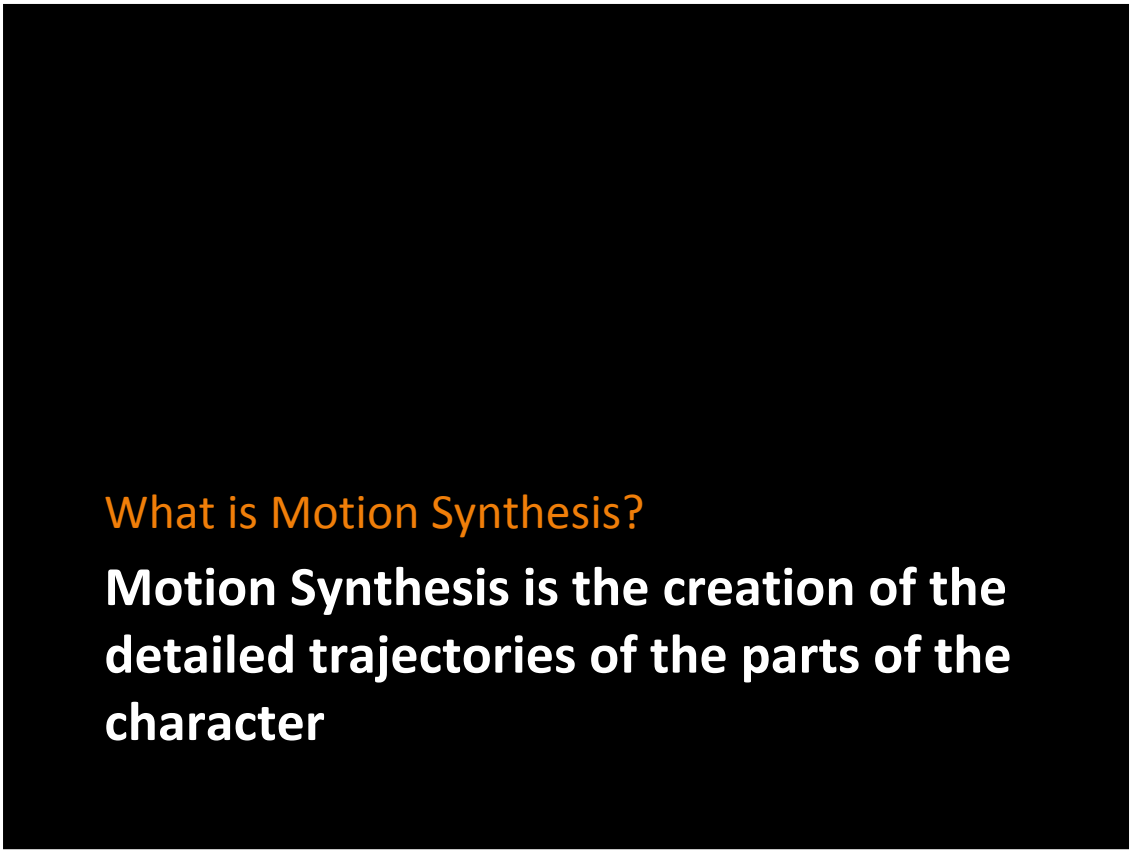
# Synthesis by Example

## Connecting Motion Planning and Example-based Movement

Michael Gleicher

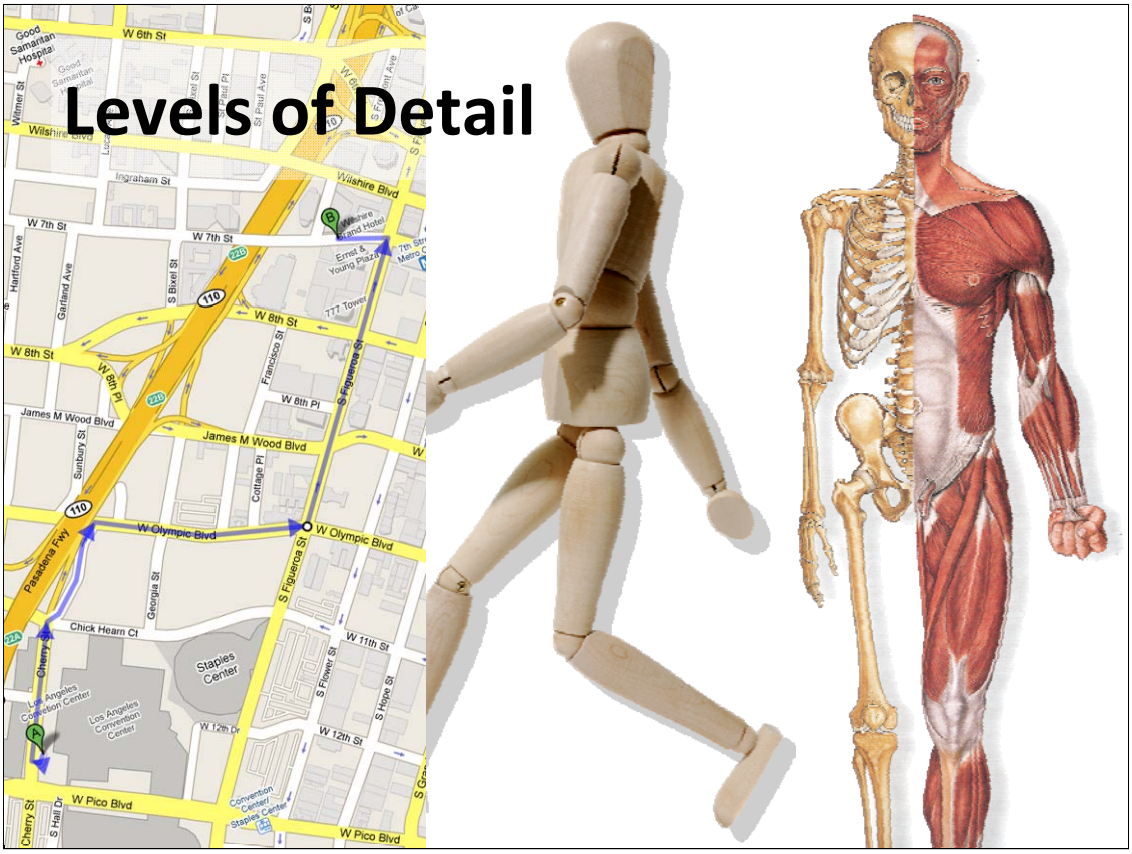
Dept of Computer Sciences
University of Wisconsin - Madison




In this talk I will connect the Motion Planning ideas discussed this morning to Motion Synthesis ideas in Computer Animation.

By Motion Synthesis, I mean creating the detailed trajectories of the various parts of the characters we're animating.

This sounds like Motion Planning – I differentiate the two by the level of abstraction that we're considering.




When we're considering how a character meets its goals – for example to get from place A to place B – we can consider many different levels of abstraction.

We might consider the route taken, the rough form of the locomotion, or even the specific details of how the movement is made.

In computer animation, we have similar levels of detail to consider.

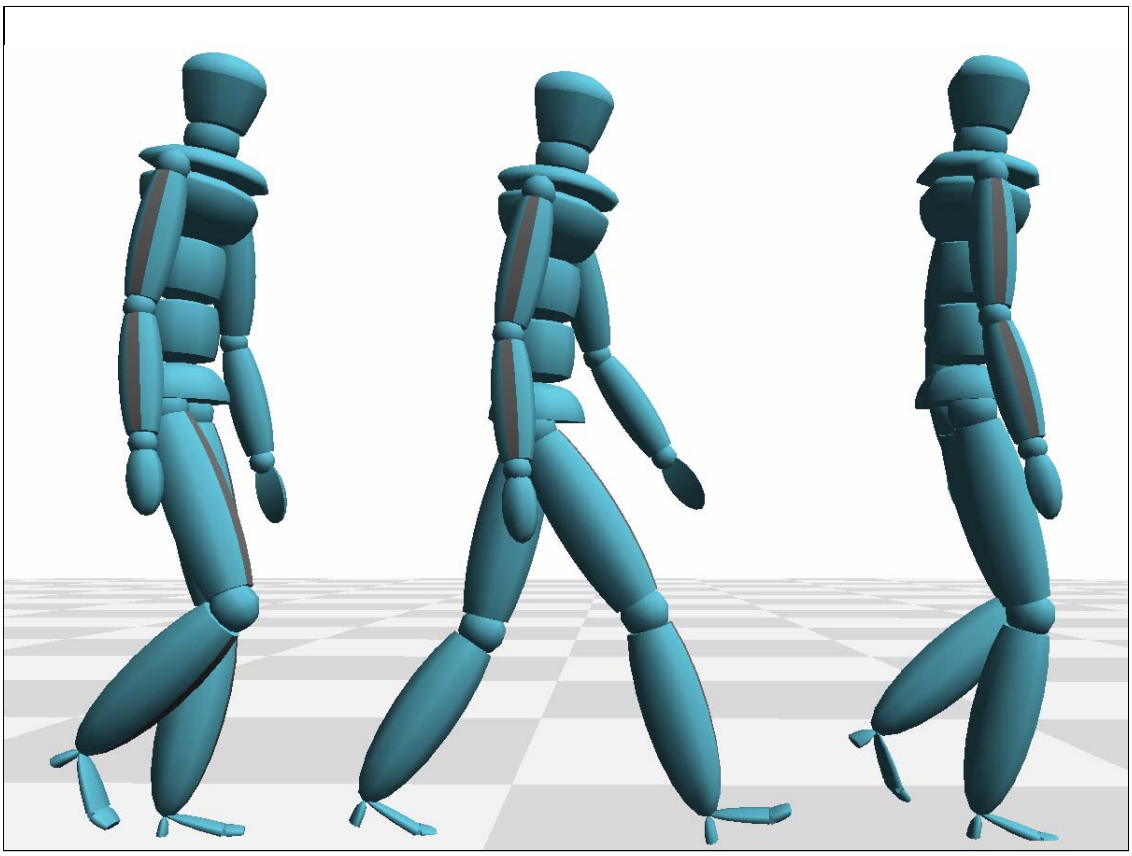
The high-level goals, such as places a character needs to get to.
A mid level representation – such as how the characters limbs move to make the motion.
Or even the specifics of how the character appears.

Each of these has a different level of detail to consider:
For high level goals like navigation, we might consider just position and orientation.
For movement, we probably want to think about the characters as an articulated figure.
And the details of the character might be a mesh.

Image credits:
Google Maps
Dora Mitsonia (stock.xcng)
My anatomy book

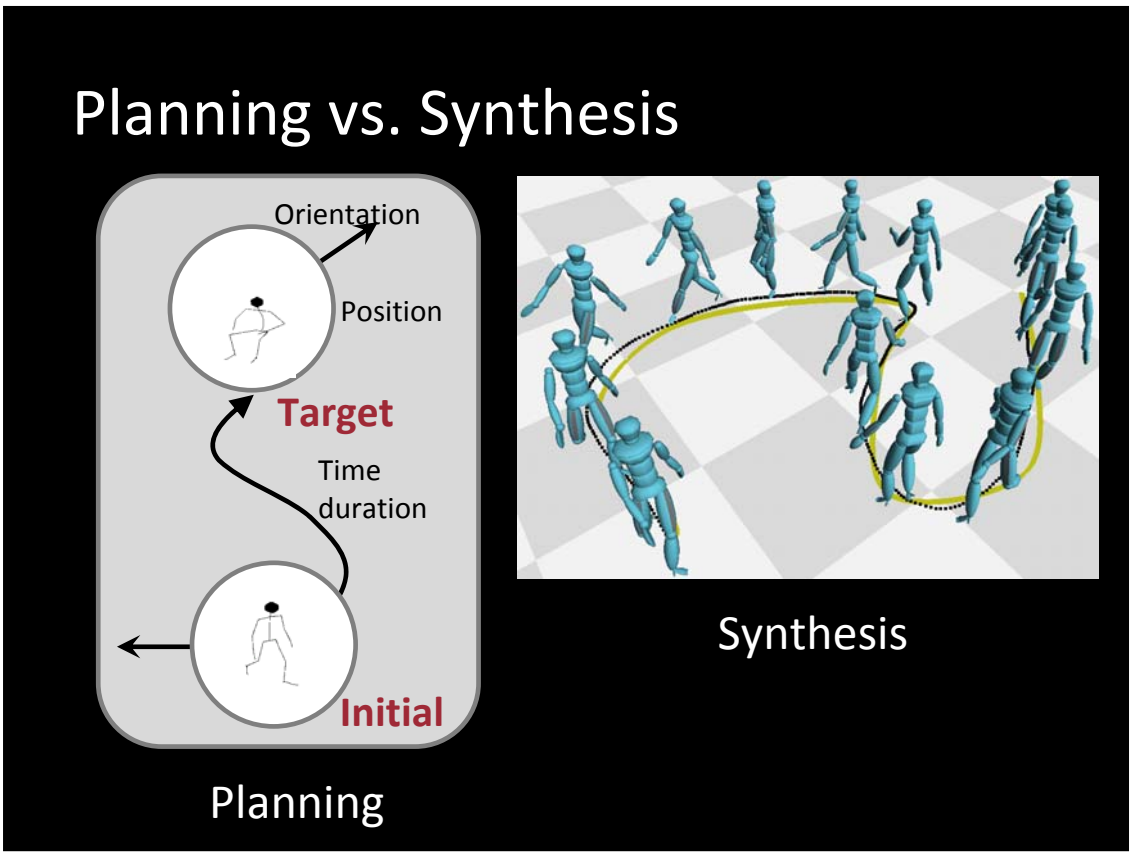
In computer animation, we tend to think about human-like character motion in terms of an articulated figure.
That is, as a skeleton of rigid bones connected by rotational joints.

Image Credit: Lynn Lancaster, stock.xchng



So, a motion, in terms of this talk, is the trajectory of the set of bones of the character over time.

It's usually represented as a root position and a set of joint angles, but the details of that don't matter for this talk




Motion planning and motion synthesis are really the same problem: figure out how to make the character move to achieve its goals.

We use two different terms to distinguish between the level of abstraction considered.

Motion Planning usually refers to high-level goals and abstract representations.
Motion Synthesis usually refers to generation of details.

Key Point:

**To get (good) detailed movements that meet high level goals, we need to combine approaches**

**Neither synthesis or planning methods can do it alone**

While planning and synthesis are similar problems (or even the same problem at different levels of abstraction), their approaches and their strengths and weaknesses are different.

Planning methods are good at dealing with the complexity of high level goals, through their use of abstract representations – but they aren't as good at getting the low-level details right to create nice movements.

Synthesis methods are good at getting the low-level details right to make good movements, but they have problems scaling to the complexity of high-level goals.

The solution is to combine approaches: to create hybrid or multi-level approaches that break the problem at different levels of abstraction and use appropriate methods for each.

To get good character motion that meets high level goals, we need to combine the apporaches.

The key point that I want to make in this talk is that neither the planning approaches, or the synthesis approaches, can solve the problem of creating high-level character control by themselves. We really need hybrid approaches that mix the two.

I will show you some hybrid approaches, and Claudia (the next talk) will discuss some other ones.

Specifically, I will talk about example-based motion synthesis methods that use motion capture data to create the movements.


Getting the details of movement right is important.

Human motion is incredibly complex and diverse. Even for something common – like walking – we all do it differently, and do it in different ways at different times.

By looking at someone's movement, you can figure out their mood, their intent, who they are, their energy level, etc.

While its possible to take on the complexity of motion algorithmically – for example to build procedural or physically-based controllers for walking – such approaches have problems scaling to all of the diverse things that people do and ways that they do it.

Photocredit: everystockphoto -> Flickr ->user atp_tyresus


Example-based approaches to creating character motion are attractive because they avoid the problem of trying to model all the subtlety, complexity and diversity of human movement.

Instead of modeling the movement, we record examples of movement (usually using motion capture) that have the properties that we need.

Photo Credits: Wisconsin State Journal, Zoran Popovic

## Example-Based Synthesis

***Capture*** the detail, subtlety and complexity

Good News:

We don't need to model all the complex things!

Bad News:

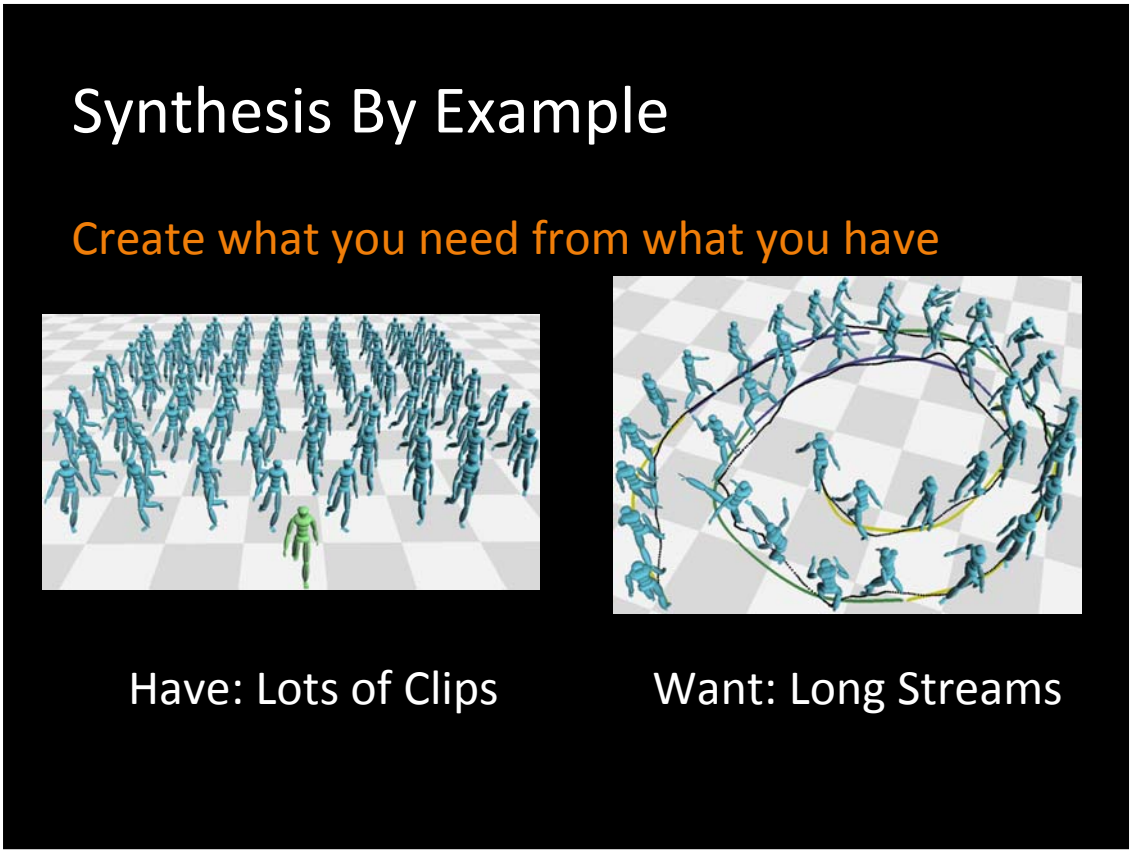
We don't have a model to generate what we didn't capture!

For specific cases, we can build algorithmic models – for example generic walking.

But if you want to get a wide variety of movements (running, jogging, sneaking, cartwheeling, …) or a wide variety of styles (specific personalities, moods, …) at best you would have to devise models for each one. For many of these, building models is really hard.

So capturing examples of movements is an attractive alternative.
We can get all the specialized motions by telling an actor to perform them.
And use generic algorithms to use these motions as we want.

There are two downsides to the example-based approach:
1)We need to be able to get examples of what we want
2)Because we have a very limited model of what the motion is, its very difficult to change it into something else




The ability to change the motion is a big deal though.

Unfortunately, in many cases, we don't know what the character is going to do ahead of time.
If the character is controlled by a player, or is autonomous, we need to generate the movement at run time.

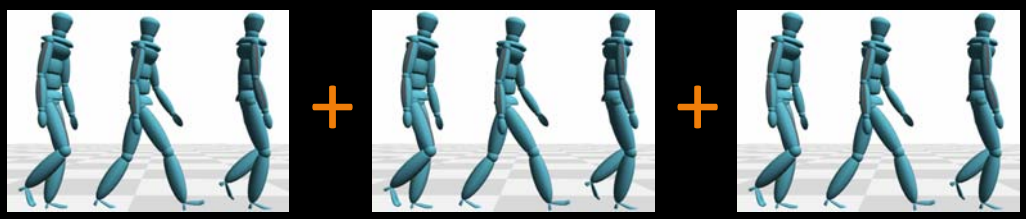
So, we need mechanisms for creating the movements that we need based on the examples that we have

This strategy is called Synthesis-By-Example

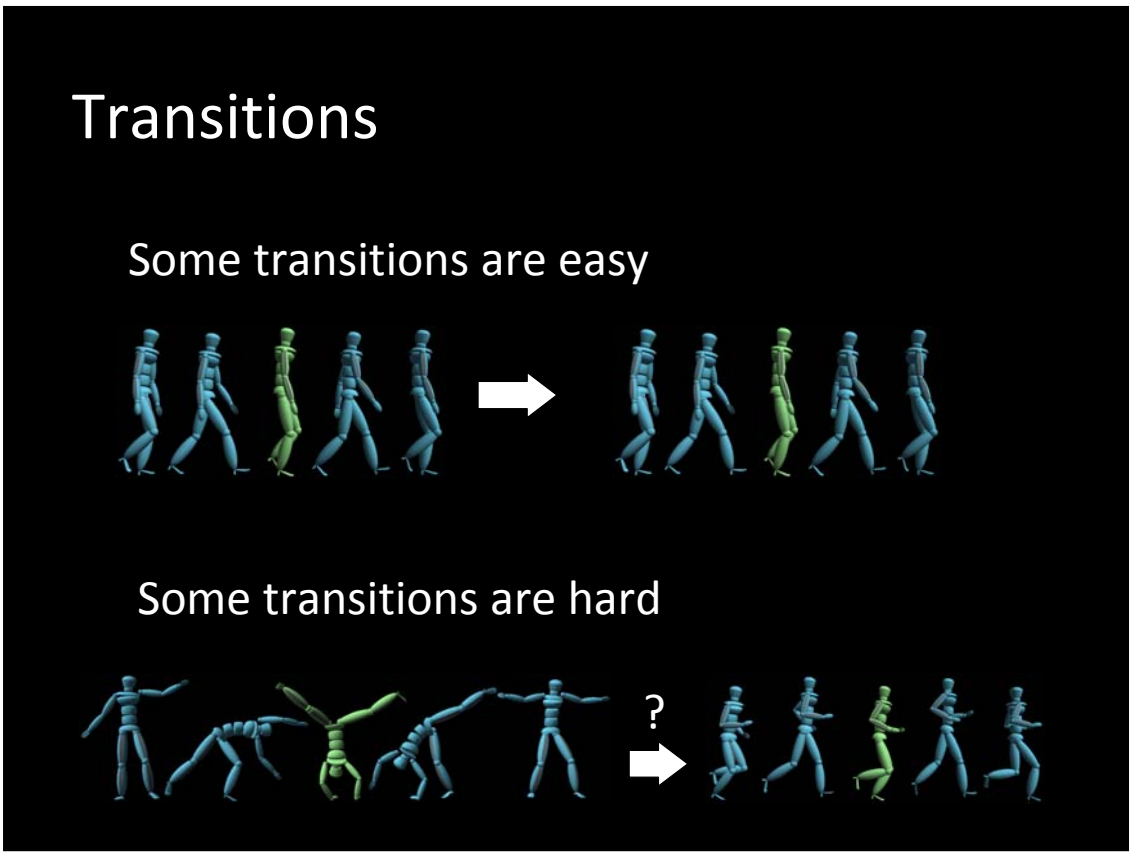
I am going to consider one important case:
How do we get long, continuous streams of motion – such as jogging from one place to the next – when we can only capture specific clips of movement.

# Concatenation

Put clip after clip after clip …



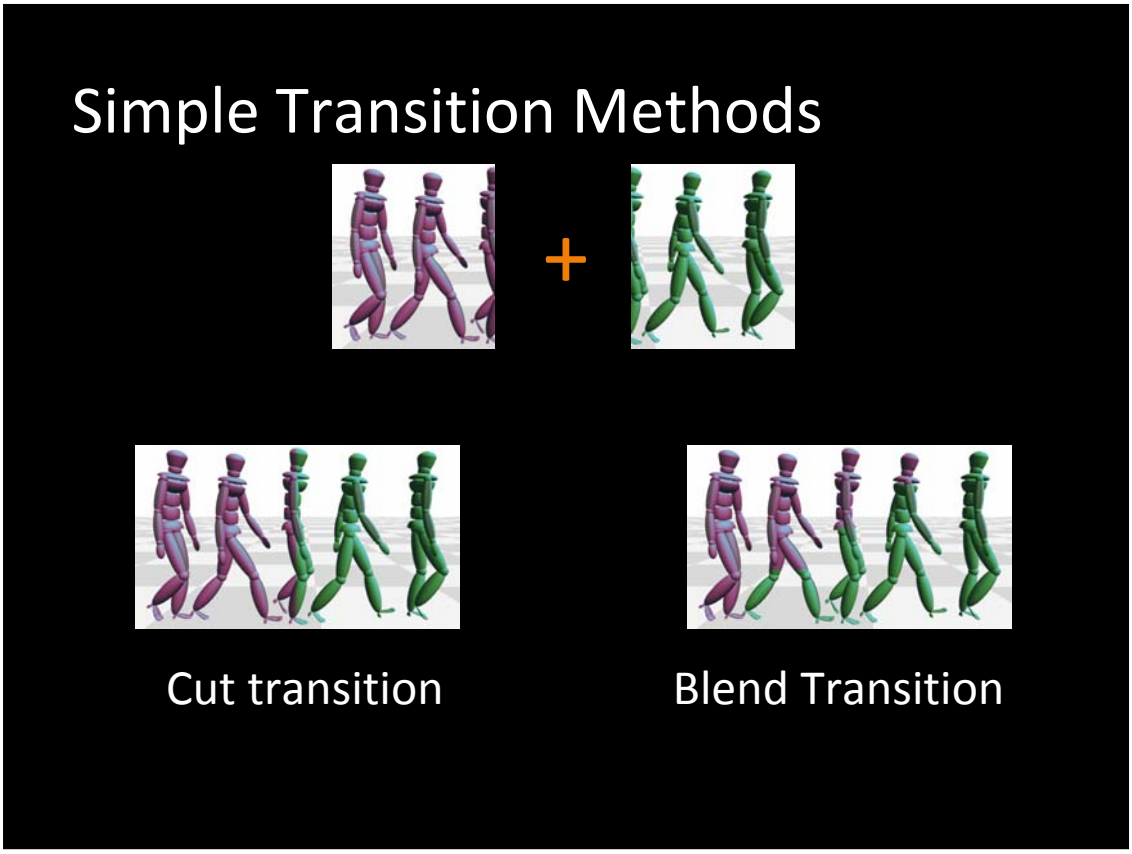
So, the basic idea is that we'll make long motions by putting a sequence of short motions together




The problem is that it isn't always easy to figure out how to get from the end of one clip to another.

If the end of one clip is similar to the beginning of the next, then creating a transition between them is easy.

If the end of one clip is very different from the beginning of the next, then creating a transition can be really hard.




If the end of the first clip is similar to the beginning of the next clip, we can use very simple methods to make the transition between them

If the clips are really similar, we can just put one after the other.
However, since the human eye is really sensitive to high frequencies, the motions must be really similar in order for them to match up.

For motions that aren't so close (basically any real motions), we can blend between the two motions.

But this only works if the motions are similar

## Concatenation-Based Synthesis

Key Idea:

- Only create transitions where simple transitions are likely to work

Historically (in practice, particularly games)

- Craft motions to have easy transitions

In Research (starting around 2002)

- Find metric to automatically determine what motions are "close" enough for transitions to apply

Kovar et al, Arikan&Forsyth, Lee et al. – All SIGGRAPH 02

The key to synthesizing motion by example has been to make effective use of simple transition schemes.

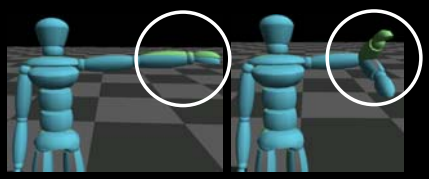
Rather than trying to develop more general transition schemes, successful methods have determined which motions the simple transition methods will work for.

In practical applications, particularly computer games, this has been done by hand-crafting the motions so that they fit together.

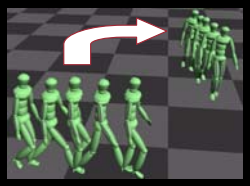
In 2002, several groups independently discovered that this process could be automated. The main idea is to define a metric that can determine what motions can have transitions made between them, and then opportunistically create the transitions.

# Determining potential transitions

- Need to account for derivative continuity
- Joint angles are difficult to compare directly
  - Effect of perturbation (e.g., rotate shoulder) depends on pose



- Need coordinate invariance
  - Different camera ≠ different motion!



A key element is a metric that determines if motions are similar enough for simple transitions to work.

There are a few key challenges in defining the metric functions:

1)The metric must consider the derivatives, so that concatenation doesn't cause inappropriate velocity discontinuities. We've found that its important to match derivatives at different temporal scales.

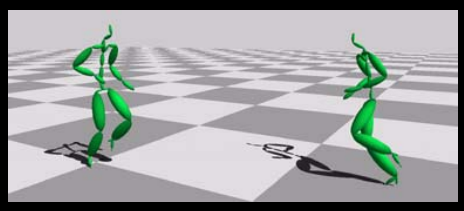
2)The actual parameters of the character can be difficult to compare directly since different joints have different amounts of effect, and the amount of this effect depends on the configuration.

3)Its important that the metric be invariant to things that do not matter. Most significantly, the metric should be invariant to global position and orientation.

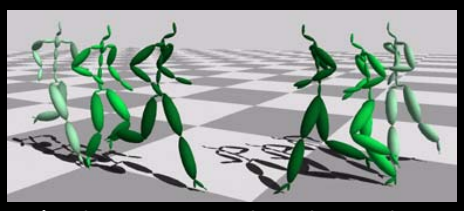
# What is Similar?

## Factor out invariances and measure

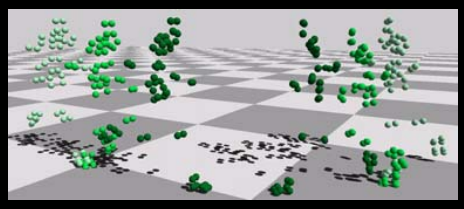
1) Initial frames



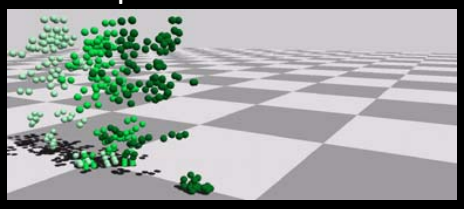
2) Extract windows



3) Convert to point clouds



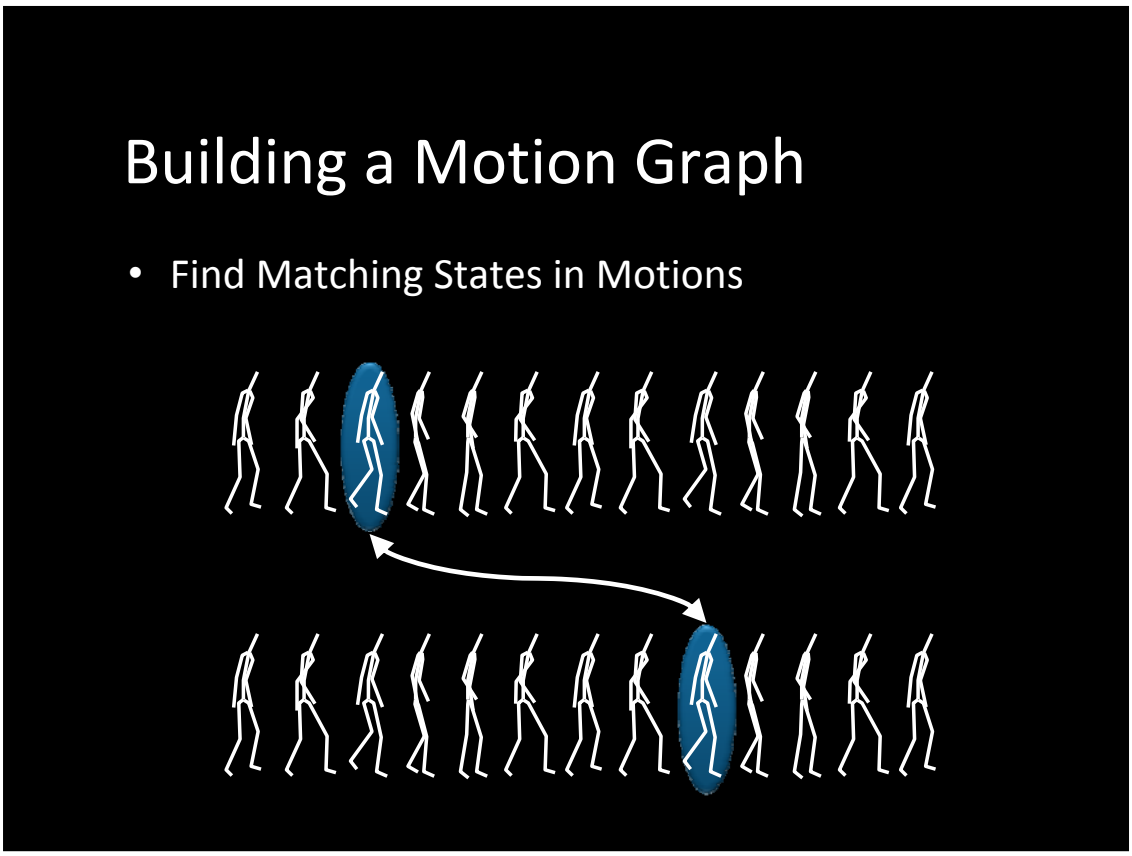
4) Align point clouds and sum squared distances



Kovar et al 2002, and others – see Kovar's thesis for discussion

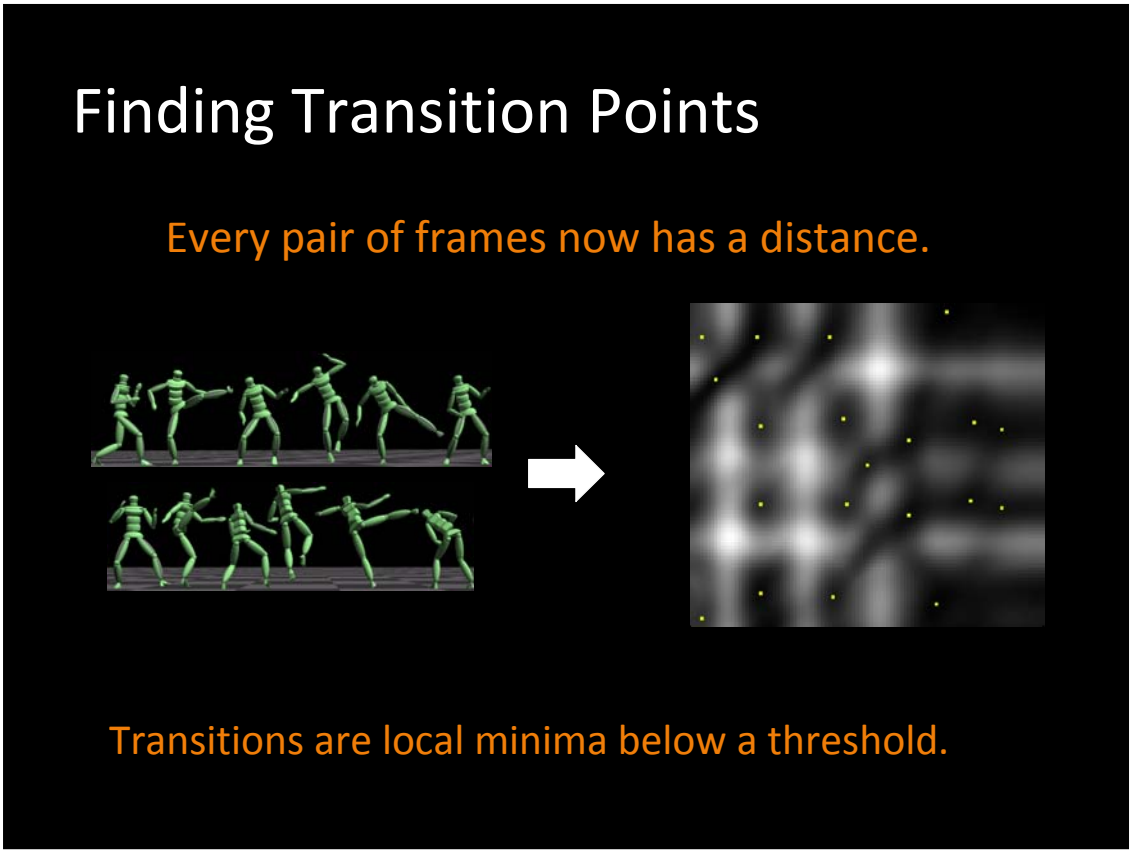
Different groups have proposed different metrics, and there have been some recent papers that compare the various approaches.

The method that we have developed, first presented in SIGGRAPH 2002, but used in a lot of our other work works in 4 phases:
For the frames we want to measure
1)We first look at a window of frames around that frame (to provide for derivative matching)
2)We then determine the positions of the joints as a point cloud for the entire window
3)We then do a rigid alignment of these point clouds
4)We then take the sum of squared distances between the corresponding points




So the metric allows us to find poses that are similar enough that we can make a transition between them.
Or possibly around them if we're doing blending.

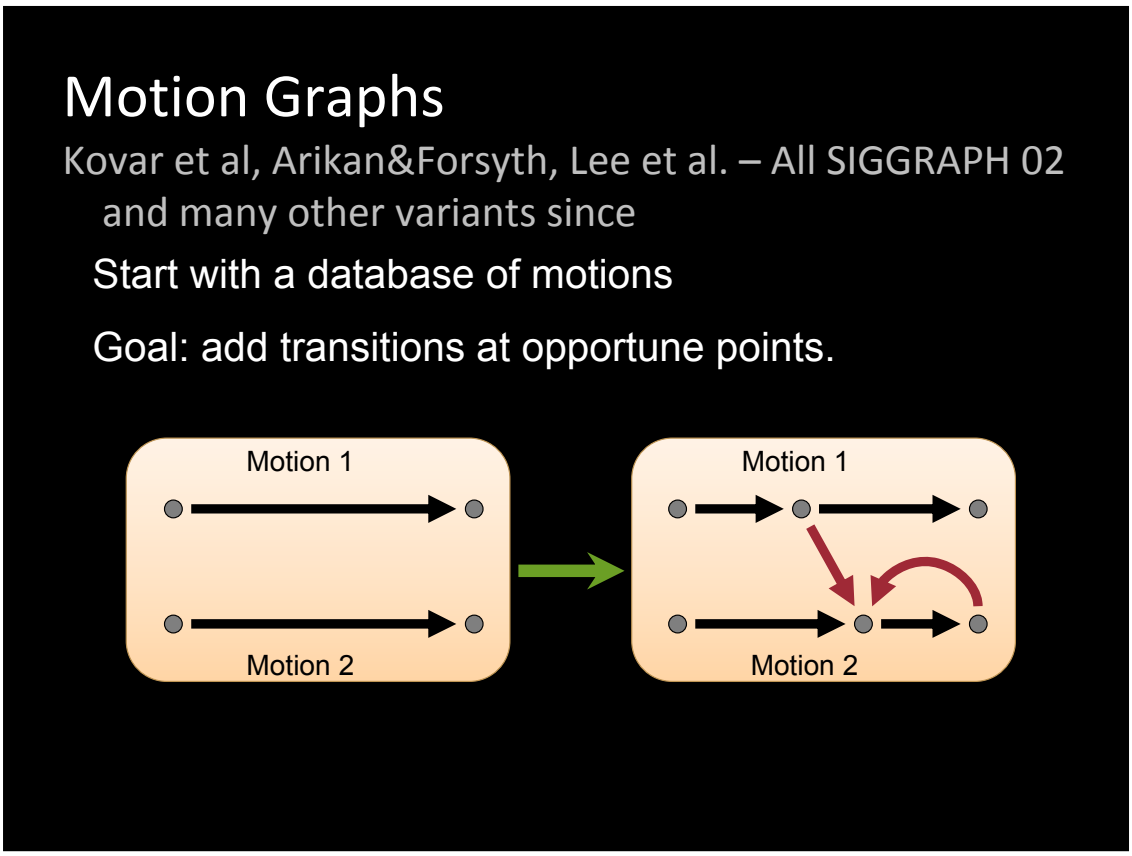
This means that we can go looking for possible transitions.




Once we have a metric to determine if a transition can be made between two frames, we can apply this metric across the database to find all possible transitions.

If the metric is below some threshold, then we can make the transition.

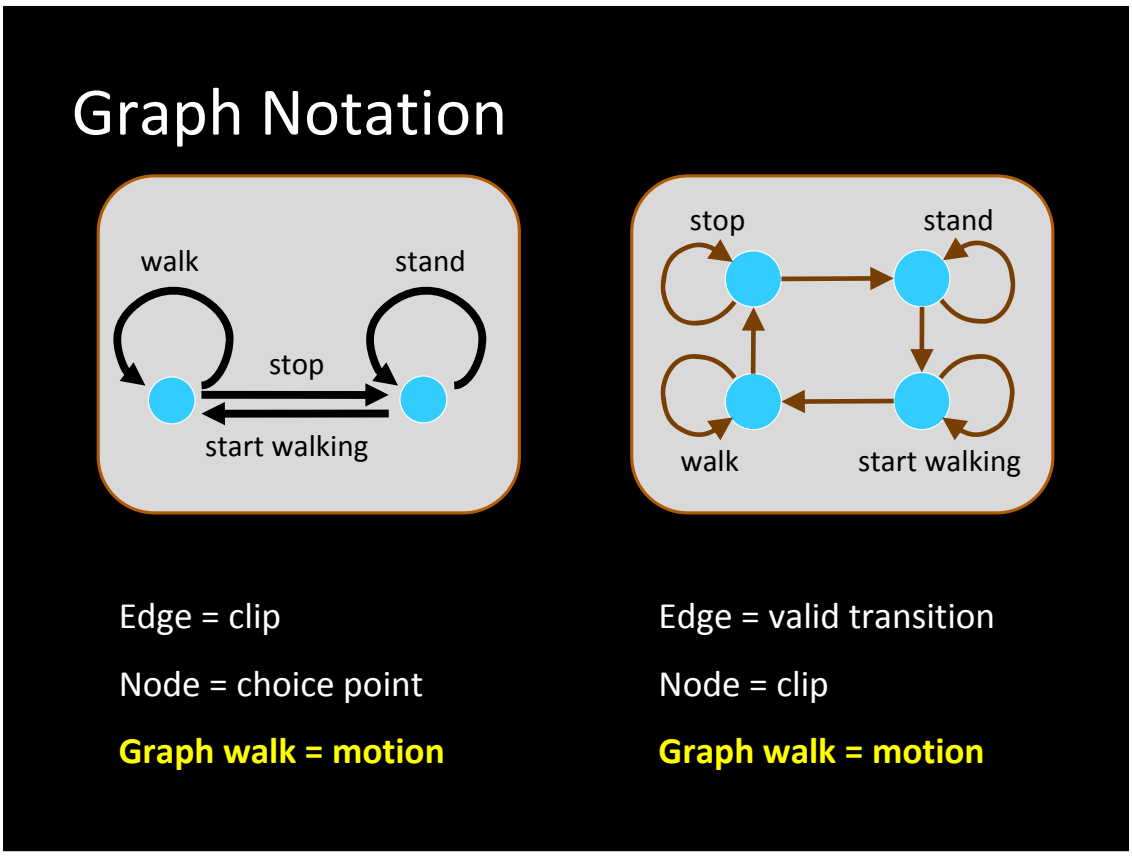
To avoid redundancy, we usually reduce the number of transitions. For example, we have looked for local minima of the metric function.




Once we're able to figure out what transitions we can make over a set of motions, we can start connecting the motions.

We can build up a graph of possible connections between the motions, turning the set of initial clips from a list of sequences into a connected graph of movements (including transitions).

We call this structure a Motion Graph.



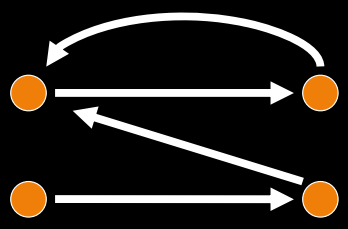

There are two different notations used in the literature – this causes no end of confusion since they are basically the same thing.

In one notation , the nodes are choice points and the edges are motion clips (potentially transitions).
In another notation, the nodes are motions and the edges are possible transitions (potentially direct successors).

But in both cases the key point is that a walk on the graph is a valid motion (a sequence of clips that “fits together”)

# Motion Graphs

Idea: automatically add transitions within a motion database



Edge = clip

Node = choice point

Walk = motion

Quality: restrict transitions

Control: build walks that meet constraints

So motion graph construction opportunistically builds a graph of good transitions from a database of movements.
We do this as a pre-process.

Then, to generate motions, we select walks on the graphs. These correspond to movements – sequences of clips that concatenate together correctly.

Motion generation from a motion graph provides good quality results because at any time its either playing an original clip, or a transition (that was determined to be good).

Motion generation from a motion graph provides flexible control because we can generate a wide range of motions of arbitrary length by creating walks on the graph.

# Using a motion graph

## Any walk on the graph is a valid motion

- Generate walks to meet goals
  - Random walks (screen savers)
  - Interactive control – pick an appropriate choice
  - Search to meet constraints
- There are many ways to use the graphs

Since any walk on the motion graph will generate a valid motion, the problem of generating motion has been reduced to searching for an appropriate walk on the graph.

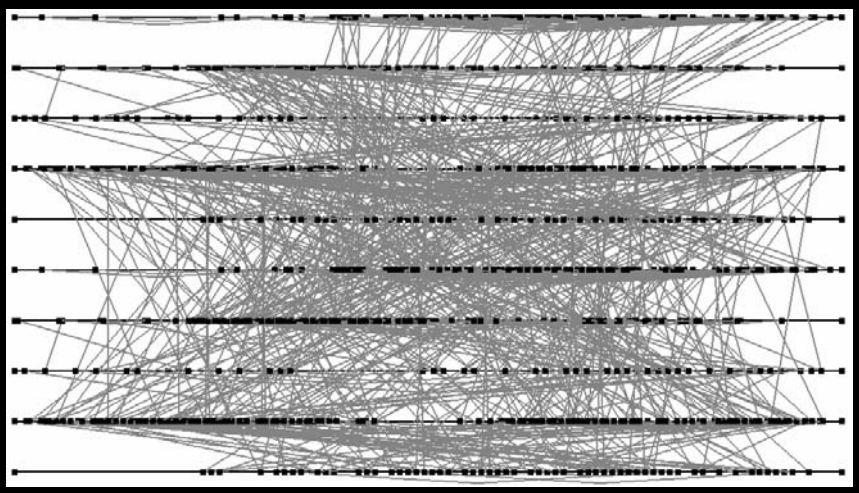
Over the years, there have been a wide variety of ways discussed for how to use a motion graph.

The two most important categories of methods are:
1)Local control – where given the input (such as a direction from a game controller, or a goal) pick the next choice based on a limited (local) view of the graph. For example, you might greedily choose the option that best moves you towards the goal, or look ahead some number of frames.
2)Search – you can do a search procedure / combinatorial optimization to find a walk on the graph that best meets a set of desired goals.

The appropriateness of these strategies depends on the application, but also, the motion graph being used.

# Structure of Motion Graphs

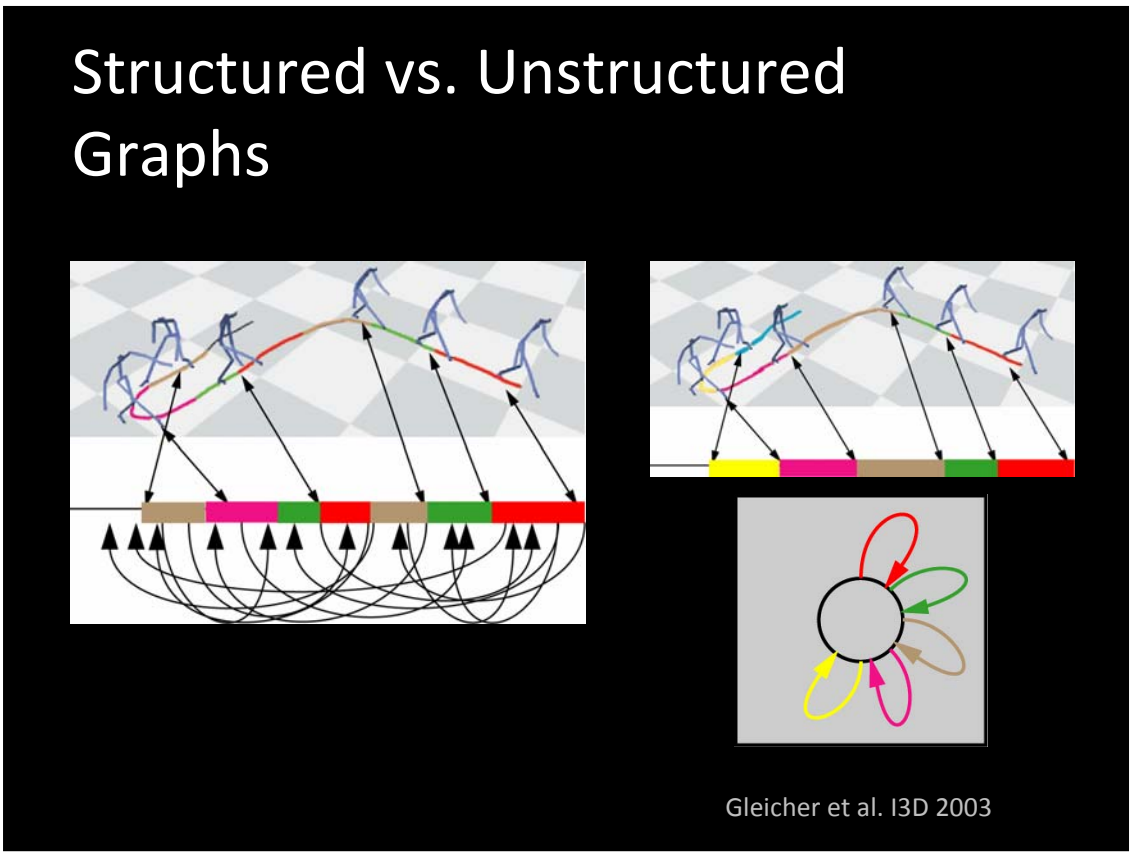
Opportunistically built graphs can be hard to search – especially for quick control



For automatically constructed motion graphs, there may not be any control over the structure of the graph.

For these complicated graphs, search is required to find ways to meet goals, since local methods might get stuck in parts of the graph that take long ways to get out of – assuming we have pruned the dead ends.

At any given time, the number of options may be limited, and the sequence of choices required to do something might be long.




In computer games, they have long used specially designed graphs.

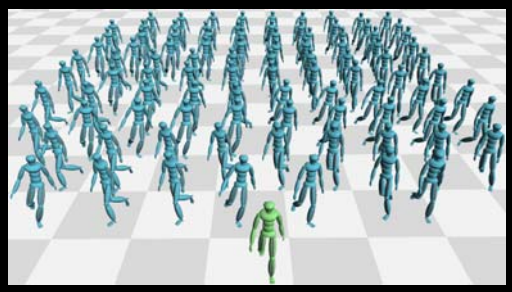
These graphs usually have a small number of highly connected nodes, with many options at any time.

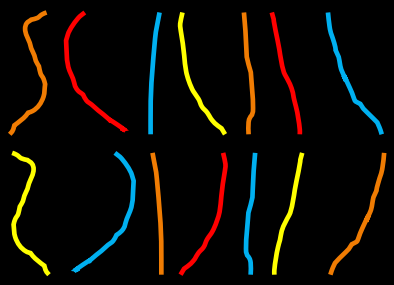
These graphs are good for interactive control, since at almost any time, many options are available. This also makes simpler search methods, such as greedy methods, more likely to work well.

We had an approach for automating the construction of these structured graphs.

# Search to goal

Search for a walk on the graph (sequence of clips) that meets the goals



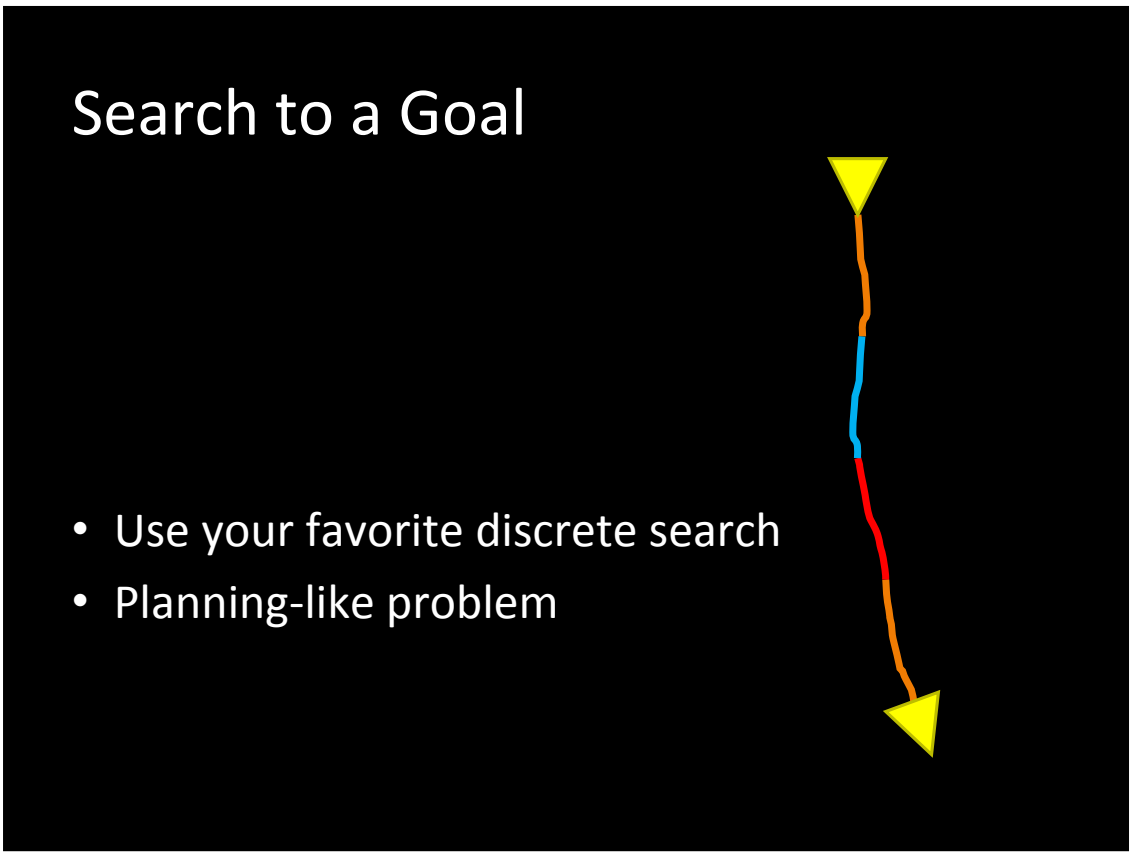


So lets consider using motion graphs for a motion planning problem.

We want to get a character from a start position to an end position.

We have a set of motion clips, each one moves the character along the floor in some way.

The graph tells us which clips can follow another clip.

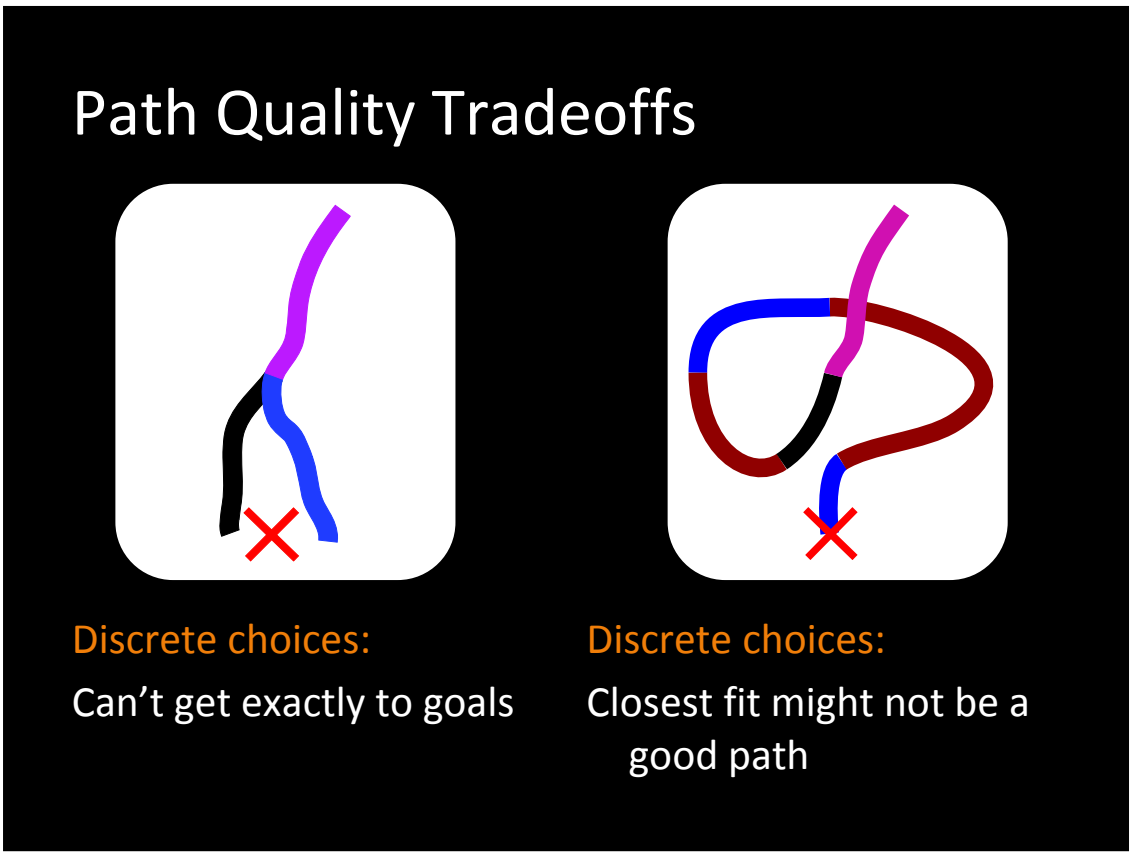



Our problem is to pick a sequence of these clips – a walk on the motion graph – such that the motion gets us from the start to the end.

This is relatively straight forward – it can be implemented with your favorite search method.

One problem is that if the goal is far away and complicated, the combinatorial explosion of possibilities makes this inefficient.

Basically, to go any distance, thinking a step at a time is just too much.



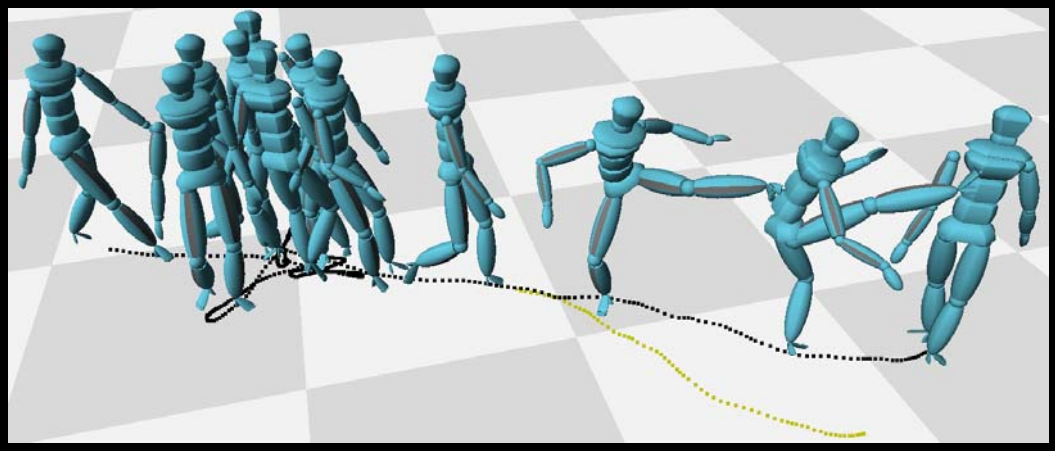

The discrete nature of the search also leads to another problem:

You might not be able to get exactly to the goal.

Or worse, there might be a way of getting exactly to the goal, but it might not be a desirable path.

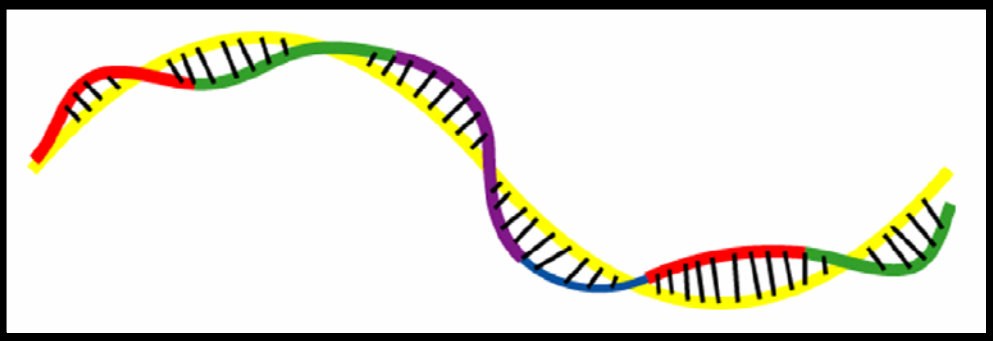
Making intelligent tradeoffs between path quality and reaching the goals can be challenging, especially when you are looking at long sequences of steps.

# Bad paths happen



So here, the character did get to its goal position and orientation, but it took a kindof circuitous route.

# Path following works better



In practice, finding a sequence that follows a path works out better for a number of reasons

1)You have local information along the whole process
2)You can determine the error along the whole path, rather the just one point
3)You can use a higher level process to determine a good path

4)In fact, you can often use a greedy “search” to follow a long a path (if you have a well-structured graph). This is effectively using the graph as a controller to drive along the path.

## Multi-Level Solutions

Different methods for different aspects

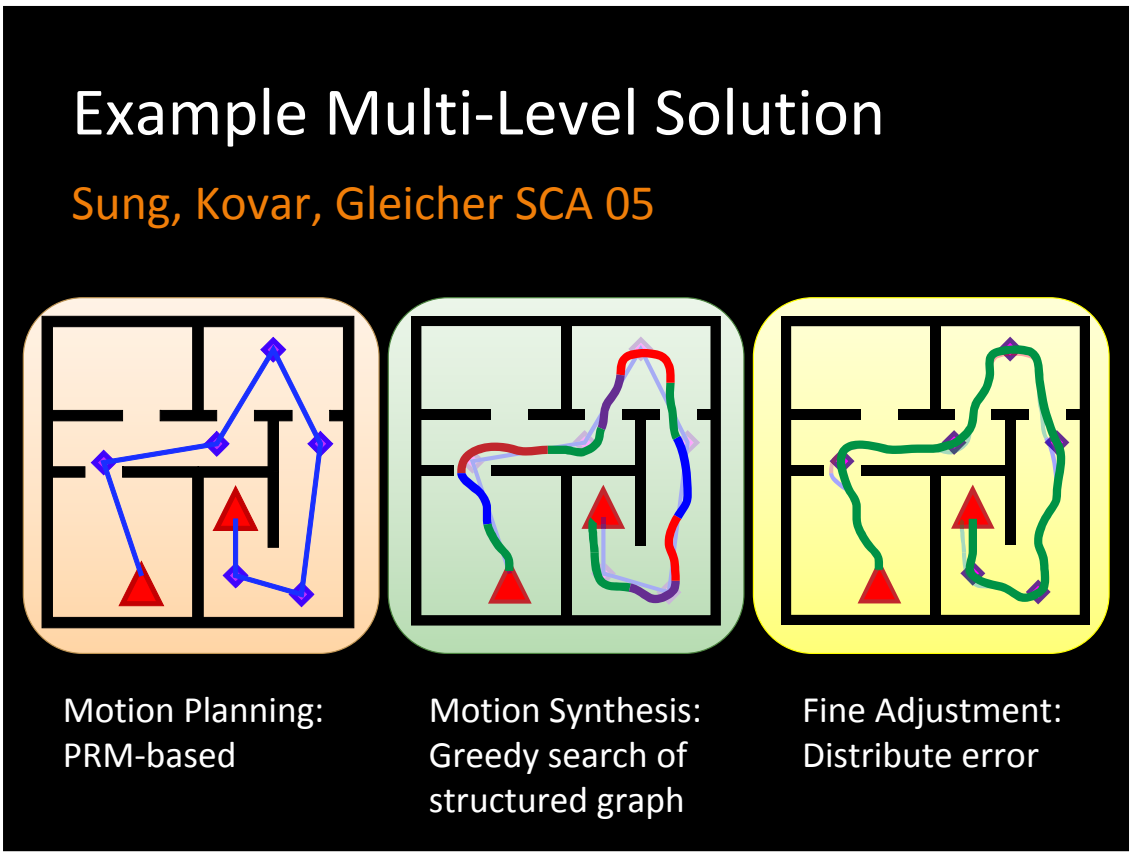
- Motion planning to get rough path
- Motion synthesis to follow path
  - Possibly only gets close
- Motion Adjustment to exactly meet goals

So the low-level motion graph search is probably not the right way to do character navigation (or other high-level goals).

1)It doesn't scale well to larger environments – small movements are too small of a granularity
2)It can't make use of all of the interesting mechanisms evolved in Motion Planning for handling very large or complex environments efficiently.
3)Looking at the details of the movement is too short-sighted for getting towards the bigger goals

This suggests a multi-level approach to creating animated characters:
1)At a high level, "motion planning" algorithms figure out a rough path based on high-level goals
2)Motion synthesis methods create specific, detailed character movements that achieve these paths
3)If discrete search is used, it might not achieve the goals precisely enough. Therefore, some adjustment or editing of the generated path might be required.

There are many examples of such multi-level approaches. Claudia will discuss several in the next talk.




As an example, I'll briefly describe a multi-level motion synthesizer that I was involved in. It was primarily the work of Mankyu Sung who was a graduate student at the time.

Mankyu had built a crowd simulator where each character had a reasonable repertoire of actions: they could walk, jog, run, pick up and set down boxes, buy tickets at a counter, and sit down on chairs. Those latter actions required them to be able to position themselves pretty precisely. And planning needed to be efficient since he was simulating a crowd.

Our approach to navigation had three levels:

1)At a high level, a probabilistic roadmap motion planner was able to choose rough paths between start and goal states efficiently.
2)Given these rough paths, a motion synthesizer based on a structured graph created a detailed character movement. A greedy search was used to choose a clip sequence that followed the path from the planner.
3)Once the initial movement was generated, it was adjusted to precisely meet the goals. Because the synthesizer would usually get close, and because the errors could be distributed over the whole motion, these adjustments could be made with a simple scheme that just displaced and rotated the character.

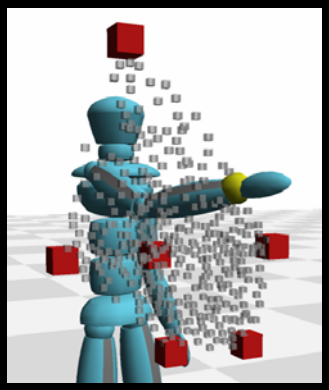
In his thesis and paper, Mankyu describes some more complex algorithms – for example, he used a greedy search forward and backwards and then found the place where these two paths were closest and joined them there.

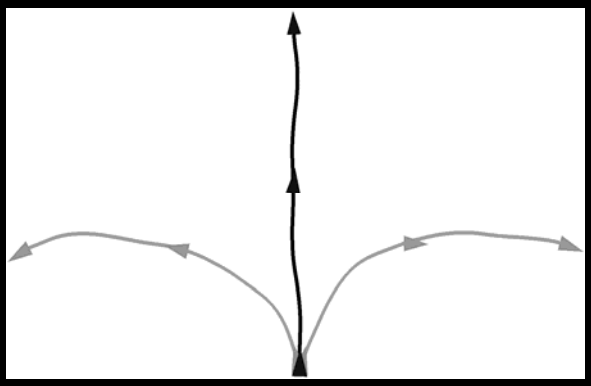
However, the key idea was to break the process of character navigation into these three parts such that each part could be done efficiently by a relatively simple method.

# Continuous Control: Blending

## Range of motions by blending examples

- Get Motions “in-between” examples
- Allows for precise control

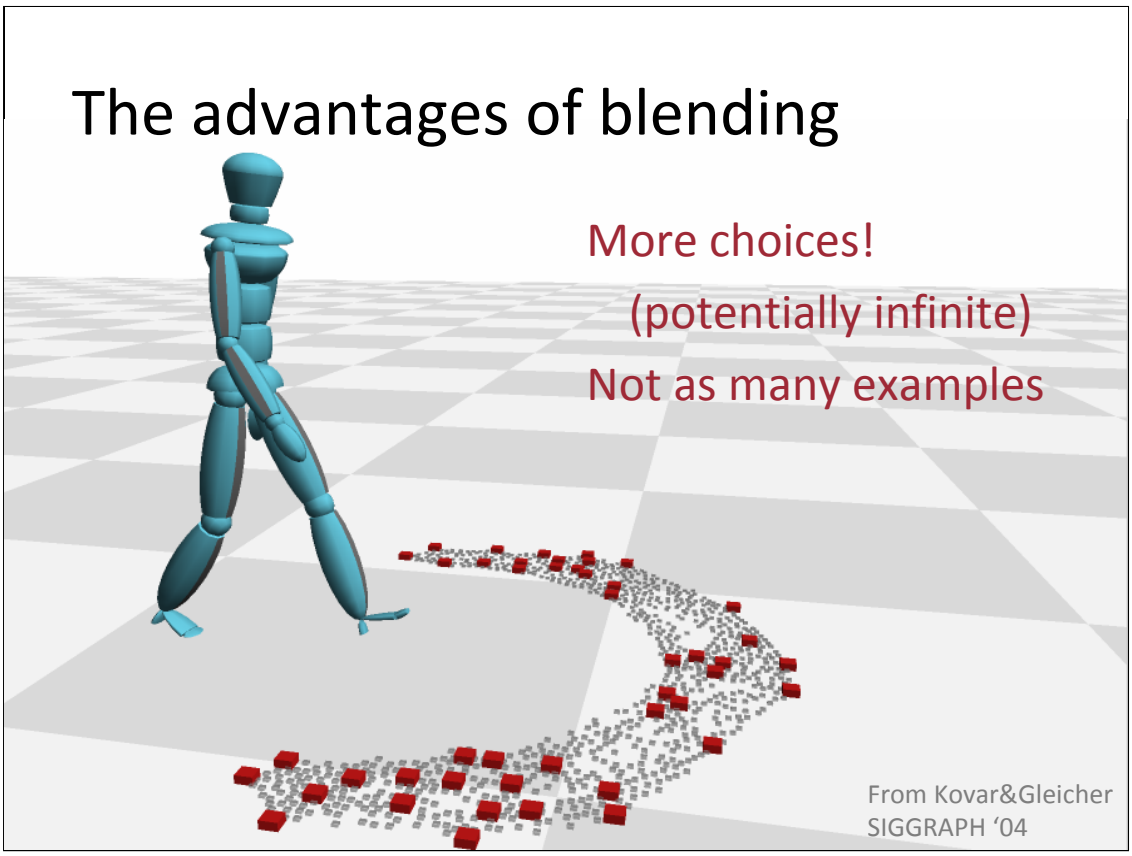




A more sophisticated way to get the precise control is by blending motions together.

For example, if we have multiple motions of a person reaching to different places, we might blend a number of them together to get motions of the person reaching to places in between.

This strategy can be applied to a variety of movements, although for some motions care must be taken in doing the blending to make it work out correctly.




The advantage of blending is that it effectively provides more choices.

If your blends are continuous, you get a continuous space of motions.

For integration with search mechanisms, you might sample to blend space to give a set of discrete choices.

What you see here is the range of places the character can get in 2 steps. The red blocks represent the sample found in the database, the gray squares represent a random sampling of the space of blends.

Blending methods provide an alternative to discrete choices, or making adjustments to individual motions to get continuous ranges of possibilities.

They tend to preserve motion quality over a wider range of movements – giving far more choices than you can get through examples alone, or by adjusting examples.

So mixing blending with a search strategy for assembling clips can provide the flexibility of assembling complex motions that precisely meet goals while preserving motion quality.

However, it is much harder to integrate blending into the search strategies.

## Parametric Graphs

Shin&Oh SCA06, Heck&Gleicher SI3D07, Safanova&Hodgins S07

- Graph of parts with blending
- Clips blended to get precise control

- Flexible synthesis and precision

- Considerable complexity
- Mixed discrete-continuous optimization

However, mixing graphs and blending (or other parametric methods) is difficult.

There are some notable examples of integrating continuous control and search in the recent literature.

Safanova and Hodgins take a brute force approach and sample the continuous blends, producing very large discrete graphs. They use sophisticated search techniques to make using these large graphs practical.

Other work, including our own, create graphs where each node of the graph is a parametrically controllable motion blender.

To date, these have mainly been used for interactive control (driving a character with a joystick), but we have experimented with using a greedy search to drive characters towards their goals.

# Review

## Human motion at various level of detail

- Blends or adjustments to get precise positions
- Concatenation to assemble motions
  - Example-based to get varied motion properties
- Higher level "planning" to get rough paths

## Summary

You can create detailed character movement that meets high-level goals

Key Idea:

Use methods appropriate for the levels of abstraction

- Motion planning to consider the high-level goals
- Motion Synthesis to create the movement details

Example-based synthesis is only one way to create the detailed movements of the characters.

It has the advantage that it uses general algorithms that can be applied to a variety of data to get a diversity of movement types and styles, without having to hand-craft a lot of controllers.