한국컴퓨터게임학회논문지
June. 2004.

게임 환경에서 동작하는 실시간 불 애니메이션을 위한 잡음 기반 텍스쳐 생성 기법

황원택* 강영민**

Noise-based Texture Generation Technique for Real-time Fire Animation in Game Environments

Won-Taek Hwang and Young-Min Kang

요 약

게임과 같은 실시간 동작 환경에서 불이나 물 등을 표현하기 위해서는 일반적으로 여러 장의 텍스쳐를 지속적으로 변경하거나 텍스쳐의 좌표값을 특정한 규칙에 따라 바꿔주는 텍스쳐 좌표 애니메이션으로 구현한다. 하지만 위와 같은 방법을 사용하였을 경우 게임 플레이어는 그 반복적인 패턴을 쉽게 알아챌 수 있기 때문에 최근의 게임에서는 노이즈 텍스쳐를 활용하여 이런 비정형적 객체들의 사실적인 텍스쳐 렌더링 기법들이 사용되고 있다. 본 논문에서는 게임과 같은 환경에서 적용 가능한 효율적인 불 애니메이션 기법을 제안한다. 이 기법은 애니메이션 생성의 효율성을 높이고 생성된 불 모양의 자연스러움을 보장하기 위해 잡음 함수와 가중치 함수를 이용한다. 본 논문에서 제안한 불 애니메이션 기법은 자연스러운 불 애니메이션 효과를 게임 환경에서 보일 수 있으며 미리 정해진 텍스쳐를 이용하는 방법에 비해 더욱 사실적인 결과를 보여주며 동일한 패턴을 반복하지 않기 때문에 가상 환경의 사실성과 몰임감을 해치지 않는다.

Key word : Noise, Texture, Fire Animation

Ⅰ. 서 론

일반적으로 불이나 물 시뮬레이션은 가상 환경의 사실성과 몰입감을 증대시키는 데 매우 중요한 역할을 한다. 하드웨어 장치의 성능이 급격히 향상됨에 따라 게임에 요구되는 사실성의 수준도 이전보다 매우 높아졌다. 게임은 사용자의 상호작용에 실시간에 반응해야 하는 동작 특성을 가지기 때문에, 실시간 동작을 보장하기 힘든 비정형 물체의 애니메이션은 지금까지 잘 다루어지지 않았다. 비정형 물체 애니메이션의 예는 옷감과 같이 외형이 바뀌는 객체의 변형을 표현하는 변형체 애니메이션 (deformable object animation)과 불과 물처럼 특정한 형태를 가지지 않고 끊어지고 다시 연결될 수 있는 객체의 애

* 정 회 원 : 한국전자통신연구원 디지털콘텐츠연구단 hwt63355@etri.re.kr(제1저자)
** 정 회 원 : 한국전자통신연구원 디지털콘텐츠연구단 ymkang@etri.re.kr(공동저자)

니메이션 등으로 구분할 수 있다. 변형체 애니메이션은 일반적으로 원래의 상태, 즉 내부 에너지가 발생하지 않는 상태로 돌아가는 움직임을 모델링하고 그 과정을 보여주는 것으로 모델 자체는 일반적으로 단순하게 정의될 수 있지만, 그 움직임을 계산하는 것이 매우 고비용의 연산을 요구하기 때문에 일반적으로 상호작용적 환경에서는 사용되지 않았다. 이와 달리 물과 불의 움직임은 그 수학적 모델을 구하는 것 자체가 옷감 애니메이션에 비해 더욱 복잡하고, 사용된 모델에 따라 사실성이 크게 좌우되기 때문에 어려운 문제로 남아 있다.

불이나 물의 애니메이션과 관련하여 사실적인 동작을 얻는 방법은 물리적 정확성을 최대한으로 고려하여 시뮬레이션을 수행하는 방법으로 포스터(Foster)와 메탁사스(Metaxas)의 유체역학 기반 기법과 존 스탐(John Stam)이나 페드위크(R. Fedwik)이 사용한 오일러 방법을 사용할 수 있다. 그러나 이러한 방법이 가상현실 환경에 적용되었을 경우은 계산 효율이 급격히 떨어져 게임과 같은 환경에 적용될 수 없다.

실시간 동작이 요구되는 시스템에서 불이나 물의 애니메이션은 일반적으로 파티클 시스템을 이용하거나 그리드의 제어점(control points)을 조절하는 방법을 사용한다. 그러나 이러한 방법은 사실적인 동작을 얻기 힘들어 미리 정의된 텍스쳐 파일을 연속적으로 보여주는 경우가 대부분이다. 미리 정의된 텍스쳐 파일을 연속적으로 보여주는 경우에 단조로운 패턴이 반복될 수밖에 없고 이러한 단조로운 패턴은 사용자의 시각에 매우 잘 포착되는 특성을 가지고 있어 게임의 몰입감을 떨어뜨린다.

사실적인 시뮬레이션과 일반적인 실시간 기법은 각각의 불 애니메이션이 어떠한 동작 특성을 갖는 환경에서 사용되는지에 따라 적절한 방법으로 선택되어야 한다. 영화의 특수효과와 같이 시간적 제약보다는 사실성이 더욱 중요한 응용분야에서는 실제 화염과 유사한 결과를 만들기 위해 가능한 정확한 물리적 모델을 이용하여 시뮬레이션을 수행하는 방법을 사용하는 것이 적합하고, 게임과 같이 사용자의 입력에 즉각적으로 반응하면서 실시간으로 장면의 변화를 생성해야만 하는 응용에서는 사용자가 부자연스러움을 느끼지 못하는 정도의 사실성을 유지하면서 가능한 빠른 속도로 애니메이션을 생성할 수 있는 기법을 사용하여야 한다.

게임과 같은 실시간 동작 환경에서 불이나 물 등을 표현하기 위해서는 일반적으로 여러 장의 텍스쳐를 지속적으로 변경하거나 텍스쳐의 좌표값을 특정한 규칙에 따라 바꿔주는 텍스쳐 좌표 애니메이션으로 구현한다. 하지만 위와 같은 방법을 사용하였을 경우 게임 플레이어는 그 반복적인 패턴을 쉽게 알아챌 수 있기 때문에 최근의 게임에서는 노이즈 텍스쳐를 활용하여 이런 비정형적 객체들의 사실적인 텍스쳐 렌더링 기법들이 사용[6]되고 있으며 쉐이더에서도 이를 가능하게 하고 있다.[7][8] 본 논문에서는 게임과 같은 환경에서 적용 가능한 효율적인 불 애니메이션 기법을 제안한다. 이 기법은 애니메이션 생성의 효율성을 높이고 생성된 불 모양의 자연스러움을 보장하기 위해 잡음 함수와 가중치 함수를 이용한다. 생성된 잡음은 가중치 함수에 의해 그 크기가 변경되는데, 잡음과 가중치를 제어함으로써 불의 모습을 제어할 수 있는 기법이다.

II. 잡음 기반 텍스쳐 생성

구름과 물의 움직임, 그리고 불이 타오르는 모습과 같은 자연 현상의 대부분은 그 움직임이나 외형은 명확하고 단순한 모델로 표현하기가 쉽지 않다. 이러한 자연현상의 대부분은 불규칙한 움직임 속에서 전체적으로 일정한 제한을 따르는 모습을 보인다. 이때 일정한 제한을 따르는 부분과 이 제한 속에서 불규칙한 변동을 보이는 특성을 각각 다른 모델로 표현하는 것이 효율적이다. 잡음을 바탕으로 자연 현상을 묘사하는 방법은 이 불규칙한 변동을 잡음을 통해 생성하고 이 잡음이 전체적인 제한 조건을 해치지 않도록 강제하는 방법을 사용하게 된다. 잡음 모델을 이용해 생성된 결과는 자연의 구름과 불처럼 지속적으로 그리고 예측하기 힘든 형태로 변형하는 객체의 애니메이션 생성에 매우 유용하다.

Ken Perlin의 이름을 딴 펄린노이즈[3]는 노이즈 생성기법의 대표적이 예로 격자(lattice)를 인덱싱 테이블 형태로 가지고 있어 이에 쉽게 접근할 수 특징을 가지고 있다. 따라서 펄린 노이즈를 이용한 텍스쳐링 기법은 규칙적인 텍스쳐 생성에 다양하게 응용될 수 있으며 역시 게임내에서도 다양하게 활용될 수 있다.

대표적인 예로 구름의 생성이나 물, 불, 나무 텍스쳐 생성에 활용될 수 있으나, 본 논문에서는 불 텍스쳐의 생성에 응용하기로 한다. 또한 좀더 사실적인 텍스쳐 생성을 위해서 옥타브를 사용한 프랙탈 노이즈[9]를 이용할 수도 있다.

1. 잡음 텍스쳐 생성을 통한 불 외형 애니메이션

본 논문의 기법은 불규칙적인 자연현상을 묘사하는데 효과적인 잡음을 이용하여 시간에 따라 변화하는 불의 외형과 색상을 생성하여 게임 환경에 적용하는 것이다. 이를 위해 잡음 가중치와 잡음을 결합하는 모델을 사용하였다.

1) 잡음 가중치 - 잡음의 전체적인 특성 제어

불모양의 텍스쳐 생성을 위해 고려해야 할 것은 전체적인 특정한 모양을 만들기 위한 잡음 가중치 함수(f_w)를 결정해 주는 것이다. 여기서 가중치 함수란 잡음의 분포밀도를 나타내는 함수로 정의한다. 예를 들어 x축에 대한 잡음 밀도를 정의하기 하기 위하여 사용한 가중치 함수가 $f_w(x) = x\,,\ where\ \ x = [0,1]$의 일차함수로 주어졌다면 이것은 x의 값이 1에 가까울수록 많은 잡음값을 가지고 있다는 것을 의미한다. 마찬가지로 y축이나 z축의 경우에도 적용시킬 수 있다. 이렇게 각 축에 대한 가중치 함수를 결정함으로써 텍스쳐의 모양을 결정해 줄 수 있다. 식 2.1은 바닥이 넓고 위가 좁은 일반적인 불의 모양을 생성하기 위해 사용한 잡음 가중치 함수이다. 식 2.1의 (a)는 x 가중치이며 (b)는 y축에 대한 잡음 가중치 함수를 나타내고 있다.

$$\text{(a)}\ f_w(x) = 1 - \sqrt{\frac{|\frac{1}{2} - x|}{\frac{1}{2}}} \qquad \text{(2.1)}$$

$$\text{(b)}\ f_w(y) = 1 - y$$

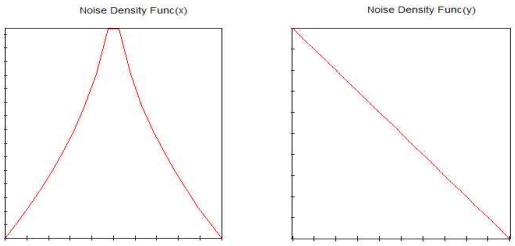
$$(\ 0<x<1,\ 0<y<1\ )$$

식 2.2는 가운데 부분이 밝고 사방으로 별처럼 빛나는 불의 모양을 생성하기 위해 사용할 수 있는 잡음 가중치 함수의 예이다. 식 2.1과 마찬가지로 식 2.2의 (a)는 x 축, (b)는 y축에 대한 가중치 함수이다.

$$\text{(a)}\ f_w(x) = 1 - \sqrt{\frac{|\frac{1}{2} - x|}{\frac{1}{2}}}$$

$$\text{(b)}\ f_w(y) = 1 - \sqrt{\frac{|\frac{1}{2} - y|}{\frac{1}{2}}} \qquad \text{(2.2)}$$

식 2.1의 (a)와 (b)에 나타난 잡음 가중치 함수를 가시화하면 그림 1과 그림 2와 같은 모양으로 나타나며 이 두 함수를 두 축에 적용하여 텍스쳐 평면 전체에 적용된 가중치 함수는 그림 3과 같은 모습을 보이게 된다. 그림 4는 식 2.2에 나타난 잡음 가중치 함수를 텍스쳐 평면에 적용한 결과를 가시화하고 있다. 그림에서 볼 수 있는 바와 같이 식 2.2를 적용할 경우에는 텍스쳐의 중심부에 가중치가 높아져 이 부분을 중심으로 불이 생성된다.



(a) 식 2.1의 x축 가중치 (b) 식 2.1의 y축 가중치

그림 1 잡음 가중치 함수의 가시화 결과 (식 2.1)

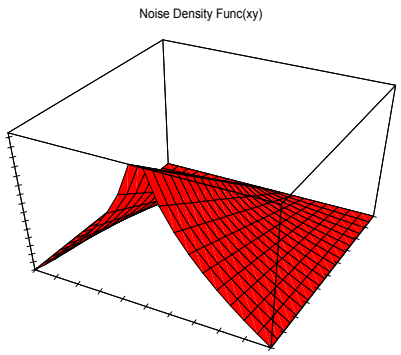


그림 2 텍스쳐 평면에 적용된 잡음 가중치(식 2.1)

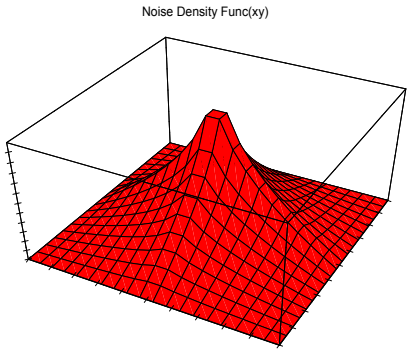


그림 3 텍스쳐 평면에 적용된 잡음 가중치(식 2.2)

2) 잡음 생성

TV 화면과 같은 불규칙적인 잡음이 섞인 화면을 렌더링 하고자 한다면 백색잡음(White noise)를 이용해도 상관없겠지만 구름이나 불 혹은 물과 같은 일정 패턴을 가지는 비정형 물체 표현을 위한 잡음 생성을 위해서는 첫째, 2D 혹은 3D 좌표 상의 값이 잡음과 1:1 매핑될 수 있어야 하며, 둘째, 잡음과 잡음 사이의 부드러운 연결을 위한 보간(interpolation)이 필요하다. 이것은 끊임없는 불규칙한 이동(turbulence displacement)의 특징을 가지는 불을 시뮬레이션 하기 위해 필요하다.

따라서, 본 논문에서는 위의 조건을 만족하는 펄린노이즈(Perlin Noise)를 사용하여 잡음을 생성하였으며 또한 좀 더 사실적인 불 텍스쳐의 생성을 위해 원래 옥타브의 1/2의 값을 재귀적으로 호출하여 더하는 방식으로 프랙탈 잡음을 사용하였다. 여기서 옥타브란 주파수를 의미한다. 즉, 옥타브가 높다는 것은 높은 주파수 대역의 잡음성분을 포함한다는 의미가 된다. [?]

3) 잡음 가중치와 잡음의 결합을 통한 불 외형 생성

다음으로 필요한 과정은 텍스쳐의 알파값에 일정한 시간간격마다 잡음 가중치 함수와 생성된 잡음을 곱하는 것이다. 진행루프의 매 프레임(frame)에 각 텍셀의 알파값을 갱신하는 방법을 사용하면 실제 게임에서 자연스러운 텍스쳐 모양의 변형이 가능하다. 그리고 바람에 흔들리는 효과를 위해 바람의 세기를 나타내는 파라메터도 노이즈를 사용하였다. 표 2.1은 매 프레임마다 호출되는 시뮬레이션을 위한 의사코드(pseudocode)이다.

표 2.1 잡음 기반 텍스쳐 생성 의사코드

```
Loop, I: Height of Image (h)
        p ← (h× scale, h, 0);
        WindParam ← Noise(p) × h × scale;
  Loop, II: Width of Image(w)
    Loop, III: Number of Octaves
      P← (w×scale+WindParam, h*scale+Velocity, 0);
      Noise(P) ← scale/2 ×Noise(P)
      add Noise(P)
    END Loop, III:
    < Ta : Texture Alpha >
    Ta ← Noise(P) × fw(x) × fw(y)
  END Loop, II:
END Loop, I:
```

표 2.1의 의사코드를 살펴보면 노이즈 값을 두 군데에서 참조하였음을 알 수 있다. 그 중 하나는 y축에 따라 변하는 바람의 세기 파라메터 값($WndParam$)이며 다른 하나는 텍셀의 투명도를 결정하는 텍스쳐 알파 값(Texture Alpha)이다. 최종적으로 가중치 함수($f_w(x)$, $f_w(y)$)에 텍스쳐 알파 값을 곱하여 불의 모양을 표현하는 텍스쳐를 생성하였다.

그림 4 생성된 노이즈를 가중치 함수와 결합한 결과

그림 4에 나타는 결과는 표 2.1의 의사코드를 통해 얻은 잡음 데이터를 앞서 살펴본 식 2.1과 식 2.2의

잡음 가중치 함수와 결합하여 생성한 결과를 각각 보여주고 있다. 그림 4에서 볼 수 있는 바와 같이 잡음 가중치를 이용하여 전체적인 외형을 제어할 수 있다.

4) 자연스러운 불의 색상 생성

앞서 설명한 잡음 생성과 잡음 가중치의 결합을 통해 실제 불과 같이 불규칙하게 움직이는 텍스쳐를 생성할 수 있었다. 그러나 이렇게 불의 외형을 생성하는 것만으로는 그럴듯한 불의 모습을 생성할 수가 없다. 자연속의 불은 타오르는 외형뿐 아니라 그 색상 역시 불규칙으로 변화하는 모습을 모이게 된다. 따라서 사실적인 불 애니메이션을 생성하기 위해서는 잡을 통해 외형을 생성한 것처럼, 자연스럽게 변화하는 불의 색상도 생성해야 한다.

이 논문에서 사용하는 불 색상 생성 방법은 앞서 사용한 불의 외형 생성과 동일한 방법을 사용한다. 불의 색상은 크게 두 부분으로 나누어 표현할 수 있는데 그 하나는 불의 전체적인 색이고 다른 한 가지 색상은 불의 중심부에서 밝게 타오르는 색이다. 사용자는 이 색상을 제어할 수 있어야 하는데, 이 두 가지 색상을 각각 C_{fire}와 C_{core}라고 정의하자. 색상 각각이 적색과 녹색, 청색 성분을 가지는 벡터라고 하면, 백색광을 표현하는 벡터 C_{white}에서 C_{fire}를 뺀 값을 C_r이라고 정의하자. 이때 백색광은 (1,1,1)로 표현되고, 다른 모든 색은 그 각각의 성분이 0에서 1사이의 값을 갖는다. 불은 전체적으로 C_{fire}의 색상을 띠지만 중심부에서는 C_{core}의 값을 가지게 된다. 이때 불의 중심부를 어떻게 표현하는가를 정하기만 하면 불의 색상을 생성하는 작업은 간단한 작업이 된다.

자연현상에서 관찰할 수 있는 바와 같이 밝은 빛을 띤 중심부의 모습 역시 고정되어 있지 않고 지속적으로 변한다. 이러한 자연스러운 움직임을 표현하기 위해 다시 한 번 잡음 함수를 이용한다. 불의 중심부, 즉 C_{core}의 값을 가지는 부분의 광강도(intensity)를 1로 가정한다. 그리고 C_{fire}의 색상을 가지는 부분은 광강도가 0이라고 가정하자. 이제 불의 색상을 결정하는 방법은 광강도 I를 결정하는 작업이다. 즉, 어떤 부분의 광강도가 I일 때, 이 부분이 가지는 색상 C의 i 번째 성분은 다음과 같이 표현된다.

$$C_{fire}[i] + IC_r[i]\,C_{core}[i]$$

불의 색상을 구하는 작업은 결국 텍스쳐 각 부분에서 I의 값을 얻는 것이다. 이 I 값을 얻는 방법은 앞서 불의 외형을 생성할 때 사용했던 잡음 가중치와 잡음의 결합 방법을 그대로 적용하면 된다. 일반적인 불의 모습을 생성하기 위해서는 불의 외형을 생성할 때 사용했던 잡음 가중치를 색상 생성에도 그대로 적용하면 된다.

2. 실시간 동작 보장을 위한 최적화

잡음 생성 기법을 이용하여 자연스러운 불의 모양과 색상을 효율적으로 생성할 수 있다. 이 기법은 불의 외형을 생성하기 위해 시뮬레이션과 같이 고비용의 계산을 필요로 하지 않기 때문에 매우 효율적으로 불 애니메이션을 생성할 수 있다.

잡음을 이용한 불 텍스쳐 생성 작업에 특별한 고비용 계산이 필요한 것은 아니지만, 게임과 같이 가능한 최소의 비용으로 작업을 처리해야 하는 환경에서는 몇 가지 성능 개선을 위한 작업을 수행할 필요가 있다.

1) 텍스쳐의 크기

사용하는 텍스쳐의 크기에 따라 본 논문의 기법은 성능 차이가 매우 크다. 예를 들어 32×32의 텍스쳐를 사용하는 경우와 64×64 텍스쳐를 사용하는 경우에 텍스쳐 갱신에 필요한 계산 비용은 텍셀의 개수에 정비례해서 네 배의 차이를 보이게 된다. 따라서 128×128 크기의 텍스쳐와 32×32 크기의 텍스쳐는 16배의 계산비용 차이를 보인다. 생성된 텍스쳐의 사실성이 게임의 몰임감을 해치지 않는 정도를 유지하면서 계산 비용을 최소로 낮출 수 있는 텍스쳐의 크기를 결정할 필요가 있다. 이 값은 실제

적용된 게임 콘텐츠가 게임 진행의 핵심적 요소를 처리한 뒤에 어느 정도의 계산 여력을 남아 있는지를 분석해야만 하며, 이와 함께 적용되는 하드웨어 플랫폼의 계산 성능에도 크게 좌우된다. 따라서 적절한 텍스쳐의 크기는 이 기법이 적용되는 시스템의 하드웨어 및 소프트웨어 성능을 분석한 뒤 최적의 값을 갖도록 한다. 일반적인 PC 환경에서 동작하는 대전 격투 게임에 적용할 경우 64×64 정도의 텍스쳐를 사용할 경우 계산 성능이 크게 저하되지 않은 상태에서 불 애니메이션을 게임 콘텐츠에 포함할 수 있었다.

2) 텍스쳐의 비월갱신(飛越更新 - interlaced texture update)

이 논문에서 제안하는 불 애니메이션 기법은 한 장의 텍스쳐를 매 프레임마다 갱신하는 방법을 사용한다. 텍스쳐의 크기가 커지면 매 프레임의 계산 비용도 같이 커지게 된다. 실제로 64×64 텍스쳐를 사용하는 경우 총 4096 개의 텍셀(texel)을 매 프레임 갱신해야 하는 부담을 가지게 된다. 컴퓨터 게임은 사용자의 입력에 따라 즉각적인 반응을 보여야 하기 때문에 처리해야할 일들의 계산 부담을 가능한 줄이는 것이 좋다. 현재의 하드웨어 성능이 앞서 예로 든 정도의 텍스쳐를 실시간에 변경하는 작업을 쉽게 해낼 수 있지만, 이 문제가 게임에 통합된 경우라면 불필요한 텍스쳐 갱신을 최대한 줄일 필요가 있다.

텍스쳐 갱신의 횟수를 최대로 줄이는 방법으로 가장 간단한 방법은 매 프레임 텍스쳐를 갱신하지 않고 몇 개의 프레임이 지나간 뒤에 한 번의 텍스쳐 갱신이 이루어지도록 하는 것이다. 이 방법은 간단하기는 하지만 매우 큰 단점을 가지고 있다. 텍스쳐의 갱신이 초당 30번 이하로 떨어지는 경우 텍스쳐 갱신에 의해 생성되는 애니메이션이 자연스럽지 않은 모습을 보인다는 것이다. 또한, 매 프레임 마다 앞서 텍스쳐를 갱신할 때가 되었는지 그렇지 않은지를 결정해야 한다. 본 논문에서는 이와 같이 텍스쳐 갱신을 매 프레임마다 하지 않고 일정 시간 지연시키는 방법을 사용하지 않았다. 대신 텍스쳐 전체를 갱신하지 않고 일부만을 갱신하여 갱신의 속도를 높이는 방법을 사용하였다.

텔레비전 신호는 비월주사(飛越走査 - interlaced scanning) 방식을 사용한다. 이 방법은 한 화면의 가로 주사선을 모두 전송하는 것이 아니라 한 번은 홀수 번째 주사선의 신호를 보내고, 다음번엔 짝수 번째 주사선의 신호만을 보내 매 화면 갱신에 필요한 전송 신호의 크기를 크게 줄일 수 있는 방법이다. 재생된 텔레비전 신호를 통해 확인할 수 있는 바와 같이 이렇게 갱신된 텔레비전 화면은 인간의 눈이 전혀 부자연스러움을 느끼지 못한다. 본 논문의 기법은 이러한 텔레비전의 비월주사 방식과 유사한 텍스쳐 비월갱신 기법을 사용한다. 텍스쳐의 비월갱신 기법은 그림 5와 같이 텍스쳐의 일부분을 묶어서 갱신이 이루어질 때에는 해당 영역의 한 텍셀만이 실제로 갱신되도록 하는 것이다. 예를 들어 가로 세로 두 개의 텍셀씩 모두 네 개의 텍셀을 하나의 그룹으로 묶었다면 그룹 전체가 갱신되는 데에는 총 네 번의 갱신이 이루어져야만 하는 것이다.

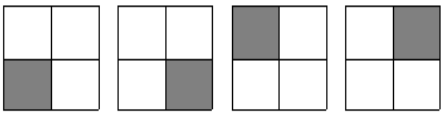


그림 5 텍스쳐 비월 갱신의 텍셀 그룹

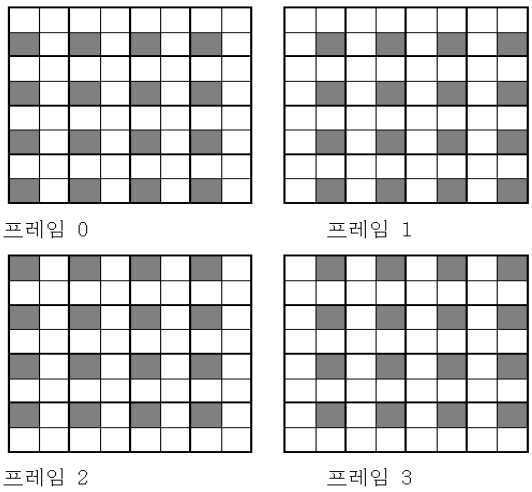


그림 6 비월갱신 사용시에 프레임별로 갱신되는 텍셀

그룹을 묶는 방법은 여러 가지 방법이 있을 수 있다. 예를 들어 3×3 텍셀을 하나의 그룹으로 묶는다면 아홉 번의 갱신이 이루어져야 한 그룹 전체가 갱신될 것이다. 이러한 비월갱신은 전체 화면이 변경되지 몇 개의 프레임에 걸쳐 변경되지 않는 상태가 발생

하지 않고 매 프레임 일부분이라도 변경이 이루어지기 때문에 사람의 눈으로 관찰했을 때 텍스쳐가 지속적으로 변경되고 있다는 느낌을 주게 되며, 그룹의 크기에 따라 계산 비용도 대폭 줄일 수가 있다. 앞에서 예를 든 2×2 텍셀 그룹을 사용할 경우에 텍스쳐 갱신에 드는 계산 비용은 $1/4$로 크게 낮아져 75%의 계산비용 절감을 이룰 수 있다.

그림 6은 이러한 비월갱신 기법을 사용했을 때 한 장의 텍스쳐가 갱신되는 모습을 보이고 있다. 그림 6의 각각의 특정한 프레임에서 갱신되는 텍셀을 회색 셀로 표현한 그림이다. 그림에서 볼 수 있는 바와 같이 특정 프레임에서 갱신되는 텍셀은 전체 텍셀의 $1/4$이며, 불과 같이 불규칙한 외형과 색상 변화를 보이는 객체에 대해 이 정도의 비월갱신을 수행할 경우 사용자의 눈에는 전혀 어색한 모습이 관찰되지 않는다.

III. 실험결과

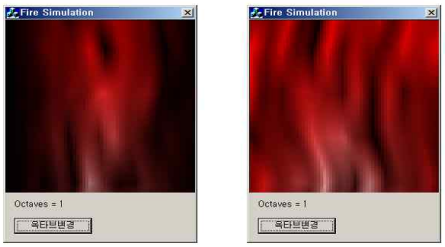
본 논문에서 사용한 잡음 기반 텍스쳐 생성은 결과를 제어할 수 있는 몇 가지 방법을 제공한다. 그 가운데 한 가지는 사용되는 노이즈의 옥타브를 조절하는 것이다. 옥타브 수를 변화시킴에 따라 더 높은 주파수를 가지는 잡음이 섞여 나오게 되므로 좀 더 사실적인 시뮬레이션 결과를 얻을 수 있다. 그림 7은 옥타브를 조절함에 따라 생성된 서로 다른 결과를 보이고 있다. 그림 7의 (a)는 옥타브를 1로 설정한 결과이며 (b)는 옥타브를 3으로 설정한 결과이다.

옥타브와 함께 불의 특성을 제어할 수 있는 파라미터로 바람의 효과를 흉내낼 수 있는 바람 파라미터를 도입하였다. 그림 8에 나타난 결과는 바람 파라미터의 값을 조절함에 불의 흔들리는 정도가 달라짐을 보이고 있다. 바람의 세기 역시 노이즈 값을 가지게 하여 시뮬레이션 하면 큰 파라미터를 가질수록 흔들림이 크게 나타남을 알 수 있다.

옥타브와 바람의 세기와 같은 파라미터와 함께 생성된 불의 모습을 제어할 수 있는 방법으로 잡음 가중치를 조절하는 방법이 있다. 잡음 가중치는 불의 전체적인 외형을 제어할 때 사용하는 것으로 생성된 잡음이 가중치 함수와 결합하여 최종적인 텍스쳐를 생성하기 때문에 불의 모습이 나타나지 않아야할 부분에서 0의 값을 갖는 가중치 함수를 사용함으로써 전체적인 외형을 제어할 수 있다. 그림 9의 (a)는 x축에 대해서는 텍스쳐 중앙을 중심으로 대칭하는 함수를 사용하고 y축에 대해서는 sin곡선을 적용시킨 경우이며 (b)는 x축과 y축 모두 sin곡선을 적용시킨 경우로, 불이 옆으로 번지는 효과를 구현할 수 있다.

(a) 옥타브 = 1 (b) 옥타브 = 3
그림 7 옥타브 변경을 통한 제어

(a) 바람 강도 = 1 (b) 바람 강도 = 2
그림 8 바람 파라미터를 변경한 결과



(a) y축에 sin 적용 (b) x,y축에 sin 적용
그림 9 잡음 가중치 변경을 통한 제어

게임환경에서 테스트 하기 위해 1:1대전게임의 배경으로 빌보드 객체를 사용하여 시뮬레이션 해 보았다. 텍스쳐는 64×64 크기와 32×32 크기를 사용해서 테스트했다. 64×64의 텍스쳐를 사용하여도 실시간 환경에서 동작하는 불을 생성하는 것이 가능하였다.

그림 10 . 객채수 5개 (74 fps)

그림 11 . 객채수 50개 (72 fps)

게임에 적용시킨 결과 빌보드 객체의 수보다는 역시 텍스쳐의 크기에 큰 프레임률의 변화를 보였다. 하지만 테스트 결과 32×32 크기의 텍스쳐를 사용했을 경우, 빌보드 객체가 5 개에서 50 개로 늘어나도 프레임률의 변화는 거의 없었다. 즉, 32×32 정도의 텍스쳐 크기를 사용하여 불 애니메이션을 생성할 경우 게임 진행에 거의 아무런 계산 부담도 주지 않는다는 것을 확인할 수 있었다. 물론 텍스쳐를 갱신하는 방법은 계산 비용을 최소로 줄이기 위해 비월 갱신 기법을 사용하였다.

IV. 결론 및 개선사항

본 논문에서는 자연스러운 불 애니메이션을 게임 환경에서 표현하기 위해 잡음 값을 이용하여 불의 텍스쳐를 생성해내는 기법을 제안하였다. 잡음은 단순한 수학 모델로 표현이 불가능한 특성을 가진 자연현상을 묘사하기에 매우 적합한 도구를 제공한다. 잡음 그 자체는 특별한 의미를 가지지 않는 값들이지만 이를 적절한 가중치나 규칙에 따라 움직이게 함으로써 구름, 물, 불과 같이 복잡하고 예측하기 어려운 동작을 보이는 자연 현상을 흉내낼 수 있다.

본 논문의 기법은 펄린 잡음을 이용하여 텍셀값을 업데이트하는 방법을 사용하였다. 불 애니메이션에서 생성되어야할 값은 불의 외형을 결정하기 위하여 불이 차지하는 영역을 결정하는 것이며, 이와 함께 자연스럽게 이글거리는 모습을 표현하기 위해 불의 색상을 결정하는 것이다. 불의 외형과 밝은 중심부의 모습을 제어하기 위해서는 잡음 데이터를 특정한 영역에서만 드러나게 하는 잡음 가중치 함수를 제어하였고, 이 영역 내에서 자연스럽고 규칙적인 모습이 생성되도록 잡음을 발생시켰다. 이러한 잡음 기반 텍스쳐 생성 기법은 매우 자연스러운 불의 모습을 실시간 환경에서 생성할 수 있었다. 따라서 이 기법을 이용하여 게임과 같이 실시간 동작이 요구되는 응용에 사실적이고 자연스러운 불의 애니메이션을 효과적으로 포함할 수 있다.

게임과 같이 실시간 동작이 요구되는 분야에서는 가능한 계산 비용을 줄이는 작업이 요구된다. 본 논문의 잡음 기반 불 텍스쳐 생성 기법은 비월갱신 기법을 통해 텍스쳐 갱신의 효율을 크게 향상시킬 수 있는 기법을 사용한다. 비월갱신 기법은 텔레비전의 비월주사와 같은 방식으로 텍셀을 갱신하는 것으로 텔레비전이 가로 주사선의 한 칸 씩 건너뛰며 갱신하는 것처럼 텍셀을 갱신하는 것이다. 이때 본 논문에서 제시한 기법은 가로선과 세로선 모두를 건너뛰며 갱신하는 기법을 사용하며, 갱신시 하나의 묶음으로 처리되는 영역을 자유롭게 설정할 수 있도록 하여

필요한 경우 계산 효율을 더욱 크게 줄일 수 있도록 하였다.

본 논문에서 제안한 불 애니메이션 기법은 자연스러운 불 애니메이션 효과를 게임 환경에서 보일 수 있는 기법이다. 이 기법은 미리 정해진 텍스쳐를 이용하는 방법에 비해 더욱 사실적인 결과를 보여주며 동일한 패턴을 반복하지 않고 계속해서 새로운 패턴을 생성할 수 있기 때문에 가상 환경의 사실성과 몰입감을 해치지 않는다.

향후 연구 과제로는 본 논문의 잡음 텍스쳐 생성 기법과 물리엔진을 연동하여 바람 이외의 다른 외부 힘에 잡음 텍스쳐가 반응하게 하는 사실적인 시뮬레이션을 효율적으로 결합하는 것이다. 이러한 물리 시뮬레이션이 효과적으로 통합된다면 게임에서의 활용폭이 더욱 넓어질 것이다.

참 고 문 헌

[1] N. Foster and D. Metaxas. Modeling the Motion of a Hot, Turbulent Gas. In *SIGGRAPH 97 Conference Proceedings, Annual Conference Series* , pages 181-188, August 1997

[2] R. Fedkiw, J. Stam, and H. W. Jensen. Visual Simulation of Smoke. In *SIGGRAPH 2001 Conference Proceedings, Annual Conference Series* , pages 15-22, August August 2001

[3] Perlin Noise available at, Http://freespace.virgin.net/hugo.elias/models/m_perlin.htm

[4] NVidia Corp. Noise, Procedural 3D Texturing. http://developer.nvidia.com/object/proctex3d.html

[5] J. Stam. Stable Fluids. In *SIGGRAPH 99 Conference Proceedings, Annual Conference Series* , pages 121-128, August 1999

[6] Unreal Animation Real-Time Textures Manual, available at http://unreal.epicgames.com/Fire/AnimatingTextures.htm

[7] Olano, M. and A. Lastra. A shading language on graphics hardware: The PixelFlow shading system. Proc. SIGGRAPH 98, July 1998,pp. 159-168

[8] Proudfoot, K., W.R. Mark, S. Tzvetkov and P. Hanrahan. A real-time programming shading system for programmable graphics hardware. Proc. SIGGRAPH2001, Aug. 2001

[9] Steven P. Worley. A cellular texturing basis function. In Holly Rushmeier, editor, SIGGRAPH96 Conference Proceedings, Annual Conference Series, p291-294.