Simulation in Computer Graphics - Project 1

Magnus Menger, Melvin Hoogendoorn and Viljan van Raaij
Group 1

1 Introduction

In this report we describe a particle system we implemented for the course *Simulation in Computer Graphics* that supports multiple different forces, constraints, integration schemes, collision objects and ways of interacting with it. In section 2 we introduce the main features of our implementation. In section 3 we first describe some testing scenes we set up that combine multiple forces, constraints and objects to simulate more complex structures like hair or cloth. We then describe some experiments we ran, to compare the performance and stability of different integration schemes in these scenes.

2 Implemented Features

For our particle system we implemented many features. Some of these features were required by the project description, while others were optional features we added on top. In this section we describe and explain all features implemented by us. In section A we explicitly list which features are optional.

2.1 Generalized Force Structure

We have implemented a generalized force structure, where all forces acting on particles are represented by force objects that share a common interface. These forces have a reference to all particles they depend on and affect. They also provide a function that computes the forces for each affected particle based on their current state and adds them to a per-particle force accumulator. Down below, we list and explain all force objects that we have implemented including the formulas used in their computation.

2.1.1 Spring Force

This is a binary force implementing Hooke's law, that tries to hold particles together at a specified rest length. For two particles i, j we denote the difference of their positions and velocities as $\mathbf{x}_{ij}$ and $\mathbf{v}_{ij}$. We denote the normalized version of a vector by a hat, for example $\hat{\mathbf{x}}_{ij} = \frac{\mathbf{x}_{ij}}{\|\mathbf{x}_{ij}\|}$. These formulas describe the forces acting on both particles:

$$\begin{aligned} F_i &= -k_s \left(\|\mathbf{x}_{ij}\| - r\right) \hat{\mathbf{x}}_{ij} - k_d \left(\hat{\mathbf{x}}_{ij} \left(\mathbf{v}_{ij} \cdot \hat{\mathbf{x}}_{ij}\right)\right), \\ F_j &= -F_i \end{aligned} \tag{1}$$

Here, k_c is the spring constant controlling the stiffness of the force, and k_d is the damping constant controlling the damping strength of the oscillation.

2.1.2 Fixed Endpoint Spring Force

This is a unary force that tries to keep a particle at a fixed rest length to some fixed point in the system. For particle i and the fixed endpoint o, we denote the difference between their positions as $\mathbf{x}_{io}$.

$$F_i = -k_d \left(\|\mathbf{x}_{io}\| - r\right) \hat{\mathbf{x}}_{io} - k_c \left(\hat{\mathbf{x}}_{io} \left(\mathbf{v}_i \cdot \hat{\mathbf{x}}_{io}\right)\right) \tag{2}$$

2.1.3 Linear Force

This is a constant unary force that can be applied to any subset of the particles in the system. For a particle i with mass m_i affected by a linear force, the force is constant independent of the particles position and velocity:

$$F_i = m_i \mathbf{g} \tag{3}$$

Here, $\mathbf{g}$ is a constant vector pointing to the direction in which the force is applied. Gravity can be implemented using this force by specifying $\mathbf{g}$ as a downward pointing vector with a magnitude equal to the gravitational constant.

2.1.4 Angular Spring Force

This is a ternary force that tries to maintain a rest angle β between three particles i, j, k, where j is the middle particle. The Torque τ applied by the angular spring depends on θ, which describes the difference of the actual angle α and the target angle β, and its derivative $\dot{\theta}$ with respect to time. Similar to the spring force it consists of a linear part and a damping part and can be computed with $\tau = k_s\theta - k_d\dot{\theta}$. We can compute θ and $\dot{\theta}$ based on the particle positions and velocities to get

$$\begin{aligned} \alpha &= \text{atan2}(\mathbf{x}_{ij} \times \mathbf{x}_{kj},\ \mathbf{x}_{ij} \times \mathbf{x}_{kj}) \\ \dot{\alpha} &= \frac{(\mathbf{x}_{ij} \cdot \mathbf{x}_{kj})(\mathbf{x}_{ij} \times \mathbf{x}_{kj} + \mathbf{x}_{ij} \times \mathbf{x}_{kj})}{(\mathbf{x}_{ij} \cdot \mathbf{x}_{kj})^2 + (\mathbf{x}_{ij} \times \mathbf{x}_{kj})^2} \\ &\quad - \frac{(\mathbf{x}_{ij} \times \mathbf{x}_{kj})(\mathbf{x}_{ij} \cdot \mathbf{x}_{kj} + \mathbf{x}_{ij} \cdot \mathbf{v}_{kj})}{(\mathbf{x}_{ij} \cdot \mathbf{x}_{kj})^2 + (\mathbf{x}_{ij} \times \mathbf{x}_{kj})^2} \\ \tau &= k_s(\alpha - \beta) - k_d(\dot{\alpha}) \end{aligned}$$

The forces acting on particles i and k are scale with the torque divided by their distance to particle j. Their direction is normal to their vector to k. The middle particle experiences the opposite forces. We denote the normal vectors with an $\mathbf{n}$, e.g. $\mathbf{n}_{ij} = \frac{1}{\|\mathbf{x}_{ij}\|} \begin{bmatrix} \mathbf{x}_{ij}^y \\ -\mathbf{x}_{ij}^x \end{bmatrix}$. The final forces are

$$F_i = -\frac{\tau}{\|\mathbf{x}_{ij}\|}\mathbf{n}_{ij}, \quad F_k = \frac{\tau}{\|\mathbf{x}_{kj}\|}\mathbf{n}_{kj}, \quad F_j = -(F_i + F_k) \tag{4}$$

2.1.5 Linear Drag Force

This is a unary force that applies a force opposite to the velocity direction of each affected particle. The magnitude of the force is linear to the velocity of the particle. For a particle i the linear drag force is computed as follows, where k_q is a strength constant:

$$F_i = -k_l \mathbf{v}_i \tag{5}$$

2.1.6 Quadratic Drag Force

This is a variation of the Linear Drag Force described above, where the magnitude of the force is quadratic to the velocity of the particle instead of linear. This was inspired by real life air resistance which also scales quadratically with velocity. For a particle i the quadratic drag force is computed as follows, where k_q is a strength constant:

$$F_i = -\frac{k_q \|\mathbf{v}\|}{2} \mathbf{v}_i \tag{6}$$

2.1.7 Gravitational Force

This is a n-ary force that mimics Newtons gravitational law by pulling a set of particles together. For each pair of particles, an attractive force is created proportional to the product of their masses and inversely proportional to the squared distance between them. For two particles i, j, with $\mathbf{x_{ij}}$ the difference of their positions, m_i, m_j their masses, G the gravitational constant, the gravitational force is computed as follows:

$$F_i = -G \frac{m_i m_j}{\|\mathbf{x}_{ij}\|^2} \hat{\mathbf{x}}_{ij} \tag{7}$$

2.1.8 Blow Force

This is an n-ary force that applies a force between a point in the system and a set of particles. The magnitude of the force is inversely proportional to the distance between the point and an affected particle. Let $\mathbf{x}_{io}$ be the difference between some affected particle i and the point o, m_i the mass of the point, k_b be a constant controlling the strength of the force, and k_m be an upper limit for the force magnitude. Then the force is computed as follows:

$$F_i = \min\left(\frac{k_b}{\|\mathbf{x}_{io}\|}, k_m\right) m_i \hat{\mathbf{x}}_{io} \tag{8}$$

This force is used for one of our mouse interactions, which is described in subsection 2.3.

2.2 Generalized Constraint Structure

Just like for the forces, we have implemented a generalized constraint structure, where constraints are represented by constraint objects with a common interface for the system. These constraint objects keep track of the particles they affect and allow to compute a constraint value $\mathbf{C}$ based on the particle positions denoted by $\mathbf{q}$ that is 0 exactly if the particles fulfill the constraints. Based on these constraint values we solve a linear system to compute so called constrained forces $\hat{\mathbf{Q}}$, that need to be applied to all particles to keep the constraints fulfilled. We will now explain the derivation of the linear system. Let $\mathbf{q}$ be the state vector containing all particle positions and $\mathbf{C}(\mathbf{q})$ be the vector-valued function containing the values of all our constraint functions, such that all constraints are satisfied if $\mathbf{C} = \vec{0}$. We assume that we are currently in a legal state, i.e. $\mathbf{C} = \vec{0} \wedge \dot{\mathbf{C}} = \vec{0}$. We are solving for $\hat{\mathbf{Q}}$, such that if added up to the other current forces in the system $\mathbf{Q}$, it will hold that $\ddot{\mathbf{C}} = \vec{0}$, ensuring that the particles will stay in a legal state. Let $J = \frac{\partial \mathbf{C}}{\mathbf{q}}$ be the Jacobian of $\mathbf{C}$. We obtain $\ddot{\mathbf{C}}$ by differentiating twice w.r.t time:

$$\dot{\mathbf{C}} = \mathbf{J}\dot{\mathbf{q}}, \qquad \ddot{\mathbf{C}} = \dot{\mathbf{J}}\dot{\mathbf{q}} + \mathbf{J}\ddot{\mathbf{q}}$$

We can replace $\ddot{\mathbf{q}}$ by $\mathbf{W}(\mathbf{Q} + \hat{\mathbf{Q}})$, where $\mathbf{W}$ is the inverse mass matrix containing the inverted mass of each particle on the diagonal:

$$\ddot{\mathbf{C}} = \dot{\mathbf{J}}\dot{\mathbf{q}} + \mathbf{JW}(\mathbf{Q} + \hat{\mathbf{Q}})$$

Setting $\ddot{\mathbf{C}}$ to zero and re-arranging gives:

$$\mathbf{JW}\hat{\mathbf{Q}} = -\dot{\mathbf{J}}\dot{\mathbf{q}} - \mathbf{JWQ}$$

To tackle numerical drift, we solve for $\ddot{\mathbf{C}} = -k_s\mathbf{C} - k_d\dot{\mathbf{C}}$ instead of $\ddot{\mathbf{C}} = 0$, where k_s and k_d are spring and damping constants. In this way particles can return to a valid state smoothly when drift occurs. We then have the following:

$$\mathbf{JW}\hat{\mathbf{Q}} = -\dot{\mathbf{J}}\dot{\mathbf{q}} - \mathbf{JWQ} - k_s\mathbf{C} - k_d\dot{\mathbf{C}}$$

Note again that $\mathbf{J}$ describes all forbidden directions, and that the constraint force $\hat{\mathbf{Q}}$ will only put work in the opposing directions. We can thus express $\hat{\mathbf{Q}}$ by $\mathbf{J}^\mathbf{T}\lambda$, such that we restrict it to the linear combinations of these forbidden directions. We then arrive at the final linear system which we solve for λ using a conjugate gradient solver:

$$\mathbf{JWJ}^\mathbf{T}\lambda = -\dot{\mathbf{J}}\dot{\mathbf{q}} - \mathbf{JWQ} - k_s\mathbf{C} - k_d\dot{\mathbf{C}} \tag{9}$$

In our implementation, we calculate the constraint force with the following steps. After all other forces have been re-calculated by the system, we loop over all constraint objects, and let them calculate their portions of the vectors $\mathbf{C}, \dot{\mathbf{C}}, \mathbf{J}$, and $\dot{\mathbf{J}}$. In this way we can build the matrices which we then in turn use to construct the right-hand-side of the linear system above using matrix and vector operations. Since these matrices are sparse, we implemented an efficient data structure that only stores non zero values and allows for fast computations on spare matrices. Next, we create a function that can compute the product of matrix $\mathbf{JWJ}^\mathbf{T}$ with a vector, which makes use of the fact that $\mathbf{J}$ and $\mathbf{W}$ are sparse, thus making it more efficient. This will allow the conjugate solver to compute the matrix vector product, without needing to construct the whole matrix beforehand. We pass this function with the constructed right-hand-side of the linear system above to the solver to obtain λ. The constraint force vector $\hat{\mathbf{Q}}$ is then obtained by multiplying vector λ with $\mathbf{J}^\mathbf{T}$, and added to the force accumulator of the concerning particle. Down below, we will describe the implemented constraints, and we will provide the formulas for their portions of global matrices $\mathbf{C}, \dot{\mathbf{C}}, \mathbf{J}$, and $\dot{\mathbf{J}}$. Note that portions $C, \dot{C}$ are scalars, and portions $\mathbf{J}, \dot{\mathbf{J}}$ are vectors.

2.2.1 Circular Wire Constraint

This is a unary constraint that restricts a particle's motion to a fixed circle. For a particle i, constraint to a circle with center $\mathbf{c}$ and radius r we can define the equations for their portions of the global matrices as follows:

$$C = (\mathbf{x_i} - \mathbf{c})^2 - r^2 \quad \mathbf{J} = 2(\mathbf{x_i} - \mathbf{c})$$
$$\dot{C} = 2\dot{\mathbf{x}}_\mathbf{i} \cdot (\mathbf{x_i} - \mathbf{c}) \quad \dot{\mathbf{J}} = 2\dot{\mathbf{x}}$$

2.2.2 Rod Constraint

This is a binary constraint that restricts two particles to be at a fixed distance from each other. For two particles i, j, constraint to be at a fixed distance d we can define the equations for the portions of the global matrices of particle i as follows:

$$C = (\mathbf{x_i} - \mathbf{x_j})^2 - d^2 \quad \mathbf{J} = 2(\mathbf{x}_i - \mathbf{x}_j)$$
$$\dot{C} = 2(\mathbf{x_i} - \mathbf{x_j}) \cdot (\dot{\mathbf{x}}_\mathbf{i} - \dot{\mathbf{x}}_j) \quad \dot{\mathbf{J}} = 2(\dot{\mathbf{x}}_i - \dot{\mathbf{x}}_j)$$

The equations for particle j can be obtained by negating the above equations.

2.2.3 Rod Constraint Variant

This is a variant to the rod constraint above. For two particles i, j, constraint to be at a fixed distance d we can define the equations for the portions of global matrices of particle i as follows:

$$C = \|\mathbf{x}_i - \mathbf{x}_j\| - d$$
$$\dot{C} = \frac{(\mathbf{x}_i - \mathbf{x}_j) \cdot (\dot{\mathbf{x}}_i - \dot{\mathbf{x}}_j)}{d}$$
$$\mathbf{J} = \frac{\mathbf{x}_i - \mathbf{x}_j}{d}$$
$$\dot{\mathbf{J}} = -\dot{C}(C+d)\frac{(\mathbf{x}_i - \mathbf{x}_j)}{(C+d)^3} + \frac{(\dot{\mathbf{x}}_i - \dot{\mathbf{x}}_j)}{C+d}$$

The equations for particle j can be obtained by negating the above equations.

2.3 Mouse Interactions

Two mouse interactions have been implemented. The first one creates a spring force between the mouse position and a nearby particle. Whenever the left mouse button is clicked, we go over all particles and check if any of their positions is within some predefined distance to the mouse position. If so, we create a spring force between that particle and an auxiliary particle. We keep updating this auxiliary particle with the mouse position while the left mouse button is still being pressed.

The second mouse interaction creates a blow force from the mouse position to all particles. The details of this blow force are described in subsubsection 2.1.8

2.4 Integration Schemes

We implemented four different integration schemes. These provide a function that progresses the state of our particle system based on some time step h. The state of the particle system consists of the positions $\mathbf{x}$ and velocities $\mathbf{v}$ of our particles. For this computation, the integration schemes can update the current state and use a function that calculates the forces $\mathbf{f}(\mathbf{x}, \mathbf{v})$ acting on particles on the current state. This together with the weight matrix $\mathbf{W}$, can be used to compute the derivative of the current state, which consists of $\mathbf{v}$ and $\mathbf{W}^{-1}\mathbf{f}(\mathbf{x}, \mathbf{v})$. The vector $\mathbf{v}$ together with $\mathbf{f}(\mathbf{x}, \mathbf{v})$ divided by particle masses forms the current derivative of the state with respect to time. Below we will describe how each integration scheme updates the state of the particle system. In subsection 3.2 we will compare these integration schemes with regards to their stability and performance.

2.4.1 Explicit Euler

Explicit Euler integration updates $\mathbf{x}$ by taking a step in the direction of the $\mathbf{v}$ with distance based on $\mathbf{h}$. The velocity $\mathbf{v}$ is then updated based on its current derivative which is calculated by computing $\mathbf{f}$ and dividing it by the particle masses. This method has an error of $O(h^2)$. The full update is as follows:

$$\mathbf{x}_{\text{new}} = \mathbf{x} + h\mathbf{v} \qquad \mathbf{v}_{\text{new}} = \mathbf{v} + h(\mathbf{W}^{-1}\mathbf{f}(\mathbf{x}, \mathbf{v})) \tag{10}$$

2.4.2 Symplectic Euler (Semi-Implicit Euler)

Symplectic Euler integration[1] is similar to Explicit Euler, with the key difference being the order of updates: the velocity $\mathbf{v}$ is updated before the position $\mathbf{x}$, allowing the new velocity to be used in the position update. This method has an error of $O(h^2)$. The full update is as follows:

$$\mathbf{v}_{\text{new}} = \mathbf{v} + h(\mathbf{W}^{-1}\mathbf{f}(\mathbf{x}, \mathbf{v})) \qquad \mathbf{x}_{\text{new}} = \mathbf{x} + h\mathbf{v} \tag{11}$$

2.4.3 Midpoint

The midpoint scheme, evaluates which state $\mathbf{x}_{\text{mid}}, \mathbf{v}_{\text{mid}}$ the system would end up in, when taking an explicit Euler step with half the timestep. It then evaluates which changes would be applied to this midpoint state with a full Euler step and applies these changes to the original states . This method has an error of $O(h^3)$. We compute this as follows:

$$\begin{aligned} \mathbf{x}_{\text{mid}} &= \mathbf{x} + \frac{h}{2}\mathbf{v}, & \mathbf{v}_{\text{mid}} &= \mathbf{v} + \frac{h}{2}\big(\mathbf{W}^{-1}\mathbf{f}(\mathbf{x}, \mathbf{v})\big) \\ \mathbf{x}_{\text{new}} &= \mathbf{x} + h\mathbf{v}_{\text{mid}}, & \mathbf{v}_{\text{new}} &= \mathbf{v} + h\big(\mathbf{W}^{-1}\mathbf{f}(\mathbf{x}_{\text{mid}}, v_{\text{mid}})\big) \end{aligned} \tag{12}$$

2.4.4 4th Order Runge Kutta

This method improves upon the previous methods by taking four evaluations of the derivative at intermediate points and combining them using weighted averages. This method achieves an error of $O(h^5)$ per step. We define intermediate estimates of position and velocity derivatives:

$$\begin{aligned} \mathbf{x}_1 &= \mathbf{x}, & \mathbf{v}_1 &= \mathbf{v}, & \mathbf{a}_1 &= \mathbf{W}^{-1}\mathbf{f}(\mathbf{x}_1, \mathbf{v}_1) \\ \mathbf{x}_2 &= \mathbf{x} + \frac{h}{2}\mathbf{v}_1, & \mathbf{v}_2 &= \mathbf{v} + \frac{h}{2}\mathbf{a}_1, & \mathbf{a}_2 &= \mathbf{W}^{-1}\mathbf{f}(\mathbf{x}_2, \mathbf{v}_2) \\ \mathbf{x}_3 &= \mathbf{x} + \frac{h}{2}\mathbf{v}_2, & \mathbf{v}_3 &= \mathbf{v} + \frac{h}{2}\mathbf{a}_2, & \mathbf{a}_3 &= \mathbf{W}^{-1}\mathbf{f}(\mathbf{x}_3, \mathbf{v}_3) \\ \mathbf{x}_4 &= \mathbf{x} + h\mathbf{v}_3, & \mathbf{v}_4 &= \mathbf{v} + h\mathbf{a}_3, & \mathbf{a}_4 &= \mathbf{W}^{-1}\mathbf{f}(\mathbf{x}_4, \mathbf{v}_4) \end{aligned}$$

The updated state is:

$$\begin{aligned} \mathbf{x}_{\text{new}} &= \mathbf{x} + \frac{h}{6}(\mathbf{v}_1 + 2\mathbf{v}_2 + 2\mathbf{v}_3 + \mathbf{v}_4) \\ \mathbf{v}_{\text{new}} &= \mathbf{v} + \frac{h}{6}(\mathbf{a}_1 + 2\mathbf{a}_2 + 2\mathbf{a}_3 + \mathbf{a}_4) \end{aligned} \tag{13}$$

1. This method was not a part of the codebase we handed in, but was later added to experiment with.

2.4.5 Implicit Euler

The implicit Euler integration scheme works by updating the state in a way, that taking a backwards Euler step for timestep h from the new location would result in the original location.

$$\mathbf{x}_{\text{new}} = \mathbf{x} + h\mathbf{v}_{\text{new}} \quad \mathbf{v}_{\text{new}} = \mathbf{v} + h\,\mathbf{W}^{-1}\mathbf{f}(\mathbf{x}_{\text{new}}, \mathbf{v}_{\text{new}}) = \mathbf{v} + h\,\mathbf{W}^{-1}\mathbf{f}(\mathbf{x} + h\mathbf{v}_{\text{new}}, \mathbf{v}_{\text{new}})$$

We define $\Delta\mathbf{x} = \mathbf{x}_{\text{new}} - \mathbf{x}$ and $\Delta\mathbf{v} = \mathbf{v}_{\text{new}} - \mathbf{v}$. This implies that $\Delta\mathbf{x} = h(\mathbf{v} + \Delta\mathbf{v})$. We use this while approximating $\mathbf{f}$ with a Taylor expansion to get the following:

$$\Delta\mathbf{v} = h\mathbf{W}^{-1}\left(\mathbf{f}(\mathbf{x},\mathbf{v}) + h\frac{\partial\mathbf{f}}{\partial\mathbf{x}}(\mathbf{v}+\Delta\mathbf{v}) + \frac{\partial\mathbf{f}}{\partial\mathbf{v}}\Delta\mathbf{v}\right)$$

We define $\mathbf{J_x} = \frac{\partial\mathbf{f}}{\partial\mathbf{x}}, \mathbf{J_v} = \frac{\partial\mathbf{f}}{\partial\mathbf{v}}$ as the Jacobian matrices of $\mathbf{f}$ with respect to particle positions and velocities and $\mathbf{f_0} = \mathbf{f}(\mathbf{x},\mathbf{v})$. With this we can rearrange our formula to a form that can be solved for $\Delta\mathbf{v}$ numerically with the conjugate gradient method [1].

$$\left(\mathbf{I} - h^2\mathbf{W}^{-1}\mathbf{J_x} - h\mathbf{W}^{-1}\mathbf{J_v}\right)\Delta\mathbf{v} = h\mathbf{W}^{-1}\mathbf{f}_0 + h^2\mathbf{W}^{-1}\mathbf{J_x}\mathbf{v} \tag{14}$$

After computing $\Delta\mathbf{v}$, we compute the new state as $\mathbf{v}_{\text{new}} = \mathbf{v} + \Delta\mathbf{v}$ and $\mathbf{x}_{\text{new}} = \mathbf{x} + h\mathbf{v}_{\text{new}}$.

Each force from our generalized force structure has an influence on only two cells per involved particle in each matrix $\mathbf{J_x}$ and $\mathbf{J_v}$. Since most forces only involve a few particles, we therefore use a sparse matrix data structure to represent these. Each force class in our system needs to implement functions for computing their contribution to $\mathbf{J_x}$ and $\mathbf{J_v}$. We implemented these functions for the fixed endpoint spring force, spring force, quadratic drag force and linear force. Scenes containing other forces than that can currently not be simulated with implicit Euler. [2] Below we list the formulas to compute the Jacobian for all forces for which we implemented implicit Euler. For the sake of brevity, do not list all derivation steps.

Spring Force: The force acting on a particle i connected to j via a spring force is described in Equation 1. Differentiating this with respect to $\mathbf{x_{ij}}$ gives

$$\begin{aligned}\frac{\partial F_i}{\partial \mathbf{x_{ij}}} =& -k_s\left[\left(1 - \frac{r}{\|\mathbf{x}_{ij}\|}\right)\cdot\left(\mathbf{I} - \hat{\mathbf{x}}_{ij}\hat{\mathbf{x}}_{ij}^\top\right) + \hat{\mathbf{x}}_{ij}\hat{\mathbf{x}}_{ij}^\top\right] \\ &- \frac{k_d}{\|\mathbf{x}_{ij}\|^2}\left[\hat{\mathbf{x}}_{ij}\mathbf{v}_{ij}^\top\cdot\left(\mathbf{I} - \hat{\mathbf{x}}_{ij}\hat{\mathbf{x}}_{ij}^\top\right)\right] \\ &- \frac{k_d}{\|\mathbf{x}_{ij}\|^2}\left[\mathbf{v}_{ij}\hat{\mathbf{x}}_{ij}^\top\cdot\left(\mathbf{I} - \hat{\mathbf{x}}_{ij}\hat{\mathbf{x}}_{ij}^\top\right)\right]\end{aligned}$$

With this we can calculate $\frac{\partial F_i}{\partial \mathbf{x_i}} = \frac{\partial F_i}{\partial \mathbf{x_{ij}}}$ and $\frac{\partial F_j}{\partial \mathbf{x_i}} = -\frac{\partial F_i}{\partial \mathbf{x_{ij}}}$. Since $F_j = -Fi$, we also have $\frac{\partial F_j}{\partial \mathbf{x_i}} = -\frac{\partial F_i}{\partial \mathbf{x_{ij}}}$ and $\frac{\partial F_j}{\partial \mathbf{x_j}} = \frac{\partial F_i}{\partial \mathbf{x_{ij}}}$. Differentiating Fi with respect to $\mathbf{v_{ij}}$ gives

$$\frac{\partial F_i}{\partial \mathbf{v_{ij}}} = -k_d(\hat{\mathbf{x}}_{ij}\mathbf{x}_{ij}^\top)$$

With this we can analogously compute $\frac{\partial F_i}{\partial \mathbf{x_i}}, \frac{\partial F_i}{\partial \mathbf{x_j}}, \frac{\partial F_j}{\partial \mathbf{x_i}}$ and $\frac{\partial F_j}{\partial \mathbf{x_j}}$.

Fixed Endpoint Spring Force: You can also compute the Jacobian for fixed endpoint spring forces with the same formulas above, by replacing $\mathbf{x}_{ij}$ with the difference in position between the single particle i and the fixed endpoint o and replacing $\mathbf{v}_{ij}$ with $\mathbf{v}_i$, since the fixed endpoint doesn't move.

Quadratic Drag Force: The quadratic drag force on a particle i is described in Equation 6. Differentiating it yields these Jacobians (note that $\mathbf{0}$ denotes a 2×2 matrix of zeros):

$$\frac{\partial F_i}{\partial \mathbf{x}_i} = \mathbf{0} \qquad \frac{\partial F_i}{\partial \mathbf{v}_i} = -\frac{k_q}{2}\left(\mathbf{v}\hat{\mathbf{v}}^\top + \|\mathbf{v}\|\mathbf{I}\right)$$

Linear Force: The formula for the linear force, as described in Equation 3, does not depend on a particles position or velocity. Therefore its Jacobians are $\mathbf{0}$ matrices.

2.5 Collisions Between Particles And Solid Objects

The collision system is composed of various collidable object types implementing the same collision interface. Currently, two collidable objects are implemented: infinite planes and circles.

A particle is considered to have collided with a solid object when it lies within a predefined distance threshold ϵ from the object's surface, and has a velocity in the direction towards the object. The simplest scenario is where the particle is close to, but has not penetrated, the surface. In this case, the normal component of the particle's velocity, denoted $\mathbf{v}_{\text{normal}}$, is reflected and scaled by a bounciness factor $b \in [0,1]$, which models energy loss during the collision. The tangential velocity component $\mathbf{v}_{\text{tangential}}$ remains unchanged. This can be formulated in the following equations, where $\hat{\mathbf{n}}$ is the unit normal vector of the surface:

$$\mathbf{v}_{\text{normal}} = -b\cdot\mathbf{v}_{\text{normal}} = -b\cdot(\mathbf{v}\cdot\hat{\mathbf{n}})\cdot\hat{\mathbf{n}} \tag{15}$$

$$\mathbf{v}_{\text{tangential}} = \mathbf{v} - \mathbf{v}_{\text{normal}} \tag{16}$$

A more complex situation arises when the particle has already crossed the surface of the solid object. To address this, a backtracking factor $t \in (0,1]$ is computed to interpolate the particle's position and velocity back to an approximate point of intersection:

$$\mathbf{v}_{\text{intersection}} = \mathbf{v}_{\text{new}} + t\cdot(\mathbf{v}_{\text{previous}} - \mathbf{v}_{\text{new}}) \tag{17}$$

$$\mathbf{x}_{\text{intersection}} = \mathbf{x}_{\text{new}} + t\cdot(\mathbf{x}_{\text{previous}} - \mathbf{x}_{\text{new}}) \tag{18}$$

2. During the writing of this report, we noticed that we missed the $\mathbf{W}^{-1}$ expressions in our handed in implementation for implicit Euler. It therefore only works correctly in scenes where all particles have mass 1. This is the case in all scenes that only use forces that support implicit Euler.

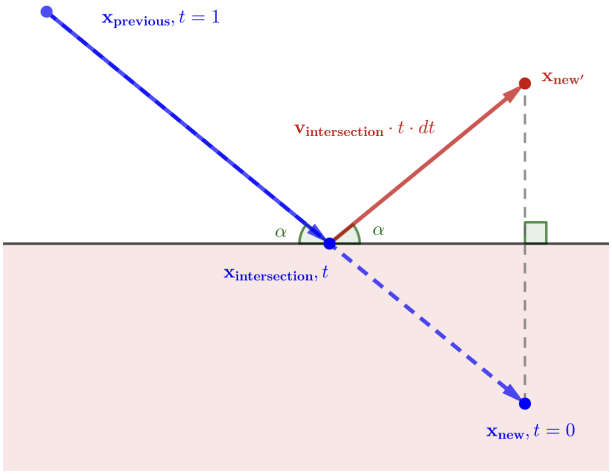


Fig. 1: Backtracking from $\mathbf{x_{new}}$ to $\mathbf{x_{intersection}}$, and then additional movement with flipped normal velocity to final particle position $\mathbf{x_{new'}}$.

A consequence of the backtracking approach is that the colliding particle does not complete its full timestep. Our current solution to this issue — although arguably suboptimal as discussed later — is as follows: after backtracking the particle to the estimated point of intersection and reflecting the normal component of its velocity, the particle is advanced a distance proportional to the remaining timestep:

$$\mathbf{x}_{\text{new}'} = \mathbf{x}_{\text{intersection}} + \mathbf{v}_{\text{intersection}} \cdot t \cdot dt \tag{19}$$

where dt is the time step of the simulation (see figure Figure 1). In practice we don't use Equation 19, but implement the following formula which gives us the exact same result:

$$\mathbf{x}_{\text{new}'} = \mathbf{x}_{\text{new}} - (2 \cdot (\mathbf{x}_{\text{new}} - \mathbf{x}_{\text{intersection}}) \cdot \hat{\mathbf{n}}) \cdot \hat{\mathbf{n}} \tag{20}$$

However, this implementation neglects any external forces that act on the particle during the remaining timestep following the collision. As a result, the post-collision movement (represented by the red arrow in Figure 1) may not be accurate.

To address this limitation, we propose an alternative: after backtracking the particle's position and velocity and reflecting the normal velocity component as before, we run a new simulation step using only the remaining time fraction $t \cdot dt$. This ensures that external forces are properly accounted for during the post-collision motion.

2.5.1 Plane Collision Object

We define a plane collision object by an origin point $\mathbf{x_0}$ and its normal $\hat{\mathbf{n}}$. A particle with the position $\mathbf{x}$ is considered to be colliding if it is within the distance ϵ from the surface of the plane:

$$(\mathbf{x} - \mathbf{x_0}) \cdot \hat{\mathbf{n}} < \epsilon \tag{21}$$

and has a velocity $\mathbf{v}$ in the direction towards the plane:

$$\hat{\mathbf{n}} \cdot \mathbf{v} < 0 \tag{22}$$

2.5.2 Circular Collision Object

We define a circular collision object by an origin point $\mathbf{x_0}$ and a radius r. A particle with the position $\mathbf{x}$ is considered to be colliding if it is within the distance ϵ from the surface of the circle:

$$|\mathbf{x} - \mathbf{x_0}| - r < \epsilon \tag{23}$$

and has a velocity $\mathbf{v}$ in the direction towards the circle origin:

$$(\mathbf{x} - \mathbf{x_0}) \cdot \mathbf{v} < 0 \tag{24}$$

2.5.3 Friction force

When a particle is sliding along the surface of a collidable object, a friction force F_f is calculated:

$$F_f = |\mathbf{F} \cdot \hat{\mathbf{n}}| \cdot k_f \tag{25}$$

where $\mathbf{F}$ represents the sum of all other external forces acting on the particle, $\hat{\mathbf{n}}$ the surface normal at the contact point, and k_f a friction coefficient. F_f is then used to slow down the tangential component of the particle's velocity, $\mathbf{v}_{\text{tangential}}$:

$$\mathbf{v}_{\text{tangential}} = \max(\mathbf{v}_{\text{tangential}} - F_f \cdot dt \cdot \mathbf{v}_{\text{tangential}}, 0) \tag{26}$$

A particle is considered to be sliding when a collision has occurred and its velocity is approximately parallel to the surface. The latter can be expressed as:

$$|\hat{\mathbf{n}} \cdot \mathbf{v}| < \epsilon \tag{27}$$

Ideally, the detection of sliding should also include a check that the particle is within an ϵ distance of the surface on both sides. However, currently we only check if the particle is within ϵ from the outside, indirectly through Equation 21 and Equation 23.

2.6 Force Visualization

We have implemented a visualization for the final resulting force that acts on each particle, an example is seen in Figure 2, where three moving particles attract each other with gravitational forces.

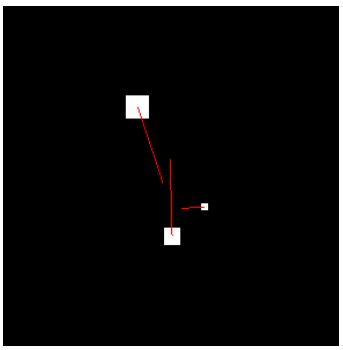


Fig. 2: Force visualization.

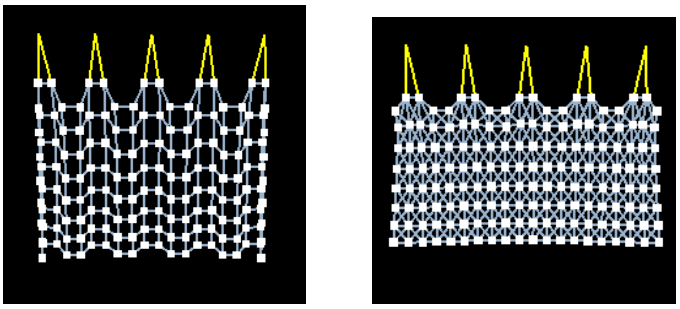
3 Experiments

3.1 Simulation Scenes

We have experimented with our particle system simulation by creating a variety of scenes. Below, we will describe these experiments and discuss what worked well and what did not.

3.1.1 Cloth Simulation

We have created a cloth simulation by forming a rectangular grid of particles connected with spring forces. Fixing the cloth to some points can be done using our fixed endpoint spring force from subsubsection 2.1.2. We also experimented doing this by applying the circular wire constraints from subsubsection 2.2.1 using a small circle radius. This however led to the fixed points becoming unstable, therefore we decided to stick with the fixed endpoint spring. Adding diagonal springs to the grid helps prevent shear deformation of the cloth. A comparison between a grid configuration with and without diagonal springs is depicted in Figure 3, where the cloth is attached to some fixed points and a gravity force acts on the particles. The diagonal configuration increases the total amount of springs and therefore the cloth becomes stiffer, which can be seen by the decreased distance between particles in the grid.



(a) No diagonal springs (b) Diagonal springs

Fig. 3: Comparison of cloth configurations.

We have experimented with wall collisions of a cloth. When a cloth with diagonal springs approaches the wall slowly, it undergoes a normal collision where the cloth maintains its shape, and bounces off the wall surface smoothly. When we let the cloth collide at a higher velocity, for example using the blow force mouse interaction, it becomes tangled. This can be seen in Figure 4. In contrast, collisions of a cloth without diagonal springs led to it becoming tangled in all cases, this due to the lack of shear resistance.

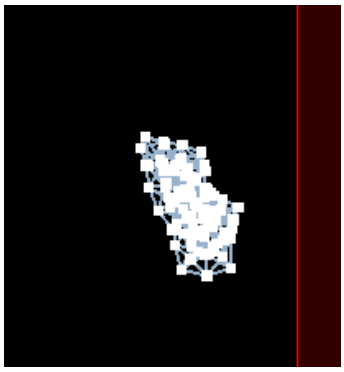


Fig. 4: Cloth with diagonal springs getting tangled after a wall collision.

3.1.2 Hair Simulation

We have created a hair simulation, which is depicted in Figure 5. Hair strands consist of a chain of particles connected with rod constraints. Angular spring forces are applied to consecutive groups of three particles along the strand. The hair strands are fixed to the top of the head by constraining the top particles with circular wire constraints from subsubsection 2.2.1. The head itself is a circular collision object, making the hair strands fall and rest along the sides of the head.

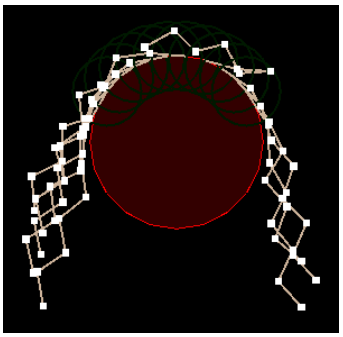


Fig. 5: Hair simulation of ten hairs with rest angle $\beta = 110$.

3.1.3 Rigid Body Structure Simulation

We have created a simulation to show of the behavior of a rigid body, which is shown in Figure 6. The rigid body consists of a small rectangular grid of particles connected with rod constraints, including connections on the diagonals. The top-left particle of the grid is constrained to a circular wire constraint with a large radius, while the bottom-right particle is subject to a fixed endpoint spring force.

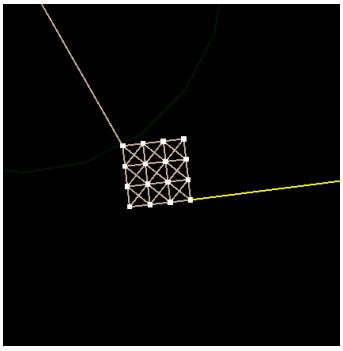


Fig. 6: Rigid body simulation.

3.1.4 Rolling Car Simulation

We have created a rolling car simulation, which is shown in Figure 7. The wheels are constructed by placing particles on a circle, and connecting them sequentially with spring forces. Additionally, all particles are connected with a spring force to a particle in the middle of the wheel to let the wheel maintain its circular shape. In our scene, we connect the center particles of the wheels with two additional particles with rod constraints to form the chassis of the car, as can be seen in the figure.

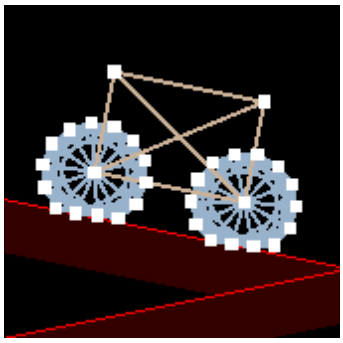


Fig. 7: Car rolling down a hill.

3.2 Comparison of Integration Schemes

To compare the integration schemes, we evaluate their stability and simulation rate across a range of time steps. By simulation rate we mean the elapsed time in the simulation for each second of computation. Only execution time of the actual physics simulation is included. In other words, any time needed for UI rendering, user input handling, etc. is ignored.

E.g.: if we can run 1000 simulation steps with a time step $dt = 0.1$ in 1 second, the average simulation rate is

$$\frac{\text{n\_simulation\_steps} \cdot dt}{\text{elapsed\_time}} = \frac{1000 \cdot 0,1}{1s} = 100/s$$

The simulation rate result in the tables 1-4 are given as 3-run averages for each scenario.

One of the more obvious results is the difference in stability between Explicit Euler and Symplectic Euler. Even though their simulation rate is about equal, the latter is a lot more stable, even at higher time steps. An additional interesting observation is how Explicit Euler, in contrast to Symplectic Euler, explodes for a simple undamped harmonic system, regardless of how small the time step is.

As expected, the more advanced integration schemes, such as Midpoint and Runge-Kutta 4, remain stable at higher time steps. In some cases, these methods also outperform simpler schemes in terms of effective simulation rate. For example, in the cloth collision scenario (see Table 2), Runge-Kutta 4 achieves a stable simulation rate of $4103/s$ at $dt = 0,3$. Explicit Euler, despite being significantly less computationally expensive, only achieves a simulation rate of $2353/s$ at $dt = 0,05$.

The implementation of Implicit Euler appears unexpectedly unstable, contradicting theoretical expectations that implicit methods should remain stable at larger time steps [2]. This larger time step should in theory compensate for the algorithm being considerably more computationally expensive. This suggests a potential flaw in our implementation. However, in cases where it does not diverge, the simulation results are similar to those of the other methods in terms of quality. We also observe that Implicit Euler is approximately ten times slower than Explicit Euler, which aligns with expectations.

Finally, all integration schemes explode whenever $dt \geq 0,4$ in the scenes from table 1-4.

3. The simulation in these scenarios are not very accurate. Particles bounces a lot when colliding and is therefore not effected as much by the friction force.

TABLE 1: Comparison of Integration Schemes for Cloth Scene from Figure 3b.

Integrator	Timestep	Avg. Simulation Rate	Stable
Explicit Euler	0,05	–	No
Symplectic Euler	0,05	1009 /s	Yes
Midpoint	0,05	523 /s	Yes
RK4 (Runge-Kutta)	0,05	245 /s	Yes
Implicit Euler	0,05	12,7 /s	Yes
Explicit Euler	0,1	–	No
Symplectic Euler	0,1	1781 /s	Yes
Midpoint	0,1	737 /s	Yes
RK4 (Runge-Kutta)	0,1	492 /s	Yes
Implicit Euler	0,1	19,0 /s	Yes
Explicit Euler	0,2	–	No
Symplectic Euler	0,2	–	No
Midpoint	0,2	–	No
RK4 (Runge-Kutta)	0,2	951 /s	Yes
Implicit Euler	0,2	–	No
Explicit Euler	0,3	–	No
Symplectic Euler	0,3	–	No
Midpoint	0,3	–	No
RK4 (Runge-Kutta)	0,3	1495 /s	Yes
Implicit Euler	0,3	–	No

TABLE 2: Comparison of Integration Schemes for Cloth Collision Scene from Figure 4. The setup is slightly modified to have a constant blow force.

Integrator	Timestep	Avg. Simulation Rate	Stable
Explicit Euler	0,05	2353 /s	Yes[3]
Symplectic Euler	0,05	2289 /s	Yes
Midpoint	0,05	1362 /s	Yes
RK4 (Runge-Kutta)	0,05	695 /s	Yes
Implicit Euler	0,05	23,0	Yes
Explicit Euler	0,1	–	No
Symplectic Euler	0,1	4158 /s	Yes
Midpoint	0,1	2261 /s	Yes
RK4 (Runge-Kutta)	0,1	1342 /s	Yes
Implicit Euler	0,1	–	No
Explicit Euler	0,2	–	No
Symplectic Euler	0,2	–	No
Midpoint	0,2	–	No
RK4 (Runge-Kutta)	0,2	2639 /s	Yes
Implicit Euler	0,2	–	No
Explicit Euler	0,3	–	No
Symplectic Euler	0,3	–	No
Midpoint	0,3	–	No
RK4 (Runge-Kutta)	0,3	4103 /s	Yes
Implicit Euler	0,3	–	No

4. Implicit Euler is not supported in the hair scene.

TABLE 3: Comparison of Integration Schemes for Hair Scene from Figure 5.

Integrator	Timestep	Avg. Simulation Rate	Stable
Explicit Euler	0,05	232 /s	Yes
Symplectic Euler	0,05	209 /s	Yes
Midpoint	0,05	130 /s	Yes
RK4 (Runge-Kutta)	0,05	62 /s	Yes
Implicit Euler[4]	–	–	–
Explicit Euler	0,1	–	No
Symplectic Euler	0,1	471 /s	Yes
Midpoint	0,1	246 /s	Yes
RK4 (Runge-Kutta)	0,1	140 /s	Yes
Implicit Euler[4]	–	–	–
Explicit Euler	0,2	–	No
Symplectic Euler	0,2	–	No
Midpoint	0,2	508 /s	Yes
RK4 (Runge-Kutta)	0,2	281 /s	Yes
Implicit Euler[4]	–	–	–
Explicit Euler	0,3	–	No
Symplectic Euler	0,3	–	No
Midpoint	0,3	–	No
RK4 (Runge-Kutta)	0,3	417 /s	Yes
Implicit Euler[4]	–	–	–

TABLE 4: Comparison of Integration Schemes for Car Scene from Figure 7.

Integrator	Timestep	Avg. Simulation Rate	Stable
Explicit Euler	0,05	1322 /s	Yes
Symplectic Euler	0,05	1317 /s	Yes
Midpoint	0,05	848 /s	Yes
RK4 (Runge-Kutta)	0,05	478 /s	Yes
Implicit Euler	0,05	–	No
Explicit Euler	0,1	2651 /s	Yes[3]
Symplectic Euler	0,1	–	No
Midpoint	0,1	1729 /s	Yes
RK4 (Runge-Kutta)	0,1	945 /s	Yes
Implicit Euler	0,1	–	No
Explicit Euler	0,2	–	No
Symplectic Euler	0,2	–	No
Midpoint	0,2	–	No
RK4 (Runge-Kutta)	0,2	–	No
Implicit Euler	0,2	–	No
Explicit Euler	0,3	–	No
Symplectic Euler	0,3	–	No
Midpoint	0,3	–	No
RK4 (Runge-Kutta)	0,3	–	No
Implicit Euler	0,3	–	No

3.3 Comparison of rod constraint variants

We created a double-pendulum scene for comparing the two different rod constraints from subsubsection 2.2.2 and subsubsection 2.2.3, of which the setup is depicted in Figure 8. We computed 3-run averages on 20 second runs using Runge-Kutta 4 as integrator. For the double-pendulum scene, the normal rod constraint obtained an average simulation rate of 4162 /s, while the rod constraint variant obtained an average simulation rate of 3703 /s. The lower simulation rate for the rod variant can be explained by the divisions required for the equations for $\dot{C}, \mathbf{J}, \dot{\mathbf{J}}$ in subsubsection 2.2.3, as they are more computationally expensive than the simple multiplications in the equations for the normal rod constraint. The division also leads to another problem: if distance d becomes too small, the terms will become really large. We also see this in practice: when we create a rod constraint variant with a very small distance d between two particles, the particles explode immediately when we start the simulation. For the normal rod constraint, this does not happen, and the particles stay in place as expected.

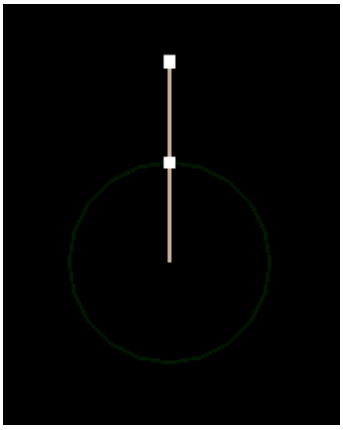


Fig. 8: Double Pendulum setup: one particle constrained by a circle, while being connected to another particle with a rod. Gravity acts on both particles.

4 Conclusion

In this project, we developed a particle system capable of simulating complex physical behaviors through various forces, constraints, integration schemes, collisions, and user interactions. We successfully simulated diverse scenes such as cloth, hair, rigid bodies, and a rolling car. Our experiments showed that Symplectic Euler offered a strong balance between performance and stability, while Explicit Euler was unstable even at small timesteps. Midpoint and Runge-Kutta 4 maintained stability at larger timesteps, sometimes outperforming simpler methods in simulation rate. Implicit Euler was less stable than expected, likely due to implementation issues. Overall, the system performed well and demonstrated realistic behavior across scenarios. Future improvements could target collision handling and implicit integration. This project provided a solid foundation for further work in physically based simulation for graphics.

APPENDIX A
FULL LIST OF OPTIONAL FEATURES

1) Fixed Endpoint Spring Force
2) Linear Drag Force
3) Quadratic Drag Force
4) Gravitational Force
5) Blow Force
6) Circular Collision Shapes
7) Collision With Tangential Friction
8) Force Visualization
9) Additional Complex Scenes (E.g. Rolling Car Simulation, Rigid Body Structure Simulation)

APPENDIX B
INDIVIDUAL CONTRIBUTIONS

Note: During this group project all team members participated very actively and with similar effort and with high collaboration on almost all task. It is therefore difficult to differentiate which part of the work should be contributed to whom. In the list below the features and tasks are listed below the team members who worked most actively on them but it does not mean that they were only implemented by a single person. For most of the features all team members were actively involved in either conception, implementation, bugfixing or optimization.

Magnus Menger:

1) Fixed endpoint spring, angular spring, linear drag, quadratic drag and gravitational forces
2) Generalized constraint structure (implemented together as group)
3) Midpoint, Runge Kutta and Implicit Euler integration
4) Force Visualization
5) Hair, rigid body and car simulation scene

Melvin Hoogendoorn:

1) Generalized force structure
2) Spring, linear, and blow forces
3) Generalized constraint structure (implemented together as group)
4) Explicit Euler integration
5) Cloth Simulation Scene

Viljan van Raaj:

1) Generalized constraint structure (implemented together as group)
2) Mouse interactions
3) Collisions between particles and solid objects
4) Experiments to compare performance of integration schemes
5) Production of the demo video

REFERENCES

[1] M. Hestenes and E. Stiefel, "Methods of conjugate gradients for solving linear systems," vol. 49, no. 6, p. 409. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/jres/049/jresv49n6p409_A1b.pdf

[2] D. Baraff and Pixar Animation Studios, "Physically based modeling," SIGGRAPH '99 Course Notes. [Online]. Available: https://graphics.stanford.edu/courses/cs448b-00-winter/papers/phys_model.pdf