※ 본 아티클은 CMP MEDIA LLC와의 라이선스 계약에 의해 국문으로 제공됩니다

Gamasutra.com

서적 발췌

GPU Gems 2:

버텍스 텍스처 변위를 사용한 사실적인 물 렌더링

Yuri Kryachko

2006년 1월 3일

http://www.gamasutra.com/features/20060103/kryachko_01.shtml

다음은 Addison-Wesley Professional에서 출판한 *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation* (ISBN 0321335597)의 18 장에서 발췌했습니다.

물 표면은 컴퓨터 그래픽, 특히 게임에서 일반적으로 등장한다. 이 장면의 사실성을 확실히 높여줄 중요한 요소가 있다. 그러나 물 표면의 움직임을 표현하는 시각적 복잡성과 물에 반사되는 빛 때문에 물을 사실적으로 표현하기가 매우 어렵다. 이 장에서는 퍼시픽 파이터라는 컴퓨터 게임에서 대양을 사실적으로 렌더링하기 위해 개발한 기술을 살펴본다.

현대 그래픽 하드웨어는 물 표면 렌더링에 사용하는 DirectX Shader Model 3.0 에 몇 가지 유용한 기능을 제공한다. 이 장에서는 이들 기능 중 하나인 버텍스 텍스처를 사용해 물 표면의 사실감을 높이는 방법을 살펴본다. 그림 18-1 은 일부 샘플 결과이다. 그리고 분기(branching)를 사용해 버텍스 프로그램의 성능을 높였다.

18.1 물 모델

물 애니메이션과 렌더링 기법은 여러 가지가 있다. 그 중 유체역학 및 Tessendorf 2001 와 같은 고속 퓨리에 변환(FFTs) 기반 기술이 가장 사실적인 결과물을 제공한다. 하지만 이들 기법은 계산량이 많아 대화형 응용프로그램에는 적당하지 못하다는 단점을 가지고 있다.

그림 18-1 변위 매핑의 장점
변위 매핑을 사용한 물 표면 렌더링(왼쪽)과
사용하지 않은 물 표면 렌더링(오른쪽)

다른 한편으로 대부분 게임은 노멀 맵으로 비주얼 디테일을 생성하는 매우 간단한 물 모델을 사용하고 있다. 하지만 이 방법으로 생성한 물 표면은 사실성이 떨어지고 표면의 물결도 충실히 재연하지 못하고 있다.

따라서 간단한 노멀 맵된 물 렌더링 기법의 속도와 FFT 기법의 비주얼을 모두 갖춘 기술이 필요하다.

18.2 구현

노멀 맵을 사용해 빛을 계산하는 렌더링 알고리즘을 따라 구현한다. 노멀 맵은 고주파에서 디테일을 충실하게 재연하기 때문에 우리는 이를 빛 계산에 이용한다. 그러나 동시에 진폭이 큰 저주파를 사용해 기하학적으로 물 메시를 교란했다.

18.2.1 물 표면 모델

물 표면 모델은 시간과 공간에 모두 바둑판 형으로된 높이맵 중첩에 기반한다. 각 텍스처는 스펙트럼의 “조화” 또는 “역타브”를 나타내고 해당 텍스처는 푸리에 합성에서 함께 추가된다. 여기서 각 값은 수평면 상의 상응하는 포인트의 높이를 나타내기 때문에 이들 텍스처를 “높이맵”이라 부른다.

높이맵 생성은 회색조 이미지를 색칠하는 것만큼 간단하다(그림 18-2 참조). 높이맵을 이용하면 단순히 그 모양을 그려 개별 물결에 이르기까지 물 애니메이션의 매개 변수를 쉽게 제어할 수 있다. 높이맵은 또한 버텍스 텍스처와도 잘 작동한다. 높이맵을 이용하면 버텍스 위치를 매우 쉽게 수직으로 바꿀 수 있다.

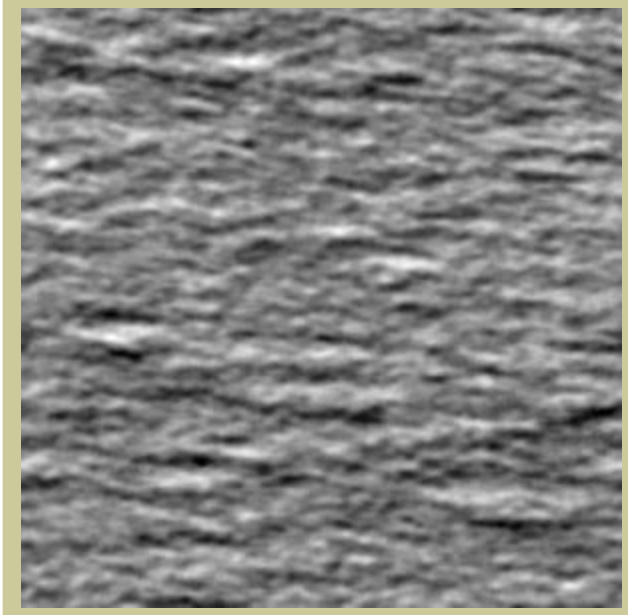

그림 18-2. 높이맵을 사용한 물 변위

몇 개의 높이맵을 다른 공간과 시간 배율과 결합하여 복잡하고 시각적으로 얽혀있는 애니메이션을 얻을 수 있다.

$$H(x, y, t) = \sum_{i=0}^{N} h\left(A_i^x x + B_i^x,\ A_i^y y + B_i^y,\ A_i^t t + B_i^t\right).$$

계수 A 와 B, 합 아래 몇 개의 항은 불필요한 반복 패턴을 최소화하면서 미적으로 가장 뛰어난 결과물를 얻기 위해 발견적으로 선택되었다. 퍼시픽 파이터에서 우리는 빛 계산에 필요한 네 가지 높이맵을 합산했고 이들 중 배율이 큰 두 개가 변위 매핑에 사용되었다. 이는 10cm 부터 40km 범위까지 움직이는 대양 표면을 묘사하기 충분했다.

18.2.2 디테일 구현

우리가 수행해야 할 모든 계산은 기하학적인 변위 계산과 빛 계산 등 두 그룹으로 구분할 수 있다. 물 표면은 섬세한 바둑판 형이기 때문에 변위맵을 버텍스 단계에서 오프로딩하는 프래그먼트 프로그램(fragment program) 수준에서 빛 계산을 수행하는 것이 합리적이다.

또한 버텍스 수준에서의 빛 계산 수행은 특히 원거리에서 시각적으로 불필요한 부분을 생성할 수 있다.

작성 시에는 버텍스 텍스처링을 수행할 수 있는 유일한 하드웨어는 GeForce 6 Series GPUs 및 NVIDIA Quadro FX GPUs 최신버전이었다. 이 하드웨어에서 버텍스 텍스처를 구현하면 버텍스 텍스처는 구성요소 텍스처당 32 비트여야 하고, 부동소수점과 가장 가까운 필터링 외에는 필터링 모드를 사용할 수 없는 등 제한 사항이 존재한다. 그럼에도 불구하고 이들 하드웨어는 이 장에서 설명하는 기술에 매우 유용하다.

18.2.3 높이맵 샘플링

구현에서는 버텍스 당 높이맵을 표본 추출하여 버텍스 프로그램에서 변위 값 결과를 계산한다. 샘플에서는 카메라 위치에 중심이 있는 방사형 그리드를 사용한다. 이 그리드는 그림 18-3 에서 나타나듯 보는 사람에게 더욱 자세한 디테일을 제공하는 바둑판 모양이다.

다음 등식은 방사형 그리드에서 버텍스 위치를 계산하는 방법을 보여준다.

$$
\begin{aligned}
r &= a_0 + a_1 i^4 \\
x_{i,j} &= r\cos\left(2\pi j/M\right) \\
y_{i,j} &= r\sin\left(2\pi j/M\right),
\end{aligned}
$$

where $i = [0..N-1], j = [0..M-1]$. We choose a_0, a_1 so that

$$
\begin{aligned}
r_0 &= a_0 = 10 \text{ cm} \\
r_{N-1} &= a_0 + a_1\left(N-1\right)^4 = 40 \text{ km}.
\end{aligned}
$$

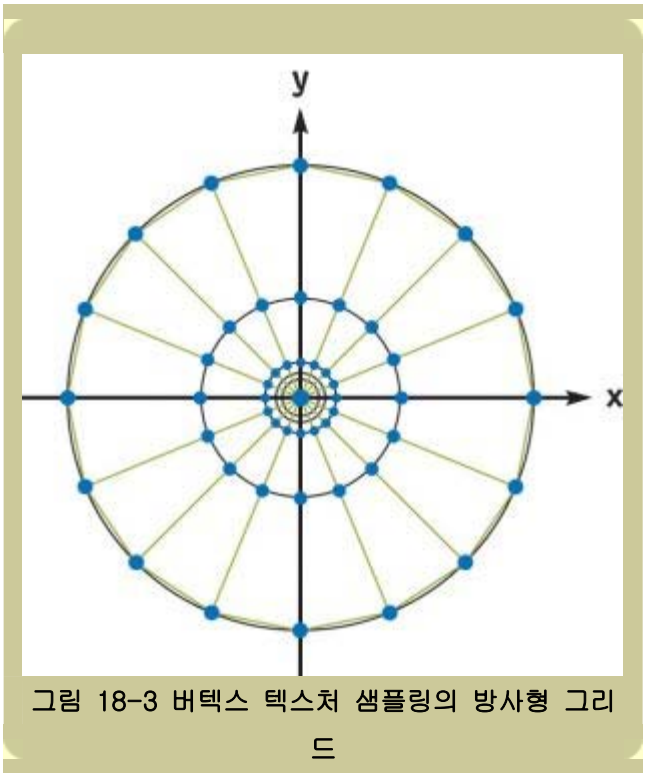

그림 18-3 버텍스 텍스처 샘플링의 방사형 그리드

이 방식을 통해 수월하게 샘플 정밀도 구성표를 제공하는 거리 기반 바둑판형을 얻었다. ROAM 또는 SOAR 지형 렌더링 알고리즘 등의 방식도 사용 가능하나 이들은 버텍스 텍스처 사용으로 얻는 이점을 모두 상쇄할 정도로 CPU 에서 막대한 양의 작업을 소화내야 한다. 적응 바둑판형과 함께 CPU 상에서 높이 필드를 렌더링하는 다른 방법은 이 책의 2 장 “CPU 기반 기하학적 클립맵을 사용한 지형 렌더링”에서 확인할 수 있다.

목록 18-1 은 방사형 그리드로 높이맵에서 샘플링을 구현하는 간단한 버텍스 셰이더이다.

```
float4 main( float4 position : POSITION,
  uniform sampler2D tex0,
  uniform float4x4 ModelViewProj,
  uniform float4 DMParameters, // displacement map parameters
  uniform float4 VOfs) : POSITION
{
  // Read vertex packed as (cos(), sin(), j)
    float4 INP = position;

  // Transform to radial grid vertex
  INP.xy = INP.xy * ( pow (INP.z, 4) * VOfs.z);

  // Find displacement map texture coordinates
  // VOfs.xy, DMParameters.x - Height texture offset and scale
   float2 t = (INP.xy + VOfs.xy) * DMParameters.x;

  // Fetch displacement value from texture (lod 0)
  float vDisp = tex2D (tex0, t).x;

  // Scale fetched value from 0..1:
  // DMParameters.y - water level
  // DMParameters.z - wavy amplitude
    INP.z = DMParameters.y + (vDisp - 0.5) * DMParameters.z;

  // Displace current position with water height
  // and project it
  return mul (ModelViewProj, INP);
}
```

목록 18-1. 방사형 그리드 기하학을 사용하여 높이맵에서 샘플링 버텍스 셰이더

18.2.4 품질 개선 및 최적화

이중선형 필터링의 패킹 높이 버텍스 텍스처 페치는 꽤 고가일 수 있다. GeForce 6 Series 하드웨어에서 단일 버텍스 텍스처 페치는 버텍스 프로그램에서 눈에 띄는 시간 지연을 야기할 수 있다. 그래서 버텍스 프로그램 내에서 텍스처 페치의 숫자를 최소화해야 한다.

또 한편으로 텍스처 값에 대해 일종의 필터링도 수행해야 한다. 그렇지 않으면 시각성이 매우 떨어질 것이기 때문이다. 일반적으로 널리 알려진 필터링 방법은 이중선형과 삼중선형 필터링이다. 이중선형 필터링은 텍스처 페치의 좌표에서 가장 가까운 텍셀 4 개의 가중 평균을 계산한다. 삼중선형 필터링은 해당하는 LOD 소수를 이용해 각각에 가중치를 주어 인접한 밉(mip) 레벨에서 이중선형 조회 결과를 평균한다. 왜냐하면 그래픽 하드웨어의 해류 생성은 어떤 형태의 버텍스 텍스처 값 필터링도 지원하지 않기 때문에 정확한 수학적 지침에 따라 셰이더에서 필터링을 에뮬레이트해야 한다. 가장 간단한 이중선형 필터도 구현했을 때 단일 필터된 값을 계산하려면 4 개의 텍스처 조회가 필요하다. 삼중선형 필터는 이보다 2 배 많은 텍스처 조회가 필요하다. 필터링에 필요한 텍스처 페치의 수를 줄이기 위해 특별한 방법으로 텍스처를 만들었기 때문에 각 텍셀은 단일 이중선형 텍스처 조회에 필요한 모든 데이터를 포함하고 있다. 우리가 사용한 높이맵은 근본적으로 단일 구성요소 텍스처이기 때문에 이것이 가능했으며 구성요소 텍스처 4 개의 단일 텍셀로 4 개의 높이 값을 패킹할 수 있다.

$$
\begin{aligned}
A_x^{i,j} &= H^{i,j}, \\
A_y^{i,j} &= H^{i+1,j}, \\
A_z^{i,j} &= H^{i,j+1}, \\
A_w^{i,j} &= H^{i+1,j+1}, \\
A_{filtered} &= F\left(A_{x,y,z,w}^{i,j}\right),
\end{aligned}
$$

여기서 $i = 0..N - 1$, $j = 0..M - 1$ 이다. H는 높이맵 값이며 F는 필터링 함수, 그리고 A는 패킹된 출력 텍스처이다.

목록 18-2 은 이전에 보여진 대로 패킹된 버텍스 텍스처로 이중선형 텍스처 조회를 구현한다.

```
float tex2D_bilinear4x( uniform sampler2D tex,
  float4 t,
  float2 Scales)

{
  float size = Scales.x;
  float scale = Scales.y;

  float4 tAB0 = tex2Dbias (tex, t);

  float2 f = frac (t.xy * size);
  float2 tAB = lerp (tAB0.xz, tAB0.yw, f.x);
  return lerp (tAB.x, tAB.y, f.y);
}
```

목록 18-2. 버텍스 셰이더에서 효과적인 이중선형 텍스처 보간 페칭을 기반으로 단일 텍스처 페치로 4개의 높이 구성요소 충당

동일한 방법을 삼중선형 필터링에도 사용할 수 있다. 삼중선형 필터링에는 소수의 LOD 값이 필요하기 때문에 카메라로부터의 거리를 LOD 의 근사값으로 사용했다. 목록 18-3 에서 코드는 패킹된 버텍스 텍스처로 삼중선형 텍스처 조회를 구현한다.

```
float tex2D_trilinear( uniform sampler2D tex,
 float4 t,
 float2 Scales)
{
 float fr = frac (t.z);
 t.z -= fr; // floor(t.zw);
 float Res;
 if (fr < 0.30)
  Res = tex2D_bilinear4x(tex, t.xyzz, Scales);
 else if (fr > 0.70)
  Res = tex2D_bilinear4x(tex, t.xyzz + float4 (0, 0, 1, 1),
                Scales * float2 (0.5, 2));
 else {
  Res = tex2D_bilinear4x(tex, t.xyzz, Scales);
  float Res1 = tex2D_bilinear4x(tex, t.xyzz + float4 (0, 0, 1, 1),
                    Scales * float2 (0.5, 2));
  fr = saturate ((fr - 0.30) * (1 / (0.70 - 0.30)));
  Res = Res1 * fr + Res * (1 - fr);
 }
 return Res;
}
```

목록 18-3. 이중선형 필터링 기술을 삼중선형 필터링으로 확장

우리는 두 밉 레벨이 큰 영향을 미치는 구역에서만 두 텍스처 조회를 수행하여 삼중선형 텍스처 페치를 최대화했다. 다른 영역에서는 LOD 값을 가장 가까운 밉 레벨에 "맞춰(snap)" 텍스처 대역폭을 줄인다.

분기로 불필요한 작업 없애기

최적화된 텍스처 필터링으로도 물 렌더링을 수행하기에는 텍스처 패치 수가 여전히 너무 많았다. 렌더된 버텍스의 총수를 줄일 수는 있으나 이로 인해 전반적인 시각적 디테일이 떨어지고 얼라이징은 증가할 수 있다.

우리는 거대한 기하학 묶음을 사용해 물을 렌더링하기 때문에 일부 삼각형은 완전히 화면 밖으로 사라진다. 하지만 이러한 삼각형에 대해서도 버텍스 프로그램은 여전히 실행되기

때문에 귀중한 컴퓨터 리소스가 낭비된다. 따라서 카메라 절두체 밖 삼각형을 계산하지 않는다면 버텍스 당 작업량을 줄일 수 있다.

버텍스 프로그램은 한번에 하나의 버텍스에서 작동하고 위상적인 정보에 접근할 수 없기 때문에 삼각형 레벨이 아닌 버텍스 당 레벨에 대해서만 결정할 수 있다. 삼각형 내의 일부 버텍스가 버텍스 텍스처링을 생략하고 일부는 그러지 않는다면 불필요한 부분을 생성할 수 있다. 우리는 실제 작업에서 이 불필요한 부분은 감지되지 않을 정도로 삼각형과 버텍스 텍스처 변위가 작다는 사실을 발견했다.

다음 의사코드는 이를 보여준다.

```
float4 ClipPos = mul (ModelViewProj, INP);
float3 b0 = abs (ClipPos.xyz) < (ClipPos.www * C0 + C1);

if ( all (b0)) {
 // Vertex belongs to visible triangle,
 // Perform texture sampling and displace vertex accordingly
}
```

위 코드에서는 클립공간 버텍스 위치를 사용해 현재 버텍스가 절두체 내에 존재하는지 판단한 다음 필요한 경우에만 계산을 수행했다.

값 C0 및 C1 은 특별한 "퍼지" 상수로써, 클리핑을 유발하기 위해 얼마나 많은 삼각형이 카메라 절두체 너머로 확장돼야 하는지 제어한다. 이 방식으로 우리는 삼각형이 여전히 보이는 절연체 밖 버텍스의 텍스처링을 생략함으로써 발생할 불필요한 부분을 없앨 수 있었다. "클리핑" 절연체를 약간 넓게 만들어 스크린 가장자리를 따라 일정량의 "보호대역" 공간을 만들었다. 생성한 수선면을 충분히 만족스러운 바둑판형이고 버텍스 텍스처 변위도 합당했기 때문에 이 간단한 방법은 실제 작업에서도 잘 동작한다.

렌더 투 텍스처 기능 사용하기

각 pass 에서 높이맵 텍스처를 단일 부동소수점 텍스처에 우선 결합시켜 속도를 높였다. 그러면 버텍스 셰이더에서 다중 필터링을 수행할 필요가 없어지기 때문이다. 그리고 우리는 현재 더욱 간결한 텍스처 포멧인 16 비트 부동소수점을 사용해 원래 높이맵을 저장한다. 또한 애니메이트된 높이맵의 순서를 3D 텍스처의 슬라이드로 저장하여 더욱 매끄러운 애니메이션을 만들어냈다.

이 최적화를 통해 우리의 렌더링 루프는 다음과 같은 두 가지 패스(pass)가 된다.

1. 단일 사변형을 fp32-텍스처로 렌더링함으로써 특별한 픽셀 셰이더를 사용해 높이맵을 결합시킨다. Texels in this texture map to the vertices of the radial mesh.
2. 위에서 설명한대로 방사형 메시 버텍스를 바꾸기 위해 생성된 높이맵을 버텍스 텍스처로 사용한다.

물결의 뒤쪽

우리는 물표면이 평평하다는 가정하의 픽셀 셰이더에서 빛을 계산했기 때문에 특정 경우 시각적으로 불필요한 부분을 만들어낼 수 있다.

그림 18-4 에서 기하학적 변위 때문에 물결이 보는 사람으로부터 밖을 향하고 있고 실제로는 보이지 않는다고 할지라도 우리는 물결의 뒷면을 볼 수 있다. 이 결과 물결의 꼭대기에서 불안정하게 밝은 영역이 나타난다.

이러한 불필요한 부분을 최소화하기 위해 우리는 보는 사람을 향해 이들을 약간 “바둑판배열”하여 빛 계산에 사용된 노멀 벡터를 조정했다. 그로 인해 물결의 앞면에 좀 더 조화를 이루게 되었다. 함께 제공하는 CD 에서 이 기술에 필요한 소스 코드를 찾을 수 있다. 그림 18-5 는 이 장에서 설명한 방법을 사용해 생성한 장면이다.

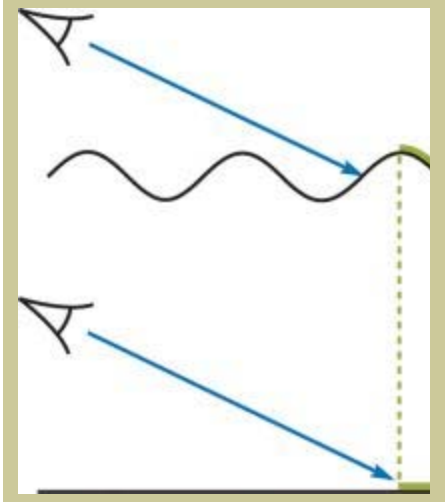

그림 18-4 불필요한 부분 렌더링의 소스
물결의 뒤쪽(녹색)은 실제로는 보이지 않지만 셰이더될 수 있다. *빛 계산에 사용된 노멀 벡터를 조절해 이러한 오류를 상당히 줄일 수 있다.*

그림 18-5. 여기서 설명한 기술로 렌더한 매우 사실적인 물

18.2.5 Rendering Local Perturbations

이제 물에 떨어지는 물체 또는 부유 물체에 의해 발생하는 로컬 짤림 현상(choppiness)을 표현해야 한다. 이는 특히 폭발, 배, 끌림 자국 등을 생성해야 하는 게임에 중요하다. 우리가 사용한 높이맵 기반의 물 표면 모델에서는 물리적으로 올바른 방법을 통합하기가 어렵게 때문에 발견적 방법을 기반의 좀 더 단순한 방법을 사용했다.

분석적인 변형 모델

가장 간단한 로컬 짤림 현상 표현 방법은 버텍스 셰이더에서 계산된 버텍스 위치와 이들을 결합함에 따라 바뀐 버텍스 위치를 분석적으로 교란한다. 폭발 장면에는 다음 수식을 사용할 수 있다.

$$I(r) = \frac{I_0 \sin(kr + \omega t) e^{-bt}}{r^2},$$

여기서 r은 폭발 중심에서 수선면까지의 거리이며 b는 불변 상수이다. I_0, w, 및 k의 값은 주어진 폭발과 그 매개 변수에 따라 선택된다.

일반 물 렌더링에서 사용했던 것과 동일한 방사형 그리드를 사용하여 렌더링 가능하나 폭발 위치의 중심에서는 그럴 수 없다.

동적 변위 매핑

또 다른 방법은 국부적으로 생성된 모든 변위를 GPU(GPGPU) 타입의 방법에 대해 일반 목적 프로그래밍을 구현한 직접 버텍스 텍스처로 렌더링하는 것이다. 이 방법으로 우리는 첫 번째 패스에서 버텍스 텍스처를 생성한 다음 실제 물 렌더링의 다음 패스에서 이용한다. 추가적으로 픽셀 셰이더에서 "옥타브"를 합산하고 기본 높이맵을 필터링함으로써 버텍스 셰이더에서 일부 작업을 오프로드할 수도 있다.

프레임에서 프레임으로 로컬 변위를 발전시킴으로써 위에서 언급한 분석적 모델 또는 셀룰라 오토마타(cellular-automata) 방식을 사용하여 변위를 계산할 수 있다. 또한 적절한 방향을 따라 텍스처를 흐림으로 처리(blurring)함으로써 바람 효과를 만들어 낼 수 있다.

그러나 50cm 해상도로 1km 의 물표면을 덮으려면 약 2048 × 2048 사이즈의 텍스처를 사용해야 한다. 이 사이즈는 텍스처 메모리와 셰이더 실행 속도에 추가적인 부담을 줄 수 있고 뷰포인트의 빠른 전환에도 문제가 될 수 있다.

그럼에도 불구하고 이들 방법을 사용해보라고 권장한다.

거품 생성

짤림 현상이 강하면 사실성을 더욱 높이기 위해 거품을 생성해야 한다. 가장 간단한 방법은 특정 높이 H_0 상의 재배치된 버텍스에서 미리 생성된 거품 텍스처와 혼합하는 것이다. 거품 텍스처의 투명도는 다음 수식을 따라 계산된다.

$$Foam.a = \text{saturate}\left(\frac{H - H_0}{H_{\max} - H_0}\right),$$

H max는 거품이 최대화되는 높이이며 H_0는 기본 높이 H는 현재 높이이다.

거품 텍스처는 거품이 생성되고 사라지는 모든 과정을 보여주도록 애니메이트될 수 있다. 이 애니메이션 장면은 수동으로 또는 프로그램적으로 생성될 수 있다.

18.3 결론

버텍스 텍스처 패치의 유연성과 동적 분기의 간략한 방법을 결합하여 대화형 속도에서 사실적인 물 표면 렌더링의 실행 방법을 발전시켰다. 여기서 설명한 방법은 "패시픽 파이터"에서 성공적으로 사용되어 10cm 에서 40km 범위의 물 표면을 사실적으로 만들어냈다. 이는 현대 모의 비행에도 적용 가능하다. 우리는 눈에 보이는 물 표면 전체에서 바둑판형의 불필요한 부분을 제거할 수 있었다.

미래의 하드웨어는 수동적으로 텍스처 값을 필터링할 과정을 없애고 버텍스 셰이더의 수행 능력을 높임으로써 구현력을 강화할 것이다.

그리고 여기서 설명한 방법은 물 표면에 부딪침과 상세한 결을 표현하는 패럴랙스 매핑과 같은 고급 음영 기술을 사용해 품질을 높일 수 있다. 8장 "거리 기능을 통한 픽셀 당 변위 매핑"에서 이와 같은 방법 하나를 확인할 수 있다.

마지막으로 반사가 큰 물 표면은 큰 빛 변동을 나타내므로 HDR(high-dynamic-range) 기술을 통해 빛 계산은 큰 이점을 얻을 수 있을 것이다.

18.4 리소스

Fournier, Alain, and William T. Reeves. 1986. *Computer Graphics (Proceedings of SIGGRAPH 86)*의 pp. 75-84 "A Simple Model of Ocean Waves"

Kryachko, Yuri. 2004. "Modelling Sea and River Water with Pixel Shader 2.0." Presentation. http://www.kriconf.ru/2004/rec/KRI-2004.Programming_20.ppt

Tessendorf, Jerry. 2001. "Simulating Nature: Realistic and Interactive Techniques,"의 *SIGGRAPH 2001* course notes, course 47 "Simulating Ocean Water."