Simulation in CG - Project 1

Thom Hurks & Jeffrey Aben

July 13, 2016

1 Introduction

In this document we will describe the work done, and decisions made regarding the particle simulation visualisation project. We will mainly focus on the mathematical formulae derived for the implementation, rather than the implementation itself, as this mainly follows the framework given in the lecture notes of Baraff.

We will start out by describing the forces present in our model in Section 2, followed by the constraints in Section 3. Afterwards we will discuss the setup we used to implement the particle system in Section 4, discuss the obtained results in Section 5 and conclude in Section 6.

1.1 Implementation

For our project we decided not to use the c++ skeleton code provided for the course, but we implemented the simulation in C#. We did use the skeleton code as a guideline and we followed the Baraff course notes, so our code should be familiar. To achieve cross-platform builds we use the Unity game engine, which can compile our C# project to Mac OS, Windows, WebGL and even iOS and Android. To make our code reusable it is not very much intertwined with Unity; we use Unity for "front-end" matters such as tracking user input events, drawing user interface components and the update loop. We do use Unity's Vector2 class, but that should be straightforward to replace by any other vector class. Our implementation works in a slightly different way depending on which platform it runs on. On a desktop platform the mouse can be used to pull on particles by using a spring after clicking on the particle; a right-click then removes the spring again. On touch-based platforms such as phones and tablets, we create springs when touching a particle and remove the spring when a touch stops. We track the touches as they change and remember their associated particles, and in this way we implement multi-touch interaction. Some forces, such as the angular springs, also visualize the forces they put on particles.

2 Forces

In this section we will address the five forces that we have included in our project. These are gravity force, viscous drag, normal (binary) spring force, the mouse spring force and the angular spring between three particles.

2.1 Gravity force

The most simple force of all. Being in a 2D setting, we chose to point this force in the negative y direction, i.e. downwards. A force $\boldsymbol{f} = \boldsymbol{g}m$ is applied to each particle, where $\boldsymbol{g} = \left(\begin{smallmatrix} 0 \\ -g \end{smallmatrix}\right)$. g is later chosen as an appropriate constant, that causes the particles to have a reasonable downwards acceleration.

2.2 Viscous drag

The viscous drag force basically hinders movement. If a particle $\boldsymbol{x}$ is moving in direction $\boldsymbol{v}$, then we apply a force $-k_d\boldsymbol{v}$ to $\boldsymbol{x}$, where k_d is the drag coefficient.

2.3 Spring force

The default spring force between two particles p_1 and p_2 is also rather standard, following Hook's law. We get the following force on p_1;

$$\boldsymbol{f}_1 = -\left(k_s * (\|\boldsymbol{l}\| - l) + k_d * \frac{\dot{\boldsymbol{l}} \cdot \boldsymbol{l}}{\|\boldsymbol{l}\|}\right) * \hat{\boldsymbol{l}} \tag{1}$$

where l denotes the rest length of the spring, k_d and k_s denote the spring constants, $\boldsymbol{l} = \boldsymbol{x}_1 - \boldsymbol{x}_2$, and $\hat{\boldsymbol{l}} = \frac{\boldsymbol{l}}{\|\boldsymbol{l}\|}$. Similarly, we get that $\boldsymbol{f}_2 = -\boldsymbol{f}_1$, which is to be expected, and get that this spring adds no energy to the system.

2.4 Mouse Spring force

Similar to the normal spring, except that we have just one particle p on location $\boldsymbol{x}$ and the mouse position $\boldsymbol{x}_m$, and a rest length of zero, we get the following force on p:

$$\boldsymbol{f} = -\left(k_s * \|\boldsymbol{l}\| + k_d * \frac{\dot{\boldsymbol{l}} \cdot \boldsymbol{l}}{\|\boldsymbol{l}\|}\right) * \hat{\boldsymbol{l}} \tag{2}$$

Where $\boldsymbol{l} = \boldsymbol{x} - \boldsymbol{x}_m$. This time however, as the mouse is not pulled in towards the particle, we have a force that adds energy to the system!

2.5 Angular springs

In this subsection we describe the decisions we made around the angular spring. We attempt to solve this problem in two different ways. We first describe an 'ad hoc' method, something that makes sense at an intuitive level. Secondly we solve the problem using energy functions.

2.5.1 Angular springs: Ad hoc

An angular spring connects three particles p_1, p_2, and p_3 together, with p_2 being the centre point point. let $\boldsymbol{r}_1 = \boldsymbol{x_1} - \boldsymbol{x_2}$ and $\boldsymbol{r}_3 = \boldsymbol{x}_3 - \boldsymbol{x}_2$ denote the positions of p_1 and p_3 relative to p_2, which we can also view as the segments that connect the particles together. The main point of interest lies in the angle θ between $\boldsymbol{r}_1$ and $\boldsymbol{r}_3$. We can find this angle by $\theta = \cos^{-1}\left(\frac{\boldsymbol{r}_1 \cdot \boldsymbol{r}_3}{|\boldsymbol{r}_1||\boldsymbol{r}_3|}\right)$. We want to apply forces to p_i such that the system moves towards a state where $\theta = \theta_0$. Using the difference in angle $\Delta\theta = \theta - \theta_o$ we can work out (through simple rotations) where $\boldsymbol{x}_1$ and $\boldsymbol{x}_3$ would have been if $\theta = \theta_0$. Let these two locations be $\boldsymbol{g}_1$ and $\boldsymbol{g}_3$. These rotations are done in a symmetrical fashion, not taking mass into account. See Figure 1 for a clarification.

The core idea is to now apply spring forces between the points $\boldsymbol{x}_i$ and $\boldsymbol{g}_i$ Now, let $\boldsymbol{f}_i$ be the force applied to $\boldsymbol{x}_i$ along the direction of $\boldsymbol{x}_i - \boldsymbol{g}_i$. The last thing we add to this is a force $\boldsymbol{f}_2$ on p_2 in order to make sure that the net-force of this spring is 0. To this end we let $\boldsymbol{f}_2 = -(\boldsymbol{f}_1 + \boldsymbol{f}_3)$. Finally, as we are dealing with springs towards fixed positions, we can use the forces for the mouse spring here. Thus, we finally get;

$$\boldsymbol{f}_i = -\left(k_s * \|\boldsymbol{l}_i\| + k_d * \frac{\dot{\boldsymbol{l}}_i \cdot \boldsymbol{l}_i}{\|\boldsymbol{l}_i\|}\right) * \hat{\boldsymbol{l}}_i \qquad \text{for } i = 1, 3 \tag{3}$$

where $\boldsymbol{l}_i = \boldsymbol{x}_i - \boldsymbol{g}_i$.

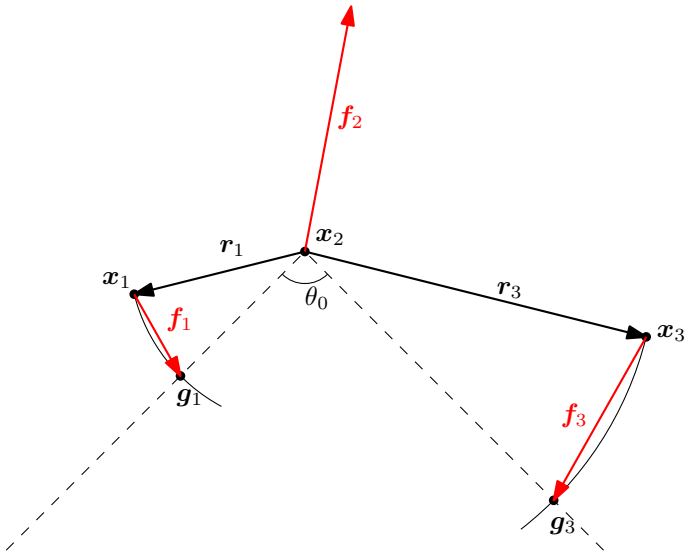

Figure 1: The important aspects of the ad-hoc angle spring.

2.5.2 Angular springs: Energy function

As above we will we working with $\boldsymbol{r}_1 = \boldsymbol{x}_1 - \boldsymbol{x}_2$ and $\boldsymbol{r}_3 = \boldsymbol{x}_3 - \boldsymbol{x}_2$.
Given a rest angle θ_0 we use the following (scalar) energy function for the angular spring:

$$C = \cos^{-1}\left(\frac{\boldsymbol{r}_1 \cdot \boldsymbol{r}_3}{|\boldsymbol{r}_1||\boldsymbol{r}_3|}\right) - \theta_0$$

Using the substitution $y = \frac{\boldsymbol{r}_1 \cdot \boldsymbol{r}_3}{|\boldsymbol{r}_1||\boldsymbol{r}_3|}$ we can write

$$\dot{C} = \frac{\delta C}{\delta y}\dot{y} \quad \frac{\delta C}{\delta y} = -\frac{1}{\sqrt{1-y^2}} \quad \dot{y} = \frac{|\boldsymbol{r}_1||\boldsymbol{r}_3| * (\boldsymbol{r}_1 \cdot \dot{\boldsymbol{r}}_3 + \dot{\boldsymbol{r}}_1 \cdot \boldsymbol{r}_3) - \boldsymbol{r}_1 \cdot \boldsymbol{r}_3 * \left(\frac{\boldsymbol{r}_1 \cdot \dot{\boldsymbol{r}}_1}{|\boldsymbol{r}_1|}|\boldsymbol{r}_3| + \frac{\boldsymbol{r}_3 \cdot \dot{\boldsymbol{r}}_3}{|\boldsymbol{r}_3|}|\boldsymbol{r}_1|\right)}{(|\boldsymbol{r}_1||\boldsymbol{r}_3|)^2}$$

This yields the rather unattractive formula for $\dot{C}$:

$$-\frac{|\boldsymbol{r}_1||\boldsymbol{r}_3| * (\boldsymbol{r}_1 \cdot \dot{\boldsymbol{r}}_3 + \dot{\boldsymbol{r}}_1 \cdot \boldsymbol{r}_3) - \boldsymbol{r}_1 \cdot \boldsymbol{r}_3 * \left(\frac{\boldsymbol{r}_1 \cdot \dot{\boldsymbol{r}}_1}{|\boldsymbol{r}_1|}|\boldsymbol{r}_3| + \frac{\boldsymbol{r}_3 \cdot \dot{\boldsymbol{r}}_3}{|\boldsymbol{r}_3|}|\boldsymbol{r}_1|\right)}{(|\boldsymbol{r}_1||\boldsymbol{r}_3|)^2\sqrt{1 - \left(\frac{\boldsymbol{r}_1 \cdot \boldsymbol{r}_3}{|\boldsymbol{r}_1||\boldsymbol{r}_3|}\right)^2}}$$

Next we need to differentiate C to $\boldsymbol{r}_1$ and $\boldsymbol{r}_3$.

$$\frac{\delta C}{\delta \boldsymbol{r}_1} = \frac{\delta C}{\delta y}\frac{\delta y}{\delta \boldsymbol{r}_1} \qquad \frac{\delta y}{\delta \boldsymbol{r}_1} = \frac{|\boldsymbol{r}_1||\boldsymbol{r}_3|\boldsymbol{r}_3 - \frac{|\boldsymbol{r}_3|}{|\boldsymbol{r}_1|}\boldsymbol{r}_1}{(|\boldsymbol{r}_1||\boldsymbol{r}_3|)^2}$$

A similar derivation holds for $\frac{\delta C}{\delta \boldsymbol{r}_3}$, which then yield the formulae:

$$\frac{\delta C}{\delta \boldsymbol{r}_1} = -\frac{|\boldsymbol{r}_1||\boldsymbol{r}_3|\boldsymbol{r}_3 - \frac{|\boldsymbol{r}_3|}{|\boldsymbol{r}_1|}\boldsymbol{r}_1}{|\boldsymbol{r}_1|^2|\boldsymbol{r}_3|^2\sqrt{1 - \left(\frac{\boldsymbol{r}_1 \cdot \boldsymbol{r}_3}{|\boldsymbol{r}_1||\boldsymbol{r}_3|}\right)^2}} \qquad \frac{\delta C}{\delta \boldsymbol{r}_3} = -\frac{|\boldsymbol{r}_1||\boldsymbol{r}_3|\boldsymbol{r}_1 - \frac{|\boldsymbol{r}_1|}{|\boldsymbol{r}_3|}\boldsymbol{r}_3}{|\boldsymbol{r}_1|^2|\boldsymbol{r}_3|^2\sqrt{1 - \left(\frac{\boldsymbol{r}_1 \cdot \boldsymbol{r}_3}{|\boldsymbol{r}_1||\boldsymbol{r}_3|}\right)^2}}$$

$$= -\frac{|\boldsymbol{r}_1||\boldsymbol{r}_3|\boldsymbol{r}_3 - \frac{|\boldsymbol{r}_3|}{|\boldsymbol{r}_1|}\boldsymbol{r}_1}{|\boldsymbol{r}_1||\boldsymbol{r}_3|\sqrt{|\boldsymbol{r}_1|^2|\boldsymbol{r}_3|^2 - (\boldsymbol{r}_1 \cdot \boldsymbol{r}_3)^2}} \qquad = -\frac{|\boldsymbol{r}_1||\boldsymbol{r}_3|\boldsymbol{r}_1 - \frac{|\boldsymbol{r}_1|}{|\boldsymbol{r}_3|}\boldsymbol{r}_3}{|\boldsymbol{r}_1||\boldsymbol{r}_3|\sqrt{|\boldsymbol{r}_1|^2|\boldsymbol{r}_3|^2 - (\boldsymbol{r}_1 \cdot \boldsymbol{r}_3)^2}}$$

Finally we need to use the chain rule to obtain $\frac{\delta C}{\delta \boldsymbol{x}_i}$. This is straightforward for $i = 1$ and $i = 3$, since $\boldsymbol{x}_1$ and $\boldsymbol{x}_3$ only occur in $\boldsymbol{r}_1$ and $\boldsymbol{r}_3$ respectively:

$$\frac{\delta C}{\delta \boldsymbol{x}_1} = -\frac{|\boldsymbol{r}_1||\boldsymbol{r}_3|\boldsymbol{r}_3 - \frac{|\boldsymbol{r}_3|}{|\boldsymbol{r}_1|}\boldsymbol{r}_1}{|\boldsymbol{r}_1||\boldsymbol{r}_3|\sqrt{|\boldsymbol{r}_1|^2|\boldsymbol{r}_3|^2 - (\boldsymbol{r}_1 \cdot \boldsymbol{r}_3)^2}} \qquad = -\frac{|\boldsymbol{r}_1||\boldsymbol{r}_3|\boldsymbol{r}_1 - \frac{|\boldsymbol{r}_1|}{|\boldsymbol{r}_3|}\boldsymbol{r}_3}{|\boldsymbol{r}_1||\boldsymbol{r}_3|\sqrt{|\boldsymbol{r}_1|^2|\boldsymbol{r}_3|^2 - (\boldsymbol{r}_1 \cdot \boldsymbol{r}_3)^2}}$$

For $\boldsymbol{x}_2$ it is another story however, since it occurs in both $\boldsymbol{r}_1$ and $\boldsymbol{r}_3$. We get the following:

$$\frac{\delta C}{\delta \boldsymbol{x}_2} = \frac{\delta C}{\delta \boldsymbol{r}_1}\frac{\delta \boldsymbol{r}_1}{\delta \boldsymbol{x}_2} + \frac{\delta C}{\delta \boldsymbol{r}_3}\frac{\delta \boldsymbol{r}_3}{\delta \boldsymbol{x}_2} = -\frac{\delta C}{\delta \boldsymbol{x}_1} - \frac{\delta C}{\delta \boldsymbol{x}_3}$$

With this we have all the required ingredients to plug in the default spring force formula:

$$\boldsymbol{f}_i = (-k_s C - k_d \dot{C})\frac{\delta C}{\delta \boldsymbol{x}_i}$$

Notice that this force again does not add energy to the system. Also note that when $\boldsymbol{r}_1$ and $\boldsymbol{r}_3$ are parallel, i.e. we are have an angle of π or zero, this system breaks down due to dividing by zero (the term in the root). In this case we abort the force application and wait for something to disrupt this 'stalemate'.

3 Constraints

In this section we will describe the various constraints we impose on some particles.
We will derive and implement multiple versions of the constraints. These versions will be derived from different, equally valid, energy functions. These in turn will yield different derivatives. While mathematically they are all equally valid, we implement them all and evaluate the stability of each version.

3.1 Rod constraint

Here we present the mathematics behind the rod constraint: the constraint that forces two particles to remain at a fixed distance from each other. Our two particles are $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$. To make the formulation of the constraint easier, we introduce the vector $\boldsymbol{l} = \boldsymbol{x}_1 - \boldsymbol{x}_2$. The two behaviour functions we explore then become:

$$C_1 = \frac{|\boldsymbol{l}|^2 - d^2}{2} \qquad C_2 = |\boldsymbol{l}| - d$$

Which we can easily differentiate to time using the product & chain rules.

$$\dot{C}_1 = \dot{\boldsymbol{l}} \cdot \boldsymbol{l} \qquad \dot{C}_2(\boldsymbol{l}) = \frac{\boldsymbol{l} \cdot \dot{\boldsymbol{l}}}{|\boldsymbol{l}|}$$

Now we differentiate C_i and $\dot{C}_i$ with respect to $\boldsymbol{l}$:

$$\frac{\delta C_1}{\delta \boldsymbol{l}} = \boldsymbol{l} \qquad \frac{\delta C_2}{\delta \boldsymbol{l}} = \frac{\boldsymbol{l}}{|\boldsymbol{l}|}$$

$$\frac{\delta \dot{C}_1}{\delta \boldsymbol{l}} = \dot{\boldsymbol{l}} \qquad \frac{\delta \dot{C}_2}{\delta \boldsymbol{l}} = \frac{|\boldsymbol{l}|\dot{\boldsymbol{l}} - (\boldsymbol{l} \cdot \dot{\boldsymbol{l}})\frac{\boldsymbol{l}}{|\boldsymbol{l}|}}{|\boldsymbol{l}|^2}$$

The corresponding formulae for $\boldsymbol{x}_i$ can be extracted straightforwardly with the chain rule.

These values can be found in two 1×2 blocks in J and $\dot{J}$ respectively. In order to make sure we did not take a wrong turn somewhere, we can verify our results by checking if $\boldsymbol{J}\dot{\boldsymbol{q}} = \dot{C}$ holds.

$$\boldsymbol{J}_1\dot{\boldsymbol{q}} = \begin{pmatrix} x_1 - x_2 \\ y_1 - y_2 \\ x_2 - x_1 \\ y_2 - y_1 \end{pmatrix}^{\mathrm{T}} \begin{pmatrix} \dot{x}_1 \\ \dot{y}_1 \\ \dot{x}_2 \\ \dot{y}_2 \end{pmatrix} = (x_1 - x_2)\dot{x}_1 + (y_1 - y_2)\dot{y}_1 + (x_2 - x_1)\dot{x}_2 + (y_2 - y_1)\dot{y}_2 = \dot{C}_1$$

$$\boldsymbol{J}_2\dot{\boldsymbol{q}} = \begin{pmatrix} (x_1 - x_2)/|x_1 - x_2| \\ (y_1 - y_2)/|y_1 - y_2| \\ (x_2 - x_1)/|x_1 - x_2| \\ (y_2 - y_1)/|y_1 - y_2| \end{pmatrix}^{\mathrm{T}} \begin{pmatrix} \dot{x}_1 \\ \dot{y}_1 \\ \dot{x}_2 \\ \dot{y}_2 \end{pmatrix} = \frac{(x_1 - x_2)\dot{x}_1}{|x_1 - x_2|} + \frac{(y_1 - y_2)\dot{y}_1}{|y_1 - y_2|} - \frac{(x_1 - x_2)\dot{x}_2}{|x_1 - x_2|} - \frac{(y_1 - y_2)\dot{y}_2}{|y_1 - y_2|} = \dot{C}_2$$

3.2 Circular wire constraint

The next constraint we consider is the circular wire constraint. Like above, it forces a particle to be a fixed distance from something else. However, contrary to the rod constraint, this time our particle needs to be a fixed distance r from some point $\boldsymbol{c}$, the centre. The big difference is that $\boldsymbol{c}$ is a static unmovable point. To keep the calculations clean, we employ the same strategy as above, and introduce the vector $\boldsymbol{l} = \boldsymbol{x} - \boldsymbol{c}$. Again, we have two variants.

$$C_1 = \left(\boldsymbol{l} \cdot \boldsymbol{l} - r^2\right)/2 \qquad\qquad C_2 = |\boldsymbol{l}| - r$$

Differentiating to time yields the same results as with the rod constraint:

$$\dot{C}_1 = \dot{\boldsymbol{l}} \cdot \boldsymbol{l} \qquad\qquad \dot{C}_2 = \frac{\boldsymbol{l} \cdot \dot{\boldsymbol{l}}}{|\boldsymbol{l}|}$$

As above, now we differentiate C_i and $\dot{C}_i$ to $\boldsymbol{l}$. Since the formulae are the same, we get the same formulae.

$$\frac{\delta C_1}{\delta \boldsymbol{l}} = \boldsymbol{l} \qquad\qquad \frac{\delta C_2}{\delta \boldsymbol{l}} = \frac{\boldsymbol{l}}{|\boldsymbol{l}|}$$

$$\frac{\delta \dot{C}_1}{\delta \boldsymbol{l}} = \dot{\boldsymbol{l}} \qquad\qquad \frac{\delta \dot{C}_2}{\delta \boldsymbol{l}} = \frac{|\boldsymbol{l}|\dot{\boldsymbol{l}} - (\boldsymbol{l} \cdot \dot{\boldsymbol{l}})\frac{\boldsymbol{l}}{|\boldsymbol{l}|}}{|\boldsymbol{l}|^2}$$

Which takes up one block of 1×2 in J and $\dot{J}$ respectively. Since we know that $\boldsymbol{J}\dot{\boldsymbol{q}}$ should equal $\dot{C}$, we can do a verification.

$$\boldsymbol{J}_1\dot{\boldsymbol{q}} = \begin{pmatrix} x - x_c \\ y - y_c \end{pmatrix}^{\mathrm{T}} \begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix} = \dot{x} * (x - x_c) + \dot{y} * (y - y_c) = \dot{C}_1$$

$$\boldsymbol{J}_2\dot{\boldsymbol{q}} = \begin{pmatrix} (x - x_c)/|x - x_c| \\ (y - y_c)/|y - y_c| \end{pmatrix}^{\mathrm{T}} \begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix} = \dot{x} * \frac{(x - x_c)}{|x - x_c|} + \dot{y} * \frac{(y - y_c)}{|y - y_c|} = \dot{C}_2$$

3.2.1 Elliptical wire constraint

The mathematics behind a constraint forcing a particle onto an elliptical wire is very similar to that of a circular wire. This time we have two centre points $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$ and define $\boldsymbol{l_1} = \boldsymbol{x} - \boldsymbol{c}_1$ and $\boldsymbol{l_2} = \boldsymbol{x} - \boldsymbol{c}_2$. We only present the formulae involved, without derivation, considering only one variant.

$$
\begin{aligned}
C_2 &= |\boldsymbol{l}_1| + |\boldsymbol{l}_2| - r \\
\dot{C}_2 &= \frac{\boldsymbol{l}_1 \cdot \dot{\boldsymbol{l}}_1}{|\boldsymbol{l}_1|} + \frac{\boldsymbol{l}_2 \cdot \dot{\boldsymbol{l}}_2}{|\boldsymbol{l}_2|} \\
\frac{\delta C_2}{\delta \boldsymbol{x}} &= \frac{\boldsymbol{l}_1}{|\boldsymbol{l}_1|} + \frac{\boldsymbol{l}_2}{|\boldsymbol{l}_2|} \\
\frac{\delta \dot{C}_2}{\delta \boldsymbol{x}} &= \frac{|\boldsymbol{l}_1|\dot{\boldsymbol{l}}_1 - (\boldsymbol{l}_1 \cdot \dot{\boldsymbol{l}}_1)\frac{\boldsymbol{l}_1}{|\boldsymbol{l}_1|}}{|\boldsymbol{l}_1|^2} + \frac{|\boldsymbol{l}_2|\dot{\boldsymbol{l}}_2 - (\boldsymbol{l}_2 \cdot \dot{\boldsymbol{l}}_2)\frac{\boldsymbol{l}_2}{|\boldsymbol{l}_2|}}{|\boldsymbol{l}_2|^2}
\end{aligned}
$$

Note how these formulae straightforwardly generalize to an n-elliptical wire constraint. We did however not implement this generalized version.

3.3 Fixed point constraint

The fixed point constraint can again be modelled a fixed point $\boldsymbol{c}$ at which $\boldsymbol{x}$ should stay. If we were to model this in the same way as above, we would create a weird scenario. Namely, the legal velocities are ill-defined at the legal position. To alleviate this, we use a vector valued energy function instead, and only have one version of our constraint:

$$\boldsymbol{C} = \boldsymbol{x} - \boldsymbol{c}$$

The derivatives involved are rather straightforward. In the results $\boldsymbol{I}$ stands for the identity matrix, and $\boldsymbol{0}$ is the 0 matrix.

$$\dot{\boldsymbol{C}} = \dot{\boldsymbol{x}} \qquad \frac{\delta \boldsymbol{C}}{\delta \boldsymbol{x}} = \boldsymbol{I} \qquad \frac{\delta \dot{\boldsymbol{C}}}{\delta \boldsymbol{x}} = \boldsymbol{0}$$

The validity check also works out:

$$\boldsymbol{J}\dot{\boldsymbol{q}} = \boldsymbol{I}\dot{\boldsymbol{x}} = \dot{\boldsymbol{x}} = \dot{\boldsymbol{C}}$$

3.4 Line constraint

Here we describe the functions used to implement horizontal and vertical line constraints. That is, constraints such that a particle's x or y coordinate is fixed. We can model these as scalar constraints. We denote the horizontal line constraint by C_{v} (it is constrained vertically) and the vertical line constraint by C_{h}. In these formulae $\boldsymbol{0}$ denotes the zero vector.

$$
\begin{aligned}
C_{\mathrm{v}}(\boldsymbol{x}) &= y_{\boldsymbol{x}} - y_c & \dot{C}_{\mathrm{v}} &= \dot{y}_{\boldsymbol{x}} & \frac{\delta C_{\mathrm{v}}}{\delta \boldsymbol{x}} &= \begin{pmatrix} 0 \\ 1 \end{pmatrix} & \frac{\delta \dot{C}_{\mathrm{v}}}{\delta \boldsymbol{x}} &= \boldsymbol{0} \\
C_{\mathrm{h}}(\boldsymbol{x}) &= x_{\boldsymbol{x}} - x_c & \dot{C}_{\mathrm{h}} &= \dot{x}_{\boldsymbol{x}} & \frac{\delta C_{\mathrm{h}}}{\delta \boldsymbol{x}} &= \begin{pmatrix} 1 \\ 0 \end{pmatrix} & \frac{\delta \dot{C}_{\mathrm{h}}}{\delta \boldsymbol{x}} &= \boldsymbol{0}
\end{aligned}
$$

We can also do this for an arbitrary line $\boldsymbol{l} = \boldsymbol{a}t + \boldsymbol{b}$, where $\boldsymbol{a}$ is a unit vector. Let $\boldsymbol{a} = \left(\begin{smallmatrix} a_x \\ a_y \end{smallmatrix}\right)$ and let $\boldsymbol{a}_\perp = \left(\begin{smallmatrix} a_y \\ -a_x \end{smallmatrix}\right)$ be a vector perpendicular to $\boldsymbol{a}$.

$$C_{\boldsymbol{l}} = (\boldsymbol{x} - \boldsymbol{b}) \cdot \boldsymbol{a}_\perp \qquad \dot{C}_{\boldsymbol{l}} = \dot{\boldsymbol{x}} \cdot \boldsymbol{a}_\perp \qquad \frac{C_{\boldsymbol{l}}}{\delta x} = \boldsymbol{a}_\perp \qquad \frac{\dot{C}_{\boldsymbol{l}}}{\delta x} = \boldsymbol{0}$$

4 Computation Environment

4.1 Implementation

Our implementation of the above forces & constraints is done in C#, and we use the Unity game engine as the environment to run our code. We switched from C++, used in the skeleton code provided to us, to C# because both authors have more experience in the language. Also, C# is a managed language with a garbage collector, which means we do not have to worry about memory leaks and many other memory safety issues during implementation. Also, because we use Unity we can run our code on many different platforms such as Windows, Mac OS, browsers (WebGL) and mobile devices.

4.2 Integrators

When implementing the framework described in the course notes of Baraff, we implemented six integration methods. The methods, in increasing order of performance are the Euler, Midpoint, Leapfrog, (Velocity) Verlet, Runga Kutta 4 methods along with one of our own integration schemes to which we will refer as Midpoint Verlet which we explain in detail below.

Notice that the assumption for the original Velocity Verlet & Leapfrog integrators, that acceleration is not dependant on velocity, do not hold in our simulation.

4.2.1 Midpoint Verlet

The idea behind Midpoint Verlet is pretty straightforward. We use subscribts to denote timesteps. In the normal Verlet the first step is to find out what $\boldsymbol{v}_{i+\frac{1}{2}}$ is. To this end, in the normal Verlet, we take half an Euler step for the velocity, that is $\boldsymbol{v}_{i+\frac{1}{2}} = \boldsymbol{v}_i + \boldsymbol{a}_i \frac{\Delta t}{2}$. The core idea of Midpoint Verlet is: "Why not replace this half Euler step, with a half Midpoint step?".
That makes the algorithm for finding $\boldsymbol{x}_{i+1}$ and $\boldsymbol{v}_{i+1}$ as follows:

1. Obtain $\boldsymbol{a}_i$ from the particle system.
2. Calculate $\boldsymbol{x}_{i+\frac{1}{4}} = \boldsymbol{x}_i + \boldsymbol{v}_i \frac{\Delta t}{4}$ and $\boldsymbol{x}_{i+\frac{1}{4}} = \boldsymbol{v}_i + \boldsymbol{a}_i \frac{\Delta t}{4}$, i.e. a $\frac{1}{4}$ Euler step.
3. Obtain $\boldsymbol{a}_{i+\frac{1}{4}}$ from the particle system.
4. Calculate $\boldsymbol{v}_{i+\frac{1}{2}} = \boldsymbol{v}_i + \boldsymbol{a}_{i+\frac{1}{4}} \frac{\Delta t}{2}$ i.e. half a midpoint step for the velocity.
5. Calculate $\boldsymbol{x}_{i+1} = \boldsymbol{x}_i + \boldsymbol{v}_{i+\frac{1}{2}} \Delta t$, i.e. the Verlet step for position.
6. Obtain $\boldsymbol{a}_{i+1}$ from the particle system, which is now in the state $\boldsymbol{x}_{i+1}, \boldsymbol{v}_{i+\frac{1}{2}}$.
7. Calculate $\boldsymbol{v}_{i+1} = \boldsymbol{v}_{i+\frac{1}{2}} + \boldsymbol{a}_{i+1} \frac{\Delta t}{2}$, i.e. the Verlet step for velocity.

This means we get the following update rule for $\boldsymbol{x}_{i+1}$ and $\boldsymbol{v}_{i+1}$:[1]

$$\begin{aligned}
\boldsymbol{a}_{i+\frac{1}{4}} &= \boldsymbol{f}\left(\boldsymbol{x}_i + \boldsymbol{v}_i \frac{\Delta t}{4}, \boldsymbol{v}_i + \boldsymbol{f}\left(\boldsymbol{x}_i, \boldsymbol{v}_i\right) \frac{\Delta t}{4}\right) \\
\boldsymbol{x}_{i+1} &= \boldsymbol{x}_i + \boldsymbol{v}_i \Delta t + \boldsymbol{a}_{i+\frac{1}{4}} \frac{\Delta t^2}{2} \\
\boldsymbol{a}_{i+1} &= \boldsymbol{f}\left(\boldsymbol{x}_{i+1}, \boldsymbol{v}_i + \boldsymbol{a}_{i+\frac{1}{4}} \frac{\Delta t}{2}\right) \\
\boldsymbol{v}_{i+1} &= \boldsymbol{v}_i + (\boldsymbol{a}_{i+\frac{1}{4}} + \boldsymbol{a}_{i+1}) \frac{\Delta t}{2}
\end{aligned}$$

[1] Note that in the formulae for $\boldsymbol{a}_{i+\frac{1}{4}}$ and $\boldsymbol{a}_{i+1}$ we do not take particle mass into account, as this might get confusing with the current indexing of variables.

Notice that we start with evaluating $\boldsymbol{a}_i$, which requires solving the ODE's of the system. If the assumption that $\boldsymbol{a}_i$ is independent of $\boldsymbol{v}_i$ would hold, we could reuse the $\boldsymbol{a}_{i+1}$ values from the previous timestep for this. Also note that this does not completely remove this assumption since we still need it in step 6. There is however little we can do about this. We also employ this extra inspection of $\boldsymbol{a}$ in the implementation of the normal Verlet integration method.

4.3 User Interface

All the constraints and forces that have multiple versions are switchable in the UI. This eases the comparison of stability and performance. We also have selectors for different testing scenarios, a slider to adjust the simulation speed and a button to reset the simulation. User interaction is done through the mouse: Left clicking a particle will grab it, attaching a mouse spring to that particle, whilst right clicking releases the current particle. On touch-based devices we use a multi-touch based system for interacting with the particles.

5 Results

In this section we go over the results. We mainly discuss stability and performance of our various constraints and forces, and all their different versions. We omit trivial results about gravity, drag, mouses springs & binary springs.

5.1 Angular springs

We start with the angular springs. There is one striking difference between the two versions: The ad hoc version has a single accepted angle: θ_0, whereas the analytical version has two accepted angles: θ_0 and $-\theta_0$.
The next difference between the two is the angles at which they behave properly. In the analytical version we get a division by zero when the particles form an angle of 0 or π. We alleviate this by introducing an ε in the square root, changing it from $\sqrt{1-y^2}$ to $\sqrt{1.05-y^2}$. The ad-hoc version does not suffer from this problem. The ad hoc version recovers very slowly when transiting from $-\theta_0$ to θ_0, which appears unnatural, since at this point its 'potential energy' should be at its highest.

Both versions have their issues. However, when combining multiple angular springs to form, for instance, a hair-like object, we suggest to use the ad-hoc version. The analytic versions will try to balance each other out, resulting in frequent angles near π, which in turn causes a very unstable simulation. Furthermore, $\dot{C}$ tends to explode quickly in the analytical version, making damping the spring a bad choice. We can however introduce a heavy viscous drag to the system, which gives us a lot of the desired stability.

It should be noted that we can also use a different energy function C based on the 'atan2' function, which should not get the divide by zero issues. We did not have time to work out and implement this version however.

5.2 Constraints

Moving on to the circular wire constraint; we have two versions, one using quadratic distances and the other using linear distances. Both versions perform well under 'normal circumstances'. However, when under a lot of forces, and heavy oscillation, it becomes evident that the linear version is more stable. This comes with the natural drawback that the formulae involved are more elaborate, and hence require more computational power per frame. This was however not a problem for the scale at which we used these constraints.

With the rod constraint we have similar results as with the circular wire constraint. The linear version appears more stable under extreme circumstances. Note however, that the tests to establish these claims were by no means controlled in a deterministic manner. It is no trivial task to create predefined scenarios where a quadratic rod breaks down while its linear counterpart does hold up. The same applies to the circular wire constraint.

The other constraints did not have multiple versions, and performed well overall.

5.3 Integrators

As stated in Section 4, we implemented six integration methods. We evaluate the performance of each integration method based on a predefined scenario, namely the 'pendulum' scenario. In the pendulum scenario there are several chains, connected through rod constraints, of various lengths attached to fixed points. We have pendulums of lengths one to ten. This scenario is the most likely to explode and is thus very suited for stability tests. We check what is the largest time step that allows the system to reach a stable state, i.e. all pendulums having a small, stable swing, within a reasonable ammount of time. We then normalize these timesteps to the timestep of the weakest integrator: Euler. Thus, Euler will get a score of 1, and having a score of x means that you can tolerate a timestep x times greater than the timestep required by Euler. The scores can be found in the table below.

Integration Method	**Score**
Euler	1
Midpoint	33
Leapfrog	120
Runga Kutta 4	205
Verlet	300
Midpoint Verlet	320

A remarkable result is that while in theory the midpoint, leapfrog and Verlet integration methods should be (more or less) equal, the difference between them is very significant. Even more remarkable is that Verlet outperforms Runga Kutta 4, while only solving the ODE's twice, whereas Runga Kutta 4 solves them four times.

To make sure that this test is not specifically biased towards Verlet integration methods, we also do another experiment, with a single particle, on a single constraint. We use an elliptical constraint with dimension 4×2, and attach a particle. For each integration method we will find the greatest initial speed we can give the particle such that it will remain on the wire. All these tests are done with the normal time step. Lastly we add a tiny amount of viscous drag to the system, to counteract the accumulation of extra speed due to numerical errors. This drag force was picked differently for each integration method, in order to keep the particle at a constant speed.

Integration Method	**Score**
Euler	2.5
Midpoint	15
Leapfrog	15
Runga Kutta 4	50
Verlet	20
Midpoint Verlet	40

These results are much more like the theoretically expected ratios between the solvers. Another noticeable result is that the Euler, Leapfrog & Verlet methods have the tendency to add energy to the system. This means that in some cases the system can't come to rest. The Midpoint,

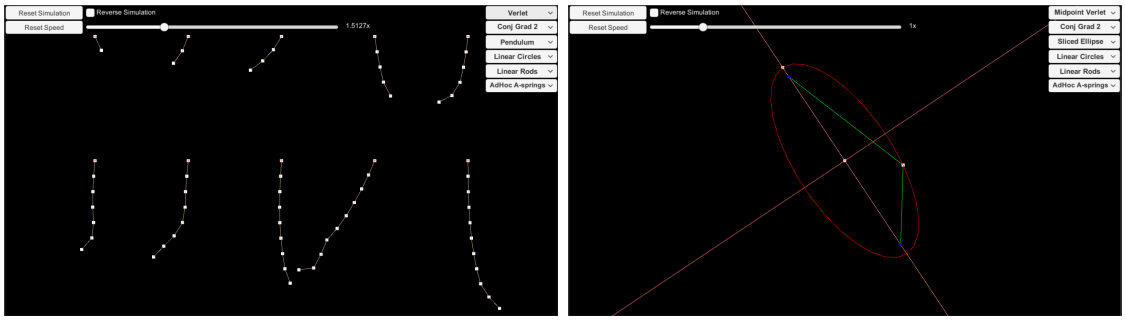

Figure 2: Two snapshots of the simulation.

Runga Kutta 4 and Midpoint Verlet methods don't seem to have this problem, or at least, it is significantly less noticeable. The only explanation for the difference between the Verlet & Leapfrog methods is that we did not add the second evaluation of $\boldsymbol{f}$ in the Leapfrog implementation, which is evident in the results.

5.4 Solvers

During the project we encountered some hard to explain bugs involving the constraints. After some debugging we discovered that something was wrong with the provided Conjugate Gradient solver. To make sure our version, that is translated to C#, or our constraints were not to blame we proceeded by implementing a simple Jacobi solver, which resolved the issue. The Jacobi solver is a lot slower than the Conjugate Gradient solver, however, so we implemented a different Conjugate Gradient variant, denoted "Conj Grad 2" in our simulator's UI, written by John Burkardt. Unfortunately, the author of this variant does not mention any mathematical specifics about the algorithm, but mentions that more information is available in the book "Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods". This second Conjugate Gradient variant does produce the expected results and is faster than our original solver. For all our results we capped the maximum number of iterations by 1000, and the convergence condition was set to $\varepsilon = 10^{-3}$.

With both of the conjugate gradient solvers we also use the solution from the previous timestep as the initial "guess" solution in order to speed up the iterative process. We do not use this approach for the Jacobi solver, because there it lead to unexpected behaviour and the simulation blowing up. We do not know enough about the Jacobi solver to determine why pre-warming does not work with that algorithm.

6 Conclusion

In conclusion the results were quite surprising. Especially the ones concerning the integration methods. The Verlet integration methods seems to perform very well, whereas the Euler method performs worse than we would have expected. Improvements would be to also evaluate $\boldsymbol{f}$ twice in the leapfrog integrator. Since the results of the Midpoint Verlet were very good, it would be interesting to see if we can improve it by evaluating $\boldsymbol{f}$ once more, in the form of a second midstep at the end, using (for instance) $\boldsymbol{a}_{i+\frac{3}{4}}$. Another interesting possibility is to mix the midpoint method with the leapfrog method, and see if that combination yields similar results.