CSE 167 (FA 2021) Rigid Motion. Final project guides

In this final project topic, we produce a physics-based animation. Specifically, we simulate rigid body motion in 3D.

In terms of affine transformations we learned in the course, it is easy to describe what happens for a rigid body. As a standard description, there is a *model coordinate* where the vertex position (and other attributes) of the geometric object are defined. Then, there is a model matrix $\mathbf{T} \in \mathbb{R}^{4\times4}$ that sends the positions in the model coordinate to the positions in the *world coordinate*:

$$\mathbb{R}^4_{\text{model}} \xrightarrow{\mathbf{T}} \mathbb{R}^4_{\text{world}}. \tag{1}$$

For a rigid body motion, it is $\mathbf{T}$ that changes over time, constituting an animation of object moving in the world. The vertex positions defined under the model coordinate remain static; they are stored in some vertex buffers initially and never require any update. Moreover, for a rigid body, the time-dependent model matrix always takes the form of "first a rotation about the center of mass,[1] followed by a translation":

$$\mathbf{T}(t) = \begin{bmatrix} 1 & & & | \\ & 1 & & \mathbf{b}(t) \\ & & 1 & | \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} & \mathbf{R}(t) & & 0 \\ & & & 0 \\ & & & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{2}$$

Here $\mathbf{R} \in \mathbb{R}^{3\times3}$ is a rotation matrix, that is $\mathbf{R}^\intercal\mathbf{R} = \mathbf{I}$ and $\det(\mathbf{R}) = 1$.

Now, how should one animate $\mathbf{R}(t)$ and $\mathbf{b}(t)$ so that the motion is physically accurate? One interesting fact is that the dynamics of $\mathbf{b}(t)$ and $\mathbf{R}(t)$ are completely decoupled. In other words, we can discuss the two dynamics separately and they don't depend on each other. The dynamics of $\mathbf{b}(t)$ is exactly the same as that of a point mass. For example $\mathbf{b}(t)$ follows

$$\mathbf{b}(t) = \mathbf{b}_0 + \mathbf{v}_0 t + \frac{1}{2}\mathbf{a}t^2 \tag{3}$$

where $\mathbf{b}_0$ is the initial position, $\mathbf{v}_0$ is the initial velocity vector, and $\mathbf{a}$ is some background acceleration vector (such as gravity).[2]

The non-trivial part is the dynamics of $\mathbf{R}(t)$, which is what you will simulate and demonstrate in this final project.

Expectation

You only need to demonstrate a rigid body rotation (without the translation part). The best example to show is the Dzhanibekov effect (see the animation seen in https://en.wikipedia.org/wiki/Tennis_racket_theorem). To validate the correctness of the animation, also show the angular velocity indeed stay on the *Poinsot's ellipsoids* (like the animation near the end of the video https://vimeo.com/286156112). We will explain them later in this document.

After this pure rigid rotation is demonstrated, you could have fun with it by adding translational part (Eq. (3)).

Extra credits will be given to projects with excellent design, exposition, and/or demonstration.

[1] We assume that the origin of the model coordinate is the center of mass of the model

[2] In general, $\mathbf{b}(t)$ satisfies Newton's second law ($F = ma$).

Animation with GLUT

In the OpenGL code with have so far, the display will be called only when some key is pressed. That is because we setup `glutKeyboardFunc` to some callback function `void keyboard(unsigned char key, int x, int y);` where we demand a `glutPostRedisplay()`. If we want to have our OpenGL program to key computing and update the screen at every frame, we set

```
glutIdleFunc( animation );
// If you want to pause, switch it to glutIdelFunc( NULL );
```

And the callback function can be

```
void animation( void ){
  // Update the state of your animated objects
  ...

  glutPostRedisplay();
}
```

Background

Since we are focusing on just the rotation part without the translation part of rigid body motions, we can ignore the 4th homogeneous coordinate. That is, everything is in $\mathbb{R}^3$, and every transformation between model and world is a $\mathbb{R}^{3\times3}$ matrix. Let $\mathbf{R}(t) \in \mathbb{R}^{3\times3}$ be the rotation matrix as the model matrix.

Note that for rotation matrices, we have

$$\mathbf{R}^{-1} = \mathbf{R}^\intercal. \tag{4}$$

We can use $\mathbf{R}^{-1}$ and $\mathbf{R}^\intercal$ interchangeably.

Angular velocity

What describes the rotation motion is the change in $\mathbf{R}(t)$ over time. Such a change is described by a vector $\boldsymbol{\omega} \in \mathbb{R}^3_{\text{world}}$ (in the world coordinate) called the **angular velocity**. The length of this vector $|\boldsymbol{\omega}|$ is the rotation speed, and the direction $\frac{\boldsymbol{\omega}}{|\boldsymbol{\omega}|}$ is the rotation axis. Put it simply, when an object has an angular velocity $\boldsymbol{\omega}$, after a short amount of time Δt, the new orientation $\mathbf{R}(t+\Delta t)$ will be further rotated by the angular velocity

$$\mathbf{R}(t+\Delta t) = \text{Rotation}\left(\Delta t|\boldsymbol{\omega}|, \frac{\boldsymbol{\omega}}{|\boldsymbol{\omega}|}\right)\mathbf{R}(t). \tag{5}$$

where $\text{Rotation}(\theta, \mathbf{a})$ is the 3D rotation matrix which rotates about axis $\mathbf{a}$ with angle θ.

The vector $\boldsymbol{\omega} \in \mathbb{R}^3_{\text{world}}$ describes how fast the orientation is modified over time. $\boldsymbol{\omega}$ is analogous to velocity. Sometimes we will also look at $\boldsymbol{\omega}$ in the model coordinate, which is given by a straightforward transformation

$$\boldsymbol{\Omega} = \mathbf{R}^{-1}\boldsymbol{\omega}. \tag{6}$$

Kinetic energy

When a body is moving, there is a kinetic energy. The kinetic energy as a function of the angular velocity is a quadratic form

$$E_{\text{kinetic}} = \frac{1}{2}\boldsymbol{\omega}^\intercal \mathbf{M}_{\text{world}}\boldsymbol{\omega} = \frac{1}{2}\boldsymbol{\Omega}^\intercal \mathbf{M}_{\text{model}}\boldsymbol{\Omega} \tag{7}$$

for some symmetric 3×3 (positive-definite) matrix $\mathbf{M}_{\text{world}}$ (for world coordinate) or $\mathbf{M}_{\text{model}}$ (for model coordinate). These matrices are called **moment of inertia**. From (7) and (6) alone we must have the transformation rule:

$$\mathbf{M}_{\text{world}} = \mathbf{R}\mathbf{M}_{\text{model}}\mathbf{R}^{-1}. \tag{8}$$

A fundamental assumption of rigid body motion is that the moment of inertia depends only on the shape of the geometry, which is completely static when we are in the model coordinate. In short:

$$\mathbf{M}_{\text{model}} \quad \text{is time-independent.} \tag{9}$$

A common practice is to set the model coordinate to align with the eigenvectors of $\mathbf{M}_{\text{model}}$ so that we can simply assume

$$\mathbf{M}_{\text{model}} = \begin{bmatrix} \mu_1 & & \\ & \mu_2 & \\ & & \mu_3 \end{bmatrix} \tag{10}$$

for 3 positive numbers μ_1, μ_2, μ_3. Every rigid object, independent how complex the shape is, boils down to these 3 numbers when it comes to the moment of inertia. The rule of thumb is that the width of the geometry in the i-th coordinate direction (in the model coordinate) is proportional to $\sqrt{\mu_i}$.

Angular momentum

The angular momentum of the system is given by the following vector in the world coordinate

$$\mathbf{L} = \mathbf{M}_{\text{world}}\boldsymbol{\omega} \in \mathbb{R}^3_{\text{world}}. \tag{11}$$

One may also look at angular momentum in the model coordinate, which is given by $\boldsymbol{\Lambda} = \mathbf{R}^{-1}\mathbf{L} = \mathbf{M}_{\text{model}}\Omega$.

Conservation law

In a rigid body motion, we have conservation of energy and conservation of angular momentum. More precisely,

- E_{kinetic} remains constant. (Conservation of energy)
- $\mathbf{L}$ remains constant (in the world coordinate). (Conservation of angular momentum)
- $\mathbf{M}_{\text{model}}$ ramins constant. (Rigidity)

These conservation laws alone allow us to develop the animation.

Core Algorithms

Quick recap:

- Moment of inertia in model coordinate $\mathbf{M}_{\text{model}} = \begin{bmatrix} \mu_1 & & \\ & \mu_2 & \\ & & \mu_3 \end{bmatrix}$. This doesn't change over time.
- Moment of inertia in world coordinate $\mathbf{M}_{\text{world}}(t) = \mathbf{R}(t)\mathbf{M}_{\text{model}}\mathbf{R}^{\intercal}(t)$ depends on time since $\mathbf{R}(t)$ does.
- Angular velocity in world coordinate $\boldsymbol{\omega}(t)$. It wobbles over time.

- Angular velocity in the model coordinate $\boldsymbol{\Omega}(t) = \mathbf{R}^{-1}(t)\boldsymbol{\omega}(t)$.
- Angular momentum in the world coordinate $\mathbf{L} = \mathbf{M}_{\text{world}}(t)\boldsymbol{\omega}(t)$ is fixed over time due to conservation of angular momentum.
- Angular momentum in the model coordinate $\boldsymbol{\Lambda}(t) = \mathbf{R}^{\intercal}(t)\mathbf{L} = \mathbf{M}_{\text{model}}\boldsymbol{\Omega}(t)$ depends on time.

A simple algorithm directly based on conservation of angular momentum is

Algorithm 1 Simple first-order method

Input: Initial $\mathbf{R}$, $\boldsymbol{\omega}$, $\mathbf{M}_{\text{model}}$, Δt

Compute world-space moment of inertia $\mathbf{M}_{\text{world}} = \mathbf{R}\mathbf{M}_{\text{model}}\mathbf{R}^{\intercal}$
Compute world-space angular momentum $\mathbf{L} = \mathbf{M}_{\text{world}}\boldsymbol{\omega}$, which is conserved over time.
for frame = 1, 2, . . . (animation sequence) **do**
    Update $\boldsymbol{\omega} \leftarrow \mathbf{M}_{\text{world}}^{-1}\mathbf{L}$
    Update $\mathbf{R} \leftarrow \text{Rotation}(\Delta t|\boldsymbol{\omega}|, \frac{\boldsymbol{\omega}}{|\boldsymbol{\omega}|})\mathbf{R}$
    Recompute $\mathbf{M}_{\text{world}} = \mathbf{R}\mathbf{M}_{\text{model}}\mathbf{R}^{\intercal}$
    Render the object using $\mathbf{R}$ for the model matrix.
end for

This method works well for small Δt. However, for larger Δt and after running the animation for a longer time, the energy will slowly drift off and not be preserved. An improvement for this algorithm is introduced in UCSD by Sam Buss in 2001.[3]

Algorithm 2 Buss' augmented second-order method (pp. 12 of the original paper)

Input: Initial $\mathbf{R}$, $\boldsymbol{\omega}$, $\mathbf{M}_{\text{model}}$, Δt

Compute world-space moment of inertia $\mathbf{M}_{\text{world}} = \mathbf{R}\mathbf{M}_{\text{model}}\mathbf{R}^{\intercal}$
Compute world-space angular momentum $\mathbf{L} = \mathbf{M}_{\text{world}}\boldsymbol{\omega}$, which is conserved over time.
for frame = 1, 2, . . . (animation sequence) **do**
    Update $\boldsymbol{\omega} \leftarrow \mathbf{M}_{\text{world}}^{-1}\mathbf{L}$
    Let $\boldsymbol{\alpha} = -\mathbf{M}_{\text{world}}^{-1}(\boldsymbol{\omega} \times \mathbf{L})$
    Let $\bar{\boldsymbol{\omega}} = \boldsymbol{\omega} + \frac{\Delta t}{2}\boldsymbol{\alpha} + \frac{(\Delta t)^2}{12}(\boldsymbol{\alpha} \times \boldsymbol{\omega})$
    Update $\mathbf{R} \leftarrow \text{Rotation}(\Delta t|\bar{\boldsymbol{\omega}}|, \frac{\bar{\boldsymbol{\omega}}}{|\bar{\boldsymbol{\omega}}|})\mathbf{R}$
    Recompute $\mathbf{M}_{\text{world}} = \mathbf{R}\mathbf{M}_{\text{model}}\mathbf{R}^{\intercal}$
    Render the object using $\mathbf{R}$ for the model matrix.
end for

Buss' algorithm preserves energy much better.

Poinsot's ellipsoids

The two conservation laws (energy and angular momentum) can be visualized by Poinsot's ellipsoids. The conservation of energy says that

$$E_{\text{kinetic}} = \boldsymbol{\Omega}^{\intercal}\mathbf{M}_{\text{model}}\boldsymbol{\Omega} = \text{constant} \tag{12}$$

[3] Original paper "Accurate and Efficient Simulation of Rigid Body Rotations" by Samual Buss, 2001.

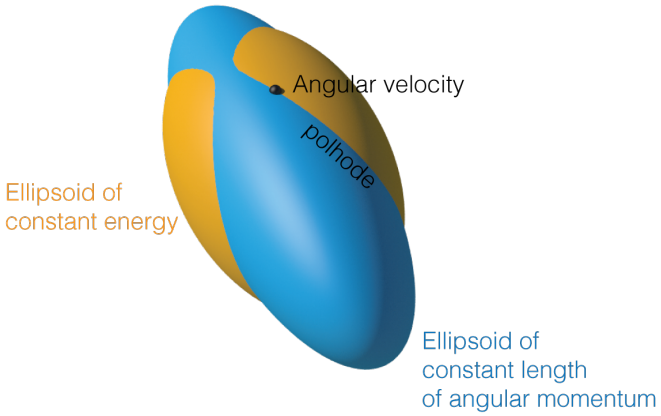

Figure 1 The angular velocity follows the polhode, the intersection of two ellipsoids.

If $\Omega = (x, y, z)^\intercal$, then this equation expands into

$$\mu_1 x^2 + \mu_2 y^2 + \mu_3 z^2 = E_{\text{kinetic}} \tag{13}$$

The set of all (x, y, z) satisfying the equation forms an ellipsoid in the model coordinate with semi-axes $\sqrt{E/\mu_1}$, $\sqrt{E/\mu_2}$, and $\sqrt{E/\mu_3}$ in the x, y, z directions.

As for conservation of angular momentum, we have $\mathbf{L}$ being constant in the world coordinate. In the model coordinate, $\mathbf{\Lambda}(t)$ is not constant. But the $|\mathbf{\Lambda}(t)|^2 = |\mathbf{L}|^2$ will be constant. Recall $\mathbf{\Lambda}(t) = \mathbf{M}_{\text{model}}\Omega(t)$. So,

$$\text{constant} = \mathbf{\Lambda}^\intercal \mathbf{\Lambda} = \mathbf{\Omega}^\intercal \mathbf{M}_{\text{model}}^\intercal \mathbf{M}_{\text{model}} \mathbf{\Omega} = \mathbf{\Omega}^\intercal \begin{bmatrix} \mu_1^2 & & \\ & \mu_2^2 & \\ & & \mu_3^2 \end{bmatrix} \mathbf{\Omega}. \tag{14}$$

This condition for $\mathbf{\Omega}$ forms another ellipsoid:

$$\mu_1^2 x^2 + \mu_2^2 y^2 + \mu_3^2 z^2 = F, \quad F \text{ is some constant} \tag{15}$$

This is the ellipsoid with semi-axes $\sqrt{F}/\mu_1$, $\sqrt{F}/\mu_2$, $\sqrt{F}/\mu_3$.

You can determine the constants E and F at the initial condition. Build the two ellipsoids *in the model coordinate*. To visually verify that our algorithm is doing the correct thing, we render the position $\mathbf{\Omega}$ and see if it lies on *the intersection of both ellipsoids* in the model coordinate. We can also transform both ellipsoid in the world and see if $\boldsymbol{\omega}$ lies on the intersection of the transformed ellipsoids.

The intersection of the two ellipsoids is called the **polhode** (Figure 1).

Tennis racket / Dzhanibekov effect

With the picture of intersecting ellipsoids in mind, it is possible to understand the Dzhanibekov effect.

The Dzhanibekov effect is the observation that sometimes a rigid rotating body will flip its orientation periodically (see the gif in `https://en.wikipedia.org/wiki/Tennis_racket_theorem`). This is somewhat counterintuitive since most of the rotating body we observe (such as the spinning earth, or a spinning top) seems to have a rotation axis pointing in a rather consistent direction. The mystery becomes clearer once we visualize the polhode curve. Suppose

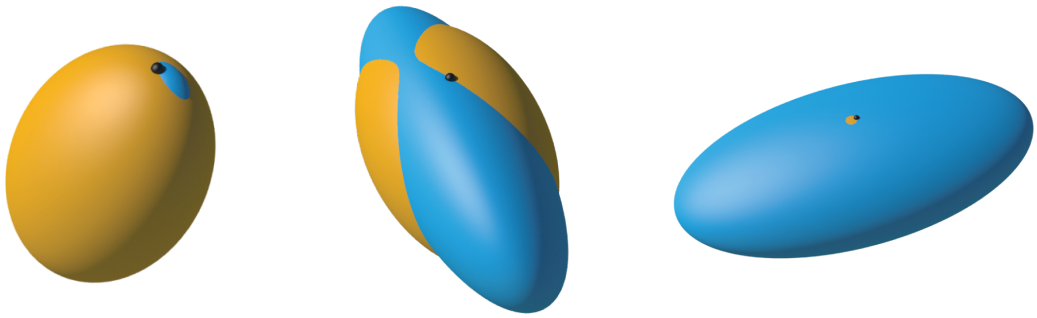

Figure 2 The polhode when the initial angular velocity is close to the axis of (left) smallest μ, (middle) intermediate μ and (right) largest μ, where μ's are diagonal values for the diagonal matrix of the moment of inertia $\mathbf{M}_{\text{model}}$.

$\mathbf{M}_{\text{model}}$ has the entries ordered $0 < \mu_1 < \mu_2 < \mu_3$. If we initialize the angular velocity so that the body rotates roughly around the 1st or the 3rd axis, then the corresponding polholde curve will be quite tiny (Figure 2 left, right). In that case, the angular velocity relative to the model won't deviate too much from the initial value. However, when the initial angular velocity is close to the 2nd axis, we have a long trajectory of the polhode (Figure 2 middle). The rotation axis naturally moves along the polhode and travels to the opposite side in the model coordinate, giving rise to the Dzhanibekov effect. Once we know the cause of the Dzhanibekov effect, it is very easy to reproduce it in real life as long as we know about which axis we should spin the object.