(19) 대한민국특허청(KR)
(12) 공개특허공보(A)

(11) 공개번호 10-2021-0015509
(43) 공개일자 2021년02월10일

(51) 국제특허분류(Int. Cl.)
*G06T 15/06* (2011.01) *G06T 1/20* (2018.01)
*G06T 1/60* (2006.01) *G06T 15/00* (2006.01)
*G06T 15/80* (2011.01)
(52) CPC특허분류
*G06T 15/06* (2013.01)
*G06T 1/20* (2013.01)
(21) 출원번호 10-2019-0094491
(22) 출원일자 2019년08월02일
심사청구일자 2019년08월02일

(71) 출원인
**한국기술교육대학교 산학협력단**
충청남도 천안시 동남구 병천면 충절로 1600 (한국기술교육대학교내)
**한국과학기술정보연구원**
대전광역시 유성구 대학로 245 (어은동)
(72) 발명자
**김덕수**
대전광역시 유성구 온천북로33번길 36-26, 403호 (봉명동, 세종타운)
**구기범**
대전광역시 유성구 노은로 353, 302동 703호(하기동, 송림마을아파트3단지)
*(뒷면에 계속)*
(74) 대리인
**특허법인오암**

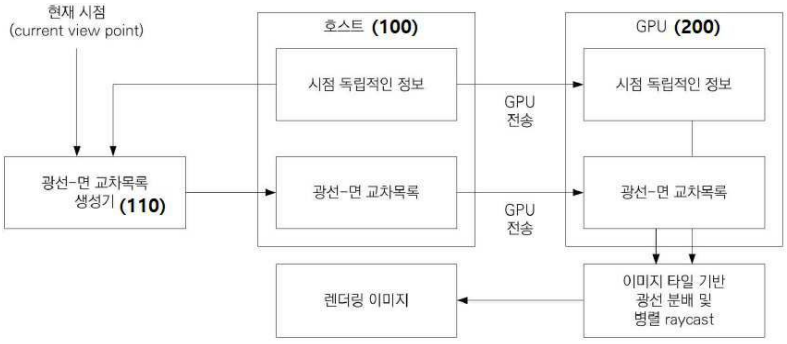
전체 청구항 수 : 총 7 항

(54) 발명의 명칭 **비정렬 격자 데이터의 렌더링을 위한 GPU 기반 레이 캐스팅 시스템 및 방법**

## (57) 요 약

본 발명의 일 실시예에 따른 GPU 기반 레이 캐스팅의 상기 GPU 기반 레이 캐스팅 시스템은, 볼륨 렌더링을 위한 레이 캐스팅 시스템에 있어서, CPU 측에서 교차 목록을 구성하되, GPU에서 상기 교차 목록을 구성하는 레이 캐스팅 알고리즘을 이용하여 어레이 기반 교차 목록을 생성하며, 상기 GPU가 상기 교차 목록을 생성하면 모든 광선에 대해 레이 캐스팅을 병렬로 수행하며, 이미지 타일 기반의 광선 분배 방법에 의해 스레드 블록에서 광선 일관성을 높이고 메모리 액세스 효율을 높일 수 있다.

*대 표 도* - 도1



(52) CPC특허분류
*G06T 1/60* (2013.01)
*G06T 15/005* (2013.01)
*G06T 15/80* (2013.01)
*G06T 2200/28* (2013.01)
(72) 발명자
**김민아**
대전광역시 유성구 어은로 57, 111동 1104호(어은동, 한빛아파트)
**이세훈**
대전광역시 유성구 가정로 65, 110동 601호(신성동, 대림두레아파트)
**이중연**
대전광역시 서구 둔산로 155, 104동 602호(둔산동, 크로바아파트)
**허영주**
대전광역시 유성구 노은로 353, 302동 703호(하기동, 송림마을아파트3단지)
**황규현**
대전광역시 서구 대덕대로175번길 40, 802호(둔산동, 두운힐스타운)

이 발명을 지원한 국가연구개발사업

| | |
|---|---|
| 과제고유번호 | 1711078787 |
| 부처명 | 과학기술정보통신부 |
| 과제관리(전문)기관명 | 국가과학기술연구회 |
| 연구사업명 | 민군융합기술연구사업 |
| 연구과제명 | 가상파랑예인수조를 위한 몰입형 가시화 기술 개발 |
| 기 여 율 | 1/1 |
| 과제수행기관명 | 한국과학기술정보연구원 |
| 연구기간 | 2018.08.01 ~ 2019.07.31 |

이 발명을 지원한 국가연구개발사업

| | |
|---|---|
| 과제고유번호 | K-19-L02-C05-S01 |
| 부처명 | 과학기술정보통신부 |
| 과제관리(전문)기관명 | 한국과학기술정보연구원 |
| 연구사업명 | 한국과학기술정보연구원 연구운영비지원(주요사업비) |
| 연구과제명 | 계산과학공학 플랫폼 기술 개발 |
| 기 여 율 | 1/1 |
| 과제수행기관명 | 한국과학기술정보연구원 |
| 연구기간 | 2019.01.01 ~ 2019.12.31 |

# 명 세 서

## 청구범위

### 청구항 1

볼륨 렌더링을 위한 레이 캐스팅 시스템에 있어서,

CPU 측에서 교차 목록을 구성하되, GPU에서 상기 교차 목록을 구성하는 레이 캐스팅 알고리즘을 이용하여 어레이 기반 교차 목록을 생성하며,

상기 GPU가 상기 교차 목록을 생성하면 모든 광선에 대해 레이 캐스팅을 병렬로 수행하는 것을 특징으로 하는 GPU 기반 레이 캐스팅 시스템.

### 청구항 2

제1항에 있어서,

상기 GPU의 스레드 블록에서 광선 간 일관성을 향상시키기 위해 이미지 타일을 기반으로 광선 그룹을 생성하며,

광선 그룹을 사용 가능한 스레드 블록에 분배하여 이미지 타일 기반 광선 분배를 수행하고 상기 스레드 블록의 스레드가 주어진 광선을 처리하여 레이 캐스팅을 병렬로 수행하는 것을 특징으로 하는 GPU 기반 레이 캐스팅 시스템.

### 청구항 3

제2항에 있어서,

상기 GPU는 상기 레이 캐스팅 알고리즘에서 주문형 VDFI 연산을 위한 스레드 로컬 버퍼를 포함하는 VDFI 버퍼를 더 포함하며,

상기 VDFI 버퍼는 워프 내의 모든 스레드가 버퍼에서 필요한 버퍼 히트를 발견할 때에만 VDFI 버퍼를 이용하며, 워프 내의 하나 이상의 스레드가 버퍼에서 필요한 정보를 찾을 수 없는 버퍼 미스의 경우, 워프 직렬화는 다른 스레드에 대한 버퍼 히트의 효과를 무효화하는 것을 특징으로 하는 GPU 기반 레이 캐스팅 시스템.

### 청구항 4

볼륨 렌더링을 위한 레이 캐스팅 방법에 있어서,

CPU 측에서 교차 목록을 구성하되, GPU에서 상기 교차 목록을 구성하는 레이 캐스팅 알고리즘을 이용하여 어레이 기반 교차 목록을 생성하는 단계;

상기 GPU의 스레드 블록에서 광선 간 일관성을 향상시키기 위해 이미지 타일을 기반으로 광선 그룹을 생성하는 단계;

광선 그룹을 사용 가능한 스레드 블록에 분배하여 이미지 타일 기반 광선 분배를 수행하는 단계; 및

상기 스레드 블록의 스레드가 주어진 광선을 처리하여 레이 캐스팅을 병렬로 수행하는 단계를 포함하는 GPU 기반 레이 캐스팅 방법.

### 청구항 5

제4항에 있어서,

상기 어레이 기반 교차 목록을 생성하는 단계는,

모든 경계면을 이미지 면에 순차적으로 투영하고, 경계면에 의해 투영된 모든 픽셀에 대해 픽셀의 ID를 $Array_p$에 추가하고 깊이가 있는 경계면 ID를 $Array_i$에 각각 추가하는 단계;

픽셀의 ID로 어레이를 정렬하고 각 픽셀의 교차 정보를 한 곳으로 모으는 단계;

크기가 픽셀 수와 같은 $Array_l$라는 새 어레이를 생성하는 단계; 및

각 픽셀에 대한 교차 목록을 깊이별로 정렬하여, 교차 목록인 $Array_i$와 $Array_l$를 얻는 단계를 더 포함하는 GPU 기반 레이 캐스팅 방법.

**청구항 6**

제4항에 있어서,

상기 GPU에 포함된 스레드 로컬 버퍼를 포함하는 VDFI 버퍼는 레이 캐스팅 알고리즘에서의 주문형 VDFI 연산을 수행하는 단계를 더 포함하는 GPU 기반 레이 캐스팅 방법.

**청구항 7**

제6항에 있어서,

상기 VDFI 버퍼는

워프 내의 모든 스레드가 버퍼에서 필요한 버퍼 히트를 발견할 때에만 VDFI 버퍼를 이용하며, 워프 내의 하나 이상의 스레드가 버퍼에서 필요한 정보를 찾을 수 없는 버퍼 미스의 경우, 워프 직렬화는 다른 스레드에 대한 버퍼 히트의 효과를 무효화하는 것을 특징으로 하는 GPU 기반 레이 캐스팅 방법.

## *발명의 설명*

### *기 술 분 야*

[0001] 본 발명은 컴퓨터 볼륨 렌더링을 위한 레이 캐스팅 시스템 및 방법에 관한 것으로 더욱 상세하게는 비정렬 격자 데이터의 렌더링을 위한 GPU 기반 레이 캐스팅 시스템 및 방법에 관한 것이다.

### *배 경 기 술*

[0002] 볼륨 렌더링(Volume Rendering)은 3차원 데이터를 2차원 영상으로 가시화 하는 기술로, 최근 컴퓨터 단층촬영(CT), 자기공명영상(MRI) 또는 3D 프린터 등의 기술이 발달하며 각광받는 분야이다. 특히 직접 볼륨 렌더링(DVR)은 가장 기본적인 시각화 방법 중 하나이며 의료 이미징, 과학 시뮬광선션 등 다양한 분야에서 널리 사용된다. 또한 레이 캐스팅은 일반성과 정확성 때문에 DVR에서 가장 널리 사용되는 알고리즘이다.

[0003] 또한 셀 투영은 비정렬 격자 데이터를 볼륨 렌더링하는데 널리 사용되는 방법 중 하나이다. 볼륨 렌더링을 수행하기 위해, 셀을 다면체 세트(예 : 정사면체)로 분해하고 다면체의 면을 이미지 평면에 투영한다. 투영 프로세스는 그래픽 하드웨어(즉, GPU)의 삼각(tri-angle) 래스터화 성능을 활용할 수 있으며, GPU 기반 셀 투영 알고리즘이 적극적으로 제안되고 있다.

[0004] 셀 투영 방법에서 렌더링 품질은 동일한 픽셀에 투영된 평면의 가시성 정렬 정확도에 따라 달라진다. 종래에는 Steven P Callahan 등이 GPU를 이용하여 가시성 정렬을 가속화하는 HAVS(Hardware-Assistant Visibility Sorting) 알고리즘을 제안하였으며, 정렬을 위해 CPU를 사용하는 것보다 높은 성능을 보였으나 정렬 정확도는 k 버퍼 크기에 의해 제한되며 동시에 텍스처 메모리 액세스에 대한 경쟁 조건 문제가 있다.

[0005] 셀 투영 방법과 달리 레이 캐스팅 알고리즘은 정확한 렌더링 결과를 보장하지만 많은 양의 연산이 필요하다. 종래에는 셀 연결 정보에 기반하여 비정렬 격자 데이터를 위한 레이 캐스팅 방법을 구현했다. 그러나 많은 메모리 공간이 필요했고 대규모 데이터 세트에 적용하기가 어려웠다.

[0006] 따라서, 전술한 문제를 해결하기 위하여 GPU에서 레이 캐스팅을 수행하고 GPU 스레드가 각 광선을 처리함과 아울러, 스레드 블록에서 광선 일관성을 높이고 메모리 액세스 효율을 높이기 위해 이미지 타일 기반의 광선 분배 방식의 비 정렬 격자 데이터를 볼륨 렌더링하기 위한 새로운 GPU 기반 레이 캐스팅 알고리즘에 대한 연구가 필요하게 되었다.

### *선행기술문헌*

### *특허문헌*

[0007] (특허문헌 0001) (비특허문헌 001) Steven P Callahan, Milan Ikits, Joao Luiz Dihl Comba, and Clau- dio T Silva, Hardware-assisted visibility sorting for unstructured volume rendering, IEEE Transactions on Visualization and Com- puter Graphics 11 (2005), no. 3

(특허문헌 0002) (비특허문헌 002) Paul Bunyk, Arie Kaufman, and Claudio T Silva, Simple, fast,

(특허문헌 0003) and robust ray casting of irregular grids, Scientific Visualization Conference, 1997, IEEE, 1997

(특허문헌 0004) (비특허문헌 003) Duksu Kim, Memory efficient parallel ray casting algorithm for unstructured grid volume rendering on multi-core CPUs, Journal of KIISE 43 (2016)

## *발명의 내용*

### *해결하려는 과제*

[0008] 본 발명의 목적은 레이 캐스팅 방법을 사용하며, 비볼록 메쉬를 지원하기 위해 픽셀별 교차 목록 개념을 사용하며, GPU의 목록에 대한 효율적인 메모리 액세스를 위해 새로운 표면 알고리즘을 사용하여 모든 면의 교차 목록을 어레이로 나타내며, 교차 목록을 사용하여 GPU에서 레이 캐스팅을 수행하고 GPU 스레드가 각 광선을 처리하며, 스레드 블록에서 광선 일관성을 높이고 메모리 액세스 효율을 높이기 위해 이미지 타일 기반의 광선 분배 방식의 GPU 기반 레이 캐스팅 시스템 및 방법을 제공하는 것이다.

### *과제의 해결 수단*

[0009] 본 발명의 일 실시예에 따른 GPU 기반 레이 캐스팅의 상기 GPU 기반 레이 캐스팅 시스템은, 볼륨 렌더링을 위한 레이 캐스팅 시스템에 있어서, CPU 측에서 교차 목록을 구성하되, GPU에서 상기 교차 목록을 구성하는 레이 캐스팅 알고리즘을 이용하여 어레이 기반 교차 목록을 생성하며, 상기 GPU가 상기 교차 목록을 생성하면 모든 광선에 대해 레이 캐스팅을 병렬로 수행하는 것을 특징으로 한다.

[0010] 상기에 있어서, 상기 GPU의 스레드 블록에서 광선 간 일관성을 향상시키기 위해 이미지 타일을 기반으로 광선 그룹을 생성하며, 광선 그룹을 사용 가능한 스레드 블록에 분배하여 이미지 타일 기반 광선 분배를 수행하고 상기 스레드 블록의 스레드가 주어진 광선을 처리하여 레이 캐스팅을 병렬로 수행하는 것을 특징으로 한다.

[0011] 상기에 있어서, 상기 GPU는 상기 레이 캐스팅 알고리즘에서 주문형 VDFI 연산을 위한 스레드 로컬 버퍼를 포함하는 VDFI 버퍼를 더 포함하며, 상기 VDFI 버퍼는 워프 내의 모든 스레드가 버퍼에서 필요한 버퍼 히트를 발견할 때에만 VDFI 버퍼를 이용하며, 워프 내의 하나 이상의 스레드가 버퍼에서 필요한 정보를 찾을 수 없는 버퍼 미스의 경우, 워프 직렬화는 다른 스레드에 대한 버퍼 히트의 효과를 무효화하는 것을 특징으로 한다.

[0012] 본 발명의 일 실시예에 따른 GPU 기반 레이 캐스팅 방법은 볼륨 렌더링을 위한 레이 캐스팅 방법에 있어서, CPU 측에서 교차 목록을 구성하되, GPU에서 상기 교차 목록을 구성하는 레이 캐스팅 알고리즘을 이용하여 어레이 기반 교차 목록을 생성하는 단계; 상기 GPU의 스레드 블록에서 광선 간 일관성을 향상시키기 위해 이미지 타일을 기반으로 광선 그룹을 생성하는 단계; 광선 그룹을 사용 가능한 스레드 블록에 분배하여 이미지 타일 기반 광선 분배를 수행하는 단계; 상기 스레드 블록의 스레드가 주어진 광선을 처리하여 레이 캐스팅을 병렬로 수행하는 단계;를 포함한다.

[0013] 상기에 있어서, 상기 어레이 기반 교차 목록을 생성하는 단계는, 모든 경계면을 이미지 면에 순차적으로 투영하고, 경계면에 의해 투영된 모든 픽셀에 대해 픽셀의 ID를 $Array_p$에 추가하고 깊이가 있는 경계면 ID를 $Array_i$에 각각 추가하는 단계; 픽셀의 ID로 어레이를 정렬하고 각 픽셀의 교차 정보를 한 곳으로 모으는 단계; 크기가 픽셀 수와 같은 $Array_l$라는 새 어레이를 생성하는 단계; 각 픽셀에 대한 교차 목록을 깊이별로 정렬하여, 교차 목록인 $Array_i$와 $Array_l$를 얻는 단계; 를 더 포함한다.

[0014] 상기에 있어서, 상기 GPU에 포함된 스레드 로컬 버퍼를 포함하는 VDFI 버퍼는 레이 캐스팅 알고리즘에서의 주문형 VDFI 연산을 수행하는 단계; 를 더 포함한다.

[0015] 상기에 있어서, 상기 VDFI 버퍼는 워프 내의 모든 스레드가 버퍼에서 필요한 버퍼 히트를 발견할 때에만 VDFI

버퍼를 이용하며, 워프 내의 하나 이상의 스레드가 버퍼에서 필요한 정보를 찾을 수 없는 버퍼 미스의 경우, 워프 직렬화는 다른 스레드에 대한 버퍼 히트의 효과를 무효화하는 것을 특징으로 한다.

## 발명의 효과

[0016] 본 발명의 GPU 기반 레이 캐스팅 시스템 및 방법은 교차 목록을 사용하여 GPU에서 레이 캐스팅을 수행하고 GPU 스레드가 각 광선을 처리하며, 이미지 타일 기반의 광선 분배 방법에 의해 스레드 블록에서 광선 일관성을 높이고 메모리 액세스 효율을 높일 수 있는 장점이 있다.

[0017] 또한, 레이 캐스팅이 정확한 렌더링 결과를 보장하며, 픽셀별 교차 목록에 의해 비볼록 메쉬를 지원하며, 중복 연산을 허용하는 경우에도 더 나은 성능을 달성하는 주문형 연산 전략을 사용할 수 있는 장점이 있다.

## 도면의 간단한 설명

[0018] 도 1은 본 발명의 일 실시예에 따른 GPU 기반 레이 캐스팅 시스템의 개념을 전체적으로 설명하기 위한 블록도이다.

도 2은 본 발명의 일 실시예에 따른 GPU 기반 레이 캐스팅 방법의 어레이 기반 교차 목록 생성 알고리즘에 대한 순서도이다.

도 3은 광선 그룹에서 광선 분배 방법을 예시적으로 보여주는 도면이다.

도 4는 크기와 특성이 서로 다른 세 개의 비정렬 격자 데이터 세트에 본 발명의 방법을 적용한 결과를 보여주는 도면이다.

도 5는 본 발명과 종래 기술을 비교하여 도 4의 데이터 세트 3의 렌더링 결과를 예시적으로 보여주는 도면이다.

도 6은 본 발명과 종래 기술을 비교하여 레이 캐스팅의 개선 성능을 보여주는 그래프이다.

도 7는 본 발명과 종래 기술을 비교하여 도 4의 데이터 세트 1에 대하여 레이 캐스팅 시간을 보여주는 그래프이다.

도 8은 볼륨 데이터 경계면의 영상평면 투영을 설명하기 위한 도면이다.

도 9는 영상 평면에 투영된 경계면을 설명하기 위한 도면이다.

도 10은 경계면과 교차하는 픽셀의 집합을 설명하기 위한 도면이다.

도 11은 도 2의 어레이 기반 교차 목록 생성 알고리즘에서 새로운 픽셀/경계면을 등록하기 전을 설명하기 위한 도면이다.

도 12는 도 2의 어레이 기반 교차 목록 생성 알고리즘에서 새로운 픽셀/경계면을 등록한 후를 설명하기 위한 도면이다.

도 13은 픽셀 ID를 이용해서 어레이를 정렬한 후를 설명하기 위한 도면이다.

도 14는 어레이 계산결과를 설명하기 위한 도면이다.

도 15는 어레이 정렬결과를 설명하기 위한 도면이다.

## 발명을 실시하기 위한 구체적인 내용

[0019] 이하에서는 도면을 참조하여 본 발명의 구체적인 실시예를 상세하게 설명한다. 다만, 본 발명의 사상은 제시되는 실시예에 제한되지 아니하고, 본 발명의 사상을 이해하는 당업자는 동일한 사상의 범위 내에서 다른 구성요소를 추가, 변경, 삭제 등을 통하여, 퇴보적인 다른 발명이나 본 발명 사상의 범위 내에 포함되는 다른 실시예를 용이하게 제안할 수 있을 것이나, 이 또한 본원 발명 사상 범위 내에 포함된다고 할 것이다. 또한, 각 실시예의 도면에 나타나는 동일한 사상의 범위 내의 기능이 동일한 구성요소는 동일한 참조부호를 사용하여 설명한다.

[0020] 본 발명의 일 실시예에 따른 GPU 기반 레이 캐스팅 시스템은 기본적으로 CPU(100)측과 GPU(200)측의 연동에 의해 기능을 수행하며, 종래의 Bunyk 알고리즘의 픽셀별 교차 목록 개념을 사용하여 비볼록 메시의 정확한 렌더링 결과를 보장한다.

[0021] GPU 기반 레이 캐스팅 시스템은 도 1을 참조하여 구체적으로 설명하면, CPU(100) 측에서 교차 목록을 구성한다. 이때, CPU(100)는 현재 시점에서 교차 목록을 생성하는 광선-면 교차목록 생성기(110)로부터 광선-면 교차목록을 제공받거나 광선-면 교차목록 생성기를 내부적으로 구비하도록 더 포함할 수도 있다.

[0022] 또한, GPU(200)에서는 CPU측과 연동하여 장치 메모리로의 효율적인 데이터 전송 및 효율적인 메모리 접근을 위해 교차 목록을 전송받아 교차 목록을 구성하는 후술할 레이 캐스팅 알고리즘을 이용하여 어레이 기반 교차 목록(광선-면 교차 목록)을 생성한다.

[0023] GPU(200)가 뷰(시점) 독립적인 정보(예를 들면, 교차 목록)을 얻으면 모든 광선에 대해 레이 캐스팅을 병렬로 수행한다.

[0024] 보다 자세하게는, GPU(200)의 스레드 블록에서 광선 간 일관성을 향상시키기 위해 이미지 타일을 기반으로 광선 그룹을 생성하고 광선 그룹을 사용 가능한 스레드 블록에 분배하는 이미지 타일 기반 광선 분배를 수행할 수 있으며, 스레드 블록의 스레드가 주어진 광선을 처리하는 레이 캐스팅을 병렬로 수행할 수 있으며, 결과적으로 렌더링 이미지가 생성된다.

[0025] 이때, 광선에 대한 레이 캐스팅 프로세스는 Bunyk 알고리즘 일부를 따르지만, 모든 면에 대해 VDFI를 유지 관리하지는 않는다. 대신 최근 GPU(200)에 대한 적절한 접근 방법으로써, 주문형 VDFI 연산 전략을 사용한다.

[0026] 본 발명이 어레이 기반 교차 목록을 구성하는 것에 대하여 보다 자세히 설명하면, 다음과 같다.

[0027] 각 광선이 교차하는 경계면을 식별하기 위해 모든 경계면을 이미지면에 투영한다. 픽셀 당 교차 목록을 작성하는 단순하고 직접적인 방법은 각 픽셀에 대해 연결된 목록를 사용한다.

[0028] 주어진 경계면의 경우, 삽입된 정렬을 사용하여 깊이별로 목록를 정렬하면서 평면이 투영된 픽셀 목록에 평면 ID를 추가한다. 그러나, 링크된 목록는 불규칙한 메모리 액세스 패턴을 이끌어 내기 때문에, GPU(200)에 바람직한 데이터 구조가 아니다. 또한 링크된 목록를 장치 메모리로 전송하려면 메모리를 할당하고 각 요소에 대해 데이터를 하나씩 복사하는 등의 복잡한 프로세스가 필요하다. 따라서 연결된 목록를 어레이 집합으로 변환하는 것이 바람직하다. 그러나 변환 과정은 모든 노드를 하나씩 방문해야 하므로 복잡한 과정을 거친다.

[0029] 픽셀별 교차 목록을 효율적으로 작성하기 위해 후술할 어레이 기반 교차 목록 생성 알고리즘을 이용한다.

[0030] 본 발명의 GPU(200) 기반 레이 캐스팅 방법은 CPU(100) 측에서 교차 목록을 구성하되, GPU(200)에서 상기 교차 목록을 구성하는 레이 캐스팅 알고리즘을 이용하여 어레이 기반 교차 목록을 생성하는 단계와, 상기 GPU(200)가 상기 교차 목록을 생성하면 레이 캐스팅을 병렬로 수행하는 단계와, 상기 GPU(200)의 스레드 블록에서 광선 간 일관성을 향상시키기 위해 이미지 타일을 기반으로 광선 그룹을 생성하는 단계와, 광선 그룹을 사용 가능한 스레드 블록에 분배하고 블록의 스레드가 주어진 광선을 처리하는 단계를 포함한다.

[0031] 도 2는 본 발명의 일 실시예에 따른 GPU 기반 레이 캐스팅 방법의 어레이의 교차 목록 생성 알고리즘에 대한 순서도이다.

[0032] 상술한 GPU(200) 기반 레이 캐스팅 방법에서 어레이 기반 교차 목록 생성하는 단계는 도 8 내지 도 15를 참조하여 설명하면, 다음과 같은 단계를 더 포함한다.

[0033] 또한 어레이 기반 교차 목록 생성 알고리즘에서는 픽셀의 ID (예 : X와 Y)를 포함하는 $Array_p$과 요소가 픽셀의 표면 ID와 깊이로 구성된 $Array_i$의 두 어레이를 사용한다.

[0034] 먼저, 첫 번째 단계에서는 볼륨 데이터를 구성하는 각각의 셀에서 모든 외부로 노출된 면(경계면, 도 9 참조)을 이미지 면에 순차적으로 투영한다(S200). 경계면을 투영한 후 경계면에 의해 투영된 모든 픽셀에 대해 픽셀의 ID를 $Array_p$에 추가하고 깊이가 있는 경계면 ID를 $Array_i$에 각각 추가한다.

[0035] 보다 상세히 설명하면, 영상평면에 투사한 경계면과 교차하는 픽셀의 ID는 $Array_p$, 경계면의 ID는 $Array_i$ 에 각각 저장한다. 즉, 도 11 및 도 12를 참조하면, 이미 저장되어 있던 $Array_p$ 와 $Array_i$의 내용은 그대로 유지한 상태에서 새로운 값을 추가(append)하는 방식으로 저장한다.

[0036] 여기서, $Array_p$는 영상평면에 투영된 경계면과 교차하는 픽셀의 ID 목록(중복 가능)을 포함하며, $Array_i$는 $Array_p$의 픽셀과 교차하는 경계면 ID 및 해당 픽셀로부터의 거리를 포함한다.

[0037] 이 첫 번째 단계 후에 두 어레이($Array_p$, $Array_i$)는 모든 픽셀에 대한 모든 교차 정보를 수집하지만 아직 정렬되지는 않는다.

[0038] 두 번째 단계에서는 픽셀의 ID로 어레이를 정렬하고 각 픽셀의 교차 정보를 한 곳으로 모은다(S202).

[0039] 바람직하게는, 도 13에 도시된 바와 같이 $Array_p$의 픽셀 ID로 $Array_p$, $Array_i$를 동시에 정렬(sort by key)한다.

[0040] 그런 다음 크기가 픽셀 수와 같은 $Array_l$이라는 새 어레이를 생성한다(S204).

[0041] 또한 새로 생성된 $Array_l$에는 $Array_l$을 구성하는 특정 픽셀 ID에 대한 정보의 시작점과 개수를 저장한다. 도 14를 참조하여 $Array_l$(P1, P2, P3, P6)을 구성하는 (x,y)값을 살펴보면, x는 $Array_p$와 $Array_i$내에서 특정 픽셀의 정보가 처음 시작되는 위치, y는 해당 픽셀 정보의 개수를 의미한다.

[0042] 또한 $Array_l$의 각 요소에는 픽셀의 인터럽션 목록에 직접 액세스하기 위한 메타 정보가 있다. 메타 정보에는 $Array_i$의 시작 인덱스와 픽셀의 인터 섹션 목록 길이가 포함된다.

[0043] 마지막으로, 도 15를 참조하면 각 픽셀에 대한 교차 목록을 깊이(거리)별로 정렬하고, 완전한 교차 목록인 $Array_i$와 $Array_l$를 얻는다(S206).

[0044] 이때 각 픽셀의 목록이 각각 독립적이므로 마지막 정렬을 병렬로 수행할 수 있다.

[0045] 또한 GPU(200)가 뷰 독립적인 정보(예 : 교차 목록)을 얻으면 모든 광선에 대해 레이 캐스팅을 병렬로 수행한다.

[0046] GPU(200)의 스레드 블록에서 광선 간 일관성을 향상시키기 위해 이미지 타일을 기반으로 광선 그룹을 만든다. 광선 그룹을 사용 가능한 스레드 블록에 분배하고 블록의 스레드가 주어진 광선을 처리한다.

[0047] 광선에 대한 레이 캐스팅 프로세스는 Bunyk 알고리즘을 따르지만 모든 면에 대해 VDFI를 유지 관리하지는 않는다. 대신, 최근의 GPU(200) 아키텍처에 대한 적절한 접근 방법으로서, 주문형 VDFI 연산 전략을 사용한다.

[0048] 종래 기술과의 비교(벤치 마크)를 위한 데이터 세트의 경우, 구축 알고리즘이 30% 연결된 목록 기반 방법과 비교할 때 평균적으로 더 많은 시간이 소요된다(1-86 %). 그러나 데이터 세트의 크기가 커짐에 따라 성능 격차가 줄어들었고 그 차이는 수치적으로 100 밀리 초 미만이었다.

[0049] 또한, 어레이 기반 교차 목록을 갖는 CPU(100) 레이 캐스팅 알고리즘(예 : Bunyk 알고리즘)은 링크된 목록을 사용하는 것보다 더 나은 성능을 나타내었다. 결과적으로 전체 볼륨 렌더링 프로세스에 연결된 목록을 사용하는 것과 호환되는 성능(± 1 %)을 달성했다.

[0050] 본 발명의 레이 캐스팅 알고리즘에 대해 보다 자세히 설명하면 다음과 같다.

[0051] 본 발명에서 광선은 서로 독립적이며, GPU(200)는 광선을 병렬로 처리한다.

[0052] 더 효율적인 메모리 액세스를 위해 광선 일관성을 고려하면서 광선을 스레드에 분배하는 방법에 대해 후술하여 자세히 설명하며 그런 다음 왜 주문형 VDFI 연산 전략을 채택하는지 후술하기로 한다.

[0053] 먼저 이미지 타일 기반 광선 분배에 대해서 설명하면, L1과 L2 캐시의 이용 효율을 높이려면 스레드 블록에서 스레드의 메모리 액세스 지역성을 향상시켜야 한다. 레이 캐스팅 프로세스의 경우 높은 지역성을 얻기 위해 그룹의 광선 간 광선 응집력을 증가시켜 안내할 수 있다.

[0054] 종래의 VF-GPU(200)는 가시성면을 기반으로 광선을 분배하여 스레드 블록에 의해 처리되는 광선 응집력을 증가시킨다. 그러나, 이 접근법은 다수의 가시적 인면들이 동일한 픽셀에 투영될 수 있기 때문에 비 볼록 메시를 가진 아티팩트를 생성할 수 있다.

[0055] 본 발명에서는 비볼록 메쉬에 대한 정확한 결과를 보장하기 위해 이미지 평면을 기반으로 광선 그룹을 만든다.

[0056] 이미지 평면을 사용하는 하나의 단순한 광선 분포는 픽셀의 순서에 따라 광선을 스레드에 할당한다(도 3의 (a)). 그러나 GPU(200)의 스레드 블록에 의해 처리된 광선은 낮은 일관성을 생성하고 캐시 활용 효율성을 낮춘다.

[0057] 스레드 블록에서 광선 간 일관성을 높이기 위해 종래의 선행기술 3과 유사한 이미지 타일을 사용한다. 이미지

타일은 도 3의 (b)에 도시된 바와 같이 특정 크기 (예를 들어, 8x8)를 갖는 정사각형 이미지이다.

[0058] 본 발명의 레이 캐스팅 방법에서, 이미지 타일로부터의 광선은 광선 그룹을 포함하고 그들은 스레드 블록에 고르게 분배된다. 더 많은 병렬 처리를 이용하기 위해 크기가 이미지 타일 크기와 동일한 2D 스레드 블록을 사용한다. 하나의 블록에 있는 각 스레드는 타일을 스레드에 할당하는 선행기술 3과 달리 하나의 광선을 처리한다. 본 발명에서는 이미지 타일 기반의 광선 분배 방법이 간단한 픽셀 순서 광선 분배 접근법을 사용하는 것에 비해 캐시 트랜잭션을 덜 필요하게 된다.

[0059] 본 발명에서의 주문형 VDFI 연산에 대해서 자세히 설명하면, 중복 VDFI 연산을 줄이면서 Bunyk 알고리즘의 높은 메모리 오버 헤드를 해결하기 위해 VF-GPU(200)는 스레드 로컬 VDFI 버퍼를 사용할 수 있다.

[0060] VF-GPU(200)에서 스레드는 다중 광선을 처리하고 이전 광선에 대해 이미 연산된 경우 표면의 VDFI를 재사용한다. 그러나 이 방법이 최근의 GPU(200) 아키텍처의 특성을 충족시키지 못하기 때문에 GPU(200) 스레드는 워프(warp) 단위로 실행되며 분기가 있을 경우 작업이 직렬화된다. 워프 내의 모든 스레드가 버퍼에서 필요한 정보 (즉, 버퍼 히트)를 발견할 때에만 VDFI 버퍼를 이용할 수 있다. 워프 내의 하나 이상의 스레드가 버퍼에서 필요한 정보를 찾을 수 없는 경우(즉, 버퍼 미스), 워프 직렬화는 다른 스레드에 대한 버퍼 히트의 효과를 무효화한다. 이 경우 버퍼 검사가 오버 헤드가 된다. 따라서 모든 스레드에 대해 높은 버퍼 적중률을 나타낼 만큼 버퍼 크기가 크지 않으면 스레드 로컬 버퍼로 성능 향상을 얻는 것이 어렵게 된다.

[0061] VF-GPU(200)(예 : NVIDIA 사의 GeForce 8800 Ultra, 128 코어, 768 MB)에 사용된 이전 GPU(200)와 달리 최근 GPU(200)에는 수천 개의 코어가 있다 (예 : Nvidia사의 GTX 1080, 2560 코어, 8 GB).

[0062] 현재 GPU(200)의 대대적인 병렬 처리를 완전히 활용하려면 수십 또는 수천 개의 스레드를 실행해야 한다. 장치 메모리의 크기도 증가했지만 코어의 증분 수를 충족시키지 못하고 비정렬 격자 데이터의 크기도 증가했다. VTK의 구현에서, 표면에 대한 VDFI는 72 바이트를 취한다. 선행기술 3에서 70 %의 적중률을 달성한 크기인 256 개의 슬롯을 각 버퍼에 할당하면 수십만 개의 스레드(예 : 262,144)를 시작할 때 수 기가 바이트(예 : 4GB)가 필요하다. 따라서 가능한 버퍼 크기는 수십 개의 슬롯으로 제한되며 모든 스레드에 대해 높은 적중률을 기대하기는 어렵다.

[0063] 본 발명의 실험에서 또한 VDFI 광선이 표면을 만날 때마다 스레드 로컬 버퍼를 사용하는 것보다 성능이 향상되지만, 연산이 중복될 수 있다. 이 이론적 및 실험적 관찰을 바탕으로 본 발명에서는 GPU(200) 기반 레이 캐스팅 알고리즘에서 주문형 VDFI 연산 전략을 사용한다.

[0064] 본 발명에서는 32GB 메인 메모리를 갖춘 GPU(200) (Nvidia사의 GTX 1080, 2560 코어, 8GB)와 쿼드 코어 CPU(100) (Intel사의 Xeon, i7-8700)로 구성된 Windows 시스템에서 볼륨 렌더링 시스템을 구현하여 종래 기술들과 비교했다.

[0065] 본 발명에서는 CPU(100)와 GPU(200)에 대한 병렬 알고리즘을 구현하기 위해 OpenMP와 CUDA를 사용하였다.

[0066] 표 1은 각 벤치 마크의 다른 통계를 도시하고 있다. 각 열은 정점, 면, 정사면체 및 경계면의 수와 정보를 저장하는 데 필요한 메모리 공간을 도시하고 있다.

[0067] 서로 다른 볼륨 렌더링 방법을 세 가지 벤치 마크 (표 1)와 비교하여 테스트했다. 모든 벤치 마크에서 본 발명에서는 1024 x 1024 이미지 해상도를 사용하고 주어진 회전축을 중심으로 5 도 회전시켜 72 시점(뷰포인트)의 볼륨 렌더링 시간을 측정했다. 표 1에서 도 4 (a)는 선행기술 2(Bunyk)에 대한 것이고, 도 4 (b)는 선행기술 1(HAVS)에 대한 것이고, 도 4 (c)는 본 발명에 대한 것이다.

표 1

| | | Vertex | Faces | Tetra | Boundary |
|---|---|---|---|---|---|
| Dataset 1 | # | 647,073 | 7,583,488 | 3,773,943 | 71,614 |
| Fig. 4 (a) | Mem. | 5 MB | 364 MB | 121 MB | 0.6 MB |
| Dataset 2 | # | 7,320,994 | 83,385,432 | 41,257,368 | 1,741,392 |
| Fig. 4 (b) | Mem. | 56 MB | 3,817 MB | 1,259 MB | 13 MB |
| Dataset 3 | # | 7,659,517 | 85,051,683 | 41,939,085 | 2,347,026 |
| Fig. 4 (c) | Mem. | 61 MB | 4,082 MB | 1,342 MB | 19 MB |

[0068]

[0069] 표 2는 3 개의 벤치 마크 데이터 세트에 대한 프레임 당 평균 렌더링 시간 (초)을 보여준다.

표 2

| (seconds) | Bunyk | Bunyk-Parallel | HAVS | Ours-Pixel | Ours-Tile |
|---|---|---|---|---|---|
| Dataset 1 | 10.31 | 2.93 | 1.04 | 0.69 | 0.57 |
| Dataset 2 | 50.65 | 15.52 | 12.44 | 5.57 | 4.83 |
| Dataset 3 | 5.31 | 2.53 | 10.62 | 1.33 | 1.29 |

[0070]

[0071] 표 2에서 Ours-Pixel은 본 발명의 접근 방식 중 하나이다. 이 방법에서는 광선 그룹을 만들기 위해 픽셀 순서를 사용한다(도 3의 (a)). 본 발명에서는 64 개의 스레드를 가진 스레드 블록을 만들었고, 본 발명에서는 262,144 개의 스레드 (4,096 개의 스레드 블록)를 사용했다.

[0072] 표 2에서 Ours-Tile은 Ours-Pixel에서 사용된 광선 분배 방법을 8 x 8 크기의 이미지 타일 기반 접근 방식 (그림 3의 (b))으로 대체하여 구현되었다.

[0073] 접근 방법의 효율성과 정확성을 비교하기 위해 이전 연구를 토대로 다음과 같은 세 가지 대체 방법도 구현했다.

[0074] 표 2에서 Bunyk은 선행기술 2에서 Bunyk 등이 제안한 비규칙 격자 레이 캐스팅의 구현이다. 종래에는 이 비규칙 격자 레이 캐스팅 알고리즘을 위해 단일 CPU(100) 코어를 사용했다. Bunyk-Parallel은 선행기술 3에서 Kim 등이 제안한 CPU(100) 기반 병렬 Bunyk 알고리즘을 구현한 것이다. 전처리 단계로 모든면의 VDFI를 연산하고 레이 캐스팅 과정에서 이를 사용한다. 본 발명에서는 이 방법에 4 개의 CPU(100) 스레드를 사용했다. HAVS는 선행기술 2에서 Callahan 등이 제안한 가장 잘 알려진 GPU(200) 기반 셀 투영 방법 중 하나이다.

[0075] 나중에 VTK 버전에는 이 방법이 없으므로 VTK 6.3 구현을 사용했다. 모든 알고리즘은 VTK 필터로 구현되었으며 VTK 시각화 프레임 워크에서 실행되었다.

[0076] 표 2의 볼륨 렌더링 시간에는 섹션 목록 작성 및 레이 캐스팅 프로세스에 대한 시간이 포함된다. Ours-Pixel 및 Ours-Tile은 CPU(100) 코어를 사용하는 Bunyk보다 각각 15.0 배 및 18.1 배 더 높은 성능을 나타냈다.

[0077] Bunyk-Parallel과 비교하여 Ours-Pixel 및 Ours-Tile은 각각 최고 4.3 및 5.1 배(평균 3.0 및 3.4 배) 높은 성능을 나타냈다. 데이터 세트 3의 경우 CPU(100)에서 실행되는 교차 목록 구성에 약 0.9 초가 걸리므로 Bunyk-Parallel의 전체 볼륨 렌더링 시간의 약 35 %를 차지하므로 이 방법의 성능이 상대적으로 낮아졌다. 본 발명의 방법과 동일한 GPU(200)를 사용하는 HAVS Bunyk-Parallel for Datasets 1 및 2에 비해 각각 최대 2.8 및 1.2 배 높은 성능을 보였다. 그러나 본 발명의 레이 캐스팅 방법은 일반적으로 종래의 HAVS보다 우수한 성능을 제공한다. 예를 들어, Ours-Tile은 세 개의 데이터 세트에 대해 각각 1.8, 2.6 및 8.2 배 더 높은 성능을 보였다. 이 결과는 본 발명의 접근법의 이점을 입증한다.

[0078] 본 발명에서는 GPU(200) 기반 레이 캐스팅 알고리즘이 최신 GPU(200) 아키텍처의 특성을 적절히 고려하기 때문

에, 그러한 높은 볼륨 렌더링 성능을 얻을 수 있었다. 데이터 세트 3에서 세포는 특정 지역에 밀집되어 있다.

[0079] 본 발명에서는 레이 캐스팅 프로세스를 수행하는 동안 초기 광선 종단 방법 을 사용하여 픽셀의 불투명도가 최대값을 만족할 때 광선에 대한 프로세스를 종료 할 수 있다.

[0080] 따라서 Dataset 3의 경우 Bunyk-Parallel 조차도 모든 표면을 고려해야 하는 HAVS 보다 우수한 성능을 보였다.

[0081] 렌더링 정확도 : 도 4는 크기와 특성이 서로 다른 세 개의 비정렬 격자 데이터 세트에 본 발명의 방법을 적용한 결과를 보여주는 도면이며, 도 5는 종래의 Bunyk, HAVS 및 본 발명의 레이 캐스팅 방법에 대한 Dataset 3의 렌더링 결과를 비교하고 있다.

[0082] 도 6은 본 발명과 종래 기술을 비교하여 레이 캐스팅의 개선 성능을 보여주는 그래프로서, Bunyk(베이스 라인)에서 달성된 각 알고리즘의 성능 향상을 보여준다.

[0083] 종래의 HAVS가 픽셀로 투영된 평면을 깊이별로 정렬하지만 정렬 결과는 k 버퍼의 크기와 종래의 Callahan 등이 언급한 텍스처 메모리 액세스의 동기화 문제에 따라 올바르지 않을 수 있다.

[0084] 결과적으로 데이터 세트 3의 렌더링 결과에서 관찰된 아티팩트(도 5의 (b)). 반면에 본 발명의 레이 캐스팅방법은 Bunyk에서와 같이 GPU(200)에 정확한 레이 캐스팅 알고리즘을 구현하여 정확한 렌더링 이미지를 생성했다.

[0085] 레이 캐스팅 성능 : 본 발명의 이점을 확인하려면 GPU(200) 기반 레이 캐스팅 알고리즘을 사용하여 레이 캐스팅 프로세스의 처리 시간만 측정하고 교차 목록 작성 시간과 같은 CPU(100)의 일반적인 처리 시간은 제외했다. 종래의 Bunyk 및 Bunyk-Parallel의 경우 모든 표면에 대해 VDFI가 연산되는 전처리 시간이 포함되었다. 본 발명의 방법에는 CPU(100)와 GPU(200) 사이의 데이터 통신 시간이 포함되었다.

[0086] Ours-Tile은 Bunyk보다 36.5 배(평균 25.5 배) 높은 레이 캐스팅 성능을 보였다. 또한 3 개의 벤치 마크 데이터 세트에 대해 Bunyk-Parallel보다 각각 5.9, 3.7 및 12.5 배 더 높은 성능을 보였다.

[0087] 더 자세히 설명하자면, 본 발명에서는 데이터 세트 3에 대해 훨씬 더 높은 성능 향상을 보였다. 이것은 광선이 실제 레이 캐스팅 프로세스동안 작은 비율의 표면을 방문하고 모든 표면에 대한 VDFI를 연산하기 위한 사전 처리 시간이 오히려 오버 헤드가 되기 때문이다.

[0088] Ours-Pixel과 비교할 때 Ours-Tile은 25 %, 23 %3 개의 벤치 마크에 대해 14 % 뛰어난 레이 캐스팅 성능을 제공한다. Professional cuda c programming의 Nsight Profiler를 사용하여 두 개의 레이 캐스팅 커널을 프로파일링했으며 Our-Tile이 Our-Pixel보다 L2 캐쉬 트랜잭션을 적게 차지한다는 것을 발견했다(예 : 데이터 집합 1의 트랜잭션이 21 % 감소). 이 결과는 이미지 타일 기반의 광선 분포 접근법이 스레드 블록에서 광선 일관성을 증가시키고 실제로 캐시 활용 효율성을 향상시킨다는 것을 보여준다.

[0089] 표 3은 서로 다른 타일 크기의 평균 레이 캐스팅 시간을 나타낸다.

*표 3*

[0090]

| Tile size | (8,4) | (8,8) | (16,8) | (16,16) | (32,16) |
|---|---|---|---|---|---|
| Dataset 1 | 485.30 | **470.62** | 476.84 | 474.40 | 586.92 |
| Dataset 2 | 3872.34 | 3813.82 | 3896.33 | **3802.77** | 4684.05 |
| Dataset 3 | 117.46 | **115.73** | 119.06 | 120.72 | 157.01 |

[0091] 표 3의 결과에서 볼 수 있듯이, 8 X 8 타일은 일반적으로 좋은 성능을 보여준다. 본 발명에서는 서로 다른 버퍼 크기로 데이터 집합 1의 레이 캐스팅 시간을 보여준다.

[0092] 본 발명에서는 큰 이미지 타일 크기는 스레드 블록에 대해 많은 수의 자원 (예 : 레지스터)을 필요로 하고 병렬 블록의 수를 감소시키기 때문에, 병렬 처리 수준(예 : 명령 처리량)을 낮추는 것을 발견했다. 한편, 8 ㅅ 4 (또는 더 작은) 이미지 타일의 경우, 각 스레드 블록은 워프만 포함하고 SM에서 사용 가능한 리소스를 완전히 활용하지 못했다.

[0093] VDFI 연산과 관련하여 보다 자세히 설명하면, 데이터 세트 1은 NVIDIA 사의 GTX 1080의 장치 메모리를 사용하여

모든면에 대해 VDFI를 로드할만큼 충분히 작다.

[0094] 종래의 Maxio 등이 제안한 Memory efficient gpu-based ray casting for unstructured volume rendering의 접근 방식과 비교하여 주문형 VDFI 연산 전략의 효율성을 검증하기 위해 다음과 같이 수행하였다.

[0095] 본 발명에서는 또한 다음의 두 알고리즘을 구현했다. GBunyk-Buffer는 각 스레드에 대해 VDFI 버퍼를 저장한다. 스레드는 VDFI 버퍼를 이용하기 위해 여러 개의 광선을 처리해야 하기 때문에 이미지 타일을 스레드에 할당했다. 그러나 스레드에 대한 많은 수의 광선이 병렬 처리를 낮추기 때문에 4 x 4 타일을 사용했다.

[0096] Ours-Full VDFI는 Our-Tile의 수정된 버전이다. 모든 표면에 대한 VDFI를 미리 연산하여 레이 캐스팅 중에 사용한다. 또한 GPU(200)에서 VDFI 연산을 수행하고 결과를 장치 메모리에 유지한다. VDFI 연산은 데이터 세트 1에 대해 약 23ms가 걸렸다. 본 발명에서는 두 알고리즘과 Ours-Tile의 레이 캐스팅 시간을 측정했다. 그 결과를 도 7에 나타낸다.

[0097] GBunyk-Buffer의 경우 각 버퍼의 슬롯 수를 4에서 256 (장치 메모리가 허용하는 최대 크기)으로 변경했다. Ours-FullVDFI는 VDFI에 대한 중복 연산을 제거하므로 Ours-Tile보다 약 8 % 향상된 성능을 보였다. 256 슬롯의 GBunyk-Buffer는 Ours-Tile과 비교하여 약 1.5 배 더 많은 메모리 액세스 요청과 최대 4.8 배의 L2 캐시 트랜잭션을 처리하는 것으로 나타났다. 모든 스레드에 대해 높은 버퍼 히트 비율을 얻기에는 버퍼 크기가 충분하지 않기 때문에 버퍼를 검사하는 데 추가적인 오버 헤드가 발생하고 많은 수의 메모리(또는 캐시) 트랜잭션이 발생한다. 본 발명에서는 L1 캐시 히트 비율도 주문형 전략에 비해 약 18.5 % 감소함을 발견했다. 이는 스레드당 버퍼 크기가 증가함에 따라 캐시 사용 효율이 감소했기 때문이다. 이 결과는 주문형 VDFI 연산 전략이 VDFI 버퍼를 사용하는 것보다 최신 GPU(200) 아키텍처에 더 적합한 선택임을 확인할 수 있다.

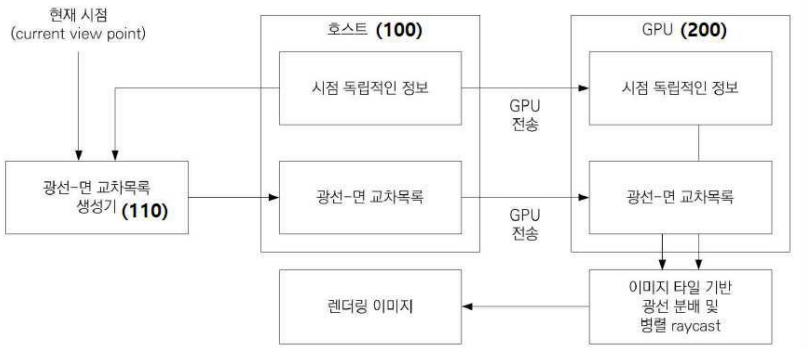
## *부호의 설명*

[0098] 100 : CPU  110 : 광선-면 교차목록 생성기

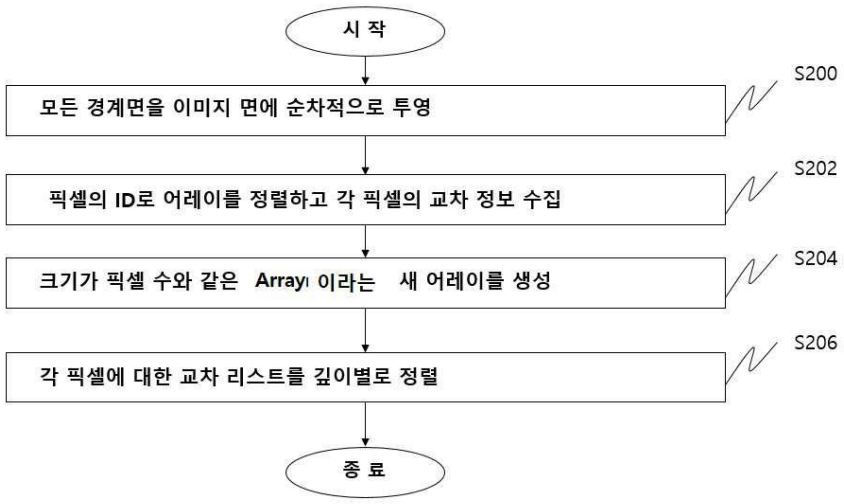
200 : GPU

## *도면*

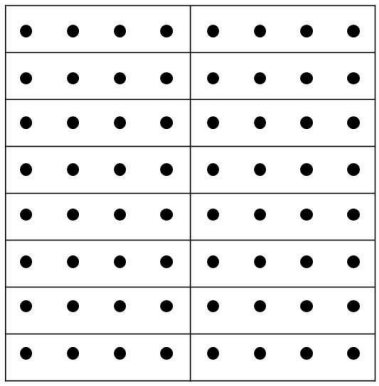
### *도면1*



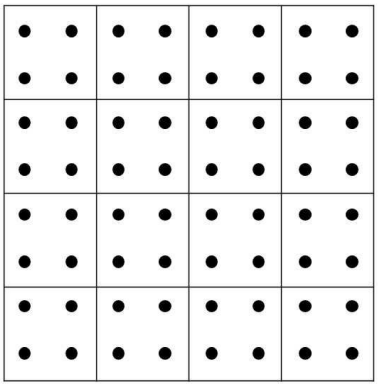
도면2




도면3

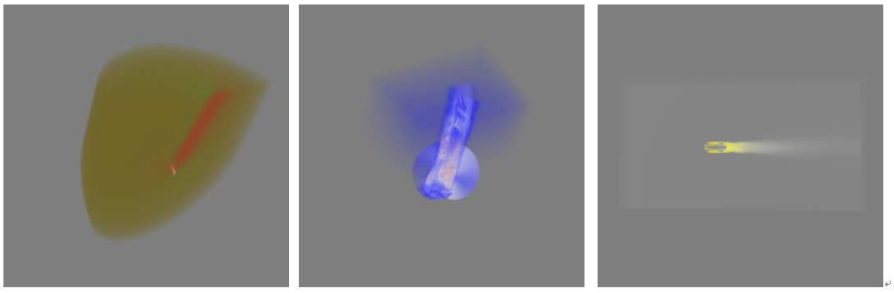


(a) Pixel-order ray distribution



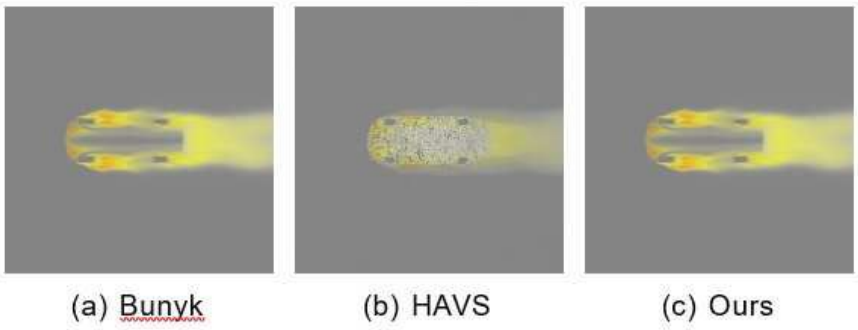
(b) Image-Tile based distribution

도면4

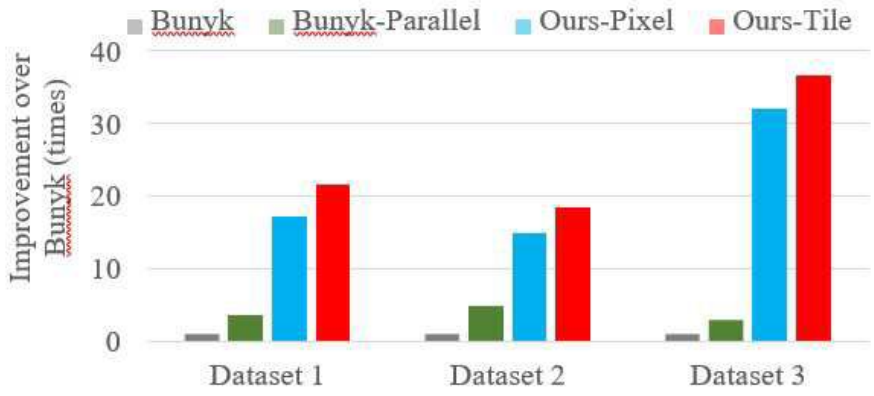


(a) Dataset 1 (b) Dataset 2 (c) Dataset 3

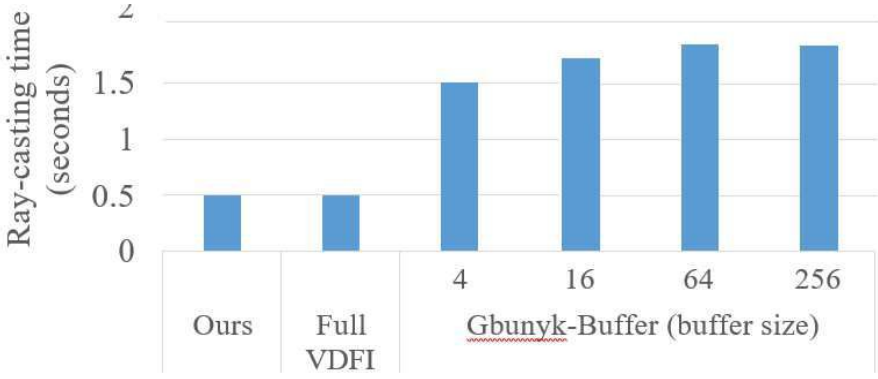
도면5


(a) Bunyk
(b) HAVS
(c) Ours

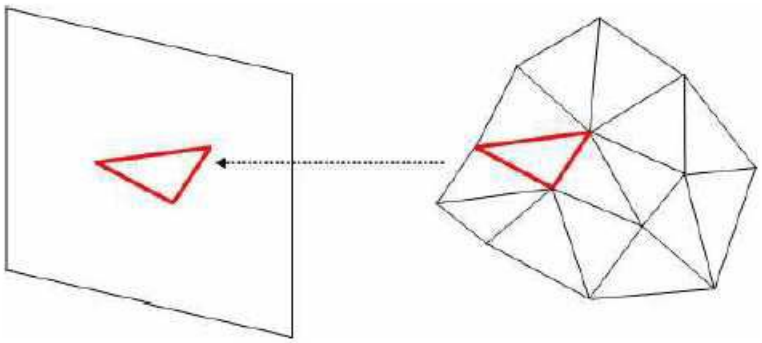
도면6


Bunyk
Bunyk-Parallel
Ours-Pixel
Ours-Tile
Improvement over Bunyk (times)
40
30
20
10
0
Dataset 1
Dataset 2
Dataset 3

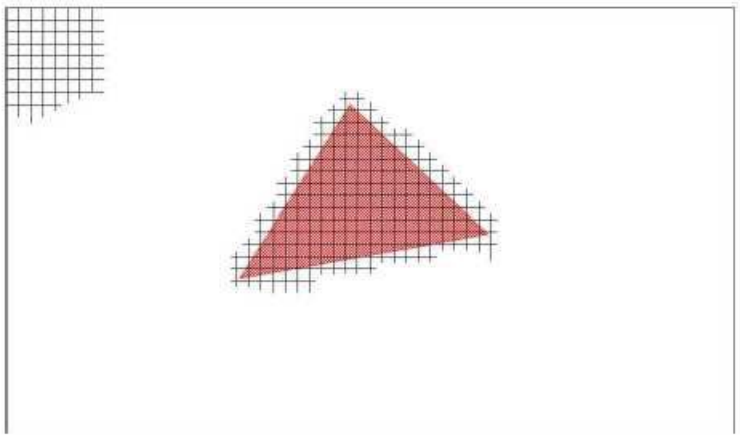
도면7


Ray-casting time (seconds)
2
1.5
1
0.5
0
Ours
Full VDFI
4
16
64
256
Gbunyk-Buffer (buffer size)

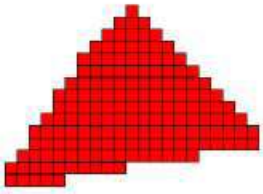
도면8



도면9



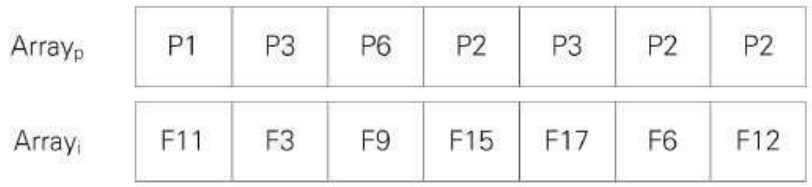
도면10



도면11



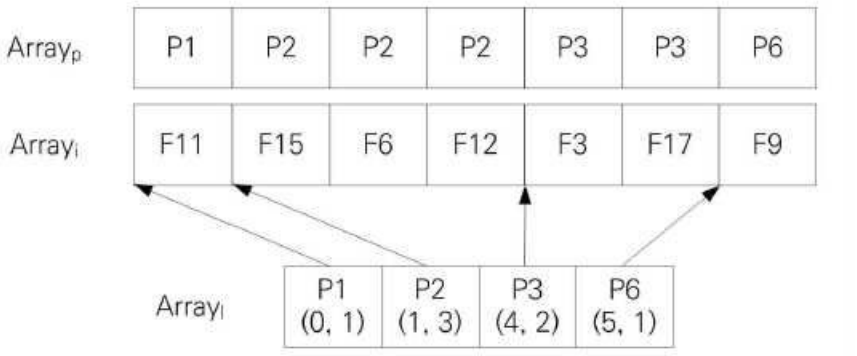
도면12




도면13




도면14




도면15

