

3차원 삼각형 메쉬의 과장을 안정적으로 표현할 수 있는 필터링과 GPU 최적화

이수빈[○], 문성혁[○], 김종현^{*}

[○]강남대학교 소프트웨어응용학부,

^{*}강남대학교 소프트웨어응용학부

e-mail: jognhyunkim@kangnam.ac.kr

Filtering and GPU Optimization to Reliably Express the Exaggeration of 3D Triangular Meshes

SuBin Lee[○], Seong-Hyeok Moon[○], Jong-Hyun Kim^{*}

[○]School of Software Application, Kangnam University,

^{*}School of Software Application, Kangnam University

● 요약 ●

본 논문에서는 법선벡터를 이용해 3D 삼각형 메쉬의 형태를 안정적으로 과장하고 GPU 기반으로 새롭게 설계하는 프레임워크를 제안한다. 우리는 High-boost 메쉬 필터링 알고리즘에서의 Aliasing 문제를 양방향 필터를 적용하여 노이즈를 제거하고 GPU 기반에서 설계해 고속화한다.

키워드: 쿠다(Computed Unified Device Architecture, CUDA), 양방향 필터(Bilateral filter), 삼각형 메쉬(Triangular mesh), GPU 최적화(GPU optimization), 과장(Exaggeration)

I. Introduction

High-Boost Mesh Filtering은 영상과 신호 처리 과정에서 사용되는 샤프닝(Sharpening) 알고리즘을 기반으로 개발된 알고리즘이다. normal을 이용한 연산으로 3D triangle mesh가 기존과 근사한 형태로 강화될 수 있도록 한다. High-Boost Mesh Filtering을 적용한 결과물은 기존보다 형태가 강화된 모습을 보인다. 그러나 aliasing과 mesh irregularization이 발생하기 때문에 결과물에 Laplacian Smoothing을 적용하여 이를 보완해 주어야 한다. mesh의 개수에 따른 CPU에서의 연산 시간을 나타내는 Fig 1을 보면 알 수 있듯이 고해상도의 mesh일수록 연산 시간이 오래 걸린다는 단점이 있다.

본 논문에서는 High-Boost Mesh Filtering에서 사용하는 boosted normal에 양방향 필터를 적용하여 aliasing 문제를 해결한다. GPU를 활용한 병렬 처리로 알고리즘 속도를 개선한다. 논문의 구성은 다음과 같다.

2장에서는 High-Boost Mesh Filtering 알고리즘에 대해 자세히 설명한다. 3장에서는 aliasing 문제를 보완한 알고리즘과 이를 GPU 기반으로 구현한 결과를 기존의 High-Boost Mesh Filtering과 비교 분석 한다.

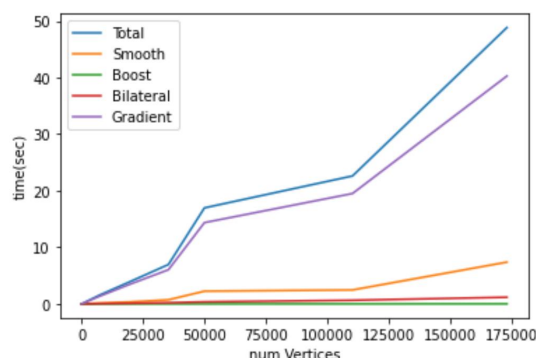


Fig. 1. calculation time for each function of High-Boost Mesh Filtering in CPU.

II. Preliminaries

2.1 High-boost mesh filtering

Hirokazu Yagou와 Belyaev, Daming Wei가 [1]에서 제안한 High Boost Mesh Filter의 알고리즘은 다음과 같다.

먼저, face T의 1-ring neighborhood faces U_i 의 normal들을

넓이 가중치를 준 mean filtering으로 평활화하여 smoothed normal 인 $n^{(k)}(T)$ 을 연산한다. k 는 smoothed normal 연산을 반복한 횟수를 의미한다.

$$n^{(k)}(T) = \frac{1}{\sum A(U_i)} \sum_{i \in N(T)} A(U_i) n(U_i) \quad (1)$$

그다음, face normal과 수식 (1)로 얻은 smoothed normal $n^{(k)}(T)$ 을 수식 (2)를 통해 boosted normal $m(T)$ 을 계산한다. α 는 boost threshold로 이를 이용해 형태를 강화하는 정도를 조절할 수 있다. [1]에서는 1.5로 α 를 설정한다.

$$m(T) = \frac{(1+\alpha)n(T) - \alpha n^{(k)}(T)}{\|(1+\alpha)n(T) - \alpha n^{(k)}(T)\|} \quad (2)$$

마지막으로, 수식 (3)과 같이 boosted normal과 face normal의 오차를 최소화하도록 Gradient를 계산하고 그 결과로 각 vertex의 위치를 이동하는 과정을 충분히 반복해 3d triangle mesh를 강화한다.

$$E_n = \sum_{i \in F_i(P)} A(R_i) (n(R_i) - m(R_i))^2 \quad (3)$$

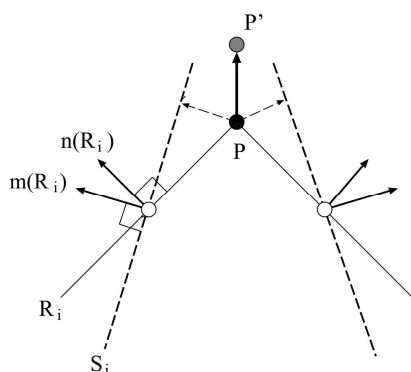


Fig. 2. normal-based error minimization in High-Boost Mesh Filtering

Fig 2는 오차를 최소화하는 알고리즘을 나타낸다. R_i 는 face를, S_i 는 boosted normal을 가지는 평면에 R_i 를 투영해 얻은 face를 의미한다. vertex P 의 1-ring neighborhood face에 대하여 R_i 와 S_i 의 넓이를 계산하고 둘의 차이를 최소화하는 gradient를 계산해 P 의 위치를 P' 로 이동한다.

$$P' \leftarrow P - \lambda \sum_i \left(\frac{\partial A(R_i)}{\partial P} - \frac{\partial A(S_i)}{\partial P} \right) \quad (4)$$

2.2 Bilateral filter

양방향 필터는 블러 알고리즘을 기반으로 Tomasi 와 Manduchi가 [2]에서 제안한 것으로 가우시안 필터의 단점을 보완했다. 주변 픽셀과의 거리와 픽셀 값의 차를 가중치로 사용하여 윤곽선은 보존하고

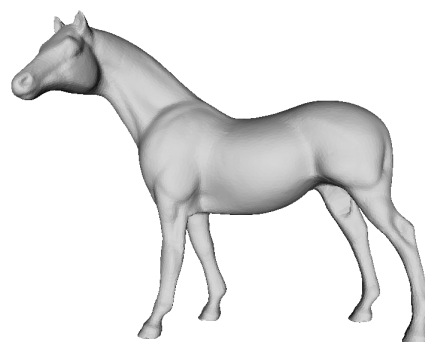
잡음은 제거한다.

$$I_p = \frac{1}{W_p} \sum_{q \in S} G_{\sigma_s}(\|p - q\|) G_{\sigma_r}(\|I_p - I_q\|) I_q \quad (5)$$

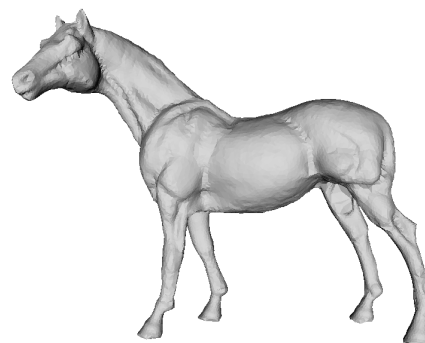
p 는 출력 픽셀의 좌표, q 는 p 의 주변 픽셀의 좌표를 의미하며 W_p 는 정규화 상수이다. 두 개의 가우시안 함수는 각각 p 와 q 간의 거리와 p 와 q 의 픽셀 값 차이로 계산한다. 가우시안 함수의 sigma로 블러 정도를 조절할 수 있다.

III. The Proposed Scheme

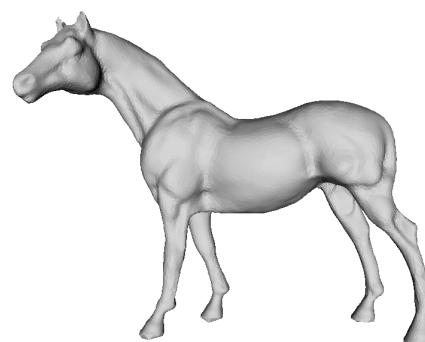
3.1 Antialiasing



(a)



(b)



(c)

Fig. 3. (a) horse original, (b) High-Boost Mesh Filtering, (c) High-Boost Mesh Filtering with Bilateral Filter

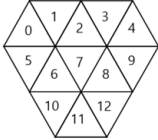
High-Boost Mesh Filtering의 단점 중 하나인 aliasing은 mesh의 형태를 강화할 때, 잡음도 함께 강화하면서 발생한다. 따라서 잡음은 제거하되 윤곽선을 보존하며 강화하기 위해 양방향 필터를 사용한다. High-Boost Mesh Filtering은 face의 boosted normal을 이용해 mesh의 형태를 강화하기 때문에 양방향 필터를 boosted normal에 적용한다. 양방향 필터를 적용할 범위는 각 face에 대하여 1-ring neighborhood faces로 설정하고 수식 (5)의 주변 픽셀과의 거리와 픽셀 값의 차이 대신 주변 triangle face의 중심 좌표 간의 거리와 boosted normal 값의 차이를 각각 가우시안 함수로 계산한다.

Fig 2는 horse 모델을 이용해 양방향 필터 적용 여부에 따른 High-Boost Mesh Filtering 결과를 비교한다. (b)와 (c)는 같은 조건에서 High-Boost Mesh Filtering을 적용한 결과이다. 둘 다 원형인 (a)보다 강화된 결과를 보이지만 잡음도 함께 강화된 (b)보다 양방향 필터로 윤곽선은 강화하며 잡음은 제거한 (c)가 더 안정적인 결과를 보인다.

3.2 High-Boost Mesh Filtering GPU 기반 구현

High-Boost Mesh Filtering의 전체 연산 시간을 각각 smoothed normal 연산 함수, boosted normal 연산 함수, 양방향 필터 함수, Gradient 연산 함수로 나눠보면 Fig 1과 같다. Gradient 연산 함수가 가장 많은 시간을 차지하고 smoothed normal 연산 함수가 두 번째로 많은 시간을 차지하며 고해상도 mesh일수록 연산 시간이 더욱 증가한다. 대부분의 시간을 차지하는 Gradient 연산 함수와 Smoothed normal 연산 함수만 GPU로 구현할 경우, CPU와 GPU 간에 데이터를 복사하는 과정이 증가해 비효율적이다. 따라서 High-Boost Mesh Filtering 알고리즘 전체를 GPU에서 구현한다. CPU와 GPU 간의 데이터 복사는 연산에 필요한 mesh 데이터를 CPU에서 GPU로 복사하는 초기 단계와 Gradient 연산까지 모두 마친 후 변경된 위치 데이터를 GPU에서 CPU로 복사하는 마지막 단계에서만 일어난다.

High-Boost Mesh Filtering 연산에 필요한 데이터 중 1-ring neighborhood faces 데이터는 각 face 및 vertex마다 데이터의 크기가 달라 GPU로 옮기기 까다로운 데이터이다. GPU에서 사용할 face와 vertex의 1-ring neighborhood faces 데이터를 Hash Table을 이용해 구현한다. Fig 4는 GPU에서 face에 대한 1-ring neighborhood faces 데이터 구조를 나타낸다. face의 1-ring neighborhood faces 데이터 전체를 하나의 배열에 저장한다. face의 index를 키, 데이터의 시작과 끝 위치를 값으로 한다. GPU에서 Gradient 연산을 할 때, 연산할 face의 index 값으로 데이터의 시작과 끝 위치를 알아내 이웃 정보가 저장된 배열에서 해당하는 위치에 있는 데이터를 사용한다.



face Index	Data Start	Data End
0	0	5
1	5	11
2	11	20
3	20	26
4	26	31
5
6		
7		
8		
9		
10		
11		
12		

Index	1-ring NbFaces
0	1
1	2
2	7
3	6
4	5
5	3
6	2
7	7
8	6
9	5
10	0
11	3
...	...

Fig. 4. Hash Table Data Structure of 1-ring neighborhood faces

smoothed normal, boosted normal, bilateral filter을 연산하는 커널에 사용하는 총 스레드 개수는 face 개수, gradient를 연산하는 커널에 사용하는 총 스레드 개수는 vertex 개수와 동일하다.

3.3 결과 분석

High-Boost Mesh Filtering은 normal과 boosted normal의 오차가 최소가 되는 위치로 vertex를 이동시킬 뿐 vertex의 개수를 늘리거나 이동 시 주변과의 관계를 고려하지 않는다. 따라서 irregularization은 여전히 발생하고 이를 해결하기 위한 smoothing 과정이 필요하다. smoothing을 사용하지 않을 때 High-Boost Mesh Filtering은 저해상도 보다 고해상도에서 더 좋은 결과를 보인다.

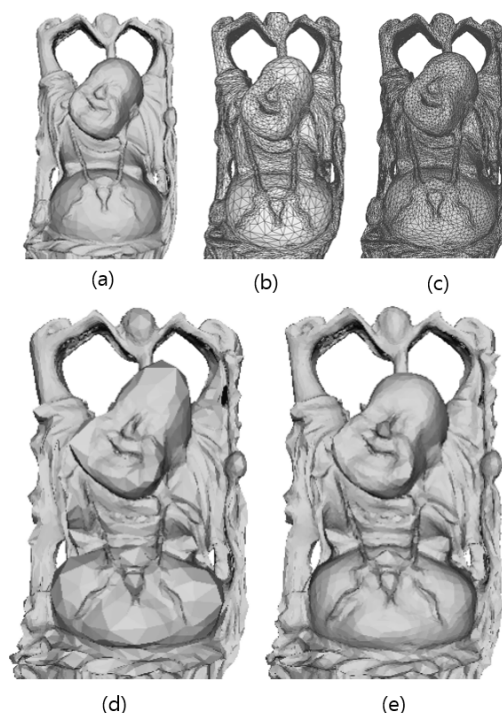


Fig. 5. (a) buddha original (b) vertex:49,990 face:100,000
(c) vertex:67,031 face:134,082
(d) High-Boost Mesh Filtering of (b)
(e) High-Boost Mesh Filtering of (c)

Fig 5는 buddha 모델을 사용해 해상도에 따른 결과를 보여준다. (b)는 원본 buddha의, (c)는 (a)의 해상도를 Subdivision으로 높은 mesh 구조를 내며 (d)와 (e)는 각각 (b)와 (c)에 High-Boost Mesh Filtering을 적용한 결과이다. (b)에서 해상도가 낮은 buddha의 배와 얼굴 부분은 irregularization으로 인해 (d)와 같이 형태는 강화되지만 부자연스럽게 변하며 smoothing을 필요로 한다. 해당 부분의 해상도를 높은 (c)의 결과인 (e)는 smoothing 없이 (d)보다 자연스러운 결과를 낸다.

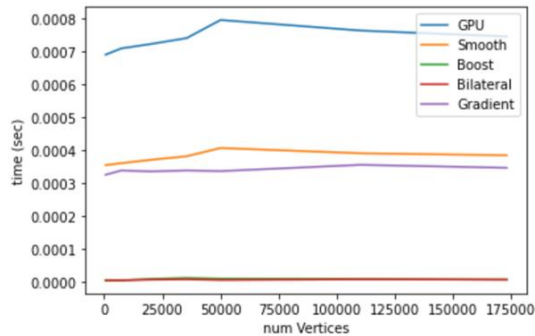


Fig. 6. calculation time for each function of High-Boost Mesh Filtering in GPU

Fig 6은 GPU 기반으로 구현한 High-Boost Mesh Filtering의 연산 시간을 나타낸다. smoothed normal 연산 반복 횟수를 100, Gradient 연산 반복 횟수를 100으로 설정해 측정한 결과이다. 평균 0.0006에서 0.0008초의 연산 시간이 소요되고 mesh의 해상도에 따른 차이가 크지 않다. CPU보다 연산 시간이 매우 단축되며 High-Boost Mesh Filtering을 mesh에 실시간으로 적용할 수 있을 정도로 빨라진다. 또한, mesh의 해상도에 따른 연산 시간의 차이가 크지 않음으로 고해상도 mesh 사용을 smoothing을 사용하지 않고 irregularization 문제를 보완하는 방법으로 고려할 수 있다.

IV. Conclusions

본 논문에서는 High-Boost Mesh Filtering 알고리즘에 양방향 필터를 적용하여 aliasing 문제를 해결하고 GPU를 활용한 병렬 처리로 알고리즘 속도를 개선하였다. 그 결과 CPU에 비해 실시간 적용이 가능할 정도로 연산 시간이 단축되었고 aliasing을 해결해 결과물이 더 자연스러운 형태로 강화되었다.

irregularization 문제는 해결되지 않아 smoothing 과정이 필요하다. 이는 단점이 남아있지만, GPU 기반으로 구현한 알고리즘은 해상도에 따른 연산 시간의 차이가 크지 않음으로 mesh의 해상도를 높여 이를 보완할 수 있다. 현재 High-Boost Mesh Filtering 알고리즘은 triangle mesh를 기반으로 하고 있지만 약간의 수정을 통해 quad mesh에서도 사용이 가능할 것으로 보인다.

REFERENCES

- [1] Hirokazu Yagou, A. Belyaev, Daming Wei, "High-Boost Mesh Filtering for 3-D Shape Enhancement".
- [2] C. Tomasi and R. Manduchi, "Bilateral filtering for gray and color images," in Proc. ICCV, pp. 839-846, Bombay, India, Jan. 1998.