Position Based Fluids

How to simulate fluids

Intro

Much bigger field than expected

Agenda:
- Background
- Smoothed Particle Hydrodynamics
- Navier-Stokes equations
- Position Based Dynamics
- Combining SPH and PBD (the paper)
- Video
- Questions?

Background

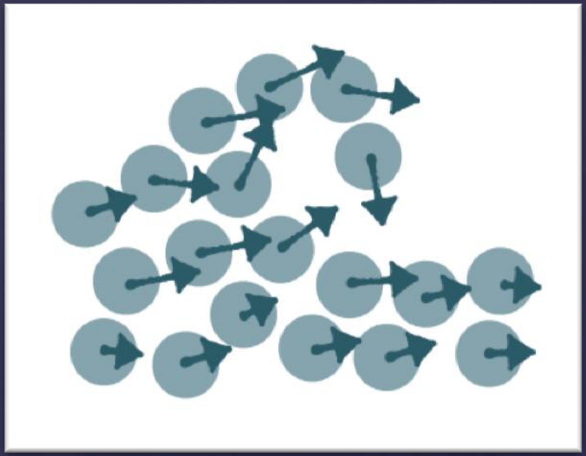
What is a fluid?

- Liquids (e. g. water)
- Gasses (e. g. air)
- Plasmas

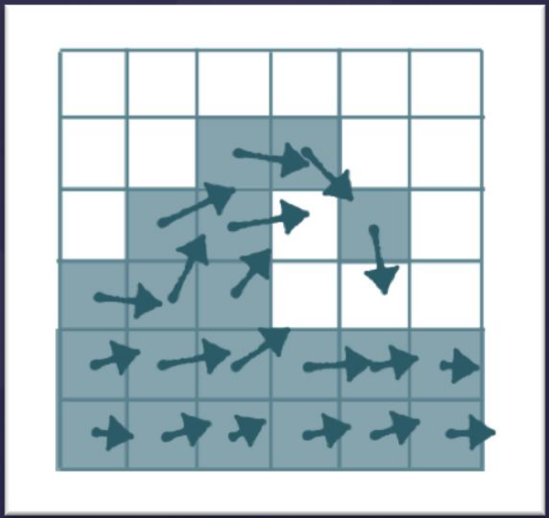
Fluids continually deforms (flows) from high pressure to low pressure

Background

Ways of representing fluids



Lagrangian
(particles)



Eulerian
(grids)

Background

Macklin and Müller's approach (the paper):

"Combines Smoothed Particle Hydrodynamic (SPH) with Position Based Dynamics (PBD)"

So, what is SPH and PBD?

Smoothed Particle Hydrodynamics (SPH)

SPH is a computational method for simulating fluid flows. SPH uses the Lagrangian representation (particles)

Introduced by Monoghan 1992 using smoothing kernels and approximations to the Navier-Stokes equations

Navier-Stokes equations

The Navier-Stokes equations describe how flows in fluids behave.

The Navier-Stokes equations were derived by Navier, Poisson, Saint-Venant, and Stokes between 1827 and 1845.

There are no analytical solutions, only numerical or simplified solutions. The Navier-Stokes equations is one of the Millennium Prize Problems

Navier-Stokes equations

The incompressible N-S equation:

$$\rho\left[\frac{\partial v}{\partial t} + v \bullet \nabla v\right] = \rho g - \nabla p + \mu \nabla^2 v$$

ϱ: density
g: gravity
μ: viscosity

p: pressure
v: velocity

$$\rho(\nabla \bullet v) = 0$$

Always solved together with the continuity equation

Navier-Stokes equations

The incompressible N-S equation:

$$\rho\left[\frac{\partial v}{\partial t} + v \bullet \nabla v\right] = \rho g - \nabla p + \mu \nabla^2 v$$

Inertial forces

Navier-Stokes equations

The incompressible N-S equation:

$$\rho\left[\frac{\partial v}{\partial t} + v \bullet \nabla v\right] = \rho g - \nabla p + \mu \nabla^2 v$$

gravity

Navier-Stokes equations

The incompressible N-S equation:

$$\rho\left[\frac{\partial v}{\partial t}+v\bullet\nabla v\right]=\rho g-\nabla p+\mu\nabla^{2}v$$

pressure forces

Navier-Stokes equations

The incompressible N-S equation:

$$\rho\left[\frac{\partial v}{\partial t} + v \bullet \nabla v\right] = \rho g - \nabla p + \mu \nabla^2 v$$

Viscous forces

Navier-Stokes equations

N-S equation for one particle:

$$\frac{dv_i}{dt} = g - \frac{1}{\rho_i}\nabla p + \frac{\mu}{\rho_i}\nabla^2 v$$

The derivative of the velocity of particle i, with respect to time (acceleration)

From N-S to SPH

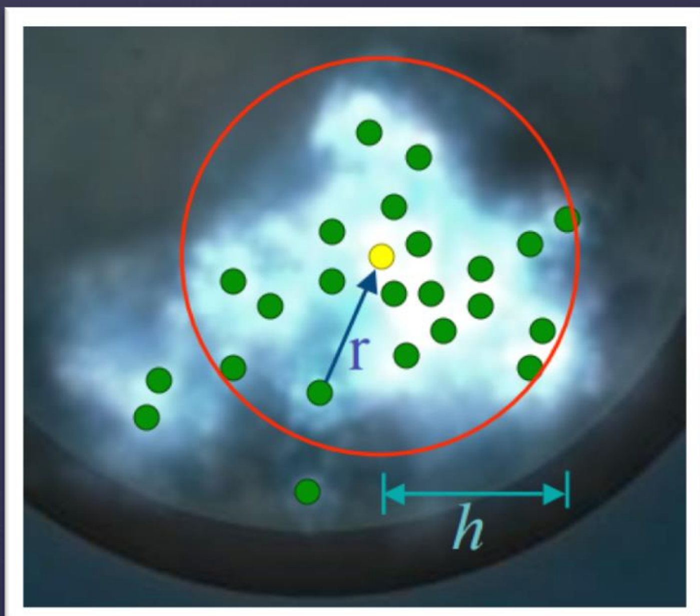
The SPH framework:

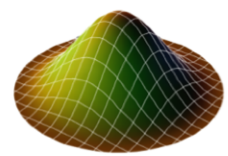
Evaluate a field anywhere by weighted sum

$$A(\mathbf{x}) = \sum_i A_i \frac{m_i}{\rho_i} W(\mathbf{x} - \mathbf{x}_i)$$

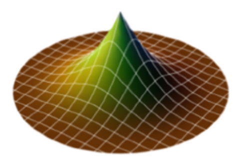
From N-S to SPH

Examples of smoothing kernels:





$$W_{poly6}(\mathbf{r}, h) = \frac{315}{64\pi h^9}(h^2 - |\mathbf{r}|^2)^3$$



$$\nabla W_{spiky}(\mathbf{r}, h) = \frac{45}{\pi h^6}(h - |\mathbf{r}|)^2 \frac{\mathbf{r}}{|\mathbf{r}|}$$

$$\nabla^2 W(r - r_b, h) \equiv \frac{45}{\pi h^6}\left(h - \left\| r - r_b \right\|\right)$$

From N-S to SPH

Approximate the N-S equations using SPH framework

$$\rho_i \approx \sum_j m_j W(r - r_j, h)$$

$$\frac{\nabla p_i}{\rho_i} \approx \sum_j m_j \left(\frac{p_i}{\rho_i^2} + \frac{p_j}{\rho_j^2} \right) \nabla W(r - r_j, h)$$

$$\frac{\mu}{\rho_i} \nabla^2 v_i \approx \frac{\mu}{\rho_i} \sum_j m_j \left(\frac{v_j - v_i}{\rho_j} \right) \nabla^2 W(r - r_j, h)$$

From N-S to SPH

Plug the equations into the original equation

$$\frac{\nabla p_i}{\rho_i} \approx \sum_j m_j \left(\frac{p_i}{\rho_i^2} + \frac{p_j}{\rho_j^2} \right) \nabla W(r - r_j, h)$$

$$\frac{\mu}{\rho_i} \nabla^2 v_i \approx \frac{\mu}{\rho_i} \sum_j m_j \left(\frac{v_j - v_i}{\rho_j} \right) \nabla^2 W(r - r_j, h)$$

$$\frac{dv_i}{dt} = g - \frac{1}{\rho_i} \nabla p + \frac{\mu}{\rho_i} \nabla^2 v$$

Updating SPH

Each time step:

- Compute acceleration **a** of each particle

$$\left.\begin{array}{l}\text{- Update velocities: } \mathbf{v} = \mathbf{v} + \mathbf{a}\,\mathrm{d}t \\ \text{- Update positions: } \mathbf{x} = \mathbf{x} + \mathbf{v}\,\mathrm{d}t\end{array}\right\}\ \text{Leapfrog integration}$$

Position Based Dynamics

Introduced by Müller et al. (NVIDIA)
At the same time by Jos Stam (Maya)
} in 2006

Popular because of generality, simplicity, robustness and efficiency

Used in PhysX, Havok Cloth, Maya nCloth and Bullet

Position Based Dynamics

Position Based Dynamics

Main idea:

- Instead of forces – constraints
- Instead of integration – constraint projection

Mass points:

mass m_i, position x_i, and velocity v_i

Constraints:

E.g. distance constraint

Position Based Dynamics

A constraint consists of:

a cardinality n_j
a function C_j
a set of indices $\{i_1,...i_n\}$
a stiffness parameter k_j
a type: *equality* or *inequality*

Example (distance constraint):

cardinality: 2
function: $C(p_1,p_2) = \| p_1 - p_2 \| - d^2 = 0$
set of indices: $[1,... N]$
stiffness parameter: range from 0-1
type: **equality**

Main PBD algorithm

(1) **forall** vertices i
(2) initialize $\mathbf{x}_i = \mathbf{x}_i^0, \mathbf{v}_i = \mathbf{v}_i^0, w_i = 1/m_i$
(3) **endfor**
(4) **loop**
(5) **forall** vertices i **do** $\mathbf{v}_i \leftarrow \mathbf{v}_i + \Delta t w_i \mathbf{f}_{ext}(\mathbf{x}_i)$
(6) dampVelocities($\mathbf{v}_1, \ldots, \mathbf{v}_N$)
(7) **forall** vertices i **do** $\mathbf{p}_i \leftarrow \mathbf{x}_i + \Delta t \mathbf{v}_i$
(8) **forall** vertices i **do** generateCollisionConstraints($\mathbf{x}_i \rightarrow \mathbf{p}_i$)
(9) **loop** solverIterations **times**
(10) projectConstraints($C_1, \ldots, C_{M+M_{coll}}, \mathbf{p}_1, \ldots, \mathbf{p}_N$)
(11) **endloop**
(12) **forall** vertices i
(13) $\mathbf{v}_i \leftarrow (\mathbf{p}_i - \mathbf{x}_i)/\Delta t$
(14) $\mathbf{x}_i \leftarrow \mathbf{p}_i$
(15) **endfor**
(16) velocityUpdate($\mathbf{v}_1, \ldots, \mathbf{v}_N$)
(17) **endloop**

(1)-(3)

initialize the state variables

(7)

estimates $\mathbf{p}_i$ for new locations of the vertices. Computed using an explicit Euler integration step

(9)-(11)

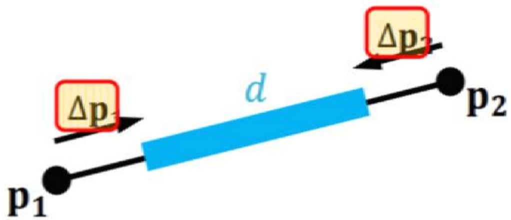
Manipulates position estimates such that they satisfy the constraints using Gauss-Seidel

(13)-(14)

The positions of the vertices are moved to the optimized estimates and the velocities are updated accordingly

Example: Distance constraint

$$C(\mathbf{p_1}, \mathbf{p_2}) = \|\mathbf{p_1} - \mathbf{p_2}\| - d^2 = 0$$



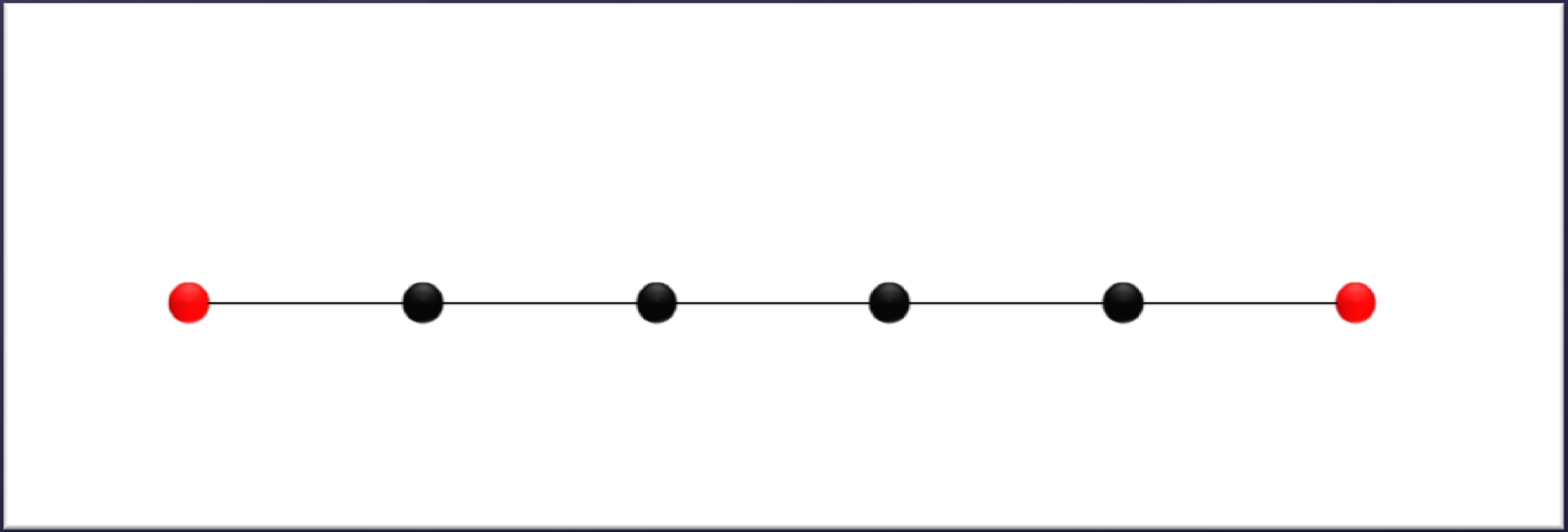
$$\Delta\mathbf{p_1} = -(|\mathbf{p_1} - \mathbf{p_2}| - d)\frac{\mathbf{p_1} - \mathbf{p_2}}{|\mathbf{p_1} - \mathbf{p_2}|}$$

$$\Delta\mathbf{p_2} = +(|\mathbf{p_1} - \mathbf{p_2}| - d)\frac{\mathbf{p_1} - \mathbf{p_2}}{|\mathbf{p_1} - \mathbf{p_2}|}$$

to satisfy the constraint, we project the particle in the valid position

Example: Distance constraint

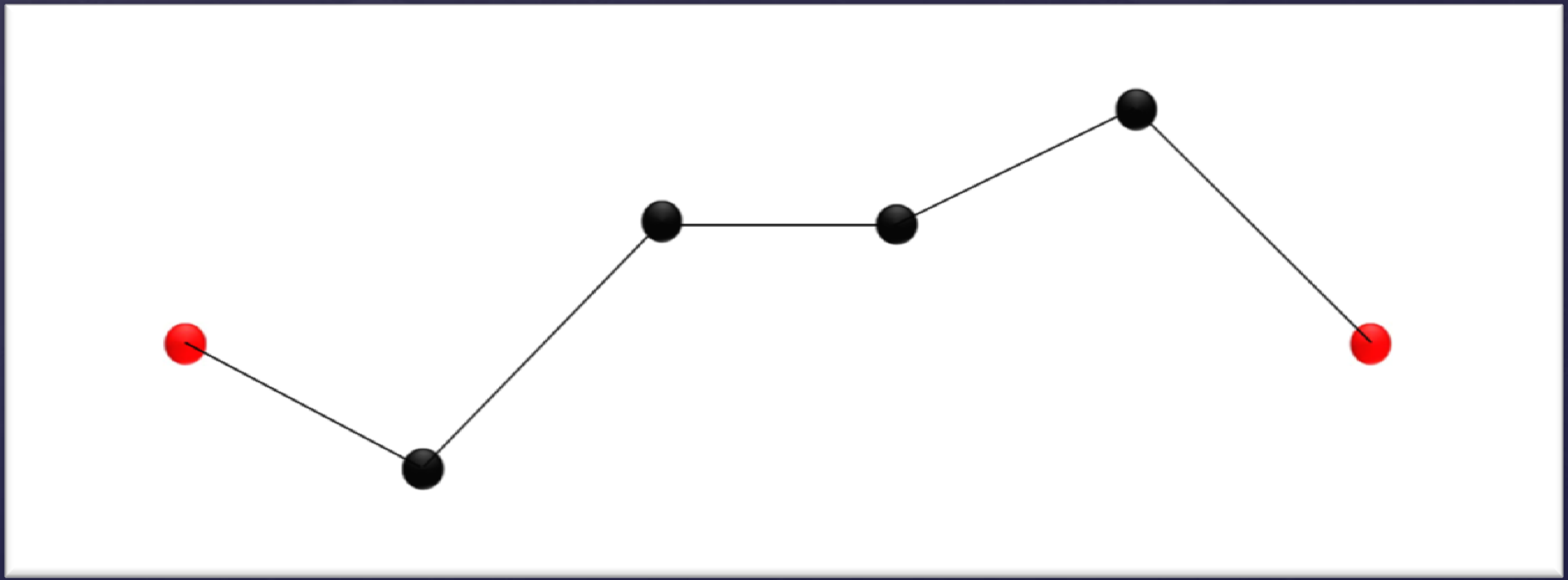
Approximate solution to a system of constraints using **Gauss-Seidel** method



Rest position

Example: Distance constraint

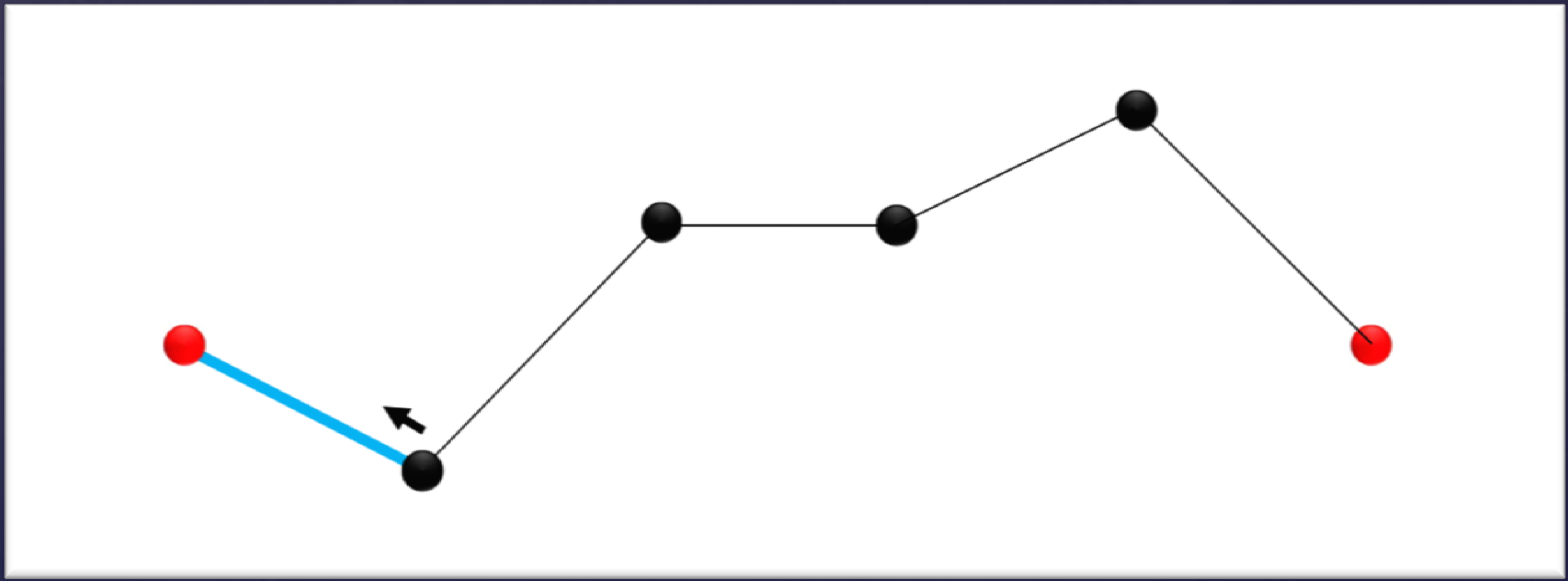
Approximate solution to a system of constraints using **Gauss-Seidel** method



The particles are displaced by external forces

Example: Distance constraint

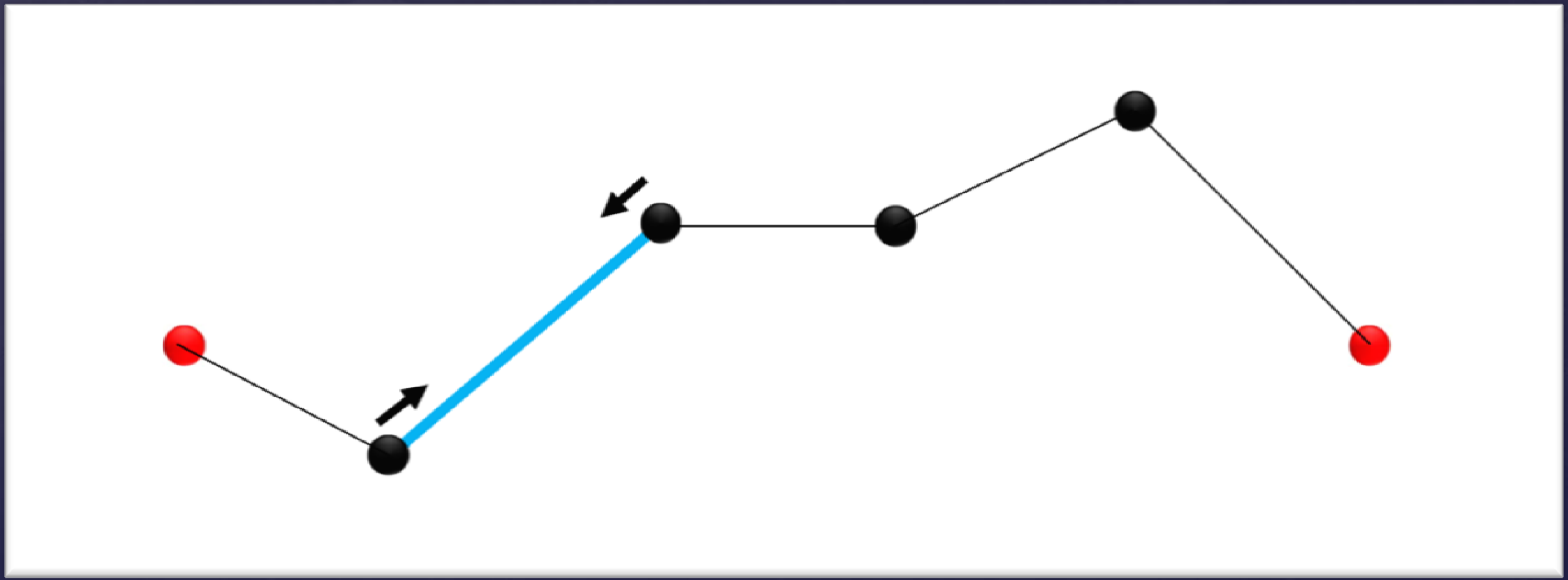
Approximate solution to a system of constraints using **Gauss-Seidel** method



The constraints are solved **sequentially**

Example: Distance constraint

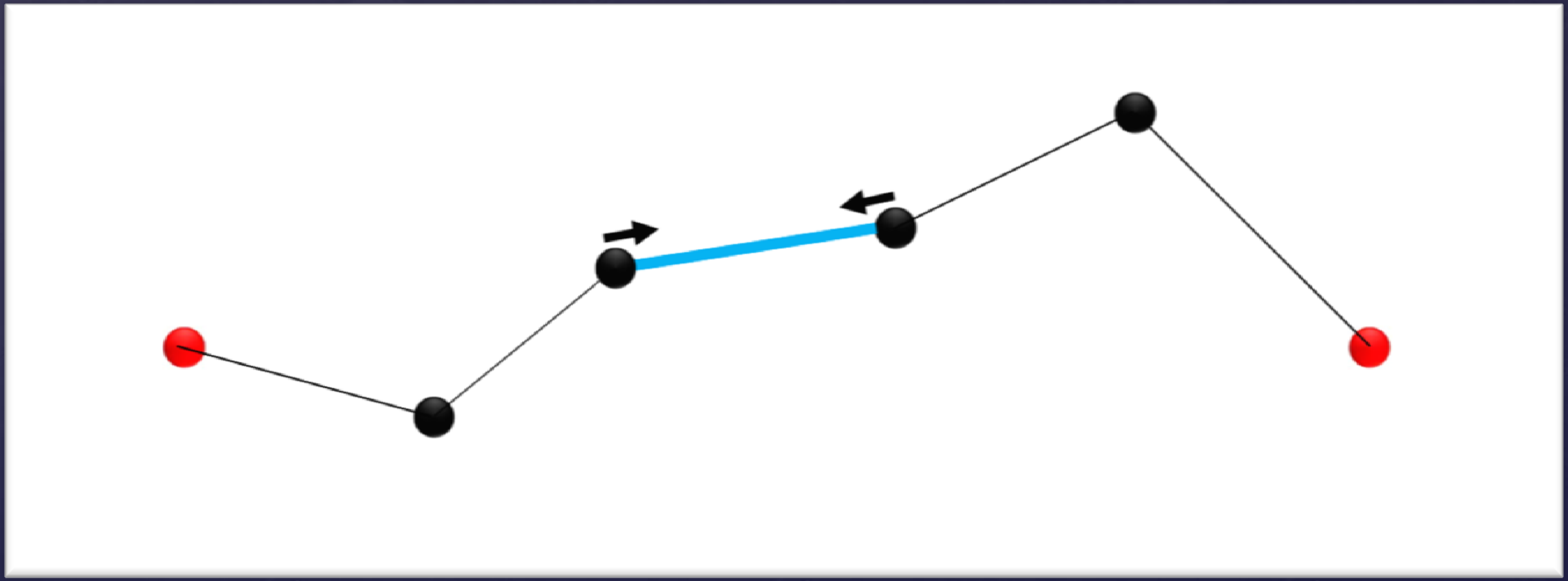
Approximate solution to a system of constraints using **Gauss-Seidel** method



The constraints are solved **sequentially**

Example: Distance constraint

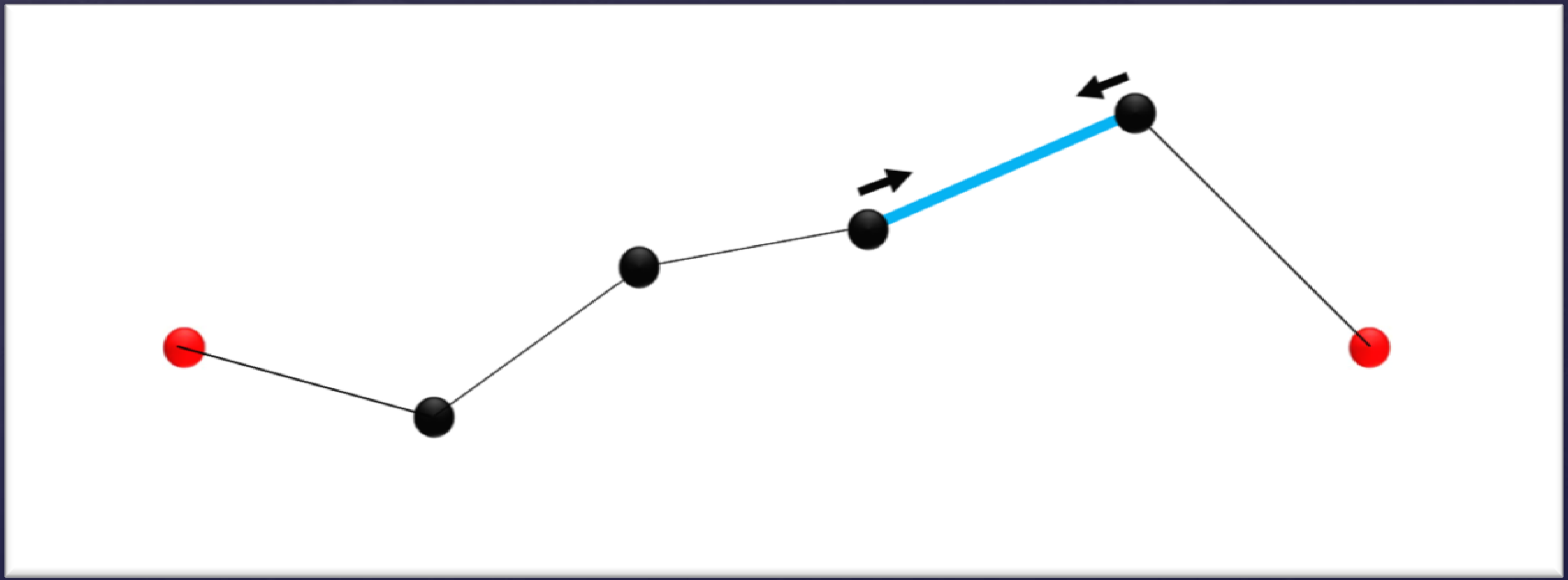
Approximate solution to a system of constraints using **Gauss-Seidel** method



The constraints are solved **sequentially**

Example: Distance constraint

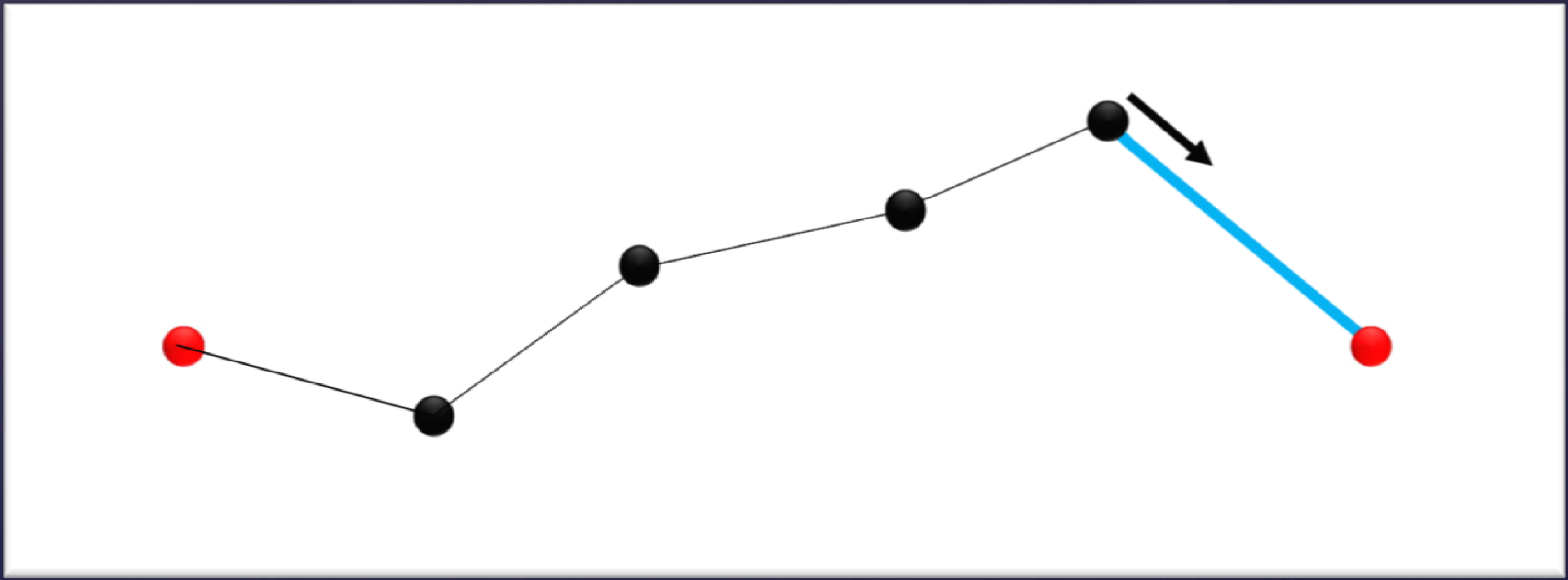
Approximate solution to a system of constraints using **Gauss-Seidel** method



The constraints are solved **sequentially**

Example: Distance constraint

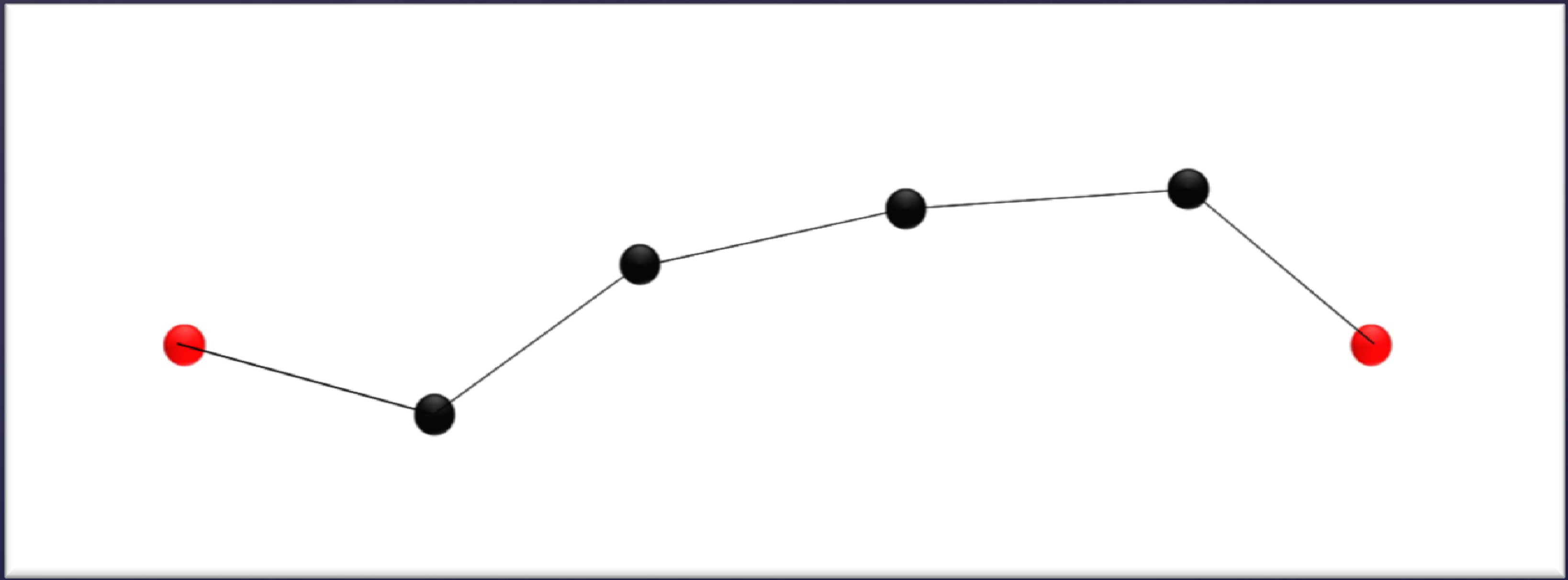
Approximate solution to a system of constraints using **Gauss-Seidel** method



The constraints are solved **sequentially**

Example: Distance constraint

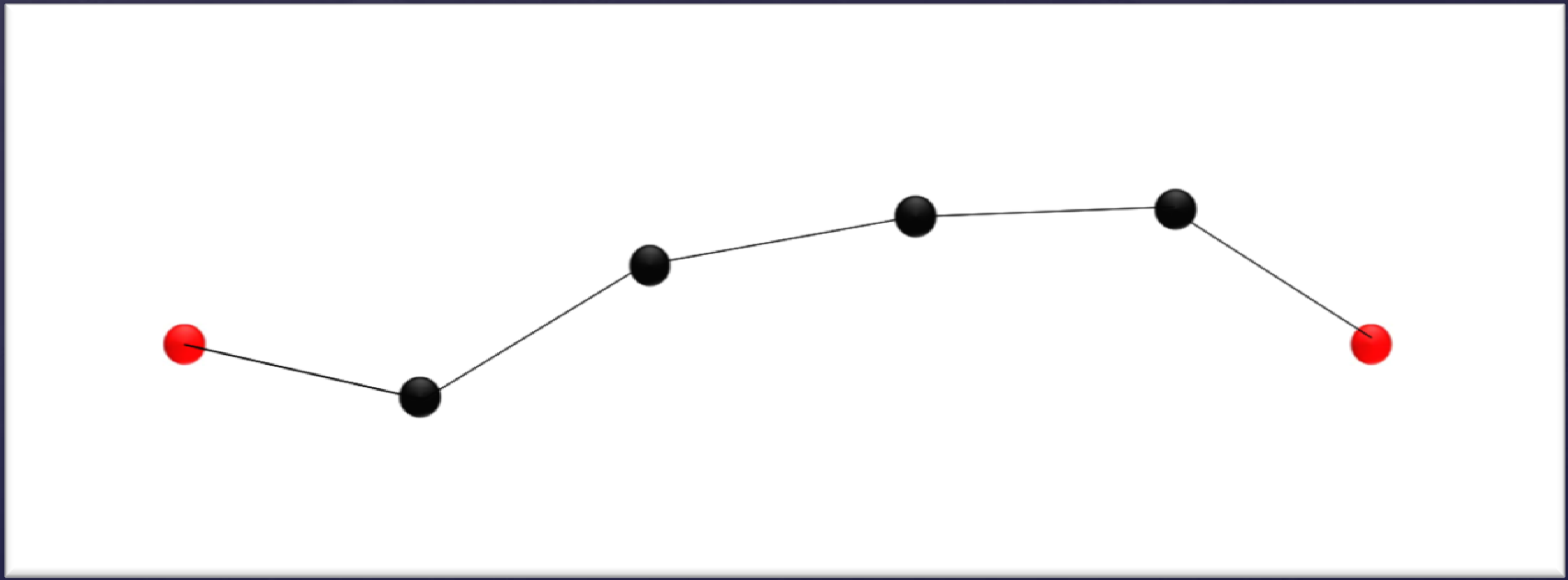
Approximate solution to a system of constraints using **Gauss-Seidel** method



Iteration 1

Example: Distance constraint

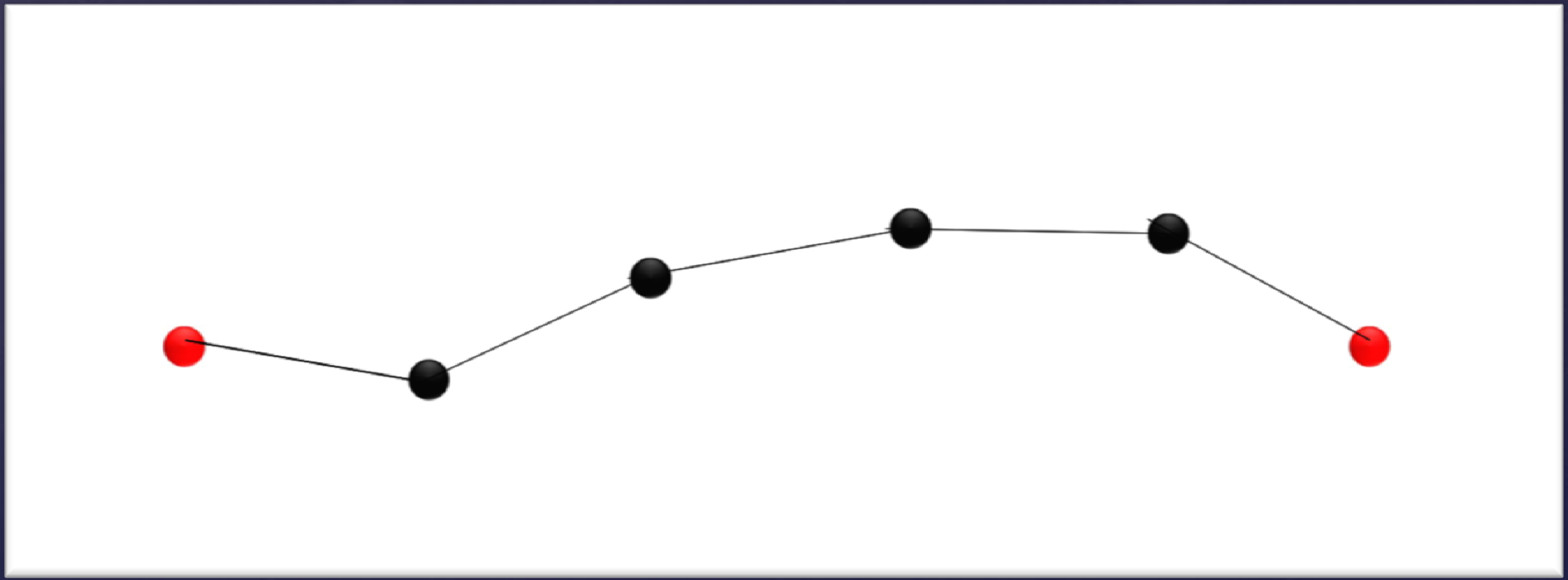
Approximate solution to a system of constraints using **Gauss-Seidel** method



Iteration 2

Example: Distance constraint

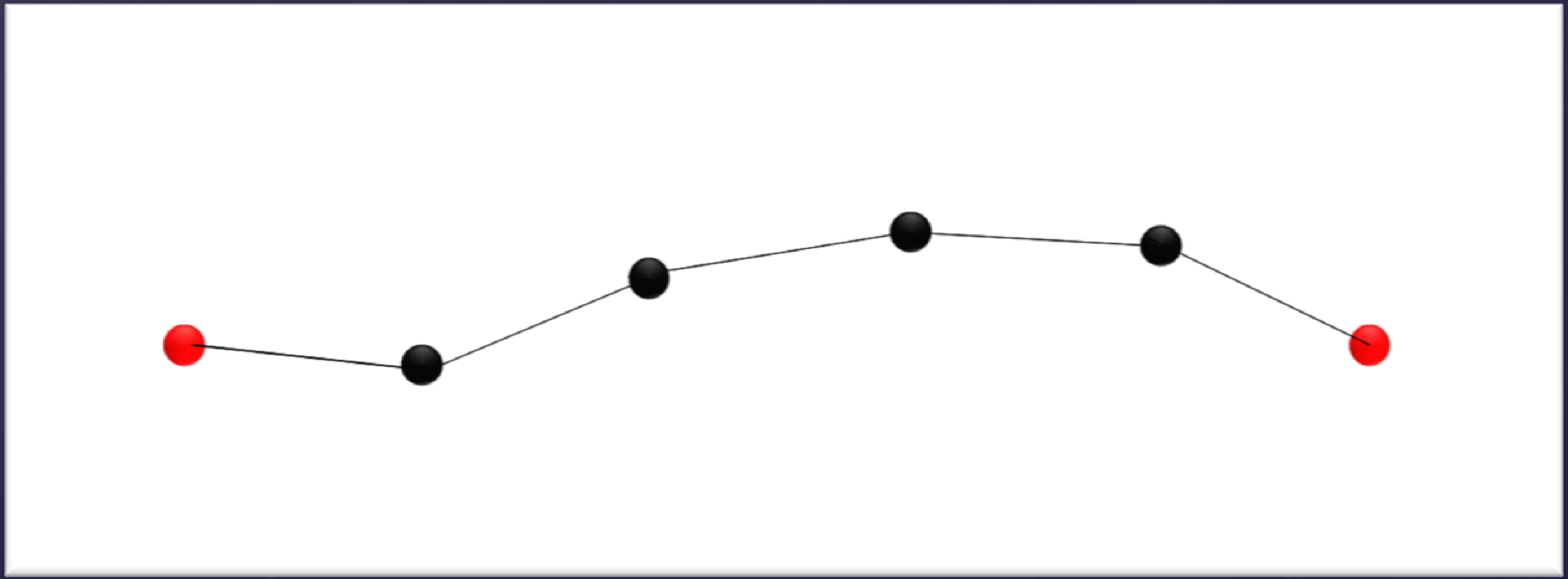
Approximate solution to a system of constraints using **Gauss-Seidel** method



Iteration 3

Example: Distance constraint

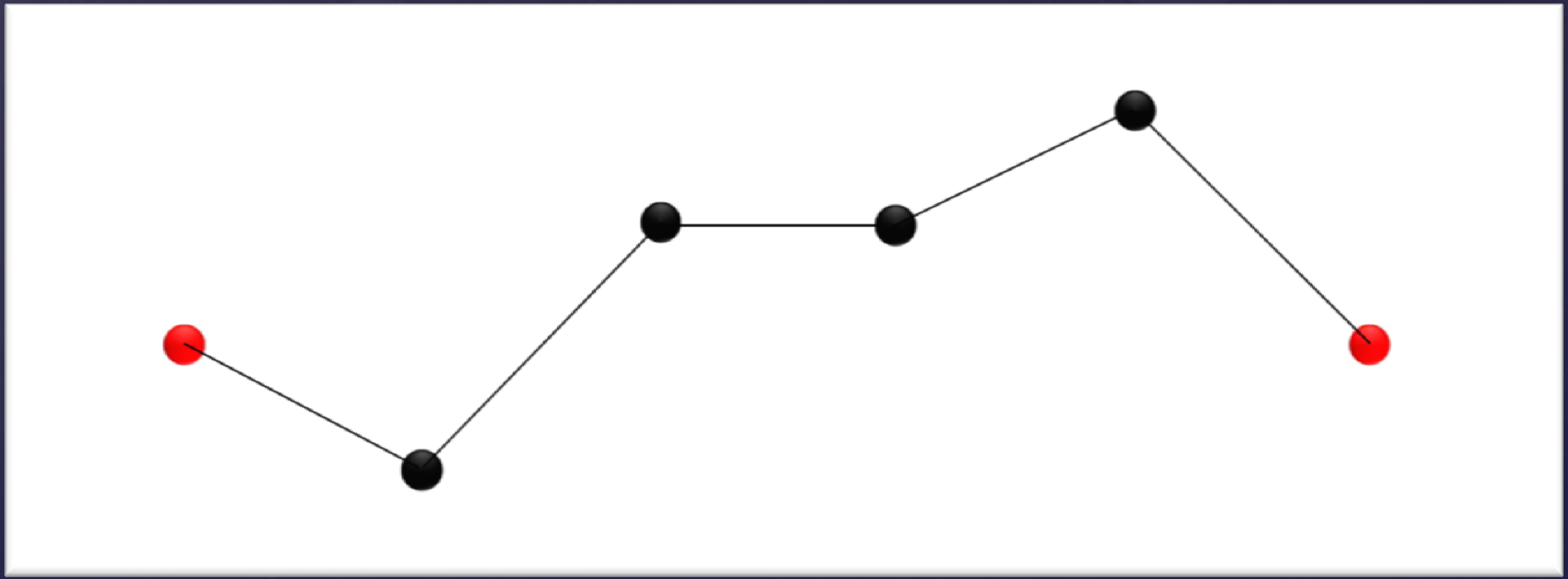
Approximate solution to a system of constraints using **Gauss-Seidel** method



Iteration 4

Example: Distance constraint

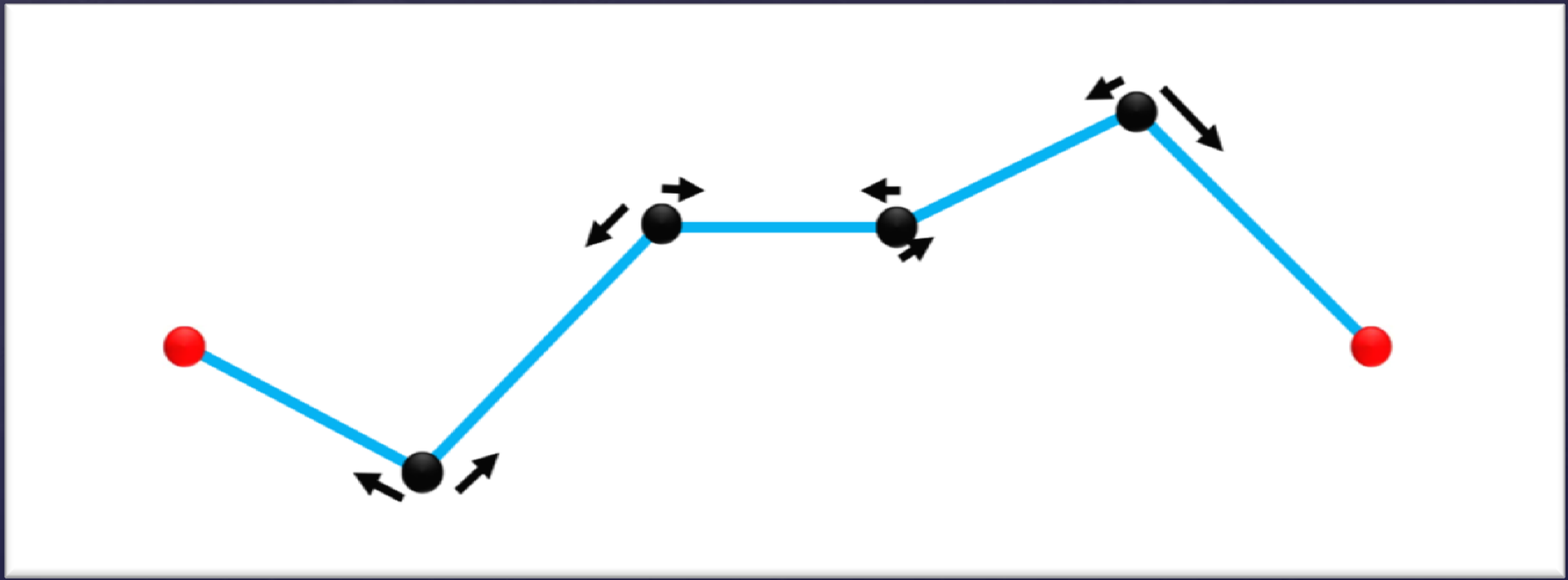
Approximate solution to a system of constraints using **Jacobi** method



The particles are displaced by external forces

Example: Distance constraint

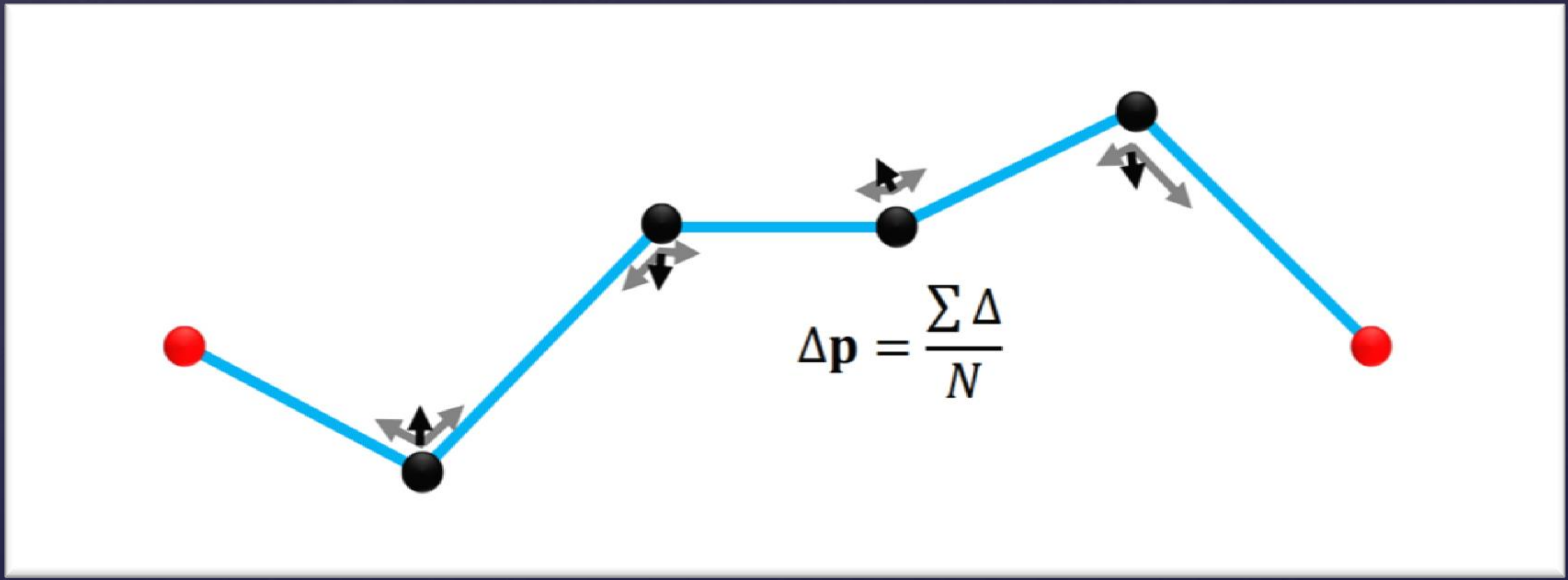
Approximate solution to a system of constraints using **Jacobi** method



The constraints are solved **in parallel**

Example: Distance constraint

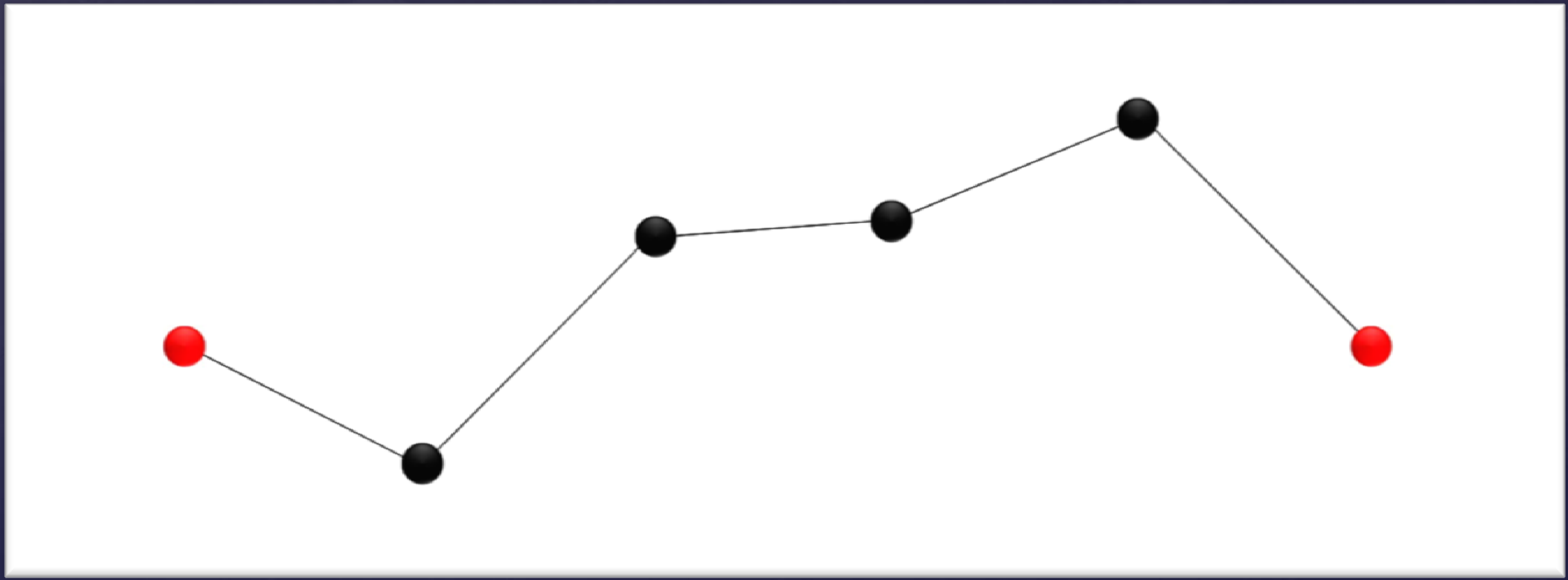
Approximate solution to a system of constraints using **Jacobi** method



The constraints are solved **in parallel**

Example: Distance constraint

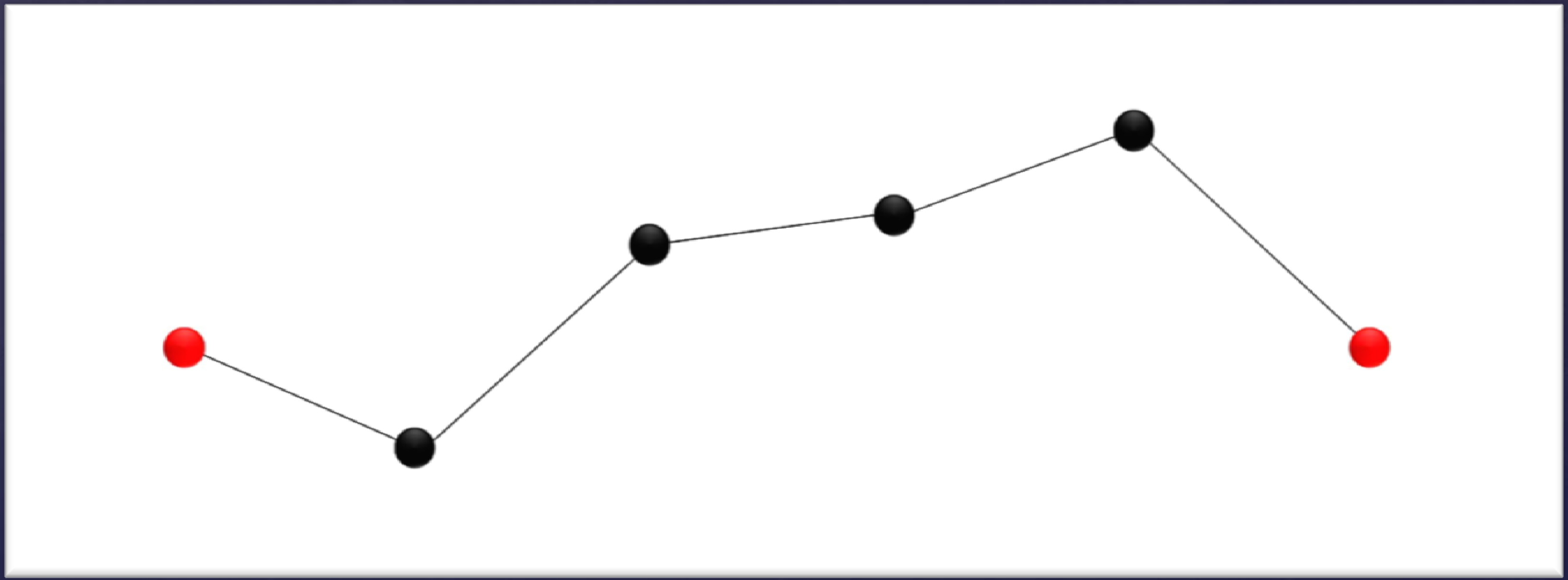
Approximate solution to a system of constraints using **Jacobi** method



Iteration 1

Example: Distance constraint

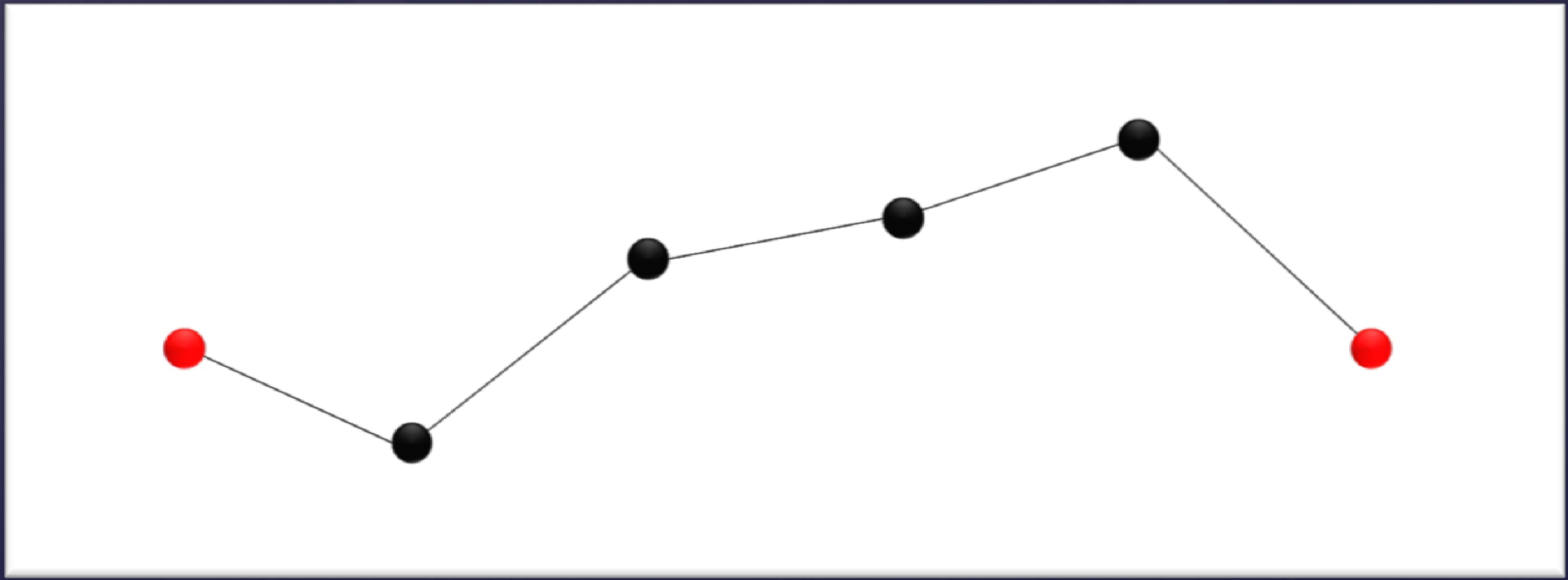
Approximate solution to a system of constraints using **Jacobi** method



Iteration 2

Example: Distance constraint

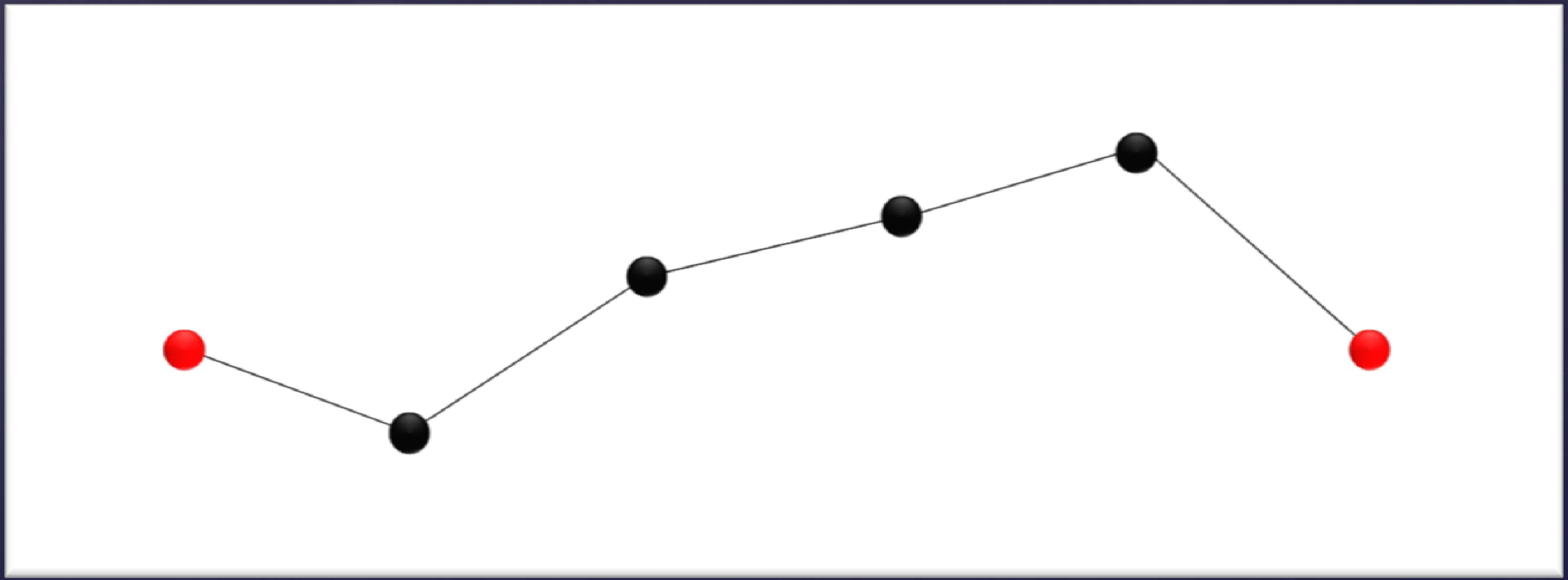
Approximate solution to a system of constraints using **Jacobi** method



Iteration 3

Example: Distance constraint

Approximate solution to a system of constraints using **Jacobi** method



Iteration 4

Gauss-Seidel vs Jacobi

- Gauss-Seidel converges much faster than Jacobi
  … but it is inherently serial

- Jacobi is trivially parallelizable
  … but the convergence rate is slow

Position Based Fluids

Finally, we have reached the actual paper by Macklin and Müller.

We want to have the nice looking results of SPH, but with the robustness and efficiency of PBD. Macklin and Müller do this by combining the two methods.

Position Based Fluids

In the paper:

1. Modify the PBD algorithm
2. Postulate constraint using SPH definitions
3. (Apply surface tension)
4. (Apply vorticity)
5. (Apply viscosity)
6. Rendering
7. Result

Modify the algorithm

PBD algorithm

forall vertices i
  initialize $\mathbf{x}_i = \mathbf{x}_i^0, \mathbf{v}_i = \mathbf{v}_i^0, w_i = 1/m_i$
endfor
loop
  forall vertices i **do** $\mathbf{v}_i \leftarrow \mathbf{v}_i + \Delta t w_i \mathbf{f}_{ext}(\mathbf{x}_i)$
  dampVelocities($\mathbf{v}_1, \ldots, \mathbf{v}_N$)
  forall vertices i **do** $\mathbf{p}_i \leftarrow \mathbf{x}_i + \Delta t \mathbf{v}_i$
  forall vertices i **do** generateCollisionConstraints($\mathbf{x}_i \rightarrow \mathbf{p}_i$)
  loop solverIterations **times**
    projectConstraints($C_1, \ldots, C_{M+M_{coll}}, \mathbf{p}_1, \ldots, \mathbf{p}_N$)
  endloop
  forall vertices i
    $\mathbf{v}_i \leftarrow (\mathbf{p}_i - \mathbf{x}_i)/\Delta t$
    $\mathbf{x}_i \leftarrow \mathbf{p}_i$
  endfor
  velocityUpdate($\mathbf{v}_1, \ldots, \mathbf{v}_N$)
endloop

PBF algorithm

for all particles i **do**
  apply forces $\mathbf{v}_i \Leftarrow \mathbf{v}_i + \Delta t \mathbf{f}_{ext}(\mathbf{x}_i)$
  predict position $\mathbf{x}_i^* \Leftarrow \mathbf{x}_i + \Delta t \mathbf{v}_i$
end for
for all particles i **do**
  find neighboring particles $N_i(\mathbf{x}_i^*)$
end for
while $iter < solverIterations$ **do**
  for all particles i **do**
    calculate λ_i
  end for
  for all particles i **do**
    calculate $\Delta \mathbf{p}_i$
    perform collision detection and response
  end for
  for all particles i **do**
    update position $\mathbf{x}_i^* \Leftarrow \mathbf{x}_i^* + \Delta \mathbf{p}_i$
  end for
end while
for all particles i **do**
  update velocity $\mathbf{v}_i \Leftarrow \frac{1}{\Delta t}(\mathbf{x}_i^* - \mathbf{x}_i)$
  apply vorticity confinement and XSPH viscosity
  update position $\mathbf{x}_i \Leftarrow \mathbf{x}_i^*$
end for

Constraint projection

The density constraint on the *ith* particle:

$$C_i(\mathbf{p}_1, \ldots, \mathbf{p}_n) = \frac{\rho_i}{\rho_0} - 1,$$

Where ρ_0 is the rest density and ρ_i is given by the SPH density:

$$\rho_i = \sum_j m_j W(\mathbf{p}_i - \mathbf{p}_j, h).$$

Constraint projection

PBD tries to find a particle position correction Δp that satisfy the constraint

$$C(\mathbf{p} + \Delta\mathbf{p}) = 0$$

Constraint projection

Doing some further math we end up with the position delta:

$$\Delta \mathbf{p}_i = \frac{1}{\rho_0} \sum_j (\lambda_i + \lambda_j) \nabla W(\mathbf{p}_i - \mathbf{p}_j, h).$$

Where lambda is the contstraint force:

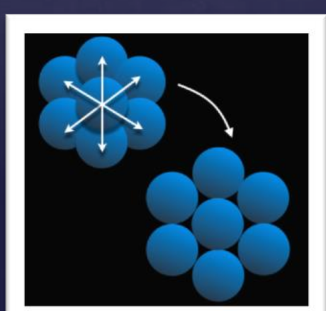
$$\lambda_i = -\frac{C_i(\mathbf{p}_1, \ldots, \mathbf{p}_n)}{\sum_k \left|\nabla_{\mathbf{p}_k} C_i\right|^2}$$

Constraint projection

Projection of the density constraint compared to the lenght constraint:

$$C_i(\mathbf{p}_1, \ldots, \mathbf{p}_n) = \frac{\rho_i}{\rho_0} - 1,$$

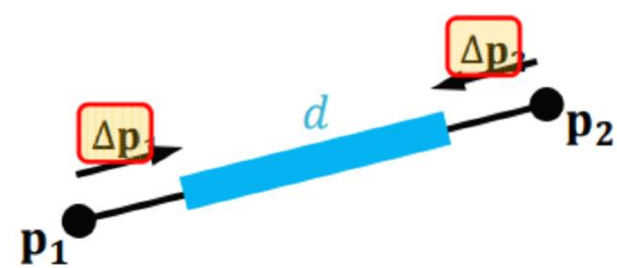
$$\Delta \mathbf{p}_i = \frac{1}{\rho_0} \sum_j (\lambda_i + \lambda_j) \nabla W(\mathbf{p}_i - \mathbf{p}_j, h).$$



$$C(\mathbf{p}_1, \mathbf{p}_2) = \|\mathbf{p}_1 - \mathbf{p}_2\| - d^2 = 0$$

$$\Delta \mathbf{p}_1 = -(|\mathbf{p}_1 - \mathbf{p}_2| - d)\frac{\mathbf{p}_1 - \mathbf{p}_2}{|\mathbf{p}_1 - \mathbf{p}_2|}$$

$$\Delta \mathbf{p}_2 = +(|\mathbf{p}_1 - \mathbf{p}_2| - d)\frac{\mathbf{p}_1 - \mathbf{p}_2}{|\mathbf{p}_1 - \mathbf{p}_2|}$$



Artificial surface tension

Projection of the density constraint compared to the lenght constraint:

$$s_{corr} = -k \left(\frac{W(\mathbf{p}_i - \mathbf{p}_j, h)}{W(\Delta \mathbf{q}, h)} \right)^n ,$$

Where Δq is a fixed constant inside the smoothing kernel

$$\Delta \mathbf{p}_i = \frac{1}{\rho_0} \sum_j (\lambda_i + \lambda_j + s_{corr}) \nabla W(\mathbf{p}_i - \mathbf{p}_j, h).$$

Vorticity and viscosity

Won't go into details

- Vorticity:

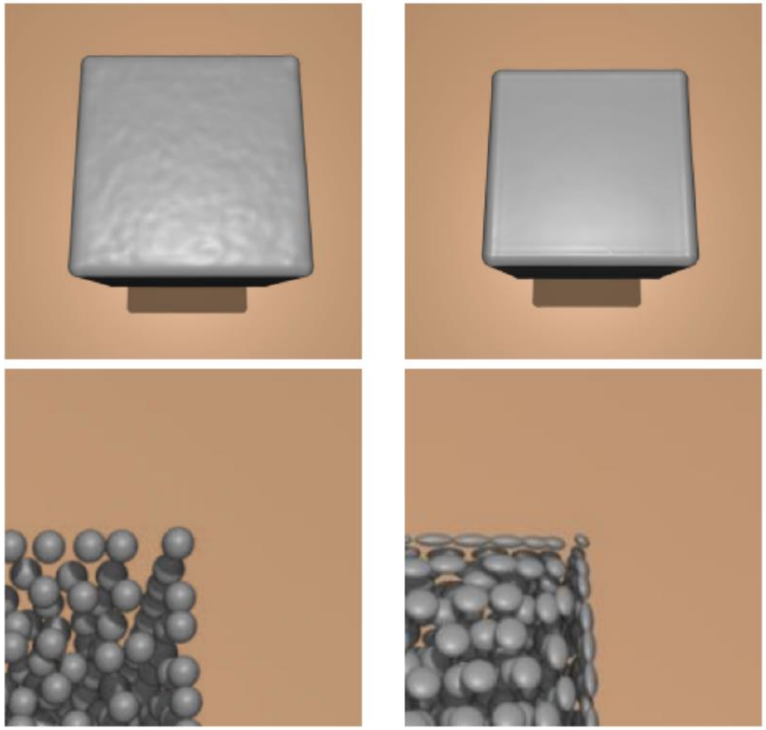
  PBD introduces damping which is unwanted. Macklin and Müller solves this by applying SPH vorticity.

- Viscosity:

  Apply SPH viscosity parameter to make the speed of the particles more coherent.

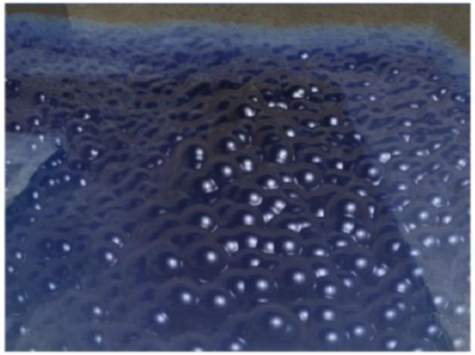
Rendering

1: Anisotropic kernels
[Yu and Turk 2013]

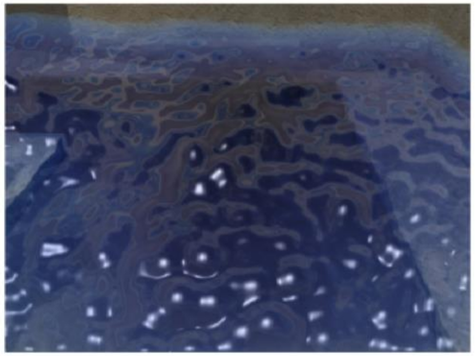


(a) (b)

2: Screen space curvature flow
[van der Laan et al. 2009]

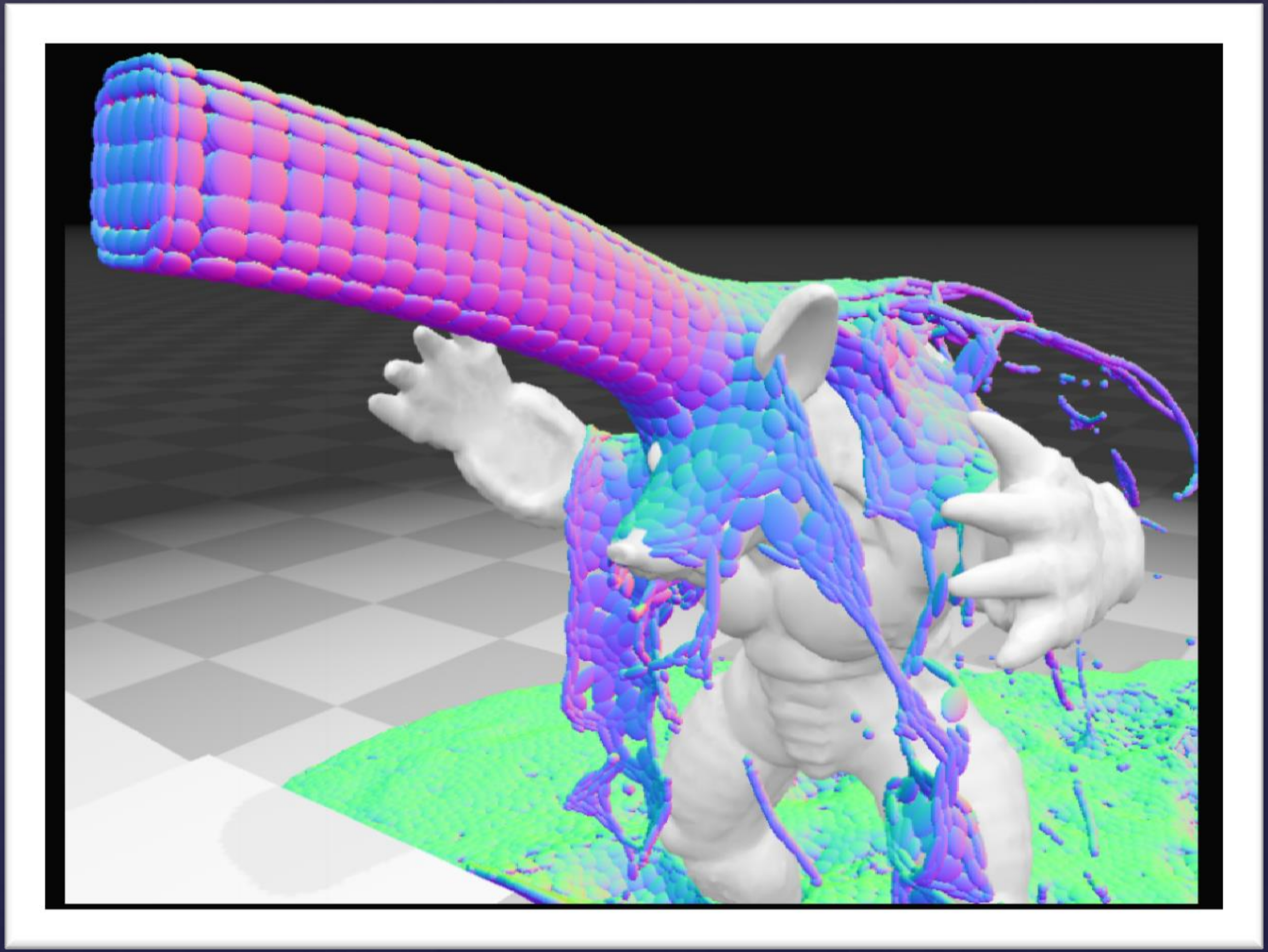


(a) Gaussian smoothing



(b) Screen-space curvature flow

Rendering

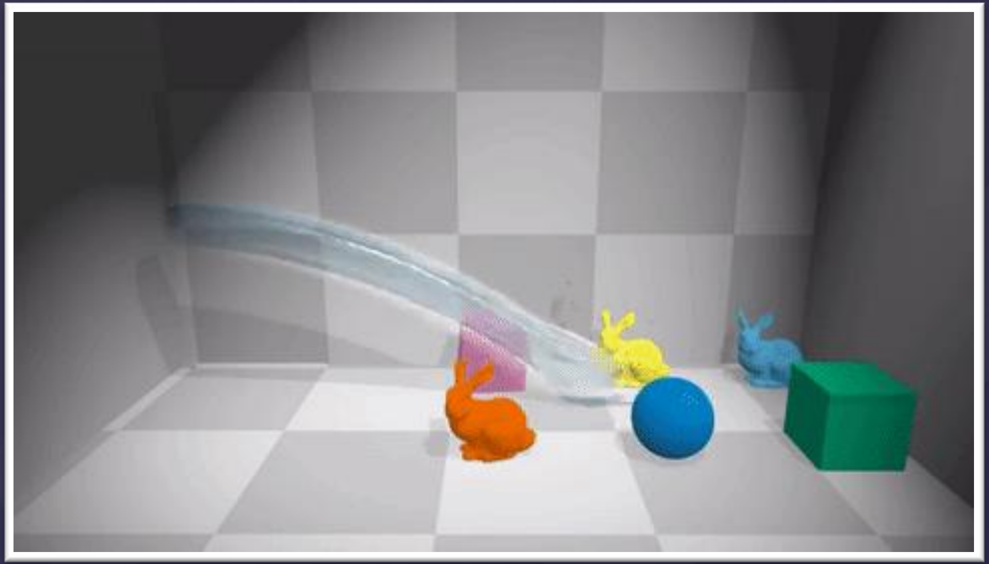


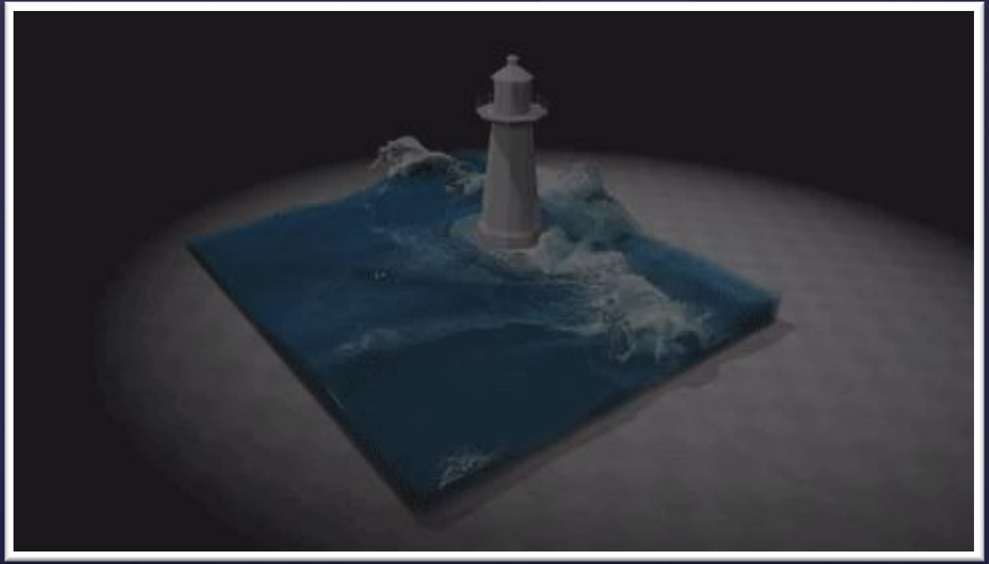
Result

Implemented using:

- 10 CUDA kernels
- GPU Hash Grid [Green 2008]
- Parallel Jacobi iteration
- 128k particles, 6 iterations, 10ms/frame on GTX680

Result





Sources

Papers:

Screen Space Fluid Rendering with Curvature Flow [van der Laan et al.]

Position-Based Simulation Methods in Computer Graphics [Bender et al.]

Reconstructing Surfaces of Particle-Based Fluids Using Anisotropic Kernels [Yu and Turk]

Position Based Fluids [Macklin and Müller]

A Survey on Position-Based Simulation Methods in Computer Graphics [Bender et al.]

Slides:

Smoothed Particle Hydrodynamics - Application Example [Alan Heirich]

Fluids in Games [Jim Van Verth]

Introduction to liquid animation and rendering [Marco Fratarcangeli]

Position based dynamics [Marco Fratarcangeli]

Position Based Fluids [Macklin and Müller]