물리 기반 시점 의존 액체 애니메이션 *

김장희°†, 임인성‡, 차득현‡

†한국전자통신연구원, ‡서강대학교 컴퓨터학과

rolldice@etri.re.kr, ihm@sogang.ac.kr, seaboy7@grmanet.sogang.ac.kr

Physically-Based View-Dependent Liquid Animation

Janghee Kim°†, Insung Ihm‡, Deukhyun Cha‡

†Electronics and Telecommunications Research Institute

‡Dept. of Computer Science, Sogang University

요 약

계산 유체 역학 분야에서는 유체 시뮬레이션 계산에 있어 계산 시간과 컴퓨터 메모리의 한계를 뛰어 넘는 유효 해상도를 달성하기 위하여 다양한 형태의 적응적 메쉬 기법들이 제시되어 왔다. 특히 최근에 컴퓨터 그래픽스 분야에서는 팔진 트리 기반의 메쉬 구조를 사용하여 중요 지역에 높은 해상도를 적용하려는 유체 애니메이션 방법이 제시되었다 [1]. 본 논문에서는 계산 시간과 메모리 사용량을 보다 절약하기 위해, 이러한 적응적 방법을 확장하여 카메라의 특성을 이용하여 보이는 지역에 상대적으로 높은 해상도의 메쉬를 적용해주는 시점 의존 방법을 제시한다. 이와 함께 시뮬레이션 과정에서 동적으로 변하는 메쉬 구조를 효율적으로 구현하기 위하여 기존의 팔진 트리와는 다른, 단순한 형태의 가변 메쉬 구조를 제시한다. 또한 실제 구현을 통하여 본 논문이 제시하는 시점 의존 기법이 유체 시뮬레이션 결과의 질을 비교적 잘 유지하면서, 계산에 필요한 자원을 효과적으로 줄일 수 있다는 사실을 보이도록 한다.

1. 서 론

최근 사실적인 유체 애니메이션 연구와 관련한 많은 성과에도 불구하고, 높은 해상도를 적용한 유체 시뮬레이션은 여전히 해결해야 할 과제로 남아있다. 높은 해상도는 보다 상세한 결과를 얻기 위해 반드시 필요하므로 그 필요성은 계속 증가하고 있다. 하지만, 삼차원 공간상에서 해상도를 높이면 격자의 수는 세제곱에 비례해서 늘어나므로 해상도는 쉽게 증가시킬 수 없다는 것이 문제이다.

적응적 그리드 (adaptive grid) 기법은 이러한 문제를 해결하는데 있어 유용한 방법의 하나로서, 최근 컴퓨터 그래픽스 분야에서는 적응적 그리드를 사용하여 물이나 연기를 세밀하게 시뮬레이션해주는 기법이 제시되었다 [1]. 이 연구에서는 좀 더 세밀한 표현이 필요한 지역에 보다 높은 해상도를 적용하기 위해 팔진 트리 기반의 구조가 사용되었다. 그리고, Navier-Stokes 방정식을 팔진 트리 기반의 구조에서 풀어냄으로써 허용할 수 있는 계산 자원 안에서 높은 해상도의 시뮬레이션을 가능하게 하였다.

유체 시뮬레이션에 적응적 그리드를 사용하는 주된 목적은 보다 중요한 지역에 집중적으로 해상도를 높히는데 있다. 따라서 우리가 얼마만큼 중요한 지역을 잘 설정하는냐에 따라 그 효과가 크게 달라질 수 있게 된다. 연기의 시뮬레이션과는 다르게 액체의 표면을 추적하는 작업은 액체 시뮬레이션에서 매우 중요한 작업이다. 왜냐하면 우리가 최종적으로 보게 되는 결과는 물의 표면이고 이 표면의 상세함에 따라 마지막으로 얻게 되는 그림의 수준이 달라지기 때문이다. 따라서 우선적으로 물의 표면에 가장 높은 해상도를 적용해야 원하는 수준의 결과를 얻을 수 있게 된다.

애니메이션 제작을 위해 유체 시뮬레이션을 할 때, 카메라에 대한 정보는 계산 효율을 올리기 위한 또 하나의 수단이 될 수 있다. 예를 들어 넓은 바다를 돌아다니는 보트를 카메라가 쫓아가는 장면을 생각해보자. 우리가 보는 장면은 결국 카메라의 위치에 의해 결정되는 뷰잉 볼륨에 의해 결정이 되므로, 보이지 않는 지역에 많은 자원을 투자하는 것은 그다지 현명한 선택이 아닐 것이다. 이런 상황에서는 보이는 지역에 대한 해상도를 보다 집중적으로 높히는 형태로 계산을 하는 것이 계산 효율을 높이는 방법이 된다.

이 논문에서는 카메라 시점에 의존하는 적응적 격자를 사용한 유체 시뮬레이션이 얼마나 효과적으로 삼차원 애니메이션에 적용될 수 있는지에 대해 알아 본다. 이를 위해 [1]에서 제시한 적응적 시뮬레이션의 방법을 확장하여 뷰잉 볼륨을 고려하여 수행한 시뮬레이션이 상당한 양의 계산 시간과 메모리를 절약할 수 있다는 것을 보이려 한다. 물론, 이 방법은 계산의 정확도가 가장 중요한 척도가 되는 시뮬레이션에는 그다지 적합한 방법이 아니다. 뷰잉 볼륨을 적용하여 적응적 격자를 구성하게 되면 추가적인 수치적 오차가 발생하기 때문이다. 하지만 제시된 방법이 계산에 필요한 자원의 사용을 상당히 줄여주면서도 우리가 원하는 상세함을 얻을 수 있는 방법이기 때문에 컴퓨터 그래픽스 분야에서는 상당히 유용한 방법이 될 수 있을 것이다.

*본 연구는 정보통신부의 ITRC 사업의 지원으로 수행되었음.

2. 관련 연구

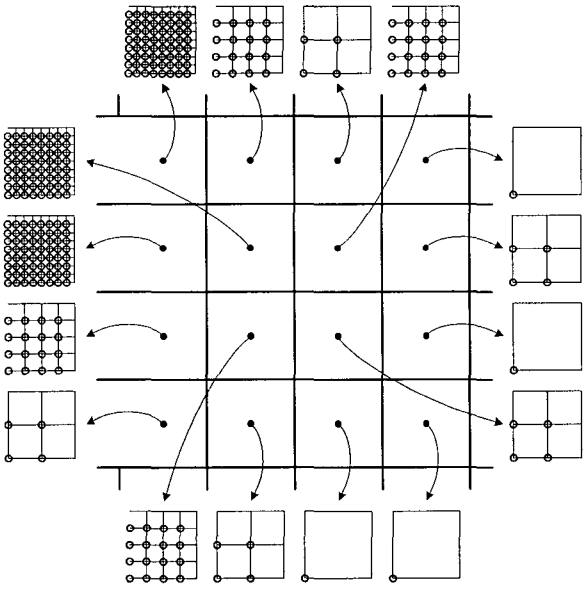
컴퓨터 그래픽스 분야에서 삼차원 공간에서 정의된 Navier-Stokes 방정식을 풀어낸 최초의 시도는 [2]에서 있었다. [3]에서는 [4]에서 제시한 semi-Lagrangian 방법을 이용하고 액체의 표면을 등위 집합 (level set) 기법을 이용해 추적하여 시뮬레이션을 보다 안정적이고 사실적으로 제작하는데 성공하였다. [5]에서는 [6]에서 제시한 입자 등위 집합 (particle level set) 기법을 액체 시뮬레이션에 적용하였고 이 새로운 방법은 복잡하게 형성되는 물의 표면을 보다 상세히 표현하는데 효과적으로 적용되었다. 또, 이런 연구들과 병행하여 많은 시뮬레이션 기술들이 연기나 가스 [7, 4, 8], 그리고 화염과 폭발 [9, 10, 11, 12]을 시뮬레이션하기 위해 개발되었다.

이런 연구들 덕분에 다양한 유체 효과에 대한 사실적인 애니메이션 제작이 가능해졌다. 하지만 그럼에도 고해상도의 시뮬레이션을 해야 할 경우 상당히 많은, 때로는 시뮬레이션이 불가능한 계산 시간과 메모리가 필요하기 때문에 원하는 해상도의 시뮬레이션을 하지 못하는 문제가 남아있다. 계산에 필요한 자원을 효율적으로 사용하기 위해 [13]에서는 큰 규모의 연기와 같은 현상의 시뮬레이션을 높은 해상도의 이차원 속도장을 그보다 낮은 해상도의 삼차원 속도장에 적용하였다. [1]에서는 팔진 트리 기반의 적응적 격자 구조를 사용해 보다 효율적으로 시뮬레이션을 하는 방법을 제시하였다. 이와는 조금 다르게 [14]에서는 부력을 가진 기체에 대한 시뮬레이션에 대해서 기체의 이동에 따라 고정된 격자의 실제 위치를 조정해 연기의 지속적인 변화를 얻어내는 방법을 제시하였다.

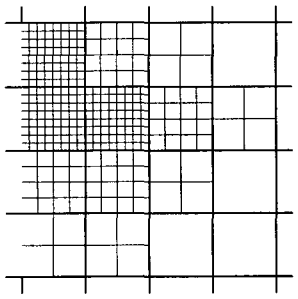
이런 결과들이 상당히 좋은 결과를 내었음에도, 유체 시뮬레이션에서의 적응적 격자의 사용은 계산에 필요한 자원의 한계를 뛰어넘어 부분적인 유체의 상세함을 증가시키려 하는 유체 시뮬레이션에서는 여전히 중요한 연구과제이다. [15]를 시작으로 여러 다양한 적응적인 데이터 구조와 계산 방법이 유체 시뮬레이션을 위해 개발되어 왔다 [16, 17, 18, 19]. 시점 의존 적응적 계산법이 컴퓨터 그래픽스 분야, 특히 [20, 21, 22]에서 연구된 다면체 모델의 단순화 작업에서 상당히 유용하게 사용되어 왔음에도 이를 유체 시뮬레이션에 적용하려는 시도는 아직 없었다. 사실 이 접근법은 유체 시뮬레이션에서는 그다지 적합하지 않을 수 있고 수치 계산의 정확성을 해칠 수 있어 부적절한 결과를 초래할 수 있다. 하지만, 컴퓨터 애니메이션 분야에서는 비슷한 결과를 얻기 위해 약간의 정확성의 희생하는 것이 종종 허용되므로 시점 기반의 계산 방법을 유체 시뮬레이션에 적용하는 것은 시도해 볼만한 일이라 할 수 있다.

3. 효율적인 적응적 격자의 구현

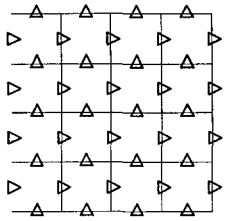
적절한 격자 구조를 설계하는 것은 계산상의 효율을 결정하는 중요한 요소이기 때문에 적응적 시뮬레이션에서 매우 중요한 일이다. 이것을 위해 동적으로 변하는 유체의 표면을 표현하기 위한 팔진 트리의 몇 가지 변형이 제시되었다 [16, 18, 19, 1]. 이런 계층적인 구조의 가장 큰 장점은 필요한 부분만 정확하게 해상도를 높힐 수 있다는 것이다. 하지만, 설계가 제대로 이루어지지 않는다면 구조를 유지하는데 상당히 많은 비용이 들고, 원하는 노드를 접근하기 위한 절차가 복잡해 결과적으로 계산 시간이 늘어난다는 단점이 있다. 이 방법은 유체 시뮬레이션에 들어가는 방대한 양의 계산 시간과 메모리를 최적화하기 위해 사용하는 것이기 때문에 적응적 격자의 설계에 대한 최적화는 대단히 중요한 문제이다.



(a) 꼭지점 할당 데이터를 위한 메모리 할당



(b) 적응적 메쉬 분할의 예



(c) 레벨 2의 벽면 속도가 메모리에 할당된 모습

그림 1: 적응적 격자의 구조. 시점 의존 시뮬레이션에 효과적으로 적용될 수 있도록 제작되었다. 전체 공간은 기반 격자로 구성되고 그 기반 격자는 메모리 포인터와 레벨 지시자로 구성된다. 매우 단순하기 때문에 구현에 큰 무리가 없다.

이 논문에서는 시점 의존 적응적 격자 적용의 구현이 효율적으로 이루어도록 하기 위해 매우 단순한 적응적 격자 구조의 사용을 제안한다. 이 방법은, 예를 들면 [15, 23, 24]에서 사용한 구조와 비슷하다. 그림 1은 제안한 계층적 격자 구조에 대해 보여준다. 우선 삼차원 상의 전체 계산 공간을 레벨 0이라 칭하는 가장 큰 격자를 이용해 구성한다 (이를 기반 격자라 함). 그리고, 정의된 레벨에 맞추어 균일하게 구성된 상세 격자가 기반 격자 위에 놓이게 된다. 레벨의 점진적인 변화를 위해 이웃한 6개의 기반 격자의 레벨 차는 많아야 1이 되도록 한다 (16개의 기반 격자로 구성된 적응적 격자의 구성 예제인 그림 1(b) 참조). 이 구조는 구조를 유지하는데 들어가는 메모리가 기반 격자의 갯수만큼의 포인터 배열밖에 없

어 그 양이 상당히 작고, 원하는 지역의 데이터에 접근하기 위해 두 번의 메모리 참조만 하면 되므로 계산 시간을 단축할 수 있다. 하지만, 기반 격자의 영역 내에서는 모두 같은 레벨이 적용되므로 불필요한 지역의 레벨을 지나치게 높힐 수 있다는 약점이 있다.

그림 1(a)는 제시한 적응적 격자 구조의 기본적 구조를 보여준다. 기반 격자의 레벨이 결정되고 나면 그것에 해당하는 크기의 메모리가 현재의 기반 격자에 할당되어 포인터로 연결된다. 이 각 꼭지점에 값을 저장하기 위해 레벨-i에 대해 $2^i \times 2^i \times 2^i$ 크기의 메모리가 할당된다. 우측 맨 끝의 값은 데이터의 중복성을 피하기 위해 할당하지 않는다.

압력 부분을 풀기 위해 꼭지점에 정의된 속도들은 격자의 벽면으로 임시적으로 가져오게 된다 (그림 1(c)). 이 때 필요한 메모리는 레벨-i에 대해 $2^{i+1} \times 2^{i+1} \times 2^{i+1}$ 크기의 메모리가 필요하게 된다. 물론 같은 레벨의 기반 격자가 연속적으로 존재할 경우 데이터의 중복성은 피할 수 없지만 압력을 풀기 위해 임시로 사용하는 메모리이므로 구현상의 무리는 거의 없었다.

지금까지 제안한 적응적 격자 구조에 대해 살펴보았다. 이 구조는 단순히 레벨 지시자가 있고 그 레벨에 맞는 메모리를 할당하는 형태로 되어 있다. 물론 이 구조는 기반 격자 내에서의 레벨이 세부적으로 나누어지지 않아 불필요한 지역의 레벨을 지나치게 높일 수 있다는 단점이 있다. 하지만, 이러한 단점에도 불구하고 구조를 유지하는데 들어가는 메모리의 양이 상대적으로 적고, 구조가 단순하여 구현에 무리가 없으며, 임의의 지점으로의 접근이 항상 $O(1)$의 시간 복잡도로 접근 가능하다는 장점은 시뮬레이션의 효율성에 상당한 이점이 된다. 이 구조의 단점을 경험적으로 어느 정도 극복할 수 있다고 생각할 때 이 구조는 시뮬레이션의 효율성을 상당히 증진시킬 수 있다.

4. 시점 의존 메쉬 분할

적응적 격자를 사용하는 중요한 목표는 관심있는 지역을 어떻게 찾아내어 그곳에 계산 자원을 집중하는가이다. 물의 표면이 지속적으로 변하고 이것을 계속 추적한다면, 물의 표면이 우리의 관심지역이 되고 이 지역에 자원을 집중해야 한다. 이러한 전략은 시뮬레이션의 정확성을 유지하면서도 상당히 많은 양의 계산 시간과 메모리를 줄일 수 있다는 것을 이미 이전 연구에서 알 수 있었다.

만약 유체 시뮬레이션이 컴퓨터 애니메이션 제작에 사용된다면 또다른 요소를 계산 효율성 증대를 위해 사용할 수 있다. 특정 장면을 제작할 때 사용되는 뷰잉 볼륨이 바로 그것이다. 만약 뷰잉 볼륨이 전체 공간 중 일부분만 차지하고 있다면 우리의 관심지역은 바로 그 공간으로 제한될 수 있다. 따라서 뷰잉 볼륨이 차지하는 공간에 보다 많은 자원을 집중하게 되면 보다 높은 계산 효율을 얻을 수 있을 것이다. 하지만, 뷰잉 볼륨 바깥의 유체도 뷰잉 볼륨 안쪽의 유체에 영향을 주기 때문에 그 영향이 최소화되는 수준에서 해상도를 결정해야 한다.

시점 의존 적응적 메쉬 분할을 사용할 경우 일반적으로 고려해야 할 사항들은 다음과 같이 정리할 수 있다.

- 먼저 보이는 지역의 해상도는 그렇지 않은 지역의 해상도에 비해 높아야 한다.
- 둘째로, 물이 있는 지역의 해상도는 기본적으로 높아야 한다. 그리고, 보이는 지역에서의 물의 표면이 있는 지역은 가장 높은 해상도를 사용한다.
- 셋째로, 물의 표면이 있는 지역은 비록 뷰잉 볼륨 바깥에 있다 하더라도 어느 정도의 해상도를 보장해 주어야 한다. 여기서의 해상도가 지나치게 낮으면 압력을 계산할 때 오차가 누적되어 전체적인 유체의 흐름이 잘못 계산되게 된다.
- 넷째로, 인접한 기반 격자 간의 레벨 차이는 많아야 1이 되도록 한다.
- 마지막으로, 뷰잉 볼륨의 길이가 상당히 길다면 멀리 있는 지역의 해상도를 가까이 보이는 지역의 해상도보다 낮출 수 있다.

이 고려사항들은 모든 문제에 동일하게 적용되는 것은 아니고 시뮬레이션에 따라 적용 정도를 적절히 달리 해야 한다. 우선 문제의 특성을 파악하고 그에 맞는 적용 범위를 결정하는 것이 안정적인 시뮬레이션에 꼭 필요하다.

5. 실험 결과

시점 의존 적응적 시뮬레이션의 효율성을 검증하기 위해, [6]에서 소개된 입자 등위 집합 기법을 구현하였다. 성능을 비교하기 위해 시뮬레이터의 내부는 3장에서 소개한 적응적 격자를 사용하도록 확장하였다. 이것을 수행할 때 [1]에서 소개한 근사화 방법으로 압력 부분을 풀 때 생성되는 선형방정식의 대칭성을 유지하였다.

이 시뮬레이션은 크게 두 가지 부분으로 나누어진다. 먼저 Navier-Stokes 방정식을 통해 속도장을 시간에 따라 진행시키고 이렇게 얻어진 속도장을 이용해 입자 등위 집합 기법에서 액체의 표면을 시간에 따라 움직인다. 속도장을 시간에 따라 진행시킬 때 가장 많은 메모리와 계산 시간을 소모하는 부분이 바로 압력 부분을 풀 때이다. 압력 부분을 풀면서 생성되는 거대한 선형방정식을 구성하기 위해 많은 메모리가 필요하기 때문이다. 시점 의존 적응적 격자를 사용하면 이런 속도장을 수정하는데 두 가지 이점이 있다. 먼저 전체 속도장과 그에 관련한 속성을 저장하는데 들어가는 메모리를 절약할 수 있다. 다음으로, 이 속도장에 따라 생성되는 선형방정식의 크기가 줄어들게 되어 사용되는 메모리와 계산 시간을 줄일 수 있게 된다.

시점 의존 적응적 격자를 사용하게 되면 [6]에서 제시한 입자 등위 집합 기법을 수행할 때에도 이점을 얻을 수 있다. 예를 들면 표면 주위에 뿌리는 입자들을 수를 보이는 지역과 그렇지 않은 지역에 따라 차등하여 생성할 수 있다는 것이다. 보이는 지역에 입자를 좀 더 많이 입자를 뿌리고 그렇지 않은 지역엔 상대적으로 적게 뿌린다면 계산 자원을 절약하면서도 우리가 원하는 효과는 얻을 수 있게 된다. 하지만, 본 논문에서는 시점 의존 적응적 격자의 효과를 보다 정확히 측정하기 위해 속도장의 수정, 특히 압력 부분을 계산할 때 필요한 계산 자원에 보다 초점을 맞춰서 비교하였다.

5.1 테스트 장면 I

그림 4는 같은 장면을 각각 다른 시뮬레이션 방법으로 제작

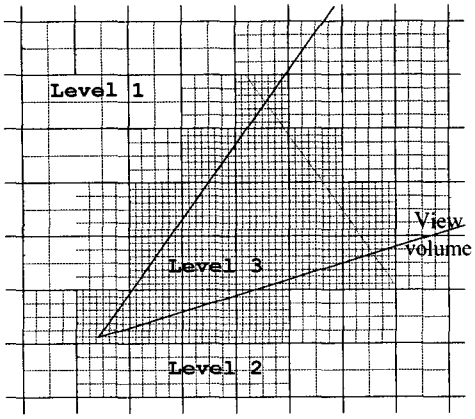


그림 2: 표면에 대한 시점 의존 메쉬 분할. 카메라에 가까이 있고 뷰잉 볼륨 안에 있는 기반 격자는 가장 높은 해상도의 레벨을 갖는다 (예를 들어 3). 그리고, 주변 격자들은 점차 낮은 레벨을 부여받는다. 물 표면에 지나치게 낮은 레벨을 부여하는 것을 방지하기 위해 물 표면에 대한 최소 레벨을 정한다 (예를 들어 1).

(*time*: sec., *memory*: MB)

	time	*order*	*nonzero*	*memory*
UNIFORM	127.2	1,973,740	7,813,844	291.08
ADAPTIVE	14.9	331,303	1,314,983	48.97
V-ADAPTIVE	8.6	178,029	704,032	26.24

표 1: 각 시뮬레이션 결과에 대한 측정치 비교. 그림 4에 소개된 장면을 각 시뮬레이션 방법으로 수행한 결과이다. *time*은 한 프레임의 속도장을 수정하기 위해 걸린 시간이다. 그리고, *order*와 *nonzero*는 압력 부분에서 생성된 선형방정식의 크기와 0아닌 원소의 갯수이고, *memory*는 이 선형 방정식을 풀기 위해 필요한 모든 종류의 메모리의 크기를 나타낸 것이다. 이 값들은 모두 430 프레임의 시뮬레이션을 모두 한 이후에 평균값을 낸 것이다.

한 결과를 보여준다. 먼저 UNIFORM은 고정되고 균일한 격자를 사용하여 $160 \times 200 \times 320$의 해상도에서 계산된 방법이다. 그에 반해 나머지 방법들은 적응적 격자를 사용한 결과들이다. 우선 전체 공간을 기반 격자로 나누고 각각의 기반 격자의 최대 해상도는 $8 \times 8 \times 8$로 정의하여 계산하였다.

ADAPTIVE는 모든 물의 표면에 가장 높은 레벨(레벨 3)로 정의하여 시뮬레이션을 하였다. 우선 물의 표면에 가장 높은 레벨을 정의하고 그 이외의 지역은 점차 레벨을 낮춰 레벨 0까지 낮추도록 설계되었다.

한편 4절에서 소개된 시점 의존 적응적 격자 방법이 V-ADAPTIVE에서 사용되었다. 그림 2은 액체의 표면에 따른 메쉬의 분할에 대해 보여준다. 메쉬를 분할할 때 카메라로부터의 거리에 따라 뷰잉 볼륨을 둘 혹은 그 이상으로 나눈다. ADAPTIVE와는 다르게 V-ADAPTIVE에서는 카메라와 가까운 안쪽의 뷰잉 볼륨 지역의 물의 표면에서만 가장 높은 해상도를 할당한다. 그리고 다른 지역은 점진적으로 레벨을 낮춰간다. 여기서는 계산상의 오차를 줄이기 위해 물의 표면에 대한 최소 레벨을 1로 정의하여 계산하였다.

5.1.1 성능 평가

각각의 방법에 대한 계산 효율을 조사하기 위해, 다음과 같은 측정 척도를 사용하였다.

- Navier-Stokes 방정식을 풀어 전체 속도장을 수정하는데 걸리는 계산 시간 (*time*).
- 압력 계산 부분에서 구성되는 선형 방정식의 크기 (*order*). *order* $\times$ *order*의 크기를 가진 대칭형 선형방정식이 만들어진다.
- 선형 방정식 내에서 0이 아닌 원소의 갯수 (*nonzero*).
- 선형 방정식을 구성하고 이를 계산하는데 필요한 메모리 크기 (*memory*). 이 메모리에는 선형 방정식의 크기와 0이 아닌 원소의 갯수가 영향을 미치고 이를 풀기 위한 임시 메모리까지 모두 포함한다. 선형 방정식의 모든 계산은 double의 정밀도로 계산하였다.

표 1에는 관련 실험 결과가 요약되어 있다. 이 실험을 위해 2.8 GHz Intel Pentium 4 CPU와 2 GB RAM이 장착된 컴퓨터를 사용하였는데, 여기에 있는 값들은 430 프레임에 달하는 모든 프레임에 대하여 평균을 낸 값들이다. 이 결과를 보면 알 수 있듯이 UNIFORM은 ADAPTIVE에 비해 계산 시간이 약 8.5배 정도 차이가 나고 있음을 알 수 있다. 이 결과도 충분히 훌륭하지만, 여기에 V-ADAPTIVE 방법을 사용하면 UNIFORM에 비해 계산 시간이 약 14.8배 정도 차이가 나 ADAPTIVE보다도 더 많은 계산 시간을 절약하고 있음을 알 수 있다.

절약된 자원의 대부분은 압력 부분을 계산할 때 생성되는 선형 방정식의 크기와 밀접한 관련이 있다. UNIFORM의 방법으로 계산할 때엔 평균적인 선형 방정식의 크기가 무려 $1,973,740 \times 1,973,740$였던 것이 V-ADAPTIVE 방법을 사용할 땐 평균적으로 $178,029 \times 178,029$ 정도의 크기가 되었다. 다른 측정 기준들도 이와 비슷한 비율로 감소한 것을 알 수 있고, 선형 방정식의 크기는 전체적인 계산의 효율성을 측정하는데 상당히 중요한 요소임을 알 수 있다.

그림 3는 매 프레임마다 변화하는 성능 측정 요소(*time*과 *order*)를 그래프로 표현한 것이다. 여기서 주목할 점은 UNIFORM과 ADAPTIVE는 시간에 따른 변화가 별로 없지만 V-ADAPTIVE는 시간이 흐름에 따라 지속적으로 자원의 사용량이 줄어든다는 점이다. 이 현상의 원인은 명확하다. 이 실험에서, 우리는 세로 60도, 가로 80도의 뷰잉 볼륨을 사용하였는데, 초기에 카메라가 복도 왼편에 있을때엔 시야가 거의 전체에 닿아 있었지만 (그림 4의 세번째 줄 참고), 시간이 흐르면서 카메라 뒷 부분의 보이지 않는 영역이 점차 넓어졌고 그에 따라 계산에 필요한 자원이 절약되었기 때문에 이런 현상이 발생한 것이다.

5.1.2 결과 이미지의 비교

실험 결과에서 보듯이 상당한 양의 계산 자원의 절감이 있었으나, 적응적 격자를 사용하는 방법은 여전히 수치적인 오차를 담고 있기 때문에 그 오차들이 결과 이미지에 영향을 줄 수 있다. 하지만, 최종적으로 나온 렌더링 결과를 살펴보면 그러한 오차가 결과에 주는 영향은 삼차원 애니메이션의 제작에 있어 받아들일만한 정도임을 알 수 있다. 그림 4의 처

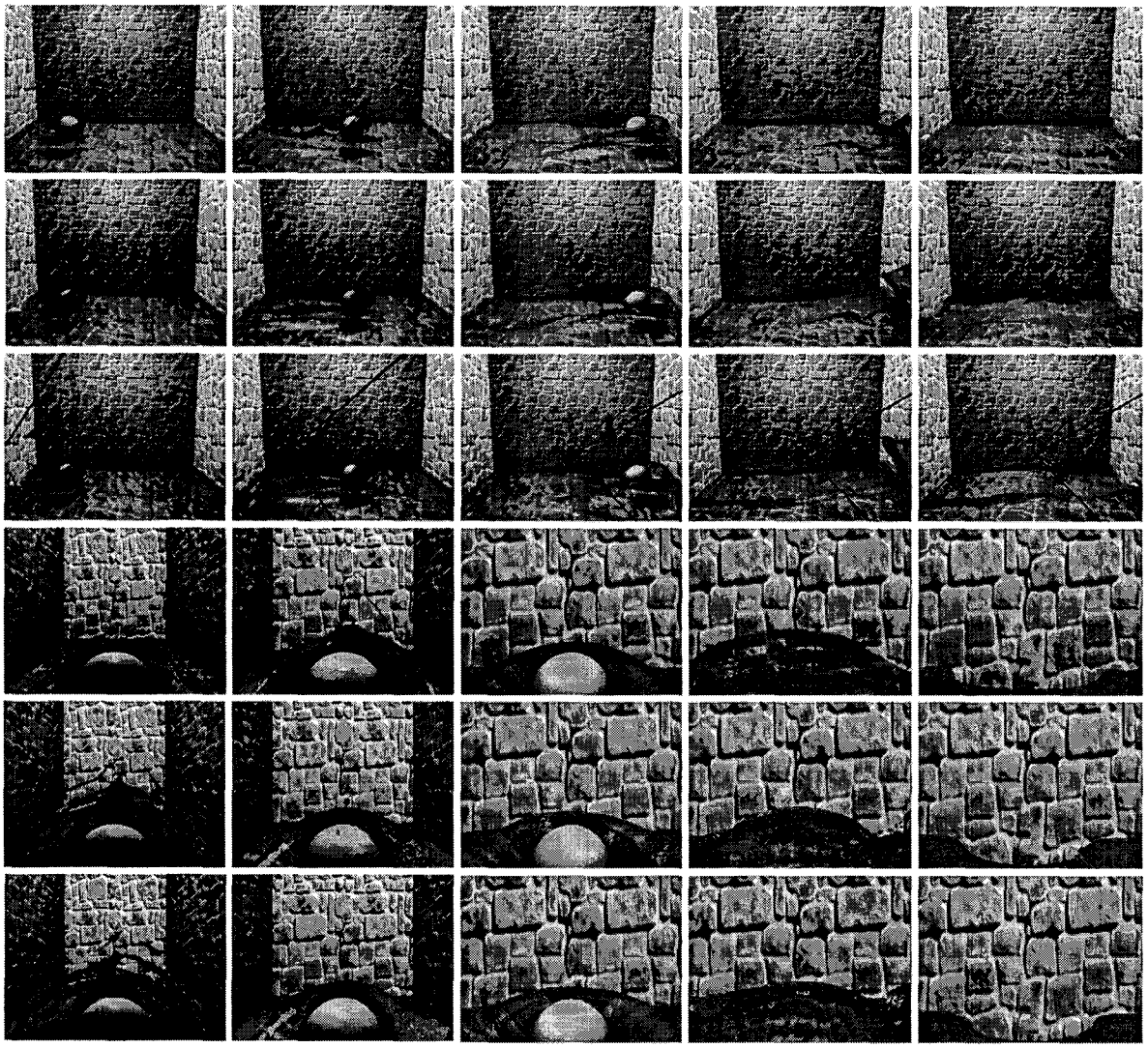
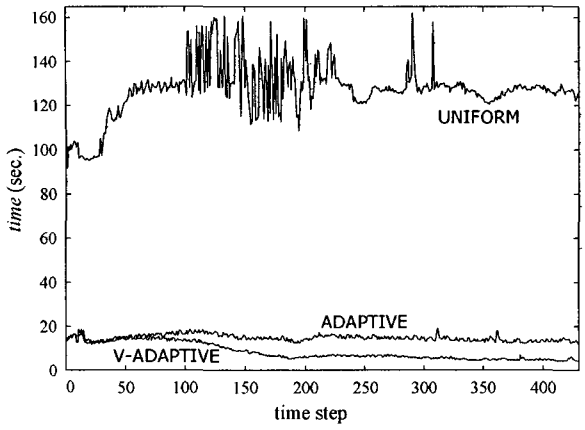
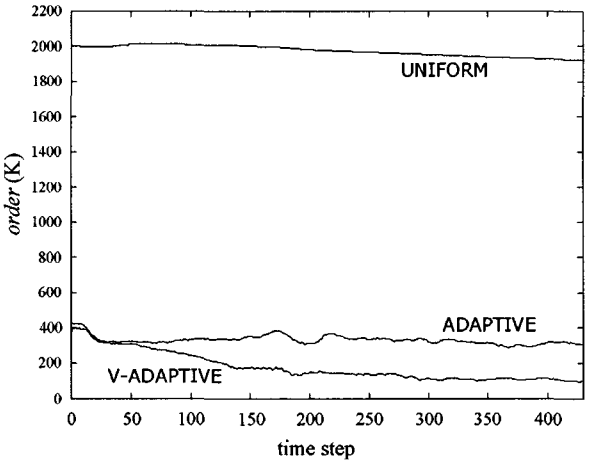


그림 4: 세 시뮬레이션 방법을 비교하기 위한 연속 장면 (프레임 번호: 60, 120, 180, 280, 400). 첫번째 세 행은 UNIFORM, ADAPTIVE, V-ADAPTIVE로 시뮬레이션된 장면의 전체를 보여주는 것이고, 나머지는 실제 카메라 위치에서 바라본 결과이다. 적용된 실제 해상도는 $160 \times 200 \times 320$이다.



(a) 속도장 계산 시간



(b) 선형 방정식의 크기

그림 3: 두 가지 성능 측정치에 대한 시간에 따른 변화. 예상대로 UNIFORM와 ADAPTIVE는 카메라의 움직임에 대해 성능이 변하지 않았지만, 뷰잉 볼륨에 들어오는 공간이 줄어들면서 V-ADAPTIVE는 성능이 지속적으로 향상되고 있음을 알 수 있다. 나머지 두 측정 기준인 *nonzero*와 *memory*도 *order*와 거의 비슷한 결과를 보인다.

음 세 줄에 보이는 연속 장면은 UNIFORM, ADAPTIVE, V-ADAPTIVE의 세 시뮬레이션 방법의 결과를 전체 장면으로 본 것이다. 시뮬레이션 방법이 바뀌면서 분명 액체 표면의 움직임이 오차에 의해 달라지는 것을 알 수 있다. V-ADAPTIVE의 방법으로 시뮬레이션한 결과는 그 오차가 뷰잉 볼륨 바깥의 부분에서 분명히 보인다. 하지만, 이 문제는 뷰잉 볼륨 바깥의 부분은 우리의 관심 지역에서 배제했기 때문에 발생하는 자연스러운 현상이다.

이러한 문제에도 불구하고, 시점 의존 적응적 시뮬레이션 방법은 분명 유용하게 사용될 수 있는 방법이다. 그림 4의 그 다음 세 줄의 연속 장면을 살펴보면 전체적으로는 유체 표면의 모양이 많이 달랐음에도 뷰잉 볼륨 안쪽의 유체 표면의 해상도에는 많은 차이가 없다. 따라서 이렇게 시점이 정해져 있는 장면의 시뮬레이션에서는 이러한 접근 방법이 상당히 유용함을 알 수 있다.

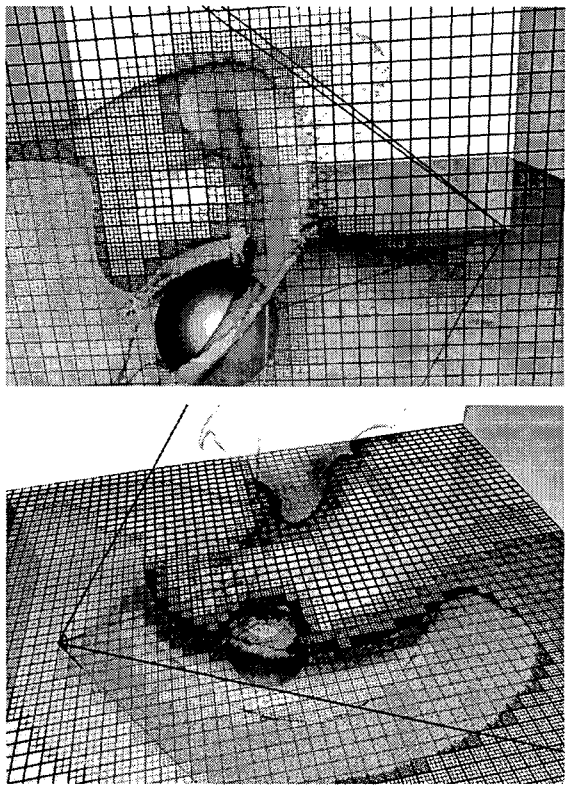


그림 5: 테스트 장면 II에 대한 격자 분할 예. 각 예제 장면에 대하여 사용 격자를 축에 수직인 단면으로 자른 예를 보여주고 있다.

5.2 테스트 장면 II

시점 의존 적응적 시뮬레이션의 효율성에 더 많은 분석을 하기 위해, 두 적응적 격자를 사용하는 방법을 이용해 $440 \times 280 \times 440$의 해상도를 이용한 시뮬레이션을 수행하였다 (그림 7 참조). 메쉬의 분할은 위의 테스트 장면과 동일한 방법을 사용하였고 (그림 5 참조), 이 실험에서 UNIFORM의 방법은 계산 자원이 가지고 있는 한계 때문에 실험에서 제외했다.

5.2.1 성능 평가

표 2의 처음 두 줄은 ADAPTIVE와 V-ADAPTIVE의 방법으로 시뮬레이션한 결과를 평균 내어 구한 값들이다. 그리고, V-ADAPTIVE에서 사용한 뷰잉 볼륨은 가로 80도, 세로 60도의 뷰잉 볼륨이다. 이 표에서 보면 알 수 있듯이, V-ADAPTIVE의 방법으로 계산한 결과가 ADAPTIVE에 비해 약 40% 정도 비용을 절감한 것을 알 수 있다. 이와 동시에 상당히 많은 양의 메모리를 선형 방정식을 풀 때 절감한 것도 알 수 있다.

표 2의 다음 두 줄에서 볼 수 있는 것처럼 카메라의 가시 지역 범위는 분명 위의 통계치에 영향을 준다. V-ADAPTIVE(4/5)와 V-ADAPTIVE(2/3)는 각각 처음의 뷰잉 볼륨에서 가장 상세하게 푸는 지역까지의 길이를 $\frac{4}{5}$와 $\frac{2}{3}$으로 줄여 계산한 것이다. 결과는 물론 줄어든 뷰잉 볼륨의 크기만

(*time*: sec., *memory*: MB)

	time	*order*	*nonzero*	*memory*
ADAPTIVE	55.9	1,230,087	4,889,637	181.87
V-ADAPTIVE	32.3	716,561	2,850,100	105.99
V-ADAPTIVE(4/5)	27.1	612,350	2,438,379	90.64
V-ADAPTIVE(2/3)	23.5	524,396	2,088,444	77.63

표 2: 각 시뮬레이션 결과에 대한 측정치 비교. 그림 8에서 보여준 장면을 두 시뮬레이션 방법으로 수행한 결과이다. 이 시뮬레이션에 사용된 실제적인 해상도는 440 × 280 × 440이다. 모든 측정 결과는 1000프레임을 시뮬레이션한 결과를 평균낸 것이다. 그리고, 가시 지역의 넓이가 수행 속도에 얼마만큼의 영향을 미치는지도 실험을 하였다. 마지막 두 행의 결과가 가장 높은 해상도를 사용하는 지역의 길이를 V-ADAPTIVE의 $\frac{4}{5}$와 $\frac{2}{3}$으로 줄여 실험한 결과이다.

큼 계산 시간과 사용 메모리도 줄어든 것을 알 수 있다. 하지만 뷰잉 볼륨이 지나치게 작으면 뷰잉 볼륨 내부의 해상도가 떨어지기 때문에 최종적으로 나온 결과 이미지의 질이 나빠질 수 있으므로 주의해야 한다.

5.2.2 결과 이미지 비교

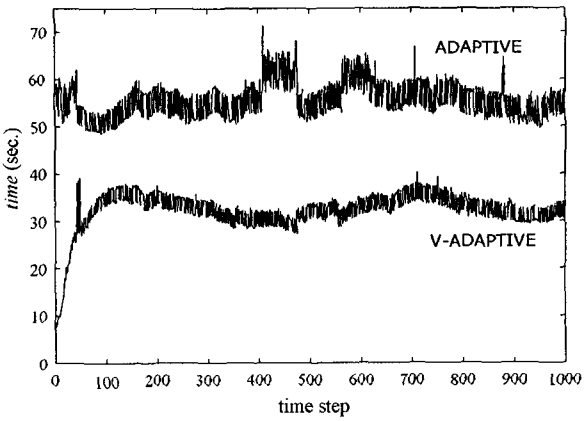
첫번째 테스트 장면에서 그 결과는 매우 만족스러웠다. 그림 7는 각각 두 방법으로 테스트 한 결과이다. 처음 절반은 나오는 것은 전체가 보이도록 한 것이고 나머지는 실제 카메라의 위치에서 바라본 장면이다. 두 시뮬레이션의 결과가 다르게 나온 것은 사실이지만, 실제 카메라의 위치에서의 그림은 그렇게 큰 차이가 나지 않는 것에 주목한다면 계산 시간과 메모리를 절약할 수 있는 V-ADAPTIVE의 방법을 선택하는 것이 현명할 것이다.

그림 8은 해상도의 차이가 얼마만큼 결과 이미지에 영향을 주는지를 알려주는 결과이다. 이 결과는 본래 카메라 위치에서 약간 뒤로 뺀 위치에서 렌더링되었다. 여기서 원래 카메라의 위치에서 보이지 않던 지역은 붉은 색을 강조하여 표현하였고 이곳에서의 물 표면의 레벨은 1로 제한된다. 그리고, 뷰잉 볼륨은 두 부분으로 나누어져 카메라 가까이 있는 지역으로 가장 높은 해상도(레벨 3)로 시뮬레이션 되는 지역은 특별한 표시를 하지 않았고, 카메라로부터 떨어진 지역으로 물 표면에서의 레벨이 2로 제한되는 지역을 녹색을 강조하여 표현하였다. 그림을 보면 가장 상세하게 풀이되는 지역은 물 표면의 상세함이 살아 있지만, 그 뒤의 붉은 색으로 표현된 지역은 그 상세함이 많이 사라진 것을 쉽게 알 수 있다.

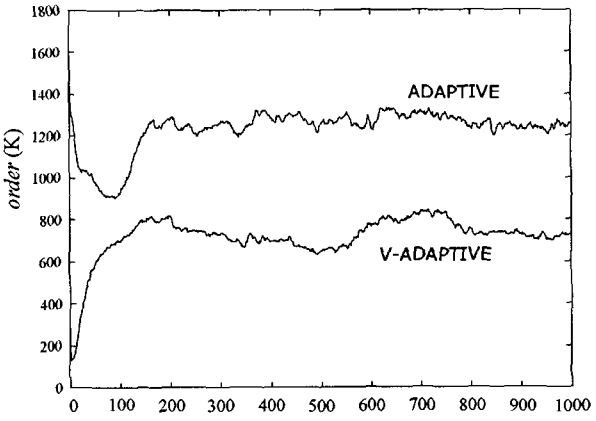
적응적 격자를 사용하여 시뮬레이션을 하게 되면 수치적 오차가 발생하고 시점 의존 적응적 격자를 사용하면 그 정도는 더 심해진다. 하지만, 실제 카메라 위치에서의 결과는 그다지 큰 차이가 없으므로 애니메이션을 제작함에 있어서는 그것이 그리 큰 문제가 되지 않는다. 게다가 이 방법은 상당히 많은 양의 계산량과 필요 메모리를 절약시켜 주기 때문에 충분히 활용할 가치가 있다.

6. 결론

물리 기반의 유체 시뮬레이션은 고해상도의 시뮬레이션이



(a) 속도장 계산 시간



(b) 선형 방정식의 크기

그림 6: 두 측정치의 시간에 따른 변화. 200프레임이 지나고 나면 뷰잉 볼륨이 완전히 시뮬레이션 영역으로 들어오게 된다. 이후로는 거의 일정하게 수치가 유지되고 있음을 알 수 있다.

힘들고 아직 연구 주제로 남아 있다. 이 논문에서는 간단히 시점 의존 적응적 격자를 사용해 보다 많은 계산 시간과 메모리를 절약하는 방법에 대해 소개하였다. 물론 그것을 사용하면서 발생하는 오차는 문제이지만, 그것이 애니메이션 제작에서는 받아들일 수 있는 오차임을 알 수 있었다. 몇가지 실험이 이를 증명하였고 이를 통해 상당히 많은 계산시간과 필요 메모리를 절약했음을 알 수 있었다. 그리고, 이 논문에서 제안한 간단하고 효율적인 적응적 격자 구조는 시점 의존 시뮬레이션 방법에 다른 구조보다 효과적으로 적용되었다. 같은 해상도가 정의되는 고정된 공간이 있다는 것이 뷰잉 볼륨에의 소속 여부를 쉽게 결정할 수 있도록 하기 때문이다.

참고 문헌

[1] F. Losasso, F. Gibou, and R. Fedkiw. Simulating water and smoke with an octree data structure. *ACM Transactions on Graphics (ACM SIGGRAPH 2004)*, 23(3):457–462, 2004.

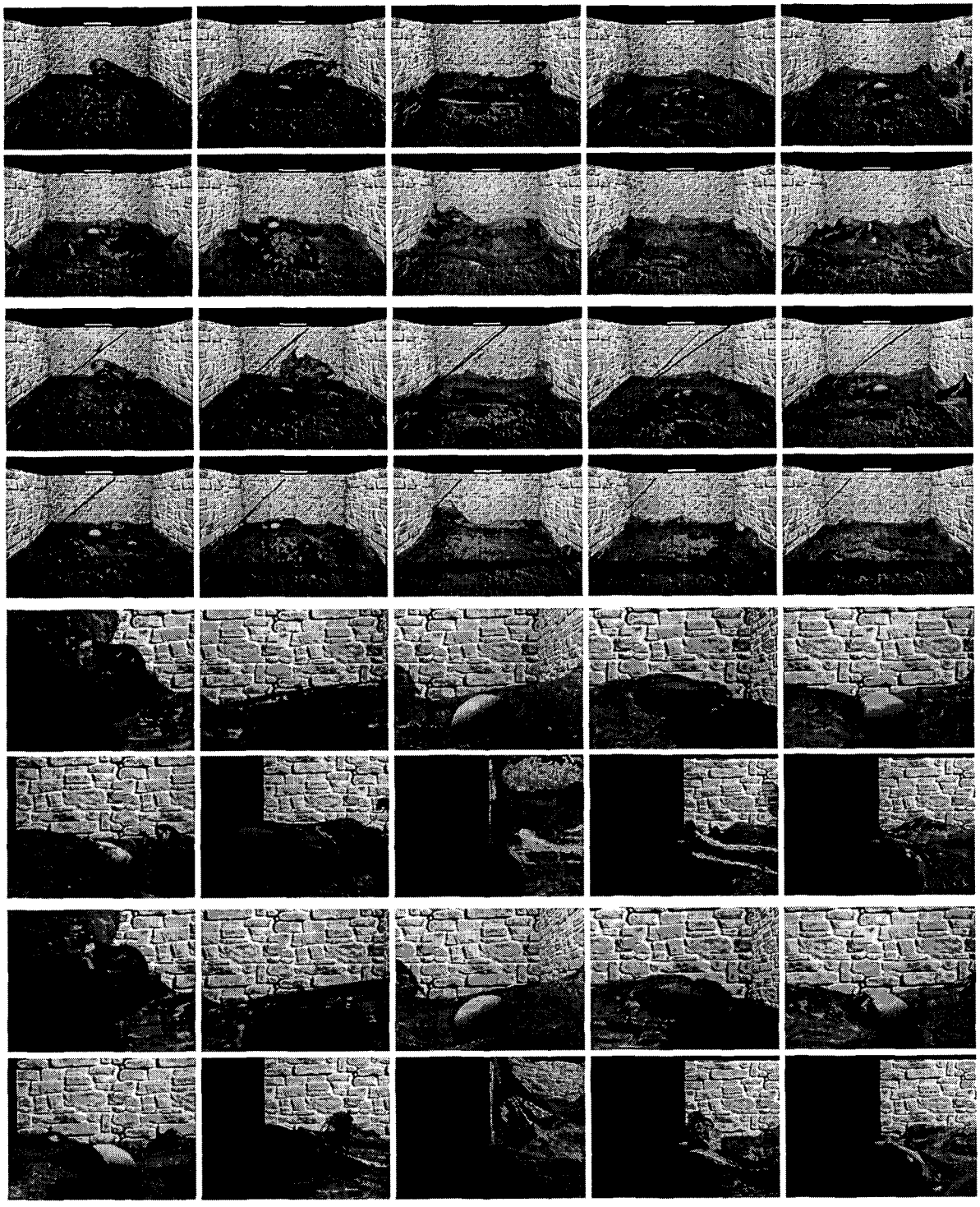


그림 7: 두 시뮬레이션 방법을 비교하기 위한 연속 장면 (프레임 번호: 100, 200, 300, ···, 1000). 다시, ADAPTIVE와 V-ADAPTIVE로 시뮬레이션한 결과의 전체와 카메라 방향에서 본 결과를 보여준다. 이 실험의 실제적인 해상도는 440 × 280×440이다. 분명 카메라 바깥쪽의 수치적인 오차의 누적이 V-ADAPTIVE 방법에서 많이 보이지만 실제 카메라로 보이는 부분에서의 차이는 거의 없는 것을 확인할 수 있기 때문에 받아들일 수 있고, 이로써 상당한 양의 계산 시간과 메모리를 절약할 수 있다.

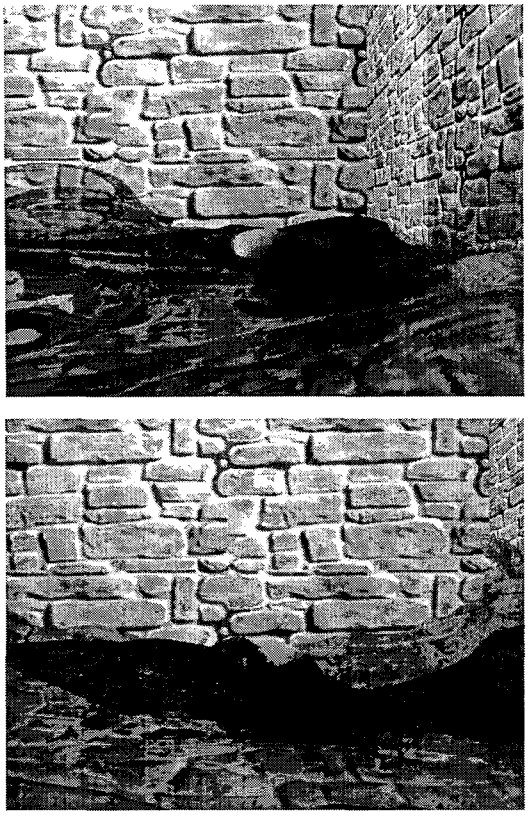


그림 8: 시점 의존 적응적 메쉬 분할의 예. 실제 카메라에서 약간 뒤로 움직인 위치에서 렌더링을 하였다. 뒷 부분의 붉은 영역이 카메라의 시야에서 보이지 않는 지역이고 원래 색으로 표현된 지역이 가장 높은 레벨이 사용된 지역이다 (레벨 3) . 카메라에서 멀리있는 지역은 녹색으로 표현되었고 이는 레벨 2를 사용하였다. 그림에서 알 수 있지만 붉은 색으로 표현된 지역의 물의 상세함이 원래 색으로 표현된 지역의 상세함보다 부족함을 알 수 있다.

[2] N. Foster and D. Metaxas. Realistic animation of liquids. *Graphical Models and Image Processing*, 58(5):471–483, 1996.

[3] N. Foster and R. Fedkiw. Practical animation of liquids. In *Proceedings of ACM SIGGRAPH 2001*, pages 23–30, 2001.

[4] J. Stam. Stable fluids. In *Proceedings of ACM SIGGRAPH 1999*, pages 121–128, 1999.

[5] D. Enright, S. Marschner, and R. Fedkiw. Animation and rendering of complex water surfaces. *ACM Transactions on Graphics (ACM SIGGRAPH 2002)*, 21(3):736–744, 2002.

[6] D. Enright, R. Fedkiw, J. Ferziger, and I. Mitchell. A hybrid particle level set method for improved interface capturing. *Journal of Computational Physics*, 183(1):83–116, 2002.

[7] N. Foster and D. Metaxas. Modeling the motion of a hot, turbulent gas. In *Proceedings of ACM SIGGRAPH 1997*, pages 181–188, 1997.

[8] R. Fedkiw, J. Stam, and H. Jensen. Visual simulation of smoke. In *Proceedings of ACM SIGGRAPH 2001*, pages 23–30, 2001.

[9] G. Yngve, J. O'Brien, and J. Hodgins. Animating explosions. In *Proceedings of ACM SIGGRAPH 2000*, pages 29–36, 2000.

[10] D. Nguyen, R. Fedkiw, and H. Jensen. Physically based modeling and animation of fire. *ACM Transactions on Graphics (ACM SIGGRAPH 2002)*, 21(3):721–728, 2002.

[11] B. Feldman, J. O'Brien, and O. Arikan. Animating suspended particle explosions. *ACM Transactions on Graphics (ACM SIGGRAPH 2003)*, 22(3):708–715, 2003.

[12] I. Ihm, B. Kang, and D. Cha. Animation of reactive gaseous fluids through chemical kinetics. In *Proceedings of Eurographics/ACM SIGGRAPH Symposium on Computer Animation 2004*, pages 203–212, 2004.

[13] N. Rasmussen, D. Nguyen, W. Geiger, and R. Fedkiw. Smoke simulation for large scale phenomena. *ACM Transactions on Graphics (ACM SIGGRAPH 2003)*, 22(3):703–707, 2003.

[14] M. Shah, J. M. Cohen, S. Patel, P. Lee, and F. Pighin. Extended galilean invariance for adaptive fluid simulation. In *Proceedings of Eurographics/ACM SIGGRAPH Symposium on Computer Animation 2004*, pages 213–221, 2004.

[15] M. Berger and J. Oliger. Adaptive mesh refinement for hyperbolic partial differential equations. *Journal of Computational Physics*, 53(3):484–512, 1984.

[16] A. M. Khokhlov. Fully threaded tree algorithms for adaptive refinement fluid dynamics simulations. *Journal of Computational Physics*, 143(2):519–543, 1998.

[17] B. Bennett and M. Smooke. Local rectangular refinement with application to nonreacting and reacting fluid flow problems. *Journal of Computational Physics*, 151(2):684–727, 1999.

[18] S. Popinet. Gerris: a tree-based adaptive solver for the incompressible Euler equations in complex geometries. *Journal of Computational Physics*, 190(2):572–600, 2003.

[19] V. Sochnikov and S. Efrima. Level set calculations of the evolution of boundaries on a dynamically adaptive grid. *International Journal for Numerical Methods in Engineering*, 56:1913–1929, 2003.

[20] J. Xia and A. Varshney. Dynamic view-dependent simplification for polygonal models. In *Proceedings of IEEE Visualization 1996*, pages 327–334, 1996.

[21] H. Hoppe. View-dependent refinement of progressive meshes. In *Proceedings of ACM SIGGRAPH 1997*, pages 189–198, 1997.

[22] D. Luebke and C. Erikson. View-dependent simplification of arbitrary polygonal environments. In *Proceedings of ACM SIGGRAPH 1997*, pages 199–208, 1997.

[23] M. Minion. A projection method for locally refined grids. *Journal of Computational Physics*, 127(1):158–178, 1996.

[24] L. Howell and J. Bell. An adaptive mesh projection method for viscous incompressible flow. *SIAM Journal in Scientific Computing*, 18(4):996–1013, 1997.