The Linear Complimentarity Problem and its Applications in Physics Based Animation

Rohan Kodati

August 2022

Abstract

The Linear complimentarity problem(LCP) is the main way to model contact between multiple rigid objects in physics based animation. It is widely used in various simulations; however, it has always been hard to understand and implement. Through this paper we will propose an implementation for a simulation which models the motion of free-falling spheres within a confined space using the LCP as the way to represent rigid body collisions.

1 Introduction

The Linear Complentarity Problem is widely used in various fields, including robotics, economics, and, the focus of this paper, physics based simulations. Many have formulated LCPs to model rigid body contact between various objects, but we will focus on the contact between spheres. The main issue with implementing the LCP in rigid body contacts is formulating the problem, and then setting up the equations to be readily solvable.
First, We will go over the Linear Complimentarity Problem to gain an in-depth understanding of it from an algebraic standpoint. From there we will go over the algorithm which we will be using to solve the LCP.
Once we go over the LCP, we are ready to talk about the simulation, mainly how we are simulating the motion of the spheres in a constrained space, and then how we are formulating the LCP to be able to resolve rigid body contacts.

2 The Linear Complementarity Problem

The LCP can seem fairly complicated, so we will break it down to the one dimensional case, and then build it up to the n-dimensional case.

2.1 1-Dimensional LCP

In one dimenstion, the complimentarity problem has two variables, x and y where $x, y \in \mathbf{R}$. The Linear Complimentarity Problem can be stated as finding solutions x and y such that they satisfy the complemtarity constraint: $y > 0 \implies x = 0$ or $x > 0 \implies y = 0$. The constraint states that either of the conditions $x > 0$ and $y = 0$ or $y > 0$ and $x = 0$ must be satisfied. The LCP is conventionally compactly written as $0 \leq y \perp x \geq 0$. It can be seen that the solution to the LCP in one dimension, if represented on a cartesian coordinate plane, would be the positive y-axis and the positive x-axis, creating a corner shape.

Next we can add a linear relationship between y and x to find more concrete solutions: $y = ax + b$ where $a, b \in \mathbf{R}$. This added relationship creates a solution space which is the intersection points between the line $y = ax + b$ and the corner shape we discussed before. This also allows us to reformulate the problem into various formats which allow us to better understand solutions in different contexts. The first reformulation is $y = ax + b, y \geq 0, x \geq 0$, and $xy = 0$. We can also rewrite it by replacing y with $ax + b$: $ax + b \geq 0, x \geq 0$, and $x(ax + b) = 0$. We can also set up the problem as an optimization problem by restating the last part of the reformulation as $x^* = \arg\min_x x(ax + b)$ subject to $ax + b \geq 0, x \geq 0$.

Another reformulation, which we will mainly be using, is the minimum map formulation. The minimum map function can be written as $h(x, y) = min(x, y)$ where $min(x, y)$ returns x if $x < y$ and returns y otherwise. The minimum map reformulation would be $h(x, y) = 0$. This can be proven because if x and y are solutions to the LCP, then we have 2 cases: $x > 0$ and $y = 0$, or $y > 0$ and $x = 0$ and in both cases the minimum map function will return 0. There are 4 cases where the minimum map function does not return 0: $x > 0$ and $y > 0$, $y = 0$ and $x = 0$, $y < 0$, or $x < 0$, and in each of these cases x and y will not satisfy the LCP.

2.2 n-Dimensional LCP

We can generalize the 1-Dimensional LCP to N-Dimensions fairly easily. First we must define some variables: $\mathbf{x}, \mathbf{b} \in \mathbf{R}^N$ and $\mathbf{A} \in \mathbf{R}^{N \times N}$. We set $\mathbf{y} = \mathbf{Ax} + \mathbf{b}$. The LCP formulation with $\mathbf{x}$ and $\mathbf{y}$ could then be written as: for each i:[1,...,N], $0 \leq \mathbf{y}_i = (\mathbf{Ax} + \mathbf{b})_i \perp \mathbf{x}_i \geq 0$. It can also be rewritten as $\mathbf{x} \geq \mathbf{0}, \mathbf{Ax} + \mathbf{b} \geq \mathbf{0}$, and $\mathbf{x}^T(\mathbf{Ax} + \mathbf{b}) = 0$ where the $\geq$ sign is an element wise $\geq$ and $\mathbf{0}$ is a N-dimensional vector of 0's. The minimum map reformulation for an N-Dimensional LCP is written as $\mathbf{H}(\mathbf{x}) = \mathbf{H}(\mathbf{x}, \mathbf{y}) = [h(\mathbf{x}_1, \mathbf{y}_1), ..., h(\mathbf{x}_N, \mathbf{y}_N)] = \mathbf{0}$.

3 Formulating an LCP to Model Rigid Body Contact

The contact model is fairly complicated and can be broken down into two steps, forming an LCP for a frictionless contact and using the frictionless contact to form an LCP for the contact with friction.

3.1 Frictionless Contact Model

Before we begin forming the LCP, we need to specify some variables. The simulation consists only of multiple spheres. n is the number of spheres in the system and m is the number of collisions in the system. To implement the model, we also need to go over some physics. λ_k will be the magnitude of the impulse acting on collision k. The model also takes place in a 3-Dimensional space so that we can split the velocity of each sphere into its x, y, and z-components. We can set s_i^k to be the k-component of the velocity of sphere i. The formulation gets somewhat confusing starting from here, so we will break the model formulation down to the 1-collision case.

3.1.1 Single Collision Contact Model

The next step is stating the LCP. We know by general physics that when two objects are in contact, there must be a collision force, or impulse, acting upon them to dictate their movement after the collision. We also know that if the spheres are not in contact, then there will be no impulse between them. Thus we can state the LCP: $d \geq 0 \perp 0 \leq \lambda$ where d is the distance between the two spheres and λ is the impulse acting upon the spheres by the collision. Modeling the contact using the distance between the spheres makes the calculations long and tricky, so we can change the LCP to be $\mathbf{v}_n = \frac{\partial d}{\partial t} \geq 0 \perp 0 \leq \lambda$ (1). This basically states that if the magnitude of the normal component of the relative velocity between two spheres is not 0, then there cannot be an impulse between them and vice-versa. This LCP also helps satisfy the non-penetration constraint which states that the spheres cannot penetrate each other. The normal component is the component of the relative velocity which is parallel to the normal vector in this collision. The normal vector is the vector upon which the impulse will act on the spheres in the contact.

The next step is finding a linear relationship between $\mathbf{v}_n$ and λ. We can write the physics-based equation for the final velocity of an object in a 1-Dimensional space after an impulse has acted upon it: $v_f = \frac{\lambda}{w} + v_i$ where v_f is the velocity of the object due to the collision, w is the weight of the object, and v_i is the initial velocity of the object. To generalize for the 3-Dimensional case, we need to write the equation for each of the x, y, and z-components. We also know that the impulse will only act on the normal vector for the collision. We can say that the normal vector is (x, y, z), and we define the normal vector to be a unit vector. From this is it easy to see that the x-component of the post-collision velocity of the first sphere would be $\frac{x * \lambda_1}{w_1} + s_1^x$ where w_1 is the weight of the sphere in question, and λ_1

is the impulse for the collision. From here we can make a simple vector to represent the post-collision velocities of the two balls which make up this single contact. We set up the final vector with each of the components grouped together. We let k_i be the k-component of the collision acting on the ith sphere. Our post-collision velocities vector would look like this:

$$\begin{bmatrix} \frac{x_1 * \lambda_1}{w_1} + s_1^x \\ \frac{x_2 * \lambda_1}{w_2} + s_2^x \\ \frac{y_1 * \lambda_1}{w_1} + s_1^y \\ \frac{y_2 * \lambda_1}{w_2} + s_2^y \\ \frac{z_1 * \lambda_1}{w_1} + s_1^z \\ \frac{z_2 * \lambda_1}{w_2} + s_2^z \end{bmatrix}$$

From here it is easy to break down this vector into matrix components. We can clearly write $\mathbf{s} \in \mathbf{R}^6$ as the column vector of the initial velocities, $\boldsymbol{\lambda} \in \mathbf{R}^1$ as the column vector of impulses. To visualize them we can write them out as $s = \begin{bmatrix} s_1^x & s_2^x & s_1^y & s_2^y & s_1^z & s_2^z \end{bmatrix}^T$ and $\boldsymbol{\lambda} = [\lambda_1]$. Next, we need to define the weight matrix $\mathbf{W} \in \mathbf{R}^{6x6}$. $\mathbf{w} \in \mathbf{R}^{2x2}$ is a diagonal matrix where the ith entry, the value in the $[i, i]$ spot, is the weight of the ith sphere. In this case $\mathbf{w} = \begin{bmatrix} w_1 & 0 \\ 0 & w_2 \end{bmatrix}$. $\mathbf{W}$ is a block diagonal matrix that is constructed by letting each element on the diagonal be $\mathbf{w}$. The last matrix we need to define is the contact Jacobian. The Jacobian is the matrix that will encode the components of the normal vector as it relates to each sphere and collision. The Jacobian will have the x-component of the normal vector acting on the 1st sphere, then the x-component of the normal acting on the second sphere, and then the y and z-components in the same format. We will call this $\mathbf{J} \in \mathbf{R}^6$ and it is a row vector. In this example, $\mathbf{J} = \begin{bmatrix} x_1 & x_2 & y_1 & y_2 & z_1 & z_2 \end{bmatrix}$. Using the defined matrices, we can rewrite the post-collision velocity matrix as $\mathbf{W}^{-1}\mathbf{J}^T\boldsymbol{\lambda} + \mathbf{s}$. It seems as if we are done and ready to move on to formulating the LCP with more collisions, but, if we remember correctly, the LCP for the impulse is between the impulse and $\mathbf{v}_n$, which is the normal component of the relative velocity between the spheres. It seems daunting to adjust our equation, but it is actually fairly simple. All we need to do to turn our component-based post-collision velocity into the magnitude of the relative velocity between the balls in the collision is to multiply the post-collision velocity vector by the Jacobian. This gives us our equation $\mathbf{v}_n = (\mathbf{J}\mathbf{W}^{-1}\mathbf{J}^T)\boldsymbol{\lambda} + \mathbf{J}\mathbf{s}$. Putting it all together, we have formulated an LCP for the single collision case: $(\mathbf{J}\mathbf{W}^{-1}\mathbf{J}^T)\boldsymbol{\lambda} + \mathbf{J}\mathbf{s} \geq 0 \perp 0 \leq \boldsymbol{\lambda}$ (2).

3.1.2 M-Collision Contact Model

Now that we understand the physics and math behind formulating the LCP for a collision, all that is left is increasing the dimensions of each of the matrix components to satisfy a system with m collisions and n spheres. The first step is creating the weight matrix $\mathbf{W} \in \mathbf{R}^{3nx3n}$. $\mathbf{W}$ is a diagonal matrix, as before, and it is constructed from three smaller weight matrices $\mathbf{w}^{nxn}$ where the ith element of $\mathbf{w}$ is the weight of the ith sphere. $\mathbf{W} = \begin{bmatrix} w & 0 & 0 \\ 0 & w & 0 \\ 0 & 0 & w \end{bmatrix}$. The next matrix we need to adjust is the impulse column vector $\boldsymbol{\lambda} \in \mathbf{R}^m$. The ith element in $\boldsymbol{\lambda}$ is λ_i, the magnitude of the impulse acting on the spheres in the ith collision. The next vector to adjust is $\mathbf{s} \in \mathbf{3n}$. $\mathbf{s}$ is a column vector where the first n elements are the x-components of the initial velocities of each of the spheres, then the next n elements are the y-components, and then the last n elements are the z-components. This is almost the exact same as in the single collision case, but is extended for every sphere in the system. The final matrix that needs to be updated is the Jacobian. The contact Jacobian, $\mathbf{J} \in \mathbf{R}^{mx3n}$, is constructed by having the ith row of $\mathbf{J}$ represent the components of the ith collision acting on each sphere. Each row will be constructed similar to the Jacobian in the single collision case. The first n elements of the ith row are the x-components of the normals acting upon the spheres in the ith collision, then the next n elements are the y-components, and the last n elements are the z-components. Since the objects are spheres, each collision will only have two spheres, so each row will only have values in the parts of the x, y, and z portions of the row which correspond to those two spheres. Using these matrices, we can construct an LCP which looks exactly the same as the LCP in the single collision case with different matrices: $(\mathbf{J}\mathbf{W}^{-1}\mathbf{J}^T)\boldsymbol{\lambda} + \mathbf{J}\mathbf{s} \geq 0 \perp 0 \leq \boldsymbol{\lambda}$.

3.2 Contact Model with Friction

Formulating the LCP for a model with friction uses the same LCP which we had in the friction-less case, but also has two more LCP's added onto it. We can once again build the contact for the single collision case and then generalize it to have m collisions.

3.2.1 Single Collision Contact Model with Friction

Since we have only one collision, we only have one normal impulse, which we will call λ_n. To model friction in the contact, we will use the 3-Dimensional linearized friction model which uses a polyhedral cone. Using this model, we have two more LCPs to define. First, we must define some more new terms. The first is $\mathbf{t}$, a set of k unit vectors that will define the polyhedral cone. The next is $\boldsymbol{\lambda}_t$ which is a vector of the tangential impulses due to friction, and $\boldsymbol{\lambda}_t = [\lambda_{t_1} ... \lambda_{t_k}]^T$. The first of the new LCPs is $\beta\mathbf{e} + \mathbf{v}_t \geq 0 \perp \boldsymbol{\lambda}_t \geq 0$ (1). $\mathbf{v}_t \in \mathbf{R}^k$ is a column vector with the tangential components of the post-collision velocities for each tangential vector,and $\mathbf{v}_t$ can be calculated in the same way as $\mathbf{v}_n$, $\beta \in \mathbf{R}$ is a constant, and $\mathbf{e} \in \mathbf{R}^k$ is a column vector of 1's. This equation mainly serves the purpose of satisfying the non-penetration constraint just as the LCP in the friction-less model did. The second LCP is the one that implements the polyhedral friction cone: $0 \leq (\mu\lambda_n - \sum_{i=1}^{k} \lambda_{t_i}) \perp \beta \geq 0$. This second LCP chooses the direction along the polyhedral cone which the tengential impulse will act (1). We can also notice that if $\beta > 0$, then the friction impulse must lie on the polyhedral cone, and if $\beta = 0$, then the two new LCPs will model static friction, which means there will be no sliding friction.

The next step is combining all three of these LCPs to make one LCP which we can solve. We can create $\mathbf{v} = [\mathbf{v}_n \ \mathbf{v}_t]^T$ and $\boldsymbol{\lambda} = [\lambda_n \ \boldsymbol{\lambda}_t \ \beta]^T$. Next we have to define the new Jacobians. There will be k new Jacobians in the model. The ith Jacobian, $\mathbf{J}_{t_i}$, will correspond to the ith tangential vector in $\mathbf{t}$ and λ_{t_i}. Similar to the Jacobian we created in the friction-less model with a single contact, $\mathbf{J}_{t_i} \in \mathbf{R}^6$ will be a row vector with the first two elements being the x-components of the ith tangential vector acting on each of the spheres, the next two will be the y-components, and the last two will be the z-components. Once we create each of the Jacobians, we can create $\mathbf{J}_t \in \mathbf{R}^{k\text{x}6}$ where $\mathbf{J}_t = [\mathbf{J}_{t_1} ... \mathbf{J}_{t_k}]$. Using these newly defined matrices and the matrices we defined earlier in the friction-less single contact model, we can create a new LCP. The trickiest part of implementing the new LCP is figuring out how to encode $\sum_{i=1}^{k} \lambda_{t_i}$, but upon further inspection it is easy to see that $\sum_{i=1}^{k} \lambda_{t_i} = \mathbf{e}^t\boldsymbol{\lambda}_t$ (2). From here we can create the matrices we will directly use to create a linear relationship between $\mathbf{v}$ and $\boldsymbol{\lambda}$. The first is the column vector $b = [\mathbf{J}_n\mathbf{s} \ \mathbf{J}_t\mathbf{s} \ 0]^T$ where $b \in \mathbf{R}^{k+2}$, J_n is the Jacobian for the normal impulses as defined in section 3.1.1, and $\mathbf{s}$ is the one we defined in 3.1.1. Next we need to specify the matrix $\mathbf{A} \in \mathbf{R}^{(k+2)\text{x}(k+2)}$. We can set $A = \begin{bmatrix} \mathbf{B}_{nn} & \mathbf{B}_{nt} & 0 \\ \mathbf{B}_{tn} & \mathbf{B}_{tt} & \mathbf{e} \\ \mu & -\mathbf{e}^T & 0 \end{bmatrix}$. Each $\mathbf{B}_{ik} = \mathbf{J}_i\mathbf{W}^-1\mathbf{J}_k^T$ and $\mu \in \mathbf{R}$ is the coefficient of friction from the second new LCP. Using these new matrices we can write our new, fully encompassing LCP for the single contact friction model as $0 \leq A\boldsymbol{\lambda} + b \perp \boldsymbol{\lambda} \geq 0$ (2).

3.2.2 M-Collision Contact Model with Friction

The process of adjusting the matrices in the friction model for m collisions and n spheres is very similar to how we adjusted the matrices in the friction-less model. We first define $\mathbf{s}$, $\mathbf{W}$, and normal Jacobian $\mathbf{J}_n$ as we did in section 3.1.2. The next step is defining $\boldsymbol{\lambda} \in \mathbf{R}^{(m(k+2))}$. Each collision will have its own set of λ_n, $\boldsymbol{\lambda}_t$, and β. To encode all of them, we can set the first m elements of $\boldsymbol{\lambda}$ to be the normal impulses of each of the collisions, then set the next m elements to be the first tangential impulses for each collision, then the next m elements would be the second tangential impulses for each collision, and repeat this process until we get through all k of the tangential impulses. Finally we set the last m to be the β values for each collision. The next matrices we need to adjust is each $\mathbf{J}_{t_i} \in \mathbf{R}^{m\text{x}3n}$. The process for adjusting each $\mathbf{J}_{t_i}$ is the same as the process for adjusting $\mathbf{J}_n$. We let the jth row of $\mathbf{J}_{t_i}$ encode the ith tangential vector of the jth collision acting on each of the spheres in the system. The first n elements in the row will have the x-components of ith tangential vector of the jth collision acting on the spheres. Just as we explained in section 3.1.2, there will only be 6 non-zero values in in the row which correspond to both of the spheres in the collision. Now that we have defined each $\mathbf{J}_{t_i}$, we can create $\mathbf{J}_t \in \mathbf{R}^{km\text{x}3n}$. We let $\mathbf{J}_t = [\mathbf{J}_{t_1} ... \mathbf{J}_{t_k}]$. Now we can form the $A \in \mathbf{R}^{m(k+2)xm(k+2)}$ and $b \in \mathbf{R}^{m(k+2)}$ matrices which we can use to create the all encompassing LCP for the m-collision friction model. We have $b = [\mathbf{J}_n\mathbf{s} \ \mathbf{J}_t\mathbf{s} \ \mathbf{0}]^T$. $\mathbf{0}$ is

an m-element column vector of 0s. Next, we can define $A = \begin{bmatrix} \mathbf{B}_{nn} & \mathbf{B}_{nt} & \mathbf{0} \\ \mathbf{B}_{tn} & \mathbf{B}_{tt} & \mathbf{e} \\ \boldsymbol{\mu} & -\mathbf{e}^T & \mathbf{0} \end{bmatrix}$. $\boldsymbol{\mu} \in \mathbf{R}^{mxm}$ is a diagonal matrix where every element is μ. Each $\mathbf{0} \in \mathbf{R}^{mxm}$ is a square matrix of all 0s. Finally, each $\mathbf{e} \in \mathbf{R}^{kmxm}$ is a block column vector where each block is an mxm diagonal matrix of 1s. Once we put all of this together we have the full LCP for a system of collisions with friction: $0 \leq A\boldsymbol{\lambda} + b \perp \boldsymbol{\lambda} \geq 0$.

4 Creating the Simulation

The simulation has multiple parts, so we can go over them in detail and in order which they show up in the algorithm. We will not go over any specific code, but just have the general algorithm for each of the parts. The frist step is undersrtanding the simulation. The simulation is of $n \in \mathbf{Z}^+$ spheres of random radii within a large hollow sphere. The spheres start in random positions, and their motion is dictated by gravity and the rigid body contacts between all of the spheres in the system, including the outer sphere.

4.1 Simulation Initialization

The first step is defining the variables. We set n to be the number of spheres, R to be the radius of the hollow, outer sphere, dt is the time step, t_f is the total time which the simulation will run, t is the current time and will be initialized to 0, k is the number of tangential vectors we will use to define the polyhedral cone as defined in section 3.2.1, and μ is the coefficient of friction for the system.

The next step is randomizing the location of the spheres and their radii. We pick the radii are i.i.d using a uniform distribution between [0.7, 1.2]. Once we pick the radius, r, for this sphere, we choose a random unit vector in the unit sphere around the origin and then multiply it by a random uniformly chosen magnitude between [0, $R-r$]. This allows the spheres to have a uniformly random position within the hollow sphere. We store each of the radii in an array and the positions within a different array. The positions vector encodes the location of the center of each sphere in a 3-Dimensional space. Next, we create an array containing the weight of each of the spheres, which is just r^3. Finally, we create a velocity vector that is constructed to be exactly the same as $\mathbf{s}$ from section 3.1.2. We let every value in the velocity vector be 0 since all spheres start at rest. Next, we need to create the $\mathbf{W}$ matrix from section 3.1.2. We take each of the values in the weight array to make the smaller diagonal matrix $\mathbf{w}$, combine them as specified in section 3 to make $\mathbf{W}$, and then invert it because we only use the inverted weight matrix. The last step of the initialization is encoding the outer hollow sphere. The collisions between the outer sphere and the smaller spheres work under the same conditions as the LCP, but we want the outer sphere to be immovable and have an infinite mass. To emulate this we have to add the outer sphere to the velocities vector and the inverted weight matrix because those are the only ones we use in the contact calculations. To add it to the velocities vector, we add more elements to the vector such that the new elements are corresponding to the first sphere in the system. We set each of these velocity components to be 0 as well. To update the inverted weight matrix, we add elements to the matrix in the positions which correspond to the first sphere in the system and make those values 0. Since it is the inverted weight matrix, this would emulate the sphere having infinite weight.

4.2 Timestep

We need to update the simulation at every time step so that we can get a clear simulation of the sphere's moving. The smaller the time step, dt, is, the more accurate the simulation will be. The downfall of choosing a small dt, however, is that it will take a much longer time to render since there are many more calculations to be made. This section will outline what happens at every time step as the simulation is running.

The first part of the time step is updating the positions and velocity vectors. To simulate a gravitational force acting on the spheres we need to decrease the y-component of the velocities of each of the spheres in the system, except for the hollow outer sphere, by $9.81 * dt$. This equation is from the kinematic equations from physics. The next step is updating the positions of every sphere except for

the outer sphere. The positions of each of the spheres will be updated by the components of its velocity multiplied by the time step. To show a concrete example, we update the position and velocity of a certain sphere where its position at time t is (x^t, y^t, z^t) and its velocity is (v_x^t, v_y^t, v_z^t). We update the velocities for gravity by doing $(v_x^t, v_y^t, v_z^t) = (v_x^t, v_y^t - (dt * 9.81), v_z^t)$, and then we update the positions by doing $(x^{t+1}, y^{t+1}, z^{t+1}) = (x^t + (dt * v_x^t), y^t + (dt * v_y^t), z^t + (dt * v_z^t))$. Using these equations, we update the positions array and the velocities vector for the corresponding parts.

4.2.1 Collision Resolution

The next part of the time step is collision detection. The first set of collisions we need to detect are the collisions between the outer hollow sphere and the other spheres in the system. To detect if a sphere is colliding with the outer sphere, we check if the magnitude of the position vector of the sphere plus its radius is larger than the radius of the outer sphere: $||(x^t, y^t, z^t)|| + r \geq R$. The next set of collisions we need to detect are between the spheres in the system that are not the outer sphere. There is a collision between two spheres if magnitude of the difference in their positions is less than the sum of their radii. We can show this concretely by having two spheres with radii r_1 and r_2 and position vectors p_1 and p_2. There would be a collision between these two spheres if $||p_1 - p_2|| \leq r_1 + r_2$.

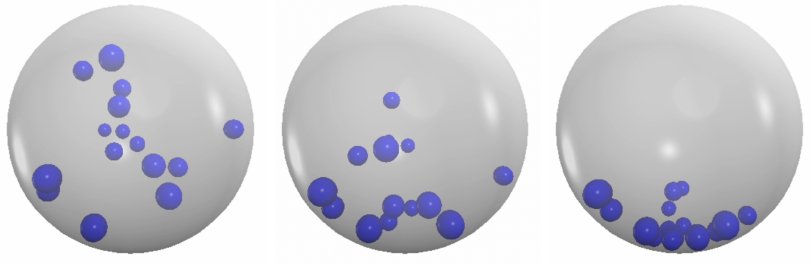
Once we detect the collisions, we need to create the A and b matrices as described in section 3.2.2 to solve the LCP. Once we solve the LCP, we can use the solution to update the velocities of the spheres. We already described how to form the A and b matrices in section 3.2.2. The only part of the formulation we need to specify is how we are finding the normals and tangential vectors for each collision. Calculating the normal for the collision is different in the outer sphere case versus the collisions between the mobile spheres. We will go through the the the calculation for the normal in a collision with the outer sphere using an example with a sphere with position vector p. The normal vector acting on the outer sphere is the norm of the position vector of the smaller sphere, $\frac{p}{||p||}$, and the normal vector acting on the smaller sphere is the norm of the negative of the position vector of the smaller sphere, $\frac{-p}{||-p||}$. Next we need to calculate the normals for the collisions between two mobile spheres. We can let the position vector of the first sphere in the collision be p_1 and we let the position of the second sphere in the collision be p_2. The normal of the impulse acting on first sphere will be $\frac{p_1 - p_2}{||p_1 - p_2||}$ and the normal of the impulse acting on the second sphere is $\frac{p_2 - p_1}{||p_2 - p_1||}$. Calculating the tangential vectors is the same for both cases of collisions. The first step is finding the tangential vectors for the first sphere in the collision by finding any unit vector that is orthogonal to one of the normals of the collision and then rotating that first tangential vector around the normal $k-1$ times. This creates k tangential unit vectors for the first ball, so then we negate each of them to find the corresponding tangential vectors for the second sphere. Once we have these vectors, we form the Jacobians and create the LCP as described in section 3.2.2.

Once we have the LCP, we can solve it to find the values of the impulses acting on each collision. There are multiple different algorithms to solve the LCP, but the one we found to be the best was the Minimum Map Newton Method. Using the solution, we will update the velocities of the mobile spheres. The $\boldsymbol{\lambda}$ has m β values which will not be used to calculate the new velocities, so we remove them from $\boldsymbol{\lambda}$. We will use the equation which we discussed in section 3.1.1, $v_f = \frac{\lambda}{w} + v_i$. First, we need to break the total impulse acting on each sphere into the x, y, and z-components and then divide each component by its weight. Once we do that, we can add it to the corresponding component of the sphere's velocity. To find the components of the impulse we multiply the new $\boldsymbol{\lambda}$ by the complete Jacobian. In this case we let the complete Jacobian $\mathbf{J} = [\mathbf{J}_n \ \mathbf{J}_t]$. We will have this new encompassing component-based impulse vector be $\lambda = \mathbf{J}^T\boldsymbol{\lambda}$. The first $n+1$ components of this vector will be the x-components of the impulses acting on each of the spheres, including the outer sphere, the next $n+1$ components of this vector will be the y-components, and the last $n+1$ components of this vector will be the z-components. Using these components we can update the velocities of the spheres in the system. We can describe this equation with an example of updating the sphere's velocity where its velocity at time t is (v_x^t, v_y^t, v_z^t), the impulse acting on it is $(\lambda_x, \lambda_y, \lambda_z)$, and the weight of the sphere is w. We update the velocity of the sphere by doing $(v_x^{t+1}, v_y^{t+1}, v_z^{t+1}) = (v_x^t + \frac{\lambda_x}{w}, v_y^t + \frac{\lambda_y}{w}, v_z^t + \frac{\lambda_z}{w})$. Once we do this the last step is to update the current time t by adding dt to it.

4.3 Numerical Implementation

The best way to create a good visual of the simulation, from my experiences of working on this project, is to save all of the positions of the spheres at each time step and the radii of the spheres. Once we save all of these we can loop over all of the data to simulate the spheres moving. I used **vpython** to visualize the spheres.

Figure 1. Visualization of a Simulation with 15 Spheres at Various Times



0 Seconds 10 Seconds 20 Seconds

Acknowledgements

I would like to thank Dr. Shravan Veerapaneni and Dr. Saibal De for helping me through this project by giving me valuable feedback and participating in intellectual discussions about this topic. I would also like to thank the University of Michigan REU program for letting me work on this project.

References

[1] Jan Bender, Kenny Erleben, Jeff Trinkle, and Erwin Coumans. Interactive Simulation of Rigid Body Dynamics in Computer Graphics. In Marie-Paule Cani and Fabio Ganovelli, editors, EG 2012 - State of the Art Reports, pages 95–134, Cagliari, Sardinia, Italy, 2012. Eurographics Association.

[2] Niebe, Sarah, and Kenny Erleben. Numerical Methods for Linear Complementarity Problems in Physics-Based Animation. Association for Computing Machinery, 2015.