한국컴퓨터그래픽스학회
Korea Computer Graphics Society Vol. 26, No. 1, P. 17~25

비정렬 격자에 대한 광선 투사를 위한 셀 사이 연결정보 추출 병렬처리 알고리즘

이지훈 김덕수*

한국기술교육대학교 컴퓨터공학부

{chosun8do, bluekds}@koreatech.ac.kr

Parallel Cell-Connectivity Information Extraction Algorithm for Ray-casting on Unstructured Grid Data

Jihun Lee Duksu Kim

KOREATECH (Korea University of Technology and Education)

요 약

본 논문은 비정렬 격자에 대한 광선투사 수행의 전처리 과정 중 하나인 셀 사이 연결정보 추출에 대한 멀티코어 CPU 기반 병렬처리 알고리즘을 제안한다. 본 연구는 기존의 직렬처리 알고리즘을 단순히 병렬화하였을 때 발생하는 동기화 문제를 확인하고, 이를 해결할 수 있는 3-단계 병렬처리 알고리즘을 제안한다. 제안하는 알고리즘은 각 단계 내에서의 스레드 간 동기화를 제거함으로서 병렬처리 효율을 높인다. 또한, 연결정보 추출 알고리즘의 핵심 연산인, 삼각형 중복 검사 과정의 메모리 접근에 대한 공간적 지역성을 높이고 캐시 활용 효율을 향상시킨다. 본 연구는 나아가, 스레드 마다 자체 메모리 풀을 사용하게 함으로서 병렬처리 효율을 더욱 높인다. 본 연구의 효용성을 확인하기 위해, 제안하는 알고리즘을 두 개의 옥타코어 CPU를 가지는 시스템에 구현하고 세 개의 비정렬 격자 데이터에 적용하였다. 그 결과, 제안하는 병렬처리 알고리즘은 스레드 수 증가에 따라 지속적으로 성능 향상을 보여주었다. 또한, 32개 스레드(물리코어 16개)를 사용하여 기존 직렬처리 알고리즘 대비 최대 82.9배 높은 성능을 보여주었다. 이는 제안하는 알고리즘의 높은 병렬처리 확장성 및 캐시 활용 효율 개선 효과를 증명하며, 대용량 비정렬 격자 처리에 대한 적합성을 보여주는 결과다.

Abstract

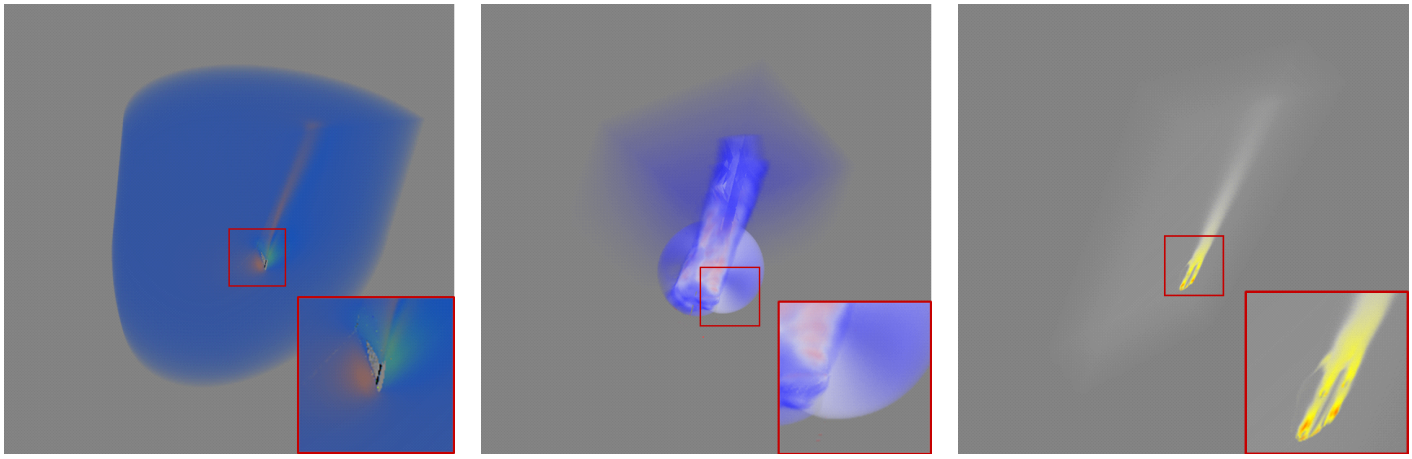
We present a novel multi-core CPU based parallel algorithm for the cell-connectivity information extraction algorithm, which is one of the preprocessing steps for volume rendering of unstructured grid data. We first check the synchronization issues when parallelizing the prior serial algorithm naively. Then, we propose a 3-step parallel algorithm that achieves high parallelization efficiency by removing synchronization in each step. Also, our 3-step algorithm improves the cache utilization efficiency by increasing the spatial locality for the duplicated triangle test process, which is the core operation of building cell-connectivity information. We further improve the efficiency of our parallel algorithm by employing a memory pool for each thread. To check the benefit of our approach, we implemented our method on a system consisting of two octa-core CPUs and measured the performance. As a result, our method shows continuous performance improvement as we add threads. Also, it achieves up to 82.9 times higher performance compared with the prior serial algorithm when we use thirty-two threads (sixteen physical cores). These results demonstrate the high parallelization efficiency and high cache utilization efficiency of our method. Also, it validates the suitability of our algorithm for large-scale unstructured data.

키워드: 연결정보, 병렬처리, 광선투사, 볼륨렌더링, 다중코어 CPU, 과학적 가시화

Keywords: connectivity information, parallel, ray-casting, volume rendering, multi-core CPU, scientific visualization

*corresponding author: Duk-Su Kim/Korea University of Technology and Education(bluekdct@gmail.com)

Received : 2019.12.13./ Review completed : 1st 2020.02.24. / Accepted : 2020.02.28.
DOI : 10.15701/kcgs.2020.26.1.17
ISSN : 1975-7883(Print)/2383-529X(Online)



(a) Dataset 1 (b) Dataset 2 (C) Dataset 3

Figure 1. Volume rendering images for the three benchmark datasets

1. 서론

직접 볼륨 렌더링(Direct Volume Rendering, DVR)은 대표적인 과학적 가시화 기법 중 하나로, M&S, 의료 등 다양한 분야에서 폭 넓게 활용된다[1]. 정형(uniform) 및 정렬(structured) 격자의 경우 데이터의 분포가 규칙적으로, 비교적 빠른 처리가 가능하며 다양한 가속 기법들이 연구되어 왔다[1]. 하지만 비정렬 격자의 경우 데이터의 분포가 불규칙적으로, 가시화 연산에 많은 시간이 소요된다[2,3]. 최근 많은 응용분야에서 비정렬 격자의 사용이 증가하고 있으며, 비정렬 격자에 대한 정확하면서도 빠른 볼륨 렌더링 알고리즘에 대한 요구가 커져가고 있다.

광선 투사(ray casting)는 이미지 평면의 각 픽셀(pixel)로 쏘아진 광선의 횡단(ray traversal)을 따라가며, 광선이 만나는 셀(cell)들의 값에 따라 픽셀의 색상을 계산하는 직접 볼륨 렌더링(이후, 볼륨 렌더링) 기법으로, 볼륨 렌더링 기법 중 가장 정확한 결과를 생성한다. 정형 및 정렬 격자는 셀 분포의 규칙을 활용하여 광선 투사를 비교적 간단히 구현할 수 있다. 하지만 비정렬 격자의 경우, 비정렬 격자를 구성하는 셀들의 기하학적 정보를 활용하여 광선이 지나는 셀들을 찾아야 한다. 이를 위해 셀 사이 연결정보(connectivity information)가 필요하며, 이는 셀을 구성하는 다면체들 사이의 면(face) 공유 정보로 표현된다[3]. 따라서 다면체들을 구성하는 면(예, 삼각형) 정보 및 그들 사이의 공유 정보를 추출하는 전처리 과정이 필요하다(3.2장).

비정렬 격자에 대한 광선 투사는 연산량이 많아 대용량 데이터에 적용하기 어렵다는 한계를 가지며, 이를 극복하기 위해 다양한 가속화 기법 및 병렬처리 알고리즘들이 연구되어 왔다[4-6]. 하지만 기존 연구들은 대부분 광선 횡단 자체의 성능을 개선하는 것에 초점을 맞추고 있으며, 연결정보 추출의 성능 향상에 대한 연구는 미미하다.

본 논문은 비정렬 격자에 대한 광선 투사 수행에 필요한, 셀 사이 연결정보를 추출에 대한 병렬처리 알고리즘을 제안한다(4장). 본 연구는 기존 직렬처리(serial) 알고리즘을 단순히 병렬화 했을 시 발생하는 종속성 문제를 해결할 수 있는 3-단계 병렬처리 알고리즘을 제안한다(4장). 제안하는 병렬처리 알고리즘의 각 단계에서는 스레드 간 동기화 없이 각 스레드들이 독립적으로 작업을 수행하며, 높은 병렬처리 효율을 얻을 수 있다. 또한, 연결정보 추출 과정의 핵심 연산인 중복 검사 과정에서 메모리 접근의 공간적 지역성(spatial locality)을 높인다. 나아가, 메모리 풀(memory pool) 기법을 적용하여, 동적 메모리 할당 요청에 따른 병렬처리 알고리즘의 성능 저하를 최소화한다.

본 연구는 제안하는 알고리즘의 효용성을 증명하기 위해, 서로 다른 세 종류의 대용량 비정렬 격자 데이터에 적용하고 16개의 CPU 물리코어를 가지는 시스템에서 성능을 측정하였다(5장). 실험 결과, 본 연구가 제안하는 병렬처리 알고리즘은 기존의 직렬 처리 알고리즘(VTK[7]의 구현) 대비, 32개 스레드(16개 물리코어)를 사용하여 최대 82.9배 높은 성능을 보여주었다. 또한, 스레드 수 증가에 따른 지속적인 성능 향상을 얻었다. 이러한 결과는 본 연구가 제안하는 병렬처리 알고리즘의 효용성 및 확장성을 증명한다.

2. 관련 연구

비정렬 격자의 볼륨 렌더링을 가속하기 위해 다양한 기법들이 연구되어 왔다. 셀 투영(cell projection)은 셀을 구성하는 면을 가시성 순서에 따라 이미지 평면(image plane)에 투영하는 볼륨렌더링 기법이다. 면을 이미지 평면에 투영하는 것은 GPU의 빠른 레스터화(rasterization) 기능을 활용하여 구현 할 수 있으며, 이를 기반으로 비정렬 격자 데이터에 대한 볼륨 렌더링을 가속하는 기법들이 제시되어 왔다[8-12]. 셀 투영을 통한 볼륨 렌더링의 품질은 같은 픽셀에

투영되는 면들 사이의 가시성 순서 판단 정확도와 밀접한 관계를 가진다. Callahan 등[11]은 GPU를 이용하여 가시성 순서 정렬을 가속하는 HAVS (Hardware-Assisted Visibility Sorting) 알고리즘을 제안하였다. HAVS는 CPU 사용 대비 높은 성능을 보여주지만, 제한적인 GPU 메모리 크기 및 GPU 스레드 사이의 동기화 문제에 따라 정확도 면에서는 한계를 보여주었다. 셀 투영 기법은 가시성 순서 판단 문제와 더불어 비 볼록 메시에 대해서는 정확한 결과를 보장하지 않는다는 한계를 가진다.

광선 투사(ray-casting)는 셀 투영 대비 많은 연산을 필요로 하지만, 정확한 볼륨 렌더링 결과를 보장한다. Garrity[2]는 셀 사이의 연결정보 정보를 사용하여 비정렬 격자에 대한 광선 횡단(traversal)을 구현하였다. Bunyk 등[3]은 광선 횡단을 위한 광선-면 교차(ray-face intersection) 검사 과정에서 중복 계산될 수 있는 정보들을 전처리 과정에서 미리 계산하여 저장해 두고, 이를 활용하여 빠르고 효율적으로, 그리고 정확하게 광선 투사를 수행하는 Bunyk 알고리즘을 제안하였다. Ribeiro 등[4]은 이웃한 광선들은 횡단 과정에서 비슷한 면들을 지난다는 특성인 광선 일관성(ray coherency)을 활용하여, Bunyk 알고리즘의 메모리 소모 문제를 해결하는 VF-Ray 알고리즘을 제안하였다. 그 결과 Bunyk 알고리즘 대비 메모리 소모량을 크게 줄이면서도, 대등한 성능을 유지 할 수 있었다. Kim[6]은 면 번호 기반의 해시(hash) 함수를 제안함으로서, 작은 크기의 버퍼에 대해 VF-Ray 대비 버퍼 적중률(hit ratio)을 크게 향상시키고 메모리 소모량을 더욱 줄였다.

위 연구들은 모두 셀 사이의 연결정보를 활용하여 광선 횡단을 수행한다. 본 논문은 이를 위한, 면 정보 및 연결정보를 추출하는 전처리 단계를 위한 병렬처리 알고리즘을 제안한다. 이는 셀 사이 연결정보를 사용하는 광선 투사 시스템에 공통적으로 적용될 수 있다.

Kim[6]은 멀티 코어 CPU를 활용하는 병렬처리 Bunyk 알고리즘을 제안하여, 8개의 CPU 코어를 이용해서 단일 코어 대비 최대 5배 높은 성능을 얻었다. Weiler 등[10]은 Garrity[2]의 광선 투사 방법을 GPU로 구현하였으며, Espinha와 Celes[13]는 Weiler 등의 방법의 정확도를 개선하였다. Bernadon 등[14]은 광선의 재진입 면을 기준으로 비 볼록 메시를 여러 층(layer) 구분하여 광선 투사를 수행하는 뎁스-필링(depth-peeling) 알고리즘을 제안하였다. 그들은 프래그먼트 쉐이더(fragment shader)를 통해 GPU 광선 투사를 구현하고, 각 층에 대해 순차적으로 광선 투사를 수행함으로서 비 볼록 메시에 대해 정확한 볼륨 렌더링 결과를 얻었다. Maximo 등[10]은 CUDA[15]를 이용해서 VF-Ray[4]를 GPU 알고리즘(VF-GPU)으로 확장하였다. 그 결과 CPU 사용 대비 3~5배 높은 성능을 얻었다. 하지만 다수의 가시 면들이 동시에 처리됨에 따라 비 볼록 메시에 대해서는 정확한 결과를 보장하지 않는다는 한계를 가진다. Gu 등[16]은 비 볼록 메시에 대해서도 정확한 결과를 보장하는 GPU 기반 광선 투사 알고리즘을 제안하였다. 그들은 최신 GPU의 메모리 구조에 적합한 알고리즘을 제안함으로서, 광선 횡단에 대해 CPU 기반 bunyk 알고리즘 대비 최대 36.5배 높은 성능을 보여주었다.

이처럼 비정렬 격자에 대한 광선 투사 성능 향상을 위해 멀티코어 CPU 또는 GPU 기반의 다양한 병렬처리 기술들이 활발히 연구되어 왔다. 하지만, 기존 연구들은 대부분 광선 횡단 단계의 병렬 처리에 집중하고 있으며, 전처리 단계에 대한 고민은 미미하다. 본 논문은 전처리 단계 중 연결정보 추출에 대한 병렬처리 알고리즘을 제안한다는 점에서, 기존 연구들과 차별성을 가진다.

과학적 가시화를 위해 널리 사용되는 오픈소스 라이브러리인 Visualization ToolKit(VTK)[7]은 비정렬 격자를 구성하는 셀들을 순차적으로 방문하며 셀을 구성하는 면 정보를 생성한다. 그리고 면 정보 생성 과정에서 해당 면이 기존에 생성된 면과 중복되는 면인지를 확인하는 방법으로, 모든 셀 사이의 연결정보를 생성한다. 하지만, 모든 면마다 중복 검사가 필요하여 연산 부하가 높다는 한계를 가진다.

본 연구는 VTK의 연결정보 추출 알고리즘을 멀티코어 CPU를 활용하는 병렬처리 알고리즘으로 확장한다(4장).

3. 광선 횡단 및 전처리 단계 개요

본 장에서는 연결정보 추출 성능 개선의 필요성에 대한 이해를 돕기 위해, 비정렬 격자에 대한 광선 횡단 알고리즘을 간단히 설명한다. 또한, 본 논문이 제안하는 병렬처리 알고리즘을 이해하기 위한 기반 지식으로 기존의 연결정보 추출 알고리즘의 개요를 제시한다.

3.1 비정렬 격자에 대한 광선 횡단

광선 횡단의 첫 단계는 각각의 픽셀로 쏘아지는 광선에 대한 진입 면(entry face)을 찾는 것이다. 이는 하나의 셀에만 속하는(외부와 만나는) 외곽 면(boundary face)을 이미지 평면에 투영(projection)하는 방법으로 찾는다. 비 볼록 다면체 메시의 경우, 한 픽셀로 투영된 외곽 면들을 깊이 값(depth 또는 Z-value)을 기준으로 정렬함으로써, 진입 순서를 결정할 수 있다. 그 중 가장 작은 깊이 값을 가지는 면이 해당 픽셀로 쏘아지는 광선의 최초 진입 면이 된다.

광선의 횡단은 진입 면에서부터 시작된다. 즉, 진입 면이 속한 셀이 광선이 처음 만나는 셀이며, 이후 광선 투사는 다음 과정을 통해 진행된다. 1) 현재 셀(다면체)을 구성하는 나머지 면들 중, 광선이 지나는 면을 찾는다. 2) 한 면은 두 개 셀(외곽 면인 경우, 한 개 셀)에 의해 공유되므로, 해당 면을 공유하는 셀 중 현재 셀이 아닌 셀이 광선이 다음에 지나는 셀이 된다. 3) 위 1~2 과정을 다음 지나는 면이 외곽 면이 아닐 때까지 반복한다. 4) 광선이 외곽 면을 만난

```
for each tetrahedron do
  for each triangle of the current tetrahedron do
      vIDs[3] = three vertex IDs of the current Tri.
      sort vIDs
      hash_key = vIDs[0] mod |hash_table|
      List = hash_table[hash_key]
      if ( Find the same vIDs in List ) then
        Create a cell connectivity information
      else
        List ← Push current triangle
      endif
  end for
end for
```

Table 1. Pseudo-code for the prior VTK's serial algorithm of extracting connectivity information

다는 것은 광선이 비정렬 격자 외부로 나왔다는 것을 뜻하며, 다음 진입 면으로 진입하여 위 과정을 반복한다.

3.2 VTK의 연결정보 추출 알고리즘

3.1장에서 본 것처럼, 비정렬 격자에 대한 광선 횡단을 위해서는 셀 사이의 연결정보가 필요하다. 비정렬 격자 데이터는 각 셀의 다면체를 구성하는 정점(vertex) 정보를 가지고 있다. 일반적으로 사면체(tetrahedral)를 많이 사용하며, 네 개의 정점 정보로 표현된다. 따라서 광선 투사 수행을 위해서는 사면체를 구성하는 정점들로부터, 사면체들 사이의 연결정보를 추출하는 과정이 필요하다.

정점 정보로 구성된 사면체 메시(mesh)의 연결정보 추출 방법에 대한 문헌(도서, 논문 등)을 찾기 어렵기 때문에, 본 논문에서는 VTK의 구현을 기준으로 기존 알고리즘을 설명한다[7].

VTK는 효율적인 광선 횡단 구현을 위해, 사면체 메시의 연결정보를 사면체들을 구성하는 삼각형 목록과 사면체 사이의 삼각형 공유 정보로 표현한다. 이를 위해서는 사면체의 정점 정보로부터 삼각형 목록을 생성하는 과정이 필요하다. 한 사면체를 구성하는 네 개의 삼각형은 네 정점을 세 개씩 묶는 방법으로 간단히 생성할 수 있다. 하지만, 인접한 두 사면체는 하나의 삼각형을 공유하므로, 이 과정에서 중복된 삼각형이 생성될 수 있다. VTK의 구현은 삼각형들의 정보를 중복 없이 저장하고, 각 사면체에는 사면체를 구성하는 삼각형의 번호를 저장한다. 또한, 각 삼각형에 자신이 속한 두 개의 사면체의 번호를 기록하는 방법으로 셀 사이 연결정보를 생성한다.

중복되는 삼각형 정보 없이 연결정보를 추출하는 VTK 알고리즘은 다음과 같다 (Table 1). 1) 사면체를 순차적으로 방문하면서, 2) 해당 사면체에 속하는 삼각형들을 하나씩 확인한다. 3) 해당 삼각형을 구성하는 정점들을 번호(vertex ID)를 기준으로 정렬한 후, 동일한 정점들로 구성된 삼각형이 있는지 확인한다. 4-1) 만약 해당 삼각형이 존재하면, 해당 삼각형의 번호를 사면체에 기록한다. 4-2) 해당 삼각형이 존재하지 않는 경우, 새로운 삼각형 정보를 생성하고 다면체에 기록한다. 5) 모든 사면체를 방문할 때까지, 1)~4)과정을 반복한다.

	Connectivity Info. build	Ray-casting/frame
Dataset 1	9.33 sec.	1.11 sec.
Dataset 2	1007.05 sec.	8.15 sec.
Dataset 3	2063.04 sec.	2.29 sec.

Table 2. Processing times by unstructured grid volume rendering algorithm of VTK for three different datasets. For the ray-casting step, it uses thirty-two CPU threads by employing on Kim's parallel algorithm[6].

이러한 과정을 통해 중복된 삼각형 정보 없이, 사면체 메시의 연결정보를 추출 할 수 있다. 하지만, 모든 삼각형마다 기존에 중복된 삼각형이 있는지 여부를 검사해야하며, 이는 많은 연산 시간을 소요하는 작업이다. 중복 검사 시 탐색 공간을 줄이기 위해 삼각형을 구성하는 가장 낮은 정점 번호를 키(Key)로 사용하는 해시 알고리즘을 사용한다. 하지만, 비정렬 격자 데이터의 크기(격자(다면체)의 수)가 커짐에 따라, 해시 테이블(hash table)의 크기가 증가하고 탐색에 걸리는 시간이 급격히 증가한다는 한계점을 가진다. 실제로 본 논문의 실험에 사용된 벤치마크 데이터들에서 광선 투사 시간(예, 1~8초/장) 대비, 전처리 시간(예, 9초~34분)이 매우 큰 것을 확인 할 수 있다(Table 2).

4. 병렬처리 연결정보 추출 알고리즘

앞서 살펴 본 것과 같이 사면체들의 연결정보를 추출하는 과정은 많은 연산을 요구한다. 본 연구는 최근 대부분의 연산 장치들이 여러 개의 CPU 코어를 가진다는 점에 착안하여, 병렬처리 연결정보 추출 알고리즘을 제안한다.

기존의 연결정보 추출 알고리즘(3.2장)을 병렬처리 알고리즘으로 확장하는 직관적인 접근법은 사면체들을 스레드(thread)들에게 균등분배하고, 각 스레드는 기존 방법을 사용하여 할당 받은 사면체들 사이의 연결정보를 추출하는 것이다. 하지만, 이 방법은 서로 다른 스레드로 분배된 사면체들 사이의 연결정보를 얻을 수 없다는 문제점을 가진다. 놓치는 정보 없이 정확한 연결정보를 추출하기 위해서는, 삼각형의 중복 확인(기존 알고리즘의 세 번째) 단계에서 다른 스레드들이 생성한 삼각형 목록들을 모두 확인해야 한다. 즉, 삼각형 중복 검사 시마다 다른 스레드들과 동기화 작업

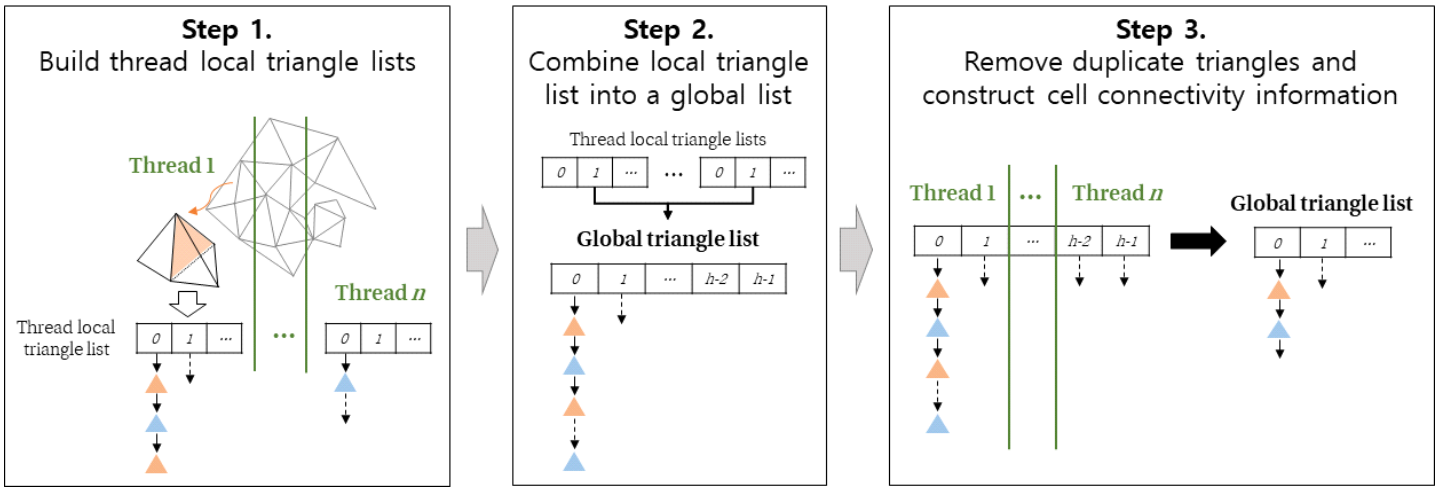


Figure 2. Overview of our parallel cell connectivity information extraction algorithm

이 필요하며, 이는 사면체 하나를 처리할 때마다 네 번씩 발생하는 매우 빈번한 작업이다. 동기화(synchronization)는 병렬처리의 효율을 떨어뜨리는 주요 병목 지점으로, 이처럼 잦은 동기화는 병렬처리가 직렬처리보다 오히려 낮은 성능을 보이는 원인이 되기도 한다. 실제로, 위의 간단한 병렬처리 알고리즘을 구현해본 결과, 8개 스레드를 사용하는 경우 본 연구에 사용된 벤치마크 데이터들에 대해, 직렬처리 대비 최대 6배(평균 3.2배) 낮은 성능을 보여주었다.

본 연구는 스레드들의 사이의 동기화를 제거하고 높은 병렬처리 효율을 얻기 위해, 3-단계 병렬처리 알고리즘을 제안한다(Figure 2). 첫 번째 단계는, 스레드 내 지역 삼각형 목록 생성단계다. 사면체들은 스레드들에게 균등하게 되며, 스레드들은 서로 동기화 없이 작업을 수행한다. 각 스레드는 할당 받은 사면체를 하나씩 방문하며 각 사면체를 구성하고 있는 네 개의 삼각형 정보를 생성한다. 이 과정에서 각 스레드는 자신만의 삼각형 목록을 만들며, 삼각형 목록은 기존 직렬 알고리즘과 같이 해시 테이블 형태로 생성된다. 하지만 기존 직렬 알고리즘과 달리 삼각형 중복 검사를 수행하지 않으며, 지역 삼각형 목록 안에 또는 다른 스레드가 생성한 삼각형 목록에 중복된 삼각형이 존재하게 된다. 즉, 사면체들 사이의 연결정보는 아직 생성되지 않은 상태로, 다음 단계들을 통해 연결정보가 만들어진다.

두 번째는, 각 스레드가 생성한 지역 삼각형 목록을 하나로 통합하는 단계다. 각 삼각형 목록은 해시 테이블로 형태로, 해시 키(hash key) 단위로 통합을 수행한다. 본 연구는 효율적인 해시 테이블 통합을 위해, 각 해시 키 마다 별도의 연결리스트(linked list)를 가지는 형태의 자료구조를 사용한다. 따라서 하나의 연결리스트를 다른 연결리스트에 연결하는 방법으로 해시 테이블을 간단히 통합할 수 있다(Figure 3의 녹색 줄). 삼각형 목록 통합은 하나의 스레드에 의해 직렬로 처리된다.

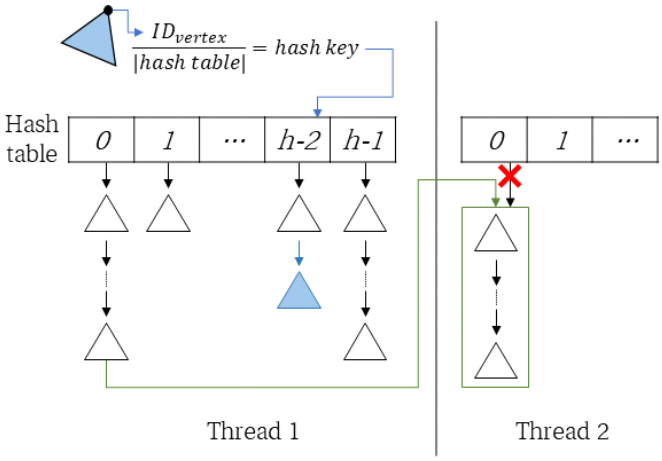


Figure 3. This figure shows the data structure for triangle lists (h : # of hash keys) and how they are combined into the global triangle list (green line).

마지막 단계는, 중복 삼각형 제거 및 사면체 사이의 연결정보 생성 단계다. 본 단계에서는 두 번째 단계에서 통합 및 생성된 삼각형 목록을 해시 키 단위로 스레드들에게 균등 분배한다. 스레드들은 자신에게 할당된 각 해시 키들에 대해 중복 삼각형 제거작업을 수행한다. 외부로 노출된 외곽 면을 제외한 모든 삼각형은 통합된 삼각형 목록에 두 개씩 존재하게 된다. 중복제거 작업은 삼각형을 순차적으로 방문하며 중복된 삼각형을 찾는 방법으로 수행된다. 탐색은 하나의 삼각형을 찾으면 종료되며, 다음 삼각형에 대한 중복 제거 작업으로 넘어간다. 중복된 삼각형을 제거 할 때는 해당 삼각형을 포함하는 사면체의 삼각형 구성 정보를 갱신하여, 사면체들 사이의 연결정보가 생성한다. 서로 다른 해시 키를 가지는 삼각형들 사이에는 중복 검사가 필요 없으며, 위 중복 제거 작업은 서로 독립적으로 수행가능하다. 따라서 모든 스레드들은 동기화 없이 주어진 중복 제거 작업

	The number of primitives				Required memory for triangle list(s)	
	Vertex	Tetra	Face	Boundary face	VTK (GB)	Ours (GB)
Dataset 1	647,073	3,773,943	7,583,488	71,614	0.41	0.82
Dataset 2	7,320,994	41,257,368	83,385,432	1,741,392	4.41	9.01
Dataset 3	7,659,517	41,939,085	85,051,683	2,347,026	4.47	9.19

Table 3. The information of benchmark unstructured grid datasets. Each column shows the number of vertex, tetrahedra, face, and boundary faces, and the required memory size for storing triangle lists by two algorithms.

을 독립적으로 처리할 수 있다.

메모리 관리: 삼각형 목록을 생성하는 과정에서 연결리스트를 사용하며, 새로운 삼각형 추가 시 동적 메모리 할당이 발생한다. 하지만, 동적 메모리 할당 요청들은 직렬화되어 처리되기 때문에 병렬처리 알고리즘의 성능 저하 요인이 된다. 본 연구는 이러한 문제를 해결하기 위해, 메모리 풀(memory pool) 기법을 활용한다[17]. 즉, 일정한 크기의 메모리 공간을 미리 잡아 둔 후, 요청이 들어오면 그중 일부를 배분해 주는 방법을 사용한다. 이는 추가적인 동적 할당 과정을 피할 수 있으며, 직렬화에 따른 성능 저하를 예방할 수 있다. 제안하는 병렬처리 알고리즘은 스레드 별 지역 삼각형 목록을 생성하며, 그 과정에서 중복된 삼각형을 생성한다. 하나의 면은 최대 두 개의 사면체에 의해 공유 될 수 있으며, 최악의 경우 직렬처리 대비 2배의 메모리가 필요하다는 것을 의미한다. 따라서 본 연구는

사면체의 수 × 4(=사면체당 삼각형 수) × 2

만큼의 삼각형을 저장할 만큼의 공간을 메모리 풀에 잡아 두고 사용한다. 또한, 메모리 풀의 공간은 각 스레드들에게 분배되면, 각 스레드는 동기화 없이 메모리 풀을 접근한다.

5. 결과 및 분석

본 연구는 제안하는 알고리즘을 두 개의 옥타코어 CPU(Intel Xeon Silver 4110, 2.10GHz) 및 384GB 메모리를 가진 윈도우즈(Windows) 시스템에서 VTK 8.2 프레임워크를 기반으로 구현하였다. 병렬처리 알고리즘은 OpenMP[18]를 기반으로 구현하였으며, 하이퍼스레딩(hyper-threading) 기능을 이용하여 최대 32개 스레드를 사용하여 성능을 측정하였다. 병렬처리 알고리즘은 동적 메모리 할당을 사용하는 버전(Ours w/o memory pool)과 메모리 풀을 사용(Ours)하는 두 가지 버전으로 작성하였다.

벤치마크: 본 연구는 서로 다른 크기를 가지는 세 가지 비정렬 격자 데이터에 대해 볼륨 렌더링을 수행하였으며(Figure 1), 연결정보 추출 성능을 테스트 하였다. 각각의 비정렬 격자 데이터에 대한 정보는 Table 3과 같다.

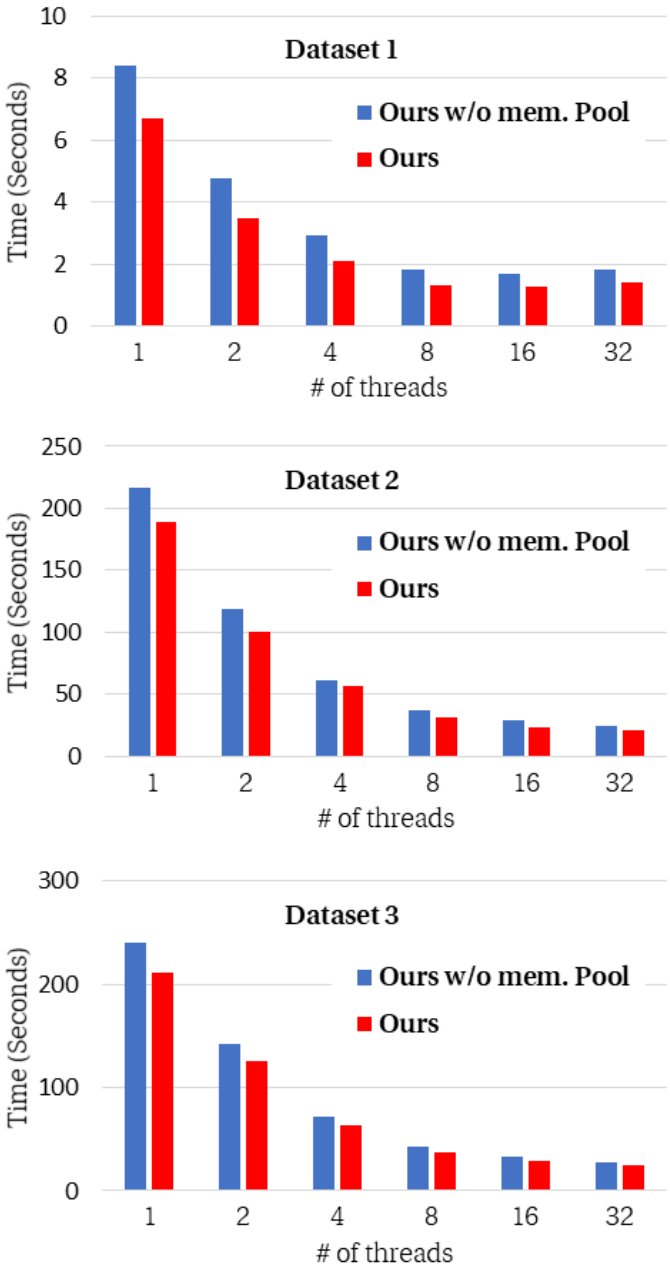


Figure 4. These graphs show the performance of two versions of our parallel algorithm depending on the number of computing threads.

Dataset 1	# of threads	1	2	4	8	16	32	VTK (serial)
	Ours w/o mem. pool	8.42	4.75	2.94	1.81	1.66	1.80	9.33
	Ours	6.71	3.46	2.11	1.33	1.27	1.41	
Dataset 2	# of threads	1	2	4	8	16	32	VTK (serial)
	Ours w/o mem. pool	216.04	118.43	61.33	36.85	28.63	24.74	1007.05
	Ours	188.62	100.11	56.18	31.19	23.50	21.29	
Dataset 3	# of threads	1	2	4	8	16	32	VTK (serial)
	Ours w/o mem. pool	239.75	142.44	72.21	43.09	32.57	28.18	2063.04
	Ours	210.97	125.86	64.08	37.59	28.21	24.89	

Table 4. Processing times (seconds) of three different cell connectivity extraction algorithms

Figure 4는 본 연구가 제안한 병렬처리 알고리즘의 스레드 수 증가에 따른 성능 변화를 보여준다. 스레드 수를 증가시키며 본 연구의 병렬처리 알고리즘(Ours)의 성능을 측정한 결과, 모든 벤치마크 데이터에서 스레드 수에 따라 성능이 높아지는 경향성을 확인할 수 있다. 상대적으로 크기가 작은 Dataset 1의 경우, 16개 스레드를 사용할 때 1개 스레드 사용 대비 5.27배 높은 성능을 보여주었으며, 32개 스레드(물리코어 16개) 사용 시에는 16개 스레드 사용대비 낮은 성능을 보여주었다. 이는 수행할 연산이 양이 많지 않아 병렬처리과정에서 발생하는 비용(메모리 풀 관리, 각 단계 사이의 동기화 등)이 상대적으로 크게 작용하기 때문이다. 하지만, 대용량 비정렬 격자 데이터인 Dataset 2와 3에서는 스레드 수 증가에 따라 지속적 성능 향상을 얻는다. Dataset 2의 경우, 단일 스레드 사용대비 4개, 8개, 16개, 32개(물리코어 16개)의 스레드 사용에 대해 각각 3.4배, 6.1배, 8.0배, 8.9배 높은 성능을 보여준다. Dataset 3에서는 4개, 8개, 16개, 32개(물리코어 16개)의 스레드 사용에 대해 각각 3.3배, 5.6배, 7.5배, 8.5배 높은 성능을 보여주었다(Table 4). 이러한 확장성(scalability)은, 본 연구가 제안한 3-단계 알고리즘이 각 단계 내에서 동기화 없이 스레드들이 독립적으로 작업을 수행하기 때문에 얻을 수 있는 결과다.

메모리 풀 활용 효율: Figure 4는 동적 메모리를 사용하는 경우와 스레드 마다 지역 메모리 풀을 사용하는 버전의 성능 비교를 보여준다. 메모리 풀을 사용하는 경우, 동적 메모리 할당 대비 최대 39% (평균 21%) 높은 성능을 보여준다. 또한 모든 벤치마크 데이터에서 스레드 수와 무관하게 성능 향상을 얻는 것을 확인할 수 있다. 이러한 결과는 병렬처리 과정에서 직렬화를 발생시키는 동적메모리 할당 과정을 제거함으로서 얻을 수 있는 결과다.

공간적 지역성 개선 효과: 흥미로운 결과 중 하나는, 본 연구의 알고리즘을 사용하는 경우, 1개의 스레드만 사용한 경우에도 기존 직렬처리 연결정보 추출 알고리즘(VTK 구현) 대비 높은 성능을 보여준다는 것이다. Dataset 1, 2, 3에 대해 기존 직렬 알고리즘 대비 각각 1.4배, 5.3배, 9.8배 높은 성능을 보여준다(Table 4). 이는 본 연구가 제안하는 3-단계 알고리즘이 중복 검사 시 기존의 방법 대비 높은 공간적 지역성(spatial locality)을 갖기 때문이다. 사면체로부터 삼각형 정보를 생성할 때 바로 중복검사를 수행하는 기존의 알고리즘의 경우, 생성되는 삼각형들 순서와 해시 키의 순서에 규칙성이 없다. 즉, 중복 검사를 위해 접근하는 해시 테이블의 영역이 불규칙하며, 이러한 불규칙한 메모리 접근은 높은 캐시(cache) 활용 효율(예, hit ratio)을 얻기 힘들다. 반면, 본 연구의 3-단계 알고리즘은 해시 키 단위로 스레드들에게 작업을 할당 하며, 각 스레드는 각 해시 키를 하나씩 처리한다. 즉, 같은 해시 키를 가지는 삼각형들 사이의 중복 검사를 수행하며, 해당 해시 키의 연결 리스트를 반복적으로 참조한다. 이처럼 메모리 접근에 있어 높은 공간적 지역성을 가지고 캐시 활용 효율을 높임으로써, 단일 스레드를 사용하는 경우에도 기존 알고리즘 대비 높은 성능 향상을 보여주었다.

그 결과, 본 연구가 제안하는 병렬처리 알고리즘은 두 개의 옥타코어 CPU를 사용하여, 기존의 VTK의 직렬 알고리즘 대비 Dataset 1, 2, 3에 대해 각각 최대 6.6배, 47.3배, 82.9배 높은 성능을 보여준다. 이러한 결과는 본 연구 결과의 우수성을 보여준다. 또한, 데이터의 크기가 크고 연결정보 추출의 시간이 큰 경우 더 높은 성능 향상을 보여준다. 이러한 결과는 제안하는 알고리즘의 대용량 비정렬 격자 데이터 처리에 대한 적합성을 증명한다.

6. 결론 및 향후 연구

본 논문은 비정렬 격자 데이터에 대한 광선투사 수행을 위해 필요한 셀 사이 연결정보 추출 성능 개선을 위한 병렬처리 알고리즘을 제안하였다. 본 연구는 기존의 직렬 처리 알고리즘을 단순히 병렬화했을 때 발생하는 처리 과정의 종속성(동기화)을 발견하고, 이를 해결 할 수 있는 3-단계 병렬처리 알고리즘을 제안하였다. 본 논문이 제안한 병렬처리 알고리즘은 각 단계에서 스레드들이 독립적으로 작업을 수행함으로써, 높은 병렬처리 효율을 얻는다는 장점을 가진다. 또한, 중복 검사 과정에서 메모리 접근의 공간적 지역성을 높임으로써 높은 캐시 활용 효율을 얻을 수 있다. 제안하는

병렬처리 알고리즘은 세 개의 벤치마크 데이터에 적용되었으며, VTK의 기존 직렬 처리 알고리즘 대비 32개의 스레드(16개 물리 코어)를 활용하여 최대 82.9배 높은 성능을 보여주었다. 이러한 결과는 본 연구가 제안하는 알고리즘의 효용성 및 대용량 데이터 처리에 있어 적합성을 증명하는 결과다.

본 연구는 멀티 코어 CPU를 사용하는 병렬처리 알고리즘을 제안하였다. 향후 연구에서는 GPU 기반 알고리즘으로 본 연구를 확장하고자 한다. 또한, 제안하는 병렬처리 알고리즘은 기존 직렬 알고리즘 대비 최대 2배의 메모리를 사용한다는 한계점을 가진다. 향후 연구에서는 성능은 유지하면 이러한 메모리 부하를 줄일 수 있는 알고리즘을 개발하고자 한다.

감사의 글

본 연구는 정부(과학기술정보통신부)의 재원으로 한국연구재단의 기초연구사업(2018R1C1B5045551)과 2020년도 한국기술교육대학교 교수 교육연구진흥과제 지원을 받아 수행된 연구입니다.

References

[1] K. Brodlie, J. Wood, "Recent advances in volume visualization", *Computer Graphics Forum*, Vol. 20, No. 2, pp. 125-148, 2001.

[2] M. Garrity. “Raytracing irregular volume data”, *Computer Graphics*, pp. 35-40, 1990.

[3] P. Bunyk, A. Kaufman, C. Silva. “Simple, fast, and robust ray-casting of irregular grids”, *Scientific Visualization Conference (dagstuhl ‘97)*, pp.30-30, 1999.

[4] Ribeiro S., Maximo A., Bentes C., Oliveira A., Farias R., “Memory-Aware And Efficient Ray-Casting Algorithm”, *Proceedings Of The Xx Brazilian Symposium On Computer Graphics And Imge Processing*, pp. 147-154, 2007

[5] André Maximo, Saulo Ribeiro, Cristiana Bentes, Antonio AF Oliveira, and Ricardo C Farias, “Memory efficient GPU-based ray casting for unstructured volume rendering”, *Volume Graphics*, pp. 155-162, 2008

[6] Duksu Kim, "Memory Efficient Parallel ray-casting Algorithm for Unstructured Grid Volume Rendering on Multi-core CPUs," *Journal of KIISE*, Vol. 43, No. 3, pp. 304~313, 2016

[7] Will Schroeder, Kenneth M. Martin, and William E. Lorensen, “The visualization toolkit (4th ed.): An object-oriented approach to 3D graphics”, *Prentice-Hall, Inc.*, 2006.

[8] Wylie B., Moreland K., Fisk L. A., Crossno P., “Tetrahedral projection using vertex shaders”, *Proceedings of the IEEE Symposium on Volume visualization and graphics*, pp. 7-12, 2002.

[9] Marroquim R., Maximo A., Farias R., Esperanca C., “GPU-Based Cell Projection for Interactive Volume Rendering”, *Proceedings of the XIX Brazilian Symposium on Computer Graphics and Image Processing*, pp. 147-154. 2006.

[10] Weiler M., Kraus M., Merz M., Ertlt, “Hardware-based view-independent cell projection”, *IEEE Transactions on Visualization and Computer Graphics*, Vol. 9, No. 2, pp. 163-175, 2003.

[11] Callahan S. P., Ikits M., Comba J. L., Silva C. T., “Hardware-assisted visibility sorting for unstructured volume rendering”, *IEEE Transactions on Visualization and Computer Graphics*, Vol. 11, No. 3, pp. 285-295, 2005.

[12] Farias, R., Mitchell, J. S., Silva, C. T., “ZSWEEP: An efficient and exact projection algorithm for unstructured volume rendering”, *Proceedings of the IEEE symposium on Volume visualization*, pp. 91-99, 2000.

[13] Espinha R., Celes W., “High-quality hardware-based ray-casting volume rendering using partial pre-integration”, *Proceedings of the XVIII Brazilian Symposium on Computer Graphics and Image Processing*, pp. 273, 2005.

[14] Bernardon F. F., Pagot C. A., Ao Luiz Dihl Comba J., Silva C. T., “GPU-based Tiled ray-casting using Depth Peeling”, *Journal of Graphics Tools*, Vol. 11, No. 3, pp. 23-29, 2006.

[15] NVIDIA, CUDA programming guide 9.2, 2018.

[16] Gu, Gibeom, and Duksu Kim. "Accurate and Memory-Efficient GPU Ray-Casting Algorithm for Volume Rendering Unstructured Grid Data", *EuroVis 2019 - Posters*, 2019

[17] Walls, Keith G. “Method for improving the performance of dynamic memory allocation by removing small memory fragments from the memory pool” *U.S. Patent* No. 5,675,790. 7 Oct. 1997.

[18] L. Dagum and R. Menon, “OpenMP: an industry standard API for shared-memory programming”, *IEEE Computational Sci. and Engineering*, Vol. 5, pp. 46-55, 1998

〈저자소개〉

이 지 훈

- 2020년 한국기술교육대학교 컴퓨터공학부 (공학사)
- 관심분야: 과학적 가시화, 병렬처리
- https://orcid.org/0000-0002-5450-4031

김 덕 수

- 2008년 성균관대학교 정보통신고학부 (공학사)
- 2014년 KAIST 전산학과 (공학박사)
- 2014년-2018년 한국과학기술정보연구원 선임연구원
- 2018년-현재 한국기술교육대학교 조교수
- 관심분야: 고성능컴퓨팅, 그래픽스/가시화, 인공지능 등
- https://orcid.org/0000-0002-9075-3983