ISBN 978-89-6211-272-3

Structured Grid 데이터에서의 스트림라인 가시화 구현

(Implementation of Streamline Visualization on the Structured Grid Data)

이 중 연 (jylee@kisti.re.kr)

한 국 과 학 기 술 정 보 연 구 원
Korea Institute of Science & Technology Information

목차

그림 차례

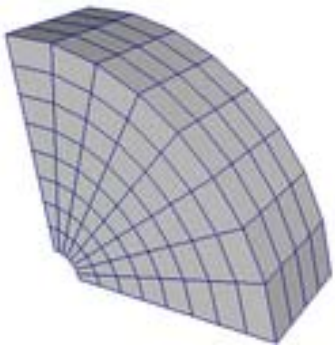
표 차례

수식 차례

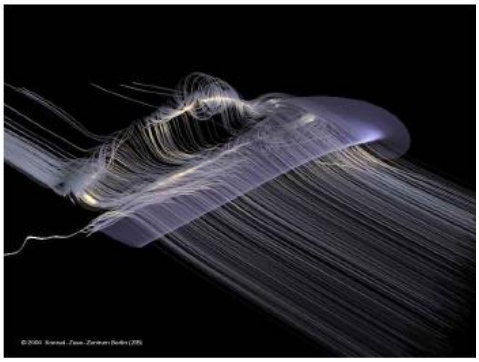
1. 서론

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(vector) 데이터는 여러 데이터 종류들 중 가장 중요하게 다루어지고 있는데, 이 데이터 타입은 스칼라(scalar) 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 structured grid 또는 unstructured grid 구조를 가지고 있기 때문에 그동안 관련 연구가 활발히 진행된 Cartesian grid 상의 데이터에 비해 많은 처리 시간을 필요로 한다.

Structured grid는 그림 1-1과 같은 구조의 격자로써, 일반적으로 격자를 구성하는 각 셀(cell)의 인덱스 자체는 (i, j, k) 형태의 Cartesian grid 위에 정의되지만, 격자의 꼭짓점의 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 Cartesian grid 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 결과 값을 얻도록 해야 한다. Structured grid 구조는 이러한 과정을 처리하기 위해 고안된 격자 구조이다.



[그림 1-1] Structured Grid



[그림 1-2] 스트림라인

이와 같이 structured grid 상의 벡터 데이터는 매우 중요한 데이터 형식이나, 국내는 물론 국외의 경우에도 이와 관련된 연구는 드문 것이 현실이다. 본 기술문서에서는 structured grid 상에 정의된 벡터 데이터에서 스트림라인(streamline)을 빠르게 생성할 수 있는 방법에 대해 기술한다.

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 1-2는 스트림라인 가시화의 실제 예이다. 이러한 스트림라인을 생성하기 위해서는 다음과 같은 과정을 따라야 한다.

1. 시작점 정의
2. 유동장을 따라 파티클 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 advection 시키는 작업이다. 이를 위해서는 우선 파티클 위치에서의 유동값을 가져와야 하는데, 이는 벡터 데이터의 샘플링을 통해 처리할 수 있다. Cartesian grid에서 정의된 데이터의 경우 샘플링을 $O(1)$에 처리할 수 있는 반면에 structured grid에서 정의된 데이터의 경우에는 특별한 자료구조를 사용하지 않는 한 $O(n)$의 복잡도를 가져 더욱 긴 시간을 필요로 한다. (여기서 n은 격자 내 셀의 수) 이러한 Structured grid 상의 데이터를 빠르게 접근하기 위해 고안한 자료구조에는 여러 가지가 있지만 대표적인 것으로는 팔진트리(octree), k-차원 트리(kd-tree), BSP 트리 (binary space partitioning tree) 등이 있다. 본 기술문서에서는 이를 위해 특별히

고안한 두 종류의 자료구조를 사용한다. 첫 번째 자료구조는 계산 공간(computational space)를 기반으로 생성한 팔진 트리이고 나머지 하나는 Structured grid를 Unstructured grid로 변환한 뒤 생성한 kd-tree이다. 이러한 두 종류의 자료구조는 실제 예제 데이터들에 적용되고 이를 이용해서 실험을 수행했다.

이와 같은 방법으로 샘플링한 유동값은 실제 advection 계산에서 사용된다. advection 계산은 오일러(Euler) 기법이나 RK(Runge-Kutta) 기법이 주로 사용된다.

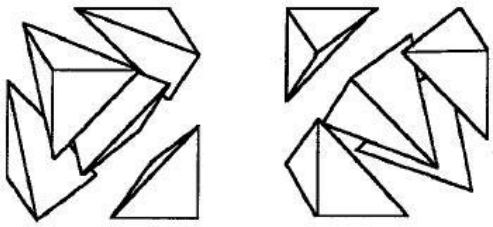
2. 데이터 샘플링

Structured grid 상의 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 기술문서에서는 이에 따라 물리공간에서 데이터를 샘플링한다. 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 데이터 값을 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 간단해실 수 있지만, 일반적으로 유동 데이터를 위한 격자구조에서 각 셀을 이루는 육면체들은 각 면이 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 unstructured grid와 같은 형태로 변경한 뒤 셀 탐색을 진행한다[2]. 사면체의 모든 면은 반드시 평면이기 때문에 문제는 더욱 간단해질 수 있다.

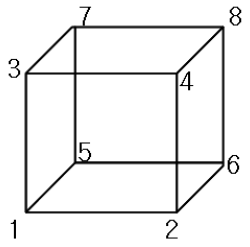
가. 셀에서의 포인트 포함 검사

Structured grid의 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 2-1과 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 2-1은 육면체의 인덱스가 그림 2-2와 같을 때 육면체를 사면체로 나누기 위한 사면체의 인덱스이다.



[그림 2-1] 육면체의 사면체화(tetrahedralization)



[그림 2-2] 꼭짓점 인덱스

[표 2-1] 육면체의 사면체화를 위한 꼭짓점 인덱스 표

사면체	육면체 꼭짓점
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 꼭짓점 중 세 꼭짓점의 좌표와 포인트의 좌표로 생성할 수 있는 다섯 개의 4×4 행렬에 대한 행렬식(determinant)들을 구한다. 이 행렬식의 부호가 모두 같으면 포인트가 사면체에 포함되는 것이고 그렇지 않으면 포함되지 않는 것이다. 사면체와 포인트의 좌표가 아래와 같을 때 이들을 이용해서 생성할 수 있는 다섯 개의 4x4 행렬은 수식 2-1과 같다.

* 사면체의 꼭짓점 좌표

V1 = (x1, y1, z1)

V2 = (x2, y2, z2)

V3 = (x3, y3, z3)

* 포인트 좌표

P = (x, y, z)

V4 = (x4, y4, z4)

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

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

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

[수식 2-1] 사면체의 세 꼭짓점 좌표와 포인트 좌표로 생성한 행렬들

나. 계층트리

Structured grid에서 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 빠르게 처리하기 위한 대표적인 자료구조로 계층트리가 있다. 본 기술문서에서는 여러 종류의 계층 트리 중 팔진트리와 kd-트리를 사용했는데, 이는 팔진트리와 kd-트리가 각각 structured grid와 unstructured grid를 효율적으로 구성할 수 있는 계층트리이기 때문이다.

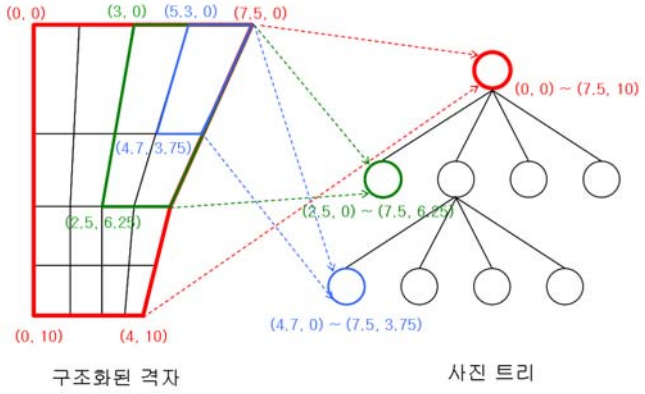
팔진트리를 사용하면 이론상 탐색에 소요되는 시간은 *O(logN)*으로 줄어들 수 있기 때문에 본 기술문서에서는 셀을 기반으로 하는 팔진트리를 생성하고 이를 이용해서 데이터 샘플링을 위한 셀 탐색을 수행하도록 했다. 팔진트리를 생성하기 위해 추가적으로 소요되는 시간은 *O(NlogN)*으로 셀의 개수가 많아지면 무시할 수 없는 시간이 소요될 수 있으나 이는 전처리과정으로 계산할 수 있으므로 큰 문제는 되지 않는다.

Structured grid 데이터의 샘플링을 빠르게 처리하기 위해 적용한 또 다른 계층구조는 kd-트리이다. kd-트리는 이진트리의 일종으로 3차원 공간상에 정의된 임의의 물체를 효율적으로 저장, 탐색하기 위한 계층구조의 일종이다. 일반적으로 unstructured grid와 같은 불규칙적인 격자구조를 표현하는데 많이 사용된다.

kd-트리 역시 팔진트리와 마찬가지로 *O(logN)*의 탐색 시간을 필요로 하는데, 팔진트리와는 달리 데이터가 x, y, z의 각 축의 길이가 달라도 compact하게 저장하는 것이 가능한 장점이 있다. kd-트리는 structured grid 보다는 unstructured grid에 최적화되어 있지만 structured grid 역시 unstructured grid로 변환이 가능하기 때문에 이러한 데이터 변환 과정을 거치면 kd-트리를 structured grid에 적용하는 것이 가능하다. 데이터 변환 과정이나 트리 생성 과정은 역시 전처리과정에서 처리가 가능하다.

1) 계산 공간 기반 팔진트리

Structured grid의 육면체를 사면체화(tetrahedralization)한 뒤 unstructured grid를 위한 탐색 알고리즘을 사용하면 structured grid의 셀 탐색이 가능해진다. 그런데 기존의 셀 탐색 기법들은 데이터 샘플링이 보다 용이하도록 고안된 structured grid의 장점을 살리지 못한 방법이다. 즉, structured grid에서 각 꼭짓점의 좌표들은 물리공간에서 불규칙하게 저장되어 있지만 그 꼭짓점들의 인덱스는 Cartesian grid에서 규칙적으로 정의되어 있고, 이를 활용하면 보다 빠르게 셀 인덱싱이 가능해진다. 팔진트리를 생성할 때 트리 구조를 Cartesian grid 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 structured grid의 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 포함되는 지를 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.

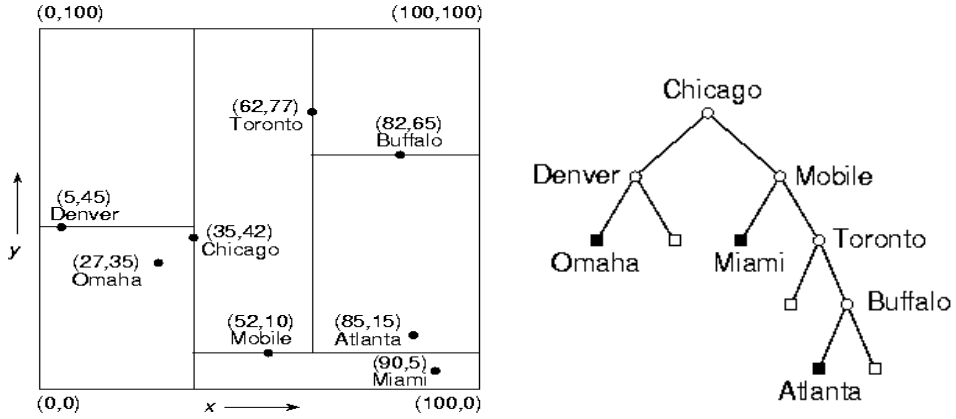


[그림 2-3] structured grid로 생성한 사(팔)진트리

여기에서 제안하는 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.

2) kd-트리

kd-트리는 이진트리의 일종으로 3차원 공간상에 정의된 임의의 물체를 효율적으로 저장, 탐색하기 위한 계층구조의 일종이다. kd-트리의 각 노드는 그 노드를 양분하는 hyperplane이 저장되는데, 나뉜 두 공간으로 자식 노드를 구성한다. (kd-tree에 대한 추가 설명 필요...) kd-트리의 노드에는 hyperplane과 함께 k차원 공간에서 정의되는 점 또는 영역이 저장되는데, 점이 저장되면 point kd-tree, 영역이 저장되면 range kd-tree라고 한다. 점이 저장될 경우에는 각 노드에 실제 데이터가 저장되는 것이고 영역이 저장될 경우에는 자식 노드들의 영역이 저장되며 실제 데이터는 단말 노드에 저장된다.



[그림 2-4] Point kd-tree

Unstructured grid를 kd-tree로 구성하는 방법에도 두 가지 방법이 모두 사용될 수 있는데, 노드에 격자의 꼭짓점을 저장할 경우에는 point kd-tree가 이용된다. 이 경우에는 점이 어떤 셀에 포함되는 지를 판단하기 위해 우선 점의 위치와 가장 가까운 꼭짓점을 찾고 그 꼭짓점으로 구성된 셀들에 대해 점이 포함되는지를 판단한다. 만약 그 점이 포함되는 셀이 존재하지 않는다면 두 번째로 가까운 꼭짓점을 찾아서 다시 그 꼭짓점으로 구성된 셀들에 대해 점이 포함되는지를 판단한다. 이 방법은 이와 같이 kd-트리에서 점과 가까운 꼭짓점들을 순차적으로 찾아야 하는데, 이 알고리즘은 $O(logN)$의 복잡도를 갖는다. 또한 데이터마다 조금씩 다르지만 일반적으로 unstructured grid의 경우 꼭짓점의 수가 셀의 수보다 적기 때문에 꼭짓점을 기반으로 한 kd-트리 탐색 알고리즘은 매우 효율적이라고 할 수 있다. 그러나 점이 전체 데이터 밖에 나갔을 경우를 위한 추가적인 처리가 필요하고 각 점을 구성하는 셀 정보를 미리 꼭짓점에 저장해야 하는 점, 그리고 꼭짓점을 찾은 뒤 추가적인 셀 탐색이 필요한 점은 단점으로 들 수 있다.

반면에 노드에 셀을 저장할 경우에는 일반적으로 range kd-tree가 이용된다. 3차원 공간의 경우에는 영역으로 상자(box)가 사용되는데, 각 노드는 자식 노드들의 경계상자(bounding box)를 영역으로 저장한다. 단말 노드에는 실제 셀 정보가 경계상자와 함께 저장된다. 이 방법은 간단한 영역 비교로 점이 현재 노드에 속하는지 속하지 않는지를 알 수 있기 때문에 빠르게 셀 탐색을 가능하게 한다. 또한 꼭짓점 기반 방법과는 달리 추가적인 저장 공간 및 탐색을 필요로 하지 않는다는 장점이 있다. 그러나 데이터의 종류에 따라 노드의 수가 많아진다는 단점이 존재한다.

다. 보간법

탐색을 통해 찾은 셀에서 파티클이 위치한 부분의 정확한 벡터 및 스칼라 값을 알기 위해서는 셀을 구성하는 각 꼭짓점에 위치한 값들을 이용해서 보간(interpolation)해야 한다. 일반적인 rectilinear grid의 경우에는 삼선형보간(trilinear interpolation)으로 이러한 문제가 해결되지만 structured grid나 unstructured grid와 같은 불규칙한 격자구조에서는 이 방법을 적용할 수 없다. 불규칙한 격자구조에서의 보간을 위한 대표적인 기법에는 Inverse Distance Weighting과 Volume Weighting을 들 수 있다.

1) Inverse Distance Weighting

x_0, x_1, ... x_7을 육면체의 각 꼭짓점들이고 그 꼭짓점들에 위치한 벡터를 v_0, v_1, ... v_7이라 할 때, 파티클 p가 위치한 지점의 벡터 v_p는 다음과 같이 각 꼭짓점들의 weighted average로 정의할 수 있다.

$$v_p = w_0 v_0 + w_1 v_1 + \cdots + w_7 v_7 = \sum_{i=0}^{7} w_i v_i$$

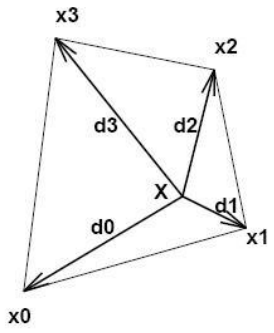
[수식 2-2] 꼭짓점에 정의된 벡터들의 가중평균벡터

이때, 각 v_i에 곱해지는 w_i는 각 꼭짓점에서의 벡터 값에의 가중치로써, 파티클 p와 각 꼭짓점의 거리에 반비례한다. 즉, 가까울수록 가중치가 높아지는 것이다. w_i는 수식 2-3과 같이 계산이 가능하다.

$$d_i = \| x_i - X \|$$

$$w_i = \frac{\frac{1}{(d_i)^2}}{\sum_{j=0}^{7} \frac{1}{(d_j)^2}}$$

[수식 2-3] Inverse Distance Weighting 기법에서의 가중치 계산



[그림 2-5] Inverse Distance Weighting 보간 기법

2) Volume Weighting

Volume weighting 보간 기법은 inverse distance weighting 기법과 마찬가지로 각 꼭짓점 값들의 가중평균을 계산해서 사용하는데, 가중치로 inverse distance 대신에 체적을 사용한다. 사면체로 된 셀 내부에 파티클 p가 존재할 때, 사면체는 다시 4개의 사면체로 분리가 가능하다. 이때, 각 셀을 구성하는 사면체의 각 꼭짓점을 x_0, x_1, x_2, x_3라 하면 분리된 사면체 T_n은 다음과 같이 구성이 가능하다.

T_0 : (x_0, x_1, x_2), T_1 : (x_0, x_1, x_3), T_2 : (x_0, x_2, x_3), T_3 : (x_1, x_2, x_3).

이렇게 분리한 사면체의 체적을 순서대로 V_0, V_1, V_2, V_3라 하고 원래 셀을 구성하는 사면체를 V_C라 하면 각 사면체에 대한 가중치는 수식 2-4와 같이 계산된다.

$$w_0 = \frac{V_0}{V_C}, \quad w_1 = \frac{V_1}{V_C}, \quad w_2 = \frac{V_2}{V_C}, \quad w_3 = \frac{V_3}{V_C}$$

[수식 2-4] Volume Weighting 보간 기법에서의 가중치 계산

이때, 꼭짓점 x_0, x_1, x_2, x_3로 구성된 사면체의 체적 V는 다음과 같이 계산이 가능하다.

$$V = \frac{|x_0x_1 \bullet (x_0x_2 \times x_0x_3)|}{6}$$

[수식 2-5] 사면체의 체적 구하기

3. Advection과 Integration

Integration 기법이란 벡터장에서 유동에 따른 한 파티클의 궤적을 시각적으로 표현하는 기법을 말한다. 이를 위해 많은 기법이 소개되었는데, 대표적으로 스트림라인, 스트림튜브, 스트림리본, 패스라인 등이 있다. 이와 같은 점의 궤적을 시각적으로 표현하기 위해 사용하는 대체 표현물을 integral object라고 부르고, 이러한 integral object들은 벡터장의 적분으로 계산이 가능하다. 대표적인 integral object 중 하나인 스트림라인을 생성하기 위해서는 스트림라인의 시작점에 파티클을 배치하고 시간에 따라 파티클이 벡터장을 이동하는 궤적을 곡선으로 표현하면 된다. 이때, 곡선이 지나가는 지점에 정의된 벡터들은 이 곡선의 접선으로 생각할 수 있다. 즉, 벡터 요소들이 불연속적으로 저장된 벡터 데이터에서, 각 벡터 요소는 아래와 같은 식으로 표현이 가능한 것이다.

$$\frac{d\mathbf{x}(t)}{dt} = \mathbf{v}(\mathbf{x}(t), t)$$

[수식 3-1] 벡터장내
벡터의 상미방 표현

여기서 t는 시간, x(t)는 시간에 따른 파티클의 위치, v는 벡터장에서의 벡터요소를 의미한다. 파티클의 초기 위치를 $\mathbf{x}_0$라고 하면 이 상미분방정식의 초기조건이 결정된다.

$$\mathbf{x}(t_0) = \mathbf{x}_0$$

[수식 3-2]
초기조건

여기에서 구하고자 하는 것은 시간의 흐름에 따른 x(t)의 변화(즉, $\mathbf{x}(t_0), \mathbf{x}(t_1)$ ...)로, 결국 주어진 상미분방정식을 풀어 x(t)를 구하는 과정이라고 볼 수 있다. 수치해석 방법으로 상미분방정식의 해를 구하는 방식으로는 오일러(Euler) 방식, Midpoint 방식, 고차원 Runge-Kutta 방식 등이 있다. 매우 빠른 속도로 풀어야 하는 경우엔 보통 오일러 방식을 많이 사용하고 속도는 느리더라도 아주 정밀하

게 풀어야 할 필요가 있을 경우에는 고차원 Runge-Kutta 방식을 많이 사용한다. Midpoint 방식은 2차원 Runge-Kutta 방식이라고 할 수도 있다.

가. 오일러 방법

오일러 방법은 가장 간단한 수치해석기반 상미방해법이다. 이 방법은 테일러 급수에서 유도되었으며 비교적 큰 오차를 보인다. 다음과 같은 초기값 문제가 있다고 하자.

$$y'(t) = f(t, y(t)), \qquad y(t_0) = y_0$$

[수식 3-3] 초기값 문제의 예

테일러 급수에서 처음 두 항은 초기 위치 $(t_0, y(t_0))$의 위치에서의 선형 근사치를 나타내는데, 이러한 테일러 급수의 성질을 이용하면 다음과 같이 y_1의 근사값을 구할 수 있다.

$$y_1 = y_0 + hf(t_0, y_0). \quad \text{단}, t_1 = t_0 + h$$

[수식 3-4] 테일러 급수를 이용한
상미방 해의 근사치

여기서, h는 step size로 작은 양수이다. 이를 일반화하면 다음과 같다.

$$y_{n+1} = y_n + hf(t_n, y_n). \quad \text{단}, t_{n+1} = t_n + h$$

[수식 3-5] 상미방 해의 근사치의 일반화

여기서, $f(t_n, y_n)$은 y_n에서의 tangent 임을 쉽게 알 수 있다. 이와 같이 테일러 급수의 처음 두 항을 이용해서 일차원적으로 상미분방정식을 푸는 방식을 오일러 방법이라 한다. 오일러 방법은 기본적으로 오차가 많은 방법으로 h가 커질수록 오차가 증가하고 0에 가까울수록 오차가 감소한다.

나. Midpoint 방법

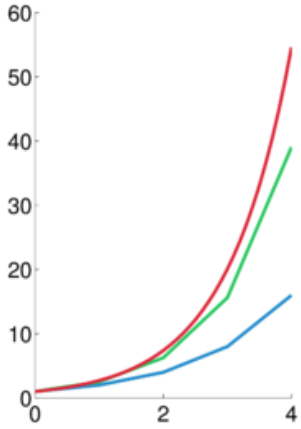
오일러 방법은 계산이 간단하지만 초기 위치에서의 tangent만을 이용하기 때문에 오차가 많은 문제가 있다. 즉, $[t_n, t_{n+1}]$에서 tangent는 연속적으로 변화하는데 오일러 기법에서는 초기의 tangent만을 이용해서 advection하기 때문에 오차가 발생하는 것이다. 이러한 문제를 보완하기 위해 초기 위치뿐만 아니라 중간 위치에서의 tangent도 함께 이용해서 계산하는 방법이 midpoint 방법이다. 이 방법에서는 초기 위치에서의 tangent를 이용해서 실제 interval의 중간 위치를 파악하고 중간위치에서의 tangent로 실제 계산을 수행한다. 오일러 방법에서 예로 들었던 상미분 방정식에 대한 Midpoint 방법에서의 y_n의 근사값은 다음과 같다.

$$y_{n+1} = y_n + hf\left(t_n + \frac{h}{2}, y_n + \frac{h}{2} f(t_n, y_n)\right)$$

[수식 3-6] Midpoint 방법을 이용한
상미방 해의 근사치

즉, 초기위치에서의 tangent인 $f(t_n, y_n)$을 이용해서 y_n과 y_{n+1}과의 중간 위치를 구하고 그 위치에서의 tangent를 구한 뒤 $\left(f\left(t_n + \frac{h}{2}, y_n + \frac{h}{2} f(t_n, y_n)\right)\right)$, 이를 이용해서 y_{n+1}을 구하는 것이다.

Midpoint 방법은 오일러 방법에 비해 상대적으로 정확하고 h값이 작아질수록 정확한 계산에 근접하는 속도가 오일러 방법에 비해 빠르다. 그림 3-1은 Midpoint 방법과 오일러 방법 그리고 정확한 계산을 비교한 그림이다. 붉은 색은 정확한 상미분방정식 계산에 의해 궤적을 구한 것이고 녹색은 midpoint 기법, 청색은 오일러 기법을 이용해서 궤적을 구한 것이다. 여기서 사용한 상미분 방정식은 수식 3-7과 같다.



[그림 3-1] 오일러 방법과 midpoint 방법의 비교

$$y' = y, \quad y(0) = 1$$

[수식 3-7] 상미분 방정식 예

다. Runge-Kutta 방법

Midpoint 방법은 오일러 방법에 비해 정확한 편이지만 여전히 오차가 작지 않다. Runge-Kutta 방법은 이러한 문제를 해결하기 위해 $[t_n, t_{n+1}]$ 내의 여러 tangent 들을 이용해서 y_{n+1}을 근사화하는 방법이다. 이 방법은 1900년경에 독일 수학자인 C. Runge와 M.W. Kutta에 의해 개발되었다. Runge-Kutta 방법에서 사용하는 tangent의 개수에 따라 차수가 정해진다. 즉, n개의 tangent를 사용하면 n차 Runge-Kutta 방법이 되는 것이다. Midpoint 방법은 사실 2차 Runge-Kutta 방법의 또 다른 이름이다. 위에서 예로 들었던 상미분방정식에 대한 4차 Runge-Kutta 근사값은 수식 3-8과 같다.

$$y_{n+1} = y_n + \frac{1}{6}h(k_1 + 2k_2 + 2k_3 + k_4)$$
$$t_{n+1} = t_n + h$$

[수식 3-8] 4차 Runge-Kutta 방법의 해

여기서, k_1, k_2, k_3, k_4는 수식 3-9와 같다.

$$k_1 = f(t_n, y_n)$$
$$k_2 = f\left(t_n + \frac{1}{2}h, y_n + \frac{1}{2}hk_1\right)$$
$$k_3 = f\left(t_n + \frac{1}{2}h, y_n + \frac{1}{2}hk_2\right)$$
$$k_4 = f(t_n + h, y_n + hk_3)$$

[수식 3-9] 4차 RK 방법에서의 k_1, k_2, k_3, k_4의 값

4. 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 데이터와 곤충 날개 데이터로 두 데이터 모두 structured grid로 총 6개의 멀티 블록(multi block)으로 구성되어 있다. 두 데이터의 블록별 정보는 표 4-1 및 4-2와 같다.

[표 4-1] KTX 데이터 설명

	Dimension	Cell 크기	크기
Block 1	12 x 53 x 48	30,528	0.93 MB
Block 2	132 x 47 x48	297,792	9.09 MB
Block 3	187 x 31 x 59	342,023	10.44 MB
Block 4	72 x 37 x 48	127,872	3.90 MB
Block 5	72 x 17 x 48	58,752	1.79 MB
Block 6	17 x 31 x 48	25,296	0.77 MB
Total		882,263	29.92 MB

[표 4-2] 곤충날개 데이터 설명

	Dimension	Cell 크기	크기
Block 1	225 x 113 x 97	2,466,225	131.71 MB
Block 2	225 x 113 x 17	432,225	23.08 MB
Block 3	97 x 113 x 17	186,337	9.95 MB
Block 4	225 x 113 x 97	2,466,225	23.08 MB
Block 5	97 x 113 x 17	186,337	9.95 MB
Block 6	225 x 113 x 97	432,225	131.71 MB
Total		6,169,574	329.49 MB

전처리 과정에서 KTX 및 곤충 날개 데이터의 팔진트리 및 kd-트리를 생성했는데, 이때 소요된 시간 및 메모리 사용량은 표 4-3과 같다.

[표 4-3] 팔진트리 및 kd-트리 생성 속도 및 메모리 사용량

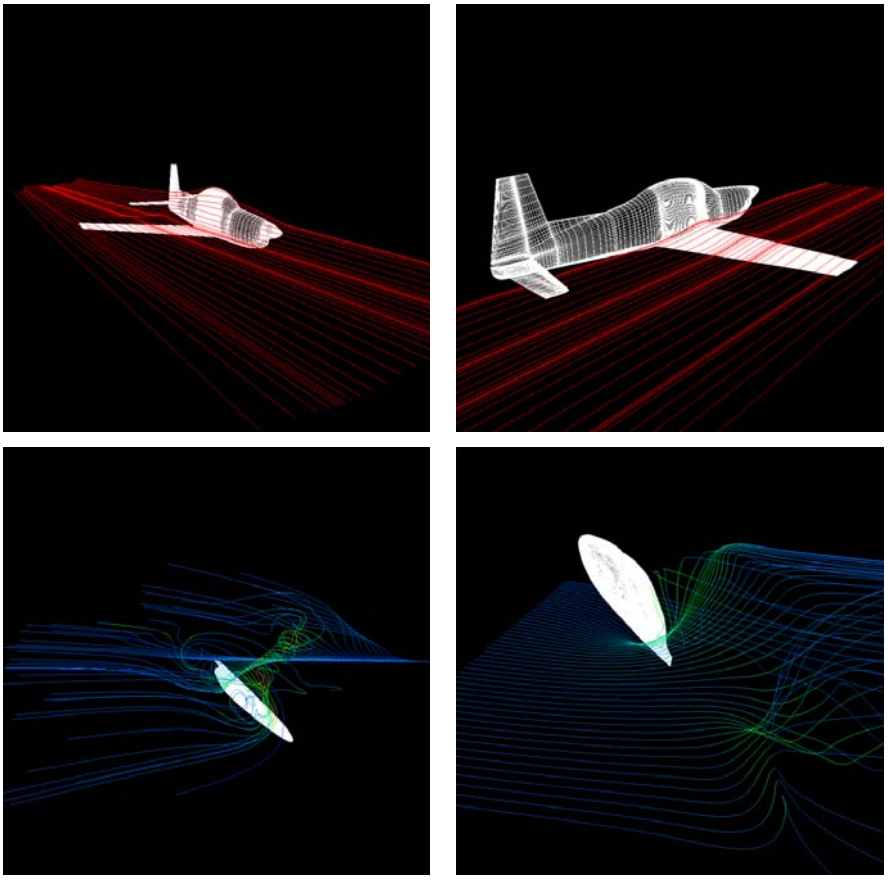
데이터	팔진트리		kd-트리	
	생성 속도 (초)	메모리 사용량 (MB)	생성 속도 (초)	메모리 사용량 (MB)
KTX	0.09	77.80	10.85	113.96
곤충날개	6.27	532.23	114.08	819.00

팔진트리 기법의 경우 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. kd-트리의 경우 생성 속도 및 메모리 사용량 모두 팔진트리 방법에 비해 많이 필요했는데, 이는 structured grid의 데이터를 unstructured grid로 변환한 뒤 생성했기 때문으로 보인다. 전체 데이터에 대해 10, 50, 100, 1000개의 스트림라인을 생성하는 총 4종류의 실험을 수행했는데, 각 스트림라인을 생성하기 위한 파티클 advection는 파티클이 도메인 밖으로 나가지 않는 한 400번씩 forward 및 backward로 실행했다. advection 알고리즘은 4차 Runge-Kutta 기법을 사용했다.

[표 4-4] 팔진트리 방법과 kd-트리 방법의 속도 비교 (단위 :초)

스트림 라인 수	KTX		곤충날개	
	팔진트리	kd-트리	팔진트리	kd-트리
10	0.47	0.42	1.13	0.18
50	3.00	2.66	5.75	0.93
100	6.55	5.56	11.49	1.85
1000	71.67	60.56	114.18	18.31

각 데이터에 대한 스트림라인 가시화 이미지는 그림 7-1과 같다.



[그림 7-1] KTX 및 곤충날개 데이터 스트림라인 가시화 결과

5. 결론

본 기술문서에서는 계산공간을 기반으로 생성한 팔진트리 및 kd-트리를 이용하여 structured grid 상의 벡터 데이터에서 빠르게 스트림라인을 생성했다. 특히, kd-트리의 경우에는 structured grid 상의 데이터를 unstructured grid로 변환한 뒤 트리를 생성했다. 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 structured grid 데이터들은 이러한 크기를 가지지 않으므로 이러한 데이터 구조에 최적화된 다른 계층 트리를 사용할 필요도 있다. 본 기술 문서에서는 이를 위해 kd-트리를 사용했다. 계층트리를 이용해서 셀 탐색을 빠르게 한다 하더라도 데이터의 크기가 매우 커지거나 파티클 advection 기법으로 4차 이상의 RK 기법을 이용하면 계산량이 많아져서 스트림라인 생성속도가 느려지게 된다. 이를 해결하기 위한 방법으로는 GPU를 이용해서 파티클 advection 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.

6. 참고문헌

[1] "Particle Tracing Algorithms for 3D Curvilinear Grids", Ari Sadarjoen, Theo van Walsum, Andrea J.S. Hin and Frits H. Post, Proceedings of Fifth Eurographics Workshop on Visualization in Scientific Computing, Rostock, Germany, 1994

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

[3] "An Efficient Point Location Method for Visualization in Large Unstructured Grids", M. Langbein, G. Scheuermann and X. Tricoche, Proceedings of Vision, Modeling, Visualization 2003, 2003