Mass-Spring Systems – Part 2
Schedule for the next few weeks

<table>
<thead>
<tr>
<th>January</th>
<th>14</th>
<th>19</th>
<th>21</th>
<th>26</th>
<th>28</th>
</tr>
</thead>
<tbody>
<tr>
<td>February</td>
<td>2</td>
<td>4</td>
<td>9</td>
<td>11</td>
<td>16</td>
</tr>
</tbody>
</table>

A1 will be out tonight


Start thinking of topics for the final project. Form teams. Talk to us as soon as possible.
Backward Euler – from last class

- Boils down to solving systems of linear equations:

\[
\begin{pmatrix}
M - h \frac{\partial F}{\partial v} - h^2 \frac{\partial F}{\partial x}
\end{pmatrix}
\Delta v = M \left(v_n - v^k\right) + hF
\]

- Matrix A is large, sparse, symmetric, (sometimes positive definite)
  - these characteristics will inform the choice of algorithm we can/should use to solve the systems of equations
Some solvers only work if $A$ is symmetric positive definite:

$$v^tAv > 0 \forall v \neq 0$$

Think of a quadratic energy function (e.g. potential energy stored in spring):

Positive Definite  

Negative Definite  

Indefinite
Analogy: Compressed Springs

compressive force $-F$

$v_1^t A v_1 > 0$

$v_2^t A v_2 < 0$

$A$ is indefinite, we are at a saddle point!
How can you tell which way particle should go?
Solving linear systems

\[ Ax = b \]

Direct Methods:

- Explicitly compute inverse (e.g. via Gaussian Elimination)
- Decompose \( A \) (LU, LDL’, etc), solve by exploiting structure
Solving linear systems

LU decomposition:

\[ A = LU \]

\[
\begin{bmatrix}
  a_{11} & a_{12} & a_{13} \\
  a_{21} & a_{22} & a_{23} \\
  a_{31} & a_{32} & a_{33}
\end{bmatrix}
= \begin{bmatrix}
  l_{11} & 0 & 0 \\
  l_{21} & l_{22} & 0 \\
  l_{31} & l_{32} & l_{33}
\end{bmatrix}
\begin{bmatrix}
  u_{11} & u_{12} & u_{13} \\
  0 & u_{22} & u_{23} \\
  0 & 0 & u_{33}
\end{bmatrix}.
\]

\[ Ax = LUx = Ly = b \]
\[ Ux = y \]
Solving linear systems

- If $A$ is symmetric, LU decomposition is unique, and is called Cholesky decomposition:

$$ A = LDL^T $$

$$\begin{pmatrix} 4 & 12 & -16 \\ 12 & 37 & -43 \\ -16 & -43 & 98 \end{pmatrix} = \begin{pmatrix} 1 & 3 & 1 \\ 3 & 1 & 9 \\ 1 & 5 & 1 \end{pmatrix} \begin{pmatrix} 1 & 3 & -4 \\ 1 & 5 & 1 \end{pmatrix}$$

```cpp
//compute x = A^-1 * b
Eigen::SimplicialLDLT<SparseMatrix> solver;
solver.compute(A);
x = solver.solve(b);
```
Solving linear systems

\[ Ax = b \]

- **Direct Methods:**
  - Gaussian Elimination
  - decompose \( A \) (LU, LDL’, etc), solve by exploiting structure
  - Exact solution \( \sim O(n^3) \) for dense matrices, constant varies
Solving linear systems

\[ Ax = b \]

- **Indirect Methods:**
  - Iteratively improve approximate solution \( x^{k+1} \)
    - Can terminate when result is “good enough”
  - Gauss-Seidel & the Jacobi Method
Solving linear systems

**Gauss-Seidel**

\[ A = L_* + U \quad \text{where} \quad L_* = \begin{bmatrix}
    a_{11} & 0 & \cdots & 0 \\
    a_{21} & a_{22} & \cdots & 0 \\
    \vdots & \vdots & \ddots & \vdots \\
    a_{n1} & a_{n2} & \cdots & a_{nn}
\end{bmatrix}, \quad U = \begin{bmatrix}
    0 & a_{12} & \cdots & a_{1n} \\
    0 & 0 & \cdots & a_{2n} \\
    \vdots & \vdots & \ddots & \vdots \\
    0 & 0 & \cdots & 0
\end{bmatrix}. \]

\[ L_* x^{(k+1)} = b - U x^{(k)}, \]

\( x^{k+1} \) can be computed in place, only one storage vector required.
Solving linear systems

- Gauss-Seidel

\[ A = L_* + U \]

where

\[ L_* = \begin{bmatrix}
    a_{11} & 0 & \cdots & 0 \\
    a_{21} & a_{22} & \cdots & 0 \\
    \vdots & \vdots & \ddots & \vdots \\
    a_{n1} & a_{n2} & \cdots & a_{nn}
\end{bmatrix}, \quad U = \begin{bmatrix}
    0 & a_{12} & \cdots & a_{1n} \\
    0 & 0 & \cdots & a_{2n} \\
    \vdots & \vdots & \ddots & \vdots \\
    0 & 0 & \cdots & 0
\end{bmatrix}.\]

\[ L_* x^{(k+1)} = b - U x^{(k)}, \]

- \( x^{k+1} \) can be computed in place, only one storage vector required
- converges if \( A \) is symmetric positive-definite
- think of it as an iterative constraint solver
Solving linear systems

◆ Jacobi Method

\[ A = D + R \quad \text{where} \quad D = \begin{bmatrix} a_{11} & 0 & \cdots & 0 \\ 0 & a_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{nn} \end{bmatrix} \quad \text{and} \quad R = \begin{bmatrix} 0 & a_{12} & \cdots & a_{1n} \\ a_{21} & 0 & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & 0 \end{bmatrix}. \]

\[ x^{(k+1)} = D^{-1}(b - Rx^{(k)}), \]

◆ \( x^{k+1} \) cannot be computed in place
◆ equivalent to solving each equation independently
◆ parallelizable
Solving linear systems

\[ Ax = b \]

**Indirect Methods:**
- Iteratively improve approximate solution \( x^{k+1} \)
  - Can terminate when result is “good enough”
- Gauss-Seidel & the Jacobi Method
- Gradient Descent & Conjugate Gradient Method
Solving linear systems

- Gradient Descent:

\[
\begin{align*}
\min_x & \quad \frac{1}{2} x^T A x - x^T b \\
& \quad r^k = b - A x^k \\
& \quad x^{k+1} = x^k + \alpha r^k
\end{align*}
\]

- Slow convergence, too much backtracking…
The Conjugate Gradient Method

Main idea:
- find basis \((p_1, p_2, \ldots)\) of conjugate search directions
  (orthogonal with respect to generalized dot product \(a^T A b = 0\))

- compute step \(\alpha\) (independently!) along each direction such that

\[
x = \sum \alpha_i p_i
\]

- Build basis iteratively. E.g., if first step was along direction \(p_1\) and gradient at step 2 is \(r_2 = \alpha A p_1 - b\), direction for step 2 is:

\[
p_2 = r_2 - \frac{p_1^T A r_2}{p_1^T A p_1}
\]
Solving linear systems

◆ Gradient Descent vs Conjugate Gradients

“An Introduction to the Conjugate Gradient Method Without the Agonizing Pain”
- Jonathan Richard Shewchuk
Solving linear systems

\[ Ax = b \]

**Indirect Methods:**
- Iteratively improve approximate solution
  - Can terminate when result is “good enough”
- Gauss-Seidel & the Jacobi Method
- Gradient Descent & Conjugate Gradient Method
- Some methods do not require matrix to be explicitly built
Questions so far?
Assignment 1 – the fun part!

- How would you model...
  - cloth
Assignment 1 – the fun part!

How would you model…

- cloth

What types of springs are required?

Structural Springs

Diagonal Springs

Stretching

Shearing
Assignment 1 – the fun part!

- How would you model…
  - shells
Assignment 1 – the fun part!

How would you model…
• shells

What types of springs are required?

- Structural Springs
- Diagonal Springs
- Interleaved Springs
- Stretching
- Shearing
- Bending
Assignment 1 – the fun part!

- How would you model...
  - fur and hairs
Assignment 1 – the fun part!

- How would you model…
  - contacts and friction
Simple Collision Response

- If in contact, project back on surface, find normal $n$
  - For ground, $n=(0,1,0)$
- Filter velocities. First, decompose into
  - normal component $v_N = (v \cdot n)n$ and
  - tangential component $v_T = v - v_N$
- Normal response: $v_{N,\text{after}} = -\varepsilon v_{N,\text{before}}$, $\varepsilon \in [0,1]$
  - $\varepsilon=0$ is fully inelastic
  - $\varepsilon=1$ is elastic
- Tangential response
  - Simple model of friction: $v_{T,\text{after}} = \alpha v_{T,\text{before}}$, $\alpha \in [0,1]$
- Then reassemble velocity $v=v_N+v_T$
Assignment 1 – the fun part!

How would you model…
- a squishy object
Assignment 1 – the fun part!

- How would you model…
  - plastic deformations
Assignment 1 – the fun part!

- How would you model…
  - viscous materials
Assignment 1 – the fun part!

- How would you model...
  - a rigid body
  - an articulated rigid body structure
Assignment 1 – the fun part!

- How would you model...
  - a tensegrity structure
Assignment 1 – the fun part!

How would you model…
- Fracture, cutting, etc
Start early. Ask questions. Have fun!!!