GPU Library CUDA를 이용한 효율적인 Delaunay 격자 생성에 관한 연구

송 지 홍[1], 강 상 현[1], 김 경 민[1], 김 병 수[2*]

A STUDY OF THE APPLICATION OF DELAUNAY GRID GENERATION ON GPU USING CUDA LIBRARY

J.H. SONG, S.H, KANG, G.M. KIM, B.S. KIM

In this study, an efficient algorithm for Delaunay triangulation of a number of points which can be used on a GPU-based parallel computation is studied. The developed algorithm is programmed using CUDA library, and the program takes full advantage of parallel computation which are concurrently performed on each of the threads on GPU. The results of partitioned triangulation collected from the GPU computation requires proper stitching between neighboring partitions and calculation of connectivities among triangular cells on CPU. In this study, the effect of number of threads on the efficiency and total duration for Delaunay grid generation is studied. And it is also shown that GPU computing using CUDA for Delaunay grid generation is feasible and it saves total time required for the triangulation of the large number points compared to the sequential CPU-based triangulation programs.

Keywords: Incremental Delaunay Triangulation, GPU(Graphic Processing Unit), CUDA(Compute Unified Device Architecture), 격자 생성(Grid Generation), 병렬 계산(Parallel Calculation)

1. 서 론

전산유체역학에 의한 유동 해석을 위해서는 해당 유동장을 유한한 숫자의 계산점, 즉 계산격자로 대체하여야 한다. 이러한 계산 격자는 그것 자체가 전산유체역학의 최종 목표가 되지는 않지만, 계산 격자의 종류나 질이 전산유체역학의 최종 목표인 유동해의 정확도, 수렴성, 그리고 계산 시간 등에 매우 큰 영향을 미치는 중요한 요소임은 잘 알려진 사실이다. 전산유체역학이 공학적 해석 도구로서 뿐만이 아니라, 설계 도구로서의 그 활용도가 점점 높아지고 있는 추세에 비하여, 아직도 격자 생성 과정은 전산유체역학 적용의 병목점으로 인식되고 있다. 특히, 3차원 볼륨 렌더링은 Delaunay 삼각화의 한 분야로 3차원 공간에 있는 스칼라형태로 분포되어 있는 불규칙한 점들의 집합을 시각화하는 것인데, 일반적으로 볼륨 렌더링을 하기 위한 점 데이터의 집합은 수십만 개에 달하므로 빠른 렌더링을 위해서는 효율적인 Delaunay 삼각화가 필수이다. 비효율적인 알고리즘을 사용하였을 경우 Delaunay 삼각화의 실행시간은 데이터의 양에 비례하여 기하급수 형태로 증가하는 경향을 볼 수 있다. [1]

GPU(Graphic Processing Unit)는 많은 양의 계산을 처리하기 위하여 특화된 멀티 쓰레드, 멀티 코어 기반의 병렬 처리 프로세서로, CPU에 비하여 저렴하면서도 수백 개의 코어를 내장하고 있어 데이터 처리 속도나 메모리 대역폭 등의 탁월한 성능을 보인다. 최근 NVIDIA에서는 CPU의 성능을 능가하는 GPU를 개발하였으며 CUDA(Compute Unified Device Architecture)을 이용하여 GPU환경에서의 연산을 가능하게 하였다.

본 논문에서는 CUDA를 이용하여 3차원 공간에 스칼라형태로 분포되어 있는 불규칙한 점들을 Delaunay 삼각화 알고리즘을 이용하여 격자를 생성하여 일반적인 CPU와 여러 개의 노드를 사용, 그리고 GPU를 사용하여 보다 효율적인 Delaunay 삼각화를 비교 연구하였다.

1 정회원, 충남대학교 대학원 항공우주공학과
2 정회원, 충남대학교 항공우주공학과
* Corresponding author E-mail: kbskbs@cnu.ac.kr

2. 관련 연구

2.1. Delaunay Triangulation

Delaunay 삼각화 알고리즘은 매우 다양하며 각 알고리즘에 따라 생성속도가 차이를 보이는데, 현재까지 연구되어지고 있는 Delaunay 삼각화 알고리즘에 대한 정의는 다음과 같다. [2]

2.1.1 Delaunay 삼각화의 정의

2차원 평면상에 점집합 P = {P_1 , P_2, ...,P_n}이 존재한다면 점집합 P의 삼각화를 정의할 수 있다. P의 점들을 직선으로 연결할 때에 평면성을 만족시키면서, 이미 연결된 직선과 새로운 직선이 교차되는 일이 없도록 두 점을 새로운 직선으로 연결하여 더 이상 두 점을 연결할 수 있는 직선이 존재하지 않을 경우 이를 점집합 P의 삼각화라 한다.

2.1.2 2차원 Delaunay 삼각화의 정의

2차원 평면상의 점집합의 Delaunay 삼각화는 다음과 같은 성질을 만족하는 삼각형들의 집합이다.

- 삼각형 꼭지점들의 집합은 2차원 평면상의 점집합으로 구성된다.
- 두 삼각형들의 교집합은 공집합 또는 하나의 공유점 또는 고유 모서리이다.
- 삼각형을 외접하는 원 내부에는 P에 속하는 다른 어떠한 점도 존재할 수 없다.(Fig. 1)

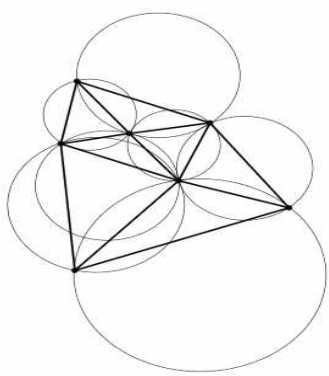


Fig. 1. Empty Circle of the Delaunay Triangulation

2.2 Incremental Delaunay Triangulation(IDT)

기존 삼각형화 구조에서 새로운 점을 추가하면서 기존의 삼각형화 결과를 수정하는 형식으로 알고리즘이 진행된다. 새로운 점이 추가되면, 기존의 삼각형을 검색하여 삼각형의 외접원이 삼각형의 세 점을 포함하는 경우를 다 찾으면 이들 꼭지점의 집합은 하나의 폴리곤을 형성하게 된다. 내부의 변들은 모두 제거하고, 새로운 점과 폴리곤의 꼭지점을 이어서 삼각형을 만들면 새로운 삼각형화 구조를 얻게 된다. 이 알고리즘은 초기에 전체의 점을 아우르는 큰 삼각형을 하나 구성하여야 하는데 이를 슈퍼 삼각형이라고 한다. 이 슈퍼 삼각형을 시작으로 해서 점을 하나씩 삽입하면서 삼각형을 만들어 나간다.[3]

2.3 CUDA(Compute Unified Device Architecture)

NVIDIA에서 개발한 CUDA는 그래픽 처리 장치(GPU)에서 수행하는 알고리즘을 코딩하는데 있어서 리눅스와 윈도우 환경에서 기존 C 컴파일러와 연동되는 방식으로 동작한다. 이는 기존의 CPU와는 달리 많은 수의 프로세서를 포함하고 있는 GPU를 이용하여 병렬 연산을 더욱 가속화하게 한다. 따라서, CPU 상의 연산 능력보다 수십 배의 성능 향상이 예상된다.

GPU를 원활하게 사용하기 위하여 Thread를 이용을 해야 하는데 Thread는 Grid와 Block을 조절하여 Thread를 수를 조절할 수 있는데 Grid와 Block의 정의는 다음과 같다.

- Grid : 2차원까지 관리할 수 있으며 kernel 함수를 호출할 때 앞에 들어가는 인자
- Blcok : 3차원 관리할 수 있으며 kernel 함수를 호출 할 때 뒤에 들어가는 인자.

주로 Block의 값을 초기화 해주고 Grid는 Block의 값을 사용해서 지정하고 Block안의 Thread수는 최대 512개로 정해져 있고 2차원으로 최대 Thread 수를 지정하려면 32*16, 3차원일때는 16*16*2와 같이 쓸 수 있다.

3. CPU 격자 생성

3.1 삼각화 알고리즘 선택

순차적 알고리즘과 병렬 알고리즘의 연산속도 차이를 나타내기 위해 기존에 나와 있는 알고리즘 중 속도가 빠른 incremental delaunay triangulation을 선택하여 격자를 생성하였다.[3]

4. GPU 격자 생성

4.1 CUDA 실행과정

4.1.1 Making Random Points

각 노드의 위치좌표 x, y를 무작위로 설정한다. 매번 같은 위치좌표를 얻기 위해 OS에서 제공해주는 랜덤 테이블을 이용하여 임의의 노드들을 생성하였다.

4.1.2 Node Range Calculation

생성된 노드들이 위치하는 범위를 계산하여 알아낸다.

4.1.2 Node Arrange for Each Grid

생성된 노드들의 인덱스를 아래의 그림((Fig. 2)과 같이 각 그리드에 맞게 재 정렬 시킨다. 초록색의 노드는 0번의 그리드에 속해 있으며 인덱스는 0에서부터 5번까지 부여된다.

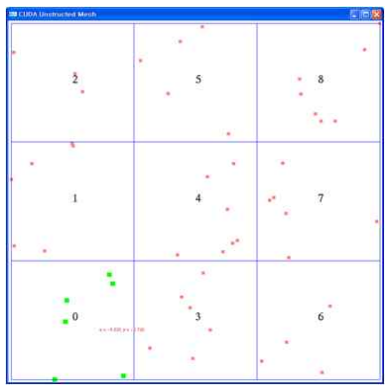


Fig. 2. Node Arrange for Each Grid

4.1.3 Parallel Calculate on Using CUDA

CUDA Library를 이용하여 병렬 계산을 하는 단계이다. 그리드의 개수만큼 쓰레드를 생성하고 하나의 그리드는 하나의 프로세서로써 연산을 수행한다. 아래의 그림(Fig. 3)과 같이 하나의 그리드는 8개의 이웃한 그리드를 가지게 되며 이웃한 그리드의 노드와 자신의 노드를 이용하여 삼각화가 이루어진다. 만약 새로 생성된 격자의 에지가 이웃한 그리드에 생성이 되면 삼각화를 중단한다. 아래의 그림((Fig. 3)은 위와 같은 알고리즘을 통하여 그리드의 격자를 나타낸 것이다.

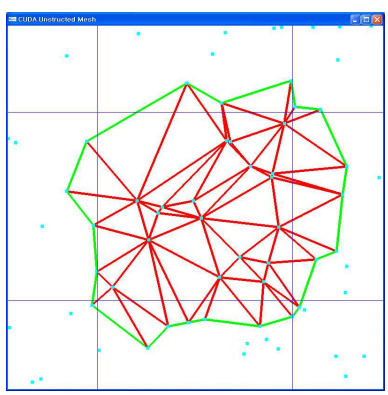


Fig. 3. Parallel Triangulation

4.1.4 Edge Swapping of Outer Boundary Cell

CUDA Library를 이용한 병렬화 작업 후 격자의 이상 유무를 확인를 해보면 아래의 그림(Fig. 4)과 같이 이웃한 그리드의 최외각 격자와 현재 자신의 격자가 서로 교차하는 현상이 발생 할 수도 있는 것을 확인하였다. 이러한 문제점이 발생하는 원인은 하나의 외접원 라인에 4개의 노드가 존재하며 그 외접원 내부에 다른 노드들이 없기 때문이다.

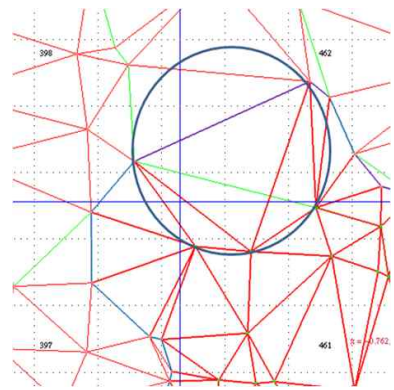


Fig. 4. Crossed Edge

또한, 그리드마다 독립적으로 삼각화를 수행하기 때문에 이웃한 그리드에서 반드시 같은 셀을 그려주지 않으므로, 아래의 그림(Fig. 5)와 같은 Edge Swapping은 필수적으로 수행하여 격자를 정리해주어야 한다.

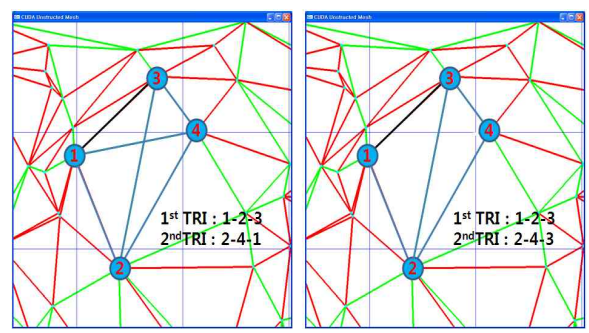


Fig. 5. Edge Swapping

4.1.5 Duplicated Triangles Remove

본 논문에 적용된 알고리즘에서 격자를 구성하는 세 노드의 그리드 정보가 모두 같다면 그 격자는 중복될 가능성이 없는 격자이다. 아래의 왼쪽 그림(Fig. 6)에서 초록색으로 그려진 격자는 중복될 가능성이 있는 격자들을 나타낸 것으로 격자의 구성 요소를 비교하여 같은 것을 찾은 후 중복되어진 격자를 제거하도록 해 주면 아래의 오른쪽 그림(Fig. 6)과 같

은 결과를 얻게 된다.

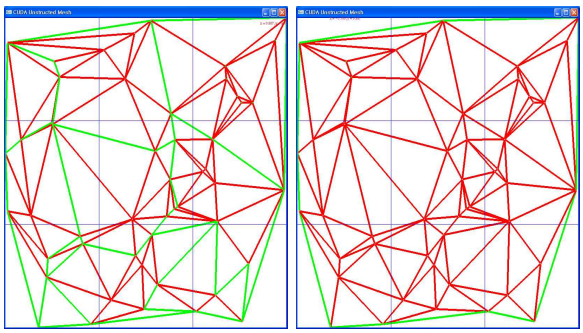


Fig. 6. Remove Duplicated Triangles

5. 테스트 결과 비교

5.1 Test Environment

가장 많이 사용하는 intel 계열의 PC를 선정하였으며 그래픽카드 또한 가장 많이 사용하는 NVIDIA GeForce 계열로 선정하였다. 자세한 사양은 아래 테이블을 참조한다.

Table 1. Computer Spec.

CPU	Inter(R) Core(TM)2 Quad 2.50GHz
RAM	3.25GB
OS	Window XP SP2 32bit
Graphic	Geforce GTX260

5.1.1 Test Cases

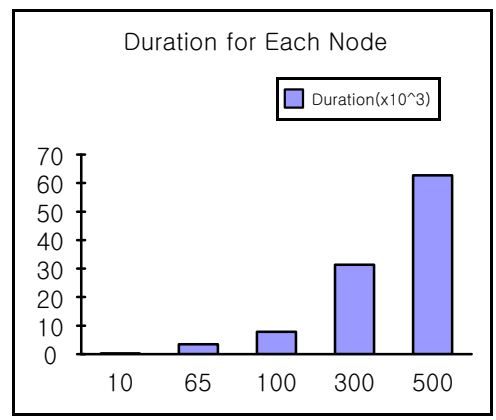
임의의 노드 개수는 10,000, 65,536, 100,000, 300,000 그리고 500,000개로 선정 하였다. 본 환경에서는 그래픽 카드가 CUDA를 위해 개발된 TESLA 계열이 아니기 때문에 더 많은 개수에 대해서는 CUDA의 run-time 제약을 받게 된다. 만약 CUDA 연산시간이 길어지면 제약으로 인해 컴퓨터가 다운 혹은 "launch time out" 에러 메시지와 함께 프로그램이 종료되는 현상이 발생 하게 된다. 그래픽 카드는 기본적으로 모니터에 비디오 데이터를 보내기 위한 장비로써 모니터의 화면 갱신율에 맞게 하드웨어적인 동기를 맞추고 있다. 만약 그래픽 프로세서에서 동작하는 CUDA 프로그램의 연산시간이 길어지게 되면 하드웨어적으로 그래픽 출력의 비안정성(픽셀의 깨짐 혹은 화면 정지 현상 등)을 막기 위해 CUDA 프로세서를 강제로 종료시키는 현상이 발생하기 때문에 50만개 이상은 테스트에서 제외를 시켰다.

5.1.2 CPU 연산결과

앞서 설명한 바와 같이 CPU 연산에서는 순차처리 알고리즘을 이용하고 반복문의 차수가 2차인 IDT를 사용하였다. 각각의 경우 대한 결과는 아래의 테이블을 참조한다.

Table 2. Duration for Node

Node	Duration
10,000	0.260(sec)
65,536	3.464(sec)
100,000	7.808(sec)
300,000	31.323(sec)
500,000	62.657(sec)



5.1.3 GPU 연산결과

CUDA는 블록과 그리드의 개수를 설정하여 사용할 쓰레드의 개수를 선정한다. 본 연구에서는 비교적 다양한 블록과 그리드를 설정하여 테스트 해 보았다.

본 논문의 4장에서 언급한 것과 같이 한 사이클의 테스트를 진행함에 있어서의 어느 순서에 있어서 시간이 소비되는지 확인하기 위하여 다음과 같은 범례를 만들어 시간 측정과 테스트를 진행하였고, 테스트를 통해 얻어진 결과들을 다음에 이어지는 그림(Fig. 7 ~ Fig. 11)들과 같이 Stack Column Chart를 이용하여 나타냈다. 각각의 차트의 몇 가지 범례들이 존재하는데 그 범례의 뜻은 다음과 같다.

1) ND : Nodes Divded
2) CUDA : CUDA Library 의 병렬 계산 진행
3) ES : Edge Swapping
4) DCR : Duplicated Cell Remove

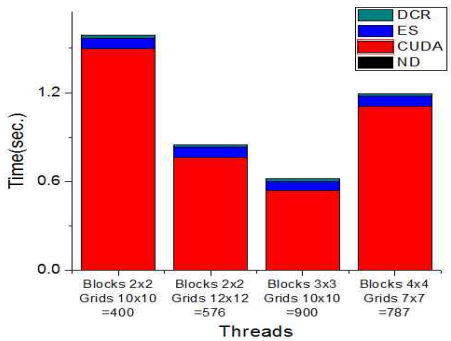


Fig. 7. Duration for 10,000 Nodes

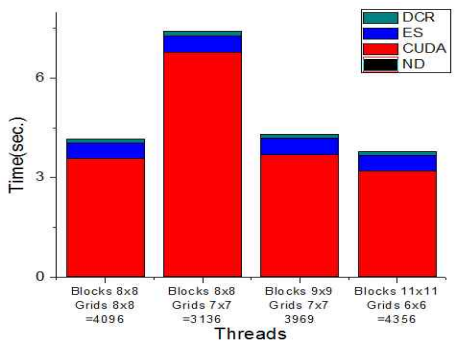


Fig. 8. Duration for 65,536 Nodes

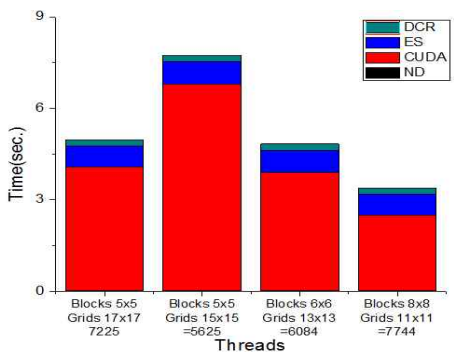


Fig. 9. Duration for 100,000 Nodes

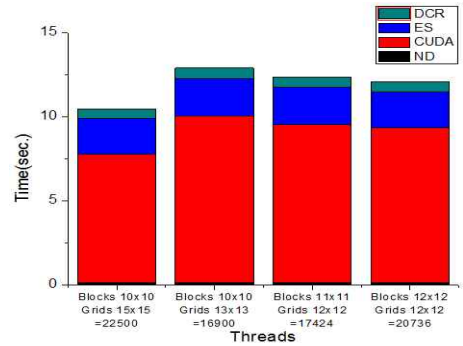


Fig. 10. Duration for 300,000 Nodes

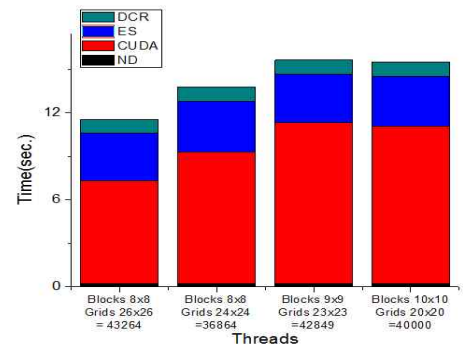


Fig. 11. Duration for 500,000 Nodes

5. 결 론

CPU와 GPU 모두 노드 개수가 증가함에 따라 격자가 만들어 지는 시간이 증가함을 보였다. 65,536개 이하에서는 단일 쓰레드의 CPU의 연산 속도가 CUDA보다 더 빠름을 보였다. CUDA의 경우 노드의 개수가 많아지면 많아질수록 연산 속도는 점점 GPU보다는 CPU의 영향을 더 많이 받는 것으로 보여진다.

참고문헌

[1] 1999, Kim B., “Automatic Multi-Block Grid Generation Technique Based on Delaunay Triangulation," 전산유체공학회 추계학술대회 논문집, pp.108~114.

[2] 2005, 박시형, “격자분할을 이용한 Delaunay 삼각화와 표면 재구성,” 박사학위논문, 건국대학교

[3] 2005, Zadravec, Mirko "An Almost distribution-independant incremental Delaunay Triangulation algorithm," The Visual computer v.21 no.6, pp.384-396