

Computer Graphics

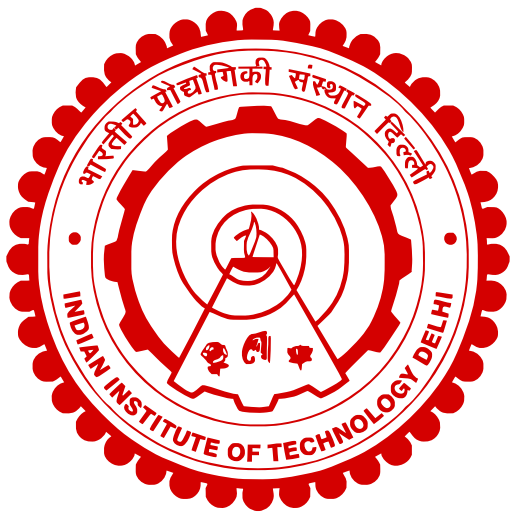
Animation

Subodh Kumar, Dept of Computer Sc. & Engg., IIT Delhi

Computer Animation



- Rendered sequence $\Rightarrow$ Object and Camera transformation
- Frame rate needs to be ≥ 10 Hz to perceive motion at all
 - $\Rightarrow$ Some can perceive flickering at 80 Hz
- Film: 24 Hz
 - $\Rightarrow$ Virtual reality requires 60+ Hz
- Frames form a sampling in time $\Rightarrow$ aliasing
- Ideally, compute average image over a finite time interval $\rightarrow$ motion blur



The 11 Principles

of artistic animation

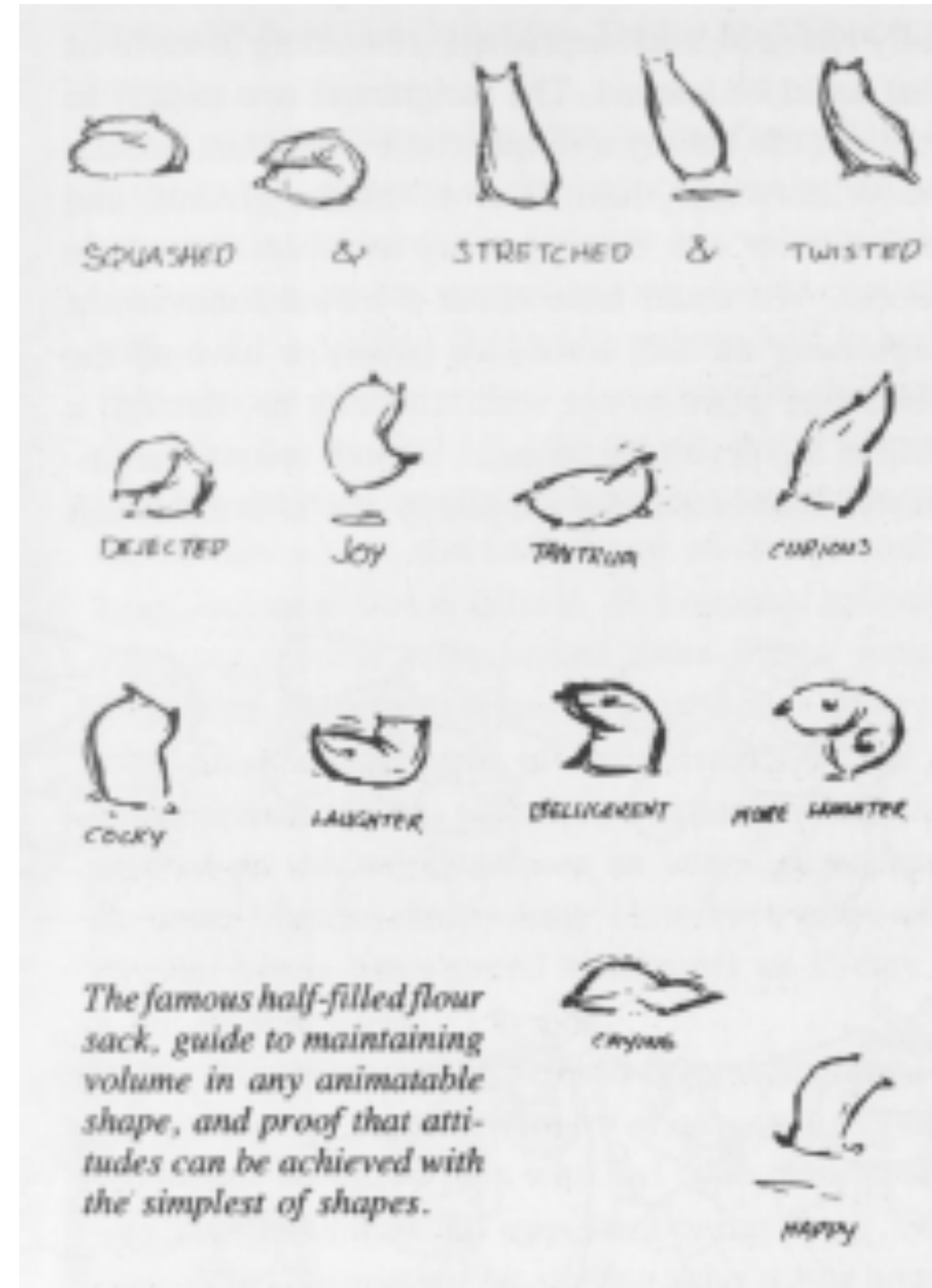
- [John Lasseter (Pixar) 1987]
 - ➔ Squash-and-Stretch, Timing, Anticipation, Follow Through and Overlapping Action, Straight Ahead Action and Pose-to-Pose Action, Slow In and Out, Arcs, Exaggeration, Secondary Action, Appeal
- Principles driven by
 - ➔ Directing viewer's attention, anticipation
 - ➔ Smoothing the motion ('easy' on the eyes)
 - ➔ Conveying emotion through motion

Conventional Animation

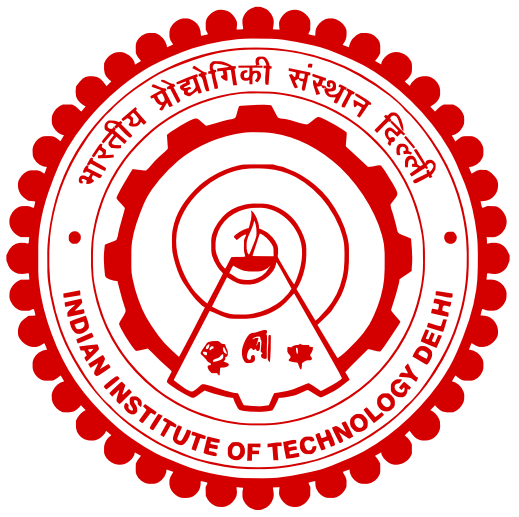


The flour sack principle

Stretch and Squash using
half filled bag of flour



Animation Techniques



- Keyframing

- ➔ Animator draws key positions (frames)
- ➔ Intermediate frames filled in: inbetweening
- ➔ aka interpolation
- ➔ Computer-aided these days

- Motion capture

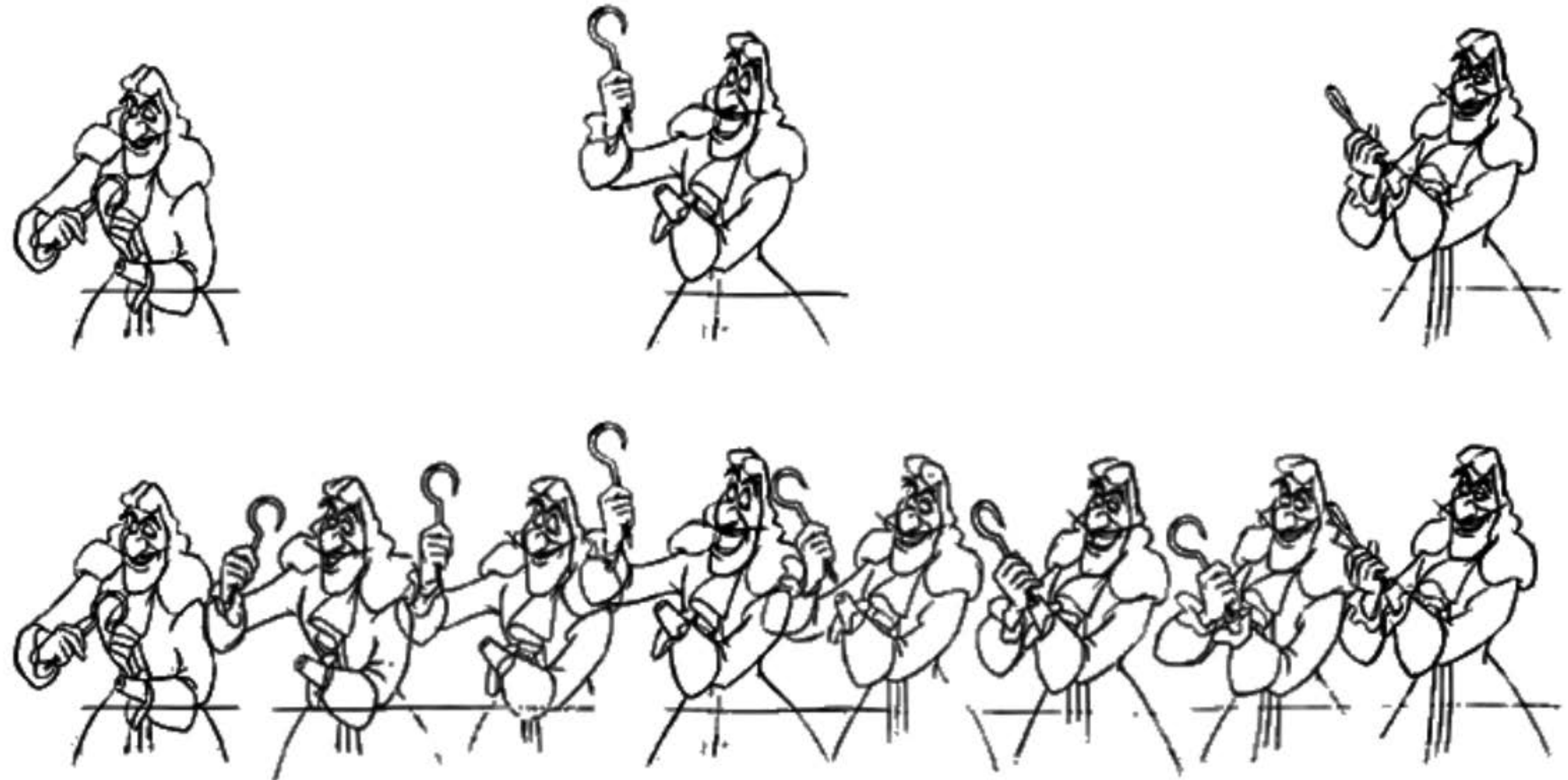
- ➔ System to capture actors' motions
- ➔ Captured data drives a virtual character

- Procedural animation

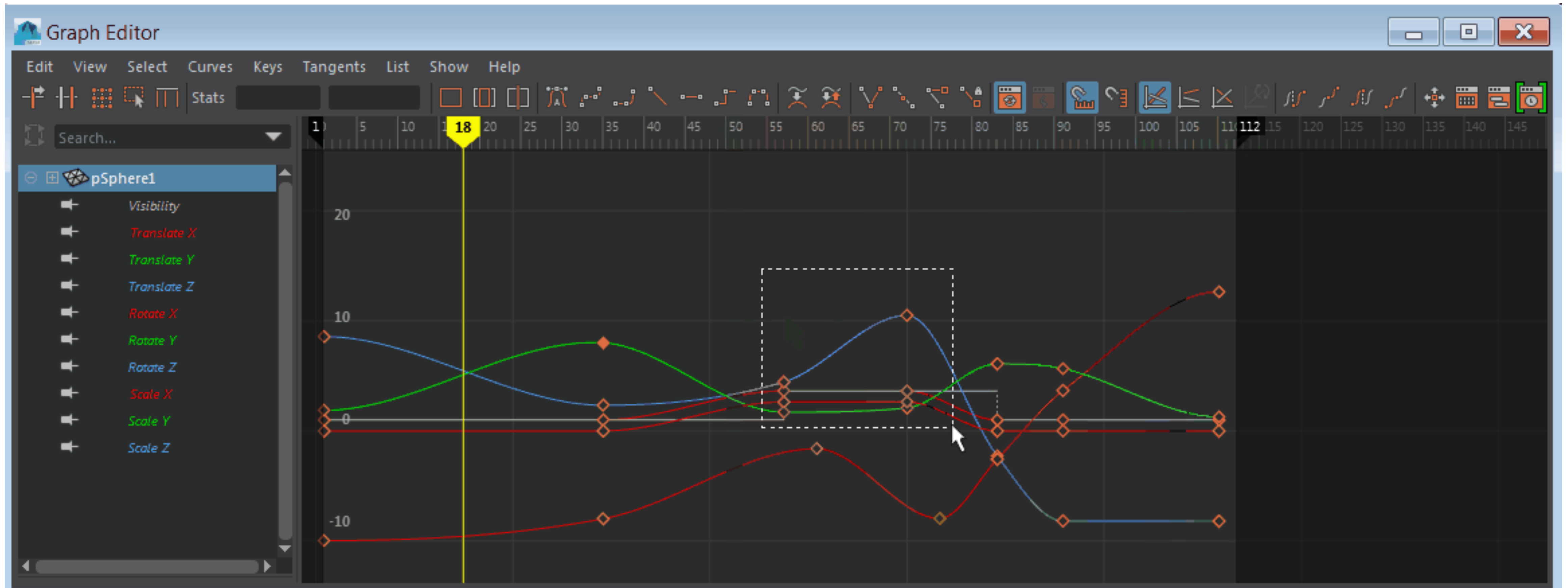
- ➔ A set of equations or rules
- ➔ Geometric and behavioral

Key-frame Animation

- In traditional (hand-drawn) animation:
- Lead animator creates **keyframes**
- Assistant creates in-between frames ("**tweening**")



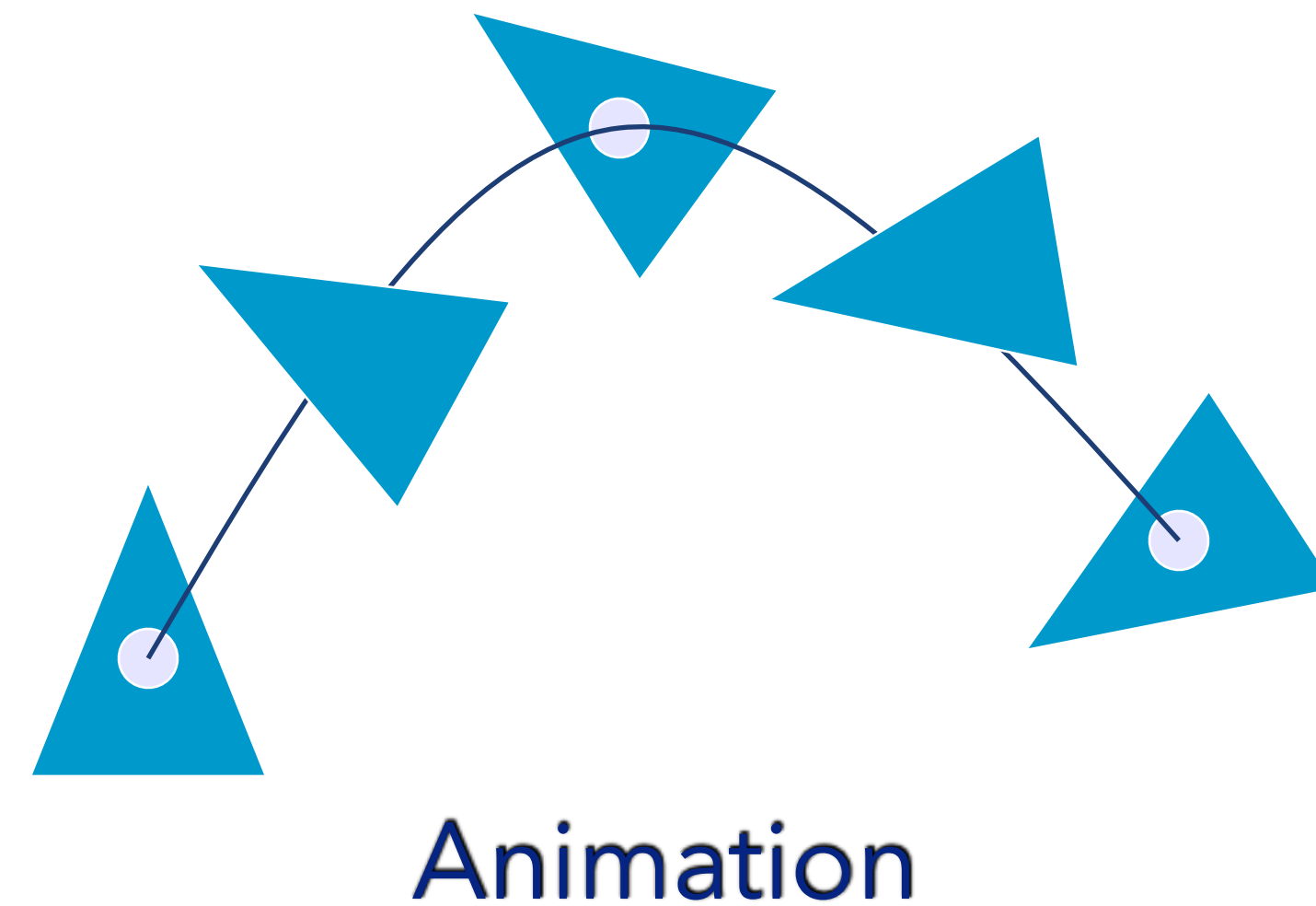
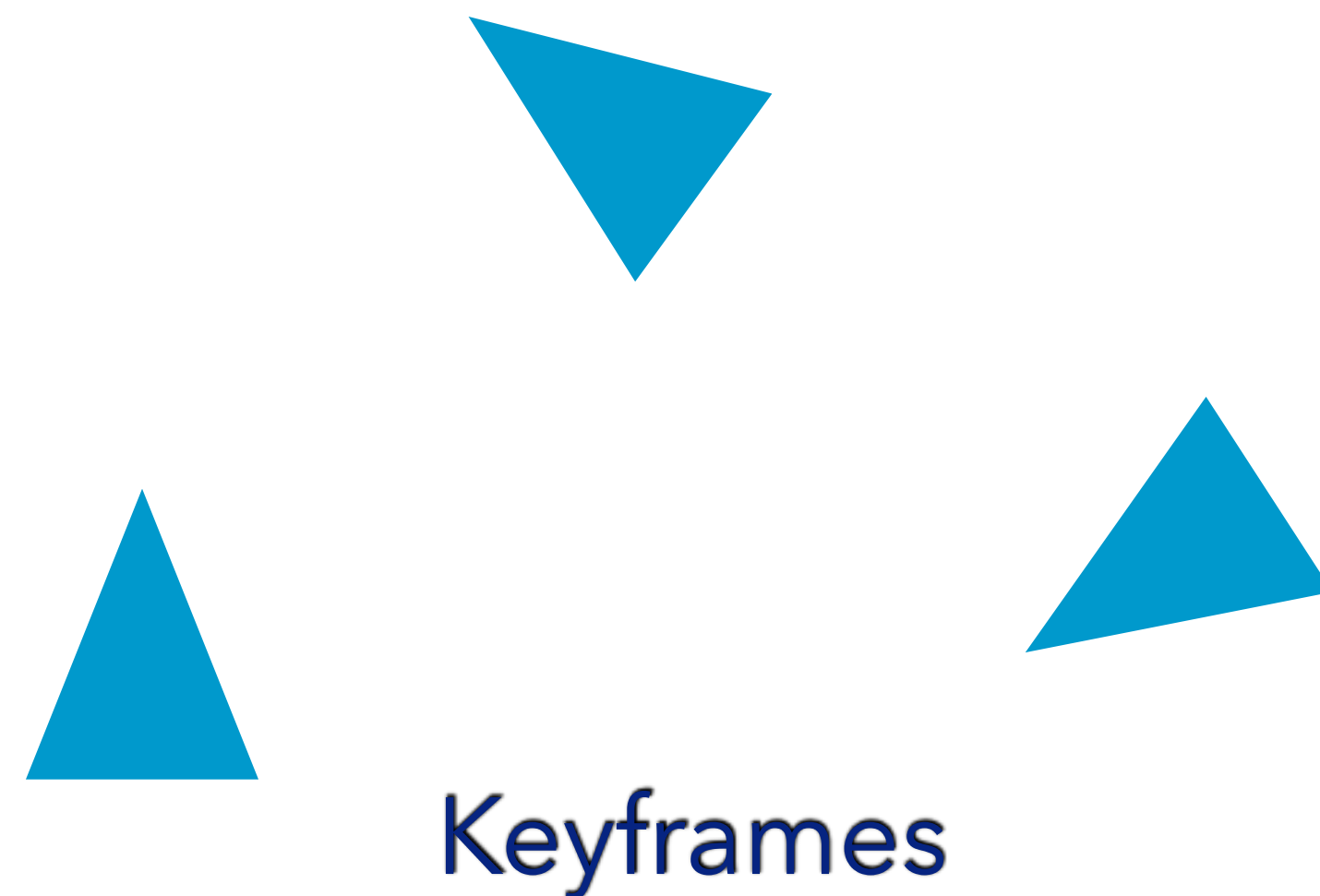
Thomas & Johnston, *The Illusion of Life*



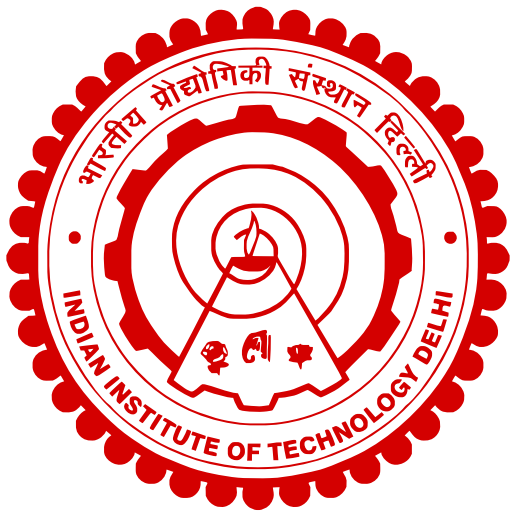
Autodesk Maya's Graph Editor

Interpolation

- Interpolating splines are smooth curves passing given control points
- Parameter value t is a function of time
 - ➔ Parameterization controls speed



Interpolation

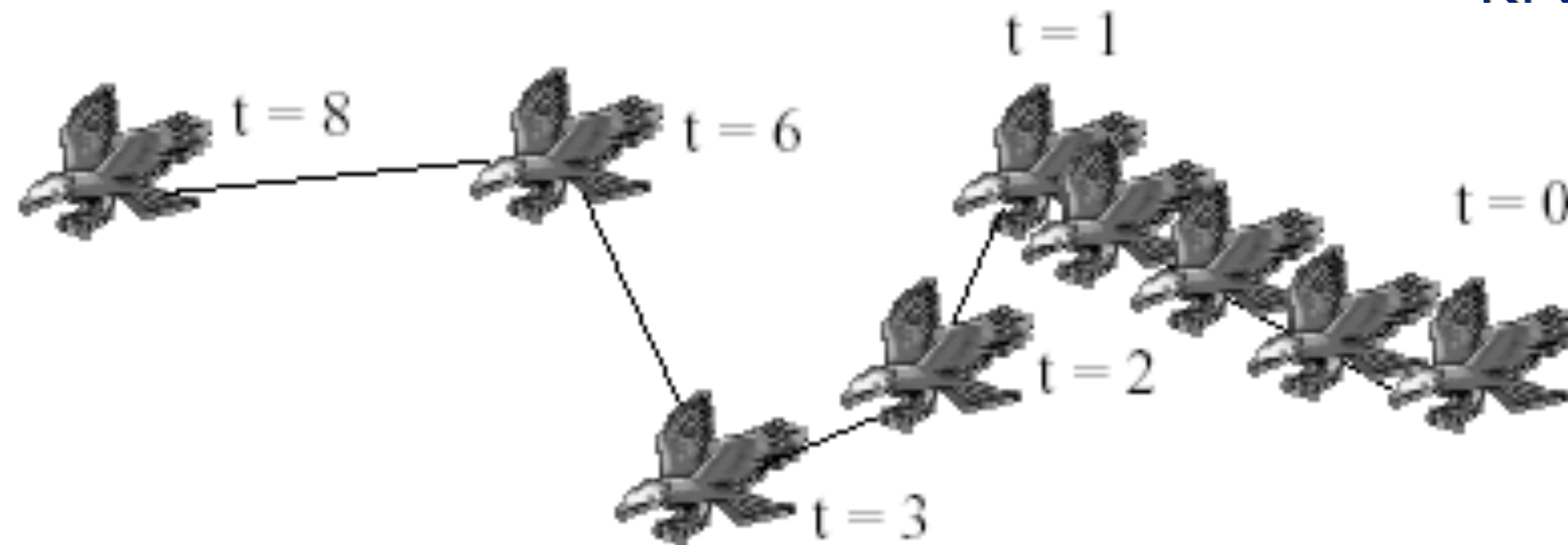


- What is interpolated:
 - ➔ Position, Orientation, Scale, Deformation, speed...
- Can you interpolate matrices?
 - ➔ Use quaternions to represent rotation and interpolate between quaternions

Linear Interpolation of Key Frames



- Discontinuities : spatial, temporal
 ➔ *Splines* may be used
- Unrealistic results



●
KF2

●
KF1

●
KF3

KF2

○ ○

○

○

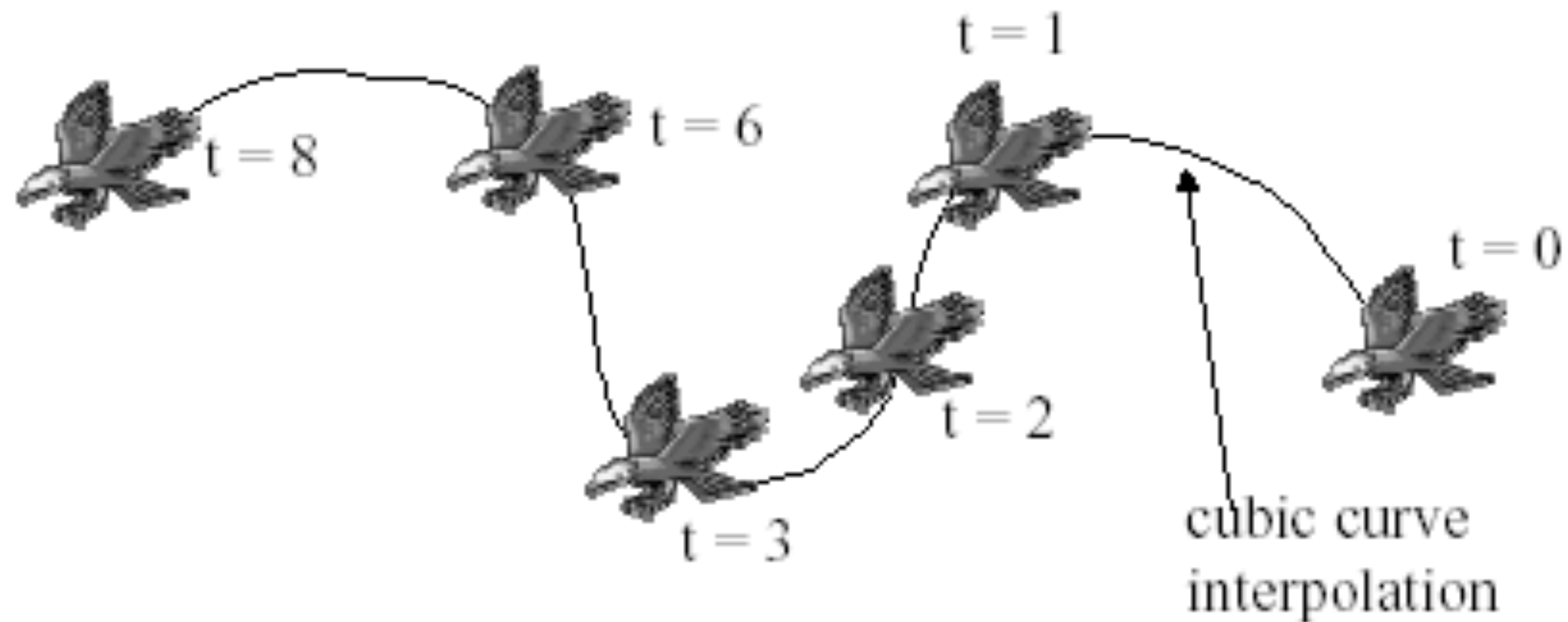
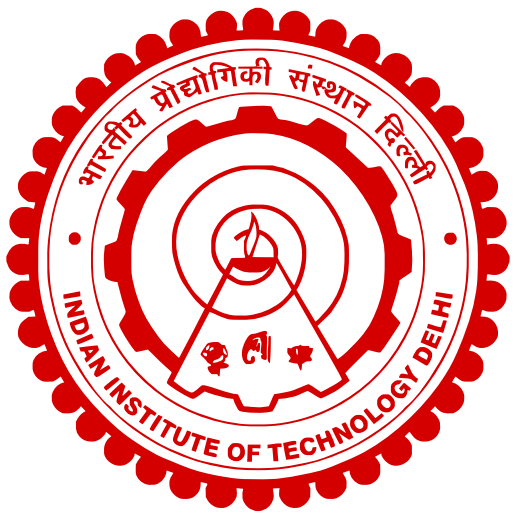
○

○

●
KF1

●
KF3

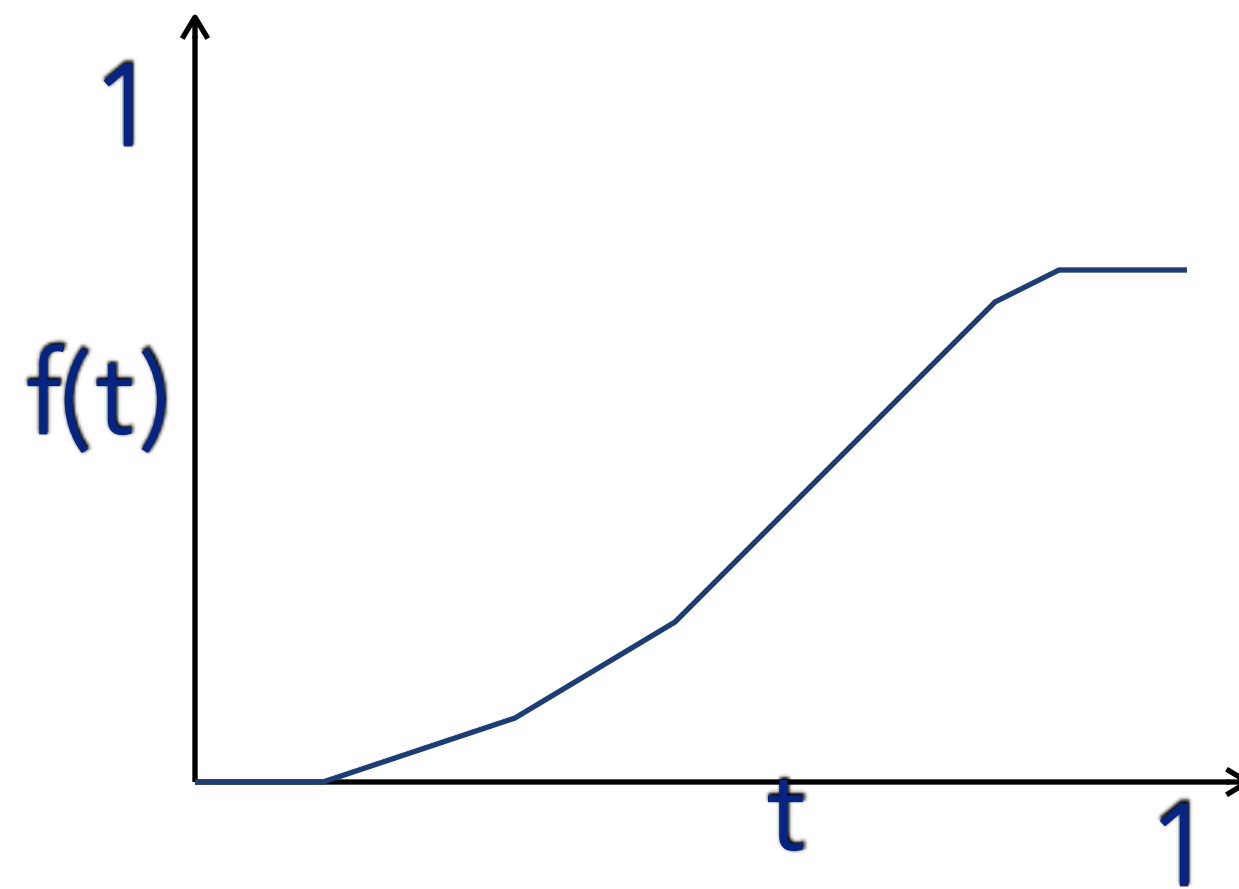
Spline Interpolation of Key Frames





Key Frame Interpolation

Using other functions (slow-in, slow-out)



v_s : attribute at start frame

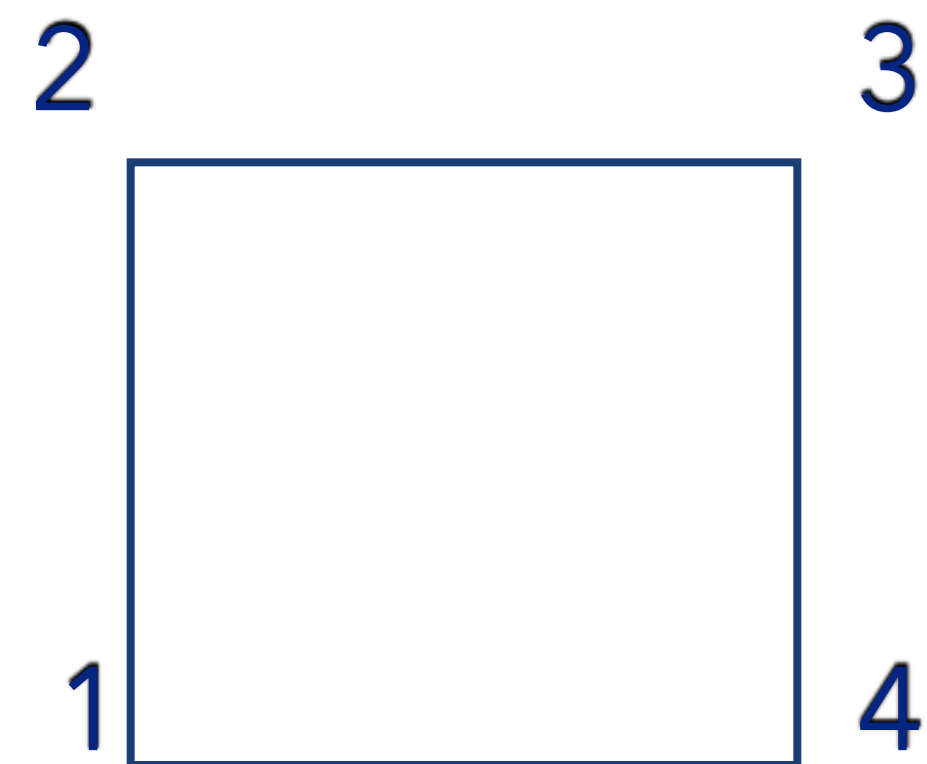
v_e : attribute at end frame

v_i : attribute at intermediate frame

$$= (1-f(t)) v_s + f(t)v_e$$



Problem with Interpolation

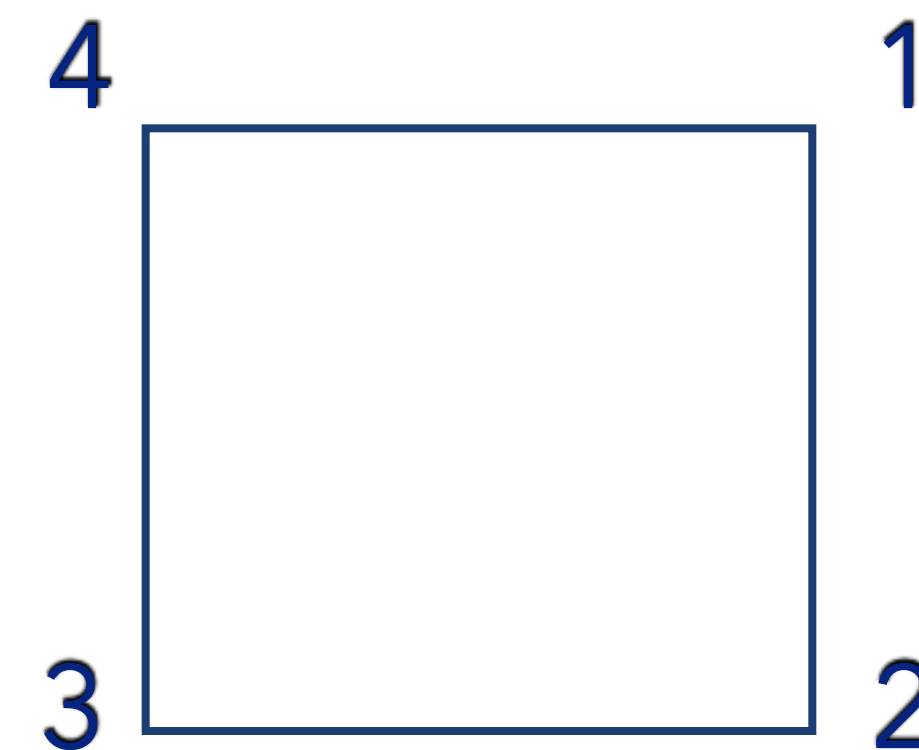


$KF1(t=0)$

??

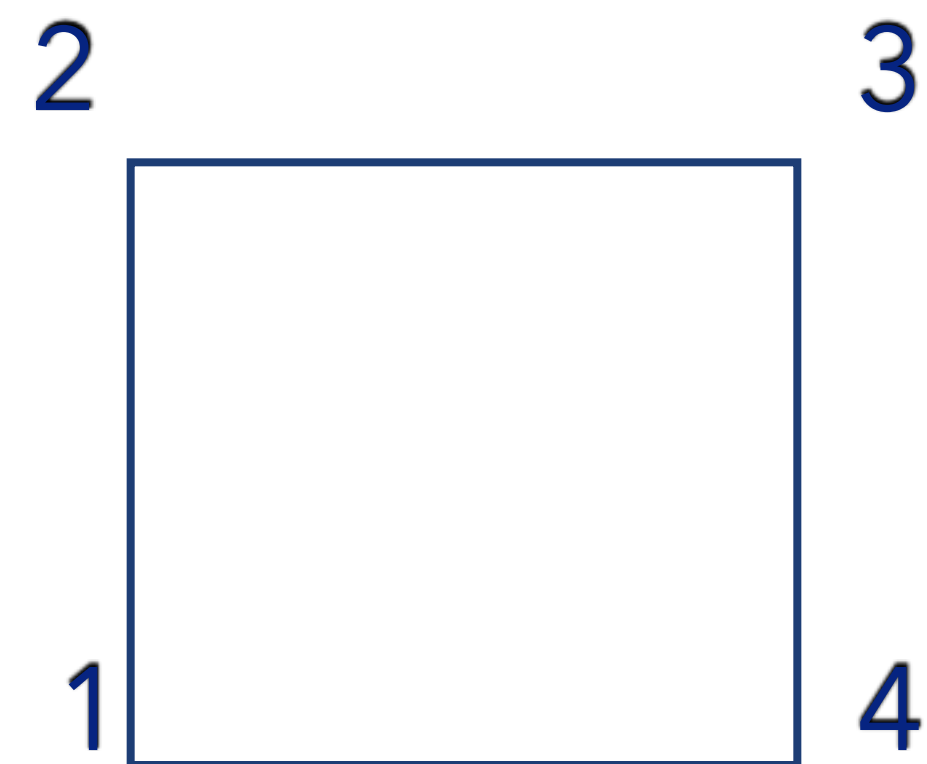
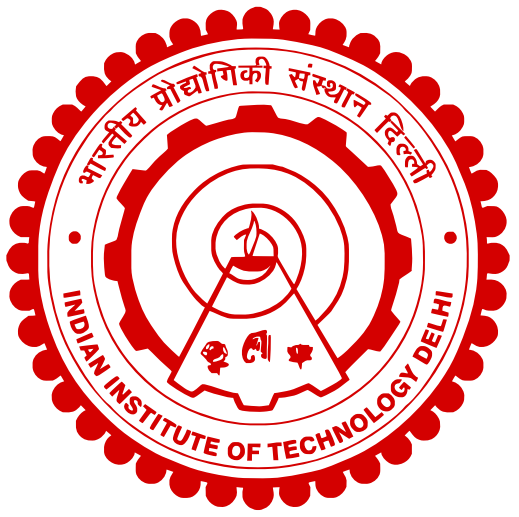
At $t = 0.5$

Rotation of square by 180°



$KF2(t=1)$

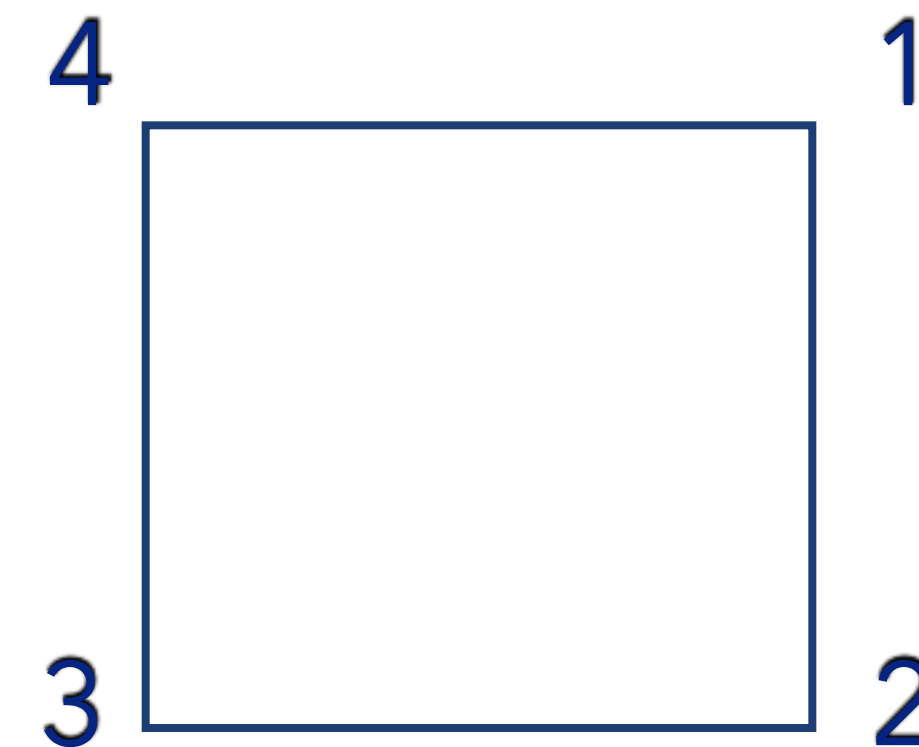
Problem with Interpolation



KF1($t=0$)

At $t = 0.5$

Incorrect Results (rotation of square by 180°)



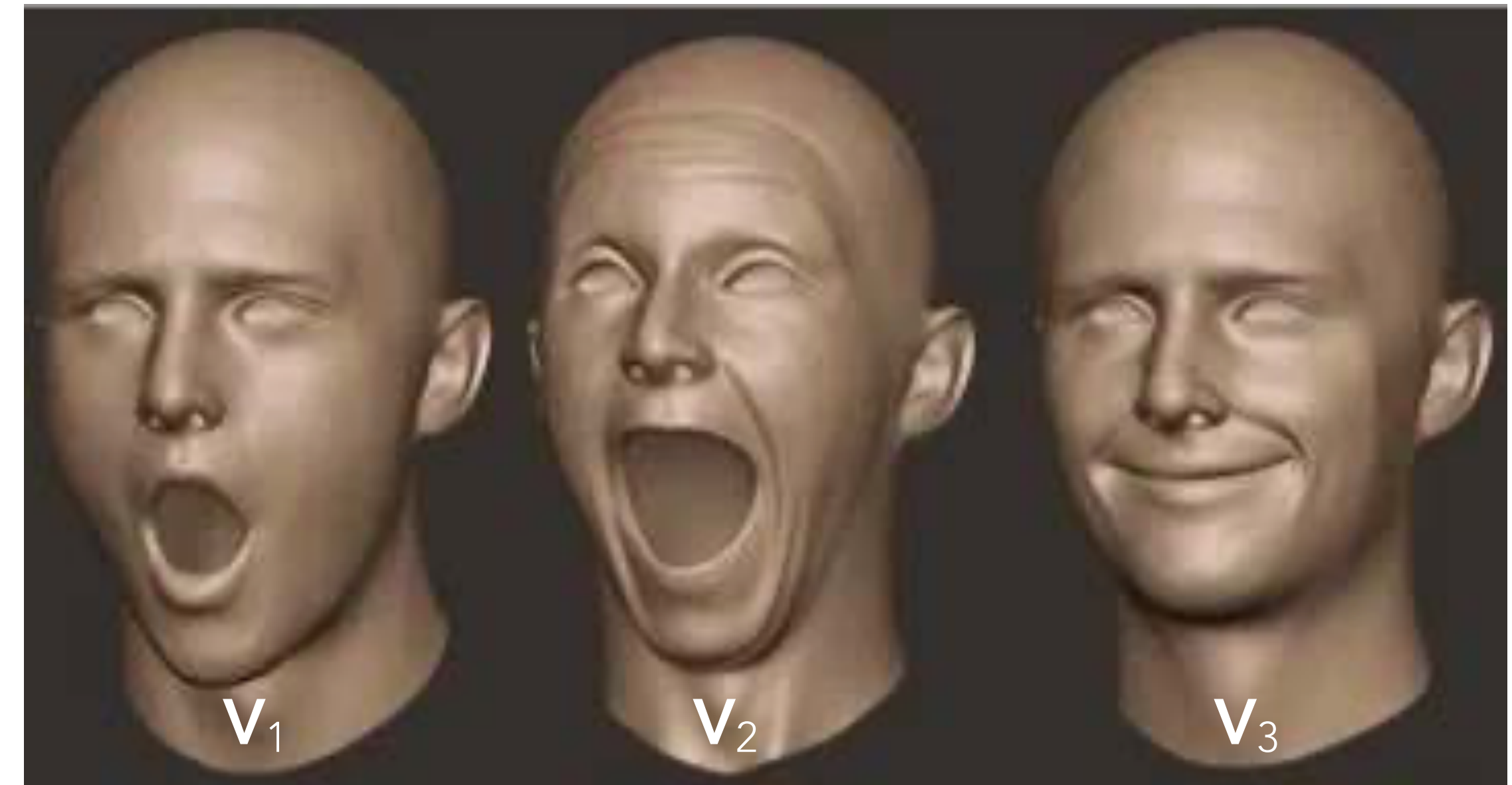
KF2($t=1$)

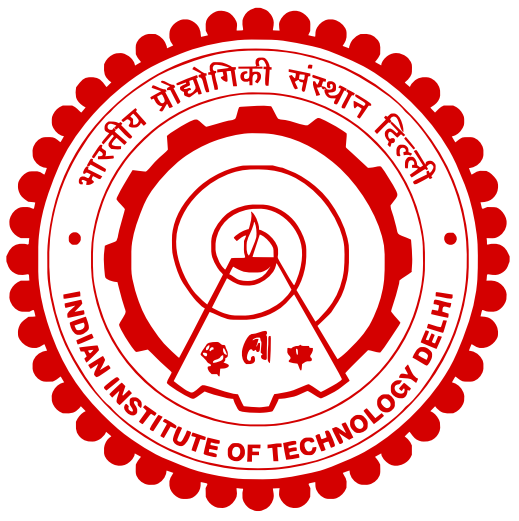
Facial Interpolation



- Deformations using bones and muscles is hard
- Generate meshes with differing expressions parameters (can be high-level: smile, frown, surprise, ... or low-level: eyebrow arch, mouth width, ...)
- For each parameter i , specify vertex positions in a **blend shape**: $\mathbf{V}_i$
- This generates a **shape space** via linear combinations:

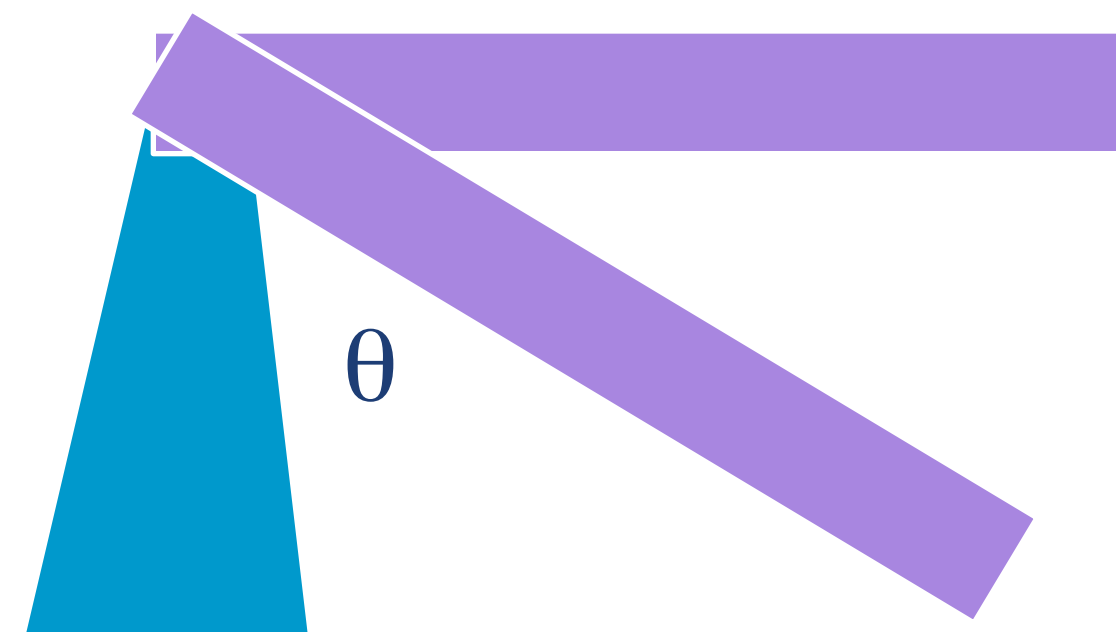
$$\triangleright \mathbf{V}(\mathbf{w}) = \mathbf{V}_0 + \sum w_i (\mathbf{V}_i - \mathbf{V}_0)$$





Parametric Animation

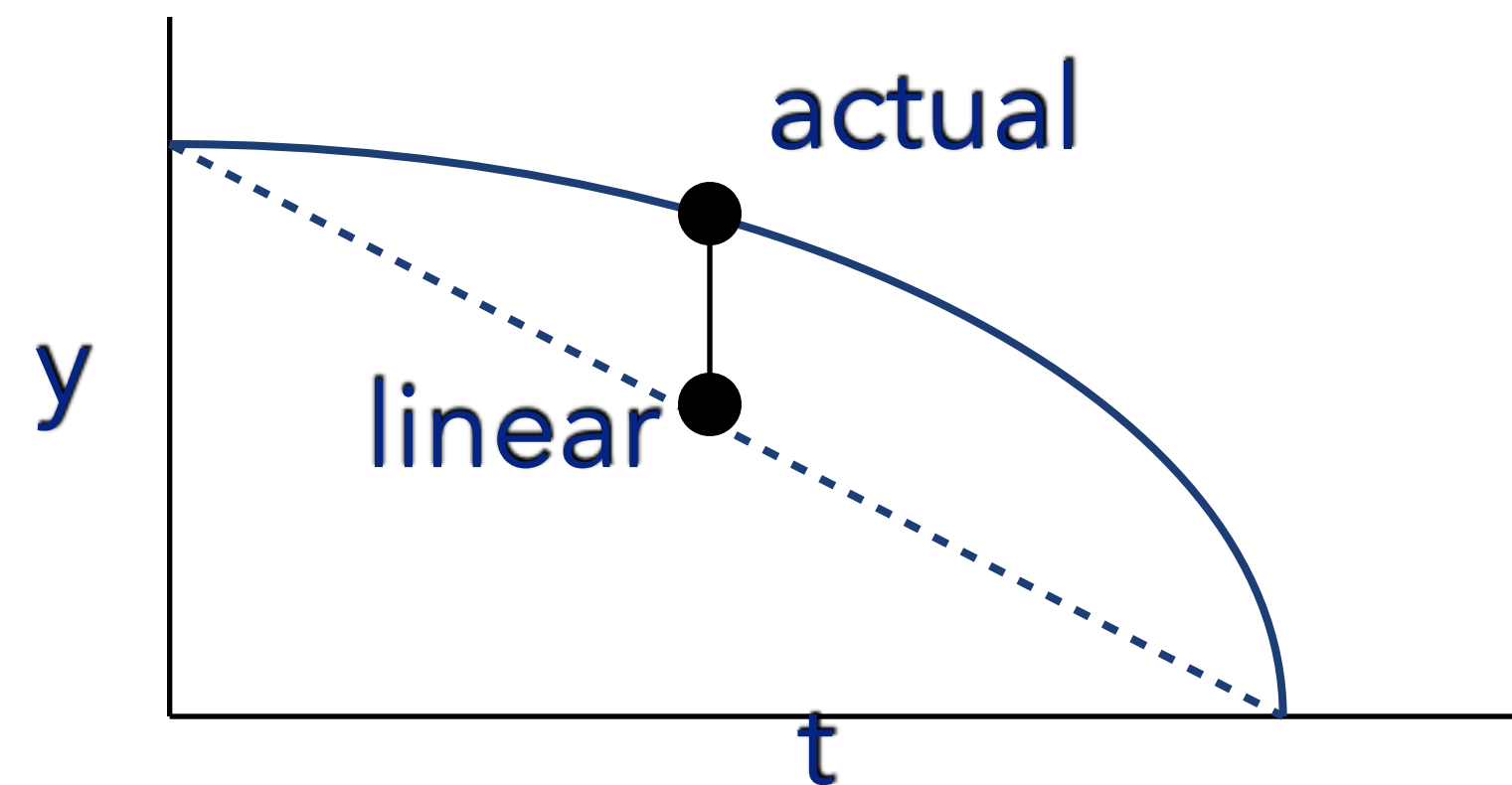
- Characteristic parameters for motion are specified and interpolated.
- Less data is required
 - ➔ e.g for motion of an arm, the parameter could be rotation angle.



Procedural Animation



Laws of motion: physical or procedural animation



Simulation

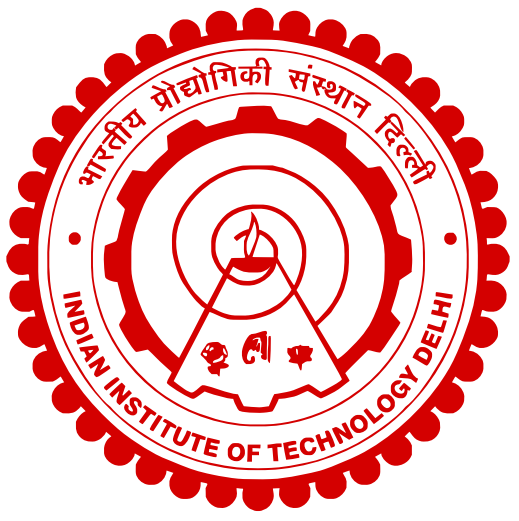
Motion Capture



CAESAR



<https://www.youtube.com/watch?v=4NU9ikjqjC0>

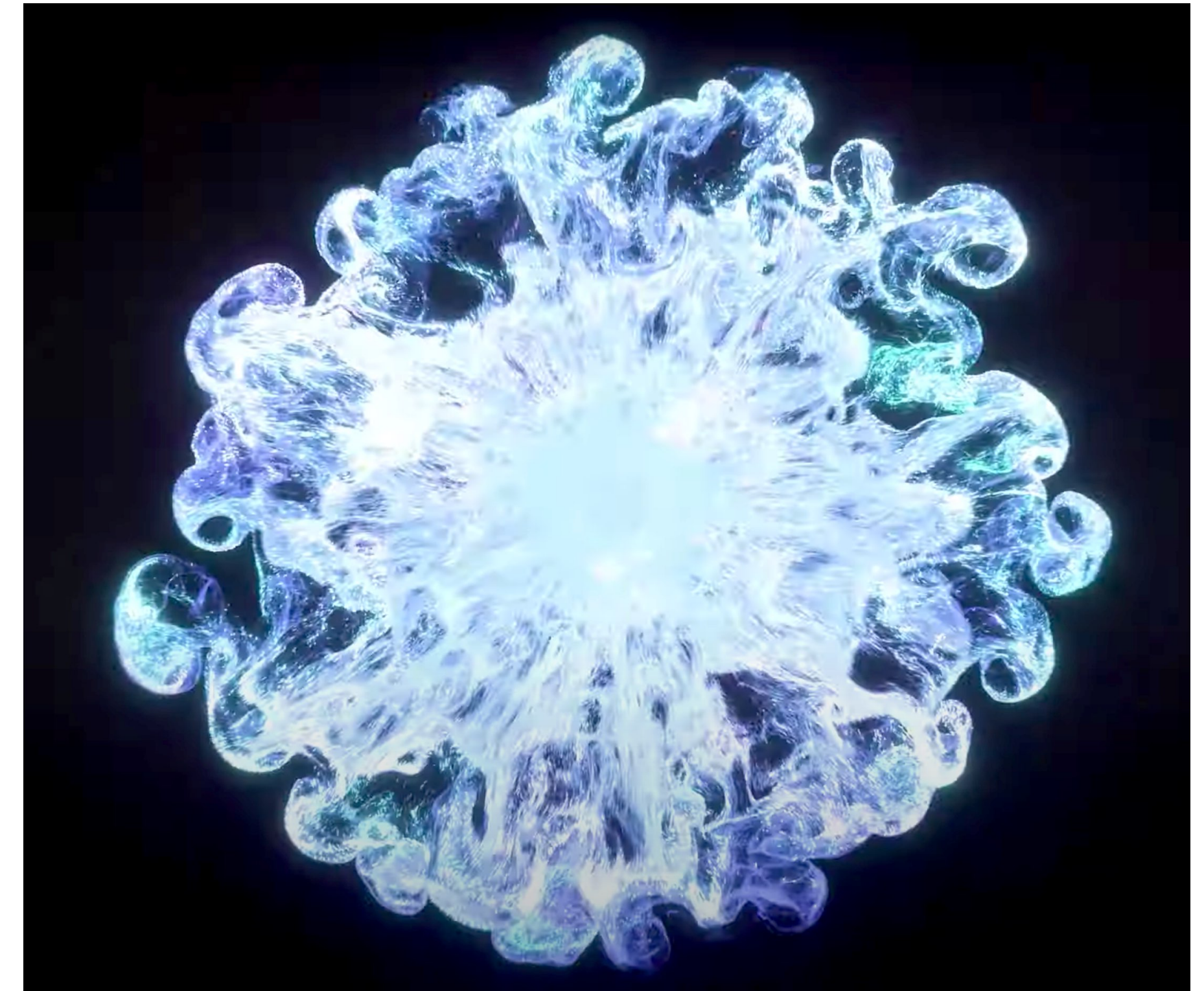


Motion Capture

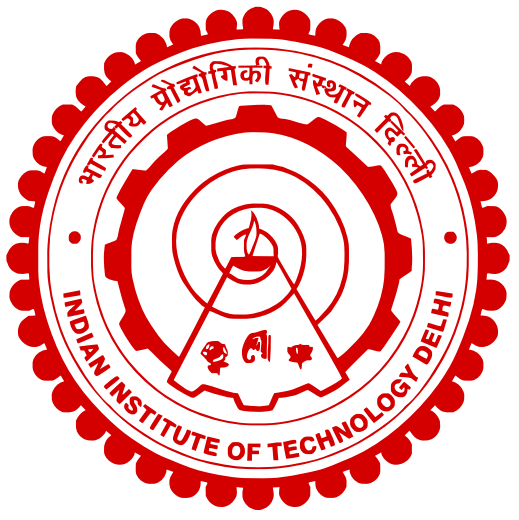
- Common techniques:
 - ➡ Optical – extract motion from video (usually many strategically placed markers)
 - ➡ Magnetic/Radio – attach magnets, transponders and use sensors to get location
- Relatively simple scenes
 - ➡ Blue/Green matte screen
- Robustness problems
 - ➡ Break scenes into small sequences and re-construct later
- Capture many sequences
 - ➡ Transition from one to the next

Procedural Animation

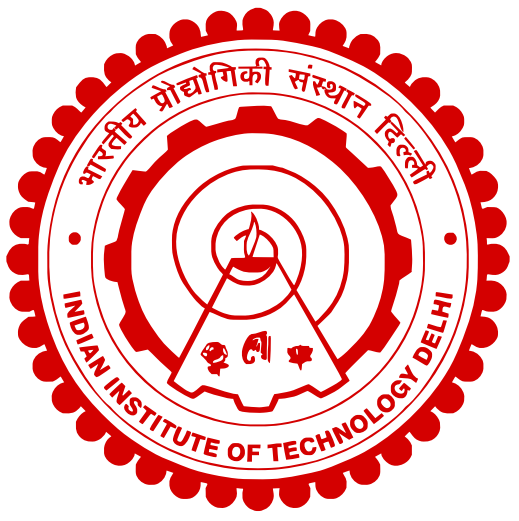
- Motion is defined algorithmically
- e.g. **particle systems**: update particles' positions based on some chosen rules, e.g.
 - ➔ directly: $\mathbf{x} = \mathbf{x}(t)$
 - ➔ using velocity field: $\dot{\mathbf{x}} = \mathbf{v}(\mathbf{x}, t)$
 - ➔ using forces: $\ddot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{v}, t)$



Procedural Animation



- Choose initial conditions
- Rules for behavior
 - ➔ Generally physics based
 - ➔ Also there are rules for transitions
- Advantage: Generality
- Disadvantage: Director's control



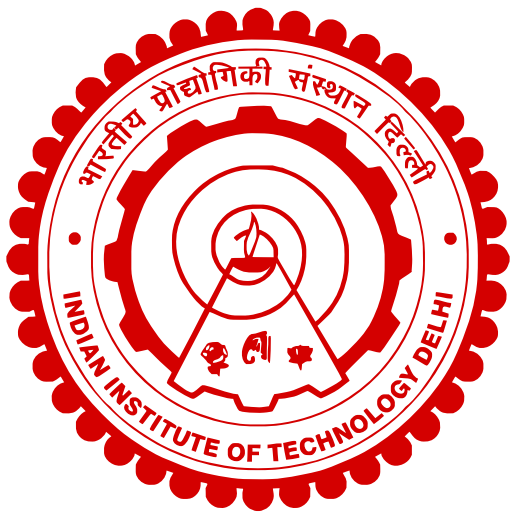
Particle Systems

- An object is represented as cloud of particles
- Particles are not static; particle system evolves
 - ➔ Non deterministic
- Particles are simple (computationally efficient) but can model complex amorphous objects and behaviors
 - ➔ Dust, Water fall, Rain, Fire, Explosion, Cloud, Stars, Grass, Fur, etc.



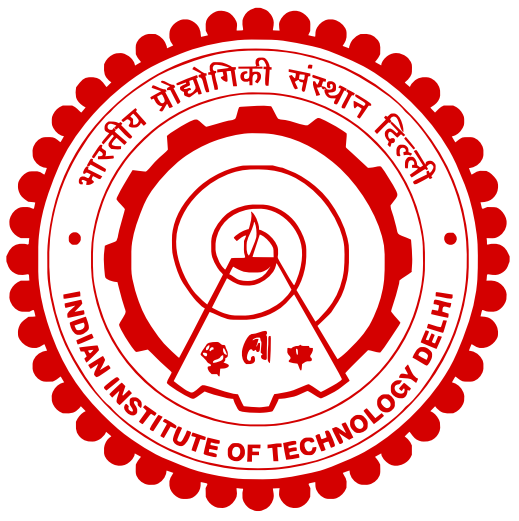
Star Trek II: The Wrath of Khan

<https://www.youtube.com/watch?v=QXbWCrzWJo4>



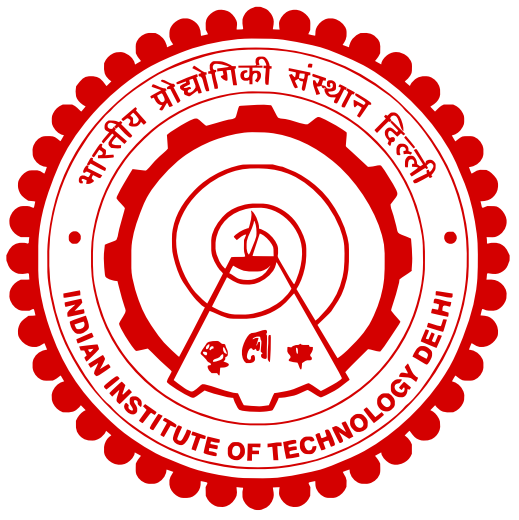
Particle Systems

- Particles exert forces and (have force exerted on them)
- Rules for particle generation, behavior and termination
 - ➡ Generate new particles with initial attributes and a lifespan
 - ➡ Attributes: color, position, shape, texture .. evolve
- Particle shaders provide color
 - ➡ Particles may also be sources of light



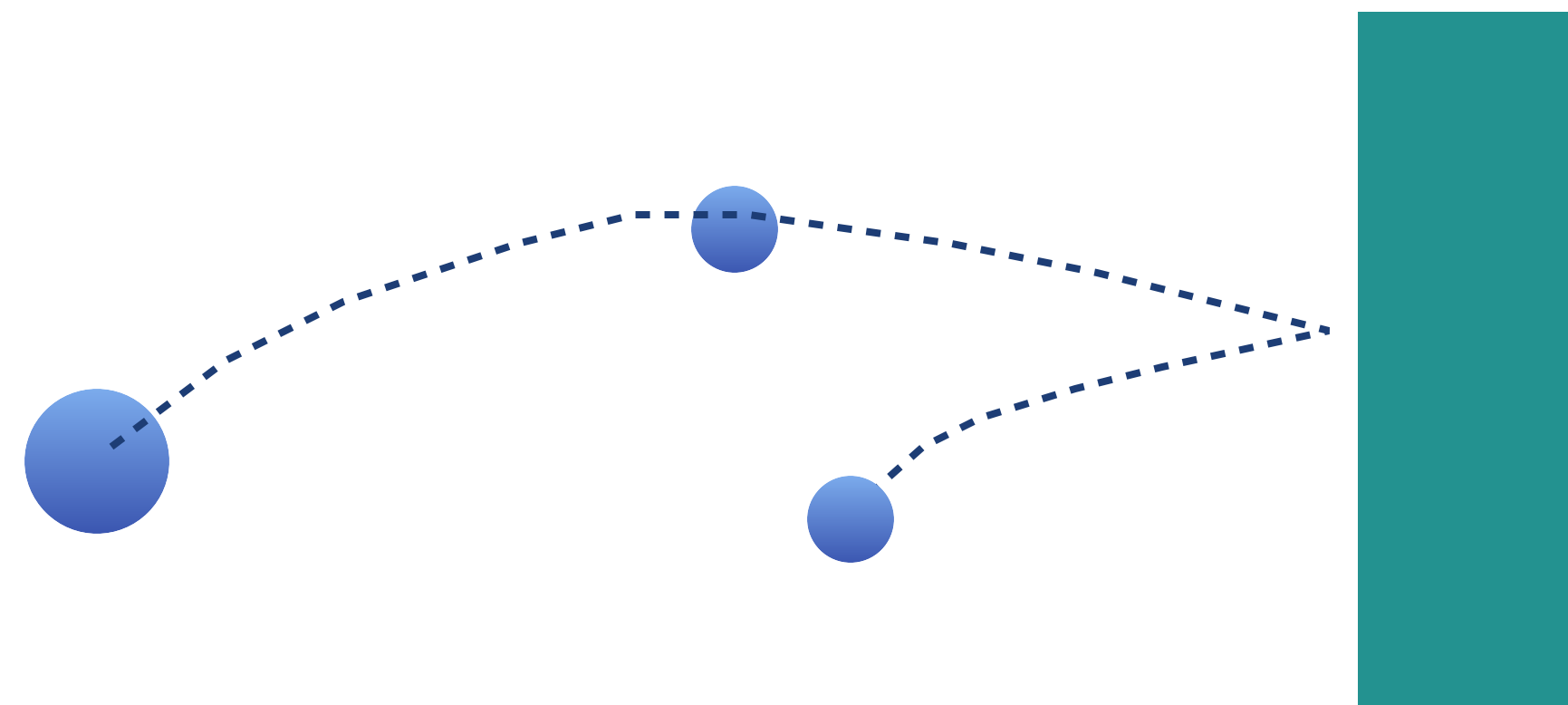
Particle Generation

- Stochastic (e.g., $N = \text{average} + \text{rand}() * \text{var}$)
- Particle attributes
 - ➔ Determine motion status, appearance, and its life in the particle system
 - ➔ (position, color, opacity, size, speed, life-span etc.)
 - ➔ Initialized at the time of creation



Particle Termination

- For each new frame, particle's life time is decremented by one
- When life time = zero, the particle is removed





Particle Animation

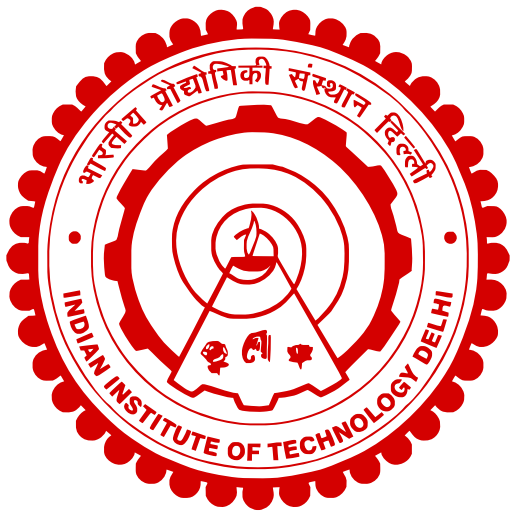
- Dynamics

$$v_i^{new} = v_i + \frac{F(x_i, v_i, t)}{m_i} \Delta t$$

$$x_i^{new} = x_i + v_i^{new} \Delta t$$

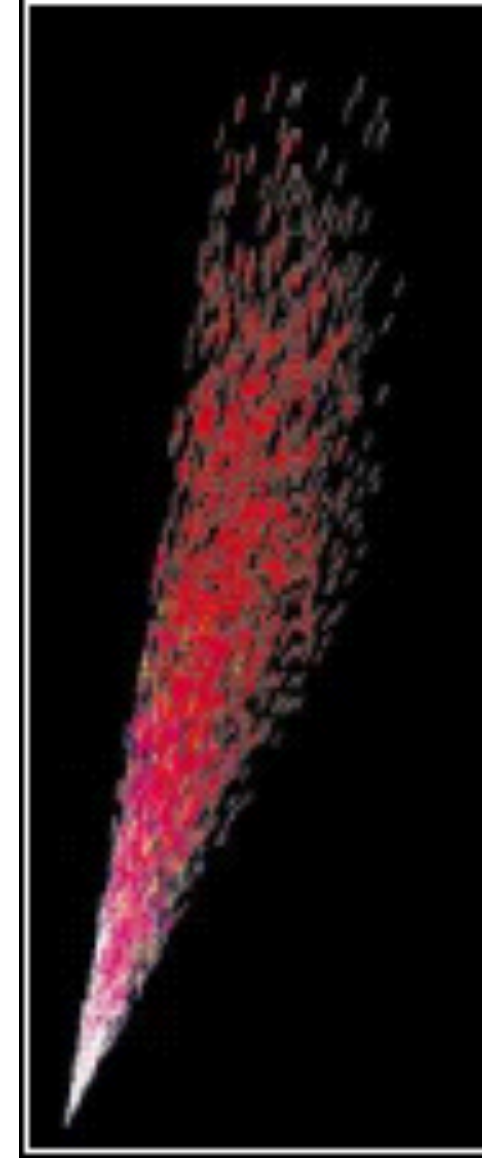
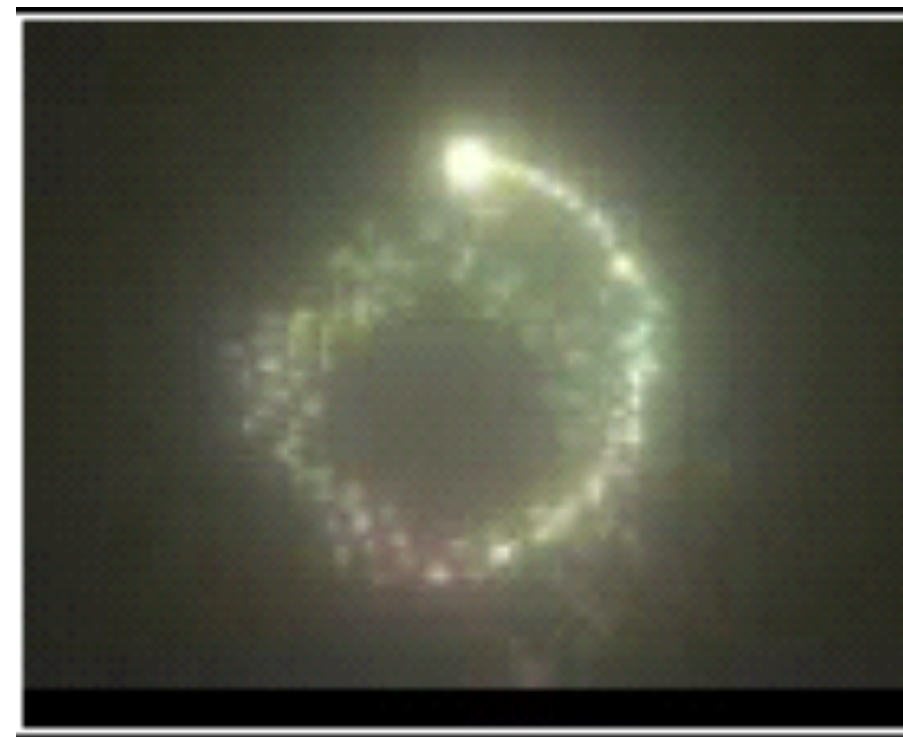
- From force find acceleration, velocity, position
- Other attributes (color, opacity, etc.) may also change with time

Particle Rendering



- Particles can be rendered as light sources
- Particles do not intersect with objects
- May ignore shadows
- These assumptions simplify the rendering and computation

Particle Systems Example



Particle Systems Example

Wrath of Khan

- Particle systems generated in concentric rings
- Number of systems based on ring circumference
- New fire particles based on distance from impact crater

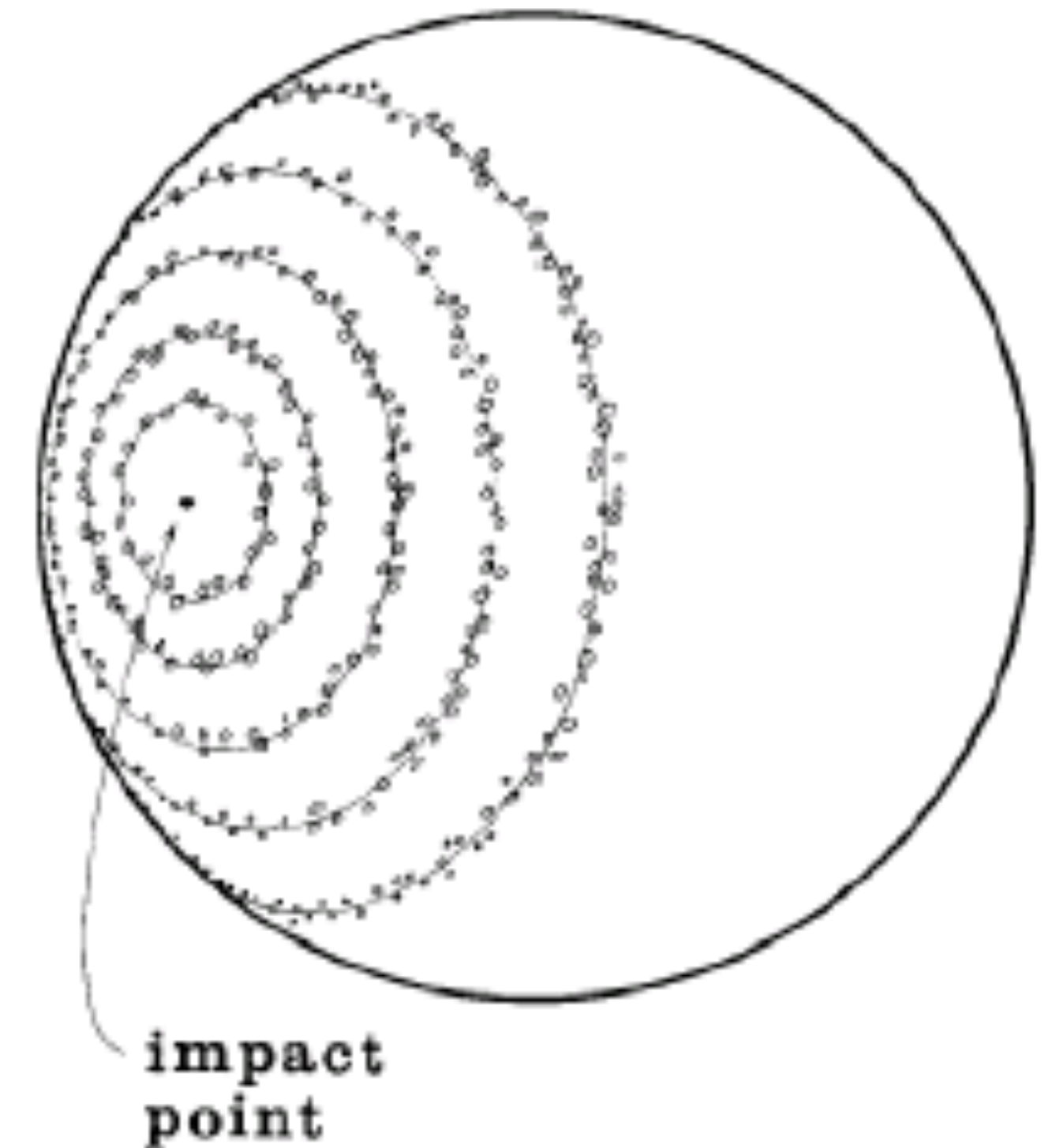


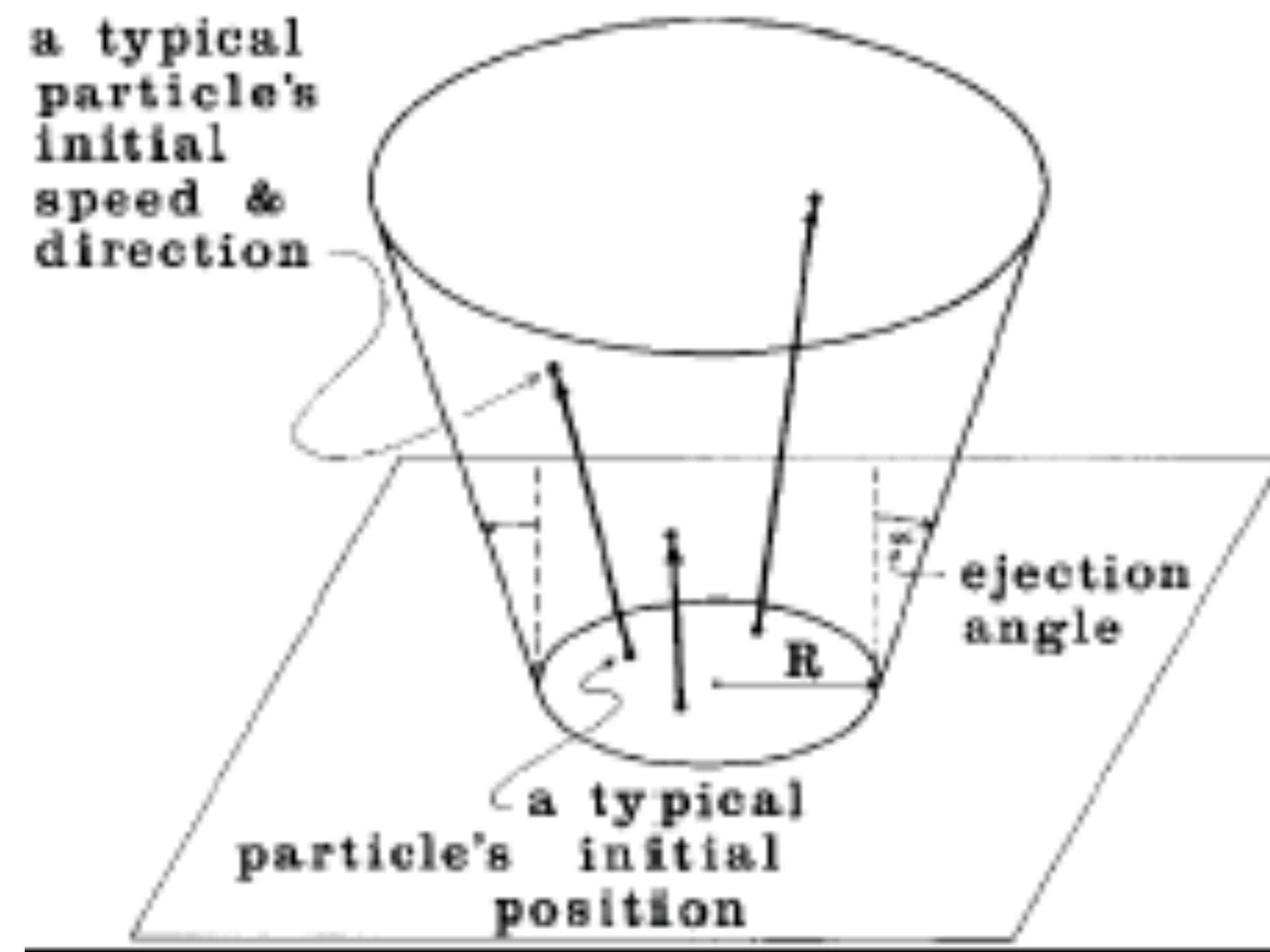
Fig. 2. Distribution of particle systems on the planet's surface.



Particle Systems Example

Wrath of Khan

- Individual particle systems look like explosions



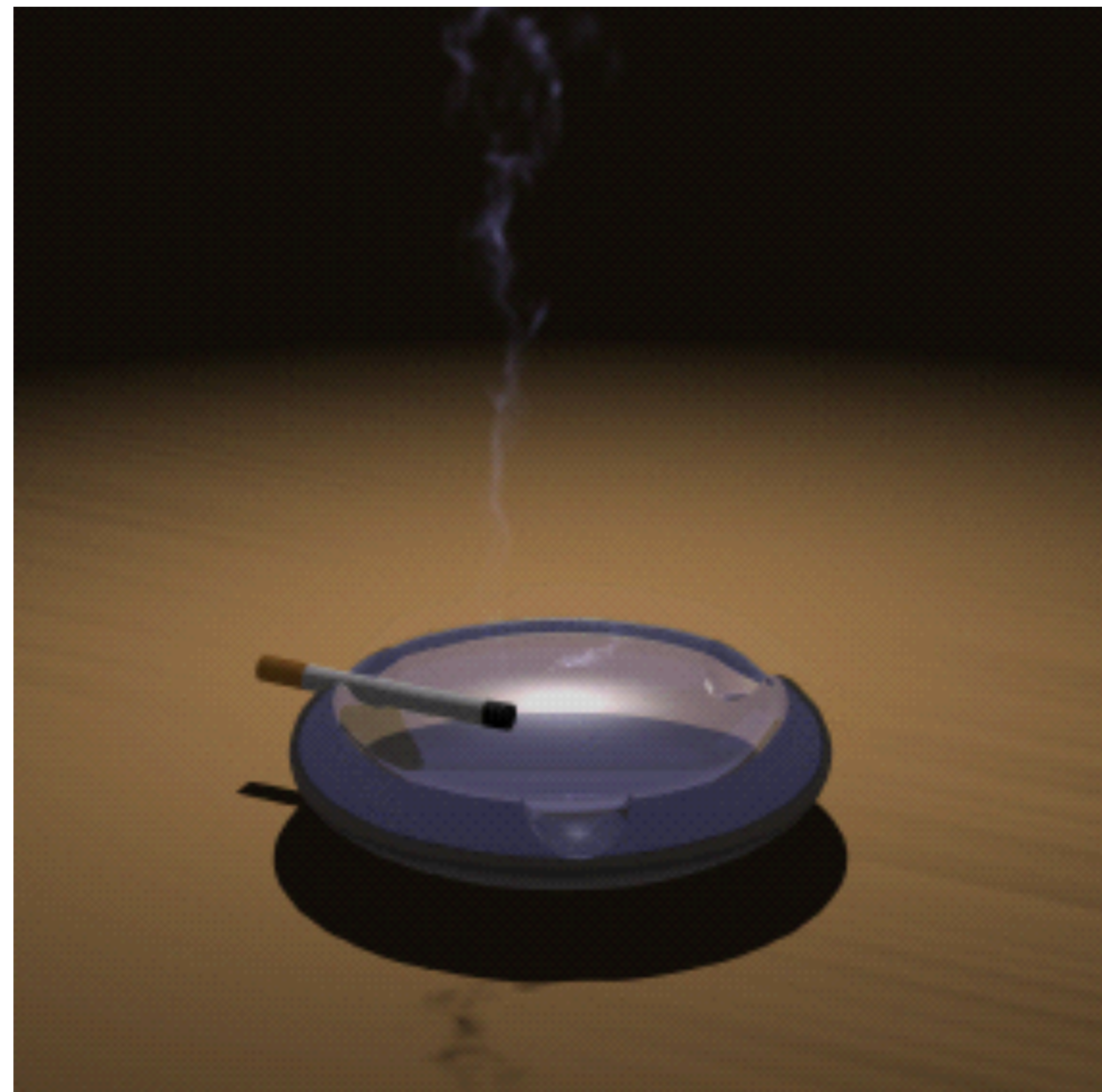
Particle Systems Example

Wrath of Khan

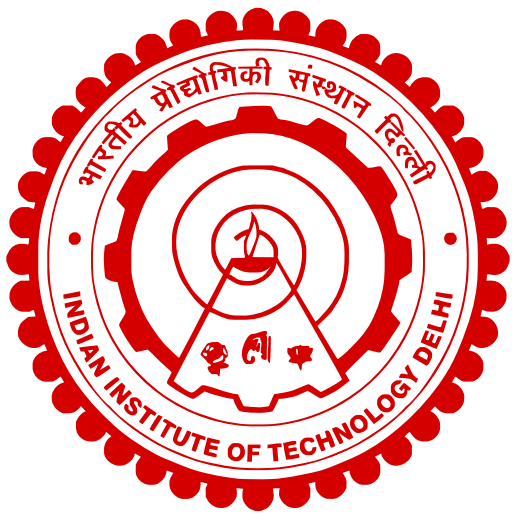
- Individual particle systems look like explosions



More Particle Systems Example

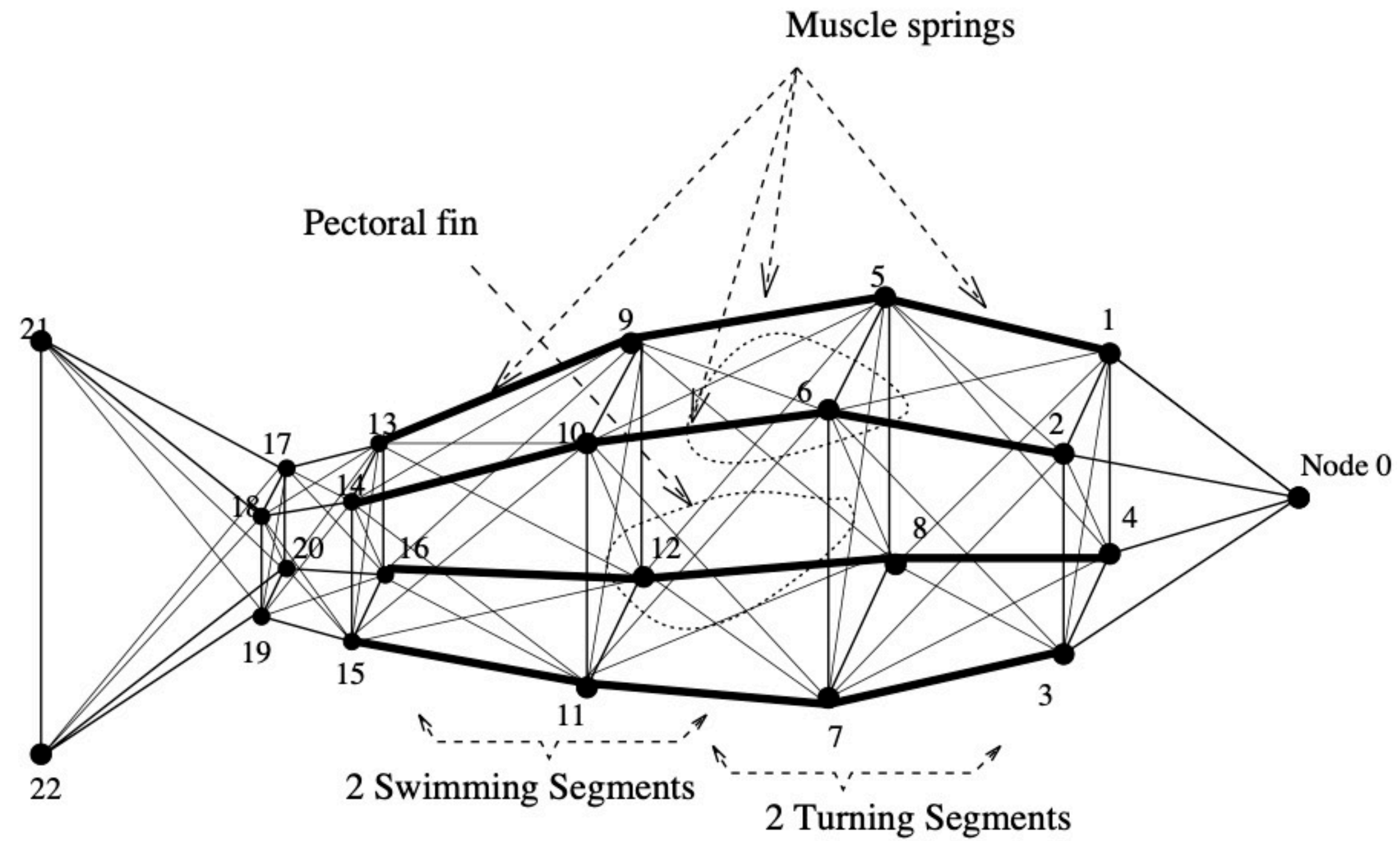


Mass Spring Systems



- Muscle, Cloth, etc.
- Model behaviour by a system of springs and masses
- Springs have a resting length and exert forces when perturbed
- Minimize sum of forces on all vertices or energy of the spring system
- Advantage
 - ➔ Simple Idea, works surprisingly well
- Disadvantage
 - ➔ Often requires "tweaking"

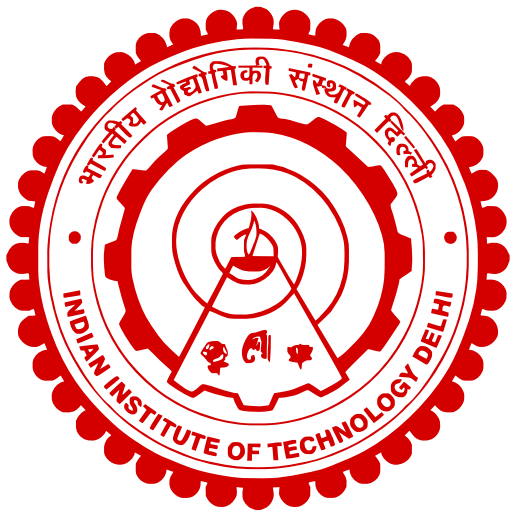
Mass Spring Fish



[Xiaoyuan Tu]



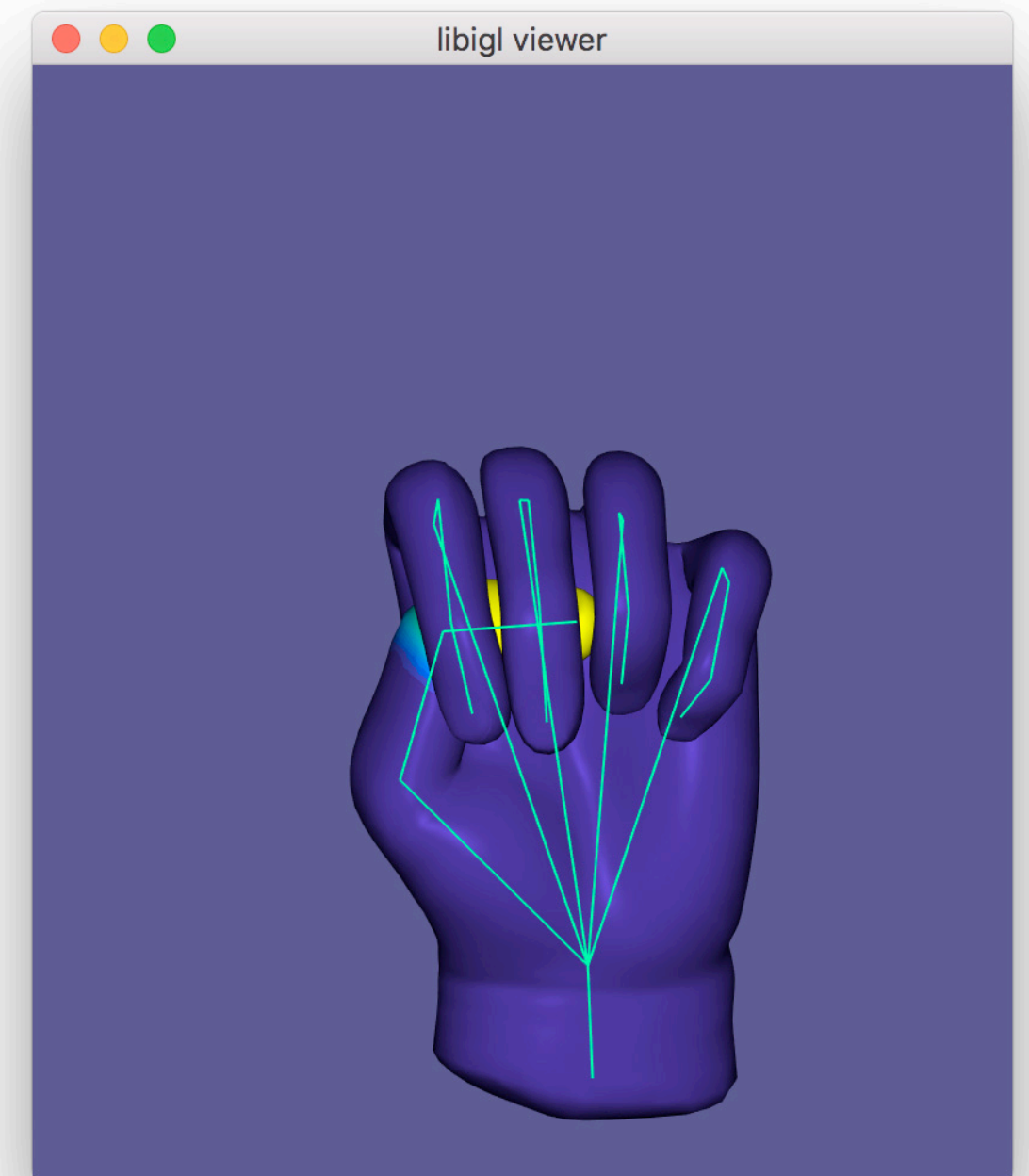
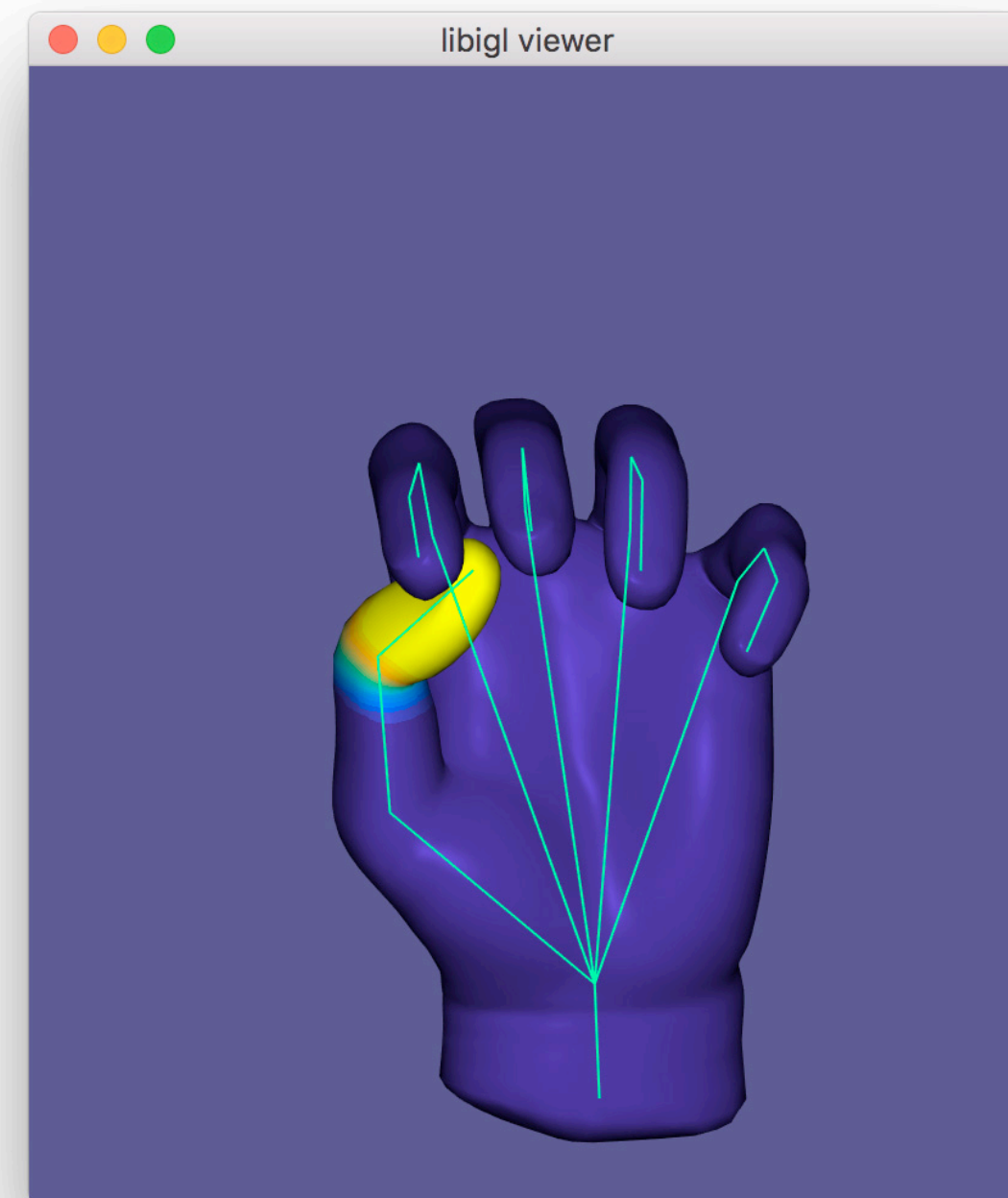
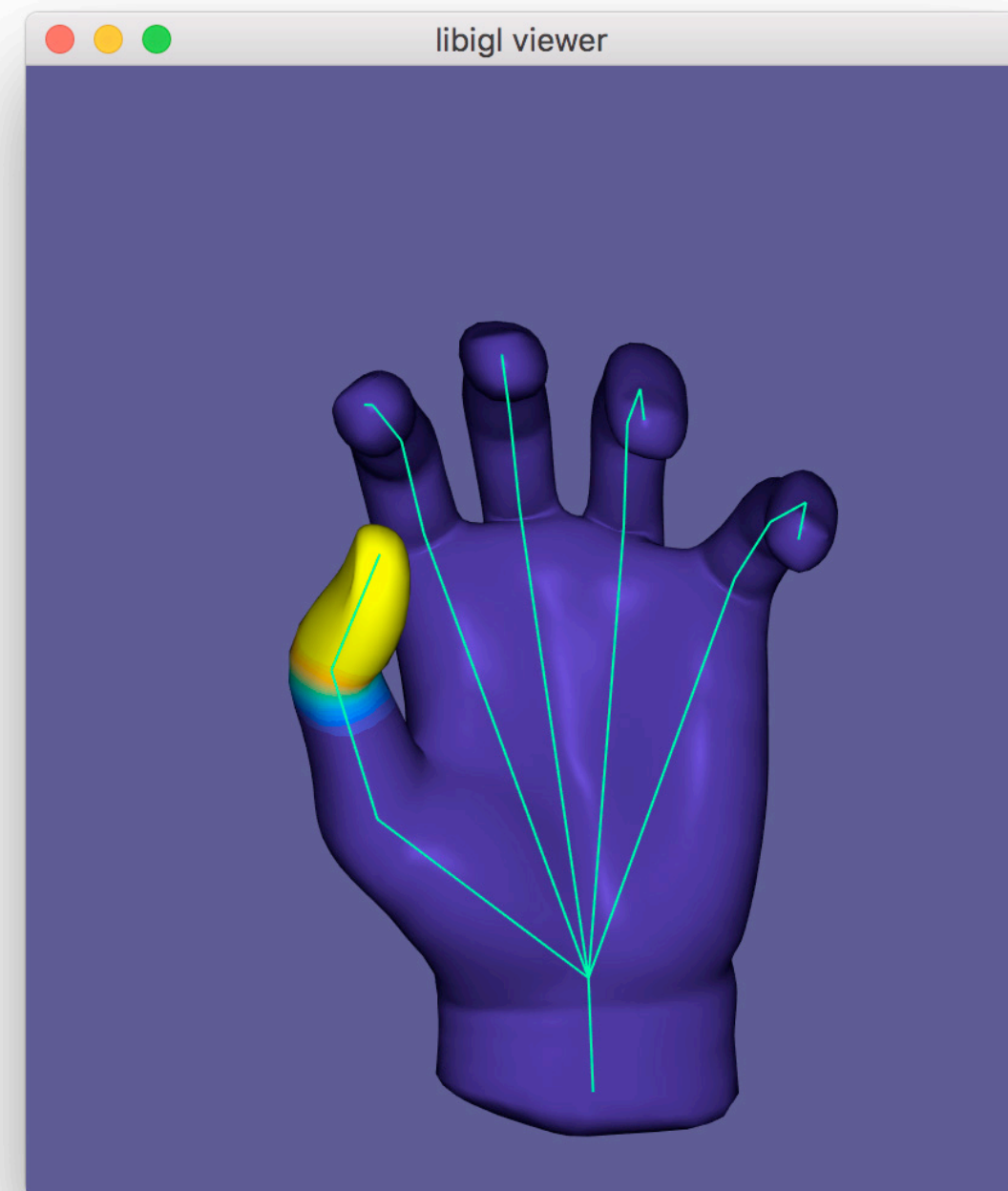
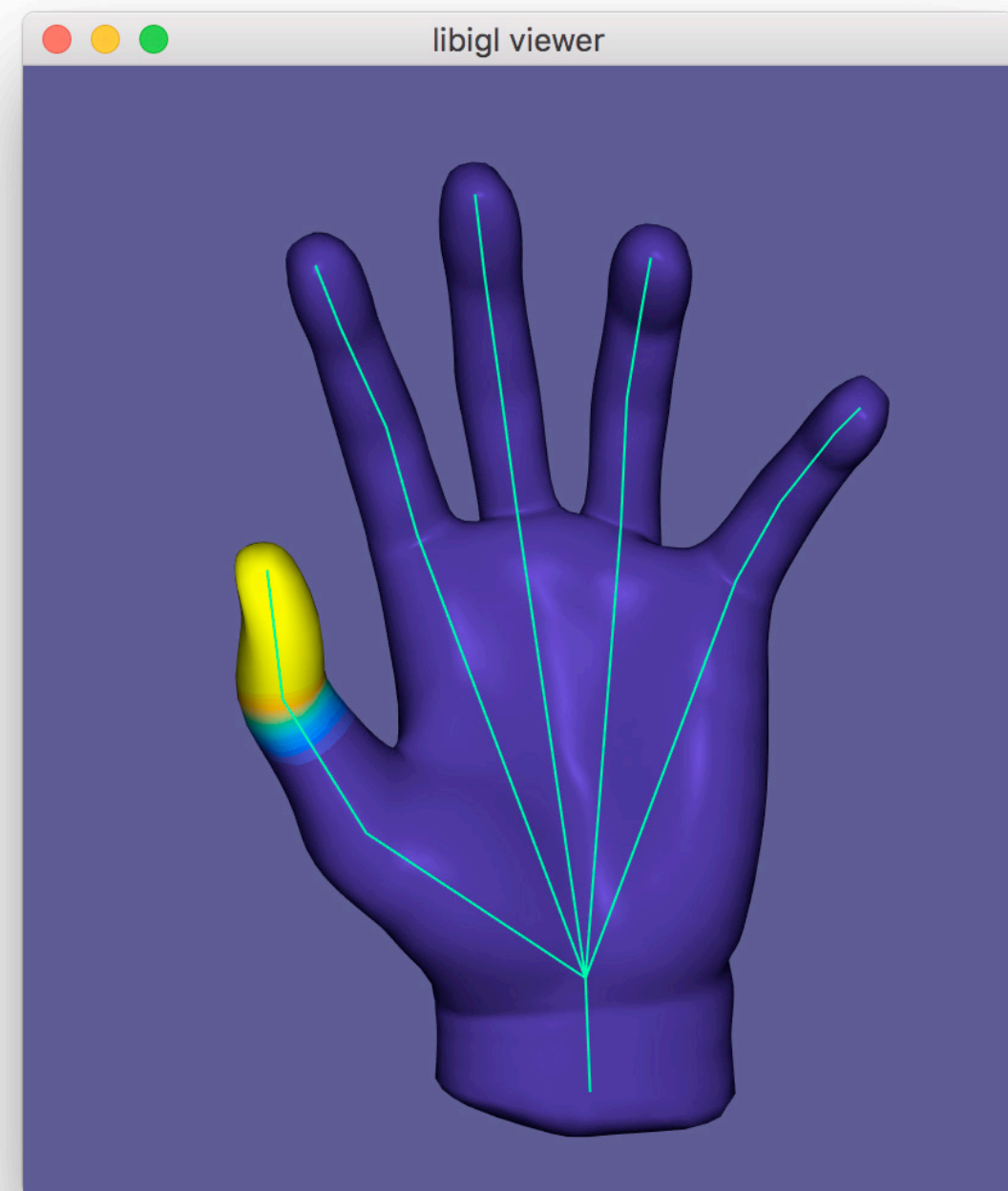
Physically-Based



- Model Physics equations
 - ➔ Find parameters and solve
- Advantages:
 - ➔ Can replicate reality
- Disadvantages:
 - ➔ Expensive computation (on the fly)
 - ➔ Hard to control (artistically)

Mesh Skinning

- Define a **skeleton** and animate its pose over time (for articulated characters)



- The skin mesh deforms following the movement of the skeleton: **skinning deformation** (based on a correspondence with the skeleton)



Hierarchical Transformations

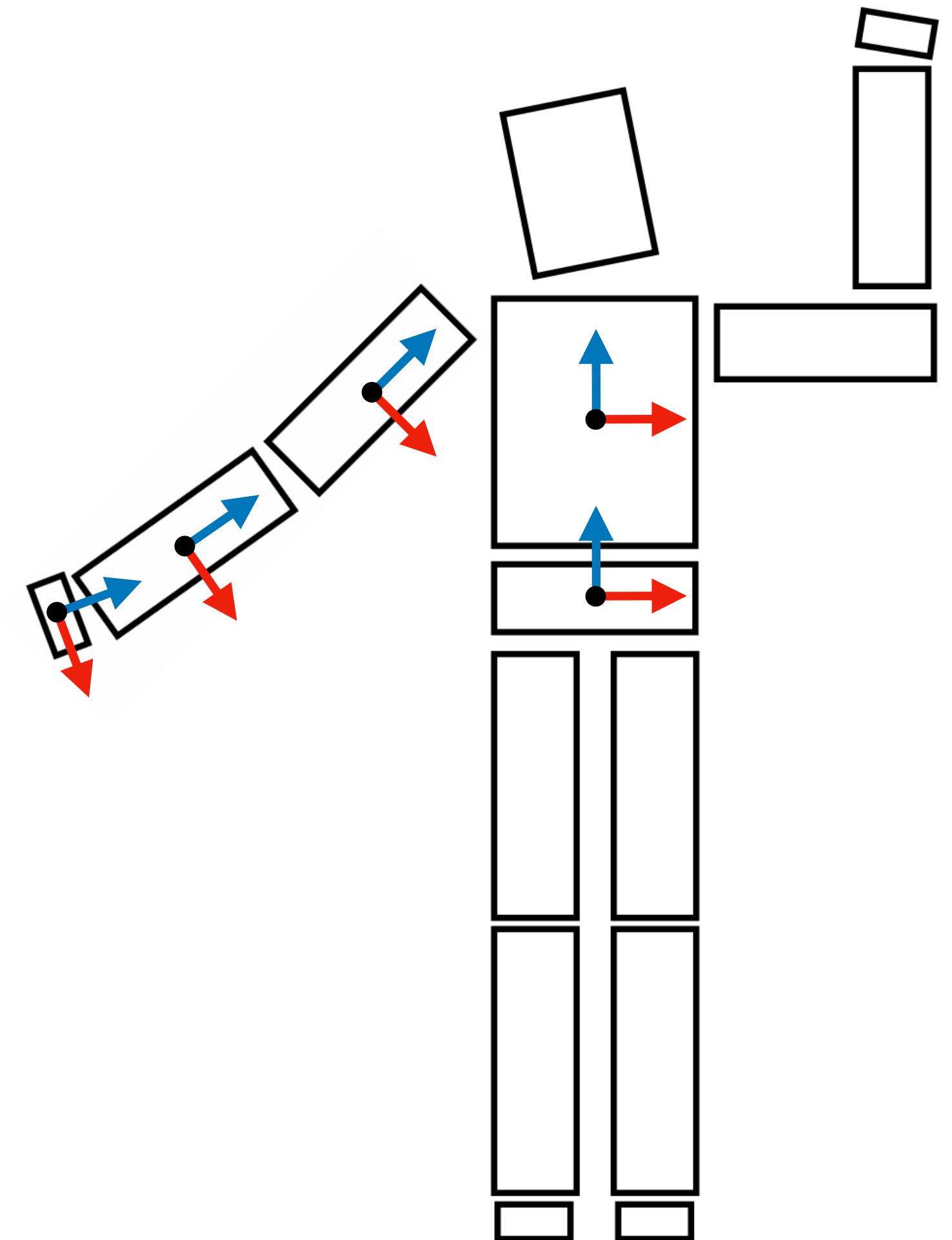
- Recall hierarchical transformations

➔ Each bone is transformed **relative** to its parent

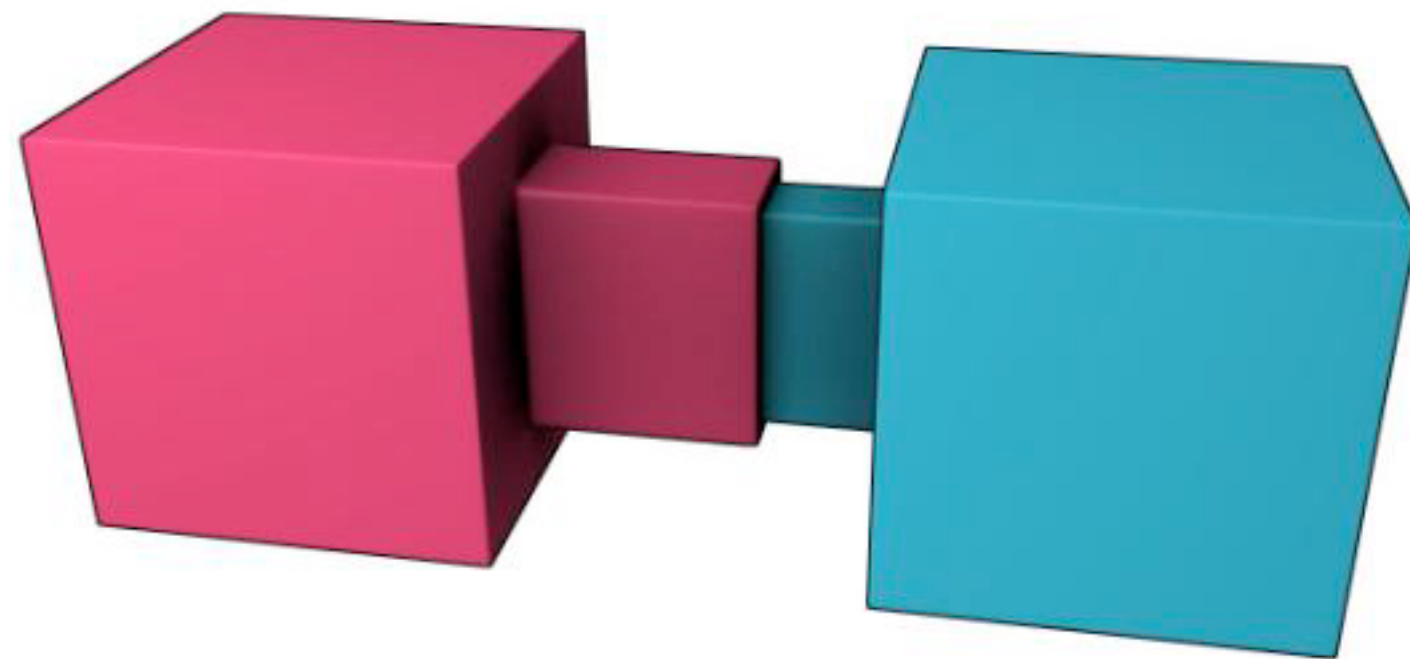
$$M_{lHand}^{world} = M_{hip}^{world} M_{chest}^{hip} M_{ulArm}^{chest} M_{llArm}^{ulArm} M_{lHand}^{llArm}$$

$$M_{lHand}^{world} = M_{llArm}^{world} M_{lHand}^{llArm}$$

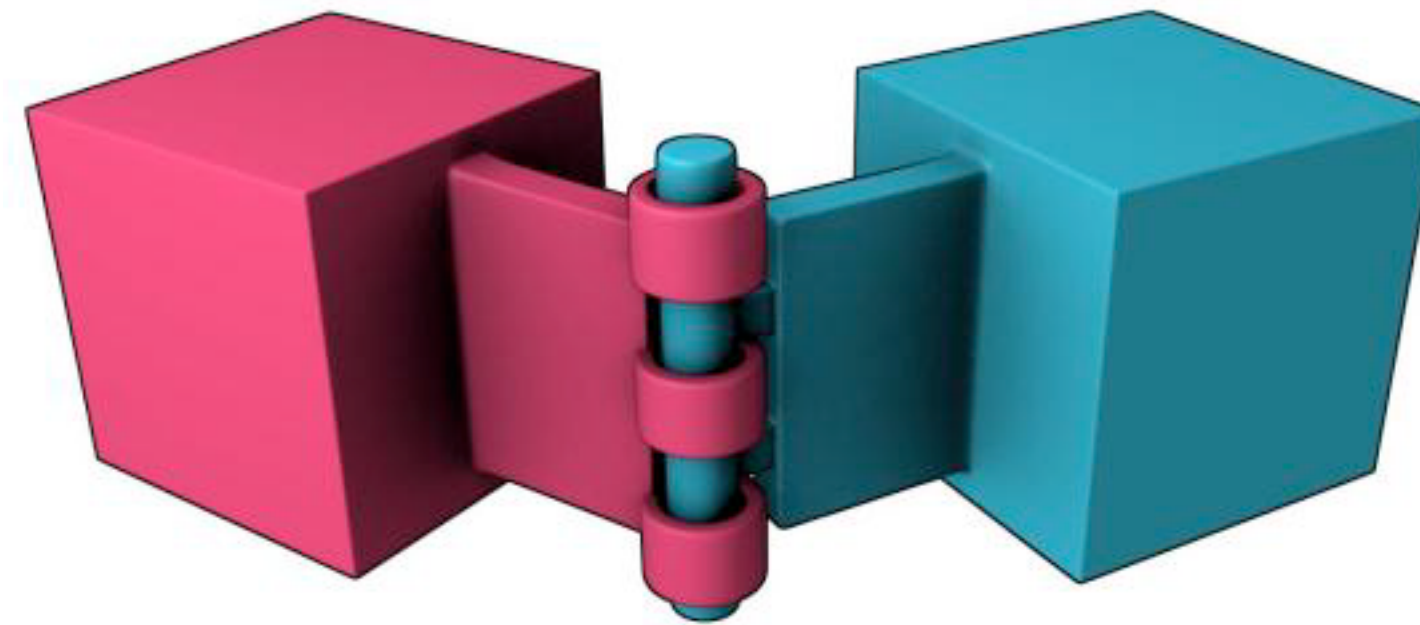
How to control transformation of child bone?



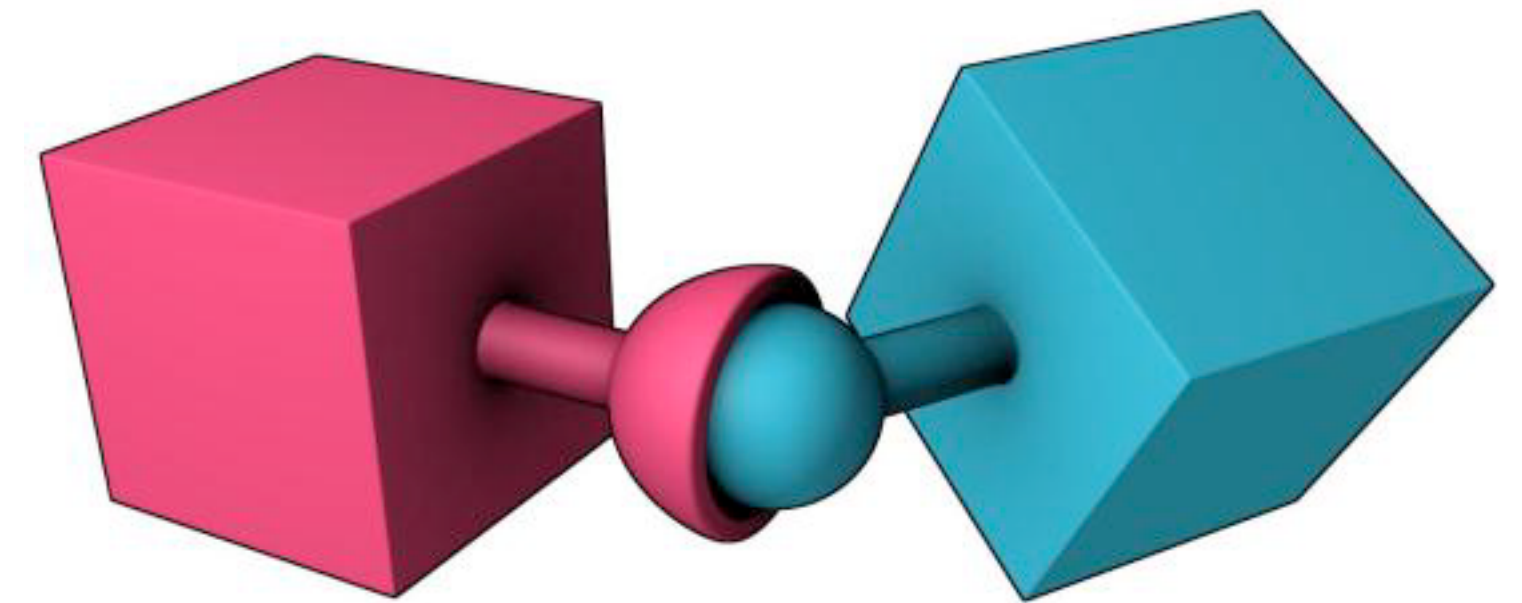
$\mathbf{M}_{\text{child}}^{\text{parent}}$ cannot be arbitrary! Child's relative motion is constrained by a **joint**, e.g.:



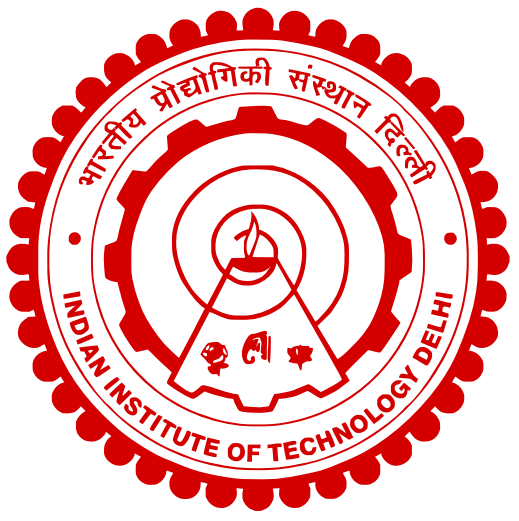
Slider
(1 DOF)



Hinge
(1 DOF)

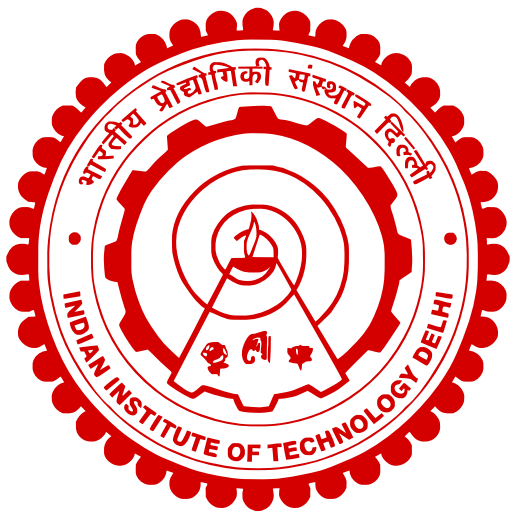


Ball-and-socket
(3 DOFs)



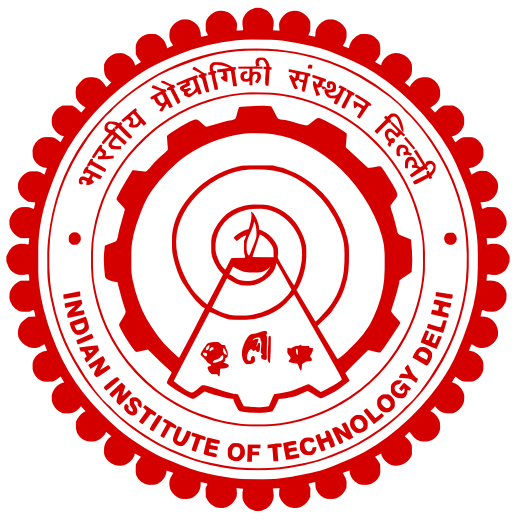
Skeleton Based Animation

- Rig:
 - ➔ Given a mesh model, create a skeleton
 - ➔ Add skin: find transformation matrices for geometry
- Animate skeleton:
 - ➔ For example, by controlling joint angles
- Skin:
 - ➔ Re-layer the skin on the moved skeleton
- Techniques:
 - ➔ Linear blend skinning (a.k.a. SSD)
 - ➔ Scattered-data interpolation
 - Given example poses, automatically fit linear skin blend model



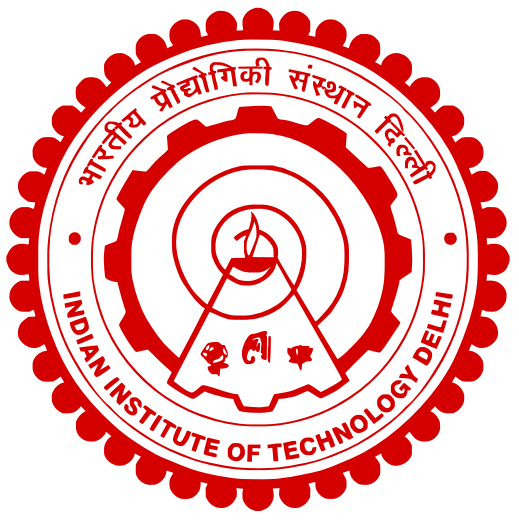
Links and Joints

- Each link has an attached coordinate system
 - ➔ origin, orthogonal directions for axes
- Independent of actual link mesh
- A joint joins two links
 - ➔ Specify in both coordinate systems, maintain transformation between links
- Constrain degrees of freedom
 - ➔ Upto three translations and three rotations
 - ➔ Typically only rotation (for rigid link animation)
 - ➔ May be decomposed into single DOF joints for simplicity



Forward Kinematics

- Pick an arbitrary link to be the root
 - ➡ form a tree
- For each joint, transform from root's coordinates to link's
- Traverse tree composing transformations
 - ➡ Leaves are called "end-effectors"
- Animate by specifying individual joint transformations
 - ➡ How does foot rest to the floor at each step?



Inverse Kinematics

- Specify target for an end-effector (Position, orientation)
- Joint angles are unknown, solve for them
 - ➔ But multiple solutions usually exist
 - ➔ Add constraints
 - e.g., keep the arm in the specified plane
 - ➔ Minimize some objective function
 - e.g., total motion:
 - sum of squares of differences from previous frame



Mesh skinning

$$v' = \sum_i^n w_i M_i v \quad \text{with} \quad \sum_i^n w_i = 1$$

Where:

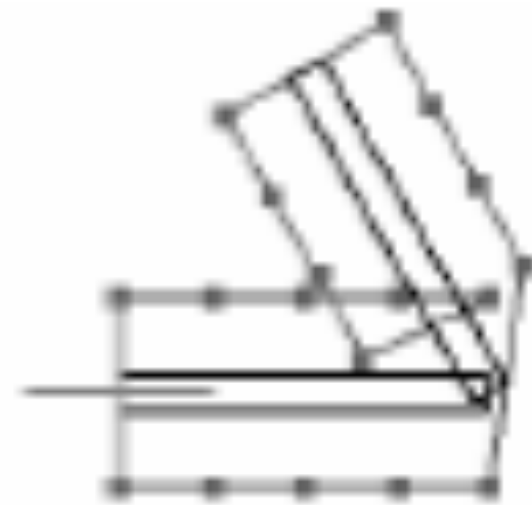
n is the number of matrices

v is the vertex position

w_i is the associated weight

M_i is the transformation matrix

Linear Blending



Bind to closest link

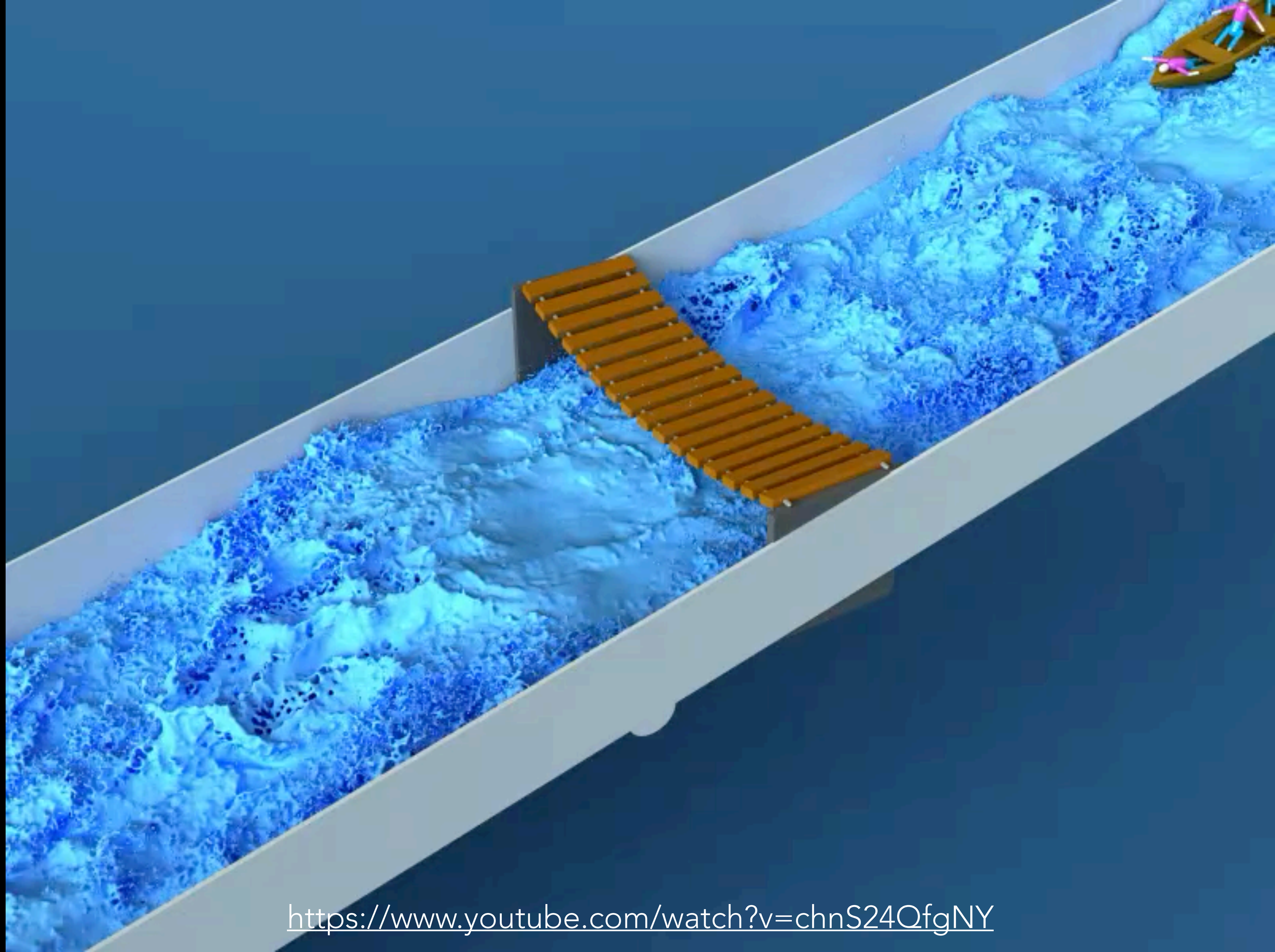
Linear blend with
two links

Linear blend
with three links

Use extra link



<https://www.youtube.com/watch?v=KDvFzFlruQ>



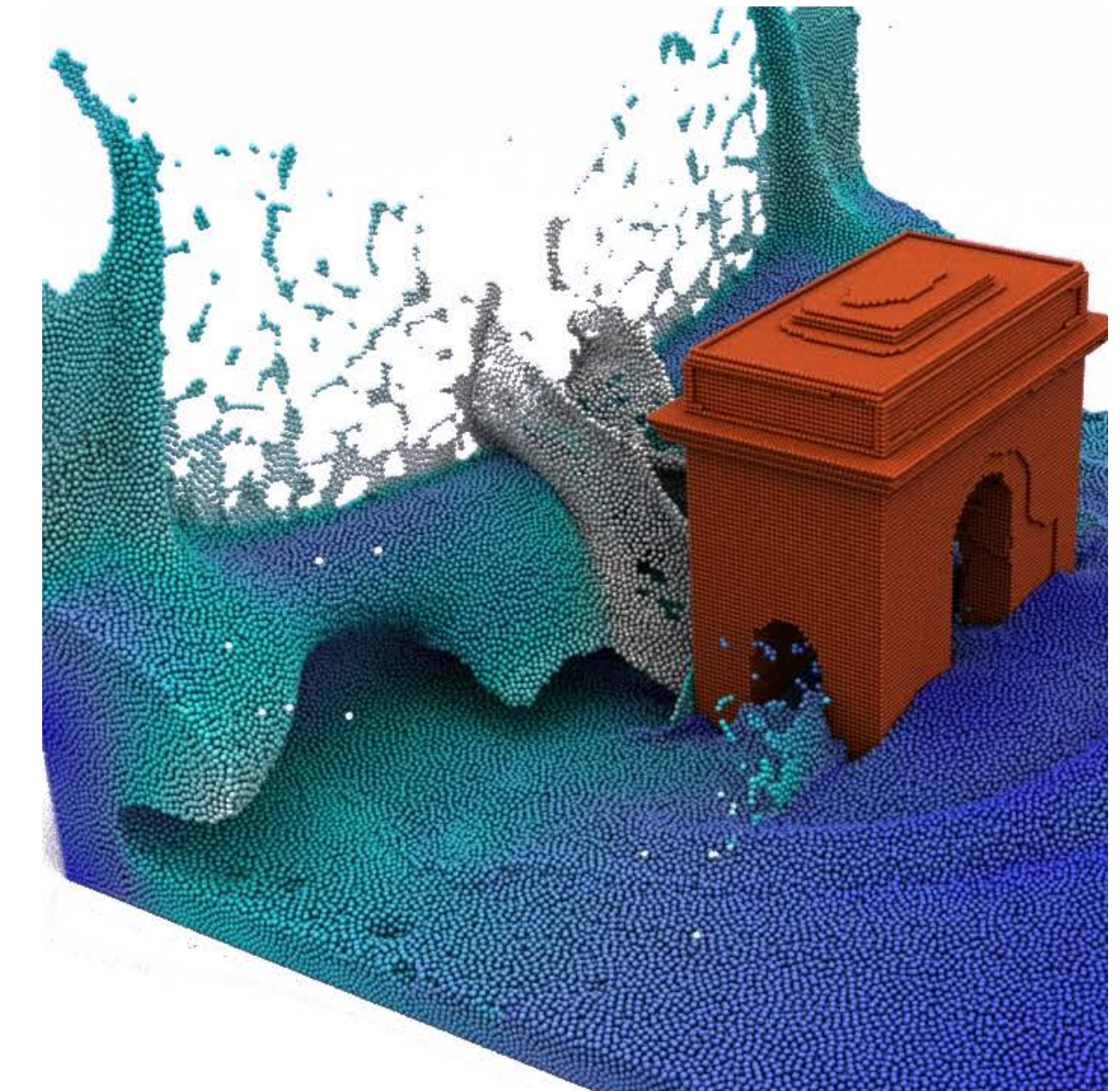
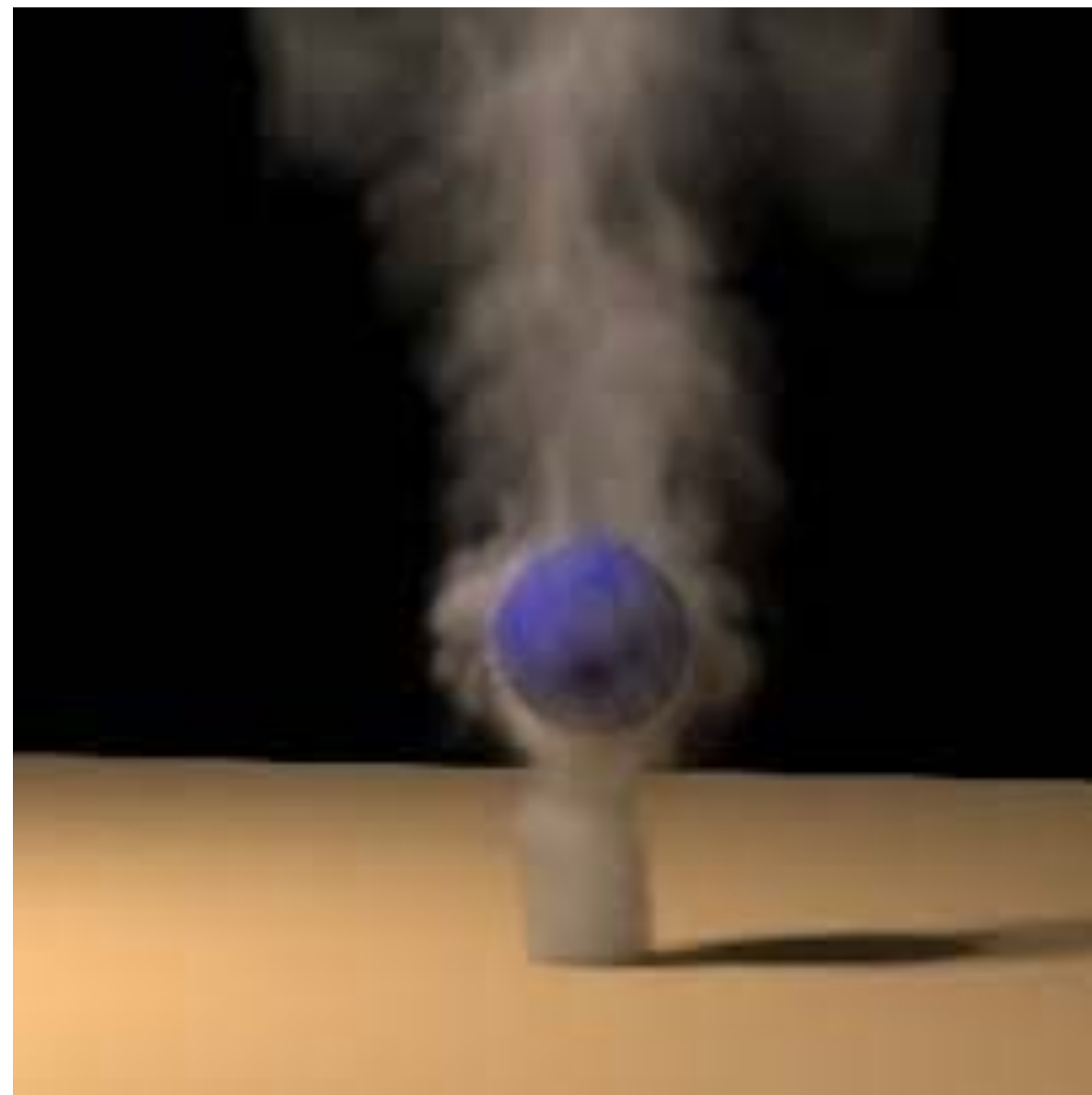
<https://www.youtube.com/watch?v=chnS24QfgNY>



Fluids

Smoke, water, fire, etc.

- Described by the Navier-Stokes equations
- Velocity field $\mathbf{v}^t(\mathbf{x})$, $\mathbf{x}$ represents position
- Forces modify $\mathbf{v}$ giving $\frac{\partial \mathbf{v}}{\partial t}$

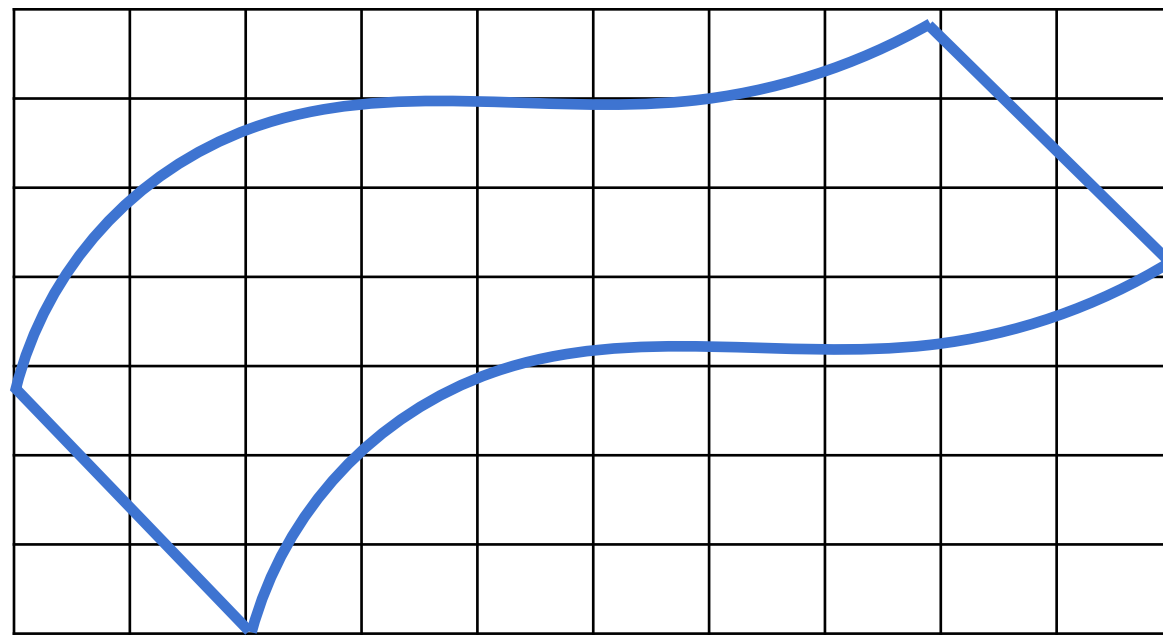




Fluid Simulation

● Navier Stokes Equations

➔ Eulerian Grid



$$\frac{\partial \mathbf{v}}{\partial t} = - \underset{\substack{\text{velocity} \\ \downarrow}}{\mathbf{v} \cdot \nabla} \mathbf{v} + \underset{\substack{\text{viscosity} \\ \downarrow}}{\mu \nabla^2} \mathbf{v} - \nabla p \underset{\substack{\text{pressure} \\ \downarrow}}{p}$$

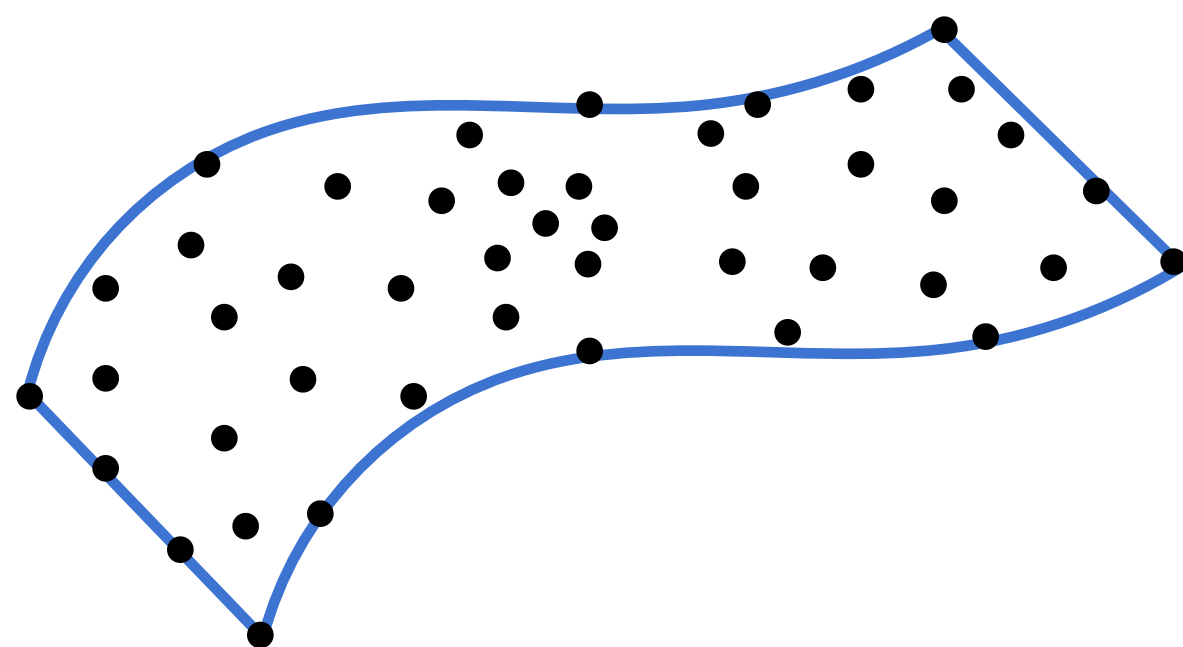
$$\nabla = \begin{bmatrix} \frac{\partial}{\partial x} & \frac{\partial}{\partial y} & \frac{\partial}{\partial z} \end{bmatrix}$$

$$\nabla^2 = \begin{bmatrix} \frac{\partial^2}{\partial x^2} & \frac{\partial^2}{\partial y^2} & \frac{\partial^2}{\partial z^2} \end{bmatrix}$$

$$\nabla \cdot \mathbf{v} = 0 \quad \text{For incompressible flows}$$

Hard to handle interfaces with other material

➔ Lagrangian particles



$$\frac{d\mathbf{v}}{dt} = \mu^2 \nabla^2 \mathbf{v} - \nabla p$$

Each particle accounts for a small mass of the fluid around it

Lower stability and accuracy

Smooth Particle Hydrodynamics



- Mesh-free: Field non-uniformly sampled by particles
 - ➡ Particle has radius (r : size), support (s : region of influence)
 - e.g., each point in the domain is supported by a small number of particles
 - ➡ mass m , position $\mathbf{x}$, velocity $\mathbf{v}$, density ρ , pressure p

- Attribute at a point is constructed using Kernel function:

$$A(\mathbf{x}) = \sum_j \frac{m_j}{\rho_j} A_j W_j(|\mathbf{x} - \mathbf{x}_j|)$$

- ➡ where Kernel function W models the strength of a particle's influence

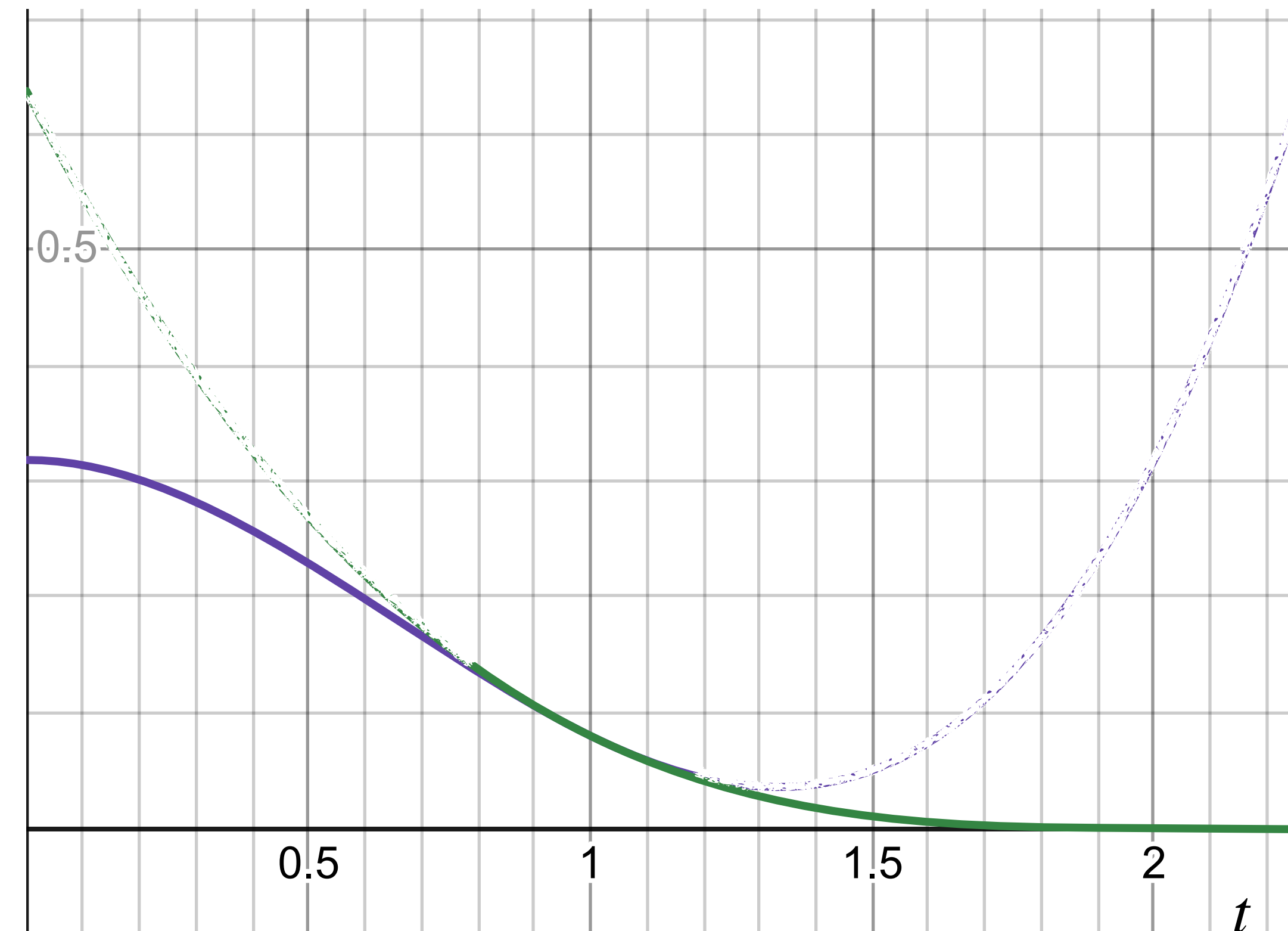


Example Kernel Function

$$W_j(\mathbf{x}) = \frac{1}{h^d} f\left(\frac{|\mathbf{x} - \mathbf{x}_j|}{h}\right)$$

d : simulation dimension
 h : smoothing radius (usually smaller than s)

$$f(q) = \frac{3}{2\pi} \begin{cases} \frac{2}{3} - q^2 + \frac{1}{2}q^3 & 0 \leq q < 1 \\ \frac{1}{6}(2 - q)^3 & 1 \leq q < 2 \\ 0 & 2 \leq q \end{cases}$$





Time Integration

$$\frac{d\mathbf{v}_i}{dt} = \mu^2 \nabla^2 \mathbf{v}_i - \frac{1}{\rho_i} \nabla p_i + \frac{\mathbf{f}_i}{m_i}$$

$$\frac{d\mathbf{v}_i}{dt} = - \sum_j m_j \left(\frac{p_i}{\rho_i^2} + \frac{p_j}{\rho_j^2} \right) \nabla W_j(\mathbf{x}_i) + \mathbf{g}$$

External acceleration

Leapfrog method:

$$\mathbf{v}_i^{n+1/2} = \mathbf{v}_i^n + \mathbf{a}_i^n \Delta t / 2$$

$$\mathbf{x}_i^{n+1} = \mathbf{x}_i^n + \mathbf{v}_i^{n+1/2} \Delta t$$

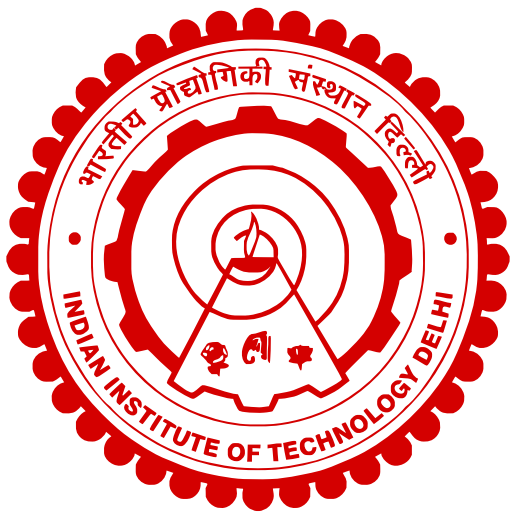
$$\mathbf{v}_i^{n+1} = \mathbf{v}_i^{n+1/2} + \mathbf{a}_i^{n+1} \Delta t / 2$$

Compare with forward Euler:

$$\mathbf{v}_i^{n+1} = \mathbf{v}_i^n + \mathbf{a}_i^n \Delta t$$

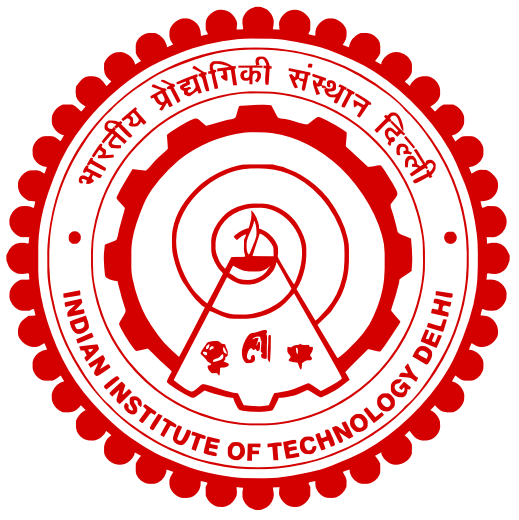
$$\mathbf{x}_i^{n+1} = \mathbf{x}_i^n + \mathbf{v}_i^n \Delta t$$

More stable (energy conserving)



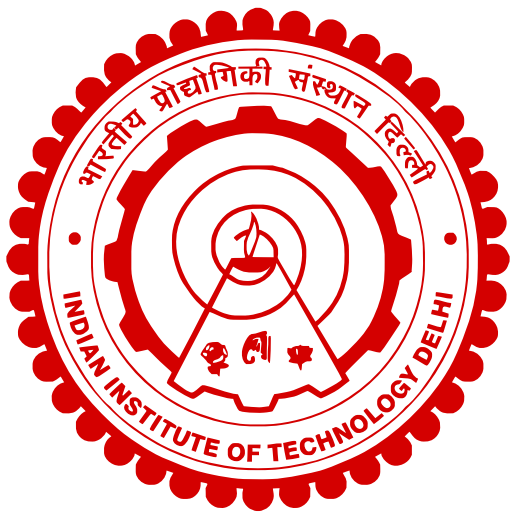
Integration Steps

- Estimate density ρ and pressure p
 - ➔ Requires finding all neighbors within influence radius
- Compute acceleration of each particle
- Integrate to compute position
- Use Δt employing CFL (Courant-Friedrich-Levy) condition:
 - ➔ no particle should move more than a ratio of its smoothing radius h
 - ➔ $\Delta t \leq \lambda \frac{h}{|v|_{\max}}$ with small λ , (e.g., 0.4)



Neighborhood Search

- To locate the neighbors within support radius impose a grid structure, storing particles per grid cell
 - ➡ Use support radius to size cells
 - ➡ Pressure limits the number of particles per cell
- To search for neighbors, loop through the adjacent cells
 - ➡ Store in Spatial hash table using grid position as key



Render SPH

- Render a splat for each particle
 - ➡ back to front order
 - ➡ Usually disks with color and transparency reducing radially (in intensity)
- Form a density volume and perform volume rendering
- Extract a triangle surface (e.g., marching cubes) and render
- For each pixel, compute the contribution of the particles