Real-time Simulation of Ocean Water Wave Using GPU

GPU를 활용한 해수면의 실시간 시뮬레이션

Gyeong-Rok Min[1] and H. Lee[1, 2]

[1] Department of Physics, Konkuk University, Seoul, 05029, Republic of Korea

[2] A graduation thesis advisor

작성일자: June 7, 2023

ABSTRACT

내용: 중력을 받는 비압축성 유체의 오일러 방정식의 표면에서의 정확한 해인 거스트너 파동함수를 모델로 하여 해수면을 시뮬레이션한다.

목표: GPU를 활용하여 해수면의 운동을 실시간으로 시뮬레이션함으로써 학부 동안에 배운 물리학의 유용성을 시각적으로 구현한다.

방법: DirectX 11 API 와 HLSL shader를 활용하여 프로그램을 작성한다.

결과: 프로그램 코드 : `https://github.com/12equal34/experiment-Gerstner_Waves`

제 1 절 서론

강원도 동해 앞바다를 바라보면 파도가 일렁이는 해수면의 운동에 대해 수학적으로 기술하고 실시간으로 시뮬레이션을 하고 싶어진다. 수면의 운동은 해당 위치에서 바닥면까지의 깊이에 따라 달라진다. 수심이 얕은 해안가 근처로 갈수록 파도가 점점 높아진다. 바람의 영향에 의해 파도는 점점 거세진다. 파도가 높이 치솟거나, 다른 파도 또는 수면과 부딪치면서 생겨나는 물보라와 거품들은 물의 표면장력을 이기지 못하고 입자들이 흩어지는 현상이다. 해안가 위에 떠다니는 거품들은 유기물들이 파도에 의해 섞이면서 비눗방울처럼 만들어진다. 지구와 달의 중력으로 인해 민물과 썰물이 오가며, 주변의 기후 및 지형 변화에 의한 해류들의 운동은 수학적으로 기술하기 어렵게 만든다. 이토록 복잡한 해안가의 풍경을 시뮬레이션하기란 쉽지 않을 것이다.

수면의 깊이가 무한하고 다른 환경의 영향(지구의 회전 운동)들을 무시한다고 가정했을 때, 거스트너 파동함수(Gerstner Waves)는 중력을 받는 비압축성 및 비점성 유체 표면에서의 오일러 방정식의 정확한 해이다. 이는 모든 표면 입자들이 각각이 따로 원운동을 하는 파동함수이다. 거스트너 파동함수는 컴퓨터 그래픽스 분야에서 파도를 렌더링하기 위해 사용되는 모델이며 실시간 시뮬레이션을 위해서 고속 푸리에 변환을 사용하여 계산한다.[1]

제 2 절 거스트너 파동함수(Gerstner Waves)

일정한 중력을 받는 2차원 상의 비점성 및 비압축성을 가진 유체의 표면을 생각하자. x축을 파동의 진행방향으로, y축을 중력의 반대방향으로 정의한다.

$\rho = 1$이라고 가정을 한 오일러 방정식은 다음과 같다.

$$\frac{\partial \boldsymbol{u}}{\partial t} + (u \cdot \nabla)u = -\nabla w + \boldsymbol{g}$$

2차원 상에서 속도장 $\boldsymbol{V} = \big(u(t,x,y),\ v(t,x,y) \big)$ 이라고 하면 다음이 성립한다.

$$\begin{cases} u_t + uu_x + vu_y = -P_x \\ v_t + uv_x + vv_y = -P_y - g \end{cases}$$

유체의 밀도가 일정하므로

$$u_x + u_y = 0.$$

한편, 수면 높이 $y = \eta(t,x)$에 대해서 평균적인 수면 높이가 0이라고 가정하면 $\int_R \eta(t,x) = 0$이 성립한다.

표면 장력을 무시한다면 경계 조건으로

$$\begin{aligned} P &= P_0 && \text{on} \quad y = \eta(t,x), \\ v &= \eta_t + u\eta_x && \text{on} \quad y = \eta(t,x), \\ (u,v) &\to (0,0) && \text{as} \quad y \to -\infty \quad \text{uniformly for } x \in \mathbf{R}. \end{aligned}$$

가 성립한다.

주어진 식들을 정리하여 풀면 다음과 같은 결과식들을 얻는다.[2]

$$\begin{cases} \eta(t,x) = a\cos(kx - wt), \\ u(t,x,y) = aw\exp(ky)\cos(kx - wt), \\ v(t,x,y) = aw\exp(ky)\sin(kx - wt), \\ P(t,x,y) = P_0 - gy + ag\exp(ky)\cos(kx - wt). \end{cases}$$

여기서 다음이 성립한다.

$$\begin{aligned} k &= 2\frac{\pi}{\lambda} \\ w &= \sqrt{gk} \end{aligned}$$

Direct3D API에서 왼손 좌표계를 사용하므로 3차원 왼손 좌표계에서 y축을 중력의 반대 방향으로, xz평면을 y축을 법선 벡터로 갖는 평면으로 정의하고 z축을 관측자의 관찰방향으로, x축을 z축과 y축의 수직방향으로 둔다. 이제 2차원에서 유도한 식을 3차원에서의 식으로 확장하면 M개의 거스트너 파동함수에 대해 m번째 파수 벡터$(k_{x,m}, k_{z,m})$과 매개변수 α, β에 대해 평균 표면 위의 한 점의 위치벡터$\boldsymbol{r} = (\alpha, \beta, 0)$와 연결된 거스트너 파동의 자유표면의 위치벡터$\boldsymbol{r} = (x,z,y)$에 대해 다음이 성립한다.

$$\begin{aligned} x &= \alpha - \sum_{m=1}^{M} \frac{k_{x,m}}{k_m} a_m \sin(\theta_m), \\ z &= \beta - \sum_{m=1}^{M} \frac{k_{z,m}}{k_m} a_m \sin(\theta_m), \\ y &= \sum_{m=1}^{M} a_m \cos(\theta_m), \\ \theta_m &= k_{x,m}\alpha + k_{z,m}\beta - w_m t - \phi_m \end{aligned}$$

여기서 다음이 성립한다.

$$\begin{aligned} k_m &= \sqrt{k_{x,m}^2 + k_{z,m}^2} \\ w_m &= \sqrt{gk_m} \end{aligned}$$

픽셀 셰이더에서 해수면의 색깔을 결정하기 위해 표면에서의 법선 벡터를 계산한다.

$$\boldsymbol{n} = \frac{\partial r}{\partial \alpha} \times \frac{\partial r}{\partial \beta}$$

$$= \begin{vmatrix} \hat{x} & \hat{z} & \hat{y} \\ 1 - \sum_{m=1}^{M} \frac{k_{x,m}^2}{k_m} a_m \cos(\theta_m) & -\sum_{m=1}^{M} \frac{k_{z,m} k_{x,m}}{k_m} a_m \cos(\theta_m) & -\sum_{m=1}^{M} k_{x,m} a_m \sin(\theta_m) \\ -\sum_{m=1}^{M} \frac{k_{z,m} k_{x,m}}{k_m} a_m \cos(\theta_m) & 1 - \sum_{m=1}^{M} \frac{k_{z,m}^2}{k_m} a_m \cos(\theta_m) & -\sum_{m=1}^{M} k_{z,m} a_m \sin(\theta_m) \end{vmatrix}$$

제 **3** 절 해수면의 실시간 시뮬레이션 프로그램

Direct3D 11 및 Win32 API를 사용하여 실시간 시뮬레이션 프로그램을 작성했다. DirectX는 1995년 9월 30일에 첫 등장해, 마이크로소프트가 Windows 95 시절부터 개발해 오고 있는 윈도우용 종합 멀티미디어 라이브러리다. 멀티 미디어 및 게임 프로그래밍에서 널리 사용되며, 구성요소로 3D 그래픽을 그리는 Direct3D API를 포함하여 현재는 버전이 12까지 나와있고 여기서는 편의상 11 버전을 이용했다.

실험에서는 임의의 8개의 거스트너 파동함수를 중첩하여 계산하였다.

```
//-----------------버텍스  셰이더    프로그램 -----------------
struct VS_In
{
    float3 pos : Position;
};

struct VS_Out
{
    float4 pos : SV_Position;
    float3 normal : Normal;
};

cbuffer Transform : register( b0 )
{
    matrix model;
    matrix modelView;
    matrix modelViewProj;
    matrix modelRotation;
    matrix cameraRotation;
};

cbuffer GlobalParameter : register( b1 )
{
    float time;
}

static const int numWave = 8;

cbuffer WaveParameter : register( b2 )
{
    struct
    {
        float kx;
        float kz;
        float k;
```

```
        float w;
        float a;
        float phi;
        float _1;
        float _2;
    } wave[numWave];
};


VS_Out main( VS_In input )
{
    VS_Out output;
    float4 pos = float4( input.pos, 1.0f );

    float S_KxDkAS = 0.0f;
    float S_KzDkAS = 0.0f;
    float S_AC = 0.0f;
    float S_KxxDkAC = 0.0f;
    float S_KzxDkAC = 0.0f;
    float S_KzzDkAC = 0.0f;
    float S_KxAS = 0.0f;
    float S_KzAS = 0.0f;
    for (int m = 0; m < numWave; ++m)
    {
        float theta = dot( float2( wave[m].kx, wave[m].kz ), pos.xz ) - wave[m].w * time - wave[m].phi;
        float AC = wave[m].a * cos( theta );
        float AS = wave[m].a * sin( theta );
        float KxAS = wave[m].kx * AS;
        float KzAS = wave[m].kz * AS;
        float DkAC = AC / wave[m].k;
        float DkAS = AS / wave[m].k;
        float KxDkAS = wave[m].kx * DkAS;
        float KzDkAS = wave[m].kz * DkAS;
        float KxxDkAC = wave[m].kx * KxDkAS;
        float KzxDkAC = wave[m].kz * KxDkAS;
        float KzzDkAC = wave[m].kz * KzDkAS;

        S_KxDkAS += KxDkAS;
        S_KzDkAS += KzDkAS;
        S_AC += AC;
        S_KxxDkAC += KxxDkAC;
        S_KzxDkAC += KzxDkAC;
        S_KzzDkAC += KzzDkAC;
        S_KxAS += KxAS;
        S_KzAS += KzAS;
    }
    pos.x = pos.x - S_KxDkAS;
    pos.z = pos.z - S_KzDkAS;
    pos.y = pos.y + S_AC;

    float3 dpos_dalhpa = float3( 1.0f - S_KxxDkAC, -S_KzxDkAC, -S_KxAS );
    float3 dpos_dbeta = float3( -S_KzxDkAC, 1.0f - S_KzzDkAC, -S_KzAS );
    float3 normal = cross( dpos_dalhpa, dpos_dbeta );
```

```
    output.pos = mul( pos, modelViewProj );
    output.normal = (float3) mul( mul( float4( normal, 0.0f ), modelRotation ), cameraRotation );
    return output;
}
//---------- 픽셀  셰이더   프로그램  -------------------------
struct VS_Out
{
    float4 pos : SV_Position;
    float3 normal : Normal;
};

cbuffer Light : register( b0 )
{
    float4 light_ambient;
    float4 light_color;
    float3 light_direction;
};

float4 main( VS_Out input ) : SV_TARGET
{
    static float4 baseColor = float4( 0.4f, 0.4f, 1.0f, 1.0f );

    float3 normal = normalize( input.normal );
    float3 ambient = light_ambient.rgb * light_ambient.a * baseColor.rgb;

    float3 diffuse = (float3) 0;
    float n_dot_l = dot( light_direction, normal );
    if (n_dot_l > 0)
    {
        diffuse = light_color.rgb * light_color.a * n_dot_l * baseColor.rgb;
    }

    return float4( ambient + diffuse , baseColor.a );
}
//---------- 응용  프로그램  코드  조각  ----------------------
// constant buffer for wave parameters in VS
{
    struct Wave {
        float wave_number_x;
        float wave_number_z;
        float wave_number;
        float wave_angular_frequency;

        float wave_amplitude;
        float wave_phase;
        float _1, _2;
    };

    std::vector<std::tuple<float, float, float, float>> init {
        {0.3f,  -0.46f, 0.2f, 0.0f },
        {-0.33f, 0.40f, 0.12f, 0.0f },
        {0.0f,  0.56f, 0.2f, 0.2f },
```

```
        {-0.33f, 0.0f, 0.12f, 0.3f },
        {0.3f,  -0.26f, 0.2f, 0.7f },
        {-0.33f, 0.20f, 0.12f, 0.9f },
        {0.17f, 0.3f,  0.3f, -1.0f},
        {-0.15f, -0.1f, 0.7f, 0.6f },
        {-0.15f, 0.1f, 0.6f, 0.5f },
        {0.15f, -0.1f, 0.8f, 0.4f },
        {0.15f, 0.1f,  0.3f, 0.3f },
        {-0.25f, -0.1f, 0.2f, 0.2f },
        {-0.15f, -0.25f, 0.4f, 0.1f },
    };

    std::vector<Wave> waves;
    waves.reserve(init.size());
    for (auto [kx, kz, a, phi] : init) {
        auto k = std::sqrtf(kx * kx + kz * kz);
        auto w = std::sqrtf(9.8f * k);
        waves.push_back(Wave(kx, kz, k, w, a, phi, 0, 0));
    }

    auto byteWidth = static_cast<UINT>(sizeof(Wave) * std::size(waves));
    ConstantBuffer waveParameterCbuf(*this, byteWidth, waves.data());
    waveParameterCbuf.SetToVertexShader(*this, 2u);
}
```

제 **4** 절 결과

실시간 해수면 시뮬레이션 프로그램의 결과 그림 1, 2를 볼 수 있다. 작성한 프로그램 및 코드는 깃헙 주소 : `https://github.com/12equal34/experiment-Gerstner_Waves` 에서 확인할 수 있다.

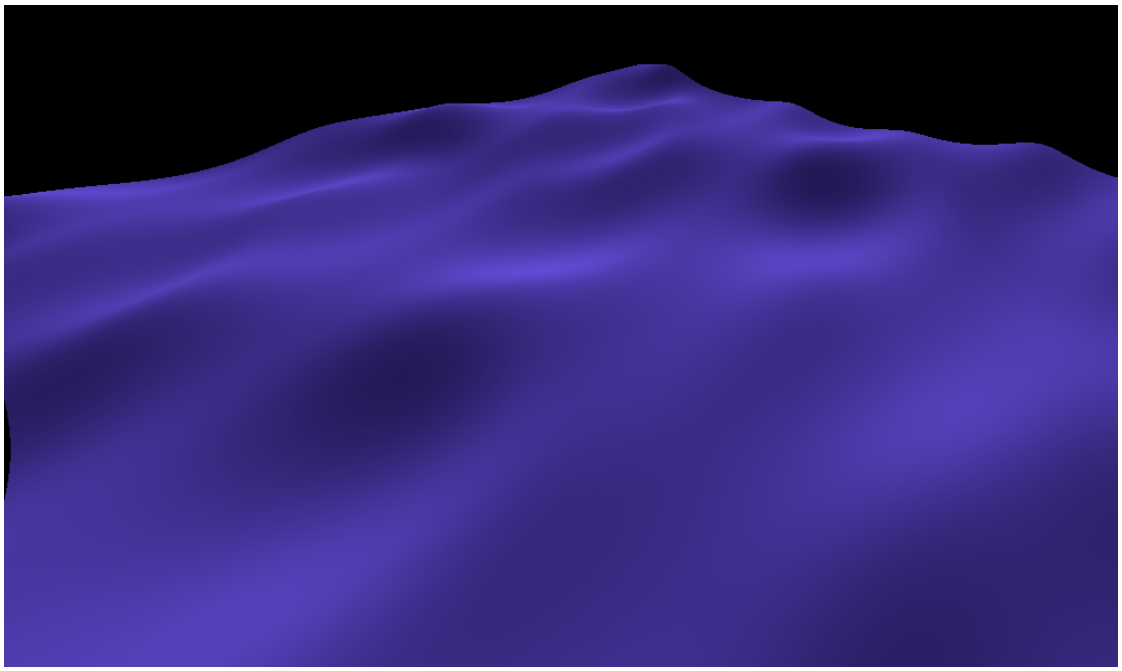

Fig. 1. 8개의 거스트너 파동함수를 중첩하여 해수면을 시뮬레이션한 결과

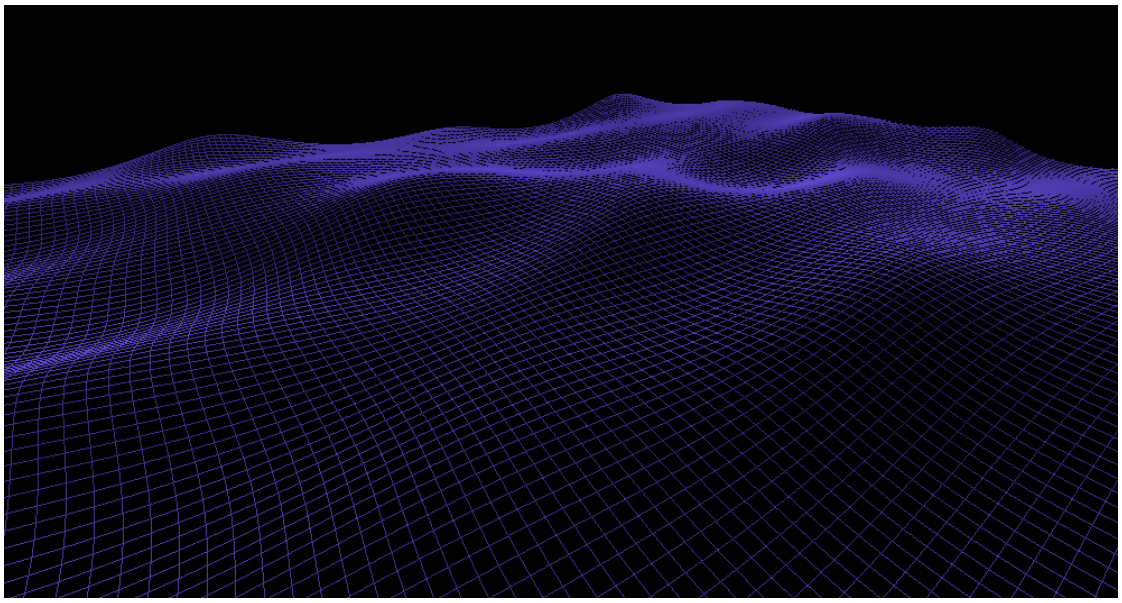

Fig. 2. 격자 모양의 해수면 시뮬레이션 사진

References

[1] Jerry Tessendorf et al. Simulating ocean water. *Simulating nature: realistic and interactive techniques. SIGGRAPH*, 1(2):5, 2001.

[2] Anca-Voichita Matioc. On particle trajectories in linear deep-water waves. *arXiv preprint arXiv:1111.2490*, 2011.