國立交通大學

資訊學院資訊科技（IT）產業研發碩士班

碩 士 論 文

賽車遊戲中的雨雪效果

Rain and Snow Effects in Racing Game

研 究 生：林九谷

指導教授：施仁忠　教授

中 華 民 國 九 十 七 年 十 月

賽車遊戲中的雨雪效果

學生：林九谷　　指導教授：施仁忠 教授

國立交通大學資訊學院產業研發碩士班

摘　　要

自然現象模擬在電腦繪圖學中一直是很重要的研究主題，但是如果要進一步將相關的技術應用於電腦遊戲上，則需要避免使用過於複雜的物理模擬計算。在本篇論文我們主要模擬賽車遊戲中的降雨及飄雪效果，使用建構於 GPU 上的粒子系統達到即時模擬與描繪之目的。雨雪效果最後整合至遊戲業界使用之 3D 繪圖引擎，配合賽車遊戲之美術場景與賽車動畫，即可呈現不同天氣之間的變化和車輪捲起雨滴(雪花)的效果。

Rain and Snow Effects in Racing Game

Student：Chiu-Ku Lin Advisors：Dr. Zen-Chung Shih

Industrial Technology R & D Master Program of
Computer Science College
National Chiao Tung University

ABSTRACT

The simulation of natural phenomena in computer graphics has been a very important topic. However, if further related technology used in computer game, you need to avoid using too complex physics simulation. In this paper we simulate the effects of rainfall and snowfall in the racing game. Using the GPU-based particle system to achieve real-time simulation and rendering. The final effects of rain and snow into the 3D graphics engine that be use by game industry. With the art scene of the racing game and the car animation, can be shown the change between the different weathers and the effect that raindrops (snowflakes)is rolled up by the wheels.

誌謝

這篇論文能夠完成，要感謝許多人。首先要感謝我的指導教授施仁忠博士在論文研究方面細心的指導，還有要感謝計算機圖學實驗室的所有成員在我的研究生涯陪著我一起度過，特別感謝實驗室學長蔡侑庭在程式效能改善方面給與的建議，以及實驗室學弟吳昱霆、張翔俊、林逸晨在雨天描繪技術部分的參與，讓最後整合出的天氣效果更加真實。還要感謝鈊象電子遊戲公司對於即時天氣系統研究計畫的大力支持，特別感謝鈊象繪圖小組在 OGRE 3D 繪圖引擎方面提供的技術支援，以及美術設計人員張韋特、賴誌宏在雨雪效果部分提供的貼圖材質與賽車環境部分提供的場景檔案。最後要感謝我的家人在我求學階段給我的鼓勵，讓我可以順利的在學校完成學業。

目錄

表目錄

圖目錄

第一章　緒論

在此章節中，會對本篇論文的研究主題先做概略性的介紹，內容包括研究動機與目的、研究方法的描述。接著再對本論文各章節將討論到的內容預先做個簡介。

1.1 研究動機與目的

近幾年來，國外有越來越多款的遊戲將天氣變化的元素加入其中，例如：微軟的「模擬飛行 2004」(Flight Simulator 2004)和 XBOX 360 上的「世界街頭賽車 4」(Project Gotham Racing，簡稱 PGR4）等幾款遊戲是較廣為人所知的作品。這些遊戲在戶外場景中融入了天氣變化的效果，透過各種天氣模式的轉變，讓遊戲畫面呈現出更為豐富的視覺效果，如毛毛細雨、傾盆大雨或是雪花紛飛等，使玩家在體驗遊戲的同時，能有身歷其境的感受。

自然現象之描繪技術在電腦繪圖學領域一直是很受注目的研究主題。雪的描繪部分，相關的研究主要分為模擬降雪時雪花飄落的描繪技術[11, 12, 14, 20]和地面產生積雪效果的描繪技術[3, 4, 13, 17]兩方面。但先前的研究為了呈現逼真的雪花及雪景繪製效果，會使用較花費時間的計算方法來實現，並不適合應用在像賽車這種與玩家有高度互動性的遊戲上。雨的描繪部分，相關的研究可分為降雨效果的描繪技術[5, 6, 15, 16, 18]和物體表面的水滴痕描繪技術[7, 8, 21]兩方面。在降雨效果的描繪方面，雖然已有可達到即時計算的方法被提出，但如要應用到實際的遊戲上，還要考量到與其它效果整合的問題。包括賽車動畫和場景光影處理等效果，這些都會額外佔據電腦的計算資源，增加遊戲整體處理運算的時間。

本論文探討能應用在數位遊戲上之即時雨雪描繪技術，使用架構於「繪圖處理單元」（Graphic Processing Unit，簡稱 GPU）之粒子系統，實現各種賽車遊戲所出現的雨雪效果，如：降雨和飄雪之效果、車輪捲起水滴或雪花之效果。最後，我們會把所開發之技術整合到遊戲業界使用的 3D 繪圖引擎中，呈現在不同的賽車場景以及跟其它描繪技術一同執行的效果。

1.2 研究方法概述

隨著GPU計算能力大幅的提升，數位遊戲的技術開發重心已經逐漸轉移至繪圖加速硬體上，以研發各式各樣「著色器程式」（Shader Program）為基本主軸，並透過妥善分配GPU與「中央處理單元」（Central Processing Unit, 簡稱CPU）的計算資源，來達到即時繪圖與擬真成像之目的。由於GPU具有可程式化、高度平行化處理以及高速浮點運算的特性，使其相當適合運用至數位遊戲加速，同時也經由許多專家學者的研究成果顯示，許多數位遊戲常用的演算法轉換至GPU上執行時，明顯地超越純粹在CPU上的執行效能，我們的系統架構圖如圖 1所示。

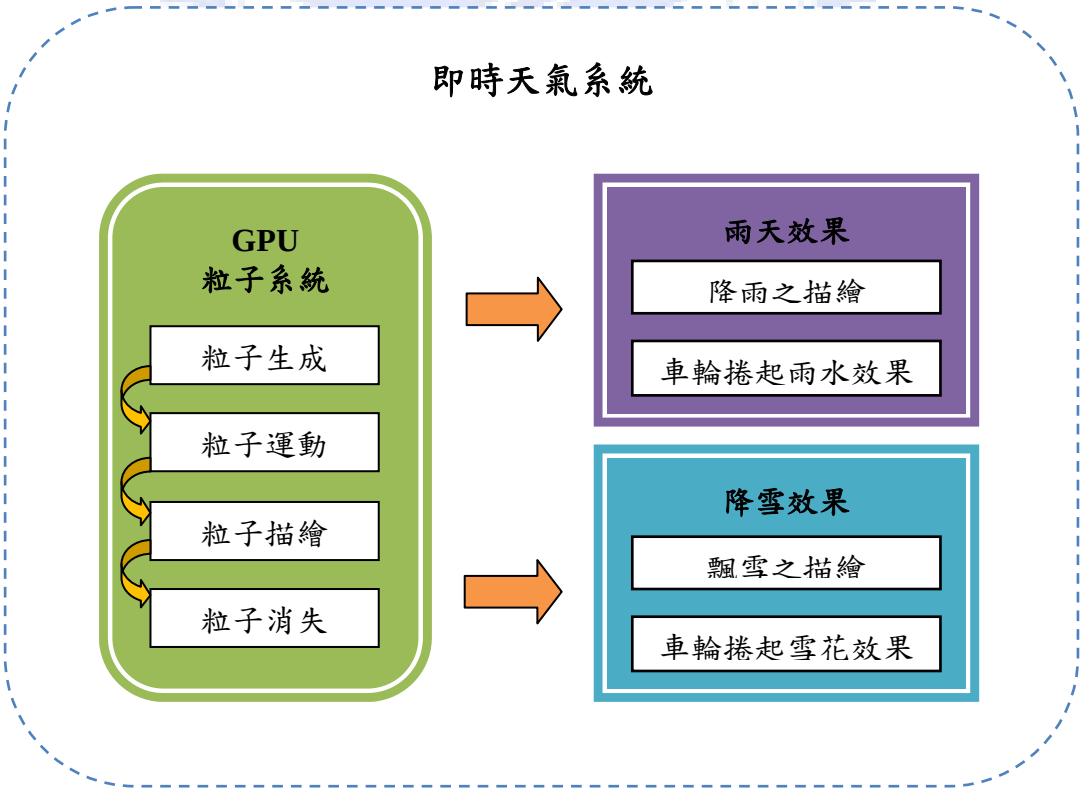


圖 1 即時天氣系統架構圖

程式的架構以開發一個即時天氣系統為基礎，此架構最終會被整合到賽車遊戲的 3D 繪圖引擎中，。我們運用 3D 繪圖引擎提供與 Shader 程式溝通的方法來實作一個 GPU 粒子系統，利用 GPU 可平行化處理的優勢來計算每個粒子移動後的位置和速度資訊。最後可透過調整 GPU 粒子系統的各項參數，將我們想要呈現的雨天和降雪效果描繪出來。

1.3 論文組織

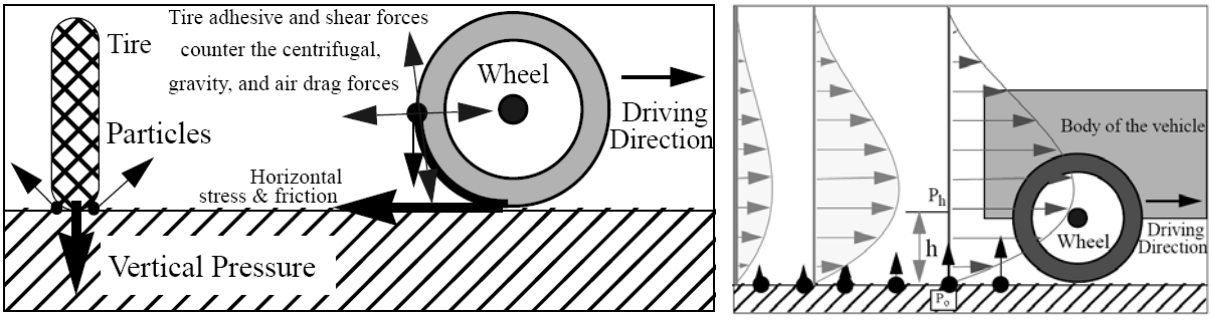
本論文接下來的章節討論如下，在第二章介紹之前相關的研究及分析這些作法的特色及優缺點。在第三章介紹我們使用的 3D 繪圖引擎及討論在此引擎架構下如何建立我們所需的 GPU 粒子系統。第四章說明我們用來描繪降雨和飄雪的方法。在第五章會展示我們的技術與賽車遊戲整合之成果。最後，在第六章回顧本論文之方法，並提出未來可能加強及改進的地方。

第二章　相關研究

2.1 粒子系統技術

粒子系統是在電腦遊戲或是動畫製作中經常被使用到的一項技術，可以被用來模擬如雨、雪、煙、塵等許多自然現象。在電腦繪圖學領域，Reeves 早於 1983 年發表了第一篇探討粒子系統的論文[14]，當中描述了粒子系統的基本動作運算和資料表示方法。後來的研究，為了讓粒子系統能模擬出更為逼真氣體流動效果，使用計算流體力學（Computational Fluid Dynamics, 簡稱 CFD）的方法來計算氣體中粒子所受到的作用力。

在日常生活中，當車輛行駛過砂石路面即會產生煙塵，Jim等人[1]有對快速移動之車輛其車輪與地面接觸所揚起之塵埃效果作即時的模擬，其研究將塵埃粒子產生的情況分成以下三種：(1)車輛之重量會造成路面的顫動及變形，將壓碎大顆的砂粒而產生細小的塵埃粒子飄向空中（如圖 2 a 左邊所示）。(2)輪胎上的塵埃粒子受到輪胎轉動之離心力與空氣中之阻力等因素影響，從輪胎表面彈向空中（如圖 2 a右邊所示）。(3)車輛高速行駛時，產生的氣流導致車底或後方的塵埃粒子飄向空中（如圖 2 b所示）。



(a)　　(b)

圖 2 塵埃產生的三種方式

由於使用 CFD 計算車輛周圍的氣流會花費大量的計算時間，Jim 等人為了達到即時互動的效果，用預先計算好的氣流場（Flow Field）來簡化數值計算之方程式，且把塵埃粒子的行為簡化成：氣流擾動、粒子動能、空中飄移三種，分別由三個不同的粒子系統來模擬。但是在賽車遊戲中，受玩家操控之賽車其行駛路徑是不固定的，無法預先建立出車輛行駛所產生的氣流場，因此 Jim 等人提出的方法不適合使用在賽車遊戲中。

在粒子系統的即時處理能力方面，描繪效能會受到 GPU 像素填充率（Pixel Fill Rate）或是 CPU 與 GPU 之間頻寬的影響。像素填充率表示每個畫格（Frame）中 GPU 能夠描繪的像素數量，只在畫面需要頻繁重複描繪的情況下，對效能才有明顯的影響，例如：粒子面積很大，則粒子之間就容易互相重疊覆蓋到。幸好一般而言，粒子系統使用越小的粒子越能模擬出逼真的視覺效果，因此像素填充率對粒子系統效能的影響較為有限。

另一個影響因素是 CPU 與 GPU 之間的傳輸頻寬，在 CPU 模擬的粒子運動要將計算結果傳送到 GPU 描繪出來，因為繪圖卡的匯流排同時也會被其它的繪圖工作所使用，所以在大部分的遊戲應用裡，粒子系統在每個畫格頂多允許 10000 個粒子存在。當粒子數量一多就必須分好幾個畫格來描繪，這樣將影響到系統的處理效能。為了解決這個問題，就有研究提出把在 CPU 做粒子運動計算部分移到 GPU 上去實行的方法[9][10]。

Kolb 等人[10]把可在 GPU 上實行粒子運動計算的方法分成 Stateless 粒子系統和 State-Preserving 粒子系統兩種，分別說明如下：

(1) Stateless 粒子系統

只使用最初設定的初始值和一個 Closed-Form 函數去描述粒子的行為，包括粒子位置的移動和屬性的改變，也被稱為參數（Parametric）粒子系統。這種方法主要的優點是不需要額外儲存每個粒子位置或屬性，就可以精確的計算出粒子的運行路徑。缺點是粒子的動作是不可改變的，這限制了粒子和環境中其它物體接觸產生即時反應的能力。所以當要製作的粒子特效不要求與場景互動，例如煙火在高空爆炸的效果，就很適合用 Stateless 粒子系統來實作。

(2) State-Preserving 粒子系統

在每個時間點，粒子位置和移動速度會受到前一個時間點之粒子位置和速度的影響，這種方法要把粒子的資料儲存在貼圖中（如圖 3所示），在過程中使用Pixel Shader反覆的去更新貼圖上的資料。由於粒子的資料有被記錄下來，State-Preserving粒子系統可以處理與動態環境下較複雜的互動，甚至是粒子之間的相互作用。而為了避免粒子資料會同時地被讀取或寫入同一個貼圖中，通常使用雙緩衝貼圖（Double Buffer Textures）的方式實作。

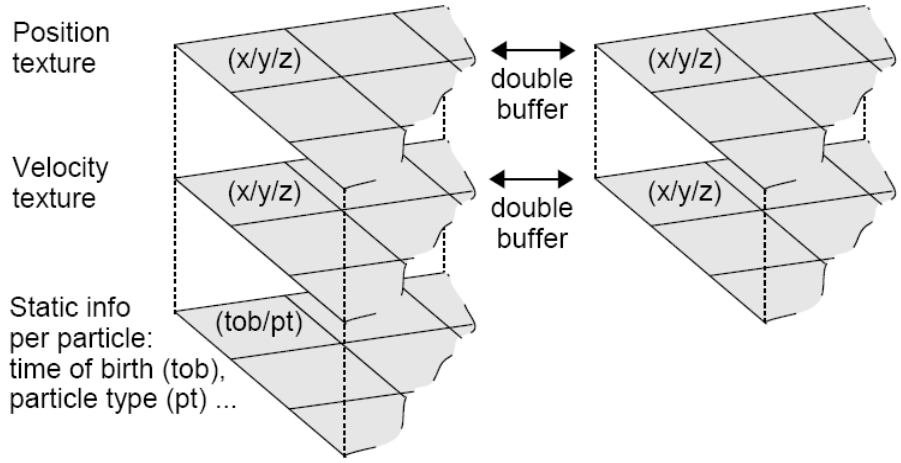


圖 3 將粒子的速度、位置等資料存在雙緩衝貼圖裡

2.2 飄雪描繪技術

要呈現出雪花紛飛的效果，粒子系統是最常被使用到的技術，但是當雪花粒子的數量越多，效能所付出的代價就越昂貴，Langer 等人[12]提出了混合粒子系統與影像式頻譜合成（Image-Based Spectral Synthesis）的方法來描繪飄雪效果。他們的方法是先用粒子系統產生少量的雪花粒子，再將用影像式頻譜合成技術產生的動態貼圖（Dynamic Texture）填入雪花粒子之間的空隙。從他們的研究結果可看出，當雪花粒子的數量龐大時，使用這種混合式方法會比只用粒子系統的方法在效能有更大的提升，但是他們的方法只適合應用在靜態的場景中。

Wang 等人[20]使用粒子系統建立出雪花與風互動的現象，空中的雪花會受到風力影響而改變行進方向，地面上的積雪也會受到風力的影響而改變形狀。他們的方法主要分為風場（Wind Field）的建立和雪效果的模擬兩個部分，分別說明如下：

(1) 風場的建立

風場是由許多網格（Lattice）所組成，空氣中的微粒子只能分佈在網格頂點上。為了加快計算處理時間，會把微粒子可移動到相鄰頂點之方向從 27 個簡化成 15 個方向(如圖 4所示)。而雪花移動後的位置就由周圍網格頂點透過內差計算來決定。

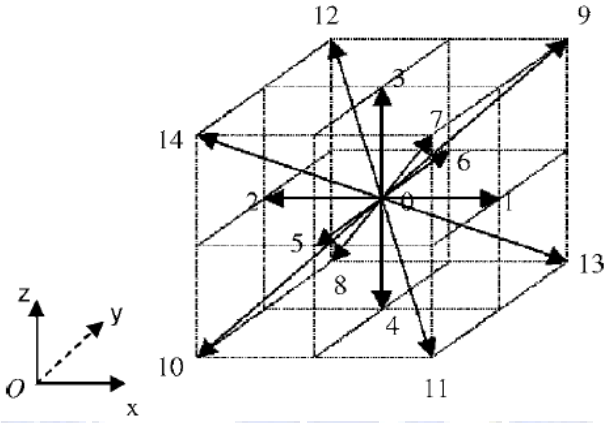


圖 4 風場的 3D 離散空間

(2) 雪效果的模擬

Wang等人[20]只考慮引力和風力對雪花之影響，主要模擬三種景象：風中飄盪的雪花、地面積起雪花的現象、地面積雪受風力侵蝕的效果。當雪花從空中降下時，由先前建立之風場來計算雪花移動的位置。當地面上同區域的雪花累積超過所設定的數量，地面高度場（Height Field）的高度開始增加。當風力超過預設值，累積在地面的雪花便會開始移動位置，且減少地面高度場的高度。最後，雪花之描繪可使用梅花形的貼圖（如圖 5所示），產生較細緻的效果。

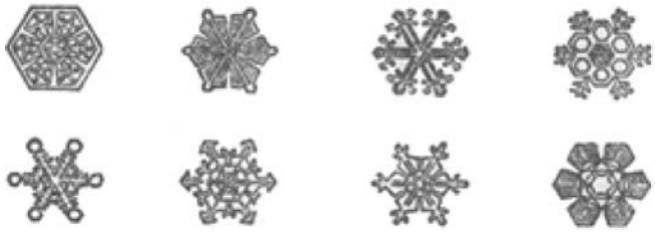


圖 5 雪花的 2D 貼圖

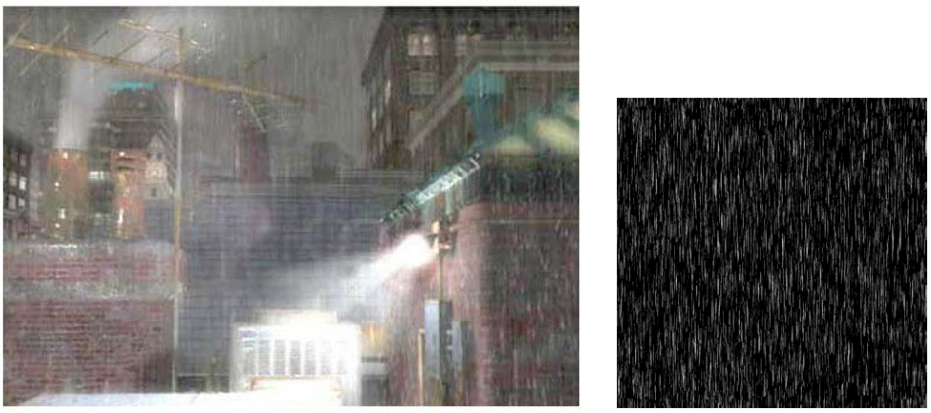
2.3 降雨描繪技術

雨是一種複雜的視覺現象，是由許多種視覺元素所組成。降雨效果可看成是由一群不規則分佈且快速落下的水滴集合，而每一個水滴都會對當時的環境光線產生反射和折射。雨滴由高處快速落下的景像，會因為人眼睛的視覺暫留問題或是攝影機快門的速度關係，看起來會像是一條條白色的線條。在電腦視覺領域，Starik 與 Werman[16]有提出在影片中加入降雨效果的方法。Garg 與 Nayar[5]則進一步提出從影片中偵測及移除降雨效果的技術。

Tatarchuk[18]在 3D 場景中建立出惡劣天氣的夜晚，天空下著雨的城市景象。當中整合了各種不同的技術來模擬下雨天所呈現的視覺效果，例如場景中遠處降雨效果是使用動態貼圖合成的方法，而近距離從屋簷上降下的雨滴則是使用粒子系統來製作。下面分別討論這兩種下雨現象所使用的技術：

(1) 降雨效果

由預先準備的降雨貼圖（如圖 6 b所示）來模擬降雨現象，美術人員可以在世界空間（World-Space）指定降雨的方向和速度來模擬不同程度之降雨效果，降雨的方向和速度會被轉換到裁剪空間（Clip-Space），再用背景捲動（Scroll）的方式來捲動降雨貼圖。在後製處理（Post-Processing）前，降雨部分被分成好幾層（Layer）來繪製，每一個層可以指定不同的降雨速度，最後再對降雨效果做模糊化的處理，建立出下雨天朦朧的景象（如圖 6 a所示）。



(a) (b)

圖 6 (a)雨夜的降雨效果 (b)使用的降雨貼圖

(2) 雨滴

要模擬從屋簷滴落的雨滴或是從排水管噴出的水滴（如圖 7a所示），這部分是利用粒子系統來實現，對於每一個粒子都會對應到一張雨滴貼圖，且粒子移動速度會影響雨貼圖的長度。每個粒子也都包含雨滴的法向量貼圖（如圖 7b所示），當實際模擬時會用來計算光線從空氣進入水滴的反射和折射。

(a) (b)

圖 7 (a)局部區域使用粒子系統的雨滴效果 (b)雨滴的法向量貼圖

第三章 GPU粒子系統和 3D繪圖引擎之整合

論文最後完成的雨雪效果必須和賽車遊戲整合在一起，因此我們直接在遊戲公司使用的 3D 繪圖引擎上進行程式開發，如此方便將美術設計人員提供的賽車場景、模型、貼圖等美術檔案載入程式中運用。本章將會介紹我們如何基於 3D 繪圖引擎之架構，實現利用 GPU 計算的即時粒子系統。

3.1 OGRE 3D繪圖引擎簡介

OGRE（Object-Oriented Graphics Rendering Engine）是一個被廣泛使用的即時 3D 繪圖引擎，它能支援 DirectX 及 OpenGL 兩種繪圖 API，並可以在 Windows、Linux、Mac OSX 等不同作業系統上跨平台執行。OGRE 同時也是開放原始碼的繪圖引擎，完全使用 C++程式語言撰寫，配合各種設計模式（Design Pattern）的運用，讓遊戲開發者不但可以清楚了解引擎內部的運作方式，也能夠有效率擴充原來引擎所提供的功能。由於只要遵循 LGPL 規範就可免費應用在商業行為上，在國外已有多款商業遊戲是以 OGRE 為繪圖核心進行開發。

3.2 GPU粒子系統架構圖

我們依據Kolb等人[10]的方法，將粒子運動會改變的粒子資料（位置、速度、生存時間）存入貼圖，過程中再利用著色器（Shader）程式取出貼圖中的粒子資料加以計算。圖 8是整個GPU粒子系統運作的架構圖，我們在這個架構下實現了從粒子產生、粒子運動、粒子描繪到最後粒子消失的處理。

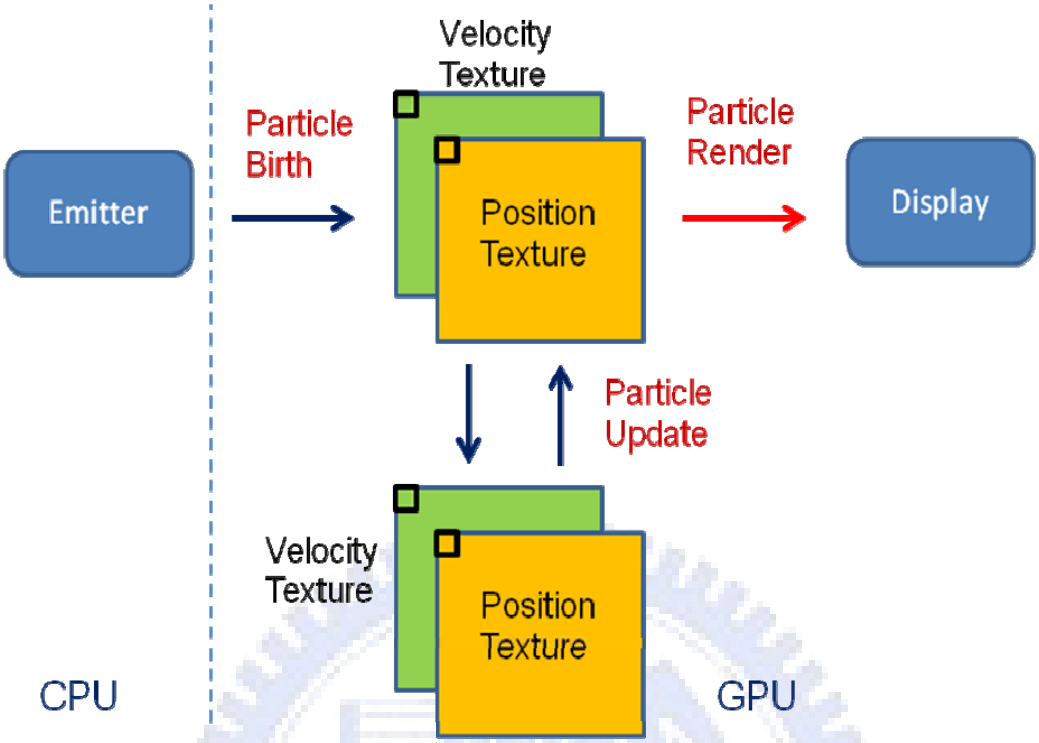


圖 8 GPU 粒子系統的架構圖

在程式流程開始，先從主程式中設定粒子發射器（Emitter）的參數。可調整參數包含了發射器位置、發射器方向、發射器範圍、粒子初速度與粒子生存時間等等。接著在每個畫格（Frame）將粒子發射器新產生的粒子資料存入兩張貼圖（Position Texture, Velocity Texture）中。粒子運動過程，由存現在粒子資料的兩張貼圖取出粒子位置、速度、生存時間，計算後將結果存入另兩張貼圖中，如此反覆進行。最後，依粒子位置貼圖的資料將粒子描繪在電腦螢幕上。

3.3 粒子生成

當粒子產生出來，要用GPU來處理粒子運動的計算，因此需要每個粒子的位置、速度、生存時間資料儲存在貼圖的像素中。我們使用RGBA-Float格式的貼圖，RGB三個分量儲存粒子位置或速度的(x, y, z)向量值，位置貼圖的Alpha分量則儲存粒子剩餘的生存時間。如圖 9所示，貼圖的大小代表此粒子系統最多可存在的粒子數目。

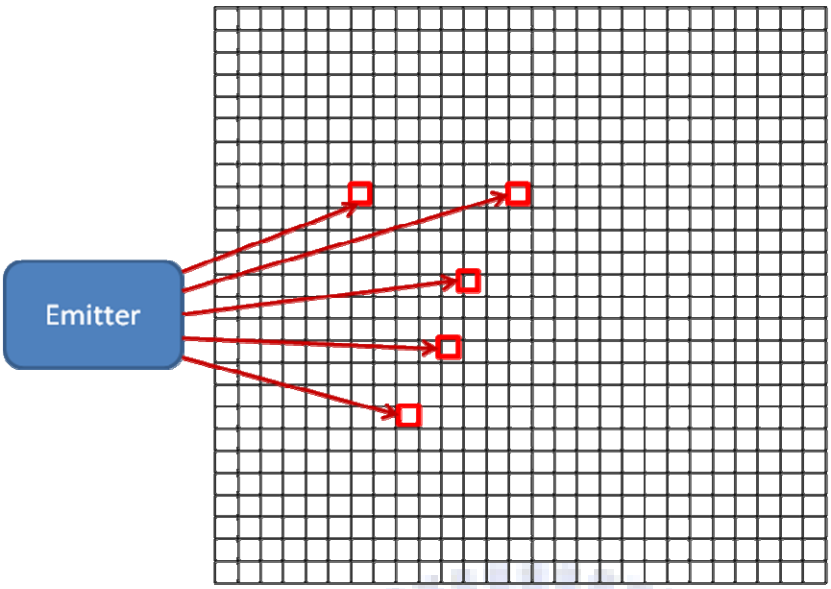


圖 9 將發射器產生的粒子資料寫入空的貼圖位置

在粒子模擬過程，新粒子從發射器不斷的被產生，舊粒子隔一段時間後也會自行消失。每個粒子在消失前都會佔據貼圖上某一像素位置，直到粒子生存時間小於零時該像素才被釋放。我們在程式裡使用佇列（Queue）記錄空出來的像素位置，讓未消失的粒子資料不會被新產生粒子覆蓋掉。

將粒子資料寫入貼圖的動作，為了避免 Lock 整張貼圖而降低系統執行效能，我們使用從程式傳頂點串流（Vertex Stream）的方法把粒子資料寫入空的像素位置。這部分處理步驟如下：

1. 用 Ogre 繪圖引擎的 ManualObject 物件，將粒子資料寫入 Vertex Shader。
2. 將存粒子位置和速度的兩張貼圖設成著色目標（Render Target）。
3. 用引擎 Compositor Render_Scene 的方法，把粒子資料寫到空的貼圖位置上。

實作方法如圖 10所示。首先建立 2D的ManualObject，其所屬的 2D座標空間是-1~1，如要把粒子資料寫入貼圖座標位置(i , j)，則需將寫入ManualObject的位置從 0~1 轉換至 -1~1 的座標空間。再使用引擎Compositor Render_Scene方法描繪ManualObject物件，把粒子資料寫入貼圖。

```
ManualObject* manual;
…
manual->position(2*i-1, 2*j-1, 0);
manual->textureCoord(p->position);
manual->textureCoord(p->velocity);
manual->textureCoord(p->timeToLive);
…
```

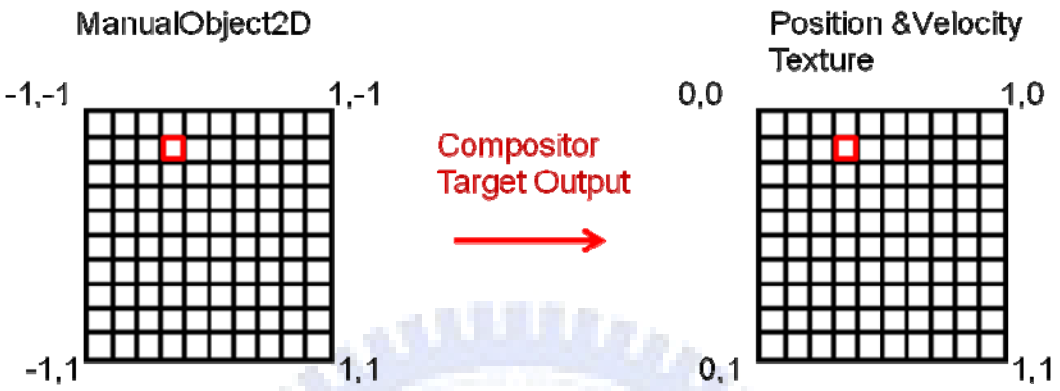


圖 10 使用 ManualObject 將粒子資料寫入空的貼圖位置

3.4 粒子運動

我們的GPU粒子系統利用四張貼圖儲存粒子資料，分別儲存前一個時間點的粒子位置與速度，以及粒子運動後的位置與速度。粒子運動的計算使用歐拉積分法（Euler Integration），如圖 11所示。

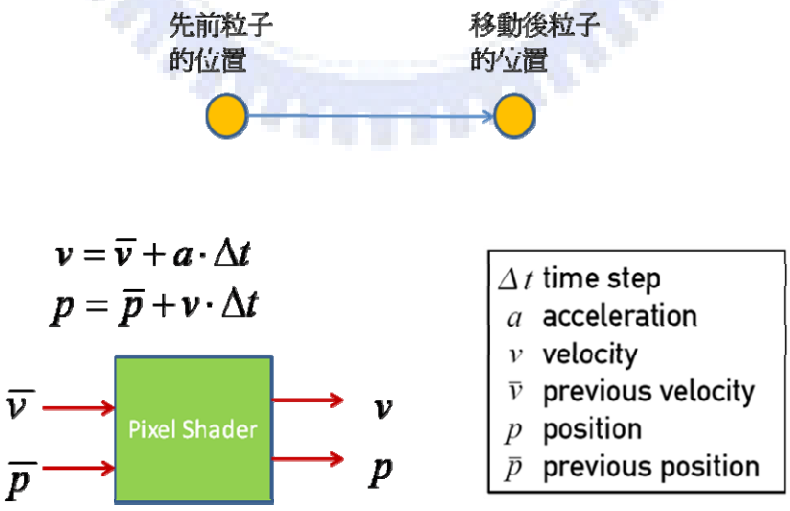


圖 11 歐拉積分法

我們用多重著色目標（Multiple Render Targets，簡稱 MRTs）之技術，將 Pixel Shader 計算出之粒子運動結果，同時寫到儲放粒子運動後的位置與速度兩張貼圖，如此避免將粒子位置與速度的計算分成兩個 Pass 來處理，可以加快計算粒子運動的時間。

3.5 粒子描繪

這部分是把粒子描繪在電腦螢幕上，完整的描繪流程如圖 12所示。首先在Vertex Shader使用頂點紋理擷取（Vertex Texture Fetch，簡稱VTF）之技術，從儲存粒子位置之貼圖取得每個粒子位於 3D場景中的位置，再透過各種粒子材質檔描繪出雪花或雨滴效果。

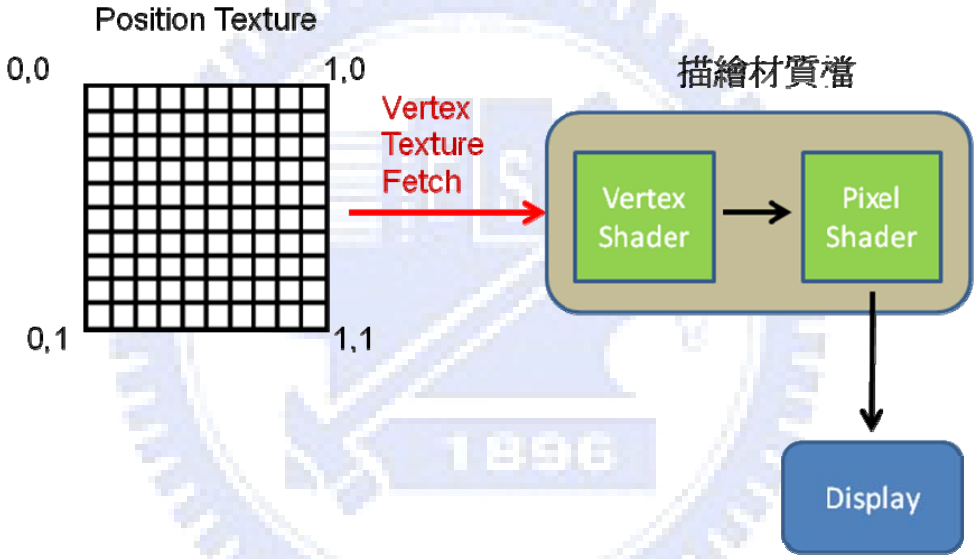


圖 12 粒子描繪的流程圖

第四章　雨和雪之描繪技術

在這一章會使用已開發的 GPU 粒子系統，產生降雨、飄雪及車輪捲起雨水（雪花）之效果。我們處理粒子描繪的方法有分為兩種：Point Sprite 和 Billboard，會依據所產生效果之需求而選擇兩者之中較適合的一種方法來實現。

4.1 飄雪之描繪

飄雪效果我們使用Point Sprite方法來描繪，在粒子描繪材質檔中指定美術設計人員提供的雪花貼圖（如圖 13所示），並在Shader程式裡計算環境光源對雪花材質的影響。

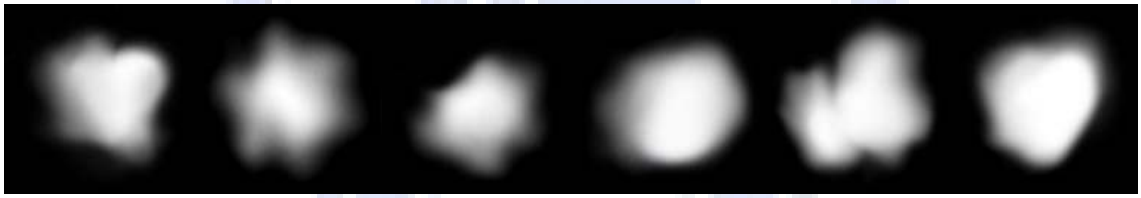


圖 13 美術設計人員提供的雪花貼圖

從天而降的雪花粒子，我們希望能產生在空中飄盪的效果，因此在計算粒子運動的Shader程式中加入一張Noise Map（如圖 14所示）去改變粒子x與y方向之移動路徑。

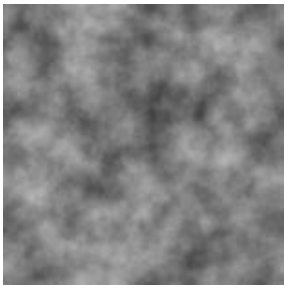


圖 14 雪花擾動的 Noise Map

4.2 降雨之描繪

降雨效果則是用Billboard的方法來描繪，主要考量是可以方便調整雨滴的長度和粗細。在Billboard的方法裡，我們有對雨滴與視角的投影效果做處理，使雨滴貼圖的法向量（Normal）不會一直正對著攝影機（Camera）的方向，避免當視角朝上看雨滴落下時，雨滴貼圖朝著Camera飛過來的不自然現象。圖 15分別展示用Point Sprite與Billboard方法描繪雨滴的效果。

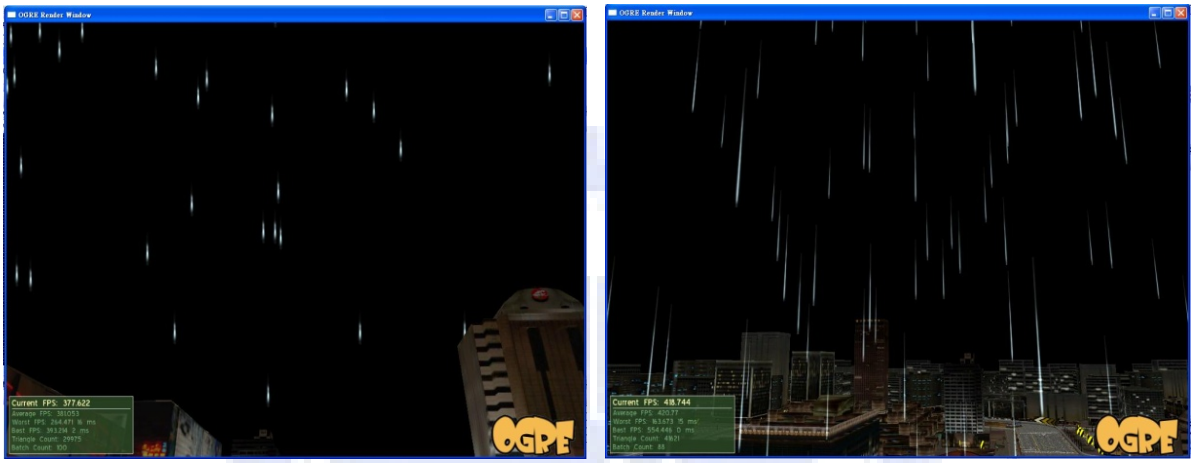


圖 15 Point Sprite 和 Billboard 描繪雨滴的效果

在粒子系統初始化時，我們就會依據儲存粒子位置與速度資料的貼圖大小產生相對應數量的空間點，每一個空間點會知道其對應到存粒子資料貼圖的座標位置。使用Point Sprite的方法，空間點的數量等於存粒子資料貼圖的大小（128 x 128 的貼圖會產生 16384 個空間點），而Billboard的方法則是此貼圖大小的 4 倍，我們用 4 個空間點來建構一個Billboard的面（如圖 16所示）。

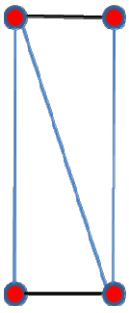


圖 16 四個空間點建構一個 Billboard

另外，使用Billboard的方法，我們可以在粒子材質檔裡調整風力向量的x與z分量來改變Billboard的角度，可形成雨滴受風力改變行進方向的效果（如圖 17所示）。

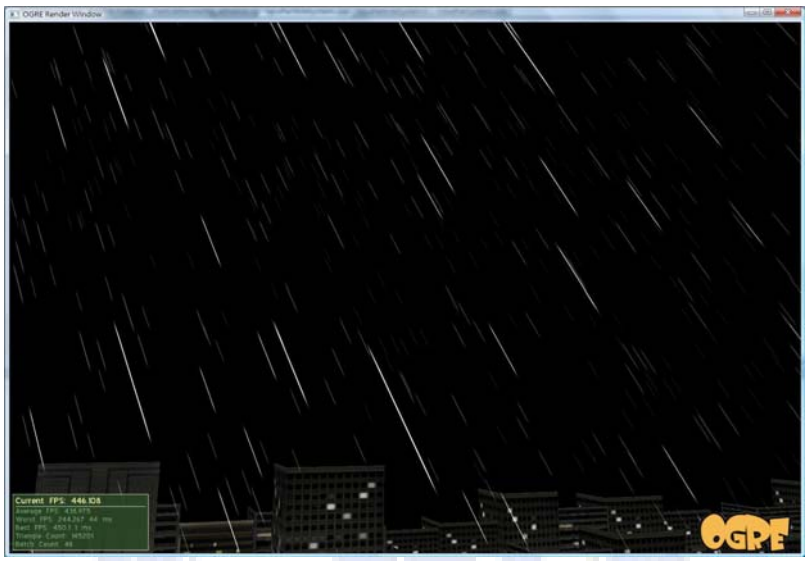


圖 17 Billboard 受風力改變方向

Billboard 平面是使用 Vertex Shader 程式建構，以取自粒子資料貼圖的粒子位置為中心點，依據材質檔設定的 Billboard 長寬與 Camera 視角方向計算出四個頂點的位置。

4.3 車輪捲起雨水(雪花)之描繪

在Point Sprite的描繪上，我們有建立出材質貼圖座標旋轉的效果（如圖 18所示）。主要會應用在如車輛激起雨水與捲起雪花上，使產生出近似氣體捲動的效果。實際做法是在Shader程式裡乘上一個旋轉矩陣。

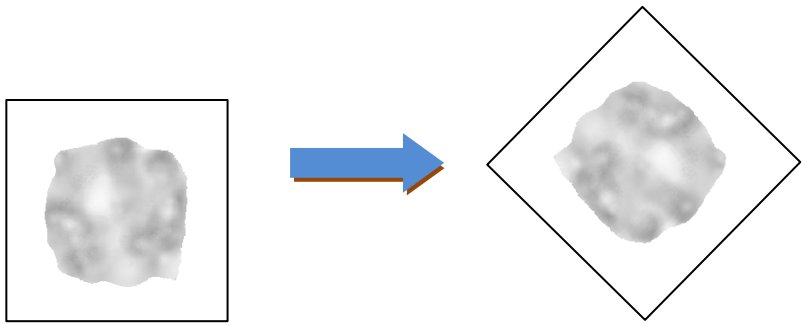


圖 18 Point Sprite 貼圖座標旋轉

最後，對本章提到的兩種粒子描繪方法做個比較。

Point Sprite 描繪方法：

1. 建立的空間點數量較少，可以節省記憶體的空間。
2. 少了由點建構面的步驟，在計算處理上更為快速。

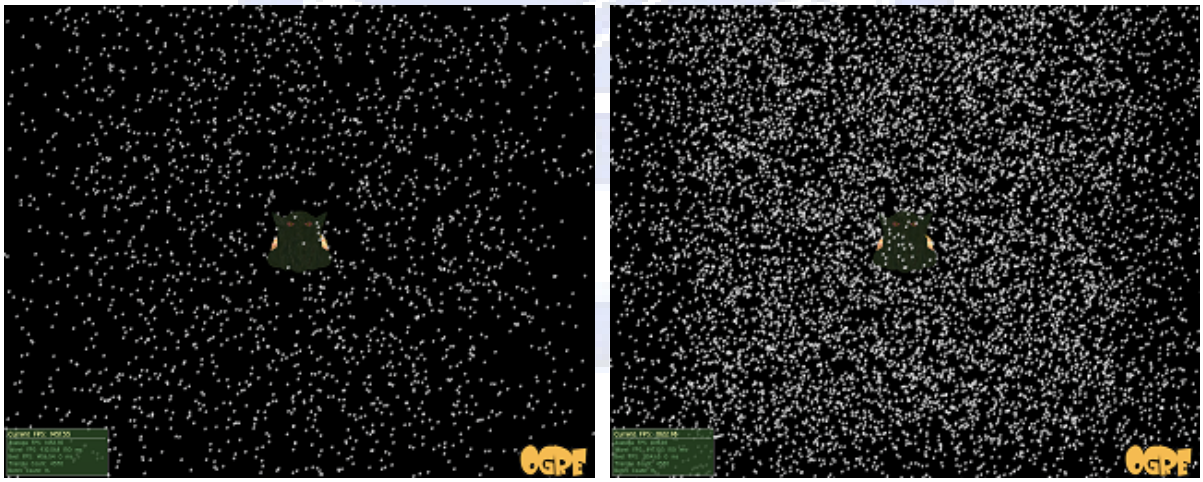
Billboard 描繪方法：

1. 可彈性調整粒子貼圖長寬。
2. 由於粒子貼圖有經過投影處理，貼圖法向量不會一直正對著 Camera 方向，有些效果看起來較為自然。

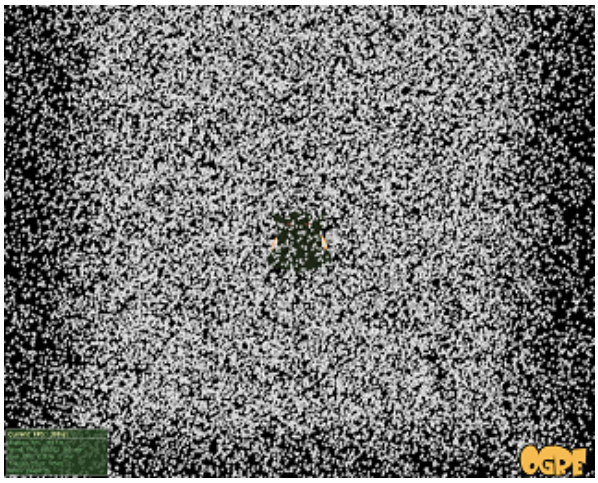
第五章　實作結果與討論

系統的開發環境，硬體部分使用配備 Intel Core2 Extreme QX9650 3.0GHz 處理器、DDR2-800 8GB 主記憶體、以及 Nvidia GeForce 9800GTX 顯示卡的電腦。軟體部分使用 Windows Vista 64 位元作業系統，OGRE 3D 繪圖引擎 1.4.8 版本。以下的程式執行效能皆是使用 OGRE 引擎的 OpenGL 描繪系統與 1280x1024 全螢幕輸出畫面下測得。

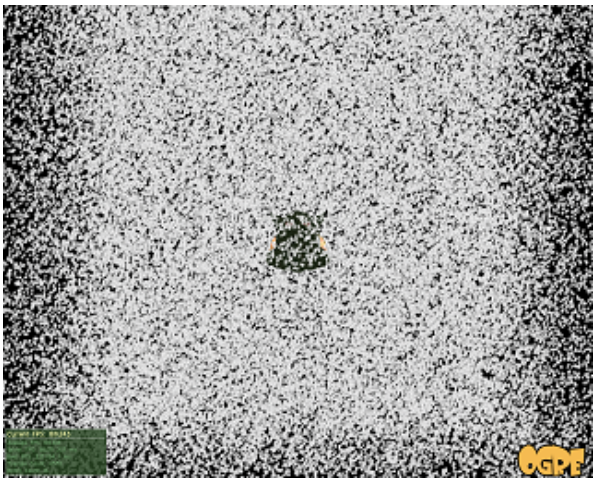
5.1. GPU粒子系統執行效能



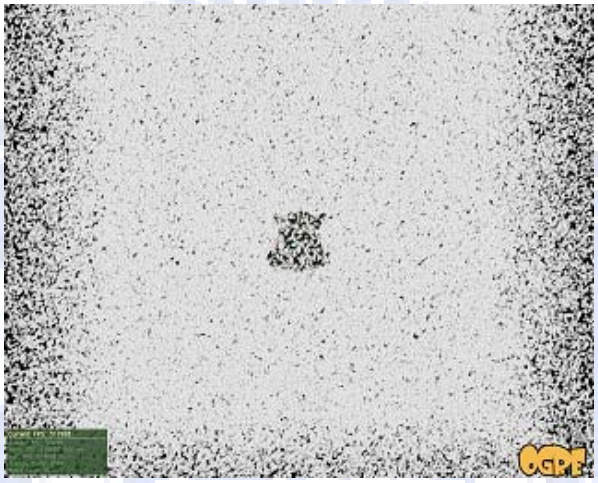
(a) 4096 個粒子　　(b) 16384 個粒子



(c) 65536 個粒子



(d) 262144 個粒子



(e) 1048576 個粒子

圖 19 使用 Point Sprite 描繪方法

表 1列出粒子數目不同時，使用Point Sprite與Billboard描繪方法其程式執行效能之FPS。

n	4,096	16,384	65,536	262,144	1,048,576
Point Sprite	1452	1023	390	189	52
Billboard	1439	1118	325	58	14

表 1 Point Sprite 與 Billboard 描繪方法的效能（FPS）比較

5.2 雨雪效果與賽車遊戲之整合

我們將本論文所開發的 GPU 粒子系統與雨雪描繪技術，配合其它的雨天描繪技術包含：「地面濕滑反射」、「地面強光反射」、「錐形車燈繪製效果」、「Rain Streak 描繪技術」與「車窗雨滴痕模擬」，整合到遊戲美術人員所提供的三個賽車場景中，最後實現的結果分別展示如下。

場景一：夜晚台北

使用技術

- ✓ GPU 粒子系統
- ✓ 地面濕滑反射效果
- ✓ 地面強光反射效果
- ✓ 錐形車燈繪製效果
- ✓ 車輪捲起雨水描繪效果
- ✓ Rain Streak 描繪技術

呈現效果

圖 20 夜晚台北場景畫面一

圖 21 夜晚台北場景畫面二

圖 22 夜晚台北場景畫面三

場景二：雨天台北

使用技術

- ✓ GPU 粒子系統
- ✓ 車窗雨滴痕模擬

呈現效果

圖 23 雨天台北場景畫面一

圖 24 雨天台北場景畫面二

圖 25 雨天台北場景畫面三

場景三：下雪台北

使用技術

- ✓ GPU 粒子系統
- ✓ 車輪捲起雪花描繪效果

呈現效果

圖 26 下雪台北場景畫面一

圖 27 下雪台北場景畫面二

圖 28 下雪台北場景畫面三

最後，在把使用技術全打開之情況下，我們整理出三個場景各自所測試出的效能數據（如表 2所示）。

	Average FPS	Best FPS	Worst FPS
夜晚台北場景	73.4	107.8	61.8
雨天台北場景	245.7	275.7	211.7
下雪台北場景	338.9	365.6	308.7

表 2 不同賽車場景的執行效能

第六章　結論與未來發展

本論文在國內外遊戲業界普遍使用的 OGRE 3D 繪圖引擎上，發展出一個運用 GPU 計算之粒子系統。將每個粒子的資料儲存於貼圖中，再使用 Shader 程式依歐拉積分法求出粒子運動後的位置。在此粒子系統的架構下可以實現各式各樣的粒子特效，現階段已有實現降雨、飄雪、車輪捲起雨水或雪花之效果，並與賽車遊戲的 3D 場景進行整合。而粒子描繪方法分成 Point Sprite 與 Billboard 兩種，可依據實際呈現效果需求選擇較適合的方法使用。

在未來發展方面，仍有可以進一步研究的方向。

(1) 粒子系統與場景的互動：

我們的 GPU 粒子系統還沒有處理對賽車場景中物體的碰撞問題，所以現在的粒子會直接穿透所接觸的物體。這部分有想到的方法是由美術人員額外提供場景的 Height Field 與賽車模型的 Bounding Box，如此我們就可以在粒子運動時處理相交測試的計算。

(2) 其它的粒子效果：

我們現在只有使用 GPU 粒子系統模擬雨及雪的效果。接下來視賽車遊戲的需求，可以加入如煙霧、爆炸等其它粒子特效。

(3) 使用 Geometry Shader 方法實作：

在最新釋出的 OGRE 1.6.0 RC1 版本，已有支援 OpenGL 的 Geometry Shader 功能。現在我們處理粒子描繪的方法，需要由四個空間點經由四次 Vertex Shader 計算來建構 Billboard，如果用 Geometry Shader 方法來建 Billboard 將減少空間點的使用數量，也可減少 Vertex Shader 處理的指令數。

參考文獻

[1] Chen, J. X., Fu, X. and Wegman, J. 1999. Real-Time Simulation of Dust Behavior Generated by a Fast Traveling Vehicle. *ACM Transactions on Modeling and Computer Simulation,* Vol. 9, Issue 2, 81-104.

[2] Drone, S. 2007. Real-Time Particle Systems on the GPU in Dynamic Environments. *ACM SIGGRAPH 2007 Courses*, pp. 80-96

[3] Fearing, P. 2000. Computer Modeling of Fallen Snow. *ACM SIGGRAPH 2000,* pp. 37-46.

[4] Feldman, B. E. and O'Brien, J. F. 2002. Modeling The Accumulation of Wind-Driven Snow. *ACM SIGGRAPH 2002 Sketches.*

[5] Garg, K. and Nayar, S. 2004. Detection and Removal of Rain from Videos. *IEEE CVPR 2004*, pp.528-535.

[6] Garg, K. and Nayar, S. 2006. Photorealistic Rendering of Rain Streaks. *ACM SIGGRAPH 2006.*

[7] Kaneda, K., Kagawa, T. and Yamashita, H. 1993. Animation of Water Droplets on a Glass Plate. Computer Animation 1993, pp 177-189.

[8] Kaneda, K., Ikeda, S. and Yamashita, H. 1999. Animation of Water Droplets Moving Down a Surface. Journal of Visualization and Computer Animation 1999, Vol. 10, Issue 1, 15-26.

[9] Kifer, P., Segal, M. and Westermann, R. 2004. Uberflow: A GPU-Based Particle Engine. *Proceedings of Eurographics Symposium on Graphics Hardware 2004*, pp. 115-122.

[10] Kolb, A., Latta, L. and Rezk-Salama, C. 2004. Hardware-Based Simulation and Collision Detection for Large Particle Systems. *Proceedings of Eurographics Symposium on Graphics Hardware 2004*. pp. 123–131.

[11] Langer, M. S. and Zhang, L. 2003. Rendering Falling Snow using an Inverse Fourier Transform. *ACM SIGGRAPH 2003 Sketches.*

[12] Langer, M. S., Zhang, L., Klein, A. W., Bhatia, A., Pereira, J. and Rekhi, D. 2004. A Spectral-Particle Hybrid Method for Rendering Falling Snow. *Proceedings of Eurographics Symposium on Rendering 2004*, pp. 217-226.

[13] Nishita, T., Iwasaki, H., Dobashi, Y. and Nakamae, E. 1997. A Modeling and Rendering Method for Snow by Using Metaballs. *Computer Graphics Forum* Vol. 16, No. 3, 357–364.

[14] Reeves, W. T. 1983. Particle Systems - A Technique for Modeling a Class of Fuzzy Objects. *ACM SIGGRAPH 1983*, pp. 91-108.

[15] Rousseau, P., Jolivet, V. and Ghazanfarpour, D. 2006. Realistic Real-Time Rain Rendering. *Computers & Graphics*, Vol. 30, Issue 4, 507-518.

[16] Starik, S. and Werman, M. 2003 Simulation of Rain in Videos. *Texture Workshop, ICCV 2003*.

[17] Sumner, R., O'Brien, J. F. and Hodgins, J. K. 1999. Animating Sand, Mud and Snow. *Computer Graphics Forum* Vol. 18, No. 1, 17–26.

[18] Tatarchuk, N. 2006. Artist-Directable Real-Time Rain Rendering in City Environments. *ACM SIGGRAPH 2006 Courses*, pp. 23-63.

[19] Van Der Burg, J. 2000. Building an Advanced Particle System. *Game Developer Magazine*, Mar. 2000, pp. 44-50.

[20] Wang, C., Wang, Z., Xia, T. and Peng., Q. 2006. Real-Time Snowing Simulation. *The Visual Computer*, Vol. 22, No. 5, 315-323.

[21] Wang, H., Mucha, P. J. and Turk, G. 2005. Water Drops on Surfaces. *ACM Transactions on Graphics*, Vol. 24, Issue 3, 921-929.