# 貝氏模型的浮水印方法
# Robust Watermarking of Bézier Based Models

鄭進和， 康竹裕， 陳柏青
Chin-Ho Cheng, Jwu-Yuh Kang, and Po-Ching Chen

輔仁大學資訊工程系
Department of Computer Science and Information Engineering
Fu Jen Catholic University, Hsinchuang, Taipei 24205, Taiwan, R.O.C.
Email:{chcheng, juyu88, poching88}@csie.fju.edu.tw

## 摘 要

近年來由於網際網路的盛行，網路的頻寬也隨著人們的需求，逐漸的增大，因此可以傳輸的檔案也可以越來越大，慢慢也衍伸出一些問題，例如智慧財產權的問題等等。像是越來越多的 CAD-based 3D 資料進入 World Wide Web，主要如 Virtual Reality Modeling Language (VRML) scenes。有些人會引用或盜用別人這些場景，而這些場景可能是人家努力許久所創作出來，然而盜用的人卻不用花太多的心力，便可以把這些場景複製下來，這樣子對這些創作人是一件非常不公平的一件事，這樣子便沒有人會想要再花心思去創作了！於是開始有人研究浮水印的技術，來保護原始創作人的智慧財產權。本篇論文主要是針對貝司曲面（Bézier Surface）所構成的模型探討如何藏入浮水印，而且還要避免浮水印遭受到外力的破壞。因為一旦受到破壞，就有可能無法驗證出這個作品是否為我們所有，因此別人就可以輕易的竊取我們的作品了，這個藏入的方法就顯得非常的重要。同時也考量到萃取浮水印的方法要簡單容易。

關鍵字：computer graphics，Bézier surface, 浮水印, 3D 立體模型

## 1. 簡介

Watermarking 已經發展了有一段的時間，大部分研究的焦點主要放在多媒體上，像是影像，聲音或是圖片上。近來網際網路的盛行，許多網頁上也有多媒體的效果出現，像是 VRML(Virtual Reality Modeling Language)的場景、3D 的動畫或是 3D 的立體模型等等，由於這些生動逼真的作品產生，漸漸就有一些著作權的問題產生。不管公司要賣出的產品，或是個人寫出的作品，都是會面臨到著作權的問題，大家也開始盡力保護自己的作品，當然也要避免抄襲到別人的東西。

浮水印已經發展了有一段的時間，大部分研究的焦點主要放在多媒體上，像是影像，聲音或是圖片上，但這些浮水印技術卻無法運用在 3D 立體模型上。在 1997 年 R. Ohbuchi, H. Masuda, 和 M. Aono[9]三位學者首度提出在 3D 立體模型上加浮水印與萃取（retrieval）浮水印的方法，從此至今有許多的學者專家投入這個領域的研究，而他們所提出的方法大多是在三角片所組成的 3D 立體模型上，利用點、線、或者是由 3D 立體模型中的多邊形(polygons)拓樸的關係，來動一些手腳以達到藏入浮水印的目的。

由於以三角片所組成的 3D 立體模型，其三角片的數量相當大而且在執行藏入浮水印或萃取浮水印需花相當時間，因此本研究的目的是在由一些 Bézier surfaces [5] 所組成的 3D 立體模型上，研究藏入浮水印和萃取浮水印的方法。使用 Bézier surfaces 的目的在於每片 Bézier surface 是由 16 點控制點所描述，因此描述整個 3D 立體模型的資料量大大的減少，而相鄰的兩片 Bézier surfaces 有一些幾何性質存在才能維持 3D 立體模型邊界面的平滑性，可以利用這些特有性質來藏入浮水印，同理也可以用類似的方法來萃取浮水印。同時，我們的浮水印方法要能夠抵抗攻擊者的攻擊，例如：平移、旋轉、或放大縮小，也就是說當模型遭受到平移、旋轉、或放大縮小時，3D 立體模型的位置、方位及大小改變以後，仍然可以正確的取出浮水印，來辯識我們的模型，防止遭受外力的破壞。另外 Bézier surfaces 還有一個與多邊形面不一樣的動作，那就是 subdivision，可以將一片 Bézier surface 分割成四小片 Bézier surfaces，而每一小片 Bézier surface 分別由 16 點控制點所描述。如果攻擊者用 subdivision 動作來攻擊，把某片 Bézier surface 做一個層次或多個層次 subdivision，這樣一來 3D 立體模型的資料已經被改變了，但外型、位置、方位及大小都沒改變，我們的浮水印演算法必須能夠把這些被 subdivision 後所得到的 Bézier surfaces 給合併起來，還原成原來的 Bézier surface，接著萃取出浮水印，來辯識我們的模型。如果攻擊形動更嚴重者---像是對一片或是多片 Bézier surfaces 做多層次不均勻的 subdivision，我們依然必須能夠萃取出浮水印，來辯識我們的模型。

論文架構如下：第二章將過去相關的研究

狀況；第三章為主文，將詳細的解說如何在 Bézier based models 上，利用 Affine Invariant Norm 的方法加入浮水印的演算法，並說明浮水印萃取相關事項，第四章則實地拿模型測試、展示實驗結果，最後以第五章做為結論並說明。

## 2．過去相關的研究

關於 Bézier 3D 立體模型的浮水印研究，至今尚未被研究過。至於其它方面的 3D 立體模型的浮水印研究，其作法最主要有下面兩大類方法：(1) 浮水印藏在幾何基本量上的演算法；(2) 浮水印藏在拓樸上的演算法。上述兩類方法將在後面分別說明。

### 2.1 浮水印藏在幾何基本量上的演算法

這類的浮水印方法就是將浮水印藏在 3D 立體模型的點座標、線段、多邊形，和多面體上面。代表性的浮水印方法是 Vertex Flood Algorithm [3, 4]、Normal Bin Encoding Algorithm [1, 2, 4]、Triangle Similarity Quadruple Embedding Algorithm [9, 10, 11]、Tetrahedral Volume Ratio Embedding Algorithm [9, 10, 11]、Yeung and Yeo's algorithm [15]、Progressive Mesh algorithm [13] 等方法。

### 2.2 浮水印藏在拓樸上的演算法

這類的浮水印方法，代表性的有 triangle strip peeling symbol sequence algorithm[9, 11]、polygon stencil pattern algorithm [9, 11] 等方法。

## 3．Affine Invariant Watermarking

我們將會利用 Affine Invariant Norm 這個概念來藏入浮水印，同時也將點分成好幾群之後，再把浮水印放入各個不同的群組中，而且還會加上多重藏入浮水印，以免被人輕易的破壞。接下來的章節將敘述 Affine Invariant Norm。

### 3.1 Affine Invariant Norm

如果使用歐基理得向量（Euclidean norm），來作為這個演算法的依據的話，容易受到別人的蓄意或是小心的破壞，例如將模型任意的旋轉、均勻或不均均的放大縮小，這樣子都很容易使 watermark 受到破壞，也就是所謂的脆弱 watermarking 。因此，我們將歐基理得向量改用 Affine Invariant norm [8] 來替代。這樣子就可以解決當模型受到別人旋轉、均勻或不均均的放大縮小時，而取不出浮水印的困境。

根據 Affine Invariant norm [8] 的定義，需要一組特徵點 $C=\{v_0, v_1, .., v_n\}$ 其中 $v_i$ 的座標是 $(x_i, y_i, z_i)$，然後可以得到這組特徵點的重心 com 如下：

$$\text{com}=(\bar{x}, \bar{y}, \bar{z})=\frac{1}{n+1}\sum_{i=0}^{n} V_i \quad \text{.....(1)}$$

亦即 $\bar{x}=\dfrac{\sum_{i=0}^{n} x_i}{n+1}$，$\bar{y}=\dfrac{\sum_{i=0}^{n} y_i}{n+1}$，$\bar{z}=\dfrac{\sum_{i=0}^{n} z_i}{n+1}$

接著定義一個矩陣

$$V=\begin{pmatrix} x_0-\bar{x} & y_0-\bar{y} & z_0-\bar{z} \\ x_1-\bar{x} & y_1-\bar{y} & z_1-\bar{z} \\ \cdots & \cdots & \cdots \\ x_n-\bar{x} & y_n-\bar{y} & z_n-\bar{z} \end{pmatrix}$$

令 $A=(n+1)(V^{T}V)^{-1}$

空間任一點 $P$ $(x, y, z)$，它相對於特徵點 C 的 Affine Invariant norm $\|P\|^2_c$ 定義如下：

$$\|P\|^2_c=\begin{pmatrix} x, & y, & z \end{pmatrix} A \begin{pmatrix} x \\ y \\ z \end{pmatrix},$$

$$P=\begin{pmatrix} x, & y, & z \end{pmatrix}$$

$$=P\ A\ P^{T}. \quad \text{.....(2)}$$

在本篇論文中 embedding primitive 是一片片的 Bézier surface，由此會計算到關於 Bézier surface（如 Figure 3.1）的 Affine Invariant norm。我們定特徵點是該 Bézier surface 16 個控制點的四個角落點，如 Figure 3.1 中的 $v_1, v_2, v_3, v_4$。也就是說特徵點集合 $C=\{v_1, v_2, v_3, v_4\}$，在下一節將證明 Affine Invariant norm 相關性質。

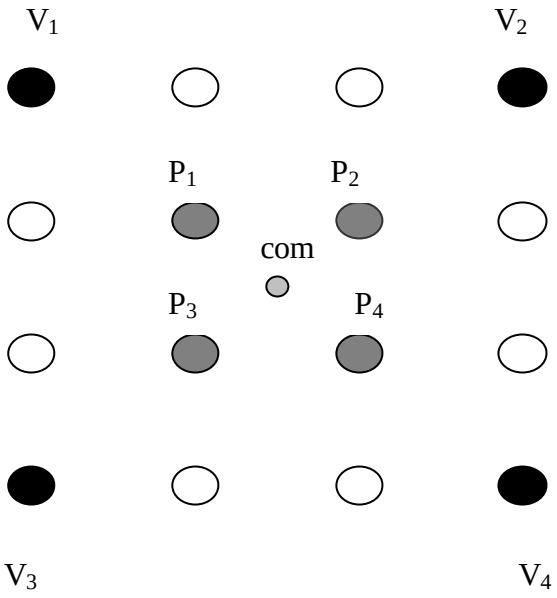



Figure 3.1

## 3.2 Affine Invariant Norm 相關性質

本節中將證明 Affine Invariant norm 可以對抗一些外來的破壞，像是旋轉、平移、或是均勻或不均勻的放大縮小。現在將用一些數學式子來證明這些性質。

假設利用 Affine Invariant norm 的性質，將 embedding 到 $P$ 點中，便可以得到下列的式子：$\|P\|_c^2 = P\ A\ P^T$.假設 $M$ 是一個組合式的矩陣，這個矩陣可能包含了旋轉、平移、或是均勻或不均勻放大縮小的資訊。當整個模型受到 $M$ 矩陣變動後，原來定義的矩陣 $V$ 勢必也會受到影響，必須要再乘上矩陣 $M$，所以會得到新的 $V'$：$V' = V\ M$。因此，$A$ 和 $P$ 都也會受到影響。因此我們可以得到改變後的 $A$ 和 $P$ 分別為 $A'$ 和 $P'$ 如下所示：

$A' = (n+1)\ ((V\ M)^T\ (V\ M))^{-1}$
$= (n+1)\ ((V')^T\ V')^{-1}$
$P' = P\ M$

所以我們可以得到一個新的 Affine Invariant norm 式子：$P'\ A'\ (P')^T$。因為 $A' = (n+1)\ (V'^T\ V')^{-1}$，$P' = P\ M$ 然後再將式子整理一下：

$P'\ A'\ (P')^T$
$= P\ M\ (n+1)\ ((V')^T\ V')^{-1}\ (P\ M)^T$
$= P\ M\ (n+1)\ ((V\ M)^T (V\ M))^{-1}\ (P\ M)^T$
$= P\ M\ (n+1)\ ((V^T M^T)\ (V\ M))^{-1}\ (P\ M)^T$
$= (n+1)\ P\ M\ ((V\ M)^{-1} (M^T V^T)^{-1})\ (P\ M)^T$
$= (n+1)\ P\ M\ M^{-1}\ V^{-1} (V^T)^{-1}\ (M^T)^{-1}\ M^T\ P^T$

(因為 $M\ M^{-1} = I$, $(M^T)^{-1} M^T = I$, I為單位矩陣)

$= (n+1)\ P\ V^{-1} V^{-T} P^T$
$= P\ (n+1)\ (V^T V)^{-1}\ P$
$= P\ A\ P^T$

由上式推導可以得到：

$\|P'\|_c^2 = P'\ A'\ P'^T = P\ A\ P^T = \|P\|_c^2$

所以由上式可以知道，經過改變後的 $\|P'\|_c^2$，還是會等於原來的 $\|P\|_c^2$。因此，當模型受到旋轉、平移、或是均勻或不均勻放大縮小的改變時，Affine Invariant Norm 的值仍然不會受到影響。

## 3.3 Affine Invariant Watermarking

已經介紹完 Affine Invariant Norm 相關的性質，接下來將介紹如何在 Bézier Surface 上使用 Affine Invariant Norm 來藏入浮水印。

首先先從一片 Bézier patch 上的十六個控制點來看（ Figure 3.1)。先挑選最外圍的四個角落點為特徵點 ($V_1$、$V_2$、$V_3$、$V_4$)，之所以會挑選這四點，是因為當模型受到細化的攻擊時，這四個點仍然不會受到改變，適合作特徵點。再利用 3.1 節中的式子（1）求出 com 。

再來我們選擇 Figure 3.1 中的 $P_1$、$P_2$、$P_3$、$P_4$ (Bézier Surface 控制點中不在邊界上的四個點) 作為藏入浮水印的點。因為這四

個點並未在四個邊上，並不會有機會跟其他的 Bézier patch 共用，當 Bézier patch 相鄰時，會共用相鄰四個點。如果選擇了共用的頂點來藏入浮水印時，當在這片 Bézier patch 藏入浮水印後，在相鄰的 Bézier patch 有可能也會對共用的頂點做同樣的事，因此共用點可能會被藏入兩次，會破壞原先所藏入的浮水印。之所以會選擇中間的四點來藏，主要就是想避免這樣的情形產生。將要藏入浮水印的四個點，分別帶入 3.1 節中的式子（2）求出 $\|P\|^2_c$ 的值。

假設一個區間的長度為 w，同時假如要藏入的浮水印的資料 為 n 個 bit (w和n為輸入的參數)。將求出的 $\|P\|^2_c$ 值除以 w，求出來的數值取整數值令為 m ，再將 m 分成 $2^n$ 組，看最後 m 是落在哪一組 （ 以 Figure 3.2 為例, n = 6)。

接下來我們直接舉例來說明如何藏入浮水印。假設要藏入的浮水印長度為 6 bits 其資料是（010000）$_2$，而浮水印的資料轉換為十進位的值為 16。同時假設要將此浮水印資料加在一點 P 上而 Affine Invariant Norm $\|P\|^2_c$ 除以 w 是 100，因為 100 mod $2^6$=36，因此必須把點 P 搬移到一個位置，使得（$\|P\|^2_c$/w) mod $2^6$ =16 （浮水印的資料），而這搬移必須在重心點 com 與點 P 連線上作最短距離的搬移（Figure 3.3)。

以下舉例說明如何加入浮水印資料的編碼方式。為了 embedding data 的方便，我們將 26 個大寫英文字母依序給予 1~26 的值，也就是 A＝1，B＝2，..Z＝26。我們要藏入浮水印為 8 個 bits，前三個 bit 為索引（index)，後五個 bit 為資料（data)。

$I_0$，$I_1$，$I_2$　　$I_3$，$I_4$，$I_5$，$I_6$，$I_7$

⇩　　⇩

Index bits　　Data bits

假設要藏入的 data 有七個字母，所以需要有三個索引位元（0　7），來決定字母的順序，又因為我們制訂最高的數值為 26，所以 data 部分只需要五個位元（0　31）就可以符合我們的需求了。例如我們要藏入 FJUCSIE 這七個字母，F＝6，J＝10，U＝21，C＝3，S＝19，I＝9，E＝5。再將這些字母轉換為要藏入的浮水印 (index + data 二進位表示式)。

| | Index | Data |
|---|---|---|
| F => | 000 | 00110 |
| J => | 001 | 01010 |
| U => | 010 | 10101 |
| C => | 011 | 00011 |
| S => | 100 | 10011 |
| I => | 101 | 01001 |
| E => | 110 | 00101 |

再將轉換十進位的數值:

F:$(00000110)_2=(6)_{10}$　J:$(00101010)_2=(42)_{10}$
U:$(01010101)_2=(85)_{10}$　C:$(01100011)_2=(99)_{10}$
S:$(10010011)_2=(147)_{10}$ I:$(10101001)_2=(169)_{10}$
E:$(11000101)_2=(197)_{10}$

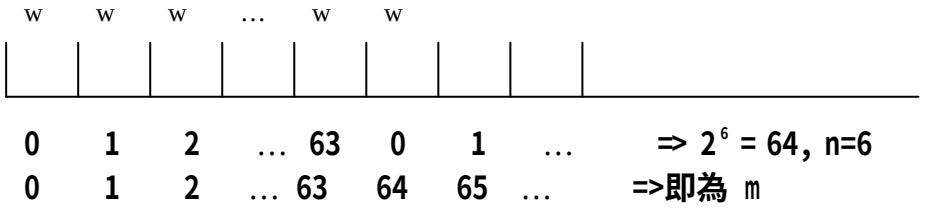



Figure 3.2

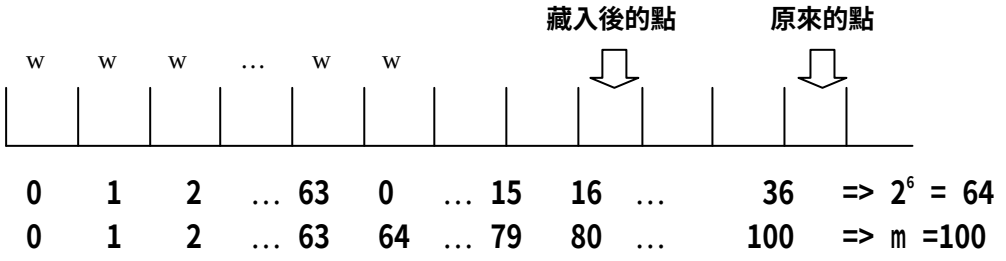



Figure 3.3

| Index | Data |
|---|---|
| F => 000 | 00110 |
| J => 001 | 01010 |
| U => 010 | 10101 |
| C => 011 | 00011 |
| S => 100 | 10011 |
| I => 101 | 01001 |
| E => 110 | 00101 |

再將轉換十進位的數值:

F:$(00000110)_2=(6)_{10}$ J:$(00101010)_2=(42)_{10}$

U:$(01010101)_2=(85)_{10}$ C:$(01100011)_2=(99)_{10}$

S:$(10010011)_2=(147)_{10}$ I:$(10101001)_2=(169)_{10}$

E:$(11000101)_2=(197)_{10}$

然後再將這些資料一一的藏入所要藏的點中，而且每個點都會藏到 data，達到之前所提到的多重藏入浮水印（Embedding multiple watermarks）的效果。

## 3.4 萃取浮水印

再來接著討論如何萃取浮水印。首先先找一片 Bézier patch，同樣的先把最外面的四個角落點取出，然後計算出 com 。再把中間的四個點帶入 3.1 節的式子（2），個別求出 $\|P\|^2_c$。求出的 $\|P\|^2_c$ 再除以 w，取整數值得到 m 。把 m mod $2^n$ (n 為藏入浮水印的位元數)，這樣便能得到所藏入的值。以之前 3.3 節所提到的例子來看，將藏入浮水印的點帶入 3.1 節的式子（2），便可以得到 80w，然後將 80 mod $2^6$ 得到 16（假設 n=6 ），因此就得到所藏入的值為 16。可以用同樣的方法把我們之前藏入七個英文字母（FJUCSIE）值的點取出來，取出之後再把這些值還原回成二進位表示式，然後前三位元為索引值（index），就可以知道這個字母排在第幾個順序，例如前三位元為（000）$_2$，換算回十進位值為 0，那我們就知道這個字母是排在第一位。接著後五個位元就是表示資料，如果這五位元為（00110）$_2$，換算回十進位值為 6，那表示這個字母為 F。綜合上述兩個條件我們就可以知道這個字母 F 是排在這些字母中的第一位。

## 3.5 Bézier Models

在這一節當中，我們首先介紹 Bézier curve 的觀念，接近介紹 Bézier surface (patch) 曲面，而 Bézier model 就是一片片 Bézier surface 所構成的 3D 模型。同時將討論 Bézier curve 的相關性質。

### 3.5.1 Cubic Bézier Curves 和 Bézier Surface Patches

若是 Degree 3 ，四個控制點（$\vec{r_0}$、$\vec{r_1}$、$\vec{r_2}$、$\vec{r_3}$）所定出的 Cubic Bézier curve， $\vec{r}(u)$ 如下：

$$\vec{r}(u)=\begin{bmatrix}u^3 u^2 u^1 1\end{bmatrix}\begin{bmatrix}-1 & 3 & -3 & 1\\ 3 & -6 & 3 & 0\\ -3 & 3 & 0 & 0\\ 1 & 0 & 0 & 0\end{bmatrix}\begin{bmatrix}r_0\\ r_1\\ r_2\\ r_3\end{bmatrix}$$

其中 $0 \le u \le 1$

Bézier patches $\vec{r}(u,v)$ 的 matrix 形式一般如下：

$$\vec{r}(u,v)=\begin{bmatrix}u^3 & u^2 & u & 1\end{bmatrix}\begin{bmatrix}B_z & r_c & B_z^T\end{bmatrix}\begin{bmatrix}v^3\\ v^2\\ v\\ 1\end{bmatrix}$$

其中 $B_z=\begin{bmatrix}-1 & 3 & -3 & 1\\ 3 & -6 & 3 & 0\\ -3 & 3 & 0 & 0\\ 1 & 0 & 0 & 0\end{bmatrix}$，

$r_C=\begin{bmatrix}r_{00} & r_{01} & r_{02} & r_{03}\\ r_{10} & r_{11} & r_{12} & r_{13}\\ r_{20} & r_{21} & r_{22} & r_{23}\\ r_{30} & r_{31} & r_{32} & r_{33}\end{bmatrix}$

u 與 v 的值在 0 與 1 之間，$r_c$ 是 16 個控制點 $r_{i,j}(0 \le i \le 3, 0 \le j \le 3)$ 所構成的矩陣。

### 3.5.2 Bernstein-Bézier Polynomial Curves

接下來我們介紹 Bézier curve 的一些特性。首先，一個 Degree $n$ 的 Bernstein-Bézier polynomial curve $\vec{r}$（u）可以用一些控制點 $\vec{r_0}$、$\vec{r_1}$、、、$\vec{r_n}$ 來定義出，

$$\vec{r}(u)=\sum_{i=0}^{n}\vec{r_i}B_{i,n}(u) = \sum_{i=0}^{n}C_i^n u^i(1-u)^{n-i}\vec{r_i}$$

，其中 $B_{i,n}(u)=C_i^n u^i(1-u)^{n-i}$ ，

$C_i^n=\dfrac{n!}{i!(n-i)!}$， $0 \le u \le 1$

### 3.5.3 Bernstein-Bézier polynomial curve 的 De Casteljau 表示式

上一小節 Bézier 所提出的 Bernstein-Bézier polynomial curve 在電腦繪圖方面使用上不是很方便。所以後來 De Casteljau 提出了 De Casteljau 演算法來改善。這裡我們推導一下，由 Bernstein-Bézier polynomial curve 到 Bézier curves 的 De Casteljau 表示的過程。

$$\vec{r}(u) = \sum_{i=0}^{n} C_i^n u^i (1-u)^{n-i} \vec{r}_i$$

$$= (1-u)^n \vec{r}_0 + \sum_{i=0}^{n-1} C_i^n u^i (1-u)^{n-i} \vec{r}_i + u^n \vec{r}_n$$

因為 $C_i^n = C_i^{n-1} + C_{i-1}^{n-1}$，所以

$$\vec{r}(u) = (1-u)^n \vec{r}_0 + \sum_{i=1}^{n-1} C_i^{n-1} u^i (1-u)^{n-i} \vec{r}_i + \sum_{i=1}^{n-1} C_{i-1}^{n-1} u^i (1-u)^{n-i} \vec{r}_i + u^n \vec{r}_n$$

$$= (1-u)\left[(1-u)^{n-1}\vec{r}_0 + \sum_{i=1}^{n-1} C_i^{n-1} u^i (1-u)^{n-1-i} \vec{r}_i\right] + u\left[\sum_{i=1}^{n-1} C_{i-1}^{n-1} u^{i-1} (1-u)^{n-i} \vec{r}_i + u^{n-1}\vec{r}_n\right]$$

$$= (1-u)\left[\sum_{i=0}^{n-1} C_i^{n-1} u^i (1-u)^{n-1-i} \vec{r}_i\right] + u\left[\sum_{i=1}^{n-1} C_{i-1}^{n-1} u^i (1-u)^{n-i-1} \overrightarrow{r_{i+1}}\right]$$

我們令 $\overrightarrow{r_{k,L}}(u)$ 表示由控制點（$\vec{r}_k, \overrightarrow{r_{k+1}}, \ldots, \vec{r}_L$）所代表的 Bernstein-Bézier polynomial curve，而且 curve 的 degree 為 (L-K)。所以，$\overrightarrow{r_{0,n}}(u) = \vec{r}(u) = (1-u)\overrightarrow{r_{0,n-1}}(u) + u.\overrightarrow{r_{1,n}}(u)$ 或

$= \overrightarrow{r_{0,n-1}}(u) + u\left[\overrightarrow{r_{1,n}}(u) - \overrightarrow{r_{0,n-1}}(u)\right]$

上述就是所謂的 Bézier curves 的 De Casteljau 表示式。換句話說，給定一個 u 值（0 到 1 之間）Bernstein-Bézier polynomial curve 上的點 $\vec{r}(u)$ 可以由一線段上，有相同 u（參數點）的兩個多項式組合而成，而且以比例 u 分割此線段。

## 3.6 Bézier Curve/Surface Subdivision and Merging

在這一節當中，我們將討論 Bézier curve/surface 的分割與合併，分述如下：

### 3.6.1 Bézier Curve Subdivision

這裡我們討論在電腦繪圖上最常使用的 Bézier curve subdivision 的技巧（u=0.5），即以取線段中點方式把 Bézier curve subdivision。4 點控制點 $p_0, p_1, p_2, p_3$ 所定義的 Cubic Bézier curve $\vec{r}(u)$ 可以在 $\vec{r}(0.5)$ 處分割成兩段。Cubic Bézier curve 左半段的控制點為 $q_0, q_1, q_2, q_3$，而右半段的控制點為 $r_0, r_1, r_2, r_3$。這些控制點之間的關係如下（如 Figure 3.4）：

可以導出下列式子(Lane 提出)

$$q_0 = p_0 \qquad r_0 = q_0$$

$$q_1 = \frac{p_0 + p_1}{2} \qquad r_1 = \frac{p_1 + p_2}{4} + \frac{r_2}{2}$$

$$q_2 = \frac{q_1}{2} + \frac{p_1 + p_2}{4} \qquad r_2 = \frac{p_2 + p_3}{2}$$

$$q_3 = \frac{q_2 + r_1}{2} \qquad r_3 = p_3$$

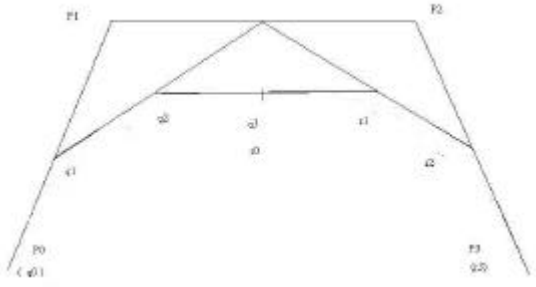


Figure 3.4

### 3.6.2 Bézier Patches Subdivision

我們從 3.5.1 節 Bézier Patch $\vec{r}(u,v)$ 的 matrix 形式，可以得到 $\vec{r}(u,v)$ 的形式如下：

$$\vec{r}(u,v)=\sum_{i=0}^{k}\sum_{j=0}^{k} r_{i,j}B_{i,n}(u)B_{j,n}(v)$$

，可再經由數學性質得到

$$\vec{r}(u,v)=\sum_{i=0}^{k} B_{j,n}(v)\left\{\sum_{j=0}^{k} r_{i,j}B_{i,n}(u)\right\}$$

。換句話說，Bézierpatches subdivision 可視為分別先對 curve 控制點 v（u）參數 splitting 再對 curve 控制點 u（v）參數 splitting。

### 3.6.3 Merging of Subdivided Bézier Patches

Bézier Patches 先天上就比較沒有抗剪面(simplication)的問題，但是相對的會有需要抗 subdivision 的問題。通常是利用，一片 Bézier Patches 分成四片的方法。subdivision 後的 Bézier Patches 控制點增加。這雖然是 subdivision 的優點，但是對於我們要加入浮水印卻是非常不利。會影響到 Bézier Patches 控制點，表示我們將來在要萃取浮水印時可能會錯誤。

我們有一個避免對抗 subdivision 的方法，就是 merging。如果我們能在萃取浮水印計算前，把 subdivision 後的 Bézier subpatches 使用 merging 的方法，將相鄰兩片 Bézier subpatches 合併的方式，恢復到還沒 subdivision 前的 Bézier Patches。那就可以由正確的 Bézier Patches 控制點計算，找出正確的浮水印。

下面我們將會介紹如何把 subdivision 後的 Bézier Patches，正確的回復到還沒 subdivision 前。假設一個 Bézier Patches 被 subdivision 的情況如下（Figure 3.5）：

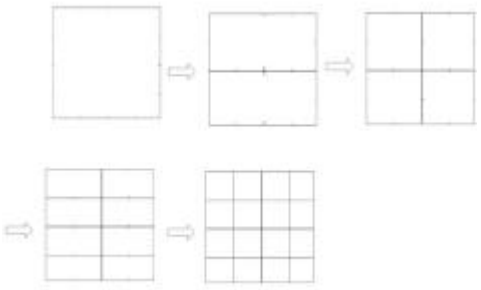


Figure 3.5

而 merging 可能的狀況（Figure 3.6），本來我們以為一定要由第一種方式如 Figure 3.6(a) 其合併的順序剛好是 subdivision 過程的相反步驟，才可以把 subdivision 後的 Bézier Patches，正確的回復到還沒 subdivision 前。但是實際上不論哪種都可以，後面我們會有說明。

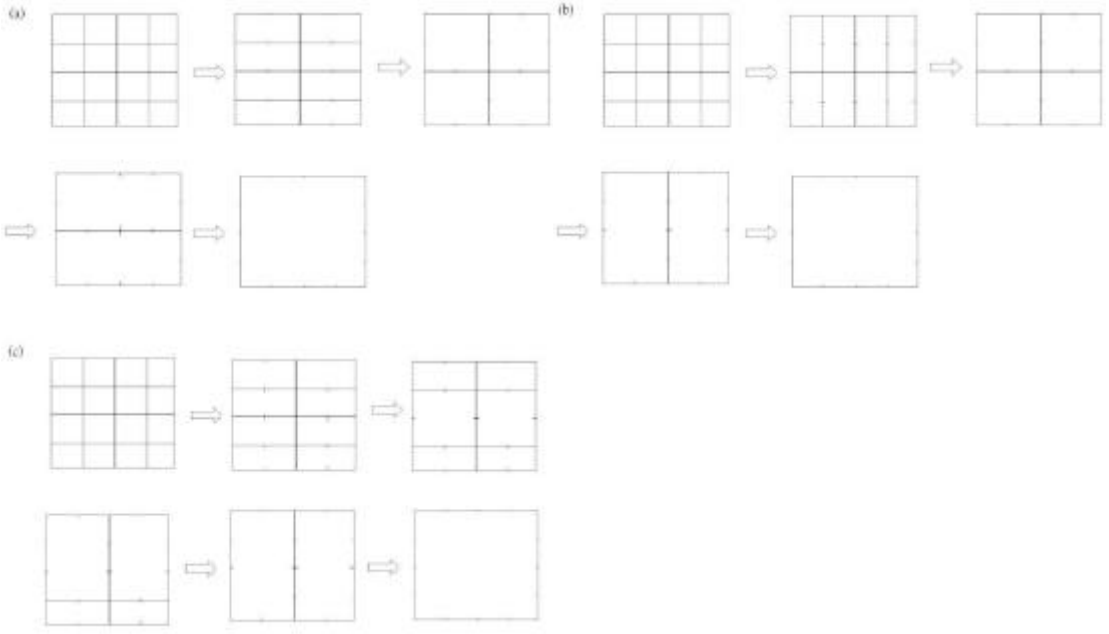


Figure 3.6

如此來看狀況可能有點複雜，我們可以由它的一維觀點來分析（Figure 3.7)。我們根據 Bézier curves 的 De Casteljau 理論，Bernstein-Bézier polynomial curve 可以由一線段上，有相同 u（參數點）的兩個多項式組合而成,而且以比例 t 分割此線段。以 n=t，控制點（$\vec{r_0}$、$\vec{r_1}$、$\vec{r_2}$、$\vec{r_3}$）為例如 Figure 3.7。

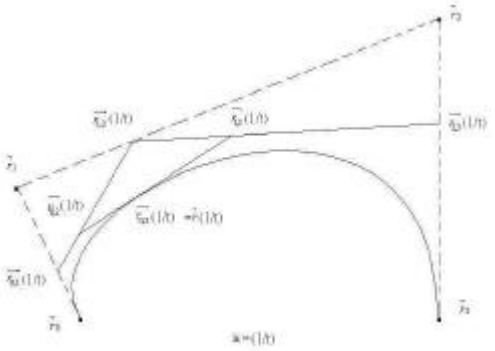


Figure 3.7

而 merging 時就可由兩段 Bézier curves，其最前兩個控制點、最後兩個控制點，分別依據 t 倍和(t/(t-1))倍找出 merging 後的控制點（$\vec{r_0}$、$\vec{r_1}$、$\vec{r_2}$、$\vec{r_3}$）。

Figure 3.5 的 Bézier Patches，它的一維觀點如下： Merging 第一種方式（Figure 3.8）

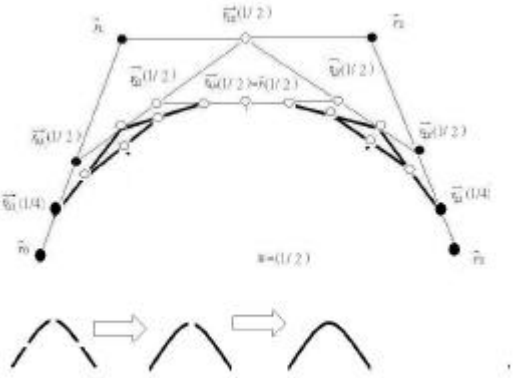


Figure 3.8

而 merging 時由 u=1/2：$\dot{r_0}$、$\dot{r}_{0,1}(1/4)$找到 $\dot{r}_{0,1}(1/2)$，由 $\dot{r_3}$、$\dot{r}_{2,3}(1/4)$找到 $\overrightarrow{r_{2.3}}(1/2)$，再由 u=1/2：$\dot{r_0}$、$\dot{r}_{0,1}(1/2)$找到 $\dot{r_1}$，最後由 $\dot{r_3}$、$\overrightarrow{r_{2.3}}(1/2)$找到 $\vec{r_2}$。其他方式 merging，如 Figure 3.9

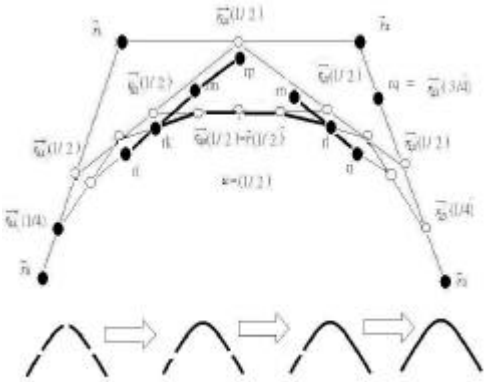


Figure 3.9

而 merging 時由 u=1/2：$\dot{r_i}$、$\dot{r_k}$找到 $\dot{r_m}$，由 $\dot{r_j}$、$\dot{r_l}$找到 $\dot{r_n}$；再由 u=2/3：$\dot{r_i}$、$\dot{r_m}$找到 $\dot{r_p}$，最後由 $\dot{r_3}$、$\overrightarrow{r_{2.3}}(1/4)$找到 $\overrightarrow{r_q}$ =$\overrightarrow{r_{2.3}}(3/4)$。最後階段由 u=1/4：$\dot{r_0}$、$\dot{r}_{0,1}(1/4)$找到 $\dot{r_1}$，由 $\dot{r_3}$、$\overrightarrow{r_{2.3}}(3/4)$找到 $\vec{r_2}$，而得到控制點（$\vec{r_0}$、$\vec{r_1}$、$\vec{r_2}$、$\vec{r_3}$）

由以上的說明，我們知道實際上不論哪種 merging 方式都可由 subdivision 後的 Bézier Patches 控制點，找到最初的 Bézier Patches 控制點（$\vec{r_0}$、$\vec{r_1}$、$\vec{r_2}$、$\vec{r_3}$），可即是正確的回復到還沒 subdivision 前的 Bézier Patches。

## 4. 實驗結果

我們將利用 Affine Invariant Norm 的方法來藏入 watermark，使用的硬體配備與相關軟體如下：

CPU => Celeron 566 MHz

顯示卡 => Geforce 2 MX 32MB（支援 OpenGL)

記憶體 => 128MB

作業環境 => Windows 98 第二版、Visual C++ 6.0、GLUT 3.7 程式庫（For Windows）

使用的模型有茶壺、湯匙、咖啡杯、滑鼠。這些模型的相關資料列在下列的表格中：

| | 頂點個數 | Bézier patches |
|---|---|---|
| 茶壺 | 306 | 32 |
| 湯匙 | 256 | 16 |
| 咖啡杯 | 251 | 26 |
| 滑鼠 | 240 | 15 |

接下來就是我們實驗的一些結果（w 值為 0.000001）：

(1)**茶壺：**Figure 4.1 **展示加入** watermark **之前與之後茶壺的圖。**

(a.)**未藏入** watermark **的原圖**

(b.)**藏入** watermark **後**

c.**藏入** watermark **後（只畫出線框）**

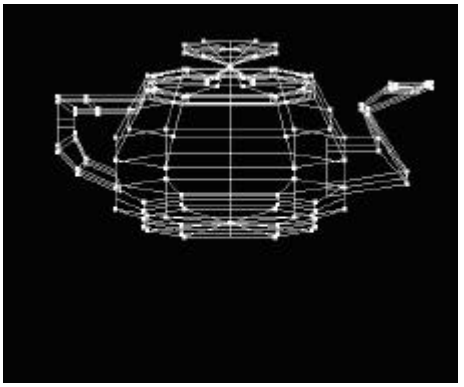
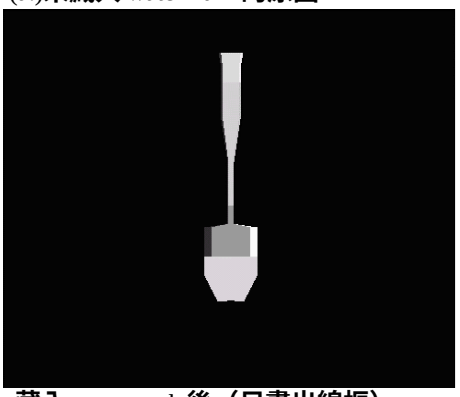


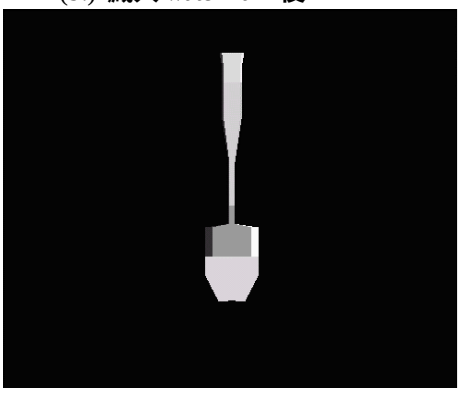
Figure 4.1

(2)**湯匙：**Figure 4.2 **展示加入** watermark **之前與之後湯匙的圖。**

(a.)**未藏入** watermark **的原圖**



(b.) **藏入** watermark **後**



c.**藏入** watermark **後（只畫出線框）**

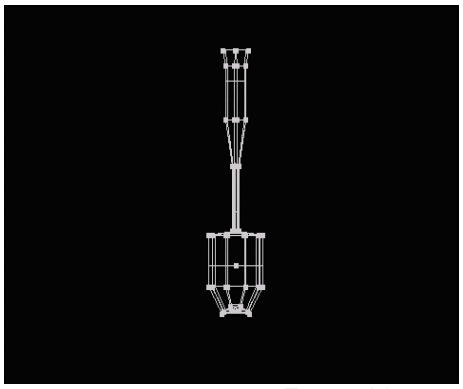


Figure 4.2

(3)**咖啡杯：** Figure 4.3 **展示加入** watermark **之前與之後咖啡杯的圖。**

(a.)**未藏入** watermark **的原圖** (b.) **藏入** watermark **後**

(c.)**藏入** watermark **後（只畫出線框）**

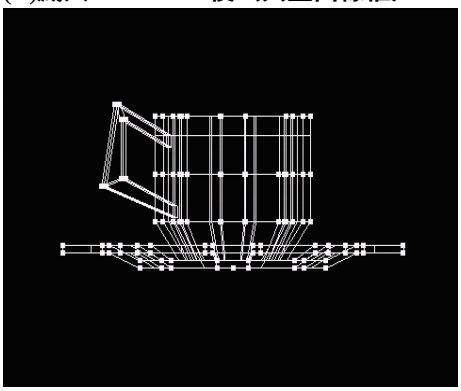
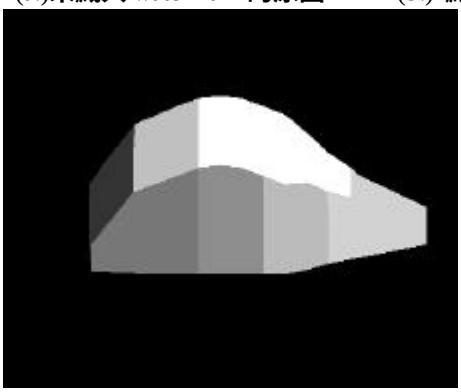
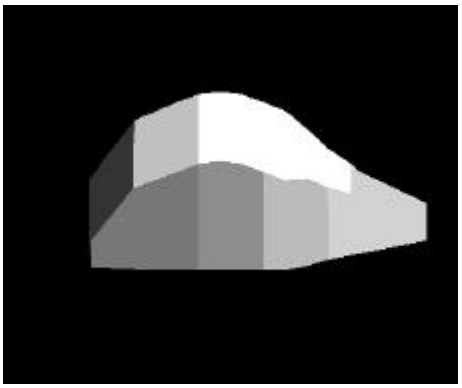


Figure 4.3

(4)**滑鼠：** Figure 4.4 **展示加入** watermark **之前與之後滑鼠的圖。**

(a.)**未藏入** watermark **的原圖** (b.) **藏入** watermark **後**





c.**藏入** watermark **後（只畫出線框）**

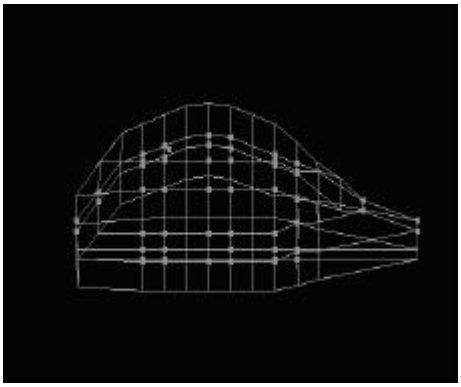


Figure 4.4

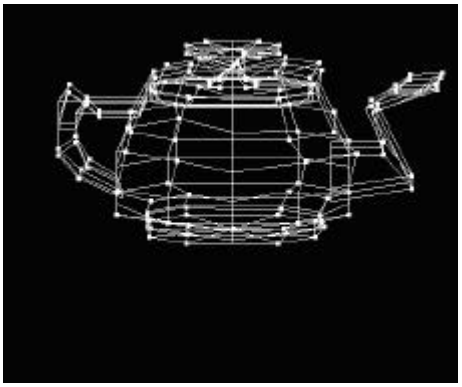



Figure 4.5

在實驗的過程中，如果藏入 watermark 的資料是固定的話，那麼 W 值的大小，便會影響到模型外觀的改變。如果我們取 W 值大於 $10^{-4}$ 的話，那麼茶壺的外觀便會開始變形，如 Figure 4.5（左圖是加 watermark 之後 shading 之後的圖，右圖為左圖的 wire frame 圖），Figure 4.5 的 W 值為 0.001，因此我們在取 W 值時，需要稍微小心一下。

## 5. 結論

我們在這裡討論了一些 watermarking 的技術，其主要的目的是希望能有一個更有效的方法，來保護我們的智慧財產權。之前所提到的 NBE 和 TSQ 的方法，像 NBE 的方法使用一般 Affine transformation（Rotation、Uniform scaling 、Non-Uniform scaling 、 Translation 的組合）的方式，可以把 NBE 加入的 Watermark 破壞掉，而 TSQ 的方法遇到座標被破壞時（randomization of coordinates）或遇到三角片減面時，TSQ 所加入的 Watermark 就會被破壞掉。

Bézier patches model 上，一個 Bézier patches 固定有 16 個控制點。試圖刪掉其中幾點或是攪亂點座標都會使 Bézier patches model 產生嚴重的變形或是 patch 與 patch 接合處會有破洞，這表示 Bézier patches model 抵抗減面的攻擊。所以我們在 Bézier patches 上，主要考慮如何對抗 Affine transformation。因此，我們提出 Affine Invariant Norm 的方法，對抗 Affine transformation ，讓這個 Watermarking 的方法，更加的健全（robust）。

## 誌謝

本研究承蒙輔仁大學研究發展處學術研究組補助研究(計畫編號：336-1-2001-1-225)，特此致謝。

## 參考文獻

[1] O.Benedens, Geometry-Based Watermarking of 3D Models, *IEEE Computer Graphics and Applications, Special Issue on Image Security*, pp.44-55, January/February, 1999.

[2] O.Benedens, Watermarking of 3D polygon based models with robustness against mesh simplification, *Proceedings of SPIE: Security and Watermarking of Multimedia Contents*, pp. 329-340, 1999.

[3] O. Benedens, Two High Capacity Methods for Embedding Public Watermarks into 3D Polygonal Models, *Proceedings of the Multimedia and Security-Workshop at ACM Multimedia 99*, Orlando, Florida, 1999, pp. 95 - 99.

[4] O. Benedens and C. Busch, Towards Blind Detection of Robust Watermarks in Polygonal Models, *Eurographics 2000 Conference Proceedings*, August 2000.

[5] P. Bézier, Numerical Control: Mathematics and Applications, *Wiley,Chichster*, UK, 1972.

[6] Y. Boon-Lock and M. Minerva, Watermarking 3D Objects for Verification, *IEEE Computer Graphics, Special Issue on Image Security*, pp. 36–45, January/February 1999.

[7] H.HOPPE, Progressive meshes, *in ACM SIGGRAPH 96 Conference Proceedings (Aug. 1996)*, pp. 99–108.

[8] G.M. Nielson and T.A Foley, A Survey of Applications of an Affine Invariant Norm, in: T. Lyche and L.L. Schumaker (eds), *Mathematical Methods in Computer Aided Geometric Design*, Academic Press, Boston, MA, 1989, pp.445-467.

[9] R. Ohbuchi, H. Masuda, and M. Aono, Embedding Data in 3D Models, in Steinmetz, et al. eds, Lecture Notes in Computer Science No. 1309, pp.1-11 (*Proceedings of the IDMS '97*, Darmstadt, Germany, September), Springer, Berlin, Germany, 1997.

[10] R. Ohbuchi, H. Masuda, and M. Aono,

Watermarking Three-Dimensional Polygonal Models, *ACM Multimedia 9*7, pp. 261–272, 1997.

[11] R. Ohbuchi, H. Masuda, and M.Aono, Watermarking Three-Dimensional Polygonal Models Through Geo-metric and Topological Modifications, *IEEE Journal on selected areas in communication*s, **1**6(4), pp. 551–559, May 1998.

[12] R. Ohbuchi, H. Masuda, and M. Aono, Geometrical and Non-geometrical Targets for Data Embedding in Three-Dimensional Polygonal Models, *Computer Communications*, Vol. 21, pp. 1344-1354, Elsevier (1998).

[13] E. Praun , H. Hoppe, and A. Finkelstein, Robust Mesh Watermarking, *SIGGRAPH 99 Proceedings*, pp. 69–76, 1999.

[14] M.G. Wagner, Robust Watermarking of Polygonal Meshes, *Geometric Modeling and Processing 2000*, April 10-12, 2000, Hong Kong.

[15] M. M. Yeung and B. L. Yeo, Fragile Watermarking of Three-Dimensional Objects, in International Conference on Image Processing, Vol. 2, pp. 442-446, 1998.