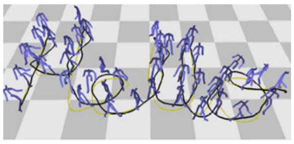
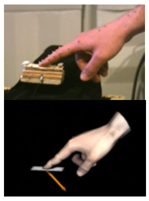
Motion capture data processing
- Motion Editing/Retargeting
- Motion Graph
- Inverse kinematic



Alexandre Meyer
M2pro

Motion capture data processing

From Data Capture to motion



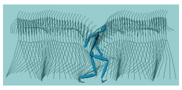
Overview: Motion data processing

In this course

- Motion warping/editing
- Motion graph/planning
- Motion retargeting

Not in this course

- Motion segmentation
- Motion compression
- Physically based animation
- Etc.



How do skeletons differ?

- Topology
  - number of bones
  - Connectivity of bones
- Joint Types
  - Bone lengths
  - Anatomical / skin relations
- Is spine in middle of body, or up the back?

Subtle Skeletal Differences

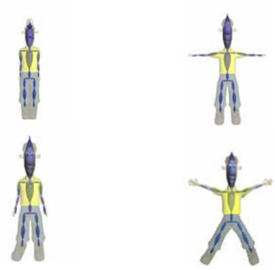
- Rest Poses (design of a skeleton)
  - Zero Pose / Base Pose
  - Dress or Binding pose
  - Frankenstein Pose
  - Da Vinci Pose
  - Rest Pose (real pose of actor)
- Need to figure out how to get between these

Subtle Skeletal Differences

- Same angles lead to different animation is rest pose is different

Rest pose

Animation with similar angles



Motion Editing

Recap on motion!

- Motion is a function of time
  - Given time, provide a pose
  - Often represented as samples
- Sparse samples + interpolation
  - Dense samples (at frames)
  - How to manipulate sets of samples?

The General Challenge

What you get is not what you want!

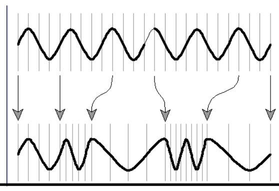
- You get observations of the performance
  - A specific performer
  - A real human
  - Doing whatever they did
  - With the noise and “realism” of real sensors
- Want something else
  - But need to preserve original
  - But we don’t know what to preserve
  - Can’t characterize motion well enough

Three Problems

- Where does X live in the data?
  - Where X ∈ {style, personality, emotion, …}
  - The things to keep or add
- Small artifacts can destroy realism
  - Eye is sensitive to certain details
- How to *specify what you want*

Manipulating motion

- Manipulate time: Motion slower or faster
  - m(t) = m0(f(t))
  - f : R - > R “time warp”
- Time scaling
  - f(t) = k t
- Time shifting
  - f(t) = t + k
- Time warping
  - Interpolate a table
  - Align events



Manipulating motion

- Manipulate value
  - m(t) = f(m0(t))
  - f : Rn- > Rn
- Scale?
  - For instance each angles x 2 → Exagerate motion
- Shift?
- Convole (linear filter)
- “Add” to another motion
  - $m(t) = m_0(t) + a(t)$

Motion Blending

- "Add" two motions together
  - Really interpolate
- m(t) = a m0(t) + (1-a) m1(t)
  - Note: this is a per-frame operation
- It works only if poses are similar!!!
- Very useful!
  - Often get small pieces of motion
  - Need to connect
  - Easy if motions are similar

Noise Removal: Signal Processing

- Noise comes from errors in process
  - Sensor errors
  - Fitting errors
  - Bad movements
- Nose is "data" that we don't want

Where's the Noise?

- Sometimes identification is easy:
  - Clearly wrong (foot through floor)
  - Marked wrong (missing data - gaps)
- More often, need to guess
  - Might be a subtle twitch…
  - Might be person shaking…
  - Might be sensor errors…

➔simply apply a filter ?

Important Intuition

- High Frequencies are Important!
- Always significant
  - Impact
  - Rapid, sudden movement
  - …

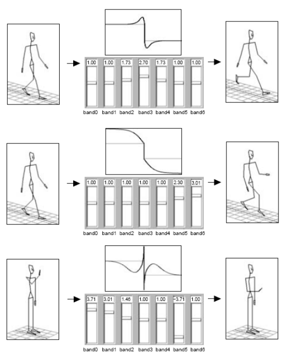
Signal processing [Unuma95]

- Fourier series
  - Coefficient motion parameters (emotion, gait)

Exaggerate jump by scaling low frequency

Motion Signal Processing [Bruderlin95]

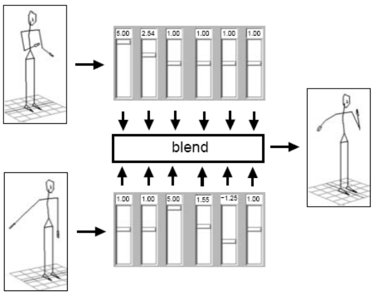
[Motion Signal Processing, Siggraph95 Bruderlin & Williams]



Motion Signal Processing [Bruderlin95]

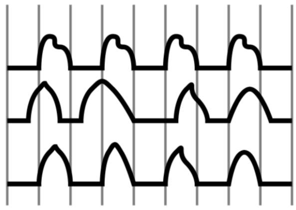
[Motion Signal Processing, Siggraph95 Bruderlin & Williams]

Motion Blending



Motion Blending

- "Add" two motions together
  - Works only if motion are synchronized



Motion Graph

Kovar, Gleicher, Pighin '02

Idea: Put Clips Together

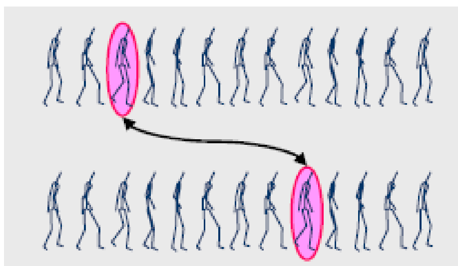
- New motions from pieces of old ones!
- Good news:
  - Keeps the qualities of the original (with care)
  - Can create long and novel "streams" (keep putting clips together)
- Challenges:
  - How to decide what clips to connect?
  - How to connect clips?

Connecting Clips: Transition Generation

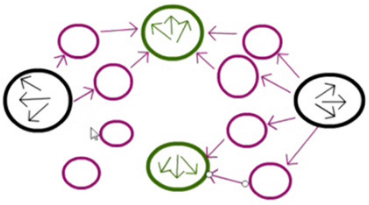
- Transitions between motions can be hard
- Motion interpolation works *sometimes*
  - Blends between aligned motions
  - Cleanup footskate artifacts
- Just need to know when is "sometime"
  - Need a distance between pose

Idea: Motion Graph

Find Matching States in Motions



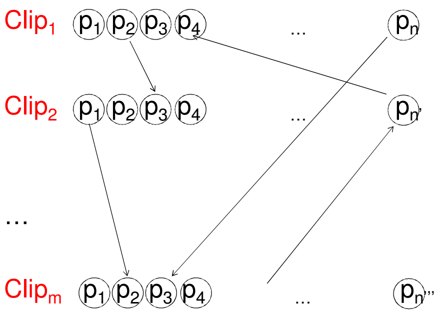
A simple motion graph



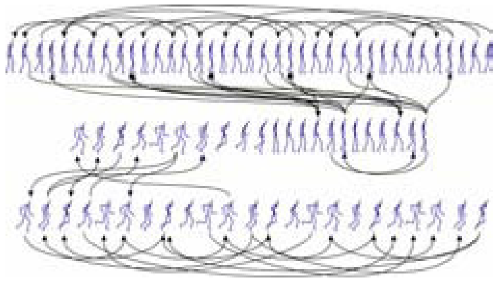
What are motion graphs?

- Directed graph representing a roadmap of motion data for a character
  - Vertex represent a pose in a motion clip
    - Vertex=(motion clip name, pose number)
  - Edges are pose transitions

A simple motion graph



A simple motion graph



A simple motion graph

- Motion Blend & Motion Graph
  - Motion Graph more examples

Building motion graphs

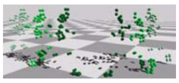
- Identify transition candidates
- Select transition points
- Eliminate problematic edges

Identify transition candidates: pose distance

- For each pose A of clip C_j, calculate its distance to each other pose B of all other clip by basically measuring volume displacement

1) Initial frames we want to compare

2) Extract windows: frame before and after



3) Convert to point clouds

4) Align point clouds and sum squared distances

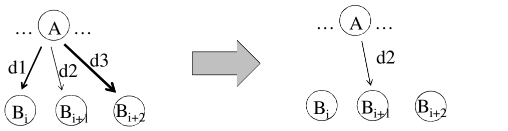
Identify transition candidates: pose distance

- For each frame/pose A, calculate its distance to each other frame/pose B by basically measuring volume displacement
- Use a weighted point cloud formed over a window of k frames ahead of A and behind B, ideally from the character mesh
- Calculate the minimal weighted sum of squared distances between corresponding points, given that a rigid 2D transformation may be applied to the second point cloud

$$\min_{\theta, x_0, z_0} \sum_i w_i \left\| \mathbf{p}_i - \mathbf{T}_{\theta, \mathbf{x}_0, \mathbf{z}_0} \mathbf{p}'_i \right\|^2$$

Select transition/edge

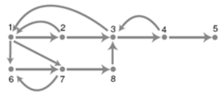
- The previous step gave us all the local minima of the distance function for each pair of points
- Now we simply define a threshold and cut transition candidates with errors above it
- May be done with or without intervention
- Threshold level depends on type of motion – eg. walking vs. ballet



We define transition only between pose with significant similitude

Eliminate problematic edges

- We want to get rid of:
  - Dead ends – not part of a cycle
  - Sinks – part of one or more cycles but only able to reach a small fraction of the nodes
  - Logical discontinuities – eg. boxing motion forced to transition into ballet motion
- Goal is to be able to generate arbitrarily long streams of motion of the same type



Using a motion graph

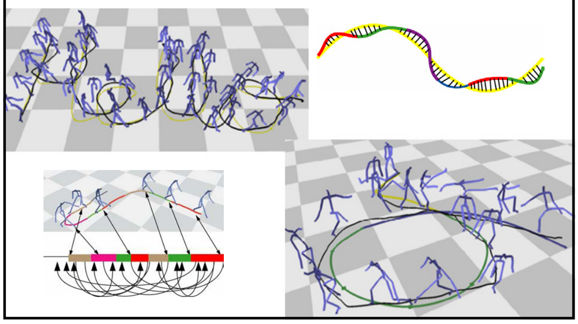
- Any walk on the graph is a valid motion
  - Generate walks to meet goals
  - Random walks (screen savers)
  - Search to meet constraints
- Other Motion Graph- like projects elsewhere
  - Differ in details, and attention to detail

Transitions

- When need to make the transition between frames A_i and B_j blend Ai through A_{i+k-1} with Bj through B_{j-k+1}
  - Align frames with appropriate rigid 2D transformation
  - Use linear interpolation to blend root positions
  - Use spherical linear interpolation to blend joint rotations

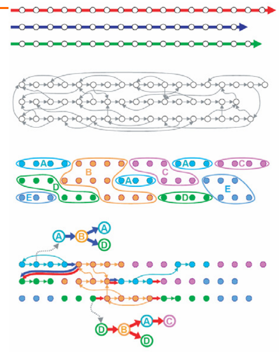
Results

+ video



Clustering a motion graph

- Clustering the graph
  - For a big graph
  - →Build a meta-graph
  - Improve the exploration



M2 38

Edition with constraints
Inverse Kinematics
Motion Retargeting

39

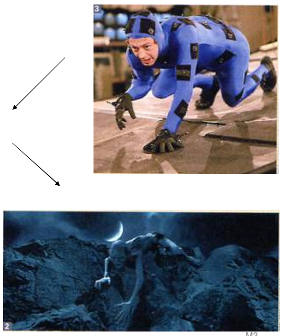
VOIR LE COURS CONSACREE ENTIEREMENT A LA CINEMATIQUE INVERSE

VOIR LE COURS CONSACREE ENTIEREMENT A LA CINEMATIQUE INVERSE

VOIR LE COURS CONSACREE ENTIEREMENT A LA CINEMATIQUE INVERSE

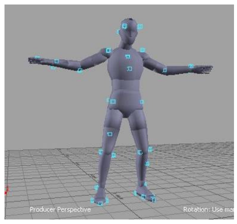
Retargeting

- capture motion on performer
  - **positions** of markers are recorded
- retarget motion on a virtual character
  - motion is usually applied to a skeleton
  - a skeleton is hierarchical
    - linked joints
  - need **rotation** data!
- need to convert positions to rotations



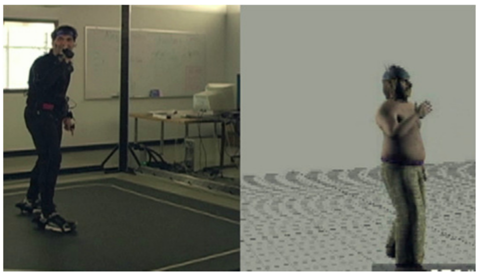
performer→ actor → character

- the **actor** is used to convert marker positions to rotational data
  - markers are handles on the actor
  - actor should have similar proportions as the **performer**
- joint rotations of the actor are applied to the character
- there are still issues with proportions



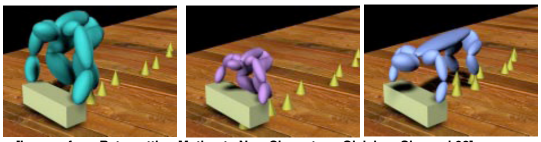
Alias Motionbuilder: actor and markers

Retargeting problems: hand problem



Problem of Hand or foot position!

- Often hand or foot positions do not match



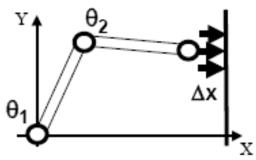
[Images from Retargetting Motion to New Characters, Gleicher, Siggraph98]

- Need to find a position with hands on the box and feet in concordance with skeleton morphology

➔Quick overview of inverse kinematic

Inverse Kinematics

- Inverse Kinematics
  - Given effectors positions, find a posture(=angles)
- Non-linear problem (position vs. angles)
  - Possibility of no or multiple solutions



Forward Kinematics

- We will use the vector:

$$\Phi = [\varphi_1 \quad \varphi_2 \quad \ldots \quad \varphi_M]$$

to represent the array of M joint DOF values
- We will also use the vector:

$$\mathbf{e} = [e_1 \quad e_2 \quad \ldots \quad e_N]$$

to represent an array of N DOFs that describe the end effector in world space. For example, if our end effector is a full joint with orientation, **e** would contain 6 DOFs: 3 translations and 3 rotations. If we were only concerned with the end effector position, **e** would just contain the 3 translations.

Forward Kinematics

- The forward kinematic function f() computes the world space end effector DOFs from the joint DOFs:
  - Forward kinematic is often easy to compute

$$\mathbf{e} = f(\mathbf{\Phi})$$

Inverse Kinematics

- The goal of inverse kinematics is to compute the vector of joint DOFs that will cause the end effector to reach some desired goal state
- In other words, it is the inverse of the forward kinematics problem
  - $f^{-1}()$ usually isn't easy to compute

$$\mathbf{\Phi} = f^{-1}(\mathbf{e})$$

Inverse Kinematics

Inverse Kinematics: many approaches

- Analytic method [IKAN, Badler]
  - Geometric based, fast
  - Ok only for few joints
- Numeric solution
  - Iterative process
  - Expensive
  - Flexible (constraints)
  - Minimization problem

Iterative Inverse Kinematics

(Gradient Descent)

Iterative IK

- Initial Position
  - Initial Angles $\begin{pmatrix} \varphi_1^0 & \varphi_2^0 & \dots & \varphi_n^0 \end{pmatrix}$

➔ $\mathbf{e}^0 = f\left(\varphi_1^0, \dots, \varphi_n^0\right)$

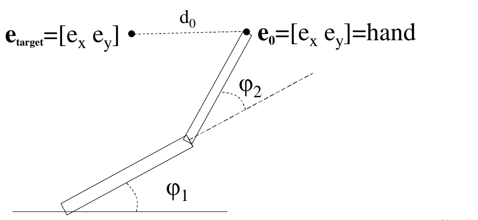
- The partial derivative of f is defined as:

$$\frac{df}{d\varphi_1} = \lim_{\Delta\varphi_1 \to 0} \frac{\Delta f}{\Delta \varphi_1} = \lim_{\Delta\varphi_1 \to 0} \frac{f(\varphi_1 + \Delta\varphi_1, \dots, \varphi_n) - f(\varphi_1, \dots, \varphi_n)}{\Delta\varphi_1}$$

$$\approx \frac{f(\varphi_1 + \Delta\varphi_1, \dots, \varphi_n) - f(\varphi_1, \dots, \varphi_n)}{\Delta\varphi_1}$$

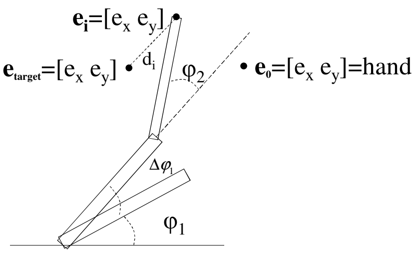
Iterative IK

- Let's say we have a simple 2D robot arm with two 1-DOF rotational joints
- And a target hand position e_{target}
- d = distance between the hand and the hand target



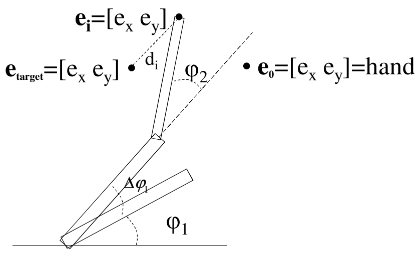
Iterative IK

1) Move a bit φ_1 → φ_1 + delta
→chose delta where d decrease
2) Do the same for φ_2
3) Iterate until d=0



Iterative IK

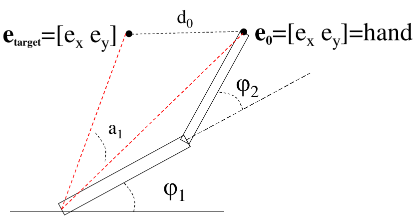
- How to find the good delta?



Cyclic Coordinate Descent

- Iterate on each joint
  - Compute and apply the analytical rotation

    $a_1\ a_2 \ldots a_i$ with a=angle between hand and target



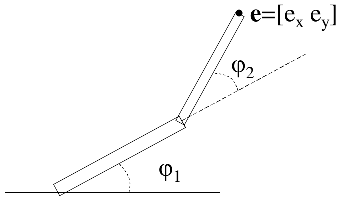
Cyclic Coordinate Descent

Repeat
    ForEach joint i Do
        a_i = compute angle between hand and target
        Rotate joint i
    EndForEach
Until d<0.01 (meaning while **e** is too far from $\mathbf{e}_{target}$)

Jacobian Inverse Kinematics

Jacobians

- Let's say we have a simple 2D robot arm with two 1-DOF rotational joints:



Jacobians

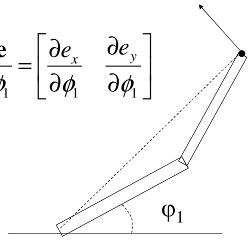
- The Jacobian matrix J(**e**,**Φ**) shows how each component of **e** varies with respect to each joint angle

$$J(\mathbf{e},\mathbf{\Phi})=\begin{bmatrix}\frac{\partial e_x}{\partial \phi_1} & \frac{\partial e_x}{\partial \phi_2}\\ \frac{\partial e_y}{\partial \phi_1} & \frac{\partial e_y}{\partial \phi_2}\end{bmatrix}$$

Jacobians

- Consider what would happen if we increased φ_1 by a small amount. What would happen to **e** ?

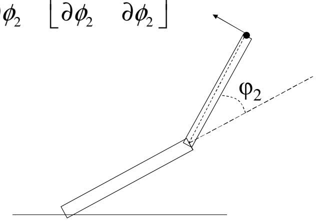
$$\frac{\partial \mathbf{e}}{\partial \phi_1}=\begin{bmatrix}\frac{\partial e_x}{\partial \phi_1} & \frac{\partial e_y}{\partial \phi_1}\end{bmatrix}$$



Jacobians

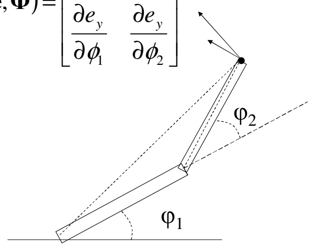
- What if we increased φ_2 by a small amount?

$$\frac{\partial \mathbf{e}}{\partial \phi_2}=\begin{bmatrix}\frac{\partial e_x}{\partial \phi_2} & \frac{\partial e_y}{\partial \phi_2}\end{bmatrix}$$



Jacobian for a 2D Robot Arm

$$J(\mathbf{e},\mathbf{\Phi})=\begin{bmatrix}\frac{\partial e_x}{\partial \phi_1} & \frac{\partial e_x}{\partial \phi_2}\\ \frac{\partial e_y}{\partial \phi_1} & \frac{\partial e_y}{\partial \phi_2}\end{bmatrix}$$



Jacobian Matrices

- Just as a scalar derivative df/dx of a function f(x) can vary over the domain of possible values for x, the Jacobian matrix J(**e**,**Φ**) varies over the domain of all possible poses for **Φ**
- For any given joint pose vector **Φ**, we can explicitly compute the individual components of the Jacobian matrix

Jacobian as a Vector Derivative

- Once again, sometimes it helps to think of:

$$J(\mathbf{e},\mathbf{\Phi})=\frac{d\mathbf{e}}{d\mathbf{\Phi}}$$

because J(**e**,**Φ**) contains all the information we need to know about how to relate changes in any component of **Φ** to changes in any component of **e**

Incremental Change in Pose

- Lets say we have a vector **ΔΦ** that represents a small change in joint DOF values
- We can approximate what the resulting change in **e** would be:

$$\Delta\mathbf{e} \approx \frac{d\mathbf{e}}{d\mathbf{\Phi}} \cdot \Delta\mathbf{\Phi} = J(\mathbf{e},\mathbf{\Phi}) \cdot \Delta\mathbf{\Phi} = \mathbf{J} \cdot \Delta\mathbf{\Phi}$$

Incremental Change in Effector

- What if we wanted to move the end effector by a small amount **Δe.** What small change **ΔΦ** will achieve this?

$$\Delta\mathbf{e} \approx \mathbf{J} \cdot \Delta\mathbf{\Phi}$$

$$so:$$

$$\Delta\mathbf{\Phi} \approx \mathbf{J}^{-1} \cdot \Delta\mathbf{e}$$

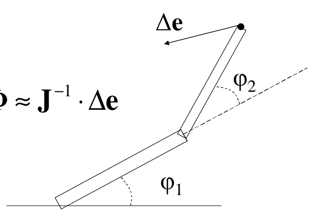
- Notice : J^{-1} maybe approximate by J^T

See this survey : http://math.ucsd.edu/~sbuss/ResearchWeb/ikmethods/iksurvey.pdf

Incremental Change in **e**

- Given some desired incremental change in end effector configuration **Δe**, we can compute an appropriate incremental change in joint DOFs **ΔΦ**

$$\Delta\mathbf{\Phi} \approx \mathbf{J}^{-1} \cdot \Delta\mathbf{e}$$



Incremental Changes

- Remember that forward kinematics is a nonlinear function (as it involves sin's and cos's of the input variables)
- This implies that we can only use the Jacobian as an approximation that is valid near the current configuration
- Therefore, we must repeat the process of computing a Jacobian and then taking a small step towards the goal until we get to where we want to be

End Effector Goals

- If **Φ** represents the current set of joint DOFs and **e** represents the current end effector DOFs, we will use $\mathbf{e}_{target}$ to represent the goal DOFs that we want the end effector to reach

Choosing Δe

- We want to choose a value for **Δe** that will move **e** closer to $\mathbf{e}_{target}$. A reasonable place to start is with

$$\Delta\mathbf{e} = \mathbf{e}_{target} - \mathbf{e}$$

- We would hope then, that the corresponding value of **ΔΦ** would bring the end effector exactly to the goal
- Unfortunately, the nonlinearity prevents this from happening, but it should get us closer
- Also, for safety, we will take smaller steps:

$$\Delta\mathbf{e} = \beta(\mathbf{e}_{target} - \mathbf{e})$$

where $0 \le \beta \le 1$

Basic Jacobian IK Technique

while ($\mathbf{e}$ is too far from $\mathbf{g}$) {

Compute $J(\mathbf{e},\mathbf{\Phi})$ for the current pose $\mathbf{\Phi}$

Compute J^{-1} // invert the Jacobian matrix

$\Delta\mathbf{e} = \beta(\mathbf{e}_{\mathbf{target}} - \mathbf{e})$ // pick approximate step to take

$\Delta\mathbf{\Phi} = J^{-1} \cdot \Delta\mathbf{e}$ // compute change in joint DOFs

$\mathbf{\Phi} = \mathbf{\Phi} + \Delta\mathbf{\Phi}$ // apply change to DOFs

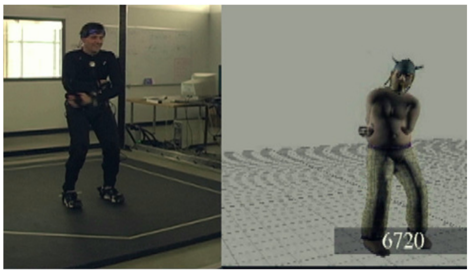
Compute new $\mathbf{e}$ vector // apply forward // kinematics to see // where we ended up

}

+ DEMO BLENDER

Back to Retargeting problems

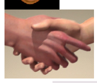
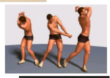
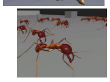
- IK may help for hand and foot but …
- many other problems than hand or foot!



Papers classification

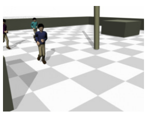
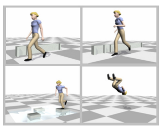
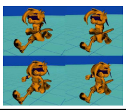
Papers Overview

Data-driven approach for

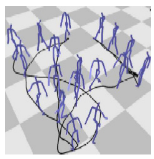
- Facial animation 
- Hand animation 
- Skin deformation 
- Animal animation 

Papers Overview

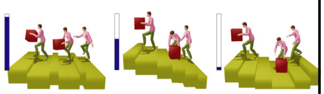
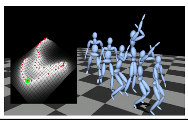
Combine mocap data with other techniques

- Motion planning 
- Physically based animation 
- Key-framing 

Papers Overview



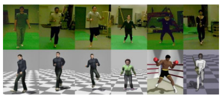
Data-driven motion synthesis

- Motion graphs/patches
- Motion interpolation 
- Statistical motion synthesis 

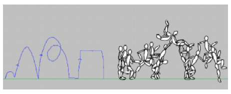
Papers Overview

Animation control

- Online animation control

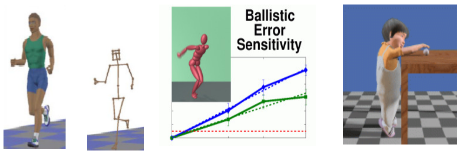


- Offline Animation control



Papers Overview

Motion perception



Iterative IK

- How to find the good delta?
  → use the gradient of f (minus of)→Gradient descent algo

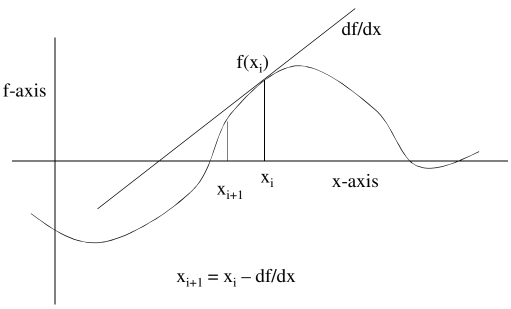
$$\nabla f = \begin{pmatrix} \dfrac{df}{d\varphi_1} & \dfrac{df}{d\varphi_2} & \cdots & \dfrac{df}{d\varphi_n} \end{pmatrix}$$

with the partial derivative of f

$$\frac{df}{d\varphi_1} = \lim_{\Delta\varphi_1 \to 0} \frac{\Delta f}{\Delta \varphi_1} = \lim_{\Delta\varphi_1 \to 0} \frac{f(\varphi_1 + \Delta\varphi_1, \ldots, \varphi_n) - f(\varphi_1, \ldots, \varphi_n)}{\Delta \varphi_1}$$

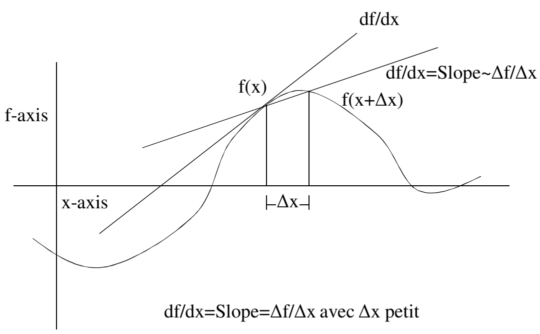
$$\approx \frac{f(\varphi_1 + \Delta\varphi_1, \ldots, \varphi_n) - f(\varphi_1, \ldots, \varphi_n)}{\Delta \varphi_1}$$

Gradient Descent



$x_{i+1} = x_i - df/dx$

Approximate Derivative



df/dx=Slope=Δf/Δx avec Δx petit

Iterative IK : gradient descent

Repeat

ForEach joint i Do

$grad_i = (df / d\varphi_i)$

$= (f(\varphi_{0,} \ldots, \varphi_i + d , \ldots, \varphi_n) - f(\varphi_{0,} \ldots, \varphi_i, \ldots, \varphi_n))/d$

EndForEach

ForEach joint i do

$\varphi_i \rightarrow \varphi_i - \beta * grad_i$

Adapt(d);

Adapt(β);

Until d<0.01 (meaning while **e** is too far from $\mathbf{e}_{\mathbf{target}}$)

M2 85