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 w/r, $v^0$, $v^0$?

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

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

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

\[ p = 0.5 p_{b1} + 0.5 p_{b2} \]

\[ p = 0.5 T_1 (T_1^0)^{-1} p^0 + 0.5 T_2 (T_2^0)^{-1} p^0 \]

\[ p = (0.5 M_1 + 0.5 M_2) p^0 \]
Smooth skinning

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

Example at middle vertex position of a bending cylinder

\[ p = 0.5p_{b1} + 0.5p_{b2} \]
\[ p = 0.5 T_1 (T_1)^{-1} p^0 + 0.5 T_2 (T_2)^{-1} p^0 \]
\[ p = (0.5 M_1 + 0.5 M_2) p^0 \]

Can be generalized to arbitrary interpolation between two bones

\[ p = \alpha p_{b1} + (1 - \alpha) p_{b2} = (\alpha M_1 + (1 - \alpha) M_2) p^0 \]
\( \alpha \): skinning weights

Can be generalized to any number of bones

\[ p = \sum_{k=0}^{N-1} \alpha_k p_{b_k} = \left( \sum_{k=0}^{N-1} \alpha_k M_k \right) p^0, \quad \sum_k \alpha_k = 1 \]
Smooth skinning - Summary

\[ p = \left( \sum_{k=0}^{N-1} \alpha_k M_k \right) p^0 = \left( \sum_{k=0}^{N-1} \alpha_k T_k (T_k^0)^{-1} \right) p \]
\[ \forall k, \omega_k \in [0, 1], \text{and } \sum_{k=0}^{N-1} \alpha_k = 1 \]

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


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), gTF (JSON)

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

collada.dae

.options = {
  "inverse" : -1,
  "uuid" : "4181C04A-B75A-3FDC-9A89-B9A1D2BCE0AF",
  "type" : "Animation",
  "name" : "MarineCv2_color",
  "url" : "MarineCv2_color.jpg",
  "image" : {
    "type" : "Image",
    "name" : "MarineCv2_color.jpg"
  }
};
Linear Blend Skinning - Limitations

- Non-trivial rigging settings
- Artifacts for large rotations: 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


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

IK: Numerical methods

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

Consider small step size \( p \rightarrow p + \Delta p \)

\[
\Delta p \simeq \left( \frac{\partial f}{\partial \theta} \right)_J \Delta \theta
\]

\( J \) - Jacobian matrix.

\( \rightarrow \) 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 $j^N \rightarrow j^{i-1} \rightarrow \cdots \rightarrow j^1$ 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

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.*

Inverse Kinematics

Example
Synthesizing and controlling skeleton animation
Blending skeleton animation

Pre-store several looping animation
Blend between animation for transition

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
Controlers

Mix between predefined motions and physics → allow user perturbations

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

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]
Automatic synthesis of skeletal animation

Seminal works
- [Evolving Virtual Creatures. Karl Sims. SIGGRAPH 1994]
- 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


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

1-Step Planning Horizon

Search for an action

desired step length
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.
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.
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

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

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

```cpp
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?