EUROGRAPHICS 2018 / D. 구티에레즈 및 A. 셰퍼
(객원 편집자)

37권(2018) 2호

GPU에서 스파스 볼륨을 사용한 빠른 유체 시뮬레이션

쿠이 우 [1] Nghia Truong [1,2] 첼 육셀과 라마 회츨라인

[1] 유타대학교 [2] 엔비디아 코퍼레이션

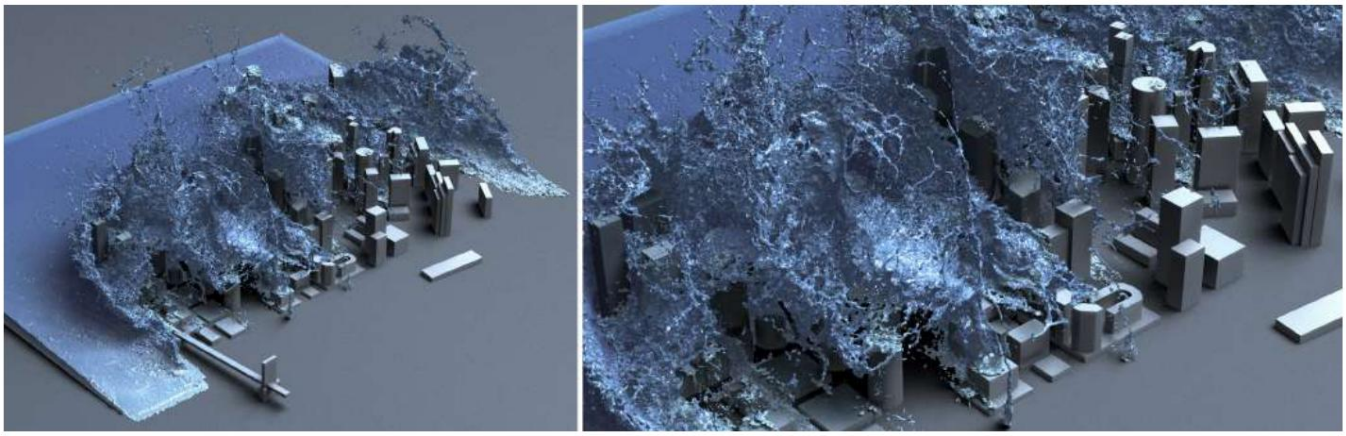

그림 1: Quadro GP100 GPU에서 평균 1.8초/프레임으로 공간적으로 희박하고 매트릭스가 없는 FLIP 솔버를 사용하여 512×256×512 그리드에서 2,900만 개의 입자로 시뮬레이션한 도시 장면.

추상적인

NVIDIAR GVDB Voxels 에 표현된 그리드의 희박한 계층 구조에 FLIP(fluid-implicit particle) 방법을 사용하여 GPU 하드웨어에서 효율적인 대규모 유체 시뮬레이션을 소개합니다 . 우리의 접근 방식은 거의 제한이 없는 시뮬레이션 영역 내에서 수천만 개의 입자를 처리합니다. 병렬 희소 그리드 계층 구성을 위한 새로운 기술과 움직이는 입자에 대한 GPU의 빠른 증분 업데이트를 설명합니다. 또한 FLIP 기술은 입자에서 복셀로 수집하는 희소하고 작업 효율적인 병렬 데이터 수집과 희소 그리드에 최적화된 매트릭스 없는 GPU 기반 공액 구배 솔버를 도입합니다. 우리의 결과는 우리의 방법이 CPU에서 실행되는 FLIP 시뮬레이션과 비교하여 GPU에서 최대 10배 더 빠른 시뮬레이션을 달성할 수 있음을 보여줍니다.

CCS 개념 • 컴퓨팅
방법론 → 물리적 시뮬레이션; 대규모 병렬 및 고성능 시뮬레이션

1. 소개

FLIP(Fluid-Implicit-Particle) 방법 [ZB05] 은 단순성과 낮은 손실로 인해 컴퓨터 그래픽에서 다양한 유형의 유체 현상을 시뮬레이션하는 데 널리 사용되었습니다. 하이브리드 기술로서 고품질 FLIP 시뮬레이션에는 많은 수의 입자와 고해상도 그리드가 모두 필요합니다. 우리의 방법은 메모리 요구 사항을 줄이고, 도메인을 동적으로 추적하고, 스파스 볼륨 표현에 상주하는 병렬 GPU 기반 계산을 사용하여 이러한 문제를 해결합니다.

우리는 GVDB 희소 복셀 데이터 구조 [Hoe16], VDB 희소 격자 계층 [Mus13] 의 GPU 친화적인 구현을 사용합니다 . 희소 그리드를 사용한 계산은 유체 시뮬레이션에 여러 이점을 제공합니다. 첫째, 유체가 차지하지 않는 빈 공간의 저장을 줄입니다. 전적으로 GPU 메모리에서 시뮬레이션을 고려하므로 이는 대규모 시뮬레이션을 실행하는 데 중요한 요소입니다. 또한 GVDB 복셀을 사용하면 그리드 크기를 미리 정의할 필요가 없습니다. 유체가 공간에서 이동함에 따라 필요에 따라 새로운 복셀이 자동으로 할당될 수 있습니다. 또한, 복셀 데이터는 밀도가 높은 복셀 그룹의 모음으로 3D 텍스처에 저장됩니다(예:

벽돌), 계산 및 하드웨어 가속 삼중 선형 필터링 중에 빠른 데이터 검색이 가능합니다. 마지막으로 GPU에서 스텐실 작업을 가속화하기 위해 앞치마 복셀을 사용하여 이웃 조회를 수행합니다.

희소 그리드에 대한 유체 시뮬레이션에는 여기에서 다루는 여러 가지 문제가 포함됩니다. 우선, 스파스 구조는 많은 수(테스트에서는 수천만 개)의 입자로부터 모든 프레임에서 재구성되어야 합니다. 우리는 트리 토폴로지를 재구성하고 업데이트하는 효율적인 알고리즘을 도입하여 이 문제를 해결합니다. 둘째, 입자-그리드 래스터화는 모든 계산을 GPU에서 유지하면서 GPU 스레드 블록과 데이터 일관성을 효율적으로 활용하는 병렬 수집 방법을 도입하여 해결하는 성능 문제를 제시합니다. 마지막으로, FLIP의 압력 솔버 단계를 효율적으로 처리하기 위해 희소 복셀 그리드에 대한 행렬이 없는 접합 기울기 솔버를 소개합니다. 희소 그리드와 매트릭스가 없는 솔버를 함께 사용하면 GPU 메모리에서 이전에 달성한 것보다 훨씬 더 큰 볼륨을 지원할 수 있을 뿐만 아니라 유사한 고밀도 멀티 코어 CPU 구현에 비해 최대 2배의 속도 향상에 도달할 수 있습니다.

우리의 주요 기여는 증분 업데이트가 있는 빠른 동적 계층 토폴로지, 하위 셀에 의한 효율적인 GPU 친화적 희소 공간 분할, 가속 입자-격자 래스터화를 통한 빠른 FLIP 시뮬레이션 및 희소에 직접 매트릭스가 없는 공액 구배 솔버로 구성됩니다. 볼륨. 이러한 각 기여에 대해 자세히 설명합니다.

2. 배경

컴퓨터 그래픽의 유체 시뮬레이션에 대한 많은 작업이 있습니다. 우리의 토론은 FLIP 및 GPU의 유체 시뮬레이션과 밀접하게 관련된 이전 작업에 중점을 둡니다. 시뮬레이션에 사용되는 GVDB 복셀 구조에 대한 간략한 설명을 제공합니다.

2.1. 관련된 일

FLIP [BR86]을 사용한 유체 시뮬레이션은 PIC(particle-in-cell) 방법 [Har64] 과 함께 컴퓨터 그래픽 [ZB05] 에 도입된 이후 널리 사용되는 접근 방식이었습니다 .

PIC와 FLIP 모두 그리드에서 수행되는 압력 해결과 함께 이류를 위해 입자를 사용합니다. FLIP은 PIC의 수치적 확산을 다루지만 수치적 에너지 소실은 다루지 않습니다 [MCP* 09]. 따라서 앞뒤 오류 보상 및 정정(BFECC) [KLLR07] 및 미분 입자 [SKK07] 방법을 사용하여 입자와 복셀 간에 데이터를 전송할 때 수치적 손실을 줄일 수 있습니다. 나중에 연구원들은 증폭된 비말 효과 [KCC* 06, GB13], 정확한 고체-유체 결합 [BBB07] 및 적절한 공기-액체 인터페이스 [BB12] 를 제공하여 PIC/FLIP 방법을 개선했습니다 . 최근 APIC(Affine Particle In-Cell) [JSS* 15] 가 도입되어 PIC의 분산 문제를 안정적으로 제거하여 입자에서 그리드로 이동하는 동안 각운동량을 정확하게 보존합니다. IVOCK(Integrated Vortic ity of Convective Kinematics) [ZBG15] 방법은 이류 방법과 독립적으로 이류 동안 소실된 와도를 대략적으로 복원하기 위해 도입되었습니다.

PIC/FLIP 방법은 컴퓨터 그래픽에서 처음 사용된 것이 모래 시뮬레이션 [ZB05] 이었기 때문에 유체 시뮬레이션에 국한되지 않습니다. 나중에 세분화된 재료의 내부 압력과 마찰 응력을 해결합니다 [NGL10]. PIC/FLIP은 자체 및 다른 객체와의 헤어 충돌을 처리하는 데에도 사용됩니다 [MSW* 09]. 최근에는 연속체 공식을 사용하여 PIC/FLIP을 일반화하는 MPM(Material Point Method)이 눈 [SSC* 13], 모래 [DBD16,KGP* 16] 와 같은 다양한 종류의 재료 시뮬레이션을 효과적으로 처리하는 것으로 나타났습니다. , 점탄성 유체, 발포체 및 스폰지 [RGJ* 15, YSB* 15] 및 이방성 탄소성 재료 [JGT17].

또한 PIC/FLIP 방법을 가속화하기 위한 광범위한 사전 작업이 있습니다. 푸아송 방정식을 스파스 선형 시스템으로 푸는 것과 관련된 그리드에 대한 압력 해석은 종종 PIC/FLIP 접근 방식의 병목 현상입니다. 하위 영역에서 병렬로 압력을 분해하고 해결하는 것은 약간의 관심을 받았습니다 [NSCL08,WST09,GNS* 12]. 다른 접근 방식은 저해상도 그리드 [PTC* 10,LZF10,EB14,ATW15] 또는 다중 그리드 주기를 켤레 기울기 솔버 [MST10] 의 선조건자로 사용합니다.

균일 그리드를 사용하는 대신 사면체 이산화 [ATW13], 원거리 필드 그리드 [ZLC* 13] 및 희소 균일 그리드 [AGL* 17] 가 공격적인 적응성을 위해 사용됩니다. 스파스 체적 구조는 또한 저장 요구 사항을 줄이기 위해 사용되었습니다 [SABS14]. 최근에는 Azevedo at el. [ABO16] 매우 거친 그리드에서 액체를 시뮬레이션하기 위해 위상학적으로 정확하고 경계를 준수하는 절단 셀 메시를 도입한 반면 Liu et al. [LMAS16] 및 Chu et al. [CZY17]은 분해된 도메인에서 푸아송 방정식을 병렬로 풀기 위한 Schur-complement를 제시합니다. Baileyet al. [BBAW15]는 또한 대규모 액체 시뮬레이션을 위해 분산 시스템에서 희소 체적 구조를 사용합니다.

압력 해결 외에도 PIC/FLIP 방법의 성능을 향상시키기 위해 다른 방법이 도입되었습니다. Lentineet al. [LCPF12]는 큰 시간 단계를 취하기 위해 정확한 레벨 세트 표현을 계산할 것을 제안했습니다. 일반적으로 FLIP에서와 같이 복셀당 8개의 입자를 사용하는 대신 Batty 및 Bridson [BB08]은 더 넓은 SPH와 같은 커널을 사용하여 입자 속도를 그리드로 전송하기 위해 더 큰 입자 반경을 가진 복셀당 하나의 입자만 사용할 것을 제안합니다. Ferstl et al. [FAW* 16] 액체 표면의 협대역 내에서만 입자를 유지하기 위해 협대역 FLIP 방법을 도입합니다.

GPU의 계산 능력은 일반적으로 유체 시뮬레이션을 위한 매력적인 하드웨어 솔루션이 되었습니다. Harris [Har05] 는 it erative Jacobi 방법을 사용하여 GPU에서 Eulerian 유체 시뮬레이션을 구현합니다. Molemakeret al. [MCPN08]은 GPU에서 반복적인 멀티그리드 기반 푸아송 솔버로 IOP(Iterated Orthogonal Projections)를 도입했습니다. Horvath와 Geiger [HG09] 는 사전 시뮬레이션된 화재 데이터를 성긴 격자로 2D 슬라이스로 나누고 여러 개별 GPU에서 병렬로 보조 Eulerian Navier-Stokes 솔버를 수행했습니다. Chentanez와 Müller는 GPU에서 압력 프로젝션을 위한 특수 멀티그리드 방법을 개발했으며 [CM11a,CM11b] 국부적으로 그리고 전역적으로 질량을 보존하는 GPU 친화적 샤프닝 필터를 도입했습니다 [CM12]. 최근 Chen et al. [CKIW15]는 실시간 페인팅 시뮬레이션을 위해 Jacobi 반복을 가속화하기 위한 GPU 기반 고정 소수점 방법을 도입했습니다.

2.2. GVDB 데이터 구조

GVDB Voxels는 [Mus13] 의 작업을 기반으로 그리드 [Hoe16] 의 희소 계층 구조에서 복셀을 효율적으로 표현하기 위한 프레임워크입니다 .

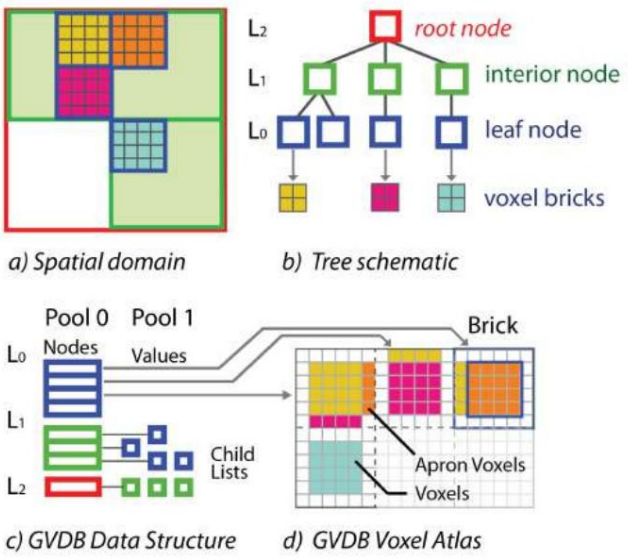

그림 2: GVDB 데이터 구조는 리프 노드가 브릭의 복셀 데이터를 가리키는 3D 그리드(b)의 트리로 공간 도메인(a)을 커버하는 희소 복셀을 나타냅니다. GVDB 데이터 구조는 두 개의 메모리 풀(c)을 사용하여 기본 노드 속성을 풀 0에 저장하고 자식 목록을 풀 1에 효율적으로 저장합니다. 리프 노드는 실제 복셀 데이터를 3D 복셀 내부에 풀링된 조밀한 3D 하위 볼륨인 브릭으로 참조합니다. 아틀라스(d)는 나중에 이웃 계산을 위해 앞치마 복셀의 추가 경계로 둘러싸여 있습니다.

GVDB 데이터 구조는 그림 2a에 표시된 것처럼 3D 그리드의 계층 구조를 사용하여 복셀이 있는 공간 도메인을 성기게 포함합니다. 구성 벡터는 각 그리드 수준의 해상도를 log2 차원의 벡터로 식별합니다. 예를 들어, 2,3,4 구성은 (2 자식이 있는 루트 노드 L2, (2가 있는 레벨 L1 의 중간 노드 및 (2 = 4,096 복셀. 이 구성은 2+3+4 (기본적으로 2개의 5개 레벨 트리, 매우 큰 주소 공간 제공)의 체적 도메인을 처리할 수 있습니다. 레벨은 0(리프 ^{2}) ^{삼} = 64 노드)에서 L(루트 노드)까지 번호가 매겨져 편리하게 허용 그림 ^{삼}) ^{삼} = 어린이 512명, 4명 2b와 같이 트리 높이를 변경할 수 있습니다 .) ^{삼}

) ^{삼} ≈ 1억 3,400만 복셀. GVDB 복셀은 최대

효율적인 GPU 메모리 관리를 위해 GVDB 복셀은 그림 2c와 같이 두 개의 노드 풀이 있는 트리 계층 구조를 저장합니다 .
주어진 수준 I의 각 노드 N(i)I은 메모리 풀 그룹 P0I에 저장 되고 별도의 풀 P1I에 저장된 세계 공간 위치, 부모 인덱스 및 자식 목록으로 구성됩니다 .
내부 노드의 경우 목록의 각 자식은 다음으로 가장 낮은 수준에서 P0l-1 의 노드에 대한 참조입니다 .
레벨 0의 리프 노드에는 자식이 없으며 대신 참조를 유지합니다.
3D 텍스처 아틀라스에 저장된 복셀 벽돌에 연결합니다.

노드 풀이 GVDB 토폴로지를 저장하는 반면, 하위 풀에서 할당된 복셀 브릭은 데이터는 그림 2d에 표시된 복셀 아틀라 ^{삼}
스의 고밀도 n 볼륨으로 표시됩니다 . 아틀라스는 3D 하드웨어 텍스처로 구현되어 3선형 보간 및 GPU 텍스처 캐시를 활성화합니다. 채널 또는 복셀 속성은 여러 아틀라스에서 지원됩니다.

3. 시뮬레이션을 위한 GVDB 복셀

원래 GVDB 데이터 구조는 하위 목록을 압축하는 데 비트마스크가 사용되는 VDB 접근 방식을 모방하도록 설계되었습니다. 실제로 토폴로지의 저장 공간은 복셀 아틀라스에 비해 작으며(<1MB) 비트마스크 압축은 노드의 동적 삽입 및 제거를 복잡하게 만듭니다. 각 노드에 대한 명시적 하위 목록을 사용하도록 GVDB 복셀을 수정합니다. 이러한 목록은 점유된 노드에서만 발생하므로 희소성은 여전히 달성됩니다. 새로운 구현은 비트마스크 테이블을 제거하고 P1의 null 값을 사용하여

비활성 하위 노드. 자식의 동적 삽입 및 제거는 이제 O(1)입니다.

이웃 계산은 각 브릭 주위에 추가 복셀 경계로 유지되는 앞치마 복셀로 가속화되고 브릭 경계를 넘어 값을 복제합니다. 앞치마 복셀을 업데이트하는 것은 올바른 값이 세계 공간의 복셀 이웃이기 때문에 사소한 일이 아닙니다(그림 2a 및 2d 참조). 앞치마 업데이트는 데이터 일관성을 유지하기 위해 반복적으로 호출되고 CG 솔버의 내부 루프 중에 사용되므로 성능이 중요합니다. 이전 기술은 각 축을 따라 하나의 커널을 시작하며, 각 스레드는 GVDB 토폴로지를 통과하여 세계 공간 이웃을 식별함으로써 아틀라스 공간에서 하나의 앞치마 복셀을 해결합니다. 우리는 순회를 수행하는 앞치마 복셀당 하나의 스레드로 가장자리와 모서리를 포함하여 브릭의 6면을 모두 덮는 단일 커널을 시작합니다. 기존의 3커널 방식에 비해 성능이 5배 향상되었습니다.

4. 시뮬레이션 알고리즘 개요

우리의 유체 시뮬레이션 기술은 알고리즘 1에 제시되어 있으며 희소 볼륨을 설명하고 GPU에서 계산을 가능하게 하기 위해 추가된 FLIP(Fluid-Implicit Particle) 방법을 따릅니다.

첫째, GVDB 토폴로지는 프레임마다 동적으로 변경되므로 트리의 전체 또는 증분 재구성부터 시작합니다(섹션 5.1 및 5.2). 이전 볼륨 데이터는 각 프레임에서 지워집니다. 둘째, 레벨 세트를 포함하여 GPU에서 점 대 복셀 래스터화를 가속화합니다.

알고리즘 1 Sparse FLIP 시뮬레이션 1: 절차

```
SparseFLIP( )
2: P ← 초기점
3: V ← GVDB 구조 4: 5: 6: 7: 8:
9:      각 프레임에 대해 if
            first frame then 수행
                Vtopo ← 전체 재구축(P)                    섹션 5.1
            else
                Vtopo ← 증분 빌드(P) end if               섹션 5.2
10:         V ← 크기 조정 및 지우기 (Vtopo)
11:         S ← 서브셀에 포인트 삽입 (V,P)                 섹션 6
12:         V(vel) ← 입자-복셀(S,P)                       섹션 7
13:         V ← 앞치마 업데이트 (ρ, vel,marker)
14:         V(벨로드 ) ← V(벨)
15:         V(div) ← 발산 (V(vel))
16:         V(ρ) ← CG 압력 해결 (V,div)                   섹션 8
17:         V(vel) ← 압력 대 속도(V(ρ))
18:         V ← 업데이트 앞치마 (V(vel))
19:         P ← 어드밴스 (V(vel),V(velold ))
20:     끝
21: 종료 절차
```

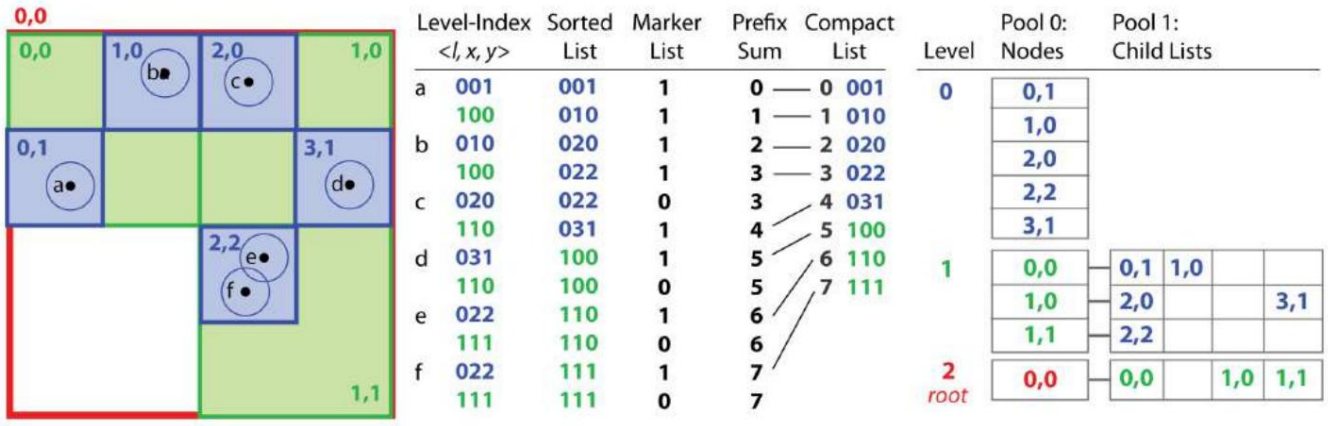

그림 3: 2D로 전체 토폴로지 재구성: 입력 포인트 a, b, c, d, e, f가 주어졌을 때 목표는 트리를 포함하는 것입니다. GVDB 노드의 인덱스 위치는 노드의 왼쪽 상단에 있습니다. 포인트를 레벨 인덱스 목록이라고 하는 x, y,z 키 목록으로 변환한 후 기수 정렬을 사용하여 모든 레벨을 동시에 정렬합니다. 그런 다음 정렬된 목록의 각 값을 이전 값과 확인합니다. 동일한 경우 마커 목록에 0 값을 설정하고, 그렇지 않으면 값을 1로 설정합니다. 마커 목록에서 출력 압축 목록에 대한 절대 위치를 제공하기 위해 접두사 합계를 계산합니다. 정렬된 목록 값은 마커 값이 1인 경우에만 접두사 위치의 압축 목록에 기록됩니다. 결과 목록은 오른쪽에 표시된 다중 레벨 트리 토폴로지를 직접 채우는 데 사용할 수 있습니다. 하위 목록의 빈 상자는 null 값입니다.

렌더링을 위해 섹션 6에서 설명한 하위 셀에 포인트를 삽입합니다 .
셋째, 앞치마 업데이트로 희박한 벽돌 경계를 처리합니다. 마지막으로 희소 복셀 그리드에 대한 매트릭스가 없는 켤레 기울기 솔버는 FLIP의 압력 솔버 단계를 효율적으로 처리하는 데 사용됩니다(섹션 8). 다른 모든 단계는 이전 및 새 그리드 속도를 사용하는 발산, 압력 대 속도 및 이류를 포함하여 GPU 가속을 사용하는 일반적인 FLIP 구현을 따릅니다.

5. 동적 토폴로지 구축

시뮬레이션 중에 각 입자를 포함하려면 희소 복셀 브릭으로 덮어야 합니다. 그러나 각 GVDB 브릭에는 수천 개의 입자가 포함될 수 있습니다. 각 입자에 대해 트리를 순회하면 병렬로도 많은 수의 쓰기 충돌과 중복 노드가 발생합니다. 다행스럽게도 각 GVDB 노드에는 고유한 절대 공간 인덱스가 있어 이에 속하는 입자를 분류하는 데 사용할 수 있습니다. 따라서 우리의 솔루션은 모든 수준에서 동시에 공간 인덱스를 기반으로 입자를 정렬한 다음 중복을 제거하는 것입니다. 나머지 집합은 모든 입자를 포함하는 고유한 노드 목록입니다.

결과적으로 모든 GVDB 노드는 병렬로 식별되고 실제 위치는 한 번만 지정됩니다. 전체 및 증분 재구축을 모두 고려합니다.
이 방법을 사용하는 GVDB 트리 계층 구조는 전체 재구축보다 훨씬 빠르게 움직이는 입자의 증분 업데이트를 사용합니다.

5.1. 전체 토폴로지 재구축

토폴로지 재구축의 목표는 파티클 세트를 포함하는 GVDB 트리의 내부 및 리프 노드 세트를 효율적으로 구성하는 것입니다.
이것은 입자 반경까지 복셀을 할당하기 위해 두 번의 패스로 수행됩니다.

5.1.1. 파티클 센터의 첫 번째 패스

첫 번째 패스는 입자 중심을 포함하는 트리 노드 세트를 결정합니다. 유체 도메인이 가상으로 제한되지 않기 때문에 트리
깊이는 다를 수 있으며 입자의 각 프레임에서 다시 계산됩니다.

경계 상자. 덮는 노드 집합은 모든 수준에서 병렬로 노드를 생성하고 중복을 제거하여 결정됩니다. 마지막으로 노드가 재할당되고 부모-자식 목록이 업데이트됩니다. 이러한 단계의 세부 정보는 다음과 같습니다.

1단계. 트리 수준 L 의 수를 결정합니다. 모든 입자를 포함하는 데 필요한 루트 노드의 수준 L을 결정하는 모든 입자를 포함하는 경계 상자를 계산하는 것으로 시작합니다. 따라서 다음 단계에서는 레벨 0에서 L-1까지의 노드만 식별됩니다. 경계 상자는 병렬 GPU 축소 알고리즘 [Har07]에 의해 계산됩니다.

2단계. 레벨 인덱스 목록을 생성합니다. 계층 구조의 모든 수준에서 각 입자를 포함하는 트리 노드의 인덱스로 구성되는 수준 인덱스 목록을 정의합니다. n을 입자의 수라고 하자. 이 목록의 총 크기는 nL이며 모든 입자에 대한 단일 패스로 생성됩니다. 수준에서 입자를 포함하는 노드의 세 인덱스 x, y 및 z는 입자 위치에서 결정됩니다.

0 ≤ ≤ L 1인 각 수준에 대해 단일 값 level-index, , x, y,z로 연결된 단일이 목록에 기록됩니다. 따라서 여러 수준에서 인덱싱된 모든 위치를 함께 정렬하여 GPU 커널 시작 횟수를 줄일 수 있습니다.

3단계. 레벨 인덱스 목록을 압축합니다. 이전 단계에서 생성된 레벨-인덱스 목록에는 각 노드에 많은 입자가 포함될 수 있으므로 중복된 값이 많이 포함되어 있습니다. 먼저 기수 정렬을 사용하여 이 목록을 압축하여 중복 값을 그룹화합니다. 그런 다음 각 요소가 이전 요소와 다르면 1로 표시하고 그렇지 않으면 0으로 표시하여 마커 목록을 구성합니다. 마커 목록에 대한 병렬 접두어 합계(스캔) [HSO07] 는 최종 압축 목록에 대한 오프셋을 제공합니다. 레벨 인덱스 목록, 마커 목록 및 접두사 합계의 길이는 동일합니다. 출력 압축 목록은 마커 목록 값이 1인 경우에만 위치에 대한 접두사 합계를 사용하여 각 고유 레벨 인덱스 노드를 찾아 구성됩니다. 압축 목록의 각 값은 토폴로지에 추가해야 하는 고유 트리 노드에 해당합니다. 모든 입자를 덮습니다. 그림 3 은 압축된 목록을 2D로 구성하는 방법을 보여줍니다.

4단계. 노드를 할당하고 초기화합니다. 압축된 레벨 인덱스 목록의 크기에 따라 GVDB의 두 메모리 풀을 할당합니다. 그런 다음 각 노드의 레벨 및 위치 값을 할당하여 노드 데이터를 초기화합니다. 모든 자식 노드 인덱스는 null 값으로 초기화됩니다.

5단계. 하위 노드 목록을 설정합니다. 마지막으로 여러 단계에서 자식 노드 인덱스를 설정합니다. 각 패스는 LN-1 에서 L0 으로 내림차순으로 레벨당 커널을 시작합니다 . 각 노드 N(i)l 에 대해 루트에서 트리를 순회하여 부모를 찾습니다. 부모 노드의 해당 자식 노드 인덱스도 동시에 설정됩니다. 유도를 통해 하향식 순서는 하위 수준을 처리하기 전에 각 부모가 토폴로지에 추가되었음을 보장합니다. 완료되면 GVDB 구조에는 모든 노드에 대한 올바른 상위 및 하위 목록이 포함됩니다.

5.1.2. 영향 상자에 대한 두 번째 패스

두 번째 패스는 복셀이 입자에서 그리드로의 속도 전송을 위한 단위 가장자리 거리 또는 레벨 세트를 생성하기 위한 반경의 두 배(렌더링용)를 커버하도록 존재하도록 보장합니다. 우리는 그림 4와 같이 입자당 경계 상자를 영향 상자라고 부릅니다. 첫 번째 패스는 입자 중심을 덮는 GVDB 노드를 생성합니다. 영향 상자를 포함하도록 레벨 인덱스 목록을 확장하기 위해 첫 번째 패스를 수정할 수 있지만 이 순진한 수정은 레벨 인덱스 목록을 8배 이상 더 크게 생성합니다. 두 번째 패스는 영향을 주는 상자가 닿는 인접 벽돌을 빠르게 생성하는 데 사용됩니다.

이 두 번째 패스의 목표는 트리에 아직 존재하지 않고 입자 중심을 포함하지 않지만 입자의 영향 상자로 인해 필요한 비교적 적은 수의 복셀 브릭을 생성하는 것입니다. 이 패스의 경우 입자의 영향을 받는 상자의 각 모서리가 이미 GVDB 노드로 존재하는지 확인하기 위해 첫 번째 패스의 2단계를 수정합니다. 코너 노드를 찾을 수 없으면 목록에 추가합니다.

누락된 노드만 원자 단위 주사위를 사용하여 목록에 추가되기 때문에 첫 번째 패스에 비해 훨씬 짧은 목록이 생성됩니다. 중복 노드가 여전히 포함될 수 있으므로 3단계를 다시 수행하여 목록을 압축합니다. 4단계와 5단계는 첫 번째 패스와 동일합니다.

5.2. 증분 재구축

FLIP 시뮬레이션의 두 연속 시간 단계에 대한 트리 토폴로지는 대부분의 입자가 동일한 노드 내에서 이동하는 반면 다른 입자는 노드를 교차할 수 있기 때문에 큰 시간 단계에서도 매우 유사합니다. 따라서 우리는 처음부터 토폴로지를 재구축하는 대신 증분 업데이트 날짜를 수행합니다. 일반적으로 각 단계에서 상대적으로 적은 수의 새 노드와 복셀만 할당하면 되며 기존 노드의 일부만 더 이상 필요하지 않습니다. 결과적으로 증분 업데이트는 토폴로지의 완전한 재구성보다 훨씬 더 효율적으로 수행될 수 있습니다.

증분 재구축은 전체 토폴로지 구성 알고리즘과 매우 유사합니다. 주요 차이점은 레벨 인덱스 목록을 생성하는 2단계에 있습니다. 원하는 증분 목록은 하위

기존 GVDB 트리에 아직 없는 새 노드만 포함하는 전체 노드 목록 집합입니다. 따라서 각 입자에 대해 입자의 영향 상자의 각 모서리를 확인하여 이미 해당 수준에 노드가 있습니다. 그렇다면 노드를 표시하십시오. 그렇지 않으면, , x, y,z가 새 노드 목록에 추가됩니다.

3~5단계는 유사하게 작동합니다. 2단계에서 기존 노드의 표시가 해제되면 해당 노드는 더 이상 필요하지 않으며 트리에서 안전하게 제거할 수 있습니다. 상향식 순서로 여러 패스를 사용하여 추가 단계로 이러한 노드를 제거합니다. 부모 노드의 해당 자식 노드 인덱스를 null 값으로 설정하여 표시되지 않은 노드를 제거합니다. 제거된 노드와 해당 브릭은 역참조되지만 메모리 풀에서 삭제되지 않으므로 각 프레임에서 다시 연결되는 새 브릭에 재사용할 수 있습니다.

6. 서브셀 분할

그림 4와 같이 각 복셀 저장 브릭을 여러 개의 작은 하위 볼륨으로 공간적으로 분할하거나 GPU 하드웨어를 위해 조직화하는 하위 셀을 도입합니다. 그런 다음 각 하위 셀에 속하거나 겹치는 모든 입자를 저장하기 위해 입자 위치 및 속도 목록을 만듭니다.

복셀 브릭의 최적 크기는 애플리케이션에 따라 다르며 성능과 점유 사이의 절충안이 있습니다 [Hoe16].
대형 브릭은 디스크 IO 및 레이트레이싱에 이상적으로 적합한 반면, 브릭당 복셀 수는 일반적으로 GPU 스레드 블록을 초과하여 로컬 계산을 비효율적으로 만듭니다. 따라서 우리는 저장을 위해 GVDB Voxel 브릭을 유지하고 하위 셀을 도입하여 스레드 블록에 대한 논리적 그룹화를 생성합니다(그림 4 참조). 하위 셀은 복셀 세트와 관련하여 계산을 위해 주어진 형상(예: 점, 다각형)을 구성합니다.

소비된 총 메모리는 다른 하위 셀과 겹치는 중복을 포함하여 모든 하위 셀에 있는 입자의 합집합입니다. 극단에서 서브셀이 하나의 복셀(13)이면 각 서브셀은 정확히 해당 복셀에 영향을 미치는 입자의 이상적인 목록이 됩니다. 그러나 전체 목록 길이는 원래 입자 목록보다 훨씬 깁니다.

서브셀이 클수록 43 = 64 복스 서브셀당 더 많은 불필요한 입자가 확인되지만 전체 목록 길다. 우리는 CUDA 워프(32× 스레드)의 배수로 서브셀당 els를 사용하 이는 더 짧아집니고 검색 길이와 메모리 소비의 균형을 맞추는 동시에 단일 GPU 스레드 블록(512× 스레드)에 맞춥니다. 하드웨어 일관성을 개선하기 위해 입자 인덱스를 저장하는 대신 위치와 속도가 각 하위 셀에 복사됩니다.

겹침으로 인한 목록의 중복으로 인해 서브셀 목록이 시뮬레이션 중에 가장 많은 메모리를 차지합니다. 메모리 사용량을 더 줄이기 위해 위치와 속도를 해당 범위로 나눈 16비트 ushort 값으로 인코딩합니다. 인코딩된 데이터는 전체 메모리의 절반만 차지합니다.

7. 입자-그리드 래스터화

FLIP 시뮬레이션의 주요 측면은 입자 속도를 복셀 그리드로 전송하는 것입니다. 가장 가까운 입자의 가중 합은 주어진 복셀에 기여합니다. 두 가지 일반적인 접근 방식은 이를 분산 또는 수집으로 수행하는 것입니다. Scatter는 여러 입자가 동일한 위치에 쓰기 때문에 텍스처 쓰기 충돌을 발생시킵니다. 또한 원자 텍스처 작업이 필요하고 불균형한 스레드 워크로드를 생성합니다. Gathering은 [KBT* 17] 에서 발견한 바와 같이 GPU에 여러 가지 이점을 제공합니다 . 각 복셀이 주변 입자의 고정된 공유 목록에서 값을 수집할 필요 없이 GPU 점유율이 향상됩니다.

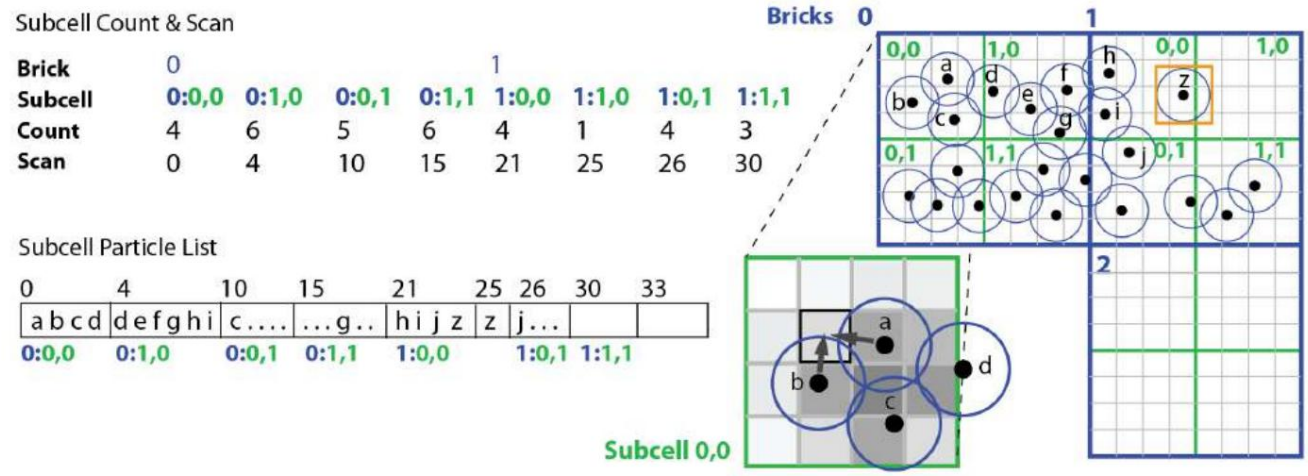

그림 4: 서브셀 분할: 서브셀은 복셀 브릭을 계산 단위로 논리적으로 세분한 것입니다. 브릭은 희소 도메인을 구성하고 레이트레이싱 및 I/O의 글로벌 성능에 맞게 크기가 조정되지만 효율적인 GPU 컴퓨팅을 위해 너무 많은 복셀을 포함합니다. 서브셀은 64개의 복셀로, GPU 스레드 블록과 CUDA 워프의 배수(32개 스레드)에 맞도록 크기가 조정됩니다. 하위 셀의 모든 복셀은 동일한 하위 셀 입자 목록을 공유합니다. 예를 들어 입자 d가 하위 셀 0,0 및 1,0에 존재하지만 복셀당 검색 길이가 줄어들기 때문에 복제가 필요합니다. 구성하는 동안 입자의 적용된 하위 셀은 입자의 영향 상자(주황색)를 래스터화하여 결정됩니다.

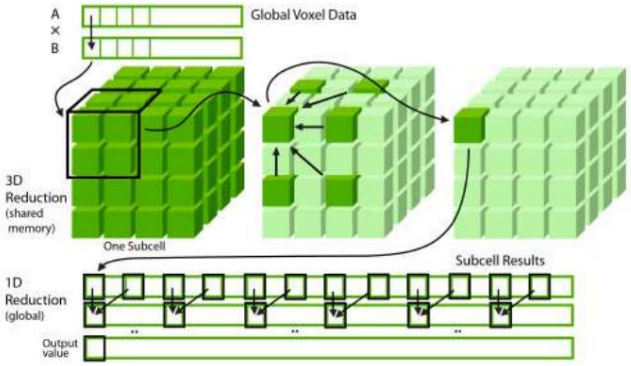

그림 5: 볼륨 축소에 의한 3D 내적: CG 솔버의 내적을 계산하기 위해 두 채널의 전역 복셀 데이터가 공유 메모리에 곱해집니다. 그런 다음 쓰기 위해 공유 메모리의 3D 축소 복셀
2의 거듭제곱에 의해 1D 어레이에 대한 ˙
서브셀당 하나의 값이 4에 적용됩니다. 마지막으로 단일 출력 값을 반환하기 위해 하위 셀 결과에 대한 1D 감소가 수행됩니다.

원자를 위해. 순진한 기술은 모든 입자에 대한 각 복셀의 거리를 확인하는 반면, 앞서 설명한 하위 셀 분할은 국소화된 고정 반경 수집을 위한 입자의 이상적인 공유 목록입니다.

모든 활성 복셀에 대해 커널이 시작되며 각 하위 셀은 GPU 그리드 블록입니다. 일반적인 수집은 일부 속성의 합을 수행하기 위해 각 입자에 한 번 액세스합니다. 렌더링을 위한 레벨 설정 값을 생성하기 위해 속도와 밀도를 수집하기 위해 여러 래스터화 커널을 구현합니다. 또한 속도 및 유체 공기 마커와 같은 많은 수집 작업이 단일 커널에 결합됩니다.

8. GPU에서 행렬이 없는 켤레 기울기 솔버

압력 해석은 비압축성 유체 시뮬레이션의 가장 중요한 구성 요소입니다. 그 목표는 선형 시스템 Ax = b를 푸는 것입니다. 여기서 x는 알 수 없는 압력 값이고, b는 발산이며, A는 고체, 유체 또는 비어 있을 수 있는 복셀 속성에만 의존하는 희소 행렬입니다. 행렬 A는 7점 라플라시안 행렬입니다. 행렬 A는 양의 정부호 대칭이므로 Conjugate Gradient(CG) 알고리즘은 가장 일반적으로 사용되는 기법 중 하나입니다. 스파스 트리 계층 구조에 직접 상주하는 스파스 복셀 그리드에서 FLIP 시뮬레이션을 위한 병렬, 매트릭스 없는 CG 솔버를 소개합니다.

알고리즘 2는 행렬이 없는 CG 솔버에 대한 의사 코드를 제공합니다. 각 CG 반복에는 하나의 SpMV(10행), 2개의 내적(11 및 15행) 및 3개의 벡터 추가(12, 13 및 17행)가 필요합니다. 벡터 r, b, q, d 및 x는 별도의 3D GVDB 텍스처에 채널로 저장됩니다. 벡터 d의 에이프런 복셀은 이웃 복셀에 액세스하는 SpMV(라인 10)에서 사용되기 전에 업데이트되어야 합니다 . 솔버 종료 기준은 파이프라인 지연을 줄이기 위해 10회 반복마다 CPU로 다시 읽어들이는 잔여 표준을 사용하여 확인됩니다(20행). 최종 입자 이류는 입자 위치에서 복셀 데이터의 간단한 샘플링입니다.

FLIP 기술에 따라 이전 및 새 그리드 속도를 샘플링하고 빼서 새 입자 속도를 제공합니다.
GVDB 복셀은 하드웨어 보간을 지원하여 임의의 위치에서 속도 값을 검색합니다.

CG 솔버에서 가장 비용이 많이 드는 연산은 SpMV(Sparse Matrix Vector Multiplication)입니다. 행렬 A는 N × N 크기의 2D 차원 행렬입니다. 여기서 N은 활성 유체 복셀의 총 수이므로 그리드 해상도와 함께 빠르게 증가합니다. 희소 행렬 라이브러리를 사용하더라도 행렬은 대규모 시스템의 경우 GPU 메모리를 빠르게 소모합니다. 따라서 행렬 없는 접근 방식 [MGSS13] 에 따라 다음과 같이 구성하여 행렬 A를 저장하지 않습니다.

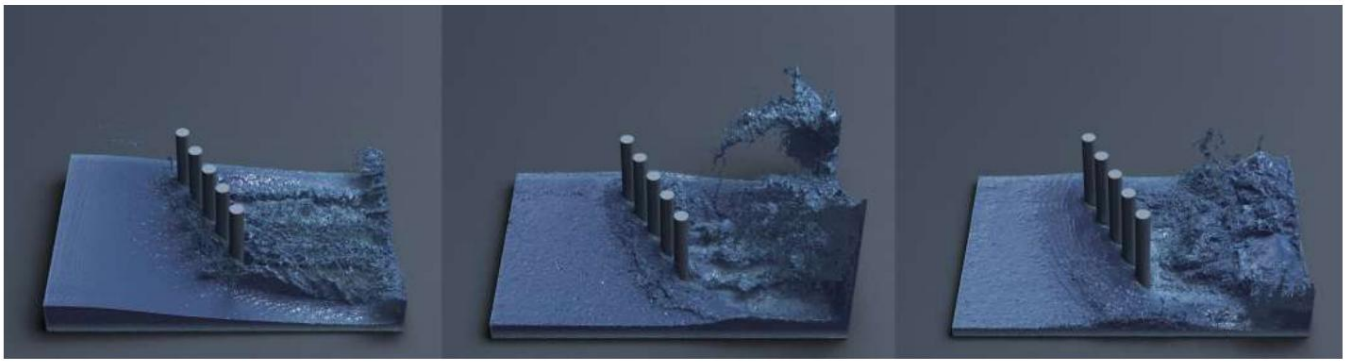

그림 6: 2,900만 개의 입자와 450×300×300의 격자 크기가 있는 열 장면. 시뮬레이션은 Quadro GP100에서 1e 4 CG 솔버 엡실론 으로 프레임당 2.4초가 걸립니다 . 두 배의 그리드 해상도에서 NVIDIAR GVDB 복셀 및 OptiX로 렌더링되었습니다 .

필요에 따라 즉석에서 직접 복셀의 행렬 값. 행렬 A의 각 행은 최대 7개의 항목만 0이 아닌 하나의 압력 방정식에 해당합니다. 대각선 값은 비고체 이웃 복셀의 수에 해당하고 나머지 6개 값은 6개의 이웃 복셀 속성에 따라 달라집니다. 벽돌과 같은 위치에 있는 에이프런 복셀로 인해 나무를 순회하지 않고 직접 텍스처 가져오기로 바로 이웃에 액세스할 수 있습니다.

SpMV를 계산하기 위해 복셀당 하나의 스레드가 시작되고 6개의 인접한 복셀 값을 확인하여 입력 벡터와 행렬 A의 행 사이의 내적을 계산합니다. 결과는 출력 벡터에 직접 기록됩니다. 예를 들어, 행 i에서 벡터 b와 A의 행렬-벡터 곱셈을 계산하기 위해 voxel i(x, y,z)의 6개 이웃을 검사하여 스칼라 s0, s1, s2, s3, s4 및 s5 를 검색 합니다 . 복셀에서 (x ± 1, y,z), (x, y ± 1,z), (x, y,z ± 1).

GVDB 앞치마 복셀로 인해 이웃은 복셀 i와 동일한 브릭에 함께 위치한 GPU 텍스처 캐시에서 이미 사용할 수 있습니다.

결과 벡터 c에 대해 우리는 ci = bi Σ 0 sn Σ를 계산합니다. δ0 억. 하는 동안 세계 공간과 아틀라스 공간 간의 매핑은 GVDB 헬퍼 함수를 사용하여 일정한 시간에 수행할 수 있습니다. 아틀라스 공간. 결과 값은 다른 출력 채널에 직접 기록됩니다.

벡터 추가는 두 채널에서 GVDB 컴퓨팅 커널을 사용하여 간단하지만 CG 솔버에서 잔차를 계산하는 데 사용되는 두 벡터의 내적은 복잡한 작업입니다.

내부 제품에 대해 3차원 병렬 축소를 수행합니다. 하나의 서브셀에 벡터 a와 b가 주어지면 각 스레드는 ai 와 bi를 곱하고 43개의 공유 메모리 에 씁니다 . 그런 다음 그림 5와 같이 공유 메모리에서 3D 축소가 수행됩니다. 8개 복셀의 각 하위 그룹은 각 반복에서 하나의 값으로 축소됩니다. 3D 축소 단계는 전역 2D 배열에 기록된 하나의 누적 값으로 끝납니다. 최종 단계에서는 각 서브셀의 결과에 대해 유사한 1D 배열 축소를 수행하여 최종 제품 값을 얻습니다.

9. 결과

우리의 결과는 FLIP의 CPU 구현에 비해 시뮬레이션 시간이 크게 향상되었음을 보여줍니다. 다양한 규모의 다양한 장면에서 테스트를 수행합니다. 성능은 토폴로지 재구축, GPU 및 CPU 모두에 대한 시뮬레이션 실험, 메모리 사용량에 대해 측정됩니다. CPU 성능은 4코어(8스레드)에서 측정됩니다.

알고리즘 2 행렬 없는 켤레 구배 솔버

```
1: 주어진 입력 발산 b, 시작 값 x, 최대 반복
    imax 및 오류 허용오차 ε 2: 절차
MatrixFreeConjugateGradientSolver(b, x,imax, ε) 3: 4: 5: 6: 7: 8: 9: 10: 11:
        i = 0 r
        = b    SpMV(x) d = r
        δnew
        = InnerProduct(r,r) δ0 = δnew
        while i <
        imax do
            UpdateApron(d) q
12:         = SpMV(d) α =
13:         δnew/InnerProduct(d,q) x = x+αd
14:         r = b    αq
15:         δold = δnew
            δnew =
            InnerProduct(r,r) β = δnew/
16:         δold d = r + βd
17:         i = i+1 if i
18:         mod 10
19:         == 0 then
20:             δnew > ε 2δ0 이면          확인을 위해 CPU로 다시 읽기
21:                 솔버를 중지하고 다음과 같은 경우 종료를 반
22:             환합니다.
23:         종료
24:     동안 종료
25: 종료 절차
```

32GB 메모리 및 GPU 성능을 갖춘 Intel Core i7 6700K 4.0GHz는 16GB GPU 메모리를 갖춘 NVIDIA Quadro GP100에서 측정되었습니다.

9.1. 토폴로지 구성

4M, 8M, 16M, 32M 및 64M 입자를 사용하여 토폴로지 구성의 성능을 그림 8의 댐 붕괴 예와 함께 측정합니다. 전체 CPU, 전체 GPU 및 증분 GPU 재구축의 성능은 표 1에 나와 있습니다. GPU에서 전체 토폴로지 재구축은 CPU 재구축보다 7~20배 빠르며 증분 재구축은 원래 CPU 재구축보다 80~180배 빠릅니다.

더 중요한 점은 증분 토폴로지 업데이트에 필요한 전체 토폴로지 빌드에 비해 2% 미만의 메모리 사용량

업데이트 목록이 훨씬 작기 때문에 GPU. 따라서 시뮬레이션에서 전체 GPU 토폴로지 빌드는 첫 번째 프레임에서만 사용된 다음 메모리가 해제되어 이후 작업에 사용할 수 있으므로 더 큰 시뮬레이션이 가능합니다.

표 1: CPU의 전체 빌드, GPU의 전체 빌드 및 한 프레임 동안 GPU의 증분(밀리초 단위)에 대한 동적 토폴로지의 성능. 데이터는 댐 붕괴 예제로 측정됩니다.

입자 수	CPU 전체	GPU 전체	GPU 증분	384	56(7×)	4.8	807	96(8×)	5.6
4M	1598	204(8×)	9.6	3249	(80×)				
800만	313(10×)	18.7	6368	366(17×)	(144×)				
16M	35.5				(167×)				
32M					(174×)				
6500만					(179×)				

9.2. 시뮬레이션 성능

우리는 GPU 시뮬레이션의 성능을 세 가지 다른 멀티스레드 CPU 구현과 비교합니다. 처음 두 가지는 성능 향상을 위해 불완전한 Cholesky 선조건자가 있거나 없는 고밀도 그리드를 사용합니다(고밀도 CPU CG/PCG). 세 번째 비교는 활성 복셀(Sparse CPU CG)을 집계하여 희소성을 달성하는 CPU의 CG 솔버에 대한 것입니다. GVDB의 브릭 기반 스토리지와 다르지만 시스템 크기가 비슷하므로 CG 솔버 성능을 조사하기 위해 이를 사용합니다.

이 세 솔버는 서로 다른 그리드 크기와 입자 수를 사용하여 비교적 간단한 두 장면에서 테스트되었습니다(그림 8 및 9 참조).
표 2에서 볼 수 있듯이 투영 단계는 CPU 기반 스파스 CG 솔버보다 최대 10배 더 빠릅니다. 모든 테스트에서 CG 허용 오차로 1e-4를 사용합니다. 1/3의 반복이 필요한 고밀도 CPU PCG와 비교할 때 우리 방법은 여전히 10배에서 28배 더 빠릅니다. 성능이 그리드 구조의 희소성에 크게 의존하는 Sparse CPU CG 솔버와 비교할 때 GPU 솔버는 여전히 6배에서 10배의 속도 향상을 달성합니다. 앞으로 우리는 병렬 하드웨어에 적합한 선조건자를 조사하여 성능을 더욱 향상시키려고 합니다.

표 3 은 시뮬레이션의 각 단계에 대한 타이밍을 보여줍니다.
증분 방식을 사용하면 토폴로지 구축 시간이 전체 시간의 3% 미만입니다. CG 솔버는 여전히 가장 비싼 부분으로, SpMV용 커널 시작, 내부 제품 2개, 벡터 추가 3개, 각 반복마다 앞치마 업데이트가 필요합니다. 하드웨어 삼선형 보간을 사용하면 최종 이류 단계에 총 시간의 약 1%가 소요됩니다. 우리는 성능이 공간의 입자 분포에 크게 의존한다는 것을 발견했습니다. 물이 튀거나 흩어지면 더 많은 벽돌이 필요하므로 사용되지 않은(공기) 복셀과 채워진(유체) 복셀의 비율이 증가합니다. 브릭 내부에 하나의 유체 복셀이 있는 한 브릭의 모든 복셀을 처리해야 합니다. 이는 토폴로지 재구축 및 업데이트 시간이 증가하는 대신 GVDB 브릭 크기를 줄임으로써 다소 완화될 수 있습니다.

9.3. 메모리 사용량

GPU에서 처리할 수 있는 시뮬레이션의 크기를 결정하는 시뮬레이션의 최대 메모리 사용량을 측정합니다. 처럼
섹션 6에서 설명한 것처럼 하위 셀 목록은 시뮬레이션 중에 가장 많은 메모리를 차지합니다. 다음으로 가장 많이 사용되는 메모리는 복셀 채널입니다.

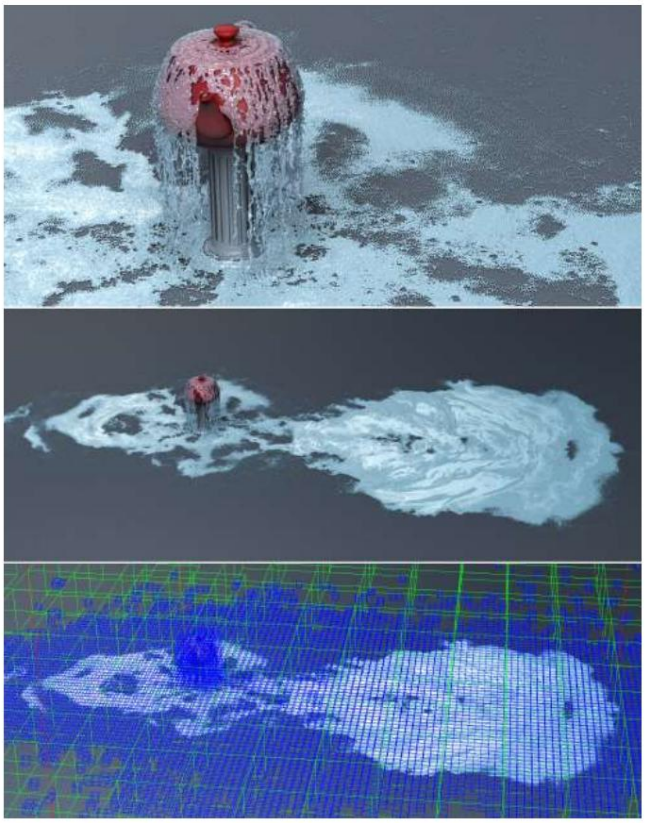

그림 7: 찻주전자 장면: 2백만 개의 입자로 확장되고 희박한 do main의 시뮬레이션. 영역은 최대 3360 × 160 × 2272에 도달하며 희박한 벽돌 indi = 32768 복셀에서 유체가 완전히 시뮬레이션됩니다. 시뮬레이션 소요 $5^{분}$ 파란색으로 표시됩니다. 각 브릭은)
(Quadro GP100에서 프레임당 21초이며, 브릭당 미사용 복셀의 비율이 높기 때문에 2M 파티클에 대한 일반적인 효율보다 낮습니다.

데이터. 모든 복셀 데이터는 13개의 GVDB 채널(밀도, 유체 마커, x, y, z 방향의 신규 및 기존 속도 구성 요소, 발산, 압력, CG 솔버에서 사용하는 벡터용 추가 채널 2개)에 저장됩니다. 단일 채널은 상대적으로 작지만 2,000만 개의 입자 장면의 경우 약 200MB이지만 13개 채널 모두 시뮬레이션 메모리의 거의 절반을 차지합니다. 채널 데이터는 유체를 덮는 벽돌을 할당하기 위해 동적으로 크기가 조정되는 3D 텍스처에 저장됩니다.

전체 토폴로지 빌드에는 정렬을 위한 긴 수준 인덱스 목록과 정렬 결과, 마커 목록 및 접두사 합계 목록을 저장하기 위한 동일한 길이의 다른 목록이 필요합니다. 그러나 전체 토폴로지 빌드 후 임시 메모리를 해제하여 GVDB 채널 데이터와 같은 다른 용도로 사용할 수 있습니다. 필요에 따라 정렬된 레벨 인덱스 목록, 마커 목록 및 접두사 합계 목록의 크기를 조정하지만 현재 레벨 인덱스 목록의 크기를 조정하지 않습니다. 목록 길이를 계산하기 위해 추가 패스가 필요하여 기본 목록이 아닌 동안 성능이 저하되기 때문입니다. 메모리 병목 현상. 전체 토폴로지 빌드에는 표 4의 증분 토폴로지 업데이트보다 2배 더 많은 메모리가 필요 하지만 증분 토폴로지 업데이트의 실제 사용량은 전체 재구축의 2% 미만입니다.

표 2: CPU, 선조건자가 있는 CPU 및 GPU에서 Matrix-Free Conjugate Gradient 솔버의 프레임당 평균 시간.

		밀도 높은 CPU CG		고밀도 CPU PCG	스파스 CPU CG 입자 #	스파스 GPU CG(GVDB)
	도메인 Iter. # 이터. 시간 총 Iter. # 이터. 시간 총 Iter. 시간 총 Iter. 시간 총 속도 향상 (ms) (ms) (ms) (ms) (ms) (ms) (ms) 100 22900 120 5870 8.3 1901 0.9 206 334 122578 370 28551 27.1 9946 3.1					
	1138 841 88959 69.7 358614 7 58614 1392 3.4 826 0.6 146 80 7444 12.0 4200 1.7 595 163 18240 29.1 13270 3.5 1596					
댐 브레이크	8M	2563	229	49		9×
	27M	3843	367	77		9×
	65M	5123	511	792 404712 14 105		10×
물방울	4M	1283	243	3402 50 17500 64		6×
	13.5M	1923	350	110 50160 92		7×
	32M	2563	456	111		8배

표 3: 시뮬레이션의 각 단계에 대한 프레임당 평균 시간. 모든 시간은 밀리초입니다. 우리의 GPU 솔루션은 도메인 그리드를 드물게 점유하는 반면, 두 개의 CPU 솔루션은 복셀로 도메인을 커버합니다. 따라서 성능 향상의 일부는 희소성으로 인한 작업 감소 때문입니다. 찻주전자 및 흐름 장면의 경우 도메인 크기는 무제한 시뮬레이션의 최대 범위입니다. 각 벽돌은 323입니다 .

	입자 #	도메인 범위 2563	# 벽돌 애비뉴/피크	CPU 프레임당 총 GPU 시간(ms)(PCG Topo 포함). Particle- CG Solve Advect Others Update to-Voxel 34 120 298						GPU 합계 프레임당 속도 향상(ms) 258 1299	
댐 브레이크	800만	3843	128 / 207	6892	6		206	4	8	3861	27×
	2700만	5123		32050	16		1138	14	11		25×
	6500만	1283		101131	36		3475	34	18		26×
물방울	4M	1923		1858	5	18	146	2	8	179	10×
	14M	2563		9067	12	56	595	7	10	680	13×
	32M		265 / 512	22080	19	144	1596	17	11	1787	12×
열	29M 450 × 300 × 300		551 / 1006	-	23	138	2260	10	12	2443	
도시	22M 512 × 256 × 512		468 / 1033	-	17	112	1721	7	11	1868	
찻주전자	2M 3360 × 360 × 2272 800 / 2976			-	20	8	983	2	70	1083	
흐름	74M 1056 × 288 × 768		665 / 812	-	68	270	3377	35	40	3790	

9.4. 표현

레이트레이싱은 NVIDIA R OptiX와 GVDB Voxels 통합을 사용하여 시뮬레이션 그리드 해상도의 두 배로 Quadro GP100에서 수행되었습니다 . Monte-Carlo 레이 트레이싱을 사용하여 하나의 기본, 하나의 반사, 두 개의 굴절 및 하나의 그림자 광선으로 각 장면을 렌더링합니다. 이 5개의 광선은 각 프레임이 픽셀당 128 샘플로 렌더링된 하나의 샘플을 구성합니다.

시뮬레이션된 물은 폴리곤으로 변환하지 않고 GVDB 복셀을 사용하여 스파스 볼륨으로 직접 레이 트레이싱되는 반면, 충돌 객체는 폴리곤 모델로 레이 트레이싱됩니다. 둘 다 하이브리드 다각형 볼륨 분산 및 음영 처리를 위해 OptiX를 사용합니다. 파티클은 레이트레이싱을 위해 설정된 협대역 레벨로 수집되며, 이를 위해서는 파티클 반경 외부의 값을 포함하도록 영향 상자를 확장해야 합니다. 일반적인 프레임은 128 샘플/픽셀로 1920×1080 해상도에서 10~30초 사이에 렌더링됩니다.

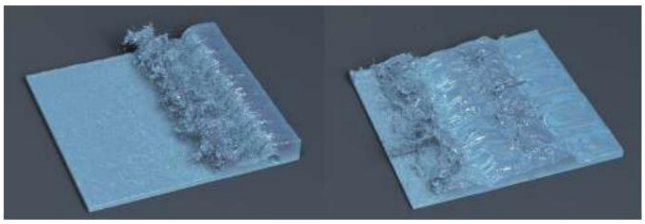

그림 8: 댐 붕괴: 2563 3843 및 5123 에서 8백만, 2천 7백만 및 6천 5백만 개의 입자로 시뮬레이션된 테스트 장면 . 2563에서의 시뮬레이션이 표시됩니다.

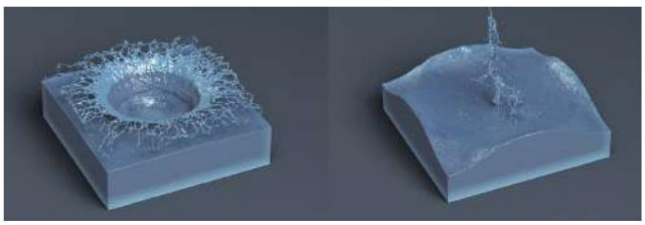

그림 9: 물방울: 1283 1923 및 2563 에서 4, 1,4, 3,200만 개의 입자로 시뮬레이션된 테스트 장면 . 2563에서의 시뮬레이션이 표시됩니다.

10. 결론 및 향후 작업

가상으로 제한되지 않은 영역의 유체에 대한 희소하고 효율적인 GPU 기반 FLIP 시뮬레이션을 제시했습니다. 우리의 방법은 희소 트리 토폴로지를 구축하고, 트리를 점진적으로 업데이트하고, 입자를 하위 셀로 효율적으로 정렬하고, 수백만 개의 입자에 대해 빠르고 매트릭스가 없는 Conjugate Gradient 솔버를 사용하여 FLIP 시뮬레이션 단계를 수행합니다. 전반적으로 유사한 CPU 기반 솔루션보다 최대 10배 빠른 시뮬레이션 시간을 달성할 수 있습니다.

드물게 배치된 브릭에서 작동하고 GPU 하드웨어에 최적화된 매트릭스 없는 CG 솔버를 설명합니다. 희소 행렬 A를 저장하는 대신 필요에 따라 즉시 복셀 값을 직접 검사하여 메모리 사용량을 크게 줄이고

GPU 메모리에서 훨씬 더 큰 시뮬레이션. 가속화된 솔버 작업에는 내부 제품에 대한 빠른 3D 축소, 이웃에 대한 효율적인 앞치마 업데이트 및 하드웨어 보간 입자 이류가 포함됩니다. 우리 시스템은 최신 GPU 하드웨어에서 프레임당 몇 초만에 최대 7,400만 개의 입자를 시뮬레이션할 수 있습니다.

우리 방법의 성능은 이미 상당히

표 4: 최대 메모리 사용량(MB). 객체는 충돌 지오메트리에 대한 복셀 그리드를 나타냅니다.

	입자 #	GVDB 토폴로지 (메가바이트)	GVDB 채널 토폴로지 (MB)	하위 셀 목록 작성 (메가바이트)	입자 데이터 (MB)	개체 합계 (MB)	(MB) 180 - 1233
댐 브레이크	800만	트) 0.06	592	120 305 445 1120 992		664 - 4283 1488 - 9708	
	2700만		1999	2530			
	6500만		4664				
물방울	4M		592	60	148	88	- 900 - 1640 -
	14M		592	206	520	308	3636 56 4583
	32M	0.08	1111	498	1258	746	
열	29M	0.09	2221	441	1120	662	
도시	22M	0.11	1851	346	876	518	46 3762 50
찻주전자	2M	0.78	2591	43	70	43	3231 - 8931
흐름	74M	0.14	2517	1127	2818	1691	

CPU 기반 솔버보다 높기 때문에 몇 가지 잠재적인 향후 개선이 예상됩니다. McAdams et al.과 유사한 멀티 그리드 기법과 같은 GPU 친화적인 프리 컨디셔너를 도입하는 것이 가능합니다. [MST10] 이지만 스파스 도메인에서 구현됩니다. 또 다른 가능한 개선 사항은 협대역 FLIP을 구현하여 입자 수를 줄이는 것입니다 [FAW* 16]. 입자 삽입 및 하위 셀로의 분류는 현재 성능 비용이 가장 높기 때문입니다. 마지막으로, GVDB 토폴로지는 공간이 트리 경계를 따라 분할될 수 있으므로 코어 외부 접근 방식에 자연스럽게 적합합니다.

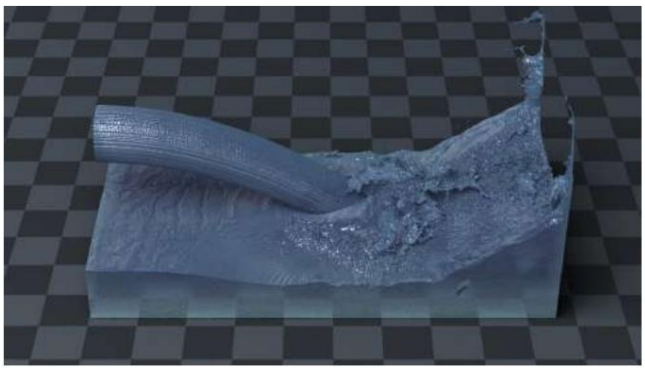

그림 10: 흐름: 도메인 1056×288×768에서 74M 입자의 시뮬레이션

향후 작업에서 몇 가지 제한 사항을 극복할 수 있습니다. 충돌은 현재 다각형 장애물 모델을 별도의 GVDB 개체로 복셀화하여 처리되며, 이는 단단한 경계 복셀을 식별하기 위해 다른 희소 그리드에서 샘플링됩니다. 복셀화할 필요 없이 이 충돌 그리드의 변환 매트릭스(로컬 참조 프레임)를 업데이트하고 경계 조건에 대한 레벨 세트를 도입하여 움직이는 강체 객체를 지원할 계획입니다.

렌더링과 관련하여 우리는 표면 폴리곤화 및 스무딩을 통해 품질을 낮추고 직접 체적 레벨 세트 레이트레이싱을 사용하여 미세한 디테일을 보존했습니다. 앞으로 우리는 현재 보이지 않는 고립된 입자를 식별하고 렌더링하려고 합니다.

레이트레이싱은 매끄러움을 위해 3선형 그래디언트를 사용하는 반면, 자유 표면 Dirichlet 경계 조건을 2차 정확도 [GFCK02] 까지 적용하지 않기 때문에 계단식 아티팩트가 보일 수 있습니다(그림 10 참조) .

우리가 아는 한 GPU에서 완전히 실행되는 최초의 공간적으로 희소하고 완전한 FLIP 솔버를 도입했습니다. FLIP 시뮬레이션 파이프라인의 모든 부분은 저장, 컴퓨팅 및 렌더링을 위해 GVDB Voxels를 사용하여 GPU 하드웨어에서 병렬 실행에 최적화되었으며, 성능이 크게 향상되는 동시에 더 큰 시뮬레이션을 처리하기 위해 메모리를 줄입니다.

감사의 말

저자는 특히 Ken Museth(Weta Digital), Gergely Klar(Dreamworks Animation) 및 Tristan Lorach(NVIDIA)의 지원과 지도에 감사드립니다. 이 작업은 NSF 보조금 #1538593에 의해 부분적으로 지원되었습니다.

참조

[ABO16] AZEVEDO VC, BATTY C., OLIVEIRA MM: 얇은 장애물과 좁은 틈이 있는 유체 흐름에 대한 기하학 및 토폴로지 보존. ACM 트랜스. 그래프. 35, 4(2016년 7월), 97:1–97:12. 2

[AGL*17] AANJANEYA M., GAO M., LIU H., BATTY C., SIFAKIS E.: 고해상도 적응성 액체에 대한 전원 다이어그램 및 희소 페이징 그리드. ACM 트랜스. 그래프. 36, 4(2017년 7월), 140:1–140:12. 2

[ATW13] ANDO R., THÜREY N., WOJTAN C.: 4면체 메시에 대한 적응력이 뛰어난 액체 시뮬레이션. ACM 트랜스. 그래프. 32, 4(2013년 7월), 103:1–103:10. 2

[ATW15] ANDO R., THÜREY N., WOJTAN C.: 액체 시뮬레이션을 위한 치수 감소 압력 솔버. 컴퓨팅 그래프. 포럼 34, 2(2015년 5월), 473–480. 2

[BB08] BATTY C., BRIDSON R.: 버클링, 코일링 및 회전하는 액체를 위한 정확한 점성 자유 표면. SCA 회보(2008), SCA '08, Eurographics Association, pp. 219–228. 2

[BB12] BOYD L., BRIDSON R.: 활발한 2상 유체 시뮬레이션을 위한 멀티플립. ACM 트랜스. 그래프. 31, 2(2012년 4월), 16:1–16:12. 2 [BBAW15] BAILEY D., BIDDLE H.,

AVRAMOUSSIS N., WARNER M.: openvdb를 사용하여 액체 배포. ACM SIGGRAPH 2015 Talks(New York, NY, USA, 2015), SIGGRAPH '15, ACM, pp. 44:1–44:1. 2

[BBB07] BATTY C., BERTAILS F., BRIDSON R.: 정확한 고체-유체 결합을 위한 빠른 변형 프레임워크. ACM 트랜스. 그래프. 26, 3(2007년 7월). 2

[BR86] BRACKBILL JU, RUPPEL HM: 플립: 2차원 유체 흐름의 적응 구역화된 셀 내 입자 계산 방법. J.컴퓨트 물리학 65, 2(1986년 8월), 314–343. 2

[CKIW15] CHEN Z., KIM B., ITO D., WANG H.: Wetbrush: 강모 수준의 GPU 기반 3D 페인팅 시뮬레이션. ACM 트랜스. 그래프. 34, 6(2015년 10월), 200:1–200:11. 2

[CM11a] CHENTANEZ N., MÜLLER M.: 솔리드 경계 조건 분리를 처리하는 멀티그리드 유체 압력 솔버. SCA 회보(2011), SCA '11, ACM, pp. 83–90. 2

[CM11b] CHENTANEZ N., MÜLLER M.: 제한된 키가 큰 셀 그리드를 사용한 실시간 오일러 물 시뮬레이션. ACM 트랜스. 그래프. 30, 4(2011년 7월), 82:1–82:10. 2 [CM12] CHENTANEZ N., MÜLLER M.: 대량 보존 오일러 액체 시뮬레이션. SCA 회보(2012), SCA '12, Eurographics Association, pp. 245–254. 2

[CZY17] CHU J., ZAFAR NB, YANG X.: 확장 가능한 병렬 유체 시뮬레이션을 위한 슈어 보완 사전 조건자. ACM 트랜스. 그래프. 36, 5(2017년 7월), 163:1–163:11. 2

[DBD16] DAVIET G., BERTAILS-DESCOUBES F.: 세분화된 재료의 연속체 시뮬레이션을 위한 반암시적 재료점 방법.
ACM 트랜스. 그래프. 35, 4(2016년 7월), 102:1–102:13. 2

[EB14] EDWARDS E., BRIDSON R.: 거친 격자가 있는 상세한 물: 표면 메시와 적응형 불연속 갈러킨 결합. ACM 트랜스. 그래프. 33, 4(2014년 7월), 136:1–136:9. 2

[FAW*16] FERSTL F., ANDO R., WOJTAN C., WESTERMANN R., THUEREY N.: 액체 시뮬레이션을 위한 협대역 FLIP. 컴퓨터 그래픽 포럼(Proc. Eurographics) 35, 2(2016), 225–232. 2, 10 [GB13] GERSZEWSKI D., BARGTEIL AW: 대규모 튀는 액체의 물리 기반 애니메이션. ACM 트랜스. 그래프. 32, 6(2013년 11월), 185:1–185:6. 2

[GFCK02] GIBOU F., FEDKIW RP, CHENG L.-T., KANG M.: 불규칙 영역에서 푸아송 방정식의 2차 정확도 대칭 이산화. J.컴퓨트 물리학 176, 1(2002년 2월), 205–227. 10

[GNS 12] GOLAS A., NARAIN R., SEWALL J., KRAJCEVSKI P., DUBEY P., LIN M.: 속도-와도 영역 분해를 사용한 대규모 유체 시뮬레이션. ACM 트랜스. 그래프. 31, 6(2012년 11월), 148:1–148:9. 2

[Har64] HARLOW FH: 유체 역학을 위한 셀 내 입자 컴퓨팅 방법. 전산 물리학 방법 3(1964), 319–343. 2

[Har05] HARRIS M.: GPU에서 빠른 유체 역학 시뮬레이션. ACM SIGGRAPH 2005 과정(뉴욕, 뉴욕, 미국, 2005), SIG GRAPH '05, ACM. 2

[Har07] HARRIS M.: cuda에서 병렬 축소 최적화. 프레젠테이션
CUDA 툴킷(2007)과 함께 제공됩니다. 4

[HG09] HORVATH C., GEIGER W.: GPU에서 지시 가능한 고해상도 화재 시뮬레이션. ACM 트랜스. 그래프. 28, 3(2009년 7월), 41:1–41:8. 2

[Hoe16] HOETZLEIN RK: GVDB: GPU에서 희소 복셀 데이터베이스 구조를 레이트레이싱합니다. 고성능 그래픽 절차(2016), HPG '16, Eurographics Association, pp. 109–117. 1, 2, 5

[HSO07] HARRIS M., SENGUPTA S., OWENS JD: CUDA를 사용한 병렬 접두사 합계(스캔). GPU Gems 3에서 Nguyen H., (Ed.). Addison Wesley, 2007년 8월, ch. 39, pp. 851-876. 4

[JGT17] JIANG C., GAST T., TERAN J.: 천, 니트 및 모발 마찰 접촉에 대한 이방성 탄성가소성. ACM 트랜스. 그래프. 36, 4(2017년 7월), 152:1–152:14. 2

[JSS 15] JIANG C., SCHROEDER C., SELLE A., TERAN J., STOM AKHIN A.: 아핀 세포 내 입자 방법. ACM 트랜스. 그래프. 34, 4(2015년 7월), 51:1–51:10. 2

[KBT 17] KLÁR G., BUDSBERG J., TITUS M., JONES S., MUSETH K.: 생산 준비 mpm 시뮬레이션. ACM SIGGRAPH 2017 Talks(New York, NY, USA, 2017), SIGGRAPH '17, ACM, pp. 42:1–42:2. 5 [KCC 06] 김지 , 차도 , 창비 , 구비 , 임이 : 격렬하게 튀는 물의 실용 애니메이션. SCA 회보(2006), SCA '06, Eurographics Association, pp. 335–344. 2 [KGP*16] KLÁR G., GAST T., PRADHANA A., FU C., SCHROEDER C., JIANG C., TERAN J.: 모래 애니메이션에 대한 Drucker-prager 탄성가소성. ACM 트랜스. 그래프. 35, 4(2016년 7월), 103:1–103:12. 2

[KLLR07] KIM B., LIU Y., LLAMAS I., ROSSIGNAC J.: 소산 및 확산이 상당히 감소된 이류. 시각화 및 컴퓨터 그래픽에 관한 IEEE 트랜잭션 13, 1(2007년 1월), 135–144. 2

[LCPF12] LENTINE M., CONG M., PATKAR S., FEDKIW R.: 매우 큰 시간 간격으로 자유 표면 흐름을 시뮬레이션합니다. SCA 회보(2012), SCA '12, Eurographics Association, pp. 107–116. 2

[LMAS16] LIU H., MITCHELL N., AANJANEYA M., SIFAKIS E.: 이기종 컴퓨팅 플랫폼을 위한 확장 가능한 슈어 보완 유체 솔버입니다. ACM 트랜스. 그래프. 35, 6(2016년 11월), 201:1–201:12. 2

[LZF10] LENTINE M., ZHENG W., FEDKIW R.: 성긴 그리드 투영만을 사용하는 비압축성 유동에 대한 새로운 알고리즘. ACM 트랜스.
그래프. 29, 4(2010년 7월), 114:1–114:9. 2

[MCP*09] MULLEN P., CRANE K., PAVLOV D., TONG Y., DESBRUN M.: 유체 애니메이션을 위한 에너지 보존 적분기. ACM 트랜스.
그래프. 28, 3(2009년 7월), 38:1–38:8. 2

[MCPN08] MOLEMAKER J., COHEN JM, PATEL S., NOH J.: 애니메이션용 저점도 유동 시뮬레이션. SCA 회보(2008), SCA '08, Eurographics Association, pp. 9–18. 2

[MGSS13] MÜLLER E., GUO X., SCHEICHL R., SHI S.: 비등방성 타원 pdes에 대한 사전 조정된 켤레 기울기 솔버의 행렬 없는 GPU 구현. 컴퓨팅 비스. 과학. 16, 2(2013년 4월), 41–58. 6

[MST10] MCADAMS A., SIFAKIS E., TERAN J.: 대형 그리드의 유체 시뮬레이션을 위한 병렬 다중 그리드 포아송 솔버입니다. SCA 회보(2010), SCA '10, Eurographics Association, pp. 65–74. 2, 10

[MSW*09] MCADAMS A., SELLE A., WARD K., SIFAKIS E., TERAN J.: 직모의 디테일 보존 연속체 시뮬레이션. ACM 트랜스.
그래프. 28, 3(2009년 7월), 62:1–62:6. 2

[Mus13] MUSETH K.: VDB: 동적 토폴로지가 있는 고해상도 스파스 볼륨. ACM 트랜스. 그래프. 32, 3(2013년 7월), 27:1–27:22. 1, 2

[NGL10] NARAIN R., GOLAS A., LIN MC: 양방향 솔리드 커플링이 있는 자유 유동성 입상 재료. ACM 트랜스. 그래프. 29, 6(2010년 12월), 173:1–173:10. 2

[NSCL08] NARAIN R., SEWALL J., CARLSON M., LIN MC: 에너지 전달 및 절차적 합성을 사용하는 난기류의 빠른 애니메이션.
ACM 트랜스. 그래프. 27, 5(2008년 12월), 166:1–166:8. 2

[PTC 10] PFAFF T., THUEREY N., COHEN J., TARIQ S., GROSS M.: 이방성 난류 입자를 사용한 확장 가능한 유체 시뮬레이션. ACM 트랜스. 그래프. 29, 6(2010년 12월), 174:1–174:8. 2

[RGJ*15] RAM D., GAST TF, JIANG C., SCHROEDER C., STOM AKHIN A., TERAN J., KAVEHPOUR P.: 점탄성 유체, 폼 및 스폰지에 대한 재료 포인트 방법. In Symposium on Computer Animation (2015), ACM, pp. 157–163. 2

[SABS14] SETALURI R., AANJANEYA M., BAUER S., SIFAKIS E.: Sp 그리드: 적응형 연기 시뮬레이션에 적용된 스파스 페이징 그리드 구조.
ACM 트랜스. 그래프. 33, 6(2014년 11월), 205:1–205:12. 2

[SKK07] 송 오유, 김동일 , 고현석 : 유체의 미세한 움직임을 시뮬레이션하기 위한 파생 입자. 시각화 및 컴퓨터 그래픽에 관한 IEEE 트랜잭션 13, 4(2007년 7월), 711–719. 2

[SSC 13] STOMAKHIN A., SCHROEDER C., CHAI L., TERAN J., SELLE A.: 눈 시뮬레이션을 위한 재료 포인트 방법. ACM 트랜스.
그래프. 32, 4(2013년 7월), 102:1–102:10. 2

[WST09] WICKE M., STANTON M., TREUILLE A.: 유체 역학을 위한 모듈 기반. ACM 트랜스. 그래프. 28, 3(2009년 7월), 39:1–39:8. 2 [YSB*15] YUE Y., SMITH B., BATTY C., ZHENG C., GRINSPUN E.: Continuum foam: 전단 의존 흐름을 위한 재료 점법.
ACM 트랜스. 그래프. 34, 5(2015년 11월), 160:1–160:20. 2

[ZB05] ZHU Y., BRIDSON R.: 유체로서의 모래 애니메이션. ACM 트랜스.
그래프. 24, 3(2005년 7월), 965–972. 1, 2

[ZBG15] ZHANG X., BRIDSON R., GREIF C.: 대류-투영 유체 솔버에서 누락된 소용돌이도 복원. ACM 트랜스. 그래프. 34, 4(2015년 7월), 52:1–52:8. 2

[ZLC 13] ZHU B., LU W., CONG M., KIM B., FEDKIW R.: 도메인 확장을 위한 새로운 그리드 구조. ACM 트랜스. 그래프. 32, 4(2013년 7월), 63:1–63:12. 2