GPU 를 이용한 이미지 기반 충돌검사

장한용 , 정택상 , 한정현
고려대학교 컴퓨터학과
jhymail@gmail.com,nanocreation@gmail.com,jhan@korea.ac.kr

Image-based Collision Detection on GPU

Han-Young Jang , TaekSang Jung , JungHyun Han
Department of Computer Science and Engineering, Korea University

요 약

This paper presents an image-space algorithm to real-time collision detection, which is run completely by GPU. For a single object or for multiple objects with no collision, the front and back faces appear alternately along the view direction. However, such alternation is violated when objects collide. Based on these observations, the algorithm has been devised, and the implementation utilizes the state-of-the-art functionalities of GPU such as framebuffer objects(FBO), vertex buffer object(VBO) and occlusion query. The experimental results show the feasibility of GPU-intensive collision detection and its performance gain in real-time applications such as 3D games.

Keyword : Computer graphics, GPU, Image-space collision detection, Visibility query

1. 서론

충돌 검사는 3D 게임이나 가상현실, 물리기반 시뮬레이션, 로봇공학 등 여러 영역에서 다루어지고 있는 문제이다. 충돌 여부를 판단하는 것은 실시간 시뮬레이션이나 동적 시스템에서 병목현상을 야기하는 부분으로 이를 해결하기 위해 크게 물체기반 충돌검사 기법(object-based collision detection)과 이미지 기반 충돌검사 기법(image-based collision detection)의 두 가지 방법이 연구되고 있다.

2. 관련 연구

물체기반 충돌검사 기법은 대부분 물체를 감싸는 구, AABB(axis-aligned bounding boxes)[2], OBB(oriented bounding box)[3] 등의 특별한 자료 구조로 표현된 바운딩 볼륨의 계층 정보를 이용하는 방법이다. 그러나 특별한 자료 구조를 구성하는데는 많은 시간이 소요되므로 매 프레임마다 바운딩 볼륨 정보가 갱신되어야 하는 역동적인 물체나 변형 가능한 물체에는 유용하지 않다.

물체기반 충돌검사와 다른 방법으로, 보는 방향에 따라 물체의 볼륨정보를 저장한 이미지를 이용한 이미지 기반 충돌 검사가 있다. 이미지 기반 충돌검사는 Shinya 와 Forgue [4]가 처음 제안하였다. 이 방법은 볼록 다면체(convex object) 를 깊이 버퍼에 그린 후 비교하여 충돌 여부를 확인한다. 이 후 그래픽 하드웨어의 기능을 이용하여 이미지 기반 충돌검사에 관한 여러 연구가 진행되었다.

주목 할만한 가장 최근의 연구로 Heidelberger 가 제안한 것이 있다[5]. 이 논문에서는 시각 방향에 나란한 광선이 물체를 통과할 때 진입점과 탈출점을 저장한 layered depth images(LDIs)를 구한 후 여러 물체의 LDI 들을 비교하여 충돌 여부를 판단한다. 이 알고리즘은 오목한 물체(concave object)도 처리 할 수 있으며 간단한 물체에 대해서 실시간으로 충돌 탐지를 할 수 있다. 하지만 복잡한 물체의 경우 매우 많은 수의 LDI 를 생성해야하는 단점이 있다.

일반적인 이미지기반 충돌검사 기법의 장점은

물체기반 충돌검사에서 바운딩 볼륨과 계층구조를 정립하는 것 과 같은 선처리 과정이 없다. 선처리 과정이 없기 때문에 구현과 적용이 간단하다. 게다가 CPU 에 비해 빠르게 발전하고 있는 GPU 를 이용한다는 점에서 CPU 만 이용하는 물체기반 충돌검사보다 이점이 있다.

반면 이미지기반 충돌검사는 GPU 에서 CPU 로 데이터 전송하는 readback 부하가 큰 단점이 있다. 예를 들면 Heidelberger 의 LDI 를 이용한 충돌검사에서[5] readback 부하를 줄이기 위해 32×32 에서 128×128 해상도의 작은 텍스쳐를 이용한다. 이때 주의 할 점은 충돌검사의 정밀도가 LDI 텍스쳐의 해상도에 비례 한다는 것이다. 낮은 해상도의 텍스쳐는 복잡한 물체에서 정확한 충돌검출을 보장해 주지 못한다. 이런 문제를 해결하기 위해 Govindaraju 는 CULLIDE 라는 알고리즘을 제안하였다[6]. CULLIDE 는 하드웨어의 visibility query 를 이용하여 잠재적 충돌가능 집합(PCS: potential colliding set)을 찾아 낸다. 하지만 CULLIDE 는 복잡한 선처리 작업을 필요로 한다.

또 다른 이미지기반 충돌검사의 단점은 렌더링 비용이 크다는 것이다. 물체의 모든 표면정보를 이미지에 담기 위해서는 여러 번의 렌더링이 필요하다.

마지막으로 이미지기반 충돌검사의 중요한 단점중의 한 쌍의 물체에 대한 충돌검사만 가능하다는 것이다.

이런 기존의 이미지기반 충돌검사의 단점을 해결하기 위해 이 논문에서는 충돌처리 전과정을 GPU 에서 수행하여 readback 을 최소화 하였고 self-collision 을 고려하지 않을 경우 렌더링 횟수를 획기적으로 절감하였으며 여러물체의 충돌검사를 한번에 처리 할 수 있는 방법을 제안하고 있다.

3. 닫힌 물체(Closed Object)의 특징

이 논문에서 제안하는 이미지기반 충돌 검사는 닫힌 물체만 고려한다. 닫힌 물체간의 충돌은 여러 이미지기반 충돌검사 논문에서 밝혀져 있듯

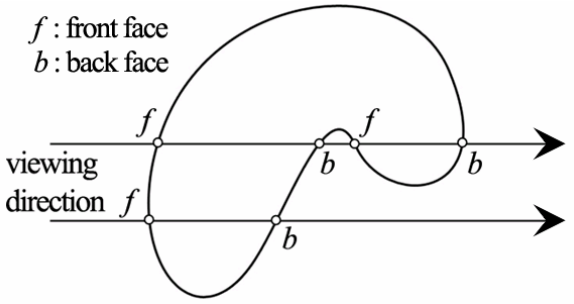

그림 1. 닫힌 물체에서 앞-뒷면 쌍

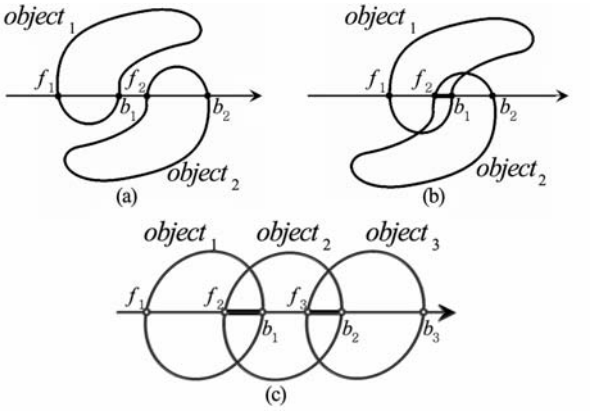

그림 2. 여러 물체에서 앞-뒷면 쌍: (a) 일치 쌍 (b) 불일치 쌍 (c) 불일치 쌍들

이 독특한 특징을 가진다.

닫힌 물체의 표면은 시각방향에 따라 앞면과 뒷면으로 나뉘어 질 수 있다. 이 때 그림 1 과 같이 시각방향에 따라 앞면과 뒷면이 반복적으로 나타나는 특징이 있다. 이것은 그림 2-(a)에서와 나타난 것과 같이 여러 물체가 있는 경우도 같은 특성을 가진다. 보다 정확히 말하자면 한 물체는 앞-뒷면을 쌍으로 가진다. 그림 2-(a)에서는 object_1 가 앞-뒷면 쌍으로 (f_1,b_1)를 가지고, object_2 에서는 (f_2,b_2)쌍을 가진다. 우리는 이러한 쌍을 일치 쌍이라고 부르겠다.

물체들 간에 충돌이 발생하게 되면 한 물체가 다른 물체 속으로 침투하게 된다. 그림 2-(b)에서 보는 것과 같이 object_1 과 object_2 가 충돌 하는 경우 (f_2,b_1)와 같은 쌍이 생성되며 이것은 일치쌍이 아니다. 우리는 이것을 불일치 쌍이라고 부르겠다.

이 것은 여러 오브젝트들 간에서도 같은 특징이 발견된다. 그림 2-(c)의 경우 (f_2,b_1), (f_3,b_2). 의 불일치 쌍이 발견된다. 그러면 충돌 검사는 불일치 쌍을 발견하는 것으로 수행될 수 있으며 (f_i,b_j), $i \neq j$ 과 같은 불일치 쌍이 발견되면 object_i 과

$object_j$ 가 서로 충돌한다고 판단할 수 있다.

4. 이미지기반 충돌검사

4-1 Depth Peeling 기법의 앞면 렌더링

이 논문에서 제안하는 알고리즘에서 모든 PCS 의 앞면은 반복적 *depth peeling* 기법을 통하여 텍스쳐에 그려진다.

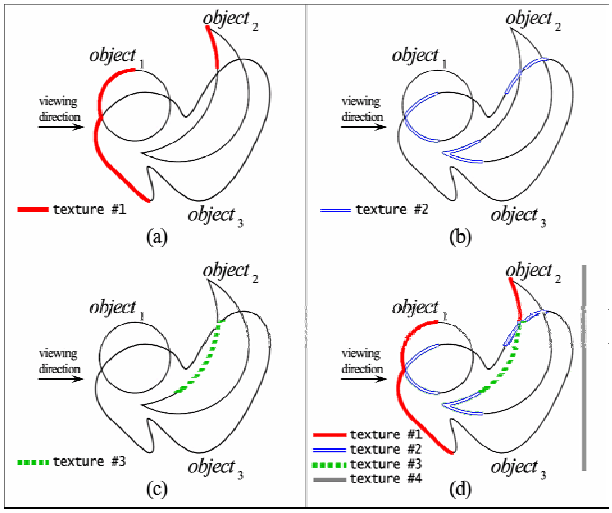

그림 3. Depth peeling iteration : (a) 초기화 (b) 첫 수행 (c) 두 번째 수행 (d) 세 번째 수행

그림 3 은 네 장의 텍스쳐에 depth peeling 을 이용하여 물체정보를 저장하는 방법을 보여주고 있다. 3-(a)에서 보여주고 있는 초기화 단계에서는 렌더링 옵션에서 depth test 를 가능하게 하고 물체의 앞면만 렌더링 하도록 설정한다. 그리고 모든 `texture#1` 의 깊이값을 갖도록 초기화 한다. 그러면 `texture#1` 에 시점에서 가까운 앞면이 렌더링 된다. 렌더링은 셰이더를 이용하여 수행되며 텍스쳐의 텍셀에는 물체의 ID 와 깊이 값이 저장된다.

`texture#2` 에 렌더링할 때는 초기화 단계와는 다른 셰이더가 사용된다. 이 셰이더는 이전 단계의 결과 텍스쳐인 `texture#1` 을 입력 받는데, 이 텍스쳐에 저장된 ID 와 다른 ID 를 가지면서 깊이가 텍스쳐에 저장된 깊이보다 깊은 값을 가지는 pixel 만 출력해 준다. 그러면 그림 3-(a)에 색칠된 부분의 정보가 `texture#2` 에 저장 된다.

렌더링이 끝나면 그래픽 하드웨어에서 지원하는 occlusion query 를 이용하여 그려진 픽셀의 개수를 확인한다. 만약 그려진 픽셀의 개수가 0 개라면 그려진 것이 아무것도 없는 것을 의미하므로 렌더링을 중지한다. 그림 3-(b)는 그려진 영역이 존재하므로 occlusion query 는 0 이 아닌 값을 반환하고 같은 방법으로 occlusion query 가 0 을 반환할 때 까지 렌더링을 반복적으로 수행된다.

두 번째 렌더링의 경우 그림 3-(c)에 보여지는 앞면이 `texture#3` 에 렌더링 된다. 이 때도 occlusion query 는 0 이 아닌 값을 반환한다. 세 번째 렌더링에서 `texture#4` 에는 그림과 같이 아무것도 그려지지 않고 occlusion query 는 0 을 리턴한다. 그러면 반복적 depth peeling 을 이용한 앞면 렌더링은 중단되며 네 텍스쳐에 저장된 정보는 그림 3-(d)와 같다.

4-2 픽셀과 텍셀 비교를 이용한 충돌검사

충돌 검사단계에서는 렌더링 설정을 뒷면만 그려지도록 한다. 전 단계에서 생성된 앞면이 그려진 모든 텍스쳐들은 이번 단계에서 셰이더의 입력정보로 사용되며 이 정보를 이용한 한번의 뒷면 렌더링으로 물체의 충돌 여부를 판단한다.

뒷면은 이전 단계에서 생성된 앞면들의 사이의 영역에 위치한 픽셀만 검사한다. 그림 3 의 경우는 모든 뒷면이 렌더링 된 앞면들 사이에 위치한다. 이 조건이 왜 필요한지는 논의 부분에서 언급하고 있다.

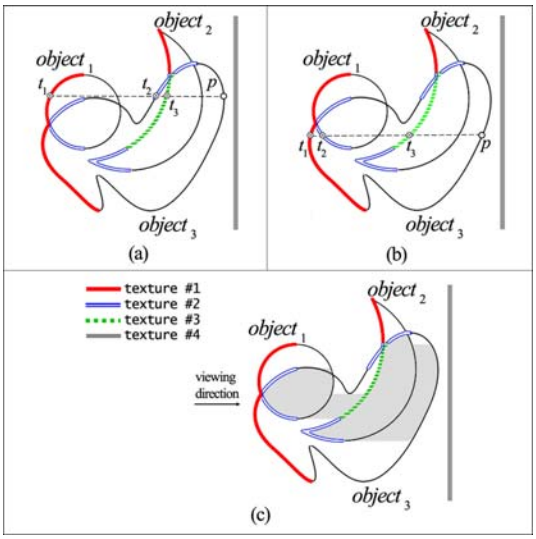

그림 4. 불일치 쌍을 통한 충돌검사 : (a) (t_3,p) (b) (t_2,p)와(t_3,p) (c) 충돌영역

뒷면이 렌더링 되어 뒷면의 한 픽셀 p 가 shader 에 의해 처리될 때 p 는 뒤에서 앞으로의 순서로 p 앞에 있는 texture 의 texel 정보를 읽어 온다. 그림 4-(a) 에서 픽셀 p 앞에는 `texture#1,#2,#3` 에서 추출된 텍셀 t_1,t_2,t_3 가 존재 한다. 픽셀 p 는 처음 t_3 값을 읽어서 자신의 ID 와 t_3 에 저장된 ID 를 비교를 해본다. t_3 는 $object_2$ 에 포함되어 있고 p 는 $object_3$ 에 포함되어 있다. 그러므로 p 와 t_3 의 ID 정보는 다르며 (t_3,p) 는 불일치 쌍을 구성하고 이 것은 $object_2$ 과 $object_3$ 가 충돌 함을 의미한다. 다음 t2 와 의 비교에서 p 와 t_2 는 같은 ID 를 가지며 일치 쌍을 구성한다. 그러면 픽셀 p 에 대한 충돌 검사는 중단 한다.

그림 4-(b)는 다른 픽셀 p 에 대한 충돌 검사를 보여 준다. 여기서 p 는 일치 쌍 t_1 을 만나기 전까지 t_2 와 t_3 를 만나 불일치 쌍을 형성하고 $object_1$ 과 $object_3$ 그리고 $object_2$ 과 $object_3$ 가 충돌 하였음을 알려 준다. 그림 4-(c)는 픽셀들이 앞쪽 텍셀들과 ID 를 비교하였을 때 불일치 쌍이 발견되는 영역을 표시한 것이다. 이 영역은 충돌 부분을 모두 포함하고 있으며 충돌 정보는 뒷면 렌더링 결과로 render target 에 저장된다. 우리의 알고리즘은 전체 충돌 검사 과정에서 이 render target 을 한번만 readback 한다.

5. 논의점들

그림 5 와 그림 6 은 충돌이 발생하는 경우와 발생하지 않는 경우에 대하여 이 논문의 충돌검사 과정을 나타낸 것이다. 2 장에서 언급하였듯이 이미지기반 충돌검사의 단점 중 가장 중요한 것은 bandwidth 제약에 따른 readback 부하가 크다는 것이다. 이 논문의 알고리즘은 모든 충돌 검사 과정이 GPU 상에서 처리되며 최종 충돌검사 결과를 PCS 를 통해 계산한 예상되는 충돌가능 지역에 대해서 한번만 readback 하기 때문에 readkback 부하가 거의 없다. 이것은 앞에서 언급한 CULLIDE 알고리즘 보다도 적은 readback 이 필요하다.

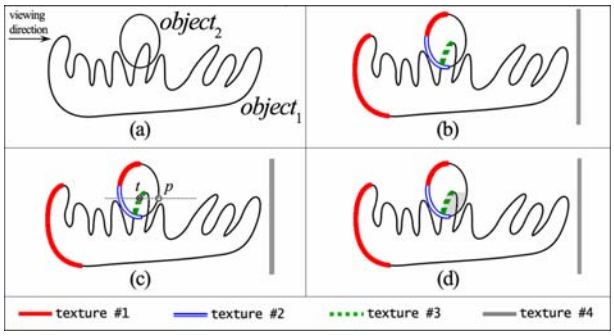

그림 5 충돌이 발생하는 예 : (a) 물체들 (b) depth-peeling 결과 (c) 불일치 쌍 (d) 충돌 영역

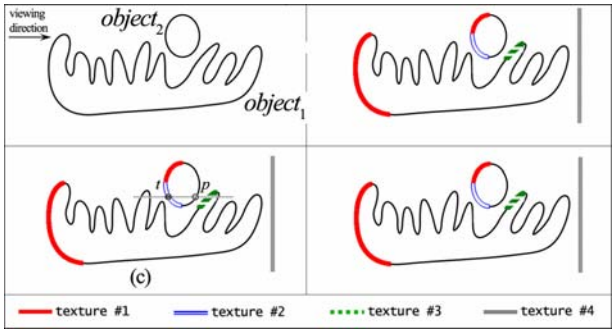

그림 6 충돌이 발생하지 않는 예 : (a) 물체들 (b) depth-peeling 결과 (c) 일치 쌍 (d) 충돌 영역 없음

readback 부하가 줄어들면 고해상도의 텍스쳐를 사용하여도 readback 의 부담이 적으며 LDIs 를 이용한 충돌검사기법에서 낮은 해상도의 텍스쳐를 사용해서 충돌검사 정밀도가 낮아 질 수 있는 한계를 극복 할 수 있다.

두 번째로 이미지에 물체의 전체 표면 정보를 담기 위해 불필요하게 여러 번 렌더링을 수행했던 기존 알고리즘과 달리 이 논문의 알고리즘은 렌더링 횟수를 획기적으로 줄였다. 그림 5 와 그림 6 의 경우 물체의 표면이 복잡한 구조를 가지고 있다. 기존의 LDIs 를 이용한 충돌검사 기법에서 그림 5 와 그림 6 의 충돌을 검사하려면 총 22 번의 렌더링이 필요하다. 그러나 이 논문에서 제안한 알고리즘은 5 번의 렌더링으로 충돌 검사를 수행한다.

세 번째로 이 논문에서 제안한 알고리즘은 한 쌍의 물체 뿐 아니라 여러 물체의 충돌 검출을 동시에 수행 할 수 있다. 이것은 그림 4-(b)에서 나타내고 있으며 때문에 3D 게임에서 전체 신의 충돌검사를 실시간에 수행 할 수 있다.

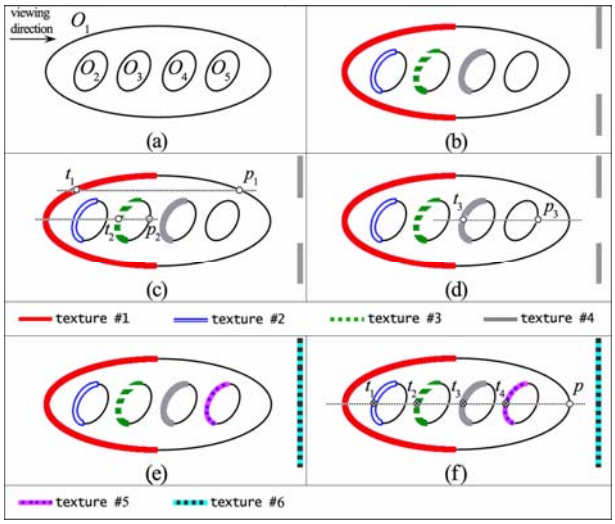

그림 7. 부분적 충돌검출 : (a) 물체들 (b) 4 개의 텍스쳐를 이용한 depth-peeling (c) 일치 쌍 (d) 검사 불능 쌍 (f) 6 개의 텍스쳐를 이용한 depth-peeling (f) 불일치 쌍들

그림 7 은 이 논문에서 제안한 알고리즘이 실패 하는 경우를 나타낸 것이다. 우리가 4 개의 텍스쳐만 가지고 depth-peeling 을 한다고 가정해 보자. 그림 7-(b)에서 나타나 듯이 필요한 모든 앞면이 4 장의 텍스쳐에 저장 되지 못한다. 이 경우 뒷면을 이용한 충돌 검사를 수행 하면 그림 7-(c)와 같이 충돌하지 않는 (t_3,p_3)의 쌍을 발견하게 된다.

이를 해결 하기 위한 방법은 depth-peeling 을 이용한 앞면 렌더링 단계에서 충분한 texture 를 제공하는 것이다. 그림 7-(e)와 7-(f)는 텍스쳐 6 개를 이용하여 정확한 충돌을 검출해 내는 예를 보여 주고 있다. 하지만 불행이도 무한한 텍스쳐를 제공하는 것은 바른 해결책이 아니다. 강건한 알고리즘을 위해서는 무한한 텍스쳐가 보장되어야 할 것이기 때문이다. 만약 그림 7-(b)와 같이 충분한 텍스쳐가 없는 경우 이 알고리즘은 텍스쳐에 저장된 앞면들 사이의 뒷면만 테스트 하고 렌더링 되지 않은 앞면에 대한 렌더링과 나머지 뒷면에 대한 충돌 검사를 위해 충돌 검사 를 다시 수행한다. 이 때는 모든 충돌 검출을 보장하지 못한다. 하지만 그림 7 과 같이 물체들이 가지는 픽셀이 정확하게 시각방향에서 겹치는 경우는 매우 드물게 발생한다. 6 장 구현에서 매우 복잡한 물체를 4 개의 텍스쳐 만으로 정확한 충돌 검사를 하는 예를 보여 준다.

6. 구현

제안된 알고리즘은 C++, OpenGL and Cg 를 이용하여 3.6 GHz 의 Pentium4 CPU 와 2GB 메모리, ATI Radeon X800XT GPU 를 갖춘 PC 에서 구현 되었다. 비동기적 GL_NV_occlusion_query 를 depth-peeling 을 위해 사용하였으며 VBO, FBO 와 같은 그래픽 하드웨어의 다양한 기능이 사용되었다. 32-bit 정밀도로 1024*1024 해상도를 가진 텍스쳐가 depth-peeling 에 사용되었다.

제안된 알고리즘은 다양한 물체를 대상으로 충돌검사를 수행했다. 그림 8 과 검사에 사용된 복잡한 물체들 보여 주고 있다. 그림 9 는 충돌 검사 시간을 측정한 것으로 왼쪽은 충돌 영역이 좁은 경우를 나타내고 오른쪽은 충돌 영역이 넓은 경우를 나타낸다.

그림 8. 검사에 사용된 모델

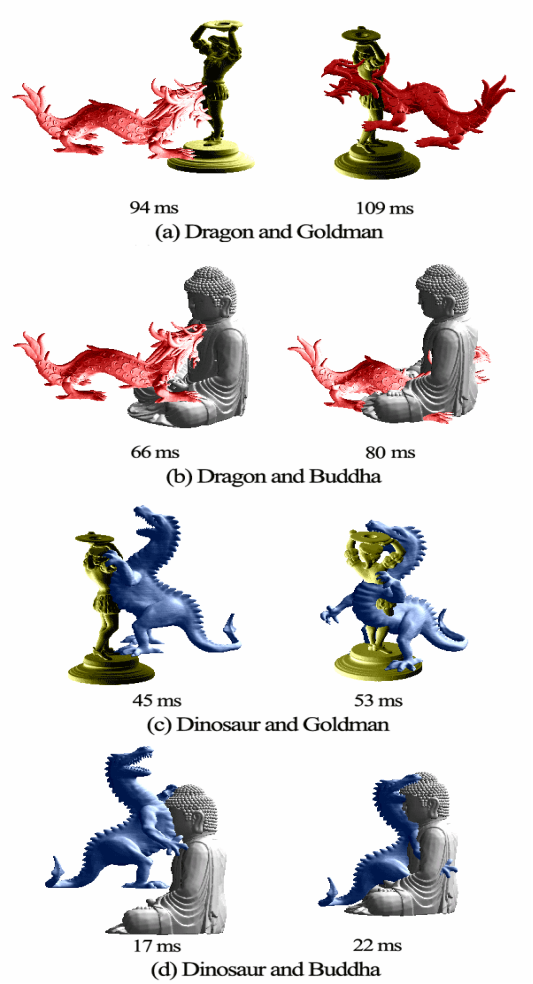

그림 9. 충돌 검사의 성능 측정

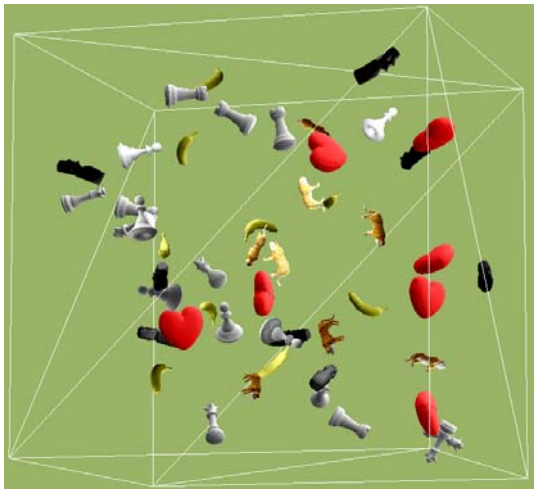

그림 10. 여러 물체에 대한 충돌 검사

물체 개수	충돌검사 수행시간	분당 충돌 발생 횟수	FPS
50	3.5ms	408	193
40	2.3ms	328	230
30	1.3ms	142	345

표 1. 여러 물체 충돌검사의 성능 측정

그림 10 은 한번에 여러 물체의 충돌 검사를 수행한 예이다. 검사에 사용된 물체는 평균 3K 개의 삼각형으로 구성되어 있다. 이 방법은 이미지 기반 충돌검사를 하기 전에 물체를 감싸는 구를 이용한 충돌 검사를 수행하여 얻은 PCS 에 대하여 충돌 검사를 수행 한다. 아래는 충돌 검사의 성능을 측정한 표이다.

7. 결론

이 논문에서 매우 효율적인 실시간 이미지기반 충돌검사 알고리즘을 제안하고 있다. CPU 에서 PCS 를 계산하고 GPU 에서 충돌 검사를 수행한다. 이 알고리즘은 readback 부하가 적으며 기존 알고리즘의 불필요한 렌더링을 줄였고 한번에 여러 물체의 충돌 검사를 수행한다. 이 알고리즘은 GPU 의 유용한 기능들을 적절히 활용하여 3D 게임에서 사용할 수 있는 실시간 충돌검사를 가능하게 하였다.

8. 참고문헌

[1] G. van den Bergen. Efficient collision detection of complex deformable models using AABB trees. Journal of Graphics Tools, 2(4):1–14, 1997.

[2] S. Gottschalk, M. C. Lin, and D. Manocha. OBBTree: A hierarchical structure for rapid interference detection. In Proc. of ACM SIGGRAPH, pages 171–180, 1996.

[3] M. Shinya and M. Forgue. Interference detection through rasterization. Journal of Visualization and Computer Animation, 2(4):131–134, 1991.

[5] B. Heidelberger, M. Teschner, and M. Gross. Detection of collisions and self-collisions using image-space techniques. In Proc. Computer Graphics, Visualization and Computer Vision WSCG’04, pages 145–152, 2004.

[6] N. K. Govindaraju, D. Knott, N. Jain, I. Kabul, R. Tamstorf, R. Gayle, M. C. Lin, and D. Manocha. Interactive collision detection between deformablemodels using chromatic decomposition. In Proc. of ACM SIGGRAPH 2005, pages 991–999, 2005.