GPU를 이용한 이미지 공간 충돌 검사 기법

GPU-based Image-space Collision Detection among Closed Objects

장한용, Han-Young Jang*, 정택상, TaekSang Jeong**, 한정현, JungHyun Han***

요약 본 논문은 GPU를 활용한 이미지 공간 실시간 충돌 검사 기법을 설명한다. 닫힌 물체들이 충돌하지 않는 경우, 뷰잉 레이를 따라 물체의 앞면과 뒷면이 번갈아 가며 나타나는 것을 확인 할 수 있다. 그러나 물체 간 충돌이 일어나는 경우 이 현상이 깨어지게 된다. 이러한 특성에 기반하여 본 논문은 충돌 검사에 필요한 최소한의 표면 정보만 텍스쳐에 기록하여 충돌 검사를 수행하는 기법을 제안한다. 이 기법은 GPU의 framebuffer object 와 vertex buffer object, 그리고 occlusion query 등의 기능을 활용한다. 이러한 GPU의 기능을 이용하면 통상적인 이미지 기반 충돌 검사에서 사용하는 multi-pass rendering 과 context switch 부하를 줄일 수 있다. 즉 기존의 이미지 기반 충돌 검사에 비해 적은 렌더링 횟수와 적은 렌더링 부하를 가진다. 본 논문에서 제안된 알고리즘은 변형체나 복잡한 물체에도 적용이 가능하며, 3D 게임이나 가상현실과 같은 실시간 어플리케이션에 적용될 수 있는 성능을 발휘한다.

Abstract 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 propose the depth peeling method which renders the minimal surface of objects, not whole surface, to find colliding. The Depth peeling method utilizes the state-of-the-art functionalities of GPU such as framebuffer object, vertexbuffer object, and occlusion query. Combining these functions, multi-pass rendering and context switch can be done with low overhead. Therefore proposed approach has less rendering times and rendering overhead than previous image-space collision detection. The algorithm can handle deformable objects and complex objects, and its precision is governed by the resolution of the render-target-texture. The experimental results show the feasibility of GPU-based collision detection and its performance gain in real-time applications such as 3D games.

핵심어: GPU, Image-space collision detection, Occlusion query

본 연구는 대학 IT연구센터 지원사업 (IITA-2005-(C1090-0501-0019)) 및 2005년 GRC 연구결과로 수행되었음.

*주저자 : 고려대학교 컴퓨터학과 대학원생
**공동저자 : 고려대학교 컴퓨터학과 대학원생
***교신저자 : 고려대학교 컴퓨터학과 교수; e-mail: jhan@korea.ac.kr

1. Introduction

Collision detection is a fundamental problem in many applications such as computer graphics and animation, 3D games, virtual reality, physically-based simulation, and robotics. It is often the major computational bottleneck in real-time simulation of complex and dynamic systems. A lot of algorithms for collision detection have been proposed, and the algorithms based on triangulated models can be classified into two broad categories. One is *object-space approach* and the other is *image-space approach*.

This paper proposes a new technique for image-space approach. The proposed algorithm is quite simple, is easy to implement using shaders, and shows superior performance. The simplicity and efficiency of the algorithm are attractive for real-time applications such as 3D games.

The structure of this paper is as follows. Section 2 reviews the related work, and discusses the advantages and disadvantages of the traditional image-based approach. Section 3 describes the features of collision among closed objects. Based on the features, Section 4 presents the collision detection algorithm, which overcomes the disadvantages of the traditional approach. Section 5 discusses the strength and weakness of the proposed algorithm. Section 6 shows the test results, and Section 7 concludes the paper.

2. Related Work

In the object-space approach, most of the proposed algorithms are accelerated by utilizing spatial data structures which are often hierarchically organized and are based on bounding volumes such as bounding spheres [1,2], axis-aligned bounding boxes [3,4], oriented bounding boxes [5], discrete orientation polytopes [6], and quantized orientation slabs with primary orientations [7]. These data structures are used to cull away portions of an object that are not in close proximity. However, the spatial data structures do not help a lot in identifying the closest features between pairs of objects in close proximity, especially for dynamic environments and deformable objects, where both of the hierarchy and bounding volumes should be updated. Some algorithms proposed for handling deformable objects either can handle simple objects only or have been designed for a limited class of objects such as cloth.[8,9]

In contrast with the object-space approach, the image-space approach typically measures the volumetric ranges of objects along the viewing direction, and then compares the ranges to detect collision. The trend started with the work of Shinya and Forgue [10], where the depth layers of convex objects are rendered into depth buffers, and then compared for interference checking. Since then, various algorithms for image-space approach have been proposed, and have attempted to maximally utilize the graphics hardware's functionality [11,12,13,14].

The most recent efforts in the image-space approach include the work by Heidelberger *et al.* [15,16], which is useful to discuss both of the advantages and disadvantages of the image-space approach. In their work, layered depth images (LDIs) are computed, one for each object, where an LDI stores entry and leaving points of parallel viewing rays with respect to an object. Then, collision is detected through Boolean intersection on LDIs. This approach can handle concave objects, and shows real-time performance for simple objects. However, LDI generation requires a considerable amount of time for objects with complex geometry.

In general, the advantages of the image-space approach can be listed as follows. Unlike the object-space approach which requires non-trivial pre-processing for computing bounding volumes and their hierarchy, the image-space approach does rarely require pre-processing. Partly due to absence of the pre-processing, the image-space approach is easy to implement. It can also effectively handle deformable objects and dynamic environments. Moreover, it usually employs graphics hardware or GPU which has been evolving at a rate faster than Moore's law, while the object-space approach performs virtually all collision tests on CPU.

The image-space approach also reveals disadvantages. First of all, virtually all of the image-space algorithms proposed so far perform collision tests using both CPU and GPU, and suffer from the limited bandwidth between them, i.e. the *readback* problem. In the LDI-based algorithm by Heidelberger *et al.* [15,16], for example, the CPU reads the LDIs from the GPU's back buffers, and then tests the LDIs for Boolean intersection. Due to the limited bandwidth, the sampling resolution (LDI resolution) is usually made low, 32x32 through 128x128. Note that, however, the accuracy of the collision detection is governed by the LDI precision, and low resolutions do not guarantee accurate detection of collision for complex objects. As an effort to overcome the readback problem, Govindaraju *et al.* [17] proposed an algorithm named CULLIDE, which computes a potentially colliding set (PCS) through hardware visibility query. However, it requires off-line pre-processing or setup stage, which is quite complex. Recently, an efficient pre-processing technique for the CULLIDE algorithm, named chromatic decomposition [18], has been suggested, but it also has limitations. For example, the objects to be tested for collision are limited to polygonal meshes with fixed connectivity. For a deformable mesh the topology of which may vary frame by frame, the time-consuming pre-processing has to be executed per each frame.

Another major disadvantage of the image-based approach lies in the rendering cost because it generally renders the entire surface areas of the objects to be tested for collision. The computational bottleneck may lie in GPU for arbitrarily-shaped complex objects.

Finally, the input to most of the image-space algorithms is limited to a pair of objects, not many objects in a large scene. Knott and Pai [19] proposed an algorithm that can handle large number of objects. However, their algorithm is based on wireframe rendering, and therefore is inherently fragile, i.e. may often miss obvious collision.

In order to resolve the problems of the traditional image-space approach, this paper proposes a GPU-based algorithm, where the entire collision test is run by GPU and readback is minimized. Assuming no self-collision, the rendering cost has been dramatically reduced. Further, it detects collision of the entire scene, not of only a pair of objects.

3. Features of Collision among Closed Objects

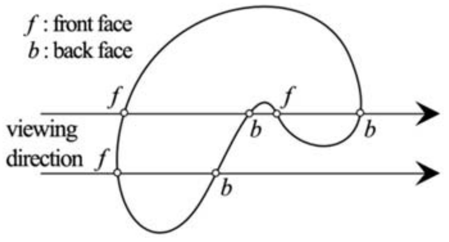

Figure 1. Front-back face pairing in a closed object

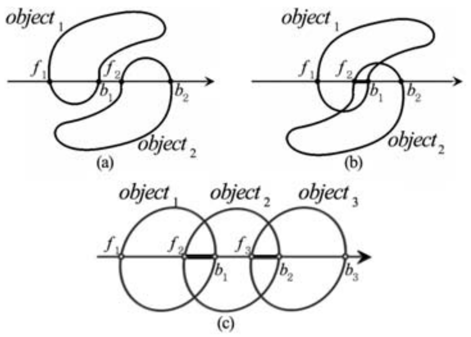

Figure 2. Front-back face pairing in multiple objects: (a)consistent pairs (b) inconsistent pair (c) inconsistent pairs

The image-space collision detection algorithm proposed in this paper handles only closed objects. Collision between closed objects reveals distinct features, which have been observed in many image-based algorithms. The front and back faces of a closed mesh appear alternately along a viewing ray, as illustrated in Fig. 1. The observation can be generalized for a scene of multiple objects with no collision, as illustrated in Fig. 2-(a). More precisely, a front face of an object is paired with a back face of the same object. In Fig. 2-(a), we can find two such pairs, (f_1,b_1) of *object*$_1$ and (f_2,b_2) of *object*$_2$. We call them *consistent pairs*.

Collision takes place when an object penetrates or touches another object. In Fig. 2-(b), *object*$_1$ penetrates *object*$_2$, and then consistent pairing is violated, i.e. right in front of b_1 lies f_2, not f_1. We call (f_2,b_1) an *inconsistent pair*. Such observation holds for multiple object collision, as illustrated in Fig. 2-(c), where we can find two inconsistent pairs, (f_2,b_1) and (f_3,b_2). Then, collision detection resorts to the task of finding inconsistent pairs (f_i,b_j), $i \neq j$, which tells us that *object*$_i$ collides with *object*$_j$.

4. Image−space Collision Detection

Each object in the scene is associated with an axis−aligned bounding box (AABB). The potentially colliding set (PCS) is computed using the AABBs. The AABBs of the PCS determine the dimension of the *orthographic view volume*, within which the image−space collision detection is invoked. The proposed algorithm can be described as *virtual ray casting* in the sense that collision is detected for a set of parallel rays.

4.1 Front−face Rendering through Depth Peeling Iteration

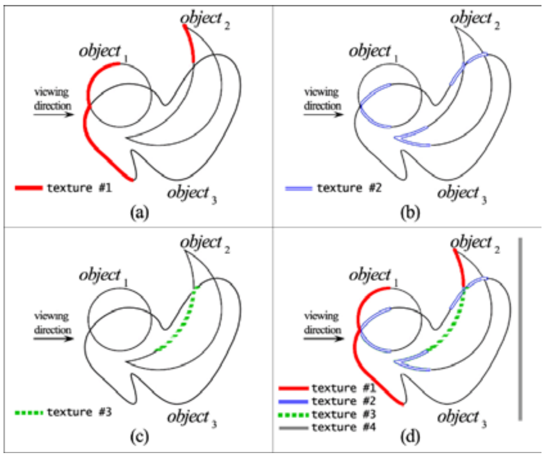

Figure 3. Depth peeling iteration: (a) initialization (b) 1st iteration (c) 2nd iteration (d) 3rd iteration

All of the front faces in the PCS are rendered onto textures through so−called *depth peeling* iteration, where the front faces are peeled off layer by layer. Fig. 3 illustrates an example with four texture maps, numbered #1, #2, #3, and #4. At the initialization stage, texture #1 is filled with the depth value of the scene background. Then, all objects in the scene are rendered with the depth test enabled. As a result, texture #1 contains the depth values of the first−layer front faces, as illustrated in Fig. 3−(a). Rendering is done by a shader, and each texel of the texture contains one more piece of information: the object ID of the rendered pixel.

The iteration starts by initializing texture #2 with the depth value of the scene background. A new shader is used for the iteration, which is different from the one used for the initialization stage. The shader takes the rendering result of the previous stage (texture #1) as a texture, and discards a pixel if it is not deeper than the one in the texture or its object ID is identical to that of the corresponding texel. Then, only the second−layer of the front faces survives, and their depth values are stored into the new texture (texture #2), as shown in Fig. 3−(b).

At the end of iteration, *occlusion query* [20] is invoked, which returns the number of pixels that have passed the depth test. If the return value is 0, it means that no object is rendered by the shader, and the iteration stops. Otherwise, the iteration resumes. In Fig. 3−(b), the second−layer front faces of the scene have been rendered. Therefore, occlusion query returns non−0 value, and the 2nd iteration starts.

During the 2nd iteration, a new texture (texture #3) is initialized with the background depth, and updated using the texture from the previous stage (texture #2). The result is shown in Fig. 3−(c). Occlusion query is invoked and returns non−0 value. The 3rd iteration starts with a new texture (texture #4) initialized with the background depth. However, nothing is rendered, and occlusion query returns 0. Therefore, the depth peeling iteration stops. Note that texture #4 is not updated at all, and all of its texels contain the background depth. Fig. 3−(d) shows the result of the depth peeling process. The depth peeling iteration is implemented using *framebuffer object* (FBO) [21], which has recently been specified as a collection of local buffers such as color, depth, stencil, and accumulation buffers. In this new specification, rendering destinations can be off−screen renderbuffers or textures. They can be shared among FBOs. Therefore, the texture rendered in an iteration will be available for the next iteration, at the minimum cost of context switching.

4.2 Collision Detection through Pixel−by−texel Comparison

Either the front faces or the back faces of a scene can be rendered by changing the culling mode. When all the front faces are rendered into textures, the culling mode is changed, and collision test starts by rendering all the back faces at a time.

A pixel from a back face can be rendered only when it is in the range delimited by the first and last textures produced in the depth peeling process. They are texture #1 and texture #4 in the example. In Fig. 3-(d), all the back faces are in the range. Section 5 will discuss why such condition is needed.

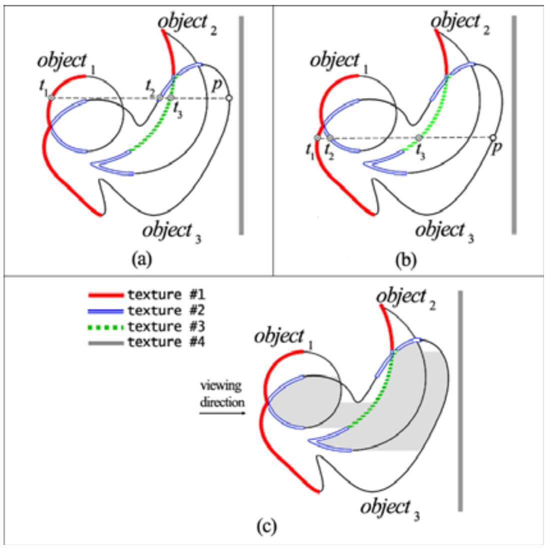

Figure 4. Collision detection using inconsistent pairs: (a)(t_3,p) (b) (t_2,p) and (t_3,p) (c) collision area

When a pixel p from a back face is rendered, the texture maps of the front faces lying in front of p are traversed in the back-to-front order. In Fig. 4-(a), three texels t_1, t_2 and t_3 (from texture #1, texture #2, and texture #3, respectively) are found to lie in front of p. The back-to-front traverse starts from t_3. By retrieving the object ID recorded in the texture, we can find that t_3 belongs to $object_2$. Note that p belongs to $object_3$. They are different objects, and therefore (t_3,p) is taken as an inconsistent pair, which judges that $object_2$ and $object_3$ collide. The back-to-front traverse continues and finds that t_2 and p are from an identical object, i.e. (t_2,p) is a consistent pair. Once such a consistent pair is found, the traverse stops. In the example, the texel t_1 is not visited. Along the viewing ray in Fig. 4-(a), collision between $object_2$ and $object_3$ is detected.

Fig. 4-(b) shows collision test with another back-face pixel. Two inconsistent pairs, (t_2,p) and (t_3,p), are found, which imply that $object_3$ collides with both of $object_1$ and $object_2$. A set of colliding-object pairs is associated with a viewing ray. For example, {($object_1$,$object_3$),($object_2$,$object_3$)} is associated with the ray in Fig. 4-(b). Of course, a set will be empty if no collision is detected. The shaded area in Fig. 4-(c) encompasses all viewing rays along which collision has been detected. The colliding-object pair sets for the shaded area are compressed and rendered into a render target, which is then readback by CPU. Our algorithm requires only a single readback for the PCS area in the screen.

5 Discussions

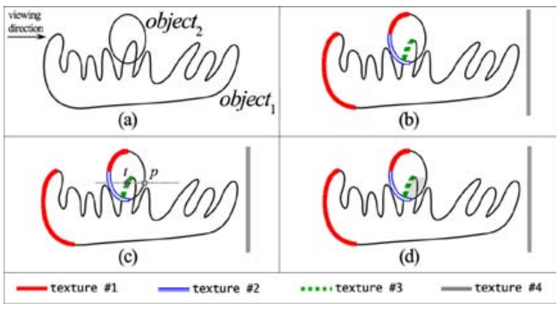

Figure 5. Collision example: (a) objects (b) depthpeeling results (c) inconsistent pair (d) collision area

As discussed in Section 2, the traditional image-based approach suffers from three major drawbacks. The algorithm proposed in this paper resolves them successfully. First of all, in the current implementation, CPU computes only the potentially colliding set (PCS), and the entire collision test is run by GPU. The authors believe that such an approach is the first of its kind in collision detection research field. As discussed in Section 4.2, CPU simply reads the collision test results (in the form of colliding-object pair sets) for the PCS, and such readback is done only once for a PCS. The algorithm does rarely suffer from the readback problem. For example, the algorithm presented in this paper requires less readback than the CULLIDE algorithm [17,18].

An important thing to note is that, unlike the LDI approach [15,16], the texture resolution governing the accuracy of collision detection can be made identical to the PCS size in the screen. Then, the precision of the collision detection becomes compatible with the user's visual perception. This makes the proposed algorithm distinguished from other real-time image-space collision detection algorithms.

The second drawback of the traditional approach has also been resolved: the proposed image−space algorithm does not necessarily render the entire surface of an object. For an arbitrarily−shaped complex object, it significantly increases efficiency. Fig. 5 illustrates the collision detection algorithm with two objects. In Fig. 5, only four textures are used for depth peeling, and inconsistent pairs are found. In contrast, for example, the LDI approach computes nine pairs of entry and leaving points (volumetric ranges) by rendering the entire surface, for the cases of Fig. 5.

Third, the proposed algorithm can detect collision of the entire scene, not of only a pair of objects. The performance gain obtained by processing the entire scene at a time is especially useful for real−time applications such as 3D games.

6. Implementation

The proposed algorithm has been implemented in C++, OpenGL and Cg on a PC with 3.6 GHz Intel Pentium4 CPU, 2GB memory, and ATI Radeon X800XT GPU. Various functionalities of the graphics hardware are exploited, e.g. the asynchronous GL_NV_occlusion_query for depth peeling and collision detection, GL_ARB_vertex_buffer_object for vertex buffer, EXT_framebuffer_object for off−screen rendering, etc. Four 32−bit floating−point textures are used for depth peeling.

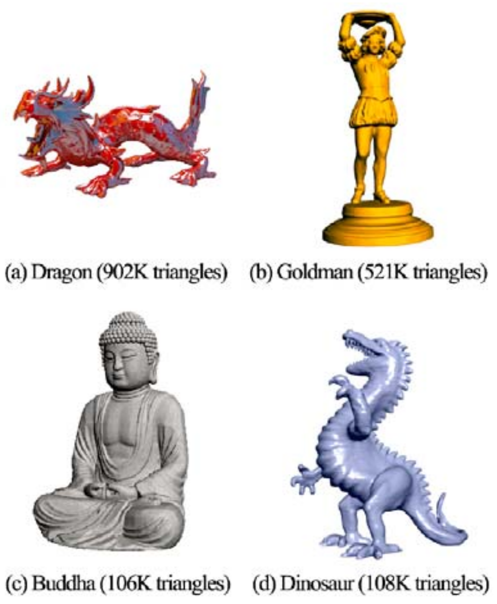
(a) Dragon (902K triangles)
(b) Goldman (521K triangles)
(c) Buddha (106K triangles)
(d) Dinosaur (108K triangles)

Figure 6. Test objects

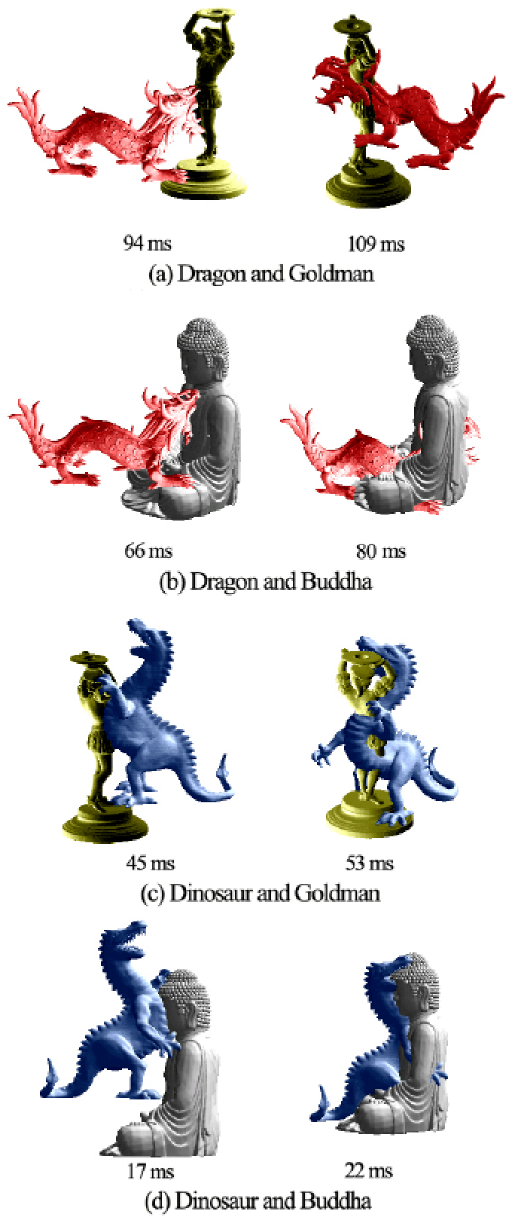

Figure 7. Performance evaluation for collision between two objects

The proposed algorithm has been tested with various objects, and shown in Fig. 6 are complex models among them. Fig. 7 lists the measured execution time for collision detection between pairs of the complex models. In each sub−figure, the right one shows heavy intersection while the left one shows light intersection. When heavily intersected, more time is need for collision detection. In the case of light intersection, objects are rendered four times on average. In the case of heavy intersection, objects are rendered six times on average.

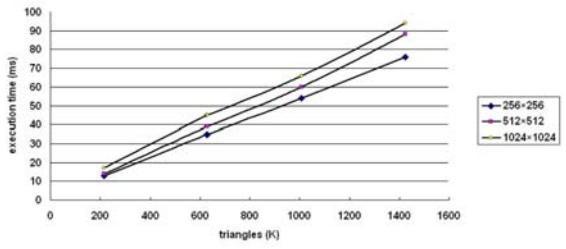

Figure 8. Performance evaluation with different texture resolutions

Fig. 7-(a) through 7-(d) are listed in the decreasing order of mesh complexity, e.g. 1423K triangles in Fig. 7-(a) and 214K triangles in Fig. 7-(d). In Fig. Fig. 8, the upper-right vertex of each curve corresponds to Fig. 7-(a) while the lower-left vertex corresponds to Fig. 7-(d), i.e. execution time is proportional to the mesh complexity. Collision detection performances are also evaluated by changing the texture resolution. Three curves in Fig. 8 show that the texture resolution does not have a great impact on the performance. Recall that the accuracy of the collision detection is governed by the texture/image precision. Therefore, in the proposed approach, highly accurate collision test can be achieved in real-time even with 1024×1024 texture.

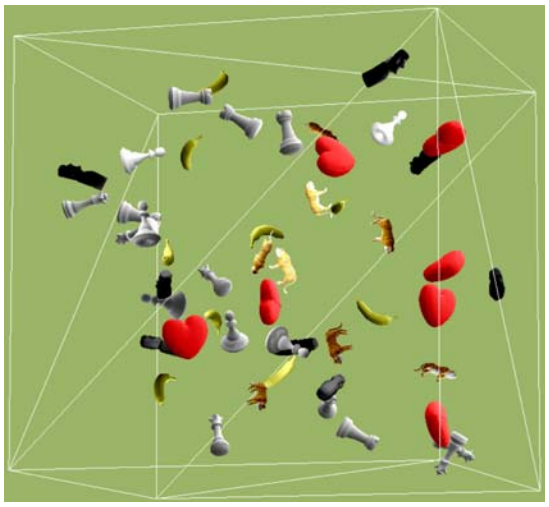

Figure 9. Multiple objects in a cube

Table 1. Performance evaluation for collision among multiple objects

No. objects	collision detection time per frame	No. collisions per min.	fps
50	3.5 ms	408	193
40	2.3 ms	328	230
30	1.3 ms	142	345

Fig. 9 shows a cube in which multiple objects are moving and colliding with each other. Each model consists of thousands of triangles, and an object has 3K polygons on average. Table 1 shows the performance evaluation measured by varying the number of objects. Unlike the collision test of Fig. 7, PCS is computed by CPU using bounding spheres, and the GPU collision detection algorithm is invoked per a PCS.

7. Conclusion

This paper presented an efficient image-space algorithm to real-time collision detection. In the current implementation, CPU computes only the potentially colliding set (PCS), and the entire collision test is run by GPU. The algorithm does rarely suffer from the readback problem. Further, the proposed algorithm does not necessarily render the entire surface of an object, and therefore the rendering cost has been reduced. The algorithm's efficiency is achieved by detecting collision of the entire scene, not of only a pair of objects. The algorithm maximally utilizes the state-of-the-art functionalities of GPU such as framebuffer objects and occlusion query. The experimental results show the feasibility of GPU-based collision detection and its performance gain in real-time applications such as 3D games.

참고문헌

[1] P. M. Hubbard. "Interactive collision detection." In Proc. of IEEE Symposium on Research Frontiers in Virtual Reality, pages 24-32, 1993.

[2] I. Palmer and R. Grimsdale. "Collision detection for animation using sphere-trees." Computer GraphicsForum, 14(2):105-116, 1995.

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

[4] G. Zachmann and W. Felger. "The boxtree: enabling realtime and exact collision detection of arbitrary polyhedra." In Proc. of Workshop on Simulation and Interaction in Virtual Environments, SIVE 95, pages 104-113, 1995.

[5] 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.

[6] J. T. Klosowski, M. Held, J. S. B. Mitchell, H. Sowizral, and K. Zikan. "Efficient collision detection using bounding volume hierarchies of k-DOPs." IEEE Transactions on Visualization and Computer Graphics, 4(1):21-36, 1998.

[7] T. He. "Fast collision detection using QuOSPO trees." In Symposium on Interactive 3D Graphics, pages 55-62, 1999.

[8] M. Teschner, B. Heidelberger, M. Mueller, D. Pomeranets, and M. Gross. "Optimized spatial hashing for collision detection of deformable objects." In Proc. of Vision, Modeling, Visualization, pages 47-54, 2003.

[9] J. Mezger, S. Kimmerle, and O. Etzmuss. "Hierachical techniques in collision detection for cloth animation." Journal of WSCG, 11(2):322-329, 2003.

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

[11] G. Baciu, S. K. Wong, and H. Sun. "RECODE: An image-based collision detection algorithm." Journal of Visualization and Computer Animation, 10(4):181-192, 1999.

[12] K. Myszkowski, O.G. Okunev, and T.L. Kunii. "Fast collision detection between computer solids using rasterizing graphics hardware." Visual Computer, 11:497-511, 1995.

[13] K.E. Hoff, A. Zaferakis, M. C. Lin, and D. Manocha. "Fast 3D geometric proximity queries between rigid and deformable models using graphics hardware acceleration." Technical report, Department of Computer Science, University of North Carolina, 2002.

[14] T. Vassilev, B. Spanlang, and Y. Chrysanthou. "Fast cloth animation on walking avatars." Computer Graphics Forum (Proc. of Eurographics01), 20(3):260-267, 2001.

[15] B. Heidelberger, M. Teschner, and M. Gross. "Realtime volumetric intersections of deforming objects." In Proc. of Vision, Modeling and Visualization, pages 461-468, 2003.

[16] B. Heidelberger, M. Teschner, and M. Gross. "Detection of collisions and self-collisions using imagespace techniques." In Proc. Computer Graphics, Visualization and Computer Vision WSCG"04, pages 145-152, 2004.

[17] N. K. Govindaraju, S. Redon, M.C. Lin, and D. Manocha. "CULLIDE: Interactive collision detection between complex models in large environments using graphics hardware." In Proc. of ACM SIGGRAPH/Eurographics Workshop on Graphics Hardware, pages 25-32, 2003.

[18] 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.

[19] D. Knott and D. Pai. "CinDeR: Collision and interference detection in real-time using graphics hardware." In Proc. of Graphics Interface "03, pages 73-80, 2003. ociety.

[20] A. Rege. "Occlusion - hp and nv extensions." In Game Developers Conference Presentation, 2002. http://developer.nvidia.com/attach/6715.

[21] S. Green. "The opengl framebuffer object extension." In Game Developers Conference Presentation, 2005. http://download.nvidia.com/developer/presentations/2005/GDC/OpenGL_Day/OpenGL_FrameBuffer_Object.pdf.

장 한 용

2005년 2월 고려대학교 컴퓨터학과 졸업(이학사). 2005년 3월 ~ 현재 고려대학교 컴퓨터학과 미디어랩 석사과정. 관심분야는 실시간 렌더링.

정 택 상

2005년 2월 고려대학교 컴퓨터학과 졸업(이학사). 2005년 3월 ~ 현재 고려대학교 컴퓨터학과 미디어랩 석사과정. 관심분야는 실시간 렌더링.

한 정 현

1996년 USC 컴퓨터학과 졸업(공학박사). 1996년 ~ 1997년 미국 상무성(NIST) Manufacturing Systems Integration Division 연구원. 2004년 3월 ~ 현재 고려대학교 컴퓨터학과 교수. 관심분야는 실시간 렌더링.