

國立交通大學

資訊科學與工程研究所

碩士論文

傅立葉立體資料描繪法之研究與轉換函數
之設計



The study of Fourier Volume Rendering and Transfer
Function design

研究生：鄭昌杰

指導教授：荊宇泰 博士

中華民國九十六年四月

傅立葉立體資料描繪法之研究與轉換函數之設計
The study of Fourier Volume Rendering and Transfer Function design

研究生：鄭昌杰

Student : Chang-Chieh Cheng

指導教授：荊宇泰

Advisor : Yu-Tai Ching

國立交通大學
資訊科學與工程研究所
碩士論文



A Thesis
Submitted to Institute of Computer Science and Engineering
College of Computer Science
National Chiao Tung University
in partial Fulfillment of the Requirements
for the Degree of
Master
in
Computer Science

April 2007

Hsinchu, Taiwan, Republic of China

中華民國九十六年四月

傅立葉立體資料描繪法之研究與轉換函數之設計

學生：鄭昌杰

指導教授：荊宇泰 博士

國立交通大學資訊科學與工程研究所

摘 要

立體資料描繪法在醫學診療及生物科技上是很常用的科學視覺化技術，但是傳統的立體資料描繪法所需的時間複雜度太大以致無法達成即時的顯像。傅立葉立體資料描繪法具有快速的繪圖時間，它對立體資料的大小為 N^3 的時間複雜度是 $O(N^2 \log N)$ ，可以滿足即時顯像的需求。但是它會有頻域空間再取樣的問題，導致重疊影像的失真現象，因此濾波器的設計對傅立葉立體資料描繪法是非常重要的研究主題。另外，如何讓傅立葉立體資料描繪法達成分類顯像的效果也是本論文的重點，我們利用了傅立葉轉換具有線性組合之特性，設計了兩種分類顯像的方式，一是預先影像處理，另一是貝茲曲線轉換函數。這兩種方式可以準確地進行資料分類，而且可以很容易的實作在目前一般個人電腦上使用的顯示卡上。

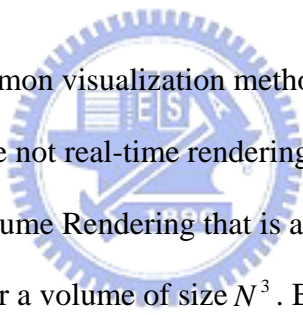
The study of Fourier Volume Rendering and Transfer Function design

Student: Chang-Chieh Cheng

Advisor: Dr. Yu-Tai Ching

Institute of Computer Science and Engineering
National Chiao Tung University

ABSTRACT



Volume rendering are common visualization methods for medical science and biological science. Traditional methods are not real-time rendering because their high complexity rendering time. The Fourier Volume Rendering that is a real-time method has computational complexity of $O(N^2 \log N)$ for a volume of size N^3 . But resampling in the frequency domain will causes replicas occur in the spatial domain if we do not use any reconstruction filter. Therefore, the design of an efficient filter is an important study issue for this paper. The classification for Fourier Volume Rendering is the main subject in this paper, too. We used the linear combination property of the Fourier transform to design two classification methods, the image preprocessing and the Bézier curve transfer function, both are accurate classification methods on rendering stage and easy to implement using graphics card in general PC.

誌 謝

本論文能得以完成，首先要感謝我的指導教授荊宇泰教授，在這兩年內由於他的專業知識傳授與細心的指導，才能讓我順利完成此論文。同時也感謝口試委員莊榮宏教授與楊傳凱教授指正論文的謬誤處及提供許多專業上的建議，使本論文能更加正確完整。感謝醫學影像實驗室的全體成員，在這兩年給予許多專業知識上及精神上的支持。

最後要感謝我的家人，感謝他們在這兩年內給我的包容與支持，讓我得以安心的完成我的碩士論文，謝謝大家。



目 錄

中文提要	i
英文提要	ii
誌謝	iii
目錄	iv
表目錄	vi
圖目錄	vii
一、	緒論	1
1.1	研究動機與目標	1
1.2	立體資料描繪法	2
1.3	分類顯像與轉換函數	5
1.4	快速傅立葉轉換	7
1.5	繪圖處理器泛用運算	7
1.6	論文架構	8
二、	傅立葉立體資料描繪法	9
2.1	傅立葉截面理論	9
2.2	頻域取樣問題	12
2.2.1	脈衝訊號	12
2.2.2	頻域空間取樣	13
2.2.3	再取樣問題	15
2.3	濾波器設計	16
2.3.1	時域空間延伸	16
2.3.2	sinc 濾波器	17
2.3.3	三線性濾波器	18
2.3.4	三次曲線濾波器	19

2.3.5	時域空間衰減補償	20
2.4	著色模型	21
2.4.1	FVR 著色模型	21
2.4.2	遠近效果	22
2.4.3	光照模型	23
三、	轉換函數的設計	25
3.1	預先影像處理	25
3.2	貝茲曲線轉換函數	27
四、	實作與結果	30
4.1	實驗環境	30
4.2	以 GPU 實作傅立葉立體資料描繪法	32
4.2.1	實作 FVR 演算法	32
4.2.2	濾波器的實作	35
4.2.3	以 GPU 實作 FFT	42
4.3	貝茲曲線轉換函數的實作	44
4.3.1	使用者介面設計	45
4.3.2	分類結果	47
五、	結論與未來展望	51
參考文獻	52

表目錄

表 4.1	不使用任何濾波器	33
表 4.2	使用三線性濾波器	36
表 4.3	使用三次曲線濾波器	39
表 4.4	三線性濾波器與時域空間延伸	41
表 4.5	FFT 的比較	44
表 4.6	貝茲曲線轉換函數	47



圖目錄

圖 1.1	人類頭部的 CT 影像	2
圖 1.2	Marching cubes	4
圖 1.3	Texture-based volume rendering	5
圖 1.4	Texture-Based Volume Rendering 與 2 維的轉換函數 ...	6
圖 2.1	2 維函數的投影	9
圖 2.2	FVR 演算法	11
圖 2.3	脈衝訊號	12
圖 2.4	單位週期脈衝訊號	13
圖 2.5	$III(Dt)$ 的時域與頻域	13
圖 2.6	頻域空間取樣過程	14
圖 2.7	取樣頻率不足下的失真	15
圖 2.8	再取樣問題	16
圖 2.9	FVR 的失真現象	16
圖 2.10	sinc 函數與 Π 函數	17
圖 2.11	線性內插	18
圖 2.12	三線性濾波器	19
圖 2.13	Catmull-Rom 曲線	20
圖 2.14	時域空間的衰減	20
圖 2.15	線性深度衰減	22
圖 2.16	遠近效果	23
圖 2.17	光照效果	24
圖 3.1	Canny edge detection	25
圖 3.2	立體資料的預先分類	26
圖 3.3	以影像預先處理的分類顯像	27
圖 3.4	轉換函數	28
圖 3.5	貝茲曲線	28

圖 4.1	FVR 實作步驟	33
圖 4.2	重疊影像的失真	34
圖 4.3	視角為 90 度的倍角	35
圖 4.4	使用三線性濾波器	37
圖 4.5	三線性濾波器時域空間衰減補償	38
圖 4.6	三次曲線內插	39
圖 4.7	使用三次曲線濾波器	40
圖 4.8	三次曲線濾波器時域空間衰減補償	40
圖 4.9	三線性濾波器與時域空間延伸	42
圖 4.10	DIF FFT	43
圖 4.11	轉換函數使用者介面	46
圖 4.12	FVR 應用程式	46
圖 4.13	頭部骨骼部份	48
圖 4.14	頭部肌肉部份	48
圖 4.15	頭部液體部份	48
圖 4.16	盆栽樹枝部份	49
圖 4.17	盆栽樹葉部份	49
圖 4.18	胸腔骨骼部份	50
圖 4.19	胸腔肌肉部份	50
圖 4.20	胸腔脂肪部份	50



一、緒 論

1.1 研究動機與目標

傅立葉立體資料描繪法(Fourier Volume Rendering, 以下簡稱 FVR), 是一種直接立體資料描繪法(Direct volume rendering)。它是以訊號處理中常用的傅立葉轉換(Fourier transform), 來產生對立體資料以任一角度所觀測的投影影像。傳統的直接立體資料描繪法在每產生一張畫面的所需時間複雜度是 $O(N^3)$, FVR 則可以透過 2D 的快速傅立葉轉換(Fast Fourier Transform, FFT)能夠在 $O(N^2 \log N)$ 的時間複雜度產生一張畫面。因此 FVR 演算法能以更快的速度讓使用者感覺到即時且互動的立體資料顯像效果。而且 FVR 所產生的影像, 是類似 X 光片的影像, 這種影像會保留許多原有資訊, 不像 Marching cubes [1]、Texture-based volume rendering [2]等其他常見的立體資料描繪法, 會把許多原有資訊去除。因此 FVR 對醫療人員的診斷上以及生物科技的研究是有一定的幫助。

目前市面上用於一般個人電腦的 3D 顯示卡都有內嵌繪圖處理器(Graphics Processing Unit, GPU), 它除了對圖形的運算能力十分強大之外, 對大量的浮點數運算能力也非常出色。而且目前 GPU 已經發展為可程式化(Programmable), 我們可以利用 GPU 進行非繪圖相關的運算, 這種技術稱為繪圖處理器泛用運算(General-Purpose Computation on GPUs, GPGPU)。因此, FVR 演算法很適合以市面上的 3D 顯示卡來實作, 除了讓顯像速度能夠再加快, 也可以加強 FVR 的顯像品質, 以及改善 FVR 的缺點。

另外, 使用者可能只想觀看有興趣的組織所產生的立體資料描繪影像, 因此分類顯像(Classification)在生物醫學影像分析是很重要的一項需求。目前全世界對於 FVR 的分類顯像之研究並沒有很多, 也尚未提出一個很有效且讓使用者感覺

便利的方法。因此，若我們能夠設計一個簡易且有效的轉換函數(Transfer Function)，就能夠讓使用者很容易地進行 FVR 的分類顯像了。

本篇論文主要是先以目前市面上用於一般個人電腦的 3D 顯示卡來實作 FVR 演算法，也就是將 FVR 演算法中較為龐大且複雜的運算工作交由 GPU 來執行，其中包含了快速傅立葉轉換、3D 頻譜資料的取樣、3D 濾波器...等。最後我們設計了一個以貝茲曲線(Bézier curve)為基礎的多項式轉換函數，來達成有效且便利的分類顯像功能。

1.2 立體資料描繪法

立體資料(Volume data)是一種三維的資料結構，常用來描述真實世界的物體之內外結構以及自然現象，如生物組織、大氣現象等。而通常它是由許多的二維平面資料所堆疊起來，如電腦斷層影像(Computed Tomography, CT)、核磁共振造影(Magnetic Resonance Imaging, MRI)等，如圖 1.1 所示為人類頭部的 CT 影像。

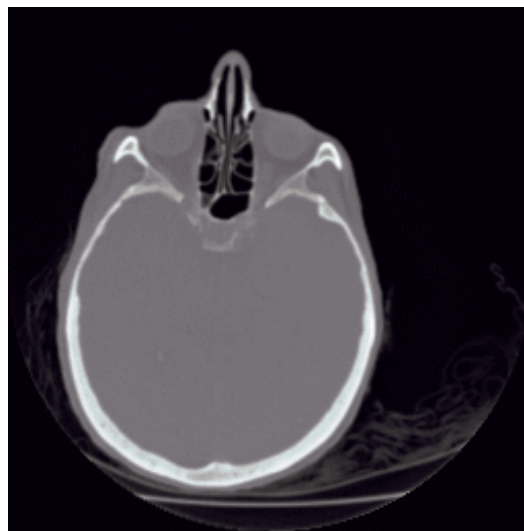


圖 1.1 人類頭部的 CT 影像

立體資料的結構中，若是以尺寸為相同大小的單位元素所組成，如與整個立體資料的尺寸等比例之立方體，我們稱這樣的立體資料為正規立體資料(Regular

volume data)，且稱每一個單位元素為 voxel。在一般的立體資料中，voxel 是一個純量值，它代表了物體中某一組織的強度大小，如溫度、密度或是影像中的灰階值。而在生物科技及醫學影像的應用上，所處理的資料形式多為正規立體資料，也就是每一個 voxel 是一個立方體，而本篇論文所使用的立體資料都只考慮正規立體資料。

立體資料描繪法(Volume rendering)是將三維的立體資料集合以二維的投影方式轉換成平面影像的一個過程，可以應用在科學計算視覺化或是生物醫學影像的視覺化。立體資料描繪法的方法大致分為表面立體資料描繪法(Surface volume rendering)以及直接立體資料描繪法(Direct volume rendering)。表面立體資料描繪法是以幾何模型結構來表現出立體資料中某一個特定的表面(Isosurface)，也就是使用者要先知道感興趣的組織的 voxel 強度值為多少，然後以此特定的強度值(Isovalue)來找出特定的表面。關於表面立體資料描繪法最具代表性的是 Lorensen [1] 所提出的 Marching cubes，他建立了 15 種基本的幾何樣式來拼湊出各種情況的 Isosurface。但此方法仍會有一些模稜兩可(Ambiguity)的情況發生，Lin [3] 提出以三線性內插(Tri-linear interpolation) [4] 來找出 isovalue 的鞍形點，以解決 Marching cubes 的模稜兩可情況。圖 1.2 是將資料大小為 $x \times y \times z = 64 \times 64 \times 150$ 的 MRI 立體資料以 Marching cubes 所描繪出。

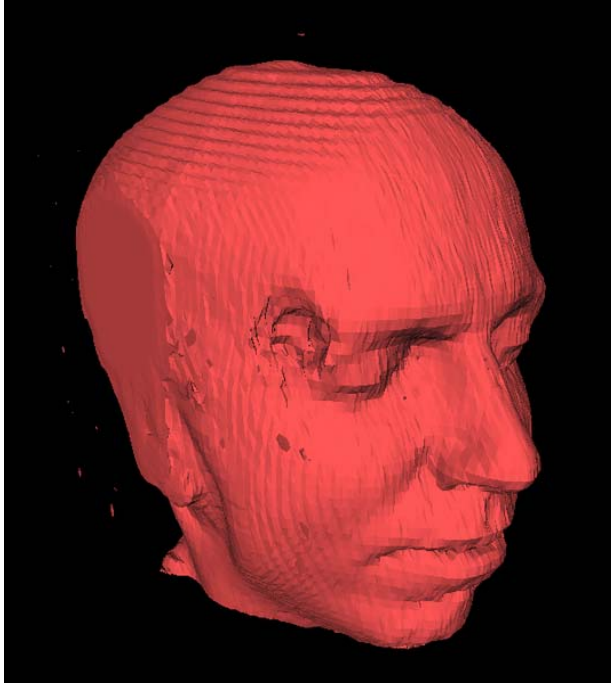


圖 1.2 Marching cubes

資料來源：Wikipedia

<http://commons.wikimedia.org/wiki/Image:Marchingcubes-head.png>

直接立體資料描繪法是模擬 X 光穿透物體衰減的物理現象，所以我們先決定一個視點，然後對一個立體資料沿著此視點方向進行積分運算，如此就會產生此視點方向的投影影像。直接立體資料描繪法的相關研究大多是以實作 Max [5] 提出的立體資料描繪積分式為主，然而近幾年由於繪圖硬體的進步，因而發展出適用在繪圖晶片以貼圖為基礎的立體資料描繪法(Texture-based volume rendering) [2]，如圖 1.3 所示。由於直接立體資料描繪法所產生的影像是類似 X 光片的影像。相較於表面立體資料描繪法所產生的影像，會有感覺比較模糊不清的情況，這是因為積分的關係而有組織重疊的現象。但相對沒有經過找 isosurface 的二值化處理，所以保留的資訊反而可能較多。



CT W:256, H:256, D:256

圖 1.3 Texture-based volume rendering

而 FVR 是一種直接立體資料描繪法，最早是由 Dunne [6] 所提出，它是以傅立葉轉換為基礎，先將立體資料轉換成頻譜資料(Spectrum)，然後在頻域空間(Frequency domain)進行取樣(Sampling)，最後再以反傅立葉轉換取得投影影像。其中在頻域空間進行取樣時會有失真問題，Malzbender [7] 設計了許多濾波器來解決這個問題。另外為了讓 FVR 所產生的影像具有遠近及光影效果，Levoy 提出如何對立體資料計算表面的法向量[8]以及設計了三種著色模型(Shading model) [9]，之後 Totsuka [10] 以迴旋積分(Convolution)來改良這三種著色模型。Levoy 所提出的著色模型對我們的研究有非常大的幫助，我們所設計的轉換函數將會應用到這種概念。

1.3 分類顯像與轉換函數

分類顯像(Classification)在直接立體資料描繪法是很重要的一項技術，所謂分類顯像意指將立體資料中每一 voxel 的強度值以某一種顏色來呈現，舉例來說明，在 CT 影像中，肌肉組織的灰階值可能介於 60 至 100 之間，我們就把 60 至 100 定義為紅色，骨骼部份的灰階值可能介於 200 至 255 之間，我們就把 200 至 255 定義

為白色。如此一來，所產生的立體資料描繪影像就可以很清楚的分辨何者為肌肉，何者為骨骼。而這種一個強度值對應一個顏色值的轉換過程，我們稱之為轉換函數 (Transfer function)。在 Texture-based volume rendering 中，Engel [11]提出了一個 2 維的轉換函數，並事先以積分方式將此轉換函數建立好，成為一張 2 維的表格，然後在產生投影影像時，將前後兩張平行於投影平面的截面(slice)的 voxel 值做為轉換函數的參數以取得顏色值。此方法之後有 Lum [12]加以改良。圖 1.4 所示為 Texture-Based Volume Rendering 以 2 維的轉換函數呈現，其中右上部分為使用者定義的強度值(橫軸)與對應顏色值(縱軸)的定義，右下部分為 2 維的轉換函數表格，左半部即為投影影像。

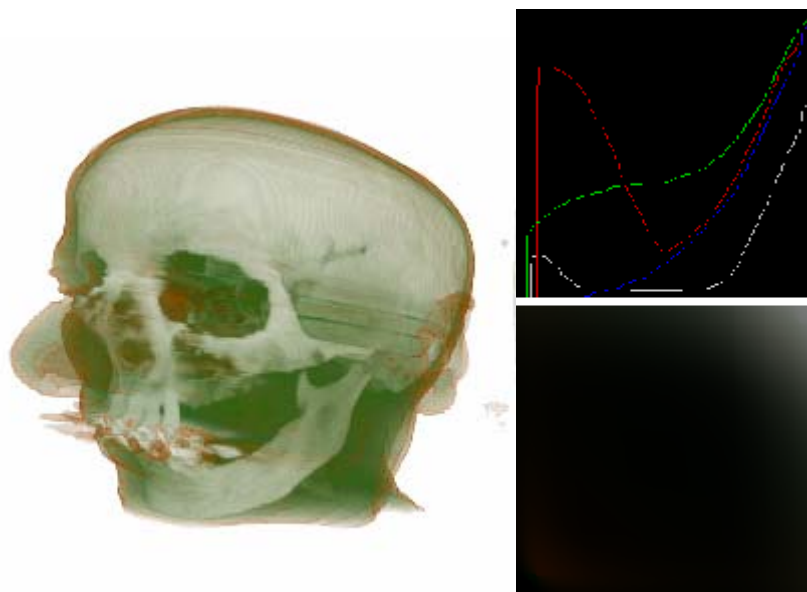


圖 1.4 Texture-Based Volume Rendering 與 2 維的轉換函數

轉換函數在 FVR 中並不容易設計，這是因為 FVR 有許多先天上的限制，第一，FVR 是在頻域空間運算，將立體資料乘上轉換函數，即兩函數在時域空間 (Time domain)相乘，等同兩函數在頻域空間進行迴旋積分，這對運算時間及空間是蠻大的負擔，無法有即時顯像的效果。第二，FVR 的產出是類似 X 光的灰階影像，若我們希望能透過轉換函數產生彩色的 RGB 影像，資料量將會增加三倍。

因此，到目前為止對於 FVR 的轉換函數之相關研究並沒有很多，只有 Nagy [13] 以步進函數(Step function)並配合 Levoy 所提出的著色模型來設計轉換函數。而我們所設計的轉換函數也是類似此方法。

1.4 快速傅立葉轉換

1965 年 Cooley 及 Tukey [14]提出 FFT(Fast Fourier Transform)演算法讓計算 DFT(Discrete Fourier Transform)的時間複雜度由 $O(N^2)$ 提升為 $O(N \log N)$ ，之後陸續有各種的實作方式，包含了適合於平行處理器運作的 Butterfly FFT [15]以及 DIF FFT(Decimation-in-frequency FFT) [16]。目前公認執行速度最快的 FFT 實作方式是由麻省理工學院兩位學者 Frigo 及 Johnson [17]所發明的 FFTW(FFT in the West)。它的原理大致是這樣，由於 FFT 是一種 Divide-and-conquer 的演算法，這種演算法的特性就是將所輸入的 N 筆資料平分成幾組，再對這幾組資料繼續平分下去，直到每組資料只會含最少的容許數量(比如一筆)，然後再一層一層合併起來(merge)。這時，每次要分成幾組就值得分析了。根據實驗結果，輸入的資料量不同，若適當給予不同分組，其執行效能會有明顯改善。換句話說，若我們對一個 Divide-and-conquer 的演算法，對每種輸入資料量都以固定的數量來分組，其執行效能將不會是最佳的。FFTW 是先以 Dynamic-programming 方式來找出分組數量的最佳解，然後再產生適合此解的一套 FFT 的程式碼，這就是一套最佳化的程式碼了。因此，在所有的 FFT 實作方式中，我們選擇 FFTW 為主要的傅立葉轉換的工具。

1.5 繪圖處理器泛用運算

繪圖處理器泛用運算(General-Purpose Computation on GPUs, GPGPU)意指以繪圖處理器(Graphics Processing Unit, GPU)進行各種類型的運算工作，除了原本的圖形運算外，還包含了物理運算[18]、線性代數運算[19]等。這是因為目前的 GPU 提

供了兩種可程化的功能：頂點著色器(Vertex shader)及像素著色器(Pixel shader)，來讓使用者在不改變原有的繪圖管線(Rendering pipeline)來變更其中的兩個程序：頂點的座標轉換及像素顏色的決定。其中像素著色器具有 SIMD(Single Instruction Multiple Data)的平行運算能力，如 ATI 最新的高階顯示卡 X1950 就提供了 48 個像素處理器來進行像素運算，nVidia 最新的 GeForce 8800 GTX 的 Stream processor 更高達了 128 個，而且這些處理器可以針對 IEEE 754 32-bit 浮點數來進行運算，使得在精密的數學運算下誤差不至於過高。因此我們若有一次需處理大量資料的情況，那麼交由 GPU 的像素著色器來進行運算是非常合適的。

傅立葉轉換十分適合以 GPU 來執行，Moreland [20]就提出一個將快速傅立葉轉換以 GPU 來執行的方法，而且利用頻譜的週期特性來節省一半的記憶體空間。之後 Jansen [16]對 DIF FFT 演算法做一個運算程序上的改變，將每個 Butterfly operator 的兩個輸出分開，使得加法的運算跟減法的運算可以各別用不同的程式碼來執行，而不是以一個程式碼來去判斷何種輸入要進行加、減法，如此更可以符合 GPU 的平行運算環境，以加速運算的時間。關於這部分的實作內容，我們將在第四章會做更詳細的介紹。然而 FVR 會用到一次的 3 維傅立葉轉換，以及每次產生投影影像所需要的 2D 反傅立葉轉換，尤其以 2D 反傅立葉轉換運算次數最頻繁，因此我們可以將它交由 GPU 來執行，以加速顯像的時間。

1.6 論文架構

本篇論文共分為六個章節：第一章為緒論，研究動機、目的以及相關研究工作。第二章為介紹 FVR 的理論、頻域空間的取樣問題以及著色模型；第三章為轉換函數的設計，介紹什麼是轉換函數，以及我們提出的兩個方法來進行分類顯像：預先影像處理、貝茲曲線轉換函數；第四章為實作與結果，將介紹如何以 GPU 來實作 FVR 演算法，以及實作分類顯像；第五章為結論與未來展望。

二、傅立葉立體資料描繪法

2.1 傅立葉截面理論

傅立葉截面理論(Fourier Slice Theorem)是由 Bracewell [21]在 1956 年所提出，亦稱傅立葉投影理論(Fourier Projection Theorem)。此理論是 FVR 的核心理論，它的大致意義是這樣：若對一個具有 n 個維度的時域函數 $f(x_0, x_1, \dots, x_{n-1})$ 進行傅立葉轉換，則可得到相同維度的頻域函數 $F(u_0, u_1, \dots, u_{n-1})$ 。若我們固定 F 的其中一個維度 u_k ，並通過原點從 F 截取出一個 $n-1$ 維度的頻域函數 F' ，之後再對 F' 進行反傅立葉轉換，則可以得到也是 $n-1$ 維度的時域函數 f' ，那麼 f' 會是沿著 x_k 軸的積分，也就是延著 x_k 軸的投影。如圖 2.1 所示，2 維的函數 $f(x, y)$ 經過線性轉換旋轉了 θ 角後會得到 $f'(x', y')$ ，然後再對 $f'(x', y')$ 沿著 x' 軸積分即得到 1 維的投影結果。

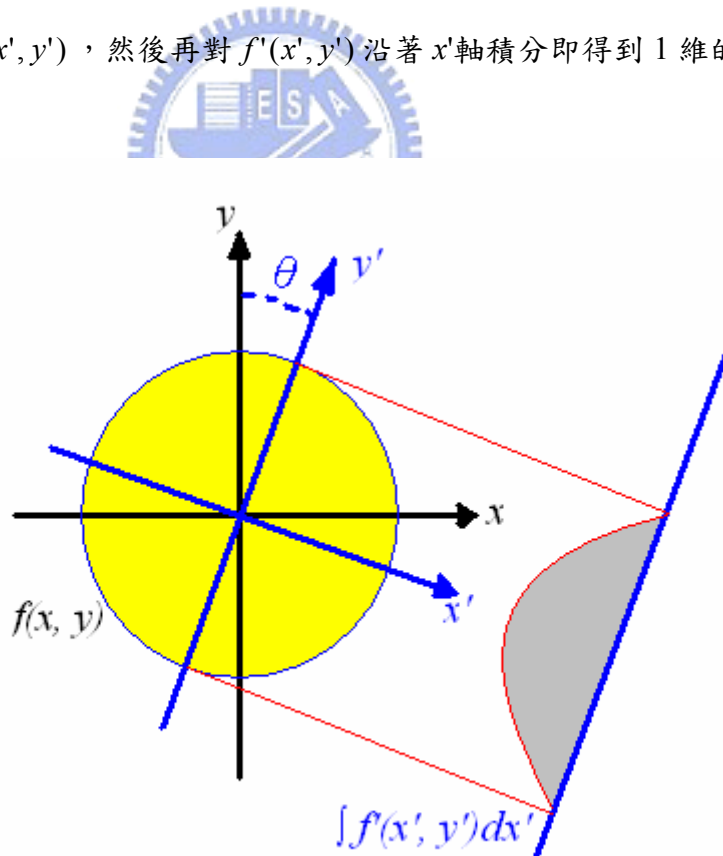


圖 2.1 2 維函數的投影

接下我們將證明傅立葉截面理論。首先，先看一下傅立葉轉換定義為何，令 $FT\{\}$ 為傅立葉轉換， $FT^{-1}\{\}$ 為反傅立葉轉換。

一維的傅立葉轉換定義如下：

$$F(u) = FT\{f(x)\} = \int_{-\infty}^{\infty} f(x)e^{-2\pi iux} dx \quad (1)$$

一維的反傅立葉轉換的定義如下：

$$f(x) = FT^{-1}\{F(u)\} = \int_{-\infty}^{\infty} F(u)e^{2\pi iux} du \quad (2)$$

二維的傅立葉轉換定義如下：

$$F(u, v) = FT_2\{f(x, y)\} = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x, y)e^{-2\pi i(ux+vy)} dx dy \quad (3)$$

二維的反傅立葉轉換的定義如下：

$$f(x, y) = FT_2^{-1}\{F(u, v)\} = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} F(u, v)e^{2\pi i(ux+vy)} dudv \quad (4)$$

三維的傅立葉轉換定義如下：

$$F(u, v, w) = FT_3\{f(x, y, z)\} = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x, y, z)e^{-2\pi i(ux+vy+wz)} dx dy dz \quad (5)$$

三維的反傅立葉轉換的定義如下：

$$f(x, y, z) = FT_3^{-1}\{F(u, v, w)\} = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} F(u, v, w)e^{2\pi i(ux+vy+wz)} dudv dw \quad (6)$$

在此我們以 3 維的函數進行證明。今我們有一 3 維的函數 $f(x, y, z)$ ，我們可對 f 進行的線性轉換 T ：

$$T\{f(x, y, z)\} = g(x, y, z)$$

根據式(5)可得到 g 的頻域函數 G ：

$$G(u, v, w) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} g(x, y, z)e^{-2\pi i(ux+vy+wz)} dx dy dz$$

接下來我們對 G 截取一個 $w=0$ 的 $u-v$ 平面：

$$\begin{aligned} G(u, v, 0) &= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} g(x, y, z)e^{-2\pi i(ux+vy)} dx dy dz \\ \Rightarrow G(u, v, 0) &= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \left(\int_{-\infty}^{\infty} g(x, y, z) dz\right) e^{-2\pi i(ux+vy)} dx dy \end{aligned} \quad (7)$$

因為 3 維的函數經過一次投影會成為 2 維的函數，所以令 $p(x, y) = \int_{-\infty}^{\infty} g(x, y, z) dz$ 代表對 g 沿著 z 軸的投影。(7)式可改寫為：

$$G(u, v, 0) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} p(x, y)e^{-2\pi i(ux+vy)} dx dy \quad (8)$$

對(8)式進行 2 維的反傅立葉轉換，可以得到投影函數 $p(x, y)$ ，得證。

$$FT_2^{-1}\{G(u, v, 0)\} = p(x, y)$$

因此，我們可以應用傅立葉截面理論來對立體資料進行任意視角的投影描繪。
FVR 演算法的步驟如下：

1. 給一組立體資料 $f(x, y, z)$ 。
2. 進行 3 維的傅立葉轉換得頻譜資料 $F(u, v, w)$ 。
3. 任取一個視點方向 E 。
4. 進行線性轉換 T ， $G(u, v, w) = F(u', v', w') = T\{F(u, v, w)\}$ ，使得 w' 與 E 同向。
5. 以 $w'=0$ 且通過原點截取一頻域函數 $P(u', v')$ 。
6. 對 $P(u', v')$ 進行 2 維的反傅立葉轉換可得 $p(x, y)$ ，即代表對時域函數 f 沿著視點方向的投影。

圖 2.2 所示為應用傅立葉截面理論的 FVR 演算法示意圖。

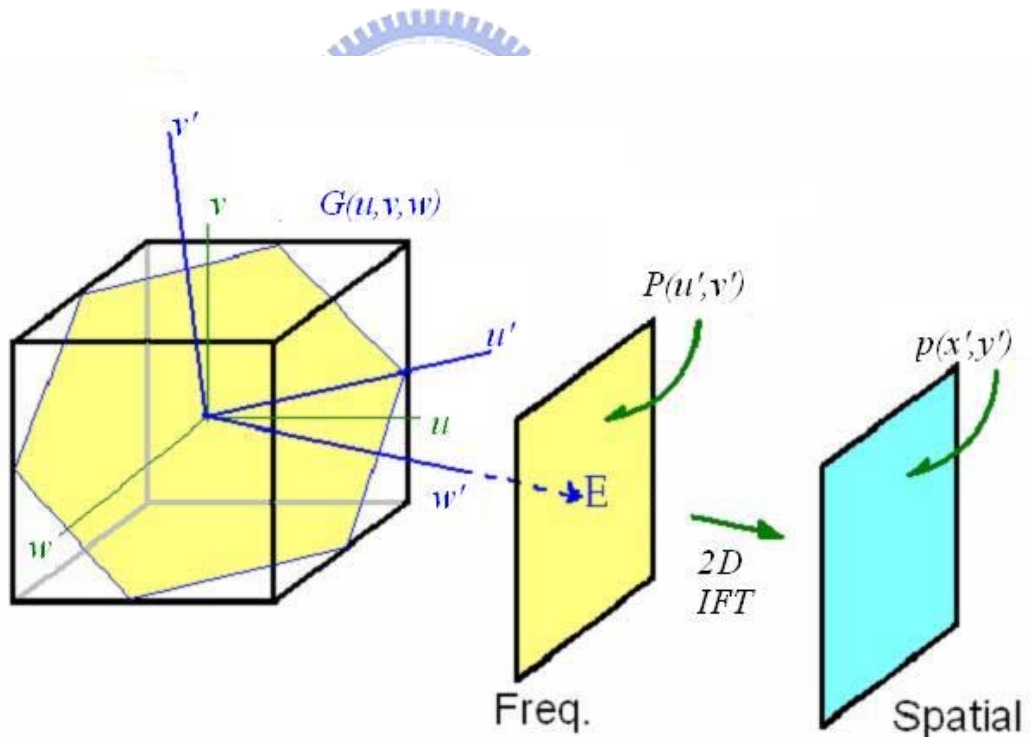


圖 2.2 FVR 演算法

我們來分析一下這個演算法。假設立體資料的大小為 N^3 ，一開始所進行的 3 維傅立葉轉換，若我們使用 FFT 來實作，則時間複雜度會是 $O(N^3 \log N)$ ，但這只會執行一次而已。接下來是截取一張平面，也就是取樣動作，若不使用任何濾波器，

我們是可以交由 GPU 的 Rendering pipeline 來完成，這樣的時間幾乎為常數時間。但是在實際的情況下是必須要使用濾波器的，若濾波器的大小為 M ，那麼就必須花費 $O(MN^2)$ 的時間。最後是 2 維的反傅立葉轉換，一樣也是 FFT 來實作，則時間複雜度會是 $O(N^2 \log N)$ 。因此，在不考慮只執行一次的 3 維傅立葉轉換下，且 $M < N$ ，則整個 FVR 演算法的時間複雜度就會是 $O(N^2 \log N)$ 。

2.2 頻域取樣問題

2.2.1 脈衝訊號

令 $\delta(t)$ 為一脈衝訊號(Impulse signal)，則：

$$\delta(t) = \begin{cases} \infty, & t = 0 \\ 0, & t \neq 0 \end{cases}$$

而且它具有這個特性：

$$\int_{-\infty}^{\infty} \delta(t) dt = 1 \quad (9)$$

其訊號圖形如圖 2.3 所示：

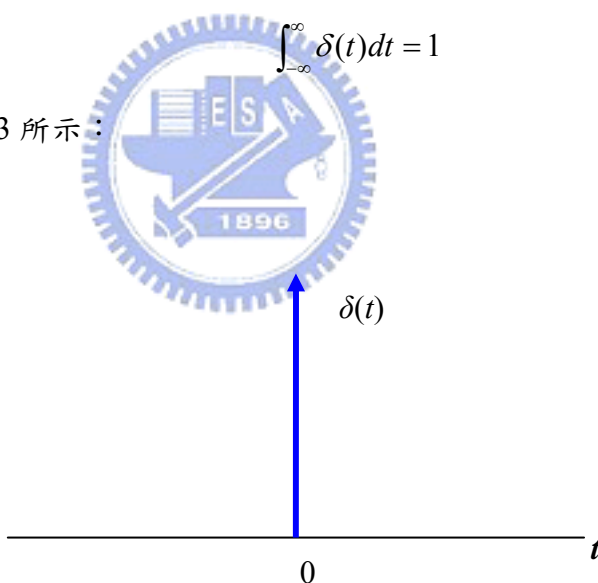


圖 2.3 脈衝訊號

定義
$$III(t) = \sum_{n=-\infty}^{\infty} \delta(t-n) \quad (10)$$

其中 III 讀音為”Shah”， $III(t)$ 具有單位週期脈衝的特性，其訊號圖形如圖 2.4 所示：

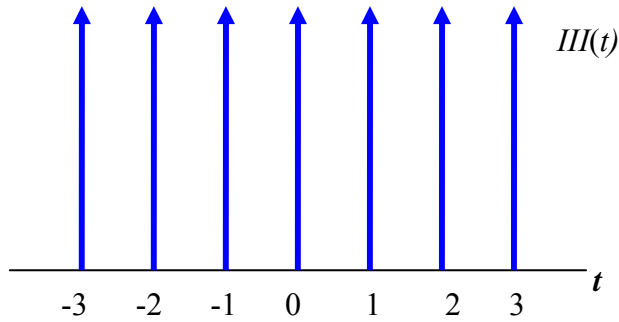


圖 2.4 單位週期脈衝訊號

若欲控制其週期，可乘上一個實數 D ：

$$III(Dt) = \frac{1}{|D|} \sum_{n=-\infty}^{\infty} \delta(t - \frac{n}{D}) \quad (11)$$

對 $III(t)$ 進行傅立葉轉換仍然會是一種週期脈衝的函數，且具有週期反比的特性：

$$FT\{III(Dt)\} = \frac{1}{D} III(\frac{x}{D}) \quad (12)$$

$$\text{或 } FT\{III(\frac{x}{D})\} = DIII(Dt) \quad (13)$$

圖 2.5 為 $III(Dt)$ 的時域與頻域的訊號圖形，左半部為時域，右半部為頻域。

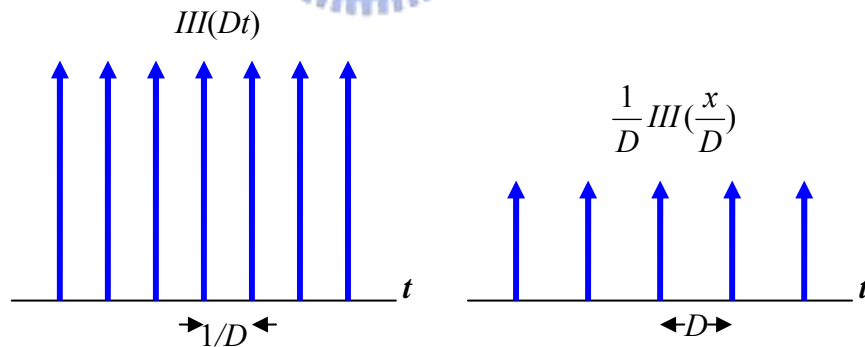


圖 2.5 $III(Dt)$ 的時域與頻域

2.2.2 頻域空間取樣

對一個連續的函數 $F(u)$ 進行取樣，等同乘上一個週期脈衝的取樣函數：

$$F_s(u) = F(u)III(Du) \quad (14)$$

其中 $F_s(u)$ 代表取樣後的函數，如圖 2.6 的(a)(c)(e)既為對 $F(u)$ 進行取樣的過程。假設 $F(u)$ 代表一個頻域函數，那麼它的取樣過程在時域的情況下，就會是一種迴旋積分的情形(Convolution)。如圖 2.6 的(b)(d)(f)所示，令 $f(x)$ 為 $F(u)$ 的時域函數，根據式(12) $III(Du)$ 的時域函數為 $\frac{1}{D} III(\frac{x}{D})$ ，那麼 $F_s(u)$ 的時域函數 $f_s(x)$ 就會是：

$$f_s(x) = f(x) * \left(\frac{1}{D} III\left(\frac{x}{D}\right) \right) \quad (15)$$

其中*代表迴旋積分。

可以發現， $f_s(x)$ 會是一個週期性的函數，若頻域空間的取樣頻率足夠，既滿足 Nyquist frequency，那麼 $f_s(x)$ 的週期訊號就不會互相影響，也就不會有失真情況發生。

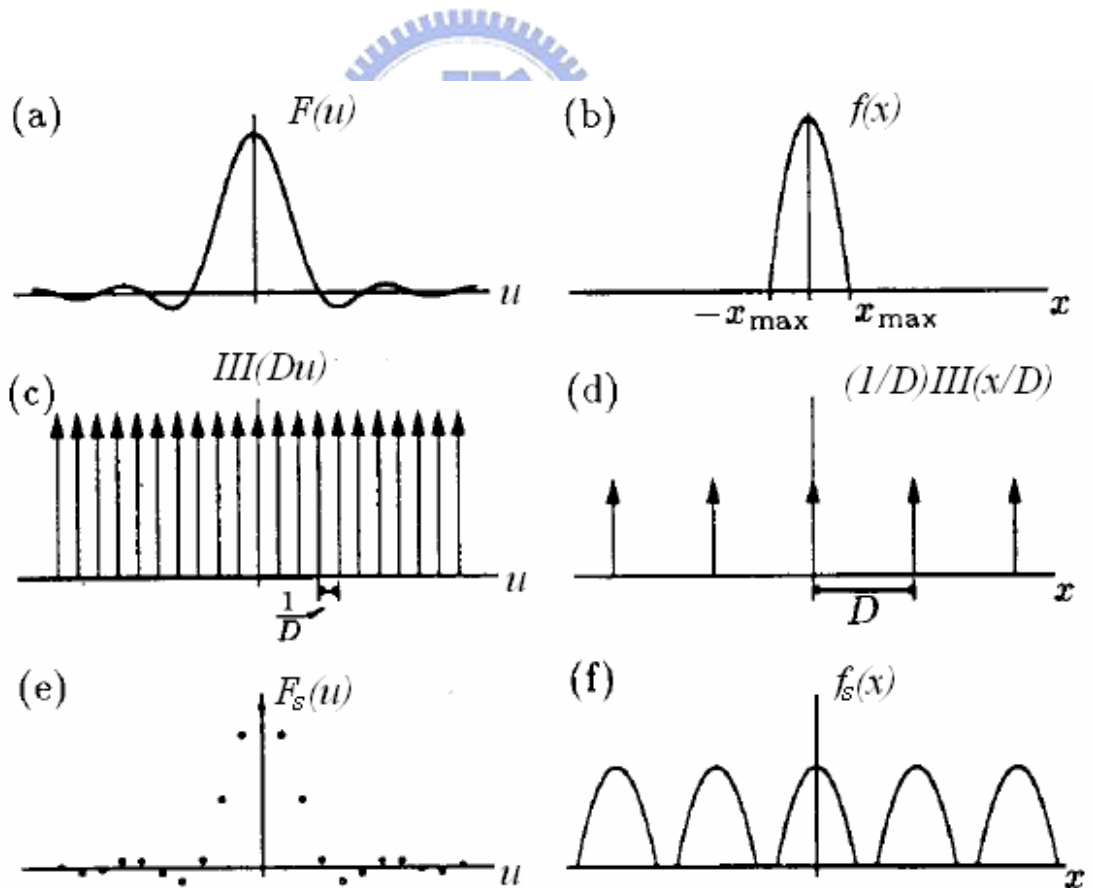


圖 2.6 頻域空間取樣過程

但若是頻域空間的取樣不足，那麼取樣函數在時域空間的頻率就會過於密集，經過迴旋積分的動作，就會生重疊訊號的失真現象。如圖 2.7 所示，(a)為以圖 2.6 的頻域函數 $F(u)$ 進行頻率較少的取樣，(b)則是該取樣函數的時域函數，(d)是在時域空間會發生嚴重的重疊失真。

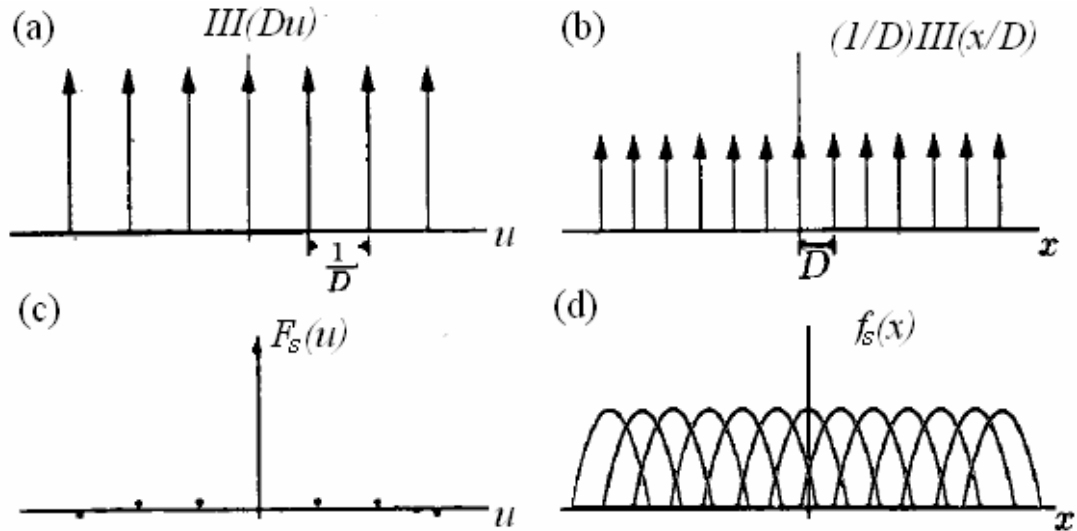


圖 2.7 取樣頻率不足下的失真

2.2.3 再取樣問題

而 FVR 演算法中，會對 3 維立體資料的頻譜資料通過原點截取一張平行於投影平面的 2 維頻譜資料，這樣的動作就是一個頻域空間的取樣動作，而且是對已取樣後的資料進行再取樣的動作(Resampling)。若是投影角度不為 90 度的倍角時，就會發生取樣頻率不足的問題。如圖 2.8 以 2 維的情況來分析，藍色的*記號為離散的頻譜資料，紅色的虛線為欲取 7 個取樣點的線段，那麼所取樣的資料，即紅框標示部份，很明顯的有取樣頻率不足之情況。同樣道理，FVR 最後的投影結果就會發生嚴重的重疊失真，如圖 2.9 所示。

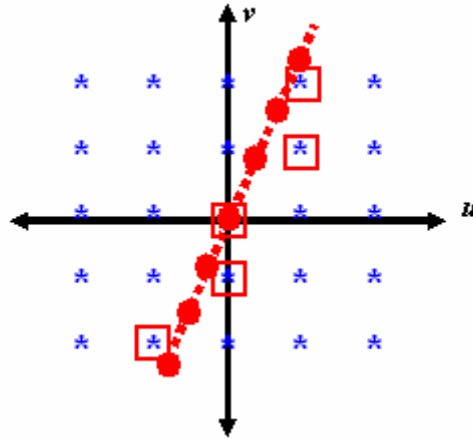


圖 2.8 再取樣問題

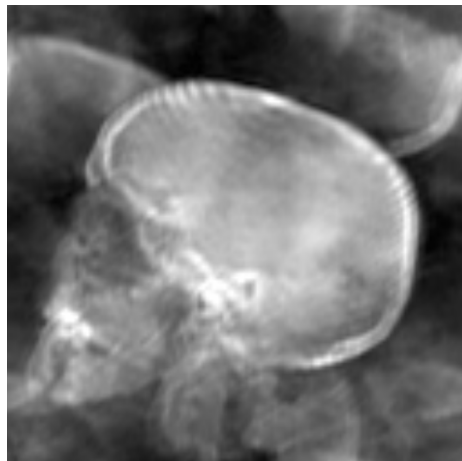


圖 2.9 FVR 的失真現象

2.3 濾波器設計

對於頻域取樣的失真問題，Malzbender [7]、Lichtenbelt [22]提出了許多解決的方法，本節將介紹這些方法。

2.3.1 時域空間延伸

取樣頻率不足的問題最直覺上的解決方法就是增加取樣頻率，我們可以將原始資料尺寸延伸兩倍，但影像的大小及位置皆不變，至於多餘的空間則以零來填補。然後在頻域取樣時以兩倍取樣頻率來取樣，理論上就可以解決失真的問題了。但這

方法會增加兩倍的記憶體空間，而且不能解決再取樣問題，仍需搭配後面將介紹的濾波器，單靠此方法並不能解決重疊影像的問題。

2.3.2 sinc 濾波器

sinc 函數的定義如下：

$$\text{sinc}(x) = \begin{cases} 1, & x = 0 \\ \frac{\sin(\pi x)}{\pi x}, & x \neq 0 \end{cases} \quad (16)$$

它的訊號圖形如圖 2.10 左半部所示：

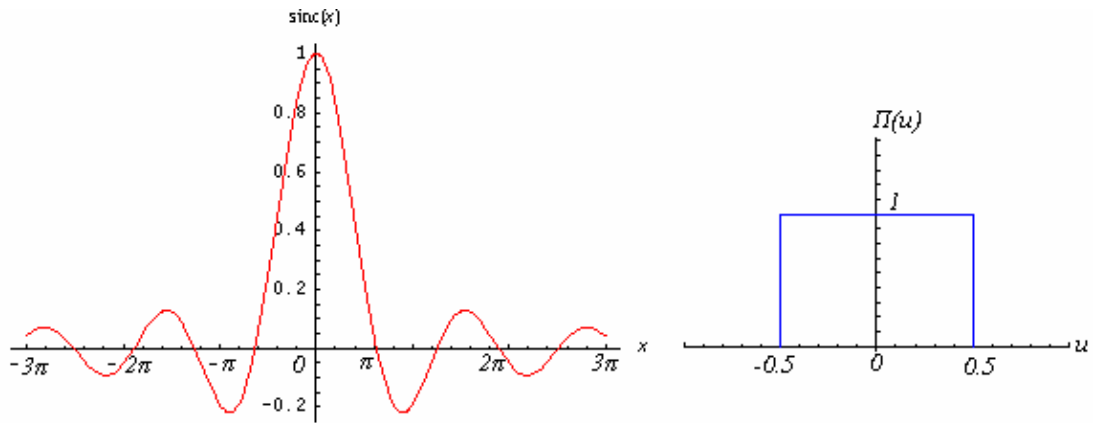


圖 2.10 sinc 函數與 Π 函數

sinc 函數有一個重要的特性，就是它在經過傅立葉轉換後會是一個矩形訊號，如圖 2.10 右半部。

$$FT\{\text{sinc}(x)\} = \int_{-\infty}^{\infty} \text{sinc}(x) e^{-2\pi i u x} dx = \Pi(u) \quad (17)$$

$$\text{其中 } \Pi(u) = \begin{cases} 0, & |u| > 0.5 \\ 1, & |u| < 0.5 \\ 0.5, & |u| = 0.5 \end{cases}$$

若是我們能夠以 Π 函數來保留訊號的中心部份，將有可能與週期訊號重疊的部份排除掉，那麼就可以消除重疊影像了。我們對式(15)乘上一個 Π 函數：

$$f_s(x) = f(x) * \left(\frac{1}{D} \text{III}\left(\frac{x}{D}\right) \right) \Pi\left(\frac{x}{a}\right) \quad (18)$$

其中 a 為寬度，兩函數相乘在頻域空間就會是兩函數的迴旋積分，那麼式(18)在頻域空間就會是：

$$F_s(u) = (F(u)III(Du)) * a \text{sinc}(au) \quad (19)$$

根據式(19)我們可以設計一個 sinc 濾波器用在取樣的過程中，以解決再取樣的問題。但由於 sinc 函數是一個無限延伸的函數，因此必須對它範圍加以限制，我們可以乘上一個 window 函數，一般常以 Hamming window 來限制 sinc 函數：

$$W(x) = 0.54348 + 0.45652 \cos\left(\frac{2\pi x}{a}\right) \quad (20)$$

其中常數 a 為 window 的寬度。

經過 window 函數限制過的 sinc 濾波器理論上是較為理想的濾波器，但是其大小要在 5^3 (voxel)以上才能發揮功能，在計算成本可說是在所有類型的濾波器中最高的。



2.3.3 三線性濾波器

我們可以在取樣過程中加上濾波器(Filter)，以解決再取樣問題，其中最簡易的濾波器就是線性濾波器(Linear filter)，而它是以線性內插(Linear interpolation) [4]的原理來設計。如圖 2.11 所示，對兩個取樣點 P_1 及 P_2 ，在位置 m 處進行一次線性內插取得 P_{int} 。

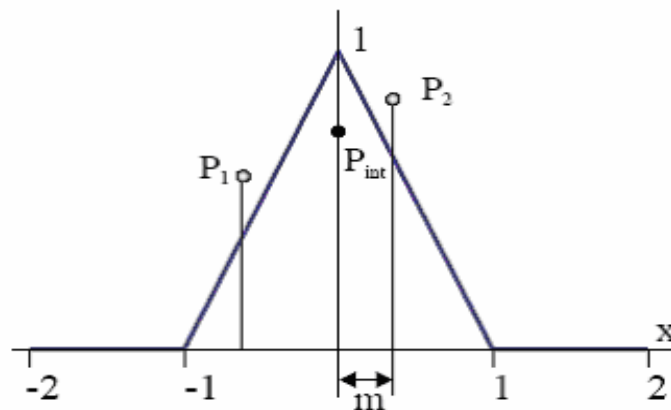


圖 2.11 線性內插

三線性濾波器(Tri-linear filter)如圖 2.12 所示，共有八個取樣點 $V_{000} \sim V_{111}$ ，先對 X 軸進行一次線性內插，可以得 $V_{x00} \sim V_{x11}$ 四個內插值，再對這四個內插值進行 Y 軸的一次線性內插，可以得到 V_{xy0} 、 V_{xy1} 這兩個內插值，最後進行 Z 軸的一次線性內插，即可得到最後的內插值 V_{xyz} 。此方法最容易實作，但並不能完全消除重疊影像的情況。

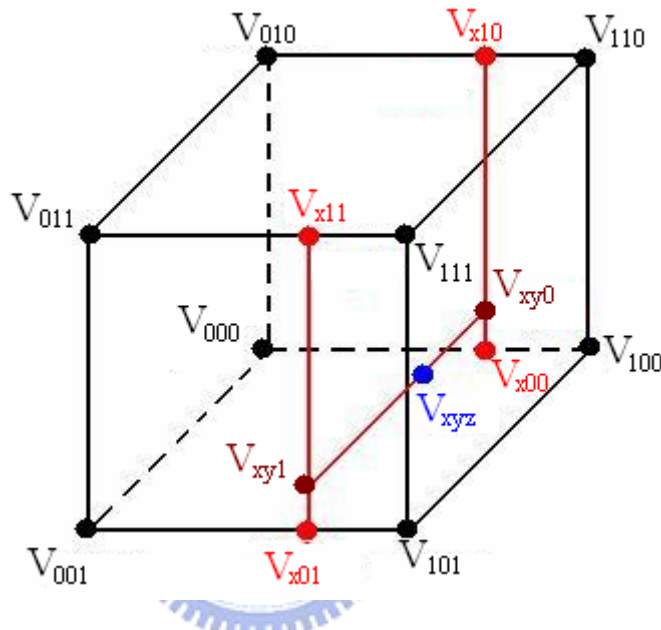


圖 2.12 三線性濾波器

2.3.4 三次曲線濾波器

三次曲線濾波器(Tri-cubic filter)與三線性濾波器不同之處在於它的內插方式是以曲線來進行，曲線方程式有很多種，其中以 Catmull-Rom 曲線[23]最適合。

Catmull-Rom 曲線方程式如下：

$$P(u) = \begin{bmatrix} u & u^2 & u & 1 \end{bmatrix} \times \begin{bmatrix} -0.5 & 1.5 & -1.5 & 0.5 \\ 1.0 & -2.5 & 2.0 & -0.5 \\ -0.5 & 0 & 0.5 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} P_{i-1} \\ P_i \\ P_{i+1} \\ P_{i+2} \end{bmatrix} \quad (21)$$

其中 u 為內插的距離， $u \in [0,1]$ ； $P_{i-1} \sim P_{i+2}$ 為四個取樣點，則內插值就會介於 P_i 與 P_{i+1} 之間，如圖 2.13 所示：

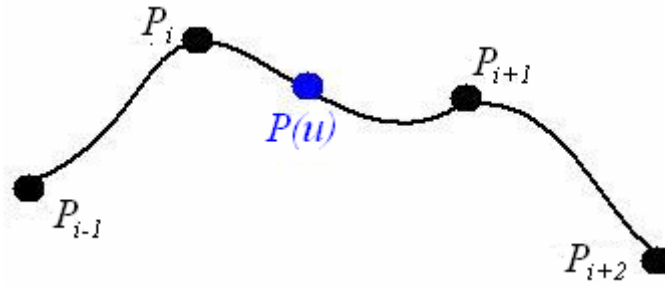


圖 2.13 Catmull-Rom 曲線

三次曲線濾波器在 FVR 演算法中，需要先取得 $4^3=64$ 個取樣點，才能得到一個內插值，雖然這對計算上將會是很大的負擔，但會比 5^3 的 sinc 濾波器計算量少，而且效果也相差不大，因此在實作上三次曲線濾波器是不錯的選擇。

2.3.5 時域空間衰減補償

上述所介紹的濾波器，皆等同對立體資料的頻域函數與濾波函數進行迴旋積分，如三線性濾波器就是與一個三角波函數行迴旋積分，這樣一來在時域空間就會發生外圍部份衰減的現象，如圖 2.14 所示，三角波函數 $H(u)$ 經過傅立葉轉換會變成 $\text{sinc}^2(x)$ ，導致投影後的結果會是中心較亮但外圍會呈現越來越暗的情況。因此，我們必須要在原始的二維資料進行補償的動作。

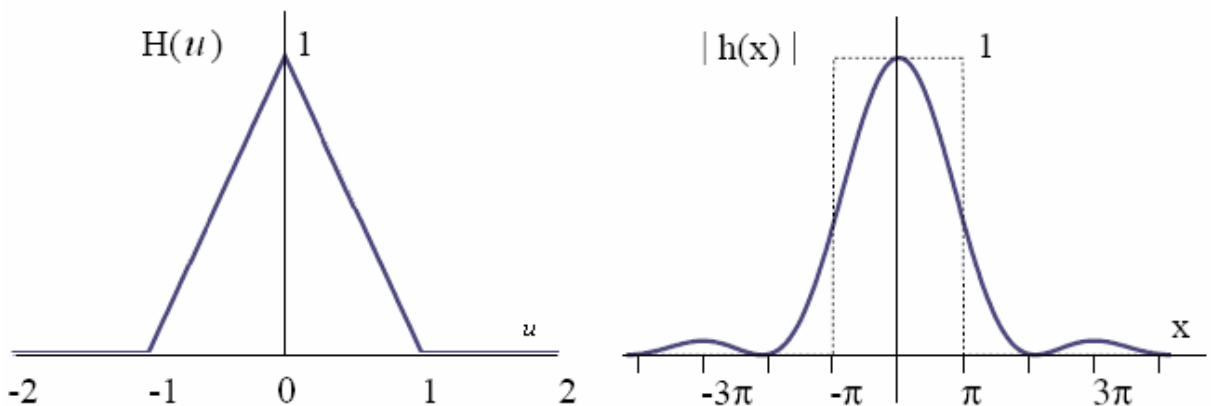


圖 2.14 時域空間的衰減

補償動作很簡單，只要將原始的立體資料乘上經過傅立葉轉換後的濾波函數之倒數即可。令 $H(u)$ 為濾波函數， $h(x)=FT\{H(u)\}$ ，則補償後的立體資料為：

$$f'(x) = \frac{f(x)}{h(x)} \quad (22)$$

2.4 著色模型

FVR 的著色模型(Shading model)是由 Levoy [9]所提出，他利用了傅立葉轉換的線性關係，將各種著色模型中的經常變動數移至 3 維的傅立葉轉換之外，剩下與立體資料位置有關的變數就可以在進行 3 維的傅立葉轉換之前預先計算，如此就可以有即時顯像的效果。

2.4.1 FVR 著色模型

根據 2.1 節所介紹的 FVR 演算法，我們可以用一個 Π_v 的運算子來表示：

$$I = \Pi_v(f(x)) = FT_2^{-1}\{FT_3\{f(x)\}\delta_v\} \quad (23)$$

其中 I 為投影影像， $f(x)$ 為立體資料， FT_2^{-1} 為 2 維的反傅立葉轉換， FT_3 為 3 維的傅立葉轉換， δ_v 代表截取一張與投影平面平行的 2 維平面資料。另外，傅立葉轉換具有線性組合的特性：

$$FT\{af(x) + bg(x)\} = aFT\{f(x)\} + bFT\{g(x)\} \quad (24)$$

其中 a, b 為常數。

若我們對立體資料乘上一個著色函數 $g(x)$ ，則式(23)可改寫為：

$$I = \Pi_v(f(x)g(x)) = FT_2^{-1}\{FT_3\{f(x)g(x)\}\delta_v\} \quad (25)$$

由式(25)可知，當 $g(x)$ 發生任何變動時，則 3 維的傅立葉轉換必須重新運算。假使我們可以把 $g(x)$ 進行分解，讓變數及與 x 形成一線性組合，就可讓 $g(x)$ 的變動不會影響 3 維的傅立葉轉換。我們希望 $g(x)$ 可以成為：

$$g(x) = g_0(x)h_0(t) + g_1(x)h_1(t) + \dots + g_n(x)h_n(t) \quad (26)$$

將式(26)代入式(25)可以得到：

$$\begin{aligned}
 I &= \Pi_v(f(x)(g_0(x)h_0(t) + g_1(x)h_1(t) + \dots + g_n(x)h_n(t))) \\
 &= FT_2^{-1}\{(h_0(t)FT_3\{f(x)g_0(x)\} + h_1(t)FT_3\{f(x)g_1(x)\} + \dots + h_n(t)FT_3\{f(x)g_n(x)\})\delta_v\} \\
 &= h_0(t)\Pi_v(f(x)g_0(x)) + h_1(t)\Pi_v(f(x)g_1(x)) + \dots + h_n(t)\Pi_v(f(x)g_n(x)) \\
 &= \sum_{i=0}^n h_i(t)\Pi_v(f(x)g_i(x))
 \end{aligned} \tag{27}$$

式(27)即為 FVR 的著色模型。

2.4.2 遠近效果

由於 FVR 演算法所產生的投影影像是對視點方向積分的結果，並沒有任何遠近效果(Depth cueing)。然而我們可以用一線性方程式來做為一個對深度具線性衰減的著色函數：

$$g(P) = \frac{\tau}{2} + \frac{\tau}{2}(V \cdot D(P)) \tag{28}$$

其中 V 為視點方向， $V=[V_x, V_y, V_z]$ ； P 為 voxel 位置， $P=[P_x, P_y, P_z]$ ； $D(P)$ 為一個線性方程式，例如把 P 成為單位向量， $D(P)=[D_x, D_y, D_z]$ ； τ 為一常數。

如圖 2.15 所示， P_1 與 V 的內積值就會大於 P_2 與 V 的內積值，則投影過程中 P_2 的 voxel 的 weight 就會比 P_1 的 voxel 少，即呈現遠近的效果。

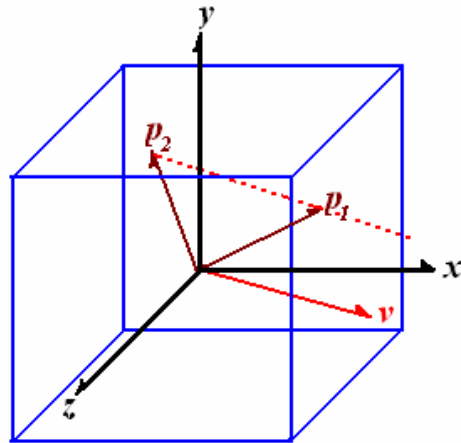


圖 2.15 線性深度衰減

整個遠近效果的著色模型就會是：

$$\begin{aligned}
 I = & V_x \Pi_v \left(f(P) \left(\frac{\tau}{2} + \frac{\tau}{2} D_x(P) \right) \right) + \\
 & V_y \Pi_v \left(f(P) \left(\frac{\tau}{2} + \frac{\tau}{2} D_y(P) \right) \right) + \\
 & V_z \Pi_v \left(f(P) \left(\frac{\tau}{2} + \frac{\tau}{2} D_z(P) \right) \right) + \\
 & \left(\frac{\tau}{2} - \frac{\tau}{2} (V_x + V_y + V_z) \right) \Pi_v (f(P))
 \end{aligned}
 \tag{29}$$

圖 2.16 是資料大小為 128^3 的 CT 頭部 FVR 結果，視點為後腦，左邊為未使用遠近效果，則發現面孔的所有細節也會呈現出來，以致無法辨識是從後腦觀測，右邊為使用遠近效果，可以很清楚地辨識是從後腦觀測。

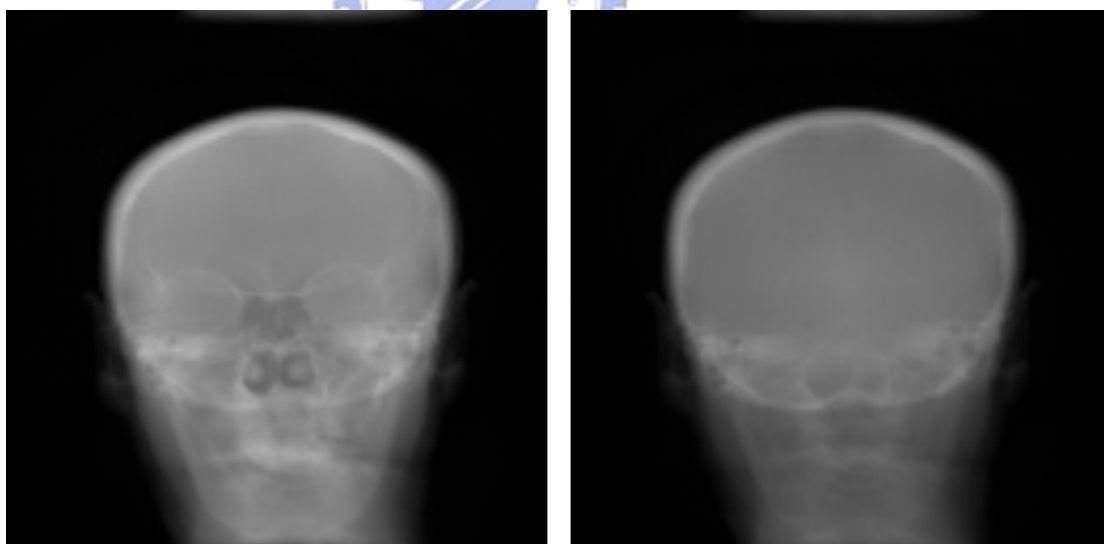


圖 2.16 遠近效果

2.4.3 光照模型

同樣地，我們也可以設計出直向光源的光照模型(Directional lighting)：

$$g(P) = \frac{\pi}{2} + \frac{\pi}{2}(L \cdot N(P)) \quad (30)$$

其中 L 為光源方向， $L=[L_x, L_y, L_z]$ ； P 為 voxel 位置， $P=[P_x, P_y, P_z]$ ； $N(P)$ 為 voxel 的法向量， $N(P)=[N_x, N_y, N_z]$ 。整個光照模型就會是：

$$\begin{aligned} I = & L_x \Pi_v \left(f(P) \left(\frac{\tau}{2} + \frac{\tau}{2} N_x(P) \right) \right) + \\ & L_y \Pi_v \left(f(P) \left(\frac{\tau}{2} + \frac{\tau}{2} N_y(P) \right) \right) + \\ & L_z \Pi_v \left(f(P) \left(\frac{\tau}{2} + \frac{\tau}{2} N_z(P) \right) \right) + \\ & \left(\frac{\pi}{2} - \frac{\pi}{2} (L_x + L_y + L_z) \right) \Pi_v (f(P)) \end{aligned} \quad (31)$$

圖 2.17 是資料大小為 64^3 的球體 FVR 光照結果，光線分別為從左邊及右邊照射。

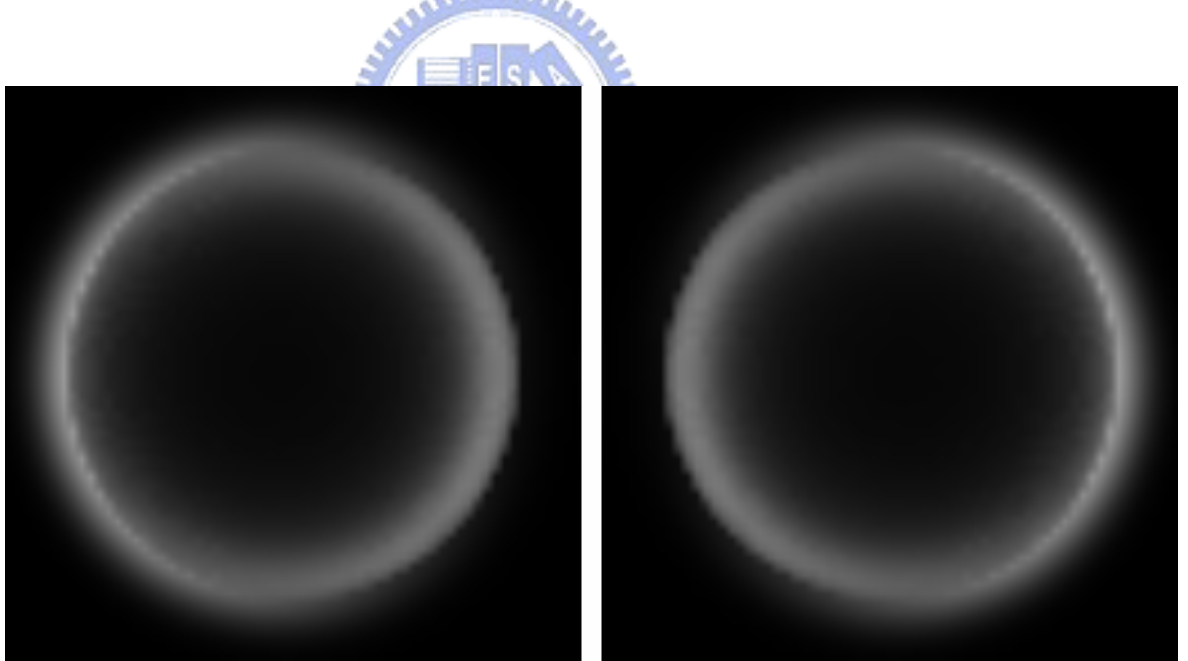


圖 2.17 光照效果

三、轉換函數的設計

第一章已經介紹過什麼是轉換函數，以及如何應用在分類顯像上。本章將介紹我們所提出的兩種分類顯像的方式：預先影像處理及貝茲曲線轉換函數。

3.1 預先影像處理

我們可以事先對原始的立體資料進行任何的影像處理，如二值化處理(Binary process)、邊緣偵測(Edge detection)、影像模糊(Blur)等技術，如圖 3.1 右半部為立體資料經過 Canny edge detection 後的影像，左半部為原始影像。

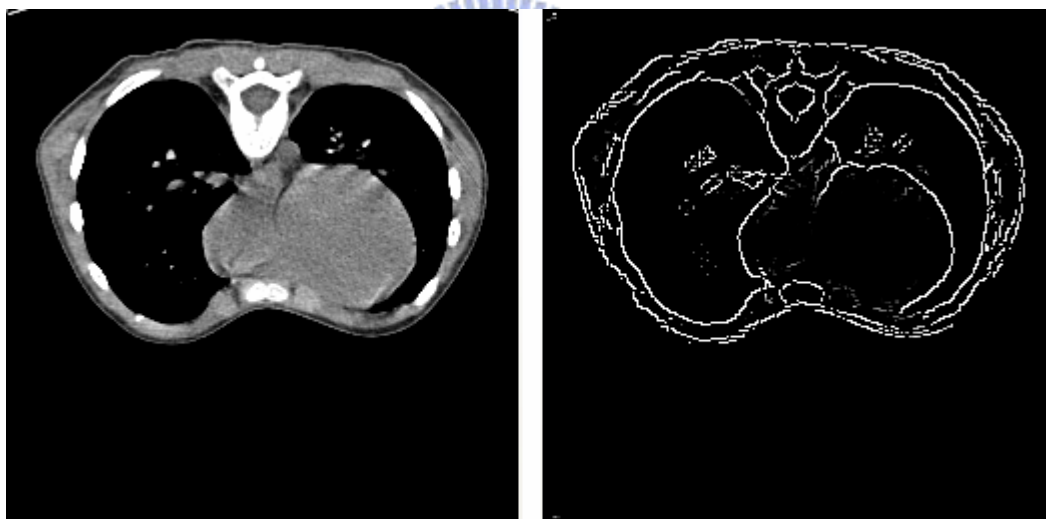


圖 3.1 Canny edge detection

當然，我們也可以事先將立體資料進行分類的動作，再利用第二章所提到的著色模型，然後於每次取樣的時候進行合併動作。我們可以應用式(27)進行分類動作：

$$I = w_0 \Pi_v(f_0(x)) + w_1 \Pi_v(f_1(x)) + \dots + w_{n-1} \Pi_v(f_{n-1}(x)) \quad (32)$$

其中 $f_0(x), f_1(x), \dots, f_{n-1}(x)$ 代表我們先將立體資料複製 n 份，然後對每一份立體資料進行不同的影像處理動作。 $w_{0 \sim n-1}$ 代表每個立體資料在投影時的加權比重。

舉例來說明，如圖 3.2 所示，我們將胸腔的立體資料複製四份，第一份立體資料(a)只保留骨骼部份，也就是灰階值在 200~255 之間(假設最大灰階值為 255)；第二份立體資料(b)只保留肌肉部份，也就是灰階值在 100~200 之間；第三份立體資料(c)只保留脂肪部份，也就是灰階值在 100 以下；第四份立體資料(d)則進行 Canny edge detection。

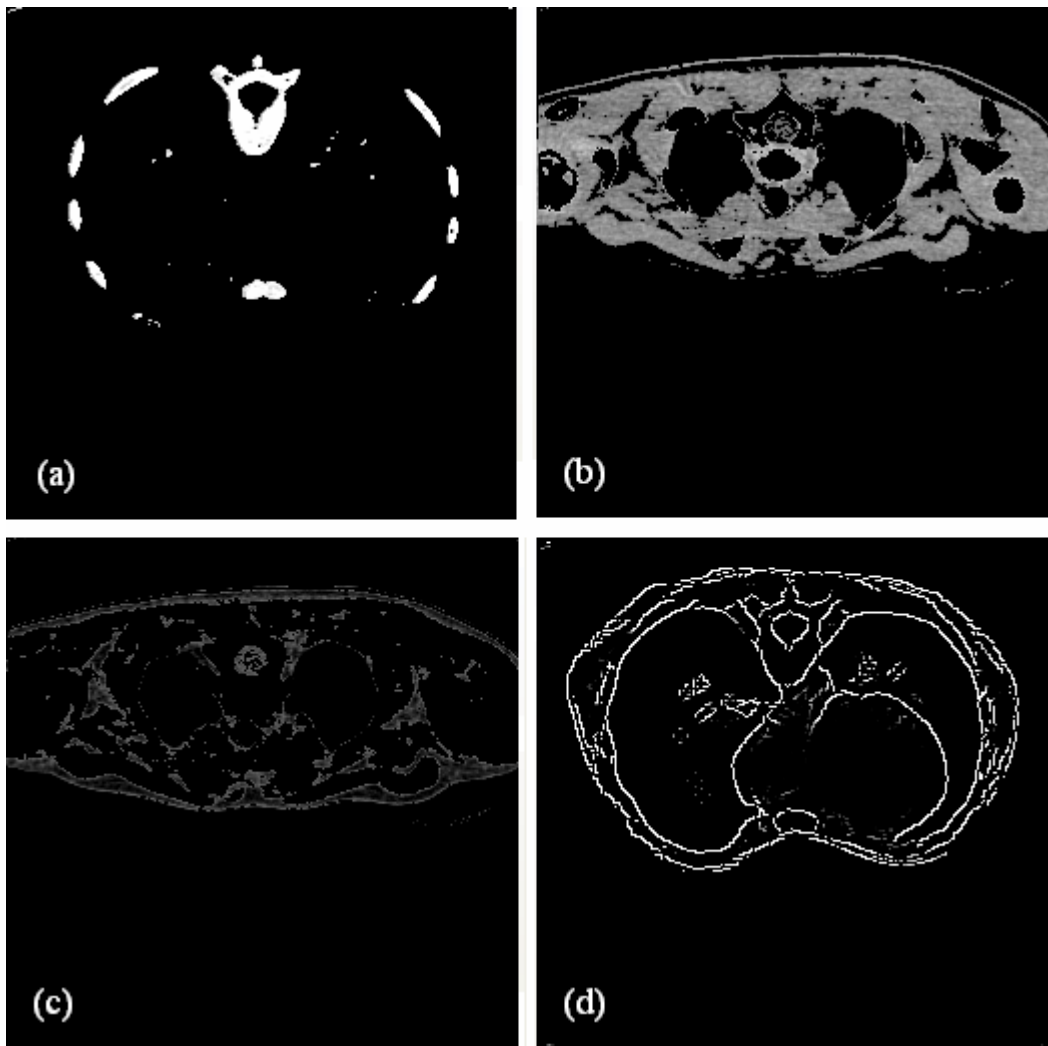


圖 3.2 立體資料的預先分類

如此一來，這四份資料透過式(32)就可以顯像的階段任意地各別調整比重，達成即時的分類顯像，如圖 3.3 所示：



圖 3.3 以影像預先處理的分類顯像

此方法的優點是我們可以很容易的指定每個分類所呈現的顏色，而且分類較為準確，投影的結果品質也比較好。但它的缺點是不易達成即時的分類顯像，若要變更分類的灰階值，則整個 3 維的傅立葉轉換必須重新運算。若想避免這個現象，就必須將立體資料所複製的份數跟灰階值的數目一樣多，這對記憶體空間來說是很大的負擔，所以此方法對於分類群組較少的情況下較為適用。

3.2 貝茲曲線轉換函數

關於 FVR 轉換函數的設計，目前只有 Nagy [13] 提出以步進函數(Step function) 做為轉換函數。他是將步進函數以傅立葉級數表示，並且把分類的門檻值移至 3 維的傅立葉轉換之外，如此就可以用式(27)的著色模型來進行分類顯像。此方法雖然可以達成即時的分類顯像，但卻只能做二值化的分類，也就是只能分出某個門檻

值以上的所有組織，而無法分出某一灰階值範圍的組織。我們希望使用者能自訂出每一個灰階值所對應的加權比重是多少，使得這樣的轉換函數就會像是一條曲線，如圖 3.4 所示，橫軸 V 為灰階值，縱軸 W 為加權比重：

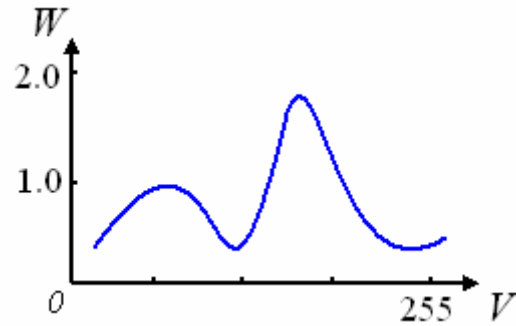


圖 3.4 轉換函數

在能夠讓使用者容易地且準確地畫出這條曲線的前提下，於是我們提出了以貝茲曲線(Bézier curve)來做為 FVR 的轉換函數。

貝茲曲線的方程式如下：

$$B(u) = \sum_{i=0}^{n-1} \binom{n-1}{i} P_i (1-u)^{n-i-1} u^i \quad (33)$$

其中 P_i 為曲線的控制點(Control point)， n 為控制點的個數， $u \in [0,1]$ 。圖 3.5 為 4 個控制點的貝茲曲線：

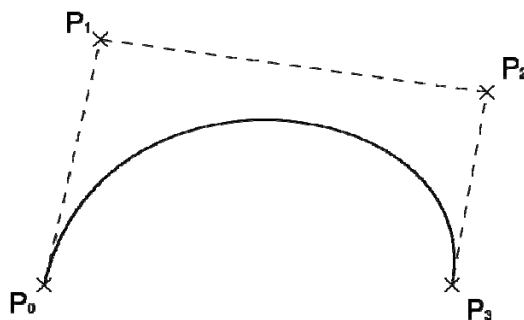


圖 3.5 貝茲曲線

由於貝茲曲線也是一種線性方程式，且曲線的控制點與 u 無關，所以成為一種 FVR 的著色模型。將式(33)做為著色函數代入式(25)：

$$\begin{aligned}
 I &= \Pi_v(f(x)B(f(x))) \\
 &= FT_2^{-1} \left\{ FT_3 \left\{ f(x) \sum_{i=0}^{n-1} \binom{n-1}{i} P_i (1-f(x))^{n-i-1} f(x)^i \right\} \delta_v \right\} \\
 &= FT_2^{-1} \left\{ \left(\sum_{i=0}^{n-1} \binom{n-1}{i} P_i FT_3 \left\{ f(x)(1-f(x))^{n-i-1} f(x)^i \right\} \right) \delta_v \right\} \\
 &= \sum_{i=0}^{n-1} \binom{n-1}{i} P_i \Pi_v(f(x)(1-f(x))^{n-i-1} f(x)^i)
 \end{aligned}$$

(34)

式(34)即為貝茲曲線轉換函數的著色模型。



四、實作與結果

4.1 實驗環境

我們以兩種不同效能的平台來做為實驗環境：

1. 低效能平台 A

- (1) CPU：AMD Sempron 2800+(1.6 GHz)
- (2) RAM：1.5 GB
- (3) GPU：ATI Mobility Radeon Xpress 200
- (4) Video RAM：64 MB
- (5) 作業系統：Microsoft Windows XP Professional

2. 高效能平台 B

- (1) CPU：Intel Pentium4 3.0 GHz
- (2) RAM：3.0 GB
- (3) GPU：ATI Radeon X850 Pro
- (4) Video RAM：256 MB
- (5) 作業系統：Microsoft Windows XP Professional



我們所使用的繪圖函式庫是 OpenGL 2.0，而上述所使用的實驗平台皆支援 OpenGL 2.0，其中我們會用到幾個重要的功能：

1. **OpenGL Shading Language, GLSL**

GPU 內有兩種處理單元，一是頂點處理器(Vertex processor)，主要功能是對輸入的頂點進行座標轉換；另一是像素處理器(Pixel processor)，主要功能是多邊形在畫面上決定好顯示區域後，且在填上該區域的每個像素前，計算這個像素的顏色會是什麼。這兩個處理器都有預設的標準動作，然而我們可

以分別為這兩種處理器寫不同的程式，進行更複雜的運算。在頂點處理器執行的程式稱為 Vertex shader，在像素處理器執行的程式稱為 Pixel shader，而這程式碼皆是以 GLSL [24]來撰寫。因此我們可以用 GLSL 來讓 FVR 演算法中許多複雜的運算工作交由 GPU 執行以及濾波器的實作。

2. 3D Texture

3D Texture 的功能可以讓我們將整個 3 維的頻譜資料載入至 Video RAM，透過 Rendering pipeline 來進行取樣的動作。我們所使用的實驗平台在規格上雖然可以一次輸入大小為 2048^3 3D texture，但是要考慮 Video RAM 的大小。若我們以 16 位元浮點數來儲存 texture，而頻譜是複數資料，若立體資料 256^3 ，那麼就需要 64MB 的 Video RAM 空間。因此平台 A 最多只能使用 256^3 的立體資料，平台 B 最多只能使用 $512*512*256$ 的立體資料。



3. Multitexturing

在實作 FVR 的著色模型時，我們必須把多個的頻譜資料取樣後進行相加的動作，Multitexturing 可以幫助我們把多張 texture 同時進行運算。我們所使用的實驗平台一次可以同時輸入 16 張 texture，然後一次最多以 8 組 texture 座標來取樣。

4. Framebuffers Objects, FBO

FBO [25]的主要功能是來讓 GPU 進行 32 位元浮點數的運算。由於 3 維頻譜資料的數值可能不是介於 0 與 1 之間，但是 GPU 會將繪圖結果的每一個像素的每一個成份，即 RGBA 的各數值，限制在 0 與 1 之間。因此我們必須使用 FBO 的功能，以離線(Off-line)的繪圖方式將繪圖結果寫在其他記憶體位置上，才能保留的完整數值而不被限制在 0 與 1 之間。而且 FBO 可以當成 texture 來使用，如此可以來完成許多遞迴的運算。

4.2 以 GPU 實作傅立葉立體資料描繪法

4.2.1 實作 FVR 演算法

Viola [26]提出以 GPU 來實作 FVR 演算法，認為這種做法是可行的而且十分合適，因此我們將以這篇論文為基礎來實作 FVR 演算法。

以 GPU 來實作 FVR 演算法的步驟如下：

1. 對原始的立體資料進行座標轉換，讓原始資料的中心座標等於原點座標。
2. 對 3 維的頻譜資料進行平移動作(Shifting)，使低頻資料集中在原點處。
3. 將平移後的頻譜資料，以 3D Texture 形式傳送至顯示卡記憶體。
4. 製作一個代理多邊形(Proxy polygon)來描述以任一視點方向所截取的平面。
此多邊形的頂點數最少有 3 個點，最多為 6 個點，且每個頂點對應一個 3 維的貼圖座標。
5. 以 GPU 來繪製代理多邊形，即會在畫面緩衝記憶體(Framebuffer)產生一塊區域。由於在 GPU 欲填上此區域上的每個像素前會呼叫 Pixel shader。我們可以把濾波動作寫在 Pixel shader 內，以解決再取樣問題。
6. 我們可以從畫面緩衝記憶體讀取出頻譜的截面資料後，再以 2 維的反傅立葉轉換得到投影結果。或是直接對畫面緩衝記憶體以 GPU 來進行 2 維的反傅立葉轉換，再從畫面緩衝記憶體讀取出投影結果。
7. 對投影結果再進行一次座標轉換即最後的結果。

這 7 個步驟我們以圖 4.1 來表示之：

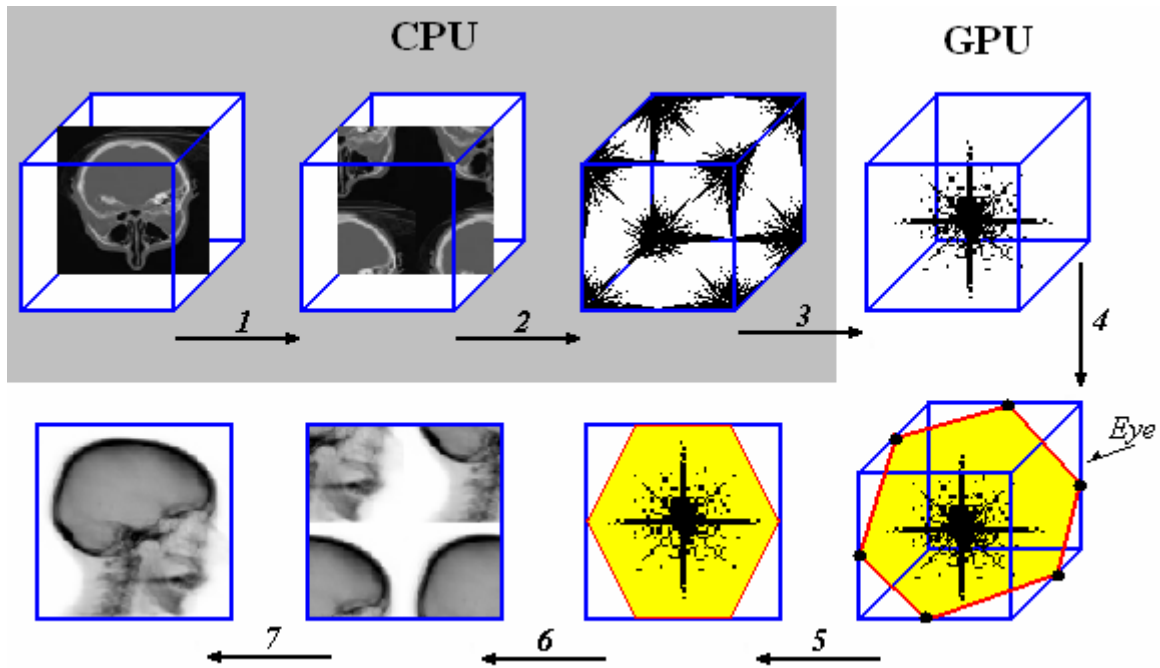


圖 4.1 FVR 實作步驟

我們以 64^3 、 128^3 、 256^3 三組不同大小的頭部 CT 立體資料來進行不使用任何濾波器及著色模型的 FVR 演算法，3 維傅立葉轉換是以標準 FFT 來進行，2 維的反傅立葉轉換是以 FFTW，其實驗數據如表 4.1 所示：

平台 A			
資料大小	64^3	128^3	256^3
實驗項目			
Pre-processing(sec.)	0.813	8.015	104.407
Frame rate(FPS)	66.67	62.25	16.129

平台 B			
資料大小	64^3	128^3	256^3
實驗項目			
Pre-processing(sec.)	0.25	3.359	33.297
Frame rate(FPS)	100	66.67	32.25

表 4.1 不使用任何濾波器

在第二章時我們有提到，在 FVR 會有再取樣的問題而導致取樣頻率不足，使得投影結果會有重疊影像發生，如圖 4.2 所示，即使是較高解析度的立體資料也是一樣，左邊為 64^3 ，右邊為 256^3 ，下方為各別的頻譜取樣結果：

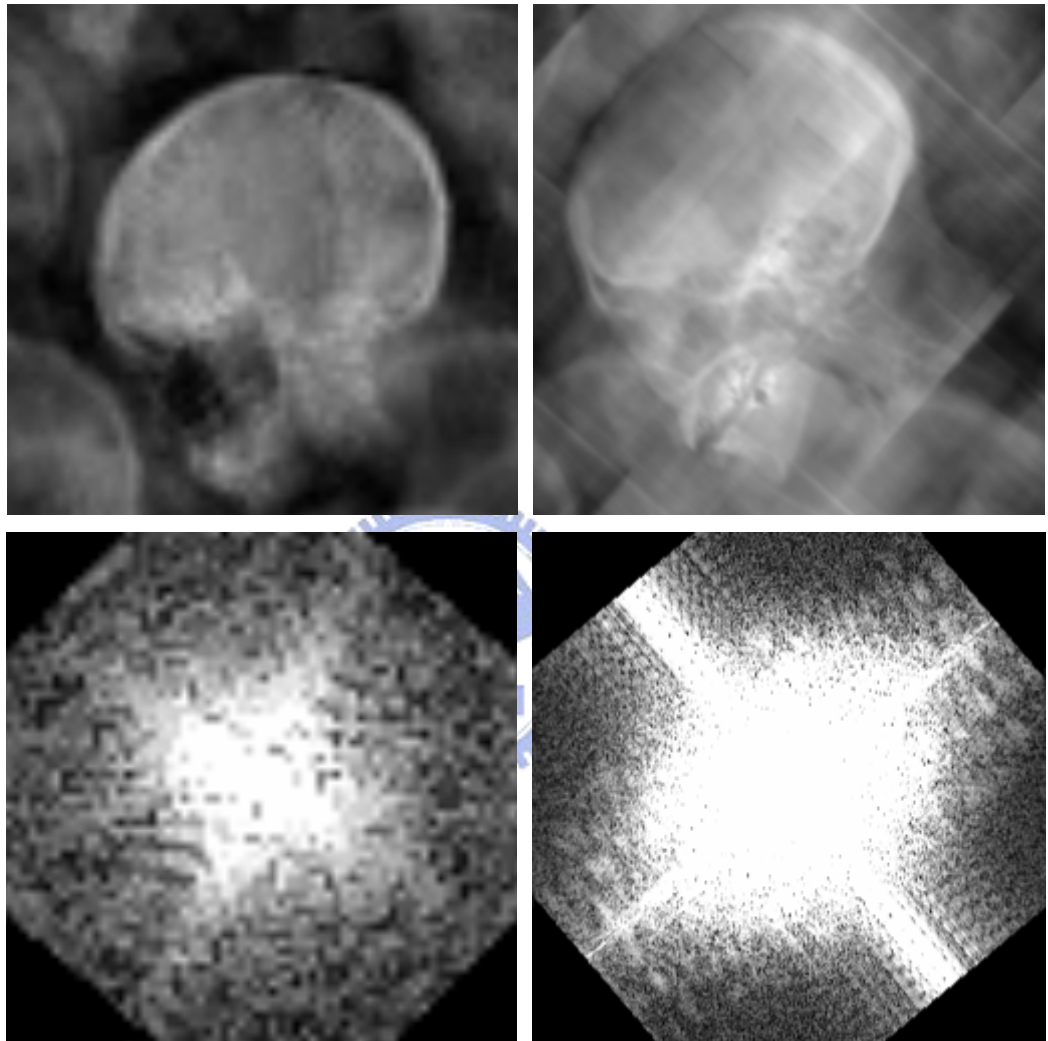


圖 4.2 重疊影像的失真

但在視角為 90 度的倍角時，就不會有重疊影像發生，如圖 4.3 所示，左邊為 64^3 ，右邊為 256^3 ，下方為各別的頻譜取樣結果：

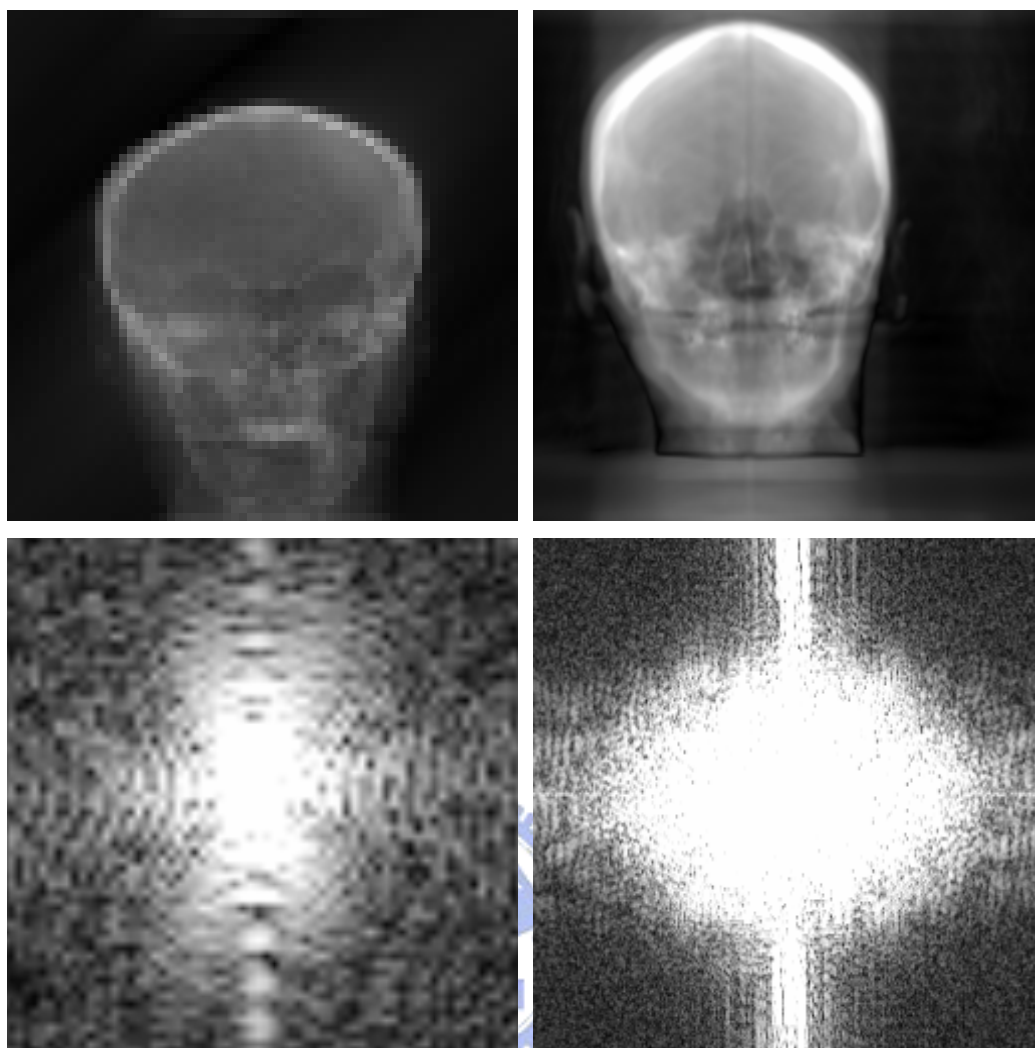


圖 4.3 視角為 90 度的倍角

4.2.2 濾波器的實作

我們主要以三線性濾波器和三次曲線濾波器為實作對象，而 Viola [26]及 Lichtenbelt [22]也提出對這兩種濾波器的實作方法。其中我們對三次曲線濾波器稍做修改，以合併三線性濾波器的方式來實作，以下為這兩種濾波器的實作過程：

1. 三線性濾波器

我們根據第二章所介紹三線性濾波器的原理以 Pixel shader 來實作，其程式的流程會是這樣：

- (1) 取得經過內插後的 3 維貼圖座標，此內插的動作是由 GPU 負責。
- (2) 由於貼圖座標的每個值都是介於 0 與 1 之間，將它們乘上立體資料的實

際大小後，取 ceil 及 floor 的整數值就會是八個取樣點的座標，小數部份的值就會是內插的比例。舉例來說明，現有一 3 維貼圖座標為 $[u, v, w] = [0.123, 0.713, 0.254]$ ，立體資料的實際大小為 $x \times y \times z = 256 \times 256 \times 256$ ，則 $[u \times x, v \times y, w \times z] = [31.488, 182.528, 65.024]$ ，其 ceil 值會是 $[32, 183, 66]$ ，floor 值會是 $[31, 182, 65]$ ，因為所需要的八個取樣點是一個立方體，而這八個貼圖座標的最大值就會 $\left[\frac{32}{256}, \frac{183}{256}, \frac{66}{256} \right]$ ，最小值就會是 $\left[\frac{31}{256}, \frac{182}{256}, \frac{65}{256} \right]$ 。那麼內插的比例就會是取小數點的部份 $[0.488, 0.528, 0.024]$ 。

(3) 依序計算 x 軸， y 軸， z 軸的內插值。

由於每次都會取樣八個點，即 fetch texture 八次，而 fetch texture 的動作對 GPU 來說是頗為費時的動作，所以加上三線性濾波器後整體效能會明顯下降，其數據如表 4.2 所示：

平台 A			
資料大小 實驗項目	64^3	128^3	256^3
Pre-processing(sec.)	0.704	7.656	111.266
Frame rate(FPS)	32.25	21.73	12.82

平台 B			
資料大小 實驗項目	64^3	128^3	256^3
Pre-processing(sec.)	0.25	2.578	35.688
Frame rate(FPS)	66.67	62.5	21.27

表 4.2 使用三線性濾波器

其結果的影像如圖 4.4 所示，左邊為 64^3 ，右邊為 256^3 ，下方為各別的頻譜取樣結果：

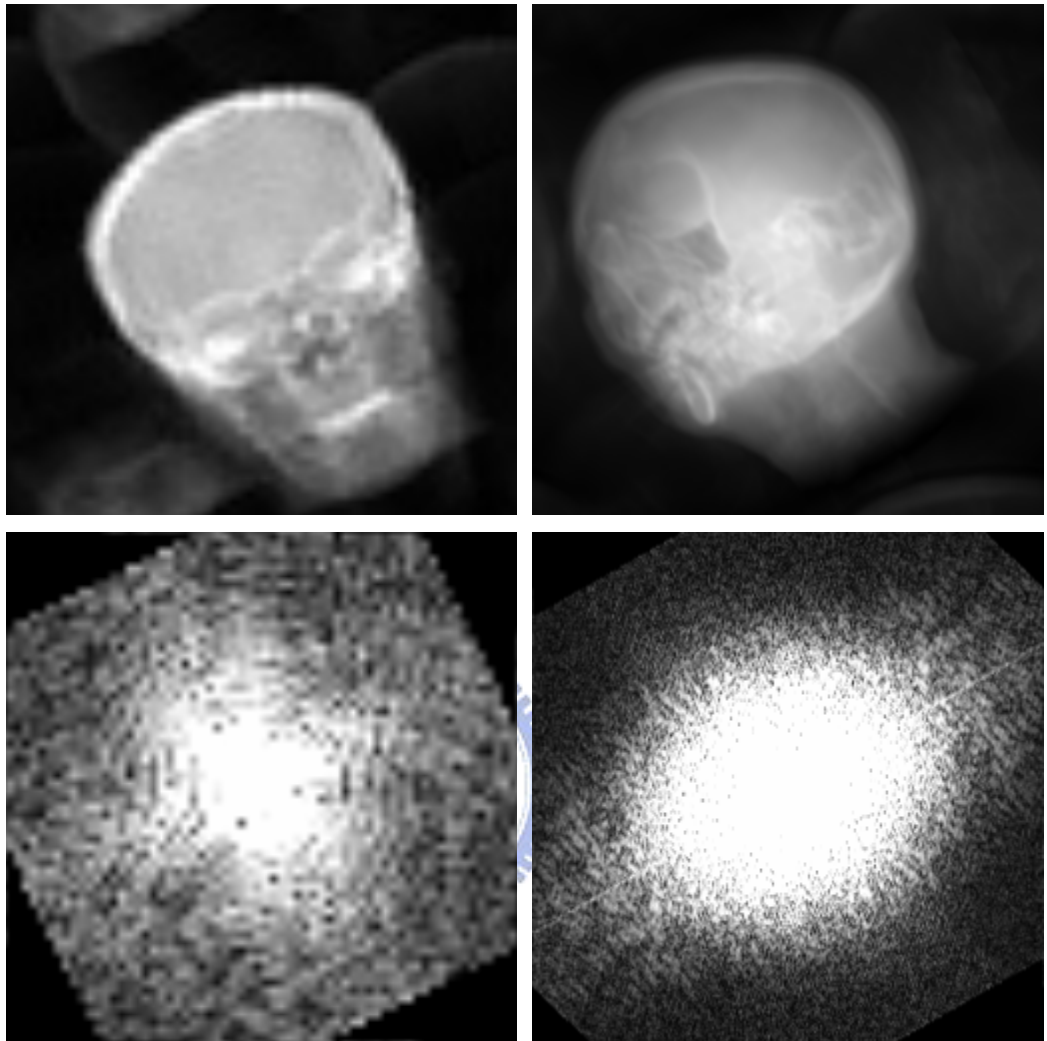


圖 4.4 使用三線性濾波器

從結果得知，雖然仍有重疊影像的情形，但比完全不使用濾波器時減少許多，影像的辨識度亦提高了許多。可是會有中間部分的強度較為明亮，而越往外面越衰減的現象，這在 2.3.5 節有提到過，必須在原始的立體資料進行補償的動作，我們以 $\text{sinc}^2(x)$ 來做為補償函數，結果如圖 4.5 所示，請與圖 4.4 的右邊為 256^3 來做比較：

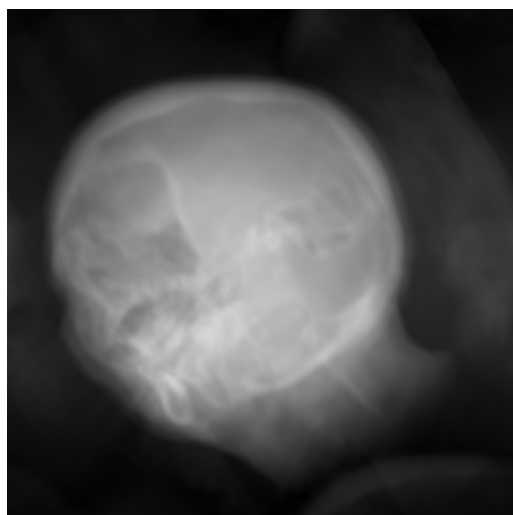


圖 4.5 三線性濾波器時域空間衰減補償

2. 三次曲線濾波器

為了能在 GPU 運作，且能取得較為正確的取樣值，我們對三次曲線濾波器稍微做了修改。首先，我們將整個濾波動作分為三大部份，也就是分為 x , y , z 軸三部份，這是因為三次曲線濾波器總共需取樣 $4^3=64$ 個取樣點，以我們的實驗平台是無法讓 GPU 一次進行那麼多的 fetch texture 動作，所以我們必須要分擔讀取 Video RAM 的動作。如此一來每次進行曲線內插只需 fetch texture 四次。因此我們必須要寫出 21 個不同的 Pixel shader 來進行曲線內插的動作，其中進行 x 軸需執行 16 次， y 軸 4 次， z 軸 1 次。

但是每次曲線內插需要四個取樣點，我們若以偏移貼圖座標來直接取得的話，效果並不是很好，因為仍然會有再取樣的問題。所以針對這四個取樣點，我們改以三線性內插來取得。如圖 4.6 所示，黑色點為由 GPU 提供的貼圖座標位置，紅色點為對 x 軸各左右偏移一個及兩個單位的貼圖座標位置，然後我們以三線性內插來取得每個紅色點的內插值，之後再以 Catmull-Rom 曲線內插方式取得藍色點的值。

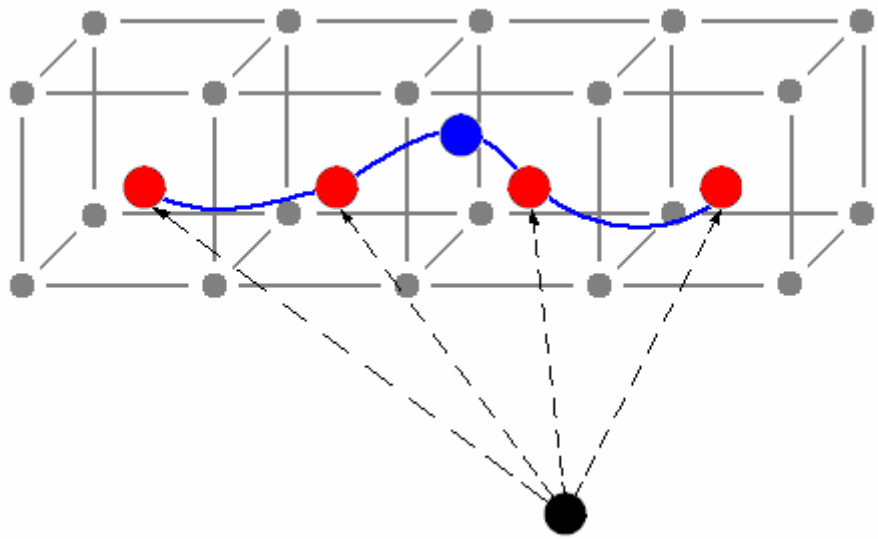


圖 4.6 三次曲線內插

其數據如表 4.3 所示：

平台 A			
資料大小	64 ³	128 ³	256 ³
Pre-processing(sec.)	0.828	7.485	100
Frame rate(FPS)	2.67	2.132	1.27

平台 B			
資料大小	64 ³	128 ³	256 ³
Pre-processing(sec.)	0.25	3.641	37.984
Frame rate(FPS)	2.78	2.66	2.54

表 4.3 使用三次曲線濾波器

其結果的影像如圖 4.7 所示，左邊為 64³，右邊為 256³，下方為各別的頻譜取樣結果：

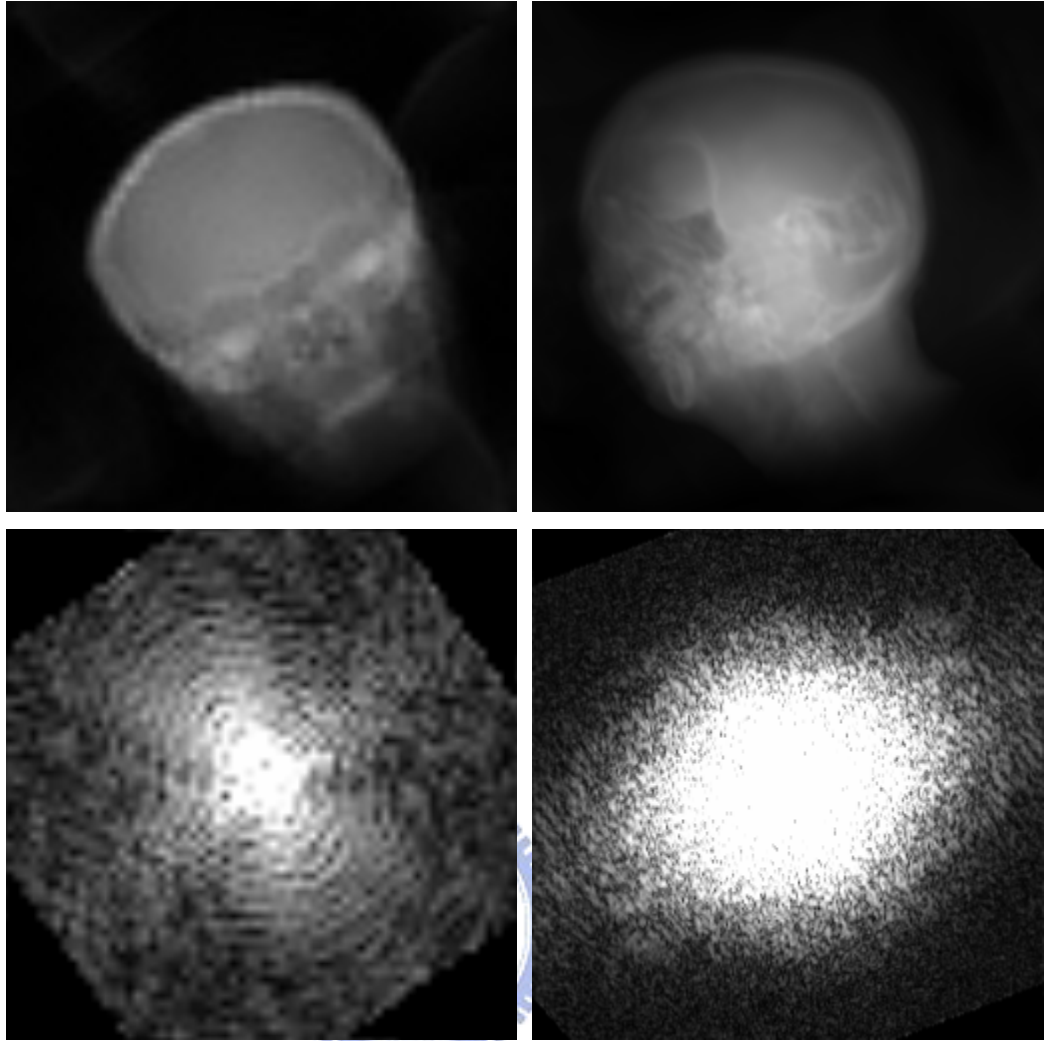


圖 4.7 使用三次曲線濾波器

從結果得知，重疊影像幾乎完全消除了，但是外圍衰減的現象卻非常嚴重，因此必須在原始的立體資料進行補償的動作，結果如圖 4.8 所示：

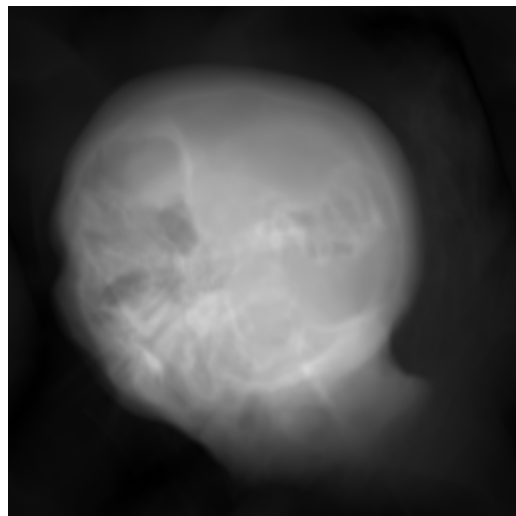


圖 4.8 三次曲線濾波器時域空間衰減補償

3. 三線性濾波器與時域空間延伸

由上述的實驗結果來分析，三次曲線濾波器的效果最好，但是所需的時間成本也是最大的。以我們的實驗平台來運作，實在無法有即時顯像的效果。在 2.3.1 節我們有提到在時域空間延伸的作法可以增加取樣頻率的能力，雖然會造成記憶體空間上的負擔，但我們發現若配合三線性濾波器會有更好的效果，其數據如表 4.4 所示：

平台 A		
資料大小 實驗項目	64 ³	128 ³
Pre-processing(sec.)	7.594	101
Frame rate(FPS)	31.25	12.82

平台 B		
資料大小 實驗項目	64 ³	128 ³
Pre-processing(sec.)	3.708	35.031
Frame rate(FPS)	66.67	31.25

表 4.4 三線性濾波器與時域空間延伸

其結果的影像如圖 4.9 所示，左邊為 64³，右邊為 128³，下方為各別的頻譜取樣結果：

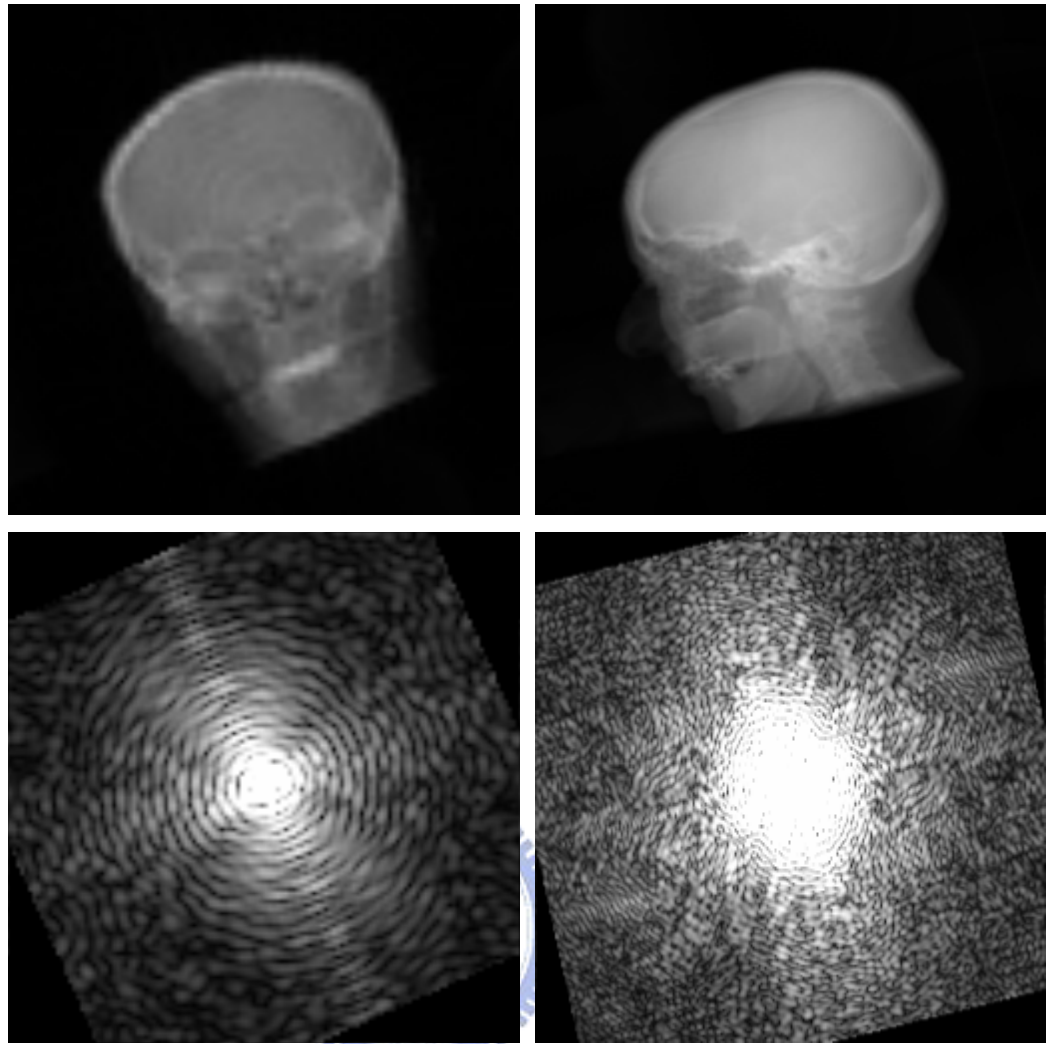


圖 4.9 三線性濾波器與時域空間延伸

4.2.3 以 GPU 實作 FFT

我們以兩種 FFT 的實作方式來進行實驗，一是 FFTW，另一是 FFT on GPU。其中 FFT on GPU 我們是以 DIF FFT 來實作，它的原理與標準的 Butterfly FFT 類似，最大不同是在進行 bit reverse 的動作是在最後的頻域空間進行，使得 Butterfly operation 的加減動作可明確的分開，亦可提升在 GPU 運算的速度。如圖 4.10 是一個對八筆資料進行 DIF FFT 的範例。

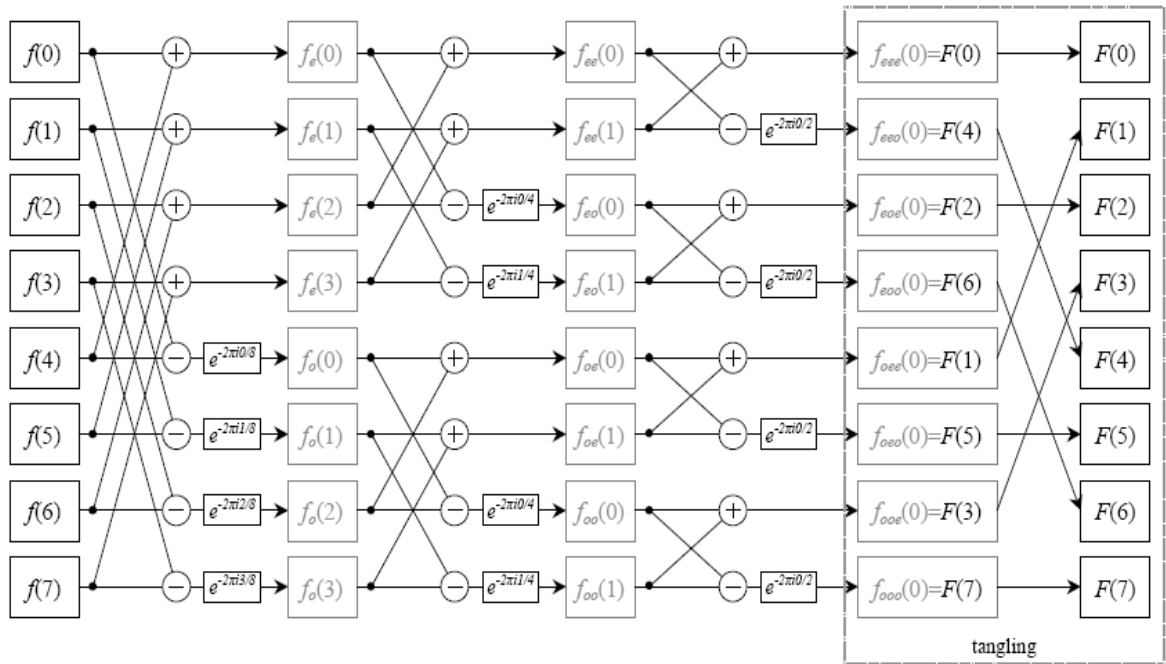


圖 4.10 DIF FFT

資料來源: T. Jansen, et al. “Fourier Volume Rendering on the GPU using a Split-Stream-FFT”



我們對每個 Butterfly operation 寫兩種 Pixel shader，分別為加法及減法。每一回合進行一次的繪圖，共需繪圖 $\log N$ 次。每次繪圖為了能讓每筆資料都能準確的進行運算，必須將原始資料以 texture 形式輸入，然後將 viewport 的大小設成與輸入的資料大小一致，然後畫出一個也是與輸入的資料大小一致的四邊形，如此 framebuffer 就會完全地填滿，四邊形內的每個 pixel 都會一對一的與輸入資料對應。為了讓執行速度再加快，我們將輸入的資料除了原始資料外，還有一張儲存 $e^{-2\pi i k l / 8}$ 的表格，因為當資料大小已知且固定的情況下，我們可以事先計算好以面對不同的資料內容，而 bit reverse 也是可以事先建立一張表格，讓查表的時間來取代計算的時間，讓整體速度可以再加快一些。待所有的運算完成後，最後 framebuffer 的內容就是最後結果，但我們不直接讀出，而是以 FBO 可以當成 texture 的特性，直接輸出在螢幕上。

我們以上述兩種不同方式來進行 2 維的反傅立葉轉換，並使用三線性濾波器，其 frame rate 的比較結果如表 4.5 所示：

平台 B			
資料大小 實驗項目	128 ³	256 ³	512*512*256
FFTW	62.5	21.276	8
FFT on GPU	21.132	21.132	16.129

表 4.5 FFT 的比較

由實驗結果發現，FFT on GPU 需要在資料量夠大的情況下才能突顯它的效果，否則在資料為 256³ 以下，以 CPU 來運算的 FFTW 效能是會比較好。

4.3 貝茲曲線轉換函數的實作

根據我們在 3.2 節所提出的貝茲曲線轉換函數著色模型，必須將原始的立體資料複製數份，各別乘上對應的係數後，再進行 3 維的傅立葉轉換。在我們的實驗過程中發現，控制點數量為六的貝茲曲線，就能夠產生出不錯的轉換函數。因此，我們最多會將原始的立體資料複製六份，而整體的速度也會降低約六倍之多。控制點數量為六的貝茲曲線方程式如下：

$$B(u) = P_0(1-u)^5 + 5P_1(1-u)^4u + 10P_2(1-u)^3u^2 + 10P_3(1-u)^2u^3 + 5P_4(1-u)u^4 + P_5u^5 \quad (35)$$

將式(35)代入式(25)：

$$\begin{aligned}
I = & P_0 \Pi_v (f(x)(1-f(x))^5) + \\
& P_1 \Pi_v (5f(x)(1-u)^4 u) + \\
& P_2 \Pi_v (10f(x)(1-f(x))^3 f(x)^2) + \\
& P_3 \Pi_v (10f(x)(1-f(x))^2 f(x)^3) + \\
& P_4 \Pi_v (5f(x)(1-f(x)) f(x)^4) + \\
& P_5 \Pi_v (f(x)f(x)^5)
\end{aligned}
\tag{36}$$

式(36)即為我們將實作的著色模型。

4.3.1 使用者介面設計

為了讓使用者能容易且準確地產生出想要的轉換函數曲線，我們以 Visual .NET C++ 2005 設計了一個使用者介面視窗，如圖 4.11 所示，圖中橫軸為 voxel 值，介於 0 與 1 之間；縱軸為 weight 值，介於 0 與 2 之間。黃色曲線即為轉換函數曲線，我們將曲線平分 6 段，每一段各由一個控制點來控制，當使用者將滑鼠游標移至該線段的區域時，就會出現藍框來標示，這時按下滑鼠左鍵並同時上下移動，就可變更線段的曲度。紅色直線代表滑鼠游標在曲線上的位置，左上角的數字代表目前的 voxel 所對應的 weight 是多少。另外，我們也可以改變控制點的數量，減少控制點可以加速顯像的速度，但相對的轉換曲線會較為不準確。

我們所設計的使用者介面具有即時變更的效果，也就是變更曲線的同時，FVR 的投影結果亦會以此轉換函數來產生。圖 4.12 為整個 FVR 應用程式的畫面。

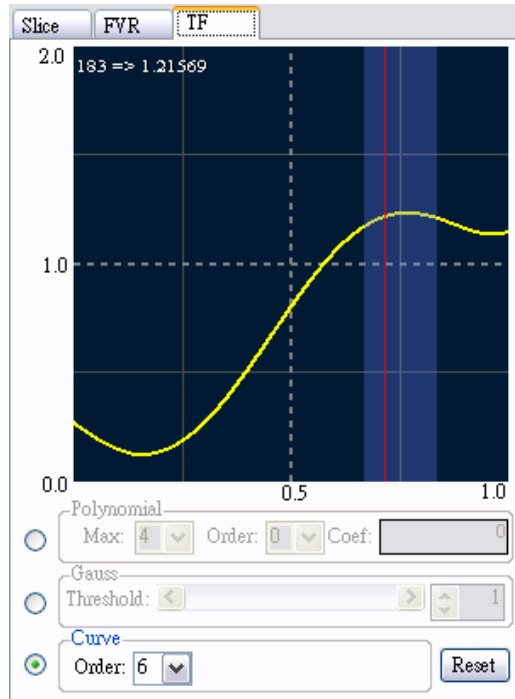


圖 4.11 轉換函數使用者介面

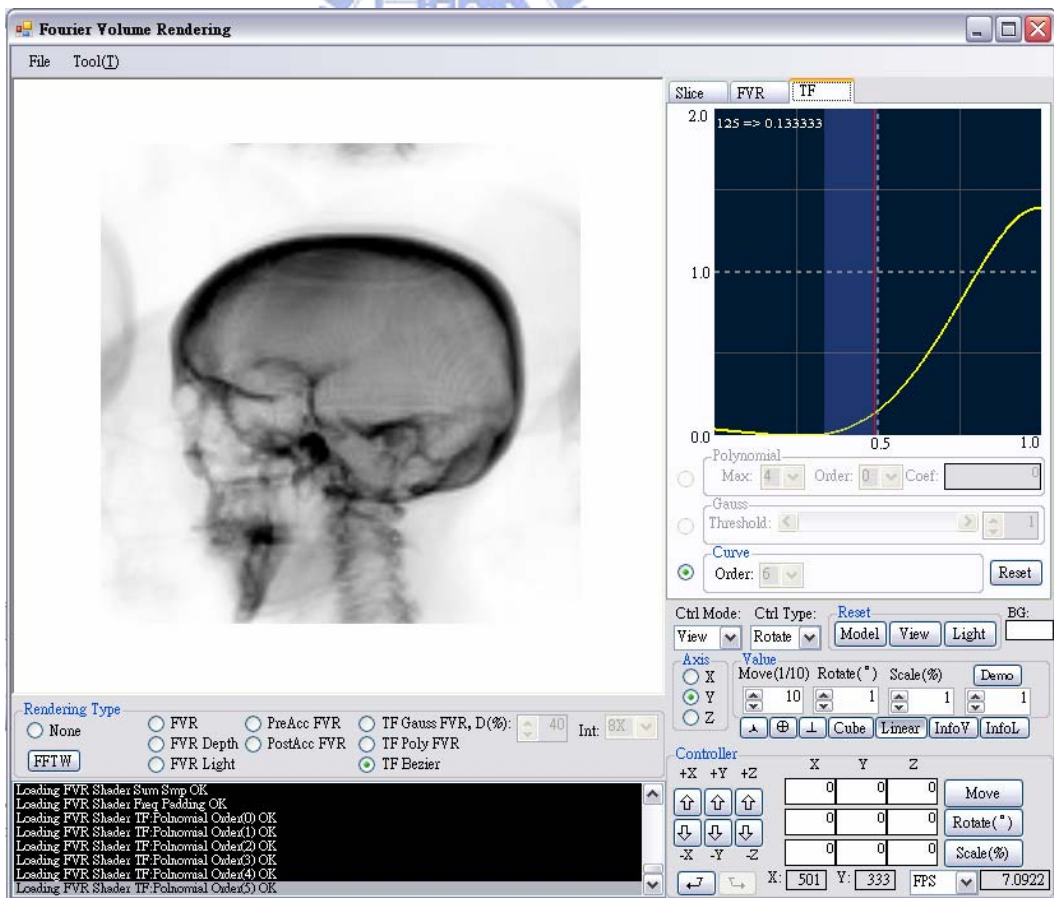


圖 4.12 FVR 應用程式

4.3.2 分類結果

我們以 6 個控制點的貝茲曲線及加上三線性濾波器來進行實驗，所得到的實驗數據如表 4.6 所示：

平台 A			
資料大小 實驗項目	64 ³	128 ³	256 ³
Pre-processing(sec.)	4.234	44.485	640.016
Frame rate(FPS)	7.09	7.09	0.39

平台 B			
資料大小 實驗項目	64 ³	128 ³	256 ³
Pre-processing(sec.)	1.172	26.375	37.984
Frame rate(FPS)	9.174	8	0.57

表 4.6 貝茲曲線轉換函數

圖 4.13，4.14，4.15 分別為對資料大小是 128^3 的頭部 CT 立體資料進行分類顯像的結果：

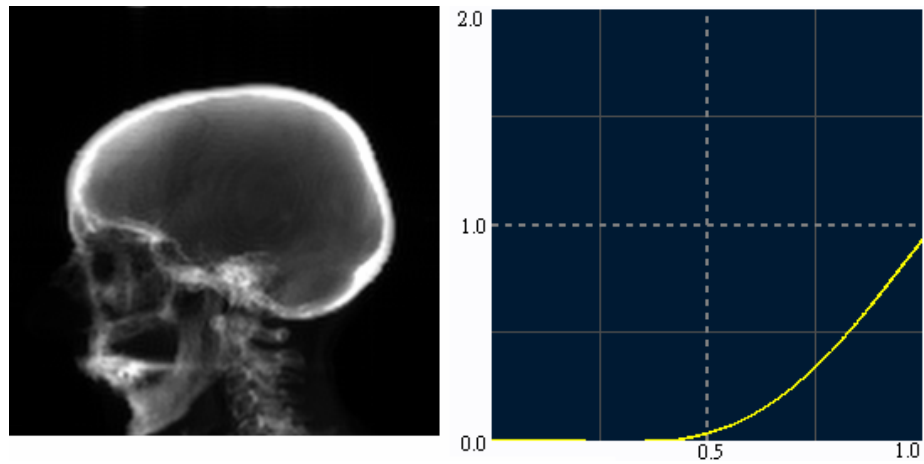


圖 4.13 頭部骨骼部份

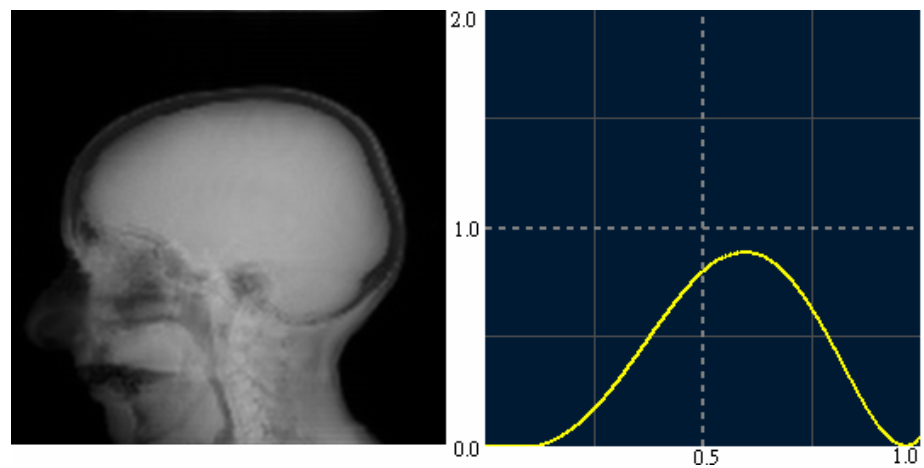


圖 4.14 頭部肌肉部份

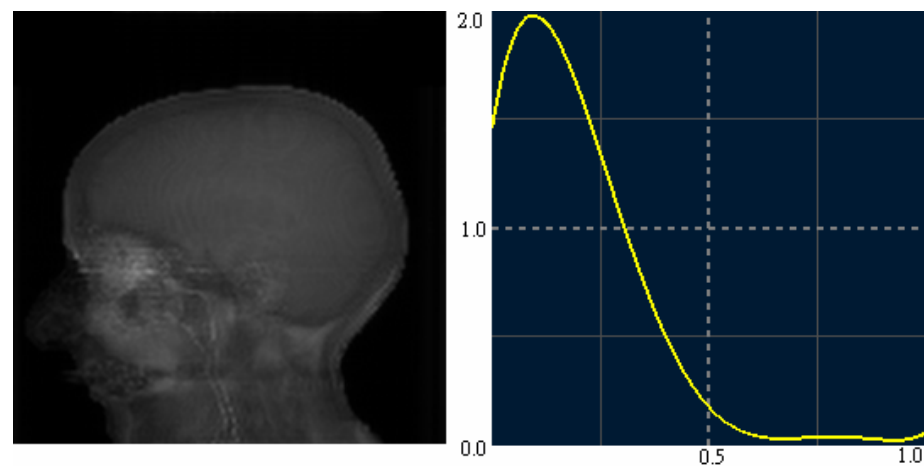


圖 4.15 頭部液體部份

圖 4.16, 4.17 分別為對資料大小是 128^3 的盆栽 CT 立體資料進行分類顯像的結果：

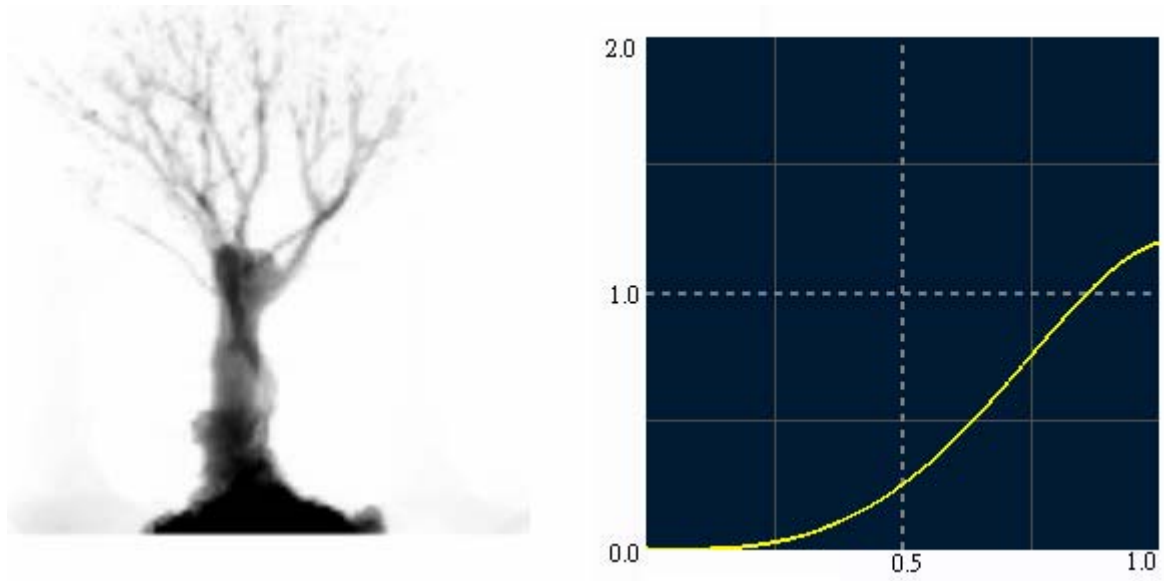


圖 4.16 盆栽樹枝部份

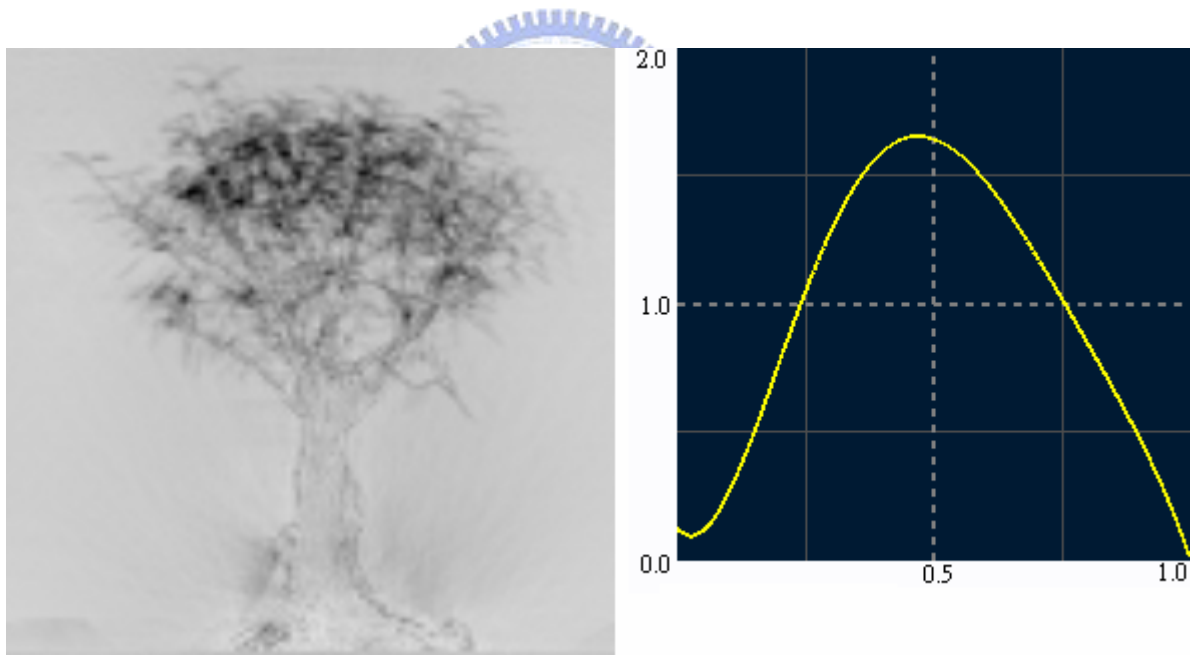


圖 4.17 盆栽樹葉部份

圖 4.18, 4.19, 4.20 分別為對資料大小是 128^3 的胸腔 CT 立體資料進行分類顯像的結果：

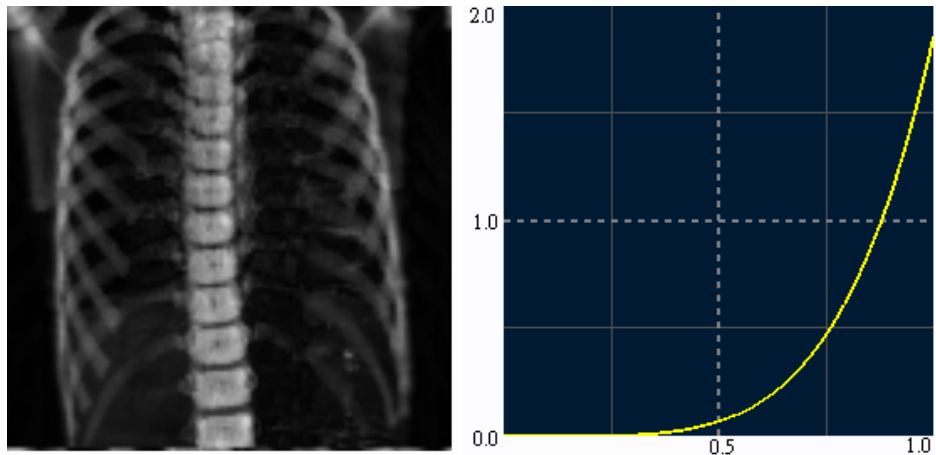


圖 4.18 胸腔骨骼部份

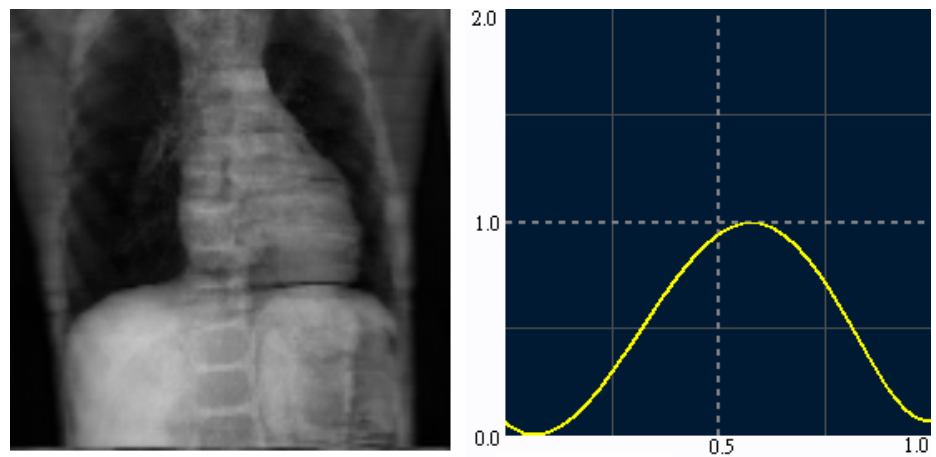


圖 4.19 胸腔肌肉部份

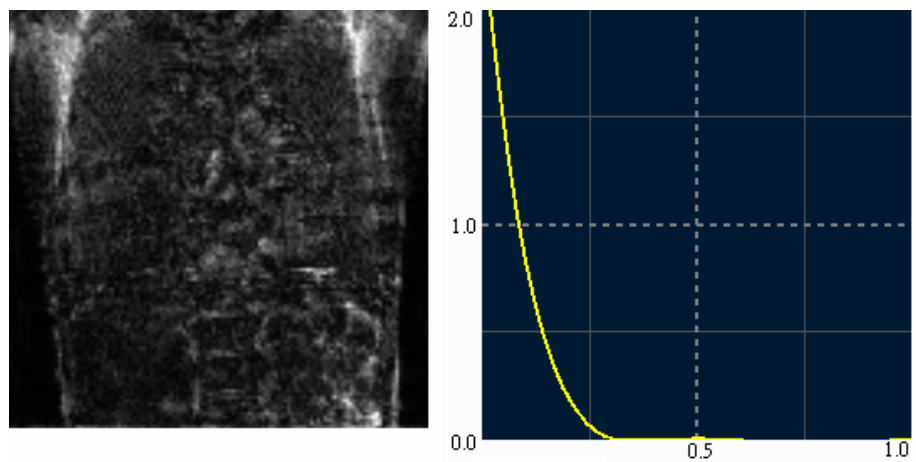


圖 4.20 胸腔脂肪部份

五、結論與未來展望

本論文對 FVR 做了一個完整的研究，並分析了 FVR 因再取樣的問題而導致重疊影像的發生，也實作了許多濾波器來進行效能與品質上的比較。我們發現將三線性濾波器與三次曲線濾波器的合併使用可以有效的消除重疊影像，但是卻帶來龐大的計算負擔。我們也發現將原始的立體資料在空間上延伸兩倍，然後配合計算負擔較少的三線性濾波器會有更好的效果，只是在記憶體空間上會多出一倍。

我們也設計了兩種進行分類影像的方式，一是預先影像處理，另一是貝茲曲線轉換函數。這兩者皆是利用傅立葉轉換具有線性組合的特性來設計，雖然 Levoy [9] [10] 已經發現這樣的特性並且設計了 FVR 的著色模型，但卻尚未提出比較有效的線性組合方程式來進行分類顯像。Nagy [13] 雖然提出一個將非線性的步進函數以線性組合方式來進行分類顯像，但分類的限制過大。比較之下，我們所提出的貝茲曲線轉換函數較為有效且容易實作，使用者也可以很容易地產生出想要進行分類的轉換函數，並且能夠即時地描繪出投影結果。

FVR 尚有許多問題需要克服，在時間上我們希望能找出更經濟且不影響品質的濾波器，因為根據我們的實驗結果，取樣濾波的動作在整體運算時間中所佔的比重是最大，而不是 2 維的反傅立葉轉換。空間上也是急需克服的問題，我們所提出的轉換函數會使用不少的記憶體空間，Hartley transform [27] 可以稍微解決這個問題，它可讓我們減少一半的記憶體空間，以及 Westenberg [28] 提出利用小波轉換(Wavelet transform)來達成 Level-of-detail。因此，我們會利用這兩項技術來改善記憶體空間的問題。

另外，我們除了期盼 GPU 的發展，能夠有更強大的功能來支援 FVR，包括效能、記憶體空間、記憶體的存取速度…等，也希望能自行設計支援 FVR 演算法的硬體，針對上述的問題加以克服。

參 考 文 獻

- [1] W. E. Lorensen, H. E. Cline, “Marching Cubes: A high resolution 3D surface construction algorithm”, Proceedings of ACM SIGGRAPH Computer Graphics '87, Vol. 21, pp. 163-169, Anaheim, California, USA, July 1987.
- [2] M. Ikits, J. Kniss, A. Lefohn, C. Hansen, “Volume Rendering Techniques”, GPU Gems, chapter 39, pp. 667-692, Addison Wesley, New York, March 2004.
- [3] C. C. Lin, Y. T. Ching, “A note on computing the saddle values in isosurface polygonization”, The Visual Computer, vol. 13, no. 7, pp. 342-344, 1997.
- [4] S. Hill, “Tri-linear Interpolation”, Graphics Gems IV, chapter X.1, pp. 521-524, Morgan Kaufmann, San Francisco, January 1994.
- [5] N. Max, “Optical Models for Direct Volume Rendering”, IEEE Transactions on Visualization and Computer Graphics, Vol. 1 , Issue 2, pp. 99-108, June 1995.
- [6] S. Dunne, S. Napel, B. Rutt, “Fast Reprojection of Volume Data”, Conference on Visualization in Biomedical Computing '90, pp. 11-18,

- Atlanta, Georgia, USA , May 1990.
- [7] T. Malzbender, “Fourier Volume Rendering”, ACM Transactions on Graphics, Vol. 12, Issue 3, pp. 233–250, July 1993.
- [8] M. Levoy, “Display of surfaces from volume data”, IEEE Computer Graphics and Applications, Vol. 8, Issue 3, pp. 29-37, May 1988.
- [9] M. Levoy, “Volume Rendering using the Fourier Projection-Slice Theorem”, Proceedings of Graphics Interface '92, pp. 61-69, Vancouver, B.C. Canada, May 1992.
- [10] T. Totsuka, M. Levoy, “Frequency Domain Volume Rendering”, SIGGRAPH '93: Proceedings of the 20th annual conference on Computer graphics and interactive techniques, pp. 271-278, Anaheim, California, USA, August 1993.
- [11] K. Engel, M. Kraus, T. Ertl, “High-quality pre-integrated volume rendering using hardware-accelerated pixel shading”, Proceedings of Eurographics/SIGGRAPH Workshop on Graphics Hardware, pp. 9-16, Los Angeles, California, USA, August 2001.
- [12] E. B. Lum, B. Wilson, K. L. Ma, “High-Quality Lighting and Efficient Pre-Integration for Volume Rendering”, Proceedings of the Joint EUROGRAPHICS - IEEE TVCG Symposium on Visualization, pp. 25-34,

Konstanz, Germany, May 2004.

- [13] Z. Nagy, G. Miiller, R. Klein, “Classification for Fourier volume rendering”, Proceedings of the Computer Graphics and Applications, 12th Pacific Conference on (PG'04), Vol. 0, pp. 51-58, Seoul, Korea, October 2004.
- [14] J. W. Cooley, J. W. Tukey, “An algorithm for the machine calculation of complex Fourier series”, Math. Comput. 19, pp. 297-301, 1965.
- [15] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, Introduction to Algorithms, 2nd edition, The MIT Press, Cambridge , Massachusetts London, England, 2001
- [16] T. Jansen, B. von Rymon-Lipinski, N. Hanssen, E. Keeve, “Fourier Volume Rendering on the GPU using a Split-Stream-FFT”, Vision Modeling and Visualization (VMV) 2004, 9th International Fall Workshop, pp. 395–403, Stanford California, USA, November 2004.
- [17] M. Frigo, S. G. Johnson, “The Design and Implementation of FFTW3”, Proceedings of the IEEE, Vol. 93, Issue 2, pp. 216–231, February 2005.
- [18] M. J. Harris, G. Coombe, T. Scheuermann, A. Lastra, “Physically-based visual simulation on graphics hardware“, Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware, pp.

- 109-118, Saarbrücken, Germany, September 2002.
- [19] J. Krüger, R. Westermann, “Linear algebra operators for GPU implementation of numerical algorithms”, SIGGRAPH 2003: International Conference on Computer Graphics and Interactive Techniques, pp. 908-916, San Diego, California, USA, July 2003.
- [20] K. Moreland, E. Angel, “The FFT on a GPU”, Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware, pp. 112–119, San Diego, California, July 2003.
- [21] R. N. Bracewell, The Fourier Transform and Its Applications, 3rd edition, McGraw-Hill, New York, June 1999.
- [22] B. B. A. Lichtenbelt, “Fourier Volume Rendering”, HP Labs Technical Reports, November 1995.
- [23] L. K. Arata, “Tricubic Interpolation”, Graphics Gems V, chapter III.3, pp. 107-109, Morgan Kaufmann, San Francisco, January 1995.
- [24] R. J. Rost, OpenGL Shading Language, 2nd edition, Addison Wesley, New York, January 2006.
- [25] E. Persson, Framebuffer Objects, ATI Technologies, Inc.
- [26] I. Viola, A. Kanitsar, M. E. Gröller, “GPU-based frequency domain volume rendering”, Proceedings of the 20th spring conference on

- Computer graphics, pp. 55-64, Budmerice, Slovakia, April 2004.
- [27] T. Theußl, R. F. Tobler, E. Gröller, “The Multi-Dimensional Hartley Transform as a Basis for Volume Rendering”, The 8-th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision '2000, vol. 1, pp. 132–139, Plzen - Bory, Czech Republic, February 2000.
- [28] M. A. Westenberg, J. B. T. M. Roerdink, “Frequency Domain Volume Rendering by the Wavelet X-ray Transform”, IEEE transactions on image processing, Vol. 9, Issue 7, pp. 1249-1261. July 2000.
- [29] A. Entezari, R. Scoggins, T. Moller, R. Machiraju, “Shading for Fourier Volume Rendering”, Proceedings of the 2002 IEEE symposium on Volume visualization and graphics, pp. 131- 138, Boston, Massachusetts, USA, October 2002.
- [30] M. Artner, T. Möller, I. Viola, M. E. Gröller, “High-Quality Volume Rendering with Resampling in the Frequency Domain”, Proceedings of EuroVis, pp. 85-92, Leeds, UK, June 2005.