특집

GPGPU 기반의 실시간 충돌감지

최유주 (서울벤처정보대학원대학교)

I. 서 론

현실 세계에서 존재하는 실제 물체의 형태 및 변형 특성 등을 모델링하여 컴퓨터상의 가상 객체로 표현하는 기술은 시술 시뮬레이션 및 가상현실 기반 컴퓨터 게임, 컴퓨터 애니메이션 분야 등에서 요구되는 중요 기술이다. 특히 사용자 조작에 의한 실시간 객체 형태 변형 기술은 대화식 가상현실 응용 분야의 필수 기반 기술로 꼽히고 있다. 변형 객체에 대한 시뮬레이션에 있어서 수행 효율성에 크게 영향을 주는 주요 요소로서 변형객체에 대한 충돌감지를 들 수 있다. 특히, 동일 변형객체 내부에서 발생할 수 있는 자체 충돌감지는 주요한 계산 병목현상의 주원인으로 지적되고 있다. 이는 서로 다른 비강체간의 상호작용 또는 비강체와 강체간의 상호작용에서 비강체는 독립된 다른 객체와의 충돌을 통하여 지속적으로 형태 변형이 이루어지므로 독립된 서로 다른 객체간 충돌 뿐 만 아니라 동일 객체내의 자체충돌이 빈번하게 발생되기 때문이다. 비강체의 경우 객체를 구성하는 모든 기본 요소들 간의 충돌이 가능하므로 자체 충돌감지를 위해서는 모든 기본 요소들의 쌍에 대한 충돌감지 처리가 요구된다.

다각형 메쉬로 표현된 강체간 충돌감지에서는 각 객체에 대한 바운딩 볼륨 계층구조(BVH: Bounding Volume Hierarchies)를 이용한 충돌감지 기법이 효과적이다. 즉, 강체의 경우, 초기의 형태가 고정적으로 유지되기 때문에 객체의 형태 특성을 표현하는 바운딩 볼륨의 계층적 자료구조를 전처리 단계에서 구축하고, 온라인 시뮬레이션 단계에서는 미리 구축된 계층적 자료구조를 이용하여 객체간 충돌을 효과적으로 감지할 수 있다. 반면, 변형객체는 초기의 형태를 온라인 시뮬레이션 단계에서 유지하지 않으므로 전처리 단계에서 객체의 형태 특성을 고려한 고정적 바운딩 볼륨 계층적 구조는 충돌감지에 그대로 효과적으로 활용되기 어렵고, 시뮬레이션 중에 지속적인 수정 작업이 병행되어야 한다. 이 또한, 높은 계산 비용을 요구하는 실시간 처리를 저해하는 요소이다. 그러므로, 비강체 변형모델의 충돌감지 기법에서는 모델의 형태를 기반으로 한 비싼 전처리 작업이나 수행 중 수정작업이 요구되는 복잡한 계층적 자료구조의 운용이 배제되어야 한다.

강체 모델간의 충돌감지에 있어서도 대상 모

델이 대규모 크기의 모델인 경우, 충돌감지에 요구되는 수행비용이 높아지게 되어 실시간 시뮬레이션 처리가 어렵게 된다. 그러므로, 대형 모델간의 충돌감지를 효율적으로 처리하기 위하여 충돌 가능성이 전혀 없는 모델들을 상세한 충돌계산의 대상에서 사전에 제외시킬 수 있는 효과적인 컬링 방법들이 고안되고 있다.

본고에서는 가상 객체의 실시간 시뮬레이션시 병목현상의 주원인으로 지적되고 있는 충돌감지를 효율적으로 처리하기 위하여 처리 효율성이 뛰어난 그래픽스 하드웨어(GPU)를 이용하는 기법들에 대하여 소개하고자 한다.

우선, 일반적인 문제를 해결하기 위한 목적으로 GPU를 활용하는 범용 계산 기법의 장점을 II장에서 설명하고, III장에서는 GPU를 이용한 충돌감지 기술들을 두 가지 부류로 분류하여 특성을 설명하고자 한다. IV장과 V장에서는 충돌감지의 효율성을 높이기 위하여 충돌 가능성이 적은 객체 및 기본 요소들을 배제시키는 GPU 기반 컬링 기법과 변형객체의 충돌감지를 위하여 텍스처를 활용한 기법에 대하여 설명한 후, 마지막으로 VI장에서 결론을 맺도록 하겠다.

II. GPGPU의 정의와 특성

최근 그래픽 렌더링 문제가 아니라 비그래픽적 문제를 해결하기 위하여 GPU를 사용하는 연구들이 활발하게 발표되고 있다[1,2]. 이와 같이 컴퓨터 그래픽스를 위한 계산만을 다루던 GPU를 사용하여 CPU에서 전통적으로 관리했던 응용 프로그램들의 계산을 수행하는 기술을 그래픽 처리 장치를 통한 일반 목적의 컴퓨팅이라 하여 GPGPU (General-Purpose computing on Graphics Processing Units)라고 부르고 있다. 범용 목적으로 GPU를 사용하는 주요 이유는 다음과 같이 요약된다.

• 수행효율성(performance) 측면

일반적으로 GPU는 대용량의 SIMD(Single Instruction Multiple Data) 병렬구조로 구성이 되어 있어서 대용량의 데이터를 CPU에 비하여 보다 높은 처리속도로 처리할 수 있다. 또한, 다수의 GPU를 동시에 사용함으로써, 수행 효율성을 선형적으로 증가시킬 수 있다.

• 작업 부하의 균형(load balancing) 측면

GPU와 CPU간의 작업부하의 균형을 유지함으로써, CPU-집약적인 응용 프로그램의 전체적인 수행 효율성을 증가시킬 수 있다.

• 발전속도(performance growth) 측면

GPU의 성능 발전 속도는 CPU의 성능 발전속도를 앞지르고 있다. Pentium 4 프로세서의 경우 평균적으로 1년에 0.65 GFLOPS의 성장속도를 보여주고 있는 반면, NVIDIA 사의 GPU의 경우, 평균적으로 1년에 24 GFLOPS의 성장속도를 보여주고 있다. CPU 대비 GPU의 빠른 성능 발전 경향은 지난 수 년간 지속되어 왔고, 이러한 경향은 앞으로도 지속될 것으로 분석되고 있다.

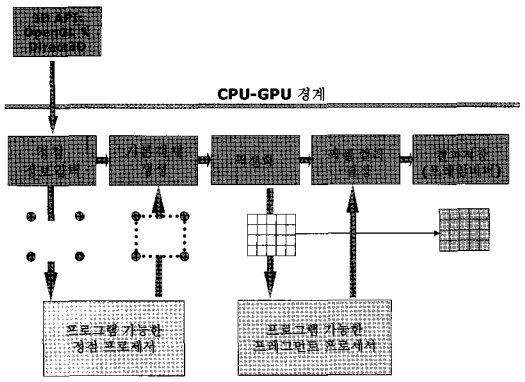
GPU는 직렬 프로세서(serial processor)가 아닌 스트림 프로세서(stream processor)이다. 직렬 프로세서는 순차적으로 하나의 명령을 수행하고, 메모리를 수정하는 반면, 스트림 프로세서는 입력 데이터 집합에 대하여 하나의 명령을 병렬로 처리하고, 출력 데이터의 집합을 동시에

발생시킨다. 각 스트림 프로세서로 전달되는 기본 데이터 요소들은 다른 데이터 요소와의 종속성이 없이 독립적으로 처리된다.

GPU에 의하여 지원되는 프로그램 가능한 그래픽 파이프라인은 <그림 1>과 같다. 일반적인 그래픽스 파이프라인에서 하나의 도형을 화면에 그리는 작업절차를 살펴보면, 우선, GPU에게 정점(vertex)의 정보가 전달되고, 정점의 정보를 이용하여 도형이 생성된다. 다음 단계에서 생성된 도형은 2차원 평면으로 투영되어 프레그먼트(fragment), 즉, 픽셀의 집합으로 표현된다. 마지막으로 래스터 명령을 통하여 각 픽셀의 색상이 결정되고, 색상이 추가된 최종 렌더링 결과는 프레임 버퍼에 저장된다. 프로그램이 가능한 그래픽스 파이프라인에서는 일련의 기존 그래픽스 파이프라인에 두 개의 프로세서 구성요소, 즉 정점 프로세서(vertex processor) 및 프레그먼트 프로세서(fragment processor)가 추가되었다. 그리고, 쉐이더(shader)라고 불리는 사용자 프로그램이 추가된 두 프로세서 상에서 수행된다. 따라서, 사용자에 의해 작성된 쉐이더의 수행결과에 따라 렌더링 결과를 사용자의 의도에 따라 바꿀 수 있다. 정점 프로세서 상에서 수행되는 정점 쉐이더는 렌더링 결과를 3차원적 정점 레벨에서 수정하고, 프레그먼트 프로세서 상에서 수행되는 픽셀 쉐이더는 픽셀레벨에서 각 픽셀의 색상을 자유롭게 수정할 수 있다.

범용 목적으로 GPU를 사용하는 응용 어플리케이션들의 경우 목적에 따라 픽셀의 색상값을 자유롭게 수정하는 픽셀 쉐이더들을 포함한다. 픽셀 단위의 픽셀 쉐이더는 실수 텍스처를 이용하여 일반적인 목적의 연산을 수행하고 그 수행결과를 프레임 버퍼에 저장하게 된다.

GPU를 이용한 범용 컴퓨팅의 처리 절차는 다



〈그림 1〉 프로그램 가능한 그래픽 파이프라인

음과 같다.

• 텍스처를 이용한 입력 데이터 저장

GPU 내에서 처리를 원하는 임의의 데이터 타입의 데이타는 실수 텍스처(floating point textures)에 저장하여 GPU로 전달 할 수 있도록 한다.

• 텍스처를 이용한 사각형 렌더링

GPU 내에서 원하는 사용자 프로그램을 수행하기 위해서는 원하는 데이터 요소의 크기에 일치하는 윈도우 상에 윈도우에 맞는 직사각형을 그리고, 그 위에 준비된 데이터 텍스처를 렌더링한다.

• 범용 목적의 픽셀 쉐이더의 수행

실수 텍스처의 형태로 입력된 입력 데이터는 텍스처 렌더링 과정을 통하여 픽셀 쉐이더의 입력으로 전달된다. 이를 이용하여 범용 목적의 쉐이더가 수행되고 수행 결과값을 생성한다.

• 수행 결과의 저장과 분석

쉐이더의 수행 결과값은 GPU 내의 프레임 버

퍼나 오프 스크린 버퍼에 픽셀 단위로 저장된다. 그래픽스 명령을 이용하여 GPU 내에 저장된 결과값을 CPU로 읽어 들이고 이에 대한 분석처리를 수행한다. 예를 들어, Open에서 한 버퍼의 사각영역을 읽어 들이기 위하여, glReadBuffer()를 이용하여 읽고자 하는 버퍼를 지정하고, glReadPixels()를 이용하여 지정된 버퍼의 원하는 영역을 읽는다. 대부분의 시스템에서 GPU의 버퍼로부터 데이터를 읽어 오는 작업은 수행 속도를 크게 저하시키고 있기 때문에 readback 하는 영역을 최소화 하는 정책들을 적용하고 있다.

III. GPU 기반 충돌감지 기법의 분류

GPU를 활용한 충돌감지 연구들을 분석하여 보면, 크게 두 가지의 방법으로 GPU를 활용하고 있는 것을 알 수 있다. 첫 번째는 이미지 공간을 기반으로 빠른 교차(intersection) 테스트를 수행하기 위하여 GPU를 활용하는 것이다. 두 번째는 기하학적 계산을 빠르게 하기 위하여 GPU를 보조 처리기(co-processor)로 활용하는 것이다.

1. 이미지 공간 기반 교차 테스트

이미지 공간 기반 교차 테스트 기술은 충돌감지 대상 객체를 컬러(color), 깊이(depth), 스텐실(stencil) 버퍼로 래스터라이징(rasterizing)한 결과를 기반으로 하는 것으로서, 2차원 이미지 공간상에서 대상 객체간의 교차가 있는지 여부를 판별하는 방법이다. 이미지 공간 기반 기술들은 래스터라이즈가 가능한 임의의 그래픽 기본요소 (Bezier, NURBS 패치, 임의의 폴리곤 패치 등)에 대해서 적용가능한 방법으로 충돌감지를 위한 복잡한 자료 구조를 필요로 하지 않는다. 그러나, 이미지 공간을 기반으로 하는 방법들은 정확한 충돌감지 결과를 얻기보다 근사값(approximation)을 알려주는 방법으로서, 실제적으로 충돌이 없는 두 객체가 이미지 공간상에서는 교차된 상황으로 인식될 수 있다. 또한, 카메라의 시선방향과 수직인 위치의 폴리곤은 이미지 공간에서 래스터라이즈가 되지 않기 때문에 교차테스트를 수행할 수 없는 문제점이 있다.

2. 3차원 공간 기반 충돌감지

3차원 공간상의 대상 객체의 기하학적 정보를 텍스처에 적절하게 저장하고 이를 렌더링 함으로써, 프로그램 가능한 픽셀 쉐이더를 통하여 GPU 상에서 충돌감지 테스트를 빠르게 처리한다. 예를 들어 Carr[3]와 Purcell[4]은 ray-triangle 교차 테스트를 처리하는 픽셀 쉐이더를 구현하여 GPU 상에서 레이 트레이싱(ray tracing)을 위한 복잡한 계산을 효율적으로 수행하였다.

IV. 가시성 테스트 기반 컬링

픽셀 쉐이더에서 객체간 충돌감지 및 자체충돌 감지를 수행하기 이전에 최근 확장된 그래픽스 라이브러리에서 지원하는 폐쇄검사(occlusion query)를 이용하여 충돌이 발생할 수 없는 삼각형 구성 요소들을 제거하고, 충돌 발생이 가능한 삼각형 집합만을 선별한다[5].

폐쇄검사의 처리 결과가 거짓(false)인 경우는 현재의 시점을 중심으로 보았을 때, 테스트 대상 객체가 다른 객체에 의하여 가려지게 됨을 나타낸다. 반대로, 폐쇄검사의 처리 결과가 참(true)

인 경우는 테스트 대상 객체가 어떠한 다른 객체에 의해서 가려짐이 없이 완전하게 가시화됨을 나타낸다. 이러한 폐쇄검사는 기본적으로 객체를 표현하기 위해 구분된 각 구성 픽셀의 깊이값(depth value)를 검사함으로써 체크된다.

1. 객체간 충돌 가능 삼각형 집합 선별

n개의 객체 O1,O2,…,On이 주어지고, 각 객체 Oi는 mi개의 삼각형 T_1^i, T_2^i, ..., $T_{m_i}^i$로 구성될 때, Oi의 하나의 구성 삼각형 T_k^i가 [조건 1]을 만족하는 경우 T_k^i는 충돌가능 삼각형 집합에서 제외된다.

[조건 1] O1,…,Oi−1,Oi+1, Oi+2,…,On ($1 \le i \le n$)의 구성 삼각형들과 T_k^i가 서로 교차하지 않는다.

[조건 1]의 만족여부를 판별하기 위하여 가시성 테스트 모드에서, 2 단계 렌더링을 수행한다. 즉, 첫번째 단계에서 O1,…, Oi−1, Oi, Oi+1, …, On 의 순서로 각 구성 삼각형들을 렌더링하고, 각 삼각형이 다른 객체의 구성 삼각형에 의하여 가려지는지 아니면 전체 삼각형이 온전하게 가시화 되는지를 폐쇄검사 함수를 이용하여 검사한다. 깊이 버퍼의 내용을 지우고 두번째 단계의 렌더링을 수행한다. 두 번째 단계에서는 1 단계의 렌더링 순서의 역순서 즉, On,…, Oi+1, Oi, Oi−1, …, O1의 순으로 각 구성 삼각형들을 렌더링하고, 각 삼각형이 다른 객체의 구성 삼각형에 의하여 가려지는지 온전히 가시화되는 지를 검사한다. 한 구성 삼각형이 양방향으로 다른 삼각형에 의한 가림 현상 없이 온전히 가시화 되는 경우, 이 삼각형은 어느 구성 삼각형과도 교차하지 않는 것으로 판단되어 충돌 가능성이 없는 삼각형으로 선별한다. 즉, 충돌가능 삼각형 집합에서 제외시킨다. 2단계 렌더링에 의한 객체간 충돌가능 삼각형 집합 선별 알고리즘은 아래와 같고, 알고리즘 적용 후 결과영상은 <그림 2>와 같다.

ALGORITHM : 객체간 충돌가능 삼각형 집합 선별 알고리즘

- First pass :

① Clear the depth buffer (use orthographic projection)

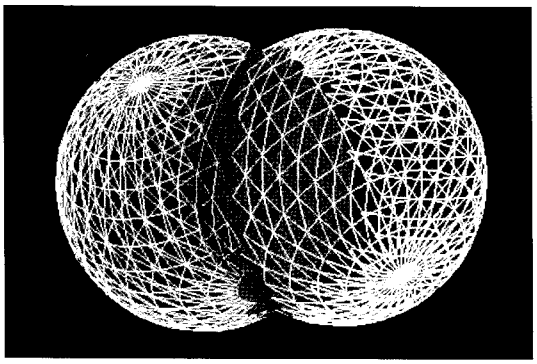
② For each object Oi,i=1,…,n
  - Disable depth mask and set the depth function to GL_GEQUAL.
  - For each sub-object T_k^i in Oi
    Render T_k^i using an occlusion query
  - Enable the depth mask and set the depth function to GL_LEQUAL
  - For each sub-object T_k^i in Oi
    Render T_k^i

③ For each object Oi,i=1,…,n
  - For each sub-object T_k^i in Oi
    Test if T_k^i is not visible with respect to the depth buffer. If it is visible, set a tag to note it as fully visible.

④ Clear the depth buffer (use orthographic projection)

- Second pass:

Same as First pass, except that the two "For each object" loops are run with i=n, …,1.



〈그림 2〉 객체간 충돌가능 삼각형 집합
(짙은색으로 표시된 삼각형)

2. 객체내 충돌 가능 삼각형 집합 선별

하나의 객체가 n개의 삼각형 P1,P2,…,Pn로 구성될 때, 각 구성 삼각형 Pi는 [조건 2]를 만족하는 경우 자체충돌 가능 삼각형 집합에서 제외된다. 자체충돌 가능 삼각형 집합은 서로 다른 두 개 이상의 삼각형이 서로 관통하는 삼각형들만을 포함한다.

[조건 2] 구성 삼각형 P1,…,Pi−1,Pi+1,…,Pn $(1 \le i \le n)$와 Pi가 서로 관통하지 않는다.

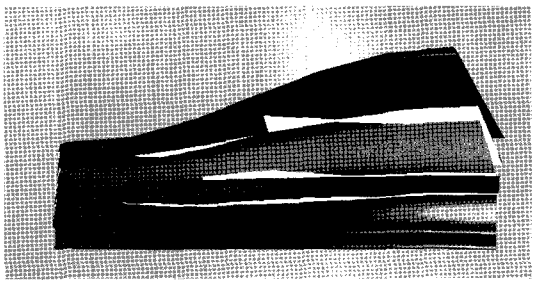
[조건 2]의 만족여부를 판별하기 위하여 가시성 테스트 모드에서, 2 단계 렌더링을 수행한다. 즉, 첫번째 단계에서 P1,…,Pi−1,Pi,Pi+1, Pi+2, …,Pn 의 순서로 각 구성 삼각형들을 렌더링하고, 각 삼각형이 다른 구성 삼각형에 의하여 가려지는지 아니면 전체 삼각형이 온전하게 가시화되는지를 폐쇄검사 함수를 이용하여 검사한다. 깊이 버퍼의 내용을 지우고 두번째 단계의 렌더링을 수행한다. 두 번째 단계에서는 1 단계의 렌더링 순서의 역순서 즉, Pn,…,Pi+1,Pi,Pi−1, …,P1 의 순으로 각 구성 삼각형들을 렌더링하고, 각 삼각형이 다른 구성 삼각형에 의하여 가려지는지 온전히 가시화되는지를 검사한다. 한 구성 삼각형이 양방향으로 다른 삼각형에 의한 가림 현상 없이 온전히 가시화 되는 경우, 이 삼각형은 어느 구성 삼각형과도 교차하지 않는 것으로 판단되어 자체충돌 가능성이 없는 삼각형으로 선별한다. 즉, 자체충돌가능 삼각형 집합에서 제외시킨다. 객체간 충돌 알고리즘과 다른 점은 폐쇄검사를 수행하기 위하여 현재 프레임 버퍼의 깊이 정보가 깊이 버퍼의 대응 값과 비교하여 더 큰 경우, 즉, 같은 에지나 면에 접하지 않으면서 깊이값이 큰 경우만 다른 삼각형에 의하여 가려지는 경우로 판단하도록 한다. 그리고 각 삼각형 단위로 깊이 테스트를 수행하고 매번 깊이 정보를 수정할 수 있도록 한다. 2단계 렌더링에 의한 객체내 충돌 가능 삼각형 집합 선별 알고리즘은 아래와 같고, 알고리즘 적용 후 결과 영상은 <그림 3>와 같다.

ALGORITHM : 자체충돌 가능 삼각형 집합 선별 알고리즘

- First pass :

① Clear the depth buffer (use orthographic projection)

② For each triangle Pi,i=1,…,n
  - Disable depth mask and set the depth function to GL_GREATER.
  - Render T_k^i using an occlusion query.
  - Enable the depth mask and set the depth function to GL_LEQUAL.
  - Render T_k^i



〈그림 3〉 자체충돌 가능 삼각형 집합
(자체충돌 가능 삼각형 흰색 표시)

③ For each triangle Pi,i=1,…,n
- Test if Pi is not visible with respect to the depth buffer. If it is visible, set a tag to note it as fully visible.

④ Clear the depth buffer (use orthographic projection)

• Second pass:

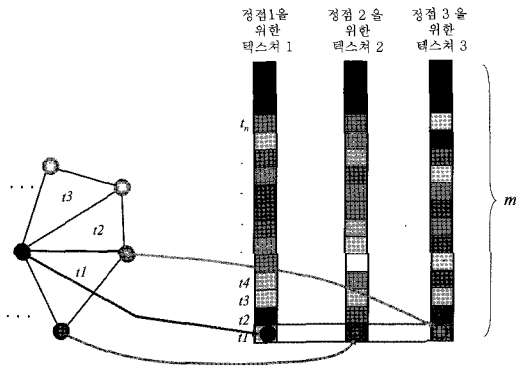
Same as First pass, except that the two "For each triangle" loops are run with i= n,…,1.

V. 삼각 메쉬쌍의 충돌감지

본 절에서는 가시성 테스트를 기반으로 충돌 가능성이 있는 것으로 판별된 삼각형들을 대상으로 가능한 모든 삼각형 쌍을 정의하고, 이를 텍스처를 이용하여 표현하는 방법과 GPU를 이용하여 삼각형 쌍의 충돌여부를 감지하는 방법에 관하여 설명하고자 한다.

1. 삼각 메쉬의 텍스처 저장

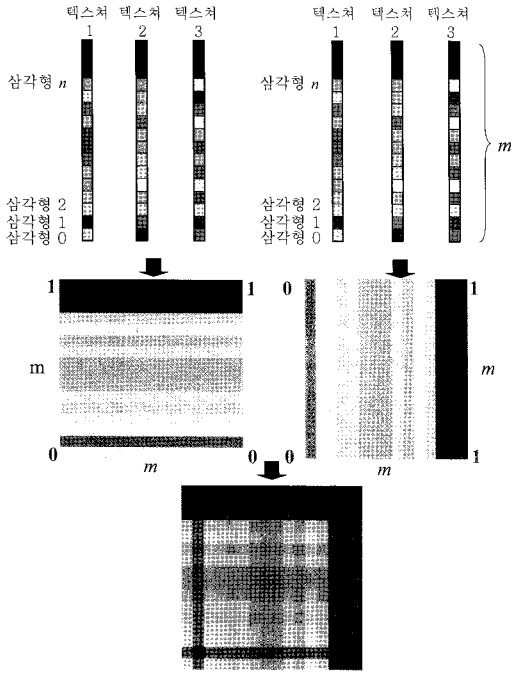
삼각형 메쉬쌍들의 충돌감지 기법에서는 충돌



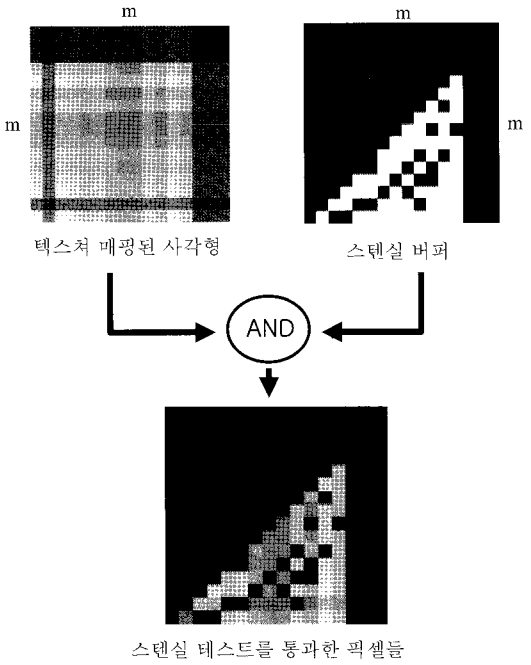
〈그림 4〉 텍스처를 이용한 삼각 메쉬의 저장

가능성이 있는 각 삼각형의 좌표를 3개의 1차원 텍스처에 저장한다. <그림 4>에서와 같이 각 삼각형의 한 정점의 좌표는 1차원 텍스처를 구성하는 하나의 텍셀(texel)에 저장된다. 즉, 정점의 x, y, z의 좌표값은 각기 텍셀의 r, g, b 필드에 저장된다. 예를 들어, <그림 4>의 왼쪽에 표시된 메쉬에서 객체를 구성하는 첫 번째 삼각형 t1의 3개의 정점 좌표는 3개의 1차원 텍스처의 첫 번째 텍셀에 저장하고, 두 번째 삼각형 요소인 삼각형 t2을 구성하는 3개의 정점의 좌표는 각기 1차원 텍스처의 두 번째 텍셀에 저장한다. 같은 방법으로 객체들을 구성하는 모든 삼각형의 좌표는 3개의 1차원 텍스처들 상에 저장이 된다. 충돌 가능성이 있는 삼각형의 총 개수가 n이라 하였을 때, n을 포함하는 가장 작은 2의 자승값 m이 텍스처의 크기가 된다.

삼각형 메쉬를 포함한 1차원 텍스쳐의 크기가 m이라 하였을 때 m×m크기의 사각형을 그리고, 그 위에 1차원 텍스쳐들을 수평과 수직방향으로 반복적으로 매핑 시킨다. 이 때, 매핑 된 텍스쳐의 픽셀값들 간에 서로 보간현상이 발생되지 않도록, m×m크기 이상의 뷰잉 윈도우를 생성하고, 원근투영 기법이 아닌 평행투영 기법으로 뷰



〈그림 5〉 1차원 텍스처를 이용한 모든 삼각쌍의 표현



〈그림 6〉 스텐실 테스트를 이용한 컬링

잉 윈도우를 설정한다. <그림 5>는 1차원 텍스쳐의 매핑 절차와 매핑 결과 영상을 보여주고 있다. 1차원 텍스쳐의 수직, 수평 방향으로의 매핑 결과 사각형을 표현하는 각 픽셀은 한 쌍의 삼각형을 표현하게 된다. 그러므로, 텍스쳐가 매핑 된 사각형은 객체를 표현하는 삼각형들에 있어서 모든 가능한 삼각형 쌍을 나타낸다.

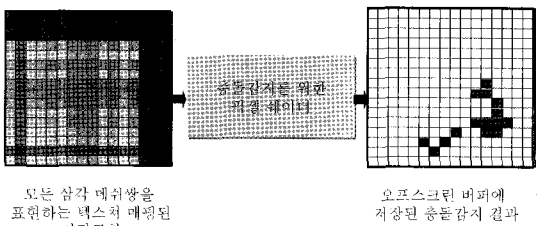
2. 위상학적 정보기반 컬링

사각형 상에 가로와 세로 방향으로 매핑된 1차원 텍스쳐에 의하여 각 픽셀은 한 쌍의 삼각형들을 표현하게 되는데, 사각형의 대각선을 중심으로 상위 좌측 영역과 하위 우측 영역이 서로 대칭적인 삼각형쌍을 표현하게 된다. 그러므로, 중복되는 삼각형 쌍에 대해서는 충돌감지 픽셀 쉐이더가 수행되지 않도록 하는 처리가 필요하다. 따라서, 하드웨어 지원 스텐실 테스트를 이용하여 중복되는 삼각형 쌍을 테스트 대상에서 제외시키기 위하여 선별적으로 픽셀 쉐이더가 수행될 수 있도록 한다. 즉, 중복되는 삼각형 쌍과 관련된 스텐실 버퍼의 픽셀을 0의 값으로 세팅하고 나머지 영역은 1의 값으로 세팅한다. 스텐실 테스트를 동작시킨 상태에서 1차원 텍스쳐가 입혀진 사각형을 렌더링 함으로써, 스텐실 버퍼의 0 값을 가지는 픽셀은 렌더링 대상에서 제외된다. 또한, 인접하고 있는 삼각형들간의 충돌감지가 수행되지 않도록, 스텐실 테스트를 이용한 해당 삼각형쌍의 배제 작업이 수행되어야 한다. 스텐실 버퍼내의 인접 삼각형쌍과 관련된 픽셀을 0 값으로 세팅함으로써, 인접 삼각형쌍에 대한 충돌감지가 수행되지 않도록 한다. 한 삼각형의 인

접 삼각형은 에지와 정점을 공유하는 삼각형을 의미한다. <그림 6>은 스텐실 마스크를 위한 스텐실 버퍼의 세팅 상태와 스텐실 테스트의 결과를 보여주고 있다. 스텐실 버퍼에서 검은 픽셀이 0값을 가지는 픽셀로서 충돌감지 대상에서 제외되는 삼각형 쌍을 표현하고 있다. 스텐실 마스크의 결과에서 0의 값을 가지지 않고 색상을 가지는 픽셀들만이 즉, 스텐실 테스트를 통과한 픽셀들만이 상세한 삼각 메쉬쌍의 충돌감지를 수행하게 된다.

3. 삼각 메쉬쌍의 충돌감지

삼각형 메쉬의 정보를 저장한 텍스처를 이용하여 사각형을 렌더링하게 되면, 각 픽셀에 대한 픽셀 쉐이더가 자동으로 수행된다. 본고에서 소개하고 있는 방법에서는 삼각형간 교차 테스트 중 수행 효율성이 가장 높다고 평가 받고 있는 Möller의 방법[6]을 픽셀쉐이더로 구현하여 삼각형간 교차를 테스트하였다. 자동으로 수행되는 픽셀 쉐이더는 각 픽셀이 표현하는 두 개의 삼각형에 대하여 교차여부를 판단하고, <그림 7>과 같이 충돌감지 결과를 GPU내의 텍스처 메모리 공간이 오프 스크린 버퍼상에 저장한다. 즉, 충돌감지 결과를 포함하는 하나의 새로운 텍스처를



〈그림 7〉 충돌감지 쉐이더를 위한 입력 텍스처와 출력 텍스처

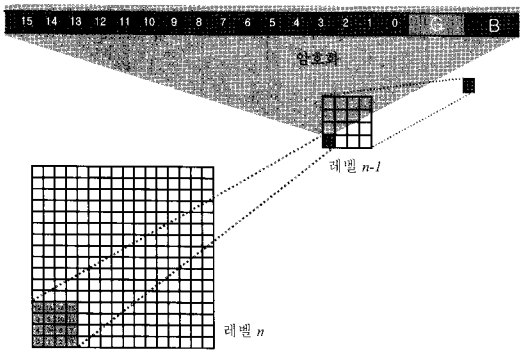
생성한다. <그림 7>에서 결과 텍스처 상에 표현된 적색의 픽셀들은 두 삼각형 쌍에 대하여 교차가 발생한 경우를 표현하고 있다.

4. 계층적 충돌 결과 조회

본 절에서는 GPU로부터 CPU로 읽어 오는 readback 단계의 오버헤드를 줄이기 위하여 조회하는 데이터의 크기를 가능한 최소화하는 기법에 대하여 설명하고자 한다.

가. 오프 스크린 버퍼의 계층적 암호화

충돌 가능 삼각형의 집합에 속한 삼각형의 개수가 n 이고, n 보다 큰 2의 자승의 숫자 중 최소의 수가 m 이라 하면, m 의 크기만한 1차원 텍스쳐에 두 객체의 기하학적 정보가 저장된다. 이 경우, 충돌감지의 최종 결과는 m x m 크기의 오프 스크린 버퍼에 저장된다. 오프 스크린 버퍼의 계층적 암호화를 위하여 최종 오프 스크린 버퍼의 내용을 텍스쳐로 하여 m/4 x m/4 의 사각형을 렌더링하고 렌더링 결과를 x m/4 크기의 상위 오프 스크린 버퍼에 저장한다. 즉, 하위 단계부터 상위 단계로 4 x 4 의 크기로 오프 스크린 버퍼의 크기를 줄여 가면서 상위 단계의 오프 스크린 버퍼에 암호화의 결과를 저장한다. <그림 8>은 암호화 방법을 보여주고 있다. 가장 하위 레벨의 오프 스크린 버퍼, 즉 최종 충돌감지의 결과는 충돌과 비충돌 이진의 결과를 갖는다. 그러므로, 최하위 레벨의 오프 스크린 버퍼의 16개 픽셀 값은 한 단계 상위레벨 버퍼의 한 픽셀에 암호화 하여 저장될 수 있다. 하위 버퍼를 4 x 4 크기의 픽셀 영역으로 나누고, 4 x 4 안에서는 각 픽셀에 대하여 0부터 15까지의 번호를 부여한다. 16개의 픽셀 값은 상위 레벨의 RED 필드

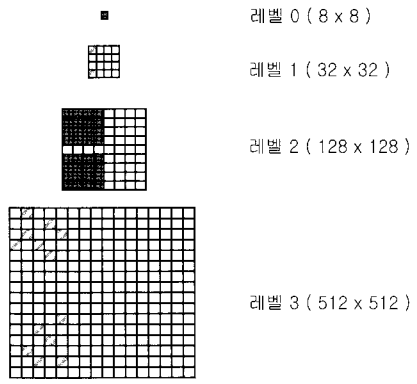


〈그림 8〉 오프 스크린 버퍼의 계층적 암호화

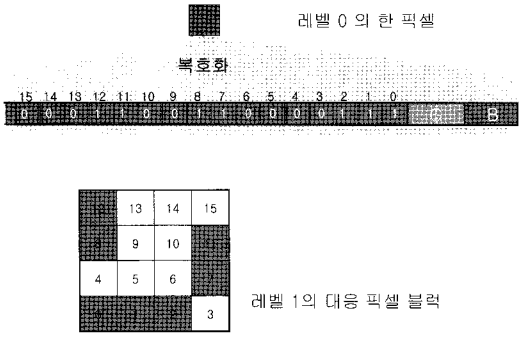
에 픽셀의 번호에 따라 해당 위치의 비트에 충돌과 비충돌의 결과를 저장한다. 충돌이 일어난 삼각형 쌍을 표현하는 픽셀의 경우는 1, 서로 분리되어 있는 삼각형 쌍을 표현하는 픽셀은 0의 값이 RED 필드의 각 비트에 저장된다. m/4 x m/4의 크기의 오프 스크린 버퍼를 암호화 하기 위해서는 마찬가지로 4 x 4 크기 단위로 암호화가 일어나는데, 단지 RED 값이 0 보다 큰 경우는 1로, RED 값이 0 인 경우에만 0으로 암호화 하여 상위 레벨의 RED 필드에 암호화 된다. 이러한 과정을 반복함으로써, 오프 스크린 버퍼의 계층적 암호 구조를 GPU의 텍스쳐 메모리 공간에 구축하게 된다.

나. 오프 스크린 버퍼 기반 계층적 복호화

<그림 9>는 오프 스크린 버퍼의 선택적 조회와 계층적 복호화 기법을 요약하여 보여주고 있다. <그림 9>(a) 는 오프 스크린 버퍼의 계층 구조와 각 계층별 조회 방법을 표현하고 있는데, 짙은 회색으로 표시된 픽셀은 명시적으로 조회된 픽셀들, 즉, GPU 메모리에서 CPU 메모리 공간으로 명시적인 READ 명령에 의하여 읽혀진 픽셀들을 의미한다. 그리고, 흐린 회색으로 표시된



(a) 오프스크린 버퍼의 계층 구조



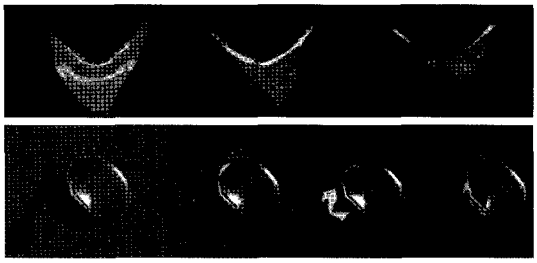
(b) 상위버퍼의 픽셀값과 하위 버퍼의 픽셀 블록간의 관계

〈그림 9〉 오프 스크린 버퍼의 선택적 조회와 계층적 복호화

픽셀들은 충돌 삼각형 쌍을 포함하고 있는 픽셀들로서 상위 버퍼의 각 픽셀의 정보를 분석하여 찾아진 픽셀들을 표현하고 있다. 즉, 흐린 회색의 픽셀들은 CPU 메모리 공간으로 데이터의 이동 없이 유추 해석된 픽셀들을 의미한다. <그림 9>(b)에서 짙은 회색의 픽셀들은 하위 버퍼의 대응 4 x 4 영역내에 충돌이 감지된 삼각형쌍을 포함하고 있는 픽셀들이 존재함을 의미하고 있다.

VI. 결 론

본고에서는 가상 객체간 충돌감지를 효율적으



〈그림 10〉 자체충돌 및 객체간 충돌 감지 결과

로 처리하기 위하여 GPU를 이용한 이미지 공간 기반 컬링기법과 3차원 기하학적 정보 기반 삼각메쉬쌍의 충돌 감지기법을 소개하였다. 또한, GPU-CPU간 데이터 조회 속도 개선을 위한 충돌결과에 대한 계층적 구조를 구축하고 이를 기반으로 충돌 부위에 대한 선택적 조회를 수행함으로써, 조회 성능을 향상시키는 방법에 대하여 설명하였다. <그림 10>은 충돌감지 결과를 보여주고 있다. 본고에서 소개한 기법은 충돌감지를 위한 전처리와 계층적 자료 구조 수정 오버헤드를 제거한 방법으로서, 입력 모델의 형태에 제약이 없이 적용가능한 방법이다.

참고문헌

[1] J. Krueger, R. Westermann, Linear algebra operators for GPU implementation of numerical algorithms, In Proceedings of SIGGRAPH, 908-916, 2003.

[2] J. Bolz, I. Farmer, E. Grinspun, P. Schroeder, Sparse matrix solvers on the GPU: Conjugate gradients and multigrid, In Proceedings of SIGGRAPH, 917-924, 2003.

[3] Nathan Carr, Jesse Hall, John Hart, The Ray Engine, In Proceedings fo the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware, pp.37-46, 2002.

[4] Timothy Purcell, Ian Buck, William Mark, Pat Hanrahan, Ray Tracing on Programmable Graphics Hardware, ACM Transactions on Graphics(Proceeding of ACM SIGGRAPH 2002), Vol.21, No.3, pp.703-712, 2002. http://graphics.stanford.edu/papers/rtongfx

[5] N.K. Govindaraju, M.C. Lin, D. Manocha, Fast self-collision culling in general environment using graphics processors, In Technical Report TR03-044 of University of North Carolina at Chapel Hill, 2003

[6] T. Möller, A fast triangle-triangle intersection test, J. Graphics Tools, 2(2), 25-30 , 1997

저자소개

최 유 주

1989년 2월 이화여자대학교 전자계산학과 학사
1991년 2월 이화여자대학교 전자계산학과 석사
2005년 2월 이화여자대학교 컴퓨터학과 박사
1991년 1월 ~ 1993년 6월 한국컴퓨터주식회사 기술연구소 주임연구원
1994년 5월 ~ 1999년 10월 포스데이타주식회사 기술연구소 주임연구원
2005년 9월 ~ 현재 서울벤처정보대학원대학교 컴퓨터응용기술학과 조교수

주관심 분야 : 컴퓨터그래픽스, 가상현실, HCI, 컴퓨터비전, 의료영상처리