Fast Collision Detection Between Complex Solids Using Rasterizing Graphics Hardware

Karol Myszkowski Oleg G. Okunev Tosiyasu L. Kunii
Department of Computer Software, The University of Aizu
Aizu-Wakamatsu, 965-80 Japan
E-mail: k-myszk, o-okunev, kunii@u-aizu.ac.jp

Abstract

We present an efficient method for collision detection between complex solid objects. The method features stable processing time and low sensitivity to the complexity of contact between objects. The algorithm handles both concave and convex objects, however, the top performance is achieved when at least one object is convex in the proximity of the collision zone (our techniques check the required convexity property as a by-product of calculations). The method exhibits real-time performance when calculations are supported by standard functionality of graphics hardware available on high-end workstations.

Key words: computer animation, virtual environment, collision detection.

1 Introduction

Collision detection is an important component of computer systems performing realistic animation, robot motion planning and simulation, virtual reality. The collision detection between moving objects is a computationally intensive task dominated by geometrical calculations. In practical applications the object models are composed of multiple elements, e.g., polygonal faces, or patches defined by parametric or implicit functions. The naive collision detection algorithm has complexity $O(n^2)$ for the number of elements n. When the contact between colliding objects is simple (e.g., isolated point), the majority of elements can be discarded using inexpensive tests like the interference of bounding volumes. In such a case only a fraction of all elements located in the proximity of the collision zone requires the more elaborate processing. However, in many applications, complex contacts involving multiple

faces or patches are possible, and trivial discarding of elements does not solve the problem.

The motivation for this work is application of collision detection to computer simulation of the human jaws articulation. The model of jaws derived from scanning of the teeth is complex (over 100,000 polygons). The dentist interactively manipulates the mandibular position inspecting jaws' occlusion and determining the regions of the contact between teeth.

The goal of this work is to develop a general purpose collision detection algorithm that satisfies the following requirements:

- Ability to handle complex colliding objects. The top performance of the algorithm is expected for a few extremely complex objects rather than for many simple objects.
- Negligible penalty for the complexity of contacts.
- Interactive time rates for the interference checking.
- Similar performance for single interference tests and equivalent tests done in a sequence of tests. This means in particular that the strength of the algorithm cannot rely on any form of space-time coherence with the interference tests performed previously.
- Insensitivity of performance to interactive changes of positions of the object. The algorithm does not exploit *a priori* given paths of motion of the objects.
- The algorithm should run on standard workstations equipped with the graphics hardware.

The requirements imposed on the complexity of objects and latency of response preclude the use of general purpose computers. A natural choice is graphics workstations, granting high polygon throughput of hardware assisted rendering. A reformulation of the collision detection algorithm is needed for efficient use of the computational power of graphics hardware. The distance or intersection checking between objects used by traditional algorithms should be replaced by depth calculations, which are directly supported by graphics accelerators. This strategy seems to be justified by rapid progress and development of rendering techniques implemented by means of specialized hardware. At the same time, it is difficult to expect a hardware support for the aforementioned traditional approaches. The algorithm presented here refers to the technologies becoming an industrial standard, such as z-buffer, color alpha-blending, pixel masks (called also *stencil buffer*; Neider et al. 1993), which are available on high-end graphics workstations. We postulate also some simple extensions of the graphics libraries (granting

the access to the hardware resources), which could even further improve the performance of the collision detection.

The next section discusses the existing methods of the collision detection. Sections 3-7 describe our hardware assisted techniques of the collision detection handling both convex and concave objects. Experimental results showing the performance and accuracy of the algorithm are presented in Section 8. Finally, Section 9 contains our conclusions and suggests some areas for future research.

2 Previous work

The conservative collision determination requires interference test between the first object and the volume swept by the second object moving relatively to the first object within time interval δt. However, this approach is computationally expensive, and the general solution for complex objects and motion paths is difficult to find (Menon et al. 1994). In order to avoid these problems, discrete sampling of the objects' motion as a function of time is usually performed, and collisions are tested for frozen positions of objects only. This is also the case when discrete sets of positions are experimentally measured in a real environment or are registered for objects interactively (Kunii et al. 1994). The discrete approach produces acceptable results for many practical applications, however, some intersections can be missed, especially for the objects exhibiting spike-like surfaces (von Herzen et al. 1990). The sampling density of the object position can be adaptively adjusted taking into account the shape of the object, its velocity, curvature of the motion path and position of the object with respect to other objects.

The intersection test between non-polygonal surfaces (Hanna et al. 1983; Phillips and Odell 1984; Aziz et al. 1990; von Herzen et al. 1990) requires substantial computational effort, which makes this approach impractical for realtime applications. The majority of the existing collision detection algorithms assumes polygonal representation of surfaces (Moore and Wilhelms 1988; Hahn 1988; Yang and Thalmann 1993; Garcia-Alonso et al. 1994). The tessellation of the surface into polygons can be a source of collision determination errors traded off by the larger amount of polygons and longer processing times (Boyse 1979). The main strategy to speedup the interference test is to exclude as many non-colliding polygons as possible using inexpensive tests. Usually a hierarchical structure is imposed on objects and corresponding polygons. On the top level, the collision interest matrix (Garcia-Alonso et al. 1994) can be built, which establishes for every pair of objects whether the interference test between them is necessary. For the pair of objects that can collide, the bounding volumes embedded into hierarchical data structures, such as octree (Moore and Wilhelms 1988; Yang and Thalmann 1993), voxel grid (Garcia-Alonso et al. 1994), and bounding volumes hierarchy (Hahn

1988) are used to exclude the polygons located in the space regions occupied by one object only. The response time for the collision detection algorithms based on the bounding volumes is very unstable. When complex contacts between objects are possible, the trivial polygons rejection fails completely, and the expensive intersection tests between polygons must be performed.

In robotics applications, the distances between the nearest points (edges, faces) in two convex polyhedra are often alternately examined (Cameron and Culley 1986; Culley and Kempf 1986). However, when the objects move, the nearest points should be updated. For complex objects this is a time consuming task, and concave objects have to be decomposed into convex pieces. The top performance of this algorithm seems to be achieved when the collision of many very simple objects is to be checked. The algorithm also takes advantage of the collision paths known in advance to decrease the frequency of the interference tests. As we have stated in the introductory section, the objectives of our algorithm are different.

An alternative approach to the collision detection is based on elimination of analytic intersection or distance checking by inexpensive, discrete tests at selected sample points. Uchiki et al. (1983) represent objects as a grid of voxels and detect the voxels assigned to more than one object. The accuracy of such an interference test is rather poor, and memory requirements are very high. Menon et al. (1994) proposed the ray representation of solids computed by "clipping" the grid of regularly spaced parallel lines against the solid. The method was developed for efficient handling of boolean operations for CSG representation of solids, including intersection, which can be used for the collision detection. However, specialized hardware architecture called RayCasting Engine is required to classify grids of parallel lines in interactive rates.

The interference detection through rasterization using z-buffer or ray tracing was proposed by Shinya and Forgue (1991). For every pixel the list of overlapping objects and corresponding depth values is calculated. The lists are sorted by z-values, and the object identifiers are grouped into pairs. The existence of a pair formed by two different objects is equivalent to a collision. Shinya and Forgue show that the complexity of the algorithm is in direct proportion with the number of polygons and the number of pixels. The algorithm performs well when the hardware accelerator can be used, otherwise it becomes too slow to be practical in realtime applications. Shinya and Forgue managed to use hardware supported z-buffer when all objects are *a priori* known to be convex. In this case, two depth maps (Z_{min} and Z_{max} values) must be stored for every object. A huge stream of bytes is copied from the depth buffer into the main memory. The sorting of depth maps and grouping objects identifiers into pairs is handled by the application software, which is slow comparing to the hardware supported realization. Also, maintenance of the lists is memory consuming. When concave objects are considered,

the graphics hardware only performs a simple test whether Z_{max} of the first object is smaller than Z_{min} of the second. When this test fails, a further processing by ray tracing or software z-buffer is required.

Rossignac et al. (1992a) propose a method for checking the interference between slices of solid objects, which are cut by two parallel cross-section planes. The hardware supported rendering using pixel masks and clipping planes is applied to determine whether the slices intersect, which is equivalent to the collision. The collision test is performed on the front and back cross-section planes, and this information is extrapolated to the interior of the slice. This approach is robust when the intersection between the slices of solids is detected, otherwise the collision cannot be excluded, and thinner slices should be considered. The recursive subdivision of slices is performed until the collision is detected or the maximum subdivision limit is reached. The algorithm usually requires many scan conversions of the solids and does not exhibit realtime performance. The processing time is sensitive to the initial guess of the collision zone. The calculation cost grows rapidly when many collision free slices should be recursively subdivided.

The techniques performing collision detection using rasterizing graphics hardware rely on the assumption that every straight line crosses the boundaries of the objects at an even number of points (Rossignac et al. 1992a); This condition holds in particular for solid objects. The proper treatment of tangent rays by scan conversion hardware is discussed in (Rossignac 1992b).

3 Overview of our approach

The algorithms proposed in this paper also use rasterizing graphics hardware and are applicable to solid objects only (i.e., bounded, closed, regular point sets with polyhedral surfaces, or other types of surfaces which are supported by the functionality of graphics hardware, e.g., NURBS). However, a smaller number of rendering passes is needed to reconstruct the collision areas comparing to the algorithm proposed by Rossignac et al. Also, more general classes of objects can be completely processed using hardware assistance comparing to Shinya and Forgue algorithm. The proposed techniques exhibit the following features improving performance and extending functionality of the algorithms discussed:

- Shinya and Forgue approach uses the *a priori* assumption that the objects are convex for effective use of hardware z-buffer. If this assumption holds, our basic algorithm seriously reduces depth maps processing performed by application software comparing to Shinya and Forgue techniques. We use the functionality of graphics hardware for depth sorting and detecting the interference regions. In many practical cases, only one rendering pass for each object is needed to detect some

collision and non-collision positions of the objects. (Shinya and Forgue algorithm can detect only some non-collision positions of the objects after single rendering pass.) Two rendering passes are only needed in more complex cases.

- No *a priori* assumption of the convexity of the objects is required by our extended algorithm. The algorithm checks for every pixel whether for a particular viewing projection used by the scan conversion algorithm the object should be treated as convex or concave. This check is obtained as a by-product of depth calculations using the standard capabilities of the graphics hardware. The appropriate algorithm handling convex or concave objects is chosen for the groups of pixels that are overlapped by the colliding objects in the same way. We show that in many cases concave objects can be processed by the cheaper algorithm for convex objects.

- The hardware assisted techniques for concave objects processing are proposed, which can handle many layers of the overlapping polygons per pixel. The maximum number of the layers is equal to the number of the bitplanes in the frame buffer.

- The accuracy of the interference test is improved.

Each of these items is described in more detail below. Our presentation develops in a step-by-step manner, starting from the simplest case of collision detection with the convexity of the objects known *a priori*. The latter assumption is not needed for an extended version of the basic algorithm, which checks the required convexity property. Finally, a general algorithm handling concave objects is proposed.

4 Convex objects

In this section we present the hardware assisted graphics algorithm for collision detection between a pair of convex objects A and B. The algorithm can be easily implemented on the commercially available high-end graphics workstations. We adjust particular operations of the algorithm to the functions supported by graphics libraries. At the end of the section we postulate some extensions for these functions, which would improve the efficiency of our techniques.

When the graphics techniques such as the ray casting or depth buffer are applied to collision detection, the problem is reduced to 1-D, and for two convex objects six trivial cases are possible (Fig. 1; bold horizontal lines mark the collision zones). Usually it is not important which particular case occurs, and it is enough to detect any kind of collision. The criterion for

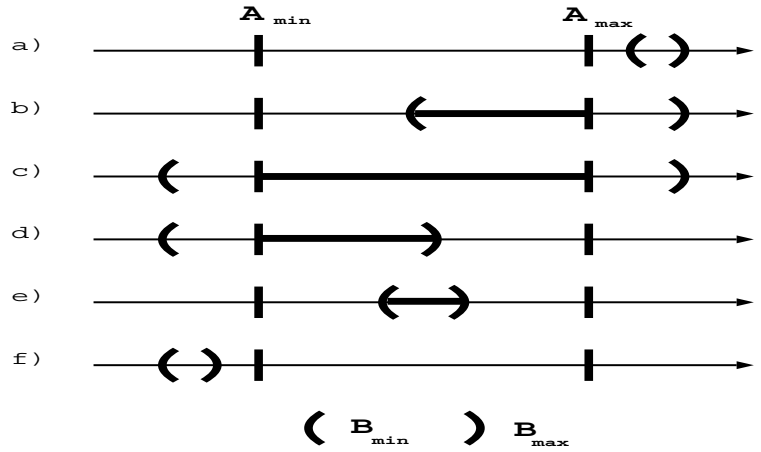

Figure 1: Interference of two convex objects in 1-D space.

interference may be stated in the following way: *There is no collision between the objects A and B if for every pixel overlapped by these objects, only the cases* a *or* f *occur.* (Of course, this statement is correct within the accuracy of the rasterization techniques, which for the particular viewing projection is determined by the frame and depth buffers resolutions.) It means that the exact sorting of intersection points is not necessary. The case *a* is equivalent to the successful test

$$Z^A_{max} < Z^B_{min} \tag{1}$$

where Z^A_{max} and Z^B_{min} stand for the maximum and minimum values of depth for the objects A and B respectively. Test (1) can be performed during a single rendering of A and B with the depth comparison enabled. However, the second rendering pass is needed when the test fails for some pixels, because the cases *f* (no collision) and *b*-*e* (Fig. 1) cannot be distinguished at the first pass. We propose the algorithm that detects conservatively not only the case *a* but also the cases *b* and *c* in the first rendering pass, thus avoiding the second rendering for some locations of A and B in space (Fig. 2).

In our formulation of the depth test we count the number of successful passes for the test

$$Z^A_{max} \geq Z^B_{arbitrary} \tag{2}$$

where $Z^B_{arbitrary}$ is the non-sorted depth value for B. The test is performed for the pixels overlapped by A.

The interpretation of the test results is straightforward. When fails only are scored, no collision is possible, because there is a clearance between A and B, or B does not overlap this pixel. The same result for all pixels means no collision, and the procedure is completed.

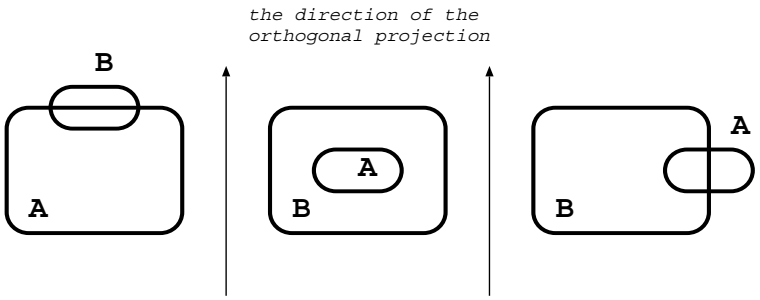

Figure 2: Locations of the objects A and B where the second rendering pass is not needed.

The collision is detected when one fail and one pass are scored (the case b or c in Fig. 1, see also Fig. 2). If the only goal of interference test is a binary decision, *collision* or *no collision*, this result completes the procedure. However, if the exact reconstruction of the whole collision area is required, then the second rendering is performed for the pixels where two passes are recorded.

After two passes of the test (2), collision cannot be excluded; the cases d, e or f are still possible (Fig. 1). The second rendering is needed to classify these cases. The depth test performed during the second rendering is

$$Z^A_{min} > Z^B_{arbitrary} \tag{3}$$

Taking into account the results of the first rendering, two passes mean no collision (the case f in Fig. 1); otherwise we have a collision (the case d or e in Fig. 1). It should be noted that the test (3) is equivalent to $Z^A_{min} \leq Z^B_{arbitrary}$ where fails and passes are interpreted in the opposite way; the test (3) was chosen for the consistency of interpretation of fails and passes in the following description of algorithms for concave and convex objects.

The outlined algorithm can be easily implemented using z-buffer rendering and pixel masks. Two operations are performed on the pixel masks: (1) setting the pixel mask `SA` when the object A overlaps the pixel, and (2) counting the number of polygons overlapping the pixel. The counter `SC` (implemented as two pixel masks) is incremented when the depth test is successfully passed for scan converted B.

```
The basic algorithm for the collision detection between
convex objects A and B
01  For all pixels set Z=0 and SA=0
02  Disable writing into the frame buffer
03  Set the Z comparison function for finding Z_max
04  Render all faces of A setting SA (search for Z_max)
05  Change the Z comparison for finding Z_min (test (2))
```

```
06  RenderObjectB()
07  second_rendering = FALSE
08  For all pixels where SA=1
09    If (SC == 1) return COLLISION
10    If (SC == 2) second_rendering = TRUE
11  If (second_rendering == FALSE) return NO_COLLISION
12  For all pixels If (SC == 2) SA=1 Else SA=0
13  Enable writing into the z-buffer
14  For all pixels set Z=INFINITY
15  Render all faces of A for pixels where SA=1 (search for Z_min)
16  Change the Z comparison for finding Z_min (test (3))
17  RenderObjectB()
18  For all pixels where SA=1
19    If (SC == 0 or SC == 1) return COLLISION
20  return NO_COLLISION

RenderObjectB()
01  Disable writing into the z-buffer
02  For all pixels set SC=0
03  Render all faces of B for pixels where SA=1
    and increment SC when the depth test passes
```

When the object A is rendered, the pixel mask `SA` is set for every pixel overlapped by A (Line 4). The object B is only rendered for pixels where `SA` is set (Line 3 in the procedure `RenderObjectB`). Pixels overlapped by A and not overlapped by B have the counter `SC`=0 when the first rendering pass is completed (the same value corresponds to the case a in Fig. 1), and are ignored in the second rendering pass (Line 12).

The whole algorithm except pixel masks examining (Lines 09, 10 and 19) was implemented using standard functions provided by the graphics libraries. It means that on high-end workstations, most of these calculations are hardware supported. Unfortunately, we were forced to examine final values of the counter `SC` by the application software for every pixel marked by `SA`. We hope that in future releases of graphics libraries, services such as checking the settings of the chosen pixel masks for the whole frame will also be hardware assisted; similar extensions of functionality have also been postulated by Rossignac et al. (1992a).

In the pseudo-code the frame buffer is not explicitly used. However, in practical implementation the access to stencil planes is granted by drawing into the frame buffer in the way controlled by the status of chosen pixel masks. As the depth of the frame buffer on the high-end workstation is at least 24 bits, the status of the pixel masks acquired during two rendering passes can be easily stored there.

In the pseudo-code, we assume that the goal of the algorithm is to detect any collision between objects. However, in fact, the algorithm calculates the complete map of the collision areas, which are directly available and processed by the application software (Lines 09, 10 and Line 19). The processing can be easily adjusted to the application needs. It is worth to note that the algorithm distinguishes all possible collision cases (Fig. 1) except the cases b and c, which are encoded in the same way in the collision map (Line 09).

In order to reduce the number of rendering passes and computations done by the application software, the following measures can be taken:

- As the initial guess, consider the object located further along the direction of the orthogonal projection as B.
- When two rendering passes are required for the particular locations of objects and two successful passes of test (2) are recorded for many pixels, the order of the considered objects can be reversed at the next animation step (the current A becomes B and vice-versa).
- When the objects are simple (can be rendered very fast) and high resolution pixel masks are used (because of the required accuracy of the collision detection), the examination of `SC` by the application software becomes the bottleneck of calculations. In such a case, a speedup can be achieved when `SC` are examined only once after completion of two rendering passes. As has already been mentioned, the frame buffer can be used to store the `SC` calculated during the first rendering pass.

It should be noted that when the collision detection is completed after the first rendering pass and the orthogonal projection parameters are not changed for the next animation step, then the rendering of A should not be repeated (Lines 01-05 can be skipped). (When A is mobile, then the depth maps can be re-used when the relative motion of B with respect to A is considered.) The rendering of B does not overwrite the depth values calculated for A (Line 01 of the procedure `RenderObjectB`). Of course, the second rendering destroys the depth maps calculated during the first rendering. When the second rendering is often required and A is complex, it may be worth to store the depth maps of A in the main memory and copy them into the depth buffer instead of repeating rendering. Direct writing into the depth buffer is supported by graphics libraries and can be faster than rendering of the complex object.

5 Convexity test

The algorithm proposed in the previous section works efficiently for convex objects; however, in practical applications we are very often facing non-convex objects. Usually it is not *a priori* known whether the object is convex

or not, e.g., when the description of shape is derived experimentally through 3-D scanning.

We may notice that the presented algorithm does not require the *global convexity* of the colliding objects. Let P be a point in the surface of an object. We call the object *convex at P in a direction π* if the intersection of the line through P in the direction π with the object is connected (that is, if it is a single segment). From the point of view of representation of the object, convexity in a direction π at a pixel p means that there are exactly two polygons in the polygonal representation of the boundary of the object whose projections in the direction π contain p (for brevity, we say that the object is *π-convex* at the pixel p). We assume here that pixels in the projections of edges are treated as belonging to exactly one of neighboring polygons; this assumption is commonly implemented in graphics hardware (SGI 1992; Neider et al. 1993). If A and B are π-convex in a given projection direction for all pixels overlapped by A and B, then the algorithm presented in the previous section works properly, and we need not care about the global convexity.

In this section the basic collision detection algorithm for convex objects (Section 4) is extended to determine the π-convexity of A and B in a given projection direction and to handle the simple cases of interference between concave objects. The π-convexity is checked for every pixel by counting the number of the overlapping polygons. When the π-convexity holds, the old approach can be used. Otherwise our strategy is to solve the collision problem for as many pixels as possible, minimizing the number of pixels where a more expensive algorithm for concave objects must be used. The extended algorithm processes completely collision between two objects, one of which is π-convex. The proposed extension of functionality requires only minor changes in the basic algorithm, which can be handled by standard graphics libraries, except inspecting the pixel masks settings.

The first rendering pass for the objects A and B is the same as in the basic algorithm. The counter `SC` is also incremented when the depth test (2) for the polygon of B passes. However, a greater number of pixel masks is needed to count all overlapping polygons and avoid the counter overflow. The number of `SC`'s pixel masks depends on the expected complexity of B, e.g., three pixel masks can handle safely 6 layers of polygons overlapping a pixel, but for 8 layers the counter clamping to its maximal value is guaranteed (SGI 1992; Neider et al. 1993). In such a case the counter overflow is easy to detect, because `SC`=7 is the value which is not allowed for solid objects. The interpretation of `SC` (Lines 08-12 in the basic algorithm for convex objects) is extended in the following way:

```
Extension #1 of the basic algorithm for concave objects
080  For all pixels where SA=1
090    If (SC odd)  return COLLISION
```

```
100    Else If (SC > 0) second_rendering = TRUE
110  If (second_rendering == FALSE)  return NO_COLLISION
120  For all pixels If (SC > 0) SA=1 Else SA=0
121  For all pixels set SC=0 (used for the convexity test of A)
...
150  Render all faces of A for pixels where SA=1
     and always increment SC (ignore results of Z comparison)
151  For all pixels If (SA == 1 and SC == 2) SCA=1 Else SCA=0
```

When SC=0 for all pixels, collision is impossible. The second rendering pass is required for pixels where collision was not detected (Line 090), and the number of the overlapping polygons is even and non-zero (Line 100). The second rendering pass is performed only for the pixels where the pixel mask SA is set on (Line 120).

When A is rendered the second time, SC counts the number of polygons overlapping the pixels where SA=1 (Line 150). After completion of rendering, one pixel mask SCA is allocated and set to 1 for pixels overlapped by two polygons (SC=2), i.e., exhibiting the property of the π-convexity for the object A (Line 151).

The second rendering of the object B and use of SC are the same as in the first pass. The interpretation of SC is more complex than in the basic algorithm (Lines 18-20, Section 4) and requires the comparison of SC calculated for the first and second rendering passes (hereafter referred as SC1 and SC2, respectively).

```
Extension #2 of the basic algorithm for concave objects
180  For all pixels where SA=1
190    If (SC2 odd) return COLLISION
191    Else If (SC1 == SC2) continue
192    Else If (SCA == 1) return COLLISION
193    Else mark this pixel to be processed by the algorithm
            for concave objects
200  If (Line 193 is never reached)
201    return NO_COLLISION
202  Else
203    Process pixels marked for the concave objects
```

Fast operations performed on stencil planes allow to mark uniquely the pixels in the frame buffer that pass tests in Lines 190-193. However, final interpretation of the frame buffer is done by the application software. Line 191 requires the copies SC1 and SC2 of the counter SC obtained at the first and second rendering passes. If the same even number of polygons overlapping the pixel passed the depth test in the first and second rendering of B, then

the collision is not possible. Line 192 examines whether the object A is π-convex at the current pixel. If this is the case, then for even SC2, and SC2 different from SC1 the collision is detected, completing the procedure for the considered pixel. However, when A is not π-convex, the further processing, described in the next section is needed.

If two objects are concave, then the more complex object should be chosen as A to avoid SC overflow. If one object is convex, then it should be rendered as A, and the complex algorithm for concave objects is not used (Line 193 is never reached).

The critical issue is the number of pixel masks supported by the graphics workstation (e.g., up to eight stencil planes on SGI's RealityEngine^2). When the second rendering pass is required, the copies of SC (SC1 and SC2) for two rendering passes are needed, plus two single pixel masks SA and SCA. If B is not complex (up to six polygon layers per pixel), then 8 pixel masks are sufficient to store all informations. Otherwise SC1 should be stored in the frame buffer before the second rendering pass is started; this is also the case when only four pixel masks are available, as e.g., in SGI's Indigo^2. In such a case operations on the pixel masks that could be supported by graphics hardware are performed by the application software.

The extended algorithm minimizes the number of pixels which should be handled by the time consuming collision detection for concave objects. However, this can require many pixel masks to process very complex objects. The simpler version of this algorithm can be implemented using only three pixel masks planes (one for SA and two for S0 and SP which replace the former SC). In the first rendering pass, B should be scan converted twice: (1) for S0 setting, for pixels overlapped by B, and (2) for SP toggling, when the depth test passes successfully. The interpretation of S0 and SP is the following:

```
Extension #1' of the basic algorithm for concave objects
(simplified version)
080  For all pixels where SA=1
090    If (S0 == 1 and SP == 1)  return COLLISION
100    If (S0 == 1 and SP == 0)  second_rendering = TRUE
110  If (second_rendering == FALSE)  return NO_COLLISION
120  For all pixels If (S0 == 1 and SP == 0) SA=1 Else SA=0
```

In the second rendering pass, B should be scan converted only once, toggling SP for every successful pass of the depth test. The final evaluation of SP is presented below:

```
Extension #2' of the basic algorithm for concave objects
(simplified version)
180  For all pixels where SA=1
190    If (SP == 1) return COLLISION
```

```
191    Else mark this pixel to be processed by the algorithm
           for concave objects
200  Process pixels marked for the concave objects
```

The simplified algorithm requires further processing for some pixels that could be fully processed by the extended version of the collision detection algorithm, because the π-convexity is not checked. Also, one extra rendering of B is required.

When the objects are concave, Shinya and Forgue perform only one test $Z^A_{max} < Z^B_{min}$, and for all remaining pixels apply the ray tracing or software z-buffer. This means that the software implemented techniques have to process many pixels, which can be completely processed by graphics hardware when our algorithm is used.

6 Concave objects

The unsolved cases for a pair of concave objects which cannot be handled by the algorithm presented in the previous section require the polygon sorting along the direction of the orthogonal projection. The similar requirement is imposed by rendering of CSG or transparent objects. The traditional methods used to solve these problems like the ray casting (Roth 1982), A-buffer (Carpenter 1984) or ZZ-buffer (Salesin and Stolfi 1990) are too slow, and a significant speedup can only be expected when hardware supported sorting of polygons is provided. The most popular approach is polygon sorting performed during the multi-pass rendering. Mammen (1989) solves the problem of transparency using the duplicate depth and frame buffers. The first buffer stores the reference depth, and the other one is used to find the nearest object located in front of this reference. The object just found becomes the reference in the next rendering pass when the sorting is continued for the next layer of polygons. The implementation of this algorithm requires flexible configuration of the frame and depth buffers offered by the Stellar graphics accelerator. Recently, low cost, single-ASIC graphics accelerator was produced for Power Macintosh (Kelley et al. 1994), which stores and sorts up to four depth values for 16 pixels long segment of the scan line. Unfortunately, similar capabilities are not yet very common on the commercially available workstations. However, the number of bits per pixel is growing for the subsequent generations of graphics hardware. Also, graphics libraries offer more and more freedom in tailoring these resources to the requirements imposed by applications. Observing these trends, we believe that hardware assisted polygon sorting will be easy to implement on many hardware platforms soon. As it was shown, some platforms offer such functionality even now.

In this section we propose an algorithm handling the collision detection for concave objects. It is assumed that all polygons are sorted along the direction

of the orthogonal projection. The realtime performance of this algorithm can only be expected when hardware supported object sorting is available. (In our current implementation the sorting is done by the application software). The layers of polygons can be rendered in the front-back or back-front order. The alpha blending is used to accumulate information about the order of the polygon layers belonging to A and B. The value $\alpha = 0.5$ is used, and the blending model is described by the formula

$$C_{acc} = C_{object} + \alpha C_{acc} \tag{4}$$

where C_{acc} stands for the accumulated value of the R,G,B color components stored in the frame buffer, and C_{object} is the corresponding component of the object color. The color of the first object is set to zero for all components. The color of the second object is set to 2^{k-1}, where k is the number of bitplanes for a single color component. The proposed blending function shifts the contents of C_{acc} right and sets the most significant bit to one or zero depending on whether A or B is rendered. As the result, zeros and ones in C_{acc} correspond to the objects A and B respectively. The collision can be detected when an odd number of the same object identifiers is found in the sequence in C_{acc}, e.g., $AABBABBA$ (00110110), $BBBAABBB$ (11100111).

One may notice that 8-bit C_{acc} encodes up to 8 layers of the overlapping polygons. However, C_{acc} consists of three RGB buffers, which can be used to encode up to 24 layers (or 32 layers when alpha bitplanes are available, e.g., RealityEngine^2). The technical problem to be solved is to prevent blending operation simultaneously for all RGB buffers. This can be easily done using masking two components and allowing the blending operation for the third. Every eight rendering passes (corresponding to 8 layers processed) writing to another color component is enabled while the component already processed is masked. Another technical problem is related with the numerical accuracy when alpha blending is performed. We did not encounter such problems on SGI's Extreme graphics hardware when $\alpha = 128/255$ was assigned. On SGI's RealityEngine^2, floating point α is supported, and our layers encoding also works fine. However, when numerical problems are expected, the least significant bit should not be used, and the next color component should be switched every seven rendering passes.

Multi-pass rendering for complex multi-layered objects is time consuming. However, the calculations are only required for pixels that cannot be processed by the simple depth comparison and counting of parity of layers proposed in the previous section. If these pixels are concentrated into small and compact image areas, then viewport parameters can be changed focusing multi-pass rendering on such areas. Many polygons outside the viewport will be immediately discarded at the clipping stage, avoiding scan conversion. On the other hand, when only a small number of dispersed pixels requires the elaborated multi-pass rendering, then the ray tracing is the best choice. The

adaptive density of traced rays can be applied to surfaces exhibiting high variation of depth for neighboring samples.

7 Accuracy issues

An important problem is the choice of the orthogonal projection that provides high accuracy of the collision detection. The projection should maximize the number of pixels falling into the collision zone. The pixels which do not overlap the objects A and B do not provide any useful information. The best depth measurement accuracy in the proximity of the collision can be expected when the projection plane is tangent to the surfaces of A and B at the point of collision. When the angle between the projection plane and polygon increases, the accuracy of depth calculations drops abruptly. However, when the contact between colliding surfaces is complex, all these requirements cannot be strictly satisfied if only one projection is considered. We propose an inexpensive algorithm for the choice of the initial guess of the projection plane taking into account the approximate collision zone calculated as the intersection of bounding volumes. Then we propose a more exact algorithm using information acquired at the previous step of the collision detection.

The bounding volume intersection test is applied to eliminate quickly the cases where the collision is impossible. First, the bounding volume of the object A transformed to the local coordinates of B is tested against the bounding volume of B. If intersection is detected, the same test in the local coordinates of A is performed. As the result, two perpendicular parallelepiped intersection volumes are obtained. For each intersection volume, the biggest wall is found. The average of the normal vectors to these two walls is the initial guess of the orientation of the projection plane used by our collision detection techniques. The orthogonal projection of the intersection volumes on this plane establishes the viewport parameters used during the scan conversion.

The accuracy of the collision detection may be improved when the projection plane is calculated as the average of the normal vectors of all polygons in the collision zone. The surface areas of the polygons can be used as weights for such averaging. The polygons in the collision zone are identified using the concept of the *item buffer* (Weghorst et al. 1984), where each polygon is drawn to the frame buffer using a unique color. In such a case drawing into the chosen bitplanes of the frame buffer should be enabled (Line 02 of the basic collision detection algorithm); other bitplanes remain reserved to store a copy of stencil planes. The application of the item buffer is just a special way of rendering and does not involve any additional cost. Of course, the processing of the item buffer and calculating the average normal vectors requires some extra time and should not be repeated for every step of the animation of the colliding objects. When changes of the objects position

are small, the same projection plane can be used for subsequent collision detection tests.

In many practical applications the expected collision zone can be guessed in advance, and then the best orthogonal projection plane can be found once forever at the preprocessing stage.

When high accuracy of the collision detection is critical and the variation of the normal vectors of polygons in the collision zone is high, then multiple projection planes can be defined. Of course this requires multiple rendering passes. Another solution, originally proposed by Shinya and Forgue (1991) is to perform exact calculations by application software for colliding polygons identified in the hardware assisted way. This solution can be extended to process spike-shaped surfaces, which can be easily missed by rasterizing graphics hardware. During scan conversion, the special bounding volumes can be rendered instead of original spike-like surfaces. When the item buffer contains the identifier of such a bounding volume, then calculation for the original surface can be conducted by the application software.

8 Experimental results

The algorithms presented in this paper were implemented on Onyx and Indigo^2 Silicon Graphics workstations. All timings given in this section were measured for RealityEngine^2 and Extreme graphics hardware (the results for Extreme are in brackets).

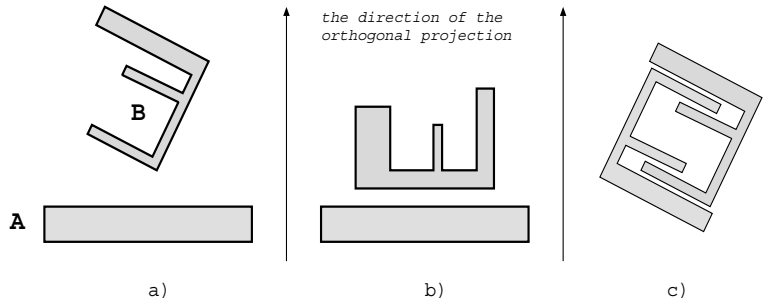

Figure 3: The examples of test objects: a) one object π-convex, b) two objects π-convex, c) general case.

In order to get easy to validate results, we evaluated the efficiency of our collision detection algorithm for multiple simple polyhedra. We simulated variable complexities of the objects by tessellating their faces into multiple triangles. Table 1 presents timings of the collision detection for scenes in Fig. 3 assuming that every scene is built of 160,000 3-D polygons and the resolution of stencil and frame buffers is 512×512 pixels. The scene in Fig. 3a, which is composed of one π-convex and one concave object (up to six lay-

	Graphics hardware	Application software			Total
		I-phase	II-phase	III-phase	
Fig3a/RE2	0.27	0.13	0.17	–	0.84
Fig3b/RE2	0.27	0.13	0.10	–	0.77
Fig3c/RE2	0.27	0.13	0.17	0.28	1.39
Fig3a/Extreme	0.52	0.18	0.25	–	1.47
Fig3b/Extreme	0.52	0.18	0.14	–	1.36
Fig3c/Extreme	0.52	0.18	0.25	0.35	2.34

Table 1: Timings of collision detection for scenes in Fig. 3 measured for SGI's RealityEngine^2 and Extreme graphics hardware. All values are given in seconds.

ers of polygons per pixel), was processed applying the algorithm described in Section 5. The basic algorithm presented in Section 4 was tested using the scene in Fig. 3b composed of two π-convex objects. A general algorithm for concave objects (Section 6) was used to check the collision between objects in Fig. 3c. Total calculation time (last, right column) was measured in the worst case scenario, i.e., when two or three (for two concave objects in Fig. 3c) rendering passes were performed. The column *Graphics hardware* presents timings of the single rendering of the objects A and B and the corresponding pixel masks processing (in practice, no difference between the first and second rendering passes was registered). The cost of examining the pixel masks by application software, including the image transfer from the frame buffer to the main memory is given separately for every rendering phase in the column *Application software*. The sub-column *III-phase* shows costs of the analysis of the order of polygons encoded by colors in the frame buffer. These calculations are relevant only for the algorithm processing two concave objects, which requires one extra rendering pass and encoding the order of polygons through the alpha blending (Section 6). We assumed that polygons are depth sorted, and the cost of sorting (at present done by the application software) was not included in Table 1. The algorithm for concave objects exhibits reasonable performance under the condition that depth sorted polygons are available, i.e., hardware support for the sorting is provided. Also, the comparison of the algorithm for convex objects (Section 4) and more general algorithm for one π-convex object and arbitrary object (Section 5) shows

that their performance is similar; however, the latter algorithm requires more pixel masks.

We noticed the linear growth of the calculation costs with respect to the number of polygons for all presented algorithms. This is not a surprise, because Shinya and Forgue (1991) also reported the linear complexity of their algorithm. However, this means that hardware assisted operations on pixel masks and the alpha blending did not affect the overall algorithm complexity.

We also tested the cost of our algorithms dependency on the resolution of the frame buffer. The rendering time was quite stable for resolutions falling into the range from 128×128 to 512×512 pixels (these results indicate an overload of geometry engines and underload of raster operations in our application), while timings of the processing of pixel masks by the application software exhibited the linear growth with respect to the number of pixels.

We applied rasterization based collision detection in simulation of human jaws articulation (Kunii et al. 1994). At present we only have access to the description of the shape of teeth (Fig. 4) measured experimentally as the height of field data using mechanical scanner. In this case the collision detection can be reduced to the trivial test (1), when the direction of the orthogonal projection corresponds to the depth axis of the scanner used during the measurement of the upper or lower jaw. However, the mechanical scanning requires a plaster mold of the jaws, and the measurement takes a long time. In practical applications, optical scanning performed for multiple viewing directions directly inside the mouth of patient is used (Duret et al. 1988). This means that the general algorithm of the collision detection for concave objects is needed. Having only simplified measurement data, we simulate such situation by arbitrary positioning of teeth with respect to the orthogonal projection direction. Additional polygons are inserted at the bottom of every tooth to make the surface closed.

Fig. 4 shows the general view of human jaws exposing the complexity of possible contacts between teeth. We focus our analysis on five teeth built of over 110,000 polygons (Fig. 5; teeth belonging to the upper jaw are brighter). An important issue is sensitivity of the algorithm to the contact complexity. The positions of the jaws shown in Figs. 5a (no collision) and 5b (collision) required only a single rendering pass, and the collision detection timings were the same within the accuracy of time measurement 0.30s (0.49s). Two rendering passes and a single alpha-blending pass were needed for the positions shown in Figs. 5c and 5d. The collision detection took 1.11s (1.78s) in both cases, excluding time of sorting polygons. Of course, when the sorting is based on multi-pass rendering, this time will be longer, proportionally to the number of layers of polygons overlapping pixels (up to 8 layers in this case). Figs. 6a and 6b present the maps of distance between teeth corresponding to Figs. 5b and 5c. Gray and pink colors depict pixels only covered by the upper and lower jaws respectively. Red and yellow colors mean interference, while

the border between yellow and green shows the contact area. In Fig. 6a the collision is just detected during free motion of the lower jaw, and the contact zone is very small (isolated pixels).

We applied ray tracing as a reference method for validation of the depth results provided by rasterizing graphics hardware. We used 16 rays per pixel, and the average distance between jaws was calculated. This distance was compared with the corresponding distance calculated using z-buffer (24 bits were used to encode the depth). Fig. 7 (upper) presents the distribution of errors over the occlusal surface of the teeth in Fig. 5c. In general the error is kept at the low level ($RMS\ Error = 0.0009\,mm$), however, in the proximity of edges of the occlusal surface it grows significantly (red color in Fig. 7; $\delta_{max} = 0.06\,mm$). The higher error is related to the orientation of polygons with respect to the orthogonal projection plane. The projection plane was adjusted to the orientation of polygons using the techniques based on the item buffer (Section 7), otherwise the error can be even more significant (Fig. 7, lower).

9 Conclusions

In this paper efficient techniques, exhibiting realtime performance for the collision detection between complex solids have been presented. The sensitivity on the complexity of the contact between objects is very low. The method handles both convex and concave solids. The algorithm checks the π-convexity of the objects as a by-product of scan conversion, and in many cases discussed in the paper, the concave objects can be completely processed within one or two rendering passes. The algorithm refers to typical functionality of graphics hardware and can be easily implemented on high-end workstations. The hardware assisted calculations constitute the main strength of the method, however, at present a missing functionality of graphics libraries forced us to perform some operations (namely, inspecting the values of stencil planes and the depth sorting of polygons) by an application software. Since similar extensions of functionality are required by other important applications (e.g., rendering of semi-transparent or CSG objects), we believe that such hardware support will soon be commonly available, even further improving the efficiency of our method.

The main drawback of the method, inherent for all rasterizing graphics techniques is the possibility of missing collisions that occur between sample points (pixels). The resolution of the frame buffer is limited, and in many cases cannot be adjusted to the requirements of the collision detection accuracy. A hierarchical, multi-resolution structure of collision maps can be built, which contains more detailed information on the regions of special interest, e.g., exhibiting a complex geometry. However, a further research is needed to find out how to construct such a structure to reduce the collision detection

errors.

Another important topic of future research is the choice of viewing parameters used for scan conversion. When the variation of the orientation of surface polygons is high, multiple directions of orthogonal projection should be used to secure high accuracy of the collision detection. An open question is the number and parameters of such multiple projections.

10 Acknowledgments

The authors would like to thank Dr. Masumi Ibusuki for educating us about the issues involved in the human jaws articulation. Thanks also to Yoshihisa Shinagawa for providing the measurement data of teeth, and Yuko Kesen, Kazuaki Yamauchi for their contributions to this work. Special thanks for proofreading of the manuscript go to Alexander Pasko and Christophe Lecerf.

Thanks are to *Fukushima Prefectural Foundation for the Advancement of Science and Education* and *Multimedia Open Network & Virtual Reality for New Education Consortium (MOVE)* for support of the Intelligent Dental Care System project.

References

[1] Graphical Library Programming Guide. Document Number 007-1680-010. Silicon Graphics, Inc., 1992

[2] Aziz N, Bata R, Bhat S (1990) Bézier surface/surface intersection. IEEE Comput Graph Appl 10(1):50–58

[3] Boyse JW (1979) Interference detection among solids and surfaces. Communications of the ACM 22:3–9

[4] Cameron SA, Culley RK (1986) Determining the minimum translational distance between two convex polyhedra. Proceedings of International Conference on Robotics and Automation 2:591–596

[5] Carpenter L (1984) The *A*-buffer, an antialiased hidden surface method. ACM Comput Graph Proc SIGGRAPH '84 18:103–108

[6] Culley RK, Kempf KG (1986) A collision detection algorithm based on velocity and distance bounds. Proceedings of International Conference on Robotics and Automation 2:1064–1069

[7] Duret F, Blouin JL, Duret B (1988) CAD/CAM in dentistry. Journal Am Dent Assoc 117:715–720

[8] Garcia-Alonso A, Serrano N, Flaquer J (1994) Solving the collision detection problem. IEEE Comput Graph Appl 14(3):36–43

[9] Hahn JK (1988) Realistic animation of rigid bodies. ACM Comput Graph Proc SIGGRAPH '88 22:299–308

[10] Hanna S, Abel J, Greenberg DP (1983) Intersection of parametric surfaces by means of lookup tables. IEEE Comput Graph Appl 3(7):39–48

[11] von Herzen B, Barr AH, Zatz HR (1990) Geometric collisions for time-dependent parametric surfaces. ACM Comput Graph Proc SIGGRAPH '90 24:39–48

[12] Kelley M, Gould K, Pease B, Winner S, Yen A (1994) Hardware accelerated rendering of CSG and transparency. ACM Comput Graph Proc SIGGRAPH '94 28:177–184

[13] Kunii TL, Herder J, Myszkowski K, Okunev O, Okuneva GG, Ibusuki M (1994) Articulation simulation for an Intelligent Dental Care System. Displays 15:181–188

[14] Mammen A (1989) Transparency and antialiasing algorithms implemented with the virtual pixel maps technique. IEEE Comput Graph Appl 9(4):43–55

[15] Menon J, Marisa RJ, Zagajac J (1994) More powerful solid modeling through ray representations. IEEE Comput Graph Appl 14(3):22–35

[16] Moore M, Wilhelms J (1988) Collision detection and response for computer animation. ACM Comput Graph Proc SIGGRAPH '88 22:289–298

[17] Neider J, Davis T, Woo M (1993) OpenGL Programming Guide. Addison-Wesley, New York

[18] Phillips MB, Odell GM (1984) An algorithm for locating and displaying the intersection of two arbitrary surfaces. IEEE Comput Graph Appl 4(9):48–58

[19] Rossignac J, Megahed A, Schneider B-O (1992) Interactive inspection of solids: Cross-sections and interferences. ACM Comput Graph Proc SIGGRAPH '92 26:353–360

[20] Rossignac J (1992b) Accurate scan conversion of triangulated surfaces. In: Kaufman A (ed) Advances in Computer Graphics Hardware VI. Springer, Berlin Heidelberg New York

[21] Roth SD (1982) Ray casting for modelling solids. Comput Graph Image Process 18:109–144

[22] Salesin D, Stolfi J (1990) Rendering CSG models with a ZZ-buffer. ACM Comput Graph Proc SIGGRAPH '90 24:67–76

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

[24] Uchiki T, Ohashi T, Tokoro M (1983) Collision detection in motion simulation. Comput Graph 7:285–293

[25] Weghorst H, Hooper G, Greenberg DP (1984) Improved computational methods for ray tracing. ACM Trans Comput Graph 3(1):52–69

[26] Yang Y, Thalmann NM (1993) An improved algorithm for collision detection in cloth animation with human body. Pacific Graphics '93 1:237–251