Human Animation for Interactive Systems:
Reconciling High-Performance and High-Quality

Michael Gleicher
and the UW Graphics Group
University of Wisconsin- Madison
www.cs.wisc.edu/~gleicher
www.cs.wisc.edu/graphics

The Vision:
Virtual Experiences

- Simulate Experiences
- More than just being in a place
- More than just appearance

- One aspect: Other people

Virtual Experience Applications

- Training
  - Emergency Rescure, Crowd Control
- Design
  - Evaluating populated spaces
- Entertainment / Commerce
  - Games, Shopping Malls, Chat

- All need people!

The Demands of Virtual Experiences

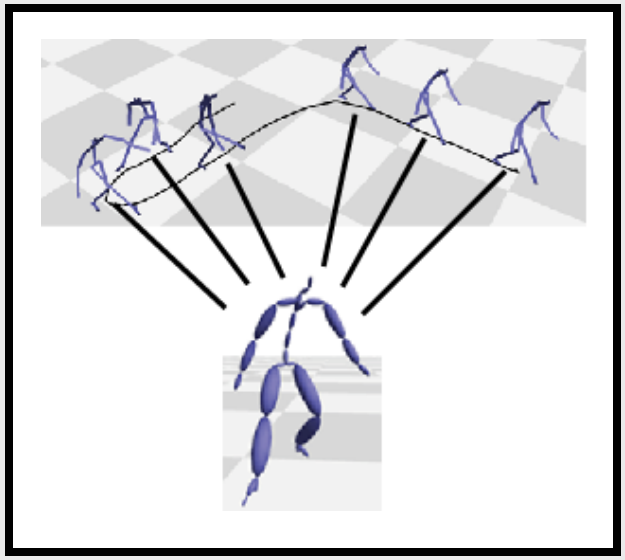
- Fast
- Responsive
- Streams of Motion
- Realistic
- Attractive
- Detailed
- Directable
- Autonomous

The problems

- What do characters do?
  - How do they choose their actions?
- How do they do it?
  - How do we generate their movements?
- What do they look like?
  - How do we draw them?

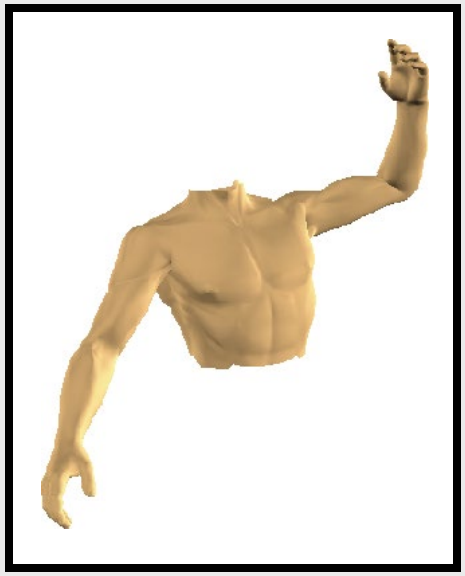
Two Sub-Projects

Snap-Together Motion



High-Quality Motion
Within Interactive Apps

Efficient Skins



High-Quality Appearance
Within Interactive Apps

How do we synthesize character motion?

- Need streams not clips
- Generated in response to action
  - Rules out capture, hand-animation, …
- Procedural Methods?
- Physics?
  - Can’t get richness and quality

Synthesis by Example

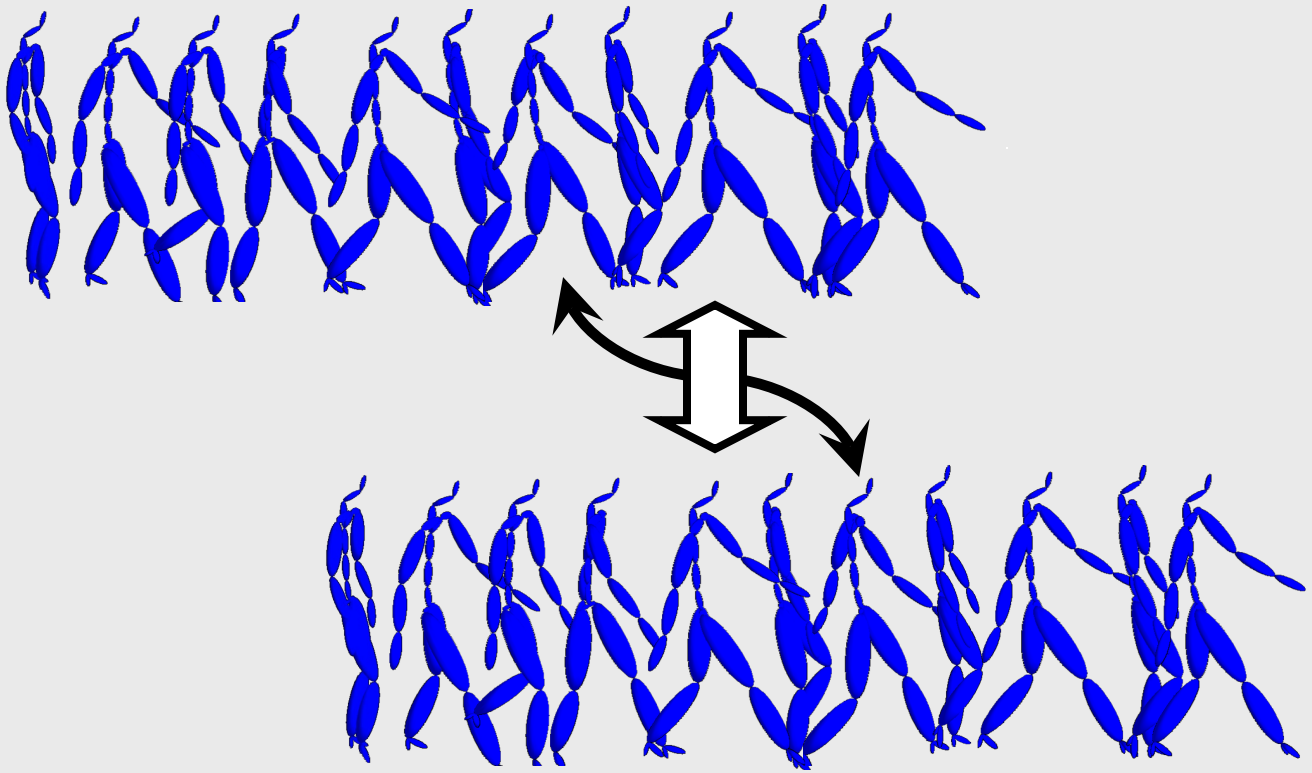
- Graph-based approaches
  - Kovar et al., Lee at al., Arikan and Forsyth
- Good:
  - High quality results
  - Flexible
- Bad:
  - Offline (need to search)
  - Limited Precomputation

We'll fix this

- Trade some quality for performance

A Review:
Synthesis by Example

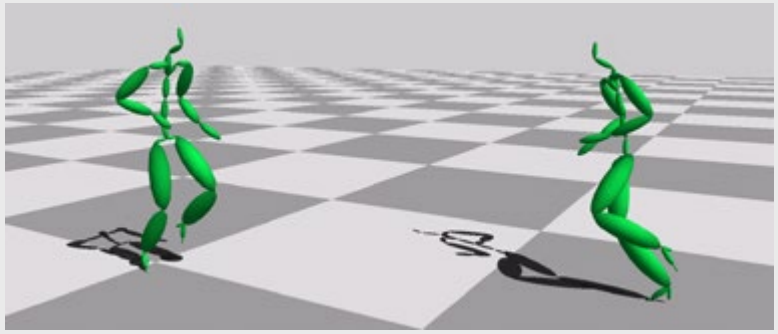
- Piece together clips of good motion
- Find “easy” transitions to make



What is Similar?

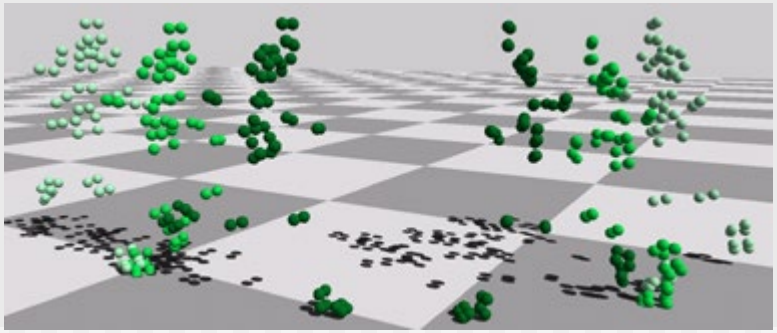
- Factor out invariances and measure

1) Initial frames

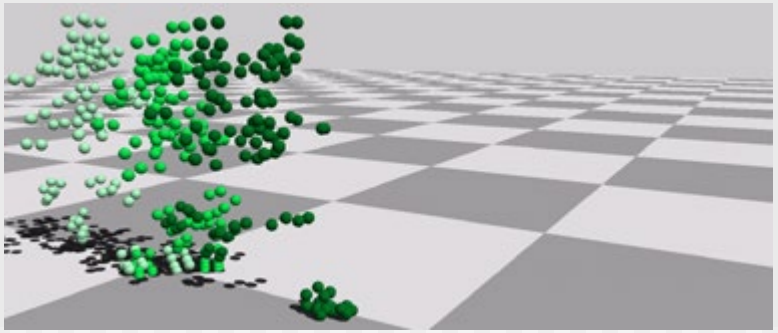


2) Extract windows

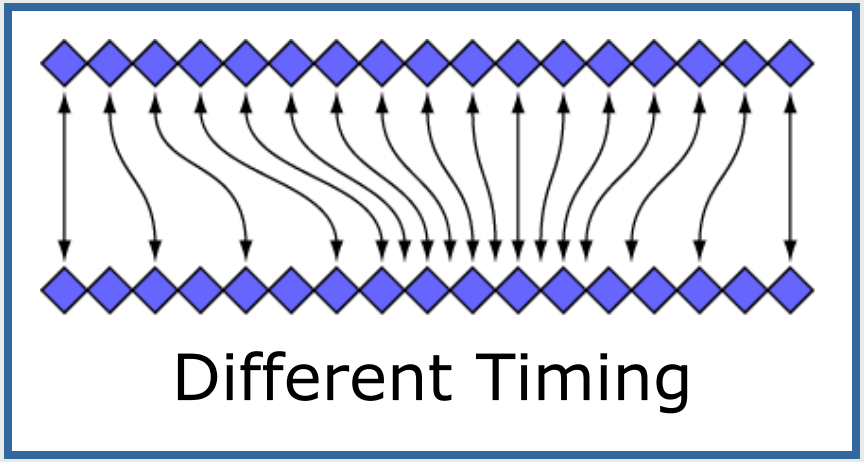
3) Convert to point clouds

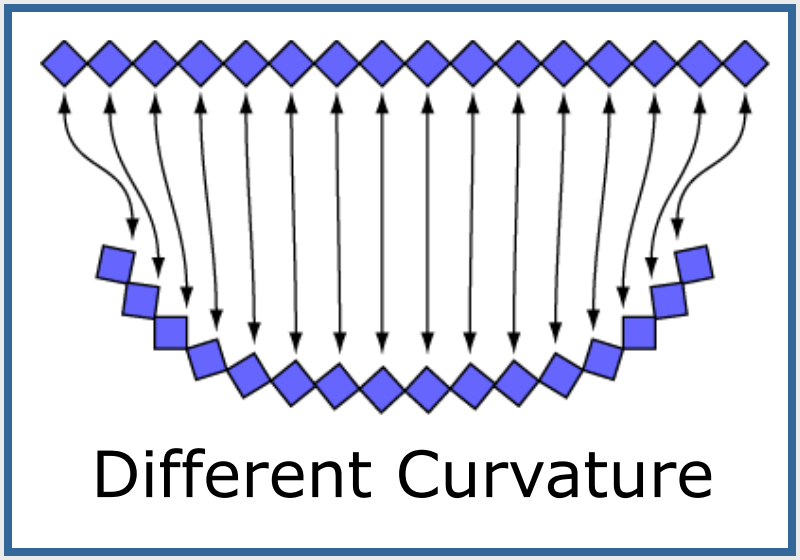


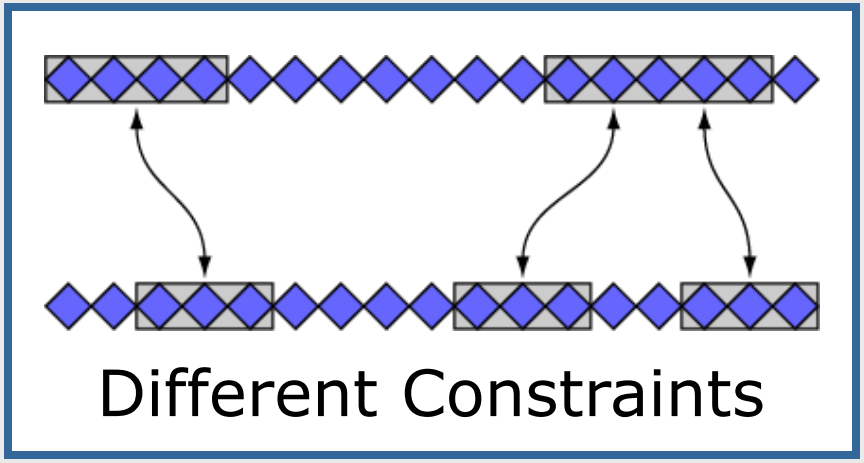
4) Align point clouds and sum squared distances



An Aside: Other Invariances
(Kovar&Gleicher '03)

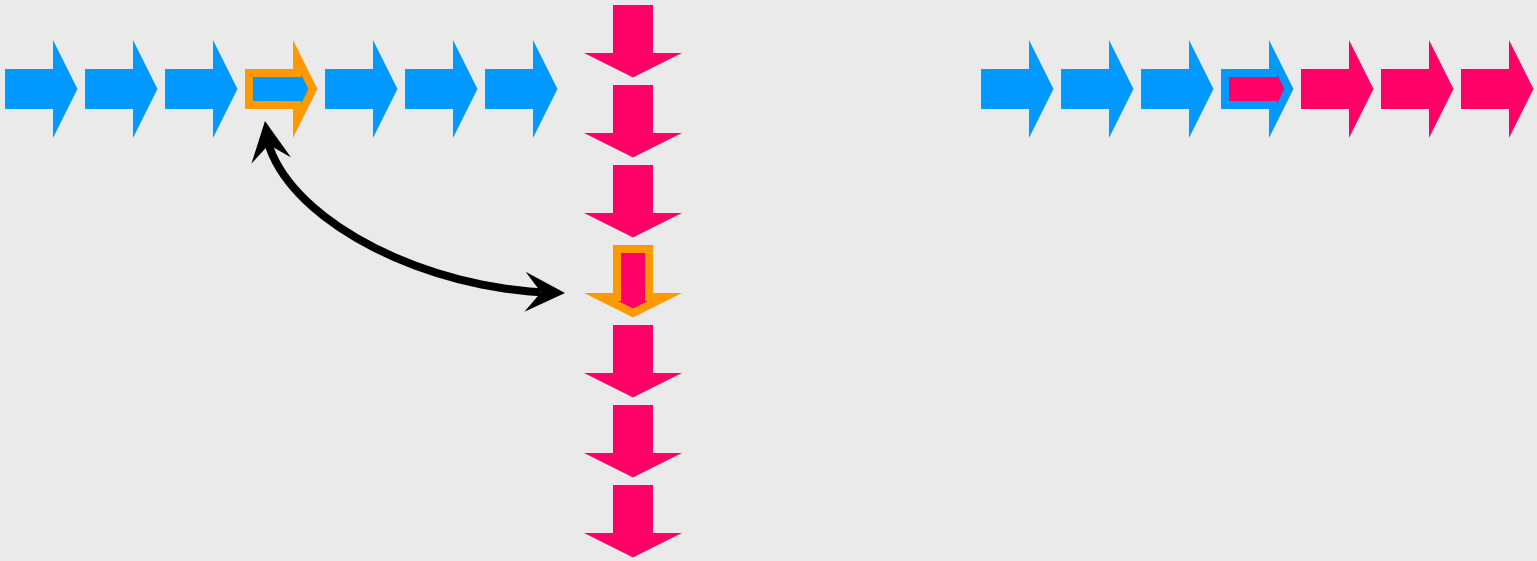






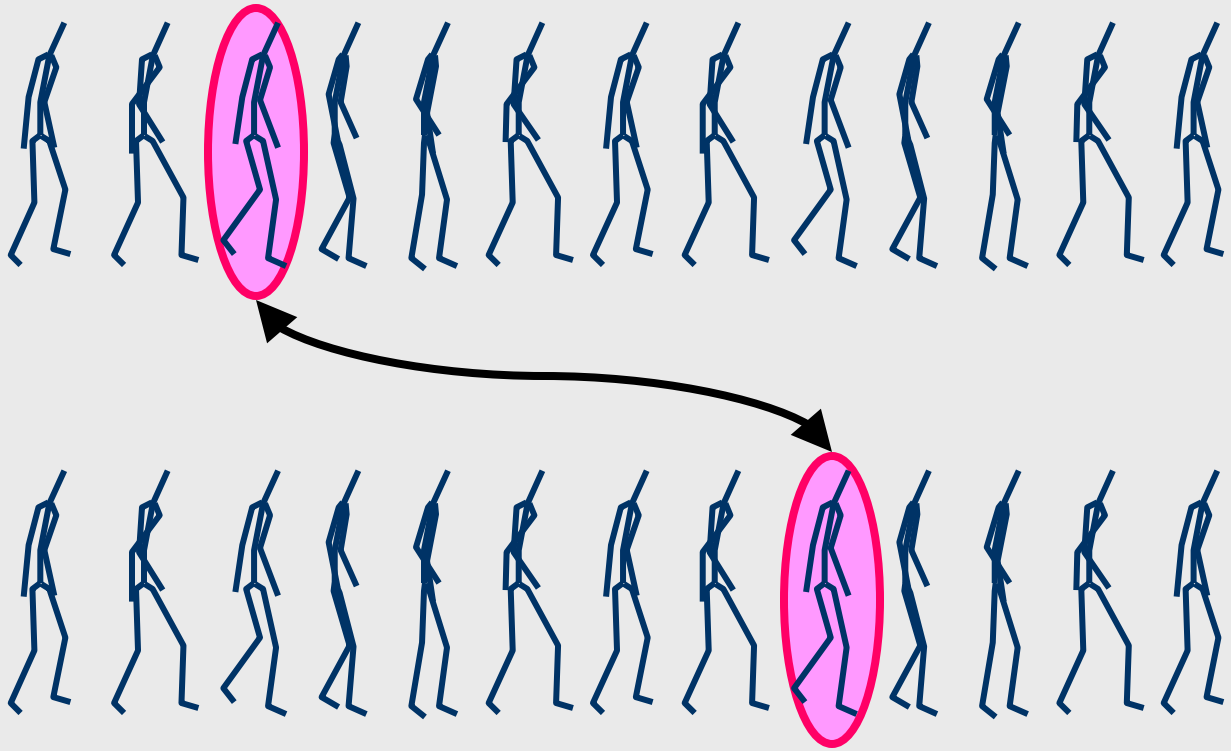
An easy point to miss: Motions are Made Similar

- "Undo" the differences from invariances when assembling
- Rigidly transform motions to connect



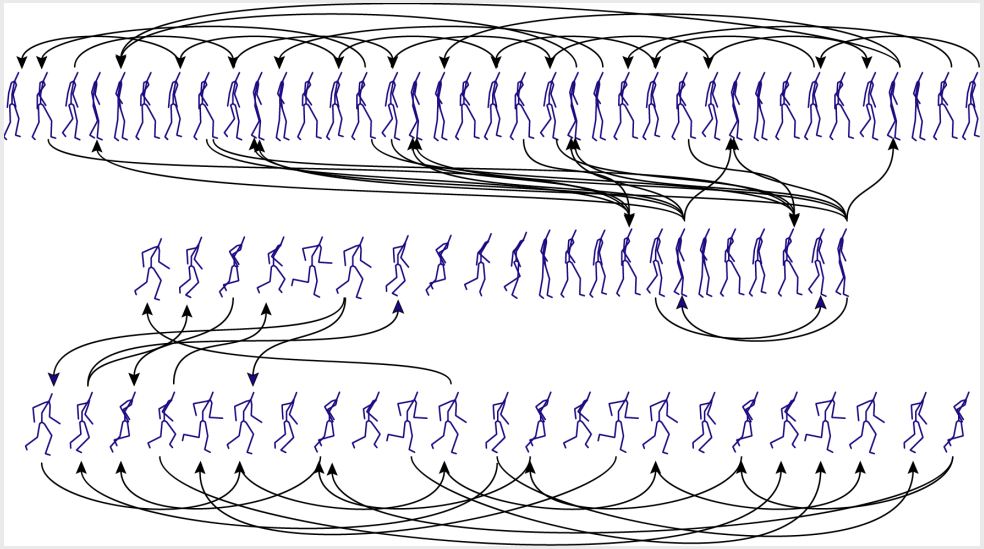
Building a Motion Graph

- Find Matching States in Motions



Automatic Graph Construction

- Find many matches (opportunistic)



- Good: Automatic
- Good: Lots of choices

Motion Graph Problems

- Graphs built opportunistically
  - Leads to unstructured graphs
  - Don’t know what’s there
  - No control over connectivity
  - No way to know what is close
  - Don’t know what character can or can’t do
- Lots of transitions
  - Can’t check them all, be conservative

Why is this OK?

- Search the graphs for motions
- Look ahead to make sure we don't get stuck
- Cleanup motions as generated
- Plan "around" missing transitions
- Optimization gets close as possible

Not OK for Interactive Apps!

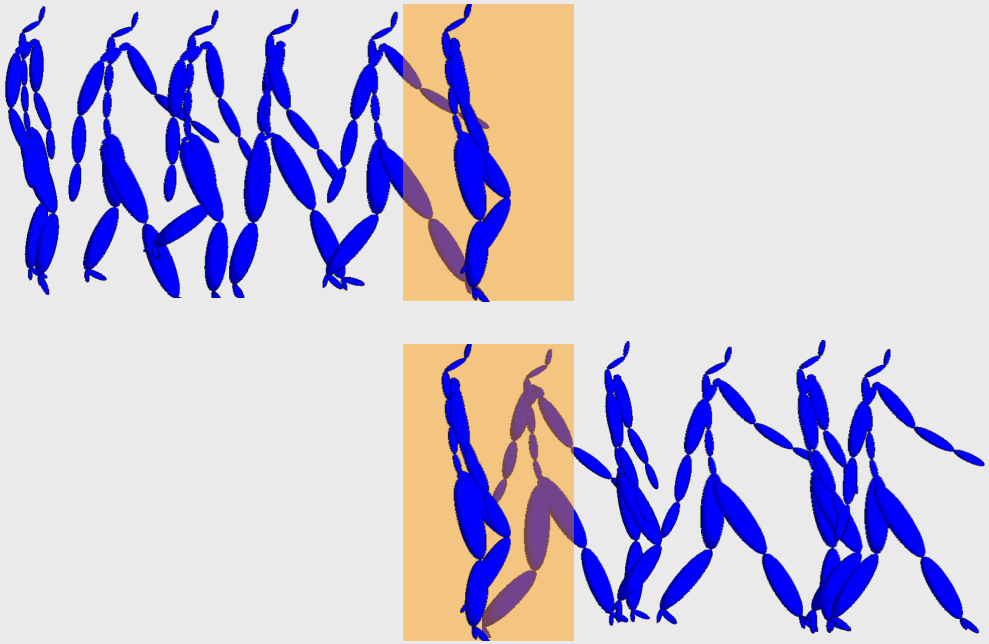
Snap-Together Motion

Three basic ideas

- Find “Hub Nodes”
- Pre-process motions so they “snap” together
- Build small, contrived graphs

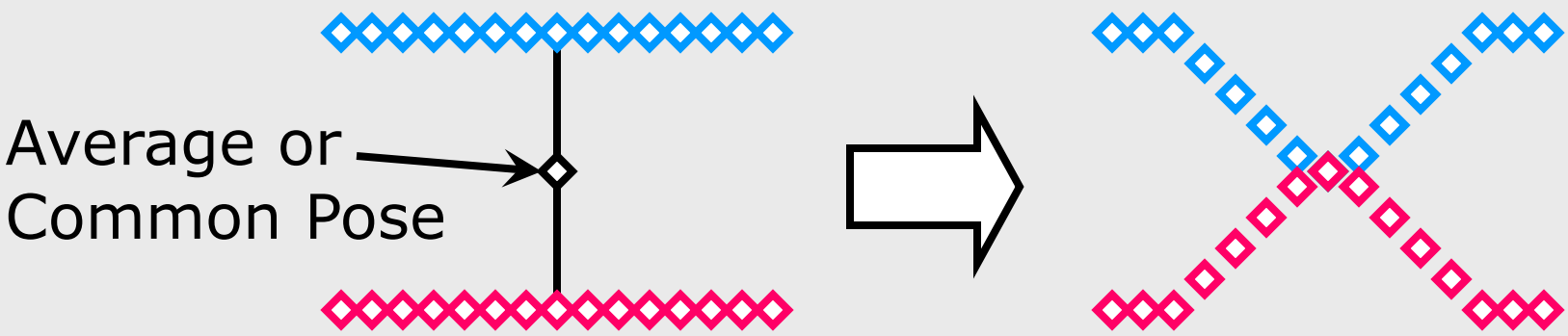
Snapable Motions

- What if motions matched exactly?
  - Match both state and derivatives
  - Match reasonably at a larger scale



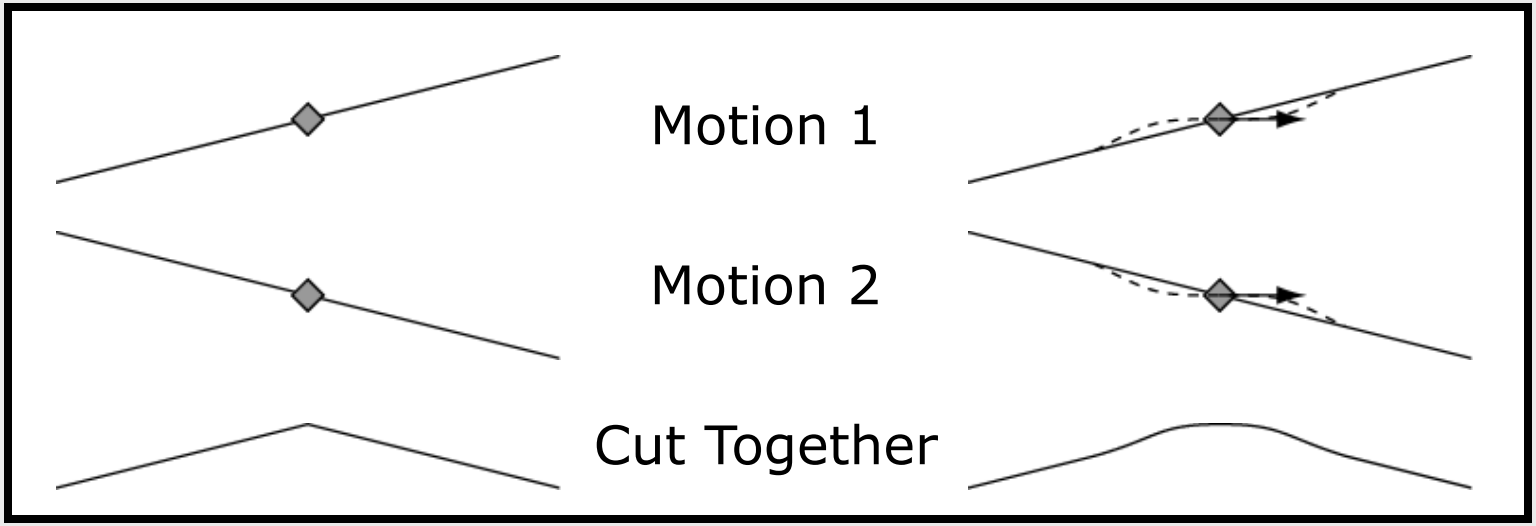
Make motions match exactly

- Add in displacement maps
- Bumps we add to motions
- Modify motions to common pose
- Compute changes at author time



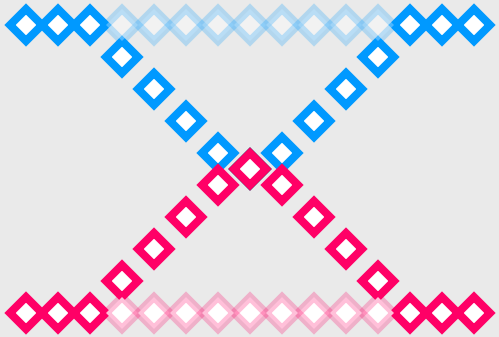
C(1) Displacement Maps

- Add displacements to make motions match in both value and derivative
- Displacement maps themselves are smoother



Some features of Cuts

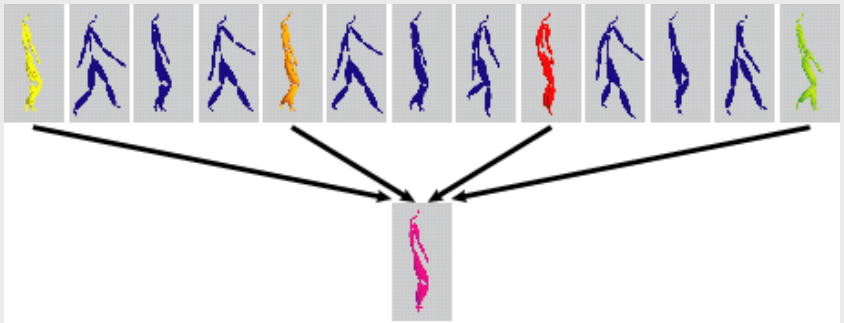
- Don’t need original motions
  - Just use displacements
  - No database growth

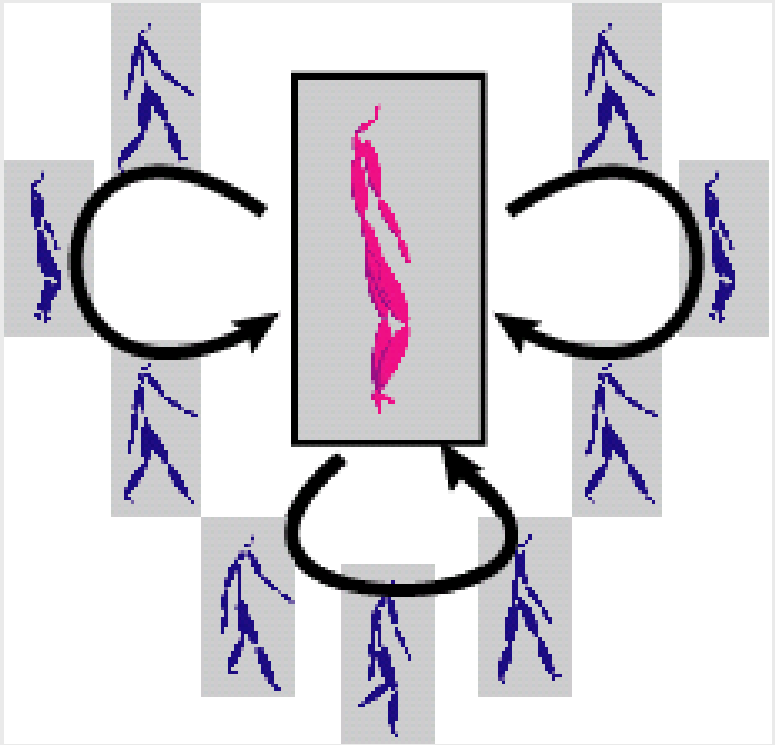


- Multi-way transitions
  - Find common point for several motions

Hub Nodes

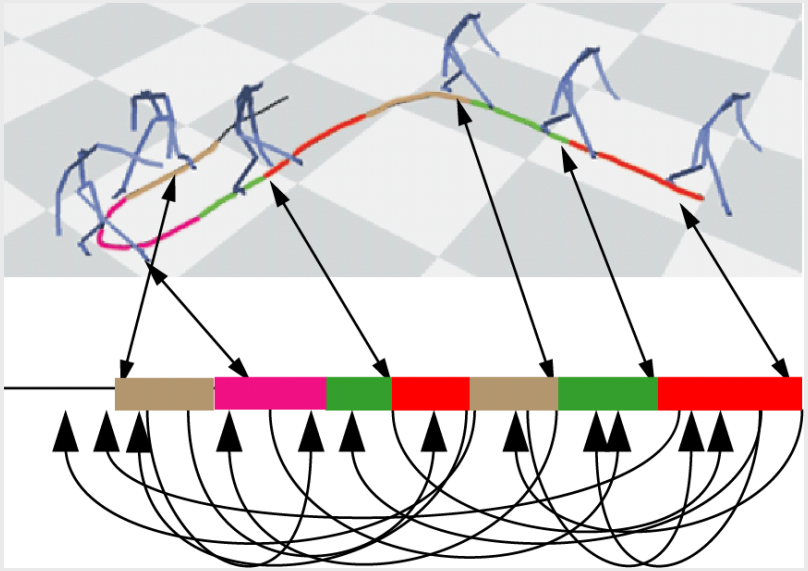
- Multi-way cuts yield Hub Nodes
- Hub nodes allow nice graphs



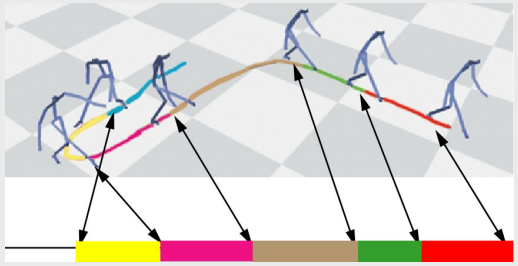


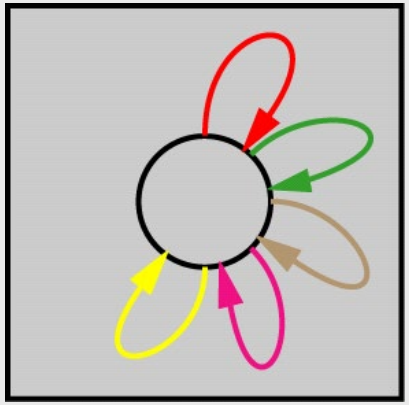
Contrived Graph Structure?

Search: Look ahead to get where you need to go

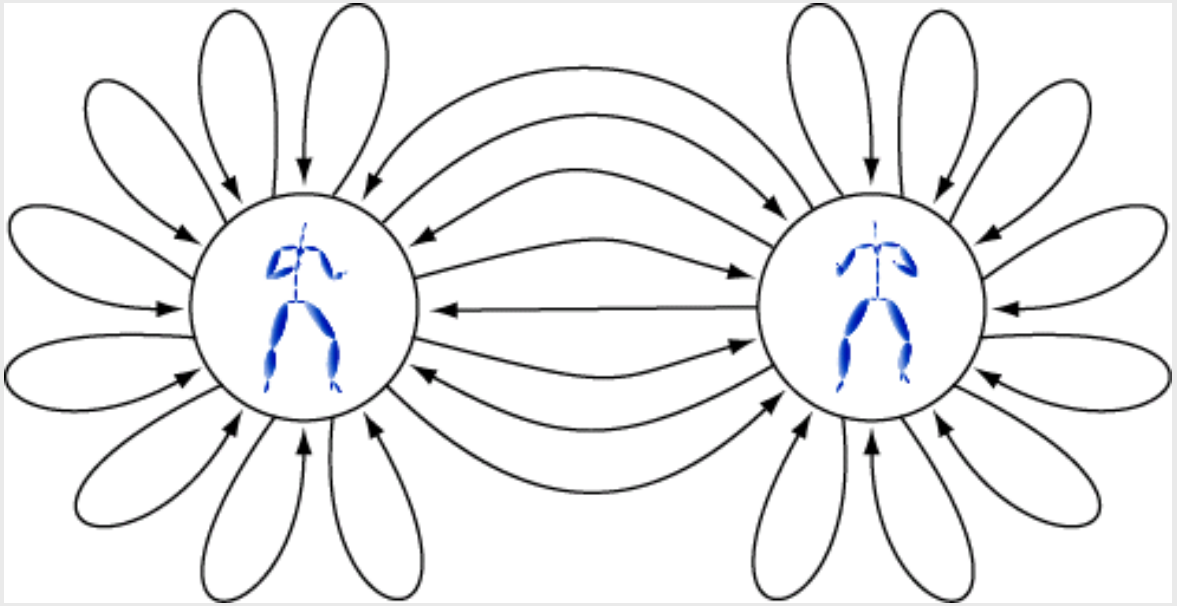


React: Always lots of choices. Something close to need.





Gamers use these



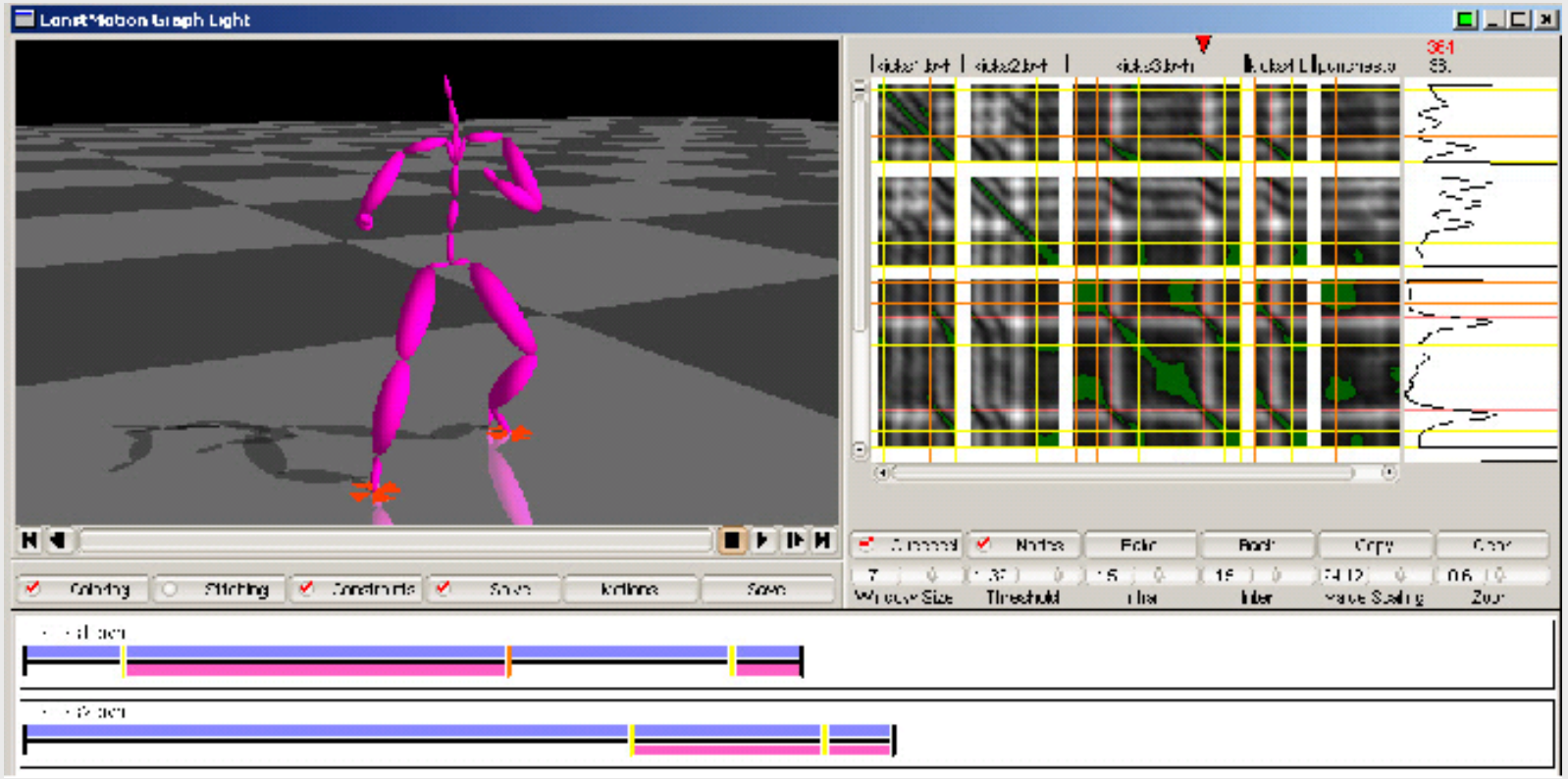
Semi-Automatic Graph Construction

- Pick set of *match frames*
  - User selects
  - System picks “best” one
- Modify motions to build hub node
- Check graph and transitions

Common poses for Multi-way Hubs

- All motions displace to one pose
- Pose must be consistent with all motions
- Average “common” pose
  - Average of poses
  - Constraints solved on frame
- Must solve constraints on all motions
  - Inconsistencies are possible

Building the Motion Graphs



Why should we care?

- Precompute everything but rigid transforms = FAST!
- Nodes with high connectivity = RESPONSIVE / CONTROLLABLE
- Add in user interaction
  - Build convenient graph structures
  - Verify transitions (less conservative)

Some results

- Corpus of Karate Motions
  - Pick two best poses
  - Map motions to game controller
  - 12 minutes – including picking buttons
- Walk
- Walk + Karate
  - 20 minutes – including picking buttons

Nothing for Free

- Less Variety
- Less "Exactness"
- Lower Quality Transitions
  - "Big" Transitions
  - Big changes to get multi-way
- Need user intervention
  - Pick Match Frames
  - Verify Transitions

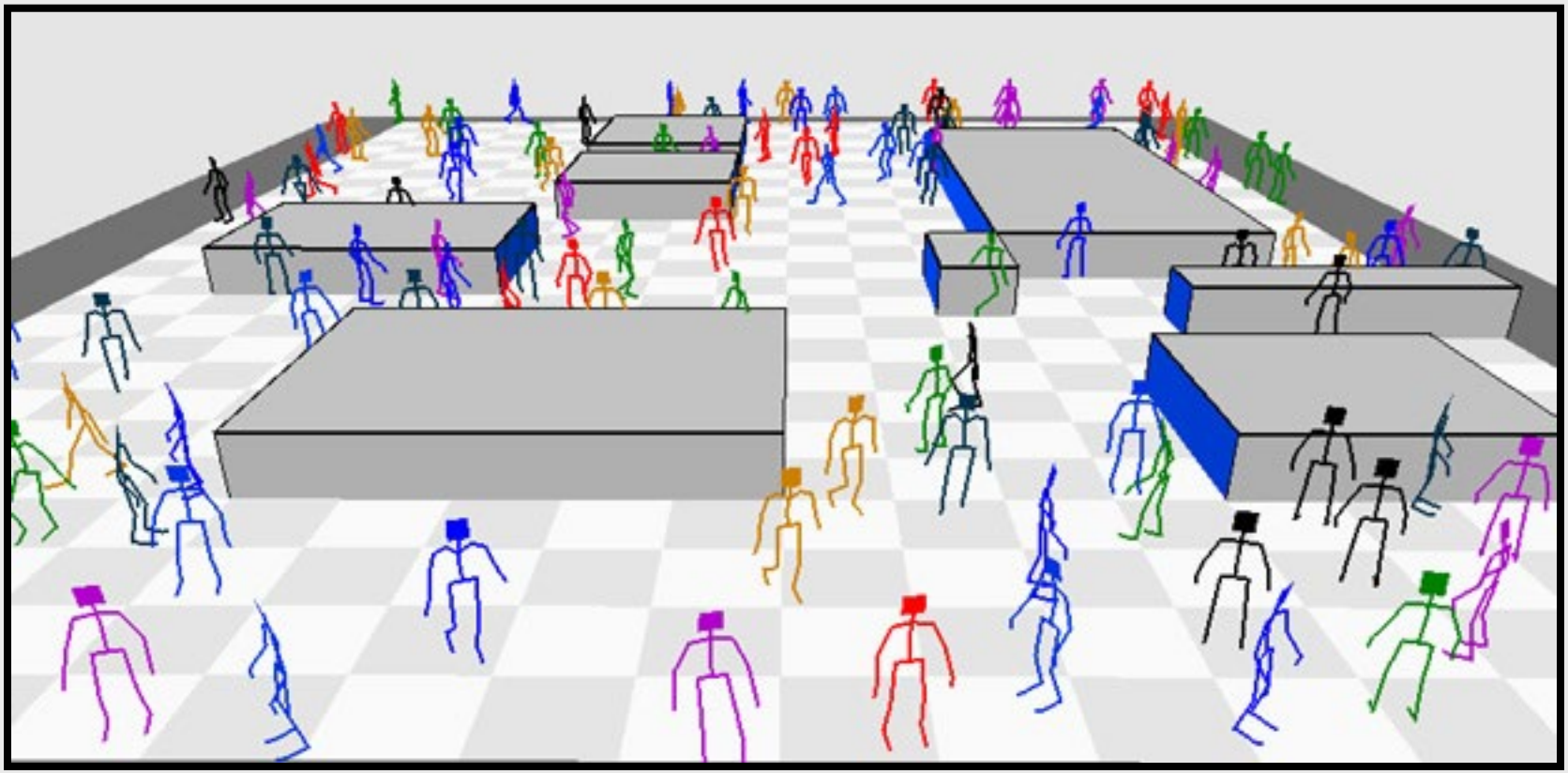
In Practice, heuristics work REALLY well – but no guaruntees

Are we done yet?

- May not have exact motions you need
  - Make them using blending techniques
- Discrete nature (choices)
  - Parametric “fat” arcs
- Better matching of dissimilar motions
  - New and better invariances
  - Motion specific invariances

- How do characters decide what to do?

Snap Together Motion in Action
Crowd Simulation



Snap-Together Motion

- Animation by Example for Interactive applications!

- User-controlled graph structures
- Multi-way cut transitions
- Pre-computed transitions

Now that we have Movements…

How to draw characters?

- Skinning:
  - Mapping controls to geometry
  - n parameters, p points

$$f_{skin} : \mathcal{R}^n \rightarrow \mathcal{R}^{3*p}$$

How to do this?

- Arbitrary code
- Need to embody art in code

- Easier: deform given geometry

$$f_{skin} : \mathcal{R}^n \rightarrow (f : \mathcal{R}^3 \rightarrow \mathcal{R}^3)$$

Where do “high-end” characters come from?

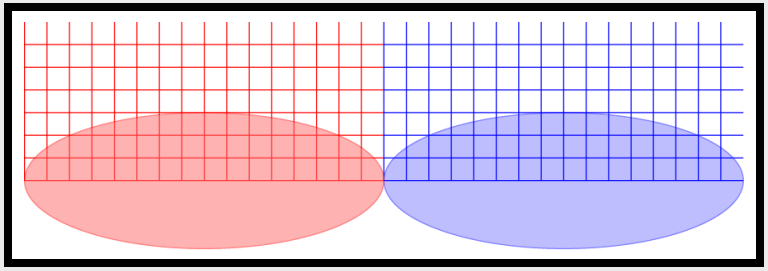
- Artists make model
- Technical director “rigs” model
  - Defines deformations
  - Arbitrary code
    - Non-linear deformations
    - Expressions
    - Code Pieces (physical simulations)
- Animation system executes rig

Where do low-end (interactive) characters come from?

- Run-times (now hardware) embody a specific mathematical model
- Riggers forced to use these abstractions
  - Skinning is setting vertex weights
  - I will explain this, and why its bad
- Couples authoring and runtime

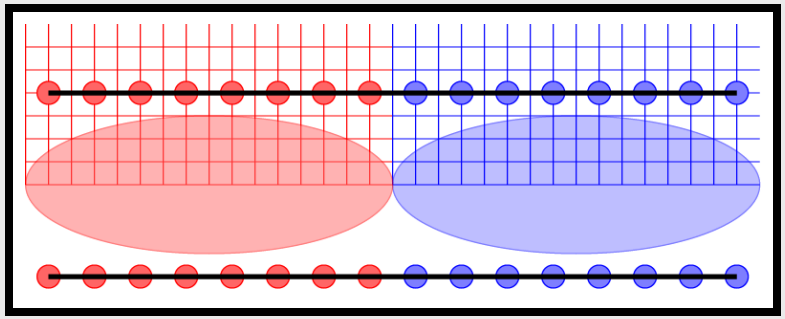
Skinning 101

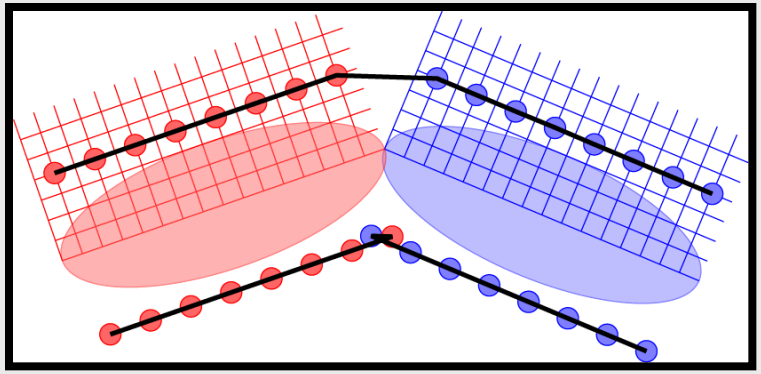
Each “bone” is a coordinate system



A coordinate system is a 4x4 matrix in graphics

Rigid Skin (Catmul ’72) put each point in one coordinate system



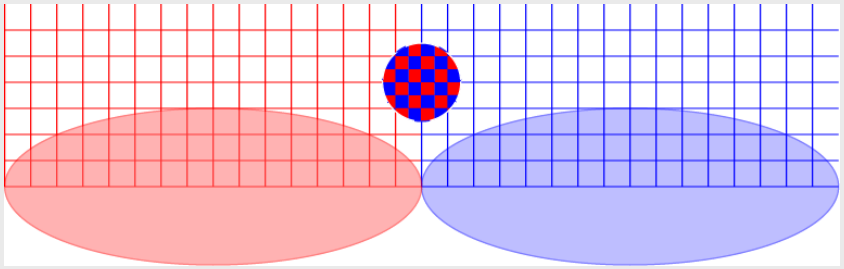


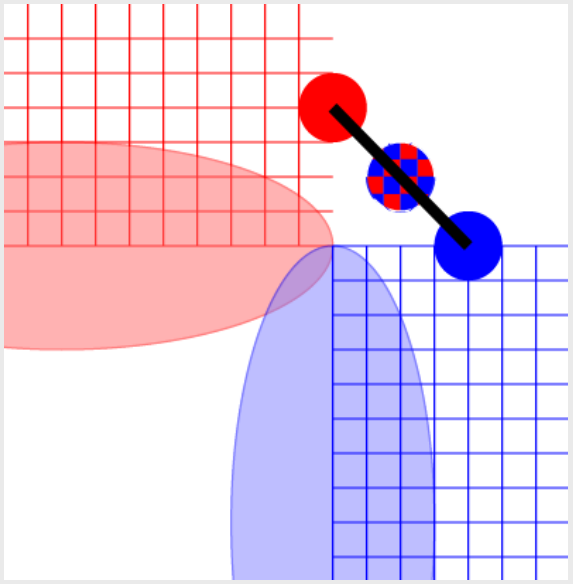
Skinning 101 (2)

Method with many names

Linear blend skinning, skeletal subspace deformation, enveloping, smooth skin

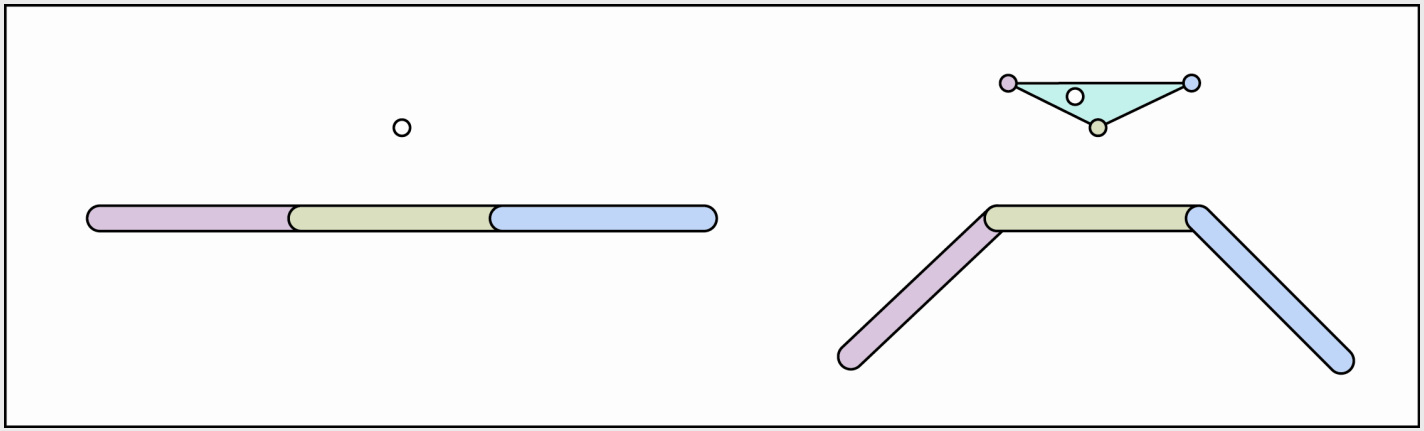
- Each point influenced by multiple bones
- Weighted linear blend





Using Linear Blend Skins

- Each point has weight vector
- Artist (rigger) sets weights
  - Possibly by direct manipulation (Mohr, Tokheim, Gleicher ’03)
  - Often by “weight painting”
- Range of effects from different weightings



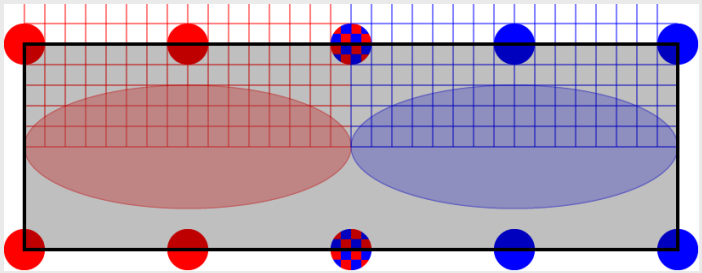
Why is this bad?

- Artists work with weights
- Good for hardware, not good for artists

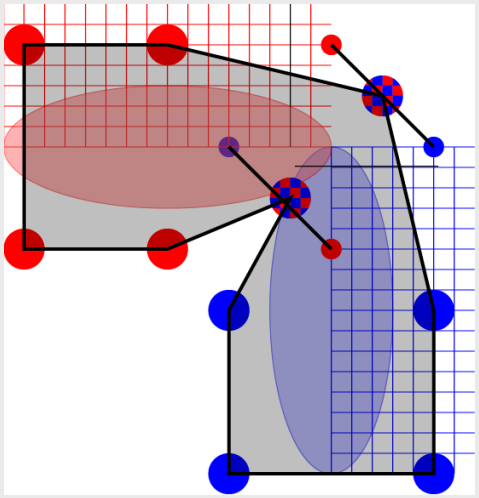
- Hard to get complex effects
- Hard to know what can be gotten

- Can lead to bad artifacts
- Hard to know if artifacts can be fixed

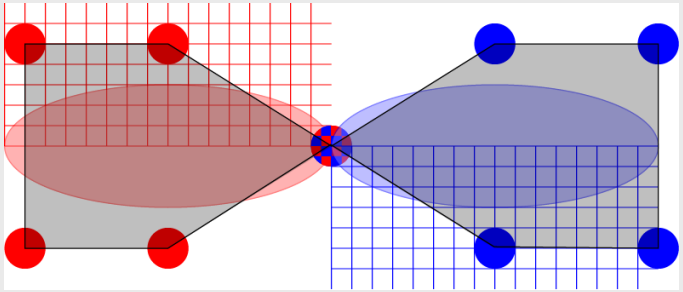
Linear-blend skin artifacts



- Pinching



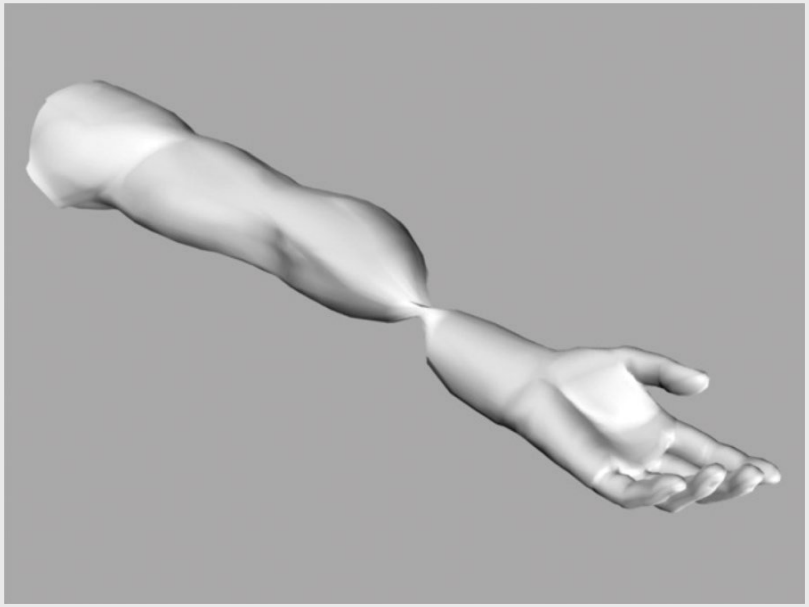
- Candy-Wrappering



Linear-blend skin artifacts

- Pinching

- Candy-Wrappering



What do others do?

- Give up on linear blend skinning
  - But can’t fit in existing run-times
- Interpolate between example shapes
  - Can’t scale and stay interactive

- Multi-weight enveloping (MWE)
  - Add more parameters to LBS model
  - Need artist input (approximate solution)
  - Automatically determine parameters
  - Mathematical tricks for stability

Our Solution: Decouple Authoring and Display

- Author skins using whatever high-end method they like
- Compute linear-blend skin that best approximates authored rig
- Optionally extend LBS model
  - Add joints since they are used automatically

Mathematically

- Skin defines deformation field

  $f_{skin} : \mathcal{R}^n \rightarrow (f : \mathcal{R}^3 \rightarrow \mathcal{R}^3)$
- Sample this function
  - Easy! Just run some animation
- Construct a good approximation
  - Form of approximation should map to existing infrastructure
  - Piecewise linear approximation with non-linear basis functions

The Details

Linear Blend Skin Model

$$\mathbf{p} = \sum_{i}^{joints} \mathbf{w}_i \, \mathbf{M}_i \, \mathbf{D}_i^{-1} \, \mathbf{p}_d$$

point poisition

weight vector

bone matrices

dress-pose matrices

dress pose point position

Example skins

For each example and each point:

- We know M_i (bone configuration)
- We know p_e (where point should be)
- We know D_i^{-1} (inverse dress pose matrix)
- We sortof know p_d (dress pose position)
- We want to find w_d (weight vector)

Optimal Linear Blend Skins

- Minimize error across examples

$$g(\mathbf{w}_i, \mathbf{p}_d) = \sum_{e}^{examples} \left\| \mathbf{p}_e - \sum_{i}^{joints} \mathbf{w}_i \, \mathbf{M}_i^e \, \mathbf{D}_i^{-1} \, \mathbf{p}_d \right\|^2$$

- Bilinear least squares problem
- Solve by alternation
  - We have a good estimate for p_d

Extending Linear Blend Skinning

- Matrices are functions of parameters

$$M_i = f_i(q)$$

- Consider as our basis functions
- To get a better approximation, pick better basis functions

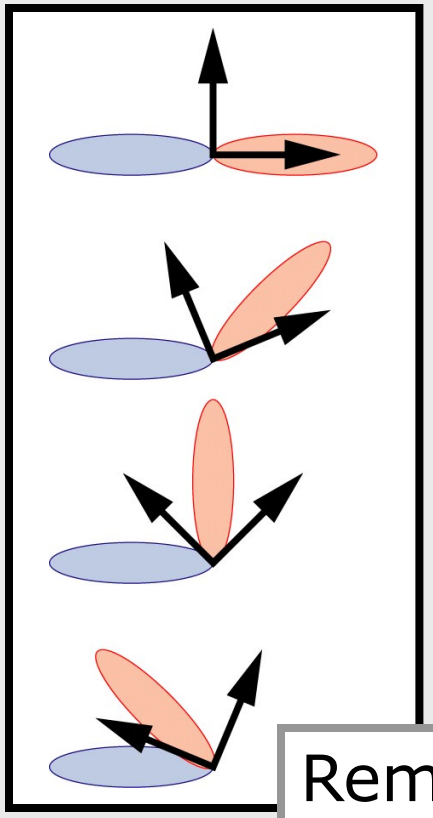
Extending the basis set

- Still within linear blend skin model
- Little computational overhead
  - Basis set is still small, many points
- User doesn’t do any more work

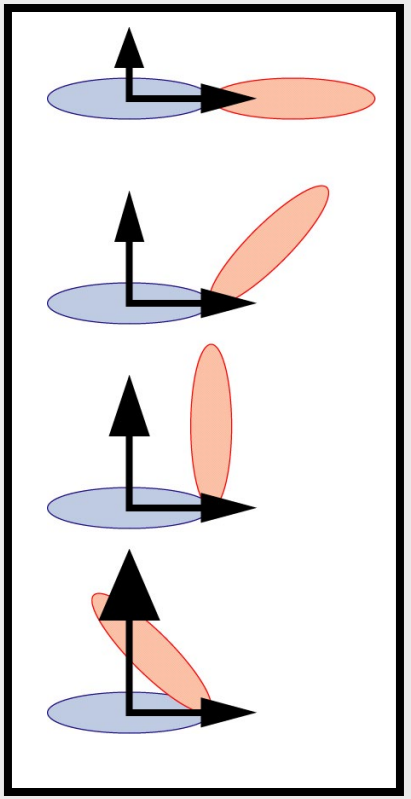
- What functions to add?
  - Today: Ad-Hoc – pick known good functions
  - Future: determine what bones to add

“Bones” to add

- Half-Angle



- Scale



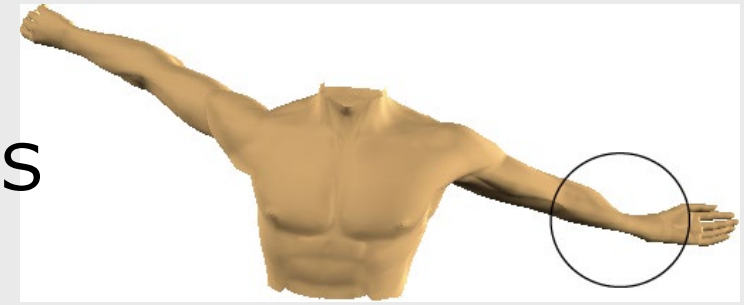
Remember:

$$R(\theta_1 + \theta_2) \neq R(\theta_1) + R_(\theta_2)$$

Results

original

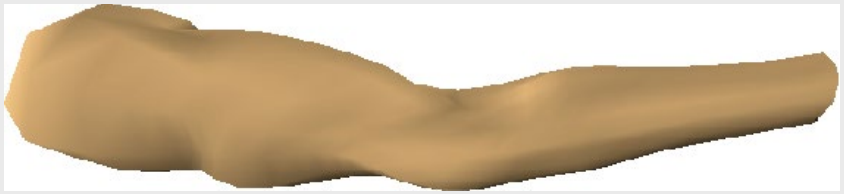
basic LBS



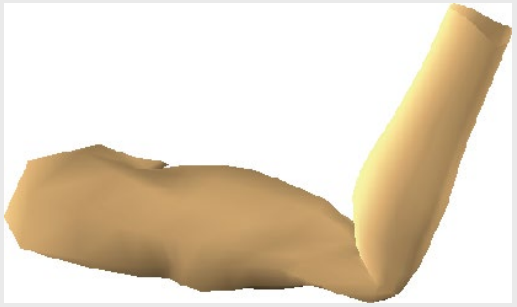
our method

More results
Bulging Muscles

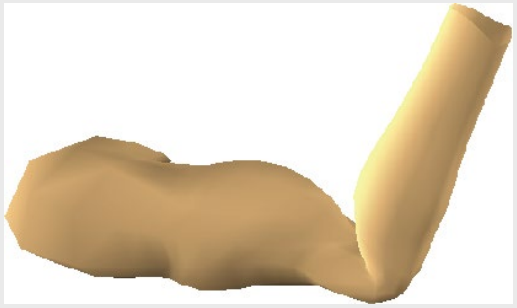
original



basic LBS



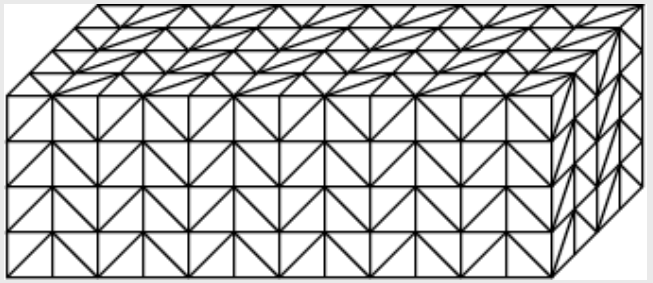
our method

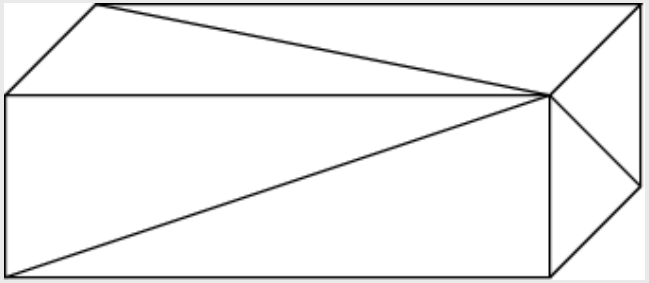


More results

An addition: Deformation Sensitive Decimation

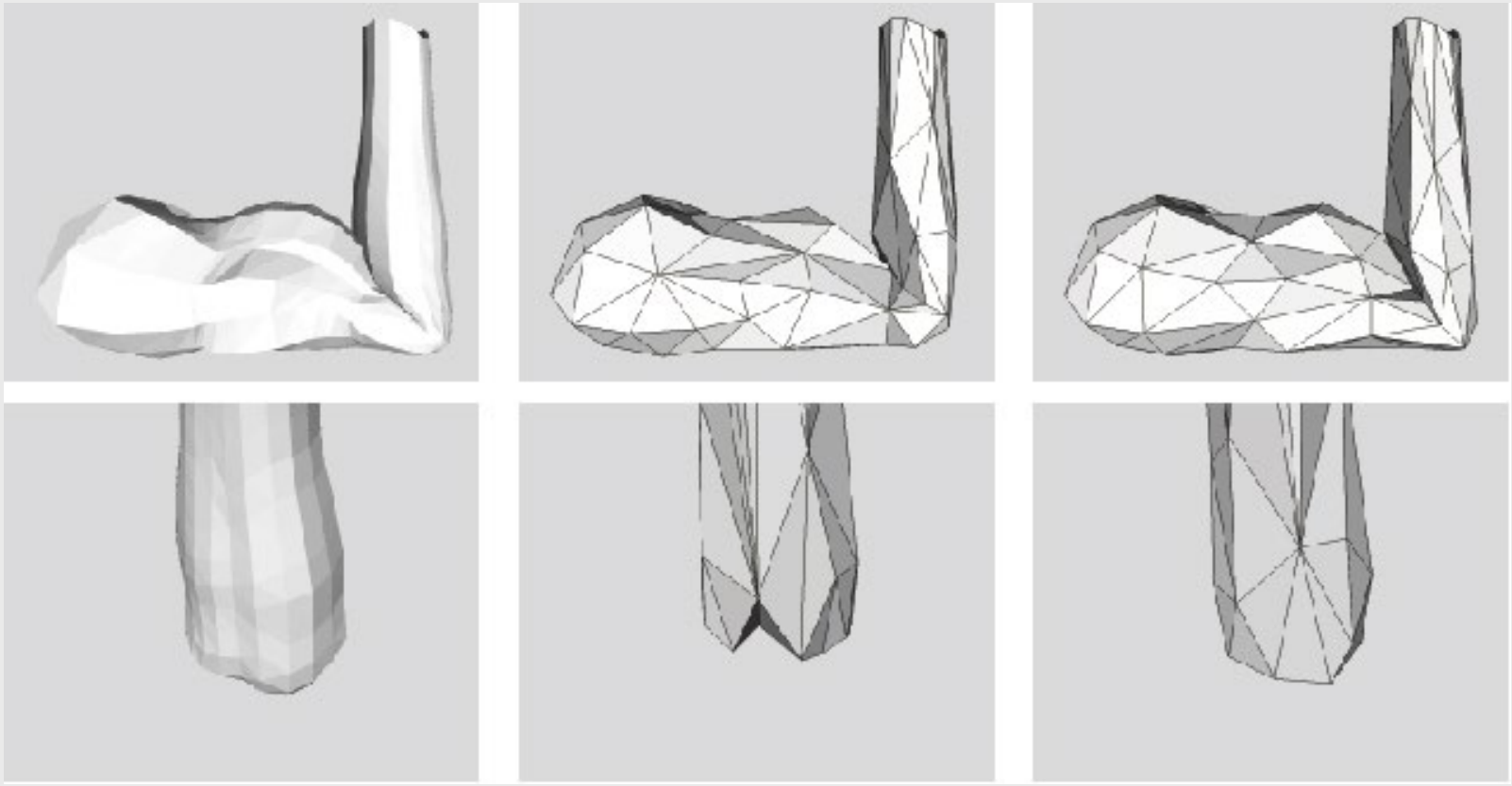
- Simplify geometry for speed
- Simplifying the rest shape doesn't work
- Extend QSLIM to look at set of examples



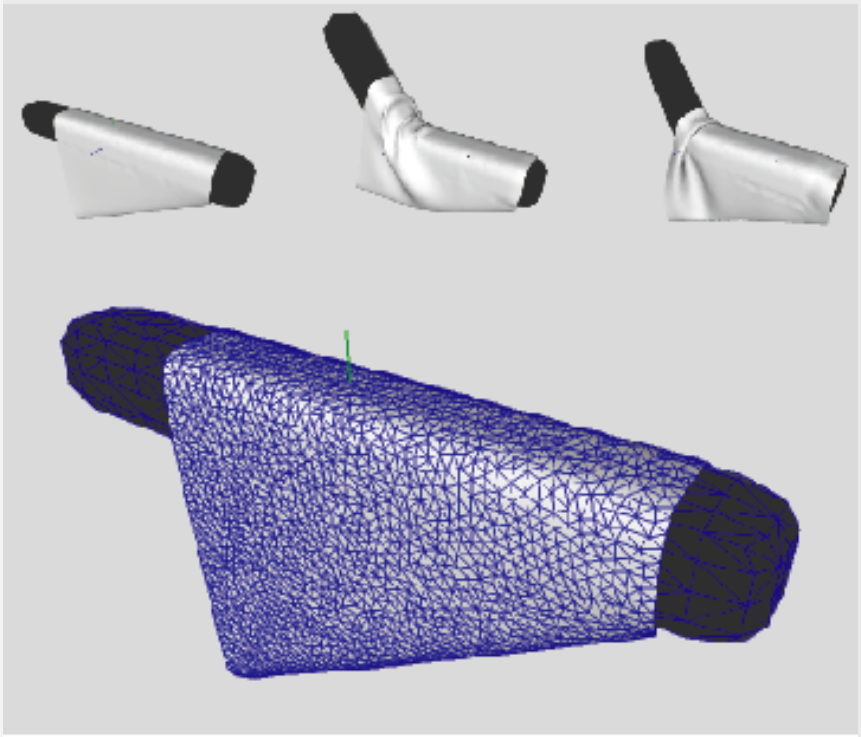


Decimation Results

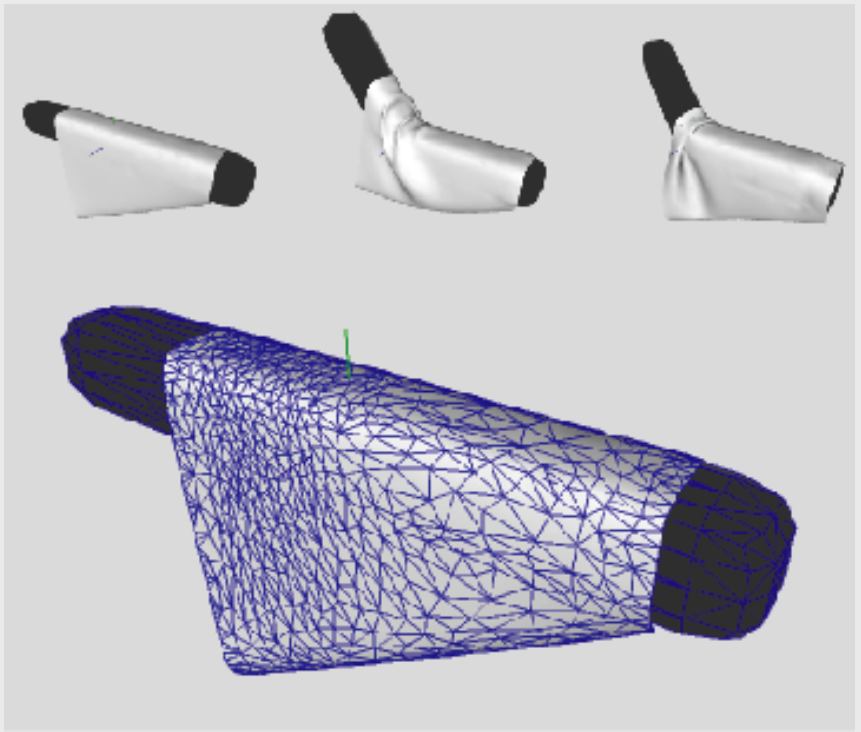
original | QSLIM 1 pose | our method



Use with Cloth Simulation



Simulation (6170 faces)



Simplified (2170 faces)

What did we just do?

- Used the existing skin model
  - Works with existing runtimes
  - (Almost) No performance penalty
- Better results with less work
  - Avoids worst linear blend artifacts
  - Avoids artists having to think about fast runtime models

Some things to think about

- Better than linear blend skinning?
  - Now that we have programmable hardware…
- Better basis sets
- Less ad-hoc basis sets

- How to author the rigs to begin with

The bigger picture:
Creating Virtual Experiences

- How do we ***author*** virtual experiences?
- How do we provide high-quality characters in virtual experiences?

Thanks!

- To the UW graphics gang.
- Animation research at UW is sponsored by the National Science Foundation, Microsoft, and the Wisconsin University and Industrial Relations program.
- House of Moves, IBM, Alias/Wavefront, Discreet, Pixar and Intel have given us stuff.
- House of Moves, Ohio State ACCAD, and Demian Gordon for data.
- And to all our friends in the business who have given us data and inspiration.