FACULTY OF ENGINEERING AND ARCHITECTURE

E016712: Computer Graphics

Animation Part 2

Lecturers: Aleksandra Pizurica and Danilo Babin

GHENT UNIVERSITY

Overview

- Animating using:
  - Motion capture
  - Free form deformation
  - Level sets
  - Skeletons
  - Boids
  - Particle systems

The material partially based on: E. Angel and D. Shreiner: Interactive Computer Graphics 6E © Addison-Wesley 2012

Typical Character

Skeleton Joints

Joints are used to create a framework for a character's hierarchy. The rotation of the skeleton joints defines the motion of the character; you can use inverse kinematics for even more control.

Character Controls

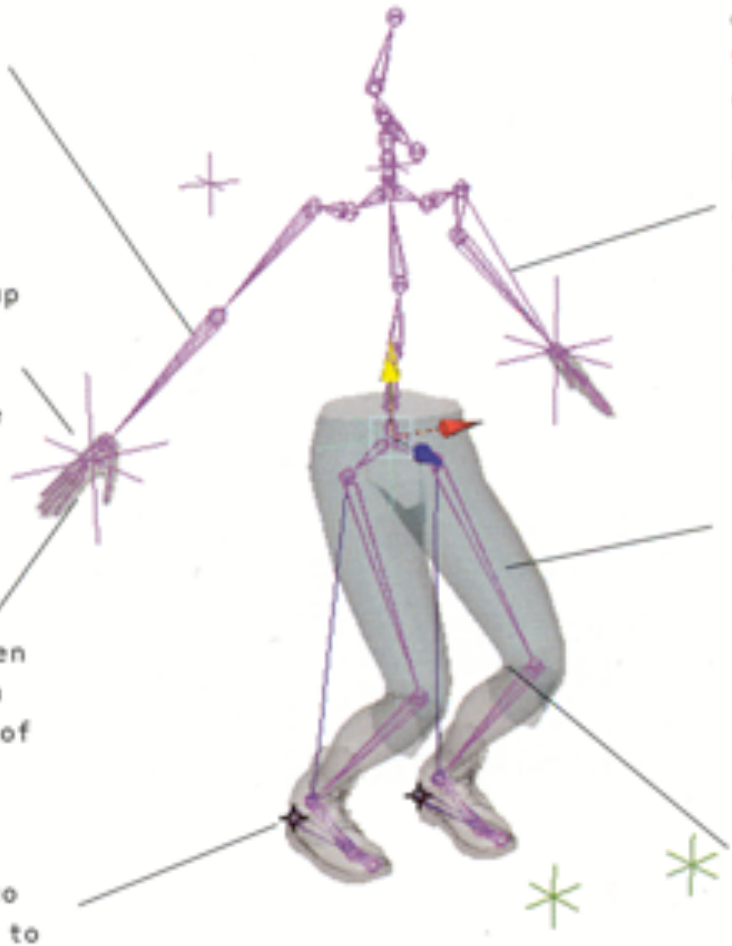
Using animation techniques such as Set Driven Key and expressions, you can set up attributes for controlling different parts of a character. For example, a hand joint could have attributes used to control the different finger joints.

Constraints

It is possible to constrain the kinematic controls of a skeleton to objects in your scene or even simple locators. You can then animate the constraint weights to make a character pick something up or grab hold of a fixed object.

Selection Handles

Selection Handles give you quick access to parts of a character's hierarchy that are to be animated. This makes it easier to work with a character after it has been rigged up for animation.



Facial Animation

To animate facial features, you can use deformers such as Blend Shape to create facial poses that can be used for talking and for showing emotion.

Kinematics

To control your skeleton joints, you can choose from forward or inverse kinematics. Forward Kinematics allows you to set the joint rotations directly. IK allows you to position IK Handles, which rotates the joints.

Bound Surfaces

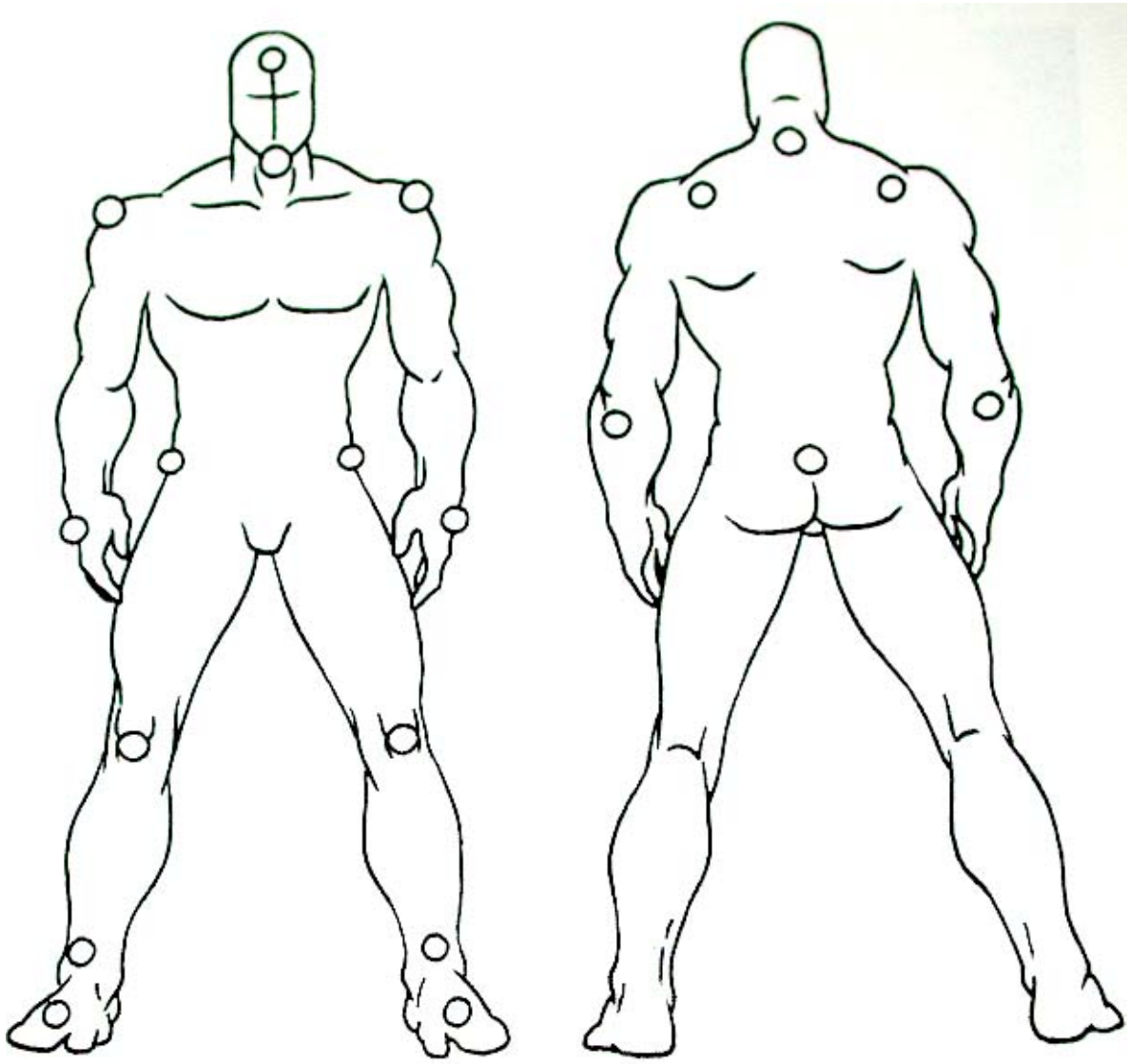
Surfaces of a character's skin and clothing can be either parented or bound to the skeleton joints to make them move together. Binding places points from a surface into clusters that are then associated with particular joints.

Deformers

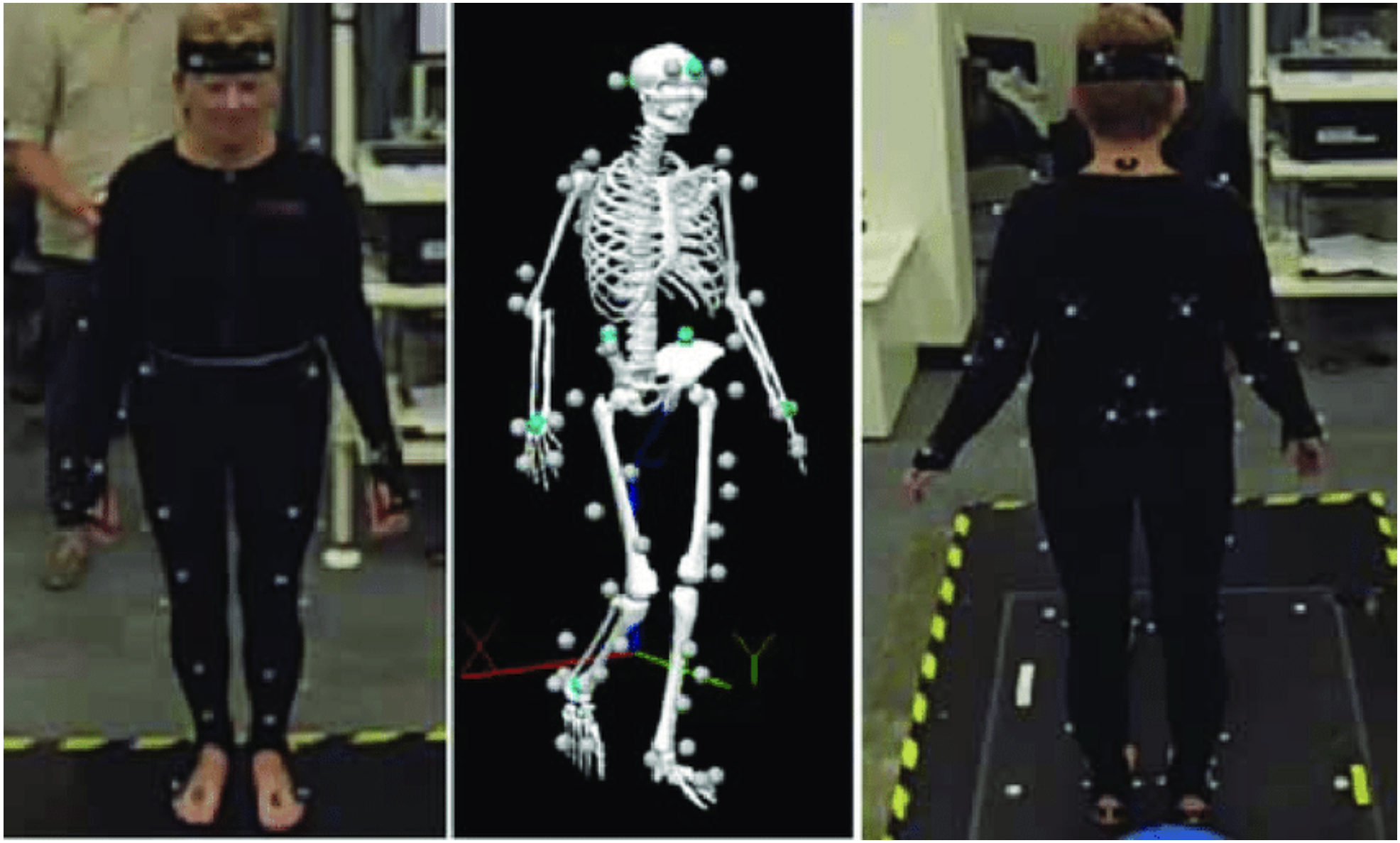
To help the surfaces bend realistically at joints, deformers such as flexors and influence objects can be used.

Motion Capture

- More realistic motion sequences can be generated by Motion Capture
- Attach joint position indicators to real actors
- Record live action



Motion Capture



Soft object animation

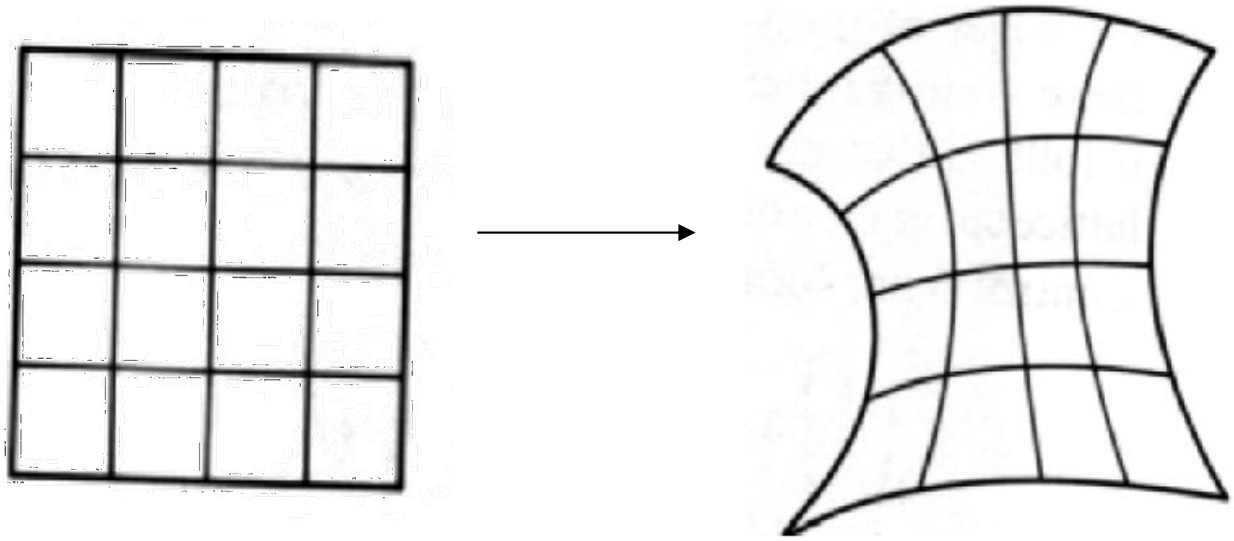
- Soft object is an object that can be deformed by the user or during the process of animation.
- Shape distortion to highlight dynamic interaction with the environment.
- Free Form Deformation (FFD) is part of the computer graphics literature on soft objects

Soft object animation

- Examples:
  - Deform the shape of a car during a collision in a racing simulation
  - Realistic deformation of an object that has a highly elastic and flexible shape
  - Facial expressions, motion of the human body, and cartoon animation
- Deformation of an object occurs by moving the vertices of a polygonal object or the control points of a parametric curve

Free Form Deformation (FFD)

- Deform space by deforming a lattice around an object



- The deformation is defined by moving the control points

Free Form Deformation (FFD)

A two-dimensional Bézier surface can be defined as a parametric surface where the position of a point **p** as a function of the parametric coordinates *u*, *v* is given by: [1]

$$\mathbf{p}(u,v) = \sum_{i=0}^{n}\sum_{j=0}^{m} B_i^n(u)\, B_j^m(v)\, \mathbf{k}_{i,j}$$

evaluated over the unit square, where

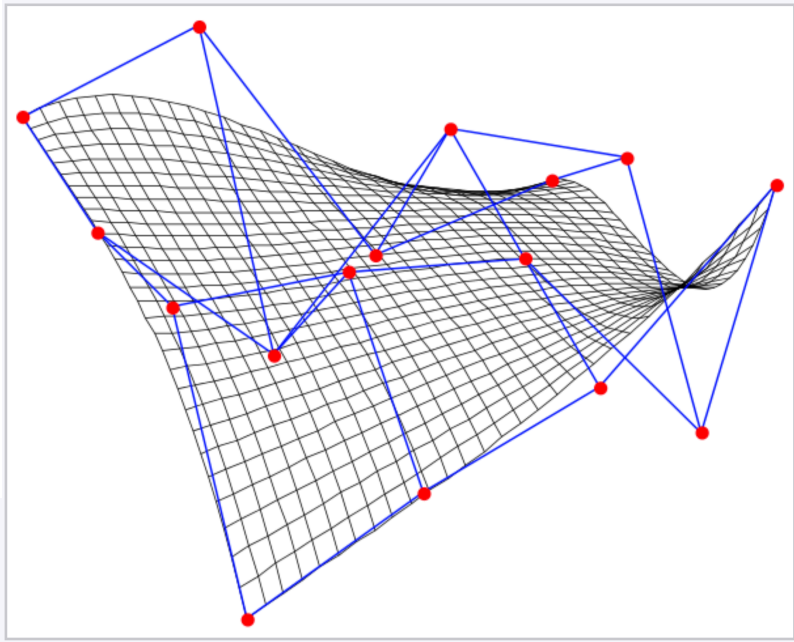
$$B_i^n(u) = \binom{n}{i} u^i (1-u)^{n-i}$$

is a Bernstein polynomial, and

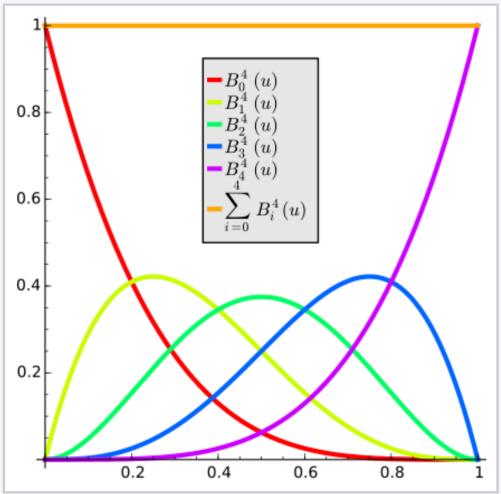
$$\binom{n}{i} = \frac{n!}{i!(n-i)!}$$

is the binomial coefficient.

$k_{i,j}$ are control points



Sample Bézier surface; red – control points, blue – control grid, black – surface approximation



Bernstein basis polynomials for 4th degree curve blending

Source: Wikipedia

Free Form Deformation (FFD)

The lattice defines a Bezier volume

$$p(u,v,w) = \sum_{i,j,k} B(u)B(v)B(w)k_{ijk}$$

1. Compute lattice coordinates

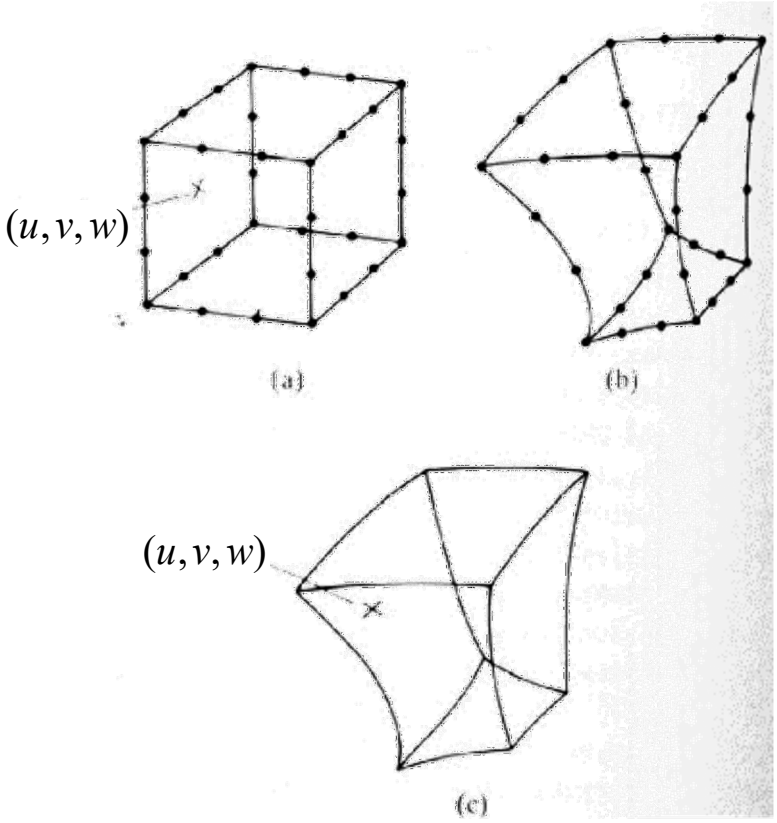
$$(u,v,w)$$

2. Alter the control points

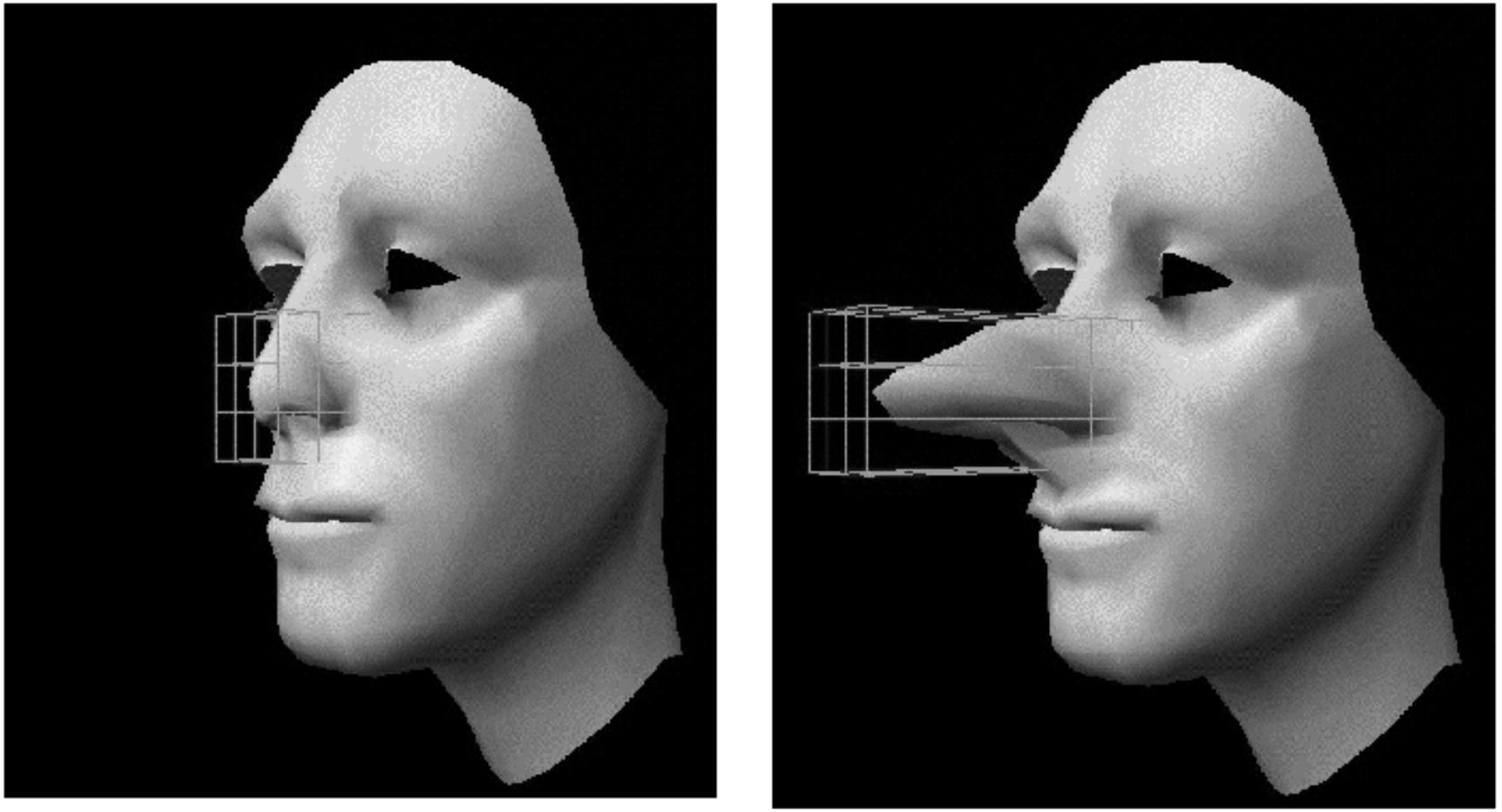
$$k_{ijk}$$

3. Compute the deformed points

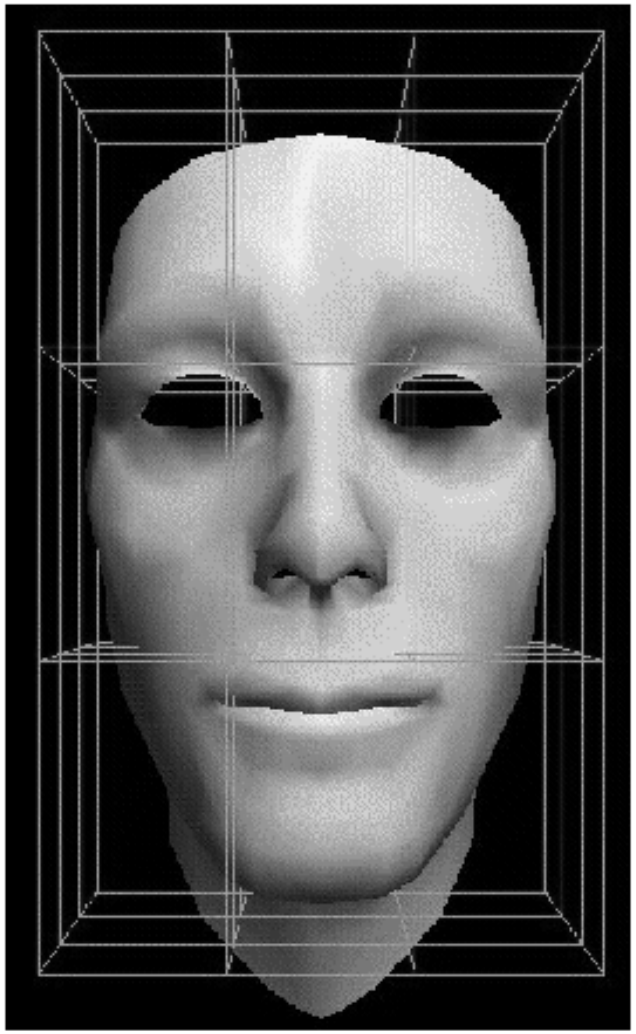
$$p(u,v,w)$$

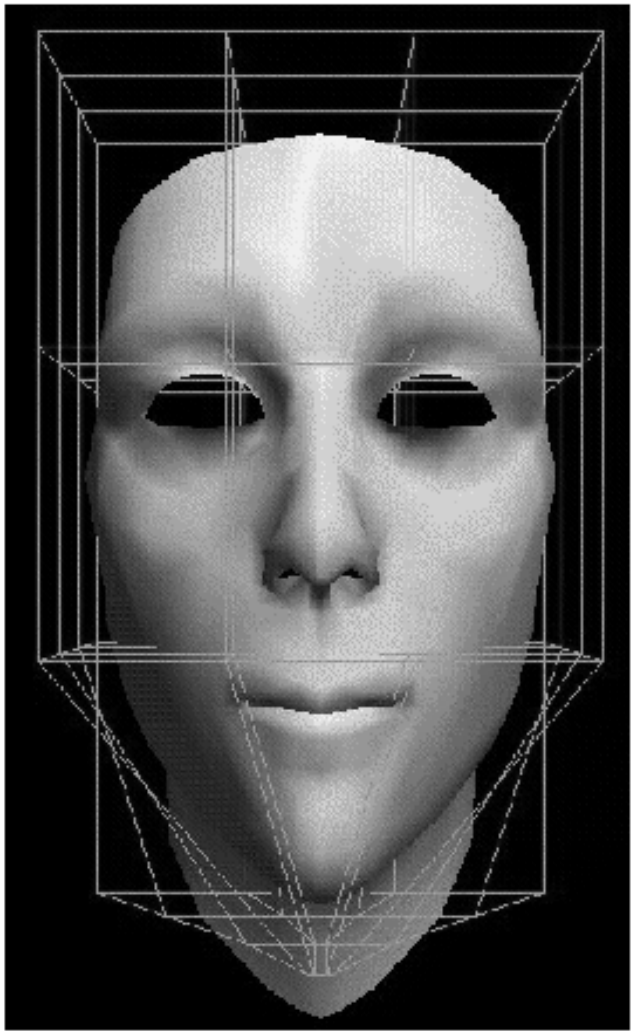


FFD Example



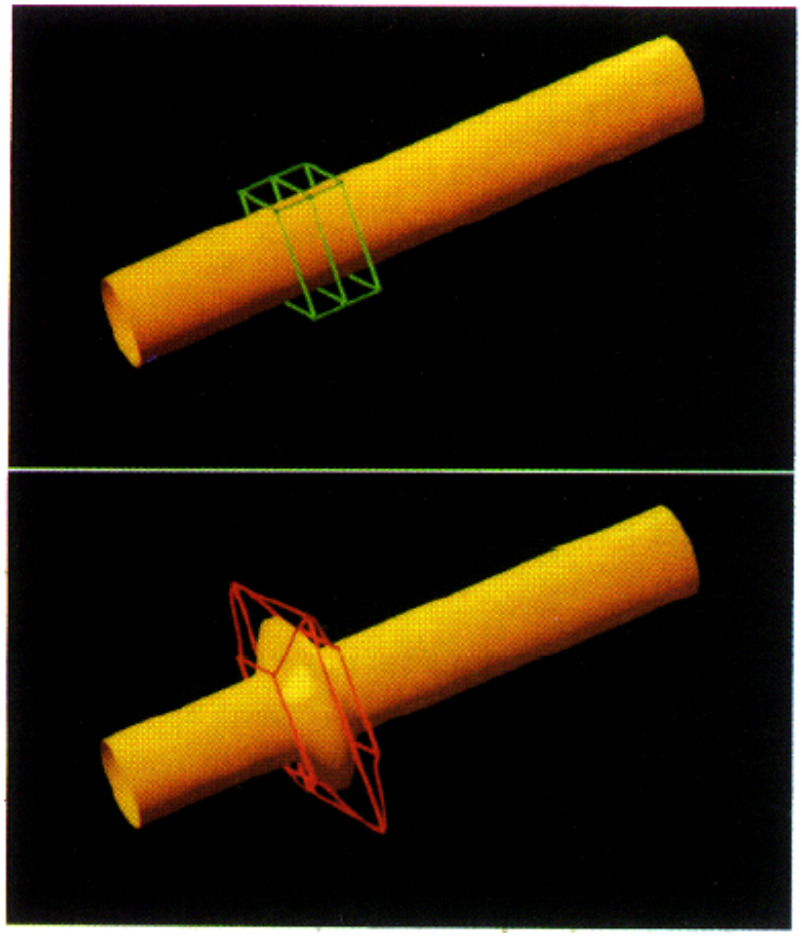
FFD Example

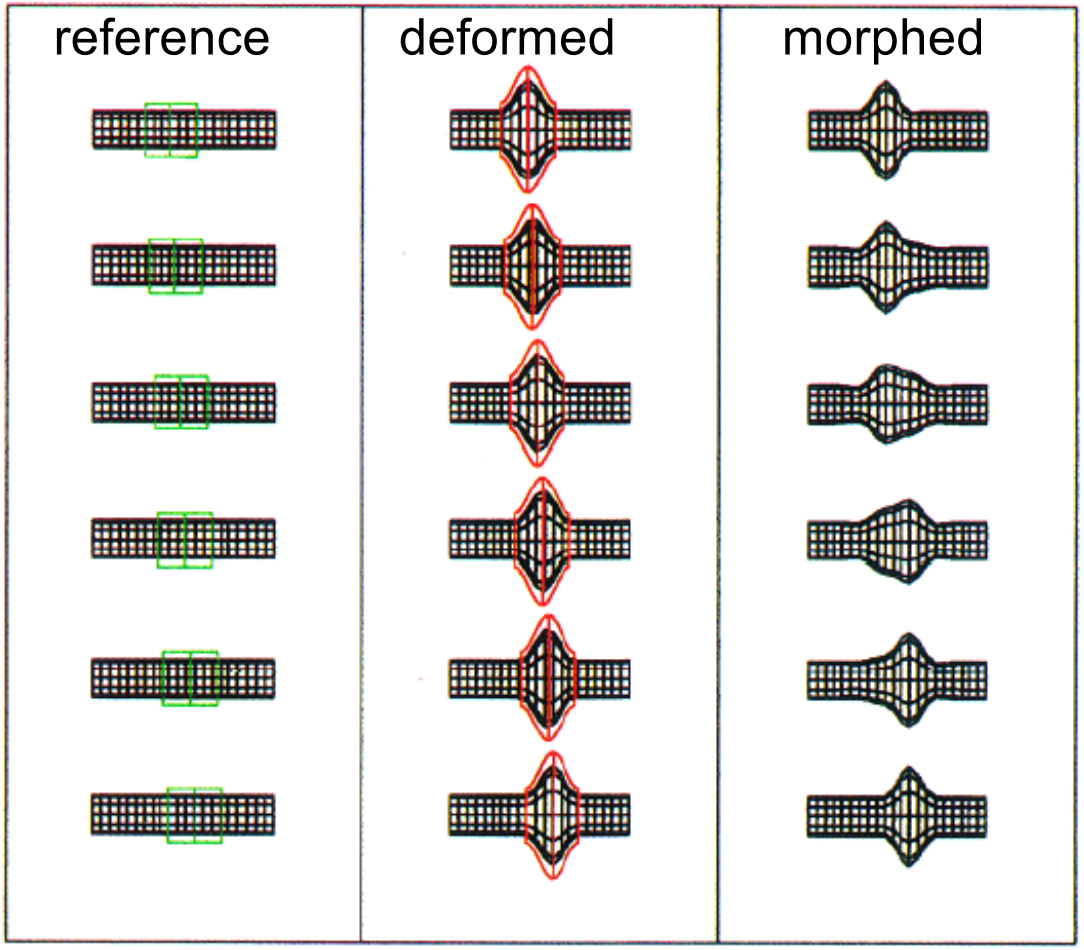




FFD Animation

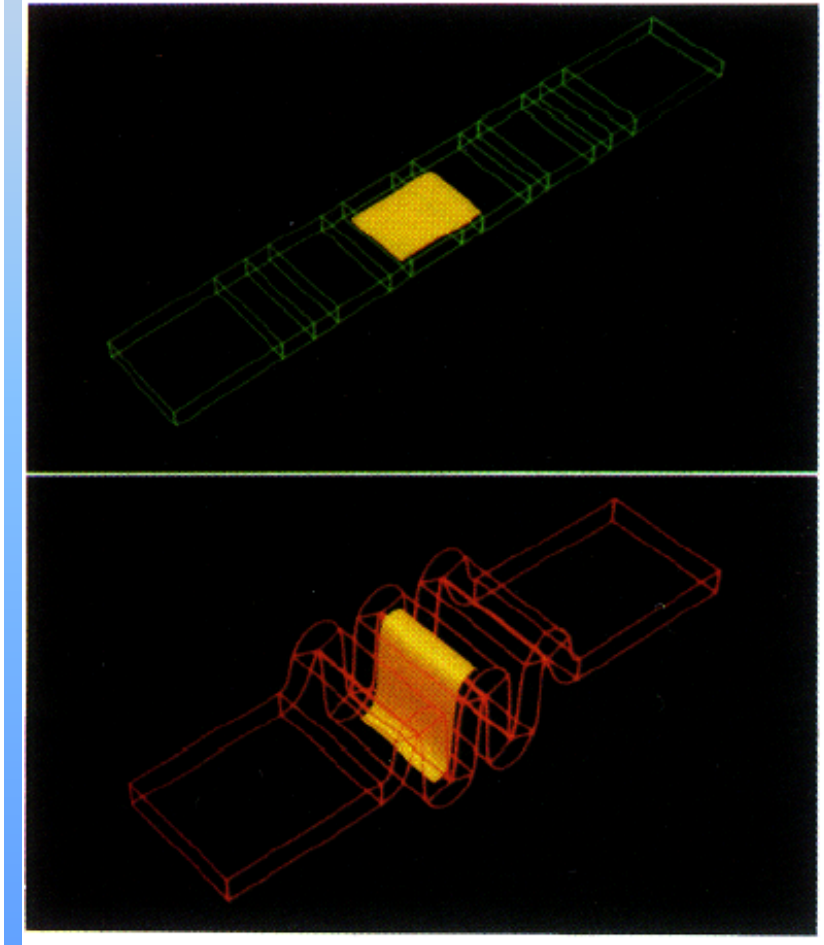
Animate a reference and a deformed lattice

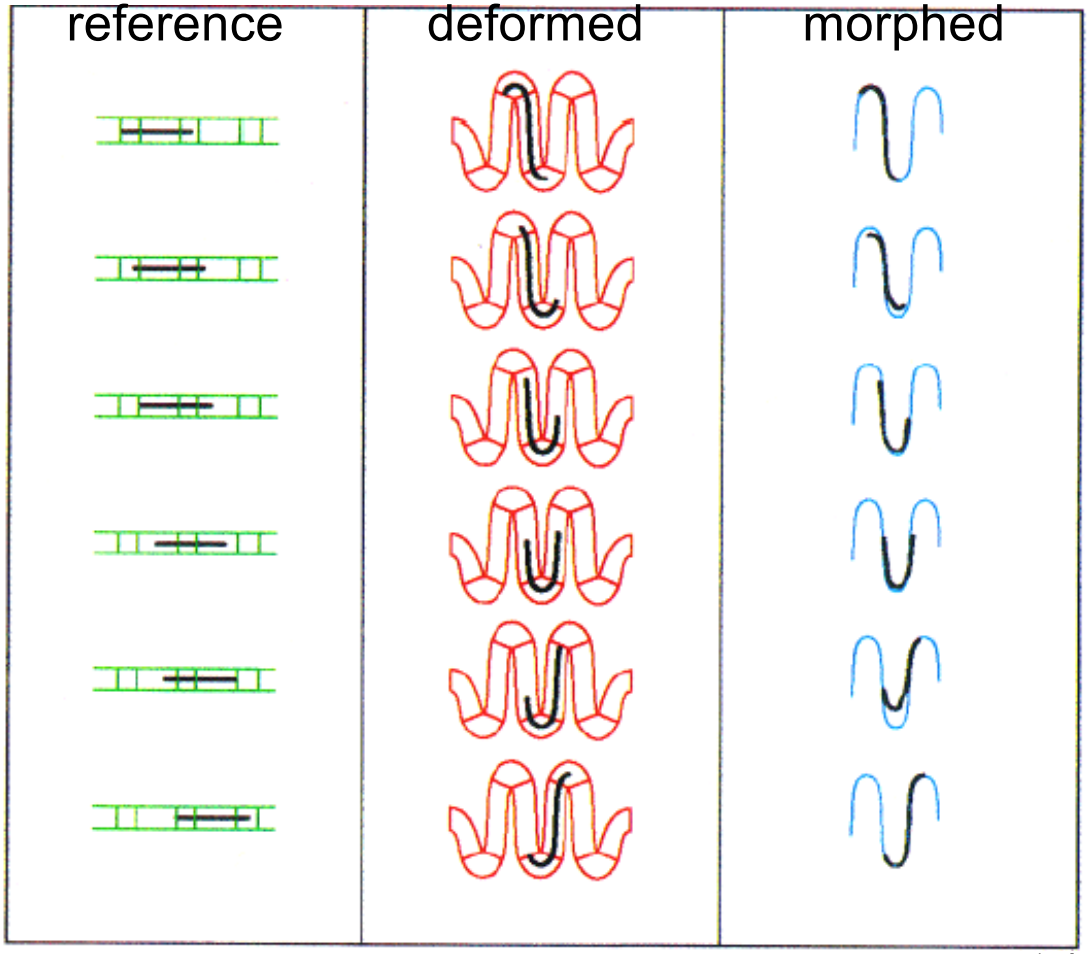




FFD Animation

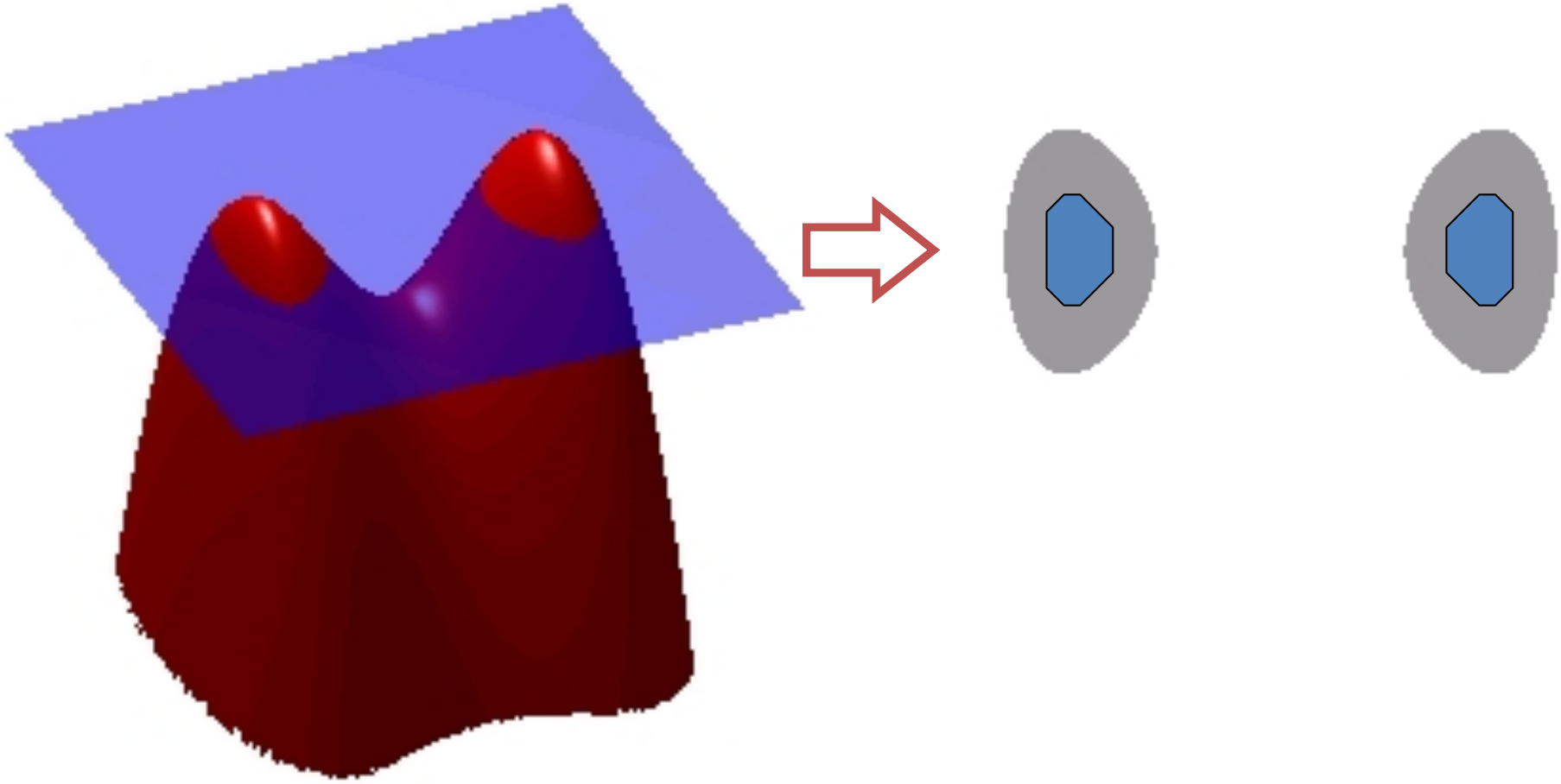
Animate the object through the lattice





Level sets

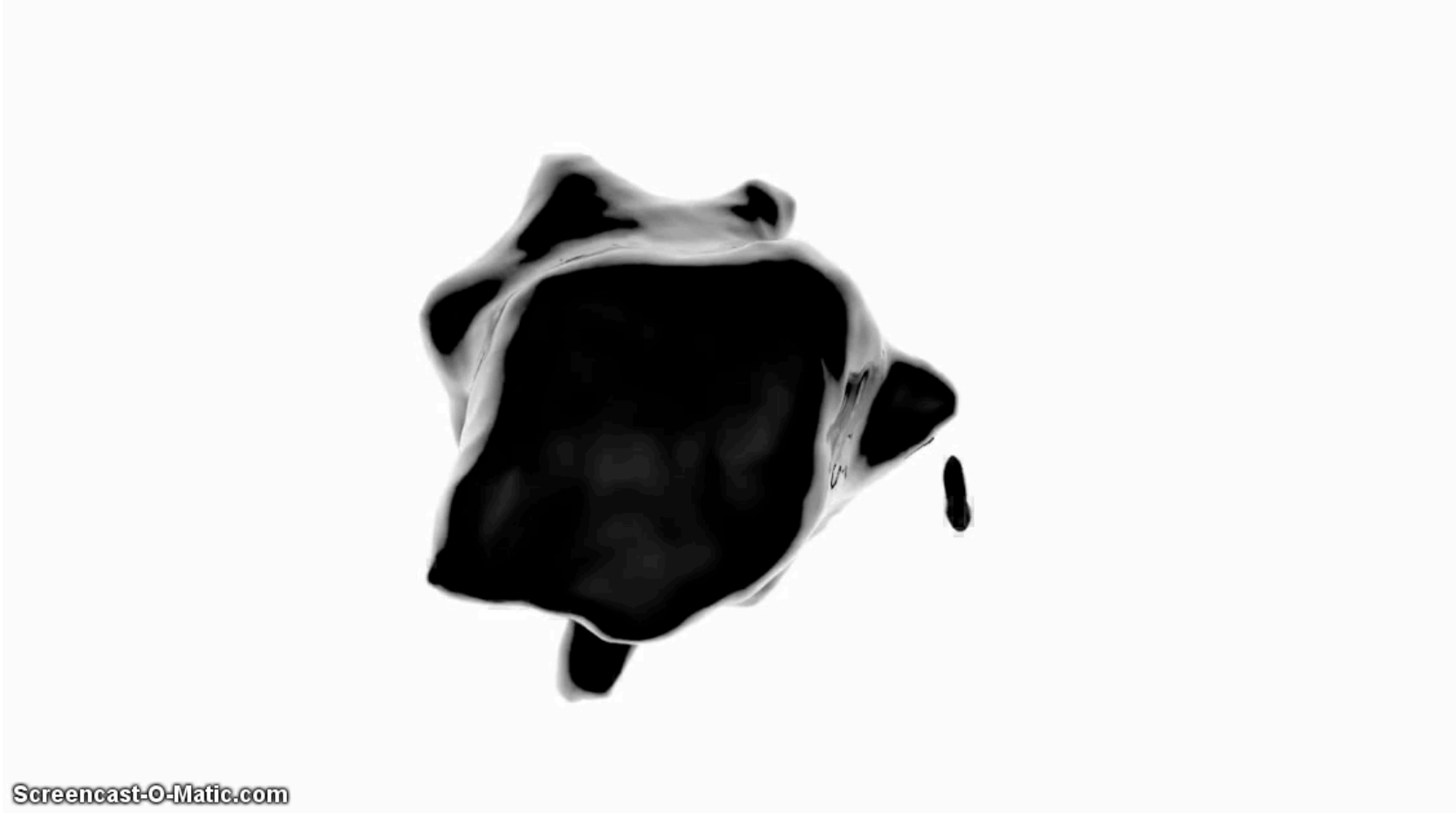
- Curve propagation based on internal and external energies
- Yield smooth segmentation results



Level sets

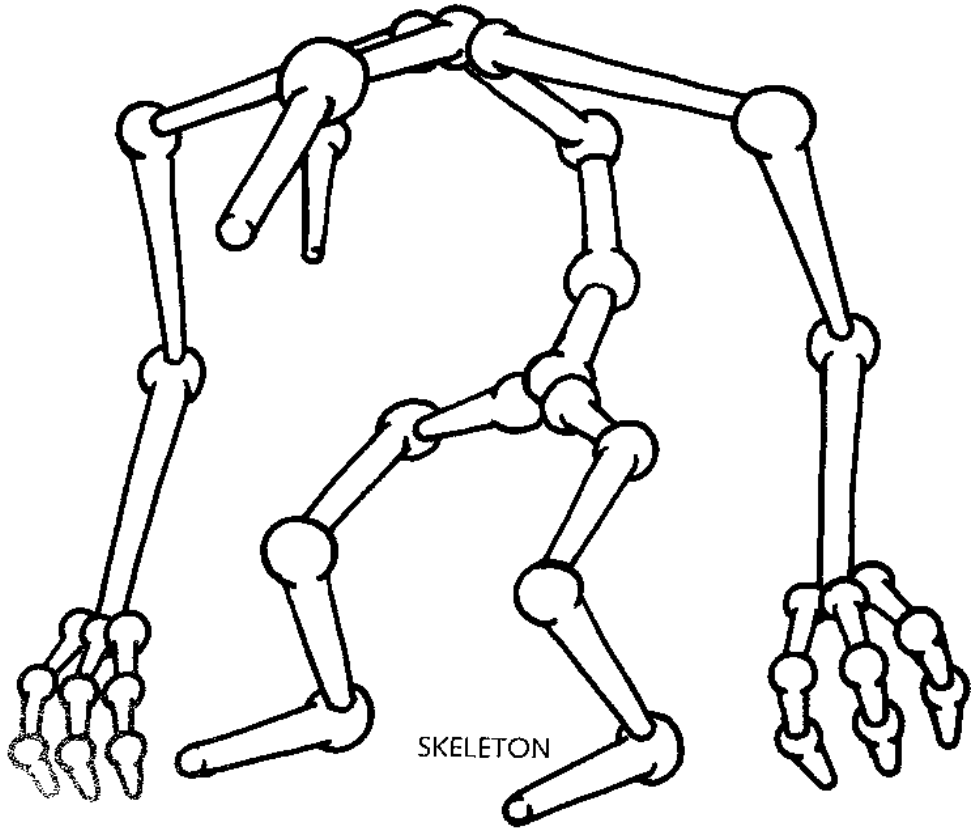
Level set morphing

- Important for computer animation.
- Simulating smooth transitions and material behavior.

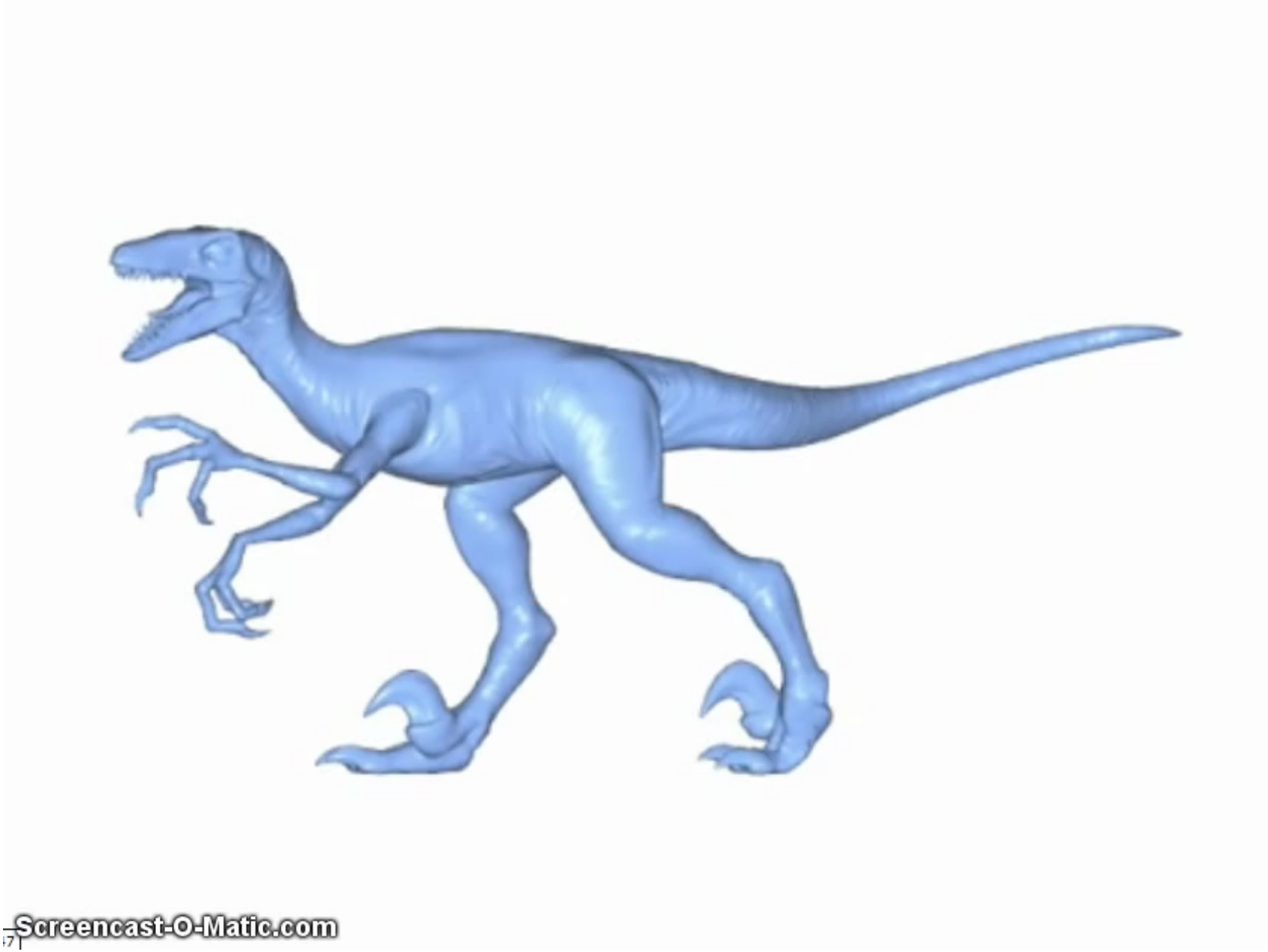


Skeletons

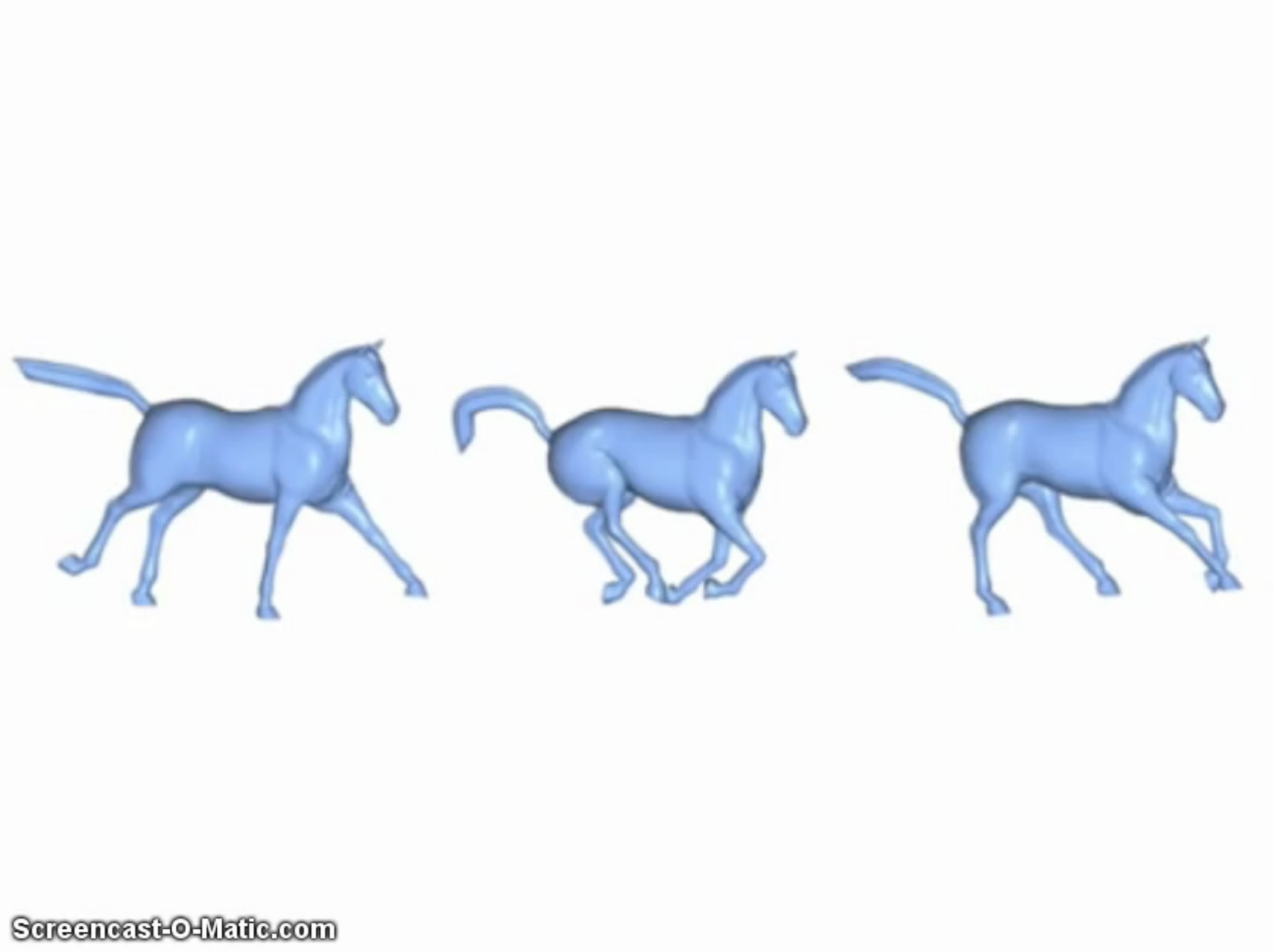
- Skeleton with joined “bones”
- Can add “skin” on top of bones
- Automatic or hand-tuned skinning



Skeletons in animation

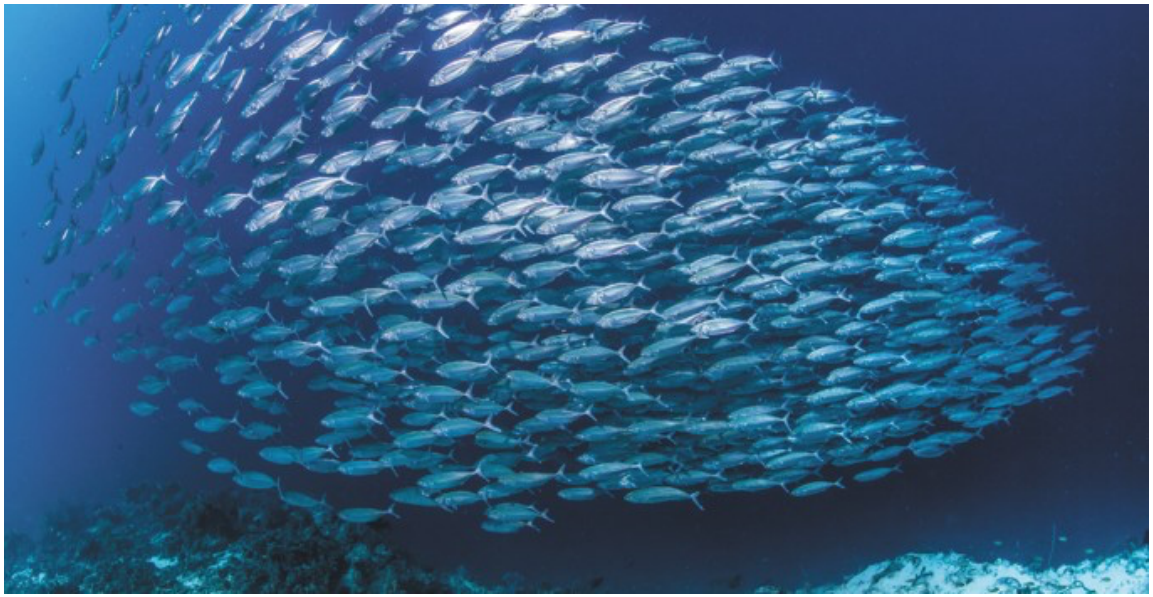


Skeletons in animation



Swarms

- Organic movement of large groups (flocks of birds, school of fish)



Swarms

- A swarm sometimes seems to behave as if it is an individual organism. Ants or wasps on a hunt for food, or on the attack, behave as if with a single mind, co-ordinating different actions with different parts of the swarm.

- A swarm (of ants/bees/locusts) often exhibits behaviours that seem clearly more intelligent than any of the individual members of it.

- The way in which swarms in some species change direction is astoundingly well co-ordinated.

- The way in which swarms in some species avoid obstacles seems to be extremely well choreographed

Boids

- Craig Reynolds is a computer graphics researcher, who revolutionised animation in games and movies with his classic paper :

- Reynolds, C. W. (1987) Flocks, Herds, and Schools: A Distributed Behavioral Model, in *Computer Graphics*, **21**(4) (SIGGRAPH '87 Conference Proceedings) pages 25-34.

- Reynold's solved the problem by trying a very simple approach, which was inspired by a sensible view of how animals actually do it.

Reynold's Rules

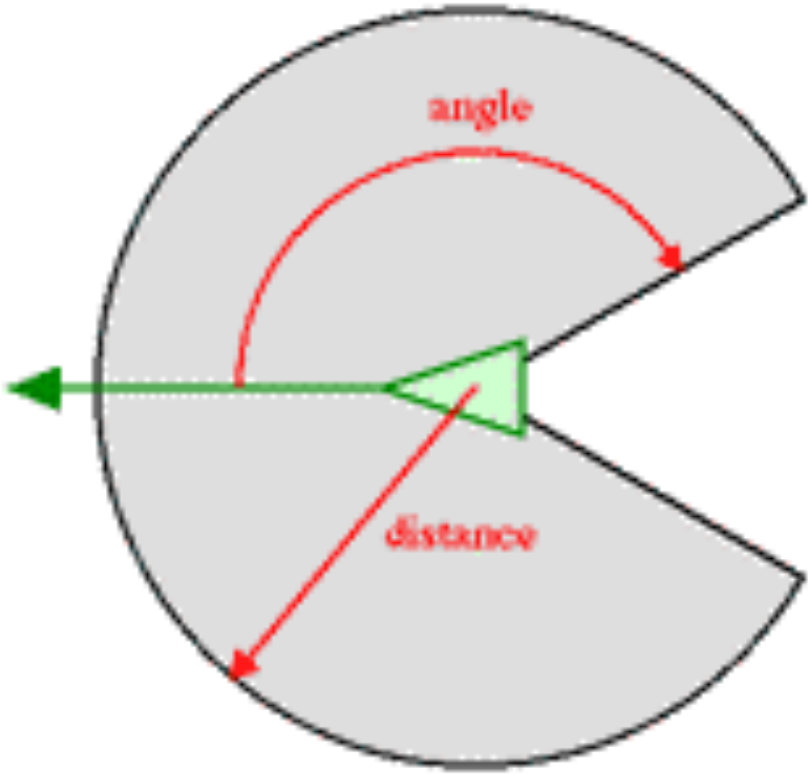
- Reynolds came up with **three simple rules** that solve this problem, resulting in entirely realistic flocking behaviour.

- To explain them, we first need to consider the **perceptual system of an individual** (which Reynolds called a boid).

- For realistic movement, you need a **realistic view of perception**, e.g. a starling's movement is not influenced at all by the flock mates that it cannot see – such as those out of its line of sight, or too far away.

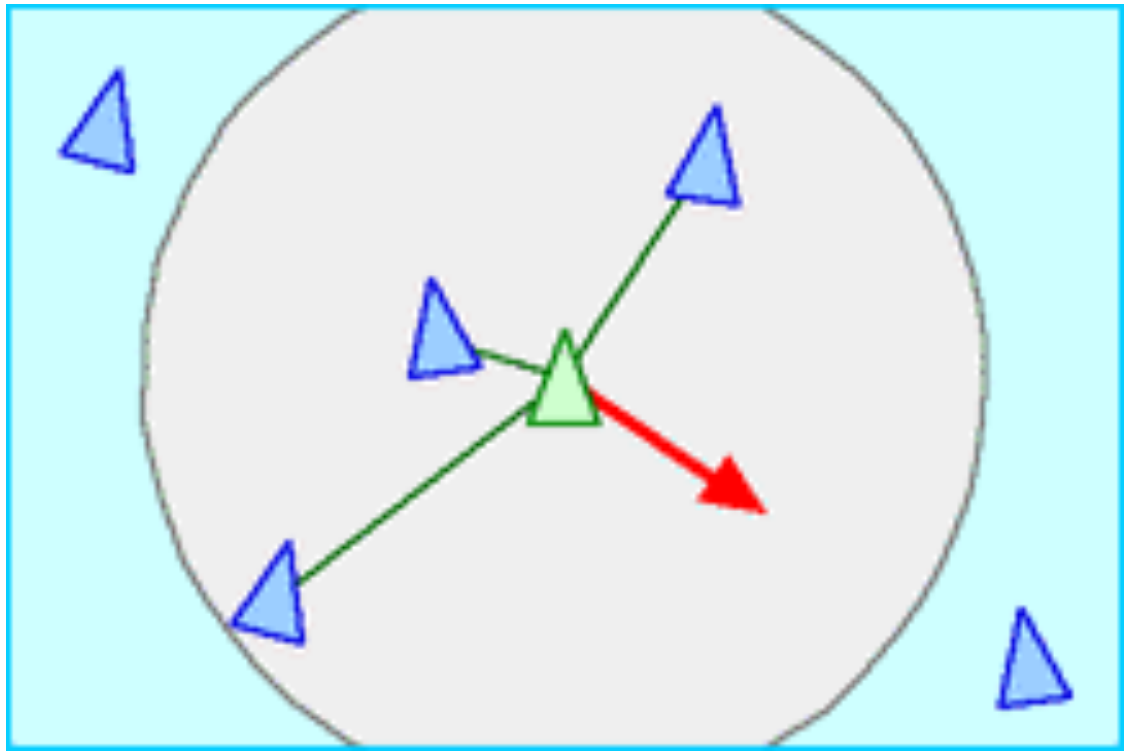
A simple sensory system

- The boid can see a certain amount ahead, and is also aware of any flockmates within limits on either side.
- Two parameters, **angle** and **distance**, define the system.
- The boid will only be influenced by those others it can sense according to these parameters.



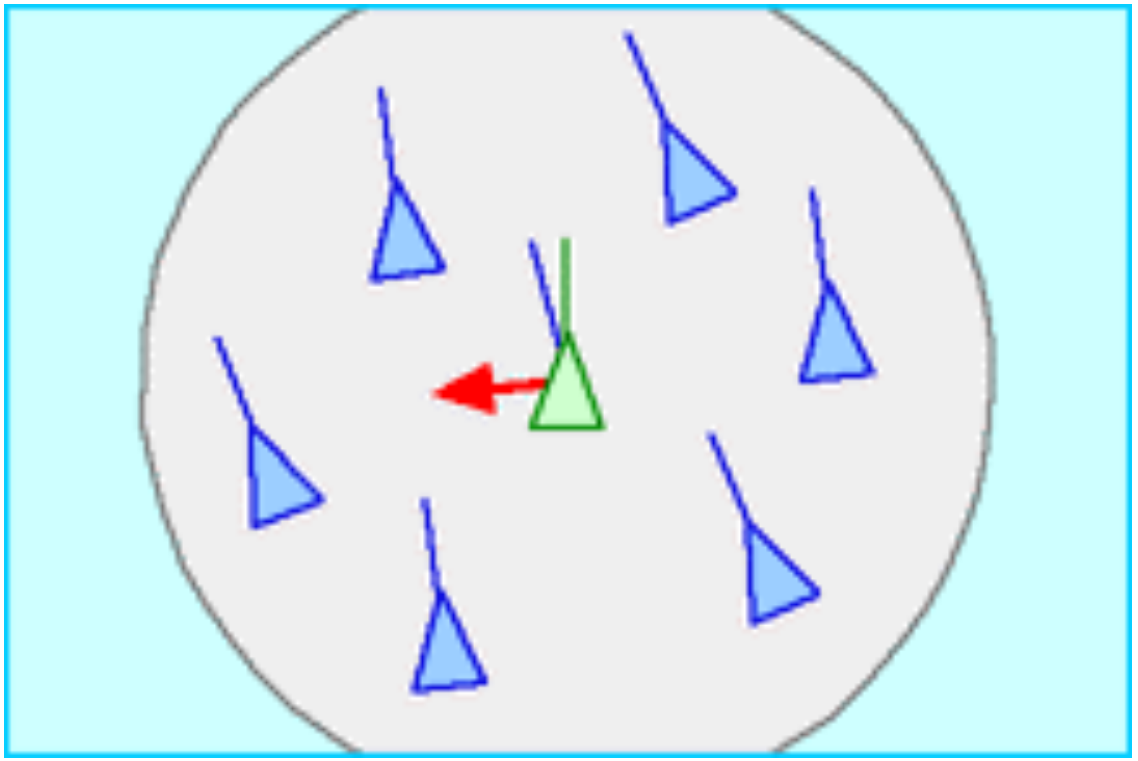
Rule 1: Separation

- At each iteration, adjust velocity to **avoid getting too close** to local (the ones it is aware of) flockmates.
- “Personal space” rule - move away sharply from very close neighbours.



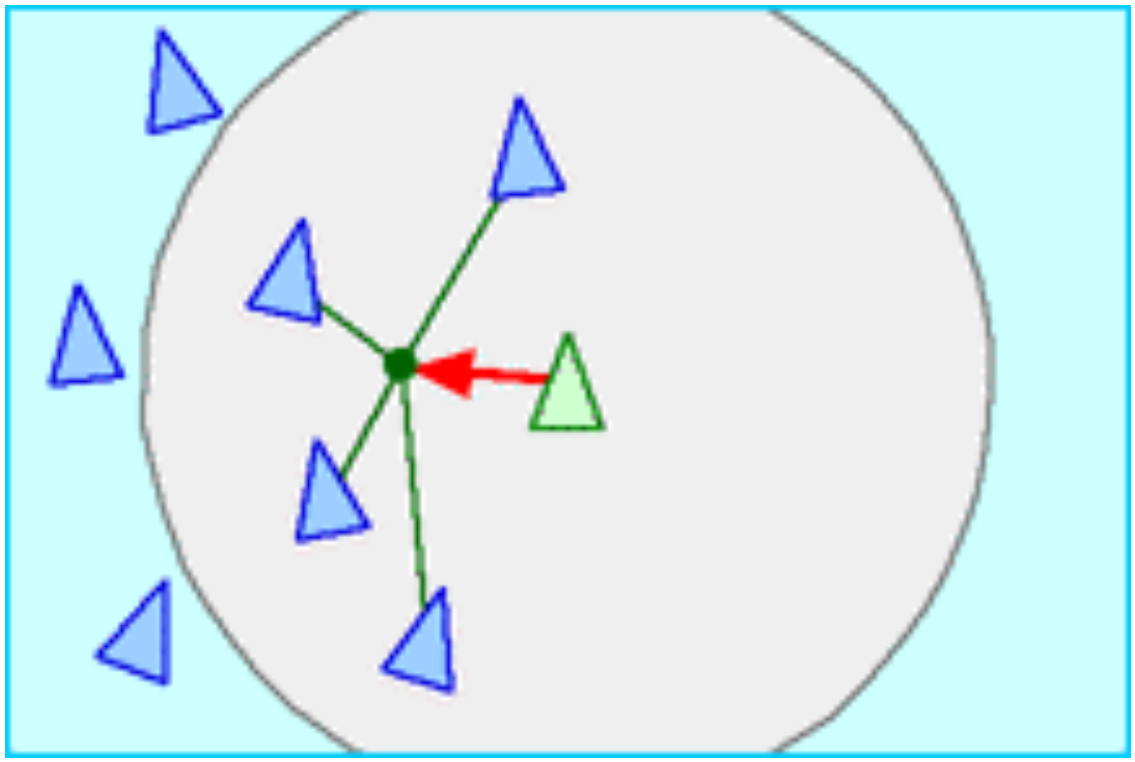
Rule 2: Alignment

- At each iteration, a boid adjusts its **velocity to match the average velocity** of its local flockmates.
- In other words, boids align themselves to their neighbors' average direction.

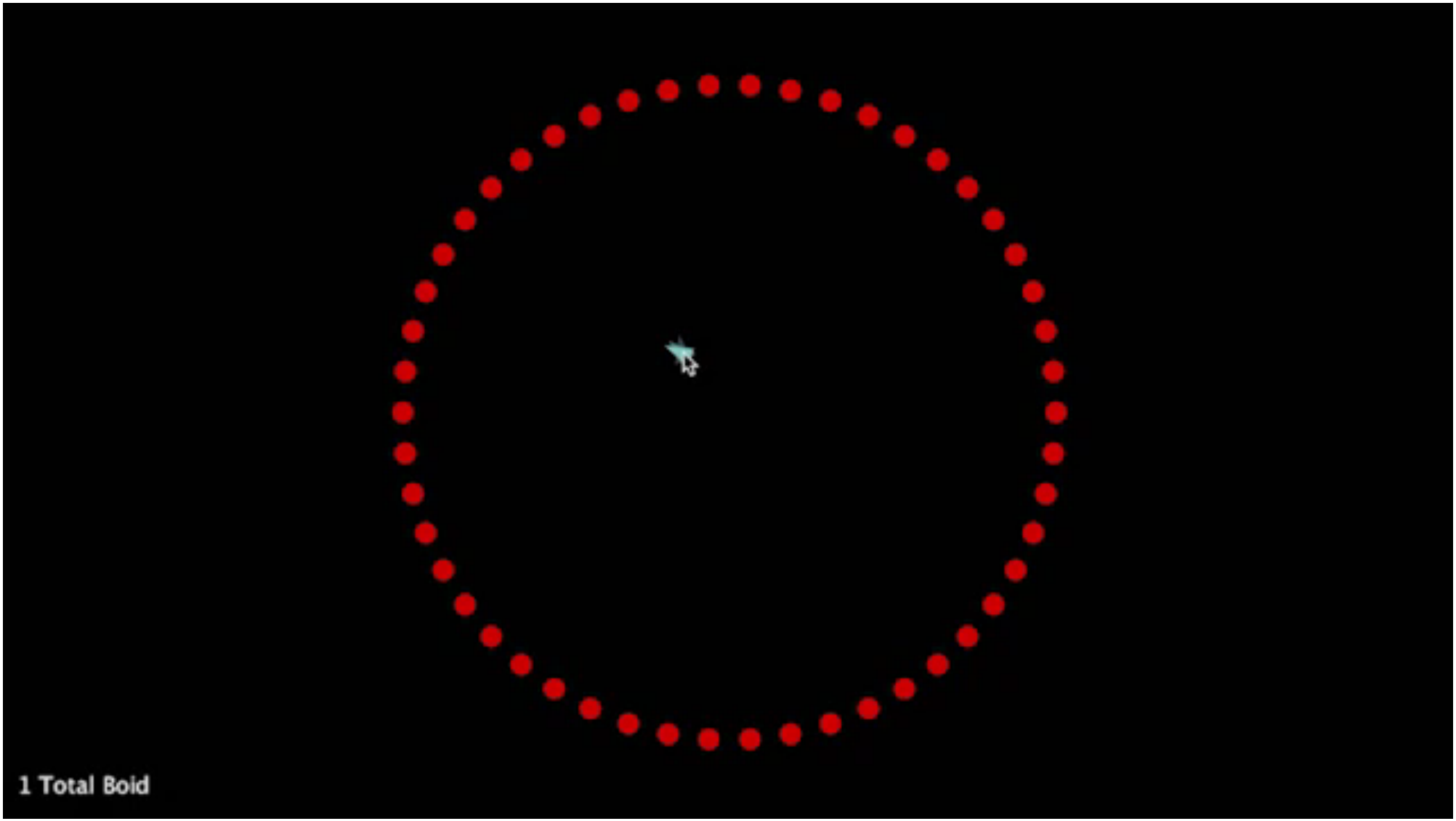


Rule 3: Cohesion

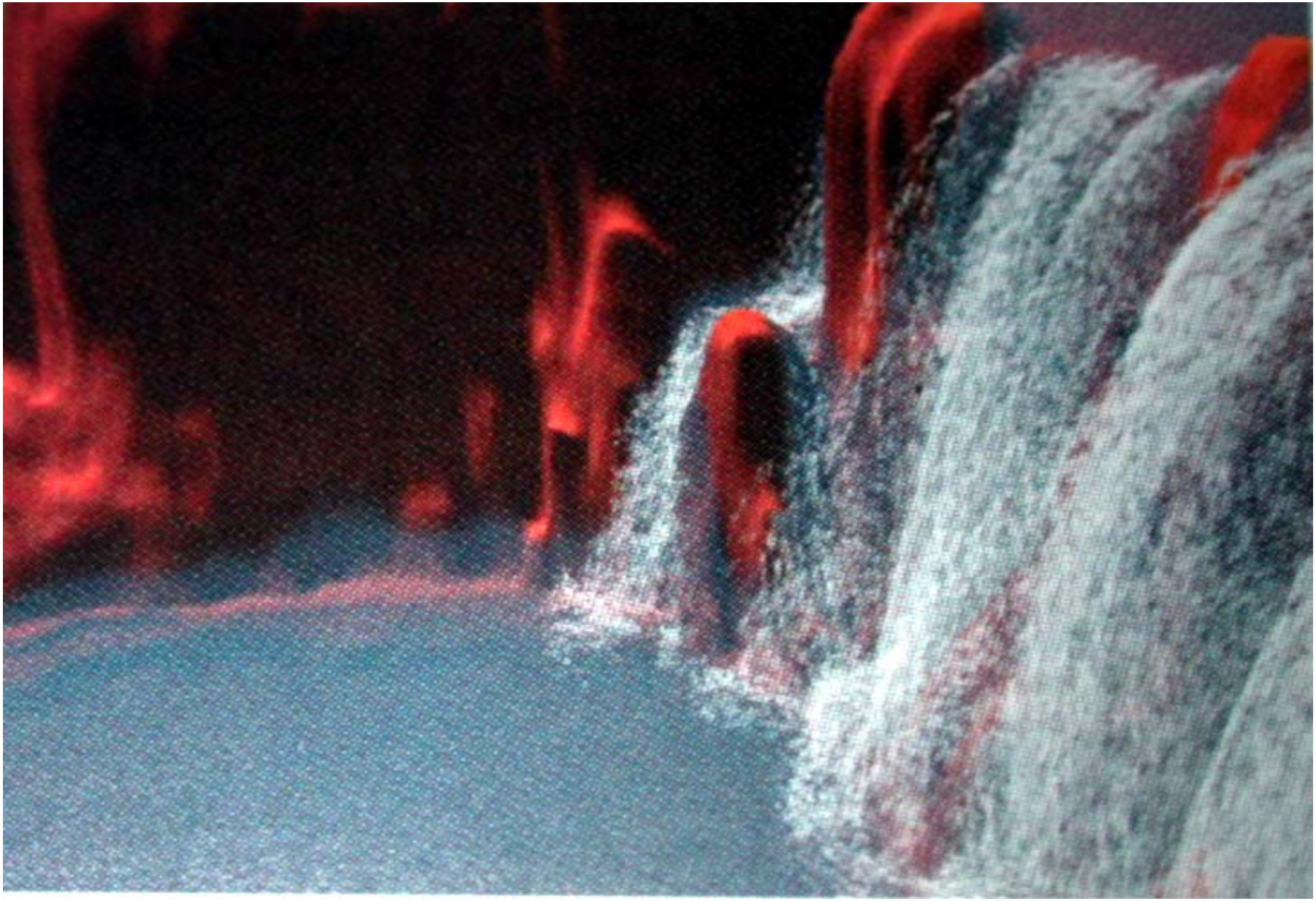
- At each iteration, a boid adjusts its **velocity towards the centroid** of its flockmates.
- In other words, boids try to move to the middle of their local neighbourhood group



Boids: Demo

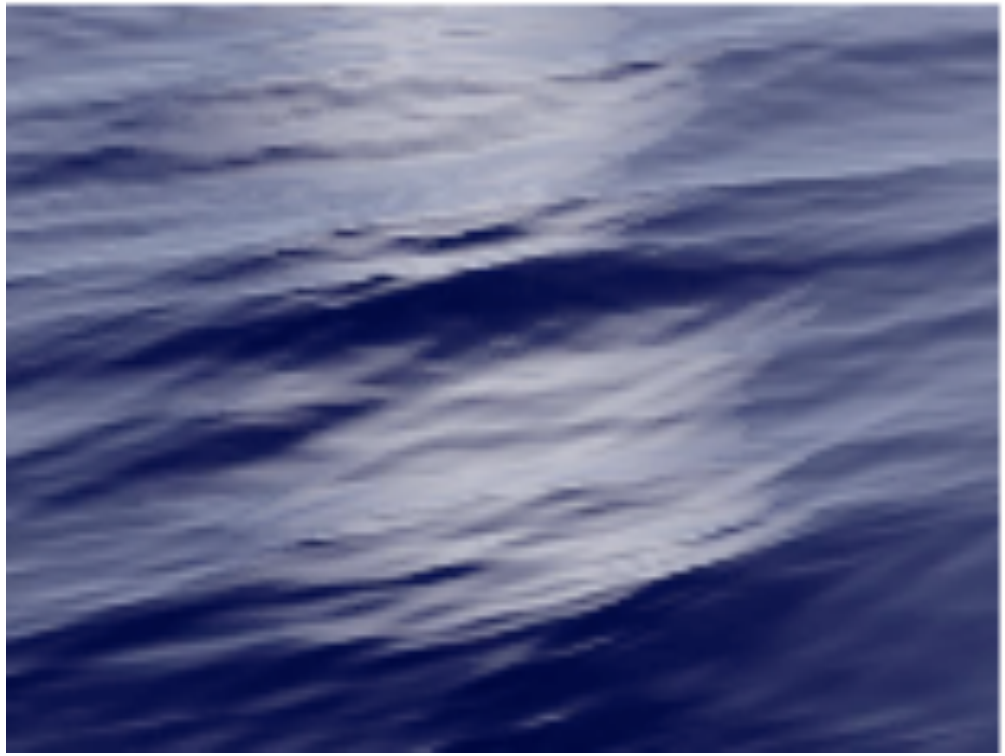


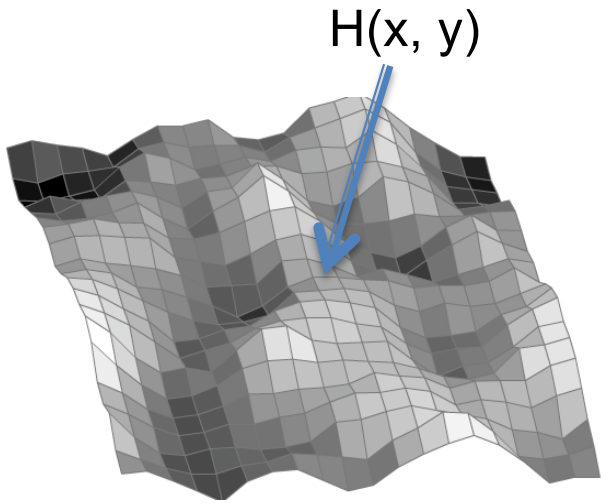
Particle systems



Real-time Fluids

- Computational Fluid Dynamics (CFD):
  - Apply only to the surface of the water (cheaper to compute than the whole volume)
- Other techniques:
  - Superimpose sine waves of a variety of amplitudes and directions.
  - Heightfield approximations: if the surface is the only interest, it can be represented using a 2d heightfield and animated by 2d wave equations with interaction forces.



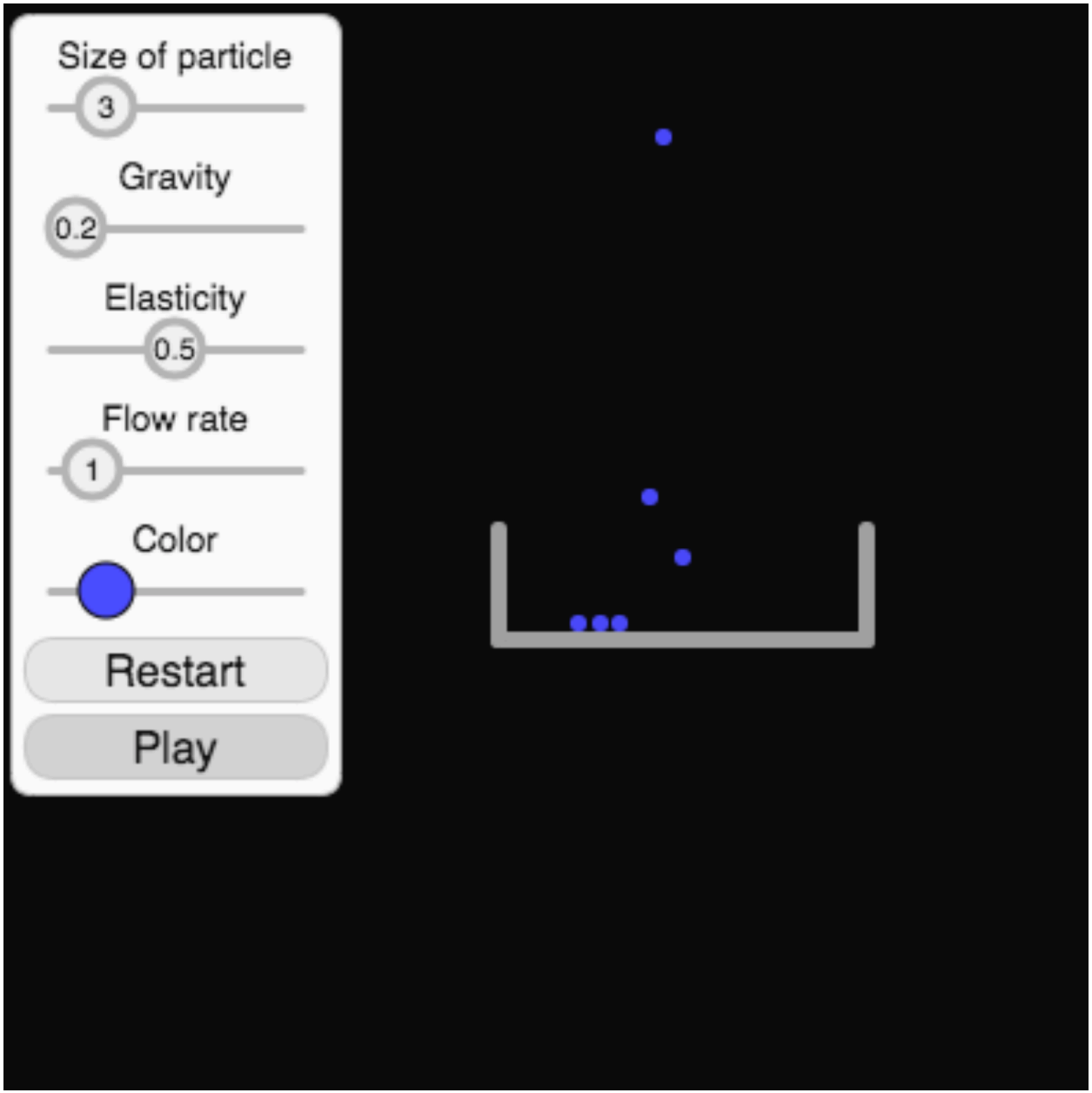


Real-time Fluids

- Particle systems: This approach is good at simulating a small amount of water such as a puddle, a bubble, or splashing fluids

Fluid Simulation Example

- Gravity: pull of force to the ground.
  - Value 0 means no gravity, so particles fly away.
- Elasticity: loss of energy after the contact with the surface.
  - Value 1 means that particle will bounce indefinitely, value 0 means it will not bounce at all.

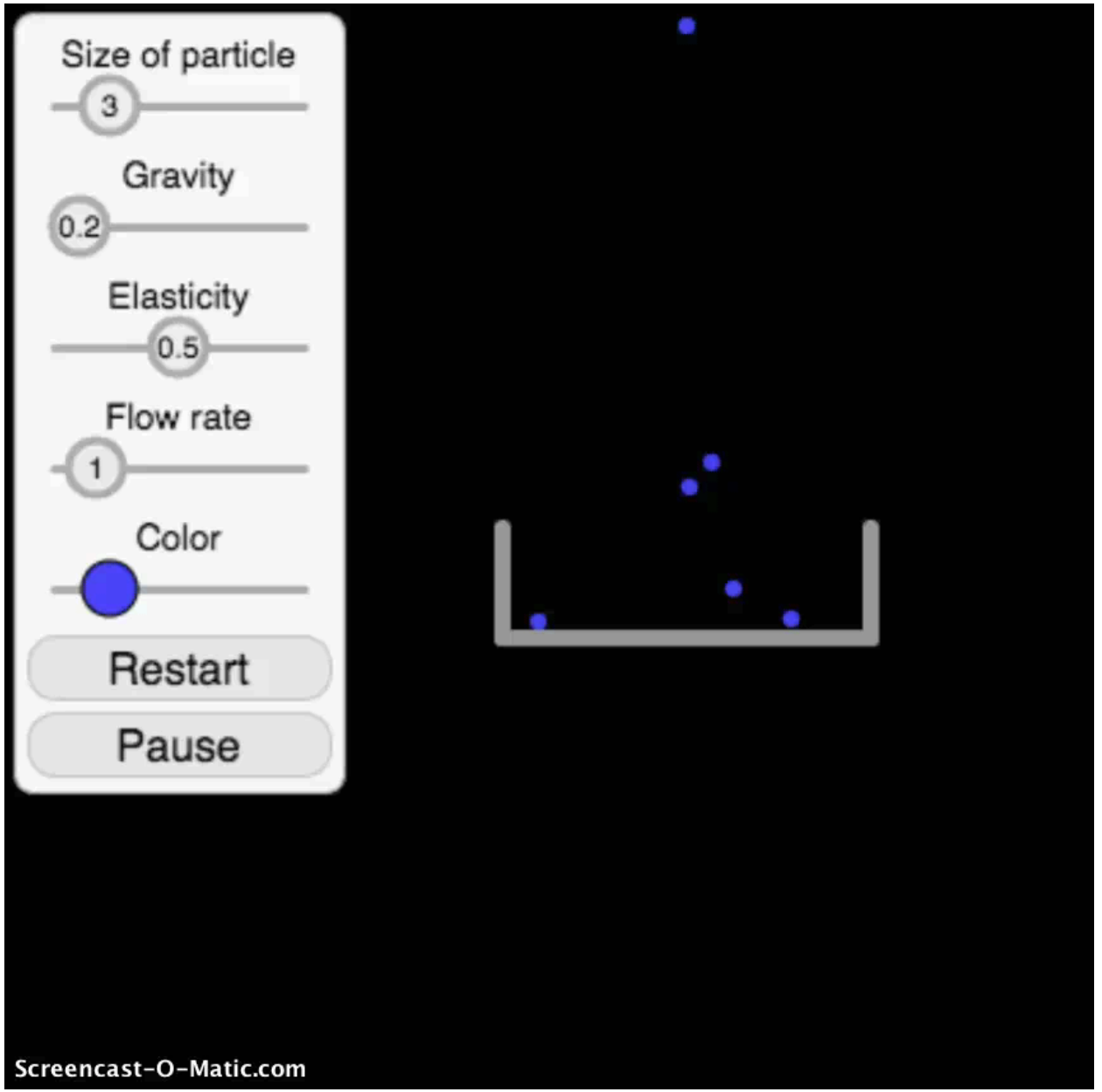


https://www.khanacademy.org/partner-content/pixar/effects/particle/pi/water-simulation

Fluid Simulation Example

- Gravity: pull of force to the ground.
  - Value 0 means no gravity, so particles fly away.
- Elasticity: loss of energy after the contact with the surface.
  - Value 1 means that particle will bounce indefinitely, value 0 means it will not bounce at all.

https://www.khanacademy.org/partner-content/pixar/effects/particle/pi/water-simulation

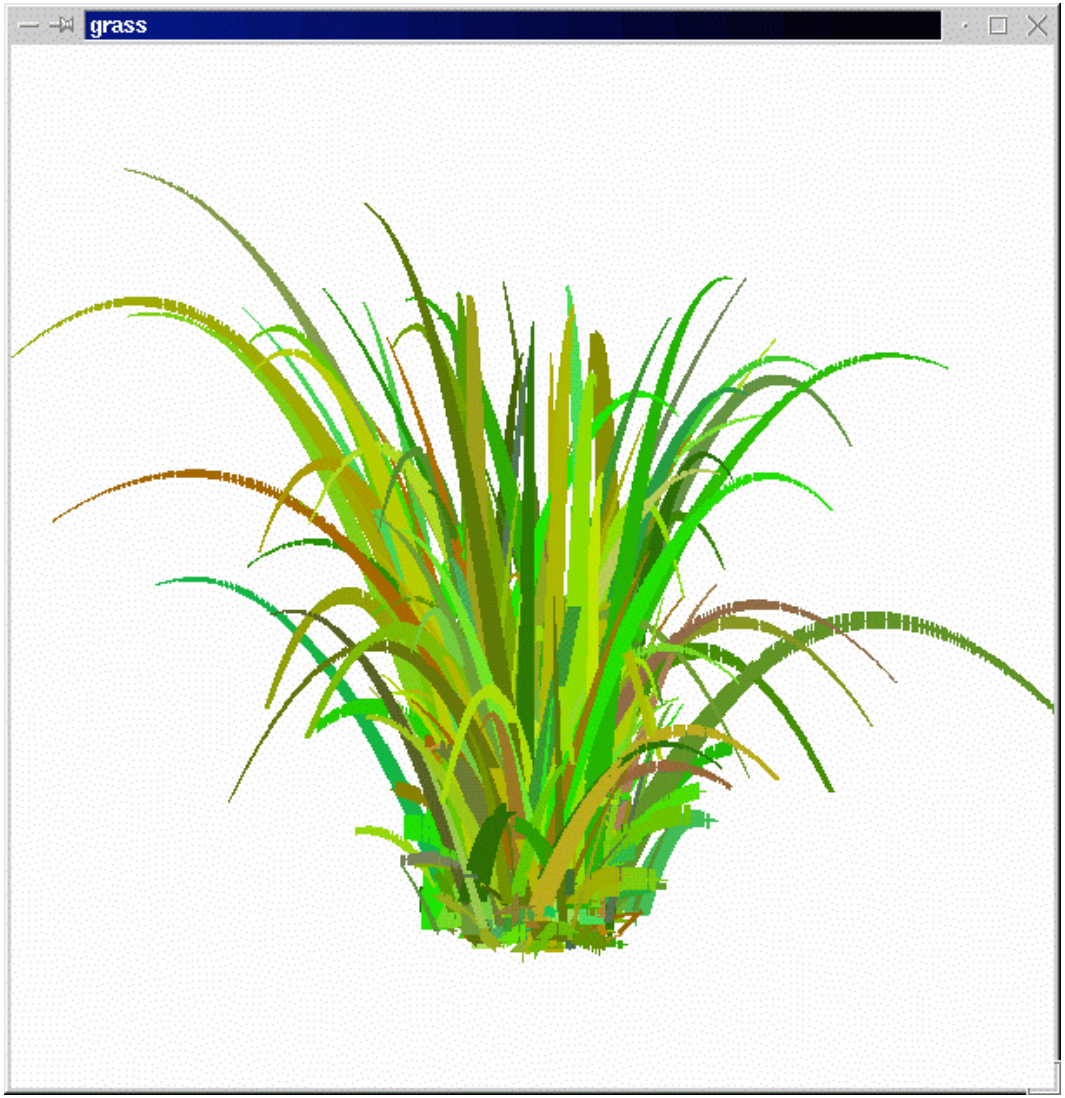


Types of Particle Systems

- Stateless Particle System
  - Particle data is computed from birth to death by a closed form function defined by a set of start values and a current time (does not react to dynamic environment)
- State Preserving Particle System
  - Uses numerical iterative integration methods to compute particle data from previous values and changing environmental descriptions.

Stateless Particle Systems

- Stateless Simulation – Computed particle data by closed form functions
  - No reaction on dynamically changing environment.
  - No storage of varying data



Particle Life Cycle

- Generation – Particles are generated randomly within a predetermined location.
- Particle Dynamics – The attributes of a particle may vary over time (e.g. color gets darker as particle cools off after explosion)
- Extinction
  - Age – Time the particle has been alive
  - Lifetime – Maximum amount of time the particle can live.
- Premature Extinction:
  - Running out of bounds (e.g. particle gets too dark to see)
  - Hitting an object (ground)
  - Attribute reaches a threshold (particle becomes transparent)

Rendering

- Expensive to render thousands of particles
- Simplify: avoid hidden surface calculations
  - Each particle has small graphical primitive (blob)
- Particles that map to the same pixels are additive
  - Sum the colors together
  - No hidden surface removal
  - Motion blur is rendered by streaking based on the particles position and velocity

State Preserving Algorithm

Rendering Passes:

- Process Birth and Deaths
- Update Velocities
- Update Positions
- Sort Particles for alpha blending (optional, takes multiple passes)
- Render particles

Birth and Death

- Birth = allocation of a particle
  - Associate new data with an available index
  - Serial process (offloaded to CPU)
  - Initial particle data determined (on CPU)

- Death = deallocation of a particle
  - Frees the index associated with particle
  - Extra pass to move any dead particles
  - In practice particles fade out or fall out of view

Update Velocities

- Global Forces
  - Wind
  - Gravity
- Local Forces
  - Attraction
  - Repulsion
- Velocity Damping
- Collision Detection

$$F = \sum_{0}^{N} F_n$$

$$F = ma$$

$$a = \frac{F}{m}$$

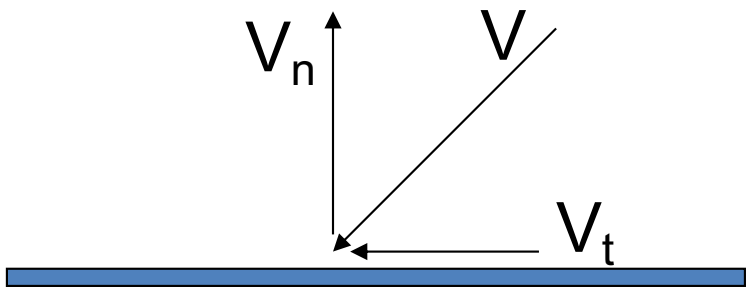
If $m = 1$: $F = a$

Velocity Damping and Un-damping

- Damping
  - Imitates viscous materials or air resistance
  - Implement by downward scaling velocity
- Un-damping
  - Self-propelled objects (bee swarms)
  - Implement by upward scaling velocity

Collisions

- Collisions against simple objects
  - Walls
  - Bounding Spheres
- Collision against complex objects
  - Terrain
  - Other objects
  - Terrain is usually modeled as a texture-based height field



v_n = normal component of velocity

v_t = tangent component of velocity

$$V = (1-\mu)v_t - \varepsilon v_n$$

μ = dynamic friction (affects tangent velocity)

ε = resilience (affects normal velocity)

Update Positions

- Euler Integration

  $p = p_{prev} + v * \Delta t$

- Verlet (for simple velocity updates, saves a rendering pass for computing velocity)

$p_{i+1} = p_i + (p_i - p_{i-1}) + a * \Delta t^2$