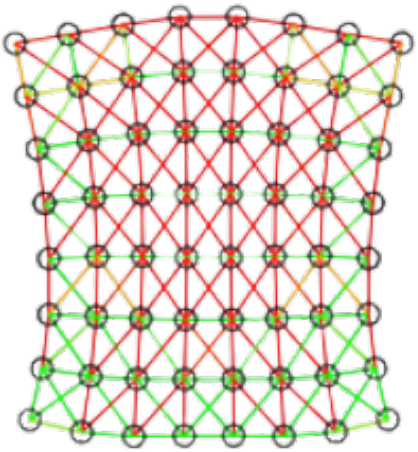
Modelling Constrained 2D Particle-Based Systems

To what extent can Penalty Constraints and Relaxed Geometric Constraints model 2D particle-based constrained dynamic systems?

Subject: Math



Word count: 3999

Contents

1 Introduction

Since the topic of choice is quite specific, this essay will begin with a breakdown of the research question. Constrained 2D particle-based systems comprise of a particle system and a set of constraints. Essentially, they are 2D physics simulations where the particles are "atoms", and the constraints are "bonds". Firstly, a particle-based system consists of a group of free-moving circles that can travel across a 2D plane. These particles (circles) have a defined radius, mass, position, velocity, and force. Moreover, particles can also interact with the surrounding given pre-defined conditions. Constraints are the pre-defined mathematical conditions that control how the particles behave, just like how a string defines the path of a pendulum. It is possible to have multiple constraints, and such systems of constraints can work together to influence the behaviour of the particles. Therefore, a constrained particle-based system is simply a set of particles that are controlled by a set of constraints. A double pendulum would be considered a constrained particle system. When visualized, systems of particles and constraints can form mesh-like objects that react under external forces (Figure 6). Some examples of common constraints include bend constraints, distance constraints, and area constraints. The goal of this essay is to establish a method that applies a mathematical constraint on a set of particles such that the system can also respond to external forces. Section 2 will formally introduce the fundamental setup of the model. Later sections will generalize, compare, and analyze the effectiveness of the penalty method (Equation 23) and relaxed geometric method (Equation 38). Further discussions about the effectiveness of each method will also be included.

A simple programmatic implementation is provided to help visualize the mathematics within this essay (Appendix A). All figures are also created by the author.

1.1 Notation

x, v, a, F	Vector quantities: position, velocity, acceleration, and force
m, r	Scalar quantities: mass and radius
$\dot{F}$	Time derivative; equivalent to $\frac{dF}{dt}$
$q_{j,i}$	the i^{th} element of q_j
Row vector notation $[x_1 \ x_2 \ \dots \ x_n]^T$	Equivalent to the column form, except it saves space $\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$
$\nabla_{q_j} C_j(q_j)$	The gradient of C in terms of q_j, $\left[\frac{\partial}{\partial q_{j,1}} C_j(q_j) \quad \frac{\partial}{\partial q_{j,2}} C_j(q_j) \quad \cdots \quad \frac{\partial}{\partial q_{j,n}} C_j(q_j)\right]$
$\nabla_{q_{j,i}} C_j(q_{j,1}, x_{j,2}, ...x_{j,n})$	This examines how each component of $q_{j,i}$ changes in the function C_j, $\nabla_{q_{j,i}} C_j(q_{j,1}, q_{j,2}, ...q_{j,n})$ Looks at the specific partial derivative with respect to $q_{j,i}$ $\frac{\partial}{\partial q_{j,i}} C_j(q_{j,1}, q_{j,2}, ..., q_{j,n})$
$x^{(k-1)}, x^{(k)}, x^{(k+1)}$	Shows the iterative steps of x from k-1, to k, then to k + 1

Table 1: Explanation of prevalent notation used in the paper

2 Introduction to Particle-Based Systems

2.1 Particle Systems and Constrained Systems

To keep everything consistent, the aforementioned qualities of constrained particle-systems will be mathematically abstracted in the following table.

p_i	A circular geometric object with properties $x_i, v_i, a_i, m_i, r_i, F_i$
p	The particle system; a concatenated list of all n particle objects $[\ p_1\ p_2\ p_3\ \cdots\ p_n\ ]^T$ (Bender)
C_j	A single constraint function • Defined by a function C_j such that $C_j : \mathbb{R}^n \to \mathbb{R}$ • A function input is called a generalized property, $q_{j,i}$ • The set of all function inputs is called the generalized properties, q_j. There will always be a $q_{j,i}$ that is the stiffness factor, k, such that $k \in q_j$. • The size of q_j is $n_j = n(q_j)$. This essay will focus on time-independent constraints, where $t \notin q_j$ (scleronmic). Additionally, all constraints will be written as an equality, $$C_j(q_{j,1}, q_{j,2}, q_{j,3}, \cdots, q_{j,n}) = 0,$$ and there will not be inequalities that results in constraint breaking (holonomic). This equation must be satisfied for the constraint to be maintained.
C	the concatenated values of all m different constraints $[\ C_1\ C_2\ C_3\ \cdots\ C_m\ ]^T$

Table 2: Mathematical abstraction of the components in a constrained particle system

2.2 Particle System Solvers and Integration Schemes

During the simulation, the constrained particle-system progresses in timesteps, Δt. This is visualized as repeated static frames of the particle system after every Δt amount of time.

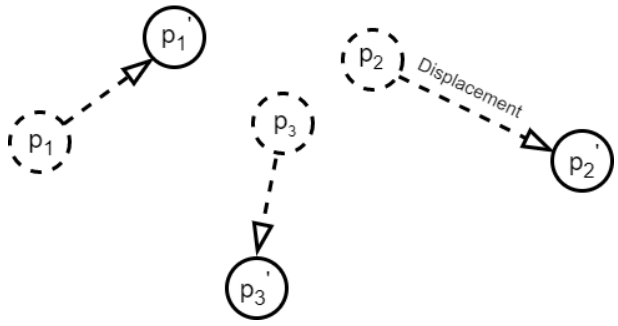


Figure 1: New particle positions after Δt

To find the displacements on a particle that has some velocity and some force, a set of kinematic differential equations are required. These set of equations are fundamental in physics and relate the position, velocity, and acceleration/force.

$$\begin{aligned} v &= \dot{x} \\ a &= \dot{v} \\ F &= ma. \end{aligned} \tag{1}$$

Complex systems are solved by decomposition into a similar form as Equation 1. Numerical integration is then used to arrive at an approximate solution. For this paper, a predictor-corrector semi-implicit Euler integration scheme is used.

2.2.1 Semi-Implicit Euler

Semi-implicit Euler is a numerical integration method that approximates the integral of a function with an algorithmic method. This method will be used to calculate new positions of the particles. The first step of semi-implicit Euler is Euler approximation. It is an algorithm that predicts the next position from a point by extending the slope at that point by a predefined step (Figure 2).

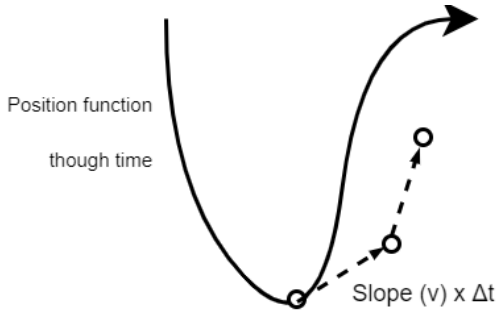


Figure 2: Visualization of Euler algorithm

Compared to normal Euler, semi-implicit Euler first completes the Euler step on velocity then uses that new value to approximate the position. According to Equations 1, the slope of a velocity vs. time curve is the acceleration, a. And the slope of a position vs. time curve is the velocity, v. Therefore,

$$\begin{aligned} v^{(k+1)} &= v^{(k)} + a^{(k)}\Delta t \\ x^{(k+1)} &= x^{(k)} + v^{(k+1)}\Delta t, \end{aligned} \tag{2}$$

and when combined,

$$x^{(k+1)} = x^{(k)} + v^{(k)}\Delta t + a^{(k)}\Delta t^2. \tag{3}$$

Implicit integration schemes solve the system both at the current time and at a future time. Typically, implicit algorithms are much more stable. At a glance, Equations 2 seem explicit as the system is solved for the future position $x^{(k+1)}$ after Δt. This is not entirely true as the velocity of the current system must be solved first, and that solved result is used to determine the new position. For lower values of Δt, error decreases notably and the system is stable like implicit methods. Whereas for larger values of Δt, the system becomes unstable like explicit methods (Liu). Because of this behaviour, this method is described as semi-implicit in literature.

The traditional Euler algorithm has an error value of $O(\Delta t^2)$ (Fitzpatrick), which is the truncation error of the algorithm. Since it only accounts for acceleration, any higher derivatives (jerk, crackle, pop) are truncated as an error. Because semi-implcit Euler is a degree higher, its corresponding error will be $O(\Delta t^3)$ (Liu).

2.2.2 Predictor-Corrector

Predictor-Corrector is an algorithmic enhancement that makes semi-implicit Euler more stable. Predictor-Corrector finds the corrective position displacements based on a predicted future position, then directly calculates the final velocity impulse with the previous position (Figure 3).

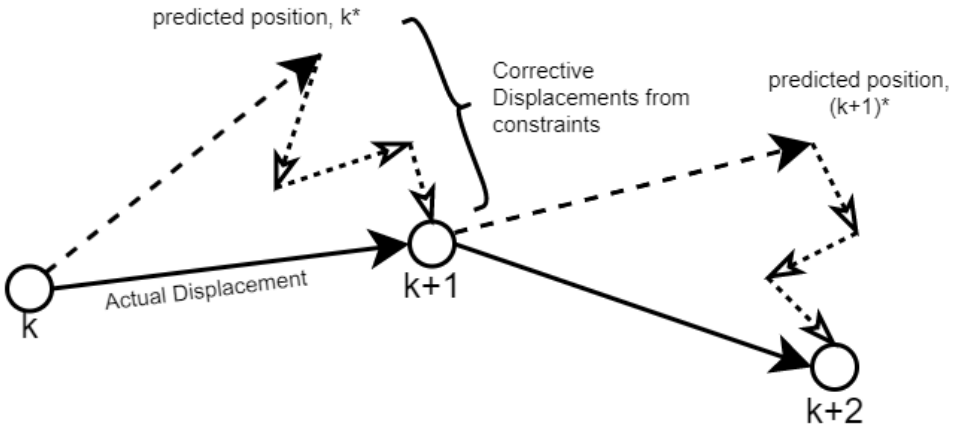


Figure 3: The predictor-corrector model over 3 iterations

To calculate the predicted position $x*$,

$$x^* = x^{(k)} + v^{(k)}\Delta t + a^{(k)}\Delta t^2.$$

Then applying the corrective displacement, $\Delta x^{(k)}$, on $x*$,

$$x^{(k+1)} = x^* + \Delta x^{(k)}.$$

The final velocity of the particle can be found using $v = \frac{\Delta x}{\Delta t}$,

$$v^{(k+1)} = \frac{x^{(k+1)} - x^{(k)}}{\Delta t}. \tag{4}$$

Clavet's paper verifies the efficacy of the Predictor-Corrector model in resolving most over-

shoot problems caused by extreme forces in the semi-implicit Euler algorithm.

2.2.3 Iterative Gauss-Seidel

The previous sections discuss how the new positions of a single particle is calculated. To find the set of all displacements, Δx, that solve the particle system, p, a Gauss-Seidel-like local iterative solver is used. Local iterative solvers converge to the global solution by repeatedly calculating the local Δx_i for each $p_i \in p$. Local iterative solvers are generally easier to implement and faster to compute (Jakobsen).

Gauss-Seidel is a divide-and-conquer algorithm that finds the approximate solution by repeatedly solving simpler counterparts. Since most constraint systems are nonlinear, an iterative "Gauss-Seidelization" (Gutiérrez) of the Gauss-Seidel algorithm is used. For a system of constraints ϕ, this is represented mathematically as,

$$\begin{aligned} x_1^{(k+1)} &= \phi_1(x_1^{(k)}, x_2^{(k)}, ..., x_n^{(k)}) \\ x_2^{(k+1)} &= \phi_2(x_1^{(k+1)}, x_2^{(k)}, ..., x_n^{(k)}) \\ &\vdots \\ x_n^{(k+1)} &= \phi_n(x_1^{(k+1)}, x_2^{(k+1)}, ..., x_n^{(k+1)}). \end{aligned} \tag{5}$$

The system of ϕ is lower triangular, with the bottom below the diagonal being $k+1$ and the rest being k iterations. In practice, this behaves as if the system updates after each constraint is solved; the updated value of the solved constraints are used to solve the next constraints. For a linear system, the rate convergence can be calculated with the spectral radius, $\rho = (cos\frac{\pi}{\epsilon+1})^2$ (Strang). Where ϵ is the number of elements in the vector Δx, and hence, the number of constraints. Generally, for k iterations, the error is multiplied by ρ^k (Strang). Admittedly, this concept is untrue for non-linear systems, but it can still approximate the expected rate of convergence. As such, the convergence rate also decreases for non-linear systems as ϵ increases. However, Gauss-Seidel can also diverge under certain conditions. (Detailed analysis will be available in Sections 3.3.2 and 4.3.2)

2.3 General Solver Algorithm

Combining all of the previous components create a general solver algorithm. Elements of this solver are influenced by Clavet's paper. Although this essay explicitly focuses on the constraint component, other rudimentary features such as collision handling are included for an enhanced user experience. (Code available in Appendix A)

Algorithm 1 General system solver for constrained particle-based systems

```
// Run solver for every frame
loop
    // Iterate iterationNumberPerFrame times per frame
    for k ← 0 to iterationNumberPerFrame do
        for all p_i ∈ p do
            // Update velocity with gravity
            v_i ← v_i + gΔt
            // Update velocity with other external forces
            v_i ← F_i/m_i · Δt
            //Save current position to previous position
            x_i^prev ← x_i
            // Move to predicted position, x*
            x_i ← x_i + v_iΔt
        end for
        // Algorithm 2, Section 3.3
        applyPenaltyConstraints()
        // Algorithm 3, Section 4.3
        applyRelaxedConstraints()
        // Additional functions for improved user experience
        resolveParticleInterations()
        resolveCollisions()
        // Equation 3: calculate the velocity
        for all p_i ∈ p do
            v_i ← (x_i − x_i^prev)/Δt
        end for
    end for
end loop
```

3 Penalty Constraints (PC)

3.1 Explanation and Generalization

Penalty constraints are a type of constraints that can be used in constrained particle systems. In essence, PCs treat all constraints as springs. This means that any deviation from $C(q) = 0$ will cause a penalty force in the opposite direction of the displacement, just like a spring (Figure 4).

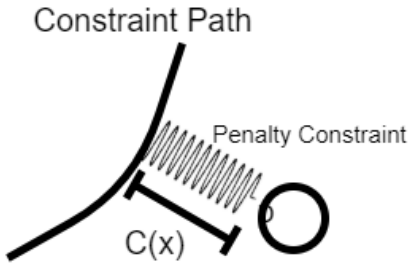


Figure 4: Penalty Constraint Visualization

The spring-like behaviour can be described using Hooke's law,

$$F = -k\Delta s. \tag{6}$$

Where k is the spring constant that determines how stiff a spring is, and Δs is how much the spring is displaced. However, this system has an obvious problem: penalty forces can increase greatly for stiff systems with higher spring constants. The excessive force impulses create large corrective displacements, and when combined with the error of semi-implicit Euler (Section 2.2.1), can cause the system to quickly become unstable.

As an example, Hooke's law will be geometrically applied on a distance constraint, where a pair of constrained particles must be a certain distance apart. Let two particles be constrained by length, r, as shown in Figure 5.

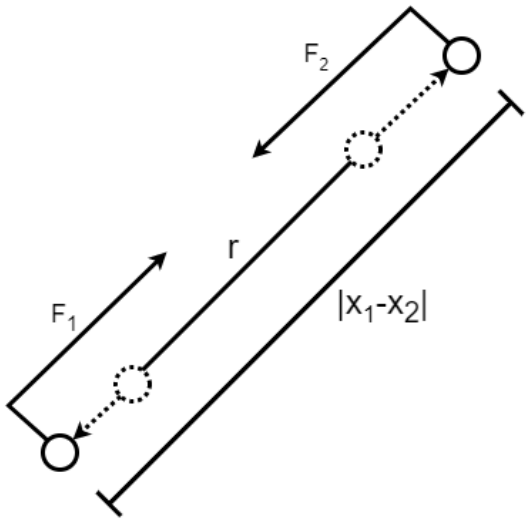


Figure 5: Distance constraint with the PC method

Given $F = -k\Delta s$,

$$\begin{aligned} F_1 &= -k(|x_1 - x_2| - r)\frac{x_1 - x_2}{|x_1 - x_2|} \\ F_2 &= k(|x_1 - x_2| - r)\frac{x_1 - x_2}{|x_1 - x_2|}, \end{aligned} \tag{7}$$

where $\Delta s = |x_1 - x_2| - r$, and $\frac{x_1 - x_2}{|x_1 - x_2|}$ is the direction unit vector. Although the geometric intuition works, it is not sufficient. Next, a generalization of the penalty method will be attempted.

Assuming that the constraints behave as springs, it is possible to calculate the energy that the constraint holds, much like a real spring. The value of this potential energy can then be used to calculate the force applied on each particle, hence resulting in some displacement. Let a constraint be $C_j(x_1, x_2, x_3, ..., x_n, k) = 0$, where x_i is the position of particle p_i. Similarly, the generalized properties will be $q_j = [\ x_1\ x_2\ \cdots\ x_n\ \ k]^T$. Then Hooke's law can be used to calculate the penalty forces for the constraint. Because the equilibrium position is $C_j(q_j) = 0$, the deviation will be $\Delta s = C_j(q_j)$. Therefore, the force will be $F_{C_j} = -kC_j$.

In physics, energy is defined as

$$E = \int_{s_i}^{s_f} F \cdot ds,$$

where $[s_i, s_f]$ is the interval of displacement that the force F is applied. Consequently, the force for a penalty constraint will be applied on the displacement interval $[0, C_j]$. Then the energy function of constraint C_j will be,

$$E_{C_j} = \int_0^{C_j} F_{C_j} \cdot dC_j$$

Using $F_{C_j} = -kC_j$,

$$E_{C_j} = \int_0^{C_j} -kC_j \cdot dC_j$$

$$= -\frac{1}{2}kC_j \cdot C_j \Big|_0^{C_j} = -\frac{1}{2}kC_j \cdot C_j - 0$$

$$E_{C_j} = -\frac{1}{2}kC_j \cdot C_j. \tag{8}$$

Conversely, the force and energy is related by the formula

$$F_x = \frac{\partial E}{\partial x}. \tag{9}$$

Therefore, the force affecting each particle position, x_i, will be

$$F_{x_i} = \frac{\partial E_{C_j}}{\partial x_i}. \tag{10}$$

This can be simplified further by substituting in Equation 8,

$$F_{x_i} = C_j \cdot \frac{\partial C_j}{\partial x_i}. \tag{11}$$

Because of the predictor-corrector model (Section 2.2.2), a corrective displacement from the resulting force must be calculated. To do so, the acceleration is first calculated with Newton's

Second law, $F = ma$. Rearranging yields,

$$a = m_i^{-1} F_{x_i}.$$

The next step involves discretizing the acceleration, a. Discretization uses the semi-implicit Euler algorithm (Section 2.2.1) to convert a into Δx. Recall,

$$x^{(k+1)} = x^{(k)} + v^{(k)}\Delta t + a^{(k)}\Delta t^2.$$

Then,

$$\Delta x^{(k)} = x^{(k+1)} - x^{(k)}$$

$$\Delta x^{(k)} = v^{(k)}\Delta t + a^{(k)}\Delta t^2.$$

In the predictor-corrector model, each corrective displacement caused by a force is independent of the particle velocity. Therefore,

$$\Delta x = a^{(k)}\Delta t^2.$$

Using the result from Newton's Second Law,

$$\Delta x = w_i F_{x_i}\Delta t^2. \tag{12}$$

where w_i is the inverse mass m_i^{-1}. Then substituting with Equation 11 yields,

$$\Delta x_i = -w_i k C_j \cdot \frac{\partial C_j}{\partial x_i}\Delta t^2. \tag{13}$$

To verify the derived formula, it will be applied to the case example stated in Figure 5. The

mathematical representation of the distance constraint will be

$$C_j(x_1, x_2, k) = |x_1 - x_2| - r. \tag{14}$$

This constraint maintains a distance r from two particles with positions x_1 and x_2. Using Equation 11, the penalty forces for each x are

$$\begin{aligned} F_{x_1} &= -k(|x_1 - x_2| - r)\frac{x_1 - x_2}{|x_1 - x_2|} \\ F_{x_2} &= k(|x_1 - x_2| - r)\frac{x_1 - x_2}{|x_1 - x_2|}. \end{aligned} \tag{15}$$

These results match the intuitive geometric results calculated earlier. This can be taken a step further with the discretization in Equation 13, of which

$$\begin{aligned} \Delta x_1 &= -w_1 k(|x_1 - x_2| - r)\frac{x_1 - x_2}{|x_1 - x_2|}\Delta t^2 \\ \Delta x_2 &= w_2 k(|x_1 - x_2| - r)\frac{x_1 - x_2}{|x_1 - x_2|}\Delta t^2. \end{aligned} \tag{16}$$

(Detailed derivation available in Appendix B)

3.2 Damping

Another major part of PCs is damping. Much like in real life, springs are affected by internal frictional forces that causes them to lose energy with time. This artificially induced energy loss can help prevent the system from becoming unstable. Typically, a damping force, D_j, is proportional to the velocity (Adams). Since this is the damping for a constraint, the respective velocity will be of the constraint itself (the rate of which the constraint contracts/extends). This is denoted mathematically as $v_{C_j} = \dot{C}_j$. Hence, D_j can be rewritten with an arbitrary damping constant, μ, as

$$D_j = \mu \dot{C}_j. \tag{17}$$

Next, to calculate how the constraint will behave with damping, the penalty forces with damping must be derived. Similar to Section 3.1, a potential energy function will be required. Let the function $F_{C_j} = -kC_j - D_j$, where $-kC_j$ is the force of the constraint and $-D_j$ is the damping force opposing the constraint (hence the negative), and by calculating the corresponding energy function,

$$E_{C_j} = \int_0^{C_j} F_{C_j} \cdot dC_j = \int_0^{C_j} (-kC_j - \mu \dot{C}_j) \cdot dC_j.$$

Expanding the integral gives

$$= \int_0^{C_j} -kC_j \cdot dC_j - \int_0^{C_j} \mu \dot{C}_j \cdot dC_j$$

$$= \int_0^{C_j} -kC_j \cdot dC_j - \int_0^{C_j} \mu \frac{dC_j}{dt} \cdot dC_j,$$

and by evaluating,

$$= -\frac{1}{2} kC_j \cdot C_j \Big|_0^{C_j} - \mu \frac{dC_j}{dt} \cdot C_j \Big|_0^{C_j}$$

$$E_{C_j} = -\frac{1}{2} kC_j \cdot C_j - \mu \frac{dC_j}{dt} \cdot C_j. \tag{18}$$

Now the individual forces can be found from Equation 18, the energy function. Recall Equation 11,

$$F_{x_i} = \frac{\partial E}{\partial x_i}.$$

Therefore,

$$F_{x_i} = \frac{\partial}{\partial x_i} \left(-\frac{1}{2} kC_j \cdot C_j - \mu \frac{dC_j}{dt} \cdot C_j \right)$$

$$= \frac{\partial}{\partial x_i} \left(-\frac{1}{2} kC_j \cdot C_j \right) - \frac{\partial}{\partial x_i} \left(\mu \frac{dC_j}{dt} \cdot C_j \right). \tag{19}$$

Next, the end term $\frac{\partial}{\partial x_i}(\mu\frac{dC_j}{dt}\cdot C_j)$ can be broken down with the product rule to

$$\mu\left(\frac{\partial}{\partial x_i}\frac{dC_j}{dt}\cdot C_j+\frac{dC_j}{dt}\cdot\frac{\partial C_j}{\partial x_i}\right).$$

Performing a swap and factor then yields

$$\mu\left(\frac{d}{dt}\frac{\partial C_j}{\partial x_i}\cdot C_j+\frac{dC_j}{dt}\cdot\frac{\partial C_j}{\partial x_i}\right)=\mu\frac{\partial C_j}{\partial x_i}\left(\frac{d}{dt}\cdot C_j+\frac{dC_j}{dt}\right).$$

Finally simplifying to

$$2\mu\frac{\partial C_j}{\partial x_i}\cdot\frac{dC_j}{dt}. \tag{20}$$

Now, substituting back and continuing with Equation 19 gives

$$\begin{aligned}F_{x_i}&=-kC_j\cdot\frac{\partial C_j}{\partial x_i}-2\mu\frac{\partial C_j}{\partial x_i}\cdot\frac{dC_j}{dt}\\ F_{x_i}&=\left(-kC_j-2\mu\frac{dC_j}{dt}\right)\cdot\frac{\partial C_j}{\partial x_i},\end{aligned} \tag{21}$$

and because μ is an arbitrary damping constant, 2μ can be condensed into μ. Further simplification yields similar results as Witkin, where

$$F_{x_i}=(-kC_j-\mu\dot{C}_j)\cdot\frac{\partial C_j}{\partial x_i}. \tag{22}$$

Lastly, using the previous method, the force from Equation 22 is discretized to calculate the position. Given $a_i=w_iF_{x_i}$ and $\Delta x_i=a_i\Delta t^2$, then

$$\Delta x_i=w_i(-kC_j-\mu\dot{C}_j)\cdot\frac{\partial C_j}{\partial x_i}\Delta t^2. \tag{23}$$

Equation 23 is the final equation for penalty constraints, which take into account damping. When the damping constant is 0, Equation 23 decomposes into Equation 13. Once again, applying the results on the constraint function $C_j(x_1,x_2)=|x_1-x_2|-r$ yields the new set

of equations,

$$\begin{aligned}\Delta x_1 &= w_1 \left(-k(|x_1 - x_2| - r) - \mu \frac{(v_1 - v_2) \cdot (x_1 - x_2)}{|x_1 - x_2|} \right) \cdot \frac{x_1 - x_2}{|x_1 - x_2|} \Delta t^2 \\ \Delta x_2 &= -w_2 \left(-k(|x_1 - x_2| - r) - \mu \frac{(v_1 - v_2) \cdot (x_1 - x_2)}{|x_1 - x_2|} \right) \cdot \frac{x_1 - x_2}{|x_1 - x_2|} \Delta t^2.\end{aligned} \tag{24}$$

(Detailed derivation available in Appendix B)

3.3 Programmatic Implementation and Analysis

For later analysis and mathematical verification, Equations 24 are implemented programmatically. (Code and demonstration available in Appendix A)

Algorithm 2 Penalty Constraint Algorithm - applyPenaltyConstraints()

```
1: //loop through all constraint solving iterations
2: for k ← 0 to constraintSolvingIterationNumber do
3:    //Iterative Gauss-Seidel step for each constraint
4:    for all C_j ∈ C do
5:       // Calculate Penalty Forces
6:       force ← calculateForce()
7:       //Discretize force to find displacement
8:       Δx ← force · Δt
9:       // Apply displacement
10:      x ← x + Δx
11:   end for
12: end for
```

3.3.1 Inconsistent Stiffness

Inconsistent stiffness is the first problem with the iterative Gauss-Seidel algorithm (Section 2.2.3). As described in Algorithm 1, the $iterationNumberPerFrame$ can be increased to solve the system multiple times every frame. Increasing the $iterationNumberPerFrame$ can speed up system convergence as the error is multiplied by p^k, where k is the iteration number (Section 2.2.3). Unfortunately, speeding up system convergence also alters constraint stiffness (Macklin). This dependence results in arbitrary stiffness units, resulting

in an unsuitable method for realistic physical modelling. To negate this issue, the iterative solver can be replaced with a global solver that directly computes the solution, which ignores any convergence errors (Witkin).

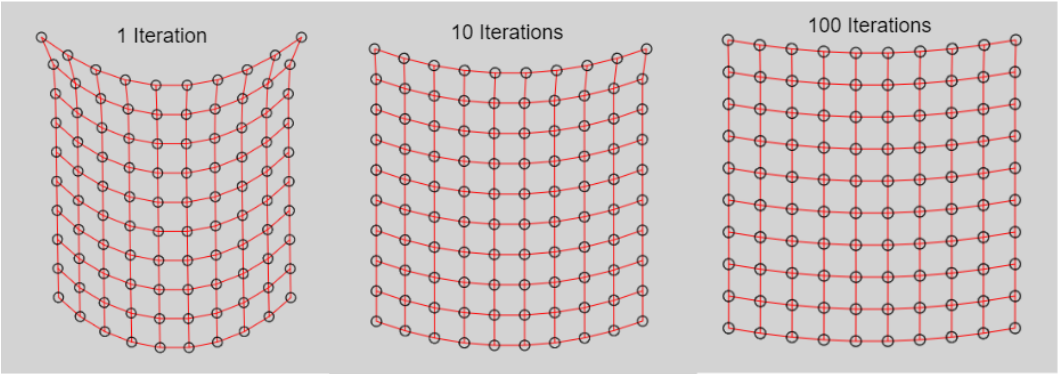


Figure 6: PC stiffness behaviour for different iterations (the uniform red colour is due to the high forces on the constraints with a stiffness of 100000)

It is observed in Figure 6 that more iterations result in higher stiffness. This is because the convergence error from the Gauss-Seidel algorithm is reduced with more iterations. More iterations also cause constraint penalty force displacements to multiply, behaving as if it was a larger displacement caused by larger stiffness (Figure 7).

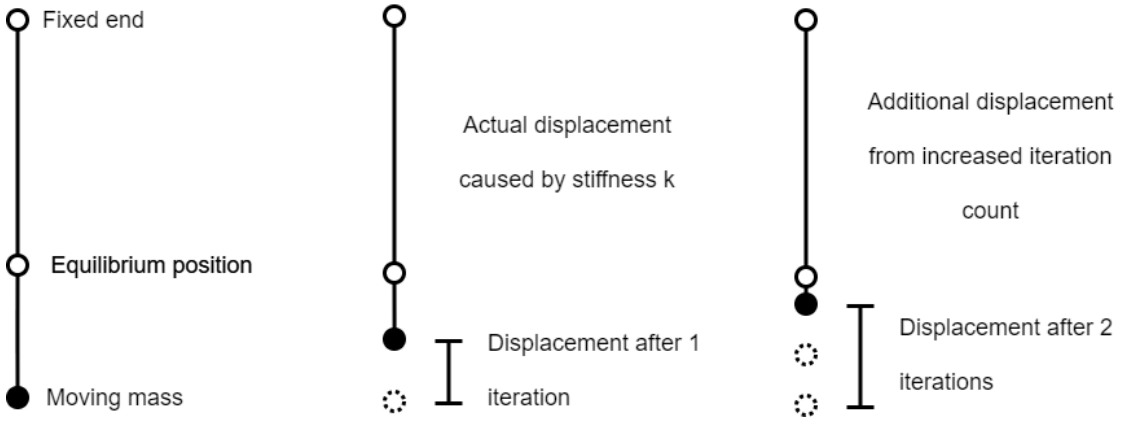


Figure 7: Increased PC stiffness with iteration number

However, using $iterationNumberPerFrame$ as a means to increase stiffness can be more stable than simply increasing k. Smaller iterative steps can help combat overshoot for large

stiffness values, as demonstrated in Figure 8.

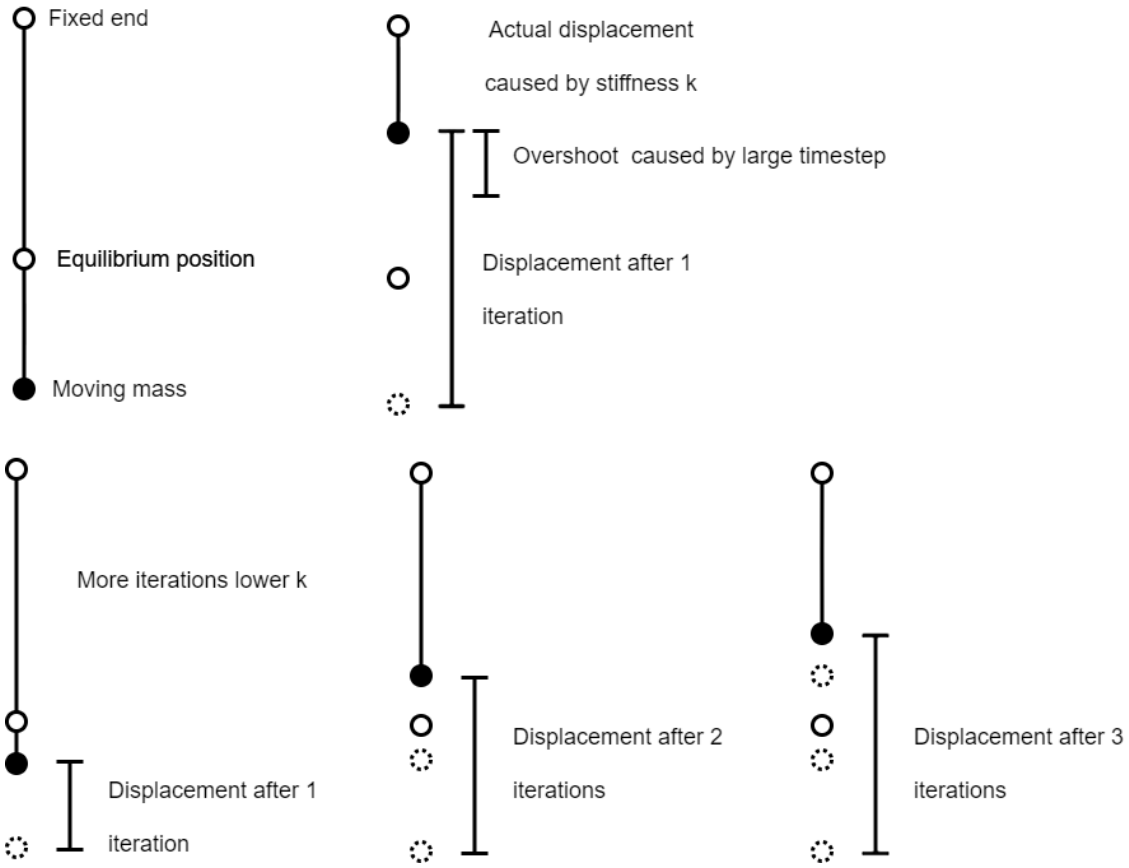


Figure 8: Induced PC stiffness stability for increased iteration number

For effective rigid systems, it is important to find an empirical balance of k, Δt, and $iterationNumberPerFrame$.

3.3.2 Convergence and Energy Conservation

Convergence and energy conservation are also key concepts of physical modelling. Inconsistencies in the energy conservation could result in unrealistic physical models. Ideal PCs do not converge; instead, they oscillate like a spring between different stable states about $C(q) = 0$ (Goldstein). Just like how a pendulum will stay forever swinging without energy loss, convergence to a static equilibrium is only induced when damping causes a energy loss within the system.

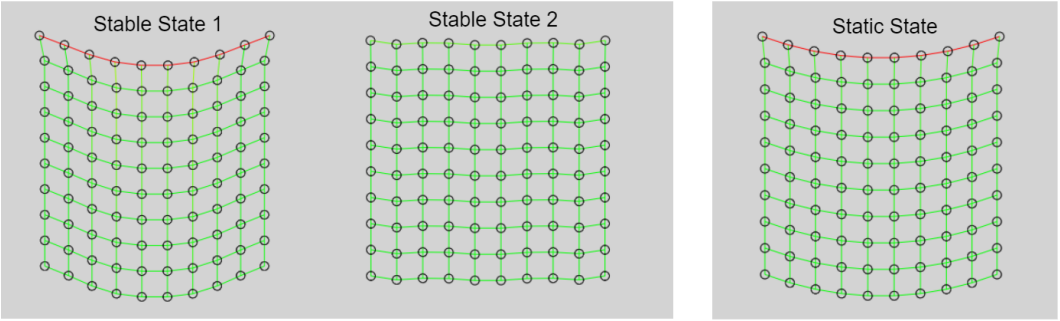


Figure 9: Stable oscillatory behaviour vs convergent behaviour

Overall system convergence is dependent on the Gauss-Seidel solver. Generally, the larger the system, the longer it takes to converge. As discussed in Section 2.2.3, increasing the $iterationNumberPerFrame$ increases the system convergence rate. However, ideal systems with no damping still seem to lose energy with time, with the rate being influenced by Δt. Increasing Δt increases the error from the semi-implicit Euler algorithm by $O(\Delta t^3)$ (Section 2.2.1). The truncated error behaves as an energy deficit and leads to faster energy loss. Similarly, if the system is large, individual constraint errors will accumulate, leading to greater energy dissipation. Larger penalty forces can also induce greater errors during the force discretization, $\Delta x_i = w_i F_i \Delta t^2$ (of which Δt also plays a role). This issue is verified with a simple pendulum simulation. The graphs in Figure 10 represent the y-position of the swinging mass through time. (Raw data available in Appendix A)

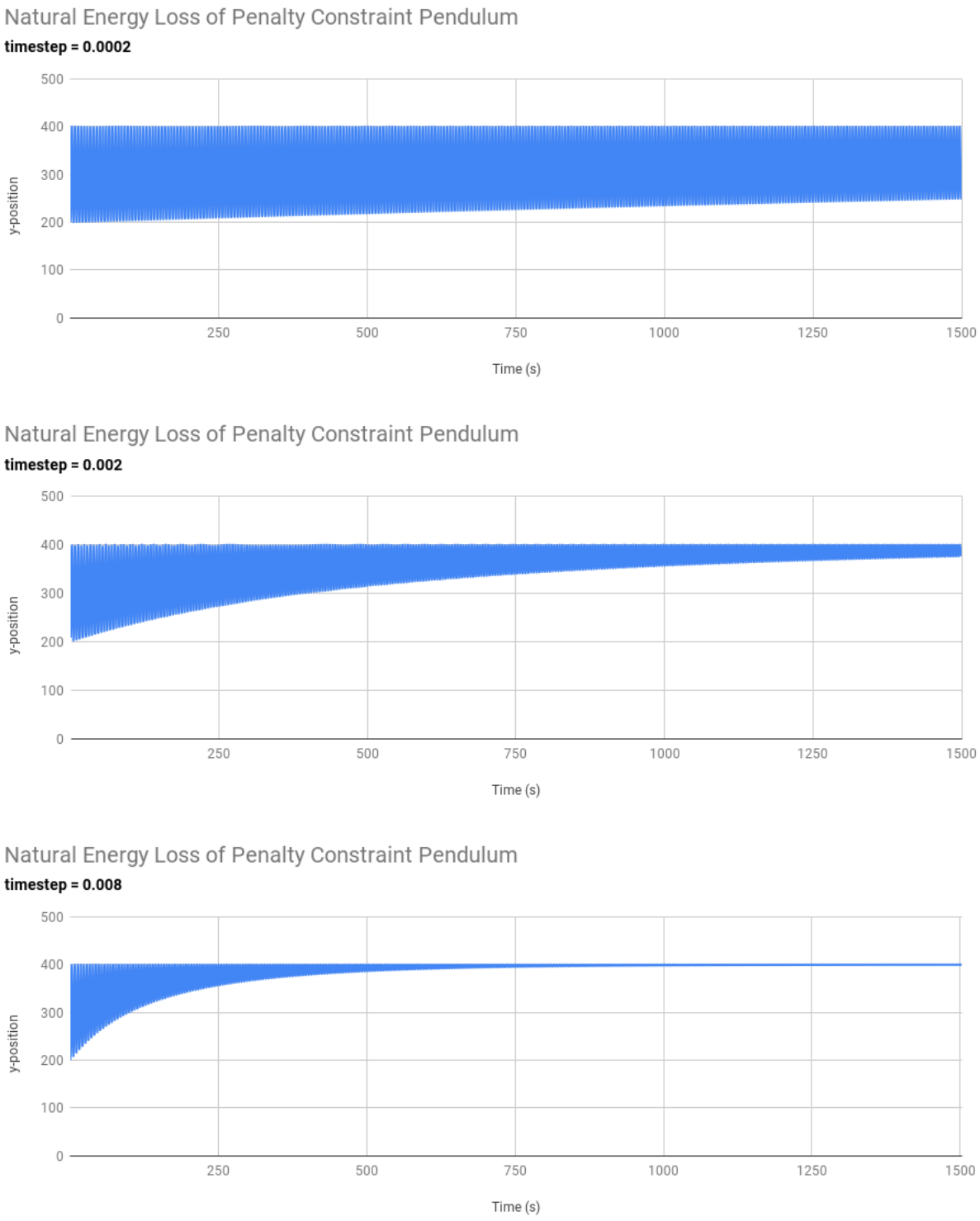


Figure 10: y-position vs time graphs of PC pendulum systems for different Δt

In contrast, systems can also be unstable and divergent, where the system gains energy and explodes out of control.

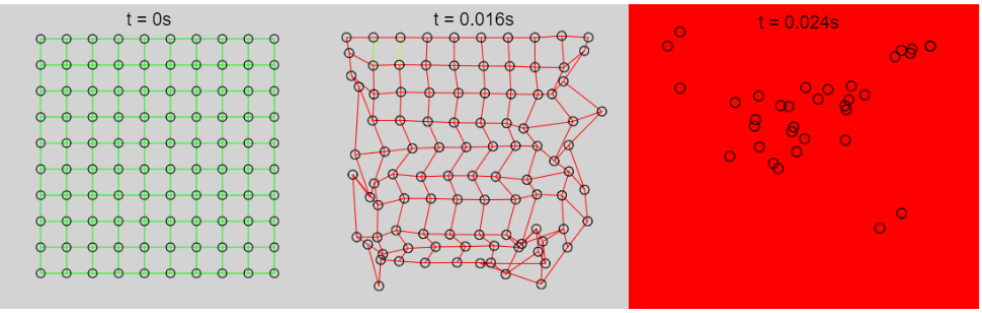


Figure 11: PC divergence at $\Delta t = 0.008$

Divergent behaviour is caused by overestimation from the semi-implicit Euler algorithm. Divergence can be resisted with the inclusion of damping, where damping provides a buffer for overshoot. Moreover, divergence can be minimized by increasing the iteration count while lowering the forces (ex. gravity, k), or decreasing Δt.

4 Relaxed Geometric Constraints (RGC)

4.1 Explanation and Generalization

RGCs approximate the true constraint path by solving the constraint geometrically. Visually, it skips all the energy derivations and it appears that the "tense" constraint is moved (relaxed) to its ideal form. The "relaxation" of this method refers to the projection of the current positions to the next closest legal positions such that the constraint function is satisfied. Once again, a straightforward geometric derivation of a simple case is presented below in Figure 12.

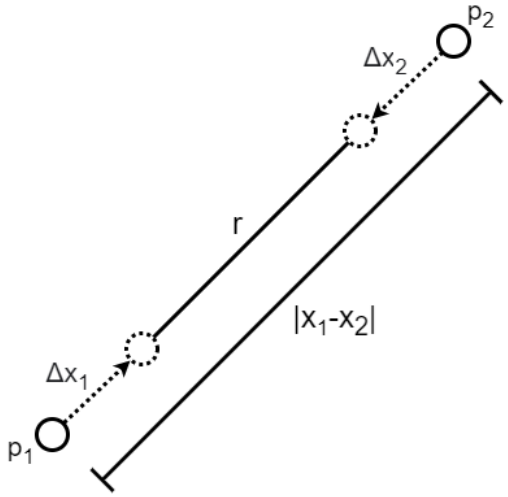


Figure 12: RGC Stretch Constraint

Where r is the ideal length and $|x_1 - x_2|$ is the current length. If p_1 and p_2 have the same mass, both will move the same amount to reach the ideal position. Therefore,

$$|\Delta x_1| = |\Delta x_2| = \frac{1}{2}(|x_1 - x_2| - r).$$

Intuitively, the contraction will be collinear with $x_1 - x_2$. Hence, the final contraction must be scaled with the directional unit vector,

$$\begin{aligned} \Delta x_1 &= -\frac{1}{2}(|x_1 - x_2| - r)\frac{x_1 - x_2}{|x_1 - x_2|} \\ \Delta x_2 &= \frac{1}{2}(|x_1 - x_2| - r)\frac{x_1 - x_2}{|x_1 - x_2|}. \end{aligned} \tag{25}$$

Equations 23 will be verified later with rigorous mathematical derivation. For now, it provides a general idea of RGCs.

As previously mentioned, RGCs are purely position-based (ignores force and energy), which offers more control. But as a result, extra caution is needed to conserve realistic physical meaning. For an accurate physical model, the system's linear momentum and

angular momentum must be conserved. In physics, this is represented by

$$\sum_{i}^{n} m_i \Delta x_i \text{ and } \sum_{i}^{n} r_i \times \Delta x_i. \tag{26}$$

Where r_i is the distance to some common rotation center of the whole particle system and x_i is the position value of a particle, p_i. They will be useful for providing the necessary information to calculate the final relaxation. The RGC process begins by setting up the system to be solved for a single constraint C_j, which is

$$\begin{aligned} q'_{j,1} &= q_{j,1} + \Delta q_{j,1} \\ q'_{j,2} &= q_{j,2} + \Delta q_{j,2} \\ &\vdots \\ q'_{j,n} &= q_{j,n} + \Delta q_{j,n}, \end{aligned}$$

such that each Δq_j moves q_j into a legal position for C_j,

$$C_j(q_j) = 0 \text{ and } C_j(q_j + \Delta q_j) = 0. \tag{27}$$

The goal for a constraint C_j is to calculate Δq_j to find $q_j + \Delta q_j$. The set of q_j can be specified further because some values $q_{j,i} \in q_j$, do not change. Namely, the mass and radius of each particle will remain constant, hence they will not need to be solved for. For this essay, the only changing values of particle p_i will only be the position, x_i (v, a, F are disregarded because RGCs are position based). Therefore, the list q_j can be further simplified into $q_j = [x_a \;\; x_b \;\; x_c \;\; ... \;]^T$. Where x is the list of all positions $[x_1 \; x_2 \; ... \; x_n]$, and $x_a, x_b, x_c ... \in x$. Solving for Δq_j is difficult because C_j is a nonlinear function, so C_j is first approximated using local linearization (Bender). This method is an extension of Newton's method of finding roots. Newton's method states for a initial point x_0, displacement h, and

function $f(x)$,

$$f(x_0 + h) = f(x_0) + hf'(x_0) + O(h^2).$$

The vector-valued function equivalent of $f'(x_0)$ is the gradient $\nabla_{x_0} C(x_0)$, which essentially describes the slope of $C(x_0)$ at the point x_0. And with $f(x) = C_j(q_j)$ and $h = \Delta q_j$,

$$C_j(q_j + \Delta q_j) = C_j(q_j) + \nabla_{(q_j)} C(q_j) \cdot \Delta x + O(\Delta q_j^2) = 0. \tag{28}$$

The error term $O(\Delta q_j^2)$ is caused by the truncation of higher degree derivatives, much like Euler's algorithm in Section 2.2.1. The erroneous displacements resulting from this error leads to energy loss and inaccurate results. To simplify the equation, the error term can be ignored, resulting in an approximation of

$$C_j(q_j) + \nabla_{q_j} C_j(q_j)^T \cdot \Delta q_j \approx 0. \tag{29}$$

Equation 29 needs to be solved to find Δq_j, but it is currently underdetermined. This indicates there must be another relationship of Δq_j. The previously discussed conservation of linear momentum (Equation 26) can be used to limit the possible values of Δq_j. Using the conservation of linear momentum, Δq_j must be in the same direction as $\nabla_{q_j} C_j(q_j)$ or else extra directions will be introduced. Therefore, Δq_j will be a scalar multiple of the vector $\nabla_{q_j} C_j(q_j)$. Now, Δq_j is rewritten as

$$\Delta q_j = \lambda \nabla_{q_j} C_j(q_j)^T. \tag{30}$$

Where λ is the extension factor (relaxation factor). However, the above relationship for Δq_j is only true for particles of same mass. When different masses are introduced, the scaling will be different as massive particles are harder to move compared to lighter particles. The next step will calculate the mass-scaling relationship from the laws of conservation of momentum.

Recall,

$$\sum_{i}^{n} m_i \Delta x_i = 0.$$

Systems with constant mass, m, behave such that,

$$\sum_{i}^{n} m \Delta x_i = 0.$$

Dividing both sides by m,

$$\sum_{i}^{n} \Delta x_i = 0.$$

To achieve the same effect for a system with different masses, each Δx_i is multiplied with its reciprocal mass $1/m_i$ (denoted as w_i) of that same particle p_i,

$$\sum_{i}^{n} m_i \frac{1}{m_i} \cdot \Delta x_i = 0 \rightarrow \sum_{i}^{n} \Delta x_i = 0. \tag{31}$$

Similarly, for angular momentum,

$$\sum_{i}^{n} r_i \times (m_i \frac{1}{m_i} \cdot \Delta x_i) = 0 \rightarrow \sum_{i}^{n} r_i \times \Delta x_i = 0. \tag{32}$$

And because $q_j \in x$,

$$\sum_{i}^{n} \Delta q_{j,i} = 0 \text{ and } \sum_{i}^{n} r_i \times \Delta q_{j,i} = 0$$

Hence Equation 32 can be modified to include the mass scaling:

$$q_j = [w_1 q_{j,1} \;\; w_2 q_{j,2} \;\; ... \;\; w_n q_{j,n}]^T$$

$$= \begin{bmatrix} w_{j,1} & 0 & 0 & 0 \\ 0 & w_{j,2} & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & w_{j,4} \end{bmatrix} \begin{bmatrix} q_{j,1} \\ q_{j,2} \\ \vdots \\ q_{j,3} \end{bmatrix}$$

$$\therefore \Delta q_j = \lambda W_j \nabla_{q_j} C_j(q_j)^T. \tag{33}$$

Where W_j is the corresponding inverse diagonal mass matrix M_j^{-1} of the set q_j. Using this equation, λ can be derived.

The goal is to find λ such that $C_j(q_j) + \nabla_{q_j} C_j(q_j)^T \cdot \Delta q_j \approx 0$ (Equation 29). Substituting with Equation 33,

$$C_j(q_j) + \nabla_{q_j} C_j(q_j)^T \cdot \lambda W_j \nabla_{q_j} C_j(q_j)^T \approx 0. \tag{34}$$

Since λ is a scalar value, it can be factored out and isolated, resulting in

$$\lambda = -\frac{C_j(q_j)}{\nabla_{q_j} C_j(q_j)^T \cdot W_j \nabla_{q_j} C_j(q_j)^T}. \tag{35}$$

Substituting back into Equation 30,

$$\Delta q_j = -W_j \frac{C_j(q_j)}{\nabla_{q_j} C_j(q_j)^T \cdot W_j \nabla_{q_j} C_j(q_j)} \nabla_{q_j} C_j(q_j)^T. \tag{36}$$

The resulting Δq_j is the vector that returns the change of all $q_{j,i}$. To get a more useful result, the function will be broken to each component of $\Delta q_{j,i}$. Firstly, $\nabla_{q_j} C_j(q_j)^T \cdot W_j \nabla_{q_j} C_j(q_j)^T$ can be simplified further.

$$\nabla_{q_j} C_j(q_j)^T \cdot W_j \nabla_{q_j} C_j(q_j)^T = \nabla_{q_j} C_j(q_j) W_j \nabla_{q_j} C_j(q_j)^T$$

$$= \begin{bmatrix} \frac{\partial}{\partial q_{j,1}} C_j(q_j) & \frac{\partial}{\partial q_{j,2}} C_j(q_j) & \dots & \frac{\partial}{\partial q_{j,n}} C_j(q_j) \end{bmatrix} \begin{bmatrix} w_{j,1} & 0 & 0 & 0 \\ 0 & w_{j,2} & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & w_{j,4} \end{bmatrix} \begin{bmatrix} \frac{\partial}{\partial q_{j,1}} C_j(q_j) \\ \frac{\partial}{\partial q_{j,2}} C_j(q_j) \\ \vdots \\ \frac{\partial}{\partial q_{j,n}} C_j(q_j) \end{bmatrix}$$

$$= \begin{bmatrix} \frac{\partial}{\partial q_{j,1}} C_j(q_j) w_{j,1} & \frac{\partial}{\partial q_{j,2}} C_j(q_j) w_{j,2} & \dots & \frac{\partial}{\partial q_{j,n}} C_j(q_j) w_{j,n} \end{bmatrix} \begin{bmatrix} \frac{\partial}{\partial q_{j,1}} C_j(q_j) \\ \frac{\partial}{\partial q_{j,2}} C_j(q_j) \\ \vdots \\ \frac{\partial}{\partial q_{j,n}} C_j(q_j) \end{bmatrix}$$

$$= w_{j,1} \left(\frac{\partial}{\partial q_{j,1}} C_j(q_j) \right)^2 + w_{j,2} \left(\frac{\partial}{\partial q_{j,2}} C_j(q_j) \right)^2 + \cdots + w_{j,n} \left(\frac{\partial}{\partial q_{j,n}} C_j(q_j) \right)^2$$

$$= \sum_u^n w_{j,u} [\nabla_{q_{j,u}} C_j(q_j) \cdot \nabla_{q_{j,u}} C_j(q_j)]. \tag{37}$$

To clarify, the notation $\nabla_{q_{j,i}} C_j(q_j)$ is the term of $\nabla_{q_{j,i}} C_j(q_j)$ corresponding to $q_{j,i}$. Substituting back to Equation 36 and expanding yields,

$$\Delta q_j = - \begin{bmatrix} w_{j,1} & 0 & 0 & 0 \\ 0 & w_{j,2} & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & w_{j,4} \end{bmatrix} \frac{C_j(q_j)}{\sum_u^n w_{j,u} [\nabla_{q_{j,u}} C_j(q_j) \cdot \nabla_{q_{j,u}} C_j(q_j)]} \begin{bmatrix} \frac{\partial}{\partial q_{j,1}} C_j(q_j) \\ \frac{\partial}{\partial q_{j,2}} C_j(q_j) \\ \vdots \\ \frac{\partial}{\partial q_{j,n}} C_j(q_j) \end{bmatrix}$$

$$= - \begin{bmatrix} w_{j,1} \frac{C_j(q_j)}{\sum_u^n w_{j,u} [\nabla_{q_{j,u}} C_j(q_j) \cdot \nabla_{q_{j,u}} C_j(q_j)]} \frac{\partial}{\partial q_{j,1}} C_j(q_j) \\ w_{j,2} \frac{C_j(q_j)}{\sum_u^n w_{j,u} [\nabla_{q_{j,u}} C_j(q_j) \cdot \nabla_{q_{j,u}} C_j(q_j)]} \frac{\partial}{\partial q_{j,2}} C(x) \\ \vdots \\ w_{j,n} \frac{C_j(q_j)}{\sum_u^n w_{j,u} [\nabla_{q_{j,u}} C_j(q_j) \cdot \nabla_{q_{j,u}} C_j(q_j)]} \frac{\partial}{\partial q_{j,n}} C_j(q_j) \end{bmatrix}.$$

Hence for each $q_{j,i}$,

$$\Delta q_{j,i} = -w_{j,i} \frac{C_j(q_j)}{\sum_u^n w_{j,u} [\nabla_{q_{j,k}} C_j(q_j) \cdot \nabla_{q_{j,k}} C_j(q_j)]} \nabla_{q_{j,i}} C_j(q_j). \tag{38}$$

Equation 38 is similar to the equation from Bender. It is important to note that Equation 38 does not contain a stiffness constant and only models perfectly rigid models ($k \to \infty$). Stiffness can introduced by multiplying $\Delta q_{j,i}$ with a reduction factor s_j (s_j replaces k as the new stiffness constant). Smaller values of s_j will result in a more flexible system; conversely, higher values of s result in stiffer systems. Hence, the final equation will be,

$$\Delta q_{j,i} = -w_{j,i} s_j \frac{C_j(q_j)}{\sum_u^n w_{j,u}[\nabla_{q_{j,k}} C_j(q_j) \cdot \nabla_{q_{j,k}} C_j(q_j)]} \nabla_{q_{j,i}} C_j(q_j). \tag{39}$$

To verify this result, it will be applied on $C(x_1, x_2) = |x_1 - x_2| - r$ (Figure 12), which results in

$$\begin{aligned} \Delta x_1 &= -s \frac{w_1}{w_1 + w_2}(|x_1 - x_2| - r)\frac{x_1 - x_2}{|x_1 - x_2|} \\ \Delta x_2 &= s \frac{w_2}{w_1 + w_2}(|x_1 - x_2| - r)\frac{x_1 - x_2}{|x_1 - x_2|}. \end{aligned} \tag{40}$$

This result looks similar to the one derived using PCs, with PCs having extra Δt terms, and RGCs having extra mass-scaling terms. The derived results for RGCs will also be programmatically implemented.

4.2 Damping

Much like PCs, damping on RGCs can also resolve divergence issues while adding a layer of realism onto the physics simulation itself. Sadly, constraint damping of rigid position-based constraints is impossible (Nealen). Damping depends on the movement of $C_j(q_j)$ about 0, and since perfectly rigid constraints always satisfy $C_j(q_j) = 0$, it cannot be affected by constraint damping. The arbitrary stiffness value s is also impossible to quantify, causing the exact damping forces to be unknown. Macklin's XPBD paper resolved this issue by using an energy-based approach to derive a compliant position-based constraint method. Their technique is similar to a combination of penalty and relaxed geometric constraints. The only method to introduce damping to the current RGC method is by applying a resistive force

onto each particle individually.

4.3 Programmatic Implementation and Analysis

Since RGCs can directly find corrective displacements, it is easy to implement and suitable for the predictor-corrector scheme. (Code and demonstration available in Appendix A)

Algorithm 3 Relaxed Geometric Constraint Algorithm - applyRelaxedConstraints()

```
1: //loop through all constraint solving iterations
2: for k ← 0 to constraintSolvingIterationNumber do
3:    //Iterative Gauss-Seidel step for each constraint
4:    for all C_j ∈ C do
5:       // Directly calculate displacement
6:       Δq_j ← calculateDisplacement()
7:       // Apply displacement
8:       q_j ← q_j + Δq_j
9:    end for
10: end for
```

4.3.1 Inconsistent Stiffness

The stiffness parameter, s, of Equation 38 is completely arbitrary. From experimentation, constraints can behave springy with $0 < s < 0.5$, and stiff with $1 < s < 1.1$. When $s > 1$, the system over-corrects and becomes more rigid, whereas for $s < 1$, the system under-corrects and behaves more compliantly. It is important to note that the system has a high chance of diverging for $s > 1.3$. Despite the theoretical rigidity for $s = 1$, the supposedly rigid constraint still exhibits some compliance (Figure 13).

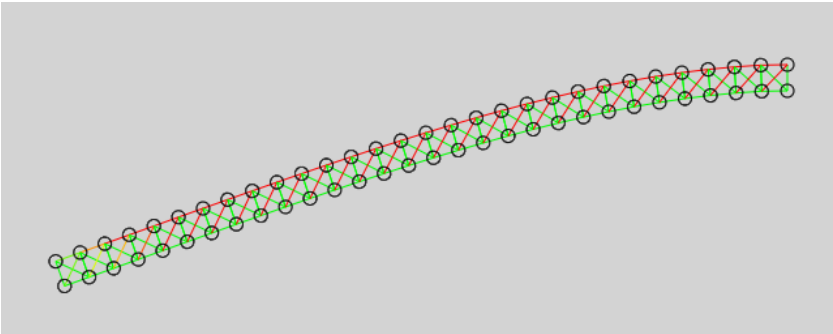


Figure 13: Compliance of a theoretically rigid beam with $s = 1$

It was observed that the *constraintSolvingIterationNumber* (Algorithm 3) significantly affects the system's stiffness. The reason for this behaviour is due to the repeated decrease of the truncation error $O(\Delta q_j^2)$ (Equation 28) as Δq_j is lowered every iteration.

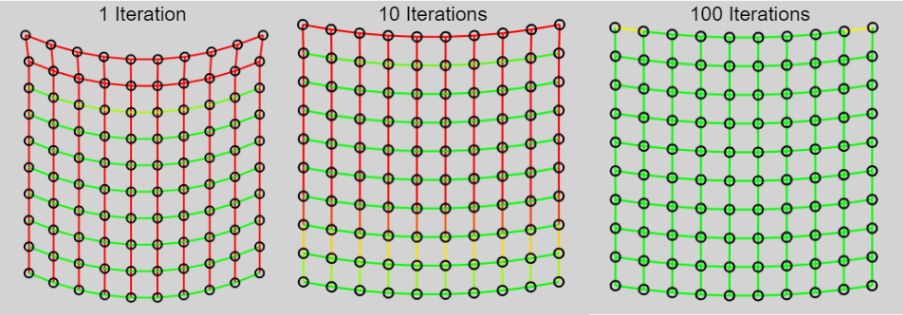


Figure 14: The stiffness increases in a cloth simulation as constraintSolvingIterationNumber increases

To resolve the lower stiffness for smaller values of *constraintSolvingIterationNumber*, the stiffness factor, s, can be increased as compensation. However, the exact scaling will need to be experimentally determined. Moreover, RGCs also exhibit different stiffness responses with different Δt (Figure 15).

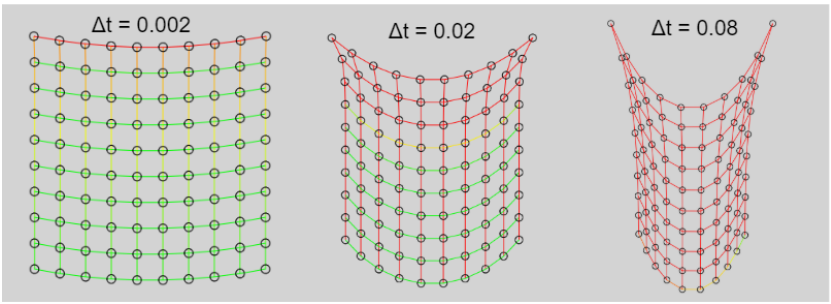


Figure 15: The stiffness decrease in a cloth simulation as the timestep increases

The system is less stiff as the Δt increases. The reason of this effect is due to the external forces acting on each RGC (Forces are not internal as Δt is not in Equation 38). In the specific example shown in Figure 15, the particles are acted on by a gravity force. To apply the gravity force, it must be discretized by Δt (Equation 3). Larger values of Δt impart a larger displacement. Larger external displacements cause larger Δq_j compensation which

increases the truncation error of the constraint (Equation 28), hence lowering the perceived stiffness.

The inconsistent stiffness values undermine RGC's credibility as a rigorous physics simulator, but as long as the initial parameters remain consistent, it can still simulate visually realistic effects (Macklin).

4.3.2 Constraint Convergence and Energy Conservation

The conditions for the convergence and stability of RGCs coincide with the conditions for PCs. Comparatively, RGCs are more stable because they are a position-based method. But also as a result, RGCs neglect energy conservation. Moreover, the constraint error - which is effected by Δq_j (Equation 28) - results in inconsistent system energies.

It is observed that the energy loss phenomenon is more apparent when the $constraintSolvingIterationNumber$, csn, is higher. Increasing csn is equivalent to accelerating the time to $csn \cdot \Delta t$. Therefore, the original error from 1 iteration will be stacked csn times, resulting in greater energy loss. A larger Δt also increases Δq_j and results in more energy instability. This phenomenon is supported with a graphical analysis of an oscillating RGC pendulum system (Raw data available in Appendix A).

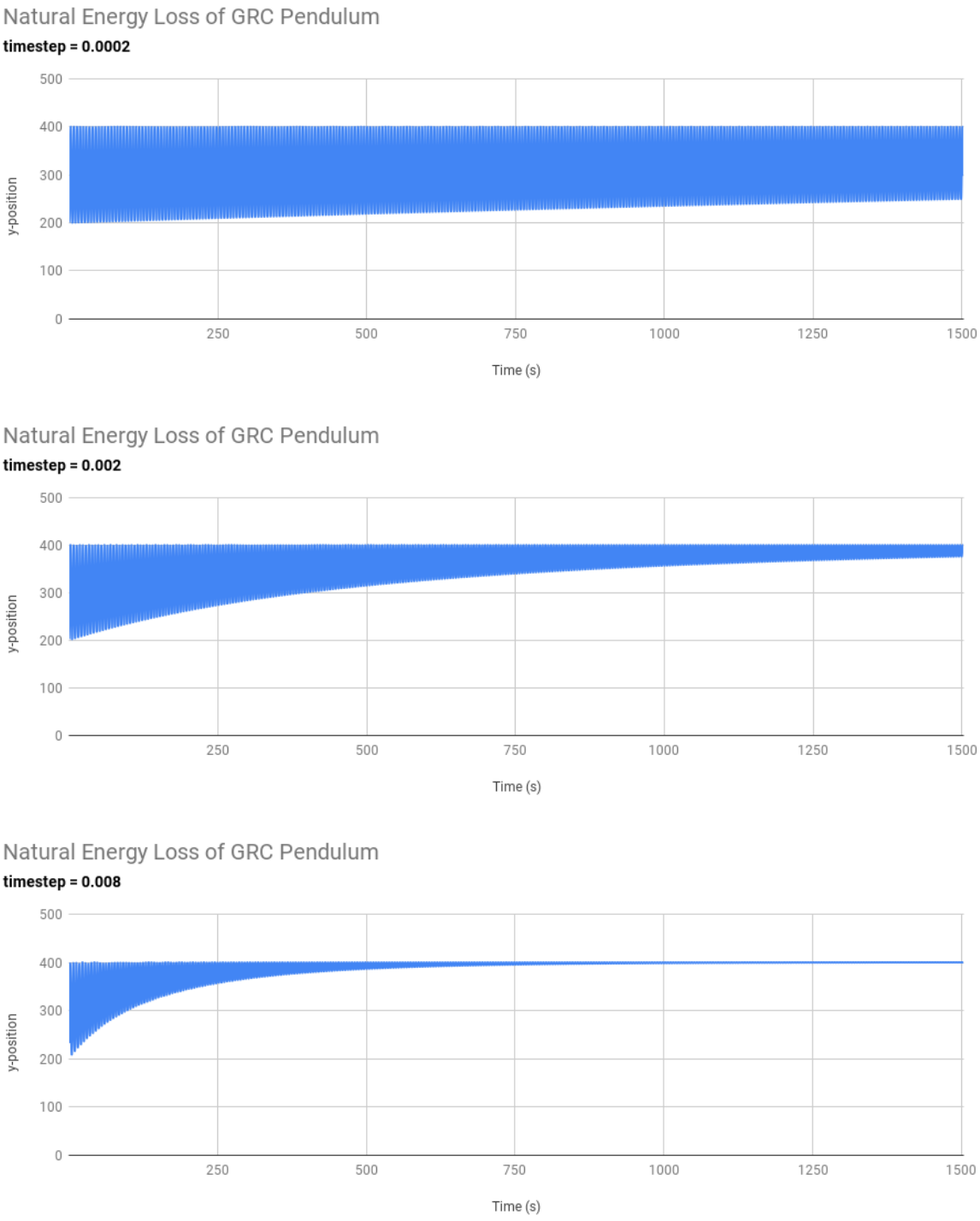


Figure 16: y-position vs time graphs of RGC pendulum systems at different Δt

System convergence depends on the Gauss-Seidel solver (Section 2.2.3), meaning higher values of $constraintSolvingIterationNumber$ result in faster convergence. Conversely, diver-

gence can occur when the truncation error overshoots. Overshoot typically occurs when a sudden external force is applied to the system (collisions and user interactions). Stiffer systems with a larger s are also more likely to overshoot and diverge. Similarly, systems are also more likely to diverge with larger values of $constraintSolvingIterationNumber$ as any overshoot will also be magnified $constraintSolvingIterationNumber$ times. Typically, the truncation error is more likely to cause undershoot, resulting in energy loss and convergence. In general, RGCs are mainly effected by divergence when there are rapid, sudden, or large movements.

5 Conclusion

With the improvement of computing technology, physical modelling has become a much more prevalent field of study. Computational modelling has opened a new perspective of how humans perceive the world. This paper discussed modelling techniques of particle-based systems with penalty (Equation 23) and relaxed geometric constraints (Equation 38). Additionally, the system also utilizes a predictor-corrector semi-implicit Euler integration algorithm with a local iterative solver (Algorithm 1). To summarize, penalty constraints are force-based, energy conservative, and easily damped. Unfortunately, they also diverge quickly for rigid systems and often require a large amounts of force penalties. On the other hand, RGCs are easier to implement, more versatile, and can be more rigid. Disadvantages of RGCs include their Δt dependence and lack of energy conservation. As the final verdict, RGCs are generally a better choice for modelling non-rigorous physics simulations. Whereas PCs are suitable for modelling non-rigid physically accurate systems.

Only a small sample of the existing modelling techniques was discussed. With all the different integration algorithms, system solvers, and constraint algorithms, the possible combinations are endless. A majority of the described techniques are rapidly evolving as technology and understanding improves with time. In the modern world where physical modelling

can encompass video game physics to highly technical engineering analysis, the field of computational mathematics is a relevant and imperative subject. Some applications of RGCs and PCs include structural modelling, stress analysis, animation, and interactive surgical simulations (Bender). Further study of this area is recommended due to the limitations imposed on this paper. Consequently, this paper was unable to analyze a major method in the literature using differential maintained constraints (Witkin). Perhaps extensions to higher dimensions along with specific analyses of different use cases would provide a more comprehensive understanding of the subject. But arguably, the most important step is to continuously experiment with new methods and resolve the current issues, elevating computer modelling to the next stage.

6 Bibliography

Adams, Douglas E. "ME 563 Mechanical Vibrations Lecture #20 - Purdue University." ME 563 Mechanical Vibrations Lecture #20, Purdue University, 2010, engineering.purdue.edu/ deadams/ME563/lecture2010.pdf. Accessed Nov 28, 2021.

Bender, Jan, et al. "A Survey on Position-Based Simulation Methods in Computer Graphics." Computer Graphics Forum, vol. 33, no. 6, 2014, pp. 228–251., doi:10.1111/cgf.12346.

Clavet, Simon, et al. "Particle-Based Viscoelastic Fluid Simulation." Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation - SCA '05, 2005, doi:10.1145/1073368.1073400.

Fitzpatrick , Richard. Numerical Errors, 29 Mar. 2006, farside.ph.utexas.edu/teaching/329/lectures/node33.html. Accessed Jun 20, 2021.

Goldstein, Herbert, et al. Classical Mechanics. Pearson, 2014.

Gutiérrez, José Manuel, et al. "The 'Gauss-Seidelization' of Iterative Methods for Solving Nonlinear Equations in the Complex Plane." Applied Mathematics and Computation, vol. 218, no. 6, 2011, pp. 2467–2479., doi:10.1016/j.amc.2011.07.061.

Jakobsen, Thomas. (2001). Advanced character physics. In-Game Developers Conference Proceedings.

Liu, Mingzhu, et al. "Convergence and Stability of the Semi-Implicit Euler Method for a Linear Stochastic Differential Delay Equation." Journal of Computational and Applied Mathematics, vol. 170, no. 2, 2004, pp. 255–268., doi:10.1016/j.cam.2004.01.040.

Macklin, Miles, et al. "XPBD." Proceedings of the 9th International Conference on Motion in Games, 2016, doi:10.1145/2994258.2994272.

Müller, Matthias, et al. "Position based dynamics." Journal of Visual Communication and Image Representation 18.2 (2007): 109-118.

Nealen, Andrew, et al. "Physically Based Deformable Models in Computer Graphics." Computer Graphics Forum, vol. 25, no. 4, 2006, pp. 809–836., doi:10.1111/j.1467-8659.2006.01000.x.

Spiegel, Murray R. Schaum's Outline of Theory and Problems of Theoretical Mechanics with an Introduction to Lagrange's Equations and Hamiltonian Theory. Schaum Publishing Co., 1994.

Strang, Gilbert. Introduction to Linear Algebra. Wellesley-Cambridge Press, 2016.

Witkin, Andrew, and David Baraff. Physically Based Modeling, Siggraph '97, 1997, www .cs.cmu.edu/ baraff/sigcourse/, Accessed 10 April 2021.

7 Appendix A: Additional Resources

All demonstrations are accessible on a modern computer browser (Google Chrome, Microsoft Edge, Brave, Firefox, Safari is untested). The demonstrations implements the stretch constraint for both RGCs and PCs, where $C(x_1, x_2) = |x_1 - x_2| - r$.

Main webpage for all demonstrations and implementations: `https://onlinedocumentation.github.io/Math-EE/`

All Source Codes: `https://github.com/onlineDocumentation/Math-EE`

Penalty Constraints demo: `https://onlinedocumentation.github.io/Math-EE/Math-EE-Penalty-Constraints/index.html`

Penalty Constraints Source Code: `https://github.com/onlineDocumentation/Math-EE/tree/main/Math-EE-Penalty-Constraints`

Relaxed Geometric Constraints demo: `https://onlinedocumentation.github.io/Math-EE/Math-EE-Relaxed-Geometric-Constraints/index.html`

Relaxed Geometric Constraints Source Code: `https://github.com/onlineDocumentation/Math-EE/tree/main/Math-EE-Relaxed-Geometric-Constraints`

Raw Data for Pendulum Graphs: `https://onlinedocumentation.github.io/Math-EE/pendulumdata.html`

8 Appendix B: Detailed Calculations

8.1 Detailed PC Stretch Constraint Derivation

8.1.1 Part 1: No damping

Given $C_j(x_1, x_2) = |x_1 - x_2| - r = \sqrt{(x_1 - x_2) \cdot (x_1 - x_2)} - r$

$$F_{x_1} = \frac{\partial E}{\partial x_1} = -kC_j \cdot \frac{\partial C_j}{\partial x_1}$$

$$F_{x_2} = \frac{\partial E}{\partial x_2} = -kC_j \cdot \frac{\partial C_j}{\partial x_2}.$$

First, solving for $\frac{\partial C_j}{\partial x_1}$ by using the chain rule then product rule

$$\frac{\partial C_j}{\partial x_1} = \frac{1}{2\sqrt{(x_1 - x_2) \cdot (x_1 - x_2)}}((1-0)(x_1 - x_2) + (x_1 - x_2)(1-0))$$

$$= \frac{2(x_1 - x_2)}{2\sqrt{(x_1 - x_2) \cdot (x_1 - x_2)}}$$

$$= \frac{x_1 - x_2}{|x_1 - x_2|}.$$

Combining yields,

$$F_{x_1} = -k(|x_1 - x_2| - r) \cdot \frac{x_1 - x_2}{|x_1 - x_2|}$$

Similarly,

$$\frac{\partial C}{\partial x_2} = \frac{1}{2\sqrt{(x_1 - x_2) \cdot (x_1 - x_2)}}((0-1)(x_1 - x_2) + (x_1 - x_2)(0-1))$$

$$= -\frac{x_1 - x_2}{|x_1 - x_2|}$$

$$F_{x_2} = k(|x_1 - x_2| - r) \cdot \frac{x_1 - x_2}{|x_1 - x_2|}.$$

8.1.2 Part 2: With damping

Penalty force discretization with damping, extending from Section 7.1.1

$$\Delta x_i = w_i(-kC_j - \mu\dot{C}_j) \cdot \frac{\partial C_j}{\partial x_i}\Delta t^2$$

First calculating $\dot{C}_j$, where

$$\dot{C}_j = \frac{d}{dt}(\sqrt{(x_1 - x_2) \cdot (x_1 - x_2)} - r)$$

$$= \frac{1}{2\sqrt{(x_1 - x_2) \cdot (x_1 - x_2)}} \left[\left(\frac{dx_1}{dt} - \frac{dx_2}{dt} \right) \cdot (x_1 - x_2) + (x_1 - x_2) \cdot \left(\frac{dx_1}{dt} - \frac{dx_2}{dt} \right) \right].$$

Of which $\frac{dx_i}{dt}$ is the velocity of x_i, v_i. Replacing results in,

$$= \frac{2(v_1 - v_2) \cdot (x_1 - x_2)}{2\sqrt{(x_1 - x_2) \cdot (x_1 - x_2)}}$$

$$= \frac{(v_1 - v_2) \cdot (x_1 - x_2)}{|x_1 - x_2|}.$$

Combining all the components from part 1 and part 2 give,

$$\Delta x_1 = w_1 \left(-k(|x_1 - x_2| - r) - \mu \frac{(v_1 - v_2) \cdot (x_1 - x_2)}{|x_1 - x_2|} \right) \cdot \frac{x_1 - x_2}{|x_1 - x_2|}\Delta t^2$$

$$\Delta x_2 = -w_2 \left(-k(|x_1 - x_2| - r) - \mu \frac{(v_1 - v_2) \cdot (x_1 - x_2)}{|x_1 - x_2|} \right) \cdot \frac{x_1 - x_2}{|x_1 - x_2|}\Delta t^2.$$

8.2 Detailed RGC Stretch Constraint Derivation

Given $C_j(x_1, x_2) = |x_1 - x_2| - r = \sqrt{(x_1 - x_2) \cdot (x_1 - x_2)} - r$

$$\Delta q_{j,i} = -w_{j,i} s_j \frac{C_j(q_j)}{\sum_u^n w_{j,u} [\nabla_{q_{j,k}} C_j(q_j) \cdot \nabla_{q_{j,k}} C_j(q_j)]} \nabla_{q_{j,i}} C_j(q_j).$$

$q_j = [x_1 \;\; x_2]^T$, therefore $q_{j,1} = x_1$ and $q_{j,2} = x_2$. Using the above equation yields,

$$\Delta x_1 = -w_1 s \frac{C_j(q_j)}{\sum_u^n w_{j,u}[\nabla_{x_u} C_j(q_j) \cdot \nabla_{q_{j,u}} C_j(q_j)]} \nabla_{x_1} C_j(q_j)$$

$$\Delta x_2 = -w_2 s \frac{C_j(q_j)}{\sum_u^n w_{j,u}[\nabla_{q_{j,u}} C_j(q_j) \cdot \nabla_{q_{j,u}} C_j(q_j)]} \nabla_{x_2} C_j(q_j).$$

Next, each section is solved by steps. First, $\nabla_{x_i} C_j(q_j)$ is calculated for x_1 and x_2,

$$\nabla_{q_{j,1}} C_j(q_j) = 1 - 0 = 1$$

$$\nabla_{q_{j,2}} C_j(q_j) = 0 - 1 = -1.$$

Then, the denominator $\sum_u^n w_{j,u}[\nabla_{x_u} C_j(q_j) \cdot \nabla_{q_{j,u}} C_j(q_j)]$ is simplified to,

$$\sum_u^n w_{j,u}[\nabla_{x_u} C_j(q_j) \cdot \nabla_{q_{j,u}} C_j(q_j)] = w_1 \cdot 1 + w_2 \cdot 1 = w_1 + w_2.$$

Lastly, substituting each component yields,

$$\Delta x_1 = -s \frac{w_1}{w_1 + w_2} (|x_1 - x_2| - r) \frac{x_1 - x_2}{|x_1 - x_2|}$$

$$\Delta x_2 = s \frac{w_2}{w_1 + w_2} (|x_1 - x_2| - r) \frac{x_1 - x_2}{|x_1 - x_2|}.$$