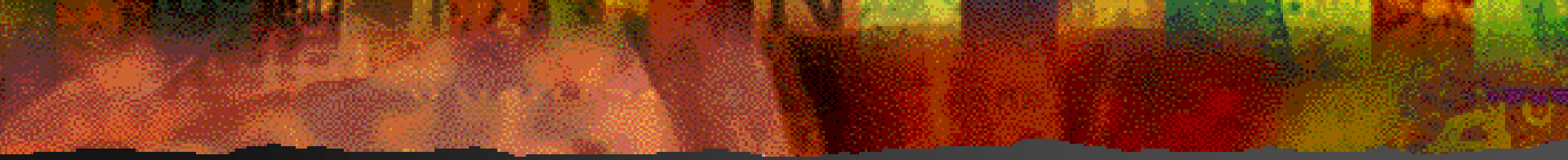


Cloth Simulation

CSE169: Computer Animation

Instructor: Steve Rotenberg

UCSD, Winter 2020

Cloth Simulation

- Cloth simulation has been an important topic in computer animation since the early 1980's
- It has been extensively researched, and has reached a point where it is *essentially* a solved problem
- Today, we will look at a very basic method of cloth simulation. It is relatively easy to implement and can achieve good results. It will also serve as an introduction to some more advanced cloth simulation topics.

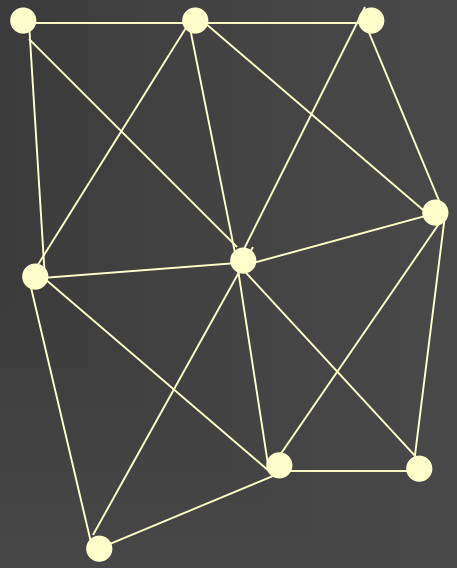
Cloth Simulation with Springs

- We will treat the cloth as a system of particles interconnected with spring-dampers
- Each spring-damper connects two particles, and generates a force based on their positions and velocities
- Each particle is also influenced by the force of gravity
- With those three simple forces (gravity, spring, & damping), we form the foundation of the cloth system
- Then, we can add some fancier forces such as aerodynamics, bending resistance, and collisions, plus additional features such as plastic deformation and tearing

Cloth Simulation

- Particle

Spring-damper



Particle

r : *position*

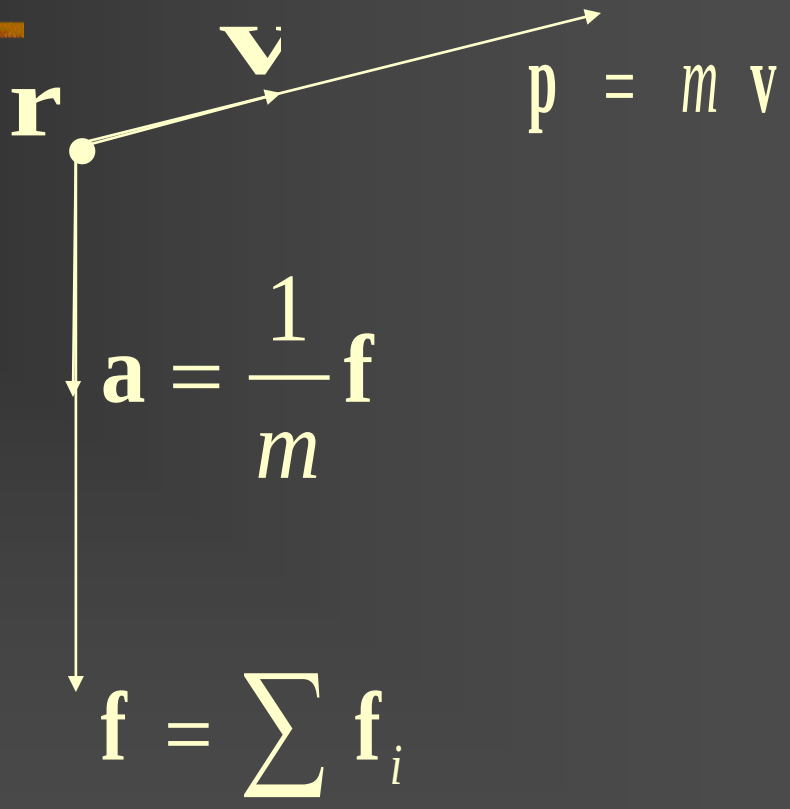
v : *velocity*

a : *acceleration*

m : *mass*

p : *momentum*

f : *force*



Euler Integration

- Once we've computed all of the forces in the system, we can use Newton's Second Law (f=ma) to compute the acceleration

$$\mathbf{a}_n = \frac{1}{m}\mathbf{f}_n$$

- Then, we use the acceleration to advance the simulation forward by some time step Δt, using the simple Euler integration scheme

$$\mathbf{v}_{n+1} = \mathbf{v}_n + \mathbf{a}_n \Delta t$$

$$\mathbf{r}_{n+1} = \mathbf{r}_n + \mathbf{v}_{n+1} \Delta t$$

Physics Simulation

General Physics Simulation:

1. Compute forces
2. Integrate motion

- Repeat

Cloth Simulation

1. Compute Forces
   For each particle: Apply gravity
   For each spring-damper: Compute & apply forces
   For each triangle: Compute & apply aerodynamic forces

2. Integrate Motion
   For each particle: Apply forward Euler integration

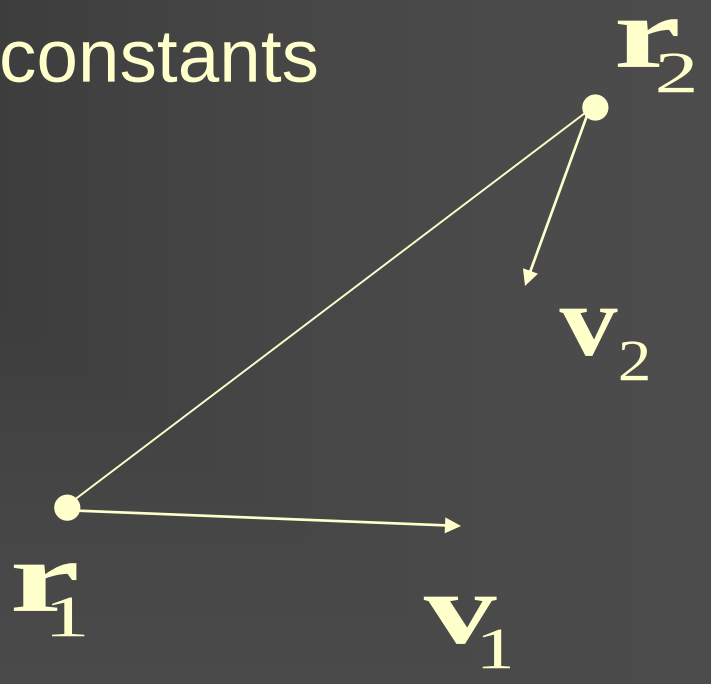
Uniform Gravity

$$\mathbf{f}_{gravity} = m\mathbf{g}_0$$

$$\mathbf{g}_0 = \begin{bmatrix} 0 & -9.8 & 0 \end{bmatrix} \quad \frac{m}{s^2}$$

Spring-Dampers

- The basic spring-damper connects two particles and has three constants defining its behavior
  - Rest length: l_0
  - Spring constant: k_s
  - Damping factor: k_d



Spring-Damper

- A simple spring-damper class might look like:

```
class SpringDamper {
    float SpringConstant,DampingFactor;
    float RestLength;
    Particle *P1,*P2;
public:
    void ComputeForce();
};
```

Spring-Dampers

- The basic linear spring force in one dimension is:

$$f_{spring} = -k_s x = -k_s \left(l_0 - l \right)$$

- The linear damping force is:

$$f_{damp} = -k_d v = -k_d \left(v_1 - v_2 \right)$$

- We can define a spring-damper by just adding the two:

$$f_{sd} = -k_s \left(l_0 - l \right) - k_d \left(v_1 - v_2 \right)$$

Spring-Dampers

- To compute the forces in 3D:
  - Turn 3D distances & velocities into 1D
  - Compute spring force in 1D
  - Turn 1D force back into 3D force

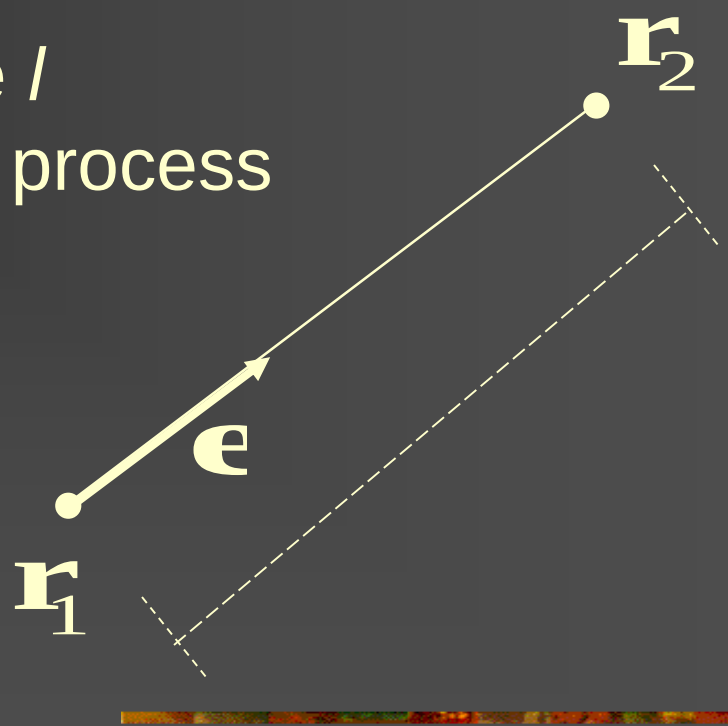
Spring-Damper Force

- We start by computing the unit length vector **e** from $\mathbf{r}_1$ to $\mathbf{r}_2$
- We can compute the distance *l* between the two points in the process

$$\mathbf{e}^* = \mathbf{r}_2 - \mathbf{r}_1$$

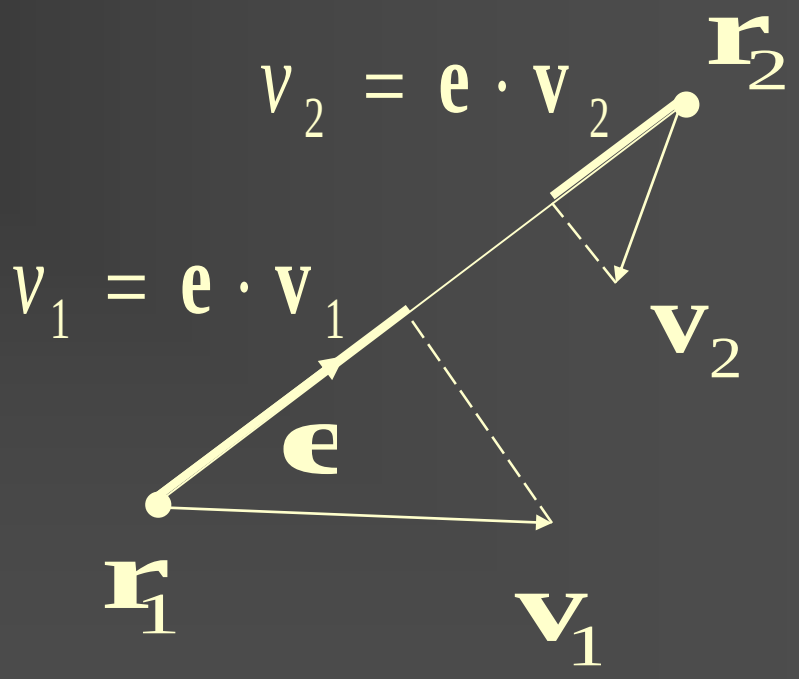
$$l = |\mathbf{e}^*|$$

$$\mathbf{e} = \frac{\mathbf{e}^*}{l}$$



Spring-Dampers

- Next, we find the 1D velocities



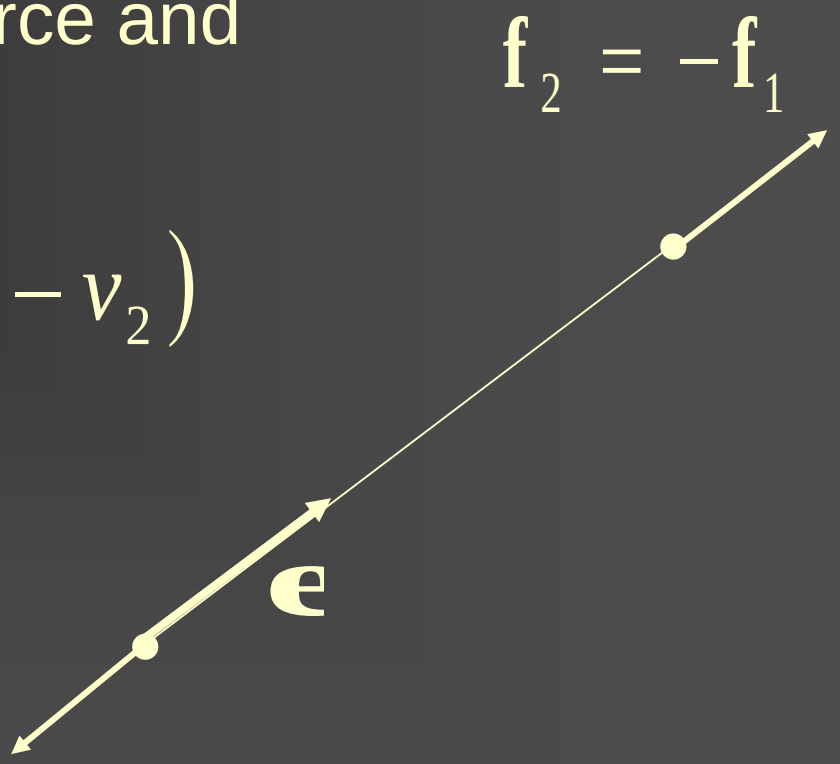
Spring-Dampers

- Now, we can find the 1D force and map it back into 3D

$$f_{sd} = -k_s\left(l_0 - l\right) - k_d\left(v_1 - v_2\right)$$

$$\mathbf{f}_1 = f_{sd}\mathbf{e}$$

$$\mathbf{f}_2 = -\mathbf{f}_1$$



Aerodynamic Force

- In the last lecture, we defined a simple aerodynamic drag force on an object as:

$$\mathbf{f}_{aero} = \frac{1}{2}\rho|\mathbf{v}|^2 c_d a\,\mathbf{e} \qquad \mathbf{e} = -\frac{\mathbf{v}}{|\mathbf{v}|}$$

ρ: density of the air (or water…)

c_d: coefficient of drag for the object

a: cross sectional area of the object

e: unit vector in the opposite direction of the velocity

Aerodynamic Force

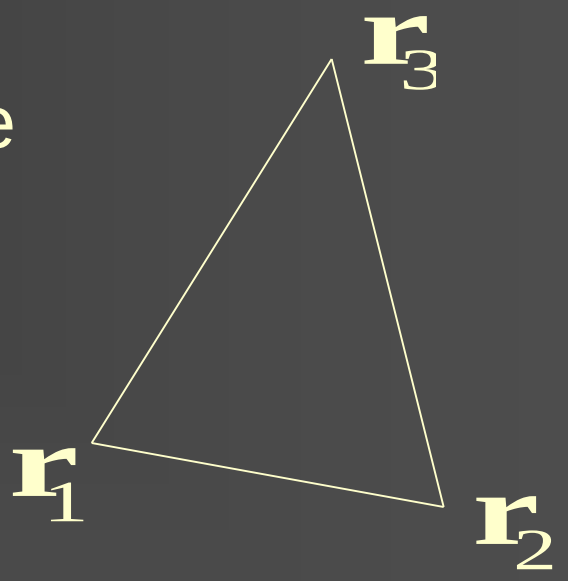
- Today we will extend that to a simple flat surface
- Instead of opposing the velocity, the force pushes against the normal of the surface

$$\mathbf{f}_{aero} = -\frac{1}{2}\rho|\mathbf{v}|^2 c_d a \mathbf{n}$$

- Note: This is a major simplification of real aerodynamic interactions, but it's a good place to start

Aerodynamic Force

- In order to compute the aerodynamic forces, we need surfaces to apply it to
- We will add some triangles to our cloth definition, where each triangle connects three particles

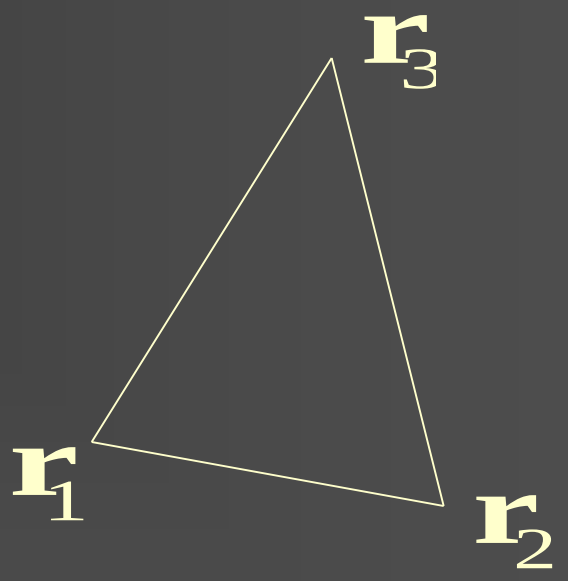


Aerodynamic Force

- In order to compute our force:

$$\mathbf{f}_{aero} = -\frac{1}{2}\rho|\mathbf{v}|^2 c_d a\mathbf{n}$$

we will need find the velocity, normal, and area of the triangle (we can assume that ρ and c_d are constants)



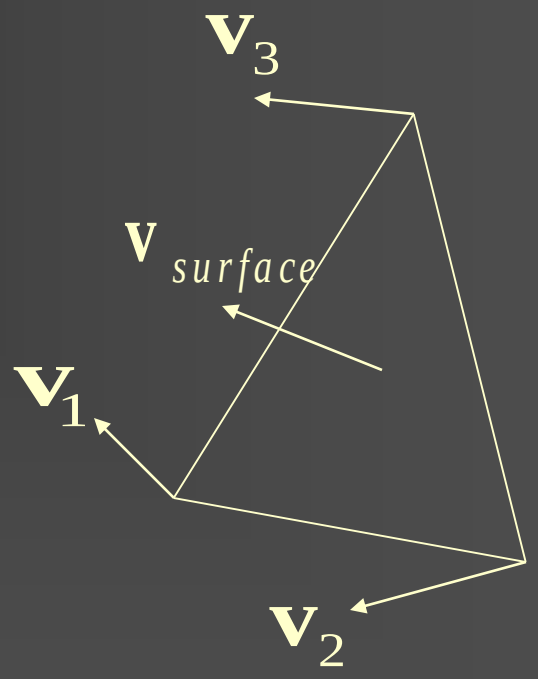
Aerodynamic Force

- For the velocity of the triangle, we can use the average of the three particle velocities

$$\mathbf{v}_{surface} = \frac{\mathbf{v}_1 + \mathbf{v}_2 + \mathbf{v}_3}{3}$$

- We actually want the relative velocity, so we will then subtract off the velocity of the air

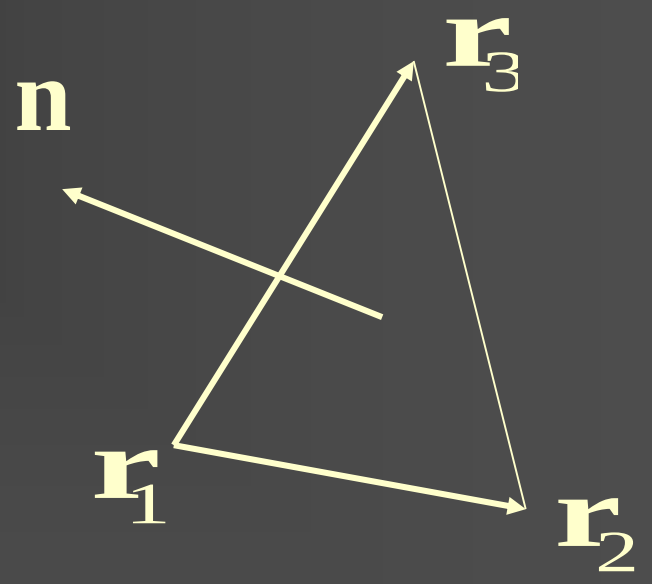
$$\mathbf{v} = \mathbf{v}_{surface} - \mathbf{v}_{air}$$



Aerodynamic Force

- The normal of the triangle is:

$$\mathbf{n} = \frac{(\mathbf{r}_2 - \mathbf{r}_1) \times (\mathbf{r}_3 - \mathbf{r}_1)}{\left|(\mathbf{r}_2 - \mathbf{r}_1) \times (\mathbf{r}_3 - \mathbf{r}_1)\right|}$$



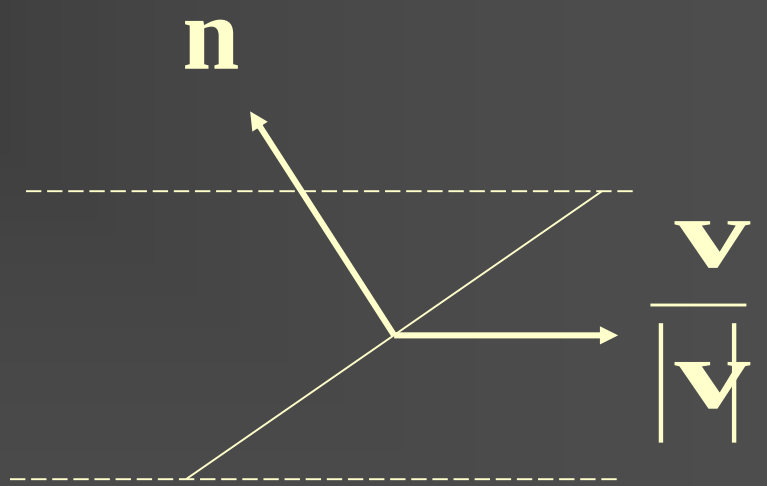
Aerodynamic Force

- The area of the triangle is:

$$a_0 = \frac{1}{2}\left|\left(\mathbf{r}_2 - \mathbf{r}_1\right) \times \left(\mathbf{r}_3 - \mathbf{r}_1\right)\right|$$

- But we really want the cross-sectional area (the area exposed to the air flow)

$$a = a_0 \frac{\mathbf{v} \cdot \mathbf{n}}{|\mathbf{v}|}$$

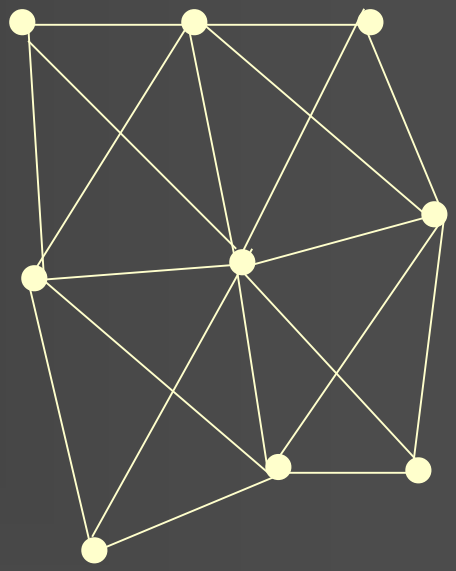


Aerodynamic Force

- The final aerodynamic force is assumed to apply to the entire triangle
- We can turn this into a force on each particle by simply dividing by 3, and splitting the total force between them

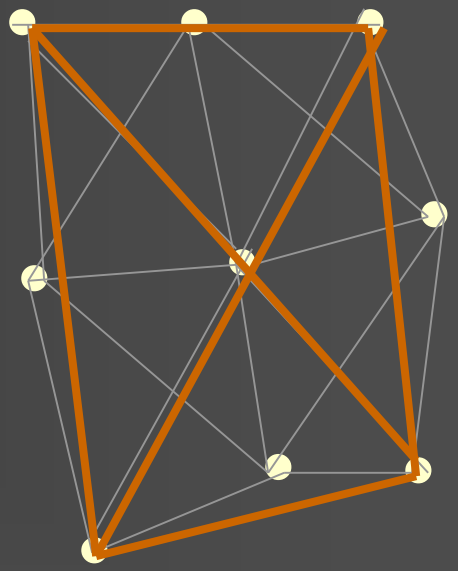
Bending Forces

- If we arrange our cloth springs as they are in the picture, there will be nothing preventing the cloth from bending
- This may be fine for simulating softer cloth, but for stiffer materials, we may want some resistance to bending



Bending Forces

- A simple solution is to add more springs, arranged in various configurations, such as the one in the picture
- The spring constants and damping factors of this layer might need to be tuned differently…



Collisions

- We will talk about collision detection & response in a later lecture…
- In the mean time, here's a very basic way to collide with a $y=y_0$ plane

```
If(r.y < y0) {
   r.y= 2*y0 - r.y;
   v.y= - elasticity * v.y;
   v.x= (1-friction) * v.x;          // cheezy
   v.z= (1-friction) * v.z;          // cheezy
}
```

Plastic Deformation

- An *elastic* deformation will restore back to its un-deformed state when all external forces are removed (such as the deformation in a spring, or in a rubber ball)
- A *plastic* deformation is a permanent adjustment of the material structure (such as the buckling of metal)

Plastic Deformation

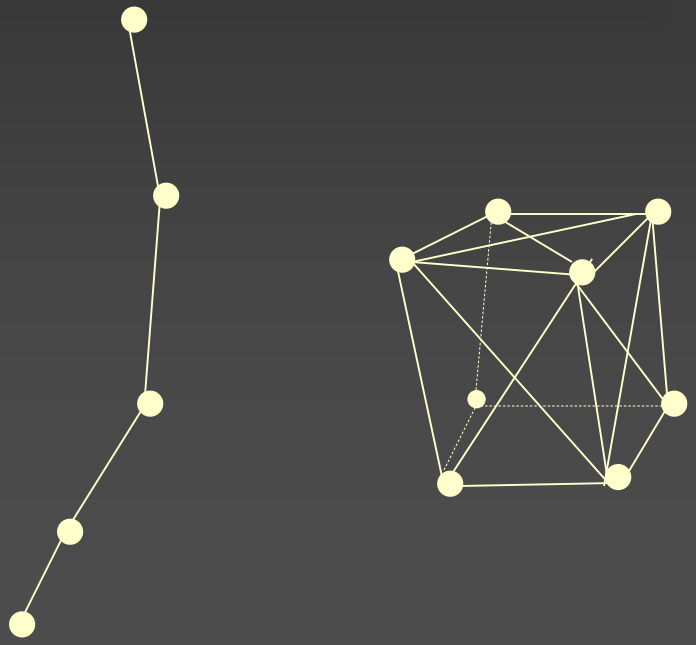
- We can add a simple plastic deformation rule to the spring-dampers
- We do so by modifying the rest length
- Several possible rules can be used, but one simple way is to start by defining an elastic limit and plastic limit
- The elastic limit is the maximum deformation distance allowed before a plastic deformation occurs
- If the elastic limit is reached, the rest length of the spring is adjusted so that meets the elastic limit
- An additional plastic limit prevents the rest length from deforming beyond some value
- The plastic limit defines the maximum distance we are allowed to move the rest length

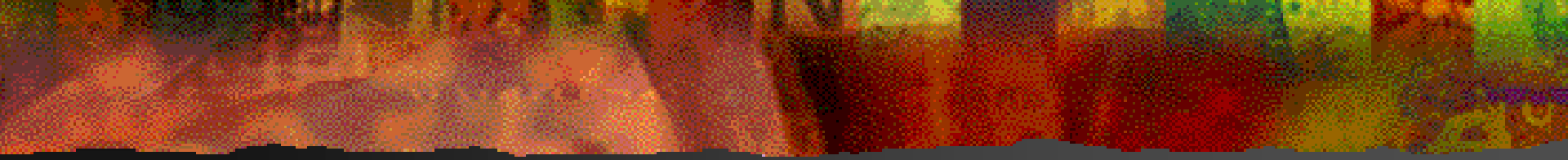
Fracture & Tearing

- We can also allow springs to break
- One way is to define a length (or percentage of rest length) that will cause the spring to break
- This can also be combined with the plastic deformation, so that fracture occurs at the plastic limit
- Another option is to base the breaking on the force of the spring (this will include damping effects)
- It’s real easy to break individual springs, but it may require some real bookkeeping to update the cloth mesh connectivity properly…

Ropes & Solids

- We can use this exact same scheme to simulate ropes, solids, and similar objects





System Stability

Conservation of Momentum

- As real springs apply equal and opposite forces to two points, they obey conservation of momentum
- Our simple spring-damper implementation should actually *guarantee* conservation of momentum, due to the way we explicitly apply the equal and opposite forces
- (This assumes that everything says within reasonable floating point ranges and we don't suffer from excessive round-off)

Conservation of Energy

- True linear springs also conserve energy, as the kinetic energy of motion can be stored in the deformation energy of the spring and later restored
- The dampers, however are specifically intended to remove kinetic energy from the system
- Our simple implementation using Euler integration is not guaranteed to conserve energy, as we never explicitly deal with it as a quantity

Conservation of Energy

- If we formulate the equations correctly and take small enough time steps, the system will hopefully conserve energy *approximately*
- In practice, we might see a gradual increase or decrease in system energy over time
- A gradual decrease of energy implies that the system damps out and might eventually come to rest. A gradual increase, however, it not so nice…

Conservation of Energy

- There are particle schemes that conserve energy, and other schemes that preserve momentum (and/or angular momentum)
- It’s possible to conserve all three, but it becomes significantly more complicated
- This is important in engineering applications, but less so in entertainment applications
- Also, as we usually want things to come to rest, we explicitly put in some energy loss through controlled damping
- Still, we want to make sure that our integration scheme is stable enough not to gain energy

Simulation Stability

- If the simulation ‘blows up’ due to artificial energy gains, then it is said to be unstable
- The basic Euler integration scheme is the simplest, but can easily become unstable and require very small time steps in order to produce useful results
- There are *many* other integration schemes that improve this behavior
- We will only briefly mention these now, but might go over them in more detail in a future lecture

Integration

- There are *many* methods of numerical integration. Some examples are:
  - Explicit Euler
  - Implicit Euler
  - Midpoint (Leapfrog)
  - Crank-Nicolson
  - Runge-Kutta
  - Adams-Bashforth, Adams-Moulton
  - etc…

Two-Level Integration Methods

- Explicit Euler: $\varphi^{n+1} = \varphi^{n} + f(t_n, \varphi^{n})\Delta t$
- Implicit Euler $\varphi^{n+1} = \varphi^{n} + f(t_{n+1}, \varphi^{n+1})\Delta t$
- Midpoint (Leapfrog): $\varphi^{n+1} = \varphi^{n} + f(t_{n+1/2}, \varphi^{n+1/2})\Delta t$
- Crank-Nicolson: $\varphi^{n+1} = \varphi^{n} + \frac{1}{2}\left(f(t_n, \varphi^{n}) + f(t_{n+1}, \varphi^{n+1})\right)\Delta t$

Multipoint Methods

- Multipoint methods fit a polynomial to several values in time. Adams-Bashforth methods use only previous values, while Adams-Moulton combine these with implicitly computed future points.
- Second order Adams-Bashforth:

$$\varphi^{n+1} = \varphi^{n} + \frac{\Delta t}{2}\left(3 f(t_n, \varphi^{n}) - f(t_{n-1}, \varphi^{n-1})\right)$$

- Third order Adams-Moulton:

$$\varphi^{n+1} = \varphi^{n} + \frac{\Delta t}{12}\left(5 f(t_{n+1}, \varphi^{n+1}) + 8 f(t_n, \varphi^{n}) - f(t_{n-1}, \varphi^{n-1})\right)$$

Runge-Kutta Methods

- The Runge-Kutta integration methods compute the value at step n+1 by computing several partial steps between n and n+1 and then constructing a polynomial to get the final value at n+1
- Second order Runge-Kutta:

$$\varphi^{n+1/2} = \varphi^{n} + \frac{\Delta t}{2} f(t_n, \varphi^{n})$$

$$\varphi^{n+1} = \varphi^{n} + \Delta t \cdot f(t_{n+1/2}, \varphi^{n+1/2})$$

Cloth Stability

- To make our cloth stable, we *should* choose a better integration scheme (such as an implicit scheme and/or using adaptive time-steps)
- It's actually not quite as bad as it sounds
- But, in the mean time, some other options include:
  - Oversampling: For one 1/60 time step, update the cloth several times at smaller time steps (say 10 times at 1/600), then draw once
  - Tuning numbers: High spring constants and damping factors will increase the instability. Lowering these will help, but will also make the cloth look more like rubber…

Advanced Cloth

Continuum Mechanics

- Real cloth simulation rarely uses springs
- Instead, forces are generated based on the the deformation of a triangular element
- This way, one can properly account for internal forces within the piece of cloth based on the theory of continuum mechanics
- The basic process is still very similar. Instead of looping through springs computing forces, one loops through the triangles and computes the forces
- Continuum models account for various properties such as elastic deformation, plastic deformation, bending forces, anisotropy, and more

Collision Detection & Response

- Cloth colliding with rigid objects is tricky
- Cloth colliding with itself is even trickier
- There have been several published papers on robust cloth collision detection and response methods