계산 공간 기반 계층 트리를 이용한 구조화된 격자 상에서의 빠른 스트림라인 가시화

이중연, 구기범, 허영주, 금복희
한국과학기술정보연구원 슈퍼컴퓨팅센터
e-mail:jylee@kisti.re.kr

Fast Streamline Visualization on Structured Grids using Computational Space Based Hierarchical Tree

Joong-Youn Lee, Geebum Koo, Youngju Hur, Bokhee Keum
Supercomputing Center, KISTI

요 약

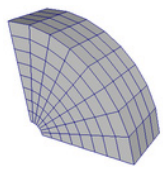
(비)구조화된 격자 상에 정의된 벡터 데이터는 다양한 과학 및 공학 분야에서 매우 중요하게 다루어진다. 이러한 데이터는 데카르트 격자 상의 데이터에 비해 많은 처리시간을 필요로 하는데, 이러한 문제는 계층 트리를 이용해서 빠르게 처리하는 것이 가능하다. 본 논문에서는 구조화된 격자 데이터에 대해 계산 공간을 기반으로한 계층 트리를 생성하고 이 트리를 이용해서 빠르게 데이터 샘플링을 처리하고자 했다. 이러한 방법을 이용해서 스트림라인 생성 시간을 평균 1800배 빨라지게 하는 것이 가능했다.

1. 서론

CFD, 천문, 대기 등 과학 및 공학의 다양한 분야에서는 여러 종류의 다양한 데이터가 다루어지고 있고, 이들을 보다 쉽게 이해하기 위해 컴퓨터 그래픽스의 힘을 빌려 데이터 가시화를 처리하고 있다. 3차원 벡터 데이터는 여러 데이터 종류들 중 가장 중요하게 다루어지고 있는데, 이 데이터 타입은 스칼라 데이터에 비해 그 크기가 크고, 많은 경우에 불안정한(unsteady) 특성을 지니고 있기 때문에 각 프레임 간 데이터의 연관성을 표출해야 하는 만큼 가시화하기에 더욱 복잡한 특성이 있다. 더욱이 이들 벡터 데이터들은 많은 경우에 구조화된 격자(structured grid) 또는 비구조화된 격자(unstructured grid) 구조를 가지고 있기 때문에 그동안 관련 연구가 활발히 진행된 데카르트 격자(Cartesian grid) 상의 데이터에 비해 많은 처리 시간을 필요로 한다. 이와 같이 (비)구조화된 격자 상의 벡터 데이터는 매우 중요한 데이터이나, 국내는 물론 국외의 경우에도 이와 관련된 연구는 드문 것이 현실이다. 본 논문에서는 구조화된 격자 상에 정의된 벡터 데이터를 계산 공간을 기반으로 생성한 계층 트리(hierarchical tree)를 이용해서 빠르게 가시화하고자 했다. 이를 위해 스트림라인(streamline) 표현 기법을 사용했고, 실제 예제 데이터를 이용해서 실험을 수행했다. 본 논문의 구성은 다음과 같다. 2장에서는 구조화된 격자에 대해 설명하고 3장에서는 이 격자를 위한 스트림라인 생성 방법을 기술한다. 4장에서는 계층 트리를 이용해서 스트림라인을 빠르게 생성하는 기법을 제안하고, 5장에서 실제 실험 결과를 설명한다. 마지막으로 6장에서 결론 및 향후 연구에 대해 논하도록 한다.

2. 구조화된 격자

구조화된 격자는 그림 1과 같은 구조의 격자로써, 일반적으로 격자를 구성하는 각 셀(cell)의 인덱스 자체는 (i, j, k) 형태의 데카르트 격자 위에 정의되지만, 격자의 꼭지점의 절대 좌표 값은 물리공간 (physical space)에서 불규칙하게 정의된다. 이 격자는 물리현상을 컴퓨터에서 시뮬레이션할 때 데이터 샘플링을 용이하게 하기 위해서 만든 격자이다. 원래 데이터는 물리공간에서 정의되는데, 이 공간에서는 데이터 샘플링에 많은 시간을 필요로 하기 때문에 전체 계산시간이 느려지게 된다. 따라서 이러한 문제를 해결하기 위해 새롭게 계산공간(computational space)을 고안하게 됐다. 즉, 컴퓨터 시뮬레이션을 할 때 물리공간에서 정의된 데이터를 모두 데카르트 격자 형태로 정의된 계산공간으로 옮기고 이를 통해 데이터 샘플링을 빠르게 처리할 수 있도록 한 것이다. 이렇게 계산된 값들은 모두 가상의 계산공간에서 정의된 좌표를 기반으로 하고 있으므로 이를 다시 물리공간으로 옮겨서 올바른 최종 결과값을 얻도록 해야한다. 구조화된 격자 구조는 이러한 과정을 처리하기 위해 고안된 격자 구조이다.



(그림 1) 구조화된 격자

3. 스트림라인의 생성

스트림라인(streamline)은 벡터 데이터 중 유동 데이터(flow data)를 가시화하는데 매우 중요하게 사용되는 표현 기법으로 운동하는 유체의 각 점에서 속도벡터의 방향이 접선이 되도록 그은 곡선을 의미한다. 그림 5는 스트림라인 가시화의 실제 예이다. 이러한 스트림라인을 생성하기 위해서는 다음과 같은 과정을 따라야 한다.

1. 시작점 정의
2. 유동장을 따라 파티클 이류 (particle advection)
3. 이동한 파티클 위치를 선으로 연결

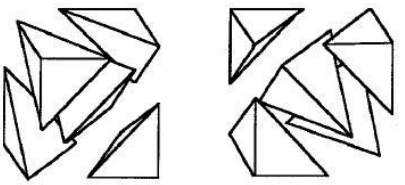
여기서 스트림라인의 생성에 가장 중요한 작업은 2번인데, 1번에서 정의한 위치에 파티클(particle)을 놓고 유동장을 따라 파티클을 계속 이류(advection) 시키는 작업이다. 이를 위해서는 우선 파티클 위치에서의 유동값을 가져와야 하는데, 이는 벡터 데이터의 샘플링을 통해 처리할 수 있다. 데카르트 격자에서 정의된 데이터의 경우 샘플링을 $O(1)$에 처리할 수 있는 반면에 (비)구조화된 격자에서 정의된 데이터의 경우에는 특별한 자료구조를 사용하지 않는 한 $O(n)$의 복잡도를 가진다. (여기서 n은 격자 내 셀의 수) 이렇게 샘플링한 유동값은 실제 이류 계산에서 사용된다. 이류 계산은 오일러(Euler) 기법이나 RK (Runge-Kutta) 기법이 주로 사용되는데, 이 논문에서는 빠른 계산을 위해 오일러 기법을 사용했다.

3.1. 데이터 샘플링

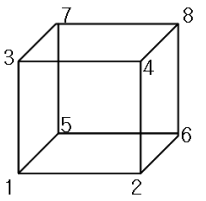
격자화된 구조상의 데이터를 샘플링하기 위한 방법에는 크게 두 가지가 있다. 하나는 물리 공간에서 샘플링하는 것이고 나머지 하나는 계산 공간에서 샘플링하는 것이다. 여기에는 장단점이 있는데, 물리공간에서는 격자가 불규칙한 형태이기 때문에 샘플링을 위한 셀 탐색에 시간이 오래 걸린다는 단점이 존재한다. 반면에 계산공간에서는 격자가 균등한 직교형이기 때문에 셀 탐색 시간이 매우 빠르지만 물리공간에서 정의된 파티클 좌표를 계산 공간으로 변환해야 한다는 단점이 있다. Sadarjoen 등은 [1]에서 두 기법의 속도를 비교했는데, 물리공간에서 계산공간으로 변환했다가 다시 물리공간으로 변환하는 과정의 시간이 너무 오래 걸리기 때문에 그냥 물리공간에서 계산하는 것이 더욱 빠르다고 했다. 본 논문에서는 이에 따라 물리공간에서 데이터를 샘플링했다. 물리공간에서는 계산공간에서와는 달리 각 셀들이 기본적으로 육면체(hexahedron)의 형태를 가지고 일부 오면체(pentahedron) 또는 사면체(tetrahedron)의 형태를 가지게 된다. 이러한 조건에서 임의의 포인트에서의 데이터값을 샘플링하는 것은 매우 복잡하다. 우선, 포인트가 포함되는 셀을 찾아야 하는데, 임의의 포인트가 임의의 육면체에 포함되는지 판단하는 알고리즘이 복잡하기 때문이다. 물론, 육면체의 모든 면이 평면이라는 보장이 있으면 알고리즘이 상당히 간단해질 수 있지만, 일반적으로 격자화된 그리드에서 각 셀을 이루는 육면체들은 각면이 평면이 아닌 경우가 다수 존재하기 때문이다. 더구나 모든 셀이 육면체라는 보장도 없고 오면체 또는 사면체일 수도 있기 때문에 문제는 더욱 복잡해진다. 이러한 이유로 많은 경우에 육면체 셀을 사면체로 분리해서 비구조화된 격자와 같은 형태로 변경한 뒤 셀 탐색을 진행한다[2]. 사면체의 모든 면은 반드시 평면이기 때문에 문제는 더욱 간단해질 수 있다.

3.2. 구조화된 격자 셀에서의 포인트 포함 검사

구조화된 격자의 각 셀에서 포인트가 포함되는지를 검사하는 것은 임의의 포인트가 임의의 육면체 안에 포함되는지를 검사하는 것과 동일한 문제이다. 이러한 문제를 해결하기 위한 가장 쉬운 방법은 육면체를 모두 그림 2와 같이 사면체로 분해한 뒤 각 사면체에 대해서 포인트가 안에 포함되는 지를 파악하는 것이다[2]. 이러한 방법을 사용하면 샘플 포인트가 속하는 셀을 찾는 알고리즘이 간단해진다. 한 개의 육면체는 최소한 5개의 사면체로 분해가 가능한데, 이 방법을 사용하면 한 개의 포인트가 비교해야 하는 셀의 개수가 5배로 늘어나게 된다. 그러나 포인트가 사면체에 포함되는지를 판단하는 알고리즘이 육면체에 비해 간단하고 사면체의 모든 면은 반드시 평면이기 때문에 고려사항이 적어지는 장점이 있다. 표 1은 육면체의 인덱스가 그림 3과 같을 때 육면체를 사면체로 나누기 위한 사면체의 인덱스이다.



(그림 2) 육면체의 사면체화(tetrahedralization)



(그림 3) 꼭지점 인덱스

<표 1> 육면체의 사면체화를 위한 꼭지점 인덱스 표

사면체	육면체 꼭지점
1	1, 2, 3, 5
2	2, 3, 4, 8
3	2, 5, 6, 8
4	3, 5, 7, 8
5	2, 3, 5, 8

임의의 포인트가 임의의 사면체에 포함되는 지 여부는 다음과 같은 방법으로 알 수 있다. 사면체의 네 꼭지점 중 세 꼭지점의 좌표와 포인트의 좌표로 생성할 수 있는 다섯 개의 4X4 행렬에 대한 행렬식(determinant)들을 구

한다. 이 행렬식의 부호가 모두 같으면 포인트가 사면체에 포함되는 것이고 그렇지 않으면 포함되지 않는 것이다.

* 사면체의 꼭지점 좌표
  V1 = (x1, y1, z1)
  V2 = (x2, y2, z2)
  V3 = (x3, y3, z3)
  V4 = (x4, y4, z4)

* 포인트 좌표
  P = (x, y, z)

* 사면체의 세 꼭지점 좌표와 포인트 좌표로 생성한 행렬들

$$D0=\begin{vmatrix} x1 & y1 & z1 & 1 \\ x2 & y2 & z2 & 1 \\ x3 & y3 & z3 & 1 \\ x4 & y4 & z4 & 1 \end{vmatrix} \quad D3=\begin{vmatrix} x1 & y1 & z1 & 1 \\ x2 & y2 & z2 & 1 \\ x & y & z & 1 \\ x4 & y4 & z4 & 1 \end{vmatrix}$$

$$D1=\begin{vmatrix} x & y & z & 1 \\ x2 & y2 & z2 & 1 \\ x3 & y3 & z3 & 1 \\ x4 & y4 & z4 & 1 \end{vmatrix} \quad D4=\begin{vmatrix} x1 & y1 & z1 & 1 \\ x2 & y2 & z2 & 1 \\ x3 & y3 & z3 & 1 \\ x & y & z & 1 \end{vmatrix}$$

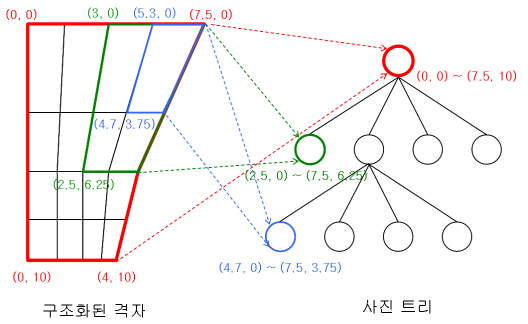
$$D2=\begin{vmatrix} x1 & y1 & z1 & 1 \\ x & y & z & 1 \\ x3 & y3 & z3 & 1 \\ x4 & y4 & z4 & 1 \end{vmatrix}$$

4. 계층 트리

구조화된 격자에서 데이터를 샘플링하는 작업의 복잡도는 *O(n)*이다. 즉, 셀의 수가 많아질수록 작업 시간이 선형으로 증가하는 것이다. 데이터를 샘플링하는 작업을 다시 이야기하면 결국 데이터를 탐색하는 것이라고 할 수 있는데, 이를 위한 대표적인 자료구조로 계층트리가 있다. 본 논문에서는 여러 종류의 계층 트리 중 팔진트리를 사용했는데, 이는 팔진트리가 구조화된 격자를 효율적으로 구성할 수 있는 계층트리이기 때문이다. 팔진트리를 사용하면 이론상 탐색에 소요되는 시간은 *O(log n)*으로 줄어들 수 있기 때문에 본 논문에서는 셀을 기반으로 하는 팔진트리를 생성하고 이를 이용해서 데이터 샘플링을 위한 셀 탐색을 수행하도록 했다. 팔진트리를 생성하기 위해 추가적으로 소요되는 시간은 *O(n log n)*으로 셀의 개수가 많아지면 무시할 수 없는 시간이 소요될 수 있으나 이는 전처리과정으로 계산할 수 있으므로 큰 문제는 되지 않는다.

4.1. 계산공간 기반 팔진트리

3장에 살펴본바와 같이 구조화된 격자의 육면체를 사면체화(tetrahedralization)한 뒤 비구조화된 격자를 위한 탐색 알고리즘을 사용하면 구조화된 격자의 셀 탐색이 가능해진다. 대표적인 비구조화된 격자의 셀 탐색 방법에는 [2]와 [3]이 있다. 그러나 이러한 방법은 데이터 샘플링이 보다 용이하도록 고안된 구조화된 격자의 장점을 살리지 못한 방법이다. 즉, 구조화된 격자에서 각 꼭지점의 좌표들은 물리공간에서 불규칙하게 저장되어 있지만 그 꼭지점들의 인덱스는 데카르트 격자에서 규칙적으로 정의되어 있고, 이를 활용하면 보다 빠르게 셀 인덱싱이 가능해진다. 팔진트리를 생성할 때 트리 구조를 데카르트 그리드 형태인 셀 인덱스를 이용해서 생성하고, 팔진트리 내 각 노드에는 그 노드의 자식 노드들을 모두 포함하는 경계상자(bounding box)를 저장하는 것이다. 이때, 경계상자의 좌표는 모두 물리좌표계로 저장해야 한다. 이러한 방법을 사용하면 구조화된 격자의 가로, 세로, 높이의 형태에 따라 최대한 균형잡힌 팔진트리를 빠르게 생성할 수 있다. 팔진트리의 각 노드가 저장하는 경계상자는 몇 번의 크기 비교만으로 포인트가 그 노드와 자식 노드들에 포함되는지를 빠르게 판단할 수 있도록 해서 탐색을 매우 빠르게 수행할 수 있도록 한다. 팔진트리의 단말 노드에는 실제 셀의 인덱스를 저장해서 최종 후보 셀을 쉽게 접근할 수 있도록 한다.



(그림 4) 구조화된 격자로 생성한 사(팔)진트리

4.2. 팔진트리를 적용한 셀 탐색

본 논문에서 구현한 셀 탐색 기법은 두 단계로 나뉘는데, 우선 팔진트리를 이용해서 전체 셀 중 포인트가 위치할 가능성이 있는 셀을 탐색하는 것이 첫 번째 단계이고, 이 셀들 중 실제 포인트가 위치하는 지를 정확히 계산하는 것이 두 번째 단계이다. 이렇게 두 단계로 나눈 것은 팔진트리의 중간 노드에서 포인트가 그 노드 혹은 자식 노드에 포함되는 지를 계산하는 연산은 경계상자를 이용한 간단한 연산으로 빠르게 처리하고 최종 후보 셀에 대해서만 정확한 연산을 수행하도록 해서 수행시간을 단축하기 위함이다.

5. 실험 및 결과

본 논문에서는 Intel Core2 Quad 2.66GHz CPU와 4GB의 메모리, 768MB의 그래픽스 메모리를 가진 NVIDIA GeForce8800 GTX 그래픽스 카드를 장착한 일반 데스크탑 PC를 이용해서 실험을 했다. 실험에 사용한 데이터는 한국형 고등훈련기인 KTX 데이터로 총 6개의 블럭, 26

프레임으로 이루어져있는 불안정한 (unsteady) 데이터이나, 본 논문에서는 한 프레임만을 사용해서 실험했다. 각 블록의 크기는 표 2와 같다. 각 셀에는 1개의 좌표, 1개의 속도벡터 그리고 2개의 스칼라가 저장되어 있으며 모든 값은 단정도 부동소수점(single precision floating point)으로 이루어져있다.

<표 2> KTX 데이터 크기

블럭	해상도	크기
1	12 X 53 X 48	954KB
2	132 X 47 X 48	9.09MB
3	187 X 31 X 59	10.44MB
4	72 X 37 X 48	3.90MB
5	72 X 17 X 48	1.79MB
6	17 X 31 X 48	790.50KB
총합	-	26.93MB

전처리 과정에서 KTX 데이터의 팔진트리를 생성했는데, 이때 소요된 시간 및 메모리 사용량은 표 3과 같다.

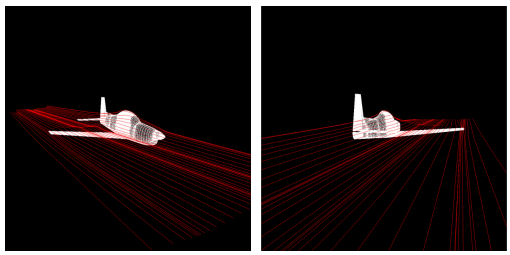
<표 3> 팔진트리 생성 속도 및 메모리 사용량

블럭	생성 속도 (초)	메모리 사용량 (MB)
1	0.01	2.25
2	0.14	26.64
3	0.17	32.69
4	0.05	9.69
5	0.02	4.56
6	0.01	1.97
총합	0.4	77.8

생성속도는 큰 문제가 없었으나 메모리 사용량은 실제 데이터의 3배에 달했다. 이는 팔진트리의 각 노드에 실제 KTX 데이터로의 포인터 및 경계상자, 그리고 팔진트리를 유지하기 위한 자체 정보 등이 저장되어 KTX 데이터의 셀에 저장되는 정보에 비해 약 1.5배정도의 저장 공간을 필요로 하고, 팔진트리의 노드 수가 본래 KTX 데이터의 셀 수에 비해 약 2배정도 많기 때문이다. 전체 KTX 데이터에 대해 10, 50, 100, 1000개의 스트림라인을 생성하는 총 4종류의 실험을 수행했는데, 각 스트림라인을 생성하기 위한 파티클 이류는 파티클이 도메인 밖으로 나가지 않는 한 200번하도록 했다. 팔진트리를 사용한 경우와 사용하지 않은 경우의 속도는 표 4와 같다.

<표 4> 기본방법과 팔진트리방법의 속도 비교 (단위 :초)

스트림라인 수	기본방법	팔진트리방법	속도차
10	16.71	0.01	1671X
50	165.73	0.07	2367.6X
100	394.44	0.19	2076X
1000	2942.26	3.27	899.8X



(그림 5) KTX 데이터 스트림라인 생성 그림

6. 결론 및 향후 연구

본 논문에서는 계산공간을 기반으로 생성한 팔진트리를 이용하여 구조화된 격자 상의 벡터 데이터에서 빠르게 스트림라인을 생성했다. 이를 통해 평균 약 1800배 정도 속도를 향상시킬 수 있었다. 만약 데이터가 2^n X 2^n X 2^n의 크기를 가진다면 팔진트리는 균형잡힌 트리가 되어 메모리 사용이나 트리 탐색 등의 측면에서 보다 최적화될 수 있을 것으로 보인다. 그러나 일반적으로 구조화된 격자 데이터들은 이러한 크기를 가지지 않으므로 이러한 데이터 구조에 최적화된 다른 계층 트리를 사용할 필요도 있다. 직교 BSP 트리가 좋은 대안 중 하나가 될 수 있을 거싱다. 계층트리를 이용해서 셀 탐색을 빠르게 한다 하더라도 데이터의 크기가 매우 커지거나 파티클 이류 기법으로 4차 이상의 RK 기법을 이용하면 계산량이 많아져서 스트림라인 생성속도가 느려지게 된다. 이를 해결하기 위한 방법으로는 GPU를 이용해서 파티클 이류 계산 및 셀 탐색을 가속할 수 있는 방법과 데이터를 여러 대의 클러스터링 노드에 분산해서 처리하는 방법이 있을 수 있다. 마지막으로 스트림라인은 유동 데이터를 가시화하기 위한 좋은 표현 방법이지만 유동 데이터가 아닌 다른 벡터 데이터는 물론 유동 데이터의 경우에도 다른 표현 기법을 필요로 할 수가 있다. 이러한 방법에는 파티클 운동, 볼륨 가시화, 스트림볼륨 가시화 기법 등이 있을 수 있다.

7. 참고문헌

[1] Ari Sadarjoen et al., "Particle Tracing Algorithms for 3D Curvilinear Grids", Proceedings of Fifth Eurographics Workshop on Visualization in Scientific Computing, 1994

[2] David K. Kenwright and David A. Lane, "Interactive Time-Dependent Particle Tracing Using Tetrahedral Decomposition", IEEE Transactions on Visualization and Computer Graphics, Vol. 2, Num. 2, pages 120-129, 1996

[3] Max Langbein, Gerik Scheuermann, Xavier Tricoche, "An Efficient Point Location Method for Visualization in Large Unstructured Grids", Proceedings of VMV 2003, Munich, 2003