RECODE: An Image-Based Collision Detection Algorithm

George Baciu[*]
gbaciu@cs.ust.hk

Wingo Sai-Keung Wong
wingo@cs.ust.hk

Hanqiu Sun
hanqiu@cs.cuhk.hk

Abstract

Object interactions are ubiquitous in interactive computer graphics, 3D object motion simulations, virtual reality and robotics applications. Most collision detection algorithms are based on geometrical object–space *interference tests. Some algorithms have employed* an image–space approach *to the collision detection problem. In this article, we demonstrate an image–space collision detection process that allows substatial computational savings during the image–space interference test. This approach makes efficient use of the graphics rendering hardware for real–time complex object interactions.*

Keywords: image–based collision detection, object interference, motion simulation

1. Introduction

Autonomous object interactions and navigation in a 3D virtual environment often require fast object interference tests, efficient and accurate computation of collision points, as well as identifying the points of contact for a possible response action. Aiming for both generality and high performance, a new breed of collision detection algorithms are currently being developed. This is motivated by the fact that object–space collision detection methods are generally dependent on the geometrical structure of objects, exhibit a high complexity even for simple combinations of polyhedra, are limited to planar bounding surfaces, and are often costly to run, either due to preprocessing, or due to run–time searches for closest features. By comparison, image-based collision detection methods are much simpler to implement, exhibit a high degree of robustness with respect to object geometry, and have a great potential to exploit the hardware–assisted rendering features of the current graphics boards. While some work has been done in this area, [20, 19, 17, 1] much remains to be investigated to improve and refine these methods.

[*]Computer Science Department, The Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong. Support provided by RGC grant HKUST 598/94E.

The paper is organized as follows. In the following section 2, we outline the background ideas stressing the difference between image–space and object–space collision detection methods together with the motivation and the main contributions of this work. In section 3 we give a perspective of some of the most relevant and recent work on collision detection methods and classify them according to the image (subsection 3.2) and object based division (subsection 3.1). In section 4 we give the details of our image–based algorithm. In section 5 we describe the experiments we performed and the performance evaluation of our algorithm in the context of the fastest algorithms currently available. In section 6 we draw some conclusions and propose some directions for further research work in the area of image–based collision detection.

2. Background

In practice, any computational processes for object interactions in a 3D environment takes us further away from the hardware rendering rates. As a fundamental task, the collision detection process should not add much overhead to the actual scene rendering itself. Our main goal is to make optimal use of the hardware rendering capabilities of conventional graphics workstations in order to achieve efficiency and flexibility within the collision detection process. We propose an effective use of the image rendering buffers during the rendering process to perform collision tests and compute collision points after reducing the possible regions of interference by object–space culling based on a combination of both aligned and oriented bounding boxes. Our approach is based on *the dimension reduction* of the 3D space leading to a simple image–space test on **M**inimal **O**verlapping **R**egions (**MOR**) which are projected by candidate object pairs onto the viewing plane. The candidate object pairs are the pairs of objects whose object–space bounding boxes overlap. Our experiments indicate that the overhead added to the hardware rendering process by the collision detection and the determination of the contact regions is relatively small on the average. Further tests show a consistent performance improvement with respect to some popular object–space collision detection algorithms. The algorithm exhibits the same degree of robustness as finding all the closest features and contact points as I-COLLIDE, while outperforming in some cases RAPID and Q–COLLIDE.

We make no assumptions about the motion of objects in the scene and there is no differentiation between the states of a collision event. For example, there is no need to differentiate between objects just before a collision, on contact, or penetrating each other. Hence, there is no need to construct special pseudo Voronoi regions and identify special cases whenever a collision event is detected due to surface penetration. Furthermore, this approach benefits from a variable image–space resolution. This allows to dynamically trade off accuracy for computational time.

The only assumption that we are currently working with is that the basic object primitives, or blocks are convex. This assumption imposes absolutely no limitations on hierarchical structures, articulated bodies, or non-convex compositions of multiple convex blocks.

3. Previous Work

Solutions to the collision detection problem can be classified according to the scene sampling in the time domain. This can be considered as either *temporally global* or *temporally local*. Within each temporal classification, the spatial sampling for collision detection can be classified into: (1) *object–space interference test*, and (2) *image–space interference test*.

3.1. Object–Space Collision Detection

In the *object–space interference test* research spans over a variety of techniques from topological analysis to computational geometry. Some precomputation is generally necessary in order to identify an interference structure, either locally or globally. Data structures, such as BSP trees [22], octrees [16, 23], voxel grid [8], or hierarchical oriented bounding boxes [11] are built for the spatial localization of regions of potential interference.

Studies in the *object–space interference* tests range from temporal and spatial localizations [4, 5, 9, 16, 12, 15, 21, 23, 24, 8, 2, 7, 13, 18, 10, 11] to global space–time analysis [3]. The techniques that tend towards a global temporal and spatial analysis are generally more robust. However, these techniques require complete knowledge of the space–time trajectories. Therefore, they are memory bound and computationally prohibitive. Furthermore, in an interactive dynamic scene with autonomous moving objects the full motion trajectories of objects are not generally known in advance. A similar situation arises in an interactive navigation through a virtual environment.

In order to achieve interactive rendering rates, the local methods, both in time and space, are more suitable. In the last few years, some of the most significant contributions in this area, both theoretical and practical, have been packaged in a series of fast, general purpose collision detection software systems such as RAPID [18], I-COLLIDE [7], Q-COLLIDE [6], V-COLLIDE [14]. Most of this work is based on the seminal work of Lin and Canny [15] who proposed an efficient computation of the closest feature set between object pairs.

The *separating axis theorem* [10] has made the elimination of object pairs in the collision detection tests fast and practical. Now, it is possible to efficiently bound objects within more tightly fit **O**riented **B**ounded **B**oxes (**OBB**) than **A**ligned **B**ounded **B**oxes (**ABB**) [11]. This dramatically improves the performance for collision tests.

Exploiting the temporal and geometric coherence has reduced the all pairs testing of arbitrary polyhedral objects from $O(N^2)$ to $O(N^2 + S)$ where N is the number of objects in the scene and S is the number of pairwise overlaps [7] when the coherence is preserved. This result is extremely useful when the relative orientation between objects does not change and collisions may occur due to translational motion alone. However, in many applications objects change their relative orientation triggering a recomputation of the closest feature sets.

The *separating vector technique* has been shown to improve the bounding box overlap tests [6]. In this work, the authors also reported that the closest feature determination may add computational overhead when objects rotate and the feature sets need to be dynamically changed all the time.

The intrinsic complexity of object–space collision detection methods arises from the following:

1. Increased geometric complexity will always result in a significant computational overhead in collision detection algorithms, regardless of their efficiency.

2. Special cases must be accounted for, e.g. pseudo Voronoi regions for penetrating surfaces.

3. Complex data structures must be maintained at all times, e.g. Voronoi regions, sorted lists, etc.

4. The entire collision detection process takes place only in the object–space and much of the information associated with rendering the scene, stored in the frame buffer, the Z–buffer, and other available image buffers is not used.

These reasons prompted us to search deeper into the rendering process for simpler, more efficient, and more elegant collision detection methods that accommodate the higher complexity of virtual environments. This directed us to look into *image–space collision detection methods*.

3.2. Image Space Collision Detection

The area of image–space collision detection has recently opened new avenues for improving the performance bounds [17]. The image–space interference test is based on projecting the object geometry onto the image plane and performing the analysis in a dimensionally reduced space with the depth map maintained in an image buffer, such as the Z–buffer.

Little has been reported on the image–space interference techniques. The pioneering work of Shinya and Forque [20] and Rossignac [19] lead to a simpler and more efficient algorithm for interference detection of interacting complex solid objects. The premise of this work is based on an enhancement in the graphics hardware support. This requires checking the change in the stencil buffer state from one frame to the next at the hardware level. If this hardware assisted test was available, the algorithm proposed by Myszkowski et al [17] would be a very practical solution to the interference problem between complex solid objects. However, such hardware enhancements are not currently available.

As our previous experiments show [1], without the hardware–assisted stencil buffer check, the collision detection becomes prohibitively slow due to the very large number of redundant memory accesses required to simulate the change in the stencil buffer state. However, we have found that it is still practical to use the conventional stencil buffer and z–buffer for collision detection provided that we reduce the region for testing the stencil buffer state [1]. We have improved upon the work done in [1] by (1) further reducing the time for the collision test, (2) introducing hierarchical support, and (3) handling *simple non-convex* shapes.

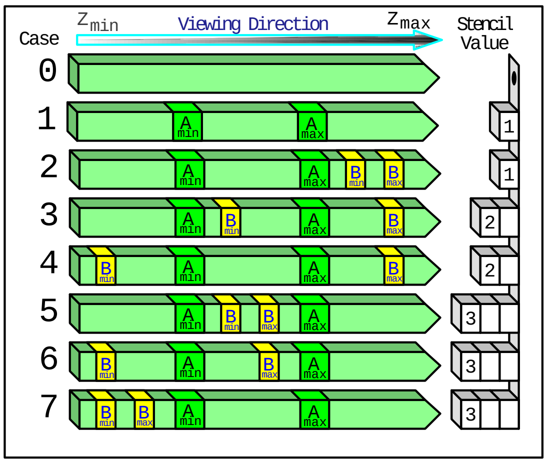


Figure 1. Possible Z-interval cases and the corresponding stencil buffer values

By fully investigating the relative performance merits of existing collision detection systems, we have found that the image–based collision detection approach is not only practical from the point of view of conventional rendering hardware, but it is also preferred in an enhanced hardware architecture. This is because, when a collision is detected, the computation of the region of interference still requires a full inspection of the stencil buffer contents. In this work, we focus on image–based interference region computations and, through practical examples, show that it is possible to perform this task for complex hierarchical objects close to frame rendering rates.

4. The RECODE algorithm

Our work has been motivated by the results reported by Myszkowski et al [17]. We start with the assumption that the basic primtive blocks of a hierarchical structure are convex. This ensures that a ray passing through any pixel (x, y) may intersect a basic block at at most two points.

We have implemented this algorithm in a general purpose collision detection system called **RECODE**, **RE***ndered* **CO***llision* **DE***tection* .

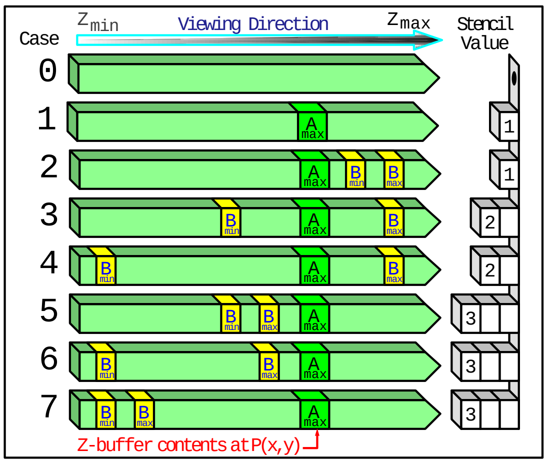


Figure 2. Testing B against A_{max}

In RECODE we reduce the three–dimensional collision detection problem to a one–dimensional interval test along the Z-axis based on the principle that a potential interference at the frame buffer resolution exists if

$$R_{xy}(\mathbf{A}) \cap R_{xy}(\mathbf{B}) \neq \phi \quad \text{and} \quad I_z(\mathbf{A}) \cap I_z(\mathbf{B}) \neq \phi \quad (1)$$

where $R_{xy}(\mathbf{X})$ is the xy–region occupied by the orthogonal projection of object **X**, and $I_z(\mathbf{X})$ is the z–interval occupied by object **X**.

In order to illustrate the basic mechanism of RECODE, we establish the following definitions: **MOR(A,B)** is *the* *minimum overlapping region* of the projection of the volume occupied by the overlapping bounding boxes of objects **A** and **B**; **VFA** is *the frame buffer viewport* allocated for rendering object **A**; **VZA** is the corresponding *Z–buffer viewport* where a depth map (either Z_{max} or Z_{min}) of object **A** is stored; **VSA** is the corresponding mask for *the stencil buffer viewport* of object **A**; **VSC** is the corresponding *counter* in *the stencil buffer viewport* of object **A**.

```
proc RECODE(A,B) ≡
  RenderingArea ← MOR(A,B)
  RenderSetZ (A , ≥ )
  RenderTestZ(B , ≤ )
  if zIntervalOverlap(A,B) = TRUE
    return collision = TRUE
  fi
  if secondRendering = TRUE
    RenderSetZ (B , ≥ )
    RenderTestZ(A , ≤ )
    if zIntervalOverlap(A,B) = TRUE
      return collision = TRUE
    fi
  fi
.
```

Figure 3. The logical flow of RECODE

The convention for testing the Z-values of two points $\mathbf{P}_1$ and $\mathbf{P}_2$ is as follows: if $Z_1 < Z_2$ is true, then $\mathbf{P}_1$ is closer to the eyepoint than $\mathbf{P}_2$ on an orthogonal projector through the pixel (x, y). The algorithm is based on counting the number of faces of two objects overlapping pixel (x, y) over the interval $[Z_{min}, Z_{max}(\mathbf{A})]$. This implies a single Z–buffer test:

$$Z(\mathbf{B}) \quad \leq \quad Z_{max}(\mathbf{A}) \tag{2}$$

where, in a normalized orthogonal viewing volume, $Z_{min}(\mathbf{X})$ is the minimum Z–value of object **X**, $Z_{max}(\mathbf{X})$ is the maximum Z–value of object **X**, $Z(\mathbf{X})$ is the Z–value of the currently rendered polygonal face of object **X**.

As shown in Fig. 3, first we determine MOR(A,B) and set the rendering viewport, VFA, to correspond only to the region covered by the MOR(A,B). In the Z–buffer, we store the maximum Z-values of object **A**, by turning off writing into the frame buffer and rendering **A** over VZA. This is done by the function $RenderSetZ(\mathbf{A}, \geq)$ in the algorithm shown in Fig. 3. The function $RenderSetZ$ generates the depth map of all the points on the backfaces of **A** by setting $Z(x, y)$ if $Z(A) \geq Z$. This map is then used as reference in the first rendering pass in order to establish a class of the possible cases of **B** overlapping **A** along the Z-axis.

For convex blocks, RECODE tests against eight independent cases, as illustrated in Fig. 1. With respect to the

```
proc RECODE(A,B) ≡
  Find $R_{xy}$(A) and $R_{xy}$(B)
  using ABB(A) and ABB(B);
  Find MOR(A,B);
  Disable writing into the frame buffer ;
  for all pixels do
      set VZA=0; VSA=0; VSC=0;
  od
  if Exist(VZA and VSA)
    Use VZA and VSA
else
    Set Z–comparison for finding $Z_{max}$(A);
    Enable the depth mask
    Set pointers to VZA and VSA
    RenderSetZ(A,≥) over $R_{xy}$(A)
    Set VSA=1 for each pixel ∈ $R_{xy}$(A)
  fi
  Disable depth mask
  RenderTestZ(B, ≤) over MOR(A,B);
  SecondRendering = FALSE;
  for all pixels ∈ MOR(A,B) ∩ VSA=1 do
      if (VSC==1) return COLLISION;
      if (VSC==2) SecondRendering = TRUE;
  od
  if (SecondRendering == FALSE)
    return NO_COLLISION;
  fi
  for all pixels do
      set VZA=0; VSA=0; VSC=0;
  od
  if Exist(VZA and VSA)
    Use VZA and VSA
else
    Enable depth mask
    Set VZA and VSA
    RenderSetZ(B, ≥) over $R_{xy}$(B)
    Set VSA=1 for each pixel ∈ $R_{xy}$(B)
  fi
  Disable the depth mask
  RenderTestZ(A, ≤) over MOR(A,B);
  for all pixels ∈ MOR(A,B) ∩ VSA=1 do
      if (VSC==1) return COLLISION;
  od
  return NO_COLLISION;
.
```

Figure 4. RECODE: An image–based collision detection algorithm

values of $A_{max} \equiv Z_{max}(\mathbf{A})$ found in the Z–buffer over VZA, the function $RenderTestZ(\mathbf{B}, \leq)$ uses test (2) in order to generate any one of the eight possible cases, given in Fig. 2.

The stencil buffer mask VSA will contain one of the values in the set $\{\mathbf{0}, \mathbf{1}, \mathbf{2}, \mathbf{3}\}$ depending on the relative depth of the faces of objects **A** and **B** in the interval $[Z_{min}, Z_{max}]$ at (x, y). The stencil buffer over VSA is initialized to zero. If there are no objects at pixel (x, y) then the corresponding stencil value remains zero. If at least one object overlaps pixel (x, y) then the corresponding stencil value is incremented to one. In case (1), no face of **B** overlaps pixel (x, y) where object **A** has a value of A_{max}. In case (2) the range on the Z-axis covered by object **B** at (x, y), $[B_{min}, B_{max}]$ does not overlap the interval $[Z_{min}, A_{max}]$. In both cases (1) and (2) there is no face of **B** that *crosses over* into the interval $[Z_{min}, A_{max}]$. Hence, the stencil value at (x, y) is one. In cases (3) and (4) the interval occupied by **B** overlaps $[Z_{min}, A_{max}]$. That is, only one face of **B** *crosses over* A_{max}. Hence, the stencil value at (x, y) is incremented to two. In cases (5–7), the range occupied by object **B** lies in the interval $[Z_{min}, A_{max}]$. That is, two faces of **B** *cross over* A_{max}. Notice that cases (4) and (5) correspond to either object **B** being larger than **A**, or object **A** being larger than **B**, respectively. The details of this algorithm are given in Fig. 4.

In the new algorithm, we minimize the stencil buffer area, that is required in order to compute the interference regions and introduce a hierarchical decomposition of larger, possibly non—convex structures.

For computing the contact region, we must ensure that the depth buffer contains the values of the boundaries of object **B**. In the original algorithm [17], the depth values of the boundaries of object **A** are returned. This happens when the second rendering is required. However, for the contact points we must have the depth values of the upper boundary of **B** and lower boundary of **B**.

5. Results

It is well known that comparing collision detection algorithms in general can be a very difficult task. As we were beginning to look into other collision detection algorithms, it has become clear that a thorough experimentation process has to be established first. A search through the literature has not revealed any comprehensive benchmarks that can be used to effectively compare different collision detection algorithms. Since we are not currently aware of a complete set of benchmarks, nor of rigorous procedures for testing general purpose collision detection algorithms, we devised some simple measures of performance based solely on time, specifically CPU time. These are easy to implement and include into a main driver that coordinates the selection of the specific collision detection algorithm. In order to take accurate measures, we implemented a benchmarking environment that considers the exact same initial conditions, object geometry, and motion paths, both rotation and translation, for all algorithms.

The most difficult task is to keep the number of collisions the same, or at least within a very narrow range from one algorithm to another. In other words, we must ensure that the motion states do not change significantly from one algorithm to another. For this reason, performing a collision response of any kind may disrupt the test environment. Since the motion states of all objects should be maintained the same in all tests, we did not include any collision response action, but merely stop the simulation in the case of a controlled experiment, or let the objects penetrate each other on an undisturbed motion path for randomly placed objects in space. Of course, this penetration should be minimal with respect to the object sizes. Therefore, we usually stopped the simulations after 100 or at most 500 frames.

We have included a microsecond resolution timer and measured the total time that each algorithm took in each frame and the time that each algorithm spent in performing only collision detection alone. It is important to observe the time patterns for both the frame rendering and collision computations, mainly to understand if the deviations from a uniform rate are due to rendering alone or collision computations. All the time measurements were done on an SGI Octane workstation powered by a 195MHz R10000 processor with 192MB of main memory.

5.1. Qualitative analysis of RECODE performance

Figure 5 illustrates three different experiments on the following convex shapes: (a) and (b) flat ellipsoids, (c) and (d) truncated cones with spherical caps, and (e) and (f) cones. The bar charts in Figure 9 show the relative performance of RECODE algorithm as the complexity of the scene, measured in the number of triangular patches, increases from 3600 to 1,152,000 triangles. The red portions of the bar charts (1) and (2) show the collision time overhead. This is the time taken by all the collision computations alone on top of the rendering time (the lower blue portions of the bars). These charts demonstrate two features of the RECODE algorithm: (1) the collision detection overhead is bounded by the rendering rates, both overall and on average; (2) the collision detection pattern remains very smooth as the complexity of the scene increases. This is shown by the standard deviation charts (3) and (4). This means that during the interactive animation sequence there are no prolonged interruptions in the display rate, as the RECODE algorithm maintains the same standard deviation from the average collision time as the rendering time itself. This is a very important aspect in collision detection algorithms, since it determines the overall quality of the animation when collision detection is turned on.

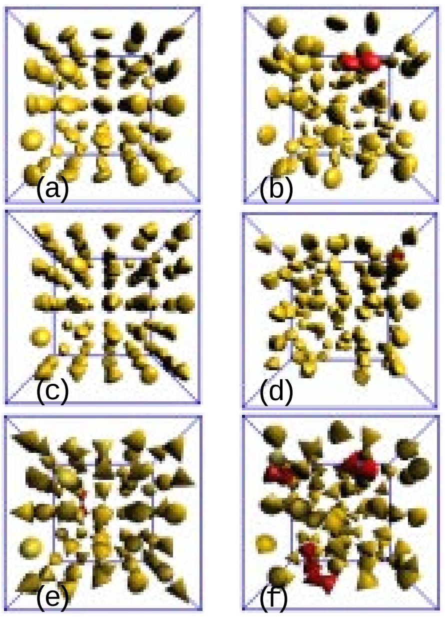


Figure 5. 100 objects in random motion

5.2. General comparison against other algorithms

We performed experiments with I–COLLIDE, RAPID (with all feature pairs turned on), RAPID1 (with only one feature pair turned on), RECODE, and Q–COLLIDE. The tests are based on measuring the frame time and the collision time for 100 cones randomly placed in space, but maintaining the same motion paths for all algorithms. As we vary the number of triangles, we observe that Q–COLLIDE performs the best, both on average and overall timings. while RECODE is the next best performer, followed in order by I–COLLIDE, RAPID, and RAPID1. We observe at this point, that in all our experiments with random object placement, Q–COLLIDE shows the best timings both overall and on average. However, Q–COLLIDE as well as RAPID tend to be less robust than I–COLLIDE and RECODE, and in some cases fail completely to detect many collisions, for example in the case of cones or pointed structures.

In this respect we have noticed some interesting behaviour. Specifically, in some simple controlled cases the high performance collision detection algorithms, such as RAPID and Q–COLLIDE, tend to exhibit very large standard deviations and often fail to detect collisions. In order to observe these effects more closely, we developed a simple controlled test called the *Comb Experiment*.

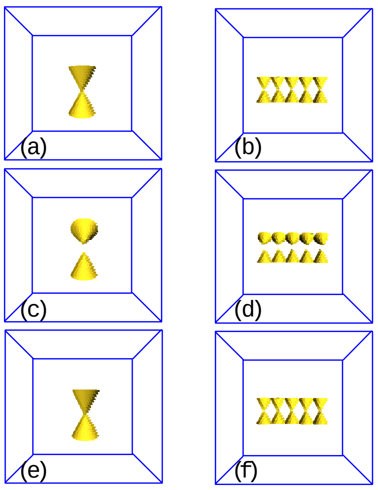


Figure 6. Comb Experiment: controlled rotations of aligned combs

5.3. The Comb Experiment

The comb experiment consists of two rows of cones with the same geometric parameters and aligned at the base, Fig. 6. We call one row of cones *a comb*. Various types of controlled motion patterns may be produced by moving one row with respect to the other. But the most desired property of the comb is that we can control the points of collision very precisely. For example, since the tip of each cone will sit on the bounding box of the cone we can observe the collision detections without the bounding boxes of a pair of cones ever intersecting until the collision event. We can also generate a motion pattern that produces no collisions but have the bounding boxes overlap at all times, causing the algorithms to compute the closest features all the time. While the translation of the combs with respect to each other keeps the closest feature set constant, their rotation will cycle through all the possible closest features. Thus, the combs can exercise every feature of the collision detection algorithm independently in a controlled fashion. Next, we describe the two most significant comb experiments.

The two combs can be set up to (a) align the cone tips through their axes, or (b) offset the cone tips of the two combs. The cases in which the cones produce interesting patterns in the performance of collision detection algorithms are: (1) pure translation, and (2) pure rotation, of

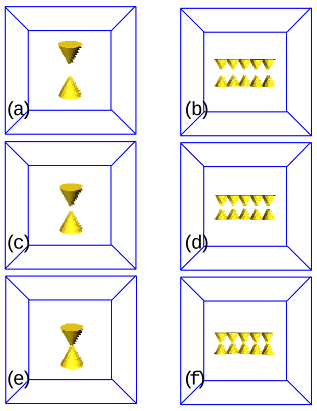


Figure 7. Comb Experiment: controlled translation of aligned combs

one comb with respect to the other. We set up two combs with only five cones each. The most interesting results we obtained are for cases (a.1) and (a.2), Fig. 6 and 7.

In the comb rotation case, Fig. 6, we observe the performance of I–COLLIDE, RAPID, RECODE, and Q–COLLIDE. RECODE, the lowest red curve, maintains the best performance in all cases above 10,000 triangles, both in the overall frame and collision times and on the average frame and collision times, charts (5) and (6). This is followed in order by I–COLLIDE, Q–COLLIDE, and RAPID. The standard deviation (bottom two charts) of RECODE is also the lowest among all algorithms as the number of triangles increases.

In the comb translation case, Fig. 7, interestingly, Q–COLLIDE cannot find any collision, even though there are exactly five points of collision between the two combs (due to the five cones in each comb). Therefore, we could not generate a performance curve for Q–COLLIDE. The curves we observed were for I–COLLIDE, RAPID, and RECODE. Again, we noticed that above 10,000 triangles, RECODE maintains the lowest collision detection time, both overall and on average followed by I–COLLIDE and RAPID. The standard deviation is also the lowest for RECODE, followed by I–COLLIDE and then RAPID.

5.4. Hand Grasping

For a more typical example, we generated the hand grasping sequence shown in Fig. 8. Table 1 shows the RECODE performance data set for the hand grasping the ball sequence. In this table, NTR is the total number of triangles

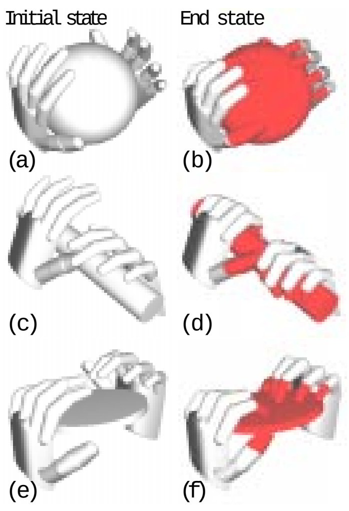


Figure 8. Hand grasping actions: initial and final states

NTR	TNF	TTF	ATF	DTF	TTC	ATC	DTC
		sec	ms	ms	sec	ms	ms
1860	75	3.78	50.1	10.9	1.3	17.0	7.3
2592	72	4.04	56.7	10.2	1.4	19.9	7.4
3744	72	4.90	67.8	11.3	1.6	22.0	8.1
4896	89	6.82	76.9	13.8	2.0	22.8	9.7
7200	72	6.91	78.6	14.2	1.9	27.0	9.9
9792	71	8.52	118.6	15.3	2.3	31.8	11.7
10944	72	9.18	128	15.6	2.4	34.0	12.7
14400	72	11.3	156	17.7	2.7	37.5	14.4
18720	72	13.7	190	19.2	3.0	41.1	15.6
21600	71	15.2	215	21.5	3.3	47.0	18.5
28512	71	19.3	272	26.0	3.9	55.3	22.0
33632	71	22.6	318	31.2	4.5	63.4	24.9
42768	68	26.8	393	34.0	5.2	76.5	30.6
63248	68	38.1	550	44.6	6.8	99.7	40.6
76032	69	45.8	666	53.4	8.2	118.6	49.9

Table 1. Collision detection performance for two hands grasping a ball

for the scene, TNF is the total number of frames that the sequence took until all fingers reached a contact point with the object grasped, TTF is the total time (including rendering and collision detection) in seconds for all frames, ATF is the average time per frame in milliseconds, DTF is the standard deviation of the frame time in milliseconds, TTC is the total collision time in seconds for the entire sequence, ATC is the average collision time in milliseconds, DTC is the standard deviation of collision time in milliseconds.

As we increase the number of triangular patches in the model, we note that the collision time maintains a fairly low overhead on top of rendering time alone. For example, from 1860 to 76,032 triangles, the average rendering time goes from 50.1 ms to 666 ms, but the average collision time alone stays well bounded between 17.0 ms and 118.6 ms. The standard deviations for collision detection alone are maintained under the standard deviation for the frame times. This again indicates a smooth motion sequence.

6. Conclusions

The results above show that the collision detection in RECODE, is bounded by the rendering rates. It is obvious that the total collision time cannot be bounded in general, as rendering itself is not bounded with the increase in the number of objects. However, much of the computational time for complex collision patterns can be permeated through the rendering pipeline. In this context, we can introduce multiple parameters to control either the level of refinement and accuracy of the collision detection or the animation time. This flexibility is greater than that given by object–space collision detection alone, since it allows an additional degree of freedom on the discretization of space by simply adjusting the viewport for rendering the candidate objects for collisions.

In conclusion, we have shown that it is possible to perform collision detection of complex objects at frame rendering rates comparable to the rates of the best collision detection algorithms available. We have also observed that among the collision detection algorithms we have tested, I–COLLIDE and RECODE are the most robust, reporting an equivalent number of collisions for each test. However, RECODE is much simpler to implement and can accommodate a larger class of objects, including surface primitives that can be rendered directly in hardware.

This promotes a relatively new direction of research in *optimal image–based collision detection algorithms*, based on considering various strategies to compute and minimize the MOR between two objects. One such possibility for convex objects is to use the separating plane itself as the image projection plane as well as in reducing the object rendering time by considering the rendering of the closest features only, rather than the entire object.

References

[1] G. Baciu and S.-K. W. Wong. Rendering in Object Interference Detection on Conventional Graphics Workstations. In *Proceedings of the Pacific Graphics'97*, pages 51–58. IEEE, Oct. 1997.

[2] S. Bandi and D. Thalmann. An Adaptive Spatial Subdivision of the Object Space for Fast Collision of Animated Rigid Bodies. In *Eurographics 95*, pages 259–270, Maastricht, August 1995.

[3] S. Cameron. Collision detection by four–dimensional intersection testing. *IEEE Trans. on Robotics and Automation*, 6(3):291–301, June 1986.

[4] J. F. Canny. Collision detection for moving polyhedra. *IEEE Trans. PAMI*, PAMI(8):200–209, March 1986.

[5] B. Chazelle and D. P. Dobkin. Intersections of convex objects in two and three dimensions. *Journal of the ACM*, 34:1–27, 1987.

[6] K. Chung and W. Wang. Quick Collision Detection of Polytopes in Virtual Environments. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, pages 125–132. ACM, July 1996.

[7] J. Cohen, M. Lin, D. Manocha, and M. Ponamgi. I–COLLIDE: An interactive and exact collision detection system for large–scale environments. In *Proc. of ACM Interactive 3D Graphics Conference*, pages 189–196. ACM, 1995.

[8] A. Garcia-Alonso, N. Serrano, and J. Flaquer. Solving the collision detection problem. *IEEE Computer Graphics and Applications*, 13(3):36–43, 1994.

[9] E. G. Gilbert, D. W. Johnson, and S. S. Keerthi. A fast procedure for computing the distance between objects in three–dimensional space. *Journal of Robotics and Automation*, RA(4):193–203, 1988.

[10] S. Gottschalk. Separating axis theorem. Technical Report TR96-024, Department of Computer Science, UNC Chapel Hill, 1996.

[11] S. Gottschalk, M. Lin, and D. Manocha. OBBTree: A Hierarchical Structure for Rapid Interference Detection. In *Computer Graphics (SIGGRAPH'96)*, pages 171–180. ACM, Aug 1996.

[12] B. V. Herzen, A. H. Barr, and H. R. Zats. Geometric collision for time–dependent parametric surfaces. *Computer Graphics (SIGGRAPH'90)*, 24(4):39–48, 1990.

[13] P. M. Hubbard. Collision detection for interactive graphics applications. *IEEE Transactions of Visualization and Computer Graphics*, 1(3):218–228, 1995.

[14] T. Hudson, M. Lin, J. Cohen, S. Gottschalk, and D. Manocha. V-COLLIDE: Accelerated Collision Detection for VRML. In *Proc. of VRML'97*. ACM, 1997.

[15] M. C. Lin and J. F. Canny. Efficient Collision Detection for Animation. In *Third Eurographics Workshop on Animation and Simulation*, Cambridge, September 1992.

[16] M. Moore and J. P. Wilhelms. Collision Detection and Response for Computer Animation. In *Computer Graphics (SIGGRAPH'88)*, volume 22(4), pages 289–298. ACM, 1988.

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

[18] M. Ponamgi, D. Manocha, and M. C. Lin. Incremental algorithms for collision detection between general solid models. In *Proceedings of ACM/SIGGRAPH Symposium on Solid Modeling*, pages 293–304. ACM, 1995.

[19] J. Rossignac, A. Megahed, and B. O. Schneider. Interactive inspection of solids: cross–section and interfences. *Computer Graphics (SIGGRAPH'92)*, 26(2):353–360, July 1992.

[20] M. Shinya and M. Forgue. Interference detection through rasterization. *Journal of Visualization and Computer Animation*, 2:131–134, 1991.

[21] J. Snyder, A. R. Woodbury, K. Fleischer, B. Currin, and A. H. Barr. Interval methods for multi–point collisions between time dependent curved surfaces. In *Computer Graphics (SIGGRAPH'93)*, pages 321–334. ACM, Aug 1993.

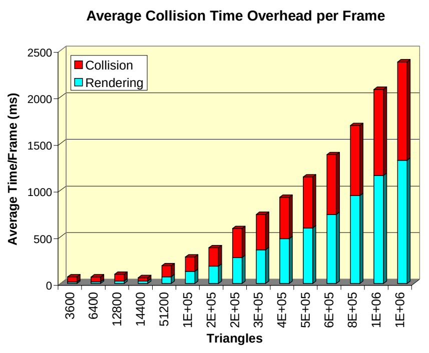
[22] W. C. Thibault and B. F. Naylor. Set operations on polyhedra using binary space partitioning trees. *Computer Graphics (SIGGRAPH'87)*, 21(4):153–162, July 1987.

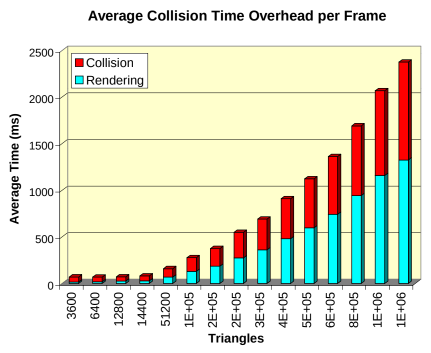
[23] Y. Yang and N. Thalmann. An improved algorithm for collision detection in cloth animation with human body. In *First Pacific Conference on Computer Graphics and Applications*, pages 237–251, 1993.

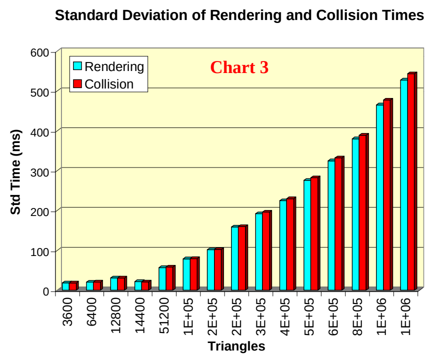
[24] M. J. Zyda, D. R. Pratt, D. Osborne, and J. G. Monahan. NPSNET: Real–time collision detection and response. *The Journal of Visualization and Computer Animation*, 4:13–24, 1993.

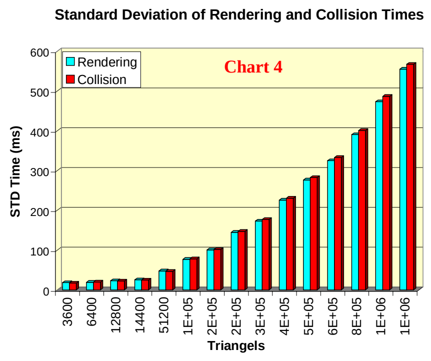
Ellipsoids in Motion

TCones in Motion

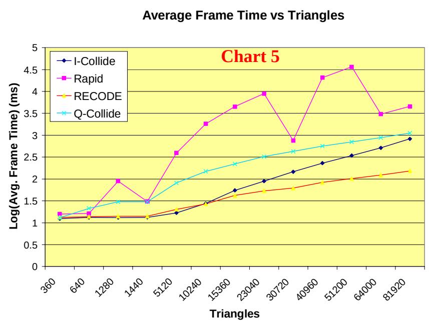


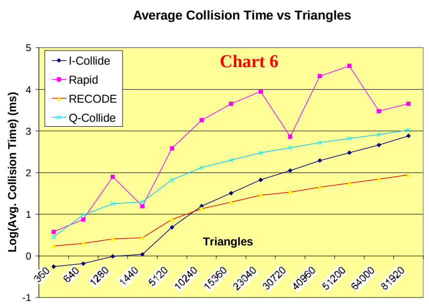


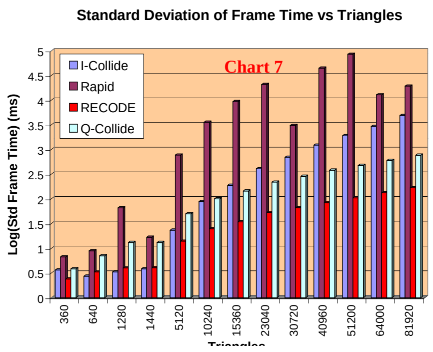




The Comb Experiment: aligned rotations







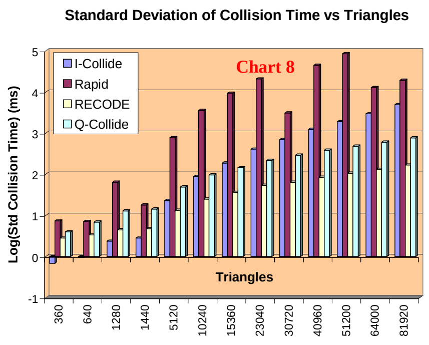


Figure 9. Collision statistics for random objects and controlled experiments