GPU를 이용한 실시간 바다 시뮬레이션 및 렌더링

이동민° 이성기
경북대학교 컴퓨터과학과
dmlee°@mail2.knu.ac.kr, sklee@knu.ac.kr

Real-time Ocean Wave Simulation and Rendering using GPU

Dongmin Lee°, Sung-kee Lee
Dept. of Computer Science, Kyungpook National University

요 약

자연현상 중의 하나인 바다의 파도는 그 규모가 거대할 뿐만 아니라 일정한 형태가 없이 바람에 의해서 계속해서 움직이며 주위의 사물과 상호작용한다. 이러한 바다를 컴퓨터를 통해 모방하고 표현하기란 쉽지 않으며 많은 계산시간을 필요로 한다. 본 논문에서는 그래픽스 하드웨어를 사용하여 움직이는 바다 영상을 실시간으로 생성하기 위한 방법을 제안한다. Gerstner 모델과 스펙트럼 모델을 기반으로 그래픽스 하드웨어에서 생성된 저해상도의 비균일격자 메쉬와 고해상도의 균일격자 법선 텍스쳐를 사용하여 바다를 표현한다. 전체과정이 그래픽스 하드웨어에서 처리되기 때문에 렌더링시에 시스템 메모리로부터 그래픽스 하드웨어로의 데이터전송에 따른 병목현상을 예방할 수 있을 뿐만 아니라 CPU자원을 차지하지 않기 때문에 컴퓨터 게임과 같이 CPU에 많은 연산이 집중되는 실시간 애플리케이션에 활용도가 크다. 또한 제안하는 방법은 잔잔한 바다뿐만 아니라 거칠고 파도가 높은 바다의 모습도 쉽고 빠르게 표현할 수 있다.

1 서론

대규모의 자연 현상을 컴퓨터로 재현해내기 위해서는 많은 계산시간이 필요하다. 특히 컴퓨터 게임과 같은 실시간 애플리케이션에서는 많은 양의 계산과 사람과의 상호작용이 동시에 요구되기 때문에 더욱 이러한 영상을 만들어 내기가 어렵다. 최근 그래픽스 하드웨어의 발전으로 개인용 컴퓨터에 소위 GPU (Graphics Processing Unit)라고 하는 추가적인 연산 장치가 탑재됨에 따라 이러한 문제를 풀기 위한 실마리를 제공하였다.

본 논문에서는 대규모 자연현상의 하나인 바다의 모습을 그래픽스 하드웨어를 사용하여 실시간으로 표현하기 위한 방법을 제안한다. 시점은 수면 위에서 자유롭게 움직이며, 수면의 파도는 바람에 의해서만 일어나고 주변의 장애물이나 수심에 의해서는 영향을 받지 않는 것으로 가정한다.

본 논문의 핵심 공헌은 스펙트럼 물결 모델과 Gerstner 물결 모델을 기반으로 이들을 합성하여 잔잔한 바다뿐만 아니라 거친 바다도 쉽고 빠르게 표현할 수 있도록 한 것이다. 또한 전체 시뮬레이션 및 렌더링 과정을 그래픽스 하드웨어에서 처리하여 일정한 영상의 질을 유지하면서 실시간으로 바다의 모습을 렌더링 할 수 있다.

본 논문의 구성은 다음과 같다. 2장에서 바다의 모습을 표현하기 위한 기존의 연구들을 살펴보고, 3장에서는 스펙트럼 물결 모델과 Gerstner 물결 모델에 대해 먼저 살펴본 후, 제안하는 합성 모델에 대해서 설명한다. 4장에서는 제안하는 모델의 그래픽스 하드웨어에서의 구현과 렌더링 방법을 설명한다. 5장에서는 제안하는 모델을 성능을 알아보기 위해서 각각 CPU와 GPU를 사용하는 두 개의 시스템으로 구현하고 이 둘의 성능을 비교하였다. 마지막으로 6장에서 본 논문의 결론을 내린다.

2 기존 연구

전통적으로 바다의 모습을 표현하는 데에는 Gerstner 모델과 스펙트럼 모델이 사용되어왔다. Gerstner 모델에서는 파도의 움직임을 일련의 물 입자들이 고정된 위치를 중심으로 원운동하는 것으로 본다. Thon은 실제와 같은 바다의 모습을 표현하기 위해서는 트로코이드의 진폭과 주파수 등의 특성 값을 올바르게 선택하여야 함을 지적하고, 스펙트럼의 유사함을 이용하여 정현파의 스펙트럼으로부터 토로코이드 스펙트럼을 추출하여 사용하였다[1]. Hinsinger는 Gerstner 모델에 적응 기법 (adative scheme)을 적용하여 실시간으로 바다의 모습을 표현하고자 하였다[2].

Mastin에 의해 처음 소개된 스펙트럼 물결 모델은 실제 바다에서 측정된 스펙트럼으로부터 Fourier 변환을 통해 바다 표면을 나타내는 높이 장 (height field)을 생성한다[3]. 하지만 바다를 둥근 정현파의 중첩으로 표현하기 때문에 주로 잔잔한 바다를 표현하는데 사용되었다. Tessendorf는 Pierson-Moskowitz 스펙트럼 대신에 Phillips 스펙트럼을 사용하고, 분산성 전파특성을 이용하여 뾰족한 파도의 모양을 나타낼 수 있는 방법을 제안하였다[4]. Mitchell은 스펙트럼 모델을 기반으로 모든 과정이 그래픽스 하드웨어에서 처리되는 바다 표현 기법을 제시하였다. 메쉬를 변형하기 위한 높이 맵과 세부적인 묘사를 위한 법선 맵을 Fourier 변환을 통해 생성하였다. 하지만 높이 맵과 화면에 그려지는 메쉬가 1:1 대응이 되도록 하여 상대적으로 좁은 영역의 수면을 표현하는데 치중하였다[5].

3 물결 모델

3.1 스펙트럼 물결 모델

스펙트럼 모델은 해수면을 여러 정현파의 중첩으로 나타낼 수 있다는 원리를 기반으로 한다. 높이 장을 정현파로 분해하거나 분해된 정현파 스펙트럼으로부터 다시 높이 장을 만드는 데에는 다수의 정현파의 합을 빠르게 계산할 수 있는 Fourier 변환이 사용된다. 스펙트럼 모델의 설명에는 [5]의 표기법을 따르고, 편의상 파도가 없는 해수면은 xz 평면상에 놓이고, $+y$축이 수면 위쪽을 가리키는 것으로 가정한다. 이 가정은 뒤에 설명할 Gerstner 모델에서도 동일하게 적용된다.

스펙트럼 모델에서 임의의 시간 t, 임의의 위치 $X=(x,z)$에서 수면의 높이 $h(X,t)$는 수식 (1)과 같이 시간에 따라 바뀌는 진폭을 가진 다수의 정현파의 합으로 정의된다.

$$h(K,t)=\sum_{K}\tilde{h}(K,t)\exp(jK\cdot X) \quad (1)$$

이때, $K=(k_x,k_z)$는 2차원 파형벡터로서, $k_x=2\pi n/L_x$, $k_z=2\pi m/L_z$이다. L_x,L_z는 각각 시뮬레이션 영역의 가로, 세로 크기이고, N, M을 각각 2차원 Fourier 변환 영역의 가로, 세로 해상도라고 할 때, n, m은 각각 $-N/2\le n<N/2$, $-M/2\le m<M/2$ 사이의 정수이다. 따라서 높이 장은 2차원 균일격자의 각 점 $X=(nL_x/N, mL_z/M)$에서 계산된다.

Fourier 변환에 의해 계산된 높이 장은 넓은 영역의 바다를 표현하기 위해 반복해서 이어 붙일 수 있고, 높이 값이 실수가 되어

야 한다. 이를 위해서는 주파수 영역의 입력 값이 복소켤레 특성 (complex conjugation property)를 만족하는 복소수이어야 한다. Tessendorf는 이러한 조건을 만족하는 입력 값을 만들기 위해서 임의의 시간 t에 주파수 영역의 입력값 $\tilde{h}(K,t)$를 수식 (2)와 같이 정의하였다. 본 연구에서는 이 방법을 그대로 따른다.

$$\tilde{h}(K,t) = \tilde{h_0}(K)\exp[jw(k)t] + \tilde{h_0^*}(K)\exp[-jw(k)t] \quad (2)$$

이렇게 함으로써 복소켤레 특성, $\tilde{h}^*(K,t) = \tilde{h}(-K,t)$를 만족하는 입력 값을 만들 수 있고, 다른 입력 값에 관계없이 특정 시간의 높이 장을 계산할 수 있다. 이 때 $\tilde{h_0}(K)$는 시간 $t=0$에서의 초기 진폭 값으로서 수식 (3)에서와 같이 백색 잡음 (white noise)을 Phillips 스펙트럼으로 필터링하여 생성한다.

$$\tilde{h_0}(K) = 1/\sqrt{2}(\xi_r + \xi_i)\sqrt{P_h(K)} \quad (3)$$

Phillips 스펙트럼에 대한 설명은 [4]를 참조한다.

높이 장뿐만 아니라 렌더링에 필요한 법선 벡터도 수식 (4)와 같이 Fourier 변환을 통해 계산할 수 있다.

$$\epsilon(X,t) = \sum_K jK\tilde{h}(K,t)\exp(jK \cdot X) \quad (4)$$

물론 높이 장을 계산한 후에 유한 차분법 (finite difference method)을 통해 법선 벡터를 계산할 수도 있으나, 이 경우 격자 간격보다 작은 물결에 의한 법선 벡터의 변화는 무시되기 때문에 보다 좋은 렌더링 결과를 얻기 위해서는 Fourier 변환을 통해 계산된 해석학적 법선 벡터를 사용하는 것이 좋다.

또한 생성된 높이 장은 정현파의 중첩이기 때문에 둥근 물마루를 가지는데 실제 바다는 파도가 거의 없이 잔잔한 경우에도 둥글기 보다는 뾰족한 물마루를 가진다. Tessendorf는 이를 표현하기 위해 Lie transform 기법을 적용하여, 수식 (5)와 같은 변위벡터를 사용하였다. 이 역시 Fourier 변환을 통해 계산된다.

$$D(X,t) = \sum_K -j\frac{K}{k}\tilde{h}(K,t)\exp(jK \cdot X) \quad (5)$$

변위 벡터를 적용한 새로운 격자의 위치는 $X+\alpha D(X,t)$가 된다. 이때 α는 변위 벡터에 의한 영향을 조절하는 상수이다. 이를 통해 트로코이드처럼 뾰족한 물마루와 평평한 골을 표현할 수 있다.

3.2 Gerstner 물결 모델

Gerstner 모델에서 바다는 안정시 위치를 중심으로 해수면에 수직으로 원운동하는 입자들에 의해 표현된다. 안정시 위치가 $X_0 = (x_0, z_0)$인 입자는 임의의 시간 t에 진폭이 A인 물결이 지나감에 따라 수식 (6)과 같이 움직이게 된다.

$$\begin{aligned} x &= x_0 + \sum A\frac{k_x}{|K|}\cos(K \cdot X_0 - wt) \\ y &= \sum A\sin(K \cdot X_0 - wt) \\ z &= z_0 + \sum A\frac{k_z}{|K|}\cos(K \cdot X_0 - wt) \end{aligned} \quad (6)$$

수식 (6)에서 A는 진폭이고, K는 파형 벡터이다.

3.3. 합성 물결 모델

스펙트럼 모델은 많은 수의 정현파의 중첩을 빠른 속도로 계산할 수 있지만 계산에 사용되는 정현파의 스펙트럼 범위가 제한적이기 때문에 바다의 거친 표면을 자세히 표현하는 동시에 큰 파도를 나타내기는 어렵다. 스펙트럼의 범위는 시뮬레이션 영역의 크기와 격자의 해상도에 따라 결정되는데 수식 (1)에서 정현파의 파장의 범위는 $\sqrt{2}L/N \sim L$ 이 된다. 현재의 하드웨어에서 실시간으로 처리되기 위해서 $N \leq 512$이하로 제한되고, 실제 바다와 같은 거친 표면을 나타내기 위해서는 파장이 짧은 정현파의 중첩이 다수 필요하기 때문에 $L \leq 512$로 제한된다. 따라서 파장이 긴 물결을 표현하기 어렵다.

물론 격자 간격과 해상도를 달리하여 좀 더 넓은 범위의 스펙트럼으로부터 Fourier 변환을 통해 생성된 높이 장을 사용할 수 있지만, 추가적인 Fourier 변환이 필요하다. 또한 계산시간을 줄이기 위해 격자간격을 크게 하면 앨리어싱 (aliasing) 때문에 영상의 질이 떨어지게 된다. 그리고 큰 물결의 경우 십여 개 정도의 정현파만으로도 충분히 사실적인 영상을 표현할 수 있기 때문에 이처럼 소수의 정현파의 중첩에 Fourier 변환을 사용하는 것은 적합하지 않다.

반면 Gerstner 모델은 스펙트럼 모델에 비해 계산량이 많아서 빠른 시간 내에 많은 수의 정현파의 중첩을 계산할 수는 없지만, 주파수 영역이 아닌 공간 영역에서 계산이 이루어지기 때문에 일정한 범위의 스펙트럼이 아닌, 임의의 스펙트럼을 가진 정현파의 중첩을 쉽게 계산할 수 있다. 또한 시뮬레이션 도중에 새로운 스펙트럼을 추가하거나 쉽게 변경할 수 있어 실시간으로 바다 표면의 특성을 변화시킬 수 있다. 뿐만 아니라 스펙트럼 모델에서 표현하기 어려운 원형 물결 등의 다양한 형태의 파도를 쉽게 표현할 수 있다.

이처럼 두 바다 모델을 조합함으로써 각각의 단점을 보완하고 장점을 살린다. Gerstner 모델에 의해 큰 물결을 표현함과 동시에 스펙트럼 모델에 의해 높은 해상도의 높이 맵과 법선 맵을 생성하여 거친 바다 표면을 세밀하게 묘사할 수 있다. 스펙트럼 모델에 의해 생성된 높이 장은 보다 넓은 영역의 바다를 표현하기 위해 반복해서 이어 붙여지는데 이 때 반복 패턴이 뚜렷하게 보이는 단점이 있다. 연속함수로 정의되는 Gerstner 모델과 합성되기 때문에 이러한 패턴이 보이지 않게 희석된다.

4 GPU 구현과 렌더링

4.1 스펙트럼 모델의 구현

GPU의 병렬 처리구조에 적합한 Decimation-in-time Fourier 변환 알고리즘을 사용한다[6]. N 개의 입력 데이터에 대해 $\log_2 N$ 개의 단계로 나누어지고, 각 단계마다 N 개의 입력데이터를 미리 정해진 상수와 곱하여 더하는 방식으로 N 개의 출력데이터를 생성한다. 각 단계에서 곱해지는 상수와 사용될 입력 데이터를 나타내는 색인은 CPU에서 미리 계산되어 텍스쳐의 형태로 비디오 메모리에 저장된다.

그래픽스 하드웨어의 렌더러블 텍스쳐 (renderable texture)는 일반 텍스쳐처럼 사용되어 데이터를 읽어올 수도 있고, 렌더링 대상이 되어 새로운 데이터를 쓸 수도 있다. 하지만 하나의 렌더러블 텍스쳐를 동시에 입력과 출력에 사용할 수 없기 때문에 소위 핑퐁 (ping-pong) 기법을 사용한다. 즉, 동일한 크기를 가진 2개의 렌더러블 텍스쳐를 만들고, 매 계산과정마다 입력 텍스쳐와 출력 텍스쳐를 맞바꾸면서 계산을 수행한다.

Phillips spectrum에 의해 필터링된 초기 진폭 값이 저장된 텍스쳐를 사용하여 매 프레임마다 수식 (2)에 의해 특정시간에서의 진폭값을 GPU에서 계산하고 이를 역 Fourier 변환하여 높이 장을 생성한다. 앞서 설명한 것처럼 높이 장뿐만 아니라 법선 벡터, 변위 벡터도 Fourier 변환에 의해 계산된다. 현재의 그래픽스 하드웨어는 MRT (Multiple rendering target) 기능을 지원하여 최대 4개의 렌더러블 텍스쳐에 동시출력이 가능하다. 또한 Fourier 변환의 입력값이 복소켤레 특성을 만족하므로 2 개의 입력 데이터 $F(u)$, $G(u)$를 $H(u) = F(u) + jG(u)$로 합성하여 Fourier 변환하면 그 결과가 $h(u) = f(u) + jg(u)$가 되어 2 개의 Fourier 변환을 한 번에 계산할 수 있다.

전체적인 시뮬레이션 과정은 그림 1과 같다. 텍스쳐에 저장되어 있는 초기 진폭 데이터 $\tilde{h_0}(K)$로부터 시간 t에서의 Fourier 변환에 대한 입력 값을 계산한다. 법선 벡터와 변위 벡터는 x, z 각 방향의 입력 데이터를 더하여 하나의 입력 데이터로 변경된다. 가로방향 1차원 Fourier 변환과 세로방향 1차원 Fourier 변환을 거쳐 최종 계산 결과는 2개의 텍스쳐에 나뉘어 저장된다. 렌더링 과정에서 높이 장과 변위 벡터는 정점 셰이더에서 샘플링 되기 때문에 하나의 RGBA 텍스쳐에 저장되고, 법선 벡터는 픽셀 셰이더에서 샘플링되기 때문에 별도의 RG 텍스쳐에 저장된다. 이렇게 함으로써 2 개의 렌더링 타겟만으로 필요한 모든 Fourier 변환을 처리할 수 있다.

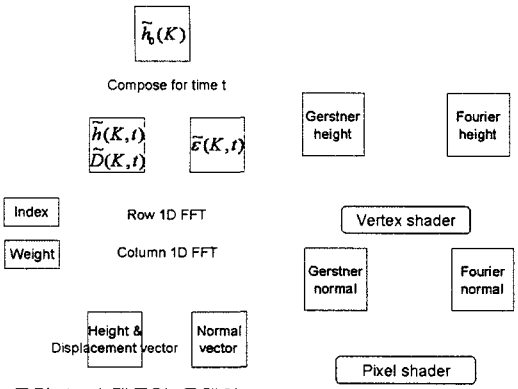

그림 1. 스펙트럼 모델의 시뮬레이션 과정.

그림 2. 렌더링 과정.

4.2 Gerstner 모델의 구현

단순히 정현파의 덧셈이기 때문에 특별한 알고리즘 없이 간단하게 구현할 수 있다. GPU의 벡터 연산을 이용하여 CPU에서 보다 빠른 계산이 가능하고 역시 계산결과를 렌더링에 바로 사용할 수 있다는 점에서 GPU로 구현하였다.

4.3 렌더링

생성된 높이 맵과 법선 맵은 그림 2에서처럼 각각 정점 셰이더와 픽셀 셰이더에서 샘플링 된다. Gerstner 모델의 높이 맵과 법선 맵은 상대적으로 큰 파도를 나타내기 때문에 각 정점에서 샘플링되고, 자세한 바다 표면의 모습을 나타내는 스펙트럼 모델의 법선 맵은 각 픽셀에서 샘플링된다.

셰이딩 방법은 일반적으로 물을 렌더링 하는 방법을 따른다. 즉, 주변환경을 나타내는 큐브맵 (cubemap)과 지역반사를 나타내는 반사 맵 (reflection map), 물 밑의 환경을 나타내는 굴절 맵 (refraction map)을 사용하여 Fresnel 항에 의해 보간된 최종 색상을 셰이딩에 사용한다.

5. 실험 및 결과 분석

우리는 제안하는 방법을 CPU와 GPU에서 각각 구현하고 이들의 성능을 알아보기 위해 렌더링시에 초당 프레임수를 비교하였다. CPU를 사용한 구현에서는 FFTW 라이브러리[7]를 사용하였다. 스펙트럼 모델의 법선 맵을 제외하고 모든 계산과 샘플링은 CPU에서 처리된다. 스펙트럼 모델의 법선 맵은 CPU에서 계산되지만, 렌더링시에 메쉬의 격자간격에 상관없이 세밀한 바다 표면을 나타내기 위해 GPU 구현에서와 같이 픽셀 셰이더에서 샘플링된다. 실험에 사용된 컴퓨터에는 1GB의 시스템 메모리와 3GHz의 Pentium 4 CPU, 그리고 256MB의 비디오 메모리를 가진 GeForce 6800 Ultra 그래픽 카드를 사용하였다. 렌더링된 윈도우의 해상도는 1024 x 768이다.

표1과 표2는 각각 스펙트럼 모델만을 사용하여 바다를 렌더링한 경우와, 스펙트럼 모델과 Gerstner 모델을 합성한 경우에 해당하는 초당 프레임수를 나타낸 것이다. 전체적으로 GPU로 구현된 시스템의 성능이 우월하며, 스펙트럼 모델과 Gerstner 모델의 해상도가 증가할수록 성능의 차이가 커진다.

표 1. 스펙트럼 모델을 사용한 경우 fps.

FFT 해상도	CPU 구현	GPU 구현
64 x 64	63	79
128 x 128	58	57
256 x 256	26	40
512 x 512	3	17

표 2. 제안하는 합성 모델을 사용한 경우 fps.

Gerstner 해상도	CPU 구현	GPU 구현
32 x 32	51	62
64 x 64	30	56
128 x 128	14	48
256 x 256	5	33

이는 GPU가 CPU와 달리 병렬처리 특성이 강해서 많은 양의 데이터를 더욱 빠르게 처리할 수 있기 때문이다. 또한 시뮬레이션과정에서 생성된 메쉬 정보와 법선 맵 등이 시스템 메모리가 아닌 그래픽스 하드웨어에 바로 저장되기 때문에 렌더링시에 데이터 전송으로 인한 지연이 발생하지 않기 때문이기도 하다.

6 결론

본 논문에서는 모든 과정을 그래픽스 하드웨어에서 처리하여 움직이는 바다의 모습을 실시간으로 시뮬레이션 하고 렌더링하는 방법을 제안하였다. Gerstner 모델을 통해 큰 파도에 의해 결정되는 바다의 전체적인 구조를 메쉬로 나타내고, 스펙트럼 모델을 통해 거칠고 세밀한 바다 표면의 모습을 노말 맵 텍스쳐로 표현한다. Gerstner 모델은 격자간격이 성근 2차원 비균일격자에서, 스펙트럼 모델은 조밀한 2차원 균일격자에서 높이 맵과 법선 맵을 각각 생성한다. 생성된 높이 맵과 법선 맵은 정점 셰이더와 픽셀 셰이더에서 샘플링되어 바다를 나타내는 메쉬의 위상을 변화시키고, Fresnel 항의 계산을 통해 실제 바다와 같은 자연스러운 셰이딩에 사용된다.

제안하는 방법을 통해 그림 3과 같이 잔잔한 바다뿐만 아니라 거칠고 파도가 높은 바다의 모습도 실시간으로 빠르게 표현할 수 있었다. 전체 과정을 GPU에서 처리함으로써 CPU에서 처리하는 것에 비해 계산 속도를 향상시켰다. 따라서 컴퓨터 게임과 같이 CPU에 많은 연산부하가 걸리는 실시간 애플리케이션에서 활용될 수 있을 것으로 기대한다.

그림 3. 제안하는 방법으로 렌더링된 바다.

참고 문헌

[1] Thon S. et al., "Ocean Waves Synthesis using a Spectrum-based Turbulence Function," *Computer Graphics International 2000*, 2000.

[2] Hinsinger D. et al., "Interactive Animation of Ocean Waves," *Proc. of the 2002 ACM SIGGRAPH/Eurographics Symposium on Compute Animation*, 2002.

[3] Mastin G. A., et al., "Fourier Synthesis of Ocean Scenes," *IEEE Computer Graphics and Applications*, 1987.

[4] Tessendorf J., "Simulating Ocean water," *In SIGGRAPH Course Notes*, Addison-Wesley, 1999.

[5] Mitchell J. L., "Real-Time Synthesis and Rendering of Ocean Water," ATI Research Technical Report, 2005.

[6] Sumanaweera T. et al. "Medical Image Reconstruction with the FFT," GPU GEMS 2, Addison-Wesley, 2005.

[7] "FFTW library," http://www.fftw.org/