눈 쌓임 현상의 실시간 표현 연구

송현우*, 장재건*, 류승택*
*한신대학교 컴퓨터공학과
e-mail:hwsong85@gmail.com, jchang@hs.ac.kr, stryoo@hs.ac.kr

A Study on Real-time Representation of Accumulated Snow

Hyun-Woo Song*, Jae-Khun Chang*, Seung-Taek Ryoo*
*Department of Computer Engineering, Hanshin University

요 약

특수 효과를 생산하기 위해 컴퓨터 그래픽의 사용은 현재 특히 엔터테인먼트와 게임 업계에 큰 결과로 적용되어지고 있다. 컴퓨터 그래픽에서 실제 효과를 나타내는데 부족한 영역은 자연 현상의 표현 영역이다. 본 논문에서는 자연 현상중의 하나인 눈이 쌓이는 효과에 대한 기존 연구에 대해 살펴보고 문제점 개선 및 향 후 연구 계획에 대하여 서술하였다.

1. 서론

눈은 자연에서 일반적인 현상이다. 눈의 효과는 장면의 분위기를 완전히 변화시킬 수 있는 힘을 가지고 있다. 그러나 눈은 일반적인 자연현상일지라도 눈의 쌓임과 녹는 현상을 실시간 렌더링을 하기 위한 연구는 많지 않다. 눈의 쌓임과 녹는 현상을 실시간으로 렌더링 하기 위해 몇몇 연구가 있었지만 아직도 컴퓨터 그래픽에서 실제 효과를 나타내는데 부족한 영역이라고 생각한다.

본 논문에서는 기존 연구에 대해 살펴보고 문제점 개선 및 향 후 연구계획에 대하여 서술하였고 구성은 다음과 같다. 2절에서는 관련연구에 대해 서술한다. 3절에서는 기존 방식의 설명과 추가된 부분에 대하여 서술하고 4절에서 결과 및 향후 연구를 서술한다.

2. 관련 연구

눈의 렌더링에 관한 연구는 1997년 Nishita[1]의 연구에서 메타볼과 볼륨 렌더링을 사용하여 눈의 쌓임에 관한 렌더링을 연구하였다. 이후 Paul Fearing[2]은 지면에서의 눈이 쌓이는 경로를 추적하여 눈의 아름다운 장면을 연출하였지만 두 논문 모두 매 프레임 연산이 오래 걸려 실시간 렌더링에는 적합하지 않은 단점이 있다.

실시간 렌더링에 적합한 연구로 Haghund[3]의 연구가 주목받게 되었다. Haghund는 각 표면에 눈의 높이 값을 갖는 매트릭스를 생성하고 Particle을 사용하여 눈송이를 표현하였으며 Particle이 지면에 닿게 되면 지면에 닿은 곳에 해당하는 매트릭스안의 높이 값을 사용하여 지면을 올리는 방법을 사용하였다. 하지만 이 방법은 장면을 렌더링 하기 전에 매트릭스를 일일이 만들어 줘야하는 단점이 있다. 이후 Ohlsson[4]은 Orthogonal Depth Map을 사용하여 눈이 쌓이는 지점을 계산하여 지면을 올리는 방법을 사용하였다. 하지만 눈이 쌓이는 영역과 쌓이지 않는 경계에 대한 샘플링 방법이 단순하여 거친 경계선이 보이고 Vertex Displacement 방법도 단순한 Vertex Normal에 의해 이동시켜 부자연스럽다.

3. 눈 쌓임 기법

본 논문의 알고리즘은 Ohlsson의 연구를 기반으로 이용한다. 여기에 추가적인 통계적 샘플링 (Stochastic Sampling)기법을 사용하여 앨리어싱을 잡음으로 바꿔 눈에 띄게 앨리어싱 효과를 줄일 수 있다. 또한 Vertex Displacement 단계에서 눈이 일정한 양이 표면에 쌓여 부자연스러움을 피하기 위해 Perlin Nose를 사용하였다.

3.1 기본 테크닉

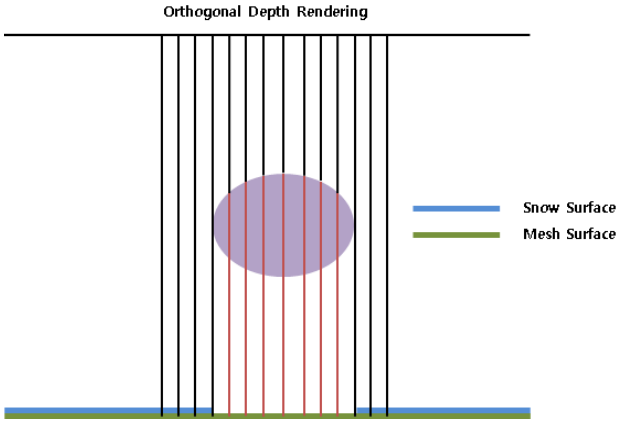
눈의 쌓임을 표현하기 위하여 노출 계수(Exposure Coefficient : Fe)와 기울임 계수(Incline Coefficient : Fi)를 사용한다. 노출 계수는 눈이 쌓이는 지점인지 아닌지에 대하여 계산하기 위한 계수이고 수식은 (식 1)과 같다.

$$F_e = \frac{\left(\sum_{\#Samples} TextureDepth - ActualDepth >= 0 \right)}{\#Samples} \quad \text{(식 1)}$$

장면을 렌더링 하기 전에 Orthogonal Depth Map을 계산하여 저장한다<그림 1>.

<그림 1> Orthogonal Depth Map

이후 장면을 렌더링할 때 렌더 타겟으로부터 Orthogonal Depth Map을 읽어 실제 장면에서의 깊이 값과 비교하게 된다. 만약 깊이 값이 같다면 눈이 쌓이는 지점이고(Fe = 1), 실제 장면에서의 깊이 값이 더 멀리 있다면 물체에 차폐되어 눈이 쌓이지 않는 지점이다(Fe = 0). 단면의 그림으로 본다면 <그림 2>와 같이 나타낼 수 있다.



<그림 2> Orthogonal Depth Map과 실제 장면에서의 깊이 값 비교

기울임 계수는 표면의 기울기에 따라 눈이 쌓이는 정도를 표현하기 위한 계수이고 수식은 (식 2)와 같다.

$$F_i = \begin{cases} \cos\theta + n, & 0 \le \theta \le 90 \\ 0, & \theta \ge 90 \end{cases} \quad \text{(식 2)}$$

θ는 Normal Vector와 Up Vector의 각도이고 n은 Perlin Noise Vector이다. Perlin Noise를 사용하여 Normal 값을 변형시켰는데 이것은 눈 표면의 난반사 효과를 나타내기 위함이다.

최종 눈의 색은 (식 3)을 사용하여 계산한다.

$$F_p = F_e * F_i$$
$$C = F_p * C_s + (1 - F_p) * C_n \quad \text{(식 3)}$$

여기서 C_s는 눈의 기본 색이고 C_n은 눈이 쌓이지 않은 표면의 색이다.

3.2 Perlin Noise

Perlin Noise 함수는 Noise 함수를 다양한 비율의 범위로 더함으로써 프랙탈한 크고 작은 변화의 현상을 재창조하며 수식은 이래와 같다.

$$FRACTALSUM(x) = \sum_i A^i noise(B^i x) \; (0 < A <= 1, 1 < B) \quad \text{(식 4)}$$

A와 B는 조절용 상수로써 A는 진폭에 영향을 주며 B는 주파수에 영향을 주고, n개의 노이즈를 합성했을 경우 n옥타브의 노이즈를 합성했다고 정의할 수 있으며 의사코드는<그림 3>과 같다.

```
int freq = 1;
int amplitude = 1;
int noise = 0;
for o in octaves
      noise += amplitude*texture(freq*uv);
      amplitude /= 2;
      freq *= 2;
endfor
```

<그림 3> Perlin Noise 의사 코드

3.3 Vertex Displacement

Vertex Displacement는 노출계수를 구하는 것과 마찬가지로 Orthogonal Depth Map을 사용하여 실제 장면에서의 Vertex의 깊이 값과 비교한다. 이를 위해 Vertex Shader에 Orthogonal Depth Map을 넘겨주기 위하여 VTF(Vertex Texture Fetch)를 사용한다. 노출계수가 구해지고 눈이 쌓이는 지점의 Vertex 위치에서 Vertex Normal의 방향으로 Vertex의 높이 값을 변경시켜 준다. 이때의 높이 값은 기본적으로 눈의 양에 의해 결정되며 Perlin Noise 값을 곱하여 Vertex의 높이 값을 랜덤하게 변경시키게 되는데 이것은 눈이 일정한 양이 쌓여 부자연스러워 보이는 것을 피하기 위함이며 수식은 다음과 같다.

$$V_n = V + A_s * F_e * N * Z_n \quad \text{(식 5)}$$

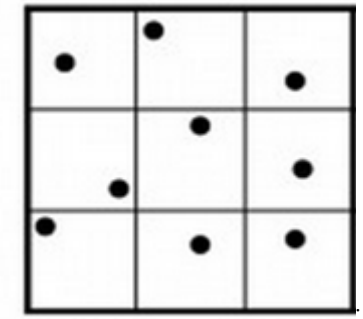
V_n은 새로운 Vertex의 위치이고 V는 현재 Vertex의 위치, A_s는 눈의 양, N은 Vertex Normal, Z_n은 Perlin Noise의 값이다.

3.4 Anti-Aliasing

Orthogonal Depth Map을 사용하여 눈이 쌓이는 지점과 쌓이지 않는 지점에 대하여 Shading을 할 때 경계에 대한 Aliasing 문제가 있다. 이것의 개선방법은 성능과 퀄리티중에 선택을 해야 한다. 퀄리티를 올리는 방법은

Depth Map의 해상도를 올리는 것이다. 또 다른 방법은 주변의 텍셀을 샘플링 하는 것인데, 샘플링 개수를 늘릴수록 퀄리티는 좋아진다. 하지만 둘 다 성능상의 문제가 있으므로 적당한 조절이 필요하다.

본 논문에서는 Orthogonal Depth Map 텍스쳐 해상도를 1024 x 1024를 사용하였으며 퀄리티를 높이는데 중점을 두었고 샘플링 방법은 Stochastic Sampling 방법을 사용하였다. 가장 공통적인Stochastic Sampling은 Jittering을 사용한다. 픽셀 당 n개의 샘플이 사용된다고 가정 하에 영역을 동일한 면적을 가지는 n영역으로 나누고 각 샘플은 이 중 한 영역의 임의의 위치에 놓인다. 최종 픽셀 색상은 샘플의 평균값을 이용한다<그림 4>. 본 논문에서는 18개의 텍셀 샘플을 사용하였다.

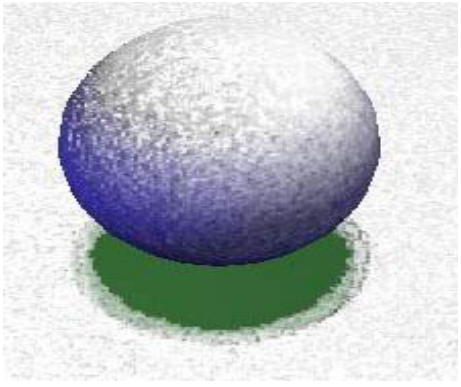


<그림 4> Stochastic Sampling

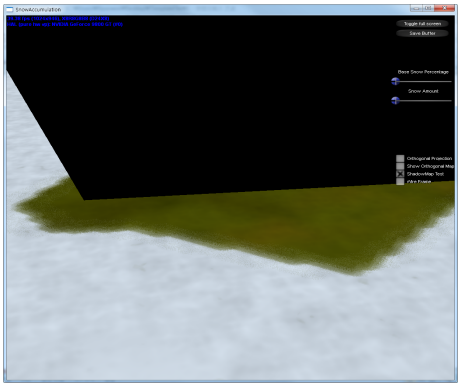
4. 결론 및 향후 연구

구현 결과의 테스트는 AMD Athlon 3000+, 4GB RAM, Geforce 9800 GT에서 실행하였으며 5000개의 삼각형 메쉬 모델이 사용되었다.

<그림 5>은 Ohlsson이 제안한 방법의 결과이고 <그림 6>는 본 논문에서 제안한 Stochastic Sampling을 사용한 방법의 결과이다.



<그림 5> Ohlsson이 제안한 방법



<그림 6> 본 논문에서 제안한 방법

Orthogonal Depth Map 텍스쳐 해상도를 512 x 512를 사용 할 때와 1024 x 1024를 사용 할 때에 FPS는 각각 160, 48 로 약 3배가량 저하되는 것을 볼 수 있었다. 이것은 컴퓨터 시스템 사양에 상당히 의존적이며 3절에서 설명하였듯이 퀄리티와 성능 중에 선택을 하여야 하는 부분이다.

향후 연구로는 바람의 방향에 의해 눈이 쌓이는 것을 위해 바람에 대한 맵의 사용과 경계선의 Vertex Displacement 시 자연스러운 곡선의 표현을 위해 Tessellation의 사용과 경계선에서의 Gaussian 함수를 사용하여 표현하는 것을 연구할 계획이다.

참고문헌

[1] Nishita T, Iwasaki H, Dobashi Y, Nakamae F. "A Modeling and Rendering Method for Snow by Using Metaballs" Computer Graphics Forum. Vol 16, No.3, C357.

[2] Fearing Paul. "Computer Modeling of Fallen Snow" Proceedings of the 27^{th} annual conference on Computer graphics and interactive techniques.

[3] Haglund H, Anderson M, Hast A "Snow Accumulation in Real-Time" Proceeding of SIGRAD 2002.

[4] Ohlsson P. "Real-time Rendering of Accumulated Snow" Uppsala Master's Theses in Computer Science 267 Examensarbete MN3.