

Cloth Simulation

Steve Rotenberg

CSE169: Computer Animation

UCSD

Winter 2021

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 simulation topics

Cloth Simulation with Springs

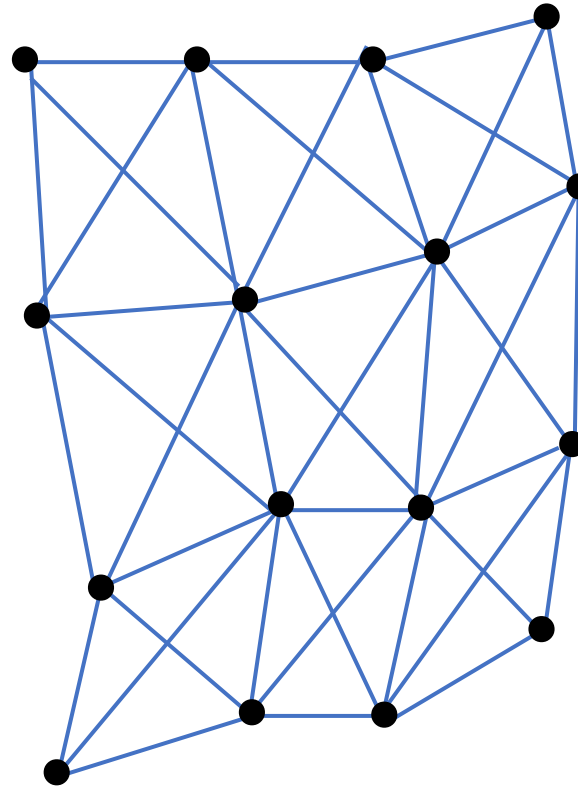
- We will model cloth as a system of particles interconnected with spring-dampers
- Each spring-damper connects two particles, and generates a force based on their positions and velocities
- Each particle is also influenced by the force of gravity
- With those three simple forces (gravity, spring, & 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 with Springs

Particle



Spring-damper



Particle

r : position

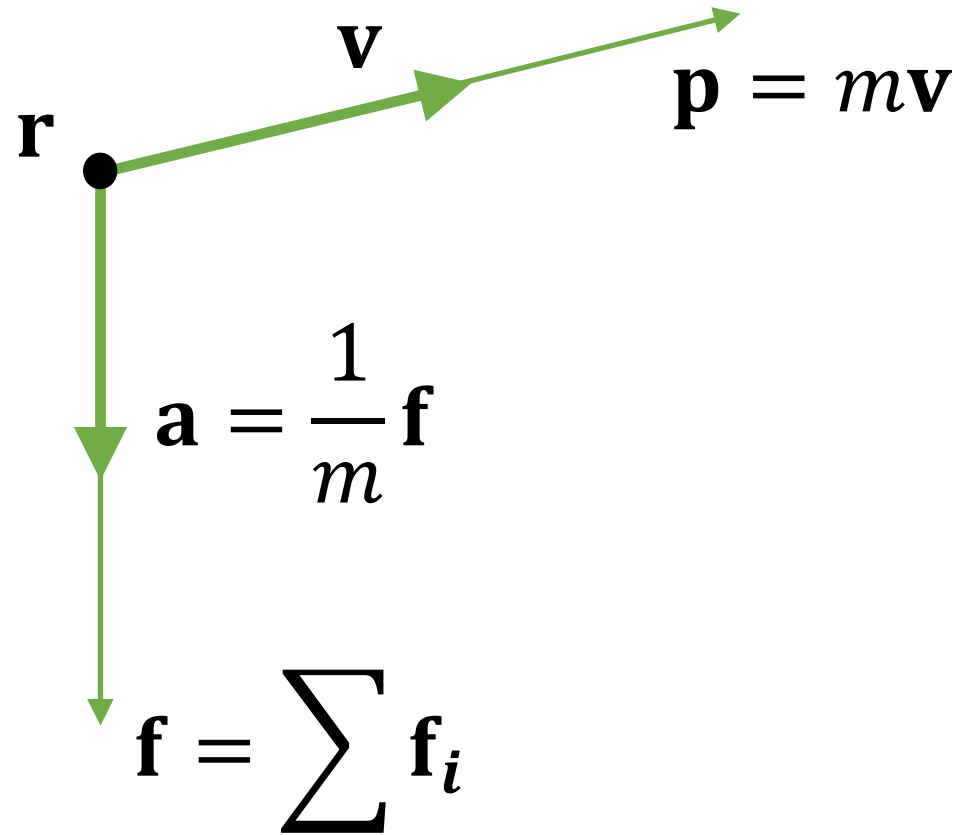
v : velocity

a : acceleration

m : mass

p : momentum

f : force



Summation of Forces

- At any given instant, multiple forces will be acting on each particle
- These will just add up to a single total force

$$\mathbf{f}_{total} = \sum \mathbf{f}_i$$

- We'll often drop the subscript and just write \mathbf{f}_{total} as \mathbf{f}

Forward Euler Integration

- Once we've computed all the forces on a particle, we can use Newton's Second Law to compute its acceleration

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

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

$$\mathbf{v}_{i+1} = \mathbf{v}_i + \mathbf{a}\Delta t$$

$$\mathbf{r}_{i+1} = \mathbf{r}_i + \mathbf{v}_{i+1}\Delta t$$

Physics Simulation

specify initial conditions

while (not finished) {

 // Simulate

 1. Compute all forces

 2. Integrate motion

 // Draw or store results...

}

Physics Simulation (slightly more complex)

specify initial conditions

while (not finished) {

 // Apply user interactions and other logic...

 // Simulate

 1. Compute all forces

 2. Integrate motion

 3. Apply constraints (collisions)

 // Draw or store results...

}

Cloth Simulation

1. Compute Forces

For each particle: apply gravity

For each spring-damper: compute & apply spring-damper forces

For each triangle: compute & apply aerodynamic forces

2. Integrate Motion

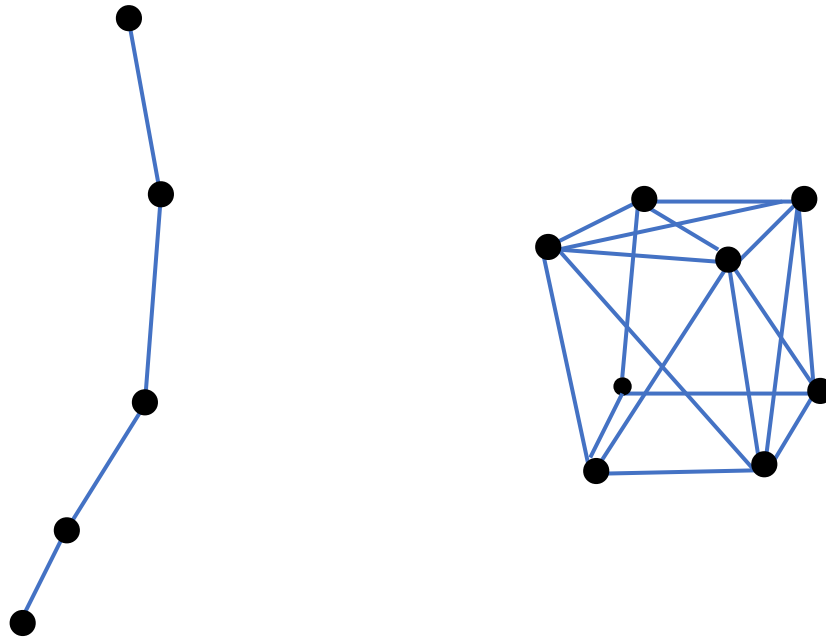
For each particle: compute acceleration and apply forward Euler integration

3. Apply Constraints

For each particle: if intersecting, push to legal position & adjust velocity

Ropes & Solids

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



Forces

Uniform Gravity

- A very simple, useful force is the uniform gravity field:

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

$$\mathbf{g}_0 = [0 \quad -9.8 \quad 0] \frac{m}{s^2}$$

- It assumes that we are near the surface of the Earth and we can approximate the gravity as constant in both magnitude and direction
- 9.8 m/s^2 is a reasonable approximation, as it actually ranges from roughly 9.76 to 9.83 around the world due to variations in altitude and local density (there are detailed maps of this available)

Springs

- We can use Hooke's Law to model simple linear spring forces:

$$\mathbf{f}_{spring} = -k_s \mathbf{x}$$

- Where k_s is the *spring constant* describing the stiffness of the spring and \mathbf{x} is a vector describing the displacement
- The spring force is therefore going to work against the displacement
- The direction of the force will be along the axis of the spring and will pull if the spring is extended and push if the spring is compressed

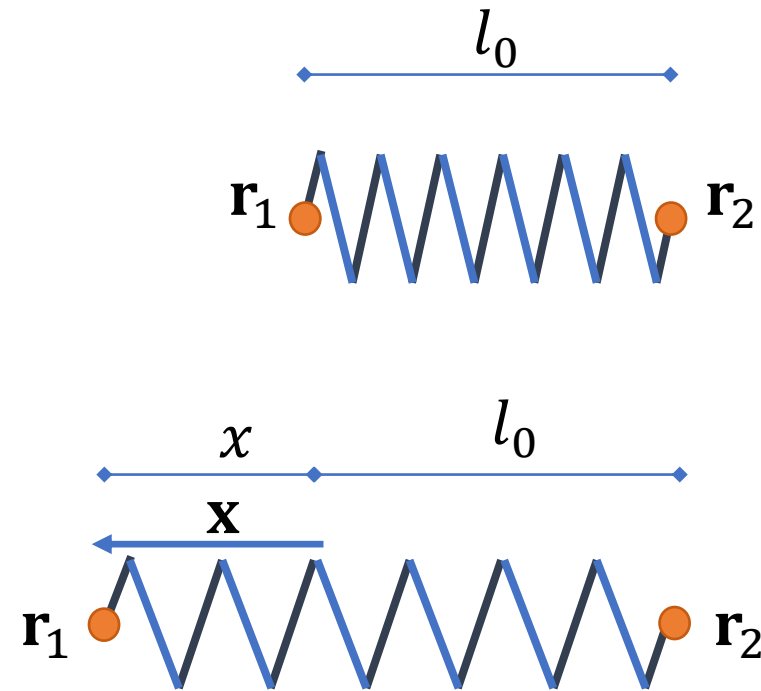
Springs

- In practice, it's nice to define a spring as connecting two particles and having a *rest length* l_0 where the spring force is 0
- This gives us:

$$x = l_0 - |\mathbf{r}_2 - \mathbf{r}_1|$$

$$\mathbf{e} = \frac{\mathbf{r}_2 - \mathbf{r}_1}{|\mathbf{r}_2 - \mathbf{r}_1|}$$

$$\mathbf{x} = x\mathbf{e}$$

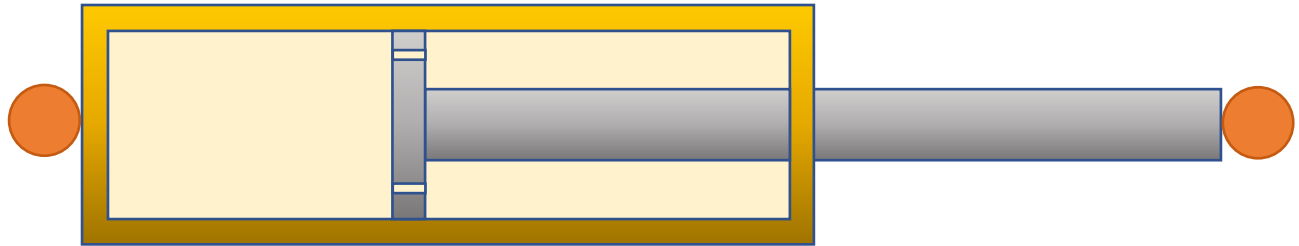
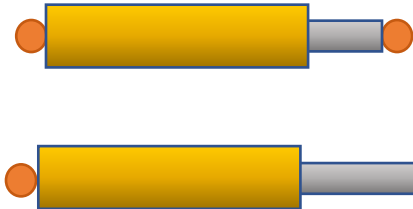


Springs

- A spring applies equal and opposite forces to two particles, and therefore explicitly obeys Newton's Third Law
- They should also obey the Laws of Conservation of Momentum and Conservation of Energy
- In practice however, how well they obey these laws is due to the numerical integration scheme used

Dampers

- Like a spring, a damper can connect between two particles
- It will create a force along the line connecting the particles that resists a difference in velocity along that line
- Car shock absorbers are examples of dampers



Dampers

- We can apply damping forces between particles:

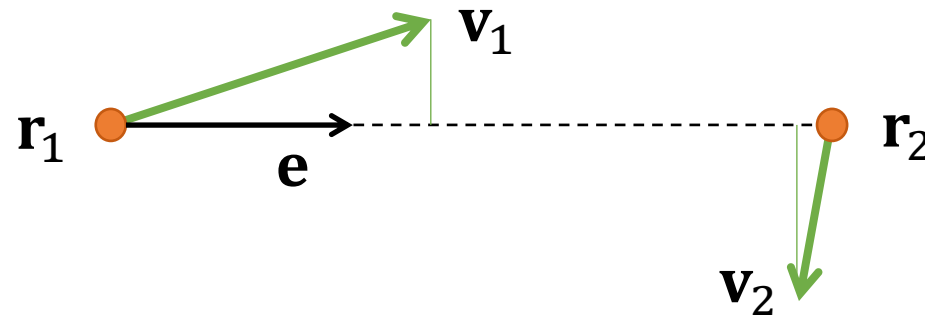
$$\mathbf{f}_{damp} = -k_d v_{close} \mathbf{e}$$

- k_d is the damping constant, v_{close} is the closing velocity, and \mathbf{e} is a unit vector that provides the direction (\mathbf{e} works the same as with springs)
- Dampers will oppose any difference in velocity between particles
- The damping forces are equal and opposite, so they should conserve momentum, but they will remove energy from the system by design

Closing Velocity

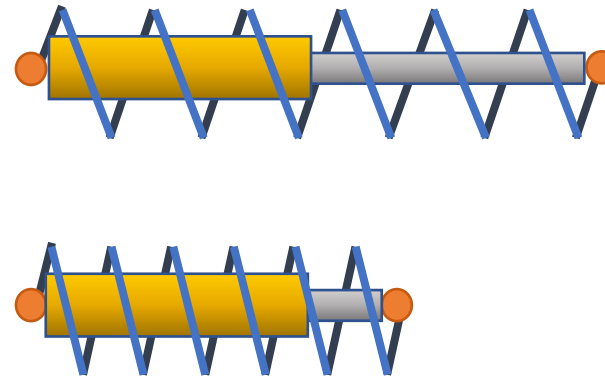
- To calculate the damping force, we need to calculate the *closing velocity* between two particles
- This is the rate that the two particles are approaching each other

$$v_{close} = (\mathbf{v}_1 - \mathbf{v}_2) \cdot \mathbf{e}$$



Spring-Dampers

- For convenience, we can combine a spring and a damper into a single object
- It will store three constants:
 - Rest length: l_0
 - Spring constant: k_s
 - Damping constant: k_d



Spring-Damper

- A simple spring-damper class might look like:

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

Spring-Damper Forces

1. Compute current length l & unit vector \mathbf{e}

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

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

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

2. Compute closing velocity v_{close}

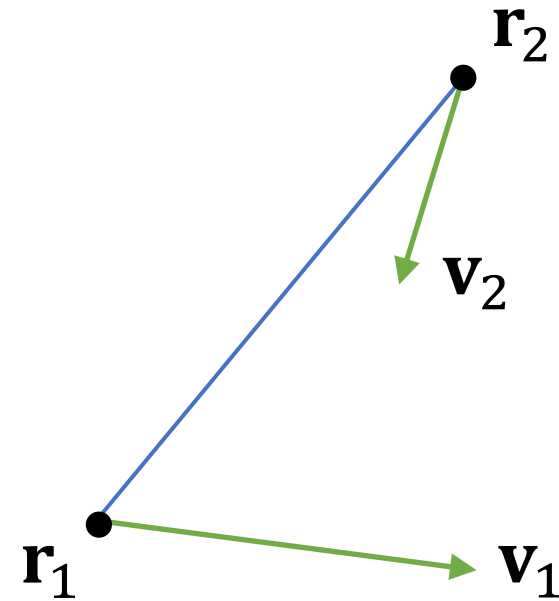
$$v_{close} = (\mathbf{v}_1 - \mathbf{v}_2) \cdot \mathbf{e}$$

3. Compute final forces

$$f = -k_s(l_0 - l) - k_d v_{close}$$

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

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



Aerodynamic Drag

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

$$\mathbf{f}_{drag} = \frac{1}{2} \rho |\mathbf{v}|^2 c_d a \mathbf{e}$$

- Where ρ is the density of the surrounding fluid (air, water, etc.), c_d is the coefficient of drag for the object, a is the cross sectional area of the object, and \mathbf{e} is a unit vector in the opposite direction of the velocity:

$$\mathbf{e} = -\frac{\mathbf{v}}{|\mathbf{v}|}$$

- Also, keep in mind that we really want the relative velocity, which is the different between the particle velocity and the average velocity of the surrounding fluid

$$\mathbf{v} = \mathbf{v}_{particle} - \mathbf{v}_{fluid}$$

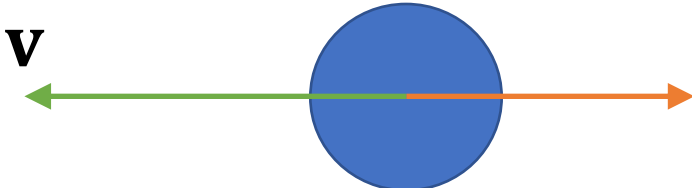
Aerodynamic Force

- Today we will extend that to a simple *flat surface* force

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

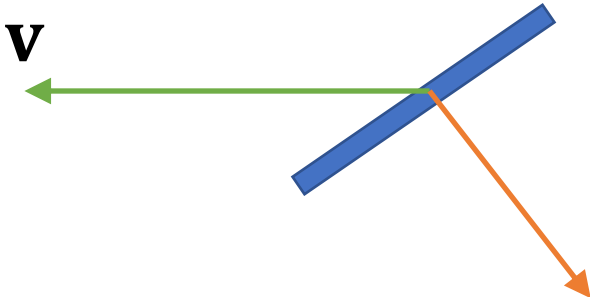
- The only real difference is that this acts along the normal instead of in the opposite direction of velocity
- This models the directionally varying force on a flat surface angled to the airflow instead of the unidirectional drag force on a symmetric object
- Note that this is a major simplification of true aerodynamic interactions, but it's a good model to start with

Aerodynamic Force



A diagram showing a blue sphere. A green arrow labeled \mathbf{v} points to the left, representing the flow velocity. An orange arrow points to the right from the center of the sphere, representing the drag force.

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



A diagram showing a blue rectangular plate tilted at an angle. A green arrow labeled \mathbf{v} points to the left, representing the flow velocity. An orange arrow points downwards and to the right from the center of the plate, representing the aerodynamic force.

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

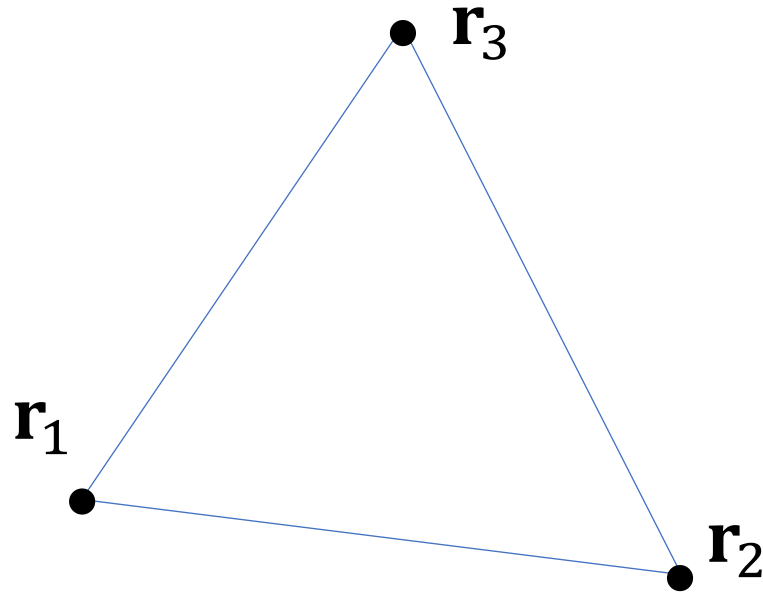
Aerodynamic Constants

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

- The fluid density ρ of air at 15° C and a pressure of 101.325 kPa (14.696 psi) is 1.225 kg/m³ and is used as a common default value
- The drag coefficient c_d for a flat plate is around 1.28, so we should use at least 1.0 or more
- The cross-sectional area a will actually vary as the surface changes its angle to the airflow or if the surface area itself changes

Aerodynamic Force

- For our cloth, we will assume the surface is made up of a bunch of triangles (the same triangles we will use to render)
- Each triangle will compute an aerodynamic force and apply it to the three particles it connects



Aerodynamic Force

- In order to compute our force:

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

- We will need to find the velocity \mathbf{v} , normal \mathbf{n} , and cross-sectional area a of the triangle (assuming ρ and c_d are constants)

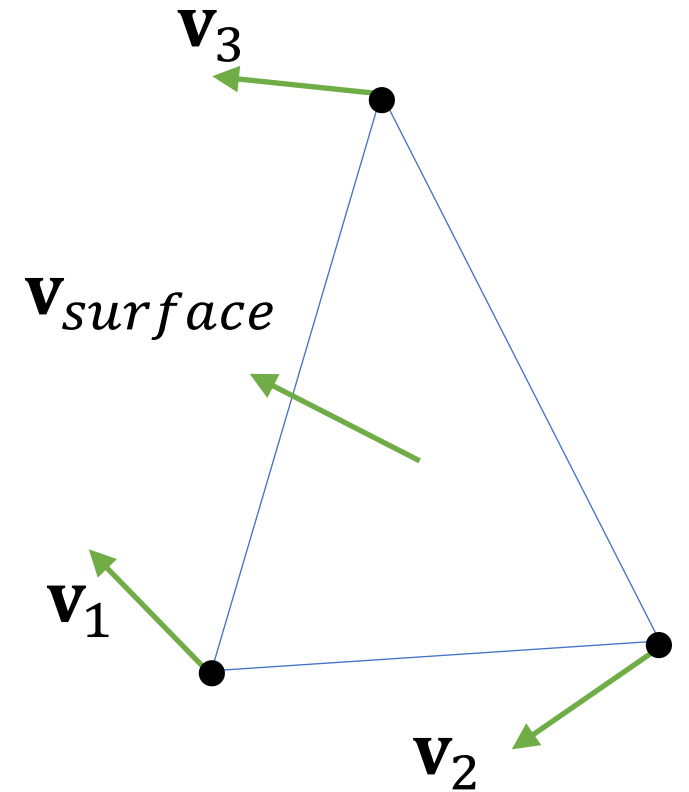
Triangle Velocity

- 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 to the airflow, so we will then subtract off the velocity of the air itself

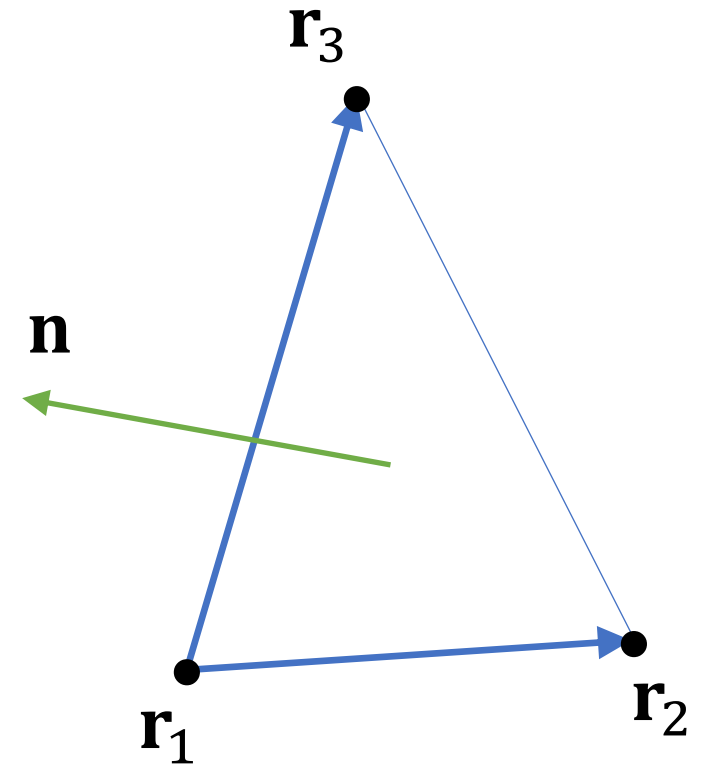
$$\mathbf{v} = \mathbf{v}_{surface} - \mathbf{v}_{air}$$



Triangle Normal

- The normal of the triangle is

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



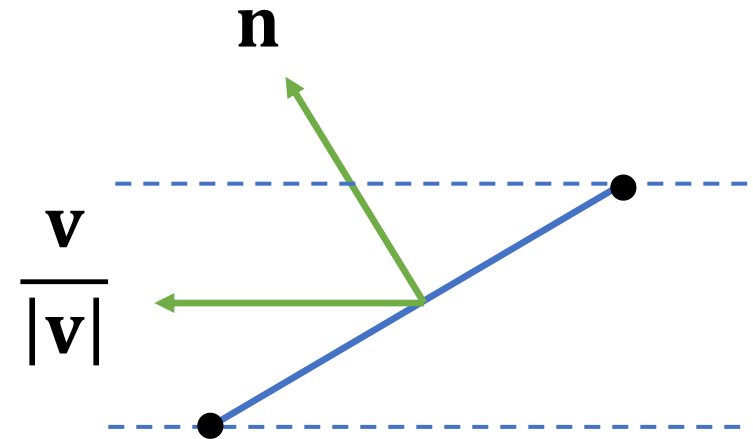
Cross Sectional Area

- The area of the triangle is

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

- But we really want the cross-sectional area which is the area viewed from the direction of the airflow

$$a = a_0 \left(\frac{\mathbf{v}}{|\mathbf{v}|} \cdot \mathbf{n} \right)$$

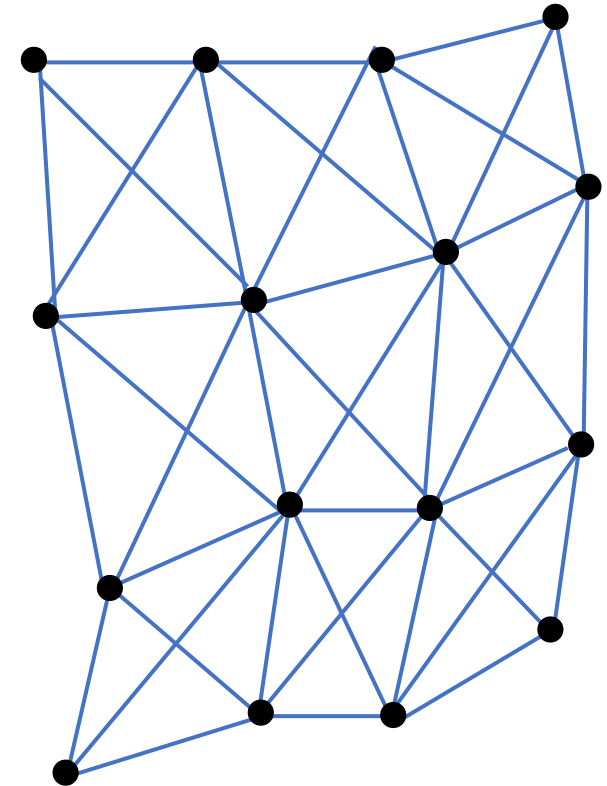


Aerodynamic Force

- The final force is assumed to apply over the entire triangle
- We can simply apply $1/3^{\text{rd}}$ of the total force to each of the three particles connecting the triangle

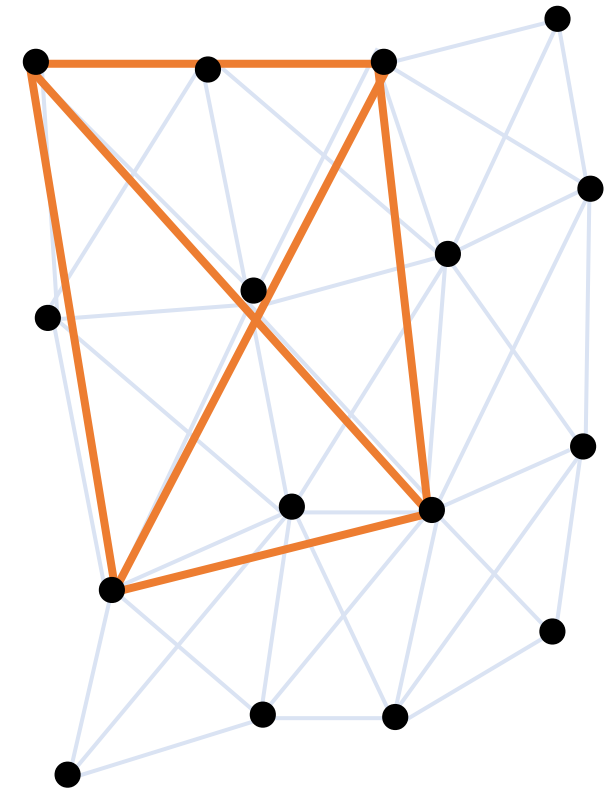
Bending Forces

- 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



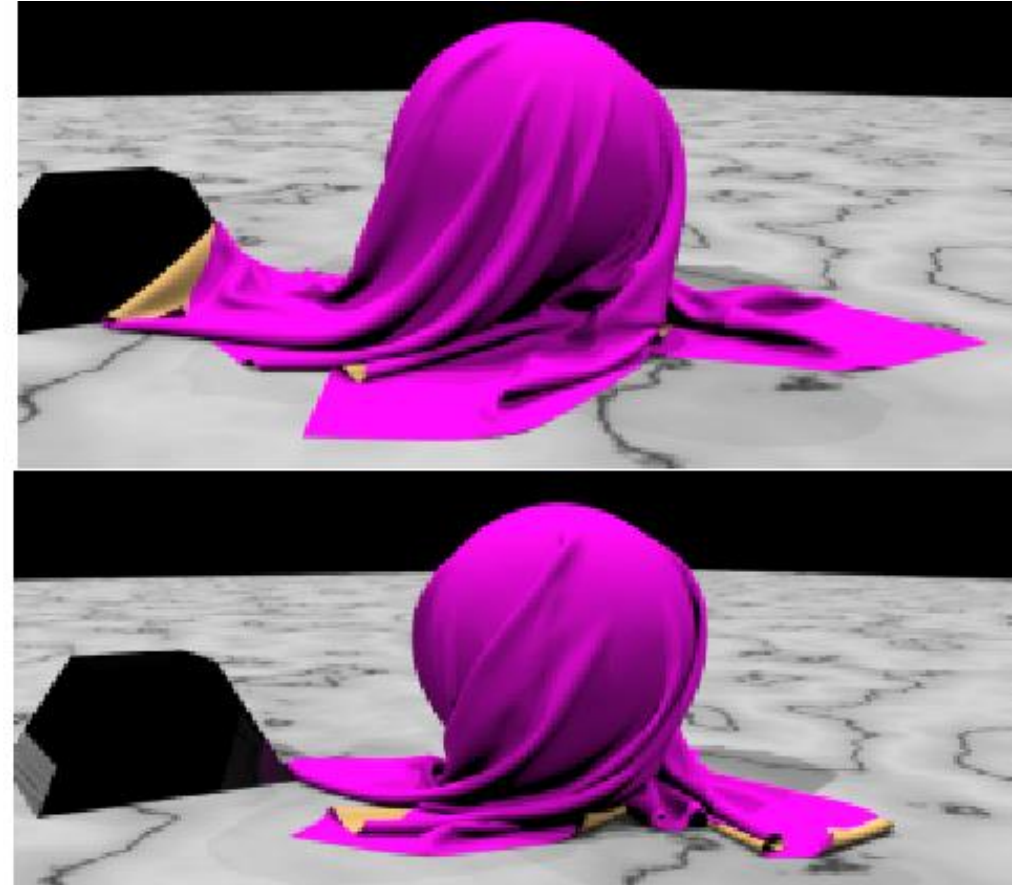
Bending Forces

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



Collisions

- General purpose collision detection with cloth is quite complicated, as one must consider self-collisions between multiple moving components
- We will discuss some collision detection and response in the next lecture



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...

System Stability

Conservation of Momentum

- Our simple spring-damper implementation should enforce conservation of momentum, due to the way we explicitly apply the equal and opposite forces
- Combined with the simple forward Euler integrator, this ensures that the total system momentum will remain constant over time (within floating point round-off errors)

Conservation of Energy

- True linear springs also should conserve energy, as the kinetic energy of particle motion can be converted to deformation energy in the springs
- The dampers however, are specifically intended to remove kinetic energy from the system (and convert it to heat)
- The Euler integration scheme we are using is not guaranteed to conserve energy, as it never explicitly deals 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 of 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, is 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
- This can happen when we have large spring constants or large damping constants
- The forward (or *explicit*) Euler integration scheme is the simplest, but can easily become unstable and require very small time steps in order to produce useful results

Backward Euler Method

- There is related method called *backward* (or *implicit*) *Euler integration* that actually can achieve perfect stability, even at very large time steps, however it still requires small time steps to achieve good accuracy
- The backward Euler method is much more complex as it requires solving for all of the forces in the system as one large set of simultaneous nonlinear equations...

Other Integration Methods

- There are numerous other integration schemes (midpoint, Runge-Kutta, Adams Moulton, general linear methods, etc.)
- Many of these have similar explicit and implicit forms
- Most of the explicit methods are focused more towards accuracy rather than stability, and have similar stability properties (or worse) than the forward Euler method
- For our purposes, the forward Euler method will be fine, but we need to be careful with the time step size

Adaptive Time Steps

- One powerful method for improving the stability of almost any integrator is use *adaptive time steps*
- With this method, we monitor the system stability and dynamically adjust the time step as necessary
- If the system gets less stable, we switch to smaller time steps
- As things stabilize, we can switch to larger time steps
- There are many other details, mainly relating to how exactly we monitor the current stability

Oversampling

- A simple approach to stabilizing the simulation (which would probably be good for project 4) is to use oversampling
- With this approach, we break a time step into a fixed number of smaller time steps
- For example, if with to simulate at 60 FPS, a single time step would be $1/60^{\text{th}}$ of a second
- If we want to do 10 oversamples, we would actually run 10 steps at $1/600^{\text{th}}$ of a second, and then draw the scene once

Project 4: Cloth Simulation

Project 4: Cloth Simulation

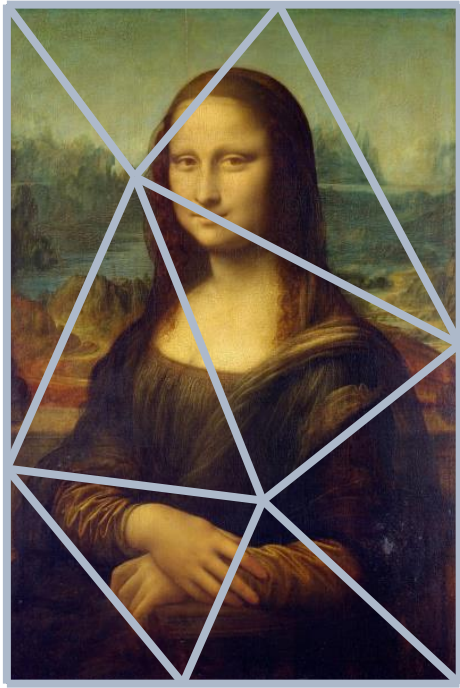
- Create a cloth simulation with spring-dampers
 - Grid of spring-dampers
 - Aerodynamic drag forces & adjustable wind speed
 - Ground plane collision handling
 - Fixed particles and effective user manipulation

Issues

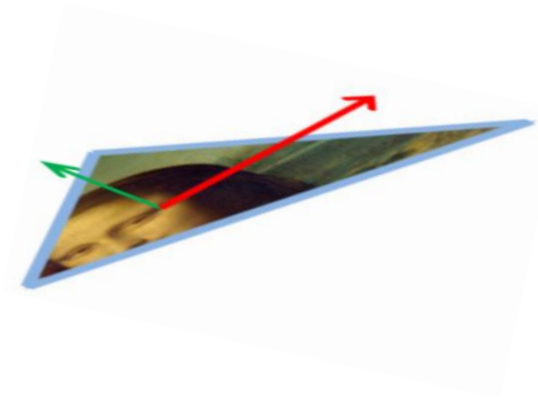
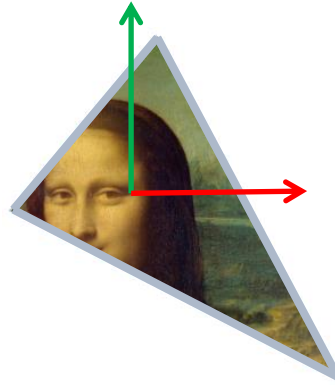
- Tuning parameters
- Stability
- 2D symmetry
- Performance
- GPU
- Rendering

Advanced Cloth

Continuum Mechanics



material space



world space

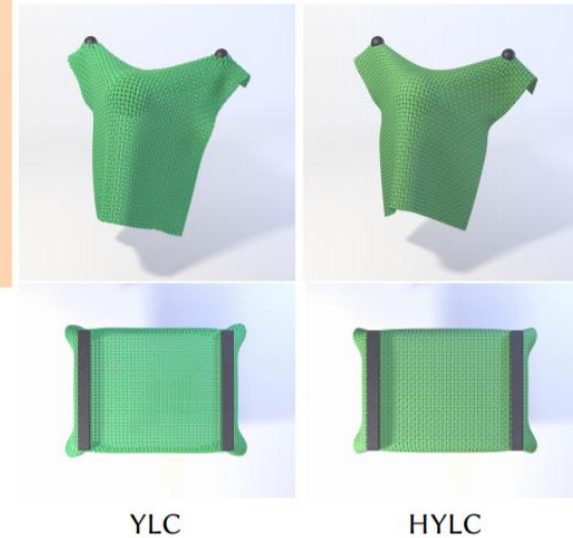
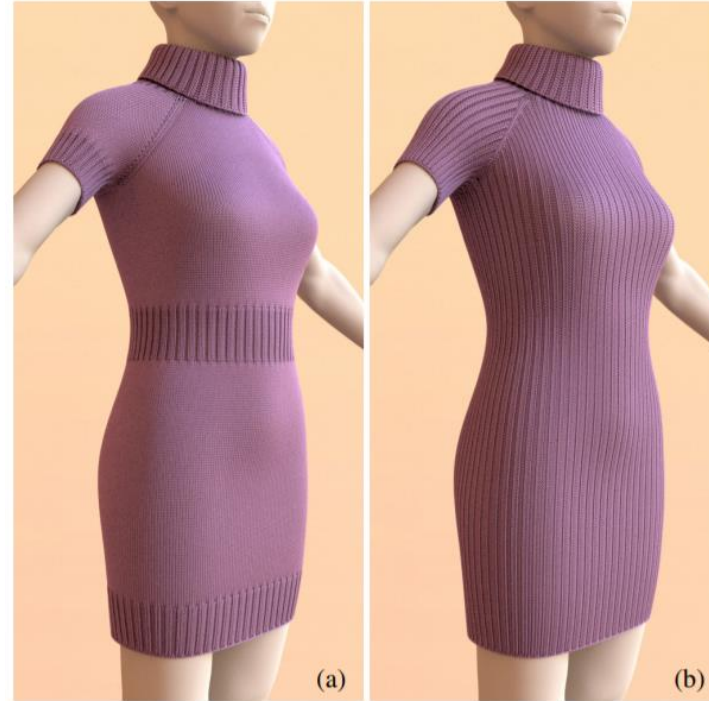
Collision Detection & Response

- “Robust Treatment of Collisions, Contact and Friction for Cloth Animation”, Bridson, Fedkiw, Anderson, 2002
- “Fast Continuous Collision Detection using Deforming Non-Penetration Filters”, Tang, Manocha, Tong, 2010
- “Local Optimization for Robust Signed Distance Field Collision”, Macklin, Erleben, Muller, Chentanez, Jeschke, Corse, 2020



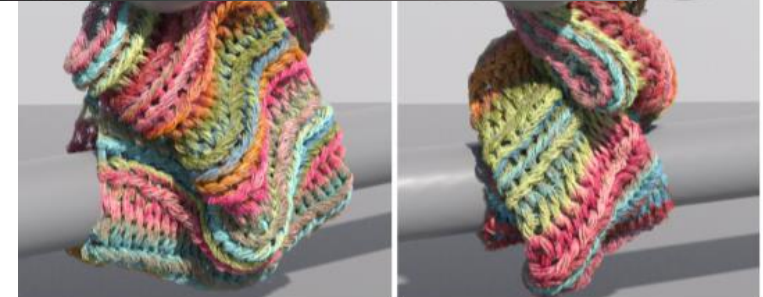
Yarn-Level Simulation

- “Simulating Knitted Cloth at the Yarn Level”, Kaldor, James, Marschner, 2008
- “Stitch Meshes for Modeling Knitted Clothing with Yarn-Level Detail”, Yuksel, Kaldor, James, Marschner, 2012
- “Homogenized Yarn-Level Cloth”, Sperl, Narain, Wojtan, 2020



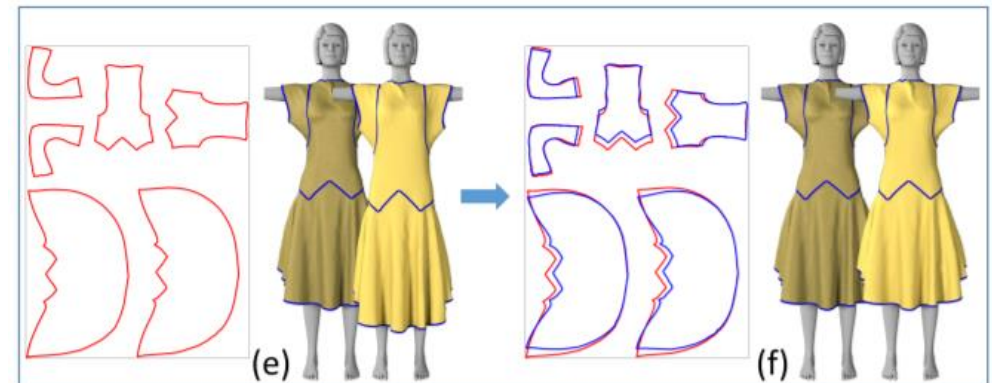
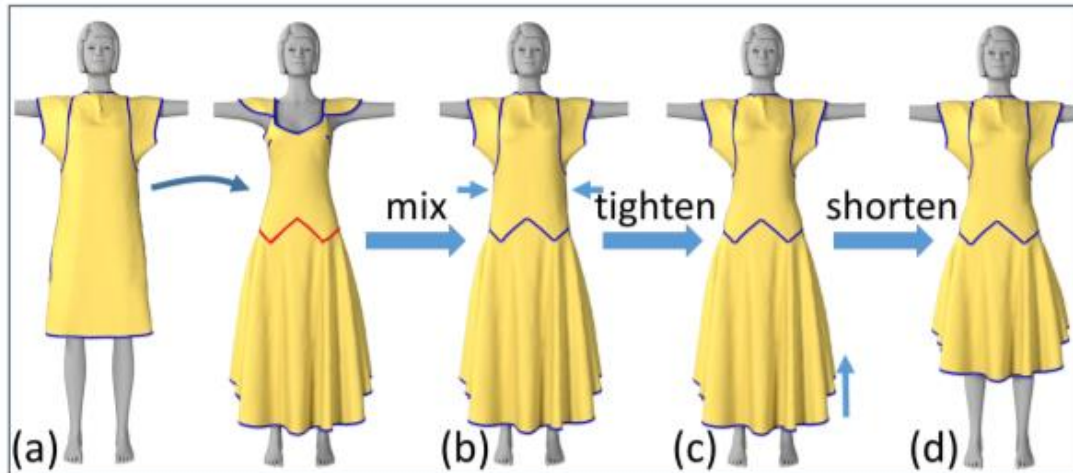
Fiber-Level Simulation

- “Anisotropic Elasticity for Cloth, Knit and Hair Frictional Contact”, Jiang, Gast, Teran, 2017



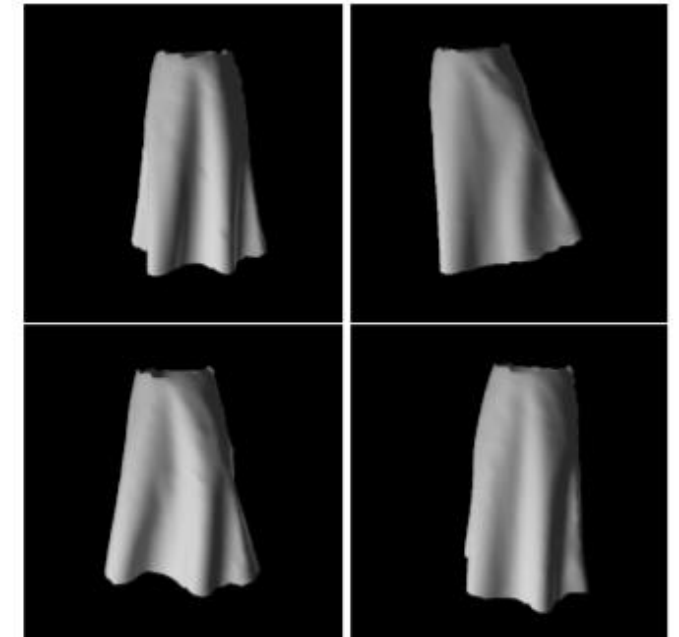
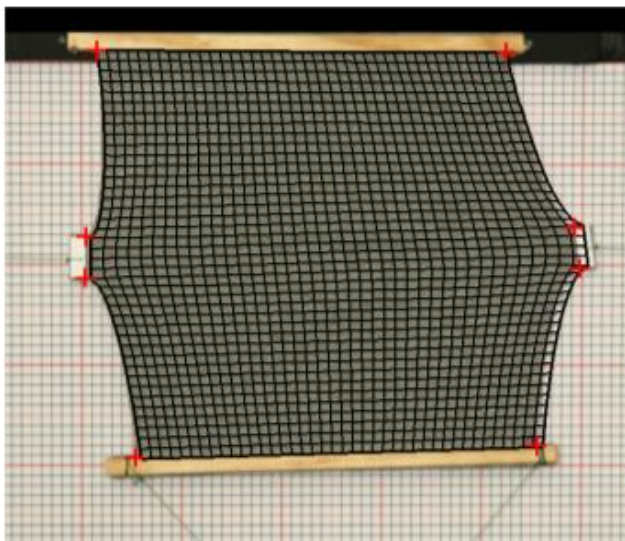
Clothing Design & Dressing

- “Design Preserving Garment Transfer”, Brouet, Sheffer, Boissieux, Cani, 2012
- “Physics-driven Pattern Adjustment for Direct 3D Garment Editing”, Bartle, Sheffer, Kim, Kaufman, Vining, Berthouzoz, 2016



Motion & Attribute Capture

- “Garment Motion Capture Using Color-Coded Patterns”, Alexa, Marks, 2005
- “Data-Driven Elastic Models for Cloth: Modeling and Measurement”, Wang, O’Brien, Ramamoorthi, 2011
- “Detailed Garment Recovery from a Single-View Image”, Yang, Amert, Pan, Wang, Yu, Berg, Lin, 2016



Multi-scale Rendering

- “Structure-aware Synthesis for Predictive Woven Fabric Appearance”, Zhao, Jakob, Marschner, Bala, 2012
- “Multi-Scale Hybrid Micro-Appearance Modeling and Realtime Rendering of Thin Fabrics”, Xu, Wang, Zhao, Bao, 2019

