

# AniMesh: Interleaved Animation, Modeling, and Editing

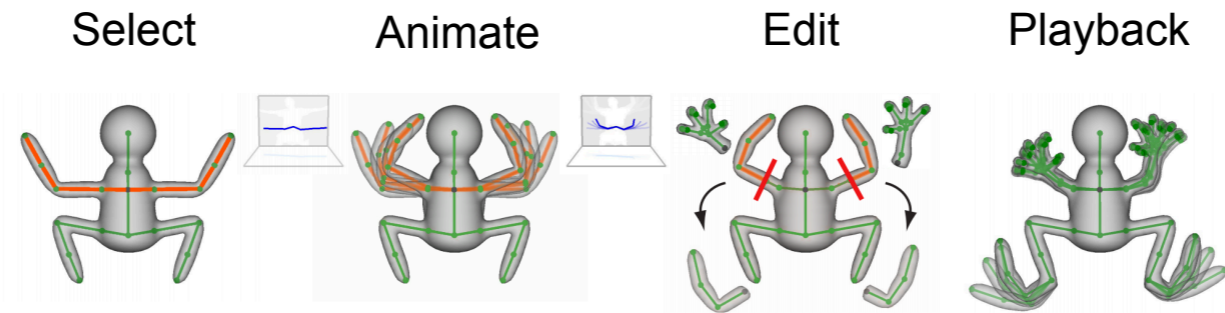
Ming Jin, **Dan Gopstein**,  
Yotam Gingold, Andy Nealen

Sponsored by  





Thanks for the introduction

# AniMesh in a Nutshell

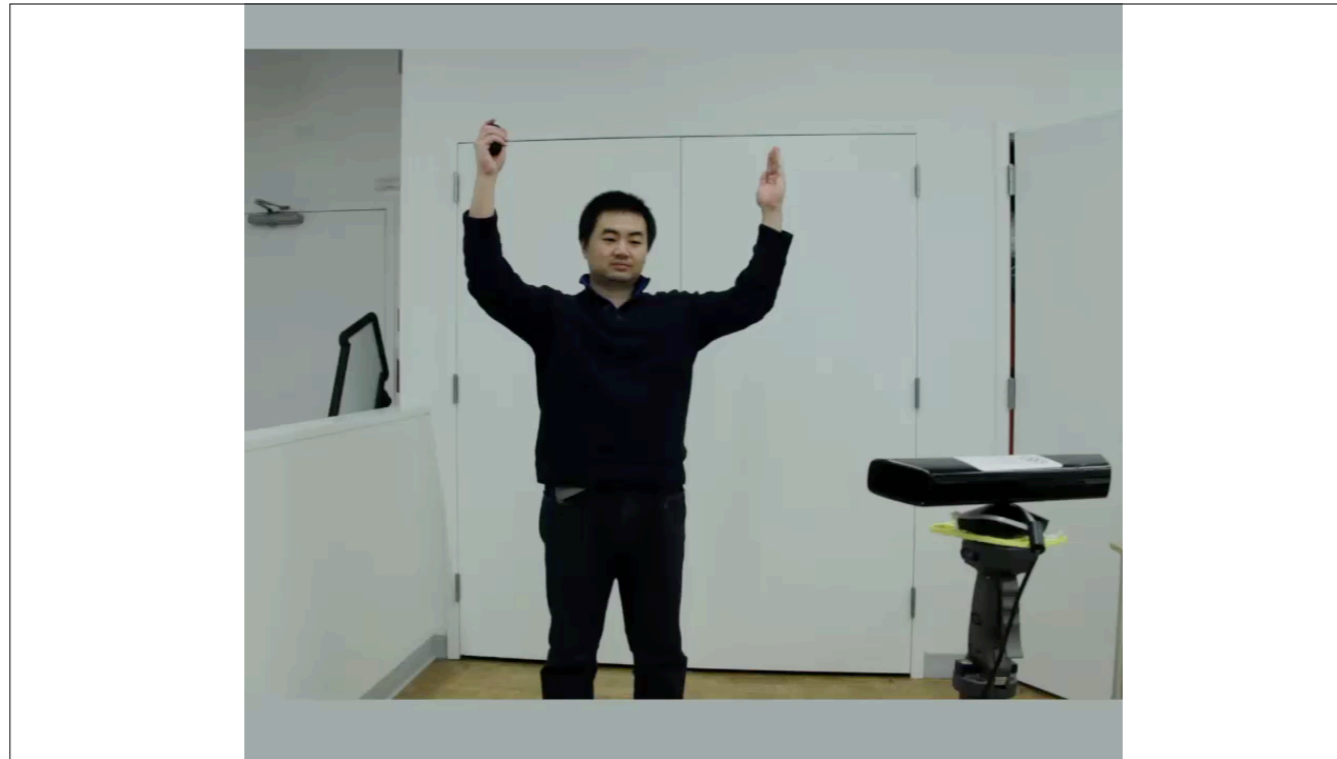


SA2015.SIGGRAPH.ORG

Sponsored by  

Today I'll be introducing AniMesh, an animation suite for rapid prototyping and design built around intuitive and natural interfaces like sketch-based modeling and performance based animation.

AniMesh encourages a non-linear workflow through its separation of modeling and performance transformations that allows motion retargeting and non-destructive modeling edits at all times.



Here we can see Ming, in front of a Microsoft Kinect, posing to select which sub-skeleton of his model to control with his gestures. In the top right of the screen you can see the Kinect render of his skeleton, and in the center we highlight the sub-skeleton that most matches his position.

After finding a skeletal mapping he likes, he goes on to animate the character by performing motions with his own body.

After he's created an animation sequence he likes, he's then free to go back and make arbitrary changes to the model skeleton. Traditionally, destructive skeletal edits like this would invalidate the animation of the rig, however in Animesh, his performance will be retargeted onto the new joint structure.

# Contributions



- Skeletal Co-abstraction
  - for source-target mapping
- Motion Retargeting
  - for animation preservation

To facilitate the intuitive interface, we created two new algorithms behind the scenes.



The first is a co-abstraction method that lets our users choose which sub-skeleton of the model to take control over

The second is a method for separating animation and modeling transformation data, and continuously blending motions between them so that animation can be preserved even after otherwise destructive modeling edits.

# Traditional Workflow



SA2015.SIGGRAPH.ORG

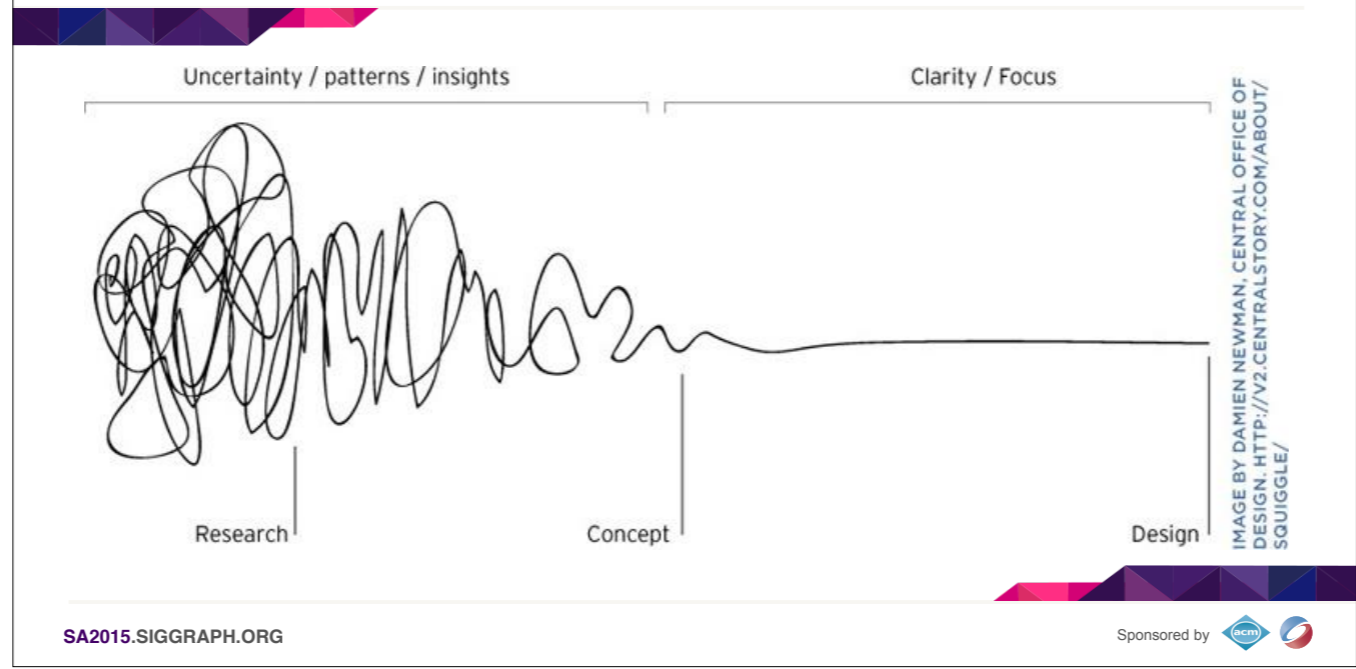
Sponsored by  

The traditional animation pipeline is linear and tightly-coupled. You need to model, then rig, then animate, and any change to the model or the rig will invalidate the animation and require prior work to be redone.

Often these jobs are performed by separate experts in each component, and it can be very time consuming to pass from modeling through to animation

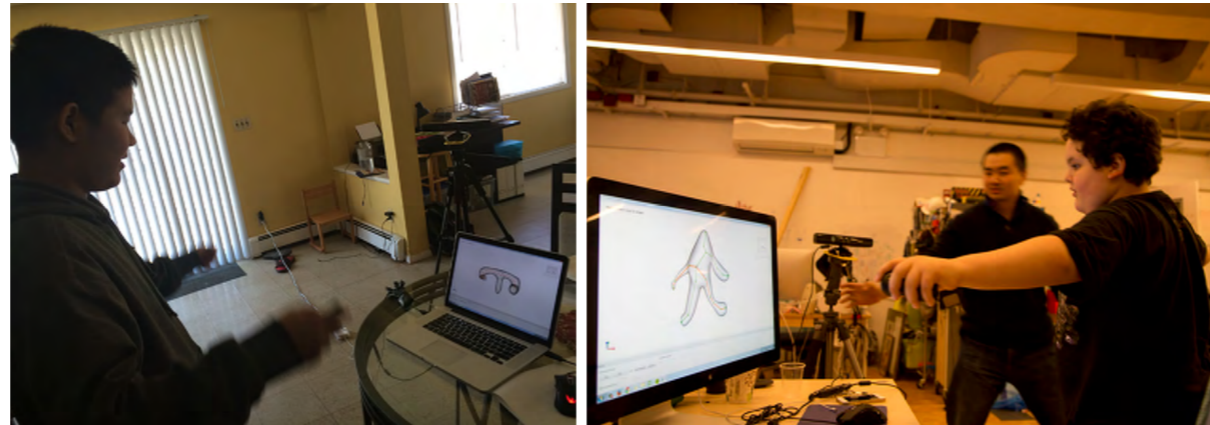
What's more, in order to be able to perform this waterfall pipeline well, you need to spend a good deal of time up front planning exactly what it is you want to accomplish

# Human-centric design





While that may be an effective model for late-stage or big budget projects with the expertise, vision, and time to drive the production, it doesn't well suit the beginning stages where ideas are vague and it's more interesting to ask "what if" than to commit to a single idea.

# Our Design Goals

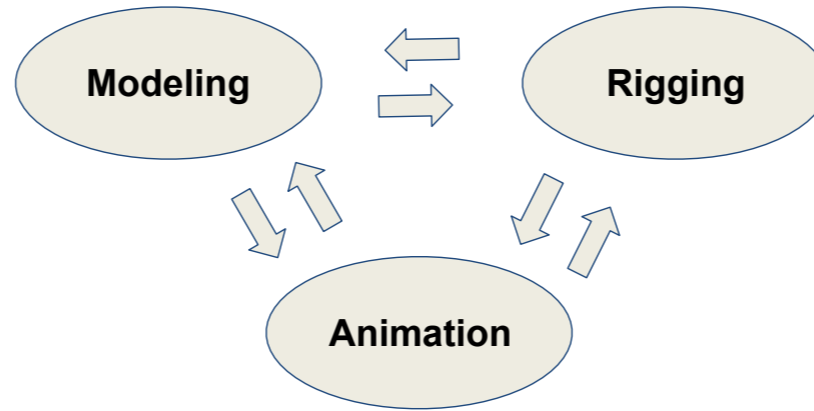


SA2015.SIGGRAPH.ORG

Sponsored by  

With that in mind we set out to design a tool that you don't need to be an expert to use. One that might help you explore your ideas, rather than to realize a well-established vision. We tried to make AniMesh as novice friendly as possible and we think we accomplished that. This came at the cost of being less suitable for high-end production tasks. For example, we do not cover issues such as footskate or non-skeletal animation. But these are primarily limitations of our implementation, and we believe with more effort in that direction this tool could be made suitable for big budget productions.

# An Alternative Workflow



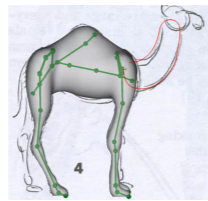
The core of AniMesh's workflow philosophy is around the idea of never locking the user into an old idea. We always allow you to re-model, with re-rigging happening on the fly. Animations can be rerecorded, moved/stretched, retargeted. And none of these actions break any of the other work you've done.



# AniMesh Pipeline



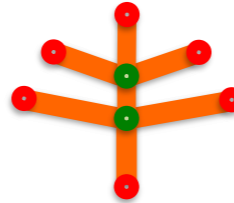
Modeling



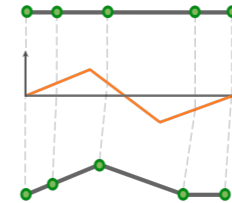
Motion Capture



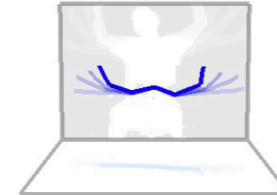
Skeletal Co-abstraction



Motion Retargeting

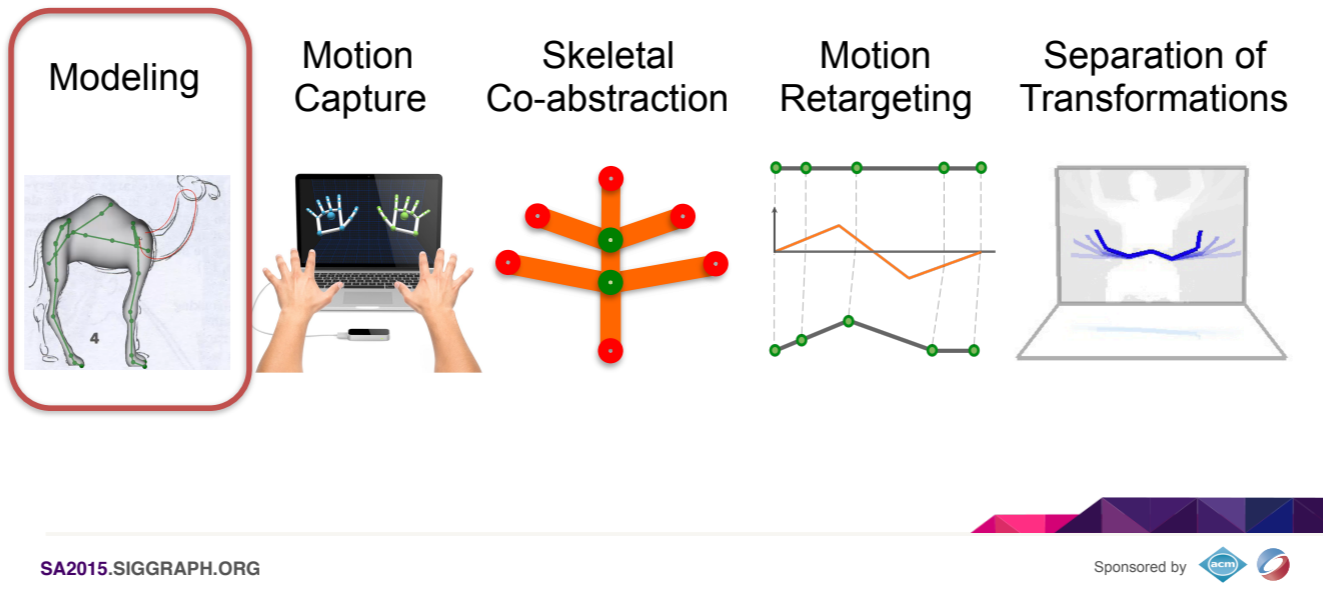


Separation of Transformations



Now I'll walk you through the technical steps that compose AniMesh. From creating a model, to mapping your body to it, and then transferring animation to it.

# AniMesh Pipeline



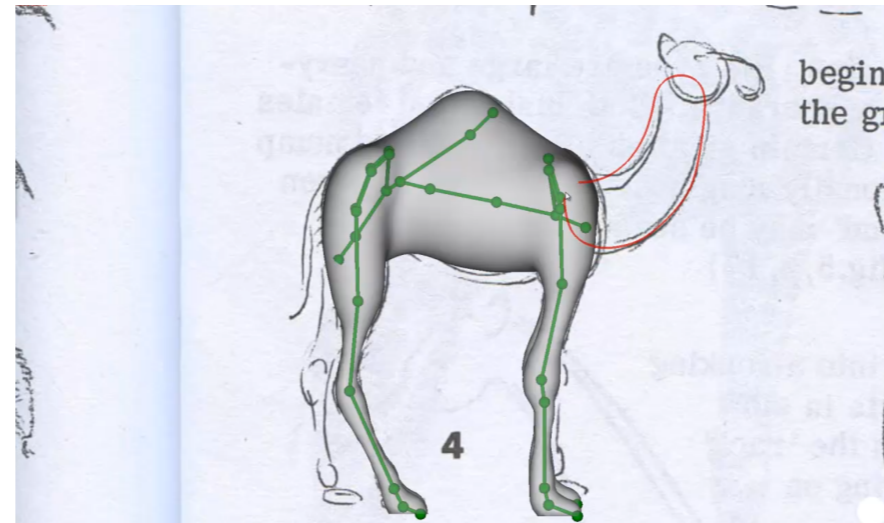
We start with modeling, creating a shape to work with throughout the process.

# Sketch-based Modeling





## RigMesh

SIGGRAPH Asia 2012



SA2015.SIGGRAPH.ORG

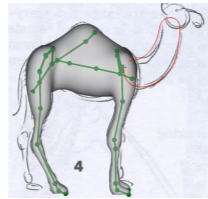
Sponsored by  

The default method of generating a model is through a sketch-based interface provided by RigMesh, a modeling tool designed the other authors of this paper. RigMesh allows 2D sketches to be expanded into 3d models automatically, with rigs generated for you as you go. Alternatively you can import rigged models from external files, and use those.

# AniMesh Pipeline



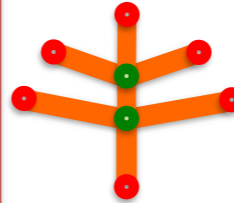
Modeling



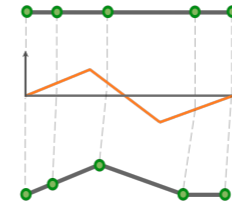
Motion Capture



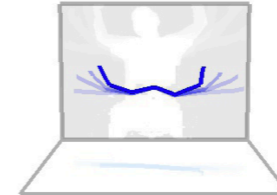
Skeletal Co-abstraction



Motion Retargeting



Separation of Transformations



After the user gets their model set up, they can control the model through a puppeteering process.

# Motion Input Sources



We can use commodity motion capture devices to input animations, and currently we have support for the Microsoft Kinect for upper-body skeleton control and the Leapmotion for hand control. But the system is designed with generality to accept any type of motion capture input, once the source has been configured.

# Sub-skeleton Selection

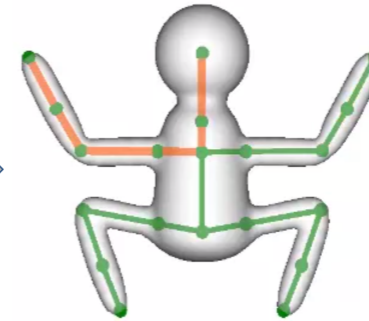
User Poses



Capture Geometry



Match to Model



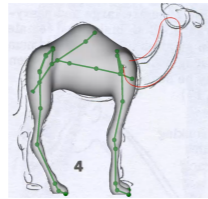
In AniMesh we don't make any assumptions about the topology of either the source or target skeletons. As a result we can't assume it's feasible to puppeteer the entire model at once, so we allow the user to select a subset of the character to control with their pose.

Here we see ming again choosing to control the head and arm of the frogman with both of his arms.

# AniMesh Pipeline



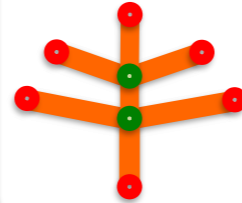
Modeling



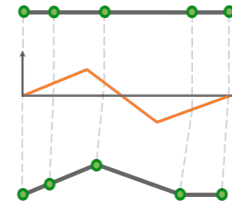
Motion Capture



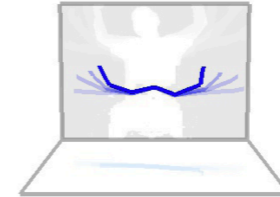
Skeletal Co-abstraction



Motion Retargeting

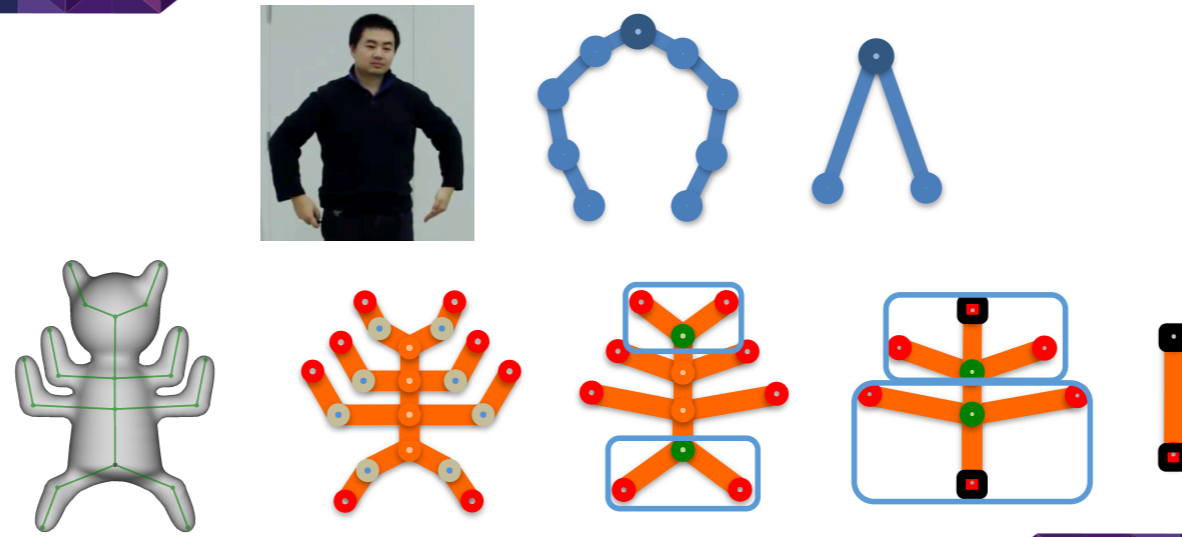


Separation of Transformations



Behind the scenes we implement the user/model mapping using an algorithm on the skeletons that's compares representative abstractions from both the source and target skeletons.

# Hierarchical Skeleton Abstraction



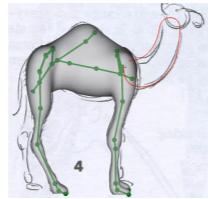
Specifically, we match the sub-skeletons by building a hierarchical level of detail tree where we iteratively simplify the structures of both the source and target skeletons, then match between them at each level of abstraction based on their geometries. On top you can see the simplification of the user's upper body skeleton. And on the bottom, the abstraction hierarchy of a 4-armed cat... thing.



# AniMesh Pipeline



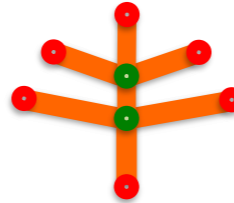
Modeling



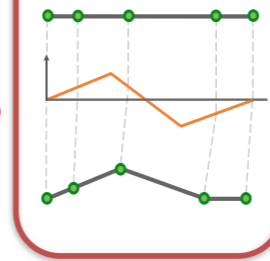
Motion Capture



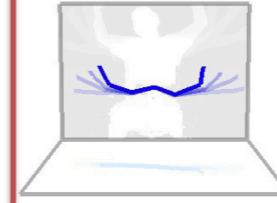
Skeletal Co-abstraction



Motion Retargeting

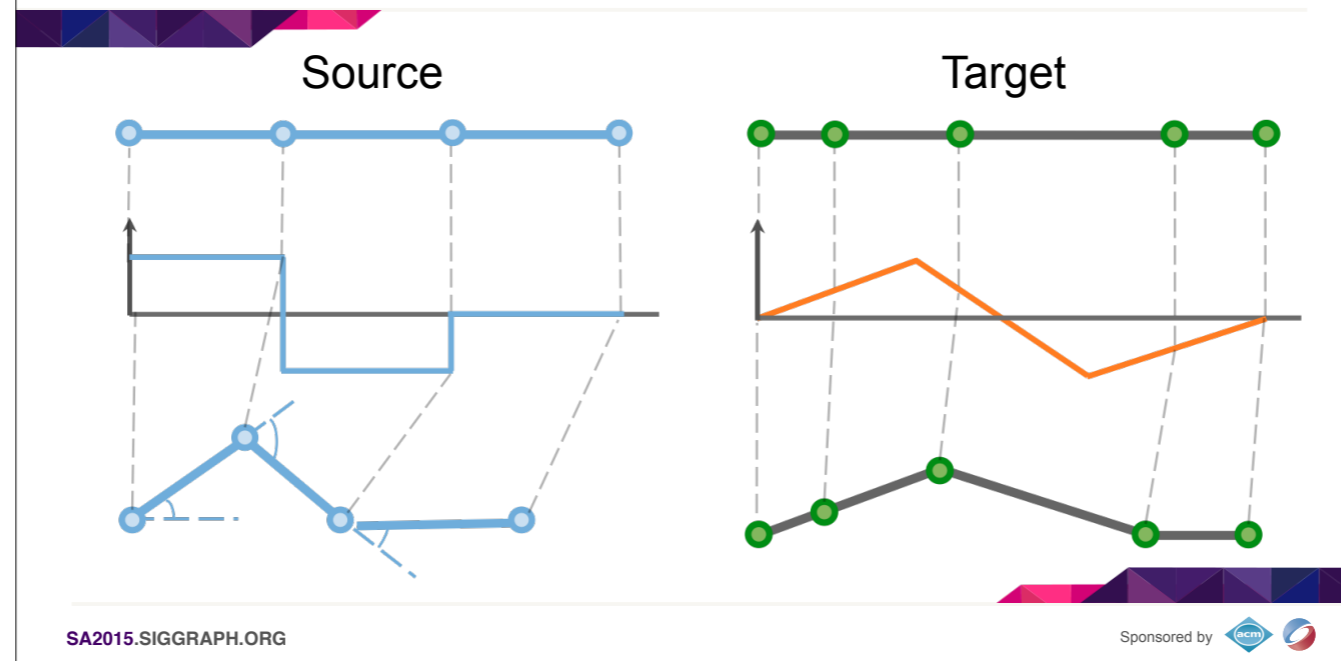


Separation of Transformations



Now that we've established a correspondence between the source and the model we need to transfer the performance between the two. Since the source and target skeletons can have arbitrary topologies, we assume that to map from one to another will require retargeting. This can be from the user to the model, or from one model to another. But since we always retarget, the "special case" of mapping animation to a different target topology, isn't a special case anymore.

# Motion Retargeting

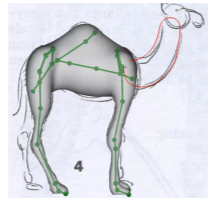


To support this retargeting we store the turning angle of the of the source during performance as an arc-length parameterization of the skeletal poly-lines. To be able to apply the turning-angle function to other joint structures, we linearly interpolate the turning-angle parameterization and re-sample it everywhere the target skeleton has a joint. While other interpolation functions may be theoretically fancier, we find that ours works very well and is extremely simple.

# AniMesh Pipeline



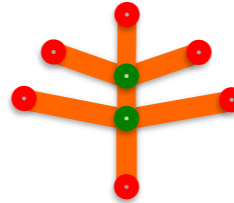
Modeling



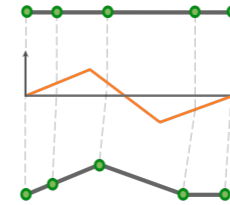
Motion Capture



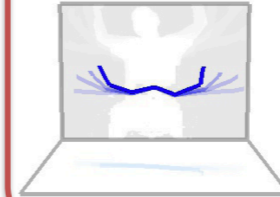
Skeletal Co-abstraction





Motion Retargeting



Separation of Transformations



SA2015.SIGGRAPH.ORG

Sponsored by  

But what if the user wants to change the model after they've already applied an animation to it? We need to make sure that the model can be updated without affecting the animation that's already been recorded.

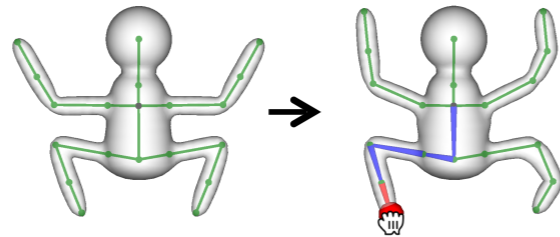
=====

In Animesh, we treat modeling operations differently from motion sequences.

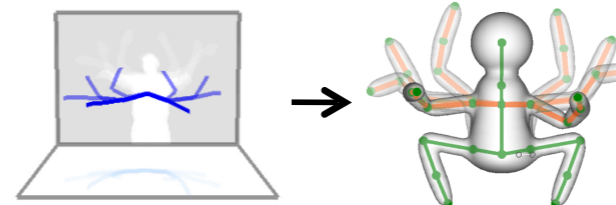
# Separation of Transformations





## Modeling transformations



## Performance transformations



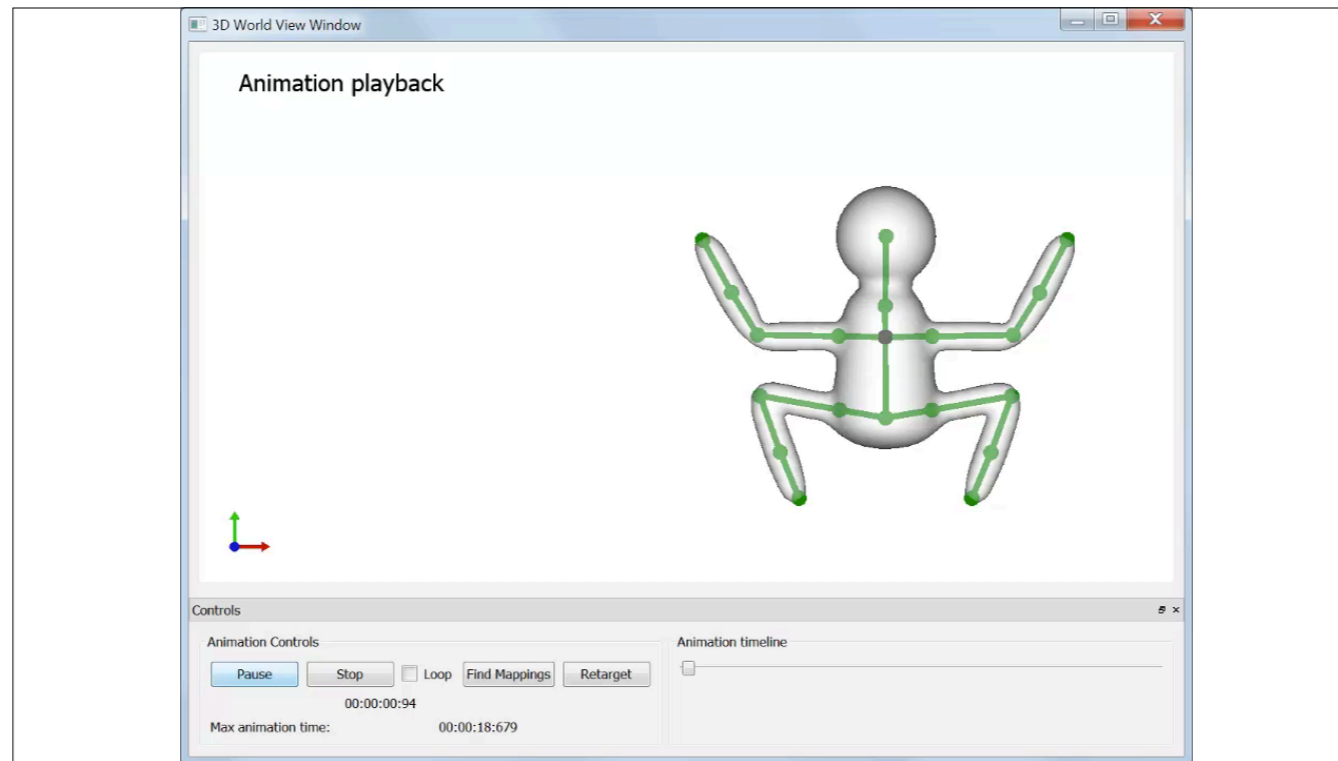
SA2015.SIGGRAPH.ORG

Sponsored by  

Since AniMesh's interface allows interactions like merging new appendages onto a skeleton which already has animations applied to it, we need to be careful to do appropriate book keeping to be apply those modeling changes, and ensure that existing animations are retroactively applied correctly to the newly modified shape. Our solution is to store modeling transformations separately from performance transformations and blend them together for playback. This way, since modeling and performance data are kept in different data structures, updating one will not destroy the other, they just to be re-merged back together again.

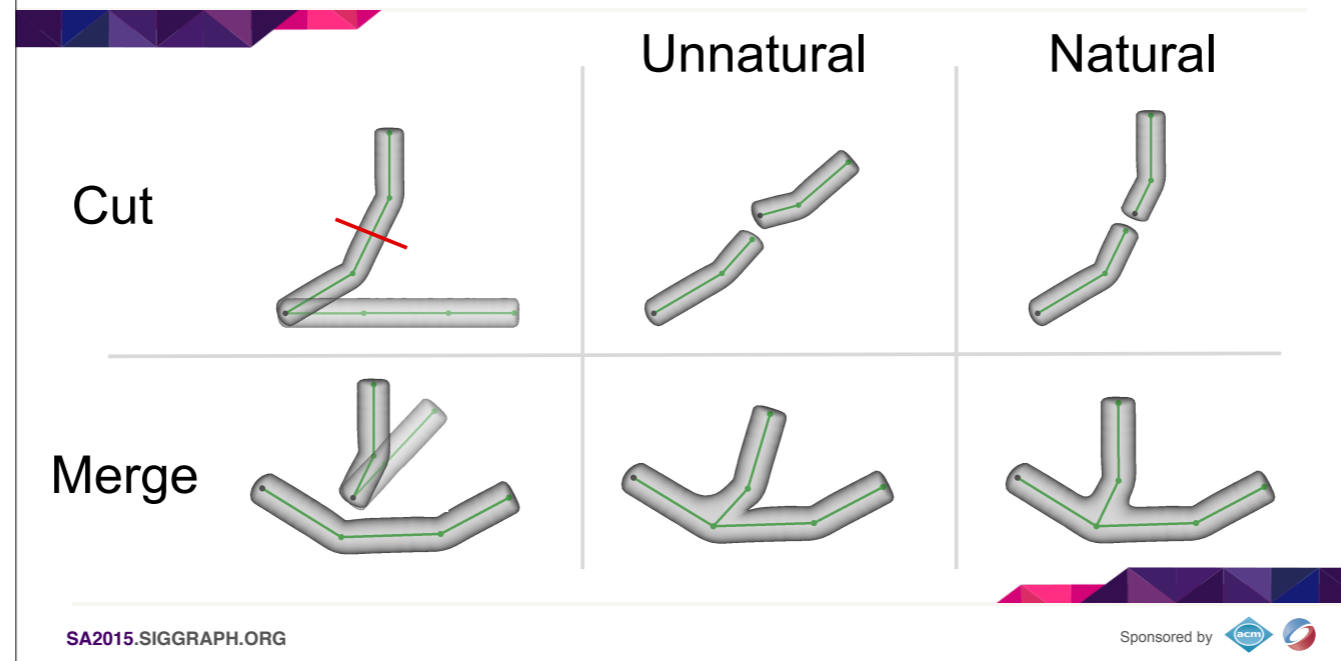
=====

We store the two types of transformations completely separate. This is how we make all modeling edits non-destructive, because they're operating on a separate datastructures from the performance data. And then regardless of what modeling edits have been performed on the model, playing back animation is consistently a matter of transferring the user's performance on to the current skeleton of the model.



To demonstrate how the separate transformations are then overlaid, we can take a model that already has an animation. Copy it and its animation over to a second model. Then we can reposition the model in various ways. But since these transformations are stored on the model, and have not changed the performance data at all, when it comes time to play back the animation, the animation is transferred onto the updated model automatically.

# Cut/Merge Handling



Unfortunately there are a couple cases where just separating modeling transformations from performance transformations doesn't get us all the way to the desired result. When cutting and merging shapes, we had to make a decision about the way to interpret these actions. Should edits happen with respect to the original shape, or with respect to the shape after it's had other modeling operations applied. We found through experimentations that, in most cases, re-applying any existing changes before doing the edit results in the most natural feeling operation.

On the left we have a skeleton being bent, and in the right two panels we have two different ways of applying merging or cutting. In the center column we don't reapply the accumulated skeletal changes to the edits, but in the right we do. We chose to go with the right-most versions because it more often represented the intent of our users.

## **Motion preserving shape editing**

Here we can see adding nodes to a skeletal chain during animation playback preserves its total rotation.

Then we can make a copy of the skeleton and merge it back on the other side to see that merging operations retain the orientations of skeletal chains from before the merge to avoid the discontinuities shown before. Similarly a cutting operation also consistently maintains the orientations of the two resulting parts.

The user can load pre-recorded animations from saved models and transfer them to other models using copy-paste functionality, even if the two models have different joint structures.

Then we can change the duration of animations as well as translate them through time to make the animations more varied and the final scene more interesting. Above you can see the Mantis is edited to have a galloping motion, even though the only input animation was a single-jointed skeleton flapping.

## **Interleaving modeling and animation for complex scenes**

We can then load a more detailed model, and reuse the animation from other characters by specifying a mapping

In this case we're mapping the arm of the frog-man the arm of the squirrel to transfer the motion identically between the two models, but any mapping would work, and here you can see we're making a less obvious mapping by assigning the motion from the leg of the frogman to the neck of the squirrel.

But, again, motion can be imported from a number of source and here we're using the LeapMotion to animate the ears using our fingers which you can see rendered in the top right.



## **Results from first-time users**

Here are some animated shapes made by first-time users of our system

The shapes in this video were made using only basic sketch/cut/merge modeling operations. And motions were all sourced from a very simple pre-recorded animation of a 90-degree rotation of a single bone.

From modeling to animation, each of these scenes were created in under 30 minutes, by users who received only 20 minutes of instruction.

# Recap



- Natural Interfaces
  - Easy for novices
- Non-destructive edits
  - Allows exploration
- Interleaved pipeline
  - Promotes iterative design

Looking beyond the technical features of AniMesh, we believe the system as a whole offers a new perspective on animation. One that won't necessarily suit every need, but that will instead offer options in situations that are otherwise poorly served. For the novice, or someone looking to explore or iteratively refine their ideas, AniMesh offers a more natural approach to animation.

----- Meeting Notes (11/3/15 18:27) -----

Say something about each of the 3 points

everything with respect to 2-d space for natural interface

Thank You

THANK YOU!