

# Collision detection for objects modelled by CSG

M. Zeiller

*Institute for Computer Graphics, Technical  
University of Vienna, Karlsplatz 13/186, A-1040  
Vienna, Austria*

## ABSTRACT

Physically based computer animation has to deal with the problem of detecting collisions of moving objects and preventing solid objects from interpenetration. This paper discusses a method for detecting collisions among complex objects modelled with the constructive solid geometry paradigm. Bounding volumes in each node of the CSG tree are used to determine as soon as possible whether a collision is likely to occur. Pairs of primitives – one of each object – are determined whose bounds cover common space. Spatial subdivision is used to detect these candidate pairs which have to be tested for collision using a geometric approach. The geometry of the intersection region is used to determine the collision point and the time and type of contact which are required for collision response.

## INTRODUCTION

In computer animation objects roll and tumble through space, change their size or shape, alter their visible appearance and many more. When they move around, it may happen that they interpenetrate. To prevent a rigid object moving through another object as if it were not present, such interpenetrations, which are called collisions, have to be detected. To achieve realistic behaviour of the objects, the correct movement of the objects after the impact has to be determined as well. Therefore two tasks have to be performed: collision detection and collision response.

In computer graphics several representations for objects are used. Many modellers are based on the constructive solid geometry paradigm. Therefore CSG objects will be frequently used in computer animation, too. To be able to use this modelling technique in dynamic simulations, collision detection and collision response for CSG based objects have to be developed. However, most of the work on collision detection has focused on polyhedral objects. This paper presents a method for exactly detecting collisions among objects modelled by CSG. Cameron [1] has introduced an approach for detecting intersections among objects defined constructively. Our method is somewhat similar since we also decompose the detection of collisions into three stages. Cameron is only interested whether there is an intersection between two

objects or not, since he is using this technique in the robotics field. However, we are dealing with collision detection for computer animation. Therefore we also have to determine the collision point and the time and type of contact which are necessary to be able to perform collision response. We will use Cameron's S-bounds theory to locate the region of a collision if there is any. If this region is not empty then a collision may occur and we have to test those primitives in the collision region whether they collide. To be able to find pairs of primitives – one of each colliding CSG object – we are using spatial subdivision to create an octree-like data structure for the CSG tree. Finally we have to determine the collision point and the time and type of contact of all pairs of primitives if they collide. Collision response will not be dealt with in this paper since analytical penalty methods can easily be applied to CSG objects as well.

The next section will give a survey on work on collision detection and collision response. The following section will present some basics on constructive solid geometry and bounding volumes. Afterwards we give an overview of how collision detection is applied to moving CSG objects. Then we describe the location of the intersection region using S-bounds. In the following section we introduce an algorithm to perform a spatial subdivision of a CSG tree which can be used to determine candidates for colliding primitives when two CSG objects intersect. This is followed by detecting a collision and the collision point between two primitives. Finally we present some ideas for future work.

## RELATED WORK

Collision detection is a kinematics problem. Two objects collide if their surfaces intersect. Collisions can be determined by geometric reasoning whether the intersection of the objects is empty or not (e.g. null object detection). An intersection depends on the shape as well as on the position and orientation of the objects. Several algorithms for detecting collisions among rigid objects have been presented. Since the shape of the involved objects greatly influences the applied methods for detecting collisions, these algorithms can be classified depending on the object representation. Most of them concentrate on polyhedral objects. Moore and Wilhelms [2] present a method for detecting collisions among convex polyhedra based upon a clipping algorithm and among flexible surfaces modelled by triangles. Hahn [3] introduces an algorithm suitable for detecting collisions among objects modelled by polygonal surfaces, too. More recent approaches for detecting collisions among moving objects like Bouma and Vanecek [4] as well as Lin and Canny [5] are also restricted to polyhedra. In contrary, Baraff [6] detects collisions among rigid bodies bounded by curved surfaces. Von Herzen et al. [7] present an algorithm to detect collisions between pairs of time-dependent parametric surfaces. Cameron's approach [1] for detecting intersections among objects defined constructively consists of three stages. First he creates a set of bounds called S-bounds around the nodes of the CSG tree. In the next stage he splits the problem into simpler subproblems using spatial subdivision. Finally he generates a set of sufficient points that are classified regarding to the objects being tested. All approaches have in common that the amount of work to detect a collision is quite remarkable. Since these tests have to be performed at each time step in the dynamic simulation, these tests should only be applied when necessary. Therefore optimizations like bounding boxes (Hahn [3]) and octrees

(Moore and Wilhelms [2]), exploiting coherence between two consecutive time steps (Baraff [6]) and even parallel architectures (Bickerstaff and Hellestrand [8]) have been proposed to speed up the computation.

Collision response is a dynamics problem. After a collision has been detected, the time, location and type of the first contact have to be determined. Based on these quantities and the preceding movement, the behaviour of the colliding objects after the impact has to be computed. To achieve a realistic behaviour according to physical laws, the linear and angular momentum has to be preserved. Friction and elasticity of the involved surfaces have to be considered as well. The main problem of collision response is to determine the contact forces between bodies in contact which can be done by analytical methods or penalty methods. Analytical methods have been presented by Moore and Wilhelms [2], Hahn [3] and Baraff [6,9]. Moore and Wilhelms present an analytical linear system solution suitable for rigid and articulated objects that calculates the impulse between two colliding objects. Hahn uses analytical and numerical solution techniques that consider only the states of the involved objects before and after the collision. In [9] Baraff determines forces that prevent a number of non-colliding rigid polyhedral objects from interpenetration by solving a system of equality and inequality constraints, whereas in [6] he determines analytical forces between curved surfaces. On the other hand, penalty methods introduce restoring forces that separate penetrating objects. Moore and Wilhelms [2] present a method based on temporary springs at collision points which can be applied to several kinds of objects.

## OBJECTS MODELLED BY CONSTRUCTIVE SOLID GEOMETRY

### Constructive Solid Geometry

Constructive solid geometry operates on parameterized instances of solid primitives that are combined by Boolean set operations (Mäntylä [10]). Several kinds of primitive shapes can be used in CSG modelling: spheres, cylinders, cones, cubes, polyhedra and even more complex shapes like torii and superquadrics. Since these primitives are represented by mathematical models, their shapes are exactly modelled.

CSG objects are usually represented by a so-called CSG tree. A CSG tree is a binary tree with Boolean set operations in its intermediate nodes and primitives in its leaf nodes. The set operations used are union ( $A \cup B$ , OR-nodes), intersection ( $A \cap B$ , AND-nodes) and difference ( $A - B$ , MINUS-nodes). We will assume regularized set operations, i.e. no 'dangling edges' will occur due to these operations. The leaf nodes represent instances of primitive shapes which are combined by the set operations. Each primitive is represented as a type identifier and a sequence of parameters describing its size, position and orientation which might as well be encoded in a transformation matrix. In this paper we will restrict these primitives to be natural quadrics (sphere, cylinder, cone) and regular convex polyhedra (Platonic solids: tetrahedra, cube, octaeder, dodecaeder, icosaeeder).

### Bounding Volumes

CSG models are used in many fields in computer graphics. Since many of these applications are computationally very intensive, the computations are accelerated by bounding volumes. These volumes enclose the object, but since

they have a much simpler shape an approximate solution can easily be obtained. Bounding volumes most frequently used are spheres, axis-aligned boxes, oriented boxes and intersections of infinite slabs. In CSG modelling a hierarchy of bounding volumes is constructed since bounding volumes can be added to each node of a CSG tree. A bounding volume can be easily computed for each primitive and stored in the respective leaf node. Additionally, bounding volumes can be computed in the intermediate nodes by combining the bounding volumes of its child nodes. The bounding volume of the root node encloses the whole object.

In this paper we will use Cameron's S-bounds [1,11,12] to calculate the bounding volumes since S-bounds are usually smaller than ordinary bounds. The upward rule computes the bounding volume of an intermediate node by combining the bounding volumes of the child nodes. The downward rule prunes the bounding volume of a child node due to the bounding volume of its parent node. A sequence of upward and downward rules is applied until a stable configuration is reached. This may even result in nodes with empty bounding volumes. In this case the respective primitive or subtree does not contribute to the entire object. We are using bounding spheres as well as axis-aligned bounding boxes to enclose the CSG objects since these volumes enable fast interference detection.

## OVERVIEW OF THE SYSTEM

Dynamic simulation treats objects as masses moving under the influence of forces and torques (Wilhelms et al. [13]). Motion is computed by solving the equations of motion describing the objects. The dynamic state at a given time step is used to compute the dynamic state at a small time increment later. These computations have to be performed at least for each frame, sometimes it is necessary to have time increments less than a frame duration. After each update of the dynamic state of the objects that results in new positions and orientations, collision detection has to be performed. If a collision has been detected, the collision point and the time and type of contact have to be determined. Using collision response techniques, the dynamic state of the affected objects after the impact can be calculated.

Solving the equations of motion results in new positions and orientations of the objects. Based on these parameters collision, detection has to be performed. In an animated scene some objects will exist that do not move. These are static objects that do not change their position and orientation during the animation. Thus we distinguish between static objects and dynamic objects that are moving around. Static objects do not have to be tested for collision among one another. On the other hand dynamic objects may collide with static objects. Therefore each static object has to be tested for interference with each dynamic object which is called static collision detection. Each dynamic object may collide with another dynamic object. Detecting collisions among pairs of dynamic objects is called dynamic collision detection.

At first we perform static collision detection and afterwards we perform dynamic collision detection. Our collision detection method for CSG objects works on two objects. Therefore all possible combinations of two objects (static-dynamic, dynamic-dynamic) have to be created which are then tested for interference. Collision detection can stop if the first pair of interfering

objects has been determined. However, if a collision between two other objects has occurred at some instant of time earlier, this will not be detected. Such a collision will probably be detected after the next evaluation of the equations of motion. However, the correct solution is to check the other objects for further collisions as well and to select the earliest one if multiple collisions should occur.

Collision detection for two CSG objects has to be performed in a common coordinate system. Usually objects are modelled in a local coordinate system and then transformed to the world coordinate system. At each frame the dynamic simulation computes new positions and orientations for a moving object which are represented in a new transformation matrix. To be able to perform collision detection all objects have to be transformed to the global coordinate system. However, this results in some problems with bounding volumes. We have decided to use spheres and axis-aligned boxes as bounding volumes because they can be tested for interference most easily. Whether two spheres interfere, can be detected by comparing the distance of the centers with the sum of the radii. Detecting an interference of two axis-aligned boxes reduces to some minimum and maximum operations. Rotating a sphere does not change its shape, but rotating an axis-aligned box will yield a box that is no longer axis-aligned. Thus transformation of a CSG object with axis-aligned bounding boxes will result in an object with non axis-aligned bounding boxes. Since interference detection of boxes that are not axis-aligned is much more complicated than interference detection of axis-aligned boxes, new axis-aligned bounding boxes have to be computed after each transformation. The transformed bounding boxes have to be enclosed by axis-aligned boxes, but unfortunately this will result in larger bounding boxes. We might as well use only spheres as bounding volumes, but boxes usually enclose an object more tightly than spheres. Therefore we have decided to use both of them, although we do not have the tightest bounding volumes that are possible.

After performing the dynamic simulation and computing new bounding volumes for the transformed objects, collision detection is applied. As already mentioned we have decomposed collision detection into three stages which will be treated in detail in the following sections. In the first stage we locate the region of a collision between each combination of two objects. If this region is empty, there is no collision and we can stop. Otherwise a collision may occur and we have to test those primitives in the collision region whether they collide. Therefore in the second stage we are using spatial subdivision to create an octree-like data structure (ECO) for the CSG tree of each object. The ECOs are intersected to find pairs of primitives – one of each colliding object. If no pair can be found the CSG objects do not collide and we can stop. Otherwise we have to determine the collision point and the time and type of contact for any pair of primitives in the third stage. If a collision has been found, collision response has to be performed. Figure 1 gives an overview of the flow of control in this system.

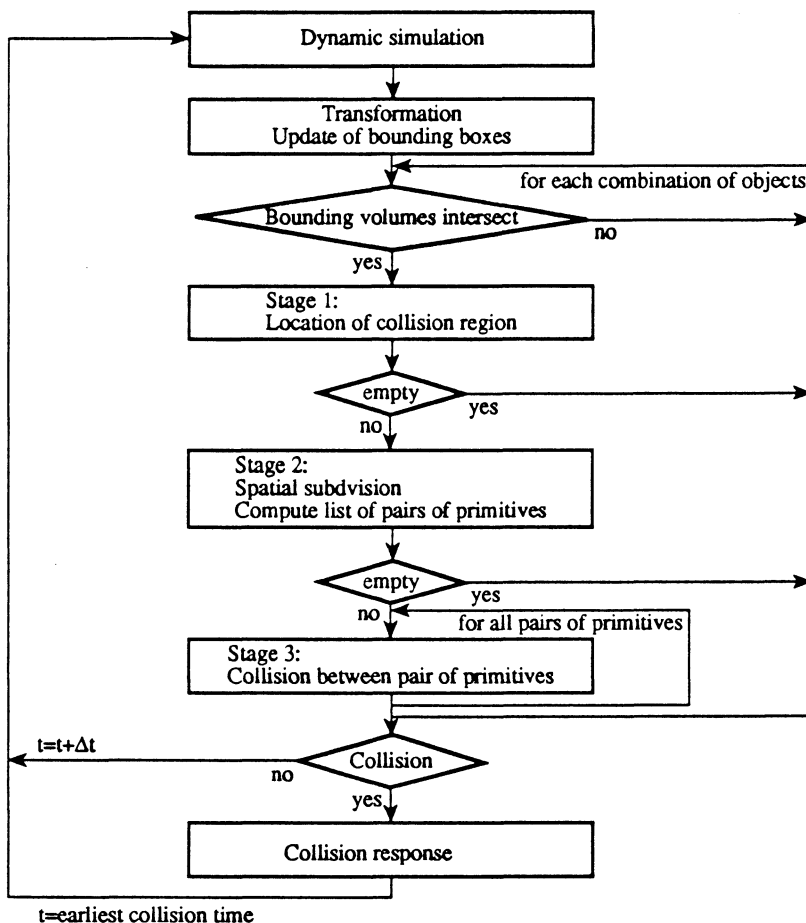


Figure 1. System overview

## LOCATING THE COLLISION REGION

If the bounding volumes of two objects do not interfere, the objects which are enclosed by these volumes obviously do not interfere as well. Therefore these two objects do not collide and collision detection stops for this pair of objects. However, if the bounding volumes interfere, further investigations have to be performed since there is at least a good chance that a collision might occur. Detecting collisions among two CSG objects can be accelerated using S-bounds. As Cameron [1,11] already pointed out two CSG trees can be combined by an intersection operation, thus forming a new CSG tree where copies of the original objects are the child nodes of the root node. The bounding volumes of the root node can be computed by combining the bounding volumes of its child nodes. The S-bounds of the new CSG tree are computed by applying downward and upward rules successively until a stable configuration is reached. If the bounding volume of the entire object, i.e. the bounding volume of the root node of the CSG tree, is the empty set, the two

original objects do not interfere. Again these objects do not collide and collision detection stops.

If the bounding volume of the root node is not empty, at least the bounding volumes of the CSG objects intersect. A prospective collision will have to occur in the enclosed region and therefore the collision point has to be inside this volume. Applying the S-bounds algorithm to the new CSG tree may have resulted in some nodes (intermediate as well as leaf nodes) with empty bounds although the bounding volume of the root node is not empty. Such nodes and their respective primitives do not contribute to the combined object. Empty bounds will seldom occur in common objects since no animator will model parts of an object that are cut away afterwards. However, since we only have combined two distinct objects symbolically to get the intersection object, the bounding volumes in several nodes will be empty. Hopefully the intersection region is small and many bounds are empty. The redundant nodes will not have to be considered for further collision detection and therefore they can be pruned from the CSG tree.

Since the two objects that shall be examined for a collision have been combined to a single CSG tree to evaluate the bounding volumes for the intersection region, we can as well remove the intersection node again. We end up with two objects which are modifications of the two original objects. Their bounds have been reduced to the region where an intersection of the S-bounds has been determined and where a collision is likely to occur. Those nodes that entirely lie outside the collision region have been pruned from the CSG trees and therefore both objects only consist of nodes that lie at least partially inside the collision region.

## PAIRS OF COLLIDING PRIMITIVES

In the previous section we have used the S-bounds theory to reduce object space to the so-called collision region where a collision takes place. If the collision region is not empty a collision is likely to occur. We have to analyze the primitives of the CSG objects to determine whether they really collide. The CSG trees of the objects have been already pruned in the previous stage and thus they only consist of those primitives which at least partially interfere the collision region. Since we use tight bounds we can assume that these pruned CSG trees are not very large and consist only of a few primitives. Therefore we would have to apply collision detection among all pairs of primitives – one from each object. If more than one collision occurs, we have to consider the collision which happens first in the following collision response step. The next section will present some algorithms to determine collisions among such pairs of primitives.

Although the CSG trees have been pruned and their internal structure is simple, the corresponding objects can still have quite complicated shapes. Just think of the 2D object in Figure 2 and its CSG tree. If two primitives have been combined by union they contribute with their whole volume to the object, but if they have been combined by intersection only those parts of their volume contribute to the object which lie in both primitives (Figure 3). The surface of  $A \cup B$  and  $A \cap B$  is made up of parts of the surface of both primitives. If a primitive B has been subtracted from another primitive A only that part of A contributes to  $A - B$  which is not in B. The surface of  $A - B$  is made up of some

part of A and some part of B, but B's surface is complemented due to the difference operation (Figure 3). Since these combinations are applied recursively in CSG modelling, the resulting volume and surface can be quite complicated. Therefore not every primitive in the CSG tree contributes to the object with its entire volume and surface and applying collision detection to such pairs of primitives would lead to incorrect results.

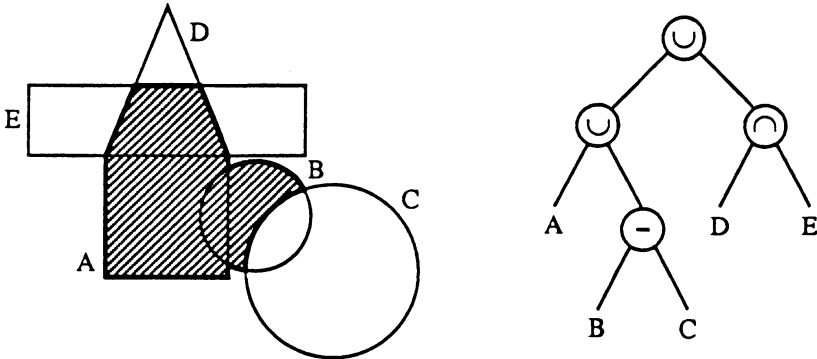


Figure 2. 2D example of a CSG object and its CSG tree

To perform collision detection we need to know the shape of the CSG object. One approach would be to spatially subdivide the objects with each voxel being full or empty (i.e. inside or outside). Collision detection could be easily applied, since if two voxels at the same location in both objects are indicated as full there is an intersection and they collide. In computer animation we have to perform collision response as well and therefore we need to determine the collision point and the type of contact which cannot be calculated from this kind of spatial subdivision. It fails for our needs because the surface of the object has been approximated by voxels, but for collision detection we need to know the exact shape. However, spatial subdivision in itself is quite useful since it helps to reduce the complexity of the object to a collection of simpler parts. In the following subsection we present a spatial subdivision scheme for a CSG object which enables us to determine pairs of primitives which have to be tested for collisions. As another advantage of this subdivision the spatial relationship of the voxels can be used to decide if some of the primitives do not have to be tested for collision.

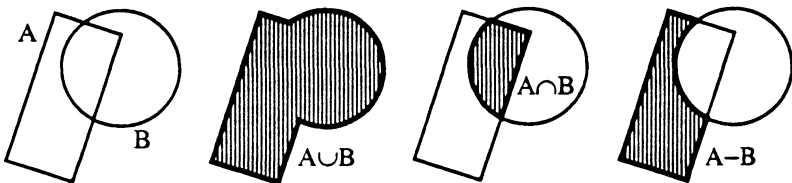


Figure 3. Set operations applied to 2D primitives

### Spatial Subdivision

As mentioned earlier an ordinary spatial subdivision of a CSG tree, like an octree with full or empty voxels, is not sufficient for our needs because the surface information gets lost due to the approximation. We would require an

octree-like description that exactly represents a CSG tree and therefore also includes information about the surfaces. As pointed out by Samet [14] several approaches exist for exactly representing polyhedral objects in an octree-like data structure that he calls PM octree. A leaf node of a PM octree can be full or empty, but it can also contain at most one face or one edge or one vertex. For the exact evaluation of CSG trees Samet [14] mentions the PM-CSG tree which is based on the work of Wyvill and Kunii [15,16]. Their approach is a variant of the PM octree in the way that their leaf nodes which are not full or empty refer to a single primitive object instead of a face, edge or vertex. They do not use all CSG operations since they restrict only to the subset of union of disjoint operands and difference. The PM-CSG tree distinguishes between five types of leaf nodes: full, empty, boundary between empty space and one primitive object, boundary between full space and one primitive object, and volume of space less than the limit of resolution.

For the aim of collision detection of CSG objects we extend the PM-CSG tree of Wyvill and Kunii [15]. According to the extended octree for representing polyhedral objects (Brunet et al. [17]) we will refer to this extension of an octree to represent an CSG tree exactly as extended CSG octree (ECO) (Zeiller [18]). In contrast to the CSG tree that is used by Wyvill and Kunii the set operations are not limited. Therefore we have to consider union, intersection and difference of two operands when we create the extended CSG octree. In extension to the PM-CSG tree we have to store more information of the underlying CSG object in the octree nodes. Again we have five types of leaf nodes:

1. A *full* node is completely embodied in the object. The corresponding voxel can be contained in more than one primitive (totally or partially). A *full* node refers to a list of primitives in which it is contained.
2. An *empty* node is not contained in the object.
3. A *p-surface* (positive surface) node is partially contained in the object. The full part of it is contained in only one primitive, the other part is empty. Therefore the boundary between full space inside the object and empty space is made up of a fraction of the primitive's surface. The *p-surface* node refers to this primitive.
4. A *n-surface* (negative surface) node is partially contained in the object. The full part of it is contained in only one primitive, the other part is empty. In contrary to a *p-surface* node the boundary between full space inside the object and empty space is made up of the surface of another primitive that has been subtracted. The *p-surface* node refers to the (positive) primitive contributing to the volume and to the (negative) primitive whose surface determines the boundary.
5. A *mixed* node is at the lowest level of resolution. More than one primitive contributes to it and therefore it should be decomposed, but due to the limited resolution it is not further subdivided. A *mixed* node refers to a list of primitives in which the voxel is partially contained.

Constructing an extended CSG octree is straightforward. The CSG tree is traversed in depth first manner. If a primitive object (leaf node of the CSG tree) is encountered, a single ECO node representing this primitive is created. At an intermediate node the two branches are evaluated and then combined according to the set operation. Tables 1, 2 and 3 represent the decision tables

for the union, intersection and difference operation. If a non grey node has to be decomposed into its eight subvoxels the only thing we have to decide is whether the subvoxels are completely inside (full), partially inside (p-surface, n-surface) or outside (empty). Since it is sufficient to perform point tests of the vertices of the subvoxels and to determine the node type from the number of vertices inside the object, this spatial subdivision can be applied to each object made up of any primitive shape where we can decide whether a point is inside or outside.

|                  | <i>grey</i> | <i>full</i> | <i>empty</i> | <i>p-surface</i> | <i>n-surface</i> | <i>mixed</i> |
|------------------|-------------|-------------|--------------|------------------|------------------|--------------|
| <i>grey</i>      | grey        | full        | grey         | grey             | grey             | –            |
| <i>full</i>      | X           | full        | full         | full             | full             | full         |
| <i>empty</i>     | X           | X           | empty        | p-surface        | n-surface        | mixed        |
| <i>p-surface</i> | X           | X           | X            | grey             | grey             | mixed        |
| <i>n-surface</i> | X           | X           | X            | X                | grey             | mixed        |
| <i>mixed</i>     | X           | X           | X            | X                | X                | mixed        |

Table 1. Decision table for union of two ECOs (commutative)

|                  | <i>grey</i> | <i>full</i> | <i>empty</i> | <i>p-surface</i> | <i>n-surface</i> | <i>mixed</i> |
|------------------|-------------|-------------|--------------|------------------|------------------|--------------|
| <i>grey</i>      | grey        | grey        | empty        | grey             | grey             | –            |
| <i>full</i>      | X           | full        | empty        | p-surface        | n-surface        | mixed        |
| <i>empty</i>     | X           | X           | empty        | empty            | empty            | empty        |
| <i>p-surface</i> | X           | X           | X            | grey             | grey             | mixed        |
| <i>n-surface</i> | X           | X           | X            | X                | grey             | mixed        |
| <i>mixed</i>     | X           | X           | X            | X                | X                | mixed        |

Table 2. Decision table for intersection of two ECOs (commutative)

|                  | <i>grey</i> | <i>full</i> | <i>empty</i> | <i>p-surface</i> | <i>n-surface</i> | <i>mixed</i> |
|------------------|-------------|-------------|--------------|------------------|------------------|--------------|
| <i>grey</i>      | grey        | empty       | grey         | grey             | grey             | –            |
| <i>full</i>      | grey        | empty       | full         | n-surface        | p-surface        | mixed        |
| <i>empty</i>     | empty       | empty       | empty        | empty            | empty            | empty        |
| <i>p-surface</i> | grey        | empty       | p-surface    | grey             | grey             | mixed        |
| <i>n-surface</i> | grey        | empty       | n-surface    | grey             | grey             | mixed        |
| <i>mixed</i>     | –           | empty       | mixed        | mixed            | mixed            | mixed        |

Table 3. Decision table for difference of two ECOs (not commutative)

The Extended CSG Octree has to be created for both of the pruned CSG objects that are tested for a collision. Since they can only collide within the collision region the ECO has to be computed for the corresponding region. A bounding cube of the collision region (an axis-aligned box) is created and for each object the ECO is created by decomposing this cube for those parts within the collision region.

### Determining Pairs of Primitives as Collision Candidates

The extended CSG octree representation of the two objects is now used to detect whether the objects collide. If they do collide those pairs of primitives are determined which have to be tested for a collision to be able to compute the collision point. Collision detection for two extended CSG octrees is very similar to combining them using an intersection operation. However, we do not



generate a new ECO, we only report if they collide and create a list of pairs of (probably) colliding primitives. Table 4 is the decision table of the collision operation applied to two extended CSG octrees.

|                  | <i>grey</i> | <i>full</i> | <i>empty</i> | <i>p-surface</i> | <i>n-surface</i> | <i>mixed</i> |
|------------------|-------------|-------------|--------------|------------------|------------------|--------------|
| <i>grey</i>      | branches    | branches    | skip         | branches         | branches         | –            |
| <i>full</i>      | X           | add pairs   | skip         | add pairs        | add pairs        | add pairs    |
| <i>empty</i>     | X           | X           | skip         | skip             | skip             | skip         |
| <i>p-surface</i> | X           | X           | X            | add pair         | add pair         | add pairs    |
| <i>n-surface</i> | X           | X           | X            | X                | add pair         | add pairs    |
| <i>mixed</i>     | X           | X           | X            | X                | X                | add pairs    |

Table 4. Decision table of collision operation (commutative)

In a *branches* case no decision can be made since at least one of the nodes is a grey node. If necessary, the larger node has to be decomposed into its subvoxels and the corresponding subvoxels of both nodes have to be compared. *Skip* means that no action has to be taken. In an *add pair* or *add pairs* case the primitives of both nodes have to be combined to a pair and inserted into the list of collision candidates. Of course, each pair of primitives has to be inserted only once. *N-surface* nodes have to be treated in a special way because they consist of two primitives with one of them complemented. Thus for those nodes an element in the list of collision candidates has to contain both of these primitives of this object and the primitive of the other object. Therefore the list of candidates is made up of three kinds of elements:

- p-pairs with two primitives with positive surface
- n-pairs with one complemented surface and one positive surface
- nn-pairs with two complemented surfaces

The collision operation is applied to the two extended CSG octrees to detect those voxels that are covered by primitives in both objects. If no such voxels can be found, i.e. the combination always ends in a *skip* decision, no primitives will be added to the list of collision candidates. If this list is empty after the collision operation has been finished, no collision has been found and collision detection for these two objects stops. Otherwise we have to perform a third stage where collisions among the primitive shapes are detected.

## DETECTING COLLISIONS AMONG PRIMITIVES

In the previous stage we have determined those pairs of primitive shapes that have to be further investigated. In the third stage of our collision detection method we have to consider only those pairs of primitives where a collision is likely to occur. Due to CSG modelling the list of primitive pairs consists of primitives with positive surface and of primitives whose surface has been complemented because a difference operation has been applied. Of course, these primitives have to be treated in different ways. We will first concentrate on collision detection of primitives with positive boundary and explain an equivalent method for complemented primitives afterwards.

### Collision Quantities

In computer animation based on dynamic simulation collision detection has to be followed by collision response. Therefore it is not sufficient to detect a

collision (yes or no), but the collision point and the time and type of contact have to be determined as well. The collision point can be defined as the point where the two primitives come into contact. However, when we detect a collision the primitives already interfere. A good heuristic in this case is to consider a so-called penetration point to be the point of greatest interpenetration (Moore and Wilhelms [2]). If the penetration point is very close to the surface, the primitives are considered to be in contact. Otherwise the collision point has to be determined by evaluating the geometry of the intersection region (see also Hahn [3] for polyhedral primitives). The time of contact can be calculated as the instant of time when the collision point lies on the surface of both primitives. We also have to determine the type of contact and the plane of collision (Hahn [3]). If quadric surfaces collide, we will gain a planar contact in most cases, since the surface normal can be derived at the collision point. It will rarely happen that two 'edges', an 'edge' and an apex or two apexes will touch. However, we are also using regular convex polyhedra as primitives and when they collide edge-edge, edge-vertex and vertex-vertex contacts may arise. Thus if we can derive the surface normal of one of the involved primitives at the collision point, this normal determines the collision plane. Otherwise the normal of the collision plane is computed by averaging the adjacent surface normals.

### Intersection of Primitives

To be able to detect a collision and to compute the quantities mentioned above, the primitives have to be intersected. The primitive shapes we are using are natural quadrics and Platonic solids, i.e. regular convex polyhedra. Therefore we can distinguish between three types of intersections:

- polyhedron-polyhedron
- quadric-quadric
- quadric-polyhedron

For the aim of collision detection, we do not have to determine the intersection curve generated from the two primitives if they really intersect. It is sufficient to calculate the quantities necessary for collision response. Therefore we can apply reduced versions of intersection algorithms.

Polyhedron-Polyhedron Intersection Several methods for detecting collisions among polyhedral objects have been presented in the literature (e.g. Moore and Wilhelms [2], Hahn [3], Bouma and Vanecsek Jr. [4], Lin and Canny [5]). We have chosen an approach based on a 3D analogy to Cyrus-Beck Clipping as is has been introduced by Moore and Wilhelms [2], but any convenient method can be used.

Quadric-Quadric Intersection The intersection of two quadric surfaces can be performed algebraically using the algebraic representations of quadrics or by geometric reasoning with the shape of the primitives (Miller [19]). Both methods are using an exact model of the surface, but algebraic representations are based on equations, whereas geometric representations directly store quantities such as size and direction vectors. In the algebraic approach any quadric surface is represented as the set of points satisfying the second degree polynomial equation

$$Ax^2 + By^2 + Cz^2 + 2Dxy + 2Eyz + 2Fzx + 2Gx + 2Hy + 2Jz + K = 0$$

with suitable values of the coefficients  $A...K$ . Therefore the intersection can be performed in a surface type independent way. We get an intersection curve

which is either a point, a line or one or two curve branches. These branches will either be conic sections or nonplanar space curves of polynomial degree 4 (Miller [19]). However, the drawback of this technique is that is not very easy to compute the collision quantities out of this curve.

On the other hand we can perform geometric reasoning based on the shape of the natural quadrics in 3D space which is determined by their geometric representation. This technique is most often faster, but unfortunately a special intersection procedure has to be implemented for each combination of primitive types. We completely loose the generality of the algebraic approach and have to write several intersection procedures if we want to add a new type of primitive. Despite these drawbacks we are using this technique whenever it is possible. In comparison to the algebraic approach, implementing this method is much more work. Several configurations of the two primitives in 3D space have to be considered and each of them requires special treatment. Figure 4 presents an example of those configurations that arise if a cylinder and a sphere are tested for collision (a. no collision, b. contact, c. collision). Both primitives have been projected onto a plane passing through the axis of the cylinder and the centre of the sphere. After we have derived the configuration we can easily compute the collision quantities. Usually determining the type of configuration can be performed very fast, thus this approach is preferable in terms of computational efficiency.

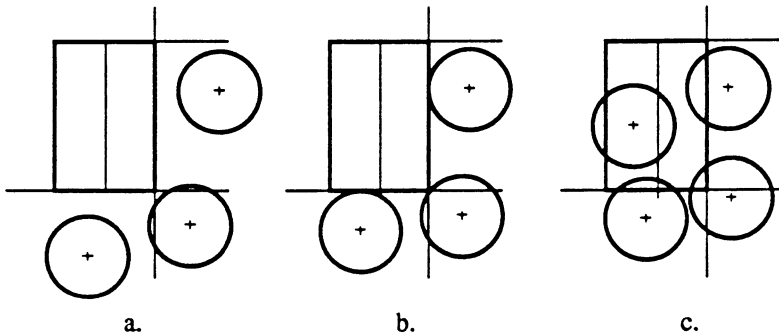


Figure 4. Cylinder and sphere

Geometric reasoning can be applied to the following pairs of primitives: sphere-sphere, sphere-cylinder, sphere-cone, cylinder-cylinder and cylinder-cone. We did not manage to use this method for cone-cone pairs as well, since despite their simple form cones cannot be treated in the same manner. For those pairs of primitives that consist of two cones we have to use a different approach. As in the geometric reasoning method we try to detect the simple configurations of the cones (e.g. an apex is inside or on the surface of the other cone) first. If we do not find such a simple configuration we apply a method that is based on the geometric approaches to intersect quadric surfaces of Miller [20]. The parametric form of one of the cones is substituted into the implicit equation of the second cone and we get the parametric form of the cone-cone intersection curve. Afterwards we have to calculate the so-called critical points on this curve. These points determine the straight lines on the surfaces of the cones with greatest interpenetration which can be used to calculate the collision point.

**Quadric-Polyhedron Intersection** The intersection of quadric and polyhedral primitives is similar to the intersection of two polyhedral objects. First the vertices of the polyhedron are tested whether they are inside or outside the quadric surface. If at least one vertex has been found which is inside the quadric, the adjacent edges are used to derive the geometry of the intersection region and the collision quantities. Otherwise an edge might pierce the quadric and the edges of the polyhedron are intersected with it. Again the collision quantities have to be derived after an intersection has been discovered. If still no collision has been detected, the bounding planes of the polyhedron have to be intersected with the quadric to determine whether the primitives collide.

### Positive and Negative Intersection

So far we have only considered positive primitives that have been added to the CSG object by union and intersection operation. Due to the difference operation we have to deal also with primitives that have been subtracted from the CSG tree thus yielding n-surface nodes in the extended CSG octree. Therefore the list of pairs of primitives consists of n-pairs and nn-pairs as well which contain complemented surfaces. These pairs have to be treated in a special way. Combining the primitives A and B by a difference operation ( $A-B$ ) means that the resulting object consists of those parts of A that do not lie inside of B. The surface of  $A-B$  is made up of some part of A and some (complemented) part of B. If collision detection between a positive primitive C and a complemented primitive has to be performed, a collision has to be reported only if the positive primitive C intersects the surface of the complemented primitive B and that part of C which is outside B lies inside primitive A. Figure 5 demonstrates a 2D example of this relationship (a. collision, b. no collision). Therefore the algorithms for intersecting primitives, that have been presented above, have to be extended to deal with complemented primitives as well.

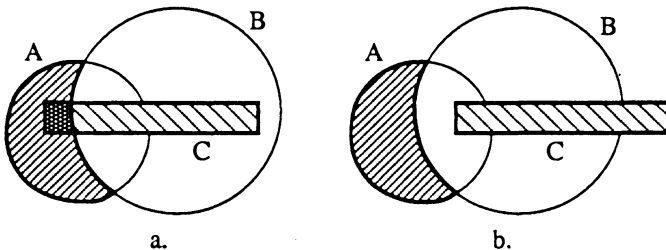


Figure 5. Collision between complemented B ( $A-B$ ) and positive C

### FUTURE WORK

We have restricted the primitive shapes of the CSG objects to be natural quadrics and Platonic solids. We could treat more complex shapes easily in the spatial subdivision stage, but using other shapes would require further intersection procedures for each new combination of primitives in the third stage where collisions among primitives are detected. Perhaps a general and fast algebraic approach can be found which can be used to calculate the collision point and the time and type of contact as well.

Primitive shapes in a CSG tree can be rotated, translated and scaled. Most CSG modelling systems enable different scaling factors in the direction of each coordinate axis. However, this would transform a sphere to an ellipsoid, a cylinder with circular cross-section to a cylinder with elliptical cross-section, etc. Therefore new shapes arise and new geometric intersection procedures have to be developed, too. On the other hand some of them may have to be solved using Miller's geometric approach (Miller [20]) because their shapes are too complicated for geometric reasoning. In the present system scaling factors have to be identical in all directions. Future systems should allow the use of different scaling factors which is another reason for investigating more general intersection methods for various primitives.

In the first stage of our collision detection algorithm we have used bounding volumes to locate the collision region. Out of the variety of possible bounding volumes we have chosen spheres and axis-aligned boxes due to their simplicity in detecting intersections. We have not performed a detailed analysis of using only bounding spheres, only bounding boxes and the combination of both until now. Other bounding volumes, that enclose the object more tightly but are more costly in intersection detection, should be investigated, too.

## CONCLUSION

We have presented a method for exactly detecting collisions among objects modelled by the constructive solid geometry paradigm. The detection of collisions between two CSG objects is decomposed into three stages. In the first stage we locate the collision region using bounding volumes. S-bounds with bounding spheres and axis-aligned bounding boxes have proven to be well suited for this problem. If the collision region is empty we can stop collision detection for these objects. Otherwise we have to test those primitives in the collision region whether they collide. We can exactly detect collisions only among two primitive. Therefore we decompose the two CSG objects into pairs of primitive shapes – one of each object. Since CSG objects may have complex shape due to the recursive application of union, intersection and difference operations, we split this problem into smaller subproblems. Both CSG objects are spatially subdivided to create an octree-like data structure for the CSG tree. These extended CSG octrees are intersected to detect those cells that are covered by primitives in both objects. If no such cells can be found, collision detection for these two objects stops. Otherwise those pairs of primitives that may collide are determined. In the third stage we have to detect collisions among all such pairs of primitive shapes. If they collide, we calculate the collision point as well as the time and type of contact.

## REFERENCES

1. Cameron, S. 'Efficient Intersection Tests for Objects Defined Constructively' *The International Journal of Robotics Research*, Vol.8, No.1, pp. 3 - 25, February 1989.
2. Moore, M. and Wilhelms, J. 'Collision Detection and Response for Computer Animation' *Computer Graphics*, Vol.22, No.4, pp. 289 - 298, August 1988.
3. Hahn, J. 'Realistic Animation of Rigid Bodies' *Computer Graphics*, Vol.22, No.4, pp. 299 - 308, August 1988.



4. Bouma, W. and Vanecek Jr., G. 'Collision Detection and Analysis in a Physically Based Simulation', pp. 191 - 203, *Proceedings of Second Eurographics Workshop on Animation and Simulation*, Vienna, Austria, 1991.
5. Lin, M. and Canny, J. 'Efficient Collision Detection for Animation', *Proceedings of Third Eurographics Workshop on Animation and Simulation*, Cambridge, UK, 1992.
6. Baraff, D. 'Curved Surfaces and Coherence for Non-penetrating Rigid Body Simulation' *Computer Graphics*, Vol.24, No.4, pp. 19 - 28, August 1990.
7. Von Herzen, B., Barr, A. and Zatz, H. 'Geometric Collisions for Time-Dependent Parametric Surfaces' *Computer Graphics*, Vol.24, No.4, pp. 39 - 48, August 1990.
8. Bickerstaff, M. and Hellestrand, G. 'A Highly Parallel Architecture for Real Time Collision Detection in Flight Simulation' *Computers & Graphics*, Vol.15, No.3, pp. 355 - 363, 1991.
9. Baraff, D. 'Analytical Methods for Dynamic Simulation of Non-penetrating Rigid Bodies' *Computer Graphics*, Vol.23, No.3, pp. 223 - 232, July 1989.
10. Mäntylä, M. *An Introduction to Solid Modeling* Computer Science Press, 1988.
11. Cameron, S. 'Efficient Bounds in Constructive Solid Geometry' *IEEE Computer Graphics & Applications*, Vol.11, No.3, pp. 68 - 74, May 1991.
12. Cameron, S. and Yap, C. 'Refinement Methods for Geometric Bounds in Constructive Solid Geometry' *ACM Transactions on Graphics*, Vol.11, No.1, pp. 12 - 39, January 1992.
13. Wilhelms, J., Moore, M., Skinner, R. 'Dynamic animation: interaction and control' *The Visual Computer*, Vol.4, No.6, pp. 283 - 295, December 1988.
14. Samet, H. 'Volume Data' Chapter 5, *The Design and Analysis of Spatial Data Structures*, pp. 315 - 376, Addison Wesley, Reading Massachusetts, 1990.
15. Wyvill, G. and Kunii, T. 'A functional model for constructive solid geometry' *The Visual Computer*, Vol.1, No.1, pp. 3 - 14, July 1985.
16. Wyvill, G., Kunii, T. and Shirai, Y. 'Space Division for Ray Tracing in CSG' *IEEE Computer Graphics & Applications*, Vol.6, No.4, pp. 28 - 34, April 1986.
17. Brunet, P., Navazo, I. 'Geometric Modelling Using Exact Octree Representation of Polyhedral Objects' in EUROGRAPHICS'85 (Ed. Vandoni, E.), pp. 159 - 169, *Proceedings of EUROGRAPHICS'85*, Nice, France, 1985. Elsevier, Amsterdam, 1985.
18. Zeiller, M. 'Extended CSG Octrees'. In preparation.
19. Miller, J. 'Analysis of Quadric-Surface-Based Solid Models' *IEEE Computer Graphics & Applications*, Vol.8, No.1, pp. 28 - 42, January 1988.
20. Miller, J. 'Geometric Approaches to Nonplanar Quadric Surface Intersection Curves', *ACM Transactions on Graphics*, Vol.6, No.4, pp. 274 - 307, October 1987.