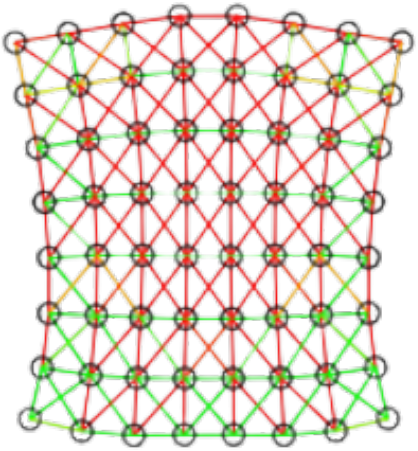
Modelling Constrained 2D Particle-Based Systems

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

The foundation of the visible universe consists of subatomic particles and atoms. As these particles interact, they form bonds and create visible matter. These intrinsic interactions between bonds and particles dictate behaviour on the universal scale. As such, a fundamental question arises; how do those atomic bonds work, and is it possible to generalize them with math? The goal of this essay is to derive different methods of modelling constrained particle based-systems. More specifically, this paper will investigate, analyze, and compare the use of penalty constraints and relaxed geometric constraints.

The essay begins with an introduction to constrained particle systems. After the background information, the essay then delves into penalty constraints (Equation 24). Along with an initial simplification with a direct case, a generalization for penalty constraints is constructed. Following the generalization, extensions, such as damping, are introduced while analyzing the programmatic implementation of penalty systems. Additional discussions about the efficacy and limitations of such constraints are also provided. Next, relaxed geometric constraints are analyzed (Equation 50). Following a similar structure as the previous section, this segment begins with an introduction and case-specific example. Correspondingly, the relaxed geometric constraint method is also generalized. Lastly, the section contains a discussion on damping, which is accompanied by an analysis of the programmatic implementation. After detailed scrutiny of both methods, a direct comparison between the two methods is examined. Lastly, future study and possible use-cases are discussed.

To help visualize the mathematics discussed within this essay, a simple programmatic implementation is provided. Further access information can be found in Appendix A.

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 $\frac{dF}{dt}$
Row vector notation $\begin{bmatrix} x_1 & x_2 & ... & x_n \end{bmatrix}^T$	Equivalent to the column form, except it saves space $\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$
$\nabla_q F(q)$	The gradient of F in terms of q, $\begin{bmatrix} \frac{\partial}{\partial q_1} F(q) & \frac{\partial}{\partial q_2} F(q) & \cdots & \frac{\partial}{\partial q_n} F(q) \end{bmatrix}^T$
$\nabla_{x_i} F(x_1, x_2, ...x_n)$	This examines how each component of x_i changes in the function F, $\nabla_i F(x_1, x_2, ...x_n)$ Looks at the specific partial derivative w.r.t. x_i $\frac{\partial}{\partial x_i} F(x_1, x_2, ..., x_n)$
$x^{(k-1)}, x^{(k)}, x^{(k+1)}$	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

A constrained particle system consists of an array of particle objects and an array of constraint objects. Object p is defined as the concatenated values of all n distinct particles, $[\ p_1\ p_2\ p_3\ \cdots\ p_n\ ]^T$ (Bender) Similarly, C is also defined as the concatenated values of all m different constraints $[\ C_1\ C_2\ C_3\ \cdots\ C_m\ ]^T$.

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

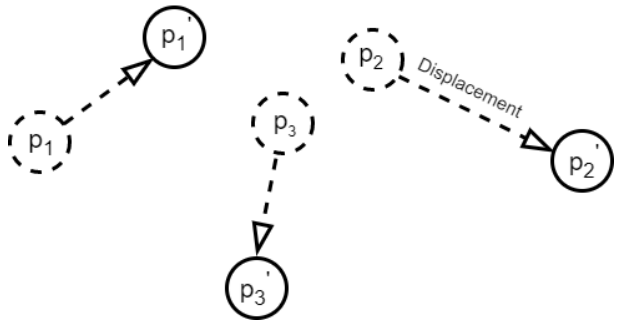


Figure 1: New particle positions after Δt

Each particle, p_i , will have a corresponding property, such that a property i $\in [1, \ldots, n]$ has components $x_i, v_i, a_i, m_i, r_i, F_i$. Particles are very easy to simulate due to their simplistic nature - they only translate.

In contrast, a constraint takes in certain properties to determine its behaviour and influence. A constraint $C_j \in C$, can be described by some function C_j, such that $C_j : \mathbb{R}^n \to \mathbb{R}$. The set of properties it receives is called the *generalized properties* q_j, where q_j is the concatenation of all the necessary generalized properties, $[\ q_{j,1}\ q_{j,2}\ q_{j,3}\ \cdots\ q_{j,n}\ ]^T$. And $q_{j,i}$ can be, but is not limited to, some properties of a particle object, such that $q_{j,i} \in \{x_i, v_i, a_i, m_i, r_i, F_i\}$. For this paper, there will always be some stiffness $k \in [0, \infty)$ where $k \in q_j$. The constraint also has some cardinality $n_j = n(q_j)$. Moreover, the constraints will be holonomic and scleronomic. Holonomic constraints satisfy the condition such that $C_j(q_{j,1}, q_{j,2}, q_{j,3}, \cdots, q_{j,n}) = 0$

(commonly named equality or bilateral constraints in literature). Scleronomic constraints have q_j such that $t \notin q_j$. Some examples of these constraints include stretch constraints, bend constraints, and volume constraints.

2.2 Particle System Solvers and Integration Schemes

The relationships in a particle system are represented by a system of differential equations that dictate their changes and behaviours. These complex systems are based upon the following basic kinematic relationships,

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

Complex systems are solved by decomposition into its basic components, such that it resemble Equations 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

The first step of semi-implicit Euler is Euler approximation, a local linearization algorithm where an approximate solution is reached by extending the slope.

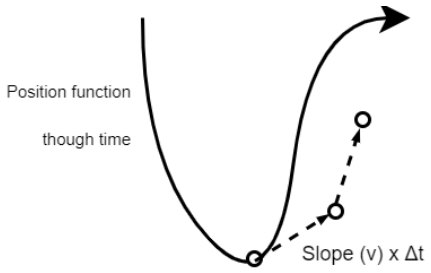


Figure 2: Visualization of Euler algorithm

Compared to normal Euler, semi-implicit Euler is one degree higher and completes the Euler

step on velocity then uses the new value for position. Numerically, this is described as

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

and when combined, it yields,

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

Semi-implicit Euler behaves implicitly depending on the size of the timestep (Muller). For lower values of Δt, position shift decreases notably and the system behaves more implicitly. Whereas for larger values of Δt, the system behaves more explicitly. This integration 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. Furthermore, this algorithm is also symplectic, meaning it theoretically conserves energy. However, in practice, the system can be both stable/convergent or unstable/divergent. To make this algorithm more accurate, the solver iteration count per frame can be increased while decreasing the timestep, Δt.

2.2.2 Predictor - Corrector

The Predictor - Corrector algorithm finds the corrective position impulses based on a predicted future position, and directly calculates the final velocity impulse with the previous position.

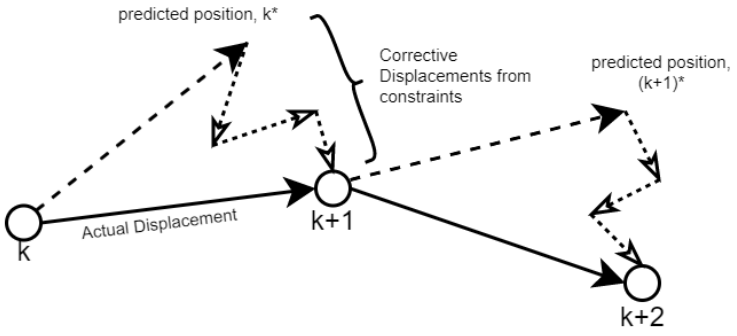


Figure 3: The predictor-corrector model over 3 iterations

This can be described mathematically as,

$$x^* = x^{(k)} + v^{(k)} \cdot \Delta t + a^{(k)} \cdot \Delta t^2$$
$$x^{(k+1)} = x^* + \Delta x^{(k)}$$
$$v^{(k+1)} = \frac{x^{(k+1)} - x^{(k)}}{\Delta t}. \tag{4}$$

The Predictor-Corrector model effectively resolves most overshoot problems from extreme forces. (Clavet)

2.2.3 Gauss-Seidel

To find the system of successive displacements, Δx, for the particle system p, the constraint system is approached with a Gauss-Seidel-like local iterative solver. Local iterative solvers are generally easier to implement and faster to compute (Jakobsen) because matrix operations used in global solvers are difficult to optimize. Furthermore, iterative solvers are more versatile and responsive. Gauss-Seidel works by decomposing a complicated system into two simpler components and repeatedly solves them until an approximate solution is reached. Since most constraint systems are nonlinear, an iterative "Gauss-Seidelization" (Gutiérrez) of the original Gauss-Seidel algorithm is used. Mathematically for a system of constraints ϕ_i, this is represented 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 x_i is lower triangular, with the bottom being $k+1$ and the top being k iterations. In practice, this behaves as if the system updates after each constraint is solved; the updated value of the solved constraint is used to solve the following constraints. Referring back to linear Gauss-Seidel, it becomes possible to analyze convergence rates of the iterative version. For a linear system $Ax = b$, A is decomposed into S and T such that $Sx^{(k+1)} =$

$Tx^{(k)} + b$ (Strang), where S contains the lower triangular matrix of A. The spectral radius of the system is $\rho(B)$, where $B = S^{-1}T$. This, unfortunately, is nonsensical since a valid S or T cannot be constructed for nonlinear systems. Luckily, $\rho(B)$ is equivalent to the eigenvalue, $|\lambda|_{max}$ of B, and $|\lambda|_{max} = (cos\frac{\pi}{n+1})^2$ (Strang). Admittedly, this concept becomes confusing outside the use of linear systems, but since the spectral radius can be calculated without knowing B, it still provides a good overview of the overall convergence rate. The n dimension of B is equivalent to the number of elements in the vector Δx. Hence, n is also equivalent to the number of constraints. Generally, for k iterations, the error is multiplied by ρ^k (Strang). Therefore, the convergence rate decreases as n increases. On the contrary, Gauss-Seidel can diverge for certain cases. Detailed analysis will be available in later sections.

Combining all of the described approaches creates 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 novelties such as collision handling are included for an enhanced user experience. (Code available in Appendix A)

Algorithm 1 General system solver

```
// 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()
    // Using Equation 3, calculate the velocity from the previous position for all particles

    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

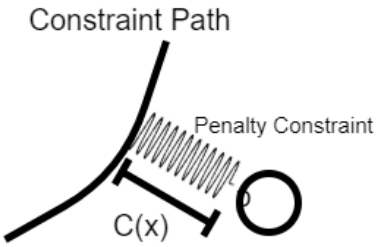


Figure 4: Penalty Constraint Visualization

Penalty constraints are straightforward - imagine attaching the particle to the constraint path with a stiff spring. However, this system has an obvious problem: the forces are enormous for stiff springs, and a rigid system would have infinitely large forces. The excessive force impulse creates large corrective displacements, which, when combined with the natural error with timestep, causes the penalty systems to quickly become unstable. According to Hooke's law, the spring force is proportional to the displacement from the equilibrium position.

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

Taking a look at a specific example, the pair of particles in the constraint can be solved geometrically to find the penalty forces. Imagine two particles constrained by length, as shown in Figure 5.

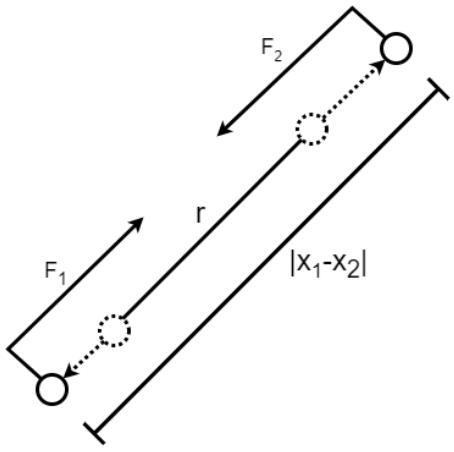


Figure 5: PC stretch constraint

Given $F = -k\Delta s$,

$$\begin{aligned} F_{p_1} &= -k(|x_1 - x_2| - r)\frac{x_1 - x_2}{|x_1 - x_2|} \\ F_{p_2} &= k(|x_1 - x_2| - r)\frac{x_1 - x_2}{|x_1 - x_2|}, \end{aligned} \tag{7}$$

where $\Delta s = |x_1 - x_2| - r$, and $\frac{x_1 - x_2}{|x_1 - x_2|}$ is the direction unit vector. Unfortunately, this approach is limited due to a lack of mathematical generalization. Witkin outlines a potential energy function derivation process that can be used to solve penalty constraints.

Let $C_j(x_1, x_2, x_3, ..., x_n) = 0$, $[\ x_1\ x_2\ \cdots\ x_n\ ]^T$, and $F_{C_j} = -kC_j$, where C_j describes the displacement of the constraint function. Then the energy of the constraint and the corresponding component for each particle p of the constraint can be derived. The energy function will be bounded by 0 and C_j; it is the work done over the interval $[0, C_j]$.

For $F_{C_j} = -kC_j$,

$$\begin{aligned} E_{C_j} &= \int_0^{C_j} F_{C_j} \cdot dC_j = \int_0^{C_j} -kC_j \cdot dC_j \\ &= -\frac{1}{2}kC_j \cdot C_j \Big|_0^{C_j} = -\frac{1}{2}kC_j \cdot C_j - 0 \\ E_{C_j} &= -\frac{1}{2}kC_j \cdot C_j. \end{aligned} \tag{8}$$

The following penalty force can be derived from the energy function E_C. Force and energy are related by the formula,

$$F_{x_i} = \frac{\partial E}{\partial x_i}, \tag{9}$$

where x_i is the moving particle. This can be simplified further by substituting in Equation 8, then

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

Therefore, the system of concatenated forces, $[\ F_{x_1}\ F_{x_2}\ \cdots\ F_{x_n}\ ]^T$, can be represented as

$$\begin{aligned} F_x &= (\nabla_x E)^T \\ F_x &= C_j \cdot (\nabla_x C_j)^T. \end{aligned} \tag{11}$$

To find the final displacement Δx, the combined force vector $F_x = [\ F_{x_1}\ F_{x_2}\ F_{x_3}\ \cdots\ F_{x_n}\ ]^T$ is discretized. The force is first decomposed to the acceleration by using Newton's Second

law, $F = ma$.

$$a = M^{-1}F_x.$$

Discretizing yields,

$$\Delta x = WF_x\Delta t^2, \tag{12}$$

where W is the inverse diagonal mass matrix M^{-1} of size $n \times n$. Then expanding Δx results in each Δx_i,

$$\Delta x_i = W\nabla_{x_i}E^T\Delta t^2$$

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

To verify the derived formula, it will be applied to the case example stated in Figure 5. The mathematical representation of the constraint function will be

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

since it maintains a distance r from two particles with positions x_1 and x_2. Using Equation 10, 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

Friction plays a major role in physical modelling, thereby playing a crucial role in penalty constraints. Moreover, many instabilities caused by the numerical integration algorithm

can be negated with an adequate amount of damping. Typically, a damping force, D_j, is proportional to the velocity. Since this is the damping for a constraint, the respective velocity will be the velocity of the constraint itself (not of the particles). This is denoted mathematically as $v_{C_j} = \dot{C}_j$. Hence, D_j can be rewritten with some arbitrary constant,

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

Next, the penalty forces with damping are derived. Similar to the first part, a potential energy function will be required.

Let the function $F_{C_j} = -kC_j - D_j$, 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 the integral,

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

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

Now the individual forces can be found from Equation 18, the energy function. Recall Equations 10 and 11,

$$F_x = (\nabla_x E)^T \ \ and \ \ F_{x_i} = \frac{\partial E}{\partial x_i}.$$

Therefore,

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

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

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

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

Performing a swap and factor then yields

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

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

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

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

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

Lastly, with the same method in the generalization, the force from Equation 20 is discretized to calculate the position. Given $a_i=w_iF_{x_i}$ and $\Delta x_i=a_i\Delta t^2$, then

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

Once again, applying the results on the constraint function $C_j(x_1,x_2)=|x_1-x_2|-r$ yields

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

(Detailed derivation available in Appendix B)

3.3 Programmatic Implementation and Analysis

To verify the mathematical derivations, the example case is 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

With the iterative Gauss-Seidel algorithm, a higher iteration number is introduced to accelerate system convergence. Unfortunately, the stiffness becomes dependent on the number of constraint solving iterations. Due to the stiffness' dependence on the iteration number and its resulting arbitrary units, this method for simulating constrained systems is not suitable for realistic physical modelling. The best way to negate this issue is by replacing the iterative solver with a global solver that directly computes the solution (Witkin). On the other hand, different iteration algorithms and constraint modelling techniques can be used to arrive at more accurate solutions.

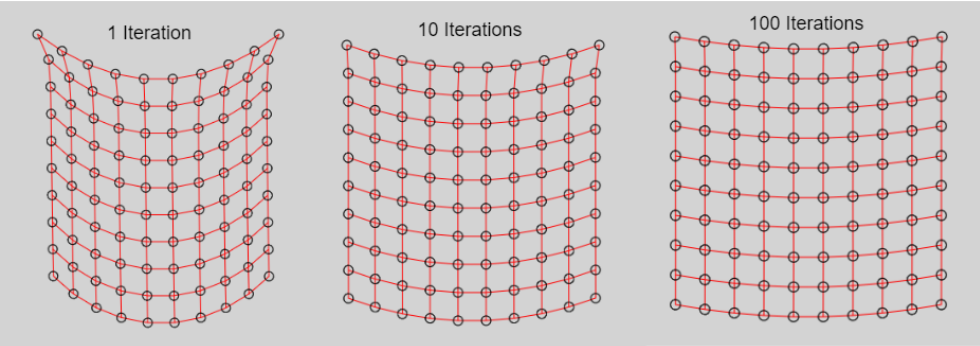


Figure 6: PC stiffness behaviour for different iterations (the uniform red colour is due to the high forces on the constraints with a stiffness of 100000)

It is observed that higher iterations result in higher stiffness. Intuitively, this makes sense because the timestep for each iteration is not compensated.

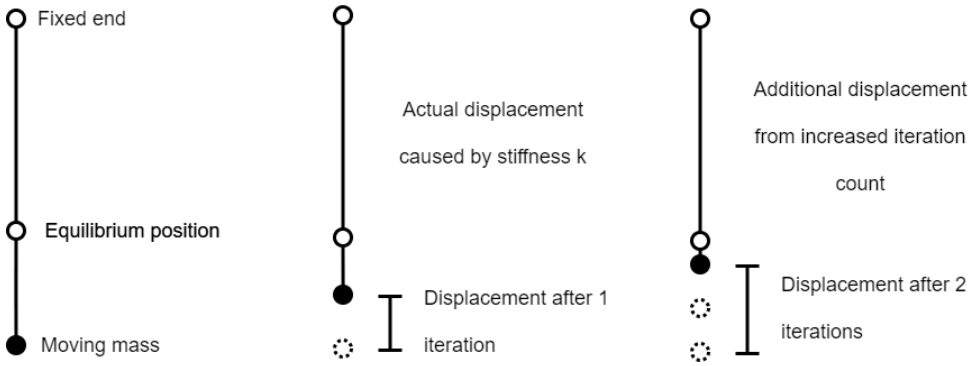


Figure 7: Increased PC stiffness with iteration number

The resulting constraint behaves as if it has larger displacement caused by larger stiffness. Compared to simply changing the stiffness constant (k), changing the number of iterations provides more continuous behaviour that can prevent overshoot. Especially when the forces or the timestep is too large, the system overshoots and diverges because it cannot counteract the penalty forces.

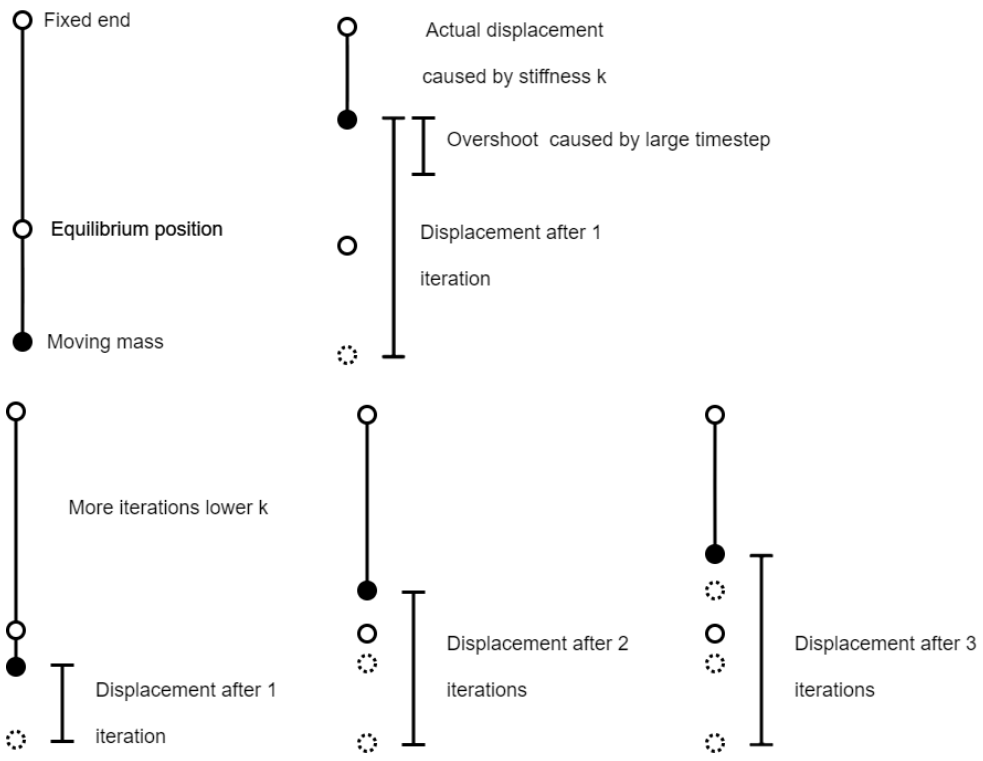


Figure 8: Induced PC stiffness stability for increased iteration number

For the best rigid systems, it is important to find an empirical balance of the iteration count, k, and timestep.

3.3.2 Convergence and Energy Conservation

Ideal spring constraints do not converge; instead, they oscillate about C = 0 between different stable states. Convergence to a static equilibrium is only induced when energy is lost within the system, which is typically caused by constraint damping or particle damping.

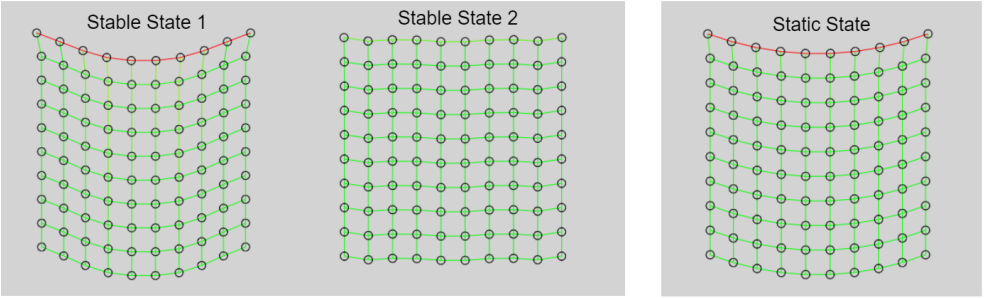


Figure 9: Stable oscillatory behaviour vs convergent behaviour

Overall system convergence is dependent on the Gauss-Seidel solver. Generally, the larger the system, the longer it takes to converge. To speed up convergence without inducing arbitrary stiffness issues, the number of iterations per frame can be increased while lowering the timestep as compensation. Although an undamped system is supposedly closed and isolated, its behaviour in practice demonstrates otherwise. Energy naturally dissipates as time passes. One factor that affects this issue is the timestep. As Δt increases, the truncation error also increases quadratically. Larger Δt values can lead to faster energy loss. Moreover, if there are more particles, there will be more truncation errors, resulting in greater dissipation. Specifically, regarding constraints, larger penalty forces can also induce greater errors during the force discretization (of which Δt also plays a role). This issue can be analyzed 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)

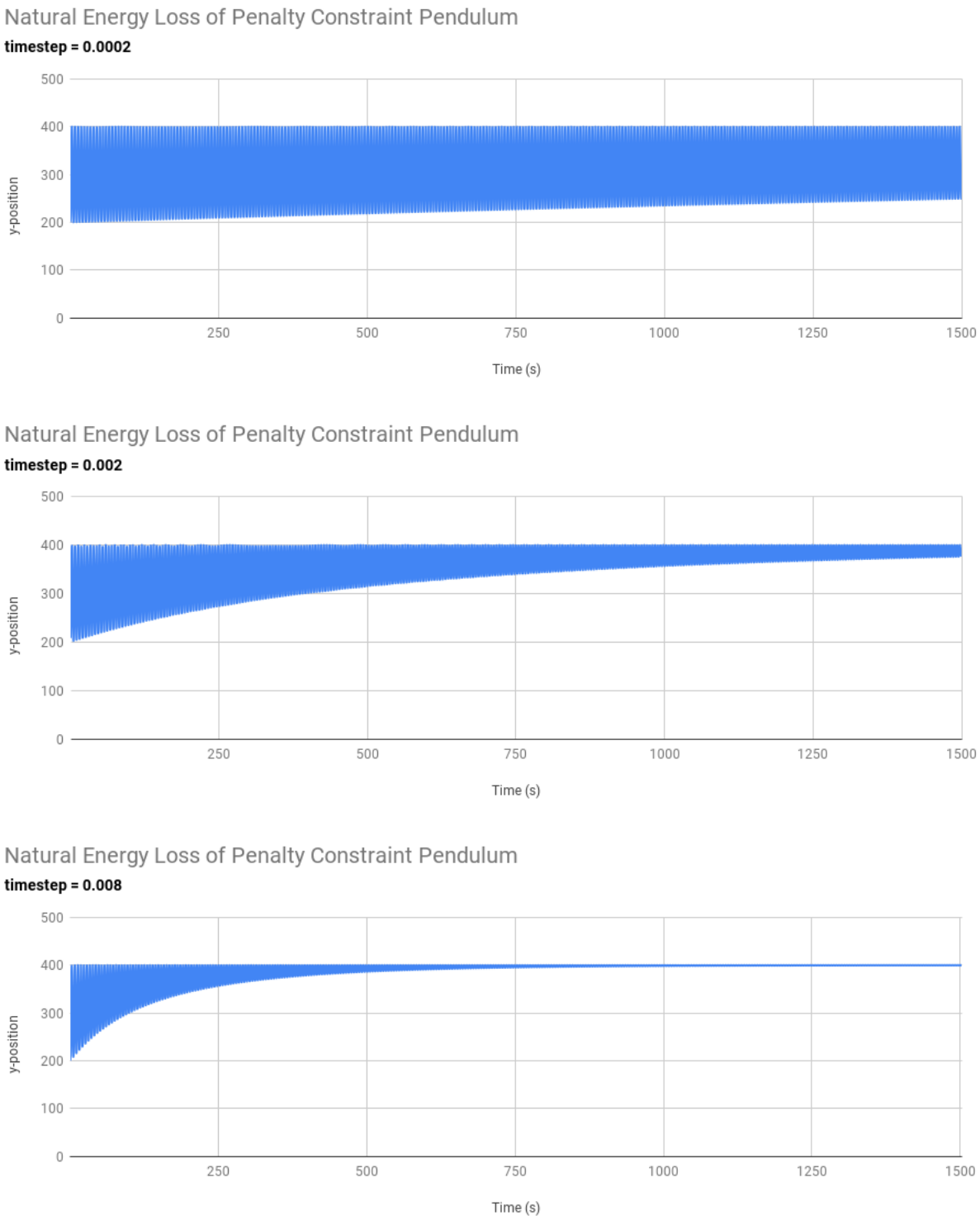


Figure 10: y-position vs time graphs of PC pendulum systems at different Δt

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

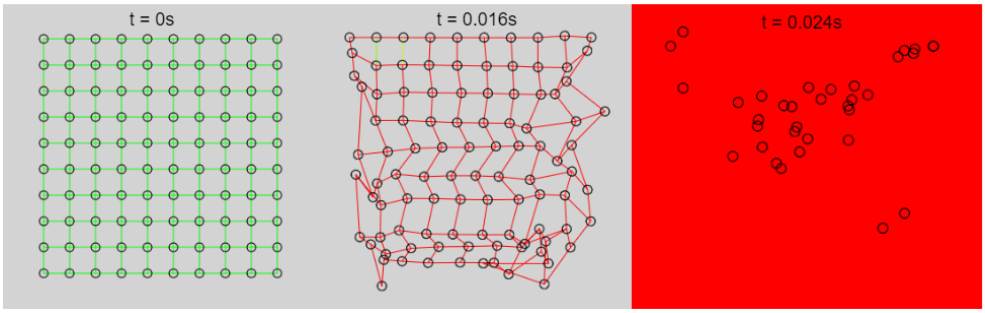


Figure 11: PC divergence at $\Delta t = 0.008$

Divergent behaviour is caused by erroneous force discretizations for each time frame. These errors propagate and result in even larger penalty forces, leading to a positive feedback loop. Divergence can be minimized with the inclusion of damping; damping provides a buffer for overshoot. Moreover, divergence can be resisted by either increasing the iteration count, lowering the forces (ex. gravity), or decreasing the timestep.

4 Relaxed Geometric Constraints (RGC)

4.1 Explanation and Generalization

RGC is a geometric method that uses mathematical relaxation. Its process involves decomposing a problem into two parts, first solving the simpler version, then repeating the process on the leftover counterpart. Consequently, it only results in an approximation of the exact solution. Visually, it appears that the "tense" constraint is corrected (relaxed) to its ideal form. THis technique is similar but less complicated than Lagrangian relaxation. The goal of this method is to project the current position to the next closest legal position that satisfies the constraint. Once again, a straightforward geometric derivation of a simple case is presented below in Figure 12.

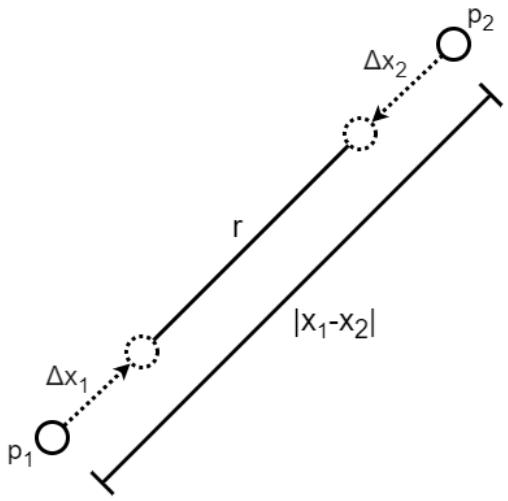


Figure 12: RGC Stretch Constraint

Where r is the correct length and $|x_1 - x_2|$ is the current length. Since p_1 and p_2 have the same mass, both will move the same amount to reach the ideal position. Intuitively, the contraction will be collinear with x_1 and x_2.

Therefore,

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

Then, multiplying the directional unit vector gives

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

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

RGCs are purely position-based, which offers more control. But as a result, extra caution is needed to conserve realistic physical meaning. To do so, the system's linear momentum and angular momentum must be conserved. Numerically, this is represented by

$$\sum_i^n m_i \Delta x_i \;\; and \;\; \sum_i^n r_i \times \Delta x_i. \tag{26}$$

Where r_i is the distance to some common rotation center of the whole particle system and x is the position value of a particle, p. These systems of equations will be useful for providing a limiting condition as well as the weighted scaling factors for different masses. Currently, it is assumed that all masses in the particle system are the same. The process starts by setting up the system to be solved for some constraint C_j, which is

$$\begin{aligned}x_1' &= x_1 + \Delta x_1 \\ x_2' &= x_2 + \Delta x_2 \\ &\vdots \\ x_n' &= x_n + \Delta x_n,\end{aligned}$$

such that

$$C_j(x_i) = 0 \;\; and \;\; C_j(x_i + \Delta x_i) = 0. \tag{27}$$

For the sake of generality, the list of generalized positions, x_i, can be joined into a vector x. Subsequently, their lists of respective displacements Δx_i become Δx. Uing the new notation,

$$x' = x + \Delta x. \tag{28}$$

Therefore,

$$C_j(x + \Delta x) = 0. \tag{29}$$

RGC calculates some Δx to satisfy C_j. Solving nonlinear constraint functions are difficult, so the constraints are first locally linearized (Bender). This method reflects that of Newton's method for finding roots. Mathematically, this is presented as,

$$C_j(x + \Delta x) = C_j(x) + \nabla_x C_j(x) \cdot \Delta x + O(|\Delta x|^2) = 0, \tag{30}$$

where $O(|\Delta x|^2)$ is the error term caused by the truncation of higher degree derivatives. The error term is closely linked to the issue of erroneous displacements. Erroneous displacemets lead to energy loss and inaccurate results. To simplify, the error term can be ignored. resulting in an approximation of

$$C_j(x) + \nabla_x C_j(x)^T \cdot \Delta x \approx 0. \tag{31}$$

Equation 29 needs to be solved to find Δx. Currently, this system is unsolvable because it is underdetermined; this indicates there must be another relationship of Δx. The previously discussed conservation of linear momentum (Equation 26) can be used to limit the possible values of Δx. Since linear momentum must be conserved, Δx must be collinear with $\nabla_x C_j(x)$ or else extra directions will be introduced. Now, Δx is rewritten as

$$\Delta x = \lambda \nabla_x C_j(x)^T. \tag{32}$$

The goal is to find the relaxation factor λ such that $C_j(x) + \nabla_x C_j(x) \cdot \Delta x \approx 0$. With some substitution,

$$C_j(x) + \nabla_x C_j(x)^T \cdot \lambda \nabla_x C_j(x)^T \approx 0. \tag{33}$$

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

$$\lambda = -\frac{C_j(x)}{\nabla_x C_j(x)^T \cdot \nabla_x C_j(x)^T}. \tag{34}$$

Hence, Δx is determined as

$$\Delta x = -\frac{C_j(x)}{\nabla_x C_j(x)^T \cdot \nabla_x C_j(x)} \nabla_x C_j(x)^T. \tag{35}$$

Next, the displacement for each particle in the system is needed. To do so, the function is broken into its vector counterparts. Firstly, $\nabla_x C_j(x)^T \cdot \nabla_x C_j(x)^T$ can be simplified further,

$$\nabla_x C_j(x)^T \cdot \nabla_x C_j(x)^T = \nabla_x C_j(x) \nabla_x C_j(x)^T$$

$$= \begin{bmatrix} \frac{\partial}{\partial x_1} C_j(x) & \frac{\partial}{\partial x_2} C_j(x) & ... & \frac{\partial}{\partial x_n} C_j(x) \end{bmatrix} \begin{bmatrix} \frac{\partial}{\partial x_1} C_j(x) \\ \frac{\partial}{\partial x_2} C_j(x) \\ \vdots \\ \frac{\partial}{\partial x_n} C_j(x) \end{bmatrix}$$

$$= (\frac{\partial}{\partial x_1} C_j(x))^2 + (\frac{\partial}{\partial x_2} C_j(x))^2 + ... + (\frac{\partial}{\partial x_n} C_j(x))^2$$

$$= \sum_j^n \nabla_{x_j} C_j(x) \cdot \nabla_{x_j} C_j(x). \tag{36}$$

To clarify, the notation $\nabla_{x_i} C_j(x)$ is the term of $\nabla_x C_j(x)$ corresponding to x_i. Substituting back to Equation 36 and expanding yields,

$$\Delta x = -\frac{C_j(x)}{\sum_k^n \nabla_{x_k} C_j(x) \cdot \nabla_{x_k} C_j(x)} \begin{bmatrix} \frac{\partial}{\partial x_1} C_j(x) \\ \frac{\partial}{\partial x_2} C_j(x) \\ \vdots \\ \frac{\partial}{\partial x_n} C_j(x) \end{bmatrix}$$

$$= -\begin{bmatrix} \frac{C_j(x)}{\sum_k^n \nabla_{x_k} C_j(x) \cdot \nabla_{x_k} C_j(x)} \frac{\partial}{\partial x_1} C_j(x) \\ \frac{C_j(x)}{\sum_k^n \nabla_{x_k} C_j(x) \cdot \nabla_{x_k} C_j(x)} \frac{\partial}{\partial x_2} C_j(x) \\ \vdots \\ \frac{C_j(x)}{\sum_k^n \nabla_{x_k} C_j(x) \cdot \nabla_{x_k} C_j(x)} \frac{\partial}{\partial x_n} C_j(x) \end{bmatrix}. \tag{37}$$

Leading to each specific Δx_i, where

$$\Delta x_i = -\frac{C(x)}{\sum_k^n \nabla_{x_k} C_j(x) \cdot \nabla_{x_k} C_j(x)} \nabla_i C_j(x). \tag{38}$$

Unfortunately, Equation 38 only works for particles with the same mass. The next logical step is to generalize the result such that it accounts for different particle masses. To resolve

this issue, some mass scaling factor will be required for each particle. Once again, the mass-scaling relationship can be derived using the conservation of momentum,

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

Systems with constant mass behave such that,

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

Then 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),

$$\sum_{i}^{n} m_i \cdot \frac{1}{m_i} \cdot \Delta x_i = 0 \rightarrow \sum_{i}^{n} \Delta x_i = 0. \tag{39}$$

A similar explanation can be used for angular momentum

$$\sum_{i}^{n} r_i \times m_i \cdot \frac{1}{m_i} \cdot \Delta x_i = 0 \rightarrow \sum_{i}^{n} r_i \times \Delta x_i = 0. \tag{40}$$

As a result, Equation 32 can be modified to include the mass scaling.

$$\Delta x = \lambda W \nabla_x C_j(x)^T, \tag{41}$$

and W is the inverse diagonal mass matrix M^{-1}. Using this equation, a new λ can then be derived. Note that $\nabla_x C_j(x)^T \cdot W \nabla_x C_j(x)^T$ still returns a scalar value despite the inclusion of W. Following the same steps as Equation 34,

$$\lambda = \frac{-C_j(x)}{\nabla_x C_j(x)^T \cdot W \nabla_x C_j(x)^T}. \tag{42}$$

Substituting back to Equation 41,

$$\Delta x = -W \frac{C_j(x)}{\nabla_x C_j(x)^T \cdot W \nabla_x C_j(x)^T} \nabla_x C_j(x)^T. \tag{43}$$

Then simplifying to find Δx_i by firstly expanding the denominator, $\nabla_x C_j(x)^T \cdot W \nabla_x C_j(x)^T$.

$$\nabla_x C_j(x)^T \cdot W \nabla_x C_j(x)^T = \nabla_x C_j(x) W \nabla_x C_j(x)^T$$

$$= \begin{bmatrix} \frac{\partial}{\partial x_1} C_j(x) & \frac{\partial}{\partial x_2} C_j(x) & \dots & \frac{\partial}{\partial x_n} C_j(x) \end{bmatrix} \begin{bmatrix} w_1 & 0 & 0 & 0 \\ 0 & w_2 & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & w_4 \end{bmatrix} \begin{bmatrix} \frac{\partial}{\partial x_1} C_j(x) \\ \frac{\partial}{\partial x_2} C_j(x) \\ \vdots \\ \frac{\partial}{\partial x_n} C_j(x) \end{bmatrix}$$

$$= \begin{bmatrix} \frac{\partial}{\partial x_1} C_j(x) w_1 & \frac{\partial}{\partial x_2} C_j(x) w_2 & \dots & \frac{\partial}{\partial x_n} C_j(x) w_n \end{bmatrix} \begin{bmatrix} \frac{\partial}{\partial x_1} C_j(x) \\ \frac{\partial}{\partial x_2} C_j(x) \\ \vdots \\ \frac{\partial}{\partial x_n} C_j(x) \end{bmatrix}$$

$$= w_1 (\frac{\partial}{\partial x_1} C_j(x))^2 + w_2 (\frac{\partial}{\partial x_2} C_j(x))^2 + \cdots + w_n (\frac{\partial}{\partial x_n} C_j(x))^2$$

$$= \sum_k^n w_k (\nabla_{x_k} C_j(x) \cdot \nabla_{x_k} C_j(x)). \tag{44}$$

Substituting back to Equation 43 and expanding,

$$\Delta x = - \begin{bmatrix} w_1 & 0 & 0 & 0 \\ 0 & w_2 & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & w_4 \end{bmatrix} \frac{C(x)}{\sum_j^n w_j (\nabla_j C(x) \cdot \nabla_j C(x))} \begin{bmatrix} \frac{\partial}{\partial x_1} C(x) \\ \frac{\partial}{\partial x_2} C(x) \\ \vdots \\ \frac{\partial}{\partial x_n} C(x) \end{bmatrix}$$

$$= - \begin{bmatrix} w_1 \frac{C(x)}{\sum_j^n w_j (\nabla_{x_j} C(x) \cdot \nabla_{x_j} C(x))} \frac{\partial}{\partial x_1} C(x) \\ w_2 \frac{C(x)}{\sum_j^n w_j (\nabla_{x_j} C(x) \cdot \nabla_{x_j} C(x))} \frac{\partial}{\partial x_2} C(x) \\ \vdots \\ w_n \frac{C(x)}{\sum_j^n w_j (\nabla_{x_j} C(x) \cdot \nabla_{x_j} C(x))} \frac{\partial}{\partial x_n} C(x) \end{bmatrix}. \tag{45}$$

Hence for each x_i,

$$\Delta x_i = -w_i \frac{C(x)}{\sum_k^n w_k [\nabla_{x_k} C(x) \cdot \nabla_{x_k} C(x)]} \nabla_{x_i} C(x), \tag{46}$$

matching the results from Bender. To verify further, Equation 46 is applied on $C(x_1, x_2) = |x_1 - x_2| - r$ (Figure 12), which results in

$$\begin{aligned} \Delta x_1 &= -\frac{w_1}{w_1 + w_2}(|x_1 - x_2| - r)\frac{x_1 - x_2}{|x_1 - x_2|} \\ \Delta x_2 &= \frac{w_2}{w_1 + w_2}(|x_1 - x_2| - r)\frac{x_1 - x_2}{|x_1 - x_2|}. \end{aligned} \tag{47}$$

This result looks very similar to the one derived using PCs, with the difference being the extra discretization and force-based approach found in PCs. and like before, will be programmatically implemented.

4.2 Damping

Damping of rigid position-based constraints is impossible. Damping depends on the value of $C(q)$, and since a perfectly rigid constraint always satisfy $C(q) = 0$, it cannot be effected by constraint damping. Macklin's XPBD paper verifies this result mathematically by using an energy-based method to derive a compliant position-based constraint solver, which is similar to a combination of the penalty and relaxed geometric constraints. The final equation is,

$$\Delta x = W\frac{-C_j(x) - \widetilde{\alpha}_j\lambda_j - \gamma_j\nabla C_j(x - x^n)}{(1 + \gamma_j)\nabla C_j(x)W\nabla C_j(x)^T + \widetilde{\alpha}_j}\nabla C_j(x)^T. \tag{48}$$

Where α_j is the inverse stiffness k_j^{-1} and $\widetilde{\alpha} = \frac{\alpha}{\Delta t^2}$. Moreover, γ_j is the compliance and damping factor, defined as $\gamma = \frac{\widetilde{\alpha_j}\widetilde{\beta_j}}{\Delta t}$. Where the constant $\widetilde{\beta_j}$ is specifcally the time-scaled damping parameter $\Delta t^2\beta_j$. The use of rigid constraints decomposes the above equation nicely; when $K \to \infty$, $\alpha \to 0$. Using the limit $\alpha = 0$, many of the terms cancel out, resulting in

$$\Delta x = W\frac{-C_j(x)}{\nabla C_j(x)W\nabla C_j(x)^T}\nabla C_j(x)^T. \tag{49}$$

This result is identical to the previously derived equation, and all the damping effects disappear. The only method to introduce damping is by applying it onto each particle individually. This is typically completed with the inclusion of fluid drag.

4.3 Programmatic Implementation and Analysis

Since RGC can directly find the displacements, it is easy to implement programmatically and is 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:       Δx ← calculateDisplacement()
7:       // Apply displacement
8:       x ← x + Δx
9:    end for
10: end for
```

4.3.1 Inconsistent Stiffness

The goal of modelling perfectly rigid constraints assume no additional Δx scaling is required as Equation 46 is assumed to produce a perfect constraint relaxation. However in reality, a new stiffness scaling parameter, s, can be introduced to further control the constraint rigidity. Unfortunately, this stiffness is completely arbitrary and has very little physical significance. Including s, the improved equation then becomes,

$$\Delta x_i = -w_i s \frac{C(x)}{\sum_j^n w_j [\nabla_j C(x) \cdot \nabla_j C(x)]} \nabla_i C(x). \tag{50}$$

Constraints can behave surprisingly springy with $0 < s < 0.5$, and stiffer with $1 < s < 1.1$. It is important to note that the system has a high chance of diverging for $s > 1.5$. For $s > 1$, the system over-corrects and behaves stiffer, whereas for $s < 1$, the system under-corrects

and behaves more compliantly. Despite the theoretical rigidity for $s = 1$, the supposedly infinitely stiff constraint still exhibits some compliance.

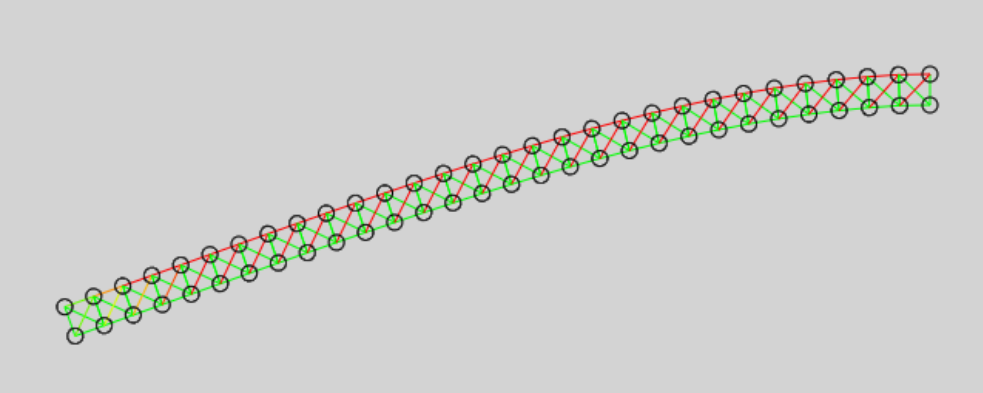


Figure 13: Compliance of a beam with $s = 1$

Similarly to PCs, it is discovered that the number of constraint solving iterations also significantly affects the system's stiffness. On the contrary, the reason for this behaviour is due to the repeated decrease of the truncation error $O(\Delta x^2)$ as Δx is lowered every iteration.

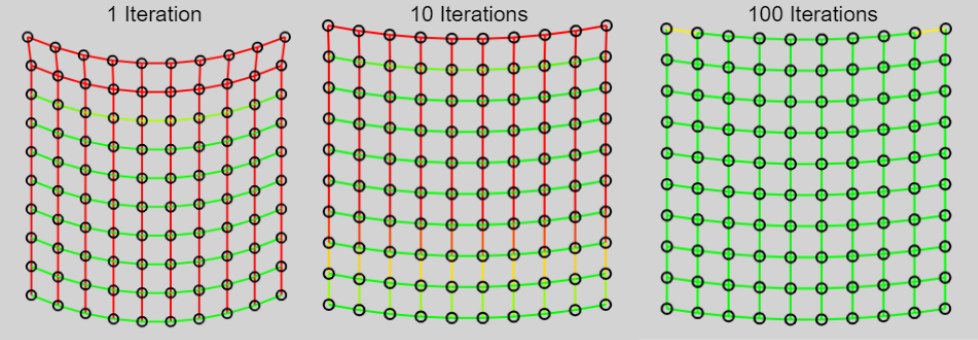


Figure 14: The stiffness increases in a cloth simulation as the number of constraint-solving iterations increases

This inconsistency can be resolved by scaling the stiffness with the number of iterations. The exact scaling of the stiffness will differ with the constraint equation itself. Moreover, the system also exhibits different stiffness responses with different timesteps.

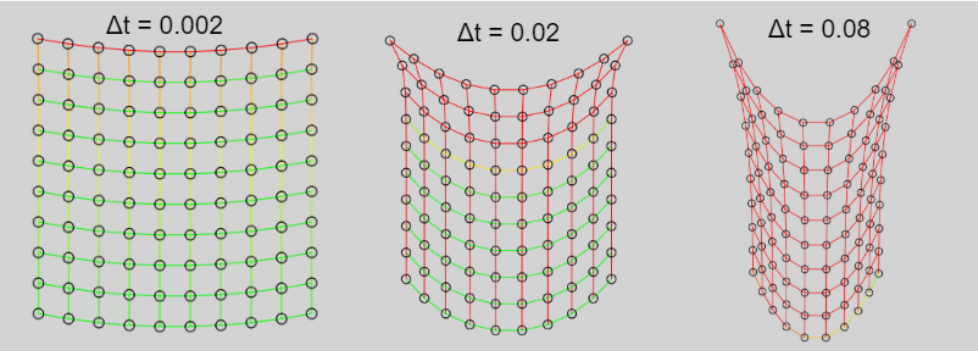


Figure 15: The stiffness decreases in a cloth simulation as the timestep increases

The system behaves less stiff as the timestep increases. This effect is caused by the larger displacement discretizations at larger timesteps. In this cloth example, the discretization of gravity increases with the timestep. In turn, resulting in larger corresponding Δx values to compensate. Once again, the larger Δx results in a greater truncation error, hence lowering the stiffness. Macklin's XPBD method extends upon the RGC method and removes the time reliance.

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.

4.3.2 Convergence and Energy Conservation

The conditions for the convergence and stability of RGCs coincide with the conditions for PCs. Comparatively, RGCs are easier to implement and are more stable because they are a position-based method. But also as a result, RGCs neglected energy conservation during its position based derivation. Moreover, the constraint error term grows quadratically with the constraint displacement, Δx, oftentimes resulting in inconsistent system energy. Similarly, increasing the timestep results in larger Δx discretizations, also contributing to the truncation error. It is also observed that the energy loss phenomenon is more apparent when there are more constraint solving iterations, η. Increasing η is equivalent to accelerating the time to $\eta \cdot \Delta t$. Therefore, the original error from 1 iteration will be stacked η times,

resulting in greater energy loss. A larger timestep also increases Δx and results in more energy instability. This phenomenon can be observed with a graphical analysis of an RGC pendulum system (Raw data available in Appendix A).

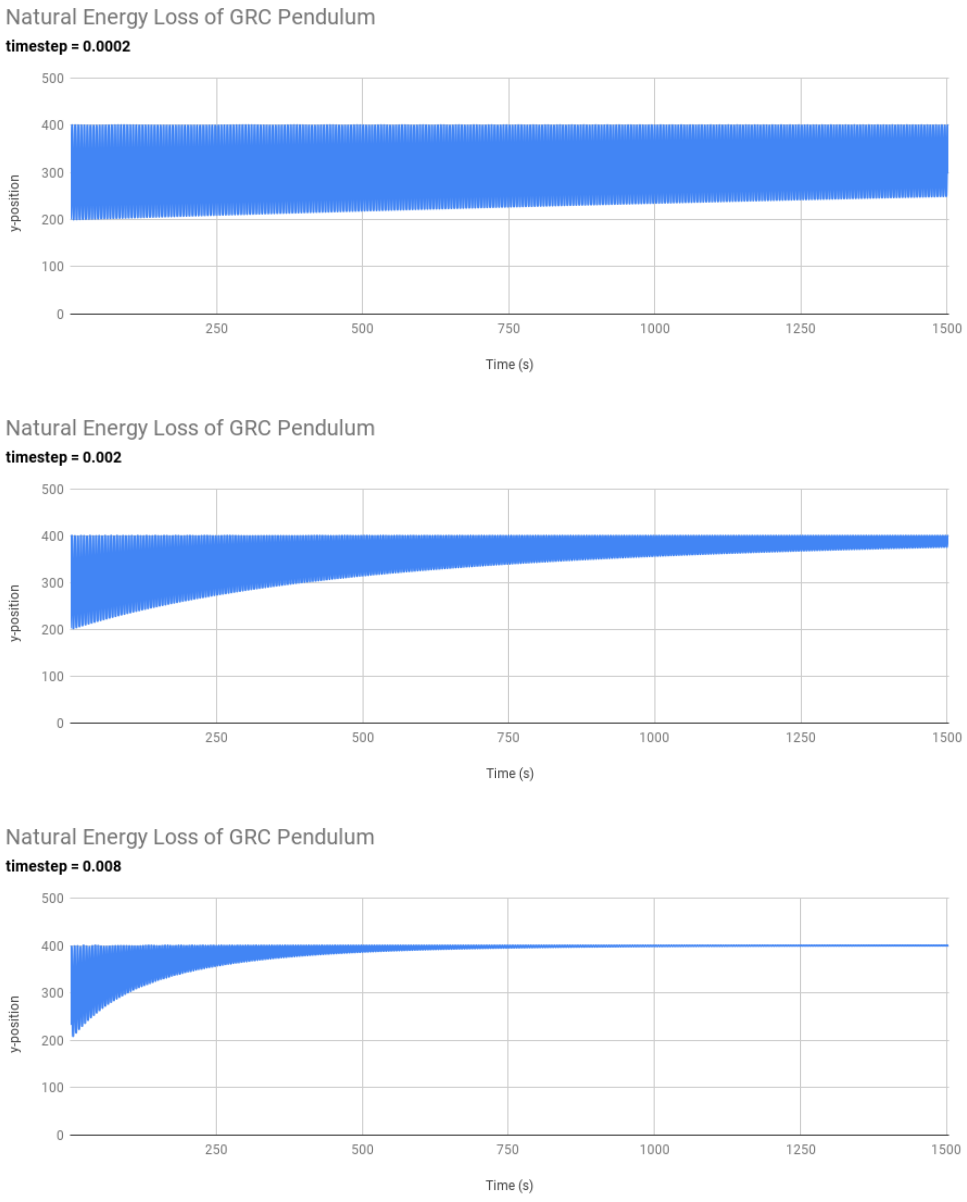


Figure 16: y-position vs time graphs of RGC pendulum systems at different Δt

System convergence also depends on the Gauss-Seidel solver; more iterations result in faster convergence. Conversely, divergence can occur when the truncation error causes overshoot. Overshoot commonly happens when an impulse is applied to the system (collisions and user

interactions). Stiffer systems with a larger s are also more likely to overshoot and diverge. Similar to PCs though, particle damping can also provide a buffer for divergent behaviour. Similarly, systems are also more likely to diverge with larger values of η; overshoot will also be magnified η times. Typically, the truncation error is more likely to causes undershoot, and therefore constraints tend to lose energy and converge with time. In general, RGCs are mainly effected by errors when there are rapid, sudden, or large movements.

5 Conclusion

Physical modelling has become much more prevalent, especially with the improvement of computing technology. Computational modelling has opened a new perspective of how humans perceive the world. This paper discussed rigid particle-based dynamic systems with penalty (Equation 24) and relaxed geometric constraints (Equation 50). Additionally, the system utilizes a predictor-corrector semi-implicit Euler integration algorithm with a local iterative solver (Algorithm 1). As such, the different aspects of each algorithm were found and analyzed. 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 easy to implement, versatile, and behaves more rigidly. 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 physical simulations. Whereas PCs are suitable for physically accurate compliant systems.

Only a small sample of the possible modelling techniques in the literature was discussed. With all the different integration algorithms, system solvers, and constraint algorithms, the resulting combinations are endless. A lot of the techniques described here are rapidly evolving as technology and understanding of the topic improves with time. In the modern world where physical modelling can encompass video game physics to highly technical engineering analysis, this 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, hence elevating computer modelling to the next stage.

6 Appendix A: Additional Resources

All demos 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`

7 Appendix B: Detailed Calculations

7.1 Detailed PC Stretch Constraint Derivation

7.1.1 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,

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

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

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

7.1.2 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

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

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

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,

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

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

7.2 Detailed RGC Stretch Constraint Derivation

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

$$\Delta x_i = -w_i \frac{C_j(x)}{\sum_k^n w_k[\nabla_{x_k} C_j(x) \cdot \nabla_{x_k} C_j(x)]} \nabla_{x_i} C_j(x).$$

Using for x_1 and x_2 give,

$$\Delta x_1 = -w_1 \frac{C_j(x)}{\sum_k^n w_k[\nabla_{x_k} C_j(x) \cdot \nabla_{x_k} C_j(x)]} \nabla_{x_1} C_j(x)$$

$$\Delta x_2 = -w_2 \frac{C_j(x)}{\sum_k^n w_k[\nabla_{x_k} C_j(x) \cdot \nabla_{x_k} C_j(x)]} \nabla_{x_2} C_j(x).$$

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

$$\nabla_{x_1} C_j(x) = 1 - 0 = 1$$

$$\nabla_{x_2} C_j(x) = 0 - 1 = -1.$$

Then, the denominator $\sum_{k}^{n} w_k[\nabla_{x_k} C_j(x) \cdot \nabla_{x_k} C_j(x)]$ is simplified to,

$$\sum_{k}^{n} w_k[\nabla_{x_k} C_j(x) \cdot \nabla_{x_k} C_j(x)] = w_1 \cdot 1 + w_2 \cdot 1 = w_1 + w_2.$$

Lastly, substituting each component yields,

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

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

8 Bibliography

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.

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.

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.

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.