# PyCUDA 프레임워크에서 볼륨 렌더링을 구현하기 위한 새로운 커널 디자인

이수호[O], 김종현[*]
[O]강남대학교 소프트웨어응용학부,
[*]강남대학교 소프트웨어응용학부
e-mail: jonghyunkim@kangnam.ac.kr

# Novel Kernel Design for Implementing Volume Rendering in the PyCUDA Framework

SooHo Lee[O], Jong-Hyun Kim[*]
[O]School of Software Application, Kangnam University,
[*]School of Software Application, Kangnam University

● 요 약 ●

본 논문에서는 계산양이 큰 볼륨 렌더링을 구현할 수 있는 파이썬 기반의 CUDA(Computed Unified Device Architecture) 커널(Kernel) 디자인에 대해서 소개한다. 최근에 파이썬은 인공지능뿐만 아니라 서버, 보안, GUI, 데이터 시각화, 빅 데이터 처리 등 다양한 분야에서 활용이 되고 있기 때문에 인터페이스만을 위한 언어라는 색을 탈피한지 오래이다. 본 논문에서는 대용량 병렬처리 기법인 NVIDIA의 CUDA를 이용하여 파이썬 환경에서 커널을 디자인하고, 계산양이 큰 볼륨 렌더링이 빠르게 계산되는 결과를 보여준다. 결과적으로 C언어 기반의 CUDA뿐만 아니라, 상대적으로 개발이 효율적인 파이썬 환경에서도 GPU(Graphic Processing Unit)기반 애플리케이션 개발이 가능하다는 것을 볼륨 렌더링을 통해 보여준다.

**키워드:** 쿠다(Computed Unified Device Architecture), 볼륨 렌더링(Volume rendering), 커널(Kernel), 파이썬(Python), 데이터 시각화(Data visualization), 빅 데이터(Big data)

## I. Introduction

GPU 리소스를 활용하는 CUDA 병렬 프로그래밍은 컴퓨터그래픽스 학문을 벗어나, 수치해석, 인공지능, 빅 데이터 등 다양한 산업에 큰 영향을 미치고 있다[1,2]. 물리 기반 시뮬레이션 중 하나인 유체 시뮬레이션에서도 CUDA의 영향을 받아, 실시간으로 압력을 계산하거나 시각화가 가능한 접근법들이 꾸준히 제안되고 있다[3,4]. 하지만, 대부분 C언어 기반의 CUDA를 이용하여 개발을 하고 있으며, 큰 계산은 C언어가 효율적이라는 장점 때문에 다른 언어들은 가벼운 처리에만 사용이 되고 있다. 인공지능 라이브러리인 텐서플로우(TensorFlow)도 핵심 부분은 C/C++기반으로 개발이 되어있고, 인터페이스 언어로 파이썬을 통해 API에 접근하도록 설계되어있다.

파이썬은 방대한 라이브러리와 오픈소스의 효율성, 객체 지향 프로그래밍 등 많은 장점을 가지고 있지만, 계산양이 크지 않은 소프트웨어 개발로만 사용이 되고 있으며, 빅 데이터를 직접 처리하는 애플리케이션으로는 활용이 좀처럼 되지 않았다. 본 논문에서는 연구 및 소프트웨어 개발로 활용이 가능한 파이썬 기반 CUDA 병렬프로그래밍을 소개하고, 그 예로 볼륨 렌더링을 PyCUDA로 개발하기 위한 커널 디자인에 대해 설명한다.

```python
import pycuda.autoinit
import pycuda.driver as drv
import numpy

from pycuda.compiler import SourceModule
mod = SourceModule("""
__global__ void multiply_them(float *dest, float *a, float *b)
{
  const int i = threadIdx.x;
  dest[i] = a[i] * b[i];
}
""")

multiply_them = mod.get_function("multiply_them")

a = numpy.random.randn(400).astype(numpy.float32)
b = numpy.random.randn(400).astype(numpy.float32)

dest = numpy.zeros_like(a)
multiply_them(
        drv.Out(dest), drv.In(a), drv.In(b),
        block=(400,1,1), grid=(1,1))

print(dest-a*b)
```

Fig. 1. CUDA sample kernel with PyCUDA.

## II. The Proposed Scheme

본 논문에서 제안하는 프레임워크는 아래와 같이 수행된다 (Fig. 1 참조). 그림은 각각의 단계를 최소화되어 표현되었다. 연기의 밀도 데이터가 입력되면, 볼륨 렌더링을 연산하기에 적합하도록 값을 변환하여 PyCUDA로 제작된 커널함수로 값을 전달한다. 그리고 디바이스 환경 내에서 볼륨 렌더링의 알고리즘 각 단계별로 연산이 되어 최종 값을 결정한다.

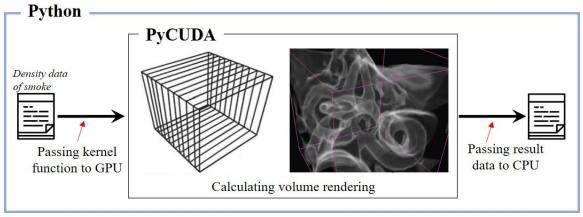


Fig. 2. Algorithm overview.

### 1. 직접 볼륨 렌더링

이번 장에서는 본 논문에서 볼륨 렌더링을 구현하기 위해 사용한 직접 볼륨 렌더링 기법에 대해서 간단하게 설명한다[5]. 직접 볼륨 렌더링은 불, 연기, 의료 영상 데이터와 같은 3차원 볼륨 데이터를 시각화하는데 사용되는 방법이다. 공간을 격자로 이산화하고, 각 복셀(Voxel)에 대해서 빛의 흡수, 방사에 대한 밀도 값을 할당하고, 레이에 따라서 시점까지 도달하는 각 복셀의 조명 값을 계산하는 방법이다. 여기서 밀도는 애플리케이션마다 거리, 밀도, 속도 등 다양한 값이 될 수 있다. 최근에 GPU를 이용한 직접 볼륨 렌더링이 있지만, 대부분 C/C++언어 기반으로만 알고리즘이 디자인되어있다.

### 2. 데이터 입력에 대한 전처리 과정

레이마칭(Raymarching)의 일종인 볼륨 데이터를 연산되기 위해선 카메라 위치로부터 시작한 레이가 밀도 복셀에 닿았을 때부터 시작되기 때문에[6], 우선 본 논문에서는 CT, MR, 시뮬레이션 등등에서 얻은 3D 볼륨 데이터 밀도 값을 파이썬 환경에서 전달한다. 입력 데이터를 커널 함수에 전달하기 위해 디바이스 내에 메모리를 할당하여 입력받을 공간을 만든다. 그리고 전달할 데이터를 GPU환경에서 사용 가능하도록 메모리 변환한다.

### 3. PyCUDA를 이용한 커널 디자인

전달받은 데이터는 직접 볼륨 렌더링의 알고리즘을 거쳐 결과 값을 도출한다. 이러한 단계들은 아래와 같은 알고리즘으로 진행된다. CUDA API가 C/ C++기반으로 동작하기 때문에 파이썬과 달리 PyCUDA 환경에서 커널함수는 C++언어로 작성해야한다 (Fig. 3 참조).

```
x = set of row volume
y = set of col volume
pos = (data.x, data.y)
rlt = 0; lrlt = 0
volume = 0

for all rayDistant in data do:
    if getVol(pos) is True Then
        add(pos) to rlt
    for all lightRayDistance in data do:
        if getVol(pos) is True Then
            add(pos) to lrlt
add(rlt, lrlt) to volume
```

Fig. 3. Pseudocode for kernel function.

볼륨 렌더링의 결과는 복셀의 투명도에 비례한 값과 조명특징이 고려된 복셀의 투명도에 비례한 값을 더하여 결과를 도출한다. 볼륨 렌더링은 모든 픽셀에서 레이마칭이 수행되고, 레이와 볼륨 데이터인 밀도값과 닿게되면 볼륨에 대한 근사가 시작된다.

레이는 설정된 거리까지의 밀도 값을 탐색하여 현재 탐색중인 좌표에 복셀의 밀도 값이 일정 수치이상 존재하면 값을 저장 한다. 그리고 다음 복셀을 레이의 방향에 따라 탐색한다. 모든 픽셀에 대한 레이로 복셀들을 탐색하였다면, 각각의 레이에서 탐색된 복셀들의 색상과 투명도를 누적하여 저장한다.

### 4. 결과 확인 및 비교

본 연구가 진행된 환경은 Ryzen2600 CPU, NVIDIA GTX 1060 3GB GPU, 8GB RAM에서 실험하였다.

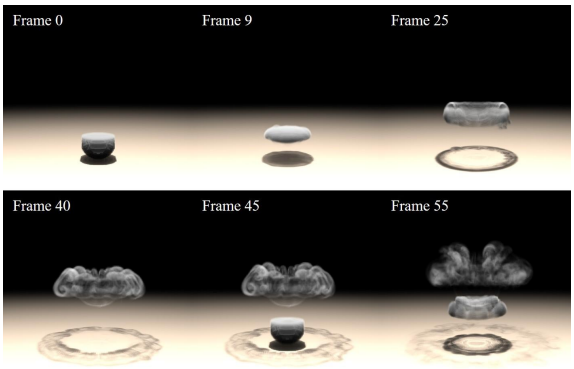


Fig. 4. Volume rendering of smoke with our method.

이렇게 연산된 값들은 3D 볼륨 값이기에 RGB값으로 변환할 수 있다. 그리고 배열로 입력받아 저장 후, 파이썬 환경으로 전달한다. 그 다음 계산된 RGB 배열을 이미지화하여 결과 이미지를 확인한다.

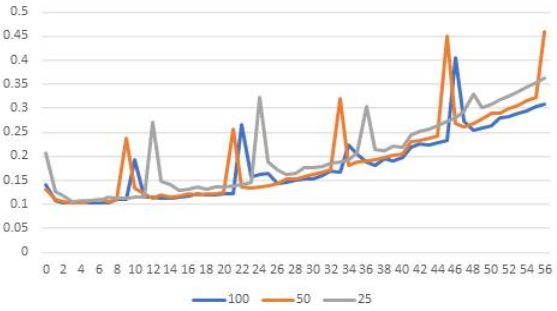


Fig. 5. block별 연산시간 비교

본 연구에서 가장 효율이 좋게 연산이 되었던 설정으로는 블록당 스레드의 개수가 8로 전체 64개, 그리드당 블록의 개수가 100개로 설정하여 연산하는 것이 전체적으로 연산시간이 적었다.

## III. Conclusions

본 논문에서는 파이썬 기반의 GPU 환경인 PyCUDA에서 계산양이 큰 볼륨렌더링 연산하는 프레임워크를 제안했다. 이를 통해서 파이썬에서도 GPU를 통해 계산양 큰 볼륨 렌더링을 빠른 시간 내에 처리될 수 있음을 보여줄 뿐만 아니라, 품질 또한 C/C++기반 GPU와 동일하게 계산되는 것을 확인하였다. 향후 연구로써 효율성 측면을 보안하여 연산시간 최소화, 정확한 볼륨 기울기 연산, 평면곡률 계산, 등등 전달 함수(Transfer function)를 적용하여 더 높은 정확성을 가진 모델을 연구할 계획이다.

## REFERENCES

[1] Januszewski, Michal, and Marcin Kostur. "Accelerating numerical solution of stochastic differential equations with CUDA." Computer Physics Communications, pp. 183-188, 2010.

[2] Ilievski, Andrej, Vladimir Zdraveski, and Marjan Gusev. "How CUDA Powers the Machine Learning Revolution." 2018 26th Telecommunications Forum (TELFOR). IEEE, 2018.

[3] Harris, Mark. "CUDA fluid simulation in NVIDIA Physx." Siggraph Asia, 2009.

[4] Amador, Gonçalo, and Abel Gomes. "A CUDA-based implementation of stable fluids in 3D with internal and moving boundaries." 2010 International Conference on Computational Science and Its Applications. IEEE, 2010.

[5] Wilson, Orion, Allen Van Gelder, and Jane Wilhelms. "Direct volume rendering via 3D textures." Ucsc-crl-94-19, Jack Baskin School of Eng., University of California at Santa Cruz, 1994.

[6] Tomczak, Lukasz Jaroslaw. "Gpu ray marching of distance fields." Technical University of Denmark, vol. 8, 2012.