HAMILTON TRUST SUMMER INTERNSHIP PROGRAMME

Trinity College Dublin - School of Mathematics

Interactively Prototyping Properties of Rigid Bodies for Physically-based Animation

Nicholas Pochinkov[1] | supervised by John Dingliana[2]

[1]Trinity College Dublin, Dublin, Ireland

[2]School of Computer Science & Statistics, Trinity College Dublin, Dublin, Ireland

Correspondence
http://pesvut.netsoc.ie
Email: pochinkn@gmail.com

Funding information
No funding was received for the duration of this internship

The aim of the research project was to produce an interactive tool that can compute various physical aspects of a body, with a focus on computing and manipulating the tensor of inertia of a complex rigid body. The methods of interaction include pulling the object on a string attached to the mouse, and "poking" the object by clicking on it. One can easily change the size and mass of the object, swap between the approximate and true tensor of inertia, as well as the forces being applied on it. The tool was created successfully and will hopefully aid people to better understand the Tensor of Inertia.

KEYWORDS
processing, rigid-bodies, simulations, physics, rotation, education

1 | INTRODUCTION

The ultimate aim of the research project was to produce an interactive tool that can compute various physical aspects of a body, with a focus on computing and manipulating the tensor of inertia of a complex rigid body. This can be used to teach people about the tensors of inertia, and to advance physics simulation in modern live graphical simulations such as the growing video game and VR industry. To code this, I used the “Processing” programming language development environment, a language favoured for its ease of use for producing graphical outputs and similarity to c, leading to a shorter time to get up and running.

2 | BACKGROUND

2.1 | Numerical Integration

Many aspects of this program require the solving of differential equations of the form $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, t)$, with **x** denoting some vector, **f** denoting a vector field and t denoting time. If we assume we have an initial starting point and a way of calculating $\mathbf{f}(\mathbf{x}, t)$. We get to the two main methods we will be using:

2.1.1 | Euler's Method

Here we get the McLauren expansion of $\mathbf{x}(t_0 + \Delta t)$:

$$\mathbf{x}(t_0 + \Delta t) = \mathbf{x}(t_0) + \mathbf{x}'(t_0)\Delta t + \frac{\mathbf{x}''(t_0)\Delta t^2}{2!} + \frac{\mathbf{x}'''(t_0)\Delta t^3}{3!} + \dots \tag{1}$$

For the Euler method, we assume that the second order derivative and those after are negligible. This gives $\mathbf{x}(t_0 + \Delta t) = \mathbf{x}(t_0) + \mathbf{x}'(t_0)\Delta t$. However, we know that $\mathbf{x}'(t) = \mathbf{f}(\mathbf{x}, t)$. Thus, we get that :

$$\mathbf{x}(t_0 + \Delta t) = \mathbf{x}(t_0) + \mathbf{f}(\mathbf{x}, t)\Delta t \tag{2}$$

This means that we can make a function that gives us $\mathbf{x}(t_0 + \Delta t)$ from any starting point $\mathbf{x}(t_0)$, the linear slope at that point $\mathbf{f}(\mathbf{x}, t)$, and the time interval over which we are integrating, Δt. This gives us an error of $O(n^2)$ inaccuracy (i.e. adding more points makes reduces the error proportional to the number of points squared)

2.1.2 | Midpoint method

However, we can also improve the rate of convergence of the from $O(n^2)$ to $O(n^3)$ (or potentially even higher, though this would require even more calculations per step). We work again with the McLauren expansion above but this time include the second derivative, and try to find x". Here we must assume $\frac{dx}{dt} = \mathbf{f}(\mathbf{x}(t))$ with **f** independent of t (or at least the dependence is negligible). (for simplicity the equation will be written as $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$). This leads us to get:

$$\ddot{x} = \frac{\delta \mathbf{f}(\mathbf{x})}{\delta t} = \frac{\delta \mathbf{f}(\mathbf{x})}{\delta \mathbf{x}} \cdot \frac{\delta \mathbf{x}}{\delta t} = \mathbf{f}' \cdot \mathbf{f} \tag{3}$$

To find this we now look at the expansion of $\mathbf{f}(\mathbf{x}_0 + \Delta\mathbf{x}) = \mathbf{f}(\mathbf{x}_0) + \mathbf{f}'(\mathbf{x}_0)\Delta\mathbf{x} + O(n^2)$.
If we do the substitution $\Delta\mathbf{x} = \frac{\mathbf{f}(x_0)\Delta t}{2}$,
Thus we get that: $\mathbf{f}(\mathbf{x}_0 + \frac{\mathbf{f}(x_0)\Delta t}{2}) = \mathbf{f}(\mathbf{x}_0) + \mathbf{f}'(\mathbf{x}_0)\mathbf{f}(x_0)\frac{\Delta t}{2} + O(n^2)$.
Substituting the above, we get that $\mathbf{f}(\mathbf{x}_0 + \frac{\mathbf{f}(x_0)\Delta t}{2}) = \ddot{\mathbf{x}}\frac{\Delta t}{2} + O(n^2)$
And if we multiply each term by Δt and rearrange, we get $\mathbf{f}(\mathbf{x}_0 + \frac{\mathbf{f}(x_0)\Delta t}{2})\Delta t - \mathbf{f}(\mathbf{x}_0)\Delta t = \ddot{\mathbf{x}}\frac{\Delta t}{2} + O(n^3)$
We substitute this into $\mathbf{x}(t_0 + \Delta t) = \mathbf{x}(t_0) + \mathbf{x}'(t_0)\Delta t + \frac{\mathbf{x}''(t_0)\Delta t^2}{2!} + O(n^3)$
And this leaves us with:

$$\mathbf{x}(t_0 + \Delta t) = \mathbf{x}(t_0) + \mathbf{f}(\mathbf{x}_0 + \frac{\mathbf{f}(x_0)\Delta t}{2})\Delta t \tag{4}$$

2.2 | Rigid-body Dynamics

The rigid bodies have different properties that we need when displaying and accurately moving them in response to stimuli. The main thing we need are:

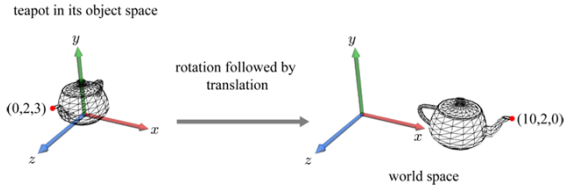
- The mass of the object
- The current position (of the center of mass)
- The current velocity
- The Tensor of Inertia of the object (equivalent to mass for rotation)
- The current orientation of the object
- The angular velocity of the object (how the orientation is changing)
- A representation of the shape of the object (basically a list of points and connections)

These however can be seen from both the body-space and from the world-space of the object which I will explain first

2.2.1 | Body-Space vs World-Space

Body space refers to the object being in a default orientation with the object being centered at the origin. We always keep a copy of the object in its body space to avoid any compounding errors slowly deforming the object, and it makes it easier to perform certain operations such as rotation.

The world space represents it's actual physical representation in the 3D world we have made up, with each of the points being where we would want them to be in the physical world. This is correctly oriented and rotated as described with the above properties.



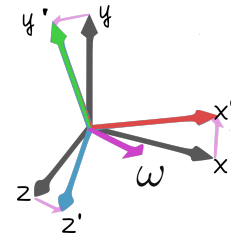
https://ya-webdesign.com/explore/clip-space-vertex-shader/

2.2.2 | Linear Properties

For our purposes, we will be assuming that mass of the object does not change, thus Force is the product of mass times acceleration $\mathbf{F} = m\mathbf{a}$ or alternatively, $\ddot{\mathbf{x}} = \frac{\mathbf{F}}{m}$. Of course, the ODE solutions discussed before were for first-order differential-equations, so we split this up into $\dot{\mathbf{x}} = \mathbf{v}$ and $\dot{\mathbf{v}} = \frac{\mathbf{F}}{m}$. Thus given an initial position and velocity (for example, 0) and a force at each step, we can use Euler's method to integrate v with respect to F/m, and integrate the position with respect to the slightly changed v.

2.2.3 | Rotation Matrix

We also need to orientate the object. As we know from mechanics, it is not possible to parametrize the orientation of a 3D object with 3 values as the rotations of the object do not commute. Instead we must use either a Rotation Matrix or a Quaternion to rotate the body. Because of my higher familiarity with Linear Algebra, we will be using the Rotation Matrix.



$R = \begin{pmatrix} x'_x & y'_x & z'_x \\ x'_y & y'_y & z'_y \\ x'_z & y'_z & z'_z \end{pmatrix}$ which shows us rotated x'-y'-z' in terms of standard x-y-z

This leads us on the problem of how to find the rotation matrix, which we will solve by initially beginning with some R (lets say I_3) and then integrating it with the angular velocity vector ω. We notice that the angular velocity gives us the linear velocity through $\mathbf{v} = \mathbf{r} \times \omega$, where **r** is the position vector (the same as **x** before but different for ease of distinction). Thus, given a starting vector $\hat{x}, \hat{y}$ or $\hat{z}$ we can get the new $\hat{x}', \hat{y}'$ or $\hat{z}'$ using the fact that $\frac{d\hat{x}}{dt} = \hat{x} \times \omega$, $\frac{d\hat{y}}{dt} = \hat{x} \times \omega$ and $\frac{d\hat{z}}{dt} = \hat{x} \times \omega$

Thus $\dot{R} = \begin{pmatrix} \omega \times \begin{pmatrix} x_x \\ x_y \\ x_z \end{pmatrix} & \omega \times \begin{pmatrix} y_x \\ y_y \\ y_z \end{pmatrix} & \omega \times \begin{pmatrix} z_x \\ z_y \\ z_z \end{pmatrix} \end{pmatrix}$

However, it is also possible to describe the cross product of a vector(the "$\omega\times$" part) as it's own matrix:

$\omega = \begin{pmatrix} 0 & -\omega_z & +\omega_y \\ +\omega_z & 0 & -\omega_x \\ -\omega_y & +\omega_x & 0 \end{pmatrix}$

Thus, using this, we can find the derivative of the rotation matrix to be:

$$\dot{R} = \begin{pmatrix} 0 & -\omega_z & +\omega_y \\ +\omega_z & 0 & -\omega_x \\ -\omega_y & +\omega_x & 0 \end{pmatrix} \begin{pmatrix} x'_x & y'_x & z'_x \\ x'_y & y'_y & z'_y \\ x'_z & y'_z & z'_z \end{pmatrix} \tag{5}$$

which we means we can integrate R using the angular velocity to find the new rotation matrix after a period of time. However because we are doing this many times over many steps, the rotation matrix can become slightly skewed over time and scale up and thus we would need to return it back to an ortho-normal matrix. As this is a 3x3 matrix, we can easily do this using Grahm-Shmidt orthogonalization. Given x_0, x_1 and x_2, to find orthogonal w_0, w_1 and w_2 we do:

$$w_0 = x_0, \qquad w_1 = x_1 - \frac{x_1 \cdot w_0}{w_0 \cdot w_0} w_0, \qquad w_2 = x_2 - \frac{x_2 \cdot w_0}{w_0 \cdot w_0} w_0 - \frac{x_2 \cdot w_1}{w_1 \cdot w_1} w_1, \qquad then \qquad w_i = \frac{w_i}{|w_i|} \tag{6}$$

Doing this after every integration stops deforming of our shapes as it goes from body space to world-space.

2.2.4 | Translation Between World and Body Spaces

Now that we have the position of the Center of Mass and rotational orientation of the body, by Chasles' Theorem, we have enough information to translate the body from body-space to world-space. To do this, we consider each of the points of the shape (stored in the PShape class). We take the PVector showing position of the point and multiply it by R to give the rotated position ($x_{rotated} = R \cdot x$). We then take this new point and add the position vector to get the world-space position ($x_{world} = x_{rotated} + r$). We then store this in a new PShape containing all the rotated points.

2.2.5 | Application of Forces

Of course, we want the object to respond to forces as they act on them. We have already seen how forces affect linear properties of the rigid-body. We will now look at how this affects the rotational aspects of the body. To calculate these, we of course need to look at the Tensor of Inertia, $\tilde{I}$ for a body made of many point pieces of mass m_i with positions x_i, y_i, z_i which is calculated as follows:

$$\tilde{I} = \begin{pmatrix} \sum_i m_i(y_i^2 + z_i^2) & -\sum_i m_i x_i y_i & -\sum_i m_i x_i z_i \\ -\sum_i m_i y_i x_i & \sum_i m_i(x_i^2 + z_i^2) & -\sum_i m_i y_i z_i \\ -\sum_i m_i z_i x_i & -\sum_i m_i z_i y_i & \sum_i m_i(x_i^2 + y_i^2) \end{pmatrix} \tag{7}$$

This can be integrated accurately, but for the current discussion, we can use an approximate Tensor of inertia for a uniform cuboid of size a, b, c and mass m, which is the size of the bounding box of the shape. This is given by:

$$I = \begin{pmatrix} \frac{1}{12}m(b^2 + c^2) & 0 & 0 \\ 0 & \frac{1}{12}m(a^2 + c^2) & 0 \\ 0 & 0 & \frac{1}{12}m(a^2 + b^2) \end{pmatrix} \tag{8}$$

We recall from mechanics that the Angular Momentum (**L**) of an object is given my it's tensor of inertia multiplied by it's angular velocity: $\mathbf{L} = \tilde{I}\omega$. If we take the derivative of this with respect to time, assuming the Tensor of inertia remains unchanged, we get: $\frac{d\mathbf{L}}{dt} = \tilde{I}\frac{d\omega}{dt}$. However, we already know that Torque $\tau = \frac{d\mathbf{L}}{dt} = (\mathbf{r} \times \mathbf{f})$. Thus, we can multiply both sides of the equation by the inverse matrix of the Tensor of Inertia (I^{-1}) to get:

$$\frac{d\omega}{dt} = \tilde{I}^{-1}(\mathbf{r} \times \mathbf{f}) \tag{9}$$

With this, given the location of the force relative to the center of mass and the force vector we have a formula for the angular acceleration, which can be used to integrate angular velocity, which can be used to integrate the rotation matrix. For our purposes, we will translate **r** and **f** from the world-space to the body space by doing the reverse of what was done before: subtract the position of the Center of Mass, and multiply the resulting vector by the inverse of the rotation matrix.

2.3 | Integrating to find the Tensor of Inertia

The method for finding the tensor of inertia was detailed in "Fast and accurate computation of polyhedral mass properties" by Brian Mirtich. [3] The method of integration described involves projecting the 3-Dimensional integrals in 3D space into 2-Dimensional integrals in 3D Space using the Divergence Theorem, then projecting these into 2-Dimensional integrals in 2D Space using Theorem 2, and then projecting these integrals into 1-Dimensional integrals in 2D Space,

which makes the computation far easier to compute. The integrals that require evaluation are as follows:

$$T_1 = \int_{\mathcal{V}} 1 dV \qquad T_x = \int_{\mathcal{V}} x dV \qquad T_{x^2} = \int_{\mathcal{V}} x^2 dV \qquad T_{xy} = \int_{\mathcal{V}} xy dV$$
$$T_y = \int_{\mathcal{V}} y dV \qquad T_{y^2} = \int_{\mathcal{V}} y^2 dV \qquad T_{xz} = \int_{\mathcal{V}} xz dV$$
$$T_z = \int_{\mathcal{V}} z dV \qquad T_{z^2} = \int_{\mathcal{V}} z^2 dV \qquad T_{yz} = \int_{\mathcal{V}} yz dV$$

These equations were integrated by the method described in the paper (which I will not repeat but it is worth a read if you wish to understand it better), but I will give a quick statement of the Thoerems used:

Divergence Theorem

$$\int_{\mathcal{V}} \nabla \cdot \mathbf{F} dV = \int_{\partial\mathcal{V}} \mathbf{F} \cdot \hat{\mathbf{n}} dA$$

$\mathcal{V}$ - region in space

$\partial\mathcal{V}$ - boundary of $\mathcal{V}$

$\hat{\mathbf{n}}$ - exterior normal

$\mathbf{F}$ - any Vector Field

Projection Theorem

$\mathcal{F}$ a polygonal region

α-β-γ coordinate system

$$\int_{\mathcal{F}} f(\alpha,\beta,\gamma) dA = \frac{1}{|\hat{n}_\gamma|} \int_{\Pi} f(\alpha,\beta,h(\alpha,\beta)) d\alpha d\beta$$

$\hat{n}$ surface normal

Π projection of $\mathcal{F}$ on α-β space

$$where: h(\alpha,\beta) = -\frac{1}{\hat{n}_\gamma}\left(\hat{n}_\alpha \alpha + \hat{n}_\beta \beta + w\right)$$

the plane in which the points lie :

$$\hat{n}_\alpha \alpha + \hat{n}_\beta \beta + \hat{n}_\gamma \gamma + w = 0$$

Green's Theorem

$$\int_{\Pi} \nabla \cdot \mathbf{H} dA = \oint_{\partial\Pi} \mathbf{H} \cdot \hat{\mathbf{m}} ds$$

Π - region in the plane

$\partial\Pi$ - single boundary

$\hat{m}$ - exterior normal along the boundary.

$\mathbf{H}$ - any vector field on Π

Line integral transverses anti-clockwise

Vertices Theorem

L_e - length of the directed line segment from (α_e, β_e) to $(\alpha_{e+1}, \beta_{e+1})$, where $(\alpha(s), \beta(s))$ gives the arc-length parameterization of the line between two vertices (a distance s from the first one). For non-negative integers p and q:

$$\int_0^1 \alpha\,(L_e\lambda)^p\,\beta\,(L_e\lambda)^q\,d\lambda = \frac{1}{p+q+1}\sum_{i=0}^{p}\sum_{j=0}^{q}\frac{\binom{p}{i}\binom{q}{j}}{\binom{p+q}{i+j}}\alpha_{e+1}^{i}\alpha_e^{p-i}\beta_{e+1}^{j}\beta_e^{q-j} \tag{10}$$

$\alpha(s)$ and $\beta(s)$ - the α- and β-coordinates of the point on this segment a distance s from the starting point.

Newell's Method

This describes a way of getting the normal to a plane of points making a shape counter-clockwise. The method of calculating involves adding up the pair of points next to each-other as such:

$$\vec{n_x} = \sum_{i=0}^{n-1}(y_i - y_{i+1})(z_i + z_{i+1}) \qquad \vec{n_y} = \sum_{i=0}^{n-1}(z_i - z_{i+1})(x_i + x_{i+1}) \qquad \vec{n_z} = \sum_{i=0}^{n-1}(x_i - x_{i+1})(y_i + y_{i+1}) \tag{11}$$

Then to find the unit normal vector, one simply normalizes the normal vector to have magnitude 1 (which can be done with the integrated PVector.normalize(); function).

3 | PROGRESS TIMELINE

Week 1

I focused on using the 3D engine implemented in the Processing language and created a basic class upon which I would build the rest of the project, named "Polygon". I also tested some of the coordinate-transfer systems already in the language, however as these do not store the points other methods were used later.

Week 2

This week, I implemented the the Rotation matrix R into the PMatrix3D class, As well as the angular velocity vector to be able to produce a spinning effect of the body used. I then created a function that integrates the rotation matrix using the given angular velocity using the midpoint method. I then added a function to orthogonalize the rotation matrix after each step to avoid the object being deformed or oversized (which was achieved with Grahm-Shmidt orthogonalization)

Week 3

I implemented a force system into the tool that makes it such that the change in velocity, position, angular velocity, rotation matrix depended on a force by mass/tensor of inertia (for a cuboid), and the change was then integrated using a combination of both the midpoint method and Euler method. The force function depended on the point in world space and the force vector, which were both returned to body-space for ease of computation.

Week 4

Week 4, I began to implement code that allows the import of any custom 3D .obj file, which required slight modifications to how other parts were processed. I also made code that sets the bounding box around the 3D object and sets the

inertia tensor to the one for the bounding box for visual testing purposes, and tested it with a simple spring attached to one of the vertices. This worked well and I then gave a seminar explaining the work I had done so far.

Week 5

Week 5, I began making a function to do integrals to get the Tensor of Inertia by the method described in the paper by Brian Mirtich [3], as well using Newell's method to find the normal to each face, and after a lot of debugging had finally made it functional and results were in the range expected.

Week 6

In the final week, I improved the interactivity of the the tool by more easily allowing the adjustment of the size and mass of the object, and also implemented a basic "poking" function that would apply a force on a point close on the object where clicked. I also made this force adjustable, which further betters the experience. I also created some basic shapes whose center of mass differed greatly from its bounding box to better show how the tensor of inertia affects rotation.

4 | CONCLUSION

In conclusion, the tool was made up to create an accurate simulation of the way objects interact with forces and how the Tensor of Inertia affects rotation. I hope the tool is useful for those learning about physical simulations. The code will be available for download as well as some sample object from :

*pesvut.netsoc.ie/researchProject*2019*.html*

ACKNOWLEDGMENTS

I would like to give thanks to my supervisor John Dingliana for assisting me with the knowledge to create this application. I would also like to acknowledge the lecture notes of David Baraff [1] publicly available as a source of my knowledge on the topic of Physically Based Modeling, and also to D. Kleppner and R. Kolenkow [2] for their book "An Introduction to Mechanics" as a helpful source to my knowledge of mechanics

REFERENCES

[1] D. Baraff, "," *Physically Based Modeling*, http://www.cs.cmu.edu/ baraff/sigcourse/index.html, 1999, pp. .

[2] R.J.K. Daniel Kleppner, *An Introduction to Mechanics*, Cambridge University Press, 2010.

[3] B. Mirtich., *Fast and accurate computation of polyhedral mass properties.*, J. Graph. Tools **February 1996** (1996), 31–50.