JKIICE Journal of the Korea Institute of Information and Communication Engineering

한국정보통신학회논문지 Vol. 23, No. 4: 416~422, Apr. 2019

절차적 함수를 이용한 GPU기반 실시간 3D구름 모델링 및 렌더링 기법

성만규*

GPU-based modeling and rendering techniques of 3D clouds using procedural functions

Mankyu Sung*

*Associate Professor, Department of Game & Mobile, Keimyung University, Daegu, 42601 Korea

요 약

본 논문은 절차적함수를 이용하여 실시간으로 3차원 구름을 모델링하고 렌더링하는 알고리즘을 제안한다. 구름 모델링은 절차적 노이즈 함수인 fbm(Fractional Brownian Motion)을 변형하여 사용하며, 이 값을 대기의 수증기 밀도 값으로 이용한다. 이 밀도 값은 파라미터로 주어진 3가지 구름의 형태를 위해 변형되며 렌더링단계의 입력 값으로 들어간다. 레이마칭(ray marching)기법을 이용한 렌더링 단계에서는 이 밀도 값을 이용하여 구름의 색상을 결정하며 이때 밀도에 따른 빛의 감소 및 산란현상은 물리적으로 계산된다. 대기모델로 렌더링 된 하늘 위에 제안한 알고리즘에 의해 구현된 구름들이 블랜딩되며, 이 때 바람의 방향에 따라 구름이 움직이도록 한다. 제안된 구름 생성 및 렌더링은 GLSL언어를 이용해서 GPU상에서 구현되었다.

ABSTRACT

This paper proposes a GPU-based modeling and rendering of 3D clouds using procedural functions. The formation of clouds is based on modified noise function made with fbm(Fractional Brownian Motion). Those noise values turn into densities of droplets of liquid water, which is a critical parameter for forming the three different types of clouds. At the rendering stage, the algorithm applies the ray marching technique to decide the colors of cloud using density values obtained from the noise function. In this process, all lighting attenuation and scattering are calculated by physically based manner. Once we have the clouds, they are blended on the sky, which is also rendered physically. We also make the clouds moving in the sky by the wind force. All algorithms are implemented and tested on GPU using GLSL.

키워드 : 실시간 렌더링, 3차원 구름 렌더링, 구름 모델링, 레이 마칭

Keywords : Real-time Rendering, 3D cloud rendering, Cloud Modeling, Ray marching

Received 18 January 2019, **Revised** 27 January 2019, **Accepted** 21 February 2019

***Corresponding Author** Mankyu Sung(E-mail:mksung@kmu.ac.kr, Tel:+82-53-580-6684)
Associate Professor, Department of Game & Mobile, Keimyung University, Daegu, 42601 Korea

Open Access http://doi.org/10.6109/jkiice.2019.23.4.416 print ISSN: 2234-4772 online ISSN: 2288-4165

Ⅰ. 서 론

건물과 같은 인공물이 아닌 자연에서 쉽게 볼 수 있는 나무나 환경, 구름과 같은 자연물에 대한 모델링 및 렌더링은 게임과 같은 콘텐츠의 사실성을 증가 시키는데 중요한 역할을 한다. 하지만, 이와 같은 자연물에 대한 사실적인 모델링과 렌더링은 인공물에 비해 많은 기술적인 어려움을 갖고 있다. 첫째, 인공물의 모습을 정의하기 위한 물리적인 법칙을 찾기 어려우며, 둘째, 이와 같은 물리적 법칙을 찾더라도, 게임과 같은 실시간 콘텐츠에 적용하기 위해서는 이 법칙을 단순화 하여 실시간으로 동작되도록 하여야 하며, 이 또한 기술적인 어려움을 갖고 있다.

본 논문은 가장 대표적인 자연물인 구름(cloud)을 실시간으로 생성하고 자연스럽게 렌더링하기 위한 알고리즘을 제안한다. 그 동안 많은 컴퓨터 그래픽스 연구자들에 의해 구름 생성 및 렌더링 기법들이 제안되었으나 제안된 알고리즘들은 물리적인 정확성에 초점을 맞추어 알고리즘의 복잡성이 증가되는 단점이 있었으며, 이로 인해 구현이 어려운 단점이 있었다[1][2][3]. 또한 자연스러운 구름 생성을 위해서는 10개 이상의 파리미터를 조절해야 해야 했었다. 본 논문에서 제안한 방법은 물리적인 정확성 보다 컴퓨터 게임에서 사용될 수 있도록, 실시간 실행가능성에 초점을 맞추었으며 자연스러운 구름 생성을 위한 필요한 파라미터간의 수를 최소화 하여, 필요한 파라미터가 자동으로 조절되도록 하였다.

제안 한 방법은 총 3가지의 서로 다른 구름 형태인, 적운(cumulus), 층적운(stratocumulus), 층운(stratus)을 생성 할 수 있으며(그림 1), 하나의 파리미터를 이용하여 하늘 상의 나타내는 구름의 볼륨이 조절 되도록 하였다. 그림 1은 고도에 따른 3개의 다른 구름 타입에 대해 나타낸다. 또한 구름의 렌더링은 물리적인 볼륨 대기 산란(volumetric atmosphere scattering)기법을 단순화 하여 구름 내부의 쉐도우(shadow)가 표현되도록 하였다. 제안한 알고리즘은 GPU에서 실행 될 수 있도록 쉐이더(Shader)로 구현되었으며 윈도우 기반 PC상에서 실험을 통해 실시간으로 수행됨을 확인하였다.

Ⅱ. 관련연구

컴퓨터 그래픽스를 이용한 3차원 구름 생성 및 렌더링기법은 많은 연구자들로부터 연구되어 왔다[1][2][4][5]. 이 방법들은 크게 실시간으로 실행되는 온라인 기법과 품질을 높이고자 오프라인에서 동작하는 오프라인 기법으로 나뉜다. 이 중 온라인 기법은 게임 등에서 사용되기 위해 실시간성이 보장되어야 한다. 이와 같은 실시간 온라인 기법을 이용한 구름의 표현은 크게 3차원 공간을 정해진 격자로 나누어서 나타내는 계층적 공간 분할(Hierarchical Space Division)하는 방법 [6], 파티클 시스템을 이용하는 방법[7],구름 밀도를 텍스처 혹은 절차적(Procedural)한 방법으로 표현하여 나타내는 내포적 표현방법(Implicit) 등으로 구분된다[8]. 본 논문은 절차적 방법을 이용하여 구름을 생성 하고자 한다. 하지만, 기존의 방법들이 대부분 3D noise 텍스처를 이용해 구름의 비정형성 나타내는 반면, 본 논문에서는 직접 noise함수를 GPU상에 구현하여 사용한다. 3D noise 텍스처를 구름 생성에 이용할 경우, 노이즈 생성을 위한 텍스처를 외부 프로그램에 의해 생성해야 하며, 이 데이터는 하나의 이미지 형태로 고정되어 있으므로 변경하기 어려운 단점이 있다[9]. 또한, 구름생성에 필요한 3D 텍스처 데이터는 메모리를 많이 차지하는 단점이 있다. GPU상에 함수형태로 구현될 경우, 필요에 따라 변경이 용이한 장점이 있으며, 이는 다양한 형태의 구름을 생성하기 위한 다양한 실험을 원활히 할 수 있다. 본 연구에서는 fbm(Fractional Brownian Motion)기법을 이용하여 noise를 생성한다. 생성된 noise는 구름내부의 수증기의 밀도 정보로 이용된다. 즉, 노이즈 값이 작으며 밀도가 작아지므로, 구름에 포함되지 않게 되며, 밀도가 클수록 우리가 아는 흰색에 가까운 색상을 나타내게 된다.

구름의 형태가 정해지면, 구름에 해당하는 픽셀의 색상을 렌더링 과정을 통해 결정해야 한다. 구름과 같은 비 정형의 모습을 가진 개체는 하나의 메쉬(mesh)형태로 표현하기 어려우므로. 기존의 실시간 렌더링 기법을 이용할 수 없다. 이를 해결하기 위해 대부분의 알고리즘들이 레이 마칭(ray marching)기법을 이용한다[10]. 레이 마칭기법을 이용할 경우, 레이와 구름과의 충돌 시 나타나는 흡수(Absorption), 투과(Transmission), 산란(Scattering)과 같은 물리적 현상을 시뮬레이션 해야 한다. 많은 연구자들은 이와 같은 물리적 현상을 단순화

하여 사용하였다[11]. 본 논문에서는 단순화된 수학적 모델에 사용되는 파라미터들을 분석 후 이 파라미터의 연관성을 파악하여 파라미터의 숫자를 줄일 수 있도록 하여, 구름의 형태와 Coverage, 구름의 색상조정 파라미터정도만을 필요하도록 하였다.

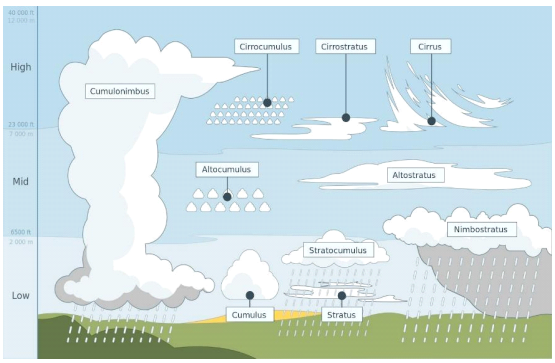


Fig. 1 Three different cloud types [12]

Ⅲ. 알고리즘

3.1. 구름 모델링

기본적인 구름 생성 알고리즘은 레이 마칭 기법을 이용한다[10]. 레이 마칭 기법은, 카메라 시점을 시작점으로(p_o), 중간에 놓여 있는 이미지 플레인 상의 각 픽셀을 타깃으로 레이를 공간상에 쏘아, 이 공간상에 signed distance function으로 정의된 오브젝트의 내부에 있는지를 확인한 후, 만일 레이 상의 한 점 p_i가 내부에 있다면 조명모델에 따라 색상을 결정하고, 그렇지 않다면 배경색을 설정한다[13]. 본 논문에서 목적으로 하는 구름의 경우, 비정형의 개체 이므로 signed distance function을 정의 하지 않고, 수증기 밀도 함수를 정의한다. 그림 2는 레이 마칭 기법을 나타낸다. 그림에서 빨간 선으로 정의된 위치는 레이 마칭을 통해 밀도를 계산하는 곳이다. 이위치 사이의 거리는 고정 위치($\triangle t$)를 정해거나, 속도 향상을 위해 가변 거리를 이용할 수 있다. 본 연구에서는 구현의 단순화를 위해 고정 위치를 이용한다.

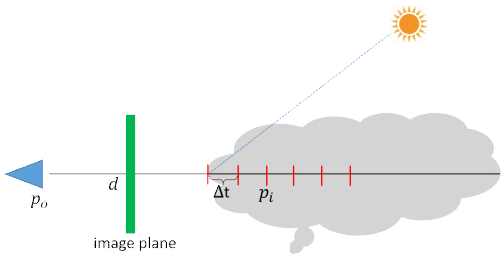


Fig. 2 Ray marching algorithm overview

하나의 레이는 아래 수식 (1)로 표현될 수 있다.

$$p_i = p_o + \triangle td \tag{1}$$

이 수식에서 p_i는 레이 상의 한 점을 나타내며 p_0는 레이의 시작점, 즉 카메라 위치를 나타내며, d는 레이의 방향, $\triangle t$는 다음 위치까지의 거리, i의 범위는 $0 \le i \le n$이다.

구름 모델링을 위해 가장 필요한 함수는 노이즈를 이용한 밀도 생성이다. 가장 대표적인 노이즈 함수는 perlin노이즈이나, 이 노이즈는 다양한 분야에 적용가능한 장점이 있는 반면, 하나의 perlin노이즈만으로는 구름 특유의 뭉쳐 있는 구름 생성을 생성하는 데는 단점이 있다. 본 연구에서는 실험을 통해 가장 정확한 구름 모습을 생성하는 fbm(Fractional brownian motion)을 이용하였다. fbm는 다수의 옥타브를 가진 노이즈들을 결합하여 사용한다. 하나의 옥타브는 하나의 주파수 영역을 나타내며, 다양한 주파수를 가진 노이즈를 결합하면, 세밀한 구름 밀도를 표현 할 수 있다. 본 연구에서는 Iñigo Quilez 가 절차적 지형 생성에서 이용한 fbm함수를 수정하여 구름 생성에 사용한다[8]. fbm을 이용해서 생성되는 노이즈 x_i 는 수식 (2)과 같다.

$$\begin{aligned} x_i &= \sum_{k=0}^{3} (x_k + a_k N(p_k)) \\ p_{k+1} &= p_k \bullet f \\ a_{k+1} &= a_k \bullet r \end{aligned} \tag{2}$$

수식 (2)에서의 상수 f 는 구름의 퍼진 정도를 나타내는 파라미터이며, r는 fbm의 옥타브 레벨에 따른 진폭감소 비율을 나타낸다. 본 연구에서는 f는 2.57정도, r은 0.5로 설정하였다. 함수 N()는 (0,1)사이의 렌덤 값을 반환 하는 함수이다. 이 함수는 3차원 위치 값 p를 입력받

는데, 이위치는 그림 2에서 빨간 선으로 나타낸 밀도 계산 위치에 해당된다. 본 연구에서는 실험으로 통해 5개의 옥타브로 이루어진 fbm이 가장 구름에 가까운 값을 반환해 줌을 확인하여 총 5개의 옥타브를 가진 노이즈를 결합하여 하나의 노이즈를 생성한다.

노이즈 함수를 통해 수증기 밀도 값(x_i)을 구한 후에는 주어진 구름의 형태에 따라 이 값을 변형해야 한다. 변형 함수 $remap(x,o_{\min},o_{\max},o'_{\min},o'_{\max})$는 주어진 값 x를 원래 범위 $[o_{\min},o_{\max}]$에서$[o'_{\min},o'_{\max}]$ 를 변경하는 함수라고 가정하면, 3가지 다른 구름 타입은 아래와 같이 이 함수를 이용하여 밀도를 변경한다[11].

- 적운(cumulus)= :
  $x_i = remap(x_i,0.01,0.3,0,1) \cdot remap(x_i,0.6,0.95,1,0)$
- 층적운(stratocumulus)
  $x_i = remap(x_i,0,0.25,0,1) \cdot remap(x_i,0.3,0.65,1,0)$
- 층운(stratus)
  $x_i = remap(x_i,0,0.1,0,1) \cdot remap(x_i,0.2,0.3,1,0)$

3.2. 구름 렌더링

구름 렌더링은 물리적인 대기 렌더링과 함께 처리한다. 본 연구에서는 평면으로 표현된 바다 위에 대기를 표현하고, 그리고 그 대기에 구름이 떠있는 모습을 목표로 한다. 레이 마칭을 통한 바다, 대기와 구름 렌더링의 대략적인 절차는 그림 3과 같다. 바다는 법선벡터가 (0,1,0)인 평면으로 표현될 수 있으므로, 레이의 방향 d와의 내적을 통해 해당 레이가 바다 부분에 충돌 되는지, 대기 부분에 충돌 되는지를 확인 할 수 있다. 만일 내적 값이 0보다 작다면 바다에 해당하므로, 바다표면의 색상을 계산하며, 그렇지 않고 0보다 크다면 대기에 해당하므로, 대기 분산산란(atmosphere scattering)을 통해 대기의 색상을 계산하고 이와 함께 수증기 밀도 모델을 이용하여 구름의 색상을 결정한다. 마지막 단계에서, 이 두 색상을 알파 값을 이용해 블랜딩 하여 최종 색상을 결정한다. 대기 렌더링은 태양의 위치에 따른 대기 산란 정도에 기반을 둔다. 지구의 대기 높이는 지상 100km정도이나 시뮬레이션은 대개 대류권과 성층권에서만 이루어지므로, 60km미만의 높이만을 고려한다. 대기 안에는 많은 입자들이 있으며, 이 입자들은 태양으로부터 나온 광자(photon)들과 충돌하면서 산란이 된다[14]. 산란의 정도는 입자의 크기와 속성에 따라 달라지나 일반적인 시뮬레이션에서는 모든 입자가 균일한 속성을 가진다고 가정하고, 대기의 높이에 따라 달라지도록 한다.

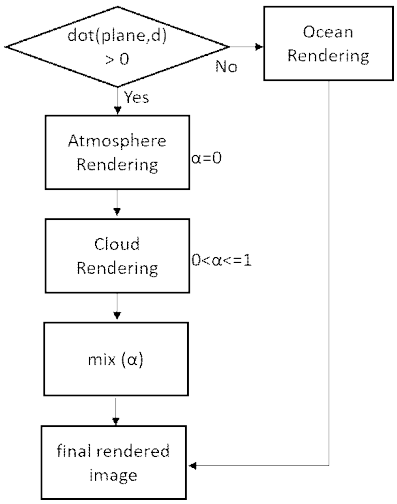


Fig. 3 Cloud Rendering Process

대기의 높이에 따라 일어나는 두 가지 산란은 Rayleigh 산란과 Mie 산란이다. Rayleigh 산란은 공기입자로 인한 산란을 의미하며, 하늘이 보통 파란색을 가지는 이유가 된다. Mie 산란은 낮은 고도의 도시에서 발생하며 스모그 등을 통해 나타나는 연무 및 수증기로 인한 산란을 의미한다. 본 연구에서는 이와 같은 산란을 포함한 대기 모델 중에 Preetham이 제안한 분석 모델을 이용하였다. 이 모델은 실시간 렌더링 가능한 장점이 있으며, 구현이 간단한 장점이 있다. 자세한 기술적 내용은 [12][15]을 참고하면 된다. 이 모델을 통해 생성된 대기 위해 구름을 생성하게 되는데, 구름으로 인해 광원으로 오는 빛의 감소되는 빛 및 투과되는 빛을 시뮬레이션 해주어야 한다. 투과되는 빛은 흔히 Beer 법칙에 의해 시뮬레이션 된다. 이 법칙을 단순화 시켜 투과되는 빛 T는 아래 수식 (3)과 같이 계산된다.

$$T_i(p_i,x_i,\triangle t) = e^{-m \cdot \triangle t \cdot x_i} \tag{3}$$

이 수식에서 T는 투과되는 빛의 크기, x_i는 밀도, p_i는 레이상의 한 점, m는 재질에 따른 상수를 의미하나, 보통은 흡수 정도를 나타내며, $\triangle t$ 수식 (1)에서의 step

크기를 의미한다.

수식 (3)은 기본적으로 구름을 투과하는 빛은 밀도가 증가함에 따라 약화됨을 의미한다.

Bees의 법칙에 의해 계산된 투과되는 빛의 양은 사실성 증대를 위해 Henyey-Greenstein(HG)의 Phase 함수를 이용해 산란(scattering)정도가 추가된다. Henyey-Greenstein은 기본적으로 Mie 산란을 단순화 시킨 모델로서, 구름을 생성하는 수증기 입자 또한 Mie산란의 영향을 받으므로, 구름이 모델을 추가하면 구름 가장자리의 밝은 부분(silver lighting)표현이 가능해 진다. HG 함수는 수식 (4)에 나타내 있다.

$$HG(\theta,g) = \frac{1}{4\pi}\frac{1-g^2}{[1+g^2-2g\cos(\theta)]^{3/2}} \quad (4)$$

이 수식에서 $\theta\,(-1 \le \theta \le 1)$는 레이의 방향과 태양광선의 방향과의 내적 값을 의미하여 $g(-1 \le g \le 1)$는 밝게 할 가장자리의 넓이를 의미하는 입력 값이다. 본 연구에서는 g 값은 0.99로 설정하였다.

수식(3)은 거리와 밀도에 따른 빛의 감소만을 나타냈으나 구름내부에서 발생하는 산란 중에는 빛의 시점방향으로 들어와서 빛의 세기를 강하게 하는 경우도 있다. 이와 같은 산란은 in-scatter산란이라고 하며, 수식 (3)의 역함수 형태를 취하게 된다. 이 in-scatter값 I는 수식 (5)와 같다.

$$I_i(p_i,x_i,\triangle t) = 1 - e^{-2\,\cdot\, m\,\cdot\, \triangle t\,\cdot\, x_i} \quad (5)$$

위에서 나타낸 Beer법칙을 통한 빛의 투과(수식(3), Henyey-Greenstein를 이용한 산란(수식(4)과 in-scatter 산란(수식5)은 수식 (6)에 의해 최종적으로 합쳐진다.

$$\begin{aligned} L_{i+1}(p_{i+1,}x_{i+1},\triangle t) = {} & L_i(p_i,x_i,\triangle t) \\ & 2\,T_i(x_i,\triangle t) \\ & I_i(p_i,x_i,\triangle t)HG(\theta,g) \end{aligned} \quad (6)$$

수식 (6)에 의해 계산된 빛의 크기 L은 최종적으로 구름의 색상 C 는 수식 (7)에 의해 변환 되며, 이때 사용되는 파라미터 b는 구름의 색상을 좀 더 어둡게 혹은 밝게 바꾸어 준다. 색상 값과 더불어 대기 렌더링결과와 블랜딩하기 위한 alpha값을 함께 계산한다.

$$\begin{aligned} C_{i+1} &= C_i + [e^h/b]L_i x_i \triangle t \\ a_{i+1} &= (1-T_i)(1-a_i) \end{aligned} \quad (7)$$

이 수식에서 $h = i/n$ 이며 자연스러운 구름 색상을 위해서는 b=1.9 값을 설정하였다. 색상값 C는 R,G,B 에 동일한 값으로 사용된다.

구름 움직임 표현을 위해서는 바람의 방향이 고려되었으며, 이를 위해 수식 (1)을 수식(8)과 같이 변경하여 사용한다.

$$p_i = p_o q + \triangle t d + w \quad (8)$$

위 수식에서 q는 offset, w는 바람의 방향벡터를 의미한다. 본 연구에서 offset는 0.01정도로 설정하였다.

Ⅳ. 실험 및 구현

3장에서 나타낸 알고리즘들은 실험을 통해 구현되었다. 시스템개발을 위한 하드웨어 환경은 Intel Xeon CPU E5-1607(32G memory)이며 그래픽스 카드는 Nvidia Geforce RTX 2070이 이용되었다. 소프트웨어 개발 환경은 Microsoft Windows 10상에서 Visual Studio를 이용하였으며, OpenGL 4.5 API 및 GLSL (OpenGL Shading Language)를 사용하여 구현되었다[16].

그림 4는 개발된 시스템을 통해 렌더링된 대기와 바다의 모습을 나타낸다. 대기의 높이에 따라 산란모델에 의해 점자 파란색으로 변함을 알 수 있다. 본 그림들은 편집없이 윈도우를 그대로 화면 캡처하여 나타낸 것이다.

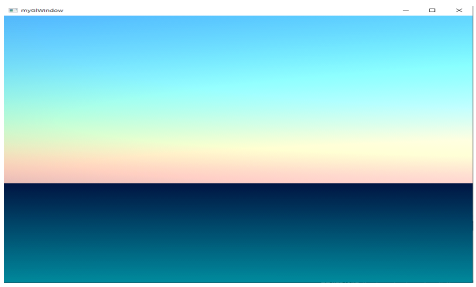
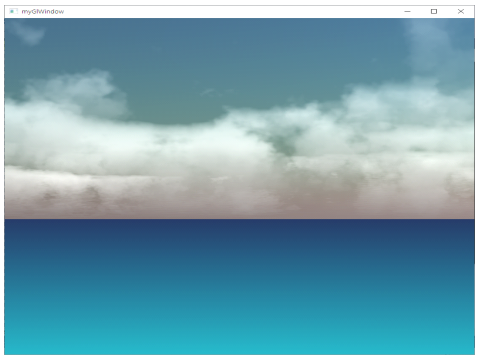
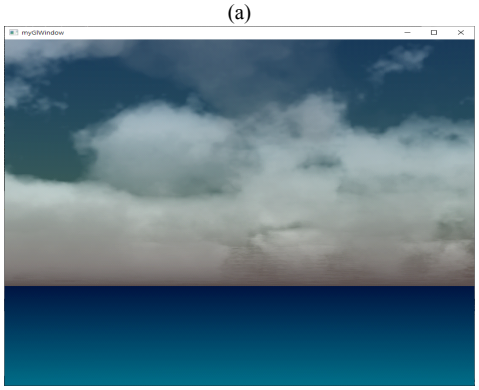


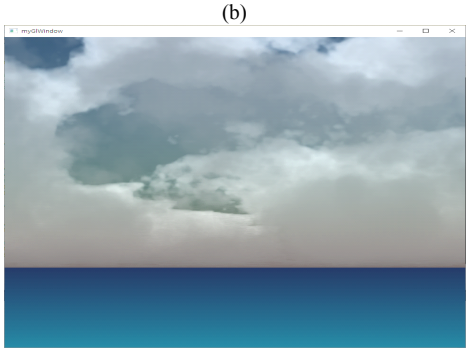
Fig. 4 Atmosphere rendering (OceanColor=(0, 0.73,0.95) ColorNearHorizon=0.0, 0.16, 0.51)



(a)



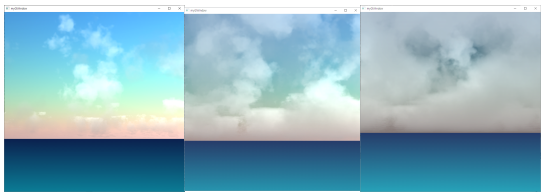
(b)



(C)

Fig. 5 Cloud Rendering: (a) cumulus (b) stratocumulus (c) stratus (f = 2.57 r = 0.5 #of octave = 5 g=0.99 b=.19)

그림 5는 세 가지 서로 다른 구름 형태인 적운, 층적운 및 층운을 나타낸 것이다. 그림 6는 coverage 값에 따른 구름의 하늘의 구름이 달라지는 모습을 나타낸 것이다. (구름 (b)는 태양방향에 있으므로 다른 그림과 대기 모습보다 밝음을 알 수 있다.)



(a) (b) (c)

Fig. 6 Cloud coverages : (a) 0.2 (b) 0.5 (c) 0.75

렌더링 속도는 구름의 형태와 상관없이 렌더링 되는 윈도우의 크기에 영향을 받는다. 표1에 해상도에 따른 렌더링 평균 속도를 확인하여 나타내었다. 참고로 본 실험에서 사용된 다양한 파라미터들은 많은 실험을 통해 가장 시각적으로 우수한 결과를 나타내는 값을 실험적으로 설정하였다. 그림 7은 본 논문에서 제안한 알고리즘과 [10]에서 제안한 렌더링 결과를 비교한 것으로 본 연구에서는 지형렌더링을 포함하지 않았다.

Table. 1 Performance

Resolution	Millisecond
1024 × 768	0.44
1280 × 1080	0.56
2048 × 1536	0.68

Fig. 7 Visual comparison of rendering result with [10] (Top : result of [10], Botton : Proposed algorithm)

Ⅴ. 결 론

본 연구는 게임등과 같은 실시간 어플리케이션을 위

해 실시간으로 구름을 생성하고 렌더링 하는 알고리즘을 제안하였다. 제안한 알고리즘은 3D텍스처를 이용하지 않고 순수한 절차적 노이즈 함수를 이용하여 구름을 생성하였으며, 대기렌더링과 구름렌더링을 알파 블랜딩에 의해 혼합하여 최종 렌더링을 수행 하였다. 노이즈 함수를 통해 얻은 결과를 수증기 밀도로 이용하였으며 레이 마칭기법을 통해 구름의 밀도에 따른 색상 값을 물리기반법칙에 기반하여 계산하였다. 추후 연구를 통해 대기의 coverage뿐 아니라, 파티클 시스템과 같이 각 수증기 입자의 수명을 조절하여 시간에 따른 구름의 생성 및 제거를 용이하게 하는 연구를 진행할 예정이다.

ACKNOWLEDGEMENT

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) grant funded by the Ministry of Education (2018R1D1A1B07048414)

References

[1] E. Cerezo, F. Pérez, X. Pueyo, F. J. Seron, and F. X. Sillion, “A survey on participating media rendering techniques,” *Visual computer*, vol. 21, no. 5, pp. 303-328, Jun. 2005.

[2] R. Hufnagel, and M. Held, “A survey of Cloud Lighting and Rendering Technique,” *Journal of WSCG*, vol. 20, no. 3, Jan. 2012.

[3] F. Häggström, “Real-time rendering of volumetric clouds,” M. S. dissertation, Umea University, 2018.

[4] A.Schneider, “The Real-time Volumetric Cloudscapes of Horizon: Zero Dawn,” SIGGRAPH Course note in Advance in Real-Time Rendering in Games, 2015.

[5] T. Akernin-Moller, E. Haines, and N. Hoffman, *Real-Time Rendering, 4th Edition*, CRC Press, 2018.

[6] C. Crassin, F. Neyret, S. Lefebvre, and E. Eisemann. “GigaVoxels: Ray-guided streaming for efficient and detailed voxel rendering,” in *Proc. ACM SIGGRAPH Symp.on Interactive 3D Graph. and Games (I3D)*, 2009.

[7] T. Nishita, Y. Dobashi, and E. Nakamae, “Display ofclouds taking into account multiple anisotropic scattering and skylight,” in *Proc. ACM SIGGRPH*, pp. 379-386, 1996.

[8] I. Quilez, Terrain Raymarching [Internet]. Available: http://www.iquilezles.org/www/articles/terrainmarching/terrainmarching.htm, 2002

[9] E. Vane, “Cloud Rendering using 3D Textures,” University of Waterloo, Technical Report, 2004.

[10] L. J. Tomczak, “GPU Ray marching of Distance Field,” M. S. dissertation, Technical University of Denmark, 2012.

[11] W. Engel, *GPU Pro 7:Advanced Rendering Technique*, CRC Press, 2016.

[12] A. J. Preeham, P. Shieley and B. Smits, “A Practical analytic model of daylight,” in *Proc. ACM SIGGRPH*, pp. 91-100, 1999.

[13] I. Quilez, 2D distance functions [Internet]. Available: https://iquilezles.org/www/articles/distfunctions2d/distfunctions2d.htm, 2018

[14] G. Bodare, and E. Sandberg, “Efficient and Dynamic Atmospheric Scattering,” M. S. dissertation, Chalmers University of Technology, 2016.

[15] E. Yosov, “High-Performance Rendering of Realistic Cumulus Clouds Using Pre-computed Lighting,” in *Proceeding of High Performance Graphics*, 2014.

[16] G. Seller, R. S. Wright, Jr, and N. Haemel, OpenGL *Super Bible : Comprehensive Tutorial and Reference, 7th Edition*, Addison- Wesley Professional, 2015.

성만규(Mankyu Sung)

1993년 2월 충남대학교 전산학과 학사
2005년 12월 (미) 위스콘신대학 컴퓨터사이언스 석사
2005년 12월 (미) 위스콘신대학 컴퓨터사이언스 박사
2006년 2월~2012년 2월 한국전자통신연구원(ETRI) 선임연구원
2012년 3월~현재 계명대학교 게임모바일공학전공 부교수
※관심분야 : 컴퓨터 그래픽스, 컴퓨터 애니메이션, 컴퓨터 게임, HCI