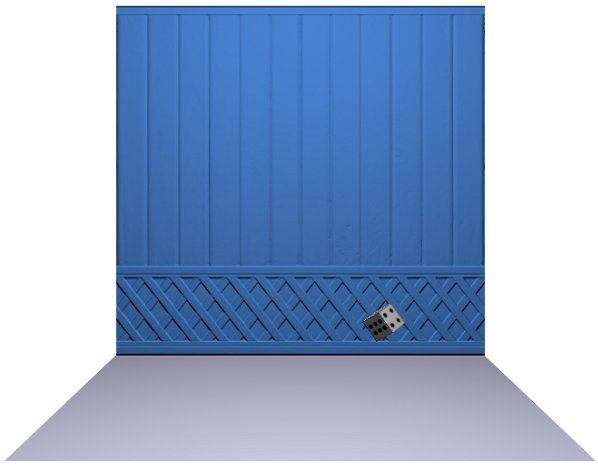
Rigid Body Simulation Practical Work

IGR202 - Computer Graphics & Virtual Reality



Lais Isabelle ALVES DOS SANTOS

January 2023

1 Introduction

The main purpose of this report is to briefly explain the implementation details of a Rigid Body simulation, using physics concepts to move a certain rigid body, in this case a dice. All the coding part for creating the simulation is mainly concentrated in *RigidSolver.hpp* and *quaternion.hpp*. To compile and work as expected, the project files and directories were refactored from the provided ones. The following commands from the *src* directory should be done to properly set the compilation and running.

```
$ mkdir build
$ cmake -B build
$ cd build; make
$ cd ..; ./tpRigid
```

2 Linear Momentum

2.1 Rigid Body Attributes

The first part to be set in the code is the rigid body attributes, which are the mass M, the inertia tensor (part of the angular momentum, but defined at the beginning of the code), which is first calculated in body space, using the notation I^0. Since this parameter acts as the "mass" of the rigid body, it must be conserved during the motion. For this reason, computing the inertia tensor in world space at each time step would add more complexity to the code, hence it is easier to compute this parameter in body space. Because this property does not change over time, it is then used to define the inertia tensor in world space I, through $I(t) = \mathbf{R}(t)I^0\mathbf{R}(t)^T$, by simply recomputing the rotation matrix at each time step.

For many type of shapes modeled as rigid body, it is possible to calculate a different inertia tensor in body space. In the case of a cube, the corresponding matrix is

$$I = \begin{bmatrix} \frac{1}{12}m(h^2+d^2) & 0 & 0 \\ 0 & \frac{1}{12}m(w^2+d^2) & 0 \\ 0 & 0 & \frac{1}{12}m(w^2+h^2) \end{bmatrix}$$

where w is the width, h is the height and d is the depth of the cube.

From I^0, it is possible to compute the inverse of this inertia tensor in order to have the angular velocity of the rigid body $\boldsymbol{\omega}(t)$. For more details on implementation, see the class *Box*.

2.2 Time Integration

The time integration is the update part that will allow the rigid body to move. It is done in the class *RigidSolver* in the function *step*. This part of the code was updated during the entire development, but in order to observe a single linear movement, the following variables were taken into consideration:

$$X(t + \Delta t)$$

$$P(t + \Delta t)$$

which are the rigid body's center of mass position and linear momentum, important to describe a motion considering the body's mass and its linear velocity.

2.3 Force and Linear Momentum

The force is the first thing to be calculated after the step function is called in a time interval of seconds. The force acting on the body each time is the gravitational force $F_g = Mg$, but in the very first step, the dice receives an external force (an impulse) to start the parabolic motion. Testing the function, the results of the linear momentum can be seen in the Figure 1.

3 Angular Momentum

3.1 Torque and Angular velocity

In order to add some rotational motion to the cube, there must be some torque to rotate the body around its correspondent center of mass. Since the torque is computed only when the external force acts, the torque at the other time steps is equal to zero. The calculation of this parameter can be given as

$$\boldsymbol{\tau}(t + \Delta t) = (\mathbf{r}(t) - \mathbf{x}(t)) \times \mathbf{F}(t)$$

(a) Time t_0 (b) Time t_n

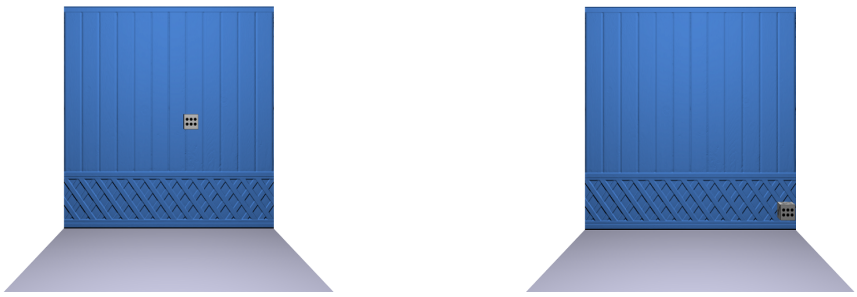


Figure 1: Linear motion of rigid body

where $\mathbf{r}(t) = \mathbf{R}(t)\mathbf{r}^0 + \mathbf{x}(t)$. Then

$$\boldsymbol{\tau}(t + \Delta t) = \mathbf{R}(t)\mathbf{r}^0 \times \mathbf{F}(t)$$

considering $\mathbf{r}^0$ as **vdata0[0]** in the code (the first vertex position - as stated in the guidance document provided).

This torque contributes to the angular momentum L and the angular velocity $\boldsymbol{\omega}$. However, just setting up this parameters contributes to the physical rotation of the rigid body. To actually see some simulation, the rotation matrix must receive these parameters to change the body's angular position. In this case, the rotation matrix is updated on the *step* function as

$$\mathbf{R}(t + \Delta t) = \mathbf{R}(t) + \Delta t\boldsymbol{\omega}(t)_*\mathbf{R}(t)$$

where $\boldsymbol{\omega}(t)_*$ is the skew matrix of the angular velocity - an easier way to compute the cross product.

Unfortunately, computing the rotation matrix as $\mathbf{R}(t + \Delta t)$ can lead to some cumulative errors, since this result does not guarantee that the matrix remains orthonormal. Thus, the cube can rotate, but it can also scale, shrink, and perform other transformations. The result of that can be seen in Figure 2.

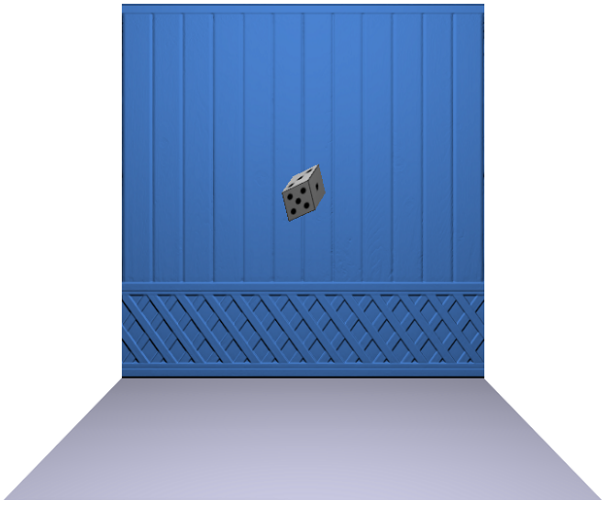


Figure 2: Rotation

3.2 Quaternion

The solution to the previous problem with the rotation matrix is the *quaternion*. To code it correctly, a class with the same name was created and inside it the functions to compute this special matrix were declared and implemented. When this part was tested, the quaternion was always zero and the dice did not make any movement other than linear. Therefore, instead of computing the quaternion as

$$\mathbf{Q}(t + \Delta t) = \mathbf{Q}\tilde{\omega}\frac{1}{2}$$

directly, the product of $\mathbf{Q}$ and $\tilde{\boldsymbol{\omega}}$ was done using the *product* function available in the quaternion class, as this is a product of two quaternion variables. The rotation matrix was then associated to the normalized quaternion matrix, which allows a "perfect" motion. Figure 3 shows the final result.

(a) Time t_0 (b) Time t_n

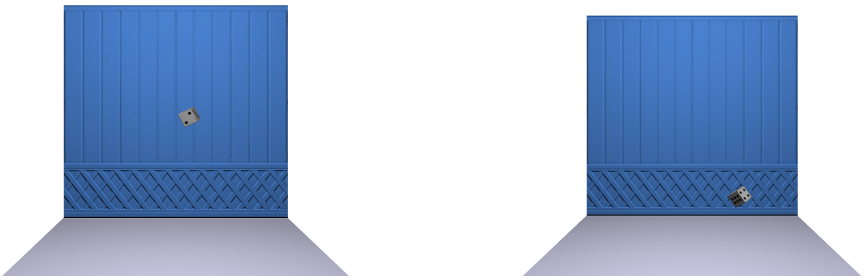


Figure 3: Rotation motion of rigid body - use of *quaternion*

4 Conclusions

Finally, the implementation of the rigid body simulation allowed to consolidate the knowledge taught during the Physics Rigid Body Simulation lessons, mainly on the quaternion part, which is a bit confusing in a theoretical way, but once the key details of this matrix are properly understood, it allows to have a good simulation.