Journal of Korea Game Society 2017 Feb; 17(1): 17-26
http://dx.doi.org/10.7583/JKGS.2017.17.1.17

게임프로그래밍

이동궤적 근사 다면체를 이용한 실시간 모션블러 기법

홍민푸, 최진형, 오경수
숭실대학교 미디어학과
{hmphuoc1985}@gmail.com, {jinham, oks}@ssu.ac.kr

Real-Time Motion Blur using Approximated Motion Trails

MinhPhuoc Hong, Jinhyung Choi, Kyoungsu Oh
Department of Media Soongsil University

요 약

최근 실시간으로 모션블러(motion blur)를 위한 연구들은 픽셀당 여러개의 시간 색상을 계산한 후 평균내는 방식으로 샘플의 수가 적을 경우 아티펙트(artifacts)나 노이즈(noise)가 발생하는 문제를 가지고 있다. 본 논문은 이러한 문제를 개선하기 위해서 이동궤적 근사 다면체(motion trail)를 이용한 실시간 모션블러 알고리즘을 제안한다. 본 논문의 알고리즘에서는 현재 프레임과 이전프레임의 삼각형으로 이동궤적 근사 다면체를 만들고 전후 관계(front-to-back) 정렬방법과 시공간차원의 비트연산(bitwise operation)을 적용하여 여러 물체가 겹치는 순간의 가시성 문제를 해결했다. 결과적으로 가려지지 않은 이동궤적 근사 다면체만을 그리기에 매끄러운 블러 효과를 얻는다.

ABSTRACT

Several algorithms have been introduced to render motion blur in real time by solving the visibility problem in the spatio-temporal domains. However, some algorithms render at interactive frame rates but have artifacts or noise. Therefore, we propose a new algorithm that renders real-time motion blur using extruded triangles. Our method uses two triangles in the previous and the current frame to make an extruded triangle then send it to the rasterization. To solve the occlusion between extruded triangles for a given pixel, we introduce a combining solution using a sorting in front to back order and bitwise operations in the spatio-temporal dimensions.

Keywords : Real-time rendering(실시간 렌더링), Motion blur(모션블러), Visibility(가시성), Shading(쉐이딩)

Received: 11, Jan, 2017 Revised: 18, Feb, 2017
Accepted: 20, Feb, 2017

Corresponding Author: Kyoungsu Oh(Soongsil Univeristy)
E-mail: oks@ssu.ac.kr
ISSN: 1598-4540 / eISSN: 2287-8211

1. 서 론

현실에서 카메라로 찍은 사진은 어느 단편의 순간을 나타낸다. 만약 사진을 찍는 순간 어떤 물체가 움직이고 있었다면, 사진에는 물체가 이동하는 방향으로 흐려지는 효과가 생길 것이다. 이것을 모션블러(Motion Blur) 이펙트라고 부른다. 모션블러는 영화와 같은 오프라인 렌더링에서 많이 사용되는 중요한 효과중 하나이다. 하지만 모션블러 효과를 실시간 렌더링에서 적용하기에는 시공간적 차원의 가시성 문제를 해결하기 쉽지 않고, 계산비용이 매우 많이 들기 때문에 현재 기술로도 여전히 어려운 상황이다[1]. 이전에 행해진 대부분의 연구들은 포인트 샘플링으로 가시성 결정 테스트를 처리했다. 하지만 포인트 샘플링 기법으로 적은 수의 샘플을 가지고 가시성을 결정한다면 결과에 노이즈가 낄 수 있고, 아티팩트들이 발생하는 문제가 있다. 고품질의 블러 효과를 얻기 위해서는 수많은 샘플들을 이용해야하는데, 여러 샘플들을 이용하면 성능이 급격하게 떨어질 우려가 있다.

블러된 이미지를 얻기 위한 몇몇의 연구들은 시공간적 바운딩 볼륨(bounding volume)을 이용했다. 바운딩 볼륨이 포함하고 있는 모든 픽셀과 볼륨의 모양, 크기 등은 어느 한 삼각형이 이전프레임부터 현재프레임까지 얼마나 이동했는지에 따라 달라진다. 본 논문에서는 이러한 기본적인 아이디어를 바탕으로 이동궤적 근사 다면체를 만든다. 본 논문의 알고리즘은 앞서 말한 이동궤적 근사 다면체를 이용하여, 어떤 한 픽셀에 대한 삼각형의 가시성을 GPU 시간에 따라 정확하게 판단하여 분석적으로 알아낸다. 그런 다음 데이터를 정렬하고 비트-연산을 사용하여 여러 이동궤적 근사 다면체들의 가시성 관계를 해결한다. 따라서 노이즈가 발생하지 않고 매끄러운 블러 이미지를 실시간으로 얻어 낼 수 있다.

본 논문의 다음 단락에서는 기존 연구들을 논할 것이고, 단락 3에서는 본 논문에서 제시한 알고리즘의 메인 아이디어를 다룰 것이다. 단락 4에서는 실험 및 결과를, 5에서는 결론을 논할 것이다.

2. 기존 연구

2.1 Post-Processing Approaches

전처리기반 [2,3,4] 알고리즘들은 특정 순간의 영상을 그리고 이를 필터로 필터링하여 모션블러 효과를 얻어 낸다. 이 방법은 한 순간에 보이는 물체들만 그릴 수 있으므로 가시성 문제가 발생한다.

이후, [5] 알고리즘은 스캐터/개더 (scatter - as-gather)를 이용하여 각 $k \times k$ 사이즈의 타일들에서 최대 크기를 갖는 속도 벡터를 찾아 저장함으로써 두 개의 중간 버퍼를 생성한다. 이 방법은 최대의 속도 벡터만을 가지고 샘플링하기 때문에, 다른 속도와 다른 방향으로 움직이는 몇몇의 물체에 대해서는 아티펙트가 발생하는 단점이 있다. 이러한 단점들은 [6]에 언급 돼있다. [7]의 방법은 최대 속도의 벡터 또는 픽셀의 속도 벡터들을 이용한 샘플링을 통해 [5]의 알고리즘을 개선하여, 좋은 품질의 블러 이미지를 얻어냈다. 하지만 이 방법 또한 문제를 완벽하게 해결하지 못했다.

대부분 알고리즘들은 물체들이 선형적인 움직임을 가진다고 가정했기 때문에 비선형 움직임이 있는 물체에 대해 아티펙트들이 발생할 수 있다. [8]은 비선형 움직임을 모션블러 이펙트 렌더링 한다. 라인들의 리스트는 각 라인의 고정된 색상들을 가지고 있고, 색상들을 이용해 커브 움직임을 표현한다. 추가적으로, 각 픽셀에 색상과 가중치를 저장하고, 스캐터 연산을 모두 저장하기 위해 상황에 따라 알파 블랜딩을 사용한다. 마지막으로 각 픽셀의 색상은 이전 프로세싱 패스에서 나온 다른 픽셀들의 기여도(contribution)에 따라 계산된다. 이 연구는 좋은 성능을 보여주지만, 가시성을 부분적으로만 처리하는 한계가 있다.

2.2 Brute-Force Approaches

[9,10]은 누적 버퍼링을 이용하여 각 픽셀에 대한 기하구조를 여러 번 렌더링하고 평균화 한다. 이 알고리즘에서 적은 양을 샘플링하게 되면 부드러운 블러 효과를 얻을 수 없게 되고, 샘플 수가 많아지게 되면 성능에 큰 영향을 미치기 때문에 실시간 렌더링에는 적합하지 않은 알고리즘이다.

2.3 Stochastic Sampling

[11]은 시공간 영역에서 광선(Ray)을 분배한 다음 가시성 결과를 통과한 모든 픽셀에 대한 색상들의 평균을 구한다. [12]에서의 알고리즘은 비슷한 방법을 이용하지만, 연속적인 시간 변화에 따른 삼각형을 이용한다는 것과 구현을 위해선 새로운 하드웨어를 필요로 하는 차이가 있다. 또 다른 알고리즘 [13]은 [12]에서 설명된 아이디어를 GPU에서 멀티 샘플 안티 알리아싱(multi-sample anti-aliasing)과 2D 스크린 공간의 컨벡스 헐(convex hull)또는 스페이스 타임 바운딩 볼륨(space-time bounding volumes)을 사용해서 구현했다. 각각의 샘플들은 움직이는 삼각형에 대한 위치를 알기위해 임의의 시간 정보를 가지고 있다. 특정 샘플의 가시성 판단은 하나의 광선과 삼각형의 교차테스트를 수행한다. 해당 픽셀의 마지막 색상을 결정짓기 위해 보이는 샘플들을 모두 처리하고 처리된 값들의 평균을 계산한다. 이 방법과 그 외 [13,14,15] 방법들을 이용하면 로우 샘플링 레이트(low samping rates) 때문에 노이즈가 발생할 수 있다.

2.4 Analytical Visibility

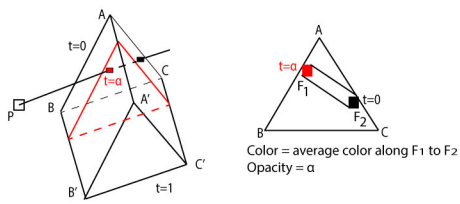
분석적 가시성 연구인 [10,16,17]의 알고리즘에서 영감을 받아 본 연구에 적용했다. [10]에 기술된 방법은 특정 해당 픽셀이 담을 수 있는 각 기하구조의 인터벌들(intervals)을 계산하는 방법을 처음으로 제시했다. 그 다음, 가려져 있는 인터벌을 처리하고 분석하여 최종 색상을 얻어낸다. 본 연구에서도 비슷한 방법으로 가려지는 상황의 해당 인터벌들을 제거하고, 추가적으로 깊이 정렬(depth-sorting)과 비트 연산(bitwise-operation)을 사용하여 각 이동궤적 근사 다면체의 인터벌의 시간을 찾아낸다. 본 논문에서 사용한 깊이 정렬알고리즘은 [16]에서 설명된 방법과 [17]의 폴리헤드론 클리퍼(polyhedron clipper)와 유사하다.

[18]에 서술된 분석적(Analytical) 모션블러는 공간과 시간의 영역차원에서 가시성 문제를 해결하기 위해서 분석적인 해결방법(analytically solved)을 제시했다. 하지만 이 알고리즘은 레스터화를 소프트웨어 단계에서 수행하기 때문에 GPU의 레스터라이저를 사용할 수 없다.

2.5 Other Approaches

그 외 참조된 연구로는 [19,20,21]처럼 가시성결과로 부터 샘플링을 분리한 (decoupling sampling) 방법, 그 외 [22,23,24]와 쉐이딩 재사용(reusing-shading) 방법이 있다.

3. 알고리즘



[Fig. 1] Color and opacity of an extruded triangle for a given pixel P. F_1 has $(t=\alpha, uv_1)$ and F_2 has $(t=0, uv_2)$

3.1 색상과 불투명도 계산

[Fig. 1]은 본 연구의 메인 아이디어를 보여준다. 한 삼각형이 이전프레임부터 $t=0(ABC)$ 현재프레임까지 $t=1(A'B'C')$ 직선적으로 움직인다고 가정하자. 픽셀 P에서 볼 때 t=0일때 F_2부분이 보이고 $t=\alpha$일 F_1이 보일 때 까지 F_2와 F_1사이부분이 보이고 그 후에는 보이지 않게 된다. 픽셀 P

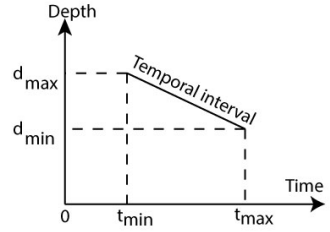
에 대해 이 삼각형의 불투명도와 색상을 생각해 보자. 삼각형의 불투명도는 삼각형이 픽셀 P에서 얼마나 오랫동안 보이는지에 따라 알 수 있다. 즉, 이 삼각형은 픽셀 P에서 시간 $t=0$에서부터 $t=\alpha$까지만 보일 것이기 때문에 이 삼각형의 불투명도는 α 라고 할 수 있다. 만약 픽셀 P에서 시간 $t=0$, $t=\alpha$ 해당하는 삼각형의 텍스쳐 좌표를 각각 $uv_{1,}uv_2$이라고 할 때. 이 삼각형의 텍스쳐 색상은 텍스쳐 좌표 $uv_{1,}uv_2$를 이어 샘플링한 색상의 평균으로 근사치를 얻어낼 수 있다.

위에 언급한 메인 아이디어를 바탕으로 이동궤적 근사 다면체를 만들어 모션블러 이미지를 실시간으로 얻어낼 수 있다. 이동궤적 근사 다면체는 시간 $t=0$, $t=1$에 해당하는 두 개의 삼각형을 이용해서 만든다. [Fig. 1]에서 각각의 정점(vertex)들은 시간정보 t와 텍스쳐 좌표를 가지고 있다. 이 이동궤적 근사 다면체는 한 삼각형이 이전프레임에서 현재프레임으로 이동하는 부분의 픽셀들을 차지한다. 삼각형기둥을 전체적으로 삼각형화 시켜 GPU의 다음 단계로 보낸다. 각 픽셀마다 GPU는 두 개의 점을 생성하고, 각각의 점들은 보간된 시간 정보(t)와 텍스쳐 좌표(u,v)를 가지게 된다. [Fig. 1]에서 두 개의 점 F_1, F_2을 예로 들 수 있다. 결과적으로 해당 픽셀에서 삼각형이 보이는 시간의 범위를 알 수 있다. 이후 위에서 설명한 방법으로 삼각형의 불투명도와 색상을 결정하는 과정을 거친다. 따라서 본 논문의 알고리즘은 분석적인 방법(analytical method)을 이용하여 가시성 문제를 해결했다.

3.2 정렬 및 블랜딩

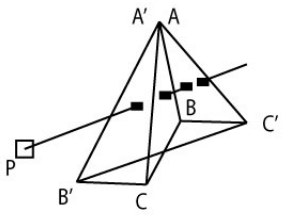
앞 절 3.1에서는 한 이동궤적 근사 다면체의 색상과 불 투명도를 어떻게 얻어내는지에 대한 설명을 하였다. 그러나 대부분의 경우는 어느 한 픽셀이 다수의 이동궤적 근사 다면체들을 담고 있기 때문에 블랜딩하기 이전에 기하구조(geometries)를 전후 순서(front-to-back order)로 정렬해야한다. 깊이와 시간 차원을 따졌을 때, 많은 이동궤적 근사 다면체들이 한 픽셀에 서로 겹쳐져있는 경우에 간단한 정렬 방법을 통해서는 완벽하게 정렬하기 쉽지 않기 때문에 본 연구는 비트연산과 깊이정렬 알고리즘을 이용한다. 이 알고리즘은 [16]과 [17]의 아이디어를 얻어 수정했다.

3.2.1 Interval Generation



[Fig. 2] An interval demonstration

본 논문에서는 주어진 픽셀에서 이동궤적 근사 다면체가 보이는 시간적인 간격을 인터벌(intervals)이라는 용어를 사용하여 나타낼 것이다. 한 인터벌은 한 이동궤적 근사 다면체의 두 개의 프래그먼트로부터 만들어진다. [Fig. 2]에서 확인할 수 있듯이, 각 인터벌에는 최소의 깊이 값과 최대의 깊이 값, 시간정보를 저장한다.



[Fig. 3] A triangle rotates from ABC, at $t=0$ to $A'B'C$, at $t=1$. For a given pixel P, instead of two fragments GPU generates four fragments.

만약 한 삼각형의 $t=0$에서의 모양 $\triangle ABC$와 $t=1$에서의 모양 $\triangle A'B'C'$이 서로 교차한다면,

[Fig. 3]처럼 한 픽셀에는 4개의 프래그먼트가 생성된다. 이런 경우엔 하나의 이동궤적 근사 다면체에서 두개의 인터벌을 만들어야한다. 우선 프래그먼트 4개를 카메라와의 거리에 따라 정렬하고, 가까운 프래그먼트들끼리 인터벌 하나를 만든다.

3.2.2 Sorting

인터벌을 깊이와 시간차원으로 이루어진 2차원 공간의 선분으로 표현한다. 각 인터벌의 직선 방정식은 시간과 깊이 값으로 계산할 수 있다. 깊이 정렬 알고리즘을 이용하여 각 인터벌들을 전후 순서로 정렬한다. 본 논문에서 사용한 깊이 정렬 알고리즘과 기본 방법과의 차이점은 아래와 같이 간략하게 요약할 수 있다.

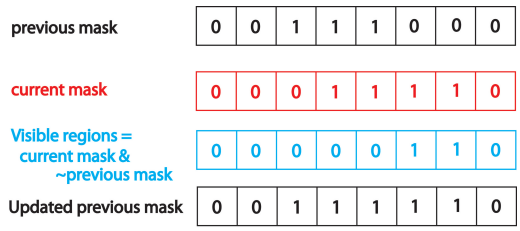
우선 모든 인터벌들의 최소 깊이 값을 가지고 오름차순으로 정렬하고, 만약 두 개의 인터벌이 교차하는 경우에, 이 교차점을 찾아서 두 인터벌을 4개의 짧은 인터벌로 나누고, 한 쌍의 인터벌 (P, Q)을 아래의 조건에 따라 검사 한다.

1.P와 Q 사이에 깊이 오버랩이 있는지
2.P와 Q 사이에 시간 오버랩이 있는지
3.뷰포인트 관점, Q가 완전히 P의 뒤에 있는지 (P의 직선의 방정식 사용)
4.뷰포인트 관점, P가 완전히 Q의 앞에 있는지 (Q의 직선의 방정식 사용)

만약 어느 조건 하나라도 $true$로 체크된다면 두 인터벌의 순서를 서로 바꾸고 그렇지 않은 경우에는 현재 순서를 유지한다.

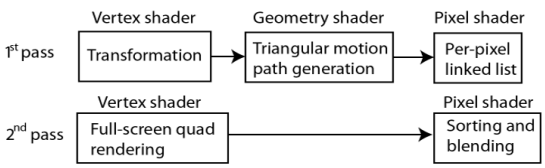
3.2.3 Blending

정렬이 끝난 뒤, 각 인터벌들의 기여도를 계산하기 위해서 전후 관계로 정렬된 인터벌의 리스트들을 탐색한다. 각 인터벌의 기여도에 따라 블랜딩의 양이 결정 된다. 한 인터벌의 기여도는 이전프레임 $(t=0)$에서부터 현재프레임 $(t=1)$까지 픽셀을 커버하는 기간이다. 본 논문에서는 시간에 따른 가시성을 비트단위로 표현하여 인터벌들의 기여도를 계산한다. 아래의 [Fig. 4]처럼 n비트 표현에서, 각 비트는 삼각형의 가시성을 나타내고, 비트가 0인 경우에는 보이지 않는 것을, 비트가 1인 경우에는 보이는 것을 의미한다. [Fig. 4]에서 “previous mask”는 이전 인터벌들의 시간차원 가시성(기여도)에 대한 비트 값들이다. 마찬가지로 “current mask”는 현재 인터벌에 대한 기여도이다. 두 개의 마스크(mask)는 4번째와 5번째 비트에서 (왼쪽에서부터) 오버랩이 있다. 따라서 현재 마스크의 기여도인 4번째 5번째 비트는 제외된다. 픽셀의 색상을 얻어내기 위해서 현재 인터벌의 기여도에 따른 색상의 평균을 계산해서 픽셀 색상에 추가한다. 그 다음 OR 연산자를 이용하여 전체적인 기여도를 업데이트한다. 전후 순서로 블랜딩 할 때, 이전 마스크의 모든 비트 값이 1이라면 탐색을 멈출 수 있다. 모든 인터벌을 블랜딩한 후, 이전 마스크에서 값이 ‘0‘인 비트의 개수에 따라 배경색을 픽셀 색상에 추가해준다.

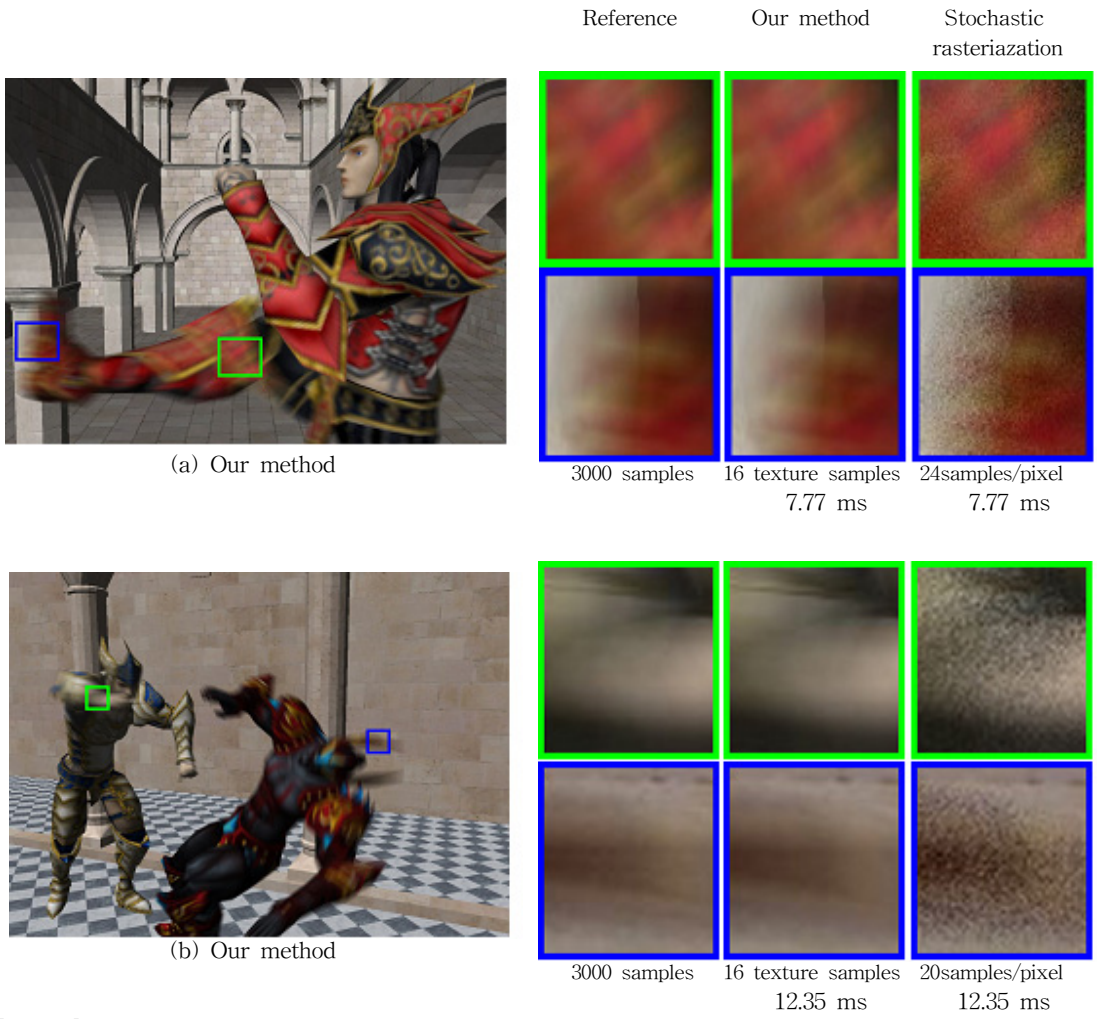


[Fig. 4] Temporal visibility is resolved by using bitwise operations. For the sake of simplicity, we only use 8-bit masks in this example.

4. 구 현



[Fig. 5] An overview of our algorithm.



[Fig. 6] Image quality comparisons between our algorithm and the stochastic rasterization at the similar render times. The reference images are rendered using the accumulation buffer with 3000 samples, our result images are rendered using 16 texture samples and the stochastic rasterization images are rendered using super-sampling 20 samples per pixel (a) and 24 samples per pixel (b).

[Table 1] Performance impact of each pass measured in milliseconds." $1^{st}pass$" and "$2^{nd}pass$"

	Fig. 6 (a) (69.4k triangles)	Fig. 6 (b) (79.9k triangles)	Fig. 7 (12 triangles)
$1^{st}pass$	0.77ms	1.09ms	0.1ms
$2^{nd}pass$	4.66ms	8.09ms	0.53ms
Frame time	7.77ms	12.35ms	1.5ms

본 논문에서 제안한 알고리즘은 [Fig. 5]에서 볼 수 있듯이 $1^{st}pass$ 과정, $2^{nd}pass$ 과정 총 2개의 단계로 구성돼있다. 오버랩 영역에 포함되어있는 모든 프래그먼트들은 $1^{st}pass$ 단계에서 스텐실 테스트를 거쳐 이후 단계에서는 제외될 것이다.

$1^{st}pass$에서 모든 정점들은 버텍스 쉐이더(vertex shader)를 거쳐 물체 공간(object space)에서 뷰 공간(view space)으로 이동되고, 지오메트리 쉐이더(geometry shader)에서는 이전프레임과 현재프레임의 6개 정점을 이용하여 이동궤적 근사 다면체를 만든다. 픽셀 쉐이더(pixel shader)에서 현재 프래그먼트의 정보를 이용하여 버퍼에 저장한다. 이 정보는 픽셀당 링크드-리스트(linked-list) 로 [25]와 [26,27]과 유사한 방법이다. 각 프래그먼트의 구조가 다르다는 차이가 있다. 한 프래그먼트에는 깊이 값, 텍스쳐 좌표, 시간, 법선벡터, 텍스쳐 ID, 넥스트 포인터(next pointer)등 여러 정보를 가지고 있다. $2^{nd}pass$에서는 [25]와 동일한 OIT(Order Independent Transparency) 알고리즘을 사용하여 해당 픽셀에 속하는 모든 프래그먼트들을 로드(load)하고, 모든 인터벌들을 정렬하고 블랜딩 한다.

5. 실험 및 결과

본 실험에서는 DirectX 11과 HLSL5.0을 이용했고, Geforce GTX980 Ti 6GB RAM, Core i5 CPU 3.4GHz과 16GB RAM을 탑재한 컴퓨터에서 수행했다. 모든 결과의 이미지는 퐁-쉐이딩(Phong Shading)을 이용했고 1024 x 768 해상도이며, 16-bit floating-point RGBA color format, 128-bit coverage mask, 픽셀당 70개의 인터벌, 16개의 텍스쳐 샘플로 구성돼있다. 프래그먼트 버퍼 크기를 픽셀 당 최대 150개의 프래그먼트를 저장할 수 있도록 변경했다.

본 논문의 실험결과는 이전 방법 “Stochastic rasterization” 및 “Reference”와 비교하여 이미지 품질을 평가한다. 본 실험의 모델은 [28]에서 제공된 모델을 사용했고, “Reference”는 [9]의 누적 버퍼 알고리즘을 이용해 3000개의 샘플을 렌더링한 결과이다. “Stochastic rasterization”은 [13]에 기술된 알고리즘과 super-sampling으로 렌더링한 결과이다. [Fig. 6]에서 볼 수 있듯이 본 논문의 결과 이미지의 화질과 속도를 비교한다. 본 실험은 움직이는 물체가 있는 장면을 렌더링 했다. 옅은 색 테두리 이미지는 결과의 일부분을 확대한 것이다. 이 이미지를 통해서 본 논문의 알고리즘이 “Stochastic rasterization”방법 보다 더 매끄럽게 블러된 것을 확인할 수 있다. 짙은 색 테두리 이미지는 배경과 움직이는 캐릭터사이를 확대한 결과이다. 즉, 서로 다른 물체사이에 있는 삼각형들의 블러 현상을 확대한 것이다. 마찬가지로 이런 부분도 “Stochastic rasterization”방법 보다 더 좋은 결과를 얻어냈다. 결과적으로 본 논문에서 제안한 알고리즘을 이용한 실험의 결과 이미지는 “Reference” 결과와 굉장히 유사한 반면에, “Stochastic rasterization”의 결과는 노이즈가 있고 아티팩트들이 발생하는 것을 확인할 수 있다. 또한 본 논문에서 제공한 알고리즘과 “Stochastic rasterization” 방법의 렌더링 속도는 같지만 이미지 화질에서 많은 차이가 있는 것을 확인할 수 있다.

[Fig. 7] A cube rotates around xyz axes. Our approximation with linear motion gives a plausible result with moderate rotation

[Fig. 7]에서는 물체가 회전하는 경우 본 논문에서 제공하는 방법의 화질을 보여준다. 회전하는 움직임도 블러 효과가 매끄럽게 표현되는 것을 확인할 수 있다.

[Table 1]은 본 논문에서 제안하는 방법을 이용한 결과 이미지의 렌더링 속도를 보여준다.

6. 결 론

본 논문에서는 실시간 렌더링에서 이동궤적 근사 다면체를 이용한 새로운 모션블러 알고리즘을 제안했다. 이 알고리즘의 메인 아이디어는 각 정점마다 시간 정보와 텍스쳐 좌표를 가지고 있는 이동궤적 근사 다면체를 이용한 것이다. 하나의 이동궤적 근사 다면체는 이전프레임에서 현재프레임으로 움직이는 여러 픽셀들을 담고 있다. 기본적인 레스터화를 거치면 2개 또는 4개의 프래그먼트가 생성되고 이 프래그먼트의 속성은 GPU를 통해서 보간 된다. 이 속성을 이용하면 어떤 픽셀에서 한 삼각형이 어느 순간에 보였는지 알 수 있다. 이 프래그먼트들은 픽셀 당 연결 리스트에 저장되고 이후에 픽셀 당 인터벌 리스트 형식으로 로드하여 사용된다. 이러한 데이터로 시공간적차원의 깊이 정렬과 비트연산을 이용하여 가시성 문제를 해결하여 매끄러운 모션블러 이펙트를 얻어냈다.

REFERENCE

[1] K. Sung, A. Pearce, C. Wang, "Spatial-temporal antialiasing", IEEE Transactions on Visualization and Computer Graphics, Vol. 8, No. 2, pp. 144 - 153, 2002.

[2] G. Rosado, "Motion Blur as a Post Processing Effect", Addison-Wesley Publishing Company, chap. 27, 2007.

[3] T. Sousa, "Crysis next gen effects", Game Developer Conference, 2008.

[4] T. Sousa, "Cryengine 3 rendering techniques:, Talk at Microsoft Game Technology conference Gamefest, 2011.

[5] M. McGuire, P. Hennessy, M. Bukowski, B. Osman, "A reconstruction filter for plausible motion blur", Interactive 3D Graphics and Games, pp. 135 - 142, 2012.

[6] T. Sousa, "Cryengine 3 rendering techniques", ACM SIGGRAPH Course Notes, 2013.

[7] J. P. Guertin, M. McGuire, D. Nowrouzezahrai, "A fast and stable feature-aware motion blur filter", High Performance Graphics, pp. 10, 2014.

[8] J. P. Guertin, D. Nowrouzezahrai, "High performance non-linear motion blur", Eurographics Symposium on Rendering-Experimental Ideas & Implementations, 2015.

[9] P. Haeberli, K. Akeley, "The accumulation buffer: hardware support for high-quality rendering", ACM SIGGRAPH Computer Graphics'90, vol. 24, pp. 309 - 318, 1990.

[10] J. Korein, N. Badler, "Temporal anti-aliasing in computer generated animation", ACM SIGGRAPH Computer Graphics'83, Vol. 17, pp. 377 - 388, 1983.

[11] R. L. Cook, T. Porter, L. Carpenter, "Distributed ray tracing", ACM SIGGRAPH Computer Graphics'84, Vol. 18, pp. 137 - 145, 1984.

[12] T. Akenine-Möller, J. Munkberg, J. Hasselgren, "Stochastic rasterization using time-continuous triangles", Graphics Hardware, pp. 7 - 16, 2007.

[13] M. McGuire, E. Enderton, P. Shirley, D. Luebke, "Real-time stochastic rasterization on conventional gpu architectures", High Performance Graphics, pp. 173 - 182, 2010.

[14] J. Munkberg, P. Clarberg, J. Hasselgren, R. Toth, M. Sugihara, T. Akenine-Möller, "Hierarchical stochastic motion blur rasterization", High Performance Graphics, pp. 107 - 118, 2011.

[15] S. Laine, T. Aila, T. Karras, J. Lehtinen, "Clipless dualspace bounds for faster stochastic rasterization". ACM Transactions on Graphics, Vol. 30, No. 4, pp 106:1 - 106:06 , 2011.

[16] M. E. Newell, R. G. Newell, T. L. Sancha,

"A solution to the hidden surface problem", Proceedings of the ACM Annual Conference, Vol. 1, pp. 443 - 450, 1972.
[17] C. W. Grant, "Integrated analytic spatial and temporal anti-aliasing for polyhedra in 4-space", ACM SIGGRAPH Computer Graphics'85, Vol. 19, pp. 79 - 84, 1985.
[18] C. J. Gribel, M. Doggett, T. Akenine-Möller, "Analytical motion blur rasterization with compression", High Performance Graphics, pp. 163 - 172, 2010.
[19] J. Ragan-Kelley, J. Lehtinen, J. Chen, M. Doggett, F. Durand, "Decoupled sampling for graphics pipelines", ACM Transactions on Graphics, Vol. 30, No. 3, pp. 17:1 - 17:17 , 2011.
[20] P. Clarberg, R. Toth, J. Hasselgren, J. Nilsson, T. Akenine-Möller, "AMFS: adaptive multi-frequency shading for future graphics processors", ACM Transactions on Graphics, Vol.33, No.4, pp. 141:1 - 141:12, 2014.
[21] R. T. Petrik Clarberg, J. Munkberg, "A sort-based deferred shading architecture for decoupled sampling", ACM Transactions on Graphics, Vol. 23, No. 4, pp. 141:1 - 141:10, 2013.
[22] M. Andersson, J. Hasselgren, R. Toth,T Akenine-Möller, "Adaptive texture space shading for stochastic render-ing", Computer Graphics Forum, Vol.33, No,2, pp. 10, 2014.
[23] P. Clarberg, J. Munkberg, "Deep shading buffers on commodity gpus", ACM Transactions on Graphics, Vol. 33, No. 6, pp. 227:1 - 227:12 , 2014.
[24] Johannes Schmid, Robert W. Sumner, H.B., M. Gross, "Programmable motion effects", ACM Transactions on Graphics, Vol. 29, No. 4, pp. 227:1 - 227:12, 2010.
[25] P. Barta, B. Kovacs, S. L. Szecsi, L. Szirmay-kalos, "Order independent transparency with per-pixel linked lists", Proceedings of CESCG 2011, 2011.
[26] M. Salvi, J. Montgomery, A. Lefohn, "Adaptive transparency", High Performance Graphics, pp. 119 - 126, 2011.
[27] C. A. Burns, "The visibility buffer: A cache-friendly approach to deferred shading", Journal of Computer Graphics Techniques (JCGT), Vol. 2, No. 2, pp. 55 - 69, 2013.
[28] M. McGuire, Computer graphics archive, http://graphics.cs.williams.edu/data, 2011.

홍 민 푸(Hong, Minh-phuoc)

2009-현재 숭실대학교 미디어학부 석박사통합과정
관심분야 : 실시간 렌더링, 모션 블러, 전역 조명

최 진 형(Choi, Jin hyung)

2016-현재 숭실대학교 미디어학부 석사과정
관심분야 : 실시간 렌더링, 모션블러, 그림자, 전역 조명

오 경 수 (Oh, Kyoung su)

2001 서울대학교 전기 컴퓨터 공학부 박사
2001-2002 ㈜ 조이멘트 개발팀장
2003-현재 숭실대학교 미디어학부 부교수
관심분야 : 실시간 컴퓨터 그래픽스, 그림자, 전역 조명, 모션 블러, 3D 모델링