

Animating Virtual Characters

Skin deformation

Skeleton based deformation - Skinning

Objective: Deform articulated character

Idea: Use skeleton to control limbs

Articulations as rotations

Animation Skeleton

Set of frames T_i : position, orientation

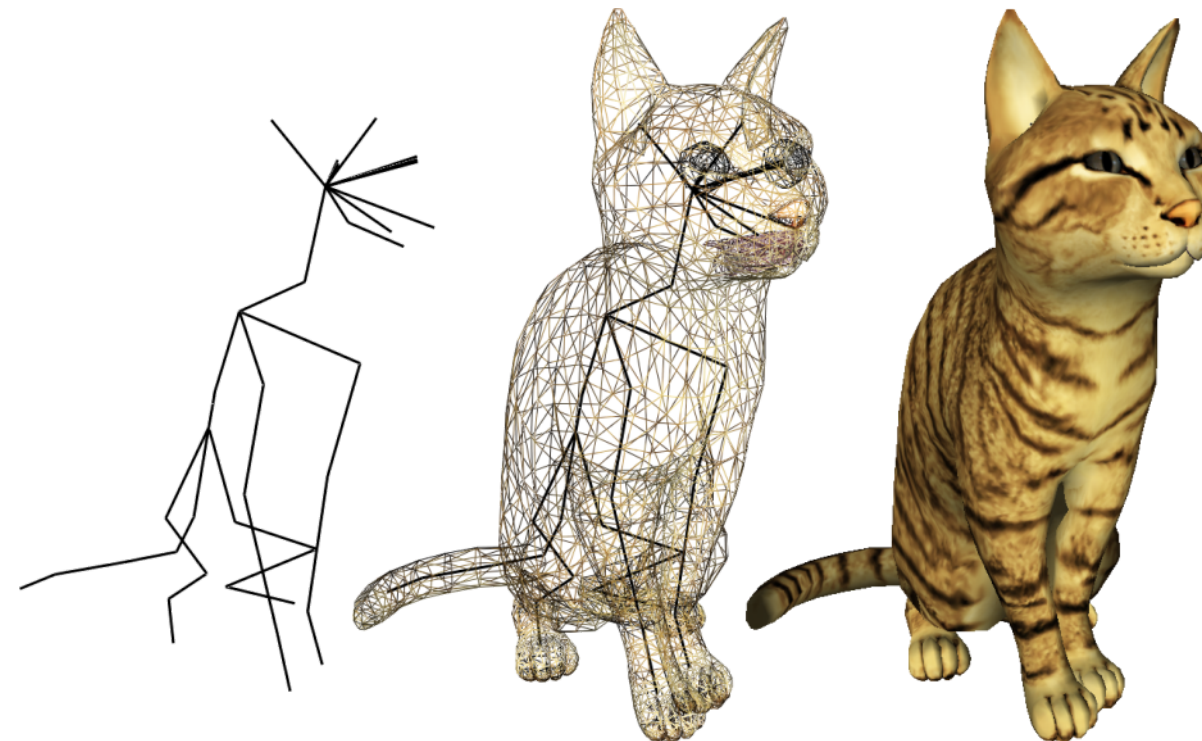
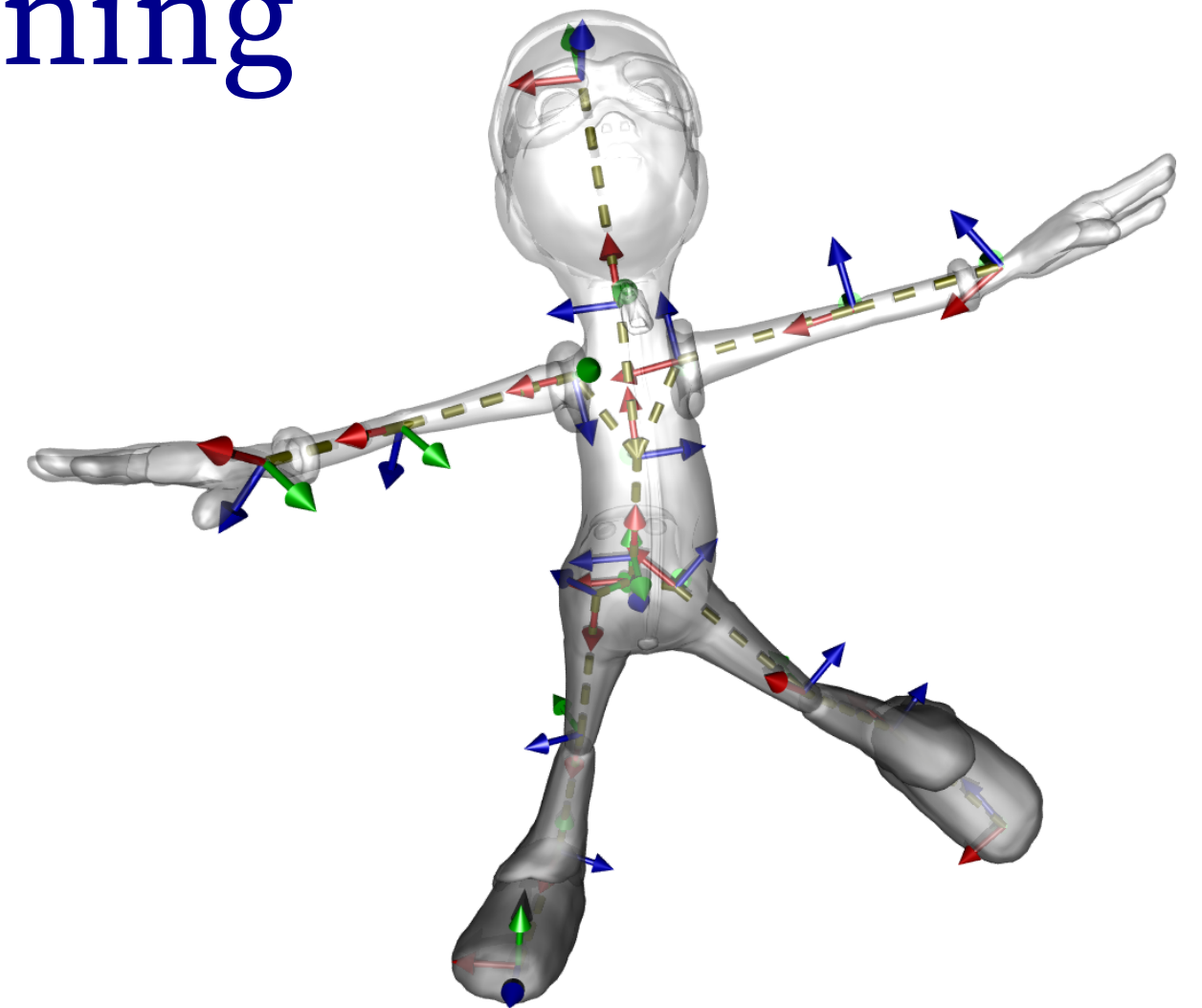
Describe non-rigid parts of the character (dof)

Terminology

Joint = A frame T_i

Bone = Segment between two joints

Note: Animation skeleton $\neq$ Anatomical skeleton



Rigid skinning

Attach rigidly subset of vertices to specific bones, described by its root joint/frame.

Vertices are following rigid deformation of their associated frame.

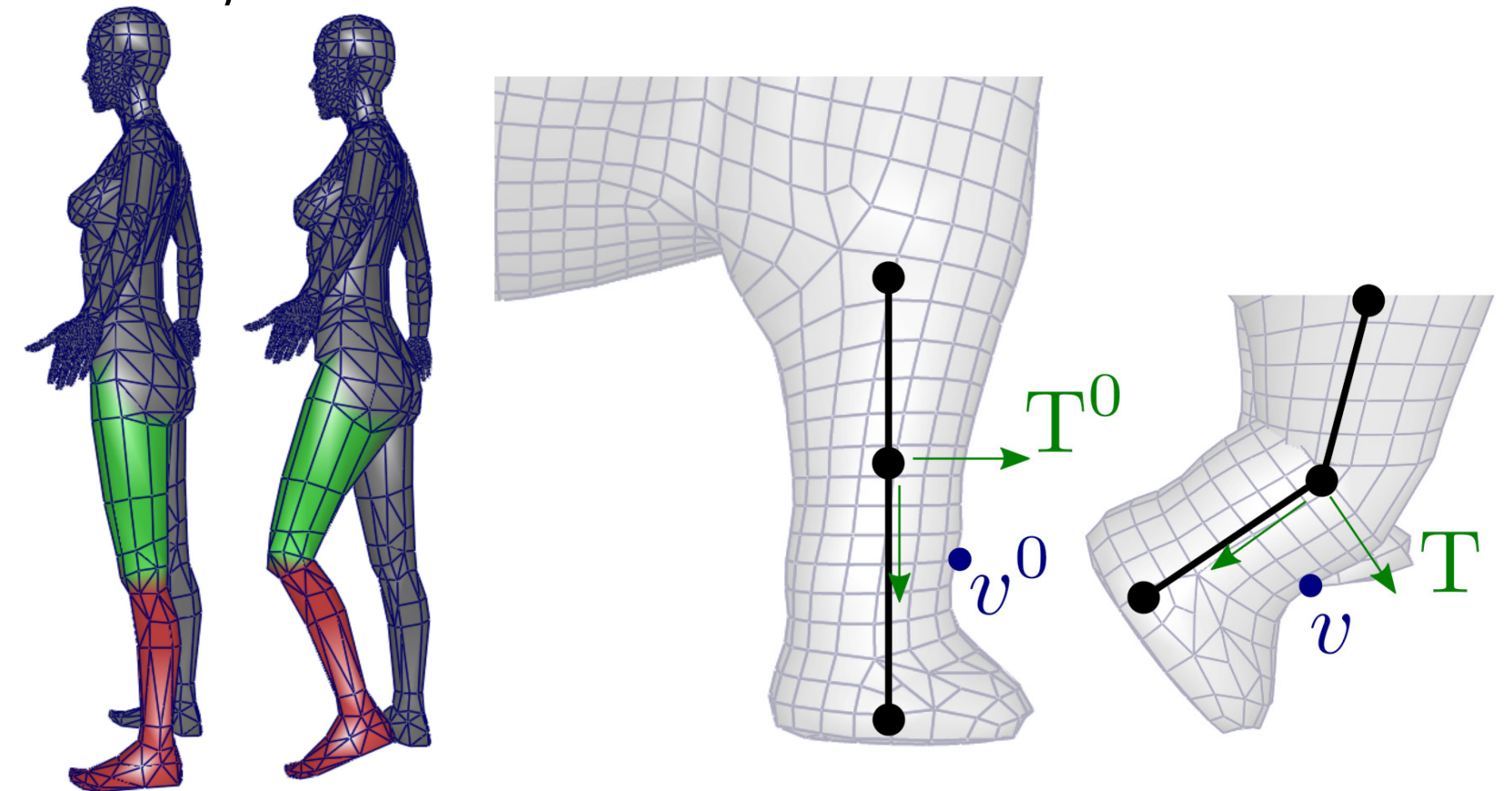
Deformation formulation

Consider, at rest pose/initial state

- A frame T^0
- A vertex v^0 attached to this frame

After deformation

- New frame T
- New vertex position v



Question: What are the new coordinates v w/r T , T^0 , v^0 ?

Rigid skinning

Attach rigidly subset of vertices to specific bones, described by its root joint/frame.

Vertices are following rigid deformation of their associated frame.

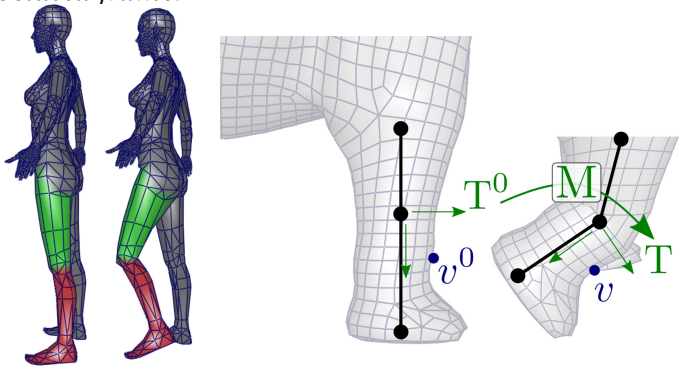
Deformation formulation

Consider, at rest pose/initial state

- A frame T^0
- A vertex v^0 attached to this frame

After deformation

- New frame T
- New vertex position v



Question: What are the new coordinates v w/r T , T^0 , v^0 ?

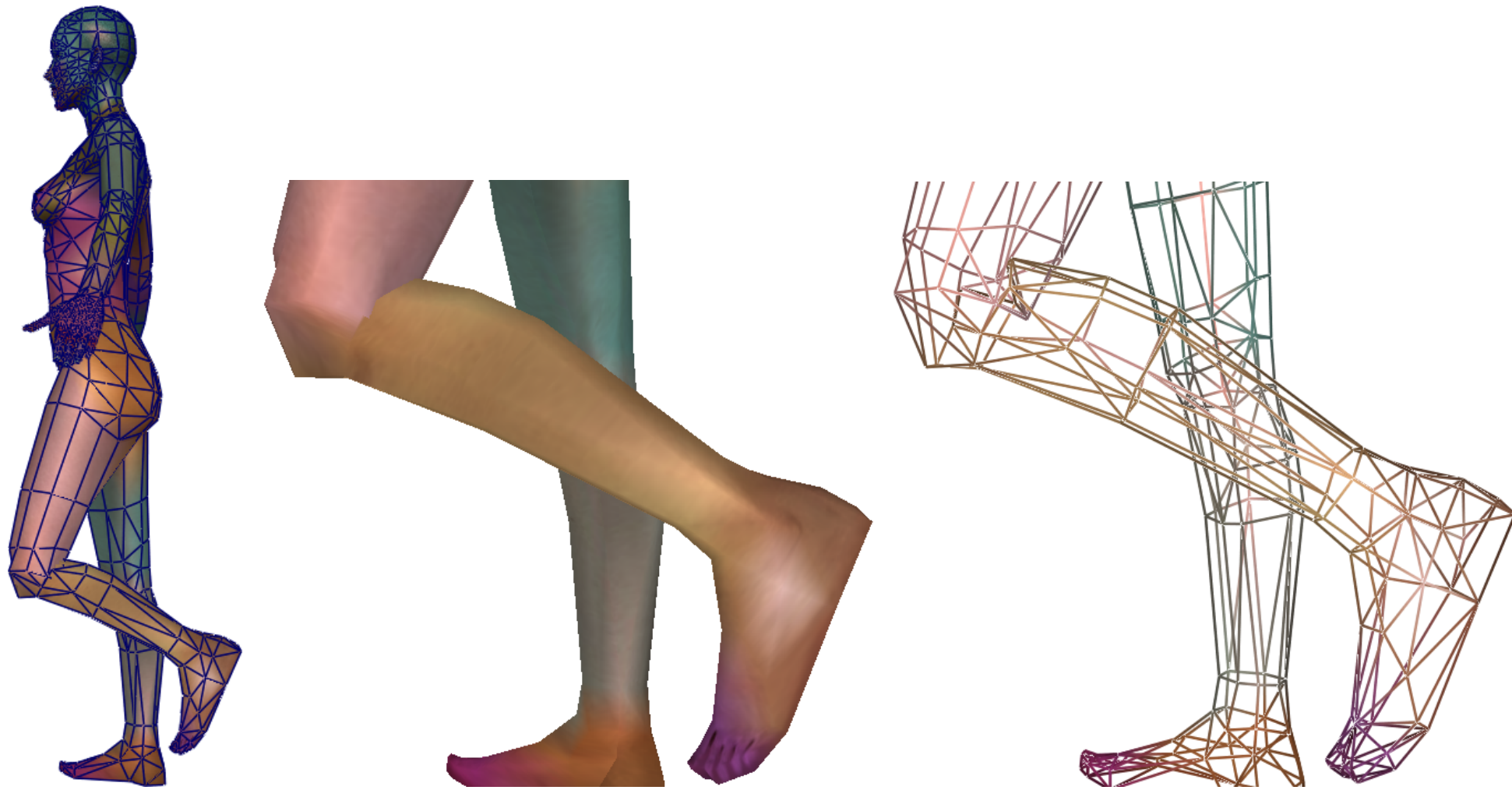
v^0 and v have similar local coordinates w/r to T^0 and T

$$\Rightarrow T^{-1}v = (T^0)^{-1}v_0$$

$$\Rightarrow v = T (T^0)^{-1}v_0 = M v_0$$

Rigid Skinning

- (+) Skeleton is easy to build
- (+) Skeleton interaction is intuitive to model rigid articulation
- (-) Discontinuities/Inter-penetrations



Idea of smooth skinning: Blend discontinuous transformation around articulation

Smooth skinning

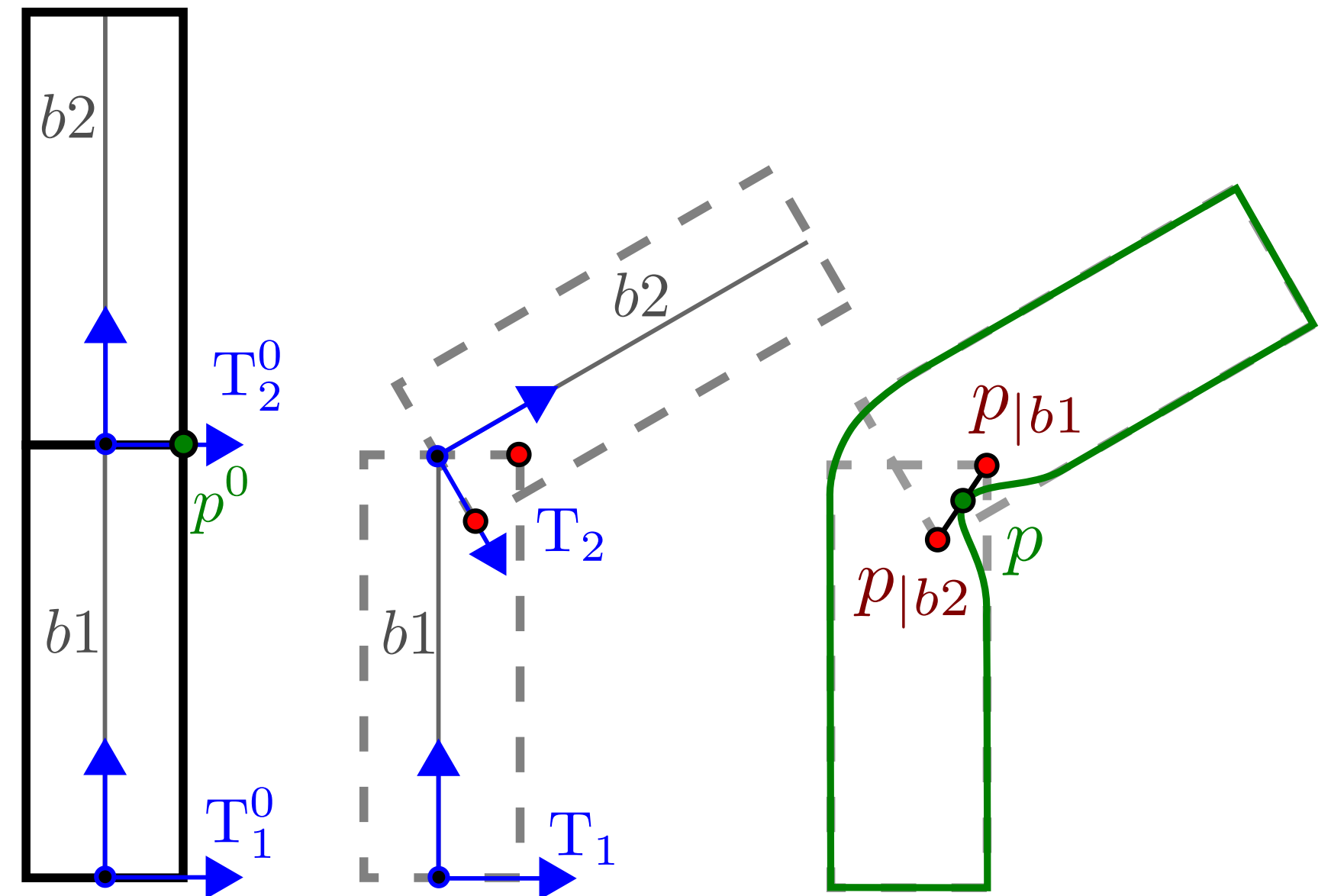
Linear Blend Skinning (LBS): Linear interpolation of positions between associated frames

Example at middle vertex position of a bending cylinder

$$p = 0.5 p_{|b1} + 0.5 p_{|b2}$$

$$p = 0.5 \underbrace{T_1 (T_1^0)^{-1}}_{M_1} p^0 + 0.5 \underbrace{T_2 (T_2^0)^{-1}}_{M_2} p^0$$

$$p = (0.5 M_1 + 0.5 M_2) p^0$$



Smooth skinning

Linear Blend Skinning (LBS): Linear interpolation of positions between associated frames

Example at middle vertex position of a bending cylinder

$$\begin{aligned} p &= 0.5\, p_{|b1} + 0.5\, p_{|b2} \\ p &= 0.5 \underbrace{T_1 (T_1^0)^{-1}}_{M_1} p^0 + 0.5 \underbrace{T_2 (T_2^0)^{-1}}_{M_2} p^0 \\ p &= (0.5\, M_1 + 0.5\, M_2) p^0 \end{aligned}$$

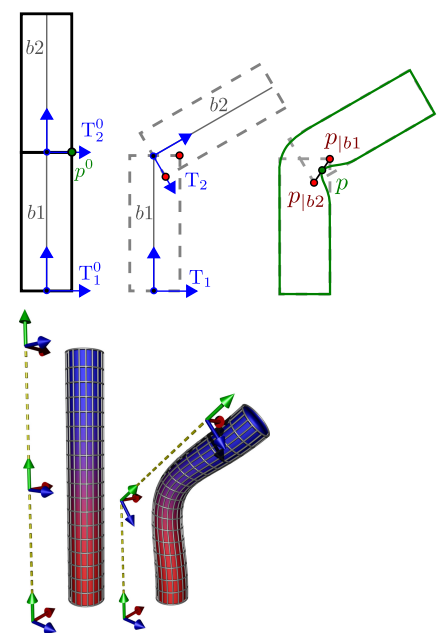
Can be generalized to arbitrary interpolation between two bones

$$p = \alpha\, p_{|b1} + (1 - \alpha)\, p_{|b2} = (\alpha M_1 + (1 - \alpha)\, M_2) p^0$$

α : skinning weights

Can be generalized to any number of bones

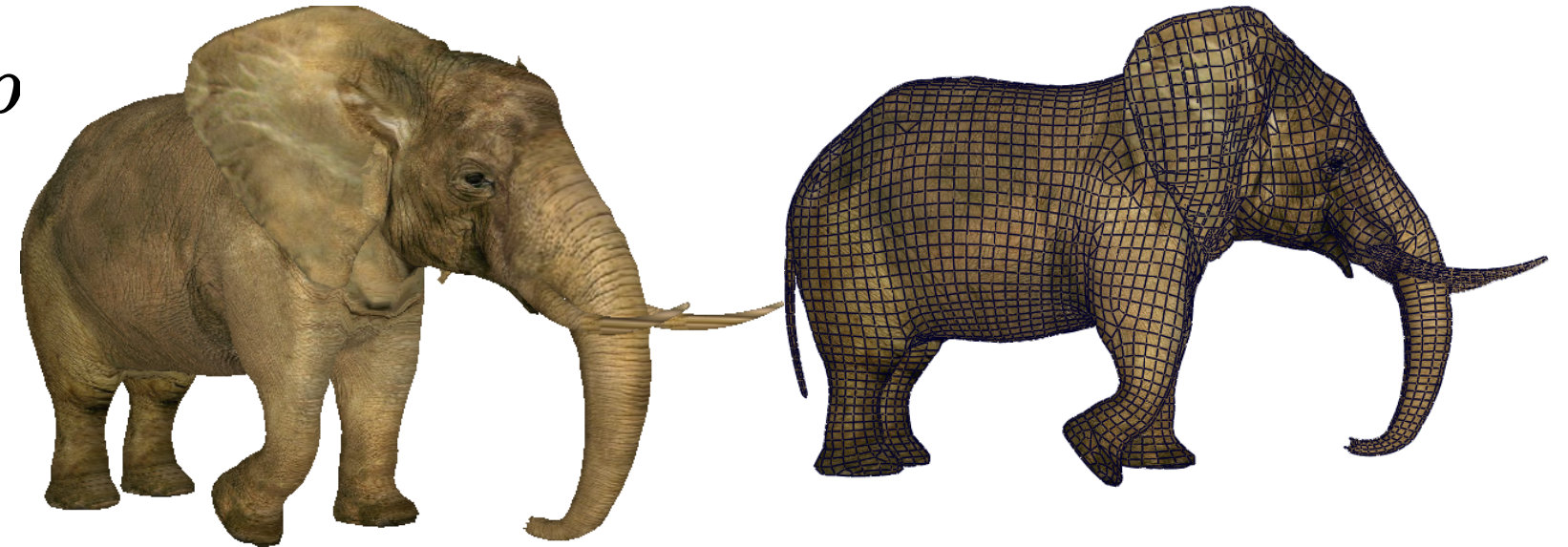
$$p = \sum_{k=0}^{N-1} \alpha_k\, p_{|bk} = \left(\sum_{k=0}^{N-1} \alpha_k\, M_k \right) p^0, \quad \sum_k \alpha_k = 1$$



Smooth skinning - Summary

$$p = \left(\sum_{k=0}^{N-1} \alpha_k M_k \right) p^0 = \left(\sum_{k=0}^{N-1} \alpha_k T_k (T_k^0)^{-1} \right) p$$

$\forall k, \alpha_k \in [0, 1], \text{ and } \sum_{k=0}^{N-1} \alpha_k = 1$

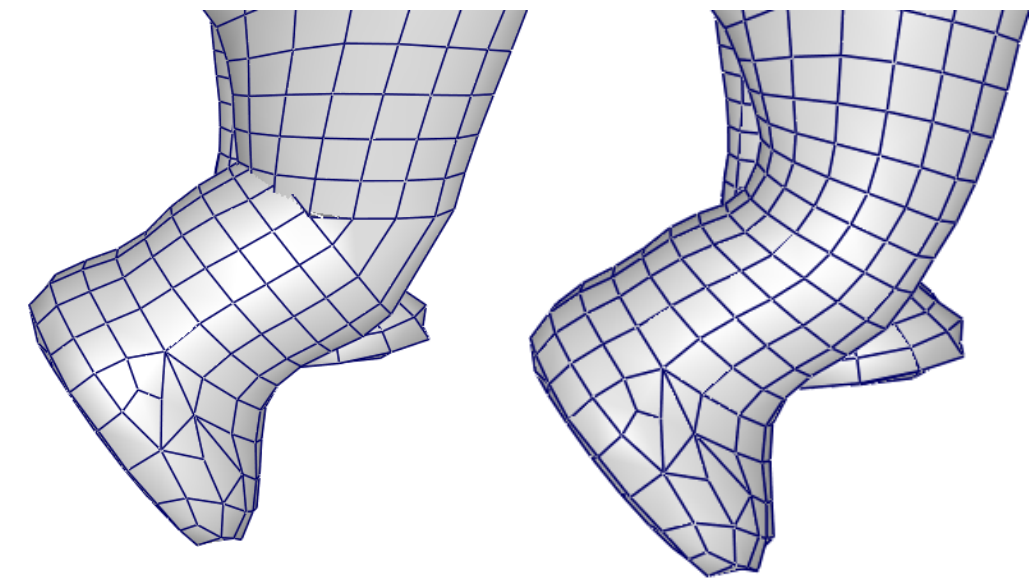


The current **standard** for almost all articulated character deformations

- Intuitive deformation
- Controlable shape (*through weights*)
- Fast to compute (GPU compatible)

matrix average, multiplication matrix-vector

⇒ Heavily used in Animation cinema & Video Game



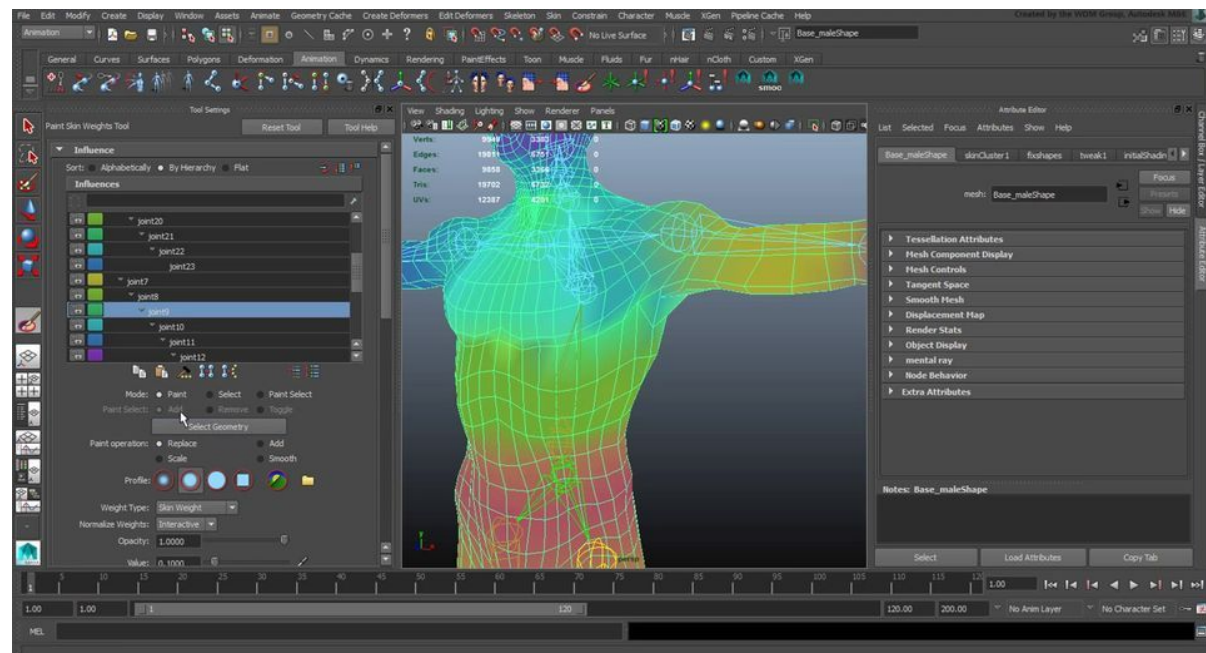
Joint-Dependent Local Deformations for Hand Animation and Object Grasping. Nadia Magnenat-Thalmann, Rochard Laperière, Daniel Thalmann. Graphics Interface, 1988

Over My Dead, Polygonal Body. Jeff Lander. Game Developer Magazine, 1998

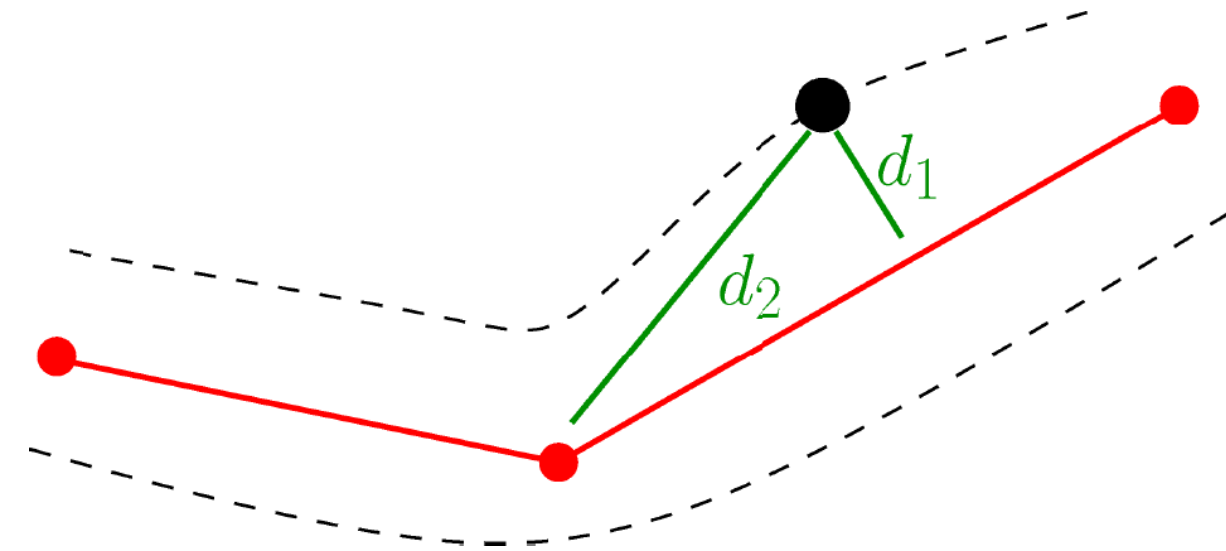
Skinning weights

How to generate skinning weights ?

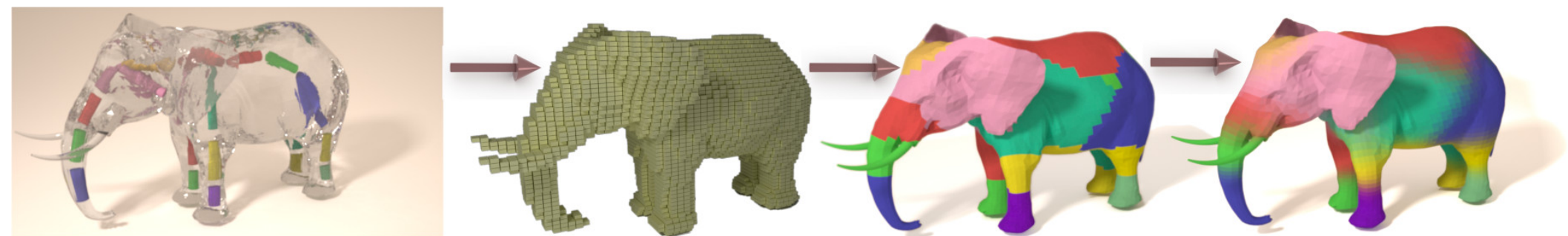
- Paint them manually
- Automatic computation



ex. Using cartesian distances: $\alpha_1 = d_1^{-1} / (d_1^{-1} + d_2^{-1})$



Or using diffusion on the volume/surface



Rigging: Associating bones and skinning weights (or any animation handle) to mesh parts

Skinning: File Format

Unfortunately few standard open format to store skinning animation data

Main open format: Collada (XML), glTF (JSON)

Common software related formats: FBX, Blend, 3DS, ...

```
krissie Maya 7.0 | ColladaMaya v2.04 | FCollada v1.14 Collada Maya Export Options:
bakeTransforms=0;exportPolygonMeshes=1;bakeLighting=0;isSampling=0;
curveConstrainSampling=0;exportCameraAsLookat=0;
exportLights=0;exportCameras=1;exportJointsAndSkin=1;
exportAnimations=1;exportTriangles=0;exportInvisibleNodes=0;
exportNormals=1;exportTexCoords=1;exportVertexColors=0;exportTangents=0;
exportTexTangents=0;exportConstraints=0;exportPhysics=0;exportXRefs=0;
dereferenceXRefs=0;cameraXFov=0;cameraYFov=1
file:///E:/maya_projets/girafe_skin/scenes/giranitex_33_bak3.ma 2008-04-16T21:56:42Z 2008-04-
16T21:56:42Z Y_UP 0.040000 0.080000 0.120000 0.160000 0.200000 0.240000 0.280000 0.320000
0.360000 0.400000 0.440000 0.480000 0.520000 0.560000 0.600000 0.640000 0.680000 0.720000
0.760000 0.800000 0.840000 0.880000 0.920000 0.960000 1.000000 1.040000 1.080000 1.120000
1.160000 1.200000 1.240000 1.280000 1.320000 1.360000 1.400000 1.440000 1.480000 1.520000
1.560000 1.600000 1.640000 1.680000 1.720000 1.760000 1.800000 1.840000 1.880000 1.920000
1.960000 2.000000 2.040000 2.080000 2.120000 2.160000 2.200000 2.240000 2.280000 2.320000
2.360000 2.400000 2.440000 2.480000 2.520000 2.560000 2.600000 2.640000 2.680000 2.720000
2.760000 2.800000 2.840000 2.880000 2.920000 2.960000 3.000000 3.040000 3.080000 3.120000
3.160000 3.200000 3.240000 3.280000 3.320000 3.360000 3.400000 3.440000 3.480000 3.520000
3.560000 3.600000 3.640000 3.680000 3.720000 3.760000 3.800000 3.840000 3.880000 3.920000
3.960000 4.000000 4.040000 4.080000 4.120000 4.160000 4.200000 4.240000 4.280000 4.320000
4.360000 4.400000 4.440000 4.480000 4.520000 4.560000 4.600000 4.640000 4.680000 4.720000
4.760000 4.800000 4.840000 4.880000 4.920000 4.960000 5.000000 5.040000 5.080000 5.120000
5.160000 5.200000 5.240000 5.280000 5.320000 5.360000 5.400000 5.440000 5.480000 5.520000
5.560000 5.600000 5.640000 5.680000 5.720000 5.760000 5.800000 5.840000 5.880000 5.920000
5.960000 6.000000 6.040000 6.080000 6.120000 6.160000 6.200000 6.240000 6.280000 6.320000
6.360000 6.400000 6.440000 6.480000 6.520000 6.560000 6.600000 6.640000 6.680000 6.720000
6.760000 6.800000 6.840000 6.880000 6.920000 6.960000 7.000000 7.040000 7.080000 7.120000
7.160000 7.200000 7.240000 7.280000 7.320000 7.360000 7.400000 7.440000 7.480000 7.520000
7.560000 7.600000 7.640000 7.680000 7.720000 7.760000 7.800000 7.840000 7.880000 7.920000
7.960000 8.000000 8.040000 8.080000 8.120000 8.160000 8.200000 8.240000 8.280000 8.320000
8.360000 8.400000 8.440000 8.480000 8.520000 8.560000 8.600000 8.640000 8.680000 8.720000
8.760000 8.800000 8.840000 8.880000 8.920000 8.960000 9.000000 9.040000 9.080000 9.120000
```

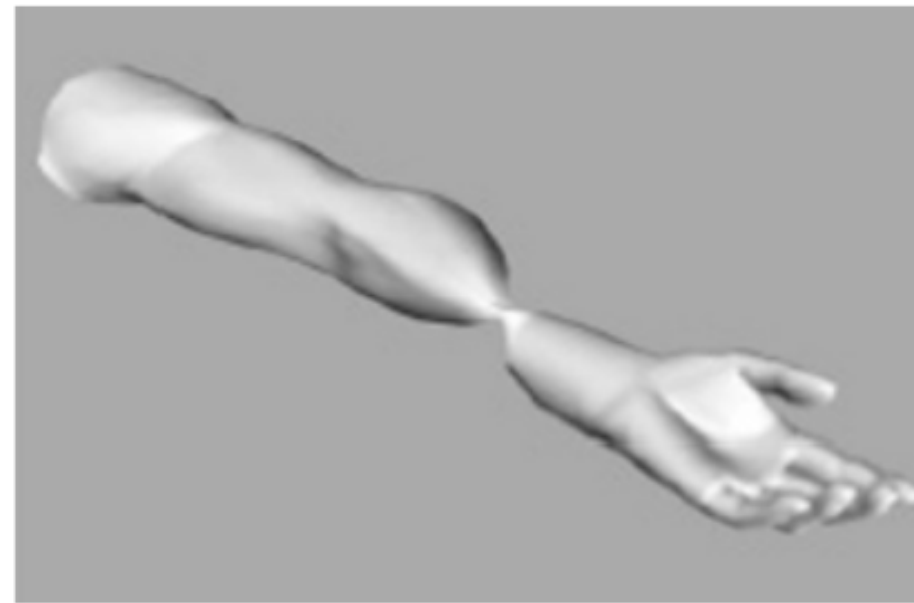
collada.dae

```
{
  "images": [{
    "url": "MarineCv2_color.jpg",
    "uuid": "1F3E9A1C-DDD2-3F4D-80C1-BC5F609DC23B",
    "name": "MarineCv2_color.jpg"
  }],
  "geometries": [{
    "type": "Geometry",
    "uuid": "4181C04A-B75A-3FDC-9A89-89A1D2BCE0AF",
    "data": {
      "uvs":
[[[0.270354,0.313478,0.271947,0.319847,0.275729,0.309655,0.274243,0.3087,0.272832,0.307868,0.269884
,0.310585,0.226132,0.380806,0.222211,0.383513,0.22838,0.387803,0.230469,0.384471,0.220584,0.387519
,0.227778,0.393438,0.236551,0.373639,0.235651,0.37412,0.236463,0.379508,0.237612,0.379488,0.239952
,0.371796,0.240752,0.379751,0.245983,0.382396,0.244072,0.372847,0.029897,0.380277,0.026492,0.37821
7,0.024403,0.381866,0.027378,0.383736,0.23588,0.388524,0.23554,0.386494,0.234758,0.383916,0.050509
,0.560407,0.047071,0.557888,0.045167,0.558824,0.048244,0.563377,0.064365,0.566106,0.055853,0.56226
3,0.05439,0.567096,0.064598,0.572284,0.04127,0.560623,0.045026,0.567068,0.051922,0.572939,0.058767
,0.576525,0.037322,0.561994,0.040429,0.570751,0.047131,0.578106,0.034887,0.563887,0.036649,0.57368
8,0.041729,0.581221,0.051832,0.587997,0.054417,0.583009,0.033166,0.564334,0.033893,0.574381,0.0382
76,0.583055,0.046687,0.590068,0.029228,0.564585,0.030582,0.575915,0.035745,0.587145,0.043417,0.594
797,0.012878,0.575167,0.025786,0.576822,0.024107,0.563407,0.011428,0.55775,0.019835,0.589523,0.031
426,0.590171,0.056273,0.542792,0.059073,0.545878,0.065176,0.5447,0.063035,0.542068,0.048447,0.5468
91,0.053936,0.549727,0.04472,0.552324,0.048873,0.553322,0.055773,0.539823,0.044725,0.545943,0.0644
16,0.538785,0.040454,0.553447,0.054384,0.536842,0.041786,0.545212,0.065807,0.534961,0.037489,0.554
075,0.049928,0.535922,0.039805,0.54519,0.035552,0.554559,0.046737,0.533753,0.037925,0.544124,0.033
877,0.554064,0.043337,0.529892,0.034539,0.543152,0.030422,0.55413,0.04709,0.510318,0.042159,0.5067
9,0.026639,0.514508,0.031141,0.519839,0.038545,0.525958,0.02404,0.532227,0.030178,0.540512,0.01661
9,0.542977,0.02613,0.552544,0.019243,0.52681,0.072806,0.54418,0.068036,0.541536,0.069568,0.544295,
0.072537,0.546332,0.074457,0.542001,0.070694,0.540527,0.075113,0.538587,0.072083,0.53632,0.070949,
0.565762,0.071836,0.562397,0.067553,0.560436,0.064941,0.562816,0.080185,0.557515,0.078931,0.555483
,0.076466,0.562489,0.076226,0.565704,0.058861,0.594811,0.064195,0.589419,0.069791,0.550726,0.06658
1,0.555646,0.071219,0.555536,0.07257,0.547845,0.061829,0.558444,0.058373,0.559956,0.058247,0.55486
4,0.053888,0.556612,0.529304,0.679074,0.528585,0.691852,0.525233,0.680917,0.521958,0.67057,0.00535
1,0.580703,0.020611,0.59504,0.002847,0.554083,0.040637,0.597389,0.01118,0.540222,0.054475,0.599187
,0.046209,0.601951,0.073623,0.5477,0.073842,0.556482,0.06595,0.597735,0.05175,0.602881,0.062732,0.
603138,0.513246,0.652152,0.507764,0.643571,0.505233,0.633772,0.522766,0.650073,0.06824,0.506816,0.
075348,0.508814,0.073568,0.50543,0.068519,0.503123,0.062065,0.55231,0.067781,0.548426,0.055314,0.5
17723,0.050992,0.513294,0.062031,0.506743,0.064039,0.51368,0.068682,0.511233,0.061823,0.58071,0.08
1265,0.530181,0.072142,0.521567,0.07258,0.520477,0.078422,0.525058,0.082656,0.542584,0.081024,0.56
```

.json

Linear Blend Skinning - Limitations

- Non-trivial rigging settings
- Artifacts for large rotations: Candy wrapper, Collapsing elbow

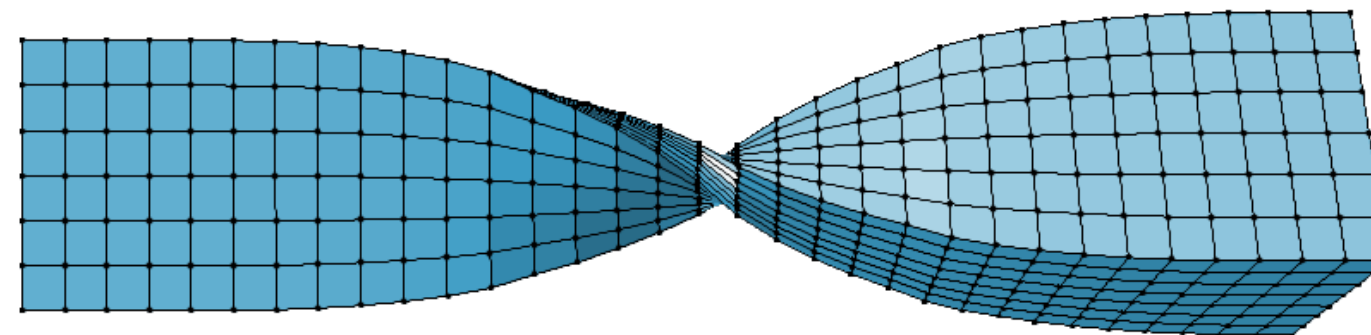


Candy wrapper



Collapsing elbow

Linear blending between rigid transformation matrices



Character Animation

Skeletal Animation

Skeleton structure

Characteristics

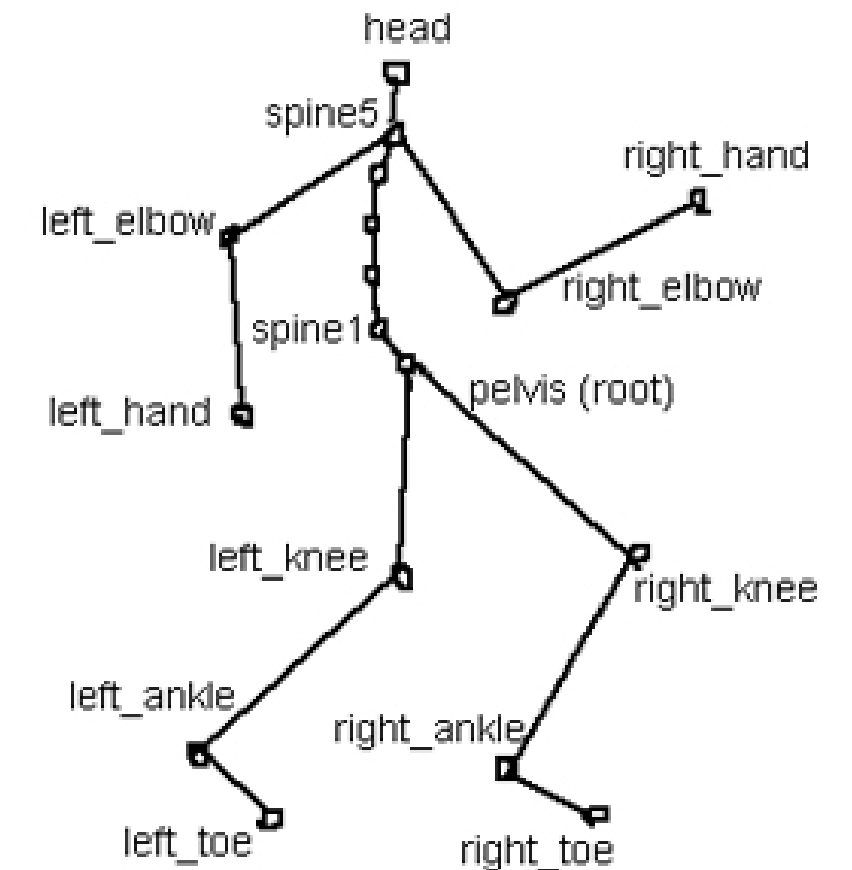
- Hierarchical representation

All children follow the transformation of the parent

Need to define a root: usually at the hips/pelvis

- Convenient to express local deformation with respect to the parent

ex. Rotate knee from 20°



Converting local to global frames/joint coordinates

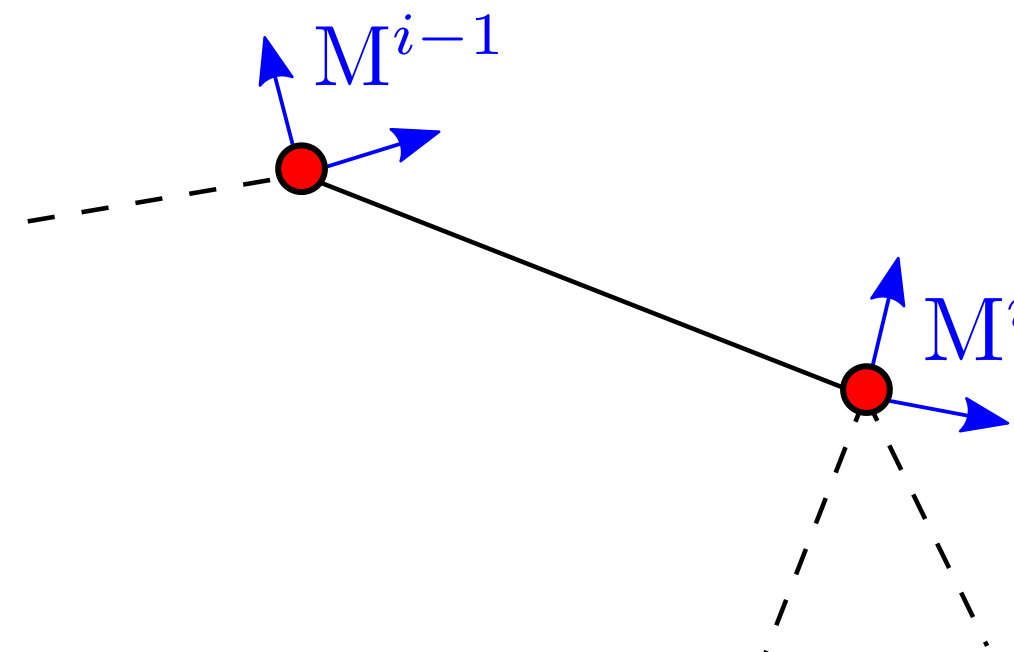
- With 4x4 matrices M

$$M_{global}^i = M_{global}^{i-1} M_{local}^i$$

- With translation t , rotation R

$$R_{global}^i = R_{global}^{i-1} R_{local}^i$$

$$t_{global}^i = t_{global}^{i-1} + R_{global}^{i-1} t_{local}^i$$



Encoding hierarchical skeleton

- Simplest encoding based on index within vector

```
Geometry=[M0, M1, M2, M3, M4, M5]
```

```
Parent = [-1,0,1,1,0,4]
```

- Convert local coordinates to global coordinates

```
local (Geometry) <- std::vector of rotation (r), translation (p)
```

```
global (Geometry) <- std::vector of rotation (r), translation (p)
```

```
global[0] = local[0];
```

```
for(size_t k=1; k<N; ++k)
```

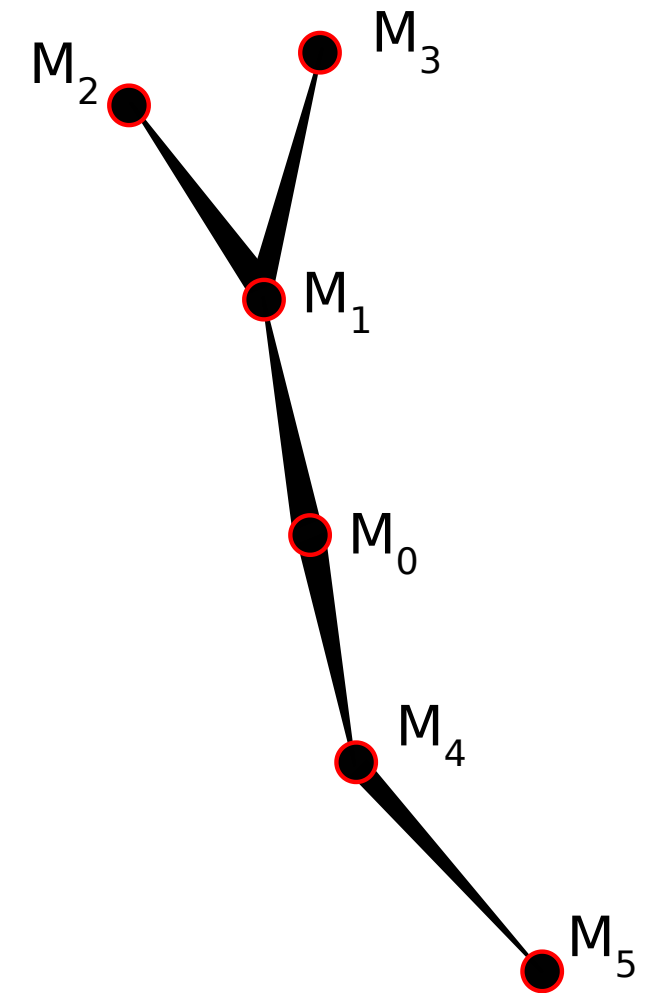
```
{
```

```
    int parent = Parent[k];
```

```
    global[k].r = global[parent].r * local[k].r;
```

```
    global[k].p = global[parent].r * local[k].p + global[parent].p;
```

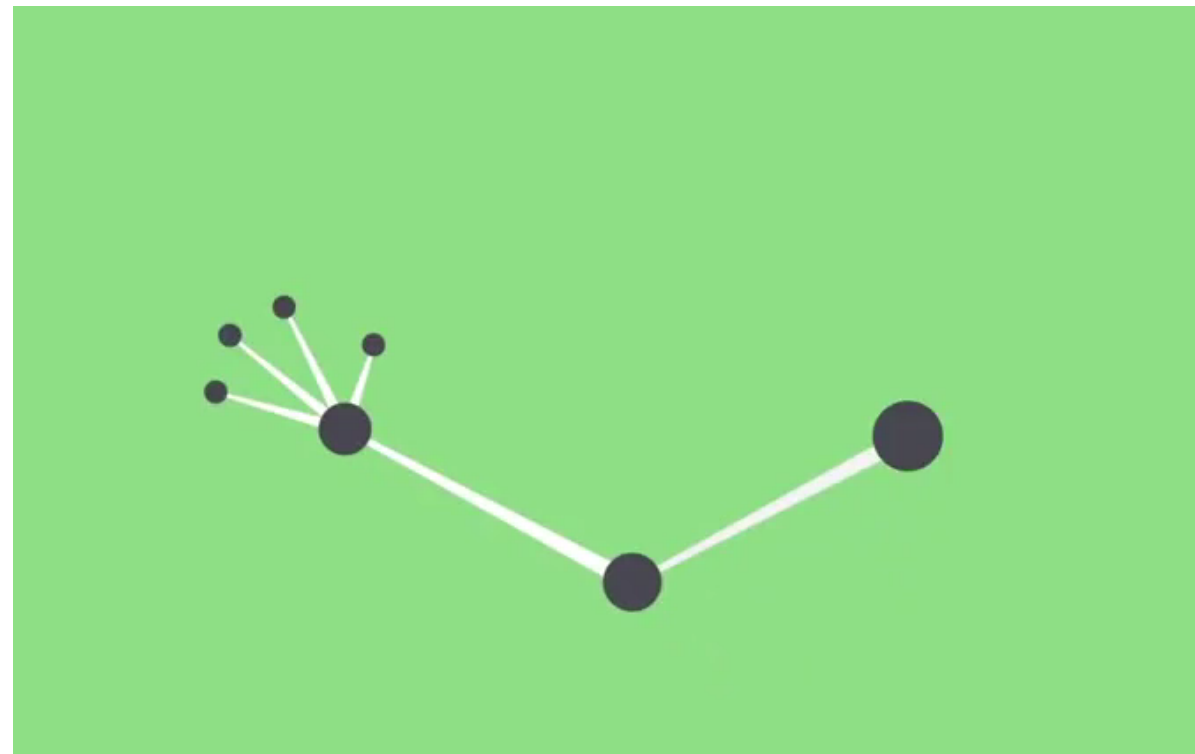
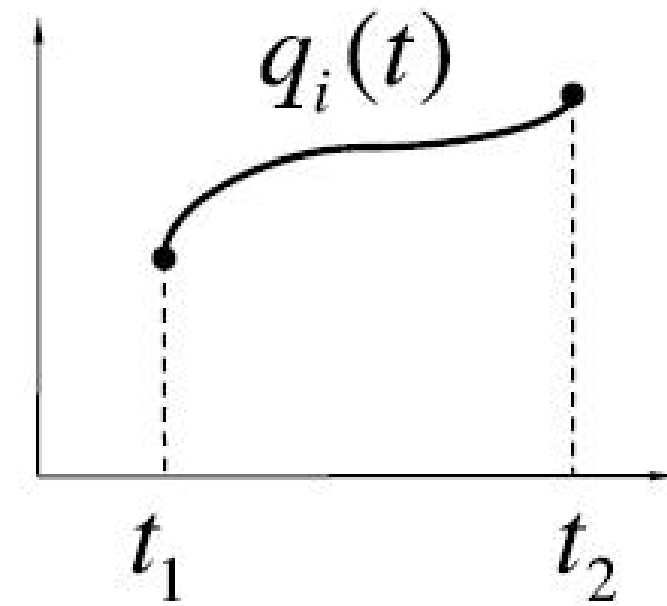
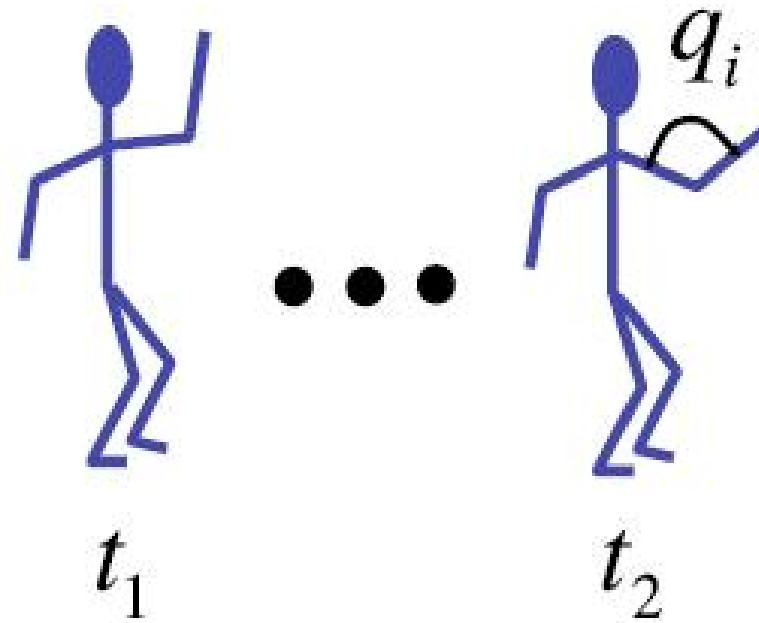
```
}
```



Forward kinematics

FK - Forward Kinematics

- Each joint angle is set manually
 - Adapted to set orientation of specific parts
 - Interpolate rotations during animation
- (+) Generates curved trajectory naturally



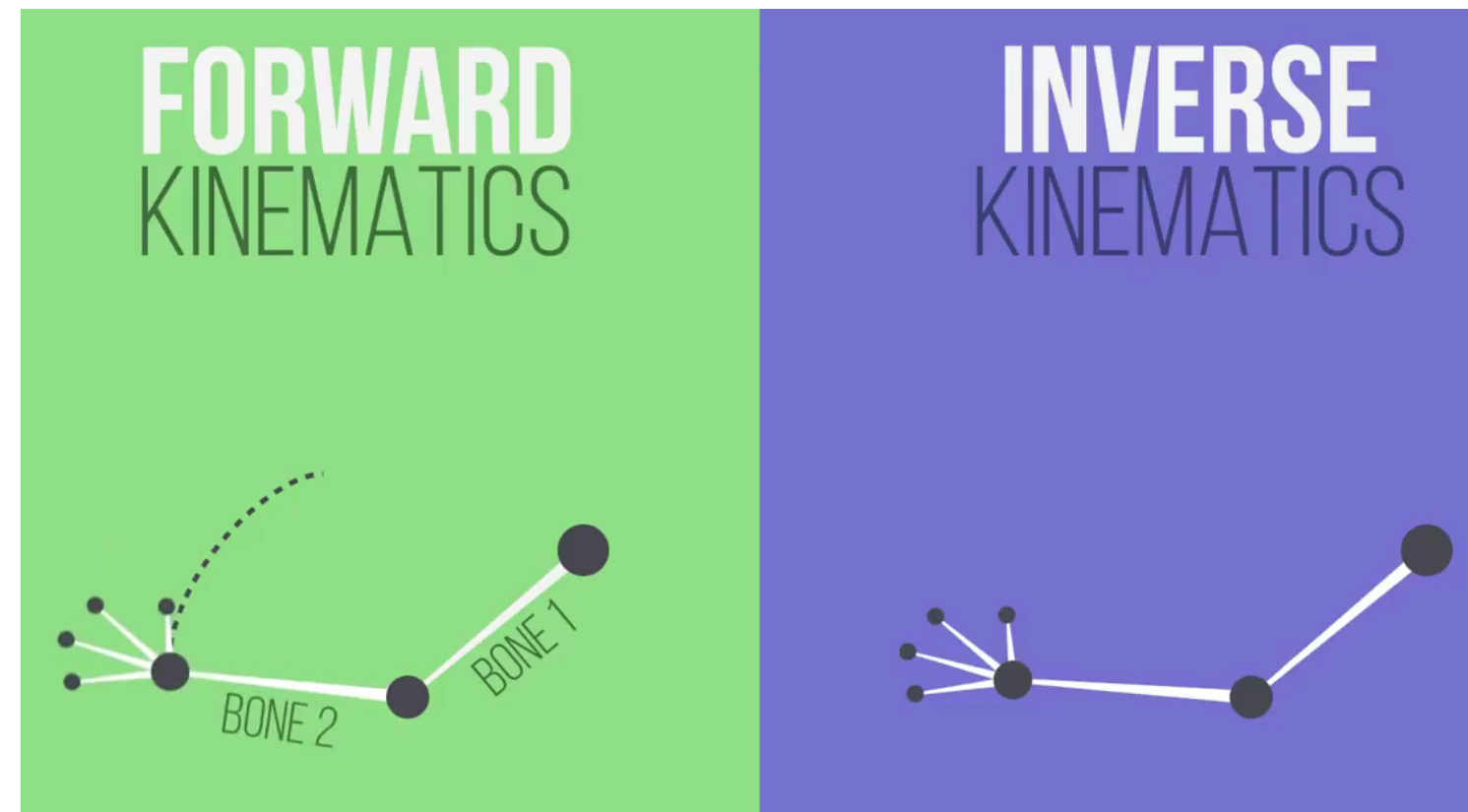
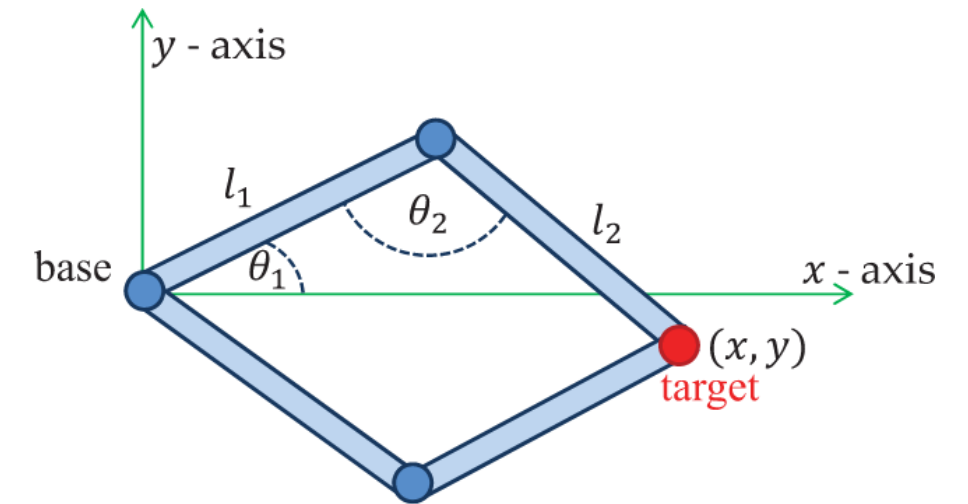
Inverse Kinematics

IK: Inverse Kinematics

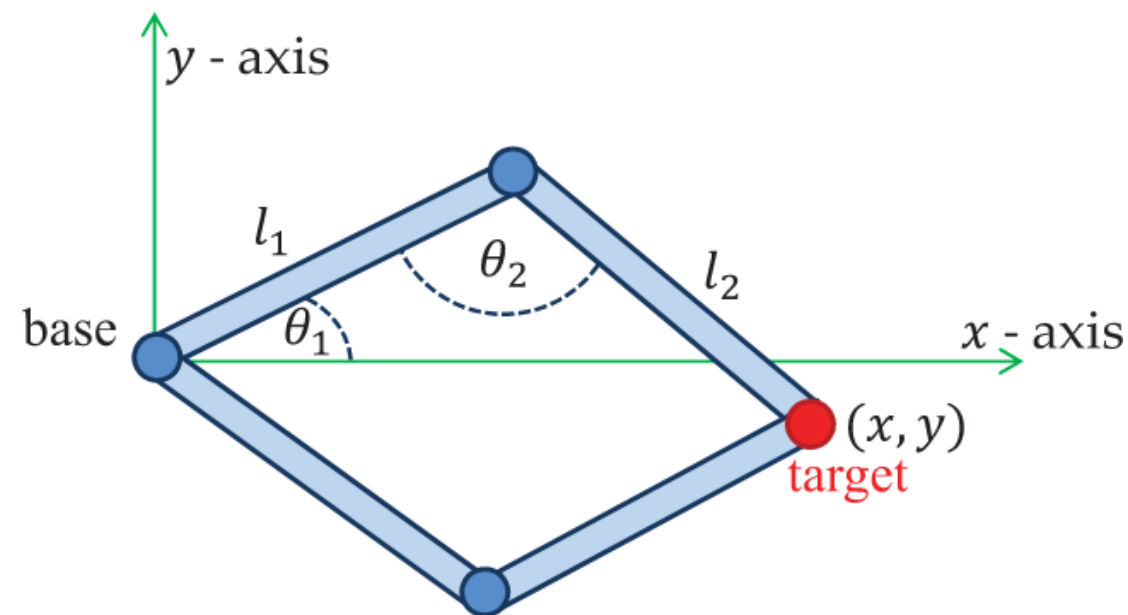
- Describe position (/and orientation) of *end-effectors* (contact, walking, etc)
- Compute joint angles reaching this position

$$p_k = p_0 + \sum_{i=0}^{k-1} l_i R_i u_i$$

R_i can be expressed with various rotation parameters (Matrices, Euler angles, axis/angle, quaternions, etc)



IK Example with two bones



Inverse Kinematics Techniques in
Computer Graphics: A Survey. A.
Aristidou et al. STAR EG. 2017.

In general the general case $p_k = f(\theta_i)$

- Look for $\theta_i = f^{-1}(p_k)$
- f is a non linear function
- There may exists multiple solutions (or none)
- Solutions may exhibits discontinuities
- Closed form solution are not available

Two solutions defined by

$$\cos(\theta_1) = \frac{l_1^2 + x^2 + y^2 - l_2^2}{2l_1 \sqrt{x^2 + y^2}}$$

$$\cos(\theta_2) = \frac{l_1^2 + l_2^2 - (x^2 + y^2)}{2l_1 l_2}$$

Some attempts for explicit solutions in specific
cases

Real-Time Inverse Kinematics Techniques for
Anthropomorphic Limbs. D. Tolani et al.
Graphical Models, 2000.

Analytical inverse kinematics with bodyposture
control. M. Kallmann. Comp. Anim. & Virt.
Worlds, 2008



IK: Numerical methods

Numerical inversion of $p = f(\theta)$, $\theta = (\theta_0, \dots, \theta_{N-1})$.

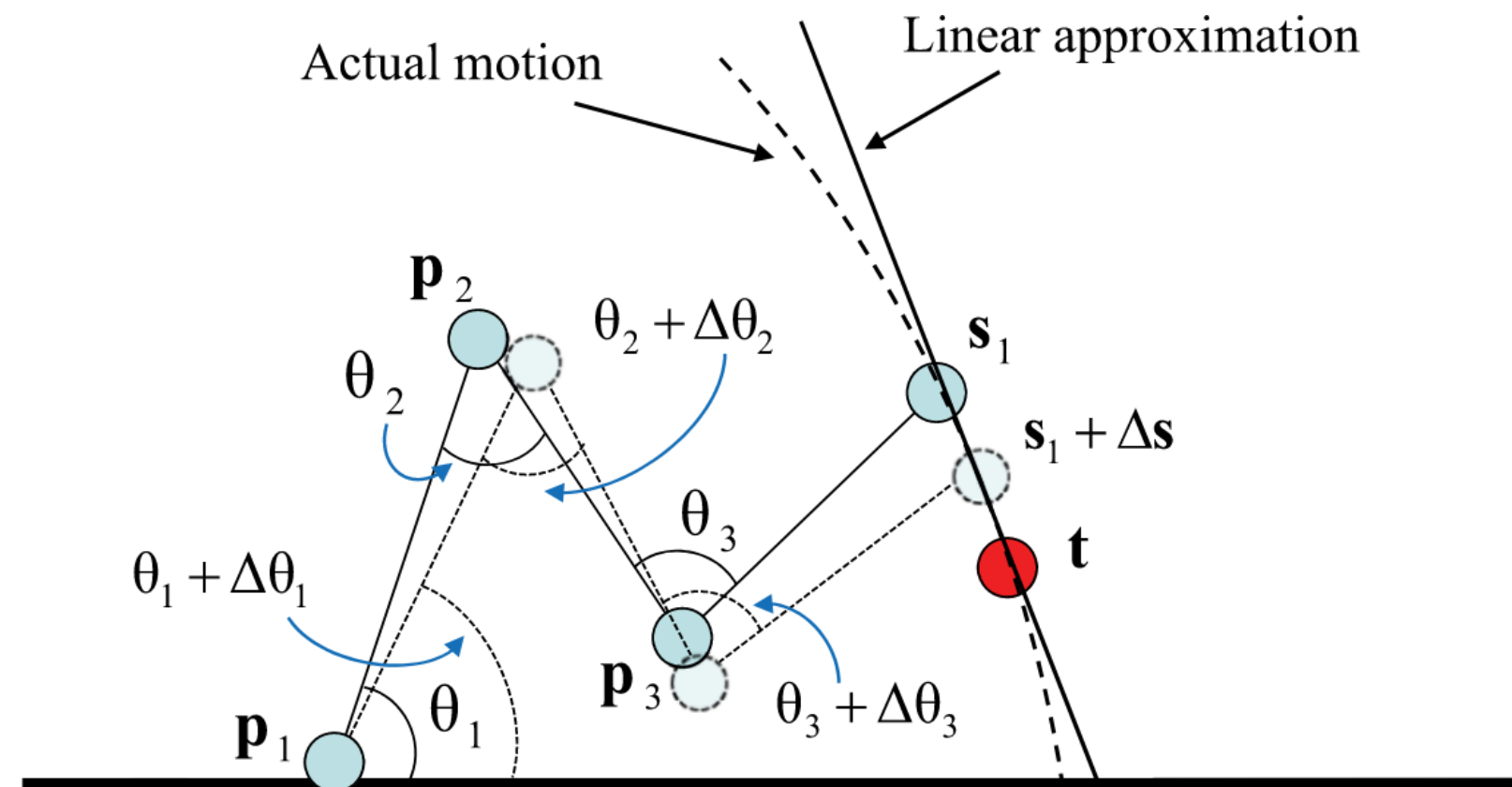
Consider small step size $p \rightarrow p + \Delta p$

$$\Delta p \simeq \underbrace{\left(\frac{\partial f}{\partial \theta} \right)}_J \Delta \theta$$

J - Jacobian matrix.

→ Not square ($3 \times N$), not invertible.

Unknown > # constraints



IK: Numerical methods

Several possible approaches to solve $J \Delta\theta = \Delta p$

- Pseudo Inverse

$$\Delta\theta = J^+ \Delta p, \text{ with } JJ^+ = I$$

$$J^+ = J^T (J J^T)^{-1}$$

- Can also be computed using SVD: $J^+ = V \Sigma^+ U^T$

$$\Sigma_{ii} = \sigma_i, \Sigma_{ii}^+ = 1/\sigma_i \text{ if } \sigma_i \neq 0, 0 \text{ otherwise.}$$

- Adding damping to compensate for singularities

$$\Delta\theta = J^T (J J^T + \lambda^2 I)^{-1} \Delta p$$

- Using Newton's methods

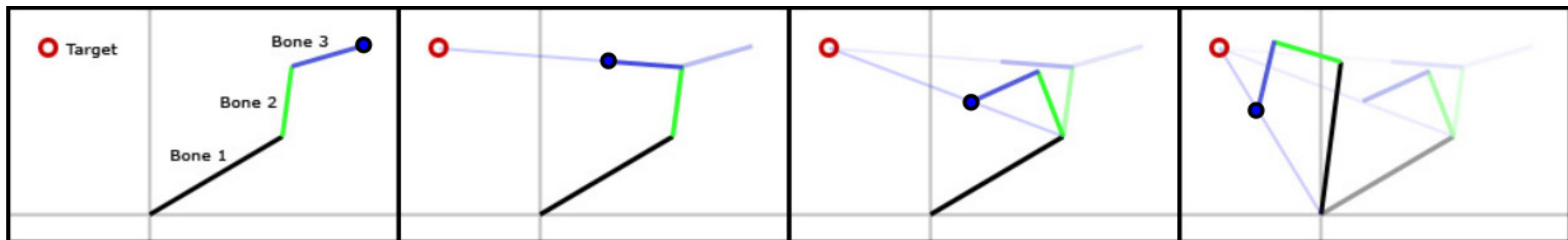
Inverse Kinematics Techniques in Computer Graphics: A Survey. A. Aristidou. STAR EG 2017

IK: Heuristic approach

Cyclic Coordinates Descent (CCD)

- Iteratively rotates joint $j^N \rightarrow j^{i-1} \rightarrow \dots \rightarrow j^1$ for the extremity (end effector) to be as close as possible from the target.
 - = *End-effector aligned with the segment (joint,target)*
- Restart until convergence

Making Kine More Flexible. Jeff Lander. Game Dev, 1998.

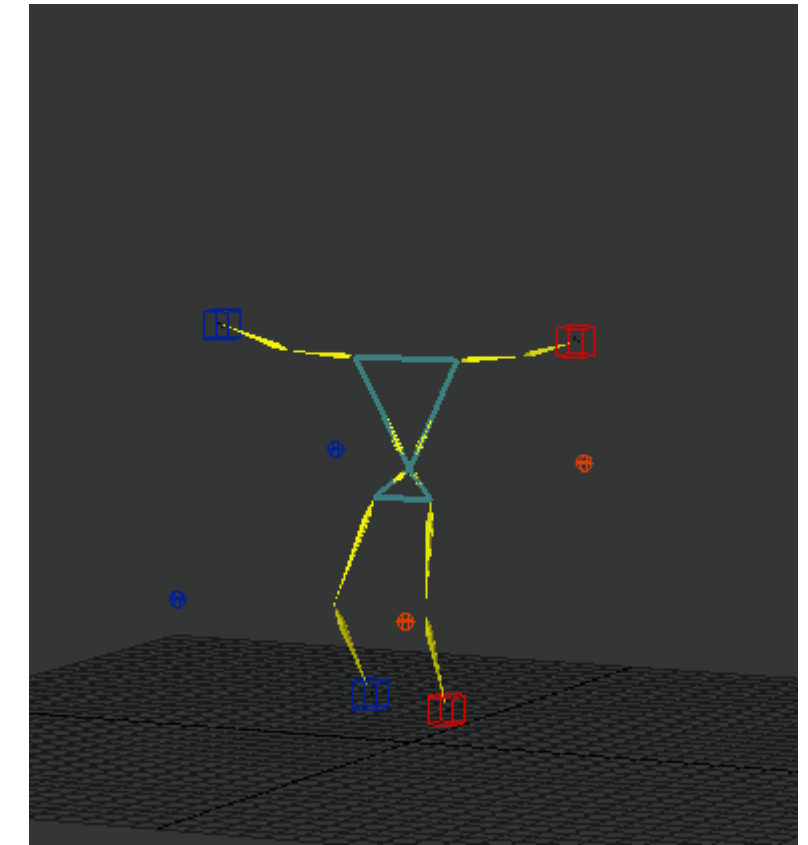


IK: Heuristic approach

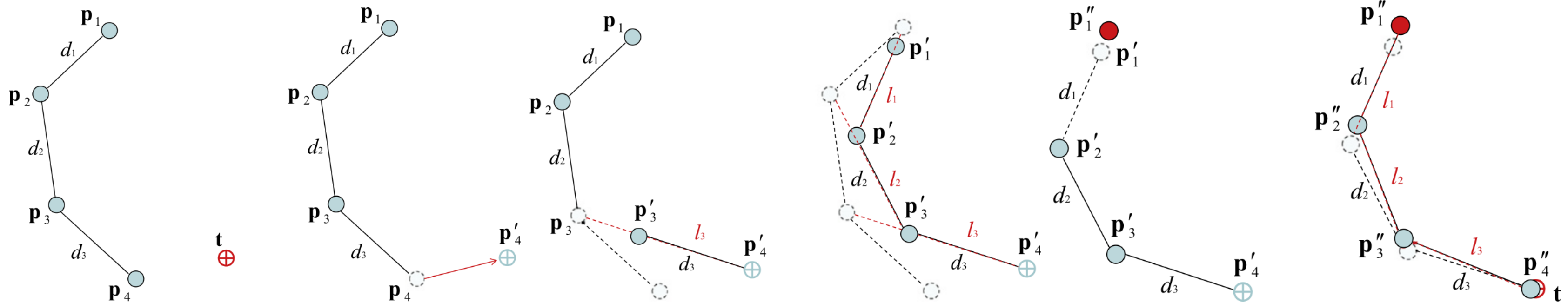
Fabrik

Iterate between

- Forward direction: Match the end-effector target
propagate changes toward previous position to match bones' length.
- Backward direction: Match the starting position
propagate changes toward following positions to match bones' length.

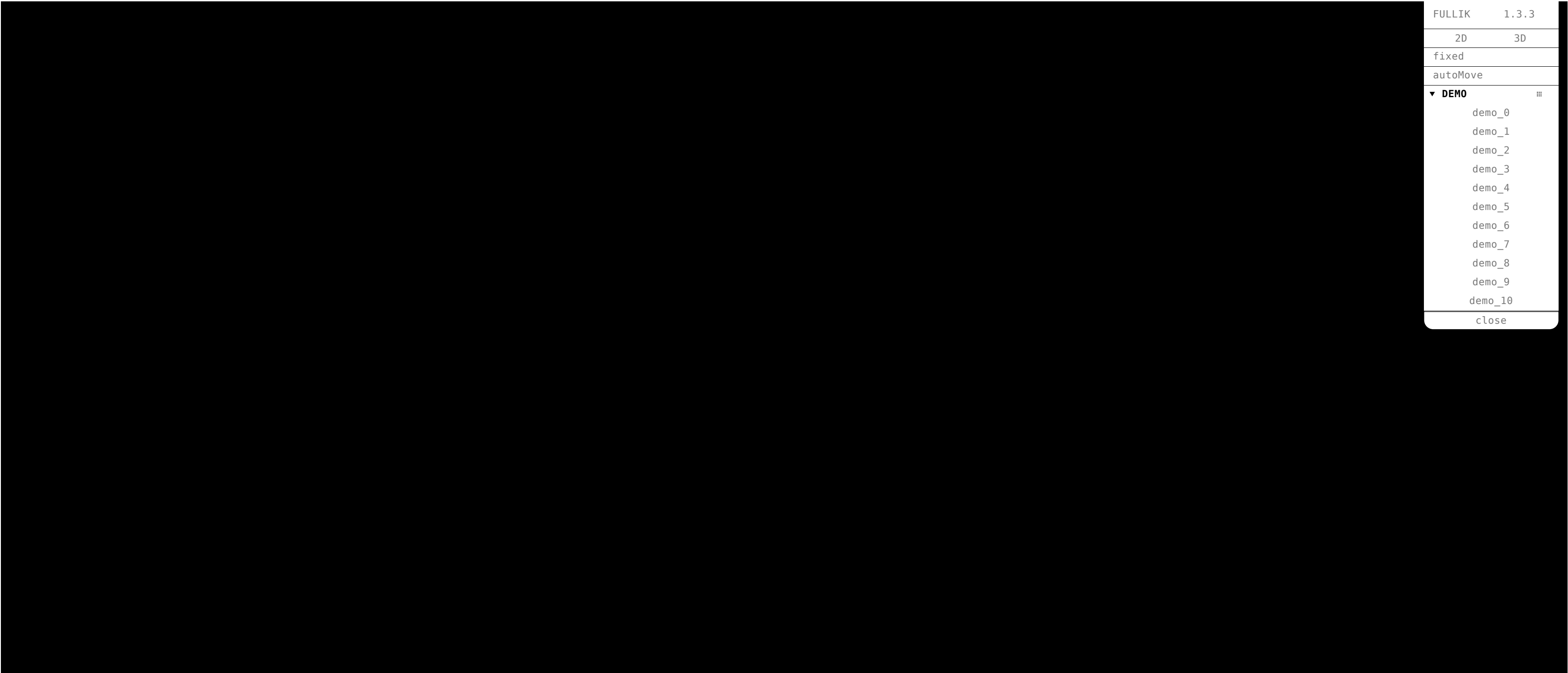


A fast iterative solver for the Inverse Kinematics problem. A.
Aristidou. Graphical Models 2011.



Inverse Kinematics

Example



Synthesizing and controlling skeleton animation

Blending skeleton animation

Pre-store several looping animation

Blend between animation for transition

```
three.js - Skeletal Animation Blending (model from realitymeltdown.com)  
camera orbit/zoom/pan with left/middle/right mouse button  
Note: crossfades are possible with blend weights being set to (1,0,0), (0,1,0) or (0,0,1)
```

Motion graphs

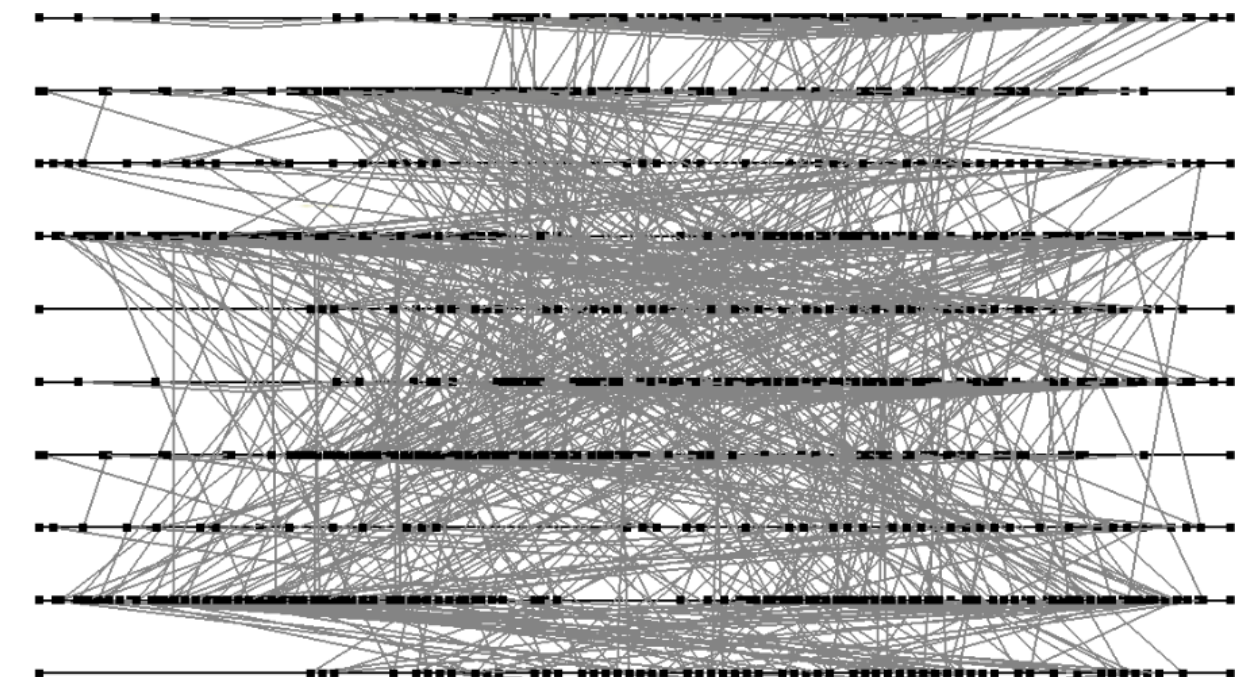
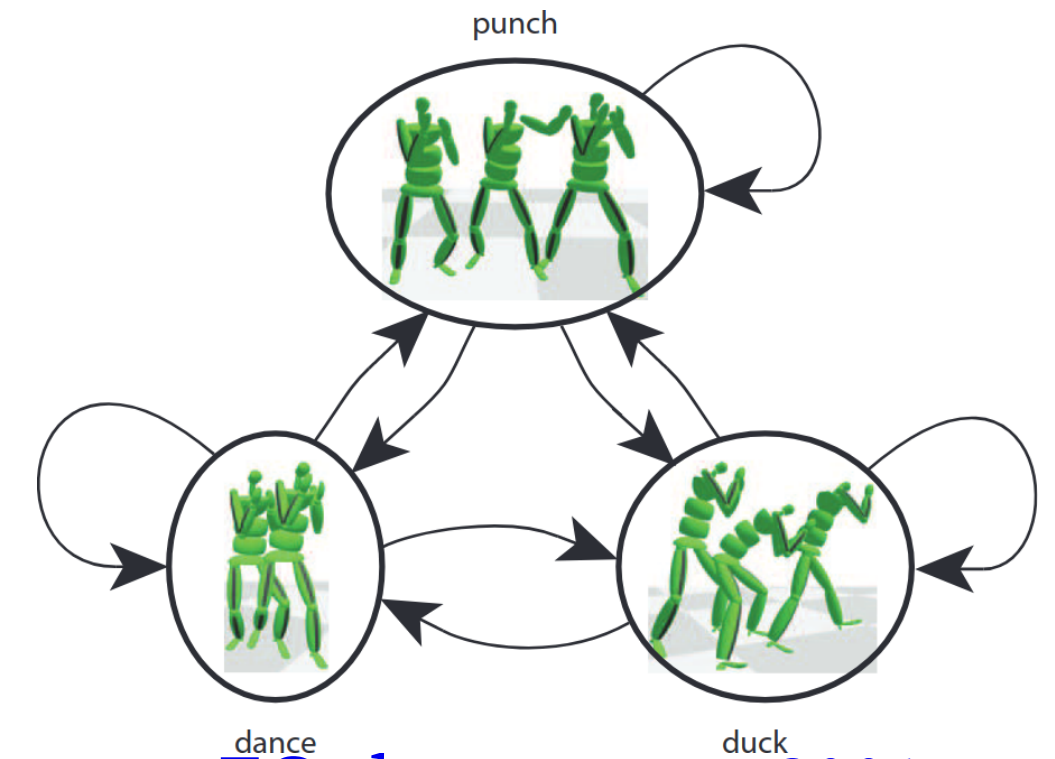
Also called Move Trees (highly used in video games)

- Stores multiple precomputed animation

Manually design, motion capture, etc

- Find optimal transitions between different motions

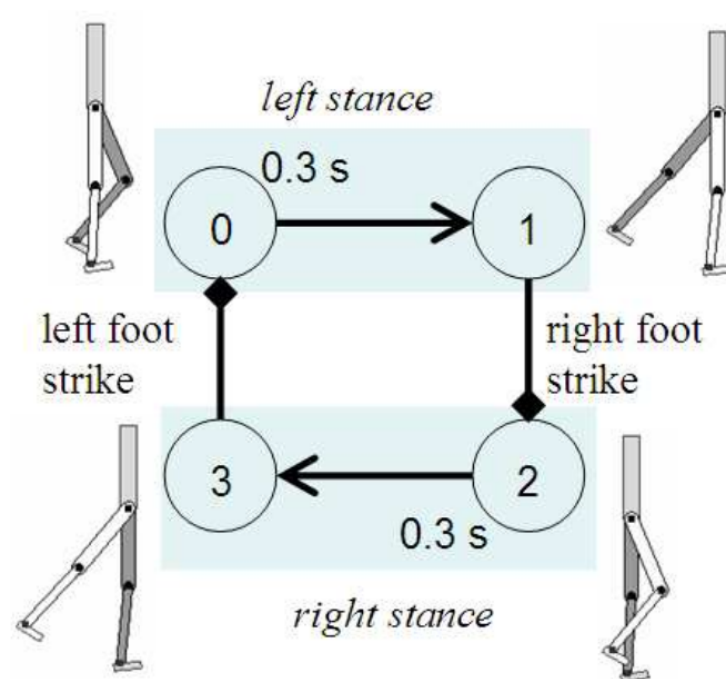
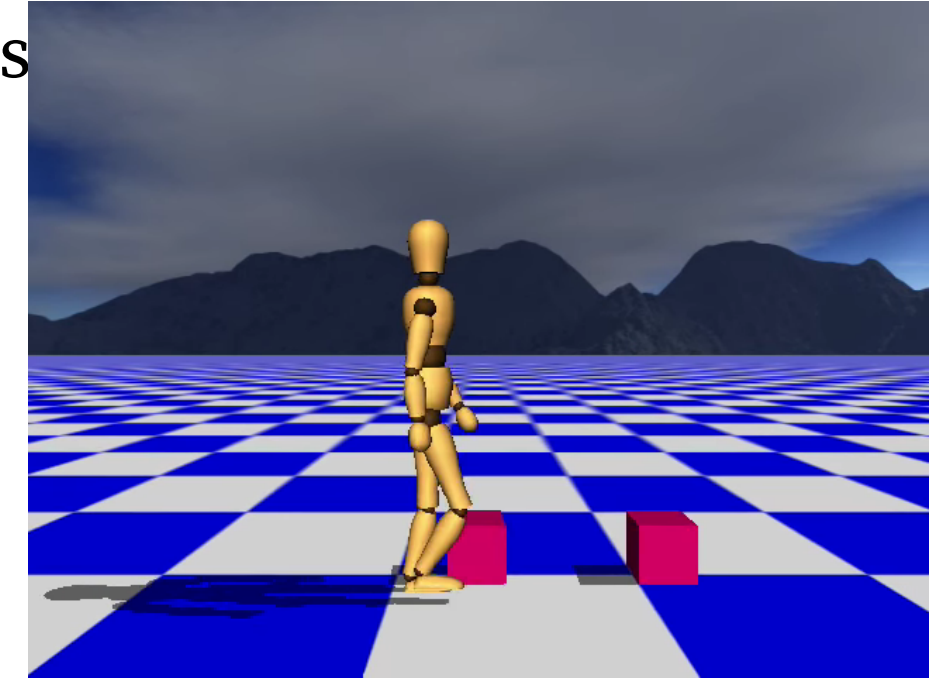
- Mizuguchi et al., Data driven motion transitions for interactive games, EG short paper, 2001
- Kovar et al., Motion Graphs, ACM SIGGRAPH 2002
- Heck and Gleicher, Parametric Motion Graphs, ACM SIGGRAPH 2007



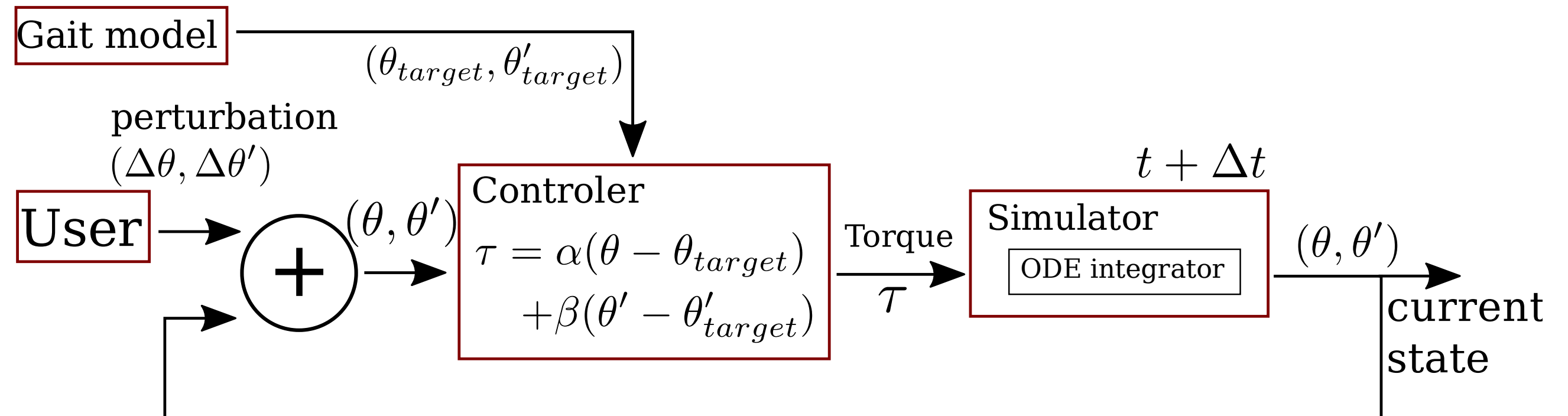
Controllers

Mix between predefined motions and physics → allow user perturbations

- 1 - Define target $(\theta_{target}(t), \theta'_{target}(t))$
pre-defined finite state machine (Gait model)
- 2 - Add user perturbation to the current state
- 3 - Use proportional derivative controllers to compute joint torque τ
- 4 - Integrate torque using rigid body simulator
- 5 - Iterate



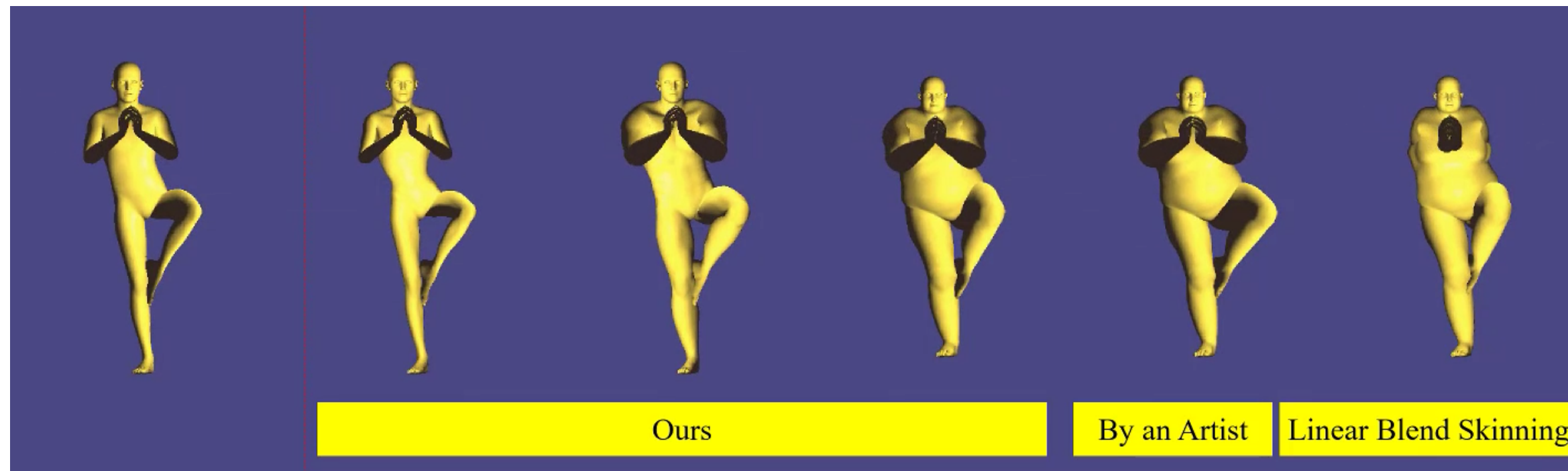
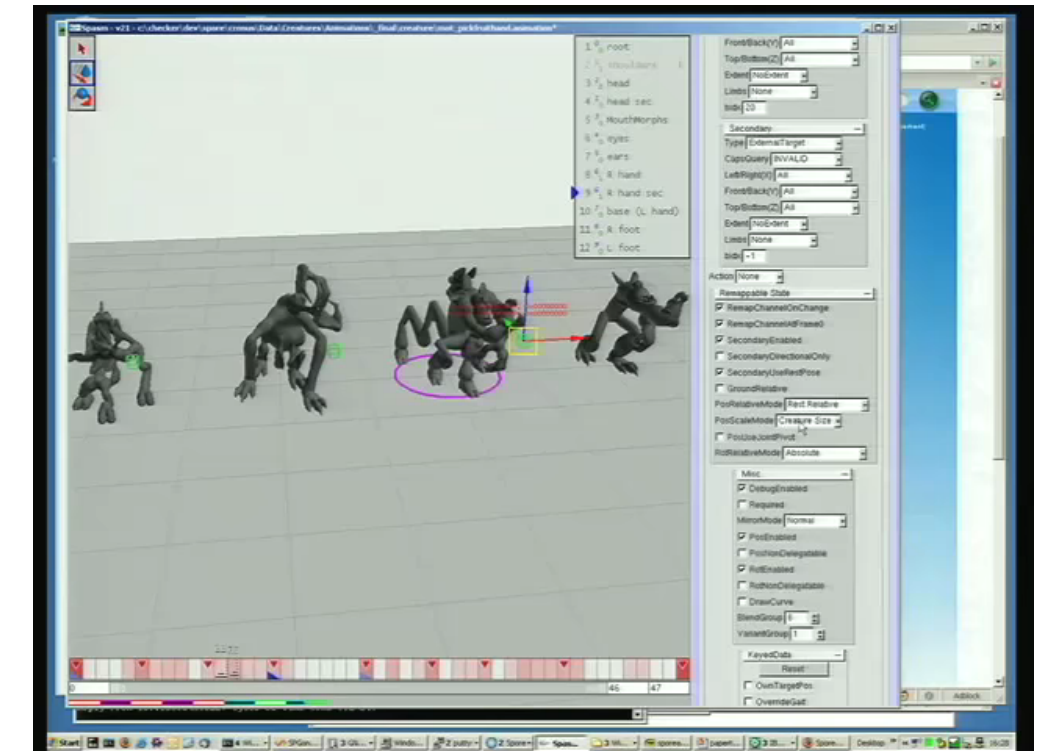
Gait model



M. Raibert and J. Hodgins. Animation of Dynamic Legged Locomotion, ACM SIGGRAPH 2001
K. Yin et al., SIMBICON: Simple Biped Locomotion Control, ACM SIGGRAPH 2007

Motion transfert

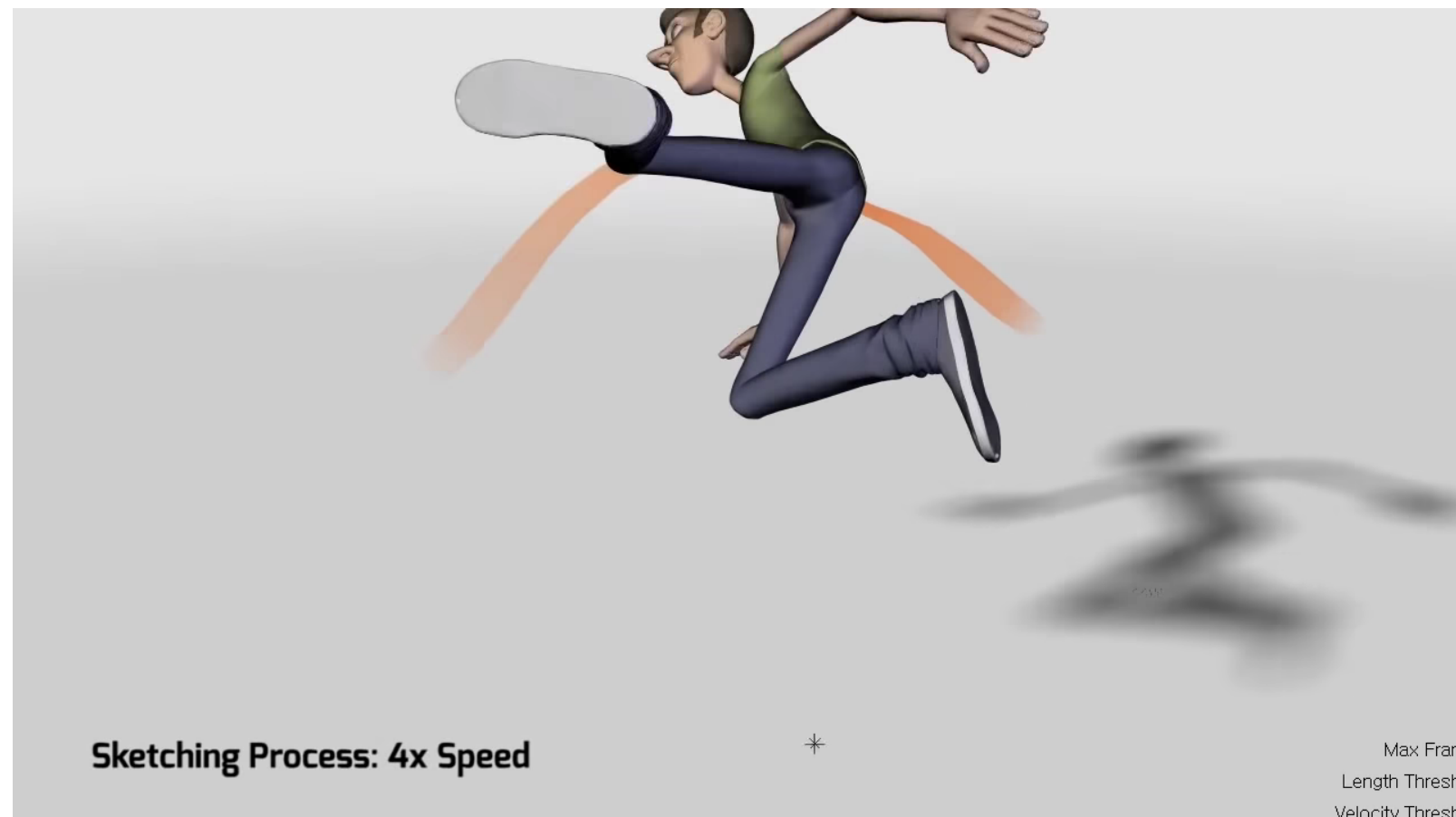
- Local coordinates mapping from skeletal motions
 - [C. Hecker et al., Real-time Motion Retargeting to Highly Varied User-Created Morphologies, SIGGRAPH 2008] (Spore)
- Including shape morphology
 - [Z. Liu et al., Surface based Motion Retargeting by Preserving Spatial Relationship, MIG 2018]



Animation design

Based on the *Line of Action*

- [Guay et al., The Line of Action: an Intuitive Interface for Expressive Character Posing, ACM SIGGRAPH Asia 2013]
- [Guay et al., Space-time sketching of character animation, ACM SIGGRAPH 2015]
- [Choi et al., SketchiMo: Sketch-Based Motion Editing for Articulated Characters, ACM SIGGRAPH 2016]



0:00 / 0:27

0:00 / 0:35

Automatic synthesis of skeletal animation

Seminal works

- [Evolving Virtual Creatures. Karl Sims. SIGGRAPH 1994]
- [Automated Learning of Muscle-Actuated Locomotion Through Control Abstraction. Radek Grzeszczuk and Demetri Terzopoulos. SIGGRAPH 1995]
- *Optimization toward objective function coupled with rigid bodies simulations*
- *Morphological variation from genetic algorithm*



Optimization of action space

Integrating on complex terrains

Offline simulation with random terrains

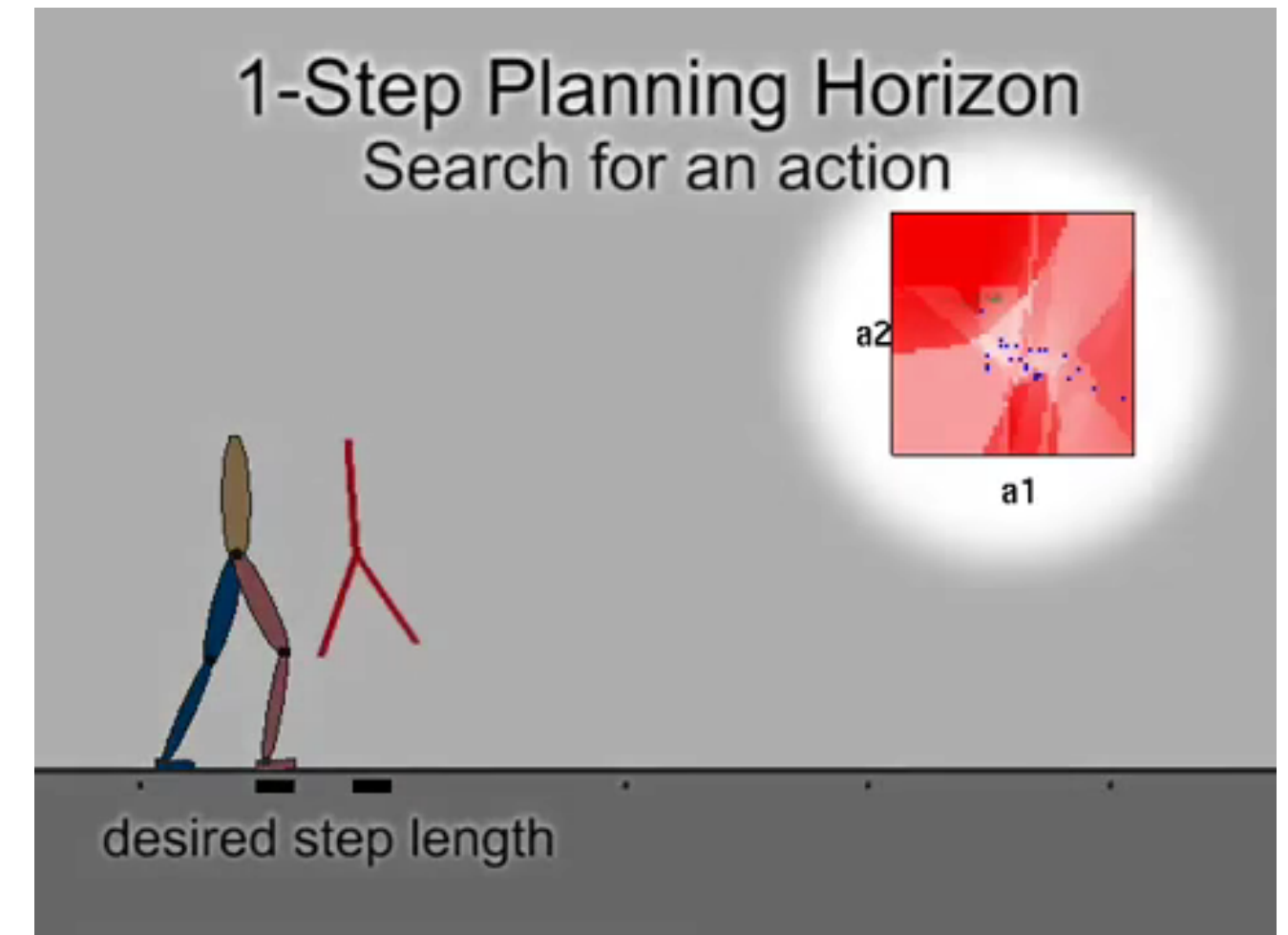
Learn "reduced action space"

Motion planning at run time

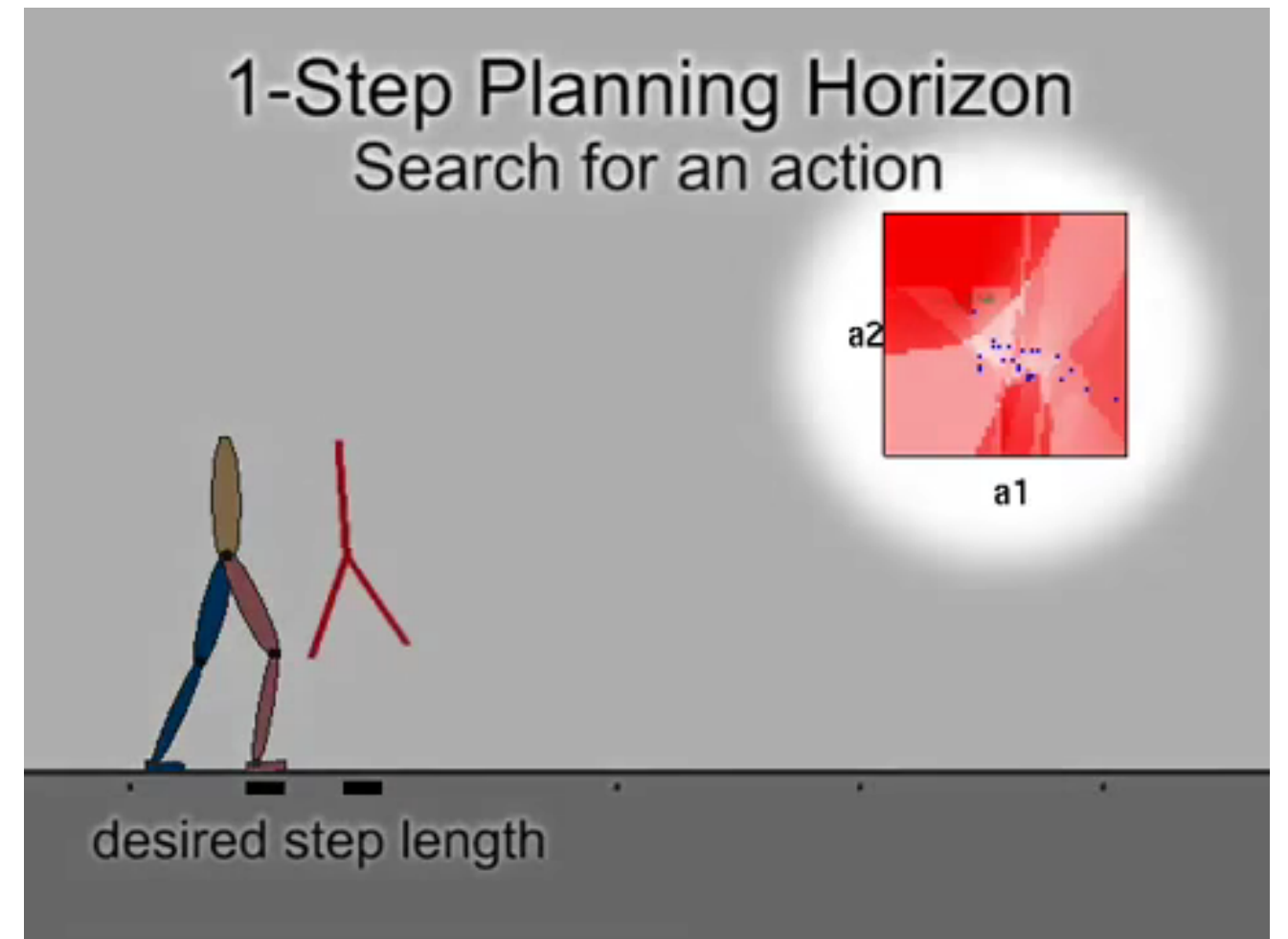
Synthesis of Constrained Walking Skills. S. Coros
et al. SIGGRAPH 2008

Generate as-diverse-as-possible variation of
possible motions

Diverse Motion Variations for Physics-based
Character Animation. S. Agrawal et al. SCA 2013.



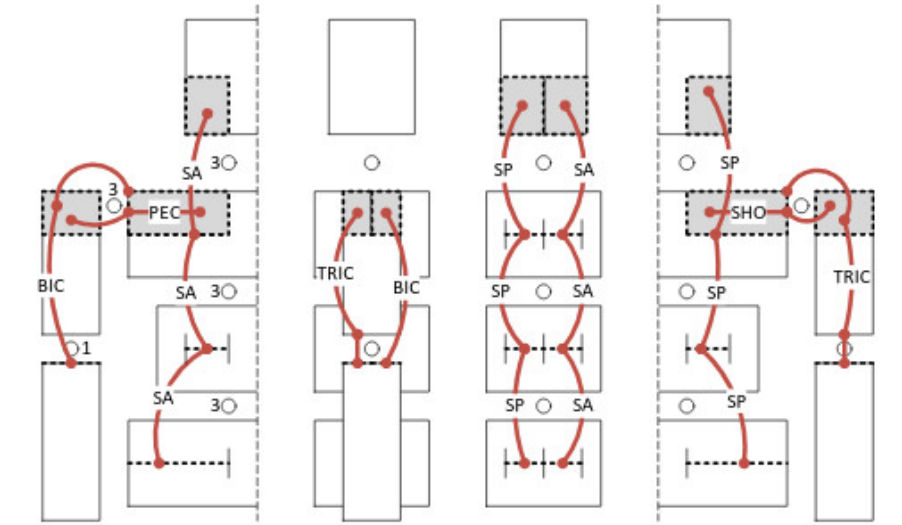
0:00 / 1:15



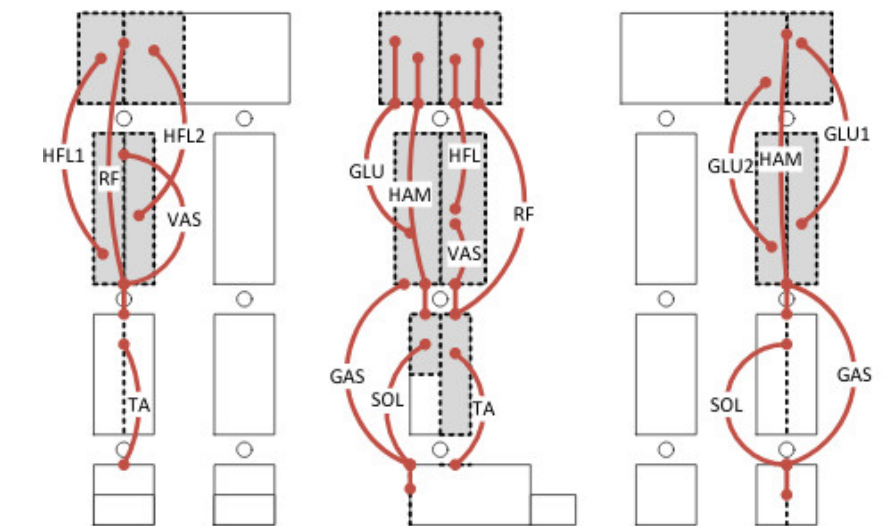
Optimizing muscle activation

- Take into account simple biomechanical model
- Optimize sequence of activation via reinforcement learning

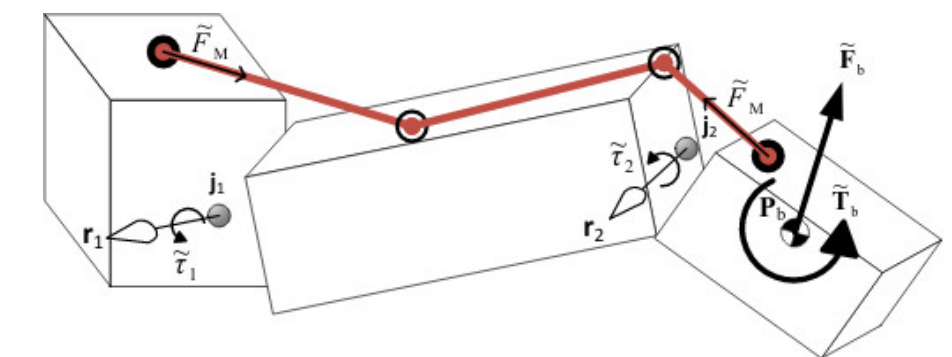
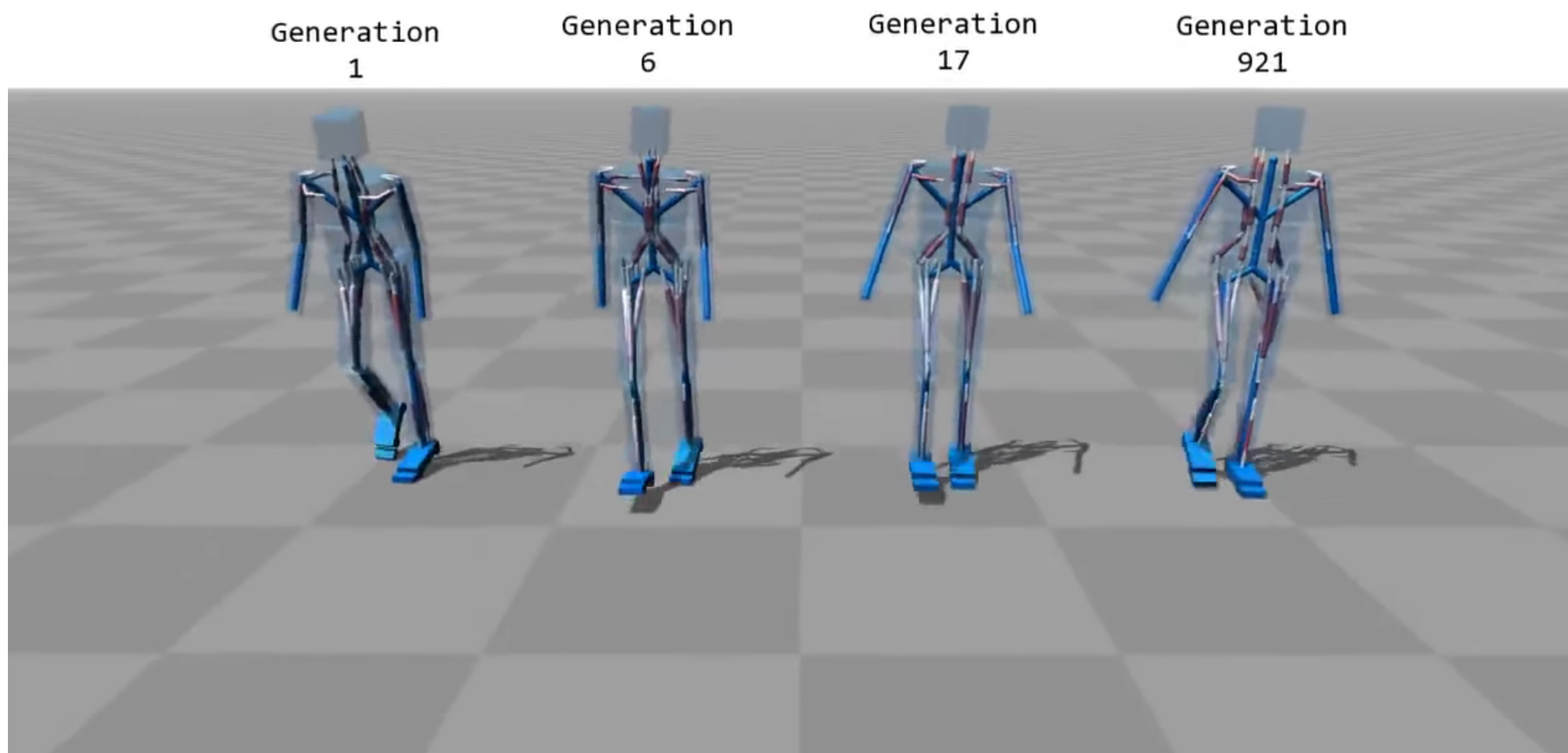
Flexible Muscle-Based Locomotion for Bipedal Creatures, T. Geijtenbeek
2013.



(a) Humanoid upper-body model: front, side arm, side body, and back.



(c) Humanoid lower-body model: front, side and back.

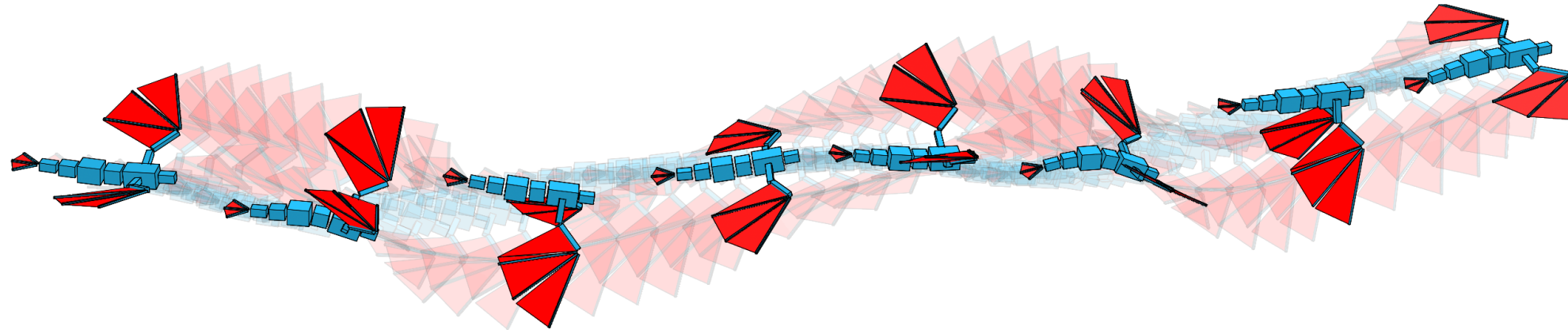


Use of deep learning

Deep reinforcement learning for complex optimization

Learn muscle activation

Won et al. How to Train Your Dragon: Example-Guided Control of Flapping Flight, ACM SIGGRAPH Asia 2017



Deep learning for real-time motion control

Learn phase of the motion cycle.

Use large data base of motion capture data

Holden et al., Phase-Functioned Neural Networks for Character



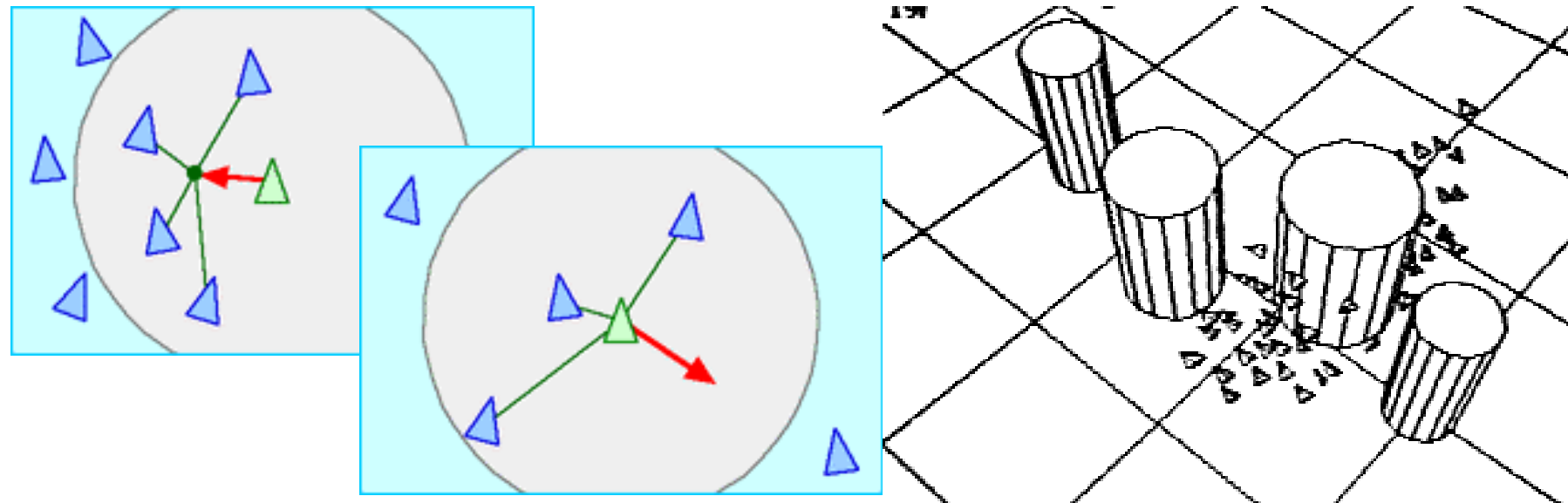
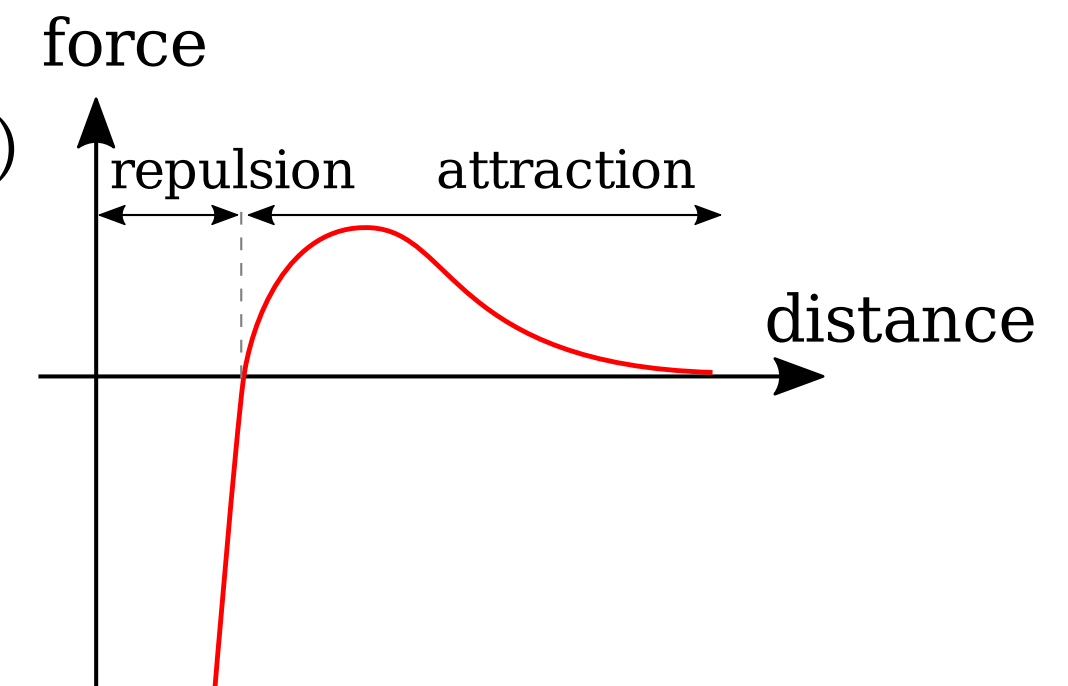
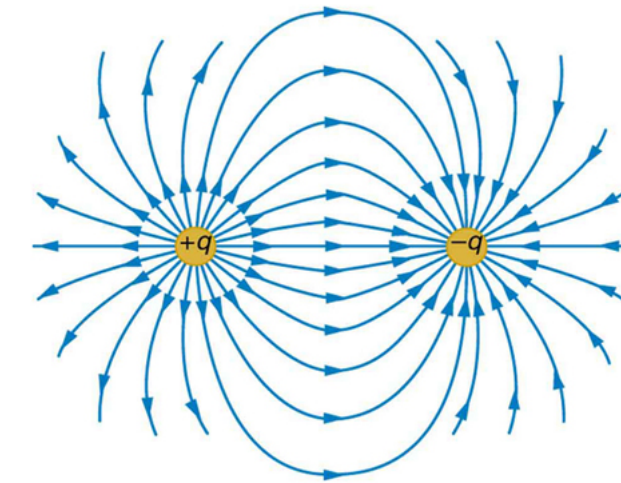
Animating crowds of characters

Interaction between particles

Interaction as **force field** (interact at distance)

Example of usage

- Models crowd of life-like characters at large scale
Inspired from physics particles forces (ex. Lennard-Jones potential)
Attraction at *long-range*
Repulsion at *short-range*
- First model: **Boids** Craig Reynolds 1987
- Extended later to human crowd modeling



Boids Model

Introduced by

- [Craig Reynolds. *Flocks, Herds, and Schools: A Distributed Behavioral Model*, SIGGRAPH 1987] [[link](#)]
- [Craig Reynolds. *Steering Behaviors For Autonomous Characters*. *Proceedings of Game Developers*, 1999] [[link](#)]

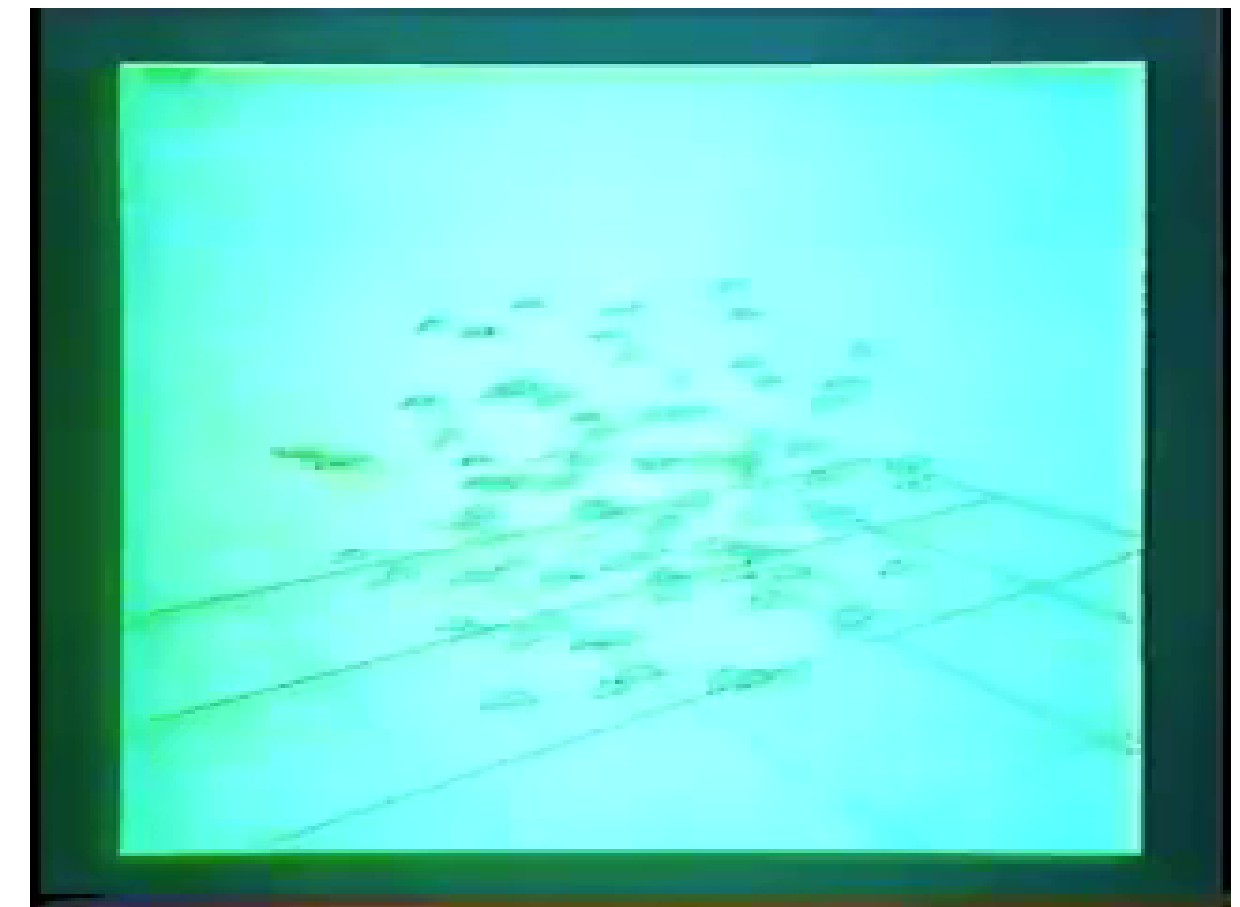
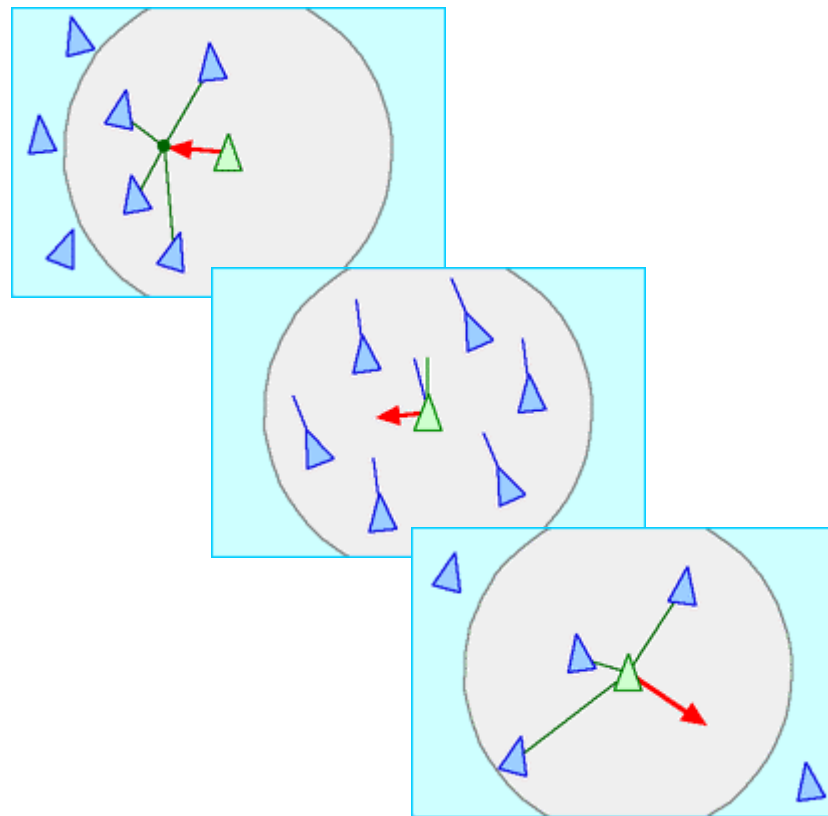
A boid is defined by its

- Position
- Speed
- Forces acting on it

Three basic local *steering behaviors* to model

- **Cohesion** between local particles
- **Alignment** between local particles
- **Separation** between too close particles

=> Leads to emerging global behaviors.



video in 1986 (C. Reynolds)

Original

Boids Model - Basic model.

- Set random initial position/speed to N particles.
- Set attraction/repulsion force depending on pairwise distances

$$F(p_i) = \sum_j f(\|p_j - p_i\|) \frac{p_j - p_i}{\|p_j - p_i\|}$$

Example

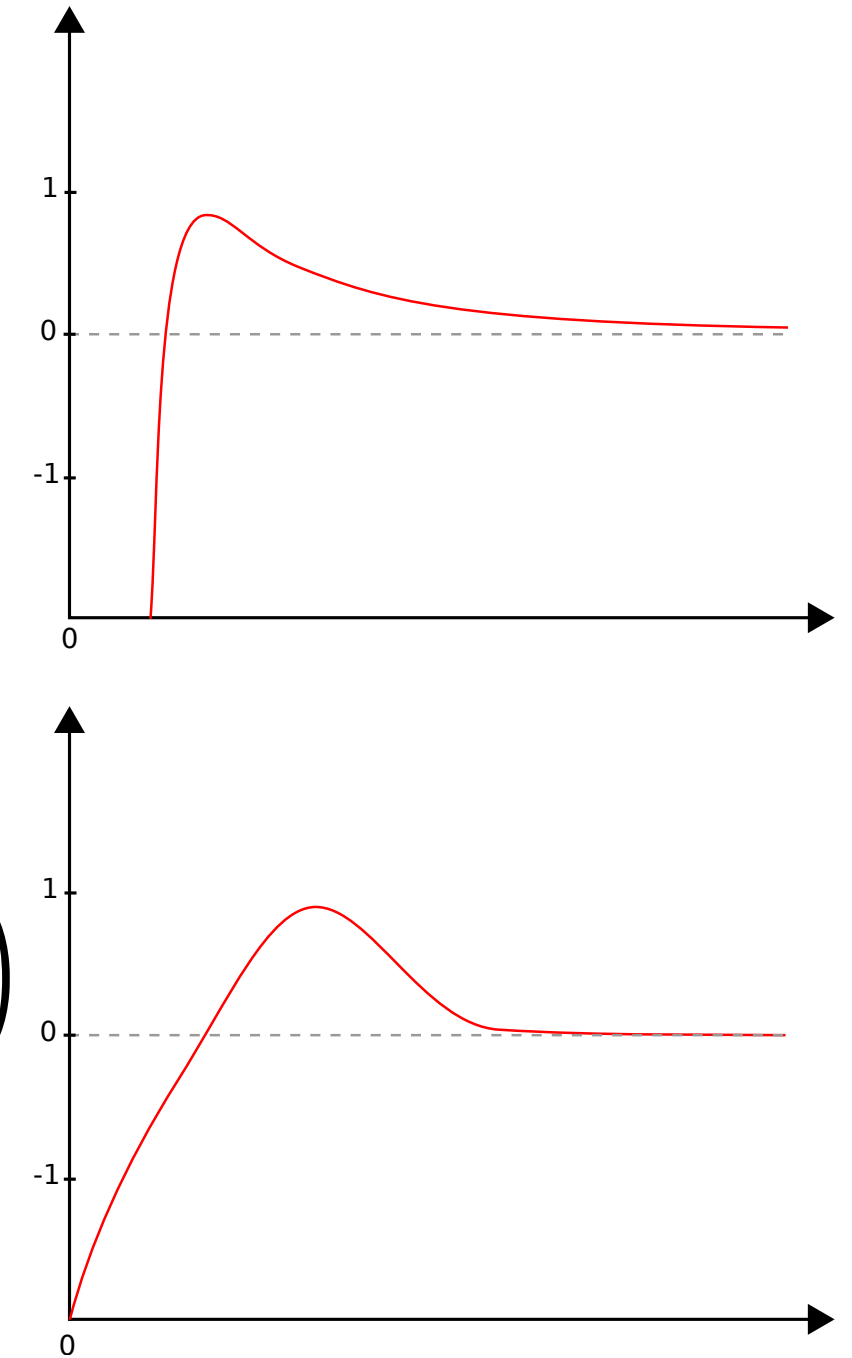
- Inverse of distance $f(x) = \frac{\alpha_1}{x^2} - \frac{\alpha_2}{x^4}$

- Exponential/Gaussian $f(x) = \alpha_1 \exp\left(-k \left(\frac{x - x_0}{x_0}\right)^2\right) - \alpha_2 \exp\left(-\frac{x}{x_0}\right)$

- Integrate position and speed through time

$$v^{t+\Delta t}(p_i) = v^t(p_i) + \Delta t F(p_i^t)$$

$$p^{t+\Delta t}(p_i) = p^t(p_i) + \Delta t v^{t+\Delta t}(p_i)$$



Boids Model - Complexity.

Trivial implementation:

```
struct particle { vec3 p, v, f; };

std::vector<particle> boids;

// Initialize N boids ...
// ...

// compute pairwise force
for(int i=0; i<N; ++i)
{
    for(int j=0; j<N; ++j)
    {
        if( i!=j )
        {
            const vec3& pi = boids[i].p;
            const vec3& pj = boids[j].p;

            boids[i].f += force( norm(pi,pj)) / (pi-pj)/norm(pi-pj);
        }
    }
}

// integration
for(int i=0; i<N; ++i)
{
    boids[i].v = boids[i].v + dt * boids[i].f;
    boids[i].p = boids[i].p + dt * boids[i].v;
}
```

- What is the complexity (wrt. N) of this algorithm ?
- Can you think of a way to be more efficient for large N ?

Boids Model - Usage and limitations

- Well adapted to flocks (birds, fishes - looking behavior)
- Display particles using 3D animated model

Additional behaviors

- Objective position/speed value
 - Constraints: Obstacle avoidance, limited velocity
 - Pursue and evade target/other particles - follow the leader, predators, etc.
-
- *Is collision between particles possible ?*
 - *Human displacement are mostly guided by vision, what key element is missing in the basic boids force-based model ?*

