Computer Animation Project Report

Breaking a Brick Wall

Dany Alejandro Cabrera
University of Victoria
Victoria, BC
dcabrera@uvic.ca

ABSTRACT

This documents serves as project report for the Computer Animation course, consisting of a simulation of a brick wall. The main challenge of this project is implementing a rigid body and collision system from scratch, in this case based on cuboid shaped colliders. Given the nature of the problem, I adopted an Oriented Bounding Box (OBB) approach, which simplifies the computations required for detecting collisions and finding the collision point to fire a collision response. Although deemed simple at early stages, the project proved challenging to implement and could easily scalate into a larger project depending on the physical considerations and optimizations included. The resulting simulation is able to demonstrate collisions between bodies, but a more realistic animation would require the implementation of additional physics features like resistance, energy damping, material density and resting conditions.

KEYWORDS

Collision detection, Oriented Bounding Box, Rigid Body Engine

ACM Reference Format:
Dany Alejandro Cabrera. 2018. Computer Animation Project Report: Breaking a Brick Wall. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, Article 4, 7 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Rigid Body Dynamics are traditionally applied to simulate the tumbling behavior of solid 3D objects with additional considerations regarding their volume and density, and thus will have more complex motion equations compared with simple points in space [8], in particular because of the additional challenges introduced with edges and faces. One of the intricacies of dealing with rigid bodies is accurately detecting collisions and generating a realistic collision response, which will increase in difficulty as we handle more complex shapes and increase the number of objects.

In this project, I intended to explore collision detection on rigid bodies by implementing a simple dynamic collision detection system to perform a real-time simulation of a brick wall being destroyed. My initial proposal included multiple considerations to make the results look more realistic, including the unevenness of density in real materials and other physical properties of real-world bricks; However, the sole implementation of a rigid body collision engine from scratch proved to be a complete challenge on its own.

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1.1 Problem Description

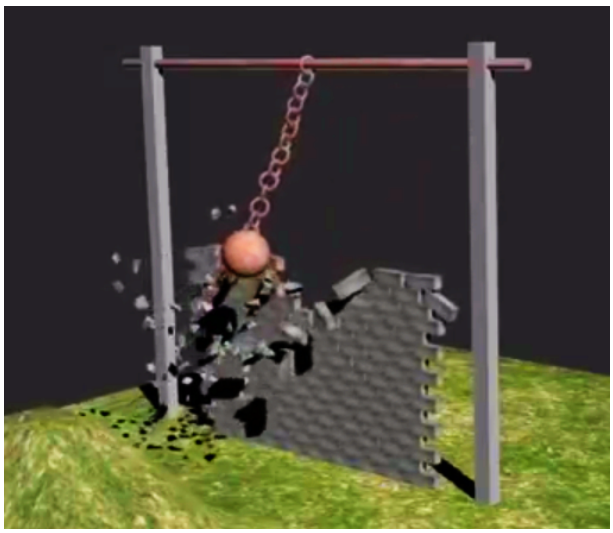


Figure 1: A ball breaks a brick wall (N-bodies collisions example [6]).

This project is inspired in the *N-bodies collisions* final project example provided during the course's laboratory section. In this animation, a ball suspended from a chain breaks a brick wall; some of the bricks stay in place, while others fly away in pieces and fall into the ground (Figure 1 presents a representative animation frame for illustration purposes). To control the project scope, I initially proposed to omit the chain (shoot the ball) and consider a flat ground surface.

Breaking a wall of bricks involves simulating numerous simple-shaped rigid bodies (boxes) colliding with each other under the influence of gravity. This problem is more complex than the collision of a single rigid body against a fixed ground or the collision of 2 rigid bodies because we expect a high number of bodies to closely interact with each other at the same time: each brick's motion will depend on the sum of multiple forces and its collision response will trigger new collisions. Sometimes the usual collision response (bouncing in the opposite direction) will be impossible to perform due to the brick already being in contact with other bodies (or

the ground), but forces should still propagate through the whole system. In practice, the corresponding rigid body engine should also handle performance considerations since the more rigid bodies in our system, the more bodies we have to perform collision detection on ($O(n^2)$ pairs).

1.2 Previous Work

Collision detection has been extensively studied not only in computer graphics, but also computational geometry and robotics. In the case of early physically-based animations, [7] presents a simulation system for rigid bodies that detects collisions and clearly explains the problems of detecting and reacting to multiple collisions in the same instant and general complex contact geometry (modeled as a graph). [9] discusses different methods for detecting and responding to collisions (including the use of springs in the response). [1] discusses collision between bodies in resting contact. [11] discusses performance in real-time simulations including the high cost of collision detection, and proposes simple and practical solutions like the use of what we currently know as a collider[1]. [4] describes collision solving in one of the early real-time simulation systems that worked on "commercial" 1987 workstations.

The topic has been continuously discussed ever since in numerous research works, and nowadays a realistic brick wall breaking animation can be easily created using commercially available animation tools or even open source 3D physics engines. Early simulations would use an AABB (Axis-Aligned Bounding Box) as a simple approach to deal with the limitations on computational power of their time, but resulted insufficient to resemble physically based animations. A more complete approach is discussed in [5] which mentions detecting collisions using the separating axis test (SAT), simulating multiple collisions with *Oriented Bounding Boxes* (OBB) and dealing with the computational complexity of the problem using a tree-like structure; this project initially took this paper as base and proposed to merge additional ideas from the early works mentioned. Many of these ideas are also discussed during the computer animation course and [8] presents a modern and didactic explanation on rigid body collisions that also serves as general guidance.

2 SOLUTION

My initial proposal was to implement a rigid body collisions engine based on the Oriented Bounding Boxes model and tree structure as described in [5] with additional improvements for this specific problem, and loosely implement the mesh fracture ideas described in [10] (specifically the ones regarding mesh division using planes) combined with my own ideas on how clay burnt bricks break to divide them in smaller simple bodies still compatible with the collision engine. However, after starting the project implementation I discovered that the number of computations (and math involved) required for body shapes different from traditional OBBs increases significantly, and the project implementation would only remain feasible if I leveraged the advantages of the symmetry and right angles from OBBs, since for the 6 faces and 12 edges that make up these bodies, their orientations result redundant and allow the consideration of a smaller set of conditions. This fact is discussed in detail in [2] which elaborates on the OBB collision calculations and later became my base reference for implementation purposes.

From a technical perspective, this simple rigid body engine is composed of multiple modules with different responsibilities, including a module for applying constant and impulse forces to the bodies, detecting collisions, producing the collision response and managing the tests between all entities with performance in mind.

2.1 Rigid Body Motion

This system requires a module that updates the position and orientation of multiple rigid bodies and handles their behavior when affected by forces (e.g. gravity). For this purpose, I implemented a simple rigid body motion engine using the basic principles described in [8] able to update force, acceleration, velocity and position in both their translational and rotational versions. An important feature is the ability to apply impulse forces to colliders, by adding the impulse to both linear and angular force (adding $\boldsymbol{F} \times (\boldsymbol{p} - \boldsymbol{X})$ in the later case to account for the distance to the center of mass). This module was designed to also be responsible of general simulation settings (e.g. integration method).

I initially proposed skipping the implementation of a collider entity separate from the actual mesh, by reusing the mesh (brick) vertices for collision detection. However, after learning about manual 3D mesh creation on the implementation phase of this project, I discovered that creating the cuboid mesh requires 24 vertices for 12 triangles, each one with their own UV coordinates and normals, in a non-trivial order. This made evident the need of implementing colliders separately, specially with collision calculations in mind.

In technical terms, a Collider class is used to store the body's mass, velocity, acceleration and forces in both linear and angular terms, as well as a flag to make it "fixed" in space when we want it to ignore external forces (e.g. the floor cuboid). To keep track of rotation I use a quaternion, which already comes implemented in the Ogre3D graphics engine along with their math-related functionality. This object has a mesh attached to it's center so we can reflect all changes into visible objects, by updating the mesh's position and orientation after the frame's calculations are over.

2.2 Collision Detection

To detect the occurrence of a collision between 2 colliders, I implement a Separating-Axis test (SAT), which can work not only for boxes but convex polyhedra as well. In simplified terms, the SAT is based on the separating hyperplane problem, which states that given two convex vertex sets A and B, either the two sets are intersecting or there exists a separating hyperplane P such that A is on one side of P and B is on the other. The SAT leverages this concept by stating that if 2 bodies are not intersecting, the separating hyperplane exists.

A simple way of understanding this concept is by following the algorithm in 2D as described in [5] and considering equation 1 for the Point-Normal form of a plane where $\{ a, b, c \}$ are the normal vector components and $\{ x_0, y_0, z_0 \}$ is the position of a point in the plane's surface. We can implement a simple algorithm for 2D collision detection as described in algorithm 1.

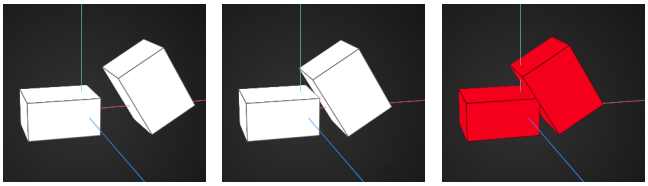
$$0 = a(x - x_0) + b(y - y_0) + c(z - z_0) \tag{1}$$

[1] Colliders define the shape of an object for the purposes of physical collisions, with an invisible shape that is usually a simpler approximation of the original.

Algorithm 1 Simple SAT (2-dimensional)

```
1:  procedure SATDETECTION(A, B)                ▷ Two bodies A and B
2:      count ← 0
3:      numEdges ← Edge count in A and B
4:      for all Edges in A and B do
5:          Get the edge's normal
6:          project A vertice into the normal
7:          determine an interval for A
8:          project B vertice into the normal
9:          determine an interval for B
10:         if intervals overlap then
11:             count ← count + 1
12:     return (count = numEdges)
```

Algorithm 1 can be used to implement a simple 3D collision test between two symmetrical bodies (cuboids) moving along the same base axis, by testing against the bodies vertices' 2D projection. Figure 2 shows this test implementation working with examples of different positions for which the body material is switched to red when a collision is detected. Similar results can be obtained for simple convex bodies as well as long as they are symmetrical along the projected axis.



(a) (b) (c)

Figure 2: Collision test that changes material to red (unshaded polygon rendering mode). (b) Shows no collision on a traditional AABB failure case. (c) Collision correctly detected.

The algorithm in 1 can be easily optimized, for example returning from the function after we detect non-intersecting intervals for the first time. There are also broad phase collision detection methods we can use to reduce computational load; for example, we can precalculate and store the maximum radius (maximum distance from the center point to a vertex) and add an initial test (akin to collision detection between spheres) that compares the distance between both object's middle point and the sum of both radius and discard collision probability if the middle points are too distant. Another option mentioned in [9] is to do AABB detection in the broad phase.

Reference [2] explains in detail how to implement the SAT in 3D. In particular, the table in Appendix A presents the non-intersection tests required for collision detection: 3 for the separating axes obtained from the faces of the collider (for testing against vertice-face collision), 3 for the same case on the other collider, and 9 for all the possible axis obtained from edge intersections presented in terms of the faces' normals, which as preiously mentioned is a clear advantage of OBBs over more irregular colliders. The 15 non-intersection tests are performed every frame on all bodies against all other bodies (a performance challenge addressed by OBB tree) and we store the first test to return true (as a test number, following the same order in the non-intersection tests table), since this information will later be used to find the collision point.

2.3 Collision Response

The basic collision response explained in [8] is to apply an impulse force to both bodies at the point of contact in opposite direction to their speed. Obtaining the intersection point is complex because after a timestep there can be multiple points of intersection (which enters the realm of solid geometry computation). However, if we focus on the first time of intersection, we know that at most cases (vertex-to-vertex, vertex-to-edge, vertex-to-face and edge-to-edge) this point is unique. Reference [2] leverages this fact to present a detailed table (shown in Appendix B) with the coefficients needed to produce the unique points of intersection depending on the particular collision detection test that was fired first; the paper proposes to keep track of the time at which each of the intersection tests trigger for the first time to find the first one fired, but for this project we managed to obtain approximate results by just considering the first intersection test triggered when evaluating all of them in the order they appear in the table.

After obtaining an approximate collision point, I apply a simple impulse to both colliders to serve as collision response, which in turn introduces both a rotational and displacement component to the force vectors of each body. For this project, I'm only applying an impulse of magnitude computed as a factor of the speed of both colliders (obtained empirically).

3 RESULTS

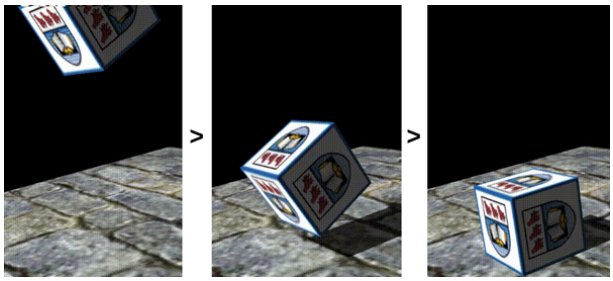


Figure 3: First test: A cube falls into a solid flat ground (sequence of animation frames, full render with stencil modulative shadows).

During the implementation of this project, my first milestone was to simulate a cube falling to the floor as shown in Figure 3. The first tests following the implementation of the collision detection and response modules made evident the need of additional elements for the simulation to look realistic (restitution, density, air resistance, energy loss, etc.); in particular a new module to handle friction. Although I wanted to implement friction as described in [8] by applying tangential forces opposing movement for every vertex in contact with a surface, at this point it was impossible due to time restrictions. Instead, I managed to get results by empirically adding damping components to my motion equations and compensating

by tweaking the gravity value of the system until the box's behavior appeared convincing.

The core test of this project pretended to showcase collisions by animating the destruction of a wall made by stacking bricks (cuboids) atop each other, by throwing another body into it. However, although the bricks were able to collide with each other (making the "stacking" possible at all), with this implementation the wall was unable to stand still for more than a few seconds, and then would spontaneously fall by itself, as shown in Figure 4. This animation can be viewed online at: https://youtu.be/-T7ay-Wlm1U

4 ANALYSIS

After verifying that the bricks were correctly stacked without intersecting each other and with the minimal space possible between them, it's safe to suppose that the wall collapses due to the instability of the system, including rounding errors which can cause iterative collision responses that propagate and resonate until the system "explodes". This is one of the problems that are solved by implementing resting conditions, and I believe lacking this module was the main contributing factor on this behavior. Although [3] describes a much more complete collision system, it mentions a similar instability problem with collapsing brick walls, in its case due to the way stress distributes differently for different brick positions, which seems to suggest that force distribution in the system might also pose challenges that escape the scope of this project.

An attentive look at the animation also reveals the weaknesses of this incomplete system: bricks slide unnaturally due to the improvised "friction" approximation, don't bounce in a natural way, and some move in directions that don't appear logical. Part of this is caused by the high gravity value used to keep the system under control, which combined with the added damping values avoid high speed values, which probably would break the collision system due to still using implicit Euler and not having implemented all of the collision non-intersection tests.

Still, bricks don't appear to be penetrating each other (even though Ogre3D's integrated shader combined with the brick material get in the way of verifying this) and successfully collide with each other.

5 CONCLUSIONS

A simple rigid body dynamics and collision detection and response system was implemented for oriented bounding box (OBB) colliders. Colliders must be implemented as separate entities from meshes even if their vertices appear to share the same position because due to the way 3D meshes are built, it would be inconvenient to rely on the mesh vertices and faces for collision calculations. OBBs have the advantage of being symmetrical and have right angles, which greatly simplify collision related calculations because OBBs can be represented as 3 axis and their corresponding extents. The lack of resting conditions in a collision system can cause appreciable instability problems and unexpected behavior. Collision systems require the implementation of multiple sub-systems for different physics considerations, and their simulation ultimately involves the implementation of a full rigid body dynamics engine, which makes it a challenging problem to solve (and justifies the use of readily-made collision engines before considering the implementation of our own).

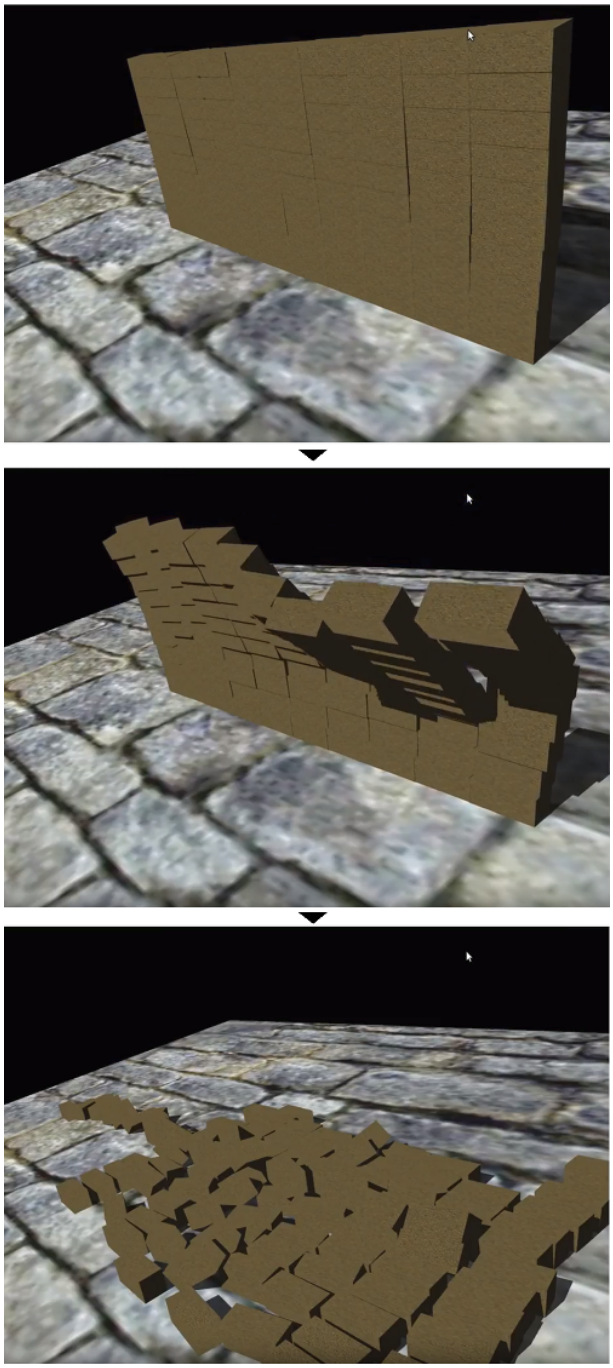


Figure 4: Core test: The brick wall (100 equal bricks, vertically stacked) showcasing collisions between OBBs, collapsing by itself.

APPENDIX A - SEPARATE AXIS TEST NON-INTERSECTION VALUES

$\mathbf{L}$	R_0	R_1	R
$\mathbf{A}_0$	a_0	$b_0\|c_{00}\| + b_1\|c_{01}\| + b_2\|c_{02}\|$	$\|\mathbf{A}_0 \cdot \mathbf{D}\|$
$\mathbf{A}_1$	a_1	$b_0\|c_{10}\| + b_1\|c_{11}\| + b_2\|c_{12}\|$	$\|\mathbf{A}_1 \cdot \mathbf{D}\|$
$\mathbf{A}_2$	a_2	$b_0\|c_{20}\| + b_1\|c_{21}\| + b_2\|c_{22}\|$	$\|\mathbf{A}_2 \cdot \mathbf{D}\|$
$\mathbf{B}_0$	$a_0\|c_{00}\| + a_1\|c_{10}\| + a_2\|c_{20}\|$	b_0	$\|\mathbf{B}_0 \cdot \mathbf{D}\|$
$\mathbf{B}_1$	$a_0\|c_{01}\| + a_1\|c_{11}\| + a_2\|c_{21}\|$	b_1	$\|\mathbf{B}_1 \cdot \mathbf{D}\|$
$\mathbf{B}_2$	$a_0\|c_{02}\| + a_1\|c_{12}\| + a_2\|c_{22}\|$	b_2	$\|\mathbf{B}_2 \cdot \mathbf{D}\|$
$\mathbf{A}_0 \times \mathbf{B}_0$	$a_1\|c_{20}\| + a_2\|c_{10}\|$	$b_1\|c_{02}\| + b_2\|c_{01}\|$	$\|c_{10}\mathbf{A}_2 \cdot \mathbf{D} - c_{20}\mathbf{A}_1 \cdot \mathbf{D}\|$
$\mathbf{A}_0 \times \mathbf{B}_1$	$a_1\|c_{21}\| + a_2\|c_{11}\|$	$b_0\|c_{02}\| + b_2\|c_{00}\|$	$\|c_{11}\mathbf{A}_2 \cdot \mathbf{D} - c_{21}\mathbf{A}_1 \cdot \mathbf{D}\|$
$\mathbf{A}_0 \times \mathbf{B}_2$	$a_1\|c_{22}\| + a_2\|c_{12}\|$	$b_0\|c_{01}\| + b_1\|c_{00}\|$	$\|c_{12}\mathbf{A}_2 \cdot \mathbf{D} - c_{22}\mathbf{A}_1 \cdot \mathbf{D}\|$
$\mathbf{A}_1 \times \mathbf{B}_0$	$a_0\|c_{20}\| + a_2\|c_{00}\|$	$b_1\|c_{12}\| + b_2\|c_{11}\|$	$\|c_{20}\mathbf{A}_0 \cdot \mathbf{D} - c_{00}\mathbf{A}_2 \cdot \mathbf{D}\|$
$\mathbf{A}_1 \times \mathbf{B}_1$	$a_0\|c_{21}\| + a_2\|c_{01}\|$	$b_0\|c_{12}\| + b_2\|c_{10}\|$	$\|c_{21}\mathbf{A}_0 \cdot \mathbf{D} - c_{01}\mathbf{A}_2 \cdot \mathbf{D}\|$
$\mathbf{A}_1 \times \mathbf{B}_2$	$a_0\|c_{22}\| + a_2\|c_{02}\|$	$b_0\|c_{11}\| + b_1\|c_{10}\|$	$\|c_{22}\mathbf{A}_0 \cdot \mathbf{D} - c_{02}\mathbf{A}_2 \cdot \mathbf{D}\|$
$\mathbf{A}_2 \times \mathbf{B}_0$	$a_0\|c_{10}\| + a_1\|c_{00}\|$	$b_1\|c_{22}\| + b_2\|c_{21}\|$	$\|c_{00}\mathbf{A}_1 \cdot \mathbf{D} - c_{10}\mathbf{A}_0 \cdot \mathbf{D}\|$
$\mathbf{A}_2 \times \mathbf{B}_1$	$a_0\|c_{11}\| + a_1\|c_{01}\|$	$b_0\|c_{22}\| + b_2\|c_{20}\|$	$\|c_{01}\mathbf{A}_1 \cdot \mathbf{D} - c_{11}\mathbf{A}_0 \cdot \mathbf{D}\|$
$\mathbf{A}_2 \times \mathbf{B}_1$	$a_0\|c_{12}\| + a_1\|c_{02}\|$	$b_0\|c_{21}\| + b_1\|c_{20}\|$	$\|c_{02}\mathbf{A}_1 \cdot \mathbf{D} - c_{12}\mathbf{A}_0 \cdot \mathbf{D}\|$

Figure 5: Values for R, R_0 and R_1 for the non-intersection test $R > R_0 + R_1$ for two OBBs, as described in [2]. Considering the OBB A described with a center C_0, axes $A_{0..2}$ and extents $a_{0..2}$ and B in the same way with C_1, $B_{0..2}$ and $b_{0..2}$. D is the center separation $C_1 - C_0$. The axes of the first box can be written as linear combinations of axes of the second box in the expression $A_i = c_{i0}B_0 + c_{i1}B_1 + c_{i2}B_2$, with coefficients c_{ij} for their $i, j = 0, 1, 2$ respective axis.

APPENDIX B - COEFFICIENTS FOR UNIQUE POINTS OF OBB-OBB INTERSECTION

L	coefficients
$\mathbf{A}_i$	$y_j = -\sigma\text{Sign}(c_{ij})b_j,\ j = 0, 1, 2$
$\mathbf{B}_j$	$x_i = +\sigma\text{Sign}(c_{ij})a_i,\ i = 0, 1, 2$
$\mathbf{A}_0 \times \mathbf{B}_0$	$x_1 = -\sigma\text{Sign}(c_{20})a_1,\ x_2 = +\sigma\text{Sign}(c_{10})a_2,\ y_1 = -\sigma\text{Sign}(c_{02})b_1,\ y_2 = +\sigma\text{Sign}(c_{01})b_2,$ $x_0 = \frac{1}{1-c_{00}^2}(\mathbf{A}_0 \cdot \mathbf{D} + c_{00}(-\mathbf{B}_0 \cdot \mathbf{D} + c_{10}x_1 + c_{20}x_2) + c_{01}y_1 + c_{02}y_2)$
$\mathbf{A}_0 \times \mathbf{B}_1$	$x_1 = -\sigma\text{Sign}(c_{21})a_1,\ x_2 = +\sigma\text{Sign}(c_{11})a_2,\ y_0 = +\sigma\text{Sign}(c_{02})b_0,\ y_2 = -\sigma\text{Sign}(c_{00})b_2,$ $x_0 = \frac{1}{1-c_{01}^2}(\mathbf{A}_0 \cdot \mathbf{D} + c_{01}(-\mathbf{B}_1 \cdot \mathbf{D} + c_{11}x_1 + c_{21}x_2) + c_{00}y_0 + c_{02}y_2)$
$\mathbf{A}_0 \times \mathbf{B}_2$	$x_1 = -\sigma\text{Sign}(c_{22})a_1,\ x_2 = +\sigma\text{Sign}(c_{12})a_2,\ y_0 = -\sigma\text{Sign}(c_{01})b_0,\ y_1 = +\sigma\text{Sign}(c_{00})b_1,$ $x_0 = \frac{1}{1-c_{02}^2}(\mathbf{A}_0 \cdot \mathbf{D} + c_{02}(-\mathbf{B}_2 \cdot \mathbf{D} + c_{12}x_1 + c_{22}x_2) + c_{00}y_0 + c_{01}y_1)$
$\mathbf{A}_1 \times \mathbf{B}_0$	$x_0 = +\sigma\text{Sign}(c_{20})a_0,\ x_2 = -\sigma\text{Sign}(c_{00})a_2,\ y_1 = -\sigma\text{Sign}(c_{12})b_1,\ y_2 = +\sigma\text{Sign}(c_{11})b_2,$ $x_1 = \frac{1}{1-c_{10}^2}(\mathbf{A}_1 \cdot \mathbf{D} + c_{10}(-\mathbf{B}_0 \cdot \mathbf{D} + c_{00}x_0 + c_{20}x_2) + c_{11}y_1 + c_{12}y_2)$
$\mathbf{A}_1 \times \mathbf{B}_1$	$x_0 = +\sigma\text{Sign}(c_{21})a_0,\ x_2 = -\sigma\text{Sign}(c_{01})a_2,\ y_0 = +\sigma\text{Sign}(c_{12})b_0,\ y_2 = -\sigma\text{Sign}(c_{10})b_2,$ $x_1 = \frac{1}{1-c_{11}^2}(\mathbf{A}_1 \cdot \mathbf{D} + c_{11}(-\mathbf{B}_1 \cdot \mathbf{D} + c_{01}x_0 + c_{21}x_2) + c_{10}y_0 + c_{12}y_2)$
$\mathbf{A}_1 \times \mathbf{B}_2$	$x_0 = +\sigma\text{Sign}(c_{22})a_0,\ x_2 = -\sigma\text{Sign}(c_{02})a_2,\ y_0 = -\sigma\text{Sign}(c_{11})b_0,\ y_1 = +\sigma\text{Sign}(c_{10})b_1,$ $x_1 = \frac{1}{1-c_{12}^2}(\mathbf{A}_1 \cdot \mathbf{D} + c_{12}(-\mathbf{B}_2 \cdot \mathbf{D} + c_{02}x_0 + c_{22}x_2) + c_{10}y_0 + c_{11}y_1)$
$\mathbf{A}_2 \times \mathbf{B}_0$	$x_0 = -\sigma\text{Sign}(c_{10})a_0,\ x_1 = +\sigma\text{Sign}(c_{00})a_1,\ y_1 = -\sigma\text{Sign}(c_{22})b_1,\ y_2 = +\sigma\text{Sign}(c_{21})b_2,$ $x_2 = \frac{1}{1-c_{20}^2}(\mathbf{A}_2 \cdot \mathbf{D} + c_{20}(-\mathbf{B}_0 \cdot \mathbf{D} + c_{00}x_0 + c_{10}x_1) + c_{21}y_1 + c_{22}y_2)$
$\mathbf{A}_2 \times \mathbf{B}_1$	$x_0 = -\sigma\text{Sign}(c_{11})a_0,\ x_1 = +\sigma\text{Sign}(c_{01})a_1,\ y_0 = +\sigma\text{Sign}(c_{22})b_0,\ y_2 = -\sigma\text{Sign}(c_{20})b_2,$ $x_2 = \frac{1}{1-c_{21}^2}(\mathbf{A}_2 \cdot \mathbf{D} + c_{21}(-\mathbf{B}_1 \cdot \mathbf{D} + c_{01}x_0 + c_{11}x_1) + c_{20}y_0 + c_{22}y_2)$
$\mathbf{A}_2 \times \mathbf{B}_2$	$x_0 = -\sigma\text{Sign}(c_{12})a_0,\ x_1 = +\sigma\text{Sign}(c_{02})a_1,\ y_0 = -\sigma\text{Sign}(c_{21})b_0,\ y_1 = +\sigma\text{Sign}(c_{20})b_1,$ $x_2 = \frac{1}{1-c_{22}^2}(\mathbf{A}_2 \cdot \mathbf{D} + c_{22}(-\mathbf{B}_2 \cdot \mathbf{D} + c_{02}x_0 + c_{12}x_1) + c_{20}y_0 + c_{21}y_1)$

Figure 6: Table with the coefficients that can be used to find the intersection point in each of the cases described in Appendix A: the first 2 correspond to the 6 non-intersection tests for faces and the remaining 9 to each of the generated along the axis of the edges. Only the first non-intersection test fired will trigger it's corresponding collision response in the table. Values follow the same conventions as described in Appendix A. Coefficients $x_{0..2}$ and $y_{0..2}$ are the displacements along their corresponding axis (along their extents) that we use to obtain the intersection coordinates; for a simplified implementation I only solve the system for one of the 2 bodies.

REFERENCES

[1] David Baraff. 1989. Analytical methods for dynamic simulation of non-penetrating rigid bodies. In *ACM SIGGRAPH Computer Graphics*, Vol. 23. ACM, 223–232.

[2] David Eberly. 1999. Dynamic Collision Detection using Oriented Bounding Boxes. https://www.geometrictools.com/Documentation/DynamicCollisionDetection.pdf/. (1999). [Online; accessed 10-December-2017].

[3] K Erleben. 2004. *Stable, robust, and versatile multibody dynamics animation.* Ph.D. Dissertation.

[4] Alejandro Garcia-Alonso, NicolÃąs Serrano, and Juan Flaquer. 1994. Solving the collision detection problem. *IEEE Computer Graphics and Applications* 14, 3 (1994), 36–43.

[5] S. Gottschalk, M. C. Lin, and D. Manocha. 1996. OBBTree: A Hierarchical Structure for Rapid Interference Detection. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '96)*. ACM, New York, NY, USA, 171–180. https://doi.org/10.1145/237170.237244

[6] VAudioFX Stunning Motion Graphics. [n. d.]. Rigid Bodies and Collision. ([n. d.]). https://www.youtube.com/watch?v=mpbWQbkl8_g#t=20m15s Accessed: 2017-02-11.

[7] James K. Hahn. 1988. Realistic animation of rigid bodies. In *Acm Siggraph Computer Graphics*, Vol. 22. ACM, 299–308.

[8] D.H. House and J.C. Keyser. 2016. *Foundations of Physically Based Modeling and Animation.* CRC Press. https://books.google.ca/books?id=vaqdDQAAQBAJ

[9] Matthew Moore and Jane Wilhelms. 1988. Collision detection and response for computer animation. In *ACM Siggraph Computer Graphics*, Vol. 22. ACM, 289–298.

[10] James F. O'Brien and Jessica K. Hodgins. 1999. Graphical Modeling and Animation of Brittle Fracture. In *Proceedings of ACM SIGGRAPH 1999*. ACM Press/Addison-Wesley Publishing Co., 137–146. https://doi.org/10.1145/311535.311550

[11] Alex Pentland. 1990. *Computational complexity versus virtual worlds.* Vol. 24. ACM.