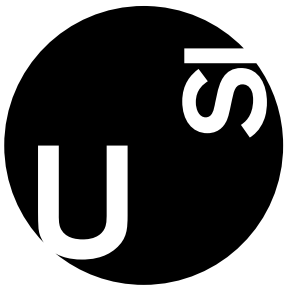


Università
della
Svizzera
italiana

Faculty
of
Informatics

Bachelor Thesis

June 7, 2023

From Flying Balls to Colliding Polygons

2D Physics Engine: Rigid Body Simulation

Arnaud Fauconnet

Abstract

Physics engines are a fun and interesting way to learn about a lot of different subjects. First the theoretical concepts, such as the equations that dictate the motion of the objects, together with their components, need to be thoroughly understood. Then there is the necessity of finding a way to represent all of those concepts in a given programming language and to make them as efficient as possible so that the simulation runs fluidly. The task to be completed here was to extend an already existing physics engine that only made circles bounce off each other. The extension was focused on having the ability to generate some arbitrary polygons and make them bounce off each other in a physically accurate way. The main issues that rose up during the development of the extension: determining the inertia of a arbitrary polygon, which is important for realistic impacts; having an accurate collision detection system, which allows the engine to know when to make two polygons bounce off each other. Once those aspects were worked on and polished, the rest of the implementation went smoothly.

Advisor
Prof. Antonio Carzaniga

Advisor's approval (Prof. Antonio Carzaniga): Date:

Contents

1 Introduction

1.1 Goal of the project

The goal of the project was to extend an existing physics engine called "flying-balls"[1] by Prof. Antonio Carzaniga. This physics engine simulated the interactions between circles in a two-dimensional space. These circles appear in the window with a random position, together with a random initial velocity vector. The simulation would then just calculate the position of each circle in the following frame and draw it in its new state. If two circles were to collide with each other, the engine would detect it and make those circles bounce off each other. The resulting position and speed would be decided by the physics equations that govern the motion of such objects.

The extension this project was asked to bring is the possibility to have more complex shapes interact with each other, such as polygons. The polygons would have to be arbitrary and bounce off other polygons present in the scene.

1.2 State of the art

There are a lot of 2D physiques engines across the internet. The purpose of this project was not to bring something new to the already existing landscape, but rather learn how to complete every step of the process (polygons generation, collision detection, kinematics resolution) from scratch, simply having a pre-existing way to represent the shapes on the screen.

[1] The state of the project before the extension can be found at `https://github.com/carzaniga/flying-balls/tree/c++-port`

2 Technical Background

The technical background is all the research related to the programming part of this bachelor project. The programming language used in this project is a mixture of C and C++, for this part, the course of Systems Programming taught by Prof. Carzaniga during the third semester. Then came the study of the starting point of the project, which was divided in the logic itself and the framework used to display the state of the simulation on the screen.

2.1 Original project

Before starting to write any code, it was necessary to study carefully the original project. The starting point of chosen for this specific project was the last commit on the `c++-port` branch. The reason for this choice is that the project originally started fully in C (which is still the case for the `main` branch) and C++ offers more functionalities that help for a smoother development process.

The life-cycle of the simulation was the typical three-step process:

1. **State initiation:** the state of the application is set with certain starting conditions;
2. **State update:** the state, at each frame, gets applied a set of rules that govern the behaviour of the application;
3. **Termination:** when the user stops the application, it actuates a number of `cleaning` up operations.

Just like any C/C++ project, the modules were split into different files, and those modules where themselves split into header files and implementation files. The header files expose the public interface which other modules can call to execute a determine function, whereas the implementation files, as the name suggests, offer the concrete implementation of the aforementioned functions. The implementation files can use some static[2] functions that it can use as auxiliary or utility functions. The header files usually expose the fields and methods of the class (or `struct`) the module is using, if any, together with one function for each of three steps of the life-cycle mentioned above.

2.2 Cairo

Cairo is a 2D graphics library with support for multiple output devices. Cairo is designed to produce consistent output on all output media while taking advantage of display hardware acceleration when available. The cairo API provides operations similar to the drawing operators of PostScript and PDF. Operations in cairo including stroking and filling cubic Bézier splines, transforming and compositing translucent images, and antialiased text rendering. All drawing operations can be transformed by any affine transformation (scale, rotation, shear, etc.). Reading the documentation[3], and more specifically the practical tutorial[4] was useful to understand how the library works.

The Cairo drawing model relies on a three-layer model, any drawing process takes place in three steps:

1. first a mask is created, which includes one or more vector primitives or forms, i.e., circles, squares, TrueType fonts, Bézier curves, etc;
2. then source must be defined, which may be a color, a color gradient, a bitmap or some vector graphics, and from the painted parts of this source a die cut is made with the help of the above defined mask;
3. finally the result is transferred to the destination or surface, which is provided by the back-end for the output.

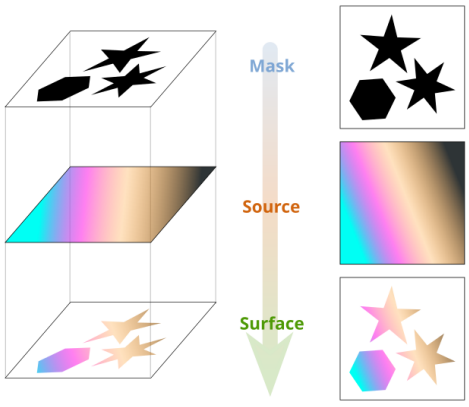


Figure 1. Cairo's drawing model

[2] static in the sense of C, i.e. visible only to the file it is declared in
[3] `https://www.cairographics.org/documentation/`
[4] `https://www.cairographics.org/tutorial/`

3 Theoretical Background

The theoretical background is everything related to the physics part of the project. It covers the calculating the inertia of different types of polygons; different algorithms to detect whether there is a collision between two polygons; the resolution of the collision, i.e. finding the final velocity vectors and angular speed of those polygons.

3.1 Moment of inertia

The inertia of an object refers to the tendency of an object to resist a change of its state of motion or rest, it describes how the object behaves when forces are applied to it. An object with a lot of inertia requires more force to change its motion, either to make it move if it's at rest or to stop it if it's already moving. On the other hand, an object with less inertia is easier to set in motion or bring to a halt.

The moment of inertia is similar but is used in a slightly different context, it specifically refers to the rotational inertia of an object. It measures an object's resistance to changes in its rotational motion and how its mass is distributed with respect to is axis of rotation.

In the case of this project the axis of rotation is the one along the z-axis (perpendicular to the plane of the simulation) and placed at the barycenter of the polygon.

The general formula for the moment of inertia is

$$I_Q = \int \vec{r}^2 \rho(\vec{r}) \, \mathrm{d}\mathcal{A} \tag{1}$$

where ρ is the density of object Q in the point $\vec{r}$ across the small pieces of area $\mathcal{A}$ of the object.

In our case, since we are implementing a 2D engine we can use the $\mathbb{R}^2$ coordinate systems, thus the formula becomes

$$I_Q = \iint \rho(x, y) \vec{r}^2 \, \mathrm{d}x \, \mathrm{d}y$$

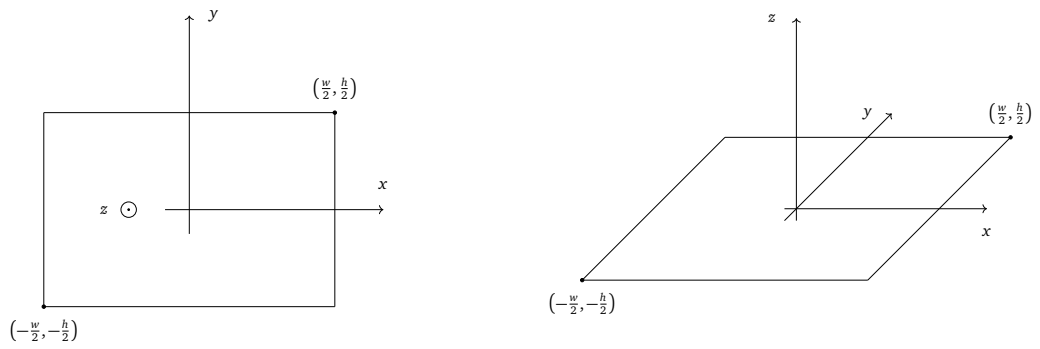
and since the requirements express that the mass of the polygons is spread uniformly across its surface, the formula finally becomes

$$I_Q = \rho \iint x^2 + y^2 \, \mathrm{d}x \, \mathrm{d}y \tag{2}$$

The bounds of the integral depend on the shape of the polygon. In the following sections, we will describe how to compute those bounds, then we will show a different technique to compute the moment of inertia of arbitrary polygons.

3.1.1 Rectangle

The moment of inertia of a rectangle of width w and height h with respect to the axis of rotation that passes through its barycenter can be visualized in the Figure 2.



(a) 2d view of rectangle with axis of rotation

(b) 3d view of rectangle with axis of rotation

Figure 2. Representation of rectangle with respect to axis of rotation z

As figure Figure 2a implies, the bounds of equation 2 are trivial to derive:

$$I_{\text{rect}} = \rho \int_{-\frac{h}{2}}^{\frac{h}{2}} \int_{-\frac{w}{2}}^{\frac{w}{2}} x^2 + y^2 \, \mathrm{d}x \, \mathrm{d}y = \frac{\rho w h}{12} \left(w^2 + h^2 \right) \tag{3}$$

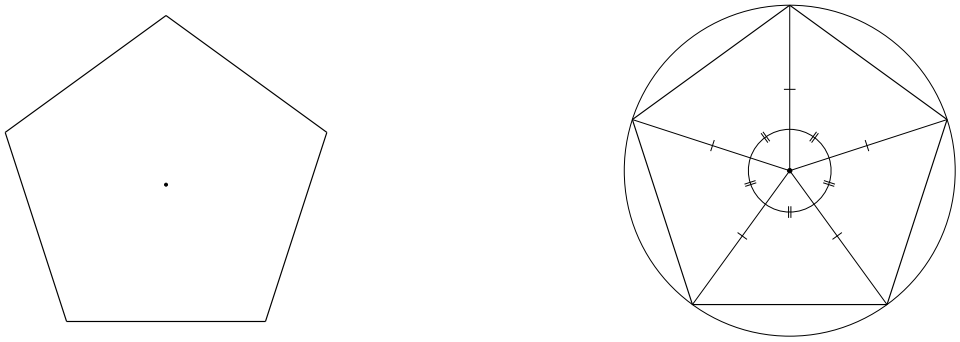
and since ρwh is the density of the rectangle multiplied by its area, we can replace this term by its mass m, thus

$$I_{\text{rect}} = \frac{1}{12} m \left(w^2 + h^2 \right) \tag{4}$$

All the steps to compute equation 3 can be found in equation 15 in Appendix A.

3.1.2 Regular Polygons

A regular polygon is a shape that has sides of equal length and angles between those sides of equal measure. A polygon of n sides can be subdivided in n congruent (and isosceles since they are all the radius of the circumscribing circle) triangles that all meet in the polygon's barycenter, as demonstrated in Figure 3b with a pentagon.



(a) Regular polygon of 5 sides with its barycenter

(b) Pentagon divided in 5 congruent triangles

Figure 3. Subdivision of regular polygons into congruent triangles

If we define one of the sub-triangle of the regular polygon as T, then we can find the moment of inertia I_T when it is rotating about the barycenter. To find the bounds of the integral in equation 2, we can take the triangle T and place it along the x-axis so that it is symmetric likes shown in figure. Assuming the side length of the polygon is l, the height of the triangle T is h and the angle of the triangle on the barycenter of the polygon to be θ, then

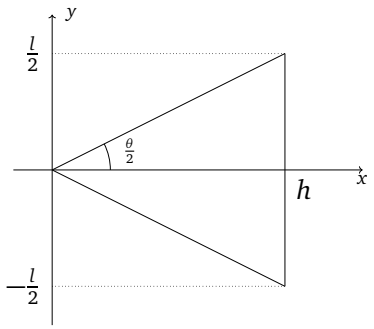


Figure 4. Sub-triangle T of regular polygon

we can see the bounds for the integral

$$I_T = \rho \int_0^h \int_{-\frac{lx}{2h}}^{\frac{lx}{2h}} x^2 + y^2 \, \mathrm{d}y \, \mathrm{d}x = \frac{m_T l^2}{24} \left(1 + 3 \cot^2 \left(\frac{\theta}{2} \right) \right) \tag{5}$$

All the steps to compute equation 5 can be found in equation 16 in Appendix A.
Now that we have the moment of inertia of the sub-triangle, we can make the link to the overall polygon. Since

$$\theta = \frac{2\pi}{n} \implies \frac{\theta}{2} = \frac{\pi}{n}$$

and the moment of inertia are additive (as long they are as they are about the same axis) we can get the moment of inertia with

$$I_{\text{regular}} = nI_T$$

and since the mass of the regular polygon m is the sum of the masses of the sub-triangle

$$m = nm_T$$

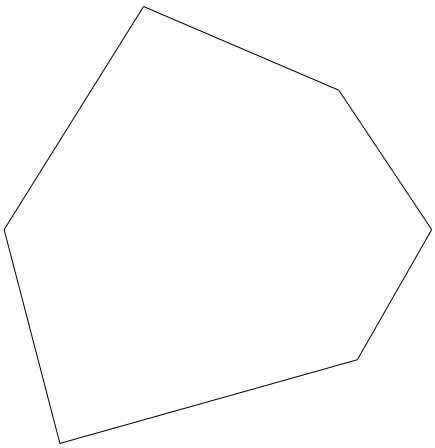
we have that

$$I_{\text{regular}} = \frac{ml^2}{24}\left(1 + 3\cot^2\left(\frac{\pi}{n}\right)\right) \tag{6}$$

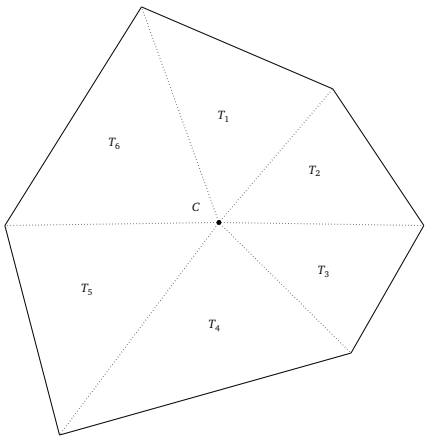
3.1.3 Arbitrary Polygons

For arbitrary polygons, we are taking a slightly different approach. Using the Cartesian coordinate system to solve the equation 2 revealed to be more cumbersome than useful. But similarly to regular polygons (c.f. Section 3.1.2), we can use the additive property of the moment inertia to divide our arbitrary polygon into sub-triangles. As opposed to regular polygons, these triangles won't be congruent, so we can't just get the moment of inertia of one of them and multiply it by the number of sides, but we need to calculate them individually. So given a polygon of n sides, we can construct n sub-triangles T_i, for $i = 1, \ldots, n$. So the moment of inertia I of the polygon will be

$$I = \sum_i I_{T_i} \tag{7}$$



(a) An arbitrary 6-sided polygon



(b) Arbitrary polygon divided into 6 sub-triangles

To calculate the moment of inertia I_{T_i}, instead of using the classical x- and y-axis as we did before, we decided to use the edges of the triangle as axis and therefore express what we need to integrate in function of those as can be seen in Figure 6.

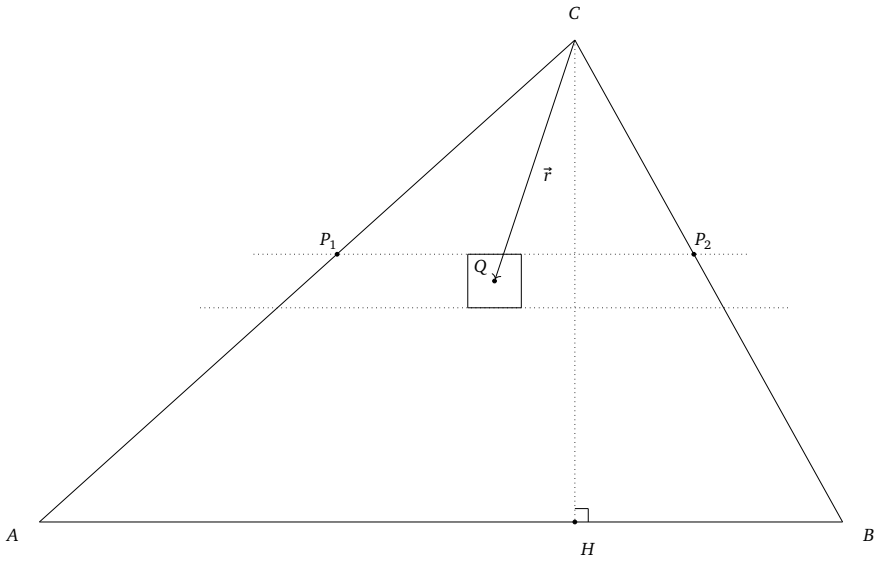


Figure 6. Sub-triangle of arbitrary polygon

In Figure 6, C represent the barycenter of the polygon (as is shown in Figure 5b). The axis we are going to integrate on are $\overrightarrow{CA}$ and $\overrightarrow{AB}$. We can now define

$$\overrightarrow{CP_1} = \alpha\overrightarrow{CA}, \qquad \overrightarrow{CP_2} = \alpha\overrightarrow{CB}, \qquad \forall\alpha \in [0,1] \tag{8}$$

and

$$\overrightarrow{P_1Q} = \beta\overrightarrow{P_1P_2}, \qquad \forall\beta \in [0,1]$$

From 8, it quickly follows that

$$\overrightarrow{P_1P_2} = \alpha\overrightarrow{AB}$$

therefore

$$\overrightarrow{P_1Q} = \beta\alpha\overrightarrow{AB} \tag{9}$$

Finally, if we put together equations 8 and 9, we have that

$$\vec{r} = \overrightarrow{CP_1} + \overrightarrow{P_1Q} = \alpha\overrightarrow{CA} + \beta\alpha\overrightarrow{AB} \tag{10}$$

Now we got the first part equation 1. To find the $\mathrm{d}\mathcal{A}$, we just need to get the area of the square that contains Q in Figure 6. Since $\|\overrightarrow{AB}\|$ represents the base of the triangle T_i, we can define

$$b = \|\overrightarrow{AB}\|$$

we consequently have that

$$\mathrm{d}\mathcal{A} = b\alpha\,\mathrm{d}\beta h\,\mathrm{d}\alpha \tag{11}$$

where $h = \|\overrightarrow{CH}\|$ is the height of triangle. We can now assemble 10 and 11

$$I_{T_i} = \rho\int_0^1\int_0^1 \vec{r}^2 hb\alpha\,\mathrm{d}\alpha\,\mathrm{d}\beta = \frac{\rho hb}{4}\left(\frac{1}{3}\overrightarrow{AB}^2 + \overrightarrow{AB}\cdot\overrightarrow{CA} + \overrightarrow{CA}^2\right) \tag{12}$$

Since $\frac{\rho hb}{2}$ is the mass of the triangle we can write the result as

$$I_{T_i} = \frac{m_{T_i}}{2}\left(\frac{1}{3}\overrightarrow{AB}^2 + \overrightarrow{AB}\cdot\overrightarrow{CA} + \overrightarrow{CA}^2\right) \tag{13}$$

All the steps to compute equation 12 can be found in equation 17 in Appendix A.
Now that we have the moment of inertia of the sub-triangle, we can make the link to the overall polygon.

$$I_{\text{arbitrary}} = \sum_i I_{T_i} = \sum_{i=1}^{n}\frac{m_{T_i}}{2}\left(\frac{1}{3}\overrightarrow{P_iP_{i+1}}^2 + \overrightarrow{CP_i}\cdot\overrightarrow{P_iP_{i+1}} + \overrightarrow{CP_i}^2\right) \tag{14}$$

where, $P_{n+1} = P_1$ in the case of $i = n$.

3.2 Collision detection

Collision detection, as the name suggests, are the algorithms used to detect whether two polygons are colliding. The result of this procedure must be an impact point and a normal vector, that will then be used for the collision resolution 3.3.

3.2.1 Separating Axis Theorem

This algorithm was the first one studied for this project and was inspired by the works of David Eberly [1]. The separating axis theorem (SAT) states that if you can draw a line between two convex objects, they do not overlap. We will call this line a *separating line*. More technically, two convex shapes do not overlap if there exists an axis onto which the two objects' projections do not overlap. We'll call this axis a *separating axis*. This concept can be visualized in Figure 7.

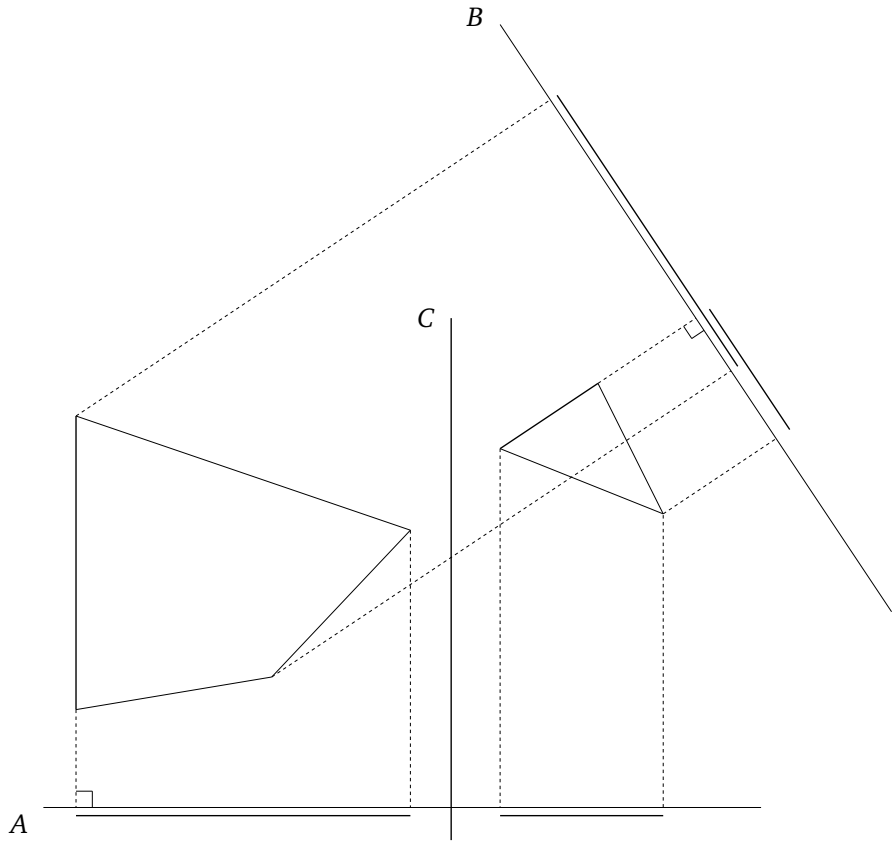


Figure 7. SAT: Separating axis (A) vs non-separating axis (B), with separating line (C)

As we can see in Figure 7, the axis B show that the projections of the both polygons overlap, but we were able to find an axis A where this is not the case. As soon as we find an axis for which the projections do not overlap, it means that the polygons are not colliding. For 2D objects, we only need to consider the axes that are orthogonal to each edge. In Figure 7, only two of those axes are shown for better readability, but they would be 7, one for each edge.

To move (or push) one polygon away from the other, we also need to find a vector that, when added to the polygons position, will make the shapes not overlap. We want the minimum displacement possible, We'll call this vector the minimum push vector (MPV). For 2-dimensional polygons, this vector will lie in some of the orthogonal axes.

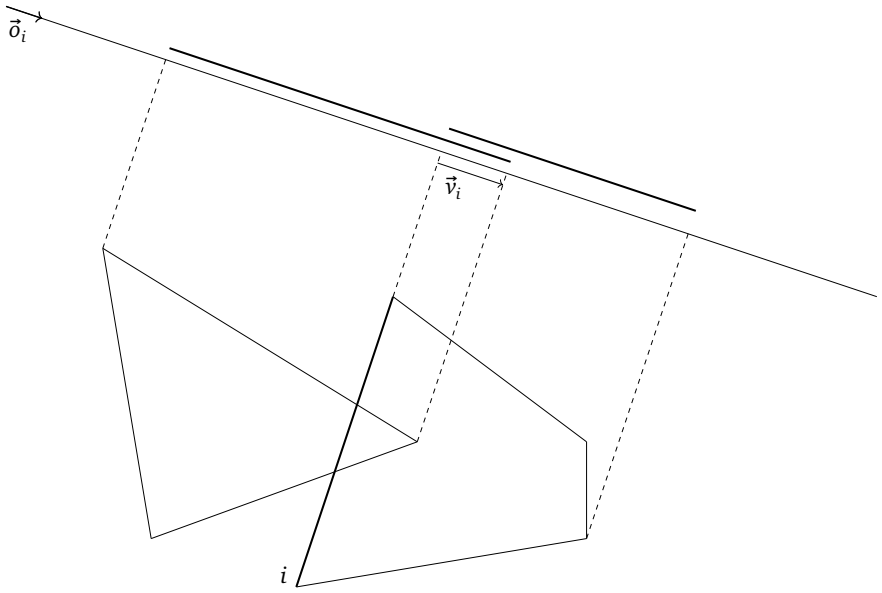


Figure 8. SAT: Minimum push vector $\vec{v}_i$ on axis defined by $\vec{o}_i$, orthogonal to edge i

The candidate MPVs $\vec{v}_i$ are the vectors that define the axis $\vec{o}_i$ (orthogonal to edge i), with $\|\vec{o}_i\| = 1$, multiplied by the minimum overlap between the two polygons, as shown in 8. The final MPV is simply the $\vec{v}_i$ with the smallest norm.

Pitfalls of SAT The issue with the SAT algorithm is that although it is good to find whether two polygons are colliding and the MPV, it isn't trivial to gather the point of impact, i.e. the vertex that is penetrating the other polygon. It is doable, but during the implementation, it came with some caveats that introduced some bugs, so we decided to switch strategy and go with an algorithm of our own.

3.2.2 Vertex collisions

3.3 Collision resolution

3.3.1 Physics

3.3.2 Solving for the impulse parameter

4 Proposed solution

5 Conclusion

References

[1] David Eberly. Intersection of convex objects: The method of separating axes. *WWW page*, pages 2–3, 2001.

A Calculations

Moment of inertia of rectangle

$$
\begin{aligned}
I_{\text{rect}} &= \rho \int_{-\frac{h}{2}}^{\frac{h}{2}} \int_{-\frac{w}{2}}^{\frac{w}{2}} x^2 + y^2 \,\mathrm{d}x\,\mathrm{d}y \\
&= 4\rho \int_{0}^{\frac{h}{2}} \int_{0}^{\frac{w}{2}} x^2 + y^2 \,\mathrm{d}x\,\mathrm{d}y \\
&= 4\rho \int_{0}^{\frac{h}{2}} \left[\frac{1}{3}x^3 + xy^2 \right]_0^{\frac{w}{2}} dy \\
&= 4\rho \int_{0}^{\frac{h}{2}} \frac{1}{3}\frac{w^3}{8} + \frac{w}{2}y^2 dy \\
&= 2\rho \int_{0}^{\frac{h}{2}} \frac{w^3}{12} + wy^2 dy \\
&= 2\rho \left[\frac{w^3}{12}y + \frac{w}{3}y^3 \right]_0^{\frac{h}{2}} \\
&= 2\rho \frac{w}{3} \left[\frac{w^2}{4}y + y^3 \right]_0^{\frac{h}{2}} \\
&= 2\rho \frac{w}{3} \left(\frac{w^2}{4}\frac{h}{2} + \frac{h^3}{8} \right) \\
&= \rho \frac{w}{3} \left(\frac{w^2}{4}h + \frac{h^3}{4} \right) \\
&= \frac{\rho w h}{12} \left(w^2 + h^3 \right)
\end{aligned}
\tag{15}
$$

Moment of inertia of sub-triangle of regular polygon Before starting the calculations, it is to be noted that according to Figure 4, we have that

$$\tan\left(\frac{\theta}{2}\right) = \frac{\frac{l}{2}}{h} = \frac{l}{2h}$$

it will be useful to simplify the result of the integral.

$$\begin{aligned}
I_T &= \rho \int_0^h \int_{-\frac{lx}{2h}}^{\frac{lx}{2h}} x^2 + y^2 \,\mathrm{d}y \,\mathrm{d}x \\
&= 2\rho \int_0^h \int_0^{\frac{lx}{2h}} x^2 + y^2 \,\mathrm{d}y \,\mathrm{d}x \\
&= 2\rho \int_0^h \left[x^2 y + \frac{1}{3} y^3 \right]_0^{\frac{lx}{2h}} \mathrm{d}x \\
&= 2\rho \int_0^h x^2 \frac{lx}{2h} + \frac{1}{3} \frac{l^3 x^3}{8h^3} \,\mathrm{d}x \\
&= 2\rho \left(\frac{l}{2h} + \frac{l^3}{24h^3} \right) \int_0^h x^3 \,\mathrm{d}x \\
&= 2\rho \left(\frac{l}{2h} + \frac{l^3}{24h^3} \right) \left[\frac{1}{4} x^4 \right]_0^h \\
&= \frac{h^4 \rho}{2} \left(\frac{l}{2h} + \frac{l^3}{24h^3} \right) \\
&= \frac{\rho l h^3}{4} \left(1 + \frac{l^2}{12h^2} \right) \\
&= \frac{m_T h^2}{2} \left(1 + \frac{l^2}{12h^2} \right) \\
&= \frac{m_T}{2} \frac{l^2}{4\tan^2\left(\frac{\theta}{2}\right)} \left(1 + \frac{4\tan^2\left(\frac{\theta}{2}\right)}{12} \right) \\
&= \frac{m_T l^2}{24} \left(1 + 3\cot^2\left(\frac{\theta}{2}\right) \right)
\end{aligned} \tag{16}$$

Moment of inertia of sub-triangle of arbitrary polygon Recall equation 10 defines

$$\vec{r} = \alpha\overrightarrow{CA} + \beta\,\alpha\overrightarrow{AB}$$

$$\begin{aligned}
I_{T_i} &= \rho \int_0^1 \int_0^1 \vec{r}^{\,2} hb\alpha \,\mathrm{d}\alpha \,\mathrm{d}\beta \\
&= \rho hb \int_0^1 \int_0^1 \left(\alpha\overrightarrow{CA} + \beta\,\alpha\overrightarrow{AB}\right)^2 \alpha \,\mathrm{d}\alpha \,\mathrm{d}\beta \\
&= \rho hb \int_0^1 \int_0^1 \left(\alpha^2\overrightarrow{CA}^2 + 2\alpha^2\beta\overrightarrow{AB}\cdot\overrightarrow{CA} + \alpha^2\beta^2\overrightarrow{AB}^2\right)\alpha \,\mathrm{d}\alpha \,\mathrm{d}\beta \\
&= \rho hb \int_0^1 \int_0^1 \alpha^3\left(\overrightarrow{CA}^2 + 2\beta\overrightarrow{AB}\cdot\overrightarrow{CA} + \beta^2\overrightarrow{AB}^2\right)\mathrm{d}\alpha \,\mathrm{d}\beta \\
&= \rho hb \int_0^1 \left[\frac{1}{4}\alpha^4\left(\overrightarrow{CA}^2 + 2\beta\overrightarrow{AB}\cdot\overrightarrow{CA} + \beta^2\overrightarrow{AB}^2\right)\right]_0^1 \mathrm{d}\beta \\
&= \frac{\rho hb}{4} \int_0^1 \beta^2\overrightarrow{AB}^2 + 2\beta\overrightarrow{AB}\cdot\overrightarrow{CA} + \overrightarrow{CA}^2 \,\mathrm{d}\beta \\
&= \frac{\rho hb}{4}\left[\frac{1}{3}\beta^3\overrightarrow{AB}^2 + \beta^2\overrightarrow{AB}\cdot\overrightarrow{CA} + \beta\overrightarrow{CA}^2\right]_0^1 \\
&= \frac{\rho hb}{4}\left(\frac{1}{3}\overrightarrow{AB}^2 + \overrightarrow{AB}\cdot\overrightarrow{CA} + \overrightarrow{CA}^2\right)
\end{aligned} \tag{17}$$