation effects, many researchers have been trying to blur the line between non-physics-based and physics-based volume deformation methods, meanwhile, emphasizing the feasibility of using hardware-based acceleration techniques for maintaining the performance of improved non-physics-based methods. However, the performance of physics-based methods can also benefit from the hardware-based accelerations.

The human visual system can react to each frame individually when the frame rate is 10 to 12 frames per second (Reader and Meyer, 2000). And 14 to 24 frames per second (fps) can offer an impression of animation. If the frame rate is over 46 fps, the human visual system cannot react to this displaying speed (Elsaesser and Barker, 1990). In other words, when displaying the same object, the human visual system cannot detect any differences between 47 fps and 77 fps. Therefore, the naked-eye observations cannot perceive the differences between both hardware-accelerated non-physics-based and physics-based deformation methods whose displaying speeds are all over 46 fps.

Besides the decreasing difference between the real-time performances of non-physics-based and physics-based caused by the developing hardware techniques, the physics-based deformation has been becoming more popular than non-physics-based one because it has the advantage of performing the physical behaviours precisely (Choi, Lee et al., 2004; Bachmann, Bouissou et al., 2009; Kobayashi, Onishi et al., 2010).

In this thesis, the criteria for evaluating the designed deformation system won't contain the definite demands in realistic applications, such as calculating the precision attributes in millimetres and predetermining the error rate of simulation works (Patete, Iacono et al., 2012). As shown in Figure 3.19, the comparison table just uses four general criteria for the performance analysis of the above mentioned physics-based volume deformation techniques: Chain-Mail algorithm (represented by CMA), FEM method and Mass-spring system (represented by MSS).

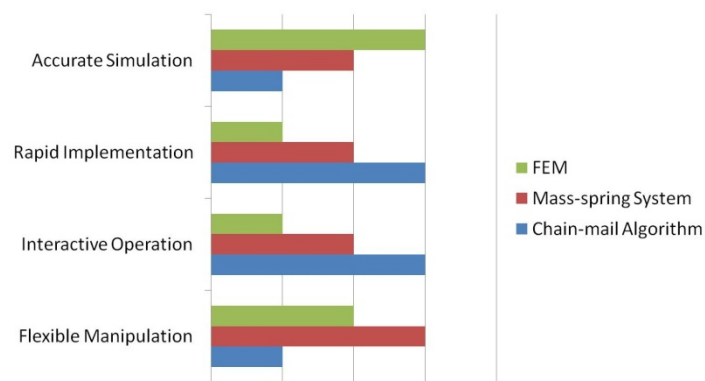


Figure 3.19 The comparison table of Chain-Mail algorithm, FEM and Mass-spring system's performances

As the greatest advantage of FEM-based volume deformation approaches, their applications are good at dividing the resulting deformed behaviours into tiny characters enough (even subdividing voxels), meanwhile only manipulate the interesting segment in volume data. Therefore, they are much popular in simulating rigid objects efficiently (Baydaa, Ling et al., 2011). Unless defining each voxel as a mass (or a chain element) in the MSS (or CMA) approaches and isolating the interesting data segment from the others, the resulting deformed behaviours cannot achieve the same precision as the FEM's results. The constrained construction of chain regions and limited movements of chain nodes makes CMA approaches only support the "coarse" motion of volumetric objects (Gibson and Mirtich, 1997). As a

result, CMA gets the lowest scores of “accurate simulation” and “flexible manipulation”.

However, the FEM-based deformation always suffered from the complicated subdivision mechanism which brings more complex computation works than the other two approaches. As a result, the low processing speeds make the real-time operations extremely difficult, and quite a few divided nodes badly restrict the flexibility of manipulations. Therefore, in both “rapid implementation” and “interactive operation” terms, FEM all gets the lowest scores. Because the simple node partition, CMA-based deformation applications can be rapidly implemented. And the CMA can only enable three statuses of the “air spring” region change. Therefore, CMA-based applications can support higher interactive rate of real-time operations than the other two approaches.

As a powerful and comprehensive deformation tool, MSS-based deformation methods have been extensively used for dynamic simulations, e.g. animation systems (Gibson and Mirtich, 1997). MSS is well-known for its plenty of connections which can form flexible frameworks and support multiform node partitions. As a result, MSS can perform more flexible results than FEM and CMA. Based on this evaluation, the devised volume deformation pipeline explained in the flowing chapters was constructed based on choosing the mass-spring system as the deformable solid, and comprised a series of solutions of improving the system performance of this design.

### **3.6 Design Criteria**

Based on the comprehensive review on the volume visualization techniques and manipulation strategies, the following design criteria for this research have been

adapted:

- First of all, the data processing stage should provide a visible analysis of any volume data set. The feasibility of the visible analysis should be demonstrated via the comparisons between automatically generated images and the results of offline manual modifications. Besides, the data processing stage also needs to provide the isolated data segments for simplifying the further processing works, e.g. enclosing the interesting data segment for reducing the lattice construction work and increasing the efficiency of the deformation system.
- The designed volume deformation method should manipulate volume objects to present deformed behaviours in a flexible and parameterized manner. By passing through an improved physics-based volume deformation approach, the resulting deformed volume models should reveal the gradient changes on their surfaces. As another outcome of this approach, the associated clipping planes should visually prove the existence of deformed interior contents and the stated coherences between exterior and interior changes. The feasibility of this deformation method should be experimented by loading different volume data sets.
- The devised volume deformation method in this project should support interactive manipulations of volume models during the real-time simulation. The gains from carrying out GPU-based implementations should be tabled in the form of given data arrays (comprising data sizes, associated parameters and frame rates). The image-based benefits of the parallel processing structure should be gathered together.

The following chapters all concentrated on improving the devised volume deformation system to meet the above criteria through overcoming corresponding

problems. As the output of the explained function module in chapter4, the extracted volume data structure and isolated data segments can enable the capability of automatic volume data analysis and flexible feature extraction. Based on these intermediate results, chapter 5 introduced a famous mesh extraction technique with associated refinement solutions to overcome the problems of control lattice construction in traditional volume deformation methods. These two chapters' achievements can help the system meet the first criterion. The second criterion of free-form volume deformation was achieved by the hybrid displacement mapping design (in chapter 6) which synchronizes the transformation for different elements. The deformation fixing function serves as the key complement of physics-based deformation process which can perform the deformation with physical properties. The third criterion requires the acceleration for achieving real-time performance. As a result, based on the GPU-accelerated designs in Chapter 7, the experimental results listed in Chapter 8 can quantify the speedup which claims the advantage of hardware-based acceleration, and exhibit final results of the devised volume deformation system to verify the feasibility of interactive operation of deformation behaviours.



## Chapter 4 Volumetric Data Processing

The volumetric data processing work introduced in this chapter aims to analyse the volume data and filter out its inherent noise by gaining the volume data structure and indicating the useless segments. As a pre-processing function module, this processing design relied on using volume segmentation methods to assist with the decision-making stage involved in manipulating volume models. In order to customize deformation behaviours, knowing about the framework of a volume model can instruct the partition and manipulation of volumetric contents. In addition, the “blank” voxels which serve as meaningless information and surround the interesting data segment(s), can be filtered out in this module and excluded from the subsequent operations to prevent processing time from being wasted on them.

Depending on the type of data processing, volume segmentation methods can be categorized into 2D (texture-based) and 3D (voxel-based) solutions. Basically, they all try to assign each voxel to a certain data segment labelled with a given mask (known as the segmentation mask). There were two types of segmentation mask. One was used to label each segment inside volume data in the form of an “ID” number. According to this ID number, the visualization process can accurately display the segment(s) with the associated optical properties in a specific TF module. The other segmentation mask was defined to make a clear spatial “nested relationship” from the volume data. The mask(s) determined whether voxels lie inside of the object

segment(s) in the manner of a binary representation. By utilizing these two masks, the volumetric data processing module can display an understandable result for analysing volume data (as shown in Figure 4.1(B)), and isolate the contents of interest selectively (as shown in Figure 4.1 (C)).

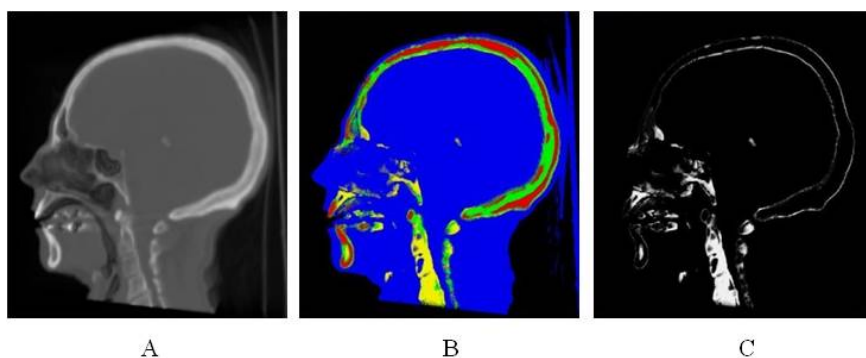


Figure 4.1 Display of a CT-scanned human head data. (Image A results from the basic DVR method. Image B simultaneously renders all parts of the head model. Image C uses the mask as a binary representation to delineate the “bony” and “boneless” parts)

The following content will cover the generation principles of these two segmentation masks with the associated usages of them.

## 4.1 Applying Clustering Methods for Classifying Volume Data

At the beginning of this devised volumetric data processing function module, the first step was to group “similar” voxels together and used the grouped result as the first kind of segmentation mask to guide the ensuing operations. Due to the fact that clustering algorithms are good at sorting pixels according to the predefined spectra, classifying voxels in volume segmentation process was implemented by extending all classical 2D clustering methods into 3D scenes. The original idea of clustering

algorithms is to group pixels by treating their coordinates and colour values as a feature space. The most popular feature space is of a 6D nature  $(x, y, r, g, b, \alpha)$ , in which  $(x, y)$  denotes the planar coordinates and  $(r, g, b, \alpha)$  represents the pixel's optical properties. In order to partition a volume data set in this process, the feature space became a 7D one, defined as  $(x, y, z, r, g, b, \alpha)$ .

In the field of clustering algorithms, there are three most prominent algorithms: connective-based algorithm (e.g. Hierarchical clustering), distribution-based algorithm (e.g. Expectation-maximization algorithm), and centroid-based algorithm (e.g. K-means clustering) (Rui and Wunsch, 2005). This section respectively implemented these typical clustering methods for classifying the same data set, and discussed the pros and cons based on their performance in volume segmentation process.

### **4.1.1 Hierarchical Clustering-based Volume Segmentation (HVS)**

As the most frequently used method for connectivity-based clustering applications, the hierarchical clustering method can be summarized as a binary tree during data clustering processes. Its leaves represent the data point and its internal nodes are the nested clusters of different sizes. Depending on the choice of data processing sequence, hierarchical clustering can be categorized into two types. One is called agglomerative hierarchical clustering, which observes the leaves firstly, and moves up to the nested clusters. Divisive hierarchical clustering method is the other type, which works in the opposite direction.

This research chose the agglomerative approach, which is more popular than the divisive one because of the traceability, as the basis of the HVS design (Cimiano, Hotho, et al., 2001). The clustering algorithm started from the voxel level. For a random voxel  $P_{voxel_i}$  belonging to a volume data set ( $P_{voxel_i}$  and  $i \in N$ ), there was a nearest voxel  $P_{voxel_{near}}$  which was paired with  $P_{voxel_i}$ . The mechanism of pairing can be written as (Szekely and Rizzo, 2005):

$$\|\overrightarrow{P_{voxel_i} P_{voxel_{near}}}\| = \min\{\|\overrightarrow{P_{voxel_i} P_{voxel_j}}\|\} \quad (\forall j, j \in N \text{ and } j \neq i) \quad (4.1)$$

The next step was merging  $P_{voxel_i}$  and its  $P_{voxel_{near}}$  into a new “voxel”, i.e. obtaining the new voxel’s properties in the form of calculated  $\overrightarrow{P_{voxel_i} P_{voxel_{near}}}$ . As a result, this generated “voxel” represented the nested cluster for both  $P_{voxel_i}$  and  $P_{voxel_{near}}$ , and joined in the next pairing operation with the other clusters. Because there was no fixed limit on the number of the first merged clusters, several voxels might miss the first pairing stage and be paired with generated clusters. By iterating the pairing and merging steps, HVS continuously generated cluster-based results, and the final result approximated to two or three clusters. As a derived result of the hierarchical clustering method, a kind of binary tree was produced to record the nested relationship of clusters (as shown in Figure 4.2 (A)). This clustering sequence led to a series of intermediate outputs (as shown in image b to f)), which represented the clustering results from Level 0 to 4. These images represented the progress of clustering volume data. Its mechanism can be written as shown in Table 4.1.

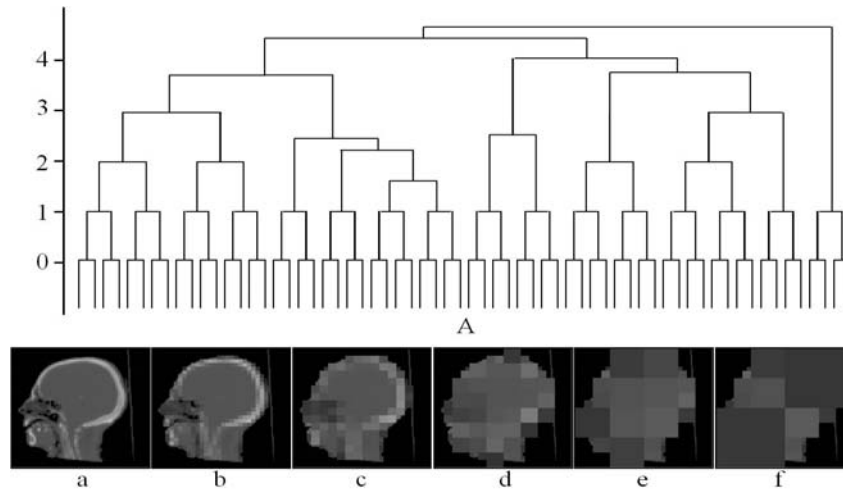


Figure 4.2 The results of HVS.

<p><i>Defining the object voxel <math>P_{voxel\_obj}</math>, and neighbouring voxels <math>\{P_{voxel\_i}\} (\forall i, i \in N \text{ and } i \neq obj)</math></i></p> <p><i>Defining the number clustering cycle <math>N</math>, and 1D array <math>I[j] (j \in [0, N])</math></i></p> <p><i>Defining the minimum distance for recording <math>Dist_{min}</math></i></p> <p><i>If <math>N &gt; 1</math> //the starting point of HVS is not the whole volume model</i></p> <p style="padding-left: 2em;"><i><math>I[N] = voxel\_obj</math> //recording the clustering sequences</i></p> <p style="padding-left: 2em;"><i>Calculate <math>Dist_{min}</math> in equation 4.1 //seeking the closest <math>P_{voxel\_nei}</math></i></p> <p style="padding-left: 2em;"><i><math>P_{voxel\_obj} = \overline{P_{voxel\_obj} P_{voxel\_nei}}</math> //pairing and merging the object voxel with this neighbour</i></p> <p style="padding-left: 2em;"><i><math>N = N - 1</math></i></p> <p style="padding-left: 2em;"><i>Return <math>I[N]</math> //Output the sequence of clustering</i></p> <p style="padding-left: 2em;"><i>Else Ending the clustering loop</i></p>
---

Table 4.1 Mechanism of HVS

### 4.1.2 Expectation-Maximization Algorithm-based Volume Segmentation (EMVS)

Among all distribution-based clustering algorithms, the expectation-maximization

(EM) algorithm is the most prominent one; it treats the object data as a statistical model and searches the given parameters (called maximum likelihood parameters) to estimate the resulting clusters. This statistical model is initialized as empty in the beginning of each clustering loop and used to contain the sampled data constantly. The resulting representation is a series of discrete distribution of corresponding points. After finishing a cluster, besides the emptied statistical model, the reset data can be divided into un-sampled data (available for one of the others clusters) and latent variable (undetectable parts in sampling process). As the information loss in EM clustering algorithm, the latent variables won't belong to any cluster. In addition to the latent variables, the rest data will anticipate in the calculation of maximum likelihood parameters.

Similar to the strategy of analysing 2D data sets in EM algorithm, EMVS also regarded the volume data as a statistical model. The number of distributions (or clusters) was predefined as  $N$ . The volume data was mathematically described by a set of voxel coordinates  $Vo\_co(x, y, z)$  with their scalar values  $Vo\_sc$  which formed a probability distribution function together:

$$p(Vo\_co_i, Vo\_sc_i | \partial) = p(Vo\_co_i | Vo\_sc_i, \partial) p(Vo\_sc_i | \partial) \quad (4.2)$$

where  $Vo\_co_i$  and  $Vo\_sc_i$  respectively represent partial coordinates and a scalar value of a random voxel belonging to the  $i_{th}$  distribution, and  $\partial$  is initialized to a nonzero parameter.

In order to extract the maximum likelihood parameters from the distributions, EMVS

relied on two alternating steps: expectation step and maximization step. The first step calculated an expected value  $M_i$  of the log likelihood function of the  $i_{th}$  distribution with respect to  $\partial$ . The corresponding log likelihood function can be written as (Dempster, Laird et al., 1977):

$$M_i = \log p(Vo\_co_i, Vo\_sc_i | \partial) = \sum p(Vo\_co_i | Vo\_sc_i, \partial) p(Vo\_sc_i | \partial) \quad (4.3)$$

After calculating the results of the log likelihood function ( $M_1, M_2 \dots M_N$ ) for all distributions, the second step was to select the maximum one as a new  $\partial$ , which was loaded in equation 4.3 for generating new expected values. This iteration of these two steps in the designed EMVS can be represented in Table 4.2.

<p><i>Defining the number of clusters <math>N</math>, voxel coordinates <math>Vo\_co</math>, voxel's scalar values <math>Vo\_sc</math>, a random parameter <math>\alpha \neq 0</math>, and a set of parameters <math>M_1 = M_2 = \dots M_N = 0</math></i></p> <p><i>For <math>i = 1</math> to <math>N</math></i></p> <p style="padding-left: 40px;"><i>Calculate <math>M_i</math> in equation 4.3</i></p> <p><i>Return <math>\{M_1, M_2 \dots M_N\}</math></i></p> <p style="padding-left: 40px;"><i>If <math>(\alpha - \max\{M_1, M_2 \dots M_N\}) \neq 0</math></i></p> <p style="padding-left: 80px;"><i><math>\alpha = \max\{M_1, M_2 \dots M_N\}</math></i></p> <p><i>Recalculate <math>M_i</math> with new <math>\alpha</math></i></p> <p style="padding-left: 40px;"><i>Else</i></p> <p><i>Return <math>(Vo\_co_i, Vo\_sc_i)</math> //output the clustered result</i></p>
--

Table 4.2 Mechanism of EMVS

After testing this EMVS on a volume data set (human head data), the resulting clustered information can be shown in Figure 4.3. Image A is the original data. Image

B, C and D are the clustered results when the number of distributions equal 3, 4 and 5 respectively.

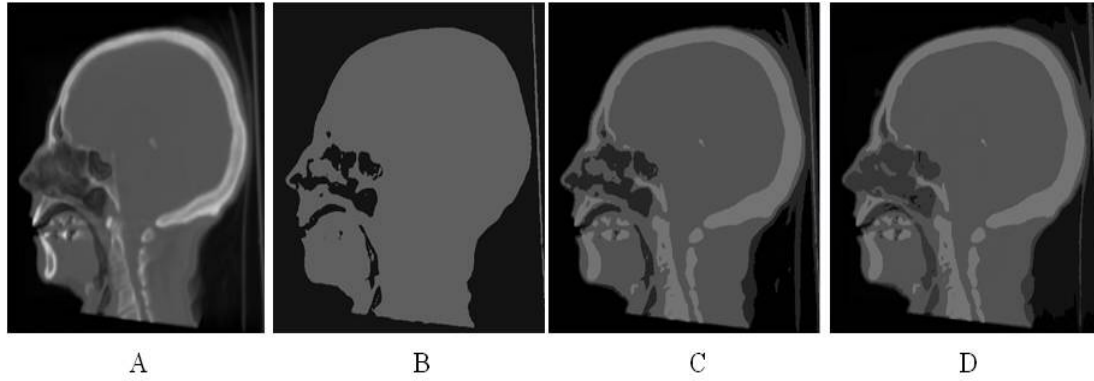


Figure 4.3 The results of EMVS.

#### 4.1.3 K-means Clustering-based Volume Segmentation (KMVS)

In order to segment a volumetric data set into a set of clusters, KMVS delineated purposely shaped clusters by locating their centroids based on the famous K-mean clustering algorithm (Kanungo, Mout, et al., 2002; MacKay, 2003). Similar to the demands in the EM algorithms, there was a predefined cluster number, which is usually represented by a  $K$ . KMVS assigned all voxels to  $K$  clusters through implementing a frequentative calculation of two important distances: distance for assignment and distance for update.

- Distance for assignment

In the beginning, KMVS acquired the predefined cluster number  $K$ , so that there was several randomly initialized centroids,  $P_{cent\_j}$  ( $j \in K$ ). After obtaining the distances

$\| \overrightarrow{P_{voxel\_i} P_{cent\_j}} \|$  from each voxel  $P_{voxel\_i}$  to all centroids  $P_{cent\_j}$ , the next step



was keeping the minimum distance  $\|\overrightarrow{P_{voxel_l} P_{cent_j}}\|$  for guiding the assignments of voxels to the related cluster of which the centroid is  $P_{cent_j}$ . Based on the Kanungo's idea of judging the minimum pixel-based values, the similar operation was created to judge the shortest distance between the object voxel and each cluster's centroid. It can be written as:

$$\|\overrightarrow{P_{voxel_l} P_{cent_j}}\| = \min \{ \|\overrightarrow{P_{voxel} P_{cent_1}}\|, \dots, \|\overrightarrow{P_{voxel} P_{cent_K}}\| \} \quad (4.4)$$

- Distance for update

After this, the next process was to create a "breakpoint" to check whether the current clustered result was the final result or still unstable output. The breakpoint was based on the distance which connected the centroid  $P_{cent_j}$  and the mean point  $P_{mean_j}$  of the cluster, assuming the number of voxels belonging to the cluster was  $M$ . After the initial assignment stage, the mean point can be calculated as in equation 4.5.

$$P_{mean_j} = \frac{1}{M} \sum_1^M P_{voxel_i} \quad (4.5)$$

The calculated  $\|\overrightarrow{P_{cent_j} P_{mean_j}}\|$  was regarded as the distance for update. In 2D-based KM clustering applications, there usually exists a kind of mechanism which switches off the cycle as soon as the distance for update equals to a predefined value. The cycle of updating centroids in KMVS was controlled by the judgement of whether the  $\|\overrightarrow{P_{cent_j} P_{mean_j}}\|$  equals zero. The mechanism of KMVS can be written as follows:

Defining the voxel  $P_{voxel_i}$ , the centroid of a cluster  $P_{cent_j}$ , the mean point  $P_{mean_j}$ , the number of clusters  $K$

Initializing the distance between voxel and centroid  $Dist_{voxel\_cent}$

Defining a fixed distance  $Dist_0$

If  $\| \overrightarrow{P_{cent_j} P_{mean_j}} \| > Dist_0$  //defining the breakpoint

$P_{cent_j} = P_{mean_j}$  //regarding the mean point as the new centroid

If  $\| \overrightarrow{P_{voxel_i} P_{cent_j}} \| \neq 0$  //sampling at the centroid

Locate  $P_{cent_j}$  in equation 4.4 //calculating the distance for assignment

Assign  $P_{voxel_i}$  with the located  $P_{cent_j}$

Calculate  $P_{mean_j}$  in equation 4.5 after assignments

Update  $P_{mean_j}$  in the breakpoint

Else

Return  $P_{cent_j}$  // output the latest centroids

Table 4.3 Mechanism of KMVS

By increasing the  $K$  value from 3 to 5, the clustered result will contain more features by detecting new clusters, be respectively shown in Figure 4.4 (B, C and D).

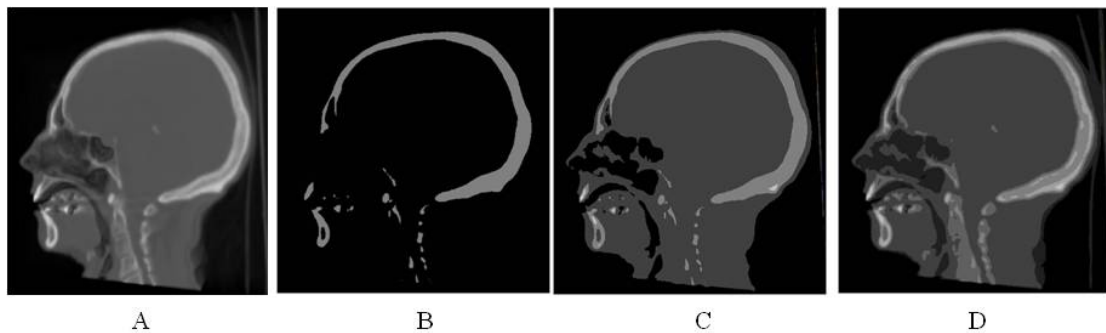


Figure 4.4 The results of KMVS.

#### 4.1.4 Evaluations of Segmentation Approaches

As volumetric applications of clustering algorithms, HVS, EMVS and KMVS successfully divided the volume data into segments, and the resulting snapshots were respectively shown in Figures 4.2, 4.3 and 4.4, also demonstrated the feasibility of developing 2D algorithms for analysing 3D datasets. In order to choose the optimal volumetric clustering method, the criteria for comparing the performance of the above three segmentation designs were based on the traditional evaluation strategies (Rui and Wunsch, 2005), and summarized as below:

- Efficiency

When classifying the same data set, all processing times were recorded in Table 4.4, according to different cluster numbers. As a pre-processing function, any processing time cost by the volume segmentation design can be acceptable. No matter how the  $K$  may be changed, HVS will cost more processing time than the others.

Cluster Number ( $K$ )	Processing Time (s)		
	HVS	EMVS	KMVS
3	179	152	144
4	316	271	264
5	571	447	438

Table 4.4 Comparison between processing times

- Applicability

Based on the Wunsch's survey, the applicability of clustering method is estimated via timing the classification of different data sets with the same clustering property. Analysing the below tabled processing time can draw a corresponding conclusion that HVS is the slowest clustering function, i.e. HVS owns the lowest applicability among these three segmentation designs.

Volume Data (KB)	Processing Time (s) and $K = 5$		
	HVS	EMVS	KMVS
Human Head Data (27.1k)	571	447	438
Engine Data (7.0k)	217	129	114
Teddy Bear Data (0.9k)	53	31	27

Table 4.5 Comparison among performances of processing different data

- Functionality

Based on the Kanungo's research work, the evidence of functionality is based on the resulting clusters for different applications. In order to generate the visible analysis of volume data, the result should offer a straightforward image, not an indistinct one.

As shown in Table 4.6, increasing  $K$  can improve the fuzzy results of HVS. However, this operation will bring a geometric rate of growth of calculation works. Although the nested relationships derived from the HVS process can be used for information retrieval applications, its clustered results are not suitable for displaying a

visible framework of the volume data, and the time-consuming processing will restrict the system performance. As a result, HVS was discarded in this volume data processing stage.



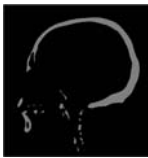

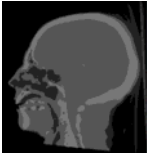


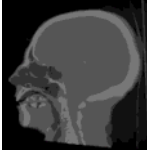
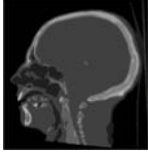
Cluster Number ( $K$ )	Clustering-based Volume Segmentation		
	HVS	EMVS	KMVS
3			
4			
5			

Table 4.6 Comparison among three kinds of clustered results.

In this volumetric data processing function module, the clustering-based segmentation designs are all expected to not only extract the volume data structure, but protect the processed results against artefacts and information loss.

Even though both EMVS and KMVS can obtain similar performances (such as clustered results and processing time), there are several differences between these two methods. Table 4.7 shows the results of using EMVS and KMVE to analyse different data sets with the same cluster number.

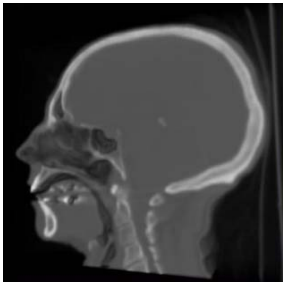
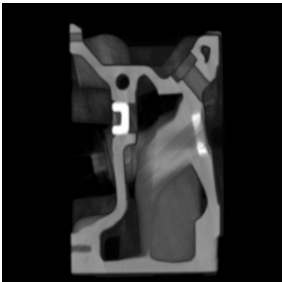
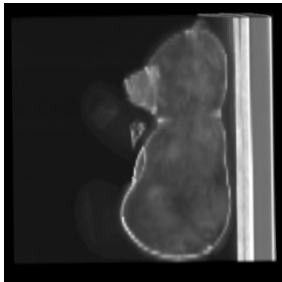
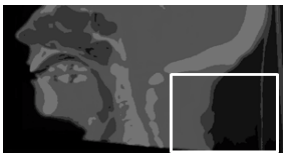
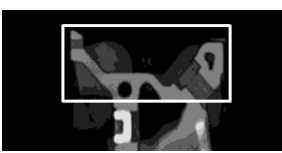


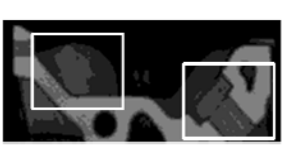

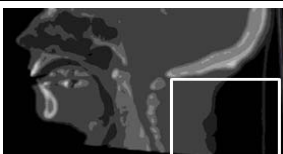
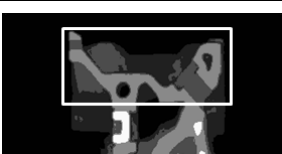




Clustering Method	$K = 5$		
	Human Head Data	Engine Data	Teddy Bear Data
			
EMVS			
			
KMVS			
			

Table 4.7 Comparison between the results of EMVS and KMVS

When segmenting the human head data, as shown in the highlighted regions, the result of EMVS contained a segment which was obtained by subdividing the background information (not contained in KMVS). More precisely, one of the clusters in EMVS was used to gather useless segment without any information regarding the head model. It can be observed that EMVS was more sensitive to noise than KMVS.

When segmenting the engine data, the EMVS result contained a large amount of

fuzzy information in each cluster which will influence the display qualities of visualizing results.

In addition, similar problems can be found in the processing of the teddy bear data. In the associated results, a few pieces of meaningful information inside the nose area were missed by EMVS. By comparing the highlighted features in the results for clustering the engine data, it can be found that KMVS is more suitable for generating the segmentation mask because its outputs can be directly rendered for labelling the clustered volume data segments.

After completing the above comparisons, KMVS was chosen as the volumetric clustering method in volume data processing stage. However, a series of problems derived from this choice. Firstly, KMVS relied on manual inputs of the cluster number  $K$ . This manual input operation led to unavoidable breaks during real-time applications. Secondly, incorrect inputs brought insufficient segmentation. More precisely, the clustered results contained segment(s) due to the incorrectly defined cluster numbers. Besides these two problems, there existed an issue of the greyscale representation of the results of KMVS, which was similar to the greyscale definition of images in 2D space, and only used shades of grey to render clustered results in the multidimensional space. The greyscale volume led to insufficient representation of complicated information with the increase of the cluster number.

## 4.2 Segmentation Improvement and Cluster Representation

### 4.2.1 Mean-Shift Clustering-based Volume Segmentation (MSVS)

In order to overcome the above mentioned problems of using KMVS, an automatic volumetric clustering method was proposed, which served as a self-driven extraction of cluster numbers for classifying the volume data, based on the idea of Mean-Shift (MS) clustering algorithm (Comaniciu and Meer, 2002). As a non-parametric technique, MS algorithm was used to analyse complicated data sets and draw the clusters automatically. “Non-parametric” means that the initialization of the MS clustering method is “one-off”, i.e. there is no need to define the cluster number before every clustering cycle.

In order to create MSVS, the first step was defining a kernel density estimator, which served as the boundaries of every iteration cycle whilst traversing through the volume data set ( $\text{voxel}_i, i \in [0, n)$ ). Similar to partial initialization designs in KMVS, there existed a multi-dimensional space,  $R^6$  consisting of coordinates and inherent scalar value. Therefore, the kernel density estimator  $\hat{f}(\text{Voxel})$  was multivariate and relied on a radially symmetric kernel  $K(\text{Voxel})$  (defined as a spherical domain in MSVS).

Based on studying the mechanism in classic 2D approach developed by Comaniciu, MSVS was built up through imitating the every process in MS algorithm and designing the 3D approach to access and calculate voxels' properties for volume-based applications. Therefore, the volumetric computation model can be written as:



$$\hat{f}(Voxel) = \frac{1}{nR^6} \sum_{i=1}^n K\left(\frac{Voxel - voxel_i}{R}\right) \quad (4.6)$$

where  $R$  is the radius of the kernel. The kernel  $K(Voxel)$  can be written as:

$$K(Voxel) = \frac{1}{\sqrt{2\pi^6}} k(\|Voxel\|^2) = \frac{1}{\sqrt{2\pi^6}} e^{(-\frac{1}{2}\|Voxel\|^2)} \quad (4.7)$$

where  $\frac{1}{\sqrt{2\pi^6}}$  is the normalized constant of  $K(Voxel)$ . Employing the kernel

$K(Voxel)$ , the equation 4.6 can be written as:

$$\hat{f}_{R,K}(Voxel) = \frac{6}{\sqrt{2\pi^6}nR^6} \sum_{i=1}^n k\left(\left\|\frac{Voxel - voxel_i}{R}\right\|^2\right) \quad (4.8)$$

The gradient of the kernel density estimator can be written as:

$$\nabla \hat{f}_{R,K}(Voxel) = \frac{12}{\sqrt{2\pi^6}nR^8} \sum_{i=1}^n (Voxel - voxel_i) k'\left(\left\|\frac{Voxel - voxel_i}{R}\right\|^2\right) \quad (4.9)$$

which yields:

$$\begin{aligned} & \nabla \hat{f}_{R,K}(Voxel) \\ &= \frac{12}{\sqrt{2\pi^6}nR^8} \left[ \sum_{i=1}^n -k'\left(\left\|\frac{Voxel - voxel_i}{R}\right\|^2\right) \right] \left[ \frac{\sum_{i=1}^n -voxel_i k'\left(\left\|\frac{Voxel - voxel_i}{R}\right\|^2\right)}{\sum_{i=1}^n -k'\left(\left\|\frac{Voxel - voxel_i}{R}\right\|^2\right)} - Voxel \right] \end{aligned} \quad (4.10)$$

The function  $\left[ \frac{\sum_{i=1}^n -voxel_i k'\left(\left\|\frac{Voxel - voxel_i}{R}\right\|^2\right)}{\sum_{i=1}^n -k'\left(\left\|\frac{Voxel - voxel_i}{R}\right\|^2\right)} - Voxel \right]$  determined the shift vector  $M$

of kernel density function, i.e. represented the change of mean values of voxels inside

the kernel. The mechanism of moving the kernel density estimator MSVS can be

written as:

*Defining the voxel  $voxel_i$ , the radius of kernel  $R$ , the location of the kernel density estimator  $L_K$ , the gradient of the kernel density estimator  $\nabla \hat{f}_{R,K}(Voxel)$*

*Calculate  $\nabla \hat{f}_{R,K}(Voxel)$  with respect to  $L_K$  in equation 4.10*

*If  $\nabla \hat{f}_{R,K}(Voxel) \neq 0$*

*Gain the shift vector  $M$  in equation 4.10 //calculating the displacement of kernel density estimator*

*$L_K = L_K + M$  //moving kernel density estimator*

*Return  $L_K$  to the calculation of  $\nabla \hat{f}_{R,K}(Voxel)$*

*Else //finishing the convergence work for every cluster*

*End*

Table 4.8 Mechanism of MSVS

With the movement of the kernel density estimator, the mean value of voxels inside the kernel was calculated as the centroid of a cluster after every shift stage. The results of MSVS are shown below:

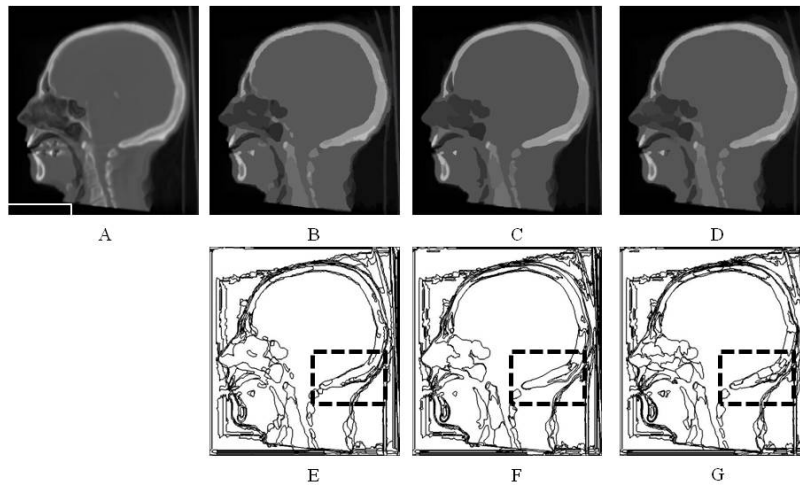


Figure 4.5 The results of MSVS.

Image A is the original data set. Image B, C and D are the results when the radius of kernel  $R$  equals 28, 14 and 7 respectively. As shown in the labelled regions in Figure 4.5 (E, F and G), minimizing the radius of spherical kernel density estimators in MSVS led to the subdivision processes. These subdivision processes brought in a sort of measurement error which was known as over-segmentation, and occurred by dividing one cell into two (or more) segments.

As mentioned at the end of section 4.1, KMVS also suffered from a kind of measurement error – under-segmentation – which meant one segment covers two (or more) cells. For example, in Figure 4.4 (B, C and D), the clustered result will show more clusters with the increase of the cluster number.

In many segmentation applications, over-segmentation was regarded as a much simpler problem than under-segmentation because it can be easily solved through “merging” operations in future processes (e.g. the displaying stage). However, in order to fix the under-segmentation, the algorithm needed a series of modifications and calculations for subdividing the current result. In addition, over-segmentation was good at generating the abundantly clustered information, so that incomplete volume data analysis in KMVS can be fixed in MSVS. Therefore, in this volumetric data processing module, MSVS served as a previewer which can automatically execute a rough analysis in the manner of producing an excess cluster number, and this number was directly used as the input of  $K$  in KMVS. This strategy is using the KMVS’s low sensibility for noise to merge the over-segmented parts. Meanwhile, this design

also prevented the segmentation process from the dependence of manual operations (e.g. merging the over-segmented parts manually).

#### **4.2.2 Design of Automatic Transfer Function (ATF)**

As mentioned in section 2.1.2, the survey of TF design in DVR-based applications claimed that multidimensional TF designs can improve the ability to isolate regions of interest, or to represent differences between features. Therefore, the volumetric data process module chose the multidimensional TF for rendering the clustering results. Besides this, the survey also emphasized the overwhelming task of configuring the multidimensional TF, and the demands for automatic or semiautomatic design. In order to facilitate the visualization of large volume data, this module employed the ATF design to automatically configure the flexible rendering mechanism. This ATF design was a kind of data-driven technique which treated the final outputs of KMVS as the parameters prepared for constructing a multidimensional TF.

First of all, the initial relationship between the scalar value and the opacity value in the TF can be shown in Figure 4.6 (A). In the results of clustering-based segmentation, the final cluster number  $K$  was used to divide this continuous information into  $K$  partitions randomly, and form a histogram (as shown in Figure 4.6 (B)). Afterwards, the next step was configuring the proportion of each partition in this diagram.

Besides the cluster number, the scalar value of a cluster centroid was used to determine the height of the corresponding rectangle. For example, both regions of the

skull and teeth obtain the same intensity value, which is recorded as the scalar value in the data acquisition period. Therefore, their clusters share the same scalar value. Without considering their coordinates, these two clusters can be assigned with the same rectangle, and rendered in the same way. In order to avoid this inefficient rendering, the coordinates of centroids were utilized to distinguish clusters which share the same scalar value. In addition, the amount of voxels in each cluster was converted into the width of rectangle.

After finishing the above mentioned data-driven configurations, the resulting histogram can be shown in Figure 4.6 (C). By accomplishing these data-driven configurations, this ATF design can render original volume data as one of important results of the volumetric data processing module (shown in Figure 4.6 (D)).

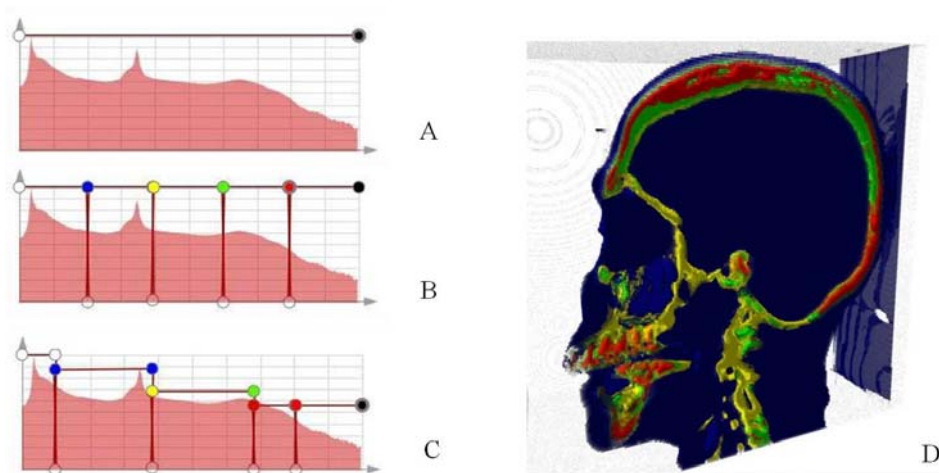


Figure 4.6 Diagrams of ATF design and the result of analysing a CT-scanned human head data

### 4.3 Designs of Boundary Extraction

After finishing the automatic visualization process which extracted the segmentation

mask to label internal structure of the volume data, the boundary extraction design, which served as a complement to the data segmentation process in this module, was constructed to detect the boundaries of interesting segment(s).

### 4.3.1 Active Contour Algorithm

Active Contour (otherwise known as Snake) is a method that delineates the outline of features in image space. According to the different types of the outline mode, this algorithm can be categorized into edge-based (Xu and Prince, 1998) and region-based (Du and Bui, 2008) methods. In order to enclose the features, all these methods rely on a deformable spline which is characterized and transformed by these two constraints: interiors and exteriors. The interior constraints are determined by the material properties of the spline, such as the mass distribution parameter and the viscosity of the neighbouring medium. The exterior ones represent a sort of state that the spline is constrainedly transformed according to the calculation of external factors. For example, after exerting a force on the spline, the resulting constraints (interior and exterior) will determine the change of sampled region based on an energy function. This function is calculated by following the equation for Energy Minimization (Kass, Witkin et al., 1988):

$$E(V) = \int_0^1 (E_{internal}(V) + E_{external}(V)) dV \quad (4.11)$$

where  $E_{internal}$  is the internal energy of the transformed spline and  $E_{external}$  serves as external energy acting on the spline.

For example, in order to detect the cavity segment from the human celiac slice, the

active contour function firstly initialises the spline and continuously compares the resulting energies. The relation between this spline and its surrounding pixels is parameterised as the exterior constraint. In equation 4.11, the corresponding external energy  $E_{external}$  is replaced by the gradient of pixel scalar value for simplifying the calculation work. Similarly, the  $E_{internal}$  represents the internal constraint and equals the number of pixels inside the spline divided by ten thousand. This configuration way is to increase the proportion of exterior constraint.

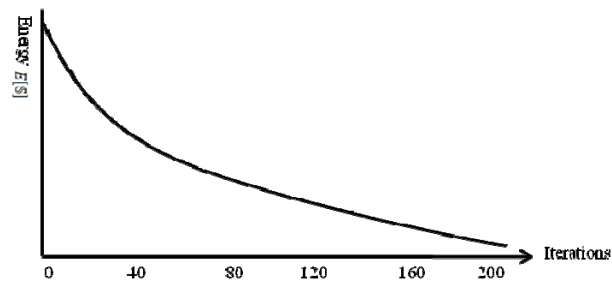


Figure 4.7 Energy  $E[x, y]$  in active contour algorithm versus the number of iteration.

As a result, before iterating this function, the current energy value is the largest one (as shown in Figure 4.7) because the pixels covered by the current spline own different scalar value. In other words, the data segment inside the initial spline (as shown in Figure 4.8 (a)) is mixture. By modifying the spline (as shown in Figure 4.8 (b-d)), the number of contained pixels and the composition of pixel value will be changed and the energy function can output new results. The active contour-based function also need to follow the proposed energy minimization strategy in traditional 2D boundary detection applications, consequently the minimum result of equation 4.11 will be kept during the iteration of modifying the spline and the energy will approach to zero as shown in Figure 4.7. Correspondingly, in the Figure 4.8 (e), the

final detected result is a homogenous part.

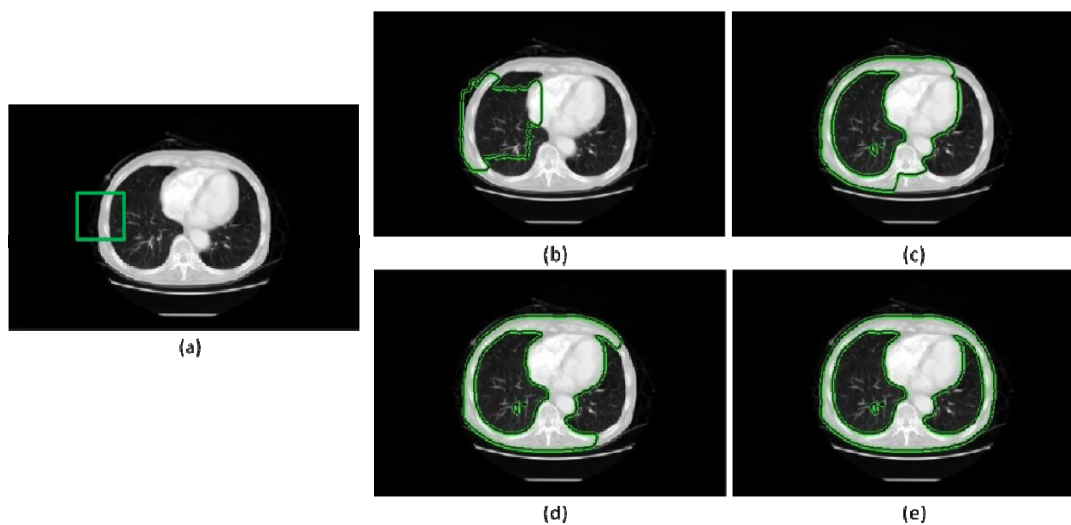


Figure 4.8 Illustrations of active contour algorithm

### 4.3.2 Region-based Active Surface Method

For processing the volume data, the boundary extraction design was implemented based on the region-based strategy of active contour method which had attracted increasing attention in analysing medical information (Du and Bui, 2008; Mille, 2009; Mohan, Sundaramoorthi et al., 2010). When using this active surface algorithm to process volume data, the linear spline will be changed into a customised cubic one (as shown in Figure 4.9).

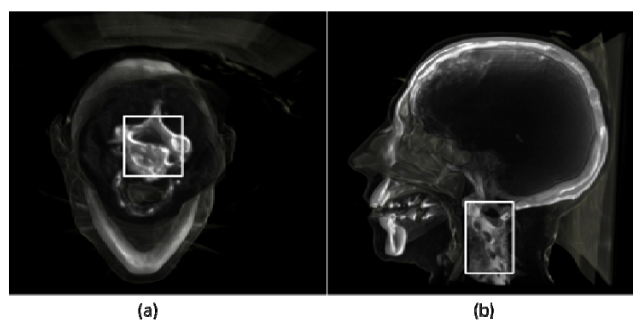


Figure 4.9 Illustrations of a domain of interest initialized in active surface algorithm. Image (a)



and (b) respectively represent the views of the cubic spline in axial and sagittal cross sections

As a result, its mathematical model was based on the change of a parameterized surface  $\mathbb{S}: S \in \mathbb{R}^6$  which outlined voxels in this boundary extraction function. By using the smoothness to represent the exterior constraint, the calculation of minimal energy inside this region-based method was represented by the equation:

$$E[\mathbb{S}] = \varphi E_{smooth} + (1 - \varphi) E_{region} \quad (4.12)$$

where  $\varphi$  is a pre-input parameter that weights the significance of the smoothness terms. Kass et al. proposed the mathematical description of a smoothness term by a first-order derivative of the object region. Based on this idea, the active surface function in this volume deformation system defines the smooth energy to describe the content underlying the spline in X, Y and Z-axial directions:

$$E_{smooth}[\mathbb{S}] = \iiint_{\mathbb{S}} \left\| \frac{\partial S}{\partial X} \right\|^2 + \left\| \frac{\partial S}{\partial Y} \right\|^2 + \left\| \frac{\partial S}{\partial Z} \right\|^2 dX dY dZ \quad (4.13)$$

Surface  $\mathbb{S}$  can separate the volume data into the interiors and exteriors.  $\mu_{in}$  and  $\mu_{out}$  respectively represent the sum of corresponding voxels' scalar values, and anticipate the evaluation of the changed region energy in the manner of dynamic parameter. By using the Chan-Vese model, the energy of the 3D domain can be calculated by:

$$E_{region}[\mathbb{S}] = \gamma_{in} \iiint_{\mathbb{S}} \mu_{in} dX dY dZ - \gamma_{out} \iiint_{\mathbb{S}} \mu_{out} dX dY dZ \quad (4.14)$$

where  $\gamma_{in}$  and  $\gamma_{out}$  are pre-defined constants for managing the proportions of interiors and exteriors respectively. The mechanism of this region-based active

surface transformation can be written in Table 4.9; its results are shown in Figure 4.10 in slices.

```

Initialize a surface  $S$  with constants  $\gamma_{in}$ ,  $\gamma_{out}$ ,  $\varphi$ 
If ( $E[S] - E[S]_{former} < 0$ )
    Calculate  $E_{smooth}[S]$  in equation 4.13
    Calculate  $E_{region}[S]$  in equation 4.14
    Update  $\mu_{in}$  and  $\mu_{out}$ 
    Calculate the energy values in equation 4.12
Else
    Break
Return  $S$ 

```

Table 4.9 Mechanism of active surface algorithm.

When using the active surface function to process the volume data sets, the resulting boundary information is a layer of detected voxels. In Figure 4.10, image A1-A5 are the 5 random cross sections of detecting the throat data segment via the initialized spline (image A0). Similarly, the detected skull data and the associated spline are respectively shown in B1- B5 and B0. They will be converted into vertices in Chapter 5.

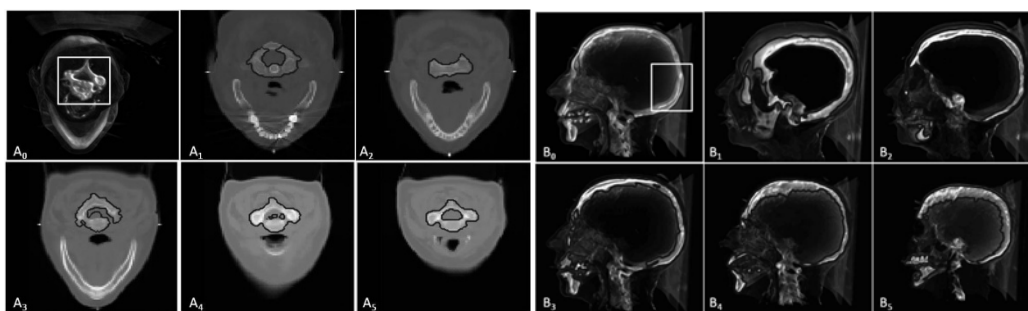


Figure 4.10 Results of active surface algorithm in volume segmentation

#### 4.4 Summary

- As illustrated in Figure 4.2, 4.3, 4.4 and 4.5, the feasibility of applying clustering algorithms to volume segmentation has been demonstrated by testing HVS, EMVS, KMVS, and MSVS clustering on different volume data sets. In Table 4.4, 4.5, 4.6, and 4.7, the evaluation of clustering method-based volume segmentation claimed the advantages of KMVS and the requirement of solutions to the segmentation errors.
- As illustrated in section 4.2, the integration of MSVS and KMVS was implemented to overcome the segmentation errors. And the clustered regions can be identified and highlighted by the ATF design. Based on this design, various operations and further analysis can be implemented.
- The region-based boundary extraction function has also been accomplished, and the extracted results demonstrated the feasibility of isolating interesting segment(s) from volume data.

The work on extracting volume control lattice introduced in the next chapter builds

upon the output from the volumetric data processing operations.

## Chapter 5 Lattice Construction and Refinement

In order to find a solution to improve the performance of the lattice construction process, the work document in this chapter focuses on developing a novel mechanism for defining control lattices and associated optimization designs prepared for the volume model manipulation stage. Most of the traditional methods, which involve manually defining the lattices and spending extra computation time on processing meaningless data, would be replaced by an automatic approach for improving the efficiency of this process.

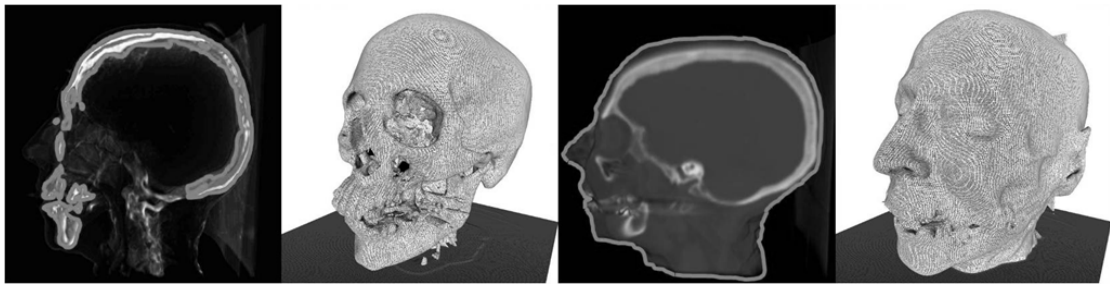


Figure 5.1 Results of constructed lattices based on the segmented information

Based on the isolated information provided by the segmentation operation described in Chapter 4, the constructed lattices can be much “closer” to the interesting segment(s) than those in manual definition methods. In other words, the usage of designed volumetric data processing function module can not only filter out meaningless parts (as shown in Figure 5.1(A and C)), but can also guide the accurate construction of lattices (as shown in images B and D). This novel lattice construction process relied on an iso-surface extraction technique – MC algorithm – to achieve the

construction of “model-fitting” lattices.

## **5.1 Marching Cubes Algorithm**

As a well-known technique, Marching Cubes algorithm serves an important indirect volume visualization approach which can represent the volume data via polygonal features, e.g. the resulting iso-surfaces can enable the volume model to support rendering methods in the field of surface modelling. However, using the iso-surface as the control lattice in volume deformation is an innovation design. The potential problem and limitation of using iso-surfaces as the control lattices for implementing a physic-based volume deformation had been mentioned in chapter 3, section 3.5. This algorithm is usually divided into two parts: extraction of vertices corresponding to a user-defined value, and calculation of the ‘normals’ at each of the vertices to accomplish triangulation tasks (Lorensen and Cline, 1987).

### **5.1.1 Sampling and Vertex Extraction Process**

By following a sort of divide-and-conquer-based sampling strategy, the MC-based lattice construction defined a cubic sampling window and makes it travel through the whole volume data. The statuses of intersections between the sampling window and sampled voxels determined the number of extracted vertices. Because each voxel’s scalar value can be converted into the values of its eight corners (as shown in Figure 5.2 (A)), the criterion for evaluating the results of various intersections was based on the comparison between each corner value and the user-defined one.

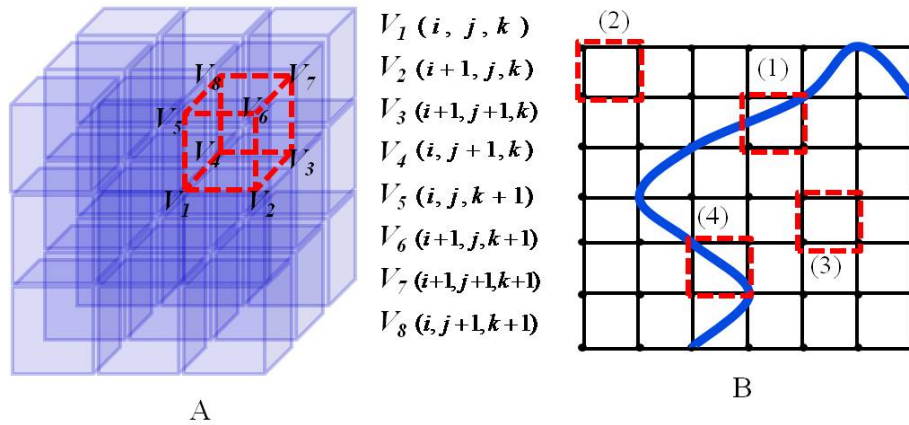


Figure 5.2 Diagram of sampling process in MC algorithm

Since the resulting iso-surface consists of extracted vertices, it can be imagined that this extracted surface intersects with associated voxels at sampling positions (1) and (4) (as shown in Figure 5.2 (B)). These voxels must comprise at least one corner which has the user-defined value. The other voxels, which are not sampled, lie on either position (2) or (3).

### 5.1.2 Triangulation Process

After ensuring that the corners fulfil the criterion, the following step is to triangulate this sampling result. Because there are eight corners in each voxel and two states, and the corner(s) is (or are) outside and inside surfaces, the triangulation process will comprise  $2^8 = 256$  types of intersections. There was a look-up table which was used as a way of indexing the associated triangulation of facets according to various surface-edge insertions (as shown in Figure 5.3 (A)) (Lorenzen and Cline, 1987). These 256 types of intersection have been represented via different combinations of 15 basic configurations (as shown in Figure 5.3 (B)).

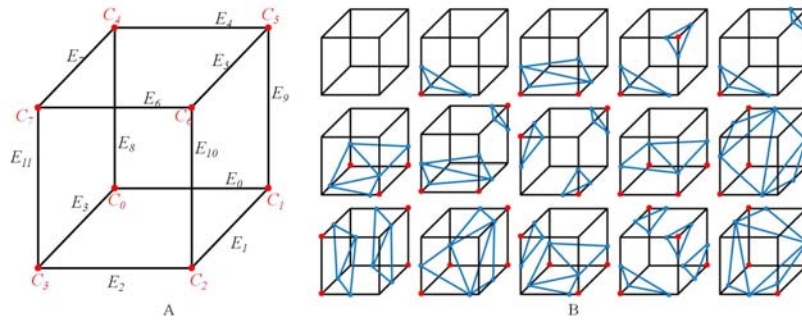


Figure 5.3 Diagram of the look-up table for surface-edge intersection

As the final process in the MC algorithm, the calculations of the unit ‘normal’ for each vertex guided the combinations of these triangular facets to form the resulting surfaces (as shown in Figure 5.4).

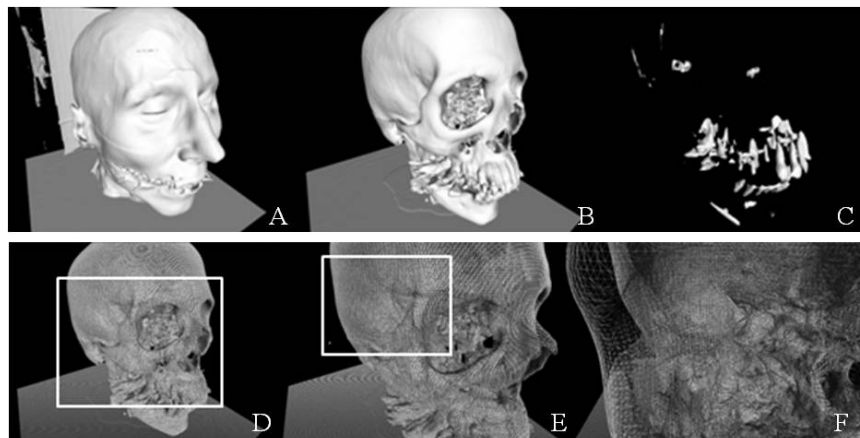


Figure 5.4 Results of extracted iso-surfaces. Images (A to C) are the results of extracting surfaces with different user-defined values. Images (D to F) gradually show the details of a wire-frame-based iso-surface.

### 5.1.3 Automatic Construction and Model-fitting Lattices

First of all, the clustering-based volume segmentation approach can freely locate each cluster and rapidly output the properties of each cluster’s centroid. This analysed information was directly used as the inputs in this MC-based surface extraction



process to replace the manually defined value, and the criterion for evaluating the statuses of intersections between the extracted surface and sampled voxels. As a result, this usage of clustered information can improve the efficiency of the lattice construction module.

Due to MC's sampling mechanism, the extracted vertices were all inside the sampled voxels. In other words, the volume of spaces between the extracted iso-surface and the underlying volumetric object was smaller than a voxel, which was usually measured at the micron level, i.e., these gaps were too small to calculate. As a result, this extracted surface can be the "model-fitting" lattice, which is the closest lattice than any manual defined one. In addition, as the outcome of Active Surface-based boundary extraction design inside the volumetric data processing module, the isolated features can be detected, to enable the efficient manipulations on the interesting segment(s). Meanwhile, the corresponding relationships between extracted vertices and sampled voxels were exported into a sort of indexing explained in Chapter 6, section 6.2.

## **5.2 Lattice Refinement**

For constructing control lattices, the key of using MC algorithm is extracting vertices from the exterior voxels in the object region. However, as the control lattice in the following deformation operations, the extracted iso-surface will increase the computation workload in deformation process because the MC algorithm's high

frequency sampling mechanism can make the control lattice contain a complicated framework and a large number of vertices. In order to maintain the system's real-time performance, the lattice refinement design follows the classical simplification strategies which originally modify the surface models for saving the storage, encoding/decoding for enhancing display effects or reconstructing for specific control. Their simplification methods will be tested and evaluated according to the criterion of efficiency in chapter 3, section 3.6. For example, as shown in Figure 5.4 (A), the total number of extracted vertices was 830K. Because, in order to keep a complete vertex-representation of the volume data, the sampling rates for the implementation of the MC algorithm was maintained at a high frequency so that at least one vertex can be extracted from the associated voxels. This complex representation can prepare abundant connections between the control lattice and the underlying volumetric object for the volume deformation operations.

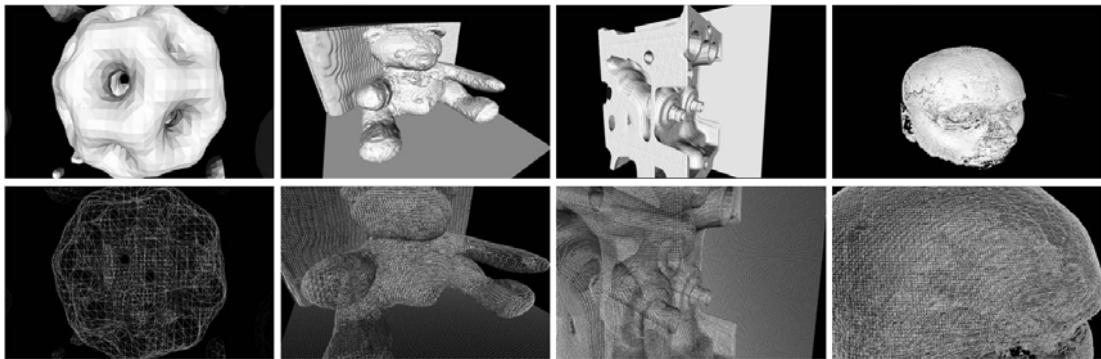


Figure 5.5 Results of extracted iso-surfaces. From left to right, the associated number of extracted vertices are 1.9k, 58K, 41k, and 950K respectively

However, because these vertices serve as the control points during the deformation process, the huge amount of control points would increase the computation workload

and reduce the interactive rate of real-time manipulations. Therefore, polygonal simplification methods were investigated to simplify the automatically constructed lattices (as shown in Figure 5.5).

As an intermediate result, only the framework of extracted iso-surfaces is considered in the following deformation process. In other words, the solution of refinement can ignore the surface features. Therefore, the additional evaluation criteria of simplification approaches will be modified when processing the extracted iso-surface in this project. As an application of mesh simplification, the refinement design will be determined based on the result of experimenting with classic approaches. According to the acknowledged review of mesh simplification methods, the extracted lattice is respectively processed by testing four prevalent methods: varying sampling rates, adaptive subdivision scheme, vertex decimation and merging approach (Luebke, 2001).

### **5.2.1 Varying Sampling Rates**

As the most direct and simplest simplification solution, varying the sampling rate can decrease the size of sampled data proportionally. However, there were several disadvantages in terms of carrying out the simplified results correctly, and manipulating the simplification freely. As shown in Figure 5.6, images B and D respectively show the simplified results of the extracted iso-surfaces in images A and C. By decreasing the sampling rate, the simplified results contained a number of losses in the features (the unacceptable gaps in images B and D) which certainly cause

the incorrect representations in deformation stage and reduce the accuracy of associated simulations. Besides, this proportional vertex management suffered from the familiar problems in traditional simplification approaches: limited simplified lattices because a few sampling frequencies available for the extraction process (Luebke, 2001).

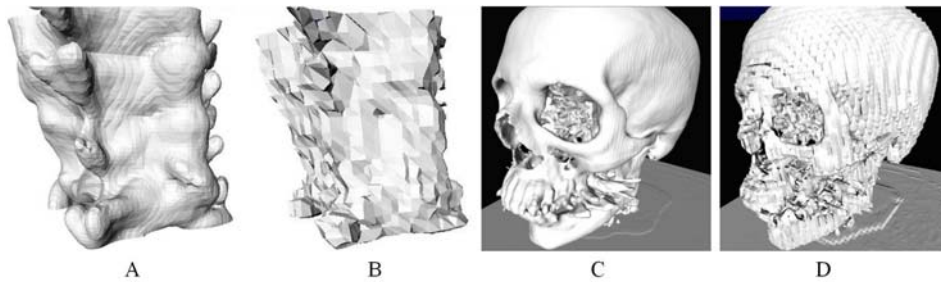


Figure 5.6 Results of decreasing sampling rate

### 5.2.2 Adaptive Subdivision Scheme

The Loop Subdivision scheme originally focused on converting arbitrary polygonal surfaces into smooth triangle-based ones by calculating new vertices. Its scheme includes adding edge-vertices and refining the changed mesh, and mainly relies on the second process, following triangular spline, in triangle-based simplifications (Faramarz, Colin et al., 2007). The lattice refinement process tested its inverse scheme for decreasing the number of rhombus within the surface from simplification level 1 to level 3 (as shown in Figure 5.7)

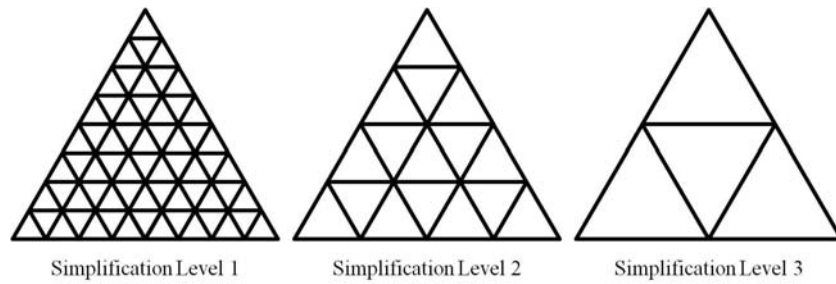


Figure 5.7 Illustrations of adaptive subdivision method

This simplification scheme was mainly used for compaction of complicated geometries or reverse engineering applications. In this project, a reverse “refinement” of constructed lattices was implemented by orienting the “former” vertex according to the current vertex and its neighbour, i.e. reversing the loop subdivision process. Then, the oriented former vertices were filtered by a subtraction process, which is an affine operation of adding points in a regular loop subdivision scheme. In the regular refinement process in loop subdivision and inverse schemes, the surface needs to be restored, so that the valence of each vertex can be changed into 6. However, in order to maintain the relationships between vertices and voxels, the surface restoring operations were disabled in the refinement process.

### 5.2.3 Vertex Decimation Approach

The vertex decimation approach iteratively located several “omissible” points, removed them and re-triangulated the result until the simplified mesh meets a user-defined criterion (as shown in Figure 5.8). As a classical simplification solution tested in this project, this method consisted of two steps: decimating vertices and preventing the local topology of the mesh from being affected by the changes caused

by the vertex movements (Schroeder, Zarge et al., 1992).

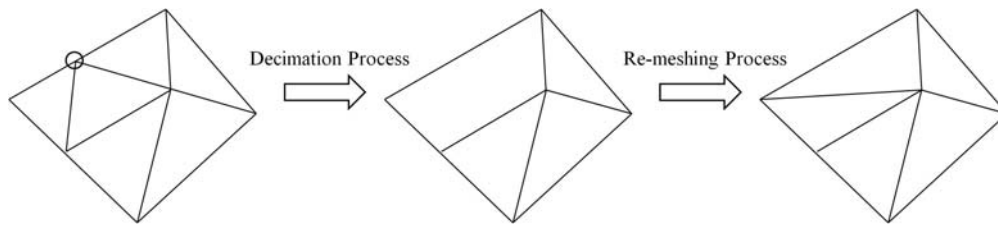


Figure 5.8 Illustrations of vertex decimation method

Before simplifying a mesh, the decimation algorithm firstly represented the local geometry and topology via a given mark. This mark may be: a simple vertex surrounded by a fixed number of neighbouring vertices; a complex vertex connected with a constant facet and a set of neighbours; a boundary vertex located on a determined fixed side within a rigid framework; an interior edge line shared by given triangles, or a corner whose structure must be protected through the simplification stage. In this vertex decimation approach, this mark served as an indicator of the simplification process, which will switch this process off if the associated local geometry and topology are changed.

#### 5.2.4 Vertex Merging Method

The vertex merging method operated the simplification process by iterating the cycle of merging two or more vertices into a single vertex (Oh & Park, 1995). As a control point inside the lattice adopted in the deformation process, every extracted vertex was addressed by a voxel for accomplishing a sort of mapping operation in Chapter 6. If several vertices are decimated, the deformation will run as usual with this incomplete

lattice. However, the merging-based simplification method led to quite a few new generated vertices which destroy the indexable relationship between extracted vertices and associated voxels. Therefore, the vertex merging method was not used as the solution for simplifying constructed lattices.

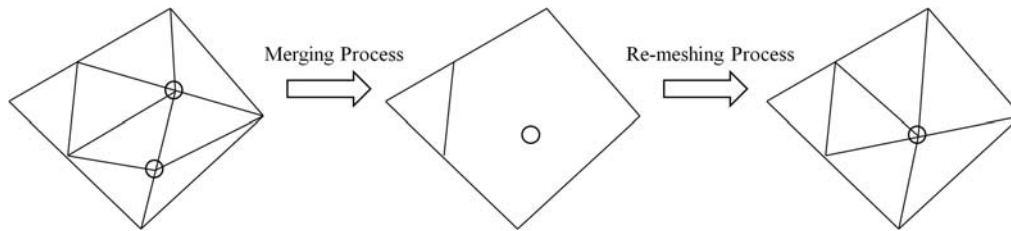


Figure 5.9 Illustrations of vertex merging method

### 5.2.5 Tests and Evaluations

In Table 5.1, the comparisons between the results of simplification using the adaptive subdivision scheme and vertex decimation method illustrated that the decimation method is more adept at capturing the exact geometry of the surface model, especially in the preservation of sharp features, than the adaptive scheme (as shown in the highlighted rectangles). As a result, simplified lattices generated by the vertex decimation method can provide more available and recognizable features. In addition, these simplified results also demonstrate the capability of these two methods for simplifying the same mesh model.


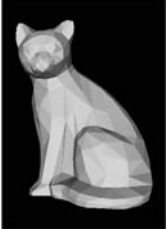


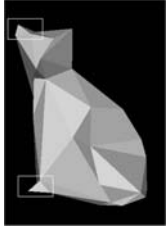
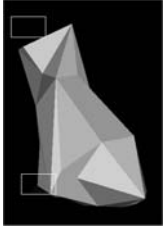
Method Num of Vert	Vertex Decimation Approach	Adaptive Subdivision Scheme
200-150		
150-90		
80-60		

Table 5.1 Comparisons between results of decimation and subdivision solutions

Table 5.2 compared the flexibilities of these two simplification solutions by testing them to process the same complicated mesh model and listing the numbers of vertices according to different simplification levels. Due to the constraint that the given geometry and topology cannot be changed, the limit for the number of vertices in decimation-based simplification solution was about 300, and it cannot provide further simplification as the subdivision-based method can. This disadvantage can restrict the flexibility of the lattices refinement process.



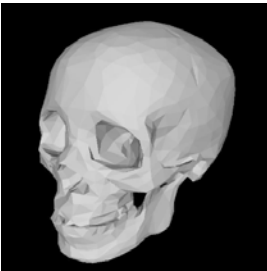
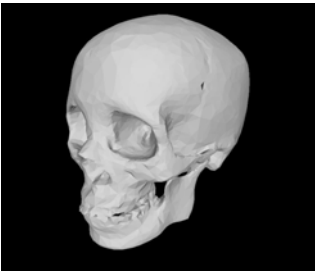
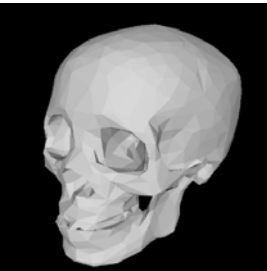
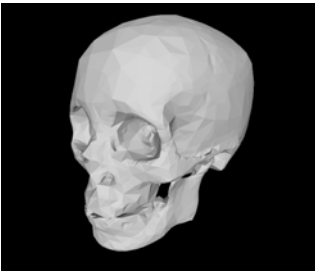
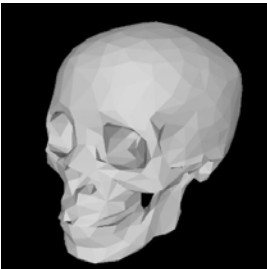
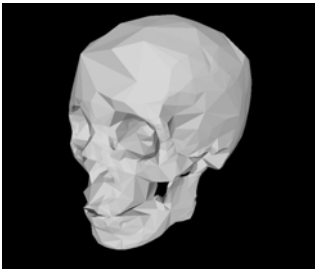
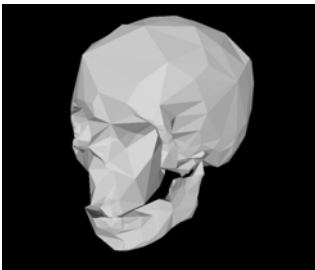
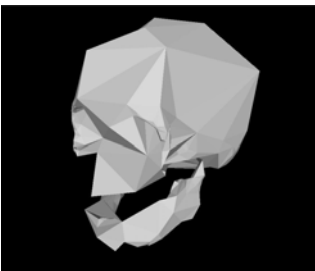
Solution Num of Vert	Vertex Decimation	Adaptive Subdivision
700-600		
600-500		
500-300		
300-150	Null	
150-100	Null	

Table 5.2 Comparison between results of decimation and subdivision solutions

As a result of this comparison, the subdivision-based method was chosen to simplify

the extracted iso-surfaces. The simplified lattices are shown in Figure 5.10 (A to G) with their wire-frame-based structures (a to g). According to different simplification levels, Figure 5.11 shows the corresponding representations of deformed lattices which will be discussed in the following chapter.

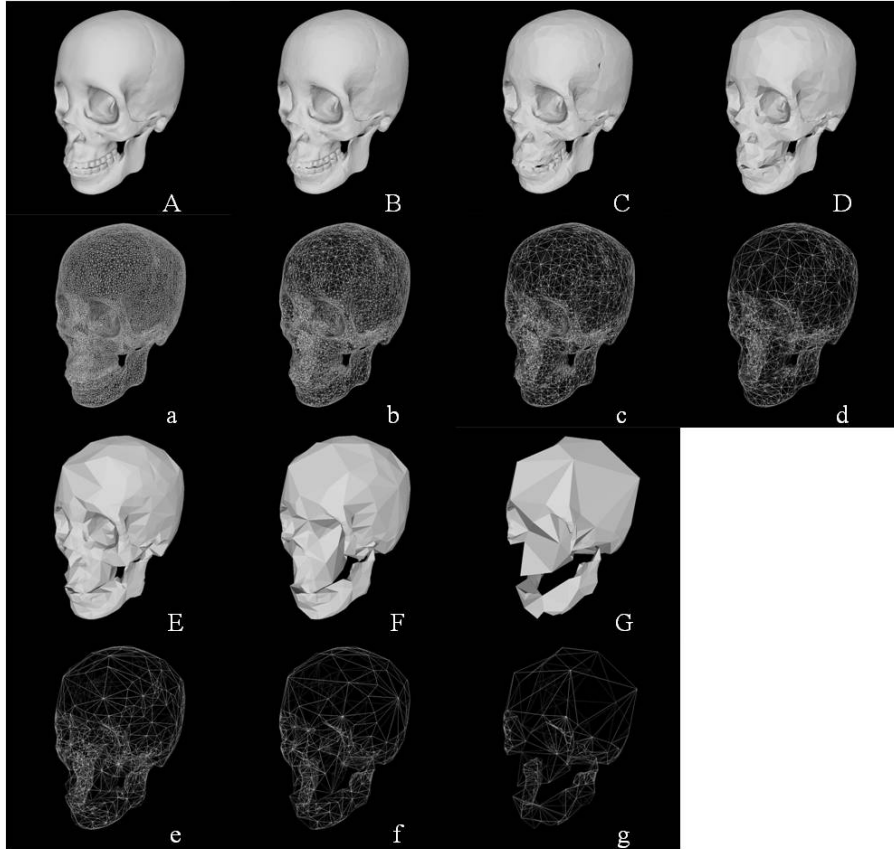


Figure 5.10 Illustrations of vertex adaptive subdivision solution

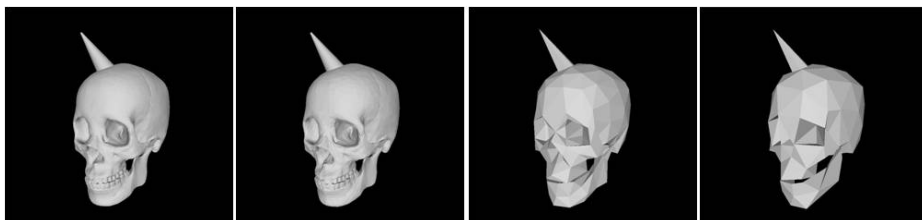


Figure 5.11 Illustrations of deformed control lattices

### 5.3 Summary

- The idea of using MC algorithm to construct the so-called “model-fitting” control lattice is based on its sampling capability which makes the extracted iso-surface perform the closest cover for the underlying object. This idea can solve the low accuracy problem inside the manually defining lattice operations.
- Derived from the active surface function design in Chapter 4, the detected information can be directly converted into the dedicated control lattice to envelop the interesting data segment(s). This data segmentation design can efficiently simplify the volume data size, and consequently reduce the workload of associated computation and data access operations. Besides, the lattice refinement function cannot only control the complexity of control lattices, but maintain the relationship which determines the correspondences between the control vertices and exterior voxels in the deformation process.” Both of these simplifications can save the processing time spent on processing the useless data and inefficient (or offline) operations, e.g. constrained deformation (Correa, Silver et al., 2010).
- There are existing hardware-driven tessellation examples, which can be adopted for square-based simplifications, e.g. the GPU-based Catmull-Clark subdivision. In this research, the basic element inside the extracted lattices was the triangle, so that the adaptive subdivision scheme was implemented in

the form of hardware-driven processes for enabling the management of triangles via an “on-the-fly” style (explained in Chapter 7, section 7.3.3).

- By following this process, the extracted surface was successfully unified and simplified, and enabled logical and mathematical relationships between the initial and final vertices for mapping operations in following deformation process.

## Chapter 6 Volume Deformation

In the terms of point-set topology, a solid object indicates a perfect closure of interiors by its surface. Technically, deforming a solid object - a volume model - should transform its internal structures along with the surface changes. In some simulations, the focus is only on the “shape” changes of the simulated objects to provide the desired visual effects.

In true volume deformation applications, the internal structure of a volume model needs to be defined explicitly through characteristics, e.g. the varied distribution gradients of voxels, to simulate the deformation behaviours often induced by physics-based forces. Therefore, the deformation process deployed in this research aimed to generate a new distribution map for voxels “under strains”, so that the final visualization could show realistic material behaviours.

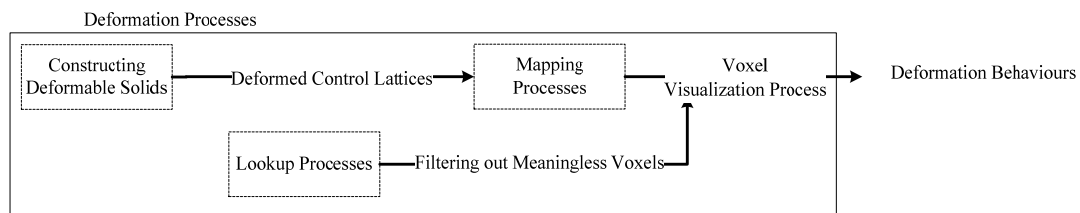


Figure 6.1 Pipeline of deformation module

The pipeline of the deformation process is shown in Figure 6.1. The following three sections will detail the principles for constructing the deformable solids, the address assignments for the participant voxels and the operation of “displacement mapping”.

The participant was used to distinguish the voxels from the others lying outside the deformation radiation. An indexing function was designed to determine the participant voxels in a deformation cycle, based on the well-known traversal mechanism in an octree data structure.

### **6.1 Constructing Deformable Solids**

Based on the resulting iso-surfaces extracted in the lattice construction process, the interesting segment in volume data can be completely enclosed. Therefore, Deformation of Geometric Models Editor (DOGME), which uses an enclosed lattice to represent the deformation and then passes the deformation characters to the underlying models, can be directly used as the manipulation strategy in this deformation design. However, in most DOGME-based deformation applications, the manual construction of control lattices can exacerbate the problem of time-consuming process, and consequently restrict the performance of real-time operations. Besides inefficient operations, the constrained assumption that the lattices are perfectly close to the underlying objects, will lead to the artefacts in the resulting deformation behaviours.

As explained in the Chapter 5, the sampling mechanism and the extraction mode in MC-based lattice construction process successfully made the extracted iso-surface serve as a “model-fitting” lattice which instinctively matches the surface features of the volumetric object. As a result, an improved DOGME (I-DOGME) was devised to

be integrated with the designed lattice construction for enveloping the only interesting data segment in the continuous volume data, and consequently prevented the extra computation time from being cost on the useless parts. In this I-DOGME method, the deformation operations were firstly implemented on the control lattice through a section concentrating on the lattice deformation. This deformation was constructed by embedding a mass-spring framework to represent the resistance force.

### 6.1.1 Lattice Deformation

By freely moving the control vertices to given positions, or optionally interpolating new vertices for restoring the surface structures, as shown in Figure 6.2 (A), the lattice can be manipulated to present the desired results without any meaning. Figure 6.2 (B) shows the 3D results of the corresponding 2D lattice deformations.

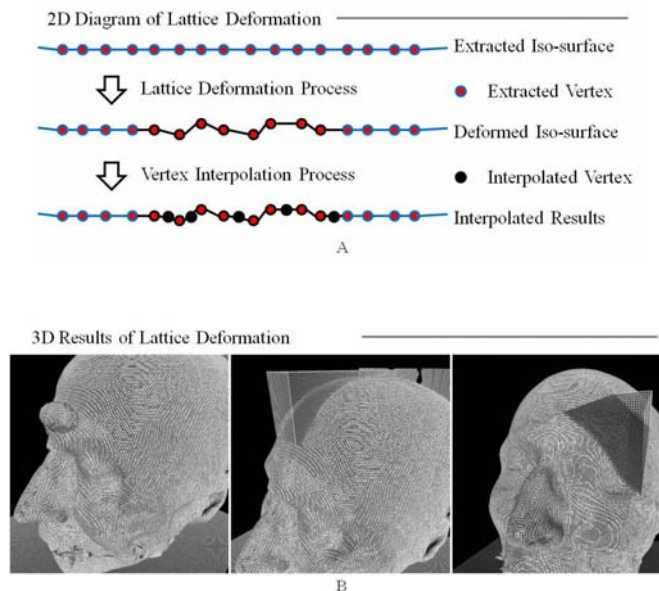


Figure 6.2 Random shape changes on the extracted lattice (in 2D and 3D)

In order to meet the requirements of complicated simulations (physics-based

deformation), the lattice was converted into deformable solids using specific mathematical models. In this volume deformation method, the constructed lattices were all triangle-based meshes and the extracted polygons were all coplanar. Therefore, the mathematical model which requires special lattices were not suitable for this applied lattices, such as the FEM model demands tetrahedral frameworks, and the Chain-Mail approach requires intersected components. In addition, applying those models will lead to the separation of volume models into a number of partitions, which often consumes substantial computational time to establish regional relationships between groups of voxels, and essential matrix calculations in the displacement mapping stage. The deformation model in this programme adopted the mass-spring model for manipulating the deformable lattices.

### **6.1.2 Embedding Mass-spring Mechanism-based Framework**

Embedding the mass-spring mechanism was a process that transformed the control lattice from a “place-holder” to a non-rigid deformable one. In order to represent elastic behaviours in the manner of mesh dynamics, the mass-spring model established a set of link-based relationships between masses in the manner of lines between vertices. Similar to the links between neighbouring vertices, a voxel-based system was carried out on the layer of exterior voxels after the displacement mapping process (as shown in Figure 6.3) (Provot, 1995):

- Links between voxels  $(a, b)$  and  $(a+1, b)$ , and voxels  $(a, b)$  and  $(a, b+1)$  were referred to as “structural springs” and were coloured in red. This can simulate



the pulling force applied on the voxels.

- Links between voxels  $(a, b)$  and  $(a+1, b+1)$  were referred to as “shear springs” and were coloured in blue; this is to simulate the transition on the cross section of the lattice. It can simulate the shear stress running in parallel to the cross section, and the transformation of forces when tensional or compression stress is applied perpendicularly.
- Links between voxels  $(a, b)$  and  $(a+2, b)$ , and voxels  $(a, b)$  and  $(a, b+2)$ , were referred to as “flexion springs” and are coloured in green. This can simulate the flexion stress (i.e. bending force) inside a layer of voxels.

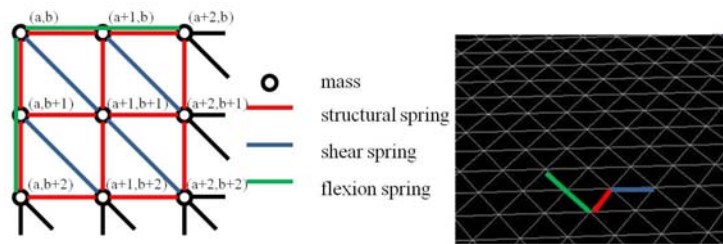


Figure 6.3 Mesh-based mass-spring system

In order to simulate the effects of different forces, the mass-spring system relied on various combinations of these links. For example, for performing elastic effects and damper frameworks caused by stretching forces, the mass-spring relationship will be limited to a single type of the 3 spring links defined above.

For simplifying the relevant calculations for the deformation process, the employed mass-spring mechanism only relied on the structural spring link, each vertex in this system was treated as one unit in mass, and the force was always applied on one control point (or vertex). In order to distinguish it from the other vertices, this control

point was named as the target vertex. For example, the target vertex  $P_{x,y,z}$  could obtain acceleration  $\alpha_{x,y,z}$  through the applied force  $F_{x,y,z}$  on it. After defining the start time  $T$  and duration  $t$ , the displacement was computed as the distance between the vertex's starting position  $P_{x,y,z}(T)$  and the end position  $P_{x,y,z}(T + t)$ , which was equal to the scalar of vector  $\left\| \overrightarrow{P_{x,y,z}(T)P_{x,y,z}(T + t)} \right\|$ . Taking into consideration that the applied forces are not constant, the resulting distance  $\left\| \overrightarrow{P_{x,y,z}(T)P_{x,y,z}(T + t)} \right\|$  cannot be obtained by calculating a simple quadratic equation with one known. As a result, the varying acceleration  $\alpha_{x,y,z}$  and a series of varying velocities  $V_{x,y,z}$  was approximately worked out during the duration  $t$  in equation 6.1, where the displacement of the vertex was calculated (Nealen, Müller et al., 2006).

$$\alpha_{x,y,z}(T + t) = \frac{F_{x,y,z}(T + t)}{1}$$

$$V_{x,y,z}(T + t) = V_{x,y,z}(T) + t\alpha_{x,y,z}(T + t) \quad (6.1)$$

$$P_{x,y,z}(T + t) = P_{x,y,z}(t) + tV_{x,y,z}(T + t)$$

After defining the movement of the target vertex, the next step was to build an iteration to decompose the applied force on the other ends of the structural springs. As shown in Figure 6.4 (A), all the decomposed forces were constrained to obtain equal scalars at the same level. The iteration continuously carried on the decomposed forces applied in the order of the Red→Orange→Green→Black, as shown in Figure 6.4 (B), where the hollow ones represent the vertices lying outside the deformation radiation.

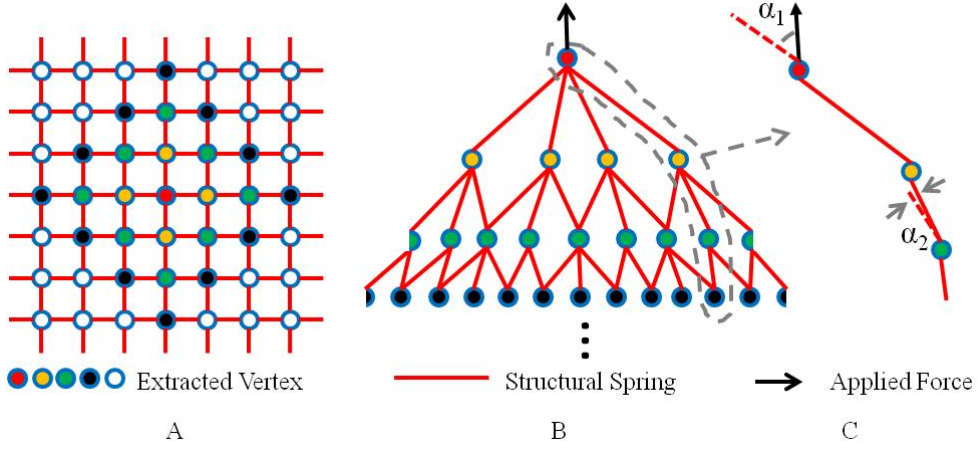


Figure 6.4 Diagrams of decomposing the applied force in mass-spring system

For example, Figure 6.4 (C) shows a component force  $F_{u,v,w}$  on the associated vertex  $P_{u,v,w}$  (the orange point) that is connected with the target vertex  $P_{x,y,z}$  (the red one). After locating the position of the moved target vertex  $P_{x,y,z}(T+t)$ , the component force  $F_{u,v,w}$  was calculated as in equation 6.2. In the Euclidean plane, the angle  $\alpha_1$  (dot product) between vectors  $\overrightarrow{P_{x,y,z}(T)P_{x,y,z}(T+t)}$  and  $\overrightarrow{P_{u,v,w}(T)P_{x,y,z}(T+t)}$  can be calculated by the equation 6.3.

$$F_{u,v,w} = F_{x,y,z} \cos \alpha_1 \quad (6.2)$$

$$\alpha_1 = \cos^{-1} \left( \frac{\overrightarrow{P_{x,y,z}(T)P_{x,y,z}(T+t)} \cdot \overrightarrow{P_{u,v,w}(T)P_{x,y,z}(T+t)}}{\| \overrightarrow{P_{x,y,z}(T)P_{x,y,z}(T+t)} \| \| \overrightarrow{P_{u,v,w}(T)P_{x,y,z}(T+t)} \|} \right) \quad (6.3)$$

And the component force  $F_{u,v,w}$  on vertex  $P_{u,v,w}$  can be written as:

$$F_{u,v,w} = F_{x,y,z} \cos \left( \frac{\overrightarrow{P_{x,y,z}(T)P_{x,y,z}(T+t)} \cdot \overrightarrow{P_{u,v,w}(T)P_{x,y,z}(T+t)}}{\| \overrightarrow{P_{x,y,z}(T)P_{x,y,z}(T+t)} \| \| \overrightarrow{P_{u,v,w}(T)P_{x,y,z}(T+t)} \|} \right) \quad (6.4)$$

Via iterating this formula calculation, all component forces on  $P_{u,v,w}$ 's neighbours can be worked out. When the angle between two vectors approximates to zero, the iteration will stop processing the branches of this vertex (the green one in Figure 6.4 (A)). In other words, its branches (coloured in black) can still receive the component forces, but not anticipates in the further computations. Based on the calculated components of the applied forces, the associated displacements of the surrounding vertices can be calculated in equation 6.1.

### 6.1.3 Implementing Resistance Force Mechanism

The lattice deformation introduced in the above sections often suffered from a lack of realism in runtime (Provot, 1995). For example, Figure 6.5 (B) exhibits a “super-elastic” effect that occurred in dragging a flap of skin on a normal human head (as show in image A).

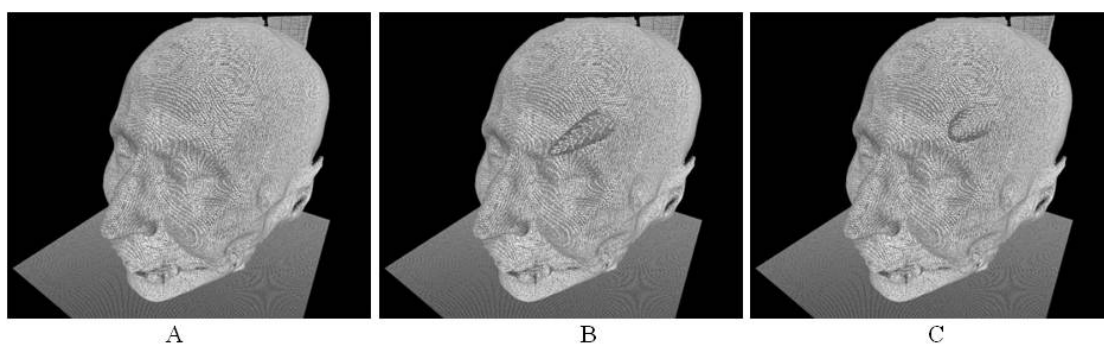


Figure 6.5 Results of lattice deformation with mass-spring system

In order to simulate the realistic and physics-based behaviours of human tissues, the deformation employed a stiffness term to “buffer” the current linear effects, as shown in Figure 6.5 (C). In reality, the stiffness term -  $F_{stiffness}$  - will increase rapidly if

the deformation extent decreases, so that the results can support limited shape changes. In the simulation of the radial force distribution, the equation of  $F_{stiffness}$  can be written as (Provot, 1995):

$$F_{stiffness}(P_{u,v,w}) = -\sum K_{x,y,z,u,v,w} \left[ \overrightarrow{P_{x,y,z}P_{u,v,w}} - L_{x,y,z,u,v,w}^0 \frac{\overrightarrow{P_{x,y,z}P_{u,v,w}}}{\|\overrightarrow{P_{x,y,z}P_{u,v,w}}\|} \right] \quad (6.5)$$

where  $K_{x,y,z,u,v,w}$  is a predefined stiffness coefficient of the structural spring between  $P_{x,y,z}$  and  $P_{u,v,w}$ , and will vary according the length of this link.  $L_{x,y,z,u,v,w}^0$  represents the original status of the spring link. Because all the stiffness forces  $F_{stiffness}$  surrounding  $P_{x,y,z}$  and the component forces on its neighbouring vertices are opposite and collinear in pairs. Therefore, the displacements of the target vertex and its neighbours  $\overrightarrow{P_{x,y,z}P_{u,v,w}}$  can be calculated after integrating the  $F_{stiffness}$  and the calculated component force  $F_{u,v,w}$  in the above section. By defining different stiffness coefficients  $K$  on the structured spring links, the resulting stiffness forces will make the deformed lattice present varied shape changes as shown in Figure 6.6. When the applied force, duration and the target vertex are all fixed, the change of stiffness parameters can determine the extent of deformation with the increase of  $K$  value, which leads to more deformation effects.

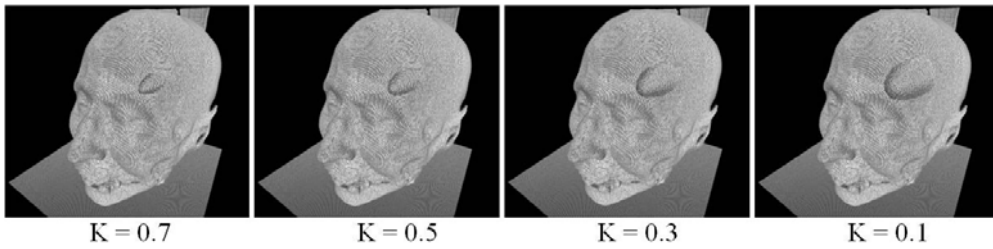


Figure 6.6 Results of lattice deformation with different stiffness coefficients

Based on this mechanism, the calculations of the radial force distribution in the lattice deformation process were determined by whether both the composition of the components of applied force and the current stiffness force all equal to zero or not. In this lattice deformation process, each vertex which lies in the distribution region were assigned with a component force, and the resulting displacement can be calculated in the following process, explained in Table 6.1.

*Define the position of target vertex  $P_{x,y,z}$ ; the position of random vertex  $P_{u,v,w}$*

*Define the component force  $F_{u,v,w}$  and a value  $F_0$*

*If  $\|\overrightarrow{P_{x,y,z}(T)P_{u,v,w}(T)}\|$  equals zero //pointing at the target vertex*

*Calculate  $\overrightarrow{P_{u,v,w}(T)P_{u,v,w}(T+t)}$  for target vertex in equation 6.1*

*Else //pointing at the neighbouring vertex*

*Calculate the component force  $F_{u,v,w}$  in equation 6.4*

*Calculate the stiffness force  $F_{stiffness}(P_{u,v,w})$  in equation 6.5*

*Calculate the integrated force  $F_{u,v,w} = F_{u,v,w} - F_{stiffness}(P_{u,v,w})$*

*Calculate  $\overrightarrow{P_{u,v,w}(T)P_{u,v,w}(T+t)}$  with calculated  $F_{u,v,w}$  in equation 6.1*

*Return  $\overrightarrow{P_{u,v,w}(T)P_{u,v,w}(T+t)}$*

*Repeat*

*Until  $F_{u,v,w} \leq F_0$*

Table 6.1 Mechanism of vertex displacement calculation

## 6.2 Displacement Mapping

### 6.2.1 Mapping Process Design

In this mapping process, the mapping matrix  $M\_Trans_{m \times n}$  served as a translator for the data mapping operation between  $n$  and  $m$  dimensional spaces, which respectively represent the lattice and the underlying volume model. The principle of the mapping process in traditional DOGME approaches is shown in Figure 6.7.

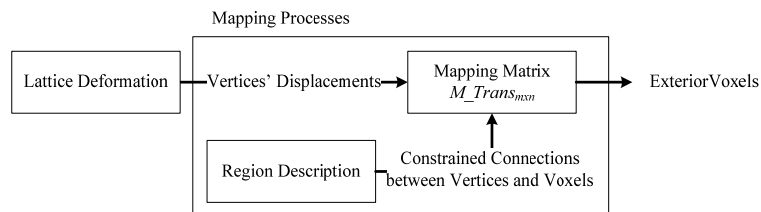


Figure 6.7 Diagram of mapping process deployed in the DOGME

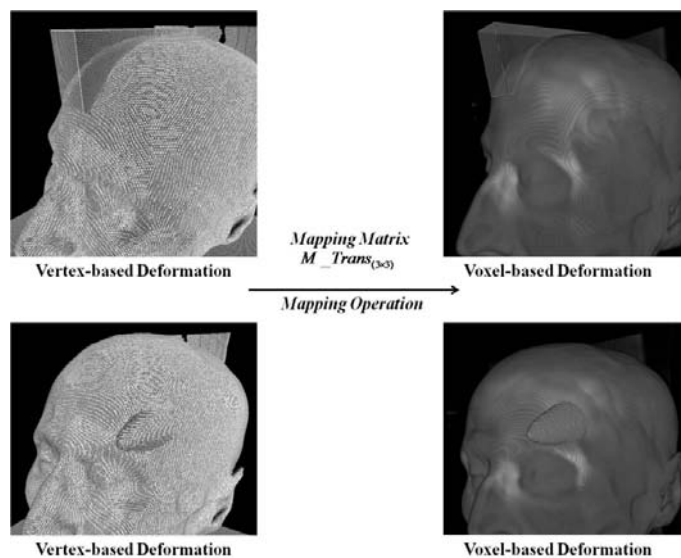


Figure 6.8 Diagram of I-DOGME

Since the deformation method described in this thesis only comprised two types of 3D elements, vertices and voxels, the mapping matrix was denoted as  $M\_Trans_{3 \times 3}$ . Figure 6.8 illustrates the vertices' displacements, named displacement constraints, in vertex space and the related results of volume deformation.

In traditional DOGME methods, after finishing the lattice deformation, the displacement constraints, registered in the form of control lattices, will be “super-imposed” onto the given area by following a set of manually defined connections between control vertices and voxels (Bechmann and Gerber, 2003). In I-DOGME, this constrained relationship was replaced by an inherent relationship which was automatically derived from the MC-based lattice construction process; meanwhile its “off-line” region description mode was improved.

### **6.2.2 Designs of Indexable Inherent Relationship (IIR)**

All vertices on the control lattice were all extracted by the MC process. In order to link these vertices with corresponding voxels, the IIR design utilized this inherent relationship to order the displacement mapping operations between voxels and vertices. Because each extracted vertex can be located on a voxel's edge (explained in Chapter 4), this voxel's coordinates was directly regarded as the vertex's. Table 6.2 shows the mechanism of indexing the voxel based on the vertices' parameters. In order to simplify the explanation of this design, the volume of the voxel and the space of the shifting sample grid were all assumed to be a uniform size.



*Define the space of shifting sampling grid, the volume of a voxel, and the volume of a sampling grid ( $S_{width0} \times S_{height0} \times S_{depth0}$ ).*

*Initialize the position of random vertex  $P_{u,v,w}(u, v, w)$*

*Initialize integer ( $T_{s_x}, T_{s_y}, T_{s_z}$ )*

*Define the size of volume model ( $W, H, D$ ).*

*Define the sequence number of each voxel  $ID_{voxel}$*

*Define the voxel array  $vol\_data[x,y,z]$*

*Calculating the times  $T_{s_x}, T_{s_y}$  and  $T_{s_z}$  of shifting grade in X-, Y- and Z-axial respectively in*

$$(T_{s_x}, T_{s_y}, T_{s_z}) = \text{int} \left( \frac{W, H, D}{S_{width0}, S_{height0}, S_{depth0}} \right) + (1, 1, 1)$$

*Using ( $T_{s_x}, T_{s_y}, T_{s_z}$ ) in fetching the voxel's sequence number*

$$ID_{voxel} = vol\_data[T_{s_x}, T_{s_y}, T_{s_z}]$$

*Return  $ID_{voxel}$*

Table 6.2 Mechanism for locating voxel's sequence number

This usage of inherent relationship between vertices and voxel played an important role in this displacement mapping process, which solved the problem of complexity caused by the manual region description in off-line mode. In addition, this usage made a further demonstration that maintaining and encapsulating the inherent relationship should be a criterion of analysing the pros and cons of four surface simplification methods in section 5.2. With the IIR design, the principle of the improved mapping process in I-DOGME method is shown in Figure 6.9

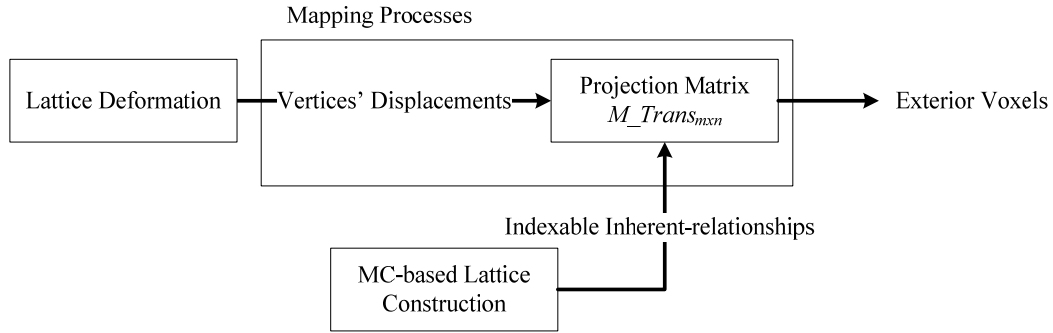


Figure 6.9 Diagram of improved mapping process deployed in the I-DOGME

After implementing the IIR design to guide the displacement mapping between control vertices and corresponding exterior voxels, the resulting deformation behaviour can be shown as the leftmost image in Figure 6.10. Modifying the stiffness coefficient can lead to different deformation results as shown in the rest images.

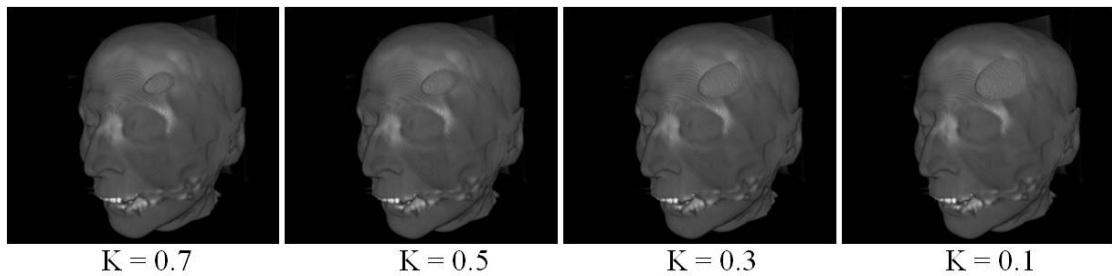


Figure 6.10 Results of volume deformation with different stiffness values

However, after applying the clipping planes to these deformation results (as shown in Figure 6.11), it was clearly visible that only a single layer of voxels in the volume model were affected.

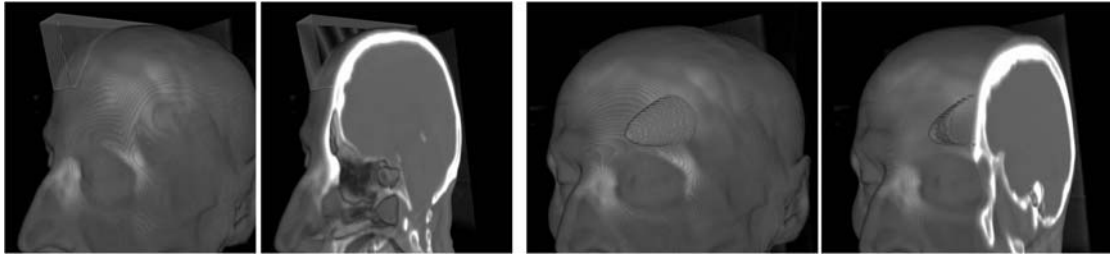


Figure 6.11 Results of clipped results after the mapping process

This is because the mapping process can only utilize the relationship between extracted vertices and exterior voxels, but cannot suffice the interiors due to the lack of connections between the exterior and interior voxels. As the extension of Figure 6.2 (A), Figure 6.12 establishes a relationship among the extracted vertices (highlighted points), the exterior voxels (red cubes) and interior voxels (black cubes). In order to accomplish this relationship, the most direct solution is connecting all interior voxels with the displacement constraints.

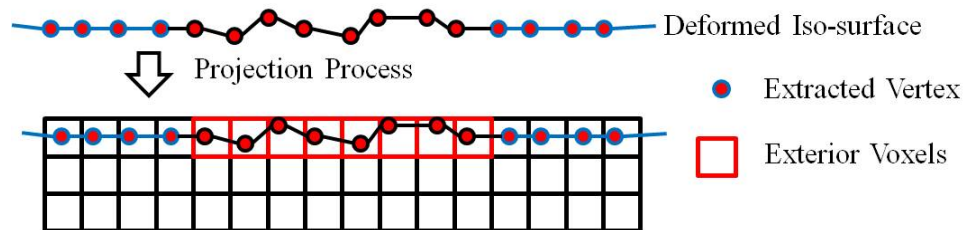


Figure 6.12 Diagram of relationships between the control vertices and underlying voxels

However, this idea will certainly cause excessive amounts of computational time in every deformation loop. For solving this problem, this design created new associated indexing operations relied on a sort of relationship-building mechanisms, to link exterior voxels (or displacement constraints) with interior voxels in the volumetric mapping stage.

### 6.3 Octree-based Lookup Function

In order to assign the displacement to more voxels (especially the interior ones), an internal relationship between exterior and interior voxels was established. Similar to the distribution of seismic wave phenomena which occurs in earthquakes, the volume deformation process was expected to perform the gradient distribution of different deformation levels in the final result. Its principle is shown in Figure 6.13.

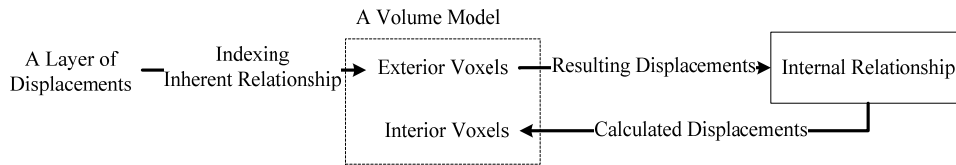


Figure 6.13 Diagram of creating the octree-based lookup function

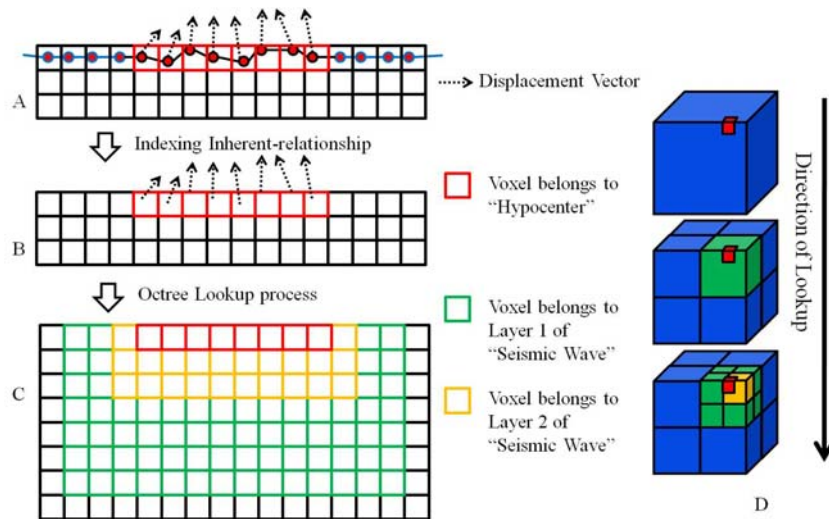


Figure 6.14 Diagram of using the octree-based lookup function

For example, as shown in Figure 6.14 (A), a layer of voxels (exteriors) obtained the displacement offsets based on the IIR design. In order to determine the internal relationship between exterior and interior voxels, an octree data structure was applied

to achieve a hierarchical management of all voxels in the form of branches and leaves. Following the lookup direction (as shown in image D), the internal voxels (green and orange ones) can be gradually located (as shown in images B and C).

In order to implement the octree data structure, the volumetric space needed to divide itself into  $8^{num\_subd}$  partitions according to the given amount of subdivision operations ( $num\_subd$ ) (as shown in Figure 6.15). To simplify the explanation of the process, the volume space and all its partitions were all defined as cubic. The portioned cubes at the same depth in the structure acquired the same size.

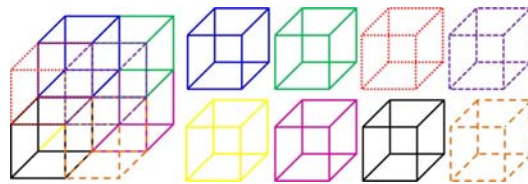


Figure 6.15 Diagram of the arrangement of 8 partitions

As a result, the subdividing process in the Figure 6.14 (D) can be represented via a kind of nested relationship between leaf nodes (shown in Figure 6.16)

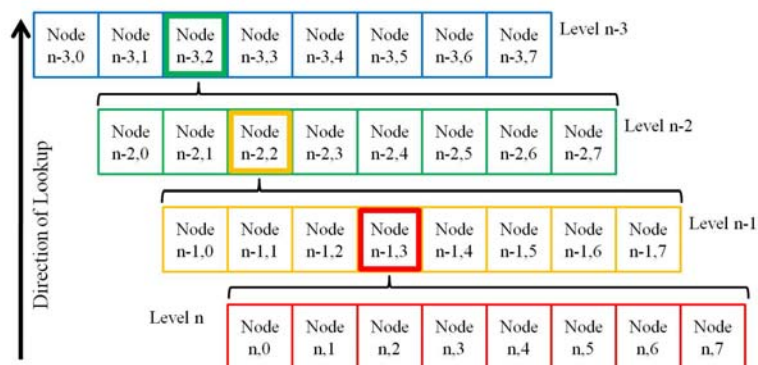


Figure 6.16 Diagram of a digital nested relationship to present the octree structure

To convert the volumetric space into an octree-based framework, an 8-bit RGBA 3D

texture called *Texture\_inter* was used to store this octree. As the basic component of these 3D textures, the Texel was used to record the distribution of nodes and the pointer-based relationships in the form of  $(R, G, B, Alpha)$ .

### 6.3.1 Implementing Octree Data Structure

To store this octree structure, the  $R, G, B$  triples were used to store the location of nodes as the texture coordinates in the octree-based lookup function. *Alpha* channel served as an indicator which determines the content stored in  $RGB$  triples ( $Alpha = 1$  means storing a leaf;  $Alpha = 0.5$  means recording the indices;  $Alpha = 0$  means an empty leaf). In order to simplify the explanation, all nodes in the octree had two coordinates: one for the volumetric space and another one for the texture space. The root node was located at  $(0, 0, 0)$  in *Texture\_inter*. This texture consisted of a number of grids (called *Grid\_inter*). For example, at depth  $D_L$ , the *Grid\_inter* was a cubic space consisting of  $2^{D_L} \times 2^{D_L} \times 2^{D_L}$  leaves. In like manner, *Grid\_inter* was only a cube of  $2 \times 2 \times 2$  leaves at the bottom level. After knowing the amount of voxels *voxel\_num*, the number of *Grid\_inter*, denoted as *Grid\_num*, will be different at each level in the data structures

### 6.3.2 Constructing Octree-based Lookup Mechanism

In texture processing stage, the octree-based lookup function was used to obtain a leaf's texture coordinates  $(e_{Leaf}, f_{Leaf}, g_{Leaf})$  (exterior voxels) at the depth  $D_L$ , and locate its coordinates. As a result, the leaf's texture coordinates can be calculated (Lefebvre, Hornus et al., 2003)

$$(e_{Leaf}, f_{Leaf}, g_{Leaf}) = \frac{(e_{(D_L-1)}, f_{(D_L-1)}, g_{(D_L-1)}) + frac((e_{Leaf}, f_{Leaf}, g_{Leaf}) \times 2^{D_L})}{(Grid\_num)} \quad (6.6)$$

where  $(e_{(D_L-1)}, f_{(D_L-1)}, g_{(D_L-1)})$  is the texture coordinates of the leaf's father node at the depth  $(D_L - 1)$ , and the *frac* function takes charge of keeping the fractional part of the parameter. The iteration of calculating texture coordinates was implemented in the process described by the pseudo code in Table 6.3.

```

Define the root node  $(e_0, f_0, g_0)$  and a random leaf  $(e_{Leaf}, f_{Leaf}, g_{Leaf})$ 
Define the number of grids Grid_num
Define the 3D textures Texture_inter
Define the depth  $D_L$  and the alpha value  $Alpha_{D_L}$ 
If ( $Alpha_{depth} < 0.9$ ) //not sampling the root node
    Calculating  $(e_{Leaf}, f_{Leaf}, g_{Leaf})$  at depth  $D_L$  in equation 6.6
    Fetching  $Alpha_{D_L}$  from Texture_inter with  $(e_{Leaf}, f_{Leaf}, g_{Leaf})$ 
        If ( $Alpha_{depth} < 0.1$ ) // empty node
            Break
        Return Grid_coord $[e_{Leaf}, f_{Leaf}, g_{Leaf}, Alpha_{D_L}]$ 
Else
    Break
Return Grid_coord $[e_{Leaf}, f_{Leaf}, g_{Leaf}, Alpha_{D_L}]$ 

```

Table 6.3 Mechanism for locating grid nodes

### 6.3.3 Accomplishing Volumetric Deformation

After finishing the octree-based lookup mechanism, *Grid\_coord* can provide the coordinates of the voxel based on its texture coordinates  $(e_{Leaf}, f_{Leaf}, g_{Leaf})$  and

depth parameter  $Alpha_{DL}$  in *Texture\_inter*. Its principle of generating the internal relationship is shown in Figure 6.17.

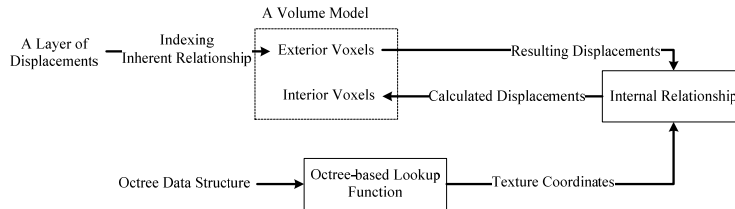


Figure 6.17 Diagram of generating internal relationship for indexing operations

The internal relationship was implemented via obtaining the interior voxels' coordinates by the octree-based lookup function. As a result, after obtaining the exterior voxels' displacements in the I-DOGME process, the interiors were assigned to calculated displacements by following the mechanism of vertex displacement calculation (explained in Table 6.1). Based on the I-DOGME and octree-based lookup function designs, the volume deformation successfully assigned computational deformations to the both exterior and interior voxels, and used a series of parameter settings to result different deformation extents (as shown in 6.18).

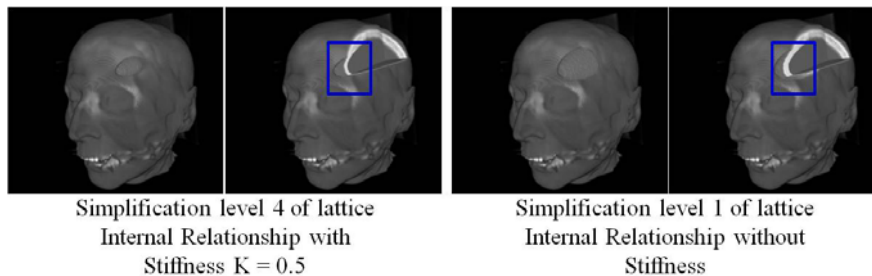


Figure 6.18 Results of volume deformation with different parameter settings



## 6.4 Fixing Deformation

As shown in Figure 6.18, the highlighted region represents the skull data belonging to rigid parts which should not perform “super-elastic” deformations. In order to fix the unrealistic features, the rigid part was firstly isolated before the deformation process, and independently manipulated with the dedicated lattices. In this solution, the connections between the rigid part and its surrounding regions are all vertices-based.

As the most prominent example, the result of bending a neck region contains rigid and soft parts simultaneously, and the distribution of every part is clear (as shown in Figure 6.19 (B)).

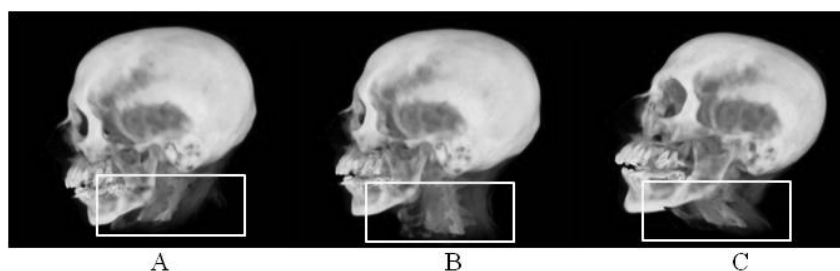


Figure 6.19 Results of fixed volume deformation

Therefore, this solution was tested on this example via implementing the isolation operations and iterating the deformation mechanism for manipulating the “cervical vertebra” data and its surrounding soft tissues respectively. As the rigid part, the “cervical vertebra” data was firstly deformed to represent the rigidly bended features. Then, its deformed control lattice between the rigid and soft parts served as an embedded lattice of the soft tissues. In other words, the vertices’ displacements in deforming rigid part were also mapped onto the soft part in the opposite direction.

After accomplish the deformation of the soft part, the results can be shown in Figure 6.19 (A and C).

### **6.5 Summary**

This chapter has presented the principle of I-DOGME deformation method, and an index mechanism for simplifying the calculation operations in displacement mapping phases. This deformation method can provide a unified approach to the specification of physics-based volume deformation in the searching interior voxels, and facilitate manipulations of volume models for showing complicated deformation behaviours. The capability of this deformation method can be demonstrated through comparisons of the results of phased improvements in constructing deformable lattices, accomplishing realistic deformation behaviours, constructing relationships between extracted vertices and voxels and determining voxels of interest.

The resulting deformation behaviours demonstrated the feasibility of utilizing this method to achieve FFD. Its advantages and versatility can be summarized as follows:

- This novel volume deformation function followed the idea of traditional DOGME method to connect the control lattice and its underlying deforming object. Based on this idea, the I-DOGME method was proposed to overcome the inherent problems insides the DOGME-based applications, and simultaneously preserve the displacement mapping operations inside volumetric space with the assistances of volumetric octree-based lookup

function. Consequently, this I-DOGME can solve the problems of manual lattice construction, and maintain the precision of transferring displacement information between the control vertices and corresponding voxels.

- This volume deformation function can perform applying forces on the volume object locally or globally. Different from the continuous and constrained deformation in the current achievements mentioned in chapter 3, this function can successfully enable partial deformation, i.e. discontinuous deformation. Besides the clustered data segments provided by the volumetric data processing function, the customisable control lattice in I-DOGME can also simplify the size of interesting data segment in deformation process for improving its efficiency. In addition, this function can also enrich more types of deformation behaviours than continuous deformation-based methods' work.
- This method can provide windows which allow real-time modification of the properties of control lattices and deformable solids. During the simulation, modifying the stiffness coefficient can enable the real-time customizations of the deformation behaviours. In addition, the lattice simplification process allows real-time management of constructed lattices, and consequently achieves variety of deformation results.
- Through performing the deformation behaviours via both vertex-based and voxel-based mass-spring systems, the progress of transformation between different elements can be technically represented in the manner of defined

deformation parameters and calculated displacements in this deformation function. As the result of this design, the whole deformation process can be restored the GPU-accelerated programme, and consequently maintain the efficiency of volume deformation function.

- This method can perform the physically precise representation of elastic and homogeneous solids. Unlike the limited deformation behaviours and unrealistic deformation results achieved by the constrained modelling strategies in non-physics-based deformation approaches, this designed volume deformation method can represent the results with free-form deformation extents by constructing lattices to partition the continuous data and customizing the deformation parameter settings for avoiding uniform transformations.

## **Chapter 7 System Integration and Acceleration**

This chapter listed the details of efficient volume deformation through constructions of GPU-parallel computing architectures. More precisely, a Single Instruction, Multiple Data streams (SIMD) architecture was constructed to solve the following problems: time-consuming iteration of mesh simplification loops in the lattice construction process, complicated indexing mechanism in running the octree-based lookup function, and huge data accessing workload in displacement mapping process. This project aimed to accomplish this design by utilizing an Nvidia graphics card and its parallel computing model (CUDA) (see Appendix A).

In the past ten years, a lot of CUDA-based inventions have been proposed for various applications. This project took advantage of several robust ideas to implement the goal of program accelerations. The design for adaptive lattice control in this project was derived from the idea of tessellation management for surface deformation (Bunnell, 2005). Besides, Bunnell also proposed the displacement mapping strategy which is a texture-based method for managing geometric transformation between surfaces. The texture mapping technique is widely used to “cause” an assembly of detailed and complex features on the surfaces of objects or 2D image planes. By developing this texture-based mapping technique for volume deformation, the spatial arrangement of voxels can be efficiently manipulated in the form of 3D textures. The efficiency of lattice modification in this project benefited from the investigation of

adaptive control of meshes. The use of octree textures to render complicated features in surface modelling applications provided a method of constructing a hierarchical structure to manage the spatial distribution of polygons (Lefebvre, Hornus et al., 2003). By applying this developed data structure to the volumetric space, each voxel can be exactly located, and consequently reacts to the deformations.

## 7.1 Preparations of CUDA-based Programming

In CUDA-based programming, the SIMD architecture is named Single Instruction, Multiple Threads (SIMT). A thread is the smallest execution unit in this programming model, and enables direct access to data arrays according to given indexes by means of texture coordinates. In other words, the whole data set can be accessed in order by an assembly of threads, and the entire data processing operation can be executed by parallelizing the computation tasks in groups of threads. A block is an assembly of threads, and uses the unique coordinates of the threads to construct the execution sequence in various ways, such as in concurrent, serial or other particular orders. Utilizing the shared memory, each block can implement the cooperation of its included threads by using the *synch\_threads* function. In addition to the concurrent execution sequence, a block can schedule various routes of progress for each thread, in order to achieve the anticipated execution sequences, such as the serial order and complicated combinations of multiple orders. As a group of blocks, a grid is the largest unit and takes charge of executing kernel functions. Apart from the cooperation operations of threads within the same block, there is no synchronization

between blocks because the shared memory is “exclusive”, i.e. every shared memory is just for one block. As a result, there cannot be any synchronization operations at block level. Figure 7.1 shows the hierarchical structure of the CUDA parallel computing model. In this project, the grids labelled with serial numbers were used to signify the usage of the GPU at different processing stages.

According to different purposes of acceleration design, there will present different sketch maps of block and thread arrangements in the CUDA programming structure. For example, in order to maintain the sequence of original sampling process inside CPU, the CUDA-based accumulation of voxels’ properties requires the arrangement of block and thread in the manner of grid 0 during the parallel processing work in GPU. Different from grid 0, the alternative arrangements for block and thread in grid 3 are respectively prepared for subdividing and simplifying operations in lattice refinement process.

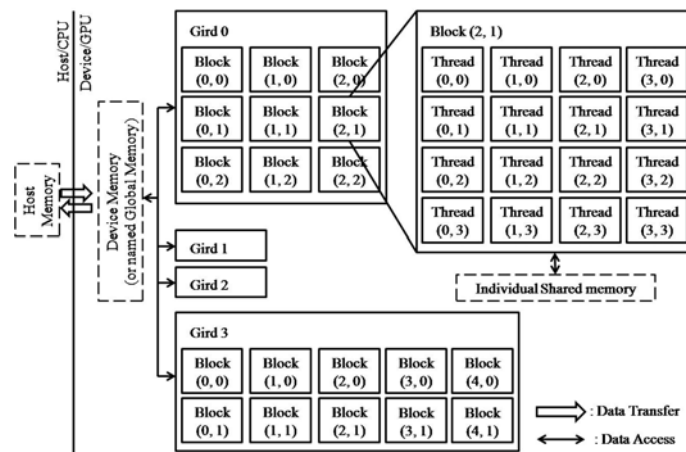


Figure 7.1 Diagram of hierarchical structure in CUDA programming model

The three investigated CUDA-based-applications mentioned in the above paragraphs

all follow the idea of constructing SIMT architecture to index partitioned data segments and synchronize the cycles of data processing by executing a kernel function. The following subsections will detail the CUDA-based procedural programming operations in the C++ programming environment, with related pseudo codes, diagrams and results.

## 7.2 CUDA-based Volume Visualization

In order to carry out a sequence of data registration, transmission, addressing and computation operations between a CPU and a GPU, this project treated the texture-based volume visualization technique as the basis in terms of system prototyping. The CUDA-based visualization pipeline is shown in Figure 7.2.

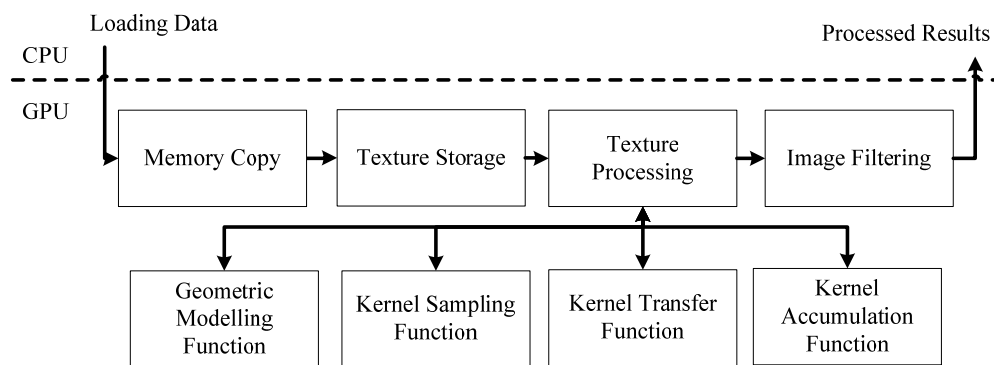


Figure 7.2 Diagram of CUDA-based volume rendering pipeline

In order to avoid complicated interpolation works and derived configuration processes, the choice of proxy geometry in this system was view-aligned textures. In this visualization process, data transfer between the CPU and GPU was implemented by parameter copying operations; for example, *cuda\_Mem\_cpy\_Host\_to\_Device*



denoted the direction of transfer operations, *cuda\_PitchedPtr* managed the properties (data source, data type and sizes of allocated memories), and various declarations of texture parameters (judgements on data normalization, filtering and addressing modes). By executing these parameter copying instructions, the volume data set could be successfully stored in the device memory in the form of 3D textures. After loading the data from the CPU, the visualization process can be divided into four stages: determining the geometric attributes of 3D textures in the geometric modelling function; configuring a rapid addressing mode in the kernel sampling function; creating a texture-based LUT in the kernel TF to “render” voxels, and constructing the kernel function to calculate the accumulation of optical properties. After executing these kernel functions, this visualization process would end by producing 2D results and uploading them to fragment operations in the CPU. In the following subsections, the four data processing stages will be detailed with their corresponding pseudo-codes.

### **7.2.1 Geometric Modelling Function**

In function, there were two kinds of size which need to be declared in advanced. One is the size of the data partitioning and the other is the size of the sampling region. Data partitioning takes charge of the *blocksize* function to create a thread-based presentation of the volume data. For example, when using *blocksize(a,b)* to represent a *sizewidth \* sizelength \* sizedepth* data set, each block can contain  $a * b$  threads. Consequently, the gridsize could be obtained from a calculation process which is simplified in terms of pseudo codes in example 7-1.

## Example 7-1. Example of calculating gridsize

```

Define sampling times (n,m);
Define data size blocksize(c,d);
Define sampling grid size gridsize(a,b);
Define division function iDivUp(x,y);
    If (x%y≠0)
        return iDivUp(x,y) = (x/(y+1));
    else
        return iDivUp(x,y) = (x/y);
//Calculate gridsize
gridsize(a,b) = iDivUp (blocksize(c,d), (n,m));
Return gridsize(a,b);

```

## 7.2.2 Kernel Sampling Function

As another important parameter in this function, the size of the sampling region was determined through setting a series of boundaries. In this visualization diagram, six imagined planes served as limitations of the sampling field (shown in Figure 7.3).

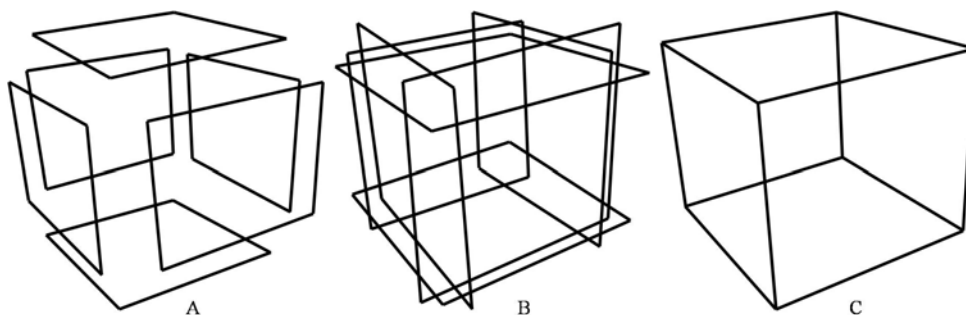


Figure 7.3 Illustrations of cubic sampling region

After finishing the data partitioning, and determining the sampling regions, the

single-channel-based sampling process in traditional texture-based volume visualization can be replaced by a multiple-channel-based sampling method. Because of the constructed SIMT architecture, the sampling operations can be divided and assigned to blocks. The sub-sampling operations in each block can be accomplished by parallelizing the executions of the kernel sampling function at thread level. In this sampling function, the *CUDA\_texture3D* function was used as the basic functional unit to retrieve the voxels directly from groups of threads.

### 7.2.3 Kernel TF

In kernel TF, the look-up-table was stored in a 1D texture. By defining *transfer\_offset* and *transfer\_scale*, the frequency of retrieval operations in 1D texture-based LUT can be determined. Therefore, the voxels extracted by the sampling process can be assigned with the retrieved values from LUT. These values were represented in terms of four dimensional float arrays (*colour (r, g, b, w)*), in which *colour.r*, *colour.g*, *colour.b* and *colour.w* respectively mean the *R, G, B* and *Alpha* value.

### 7.2.4 Kernel Accumulation Function

As the complement of the kernel function and the preparation for fragment operations in the CPU, the kernel accumulation function accumulated the optical values, converted the results into desired formats, and implemented self-labelling for rendering the results in the final display. The kernel accumulation function can be classified by means of the following pseudo codes:

Example 7-2. Example of kernel accumulation function

```
Define colour value   Color(r,g,b,a);
Define accumulated colour value   Sum(R,G,B,A);
Define the maximum number of circle   max_circle;
Define the upper limit for blending process   Opaci;
Define the block ID   BlockIdx(x,y);
Define the block dimension   BlockDim(x,y);
Define the Thread ID   ThreadIdx(x,y);
Define index number   index_number_(x,y);
//real-time calculation of index number
index_number_(x,y) = BlockIdx(x,y)*BlockDim(x,y) + ThreadIdx(x,y);
for (i = 0, i<max_circle; i++) //execute the accumulation circle
{
    Color(r,g,b) = Color(r,g,b) * Color.a; //multiply with alpha value
    /Sum(R,G,B,A) = Sum(R,G,B,A) + Colour(r,g,b,a) * (1 - Sum.A);
    if (Sum.A > Opaci)
        break;
}
Return Sum(R,G,B,A);
```

By using the serial numbers of the associated blocks and enveloped threads, the self-labelling tool “painted” each thread by following the thread’s numbers in blocks. In order to avoid the inaccurate computations caused by the various parameter values (such as 256 microns, 3 microns per 1 sampling shift distance, and 2048 units), there was a “clump”-like data normalization process, which constrains these values to be indexed using a special thresholding, [0, 1].

### 7.3 CUDA-based Lattices Construction

In I-DOGME design, an advanced lattice construction method can help to increase the efficiency of the whole deformation system. The combination of a mesh subdivision scheme and MC-based iso-surface construction can improve the accuracy and configurability of the lattices consecution process, and the CUDA-based combination can further improve the system in terms of meeting the requirement of a high interactive rate.

The pipeline of CUDA-based lattices construction is shown in Figure 7.4. Besides the common data transfer and volume data storage, this lattices construction process carried out an adaptive control of tessellation in extracted iso-surface, by designing three kernel functions. In the following subsections, the implementations of these kernel functions will be explained respectively.

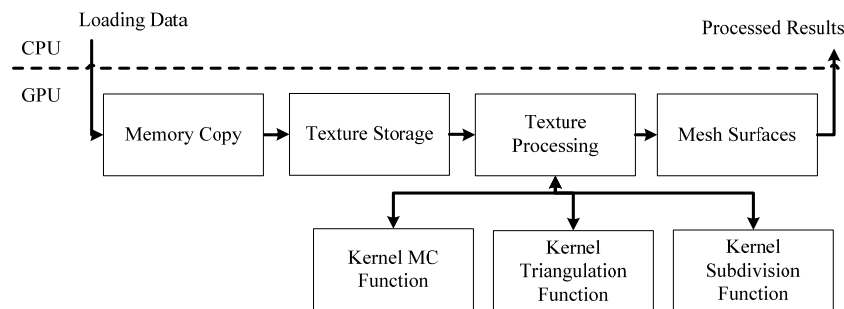


Figure 7.4 Diagram of CUDA-based lattice construction

#### 7.3.1 Kernel MC Function

After accomplishing the regular thread and block operations described in section

7.2.1, the serial sampling sequence in CPU-based MC process was converted into a set of synchronized executions of sampling windows. In other words, the CUDA-based MC process can simultaneously enable the vertices extraction processes on threads inside each block. The mechanism of parallelized vertices extraction in the kernel MC function is shown in example 7-3.

Example 7-3. Example of kernel MC function

```
Define the extracted vertex  ver_pos(x,y,z);
Define the sampling window  sampleWinsize(A,B,C);
Define the "leg" among sampling windows  sampleWinShiftsize(a,b,c);
Define the location of the sampling window  sampleWinPos(e,f,g);
Define the size of memory for a sampling window in manner of mask
                                         sampleWindSizeMask(h,i,j);
Define voxel size  unit3 voxelsize(1,1,1);
Define index number for classifying vertices  unit index_I;
//calculate current position of sampling windows
    sampleWindPos.e = index_I & sampleWinSizeMask.h;
    sampleWindPos(f.g) =
        (index_I >> sampleWinShiftsize(b,c)) & sampleWinSizeMask(i,j);
//locating vertices via calculating their coordinates
    ver_pos(x,y,z) = (-1.0,-1.0,-1.0) + (sampleWinPos(e,f,g) * voxelsize(1,1,1));
Return ver_pos(x,y,z);
```

As common properties in the MC process, a series of parameters need to be predefined, e.g. the size of the sampling window *sampleWinsize(A,B,C)*. According to the determined sampling frequency, the MC process treated a given

number of voxels as a standard volume. Based on this setting, all size parameters in MC process were all defined in terms of multiple standard volumes. In this project, MC was used to generate lattices because of its capability of passing through every voxel. Therefore, the standard volume in this kernel MC function was defined as a voxel size.

The volume of the one-off sampling space was written as a  $2^3 * \text{sampleWinSize}(A - 1, B - 1, C - 1)$  which was designed for avoiding sampling starting point twice. In the same way, the  $\text{sampleWinShiftSize}(a, b, c)$  represented the shift control of the sampling window, i.e. the uniform space between each two neighbouring sampling spaces was  $2^3 * \text{sampleWinShiftSize}(a - 1, b - 1, c - 1)$ .  $\text{voxelSize}(u, v, w)$  meant that each voxel occupied a  $2^3 * \text{voxelSize}(u - 1, v - 1, w - 1)$ . For simplifying the calculation workload, the  $\text{voxelSize}(u, v, w)$  was initialized to  $\text{voxelSize}(1, 1, 1)$ .

### 7.3.2 Kernel Triangulation Function

After finishing the sampling process, this kernel function served as a vertex shader to triangulate numerous polygons based on the extracted vertices. All polygonization modes (named cube configuration in MC-based applications) were indexed in the *ver\_Tex* texture.

Before the triangulation process, the extracted vertices were classified into a number of assemblies according to the different exterior voxels. The mechanism of locating these voxels is shown in Example 7-4.

Example 7-4. Example of kernel triangulation function (I)

```
Define index number for accessing voxels  index_inter_I;
Define voxel position  position (x,y,z);
//judging the state of intersection between sampling windows and voxels
position(x,y,z) = min(position(x,y,z), size(x,y,z)-1);
//calculate an index number
index_inter_I = (position.z*size.x*size.y) + (position.y*size.x) + position.x;
//fetch the voxel from the volume data
Return texture index_inter_I;
```

Example 7-5. Example of kernel triangulation function (II)

```
Define the value for iso-surface extraction  iso_value;
Define the sampling array  sampleWin;
Define the sampling status  sample_field;
Define the cube configurations  index_cube;
For (i=0, i<8, i++)
{
sample_field [i] = sampleWin (volume_data, sampleWinPos, sampleWinsize);
index_cube= uint (field[i]< iso-value)*2^i; //256 cube configurations
}
Return Index_cube;
```

After locating voxels, the inherent relationship between vertices and voxels in IIR design could be constructed, i.e. the number of vertices extracted from each exterior voxel could be obtained. The combination of vertices could be determined precisely, meanwhile, the cube configuration could be determined correctly. As a result, a mesh



surface could be accurately constructed through the triangulation process. The mechanism is shown in Example 7-5.

### 7.3.3 Kernel Subdivision Function

Based on the idea of tetrahedral-based subdivision scheme in CUDA-based Catmull-Clark application, the triangle-based application was devised and implemented to serve as the kernel subdivision function for achieving flexible lattice refinement operations. Before being stored into textures, the object mesh needs to be separated into triangle-based units through step\_1 in Figure 7.5. Then, these separated units will be treated as individual object in the following subdividing and simplification operations.

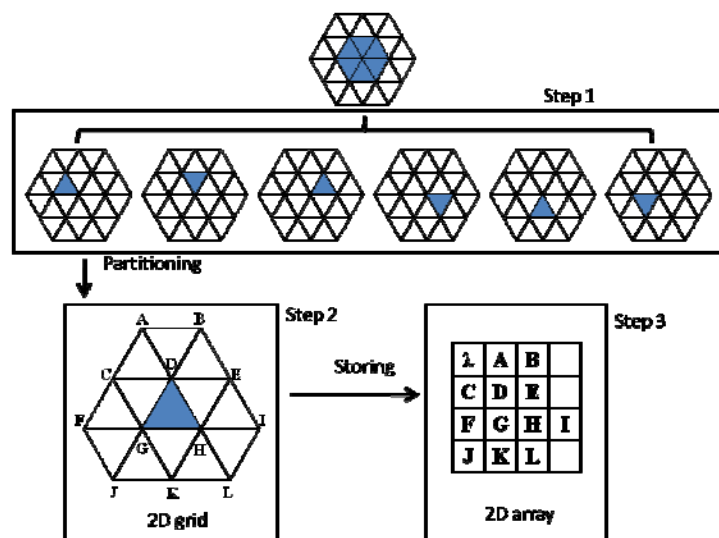


Figure 7.5 Diagram of triangulizing polygons

Afterwards, in step\_2, each unit was stored in the manner of texture units by indexing its vertices and surrounding points. Consequently, the results of associated subdividing and simplification operations on the each unit will be represented by the

modification of the above mentioned points. Besides, step\_3 revealed the state of sorting a unit in a 2D array which records the points' coordinates within the plane of this unit. The mechanism of kernel subdivision function is shown in Example 7-6.

Example 7-6. Example of kernel subdivision function

*Define the initial vertices  $vertex(x,y,z)$ , index number  $index\_ver(x,y)$  and modified vertices  $sub\_vertex(x,y,z)$ ;*

*Define weight parameters for subdivision  $\alpha(1/8,6/8,1/8)$ ,  $\beta(3/8,5/8,0)$ ;*

*Define two data array for storing initial coordinates  $ver\_co$  and fetching coordinates  $ver\_in\_co$ ;*

*if ( indicate = 0) //regular scheme*

*{  $vertex(x,y,z) = (ver\_co, index\_ver.x, 0)$ ;*

*$sub\_vertex(x,y,z) = vertex(x,y,z) + text3D(ver\_inte\_co, index\_ver.x, 0, 0) +$*

*$text3D(ver\_inte\_co, 0, index\_ver.y, 0) + text3D(ver\_inte\_co, index\_ver(x,y), 0)$ ;*

*}*

*Return  $sub\_vertex(x,y,z) * \alpha$ ;*

*else //reverse scheme*

*{  $Sub\_vertex(x,y,z) = (ver\_co, index\_ver.x, 0)$ ;*

*$vertex.x = sub\_vertex.x + text3D(ver\_inte\_co, index\_ver.x, 0, 0).x$*

*$+text3D(ver\_inte\_co, 0, index\_ver.y, 0).x$*

*$+ text3D(ver\_inte\_co, index\_ver(x,y), 0).x$ ;*

*$Vertex(y,z) = sub\_vertex.z + text3D(ver\_inte\_co, index\_ver.x, 0, 0).z$*

*$+ text3D(ver\_inte\_co, 0, index\_ver.y, 0).z$*

*$+ text3D(ver\_inte\_co, index\_ver(x,y), 0).z$ ; }*

*Return  $vertex(x,y,z) * \beta$ ;*

## 7.4 CUDA-based Displacement Mapping

In this system, displacement mapping process existed between extracted vertices and exterior voxels, and between exterior and interior voxels. The mechanism of displacement mapping process is shown in Figure 7.6. This design can solve the problem of time-consuming mechanisms of recursive octree transversal, and the associated indexing operations were all implemented on a CUDA-based octree data structure to achieve the goal of acceleration.

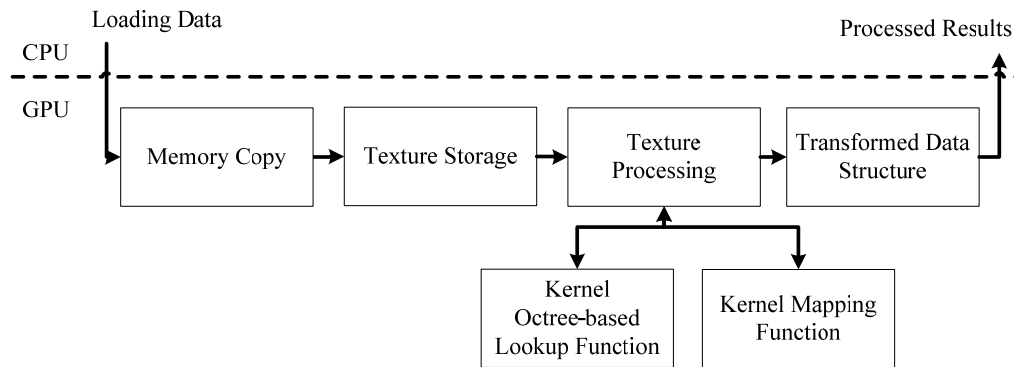


Figure 7.6 Diagram of octree data structure and the displacement mapping design

### 7.4.1 Kernel Octree-based Lookup Function

In this constructed hierarchical data structure, the voxels were represented in a part of the octree in the form of a terminal or non-terminal octant, and the nested relationships between different nodes determined the assignments of displacements to corresponding voxels. In the I-DOMGE process, the number of exterior voxels was determined by the status of modified lattices, e.g. the higher simplification level will cause less voxels to be influenced because the decreased amount vertices in the

lattices or vice versa. In the same way, the depth in the octree data structure determined the number of voxels which can be located by the lookup function. Figure 7.7 shows different results of tree lookup function with different depths (represented via levels). By changing the depth attributes in the lookup function, the final results of the tracking process will perform various statuses in the form of the numbers of voxels.

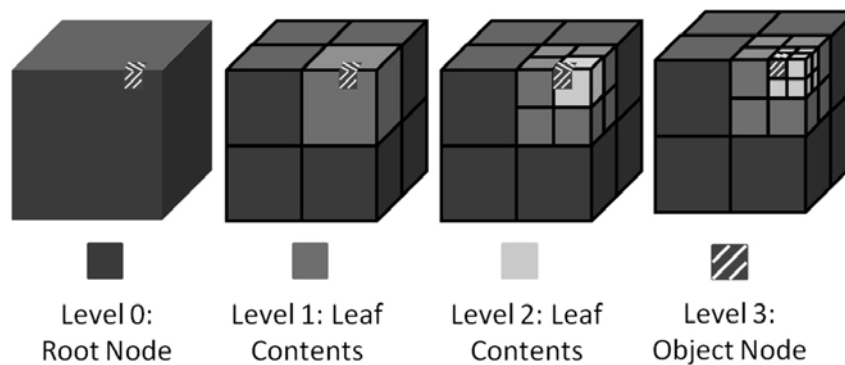


Figure 7.7 Results of octree-based lookup function

In this octree-based lookup functions, there was an indirection “pool” which is for retrieving the newest lookup results. If the results denote an index, the lookup function will carry on to a deeper level. Otherwise, the latest result in the indirection pool will be treated as the final output. In the volume deformation process, the indirection pool was designed to enable a real-time data storage record the latest voxels’ sequence number. These sequence numbers can form a “connection” between the exterior voxel and its underlying interiors, and represented it in the form of a group of leaf nodes at different depth levels in octree data structure. In other words, the “connection” recorded a set of nested relationships (as shown in Figure 7.7). The mechanism of the kernel tree lookup function is shown in Example 7-7.

## Example 7-7. Example of kernel tree lookup function

```

Define the location of object node(s)  node_co (x,y,z);
Define the location of object node(s) in indirection pool  node_inte_co (x,y,z);
Define a texture for storing intermediate coordinates texture  co_inte_tex;
Define a texture for storing initial coordinates texture  co_tex;
Pointer to a data array for storing coordinates  coord;
Pointer to a data array for storing coordinates on the connection  conne_co;
Define the size of volumetric objects  volsize (w,l,d);
Define the upper limitation for lookup function  lookup_max_depth;
Define index number for accessing coordinates  index_co (x,y);
Creating octree-based lookup (node_co, node_inte_co, coord);
{
    node_inte_co = make_float3 (0.0, 0.0, 0.0); //initialize indirection pool
    node_co = make_float4 (node_co (x,y,z), 0.0);
    for (i = 0, i < lookup_max_depth; i++)
        {
            node_co = texture1D (co_tex, volsize.d * index_co.x * index_co.y);
            node_inte_co = make_float3 (node_co (x,y,z));
            Comparing node_co.w with 0.9 and 0.1;
        }
    Return conne_co[index_co(x,y)] =
        (node_inte_coord(x,y,z), (volsize.d * index_co.x * index_co.y
        + volsize.l * index_co.y + volsize.w));
}

```

**7.4.2 Kernel Mapping Function**

Example 7-8 shows the mechanism of the kernel mapping function.

Example 7-8. Example of kernel mapping function

```

Pointer to a data array for storing voxels' displacements  voxel_co;
Pointer to a data array for storing displaced coordinates  disp_tex;
Define the upper limitation for indexing  index_label_max;
Define the location of object node(s)  disp_co (x,y,z);
Define the changed location of object node(s)  disp_inte_co (x,y,z);
Define the texture coordinates  index_co (x,y,z);
Pointer to a data array for storing coordinates  conne_co;
Define the size of volumetric objects  volsize (w,l,d);
//displacement mapping function
Creating disp_tex (disp_co, disp_inte_co, conne_co);
{
  for (i = 0, i < index_label_max; i++)
    {
      index_co (x,y,z) = texture3D(conne_co);
//indexing displacement values
      disp_inte_co(x,y,z) =
          texture1D (dis_tex, (volsize.d * index_co.x * index_co.y
              + volsize.l * index_co.y + volsize.w));
//attach a displacement map to the original coordinates recorded in pool
      disp_inte_co(x,y,z) = disp_inte_co(x,y,z) + disp_co(x,y,z);
    }
Return  voxel_co [index_label(x,y)] = make_float4 (node_inte_co (x,y,z),
          (volsize.d * index_co.x * index_co.y
          + volsize.l * index_co.y + volsize.w));
}

```

Based on the implementation of the kernel octree-based lookup function, the related regions surrounding the control point can be efficiently located and the connections can be recorded in *conne\_co* texture with their coordinates. Consequently, the new coordinates calculated by manipulating the mass-spring system can avoid being assigned to other meaningless regions. For example, the *conne\_co* ( $x, y$ ) can index a cluster of nodes processed in the ( $x, y$ ) block. By using *conne\_co*, an efficient carrier can be designed to transfer the coordinates between nodes and corresponding voxels in the form of 3D displacement mapping mode.

## **7.5 Summary**

### **7.5.1 SIMT Architecture**

As described in this chapter, all volume-based processes were parallelized and synchronized in the SIMT architecture. By determining the properties of thread and block, volumetric content were averagely partitioned and assigned to blocks and underlying threads. As a necessary preparation for various parallelization designs, SIMT architecture labelled every block and associated threads with a unique indexable serial number, which serves as recording the sampling sequence in synchronization process.

### **7.5.2 Synchronizing Kernel Functions**

By taking advantage of GPU programming, single-channel-based processes were converted into multiple-channel-based ones by means of simultaneous executions of

multiple kernel functions. Compared with corresponding CPU-based implementations, the CUDA-based acceleration designs in this system did help with achieving high efficiencies, supporting complicated processes and improving the trade-offs between effect and speed.

In the next chapter, the system will be tested by experimenting on each processing stage. The increased efficiency, improved visual effects, configurable operations and derived benefits of the purposive designs will be listed, in order to testify to the feasibility of this designed volume deformation system. The contribution to knowledge will be evaluated by a series of comparisons with similar research achievements.



## Chapter 8 Test and Evaluation

After completing the functional module designs and accelerated implementations, a series of tests were conducted to evaluate the performances of three key processes: volume data segmentation, lattice construction, and interactive deformation, which were presented in this chapter. The results of the experiments can be used to assess the validity and effectiveness of this interactive volume deformation (IVD) designs.

### 8.1 Efficiency Evaluation on Volume Segmentation

The segmentation designs in the volume data processing module tried to extract two kinds of segmentation masks from the volume data. One mask was used to number all classified segments inside the volumetric space. The other one was used to isolate the interesting segment(s) from the same space. Section 4.1 mainly focused on the usage of the first mask. With the context of segmentation improvement and representing clusters, the performances of the second mask were covered in the subsequent sections, as the key information in the other two experiments.

Although clipping techniques can provide a rapid presentation of the volume models' interiors, a greyscale visualization of volume data cannot fully describe the differences between data segments which share the same scalar value, i.e. the regions rendered with the same intensity information (e.g. the highlighted regions in Figure 8.1 (A)). Therefore, the multidimensional TF was devised to implement a further

segmentation process visually (as shown in image C). However, in using different colours to emphasize the differences between these two regions, the associated trial and error tests in modifying TF manually (as shown in image B) cost too much time, because of the lack of a standardized colouring plate.

By designing an automatic clustering-based segmentation method, the clustered results can be used to automatically generate a standardized colour combination (as shown in image D) to replace the manual configurations. This devised function not only achieved a data-driven analysis of the volume data, but output similar effects to the traditional multidimensional TF's results. Both TF function designs illustrated in Figure 8.1 can separate the same volume data sets into five parts with rendering them in corresponding colour properties.

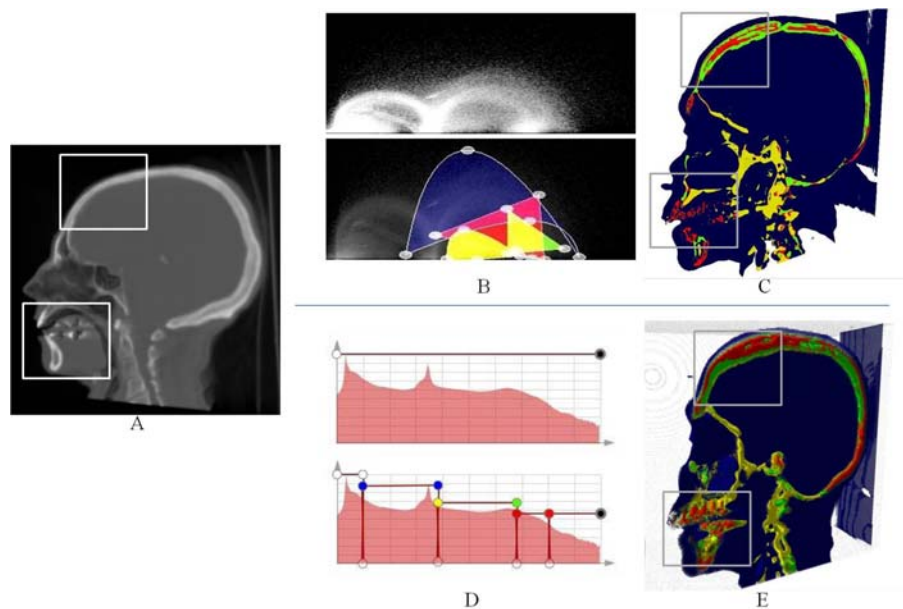


Figure 8.1 Results of DVR with a 2D TF and the ATF design

Table 8.1 records the performance of testing the volume data processing module with

different volume data. By comparing the  $K$  values generated by MSVS and KMVS, it can be observed that this module filtered the over-segmentation generated in MSVS via using KMVS's low sensitivity to tiny features, and avoided the extra computations for fixing under-segmentation in KMVS. With an increase in volume data size, the reduced cluster number can demonstrate the advantage of the integration of MSVS and KMVS, and the visualized results are shown in Figure 8.2.

	Data Size (KB)	MSVS Processing Time (s)	Generated K in MSVS	KMVS Processing Time (s)	Output K in KMVS
Inner Ear	0.48k	430	7	24	4
Teddy Bear	0.9k	541	16	61	4
Engine Data	7.0k	1k	9	109	3
MRI Human Head	16.0k	3k	31	222	5
CT Human Head	27.1k	4k	65	438	6
Celiac Data	237.0k	10k	112	600	9

Table 8.1 Results of using KMVS and MSVS to process different data sets

As a pre-processing operation, the automatic data analysis just served as a one-off guide for enabling a comprehensive display of the volume data, and was not iterated in the subsequent operations. Therefore, the performance of future deformation processes will not be restricted by a lengthy processing time.

After accomplishing this volumetric data processing function, different volume data sets were tested and their processing results were shown in Figure 8.2. The coloured features in different data sets can verify that the uniform configuration of rendering properties in ATF function can enable all elastic parts (such as the cartilage appendage in ear data, the twistable fastener in the nose area of teddy bear data, the stratum corneum in MRI human head data, the gum in CT-scanned human head data, and the soft tissue in human celiac data) to be rendered in yellow. In the same way, the high density parts (the auditory canal part, the teddy bear's crust, the engine's framework, the soft tissue canned by MRI, the mixed osseous features in CT data, and the human backbone) were highlighted in green. And the intermediate data was fulfilled with blue features.

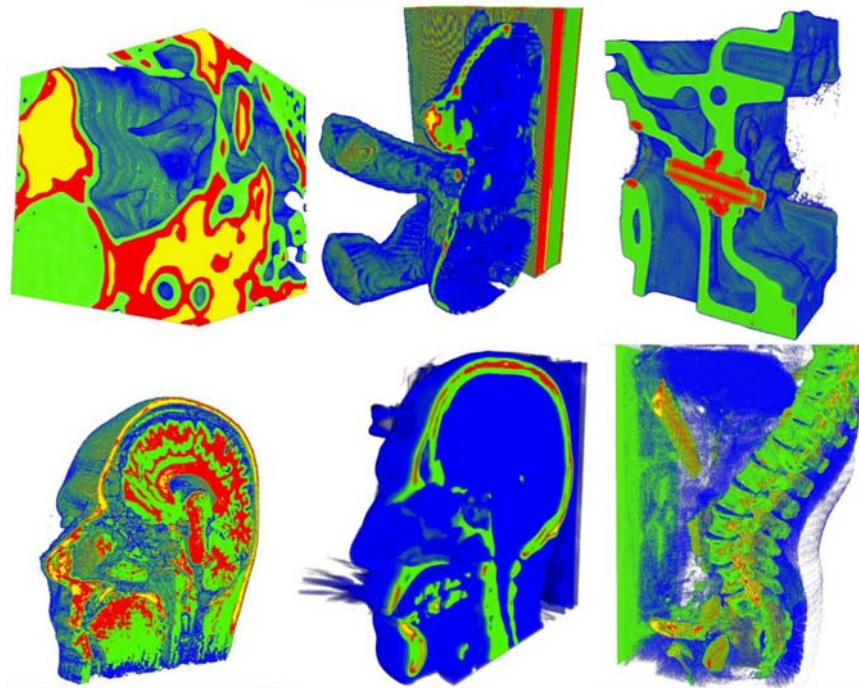


Figure 8.2 Results of the volume data processing module

## 8.2 Effectiveness Test on Lattice Construction

Because the visualized results can only trace the boundaries of clusters visually, specific operations were required to parameterize the interesting segment(s) for further operations. In other words, the visualization process cannot prevent unnecessary parts from joining all the subsequent computations. Therefore, the second segmentation mask for isolating interesting data segment(s) from the continuous volume was iterated for localizing the related voxels spatially. In addition, the size of the processing data was decreased by using this mask to filter out the unnecessary parts. The associated computation workloads was reduced and the system efficiency was improved as well as.

In the lattice construction process, this mask shortened the sampling range before executing the MC algorithm. As shown in Figure 8.3, these isolated part (delineated by blue lines in image A, C, E and G) were respectively represented via corresponding iso-surfaces (revealed in image B, D, F and H). The numbers of extracted vertices inside the isolated results are 13K, 2K, 21K and 109K, different from the previous ones 114M, 371M, 220M and 1200M. The associated frame rates will be listed in the section 8.4. As a result, the constructed lattices for volume deformation can be the “model-fitting” one which closely envelops the deformation object.

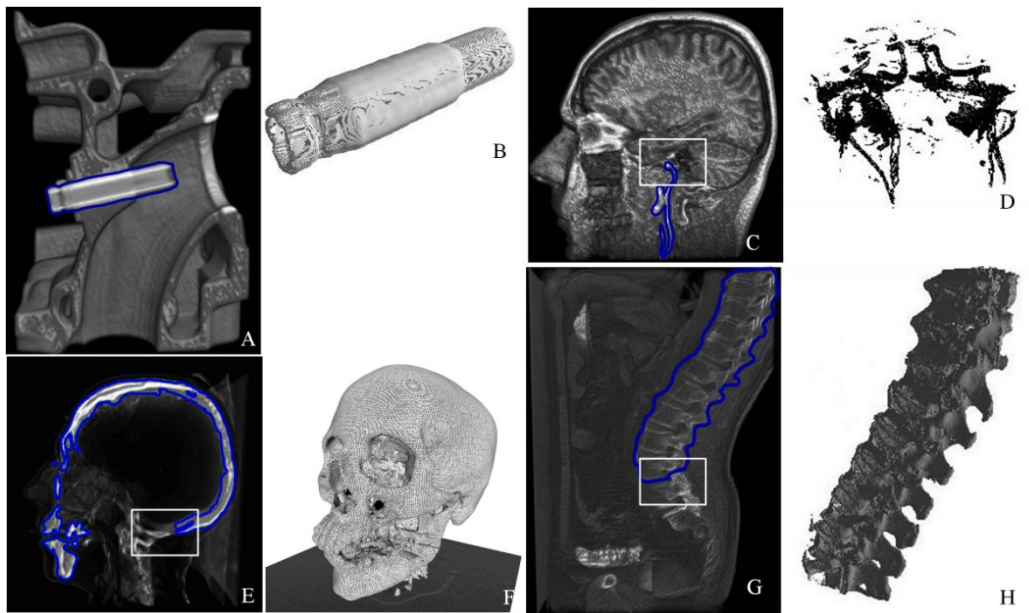


Figure 8.3 Results of extracting lattices from the isolated segments

### 8.3 Flexibility Assessment on Interactive Deformation

In this thesis, the principle of the devised FVD method consisted of three concept designs. The first design, I-DOGME, parameterized the applied forces through implementing a general mesh deformation solution onto the constructed lattice.

For indexing the displacement mapping operations between the vertices on the control lattice and the exterior voxels, the second design used the IIR derived from the MC-based lattice construction process to form a dedicated LUT. In order to address each vertex in a unique index, no new vertices are permitted to be generated after the lattice refinement process.

The third design involved the construction of a volumetric displacement diagram to characterize the interior voxels' movements. If there is no further “depth” calculation,

its mechanism will be partly similar to the latest non-physics-based volume deformation approach, whose crucial achievements are illustrated in Figure 8.4 (A to D). Correspondingly, images F to D show the similar outputs implemented by the third design without the “depth” calculation.

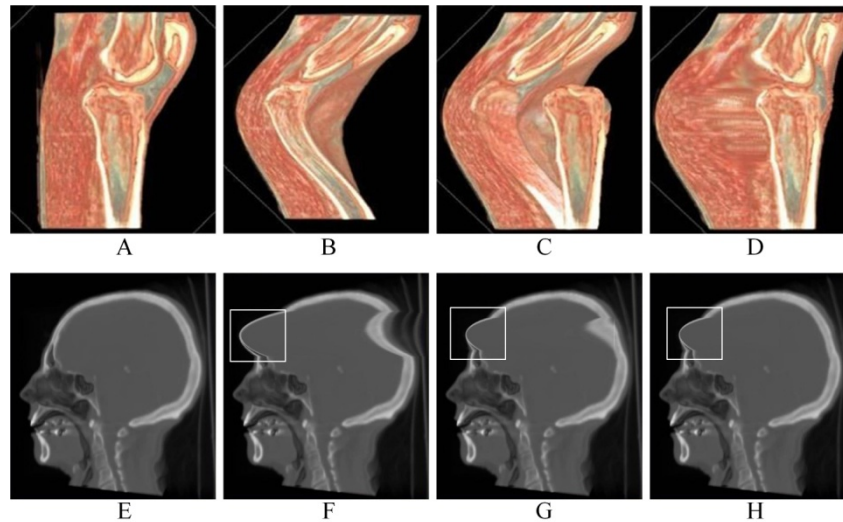


Figure 8.4 Results of non-physics-based deformation. Images B to D respectively represent the results of axis-aligned linear transformation, feature-aligned isolation operation, and constrained interpolation design (courtesy of Correa *et al.*)

As shown in Figure 8.4 (F to H), the highlighted behaviours, namely the axis-aligned results, exhibit a series of unnatural shape changes which merge the skin data and skull data together. Different from the manually constrained operations in this latest non-physics-based volume deformation approach, the third design utilized an octree-based lookup mechanism to provide a set of internal relationships for connecting exterior and interior voxels, and treats the relative distances between vertices and voxels as the “depth” parameters to implement a gradient distribution of displacements (as shown in Figure 8.5). In other words, the calculated depth values

were regarded as the specific factors which parameterize the gradient changes in the distances.

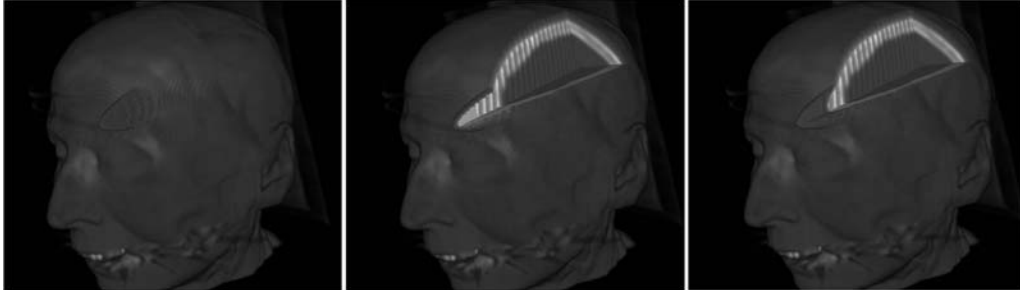


Figure 8.5 Results of the IVD method

After implementing these three concept designs, the above deformation behaviours can demonstrate the feasibility, applicability and efficiency of the devised IVD method system.

#### **8.4 System Run-time Performance Evaluations**

The run-time evaluation work was based on performances obtained on a consumer grade desktop which was mainly equipped with an Intel Core 2 Quad Q9400 CPU, 4G RAM and a Nvidia GeForce GTX 260 graphics card. This section divides the evaluation work into four parts.

- Volume data analysis



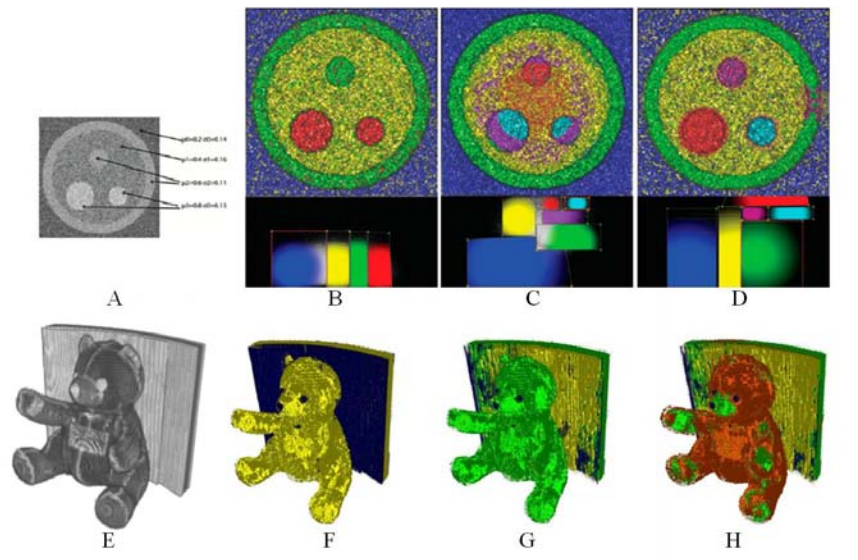


Figure 8.6 Comparative results of different TF designs. Images B to D are the results of latest TF designs (courtesy to Liang *et al.*)

In Figure 8.6, images F to H respectively show the visual results of automatic volume data analysis generated in the volumetric data analysing process. Images (B to D) show the results of different chains of multiple TFs (e.g.  $2D + 1D + 1D + \dots$  or  $2D + 2D + 1D + 1D \dots$  or even  $2D + 2D + 2D + \dots$ ) decided in the latest TF design (Zhou, Schott et al., 2012). The results all highlight the specificities inside volume data. The automatic TF design described in this thesis can efficiently output similar results to those designs which need to rely on expensive trial and error operations for evaluating the chain of multiple TFs in the TF combination approaches.

- GPU-acceleration design

Figure 8.7 delineates a speed distribution plot which records different performances of using IDV to deform different volume data sets on different platforms. From this figure, the advantage of GPU-based volume deformation system over the CPU-based

can be clearly represented even processing simple data sets. With the increase of volume data size, the benefit from implementing GPU platform is gradually decreased, but the real-time record of frame rates of GPU-based acceleration designs is still higher than the corresponding programs in CPU. More precisely, the GPU platform can enable more volume deformation program to run at a “responsive” speed (framer rate is between 14 and 24 fps) than CPU works.

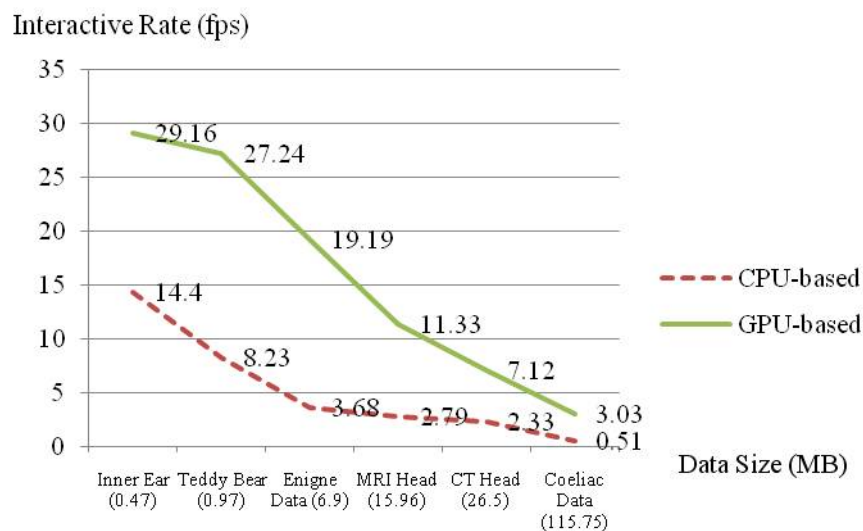


Figure 8.7 Comparative results of different implementations

Figure 8.8 illustrates the comparison between the devised IVD method and the latest volume deformation solution (named constrained illustrative volume deformation (Correa, Silver et al., 2010)) in processing volume data sizes (less than 40MB). It can be seen that the performance data for the IVD meet the criteria for interactive deformation, alongside which the constrained illustrative volume deformation also followed in its system performance evaluations.

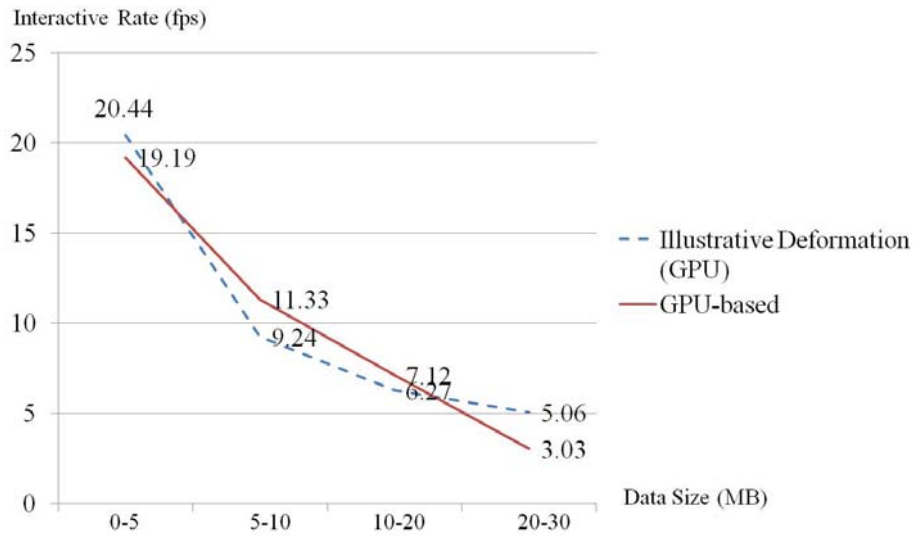


Figure 8.8 Comparative results of different deformation solutions

- Adaptive lattice simplification

The lattice simplification served as an optimization of the lattice construction process, which reduces the number of extracted vertices in order to generate different levels of resolution, as illustrated in Figure 8.6. Since the voxels' displacements derive from these vertices' properties, the simplified lattices can enable the result to represent different deformation extents (as shown in Figure 8.6 (A to D)).

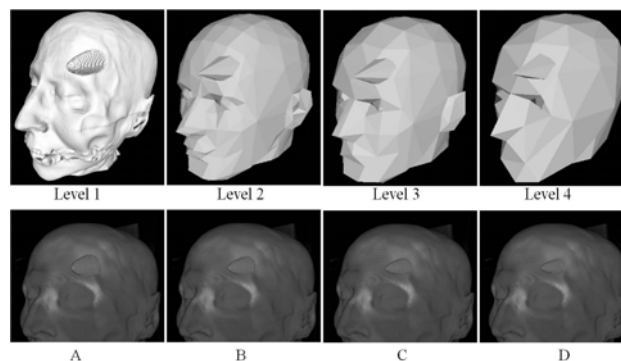


Figure 8.9 Results of deformed behaviours resulting from different lattice simplification levels

Furthermore, the simplified lattices can accelerate the deformation process. Figure 8.10 shows the interactive rates of testing different levels of resolution. The lattice refinement will overlap several vertices during the reverse subdivision process. Consequentially, the simplified control lattice might loss a few overlapt vertices' displacements and lead to a tiny influence of resulting deformation behaviours. However, as shown in Figure 8.10, the higher simplification level number will lead to higher frame rates, especially work on processing the large volume data set, e.g. maintaining the deformation of human celiac data at 19.36 fps on the simplification level 4.

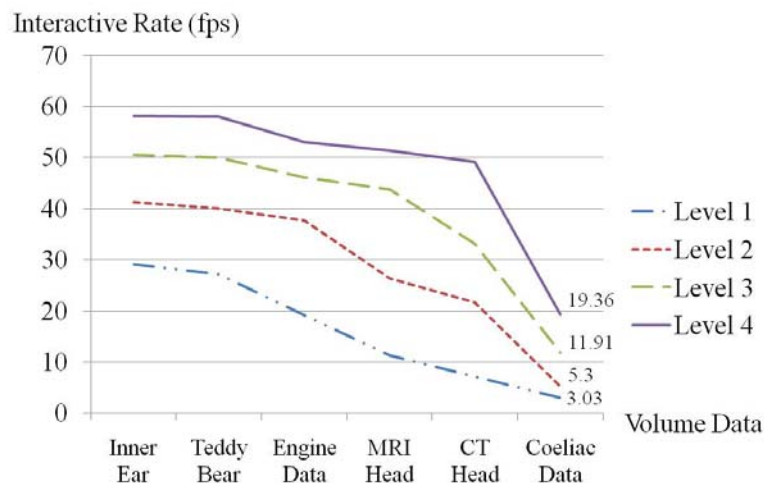


Figure 8.10 Comparative results of different lattice simplification levels

- Freeform deformation behaviours

Figure 8.11 illustrates a number of deformation results of the IVD method. In images (A to H), the results of non-physics-based deformation were generated by implementing a series of linear transformation on the extracted lattices. As shown in image I to L, the deformation approach enabled the representation of gradient changes

in the deformed areas.

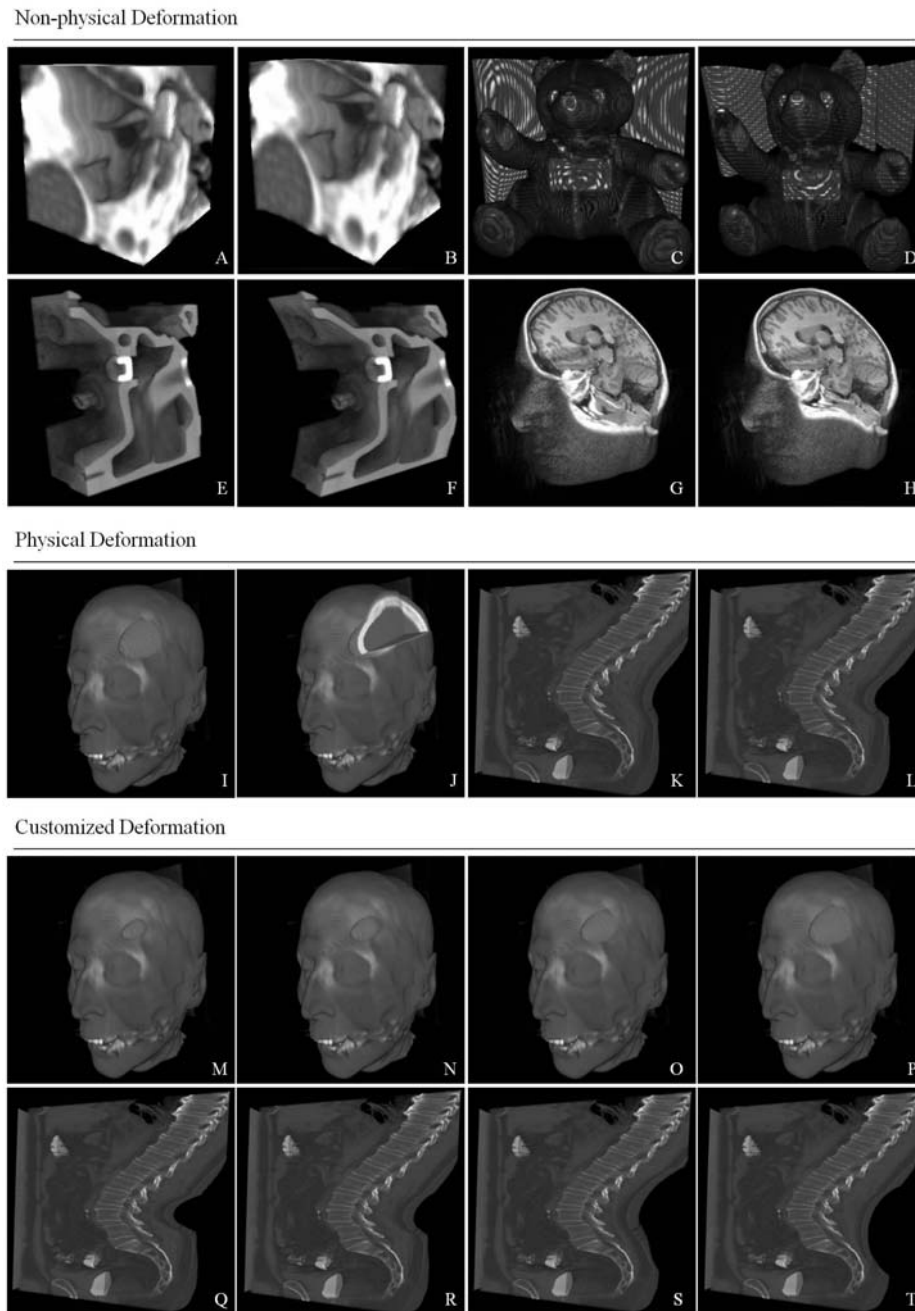


Figure 8.11 Various results of IVD method

In addition, by configuring the simplification level of the lattices, the variables of the mass-spring mechanism and the properties of the volumetric displacement diagram, the deformation mechanism can be modified to output various customized

deformation results (shown in image M to T).

## 8.5 Summary

This chapter has described the system test operations which were implemented from a quantitative perspective. Firstly, the important concept designs were summarized with presenting their test results. Then, the performance evaluation began to quantify the achievements of the IVD system through a series of contrasts in interactive rates. Simultaneously, a number of deformation results were listed to demonstrate the capabilities of the system in terms of freeform deformations and customized manipulations. At the end of this chapter, this section aims to claim the advantages of the IVD approach by comparison with the latest physics-based and non-physics-based volume deformation approaches (assisted breast survey (Patete, Iacono et al., 2012) and constrained illustrative volume deformation (Correa, Silver et al., 2010)).

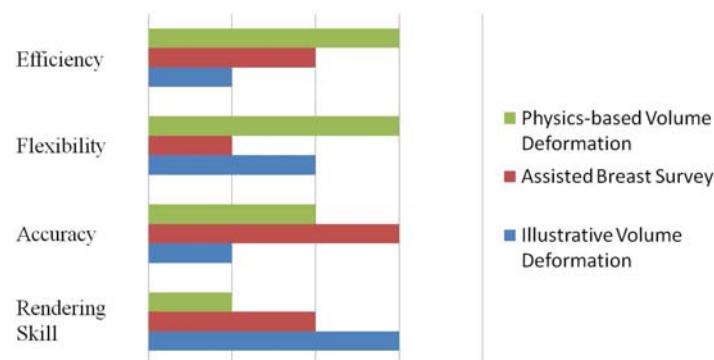


Figure 8.12 Comparison table showing performances of IVD, assisted breast survey (ABS) and constrained illustrative volume deformation (CIVD)

As shown in Figure 8.12, the IVD approach represents the highest efficiency because

its automatic visualization diagram and accurate data deformation. It is more convenient than the constrained off-line interpolation operations in CIVD and the semi-automatic procedures in ABS. As a dedicated clinical analysis application, ABS gives the highest accuracy because it studies the deformation behaviours in microns. However, the specific requirements of clinical simulations restrict the usage of the ABS approach. In addition, the non-physics-based deformation cannot assist CIVD in implementing true flexible deformations as well as the physics-based approach's results. Due to the constructed mass-spring system in both the IVD and ABS approaches, their deformation behaviours are more accurate than the manually constrained results in CIVD. The biggest advantage of CIVD is enabling volume shading terms in rendering deformation results. The other two approaches just focus on the deformation extents and leave the associated rendering designs to future operations.

By summarizing this comparison table, this chapter has managed to demonstrate the feasibility of the IVD design and the test results have given enough evidence of the applicability of this design in real-time manipulation applications.

## Chapter 9 Conclusions and Future Work

This thesis presented a novel physics-based volume deformation approach that enables real-time manipulations of volume data. A new notion of displacement mapping was devised for processing the voxels' displacement parameters, so that the presented approach could perform a series of flexible and interactive volumetric deformation behaviours. The implementation of this deformation approach was accomplished in 4 key phases, volume data processing, lattice manipulation, deformation control and GPU-accelerated implementation.

The construction of volumetric data processing function can perfectly solve the limitations of complicated volume data analysis and successfully make the visual results meet the pre-defined criteria of visualization term. Besides, this function simultaneously prepares the isolation of interesting data segment(s) for the following lattice and deformation operation. About the associated rendering of deformed features is left to in the future work plan.

As an intermediate process in this deformation system, the lattice manipulation succeeds in building up model-fitting lattice, mapping the vertices displacements onto voxels and maintaining the system performance via its lattice refinement function. However, its shortage in preserving the integrity of displacement information during the lattice simplification phase can lead to a few artefacts. For example, according to the mesh deformation tested by Patete, the influence of displacement will lead to the



0.01%-1.0% difference to the real result (Patete, Iacono et al., 2012). With the idea of further rendering, the loss of displacement will be mentioned in the future work plan.

In addition to the visualization improvements, the novel physics-based volume deformation design is another important part in this project. First of all, this deformation function can not only accomplish both physics-based and non-physics-based volume deformation behaviours, but support customisable and localisable manipulation of interesting data segment(s).

The GPU-based acceleration designed had sufficiently exhibited its power of maintaining the designed volume deformation system to enable the real-time customisation operations at an interactive rate (over 14 fps).

## **9.1 Conclusions**

### **9.1.1 Efficient Volume Data Processing**

Chapter 3 described a solution which analyses the volume data structure and enables a volumetric features extraction function by using two kinds of segmentation masks. By developing the image segmentation algorithms to classify the volume data, an automatic volume segmentation process was constructed to generate one kind of segmentation mask which records the properties of the volumetric data segments. This mask was rendered by a set of visual information which was generated by an online LUT in the ATF design, which was designed to paint the associated data segments in the final display in the data-driven mode. This visual information can describe the

volume data structure, based on which the other kind of segmentation mask was used to isolate the interesting segment(s) from the volumetric space.

As a data pre-processing approach, the volume data processing was designed to complete the information extraction before starting any further processing operations. Therefore, its processing time was not considered in the evaluations of the performance of the deformation processes.

### **9.1.2 Adaptive Lattice Manipulation**

The construction of control lattices played an important part in the deformation preparation stage. Chapter 4 explained a solution for constructing lattices to enclose the volume data. The feasibility of the solution was proven by the fact that its MC-based lattice construction successfully generated “model-fitting” control lattices which could completely enclose the deformation object and precisely match its outlines. It could avoid the manually determined boundaries which usually cover a number of unnecessary parts, and result in constrained assumptions.

Due to the high sampling frequency of the method, the extracted lattices comprised overabundant control points. Therefore, a number of mesh simplification methods were tested to solve the problem of oversampling successfully. Besides the criteria for different simplification mechanisms, the mesh simplification design explained in Chapter 4 laid down a new one, which stated that both the generation of new vertices and the merging of several vertices into a new one were not permitted. This criterion was intended to ensure a relationship by which each extracted vertex could be

addressed to a voxel by indexing it in the IIR design.

### **9.1.3 Flexible Deformation Control**

After finishing the lattice construction, the deformation preparation was continued by executing the other important part: displacement mapping. In Chapter 5, the framework presented for a new displacement mapping method consisted of two conversions. The first conversion used the vertices' displacements to parameterize the deformation on the control lattices, and mapping them to exterior voxels. This conversion relied on the IIR between the extract vertices and a layer of exterior voxels. The second conversion was between the exterior voxels and the underlying volumetric space. In order to implement a resulting distribution of movements inside this space, the notion of a volumetric displacement diagram was implemented by devising an octree-based lookup function to locate the interior voxels, and assigning the computed displacements to them. Based on these two conversions, the volume deformation could be successfully implemented, so that the deformation operations could be parameterized within the associated variables.

### **9.1.4 GPU-accelerated System Integration**

Chapter 7 encoded a series of GPU-accelerated implementations. The time-consuming data access and computation operations were separated into a set of sub-tasks. By describing the associated principle of constructing kernel function and iterating it on all subtasks simultaneously in the CUDA-based parallel processing framework, the large vertex extraction workload and the complicated conversions design in

deformation preparation could be accomplished efficiently. Based on the results of the system evaluation described in Chapter 8, it has been proven that GPU-based implementations can enable rapid deformation representations and interactive manipulations of volume data.

## 9.2 Future Work

The novel physics-based volume deformation pipeline presented in this thesis can manipulate the voxels' displacements to achieve complicated simulations and freeform deformations.

There are a number of imminent steps that once realized could further improve the IVD system. First of all, the volume data processing can be implemented on GPU, so that its accelerated performance can enable a rapid display of analysis results. In order to highlight the given features in deformation results, the volume data processing should allow a series of real-time (or near real-time) modifications for customizing them. Its GPU-accelerated implementations will rapidly provide direct feedback of the modifications of volumetric data analysis module. Secondly, a new approach to local mesh simplification, which divides the control lattice into useful and useless meshes, can be implemented to reduce the number of vertices in the simplification process. Meanwhile, the mesh simplification design can be assisted by a dedicated function which serves to preserve sharp features during the simplification period. Thirdly, the deformation behaviours can be enriched by implementing discontinuous

operations and results, both of which can be used to simulate ruptures in manipulating rigid materials.

In addition, the GPU-based visualization pipeline has shown potential for adding advanced rendering terms to the final results. The envisaged future works can extend the current system to comprise a comprehensive set of latest illumination terms through enabling layer-based or voxel-based rendering mode. For example, the rendering effects will be generated by the approximate Monte-Carlo light transports on the layers of voxels. For high-precision applications, e.g. clinical simulations, the computed deformation results should not only manipulate the large volume data accurately, but also shade correct illumination effects corresponding to different tissue substances through implementing the lighting scattering methods.

## References

Arens, S. and G. Domik (2010). "A survey of transfer functions suitable for volume rendering." *Volume Graphics 2010*: 77-83.

Aykanat, C., B. B. Cambazoglu, et al. (2007). "Adaptive decomposition and remapping algorithms for object-space-parallel direct volume rendering of unstructured grids." *Journal of Parallel and Distributed Computing*, 67(1): 77-99.

Azar, F. S., D. N. Metaxas, et al. (2002). "Methods for modeling and predicting mechanical deformations of the breast under external perturbations." *Medical Image Analysis*, 6(1): 1-27.

Bachmann, D., S. Bouissou, et al. (2009). "Analysis of massif fracturing during Deep-Seated Gravitational Slope Deformation by physical and numerical modeling." *Geomorphology*, 103(1): 130-135.

Bathe, K. J. and H. Zhang (2009). "A mesh adaptivity procedure for CFD and fluid-structure interactions." *Computers & Structures*, 87(11-12): 604-617.

Baydaa, H. M., X. Ling, et al. (2011). "3D FEM numerical simulation of seismic pile-supported bridge structure." *Applied Sciences, Engineering and Technology*, 3(4): 344-355.

Bechmann, D. and D. Gerber (2003). "Arbitrary shaped deformation with DOGME." *The Visual Computer*, 19(2-3): 175-186.

Bennebroek, K., I. Ernst, et al. (1997). "Design principles of hardware-based phong shading and bump-mapping." *Computers & Graphics*, 21(2): 143-149.

Binotto, B., J. L. D. Comba, et al. (2003). "Real-time volume rendering of time-varying data using a fragment-shader compression approach." *Proceedings of the*

2003 IEEE Symposium on Parallel and Large-Data Visualization and Graphics, IEEE Computer Society: 10.

Blinn, J. F. (1994). "Compositing, part 1: theory." *IEEE Comput. Graph. Appl.*, 14(5): 83-87.

Bordemann, M. (2008). "Deformation quantization: a survey." *Journal of Physics: Conference Series*, 103(1): 012002.

Britz, D., J. Strutwolf, et al. (2011). "Digital simulation of thermal reactions." *Applied Mathematics and Computation*, 218(4): 1280-1290.

Bunel, M. (2005). "Adaptive tessellation of subdivision surfaces with displacement mapping." In *GPU Gems2*: 109-122

Caban, J. J. and P. Rheingans (2008). "Texture-based transfer functions for direct volume rendering." *Visualization and Computer Graphics, IEEE Transactions*, 14(6): 1364-1371.

Cai, W. and G. Sakas (1998). "Maximum intensity projection using splatting in sheared object space." *Computer Graphics Forum*, 17(3): 113-124.

Carmona, R. and B. Froehlich (2011). "Error-controlled real-time cut updates for multi-resolution volume rendering." *Computers & Graphics*, 35(4): 931-944.

Chan, T. and L. Vese (2001). "Active contour without deges." *IEEE Transactions on Image Processing*, 10(2): 266-277.

Chen, M., R. H. Clayton, et al. (2003). "Visualising cardiac anatomy using constructive volume geometry." *Proceedings of the 2nd international conference on Functional imaging and modeling of the heart*: 30-38.

Chen, M., D. Silver, et al. (2003). "Spatial transfer functions: a unified approach to specifying deformation in volume modeling and animation." *Proceedings of the 2003 Eurographics/IEEE TVCG Workshop on Volume graphics*: 35-44.

Chen, M. and J. V. Tucker (2000). "Constructive volume geometry." *Computer Graphics Forum*, 19(4): 281-293.

Cho, K. H., S. J. Cho, et al. (2012). "Dose responses in a normoxic polymethacrylic acid gel dosimeter using optimal CT scanning parameters." *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 675: 112-117.

Choi, S. M., Y.K. Lee, et al. (2004). "Quantitative analysis of gated SPECT images using an efficient physical deformation model." *Computers in Biology and Medicine*, 34(1): 15-33.

Choi, J. J., B. S. Shin, et al. (2000). "Efficient volumetric ray casting for isosurface rendering." *Computers & Graphics*, 24(5): 661-670.

Cimiano, P., A. Hotho, et al. (2004). "Comparing conceptual, divisive and agglomerative clustering for learning taxonomies from text." *Proceedings of the European Conference on Artificial Intelligence*: 435-439.

Comaniciu, D. and P. Meer (2002). "Mean Shift: a robust approach toward feature space analysis." *IEEE TRANSACTIONS on Pattern Analysis and Machine Intelligence*, 24(5): 603-619.

Cormack, A. M. (1979). "Early two dimensional reconstruction and recent topics stemming from it." *Science* 26 September 1980, 209(4464): 1482-1486.

Correa, C. and K. Ma. (2008). "Size-based Transfer Functions: A new volume exploration technique." *IEEE Transactions on Visualization and Computer Graphics*, 14(6): 1380-1387.



Correa, C. D., D. Silver, et al. (2010). "Constrained illustrative volume deformation." *Computers & Graphics*, 34(4): 370-377.

Corrigan, A., F. Camelli, et al. (2011). "Semi-automatic porting of a large-scale fortran CFD code to GPUs." *International Journal for Numerical Methods in Fluids*, 69(2): 314-331.

Courtecuisse, H., H. Jung, et al. (2010). "GPU-based real-time soft tissue deformation with cutting and haptic feedback." *Progress in Biophysics and Molecular Biology*, 103(2-3): 159-168.

De Vaal, M. H., J. Neville, et al. (2011). "Patient-specific prediction of intrinsic mechanical loadings on sub-muscular pectoral pacemaker implants based on an inter-species transfer function." *Journal of Biomechanics*, 44(14): 2525-2531.

Dempster, A. P., N. M. Laird, et al. (1977). "Maximum likelihood from incomplete data via the EM algorithm." *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1): 1-38.

Drager, C. (2005). "A chainmail algorithm for direct volume deformation in virtual endoscopy applications." *Institute of Computer Graphics and Algorithm, Vienna University of Technology*. PHD: 128.

Drebin, R. A., L. Carpenter, et al. (1988). "Volume rendering." *SIGGRAPH Comput. Graph.*, 22(4): 65-74.

Du, X. and T. D. Bui (2008). "A new model for image segmentation." *IEEE signal process.*, 15: 182-185.

Dumas, L., B. Druetz, et al. (2009). "A fully adaptive hybrid optimization of aircraft engine blades." *Journal of Computational and Applied Mathematics*, 232(1): 54-60.

Elsaesser, T. and A. Barker (1990). "Early cinema: space, frame, narrative." BFI Publishing.

Engel, K., M. Hadwiger, et al. (2004). "Course notes 28: real-time volume graphics." Special Interest Group on Graphics and Interactive Techniques.

Engel, K., M. Kraus, et al. (2001). "High-quality pre-integrated volume rendering using hardware-accelerated pixel shading." Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware: 9-16.

Engel, K., F. Oellien, et al. (2000). Client-server-strategien zur visualisierung komplexer struktureigenschaften in digitalen dokumenten der chemie. IT+TI 6/2000 Informationstechnik und Technische Informatik: 17-23.

Fang, G. (2001). "A comprehensive overview of basic clustering algorithms."

Fang, L., Y. Wang, et al. (2002). "Fast maximum intensity projection algorithm using shear warp factorization and reduced resampling." Magnetic Resonance in Medicine, 47(4): 696-700.

Faramarz, F. S., S. Colin, et al. (2007). "Reverse loop subdivision for geometry and textures." IRANIAN Journal of Mathematical Sciences and Informatics, 2(1): 21-37.

Fluck, O., C. Vetter, et al. (2011). "A survey of medical image registration on graphics hardware." Computer Methods and Programs in Biomedicine, 104(3): 45-57.

Forsberg, F., R. Mooser, et al. (2008). "3D micro-scale deformations of wood in bending: Synchrotron radiation muCT data analyzed with digital volume correlation." Journal of Structural Biology, 164(3): 255-262.

Gibson, S. and B. Mirtich (1997). "A survey of deformable modeling in computer graphics." Cambridge.

Gibson, S. F. F. (1997). "3D chainmail a fast algorithm for deforming volumetric objects." Symposium on interactive 3D graphics: 149-154.

Gluchoff, A. (2005). "Pure mathematics applied in early twentieth-century America: The case of T.H. Gronwall, consulting mathematician." *Historia Mathematica*, 32(3): 312-357.

Green, S. (2005). "Volumetric particle shadows." Nvidia Corporation: 1-13.

Hadwiger, M., P. Ljung, et al. (2009). "Advanced illumination techniques for GPU-based volume raycasting." ACM SIGGRAPH 2009 Courses: 1-166.

Hadwiger, M., C. Sigg, et al. (2005). "Real-time ray-casting and advanced shading of discrete isosurfaces." *Computer Graphics Forum*, 24(3): 303-312.

Han, D., J. Keyser, et al. (2009). "A local maximum intensity projection tracing of vasculature in knife-edge scanning microscope volume data." Proceedings of the Sixth IEEE international conference on Symposium on Biomedical Imaging: From Nano to Macro: 1259-1262.

Hernandez, I., C. Mateos, et al. (2009). "Lie theory: applications to problems in mathematical finance and economics." *Applied Mathematics and Computation*, 208(2): 446-452.

Hladuvka, J., A. Konig, et al. (2000). "Curvature-based transfer function for direct volume rendering." Spring Conference on Computer Graphics: 58-65.

Hounsfield, G. N. (1979). "Computed medical imaging." Nobel Lecture 8.

Jaganathan, S., H. V. Tafreshi, et al. (2008). "A realistic approach for modeling permeability of fibrous media: 3-D imaging coupled with CFD simulation." *Chemical Engineering Science*, 63(1): 244-252.

Kanungo, T., D. M. Mount, et al. (2002). "An efficient k-means clustering algorithm: analysis and implementation." *Journal of IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE*, 24(7): 881-892.

Kass, M., A. Witkin, et al. (1988). "Snakes: active contour models." *International Journal of Computer Vision*, 1(4): 321-331.

Kaul, U. K. (2010). "Three-dimensional elliptic grid generation with fully automatic boundary constraints." *Journal of Computational Physics*, 229(17): 5966-5979.

Keeve, E., S. Girod, et al. (1996). "Anatomy-based facial tissue modeling using the finite element method." *Visualization '96. Proceedings*: 21-28

Kniss, J., G. Kindlmann, et al. (2002). "Multidimensional transfer functions for interactive volume rendering." *IEEE Transactions on Visualization and Computer Graphics*, 8(3): 270-285.

Kniss, J., S. Premoze, et al. (2003). "A model for volume lighting and modeling." *IEEE Transactions on Visualization and Computer Graphics*, 9(2): 150-162.

Kobayashi, Y., A. Onishi, et al. (2010). "Development of an integrated needle insertion system with image guidance and deformation simulation." *Computerized Medical Imaging and Graphics*, 34(1): 9-18.

Konig, A. H. and W. M. Groller (2001). "Mastering transfer function specification by using volumepro technology." *Spring Conference on Computer Graphics 2001*, 17: 279-286.

Kotava, N., A. Knoll, et al. (2012). "Morse-male decomposition of multivariate transfer function space for separably-sampled volume rendering." *Computer Aided Geometric Design*.

Kruger, J. and R. Westermann (2003). "Acceleration techniques for GPU-based volume rendering." Proceedings of the 14th IEEE Visualization 2003 (VIS'03), IEEE Computer Society: 38

Kurzion, Y. and R. Yagel (1997). "Interactive space deformation with hardware-assisted rendering." IEEE Comput. Graph. Appl., 17(5): 66-77.

Lefebvre, S., S. Hornus, et al. (2003). "Octree textures on the GPU." In GPU Gems2: 595-613.

Lefohn, A., J. Kniss, et al. (2003) "Implementing efficient parallel data structure on GPUs." In GPU Gems2: 521-545.

Leu, A. and M. Chen (1999). "Modelling and rendering graphics scenes composed of multiple volumetric datasets." Computer Graphics Forum, 18(2): 159-171.

Levoy, M. (1988). "Display of Surfaces from Volume Data." IEEE Comput. Graph. Appl., 8(3): 29-37.

Levoy, P. L. a. M. (1994). "Fast volume rendering using a shear-warp factorization of the viewing transformation." SIGGRAPH '94 Proceedings of the 21st annual conference on Computer graphics and interactive techniques: 451-458.

Lewis, J. P., M. Cordner, et al. (2000). "Pose space deformation: a unified approach to shape interpolation and skeleton-driven deformation." Proceedings of the 27th annual conference on Computer graphics and interactive techniques, ACM Press/Addison-Wesley Publishing Co: 165-172.

Li, Y. and K. Brodlie (2003). "Soft object modelling with generalised chainmail - extending the boundaries of web-based graphics " Computer Graphics Forum, 22(4): 717-727.

Lorensen, W. E. and H. E. Cline (1987). "Marching cubes: a high resolution 3D surface construction algorithm." *ACM SIGGRAPH Computer Graphics*, 21(4): 163-169.

Luebke, D. (2001). "A developer's survey of polygonal simplification algorithms." *Computer Graphics*, 21(3): 24-35.

MacKay, D. (2003). "Chapter 20. An example inference task: clustering." *Information Theory, Inference and Learning Algorithms*: 284-292.

Mark, W. R., G. Steven, et al. (2003). "Cg: a system for programming graphics hardware in a C-like language." *Proceeding of the conference on ACM SIGGRAPH '03*: 896-907.

Maruya, J., K. Nishimaki, et al. (2010). "Hyperperfusion syndrome after neck clipping of a ruptured aneurysm on a dolichoectatic middle cerebral artery." *Journal of Stroke and Cerebrovascular Diseases*, 20(3): 260-263.

Matsuura, Y., S. Oharu, et al. (2003). "Mathematical approaches to bone reformation phenomena and numerical simulations." *Journal of Computational and Applied Mathematics*, 158(1): 107-119.

Mille, J. (2009). "Narrow band region-based active contours and surfaces for 2D and 3D segmentation." *Computer Vision and Image Understanding*, 113(9): 946-965.

Nakao, M. and K. Minato (2010). "Physics-based interactive volume manipulation for sharing surgical process." *Information Technology in Biomedicine, IEEE Transactions on*, 14(3): 809-816.

Natsupakpong, S. and M. C. Cavusoglu (2010). "Determination of elasticity parameters in lumped element (mass-spring) models of deformable objects." *Graphical Models*, 72(6): 61-73.

Nealen, A., M. Müller, et al. (2006). "Physically based deformable models in computer graphics." *Computer Graphics Forum*, 25(4): 809-836.

Nicodemus, F. E. (1965). "Directional reflectance and emissivity of an opaque surface." *Applied Optics*, 4(7): 767-773.

Nienhuys, H. W. and A. F. v. d. Stappen (2000). "Combining finite element deformation with cutting for surgery simulations." *Eurographics 2000*: 143-152.

Nishita, T., Y. Dobashi, et al. (1996). "Display of clouds taking into account multiple anisotropic scattering and sky light." *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*: 379-386.

Oh, K. M. and K. H. Park (1995). "A vertex merging algorithm for extracting a variable-resolution isosurface from volume data." *Systems, Man and Cybernetics, 1995. Intelligent Systems for the 21st Century*, 4: 3543-3548.

Olabarriaga, S. D. and A. W. M. Smeulders (2001). "Interaction in the segmentation of medical images: A survey." *Medical Image Analysis*, 5(2): 127-142.

Park, S. and C. Bajaj (2007). "Feature selection of 3D volume data through multi-dimensional transfer functions." *Pattern Recognition Letters*, 28(3): 367-374.

Patete, P., M. I. Iacono, et al. (2012). "A multi-tissue mass-spring model for computer assisted breast surgery." *Medical Engineering & Physics*.

Petersch, B., M. Hadwiger, et al. (2005). "Real time computation and temporal coherence of opacity transfer functions for direct volume rendering of ultrasound data." *Computerized Medical Imaging and Graphics*, 29(1): 53-63.

Pfister, H., B. Lorensen, et al. (2001). "The transfer function bake-off." *Computer Graphics and Applications, IEEE*, 21(3): 16-22.

Pinto, F. d. M. and C. M. D. S. Freitas (2006). "Two-level interaction transfer function design combining boundary emphasis, manual specification and evolutive generation." *Computer Graphics and Image Processing*, 2006. SIBGRAPI '06. 19th Brazilian Symposium on: 281-288.

Pinto, F. d. M. and C. M. D. S. Freitas (2008). "Volume visualization and exploration through flexible transfer function design." *Computers & Graphics*, 32(5): 540-549.

Provot, X. (1995). "Deformation constraints in a mass-spring model to describe rigid cloth behavior." *Graphics Interface*: 141-155.

Qian, Z., P. H. Joshi, et al. (2011). "Relationship between chest lateral width, tube current, image noise, and radiation exposure associated with coronary artery calcium scanning on 320-detector row CT." *Journal of Cardiovascular Computed Tomography*, 5(4): 231-239.

Radon, J. (1986). "On the determination of functions from their integral values along certain manifolds." *Medical Imaging, IEEE Transactions on*, 5(4): 170-176.

Rajon, D. A. and W. E. Bolch (2003). "Marching cube algorithm: review and trilinear interpolation adaptation for image-based dosimetric models." *Computerized Medical Imaging and Graphics*, 27(5): 411-435.

Ray, H., H. Pfister, et al. (1999). "Ray casting architectures for volume visualization." *IEEE Transactions on Visualization and Computer Graphics*, 5(3): 210-223.

Reader, P. and M. P. Meyer (2000). *Restoration of motion picture film*, Butterworth-Heinemann: 23-24.

Rieder, C., S. Palmer, et al. (2011). "A shader framework for rapid prototyping of GPU-based volume rendering." *Computer Graphics Forum*, 30(3): 1031-1040.



Roy, S. and G. A. Ahmed (2011). "Monte carlo simulation of light scattering from size distributed sub-micron spherical CdS particles in a volume element." *Optik - International Journal for Light and Electron Optics*, 122(11): 1000-1004.

Rui, X. and D. Wunsch II (2005). "Survey of clustering algorithms." *Neural Networks, IEEE Transactions on*, 16(3): 645-678.

Sakamoto, N., T. Kawamura, et al. (2010). "Improvement of particle-based volume rendering for visualizing irregular volume data sets." *Computers & Graphics*, 34(1): 34-42.

Sato, N., N. Shiraga, et al. (1998). "Local maximum intensity projection." *Journal of Computer Assisted Tomography*, 22(6): 912-917.

Sauvage, B., S. Hahmann, et al. (2008). "Detailed preserving deformation of B-spline surface with volume constraint." *Journal of Computer Aided Geometric Design*, 25(8): 678-696.

Schroeder, W. J., J. A. Zarge, et al. (1992). "Decimation of triangle meshes." *SIGGRAPH Comput. Graph*, 26(2): 65-70.

Stankevich, D., Y. Shkuratov, et al. (2003). "Computer simulations for multiple scattering of light rays in systems of opaque particles." *Journal of Quantitative Spectroscopy and Radiative Transfer*, 76(1): 1-16.

Strengert, M., M. Magallán, et al. (2005). "Large volume visualization of compressed time-dependent datasets on GPU clusters." *Journal of Parallel Computing*, 31(2): 205-219.

Sugihara, M., B. Wyvill, et al. (2010). "WarpCurves: a tool for explicit manipulation of implicit surfaces." *Computers & Graphics*, 34(3): 282-291.

Szekely, G. J. and M. L. Rizzo (2005). "Hierarchical clustering via joint between-within distances: extending ward's minimum variance method." *Journal of Classification*, 22(2): 151-183.

Tappenbeck, A., B. Preim, et al. (2006). "Distance-based transfer function design: specification." *Methods and Applications. Simulation and visualization*: 259-274.

Tatarchuk, N., J. Shopf, et al. (2008). "Advanced interactive medical visualization on the GPU." *Journal of Parallel and Distributed Computing*, 68(10): 1319-1328.

Tejada, E. and T. Ertl (2005). "Large steps in GPU-based deformable bodies simulation." *Simulation Modelling Practice and Theory*, 13(8): 703-715.

Vigneron, L. M., R. C. Boman, et al. (2008). "Enhanced FEM-based modeling of brain shift deformation in image-guided neurosurgery." *Journal of Computational and Applied Mathematics*, 234(7): 2046-2053.

Wallis, J. W., T. R. Miller, et al. (1989). "Three-dimensional display in nuclear medicine." *Medical Imaging, IEEE Transactions on*, 8(4): 297-230.

Weiskopf, D., K. Engel, et al. (2003). "Interactive clipping techniques for texture-based volume visualization and volume shading." *Visualization and Computer Graphics, IEEE Transactions on*, 9(3): 298-312.

Weiskopf, D., M. Hopf, et al. (2001). "Hardware-accelerated visualization of time-varying 2D and 3D vector fields by texture advection via programmable per-pixel operations." *Proceedings of the Vision Modeling and Visualization Conference '01*: 439-446.

Weng, T. L., S. J. Lin, et al. (2002). "Voxel-based texture mapping for medical data." *Computerized Medical Imaging and Graphics*, 26(6): 445-452.

Westermann, R. and T. Ertl (1998). "Efficiently using graphics hardware in volume rendering applications." Proceedings of the 25th annual conference on Computer graphics and interactive techniques, ACM: 169-177.

Westermann, R. and B. Sevenich (2001). "Accelerated volume ray-casting using texture mapping." Proceedings of the conference on Visualization '01. IEEE Computer Society: 271-278.

Wittenbrink, C. M., T. Malzbender, et al. (1998). "Opacity-weighted color interpolation, for volume sampling." Proceedings of the 1998 IEEE symposium on Volume visualization: 135-142.

Wu, Y., V. Bhatia, et al. (2003). "Shear-image order ray casting volume rendering." Proceedings of the 2003 symposium on Interactive 3D graphics: 152-162.

Xu, C. and J. L. Prince (1998). "Shankes, shapes and gradient vector flow." IEEE Trans. Image process, 17(3): 359-369.

Yuan, Z. Y., Y. Y. Zhang, et al. (2010). "Real-time Simulation for 3D Tissue Deformation with CUDA Based GPU Computing." Journal of Convergence Information Technology, 4(4): 109-119.

Zhang, Y., D. Zhu, et al. (2011). "Importance sampling for volumetric illumination of flames." Computers & Graphics, 35(2): 312-319.

Zhou, N., T. Matsumoto, et al. (2010). "Pore-scale visualization of gas trapping in porous media by X-ray CT scanning." Flow Measurement and Instrumentation, 21(3): 262-267.

Zhou, L., M. Schott, et al. (2012). "Transfer function combinations." Computers & Graphics.

## Appendix A

Because of the benefits from applying GPU, the traditional computing process is evolving from “central processing” on the CPU to “co-processing” on the CPU and GPU. In order to support this new computing paradigm, Nvidia built up CUDA, a parallel computing architecture, to facilitate the heterogeneous computing with CPU and GPU. As a subset of C with dedicated extensions, CUDA is a programming model that enables dramatic increases in computing performance by harnessing the power of the GPU.

CUDA serves as a technical partition of an object task into a certain amount of subtasks and assigns them to be accessed and processed in the manner of threads in GPU. Although both processing mechanisms are based on the thread, the advantage of GPU's threads over CPU's contains: very little creation overhead (i.e. GPU can own more threads than CPU) and indexable ID for rapid switching (i.e. offering faster thread management). Besides, as a special feature of GPU, the inherent threads are available for synchronisation which can overcome the problems of limited memory bandwidth and redundant computation. Consequently, utilizing GPU to process the computation works can save processing time and increase system efficiency.

Inside this cooperative processing, CPU and GPU are two distinct processors and require two different types of memory: host memory (only available in CPU but not accessible for GPU's threads) and device memory (dedicated storage unit in GPU).

For the communication between them, there constrainedly exists a series of memory allocation operations which standardise the data transfer tasks.

As a powerful tool of GPU-based acceleration, CUDA has been becoming popular in various categories of recent graphics cards, representing a significant installed base for different application developers or researchers.

## **Appendix B**

Consumer grade is for describing a gear manufactured for general users who want ordinary application with an acceptable price of it. Different from professionals, the consumer grade gear is normally designed to suffice for various user demands without any special service restrictions.

As an example of consumer grade gear, the consumer grade computer owns various features: low prices, low quality, popular, convenient maintenance and management, etc. Although it cannot achieve the same high-performance as the professional gear (the workstation), the consumer grade computer has been becoming more powerful through benefitting from the hardware development the design and application of new materials in the past two decades.