한국게임학회 논문지
제3권 제1호 2003년 3월

텍스쳐 매핑과 파티클 시스템을 이용한 눈 내리는 전경의 애니메이션

이 상락*
컴퓨터 공학과 부교수
srlee@incheon.ac.kr

Animation of Snowy Scenery Using Texture Mapping and Particle Systems

Sang-Rak Lee
Incheon Univ. Computer Dept.

요 약

본 논문에서는 텍스쳐 매핑과 파티클 시스템을 이용하여, 눈 내리는 장면의 애니매이션을 OpenGL로 구현하는 두가지 방법을 제시하고 이들의 실행 결과를 비교 분석하였다. 하나는 glPointSize()함수를 사용하는 것이고 또 다른 하나는glutSolidSphere()함수를 사용하는 것이다. 실행 결과 전자가 후자보다 실행 결과가 빨라서 속도가 느린 PC나 내리는 눈의 개체 수가 많은 경우에 사용하기에 더 적절한 방법이라 생각된다. 또 전자의 방법을 이용한 구현 과정에서 눈의 모양과 눈의 색깔의 제어가 서로 상충되는 사실을 발견하고 이미지를 조작하여 그것을 해결하는 방법을 제시한다.

Abstract

This paper deals with animation of snowy scenery using texture mapping and particle system. Two different OpenGL programs were prepared and compared for implementation and evaluation of their performance. One used glPointSize(), and the other used glut SoildSphere(). Result of implementation revealed that the first was faster than the second. So the first is considered to be more useful for the slow PC. In relation to the first method, I found that control of shape of snow flake conflicts with that of color. Fortunately, I found out an esoteric method which handles the back ground image for scene that is used in texture mapping. I present that method.

Key Words : Textuer mapping, Particle Systems, OpenGL, animation, snowy scenery

1. 서론

구름, 연기, 나무나 숲, 의복, 폭발, 강설, 강우 등과 같은 자연 물체나 자연 현상의 모델링은 오늘날 컴퓨터 그래픽에서 새로운 관심의 대상이 되고 있다. 따라서 이들에 대한 연구도 활발히 이루어지고 있으며, 대상에 따라 적절한 여러 가지 모델링 기법이 제시되고 있다[1][2].

자연현상 중에서도 눈은 특히 사람들의 관심을 끌기에 충분하다. 눈이 내린 후면 만물은 은빛의 담요로 뒤덮이고 세상은 환상의 세계로 변한다. 또 눈이 휘날리는 설경은 사람들의 마음을 황홀하게 만들기도 한다.

눈은 이와 같이 우리에게 가까이 있음에도 불구하고 컴퓨

* 이 논문은 2002년도 학술 연구 조성비 지원에 의해 수행되었음
정회원 : 인천대학교 컴퓨터 공학과

터 그래픽에서 이와 관련된 연구나 작품은 크게 많지 않다. 일반적으로 눈 덮인 전경은 텍스쳐 매핑을 이용하면 쉽게 렌더링 할 수 있다. 또 눈이 내리는 장면은 파티클 시스템을 이용한다. 그러나 지금까지의 연구들 중에서 이들 두 기법을 동시에 사용하여 눈 내리는 장면을 렌더링 한 것은 발견하지 못 하였다. 특히 파티클 시스템과 관련된 지금까지의 연구들은 대부분 검은색 바탕에서 자연물체를 렌더링하고 있기 때문에 텍스쳐 매핑을 한 상태에서 물체의 색깔에 대한 언급은 없다[2].

본 논문에서는 텍스쳐 매핑과 파티클 시스템을 이용하여, 눈 내리는 장면을 렌더링 하였다. 그런데 렌더링 과정에서 눈의 모양과 눈의 색깔을 제어하는 것이 서로 상충되어, 이들을 의도적으로 제어하는 것이 쉽지 않았다. 그러나 우연하게도, 이미지를 조작하여 눈의 색깔을 의도적으로 제어할 수 있는, 다소 비법에 가까운 방법을 발견하였다.

본 연구에서는 눈 내리는 장면을 구현하기 위하여 두가지 OpenGL 프로그램을 준비하였다. 하나는 glPointSize()함수를 사용하는 것이고 또 다른 하나는 glutSolidSphere()함수를 사용하는 것이다. 그리고 이들의 실행 결과를비교 분석하여 장, 단점을 제시한다.

본 논문은 다음과 같이 구성되었다. 2장에서는 눈과 관련된 선행 연구를 살펴본다. 3장에서는 배경을 나타내기 위한 텍스쳐 매핑과 이미지의 준비에 대하여 기술한다. 4장,5장,6,7장은 본 논문의 주요 내용으로, 눈 내리기를 위한 두 가지 방법을 소개하고 텍스쳐 매핑을 활성화 시킨 상태에서 눈의 모양과 색깔의 관계를 살펴본다. 마지막으로 두 가지 방법의 실행 시간을 비교한다.

2. 선행 연구

눈은 우리들과 매우 친밀함에도 불구하고 컴퓨터 그래픽을 위한 눈의 모델링을 위한 연구는 별로 없다. 이들 연구를 두 가지로 분류하면 하나는 눈이 쌓인 장면을 만들어 내는 것이고, 또 하나는 눈이 내리는 것과 관련된 것이다.

Premoz 등 [4]은 눈으로 덮인 전경의 산악지형을 만들어 내었다. 항공 촬영 사진에 약간의 눈이 덮이도록 하였다. 그러나 이들의 방법은 항공사진 촬영의 속성상 세세한 부분의 눈이 쌓인 모양을 만들어 내기에는 제약이 있다.

Muraoka 등 [5,6]은 전경에 볼륨 요소를 낙하시켜 두꺼운 눈이 두껍게 내린 장면을 만들어 낼 수 있었다. 그러나 불행하게도 제한된 수의 입자를 사용하여 전체의 전경을 덮으려 하다보니 하나의 입자가 확대되어 기본이 되는 표면의 세부사항이 가려져 제대로 나타나지 못하는 단점이 있다.

Paul Fearing[7]은 누적 모델(accumulation model)과 안정 모델(stability model)을 도입하여 눈이 쌓이는 장면을 렌더링 하였다. 특히, 누적 모델링은 사이트(site)라 불려지는 표면 다각형에서 입자를 하늘 방향으로 쏘아 올려 충돌 검사를 하여 눈이 내릴 때, 중간에 차폐물이 존재하는지를 검출하였으며, 안정 모델에서는 눈이 일차의 차폐물에 누적된 후 어느 정도를 넘으면 눈사태를 일으켜 그 밑으로 떨어지는 현상을 해결하였다.

이 이외에 눈이 쌓인 정지된 장면과 관련된 연구는 [8]-[10]에서 찾아볼 수 있다.

움직이는 눈송이의 렌더링은 Karl Sims[8]에서 찾아볼 수 있다. 그러나 Sims는 눈의 움직임에만 관심이 있을 뿐 눈의 색깔에 대한 깊은 언급은 없다. 또, [3]에서는 java로 구현한 눈 내리는 장면을 볼 수 있지만, 눈의 수가 많아지면 속도감 때문에 현실감이 떨어진다.

3. 텍스쳐 매핑과 배경 이미지의 준비

눈이 내린 배경은 눈이 쌓여 있는 이미지를 준비하고 텍스쳐 매핑을 이용하여 만들어 낸다. 텍스쳐 매핑기법은 많은 그래픽 관련 서적에 소개되고 있다. 특히 [12][14]에서 자세히 설명되고 있으며 [11]는 여러 가지 텍스쳐 매핑 기법을 조사 제시하고 있다. 또 Davis Ebert 등[13]은 이미지 기반의 텍스쳐 매핑 기법을 자세히 소개하고 있다.

OpenGL에서 텍스쳐 매핑을 구현하기 위해서는 파일 형태가 bitmap 형식이어야 하고, 이미지의 크기가 2^n이 되어야 하며 최대 256×256이 허용된다. 따라서 이미지의 크기가 맞지 않거나 파일 형태가 bitmap이 아닐 경우에는 Photoshop이나 Flash등을 이용하여 이미지를 변형하여 사용한다. 256×256 크기의 이미지를 텍스쳐 매핑을 한 후 디스플레이 화면의 창을 다시 확대하면 이미지의 질이 떨어져 흐리게 보인다. 이 점을 개선하려면 원하는 이미지를 256×256의 사이즈를 가지는 작은 이미지로 쪼갠 후 이를 각각의 다각형에 매핑하고 OpenGL의 Viewport 기능을 이용하여 화면에 나타내면 해결될 수 있을 것이다.

4. 눈 내림

(1) 눈의 이동

눈의 움직임의 애니메이션은 파티클 시스템을 이용하였다. 파티클 시스템기법은 [2][8] 등에서 찾아볼 수 있다.

본 논문에서 렌더링하고자 하는 눈송이의 속성은 크기, 위치(x, y, z좌표)이며 다음과 같은 구조체 형태로 표현하였다.

```
typedef struct{
        float size;
        float x, y, z;
}snow;
```

눈의 위치는 시간이 지남에 따라 조금씩 아래로 떨어지게 하기 위하여 약간의 변화를 주어야 하며, 이를 위해 dx, dy, dz변수를 도입하며,각각의 눈송이마다 변위되는 크기를 달리하기 위하여 이들은 랜덤으로 처리한다. 10개정도의 dx, dy, dz 각각의 변위량을인위적으로 설정하여 배열에 저장한 후 랜덤으로 첨자 하나를 생성하여 변위량을 참조하게 하면 각각의 변위량을 랜덤으로 발생시키는 것보다 시간을 단축할 수 있다.

dx, dz는 눈이 수직으로 떨어지는 것을 막으며, 또 dz는 OpenGL의Perspective기능과 함께 사용하여 시야에서 보이다 사라지다 하는 효과를 만들어 낼 수 있다.눈을 내리게 하는 애니메이션을 위해 아래 설명과 같은 기능을 가진함수를 정의하여 사용하였다.

```
initialization( );
: 초기에 원하는 개수만큼의 눈을 생성하여 배열에 저장
flake_generator( );
: 눈송이가 바닥에 다 떨어지거나 내리는 도중 옆으로 벗어나면 다른 눈송이를 생성
flake_manager( );
: 눈송이가 바닥에 닿았는지, 내리는 도중 창 범위를벗어났는지를 확인
snow_falling( );
: 눈이 내리는 전체과정을 통합하여 animation을 생성
```

(2)눈 모양의 처리

눈의 모양과 하강을 처리하기 위해서는 두 가지 방법이 사용될 수 있다.

① 점으로 처리

눈을 하나의 점으로 처리하여 크기는 점의 크기로 조절한다. OpenGL에서 제공되는 GL_Point_Size 및 GL_Vertex( ) 를 이용한다.

② 작은 구로 처리

눈을 하나의 구로 나타내고 아래로의 이동은 평행이동 변환을 이용한다. 즉 , GlutSolitSphere( ) 함 수 와 glTranslate( )를 이용한다.

(3) 눈 내리기 animation

눈이 내리는 장면을 애니메이션으로 나타 내기 위한 알고리즘은 다음과 같다.

```
initialization( );
snow_falling( );
 {
        movement of snow flake;
        flake_manager( );
         {
        눈송이가 바닥에 닿거나 창을 벗어나면
          flake_generator( );
        }
 }
```

5. 눈의 모양과 색깔

앞에서 설명한 눈을 나타내는 방법 중에서 GlutSolitSphere( ) 함수를 사용하여 눈을 구로 나타내는 방법을 사용하면 눈의 모양이나 색깔을 의도한대로 제어하는 데는 큰 문제가 없다. 그러나GL_Point_Size( ) 와 GL_Vertex( )를 이용하여 눈을 점으로 나타내는 경우는 눈의 모양과 색깔의 제어가 서로 상충되어 흰색의 그럴듯하게 보이는 눈을 만들어 내지 못한다. 다시 말하면 흰색의 눈송이가 만들어지도록 하기 위하여 눈송이의 색깔을 glColor(1,1,1)로 지정하여도 TEXTURE와 POINT_SMOOTH의 상태를 Enable 상태로 하거나 Disable 상태로 하는상황에 따라 눈송이의 색깔이 의도한 흰색으로 나타나지 않는 현상을 보인다.물론 텍스쳐 매핑을Disable 한 후, 눈을 내리게 하면 흰색의 눈을 얻을 수는 있다. 그러나 이 경우 비록

Point_Smooth을 활성화하여도 둥그런 모양의 눈이 만들어지지 않는다.

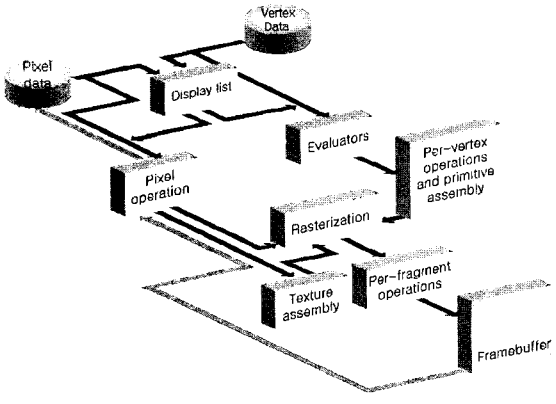
아래의 〈표1〉은 이 현상을 정리한 것으로 흰색의둥그런 눈송이를 나타내는 것은 불가능하다.

TEXTURE	POINT_SMOOTH	결 과	
		눈의 모양	나타난 눈의 색깔
Enable	Enable	원형	흰색이 아님
Enable	Disable	사각형	흰색이 아님
Disable	Enable	사각형	흰색
Disable	Disable	원형	흰색이 아님

〈표 1〉상태의 설정에 따른눈의 모양 및 색깔의 관계
〈Table 1〉 Relation between shape and color of snow flake

위의 현상의원인은 본 저자는 지금까지 정확히 알 수가 없다. 다만OpenGL에서 디스플레이 하고자 하는 객체의 처리 과정에 기인하지 않을까 추측하고 있다. 즉, (그림-1) 에서와 같이 텍스쳐 매핑이 처리되기 전에 객체의 컬러를 먼저 계산한다. 그리고 텍스쳐 매핑에 의해 픽셀의 컬러가 결정되면 래스터과정에서 결정된픽셀의 컬러는 이 색깔로 대치된다[15]. 따라서 이러한 현상의 원인은 처리 과정의 파이프 라인에서 텍스쳐 매핑후의 픽셀 색상은 래스터 과정의 색상으로 되돌아 갈 수 없기 때문이 아닌가 추측된다.

다행히도 우리는 이 문제의 해결 방안을우연히 발견하였다. 즉, 배경으로 사용하는 이미지의 네 귀퉁이의 픽셀을 흰

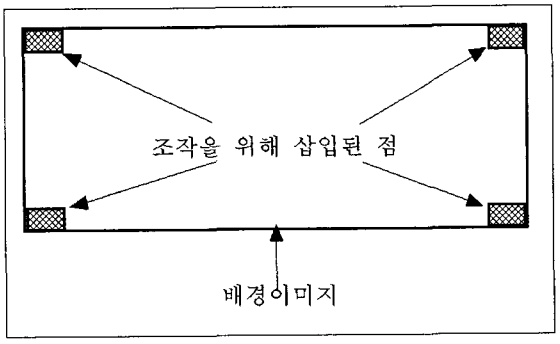


(그림 1) OpenGL에서 처리 순서
(Fig. 1) Order of Operations in OpenGL

색의 점으로 처리한 후 프로그램에서 사용하면 완벽한 희색의 눈송이를 만들어 낼 수 있었다. 나아가서 눈의 색깔은 귀퉁이 찍은 점의 개수에 따라 어떻게 달라지는 가를 알아보기 위하여 아래의 이미지 조작 실험을 하였다.

6.배경 이미지 조작 실험

프로그램에서 한 객체의색깔을 흰색으로설정을 한 후 (그림 2)와전경의 배경 이미지로 사용할 텍스쳐 이미지의 네 귀퉁이에 색깔과 개수를 달리하면서 점을 찍어 넣는다.



(그림 2)이미지 조작을 위한 점의 삽입
(Fig. 2) Insertion of pixel for image manipulation

그리고 조작된 이 이미지를 사용하였을 때객체의 색깔이 무슨 색으로 나타나는지를 Photoshop의 pixel 정보 pallet를 사용하여 조사하였다. 조사 결과는 아래 〈표2〉와 같다.

표는 점의 색깔과 개수를 달리하였을 때 나타나는 물체의 색깔을 Red Green Blue의 요소별로 나타내었으며 각 격자내의 값은 각 요소의 색상을 8 bit를 사용하여 나타낸 값이다.

〈표2〉를 보면 점의 개수가 많을수록 화면상의 물체의 색깔은 점의 색깔에 가까워진다는 것을 알 수 있다.

이러한 관계를 식으로 나타내면 아래와 같다.

$C = C_d + 63 * (4 - n) * C_1 + 63 * (4 - n) * C_2$

C : 물체의 색깔

C_d : 점의 색깔에서 우선 요소의 값 (255)

C_1 , C_2 : 점의 색깔에서 우선 요소가 아닌 다른 두 요소의 값

n : 점의 갯수

Dot		0개	1개	2개	3개	4개
Black	R	255	191	127	63	0
	G	255	191	127	63	0
	B	255	191	127	63	0
Red	R	255	255	255	255	255
	G	255	191	127	63	0
	B	255	191	127	63	0
Blue	R	255	191	127	63	0
	G	255	191	127	63	0
	B	255	255	255	255	255
Green	R	255	191	127	63	0
	G	255	255	255	255	255
	B	255	191	127	63	0

〈표2〉이미지 조작에 따른 객체의 색상 변화
〈Table 2〉Relation between number of dot and object color

7. 눈의 모양을 위한 두 방법의 비교 실험

눈을 나타내는 방법으로는 점으로 나타내는 방법과 구로 나타내는 방법이 있음을 설명하였다.

이 두 방법의 우열을 가리기 위해 실행 시간을 측정 비교해 보았다. 아래 (그림3)는 눈의 개수를 달리 하면서 이 두 방법의 실행시간을 측정한 실험 결과이다.

Dot		0개	1개	2개	3개	4개
Black	R	255	191	127	63	0
	G	255	191	127	63	0
	B	255	191	127	63	0
Red	R	255	255	255	255	255
	G	255	191	127	63	0
	B	255	191	127	63	0
Blue	R	255	191	127	63	0
	G	255	191	127	63	0
	B	255	255	255	255	255
Green	R	255	191	127	63	0
	G	255	255	255	255	255
	B	255	191	127	63	0

(그림3)실행 시간 비교 결과
(Fig. 3) Result of run time comparison

*참고

실행환경

· CPU : Intel Pentium III 667MHz

· Graphic Card chipset : NVIDIA TNT2 M64 (32M)

위의 결과를 보면 눈을 구로 나타내는 방법은 점으로 나타내는 방법보다 시간이 많이 걸림을 알 수있다. 이것은 전자의 방법이 Translate()와 같은 함수를 더 호출하기 때문이다. 더욱이 glutSolidSphere()와 같은 함수는 많은 시간이 소요되기 때문에 전자의 방법은 속도가 느린 PC에서는 적용하기에 적절하지 못할 것으로 생각된다.

8. 결론

본 연구에서는 눈이 내리는 장면을 실제로 렌더링 하는 방법을 소개하고 이 과정에서의 문제점과 해결 방안을 제시하였다. 눈을 구로 처리하면 동그란 모양의 흰 눈이 내리는 전경을 렌더링 할 수 있다. 그러나 이 방법은 눈의 개수가 많아지면 시간이 많이 걸리기 때문에 속도가 느린 pc에서는 이용하기가 부적합하다.

본 연구에서 찾아낸 방법과 같이 눈을 점으로 렌더링 하되 흰색의 둥근 눈을 만들기 위해서 이미지를 조작하는 것이 더 적절한 방법일 수도 있다. 이미지를 직접 조작하는 대신 프로그램에서 이를 처리하여도 될 것이다.

차후에는 OpenGL에서 컬러가 처리되는 과정을 보다 자세히 조사하여 본 논문에서 제시한 방법의 원인을 밝히어 그 정당성을 보이는 연구가 필요하다. 또 그래픽에서 빛의 효과를 주고자 하는 경우 이 논문에서 제시한 바와 같이 이미지를 조작하면 물체의 색을 원하는 대로 제어할 수 있는지의 여부에 관한 연구를 할 계획이다.

참고: 이 논문에서 사용된 프로그램은 화면 보호용 프로그램이나 인터넷 상에서송년 카드로 활용 될 수 있다. 부록으로 OpenGL 프로그램의 전체 구성과 화면에서 디스플레이 되는 애니메이션 장면을 제시한다.

참고 문헌

[1] 이상락, "자연 물체의 모델링 기법에 관한 연구", 인천대학교 공학기술연구, 제17집 제1권, pp, 2002.

[2] William T. Reeves, "Particle System - A Technique for Modelling a class of Fuzzy Object", ACM Transactions on Graphics, Vol 2 No.2, pp91-108, April 1983.

[3]http://my.netian.com/~jjho13/sm-web%20site/java/snow.htm

[4] S. Premoz, W. Tompson, P. Shirley, "Geospecific Rendering of Alpine Terrain", In Eurographics Rendering Workshop, European Association for Computer Graphics, June 1999.

[5] K. Muraoka and N. Chiba, "A Visual Simulation of Melting Snow", The Journal of the Institute of Image Electronics Engineers of Japan, 27(4) : pp327-338, 1998.

[6] K. Muraoka, N. Chiba and I. Ohtawara, "Snow fall Model For Simulating Close Views of Snowy Landscapes", The Journal of the Institute of Television Engineers of Japan, 49(10), pp. 1252-1258, 1995.

[7] Paul Fearing, "Computer Modelling of Fallen Snow", Computer Graphics Processing, Annual Conference Series, 2000, pp. 37-46, 2000.

[8] Karl Sims, "Particle Animation And Rendering Using Data Parallel Computation", Computer Graphics (SIGGRAPH 90 Conference Proceedings), 24(4) : 405-413, August 1990.

[9] T. Nishita, H. Iwasaki, Y. Dobashi, and E. Nakami, "A Modeling And Rendering Method for Snow By Using Metaballs", In Proc EuroGRAPHICS, vol 16, European Association for Computer Graphics, 1997.

[10] R. Summer, J. O'Brien, and J. Hodgins, "Animating Sand, Mud and Snow", In Proceeding of Graphics Interface, pp. 125-132, Canadian Information Processing Society; 1998.

[11] Paw. S. Heckbert, "Survey of Texture mapping", IEEE Computer Graphics and Application, pp. 55-69, Nov 1986

[12] Tomas M ller, Eric Haines, "Real Time Rendering", A K peters, 1999.

[13] David S. Ebert, et. al, "Texturing and Modeling, Ap Professional, 1998.

[14] Alan Watt, "3D Computer Graphics", 3rd Edition, Addison Wesley, 2000.

[15] Mason Woo, Jackie Neider, Tom Davis, and Dave Shreiner, "OpenGL Programing Guide third edition", OpenGL ARB, Feb 2000

부록. snowscene.c를 이용한 눈 내리는 애니메이션 연하 카드

이상락

1971년 서울대학교 과학교육과 졸업(이학사)
1984년 광운대학교 대학원 전자계산학과 졸업(이학석사)
1995년 광운대학교 대학원 전자계산학과 졸업(이학박사)
1983년 ~ 1988년 한국교육 개발원 연구원
1988년 ~ 현재 인천대학교 부교수
관심분야 : 컴퓨터그래픽스의 기하모델링, 애니메이션 등