Motion Graphs: A Comprehensive Analysis

Realistic and Controllable Motion Synthesis from Motion Capture Data

Technical Report

November 11, 2025

Abstract

This report provides a comprehensive analysis of the seminal paper ”Motion Graphs” by Kovar, Gleicher, and Pighin (2002). We examine the paper’s core objectives of creating realistic, controllable motion from motion capture data, delve into the mathematical foundations and implementation details, evaluate its contributions to computer graphics and animation, and discuss its limitations along with subsequent advancements in the field.

1 Introduction and Objectives

1.1 What the Paper Aims to Do

The Motion Graphs paper addresses a fundamental challenge in computer animation: how to create realistic human motion that can be dynamically controlled while maintaining the quality of motion capture data.

The Core Problem: Motion capture provides highly realistic human movement, but it’s essentially a recording technique. You can replay what was captured, but modifying it is difficult. If you need a character to walk around a virtual room with specific turns and stops, you’d traditionally need to capture exactly that motion—an expensive and time-consuming process.

Simple Analogy: Think of traditional motion capture like having video clips of someone walking. You can play these clips, but if you need the person to walk in a different pattern, you’d need to film new footage. Motion graphs are like having a smart system that can seamlessly splice and transition between different walking clips to create new walking patterns you never filmed.

1.2 How It Works: High-Level Overview

The paper’s solution involves three main steps:

1. **Build a Motion Graph:** Automatically analyze a database of motion capture clips to find places where different motions can be smoothly connected. This creates a directed graph where:
   - **Edges** represent pieces of motion (either original or transitions)
   - **Nodes** represent connection points where motions can switch
2. **Create Transitions:** Automatically generate smooth transitions between similar motions so the character can switch from one action to another without jarring discontinuities.
3. **Extract Motion:** Use graph search algorithms to find paths through the motion graph that satisfy user requirements (like ”walk along this path” or ”sneak for 5 seconds then walk”).

Real-World Example: The authors took just 78.5 seconds of motion capture footage of someone walking randomly and used it to generate new motions following arbitrary paths drawn on the ground, including smooth curves and sharp turns—movements that weren't explicitly in the original data.

2 Mathematical Background and Implementation

This section details the mathematical foundations that make motion graphs work.

2.1 Motion Representation

Motion capture data represents a character's pose at discrete time intervals. Each frame $\vec{f}$ consists of:

$$\vec{f} = (R_{root}, q_1, q_2, \ldots, q_n) \tag{1}$$

where:

- $R_{root} = (x, y, z)$ is the 3D position of the root joint
- q_i are quaternions representing joint orientations
- n is the number of joints in the skeleton

Why This Matters: This representation separates the character's position in space from its pose, which is crucial for comparing and blending motions.

2.2 Detecting Candidate Transitions

2.2.1 The Distance Metric

The core mathematical challenge is determining when two frames are "similar enough" to create a smooth transition. The paper uses a point cloud-based distance metric.

Point Cloud Representation: Instead of comparing joint angles directly, the algorithm compares the 3D positions of points attached to the character's skeleton. Think of placing dozens of markers all over a person's body and comparing where these markers are in 3D space.

For two frames $\vec{f_i}$ and $\vec{f_j}$, the distance is computed over temporal windows of length k (typically $\frac{1}{3}$ second):

$$D(\vec{f_i}, \vec{f_j}) = \min_{\theta, x_0, z_0} \sum_p w_p \| p_i - T_{\theta, x_0, z_0}(p'_i) \|^2 \tag{2}$$

where:

- p_i and p'_i are corresponding points in the two point clouds
- w_p are weights (higher for important joints like feet)
- T_{θ, x_0, z_0} is a 2D rigid transformation (rotation θ + translation (x_0, z_0))

Why Minimize Over Transformations? Motion is invariant to position and orientation on the ground plane. A person walking north is fundamentally doing the same motion as walking east—just rotated 90 degrees. The optimization finds the best alignment before comparing.

2.2.2 Closed-Form Solution

The paper derives a closed-form solution for the optimal transformation. Given point clouds with points (x_i, z_i) and (x'_i, z'_i):

$$\theta = \arctan\left(\frac{\sum_i w_i(x_i z'_i - x'_i z_i) - \frac{1}{\sum_i w_i}(\bar{x}\bar{z'} - \bar{x'}\bar{z})}{\sum_i w_i(x_i x'_i + z_i z'_i) - \frac{1}{\sum_i w_i}(\bar{x}\bar{x'} + \bar{z}\bar{z'})}\right) \tag{3}$$

$$x_0 = \frac{1}{\sum_i w_i}(\bar{x} - \bar{x'}\cos\theta - \bar{z'}\sin\theta) \tag{4}$$

$$z_0 = \frac{1}{\sum_i w_i}(\bar{z} + \bar{x'}\sin\theta - \bar{z'}\cos\theta) \tag{5}$$

where $\bar{x} = \sum_i w_i x_i$ and similarly for other barred terms.

Computational Relevance: This closed-form solution is critical for efficiency. Computing distances for all pairs of frames in a database has $O(F^2)$ complexity, where F is the number of frames. With a closed-form solution, each distance calculation is fast enough to make this feasible.

2.2.3 Finding Local Minima

After computing distances for all frame pairs, the algorithm identifies local minima—"sweet spots" where transitions are locally optimal. A frame pair $(\vec{f_i}, \vec{f_j})$ is a local minimum if:

$$D(\vec{f_i}, \vec{f_j}) < D(\vec{f}_{i+\delta_1}, \vec{f}_{j+\delta_2}) \quad \forall(\delta_1, \delta_2) \in \text{neighborhood} \tag{6}$$

Why Local Minima? Not every frame pair with reasonable distance makes a good transition. Local minima represent frames that are not just similar, but more similar than nearby alternatives—these tend to produce the smoothest transitions.

2.3 Creating Transitions

2.3.1 Blend Function

When a candidate transition is accepted, a blend is created between frames $\vec{f_i}$ through $\vec{f}_{i+k-1}$ and frames $\vec{f}_{j-k+1}$ through $\vec{f_j}$.

First, apply the aligning transformation T_{θ,x_0,z_0} to the second motion. Then for frame p of the transition $(0 \leq p < k)$:

$$R_p = \alpha(p)R_{\vec{f}_{i+p}} + [1 - \alpha(p)]R_{\vec{f}_{j-k+1+p}} \tag{7}$$

$$q_p^m = \text{slerp}(q^m_{\vec{f}_{i+p}}, q^m_{\vec{f}_{j-k+1+p}}, \alpha(p)) \tag{8}$$

where R_p is the root position, q_p^m is the quaternion for joint m, and slerp is spherical linear interpolation.

2.3.2 Blend Weight Function

The blend weight function $\alpha(p)$ must ensure C^1 continuity (smooth velocities):

$$\alpha(p) = \begin{cases} 1 & p \leq -1 \\ 2\left(\frac{p+1}{k}\right)^3 - 3\left(\frac{p+1}{k}\right)^2 + 1 & -1 < p < k \\ 0 & p \geq k \end{cases} \tag{9}$$

Why This Form? This cubic polynomial ensures:

- $\alpha(-1) = 1, \alpha(k) = 0$ (correct boundary values)
- $\alpha'(-1) = 0, \alpha'(k) = 0$ (smooth velocity at boundaries)
- Smooth transition from first motion to second

Effect on Results: Without smooth blending, transitions would have visible jerks or pops. The C^1 continuity ensures velocities change smoothly, making transitions imperceptible to viewers.

2.4 Graph Structure and Pruning

2.4.1 Strongly Connected Components

To ensure infinite motion generation, the graph must be properly connected. The algorithm computes Strongly Connected Components (SCCs) using Tarjan's algorithm.

Definition: An SCC is a maximal set of nodes S where for any two nodes $u, v \in S$, there exists a path from u to v and from v to u.

Simple Explanation: Imagine you're in a network of one-way streets. An SCC is a group of intersections where you can get from any intersection to any other by following the one-way signs, and eventually return to where you started.

The algorithm retains only the largest SCC for each type of motion (walking, sneaking, etc.). This guarantees:

1. Arbitrarily long motion can be generated
2. The motion stays within one action type (no forced switches from walking to ballet)
3. Maximum database coverage

2.4.2 Graph Complexity

For a database with F frames:

- Transition detection: $O(F^2)$ distance calculations
- Graph construction: $O(E)$ where E is the number of accepted transitions
- SCC computation: $O(V + E)$ where V is the number of nodes

In practice, motion graphs are sparse (few transitions per node) because most frame pairs are too dissimilar.

2.5 Motion Extraction via Graph Search

2.5.1 Optimization Framework

Motion extraction is formulated as finding a graph walk $w = [e_1, e_2, \ldots, e_n]$ that minimizes:

$$f(w) = \sum_{i=1}^{n} g([e_1, \ldots, e_{i-1}], e_i) \tag{10}$$

where $g(w, e)$ is the incremental error of adding edge e to walk w.

Key Constraint: $g(w, e) \geq 0$ for all w, e. This monotonicity enables efficient branch-and-bound search.

2.5.2 Branch and Bound Algorithm

The search uses branch and bound with the following property:

$$f(w) \leq f(w + v) \quad \forall v \tag{11}$$

This means $f(w)$ is a lower bound on any extension of w.

Algorithm:

Algorithm 1 Branch and Bound Search

```
1:  $w_{opt} \leftarrow$ initial complete walk
2:  $f_{opt} \leftarrow f(w_{opt})$
3:  function SEARCH($w$)
4:      if $f(w) \geq f_{opt}$ then
5:          return                                        ▷ Prune this branch
6:      end if
7:      if $w$ is complete then
8:          if $f(w) < f_{opt}$ then
9:              $w_{opt} \leftarrow w$
10:             $f_{opt} \leftarrow f(w)$
11:         end if
12:         return
13:     end if
14:     for each valid edge $e$ from end of $w$ do
15:         SEARCH($w + e$)                               ▷ Recursively search
16:     end for
17: end function
```

Efficiency Trick: The algorithm uses incremental generation, searching for optimal paths of n frames (80-120 frames or 2.7-4 seconds), retaining the first m frames (25-30 frames or 1 second), then searching again. This keeps the search tree manageable.

2.6 Path Synthesis: A Concrete Application

2.6.1 Problem Formulation

Given a desired path P (a curve in 2D), find a graph walk that makes the character follow P as closely as possible.

Let $P(s)$ be the point on path P at arc-length s from the start. Let $P'(s)$ be the actual path traveled. The error function is:

$$g(w, e) = \sum_{i=1}^{|e|} \|P'(s(e_i)) - P(s(e_i))\|^2 \tag{12}$$

where $s(e_i)$ is the arc-length along P' at frame i of edge e.

Why This Works: By penalizing deviation from the path at each frame, the algorithm incentivizes the character to stay close to P. The squared distance makes large deviations especially costly.

2.6.2 Arc-Length Parameterization

The key insight is using arc-length parameterization rather than time:

- Frame i of the walk has traveled arc-length s_i along P'
- Compare position to $P(s_i)$ rather than $P(t_i)$

Why This Matters: Different motions have different speeds. A sharp turn covers less distance per frame than walking straight. Arc-length parameterization doesn't penalize the character for slowing down when necessary—it only cares about staying near the path.

Example: Imagine asking someone to walk along a wavy line while maintaining constant speed. That's hard! Arc-length parameterization instead says ”walk along this line at whatever speed is natural”—much easier and more realistic.

2.6.3 Handling Multiple Motion Types

For paths requiring different actions (walk then sneak), the path is divided: $P = P_1 \cup P_2$.

The algorithm tracks which sub-path is being fit and only allows transitions between motion types near the boundary:

$$\text{Allow type switch if: } \|P'(\text{current}) - P_1(\text{end})\| < \epsilon \tag{13}$$

This prevents random switching while ensuring transitions happen near the specified location.

3 Achievements and Benefits

3.1 Key Achievements

3.1.1 Automation

What They Did: The system automatically analyzes motion capture data and builds a usable structure without manual intervention (except setting quality thresholds).

Why It Matters: Previous systems required carefully scripted motion capture sessions where performers would explicitly create connections between motions. This was labor-intensive and required advance planning. Motion graphs work with ”donated” data captured without specific connection planning.

Real Impact: The paper demonstrates generating varied motions from just 78.5 seconds of walking footage—showing that you don't need massive databases to get useful results.

3.1.2 Quality Preservation

What They Did: Generated motion consists primarily of original motion capture data with only brief transitions.

Why It Matters: Motion capture is used specifically because it looks realistic. By using original data as much as possible and only creating short transitions between similar poses, the system maintains this realism.

Technical Advantage: Unlike statistical models that might average motions together or synthesize from learned distributions, motion graphs guarantee that most frames are actual captured motion.

3.1.3 Computational Efficiency

What They Did: Generated animations in near real-time (computation time animation length).

Example Results:

- 54.9 seconds of walking motion: 58.1 seconds computation
- 87.7 seconds of martial arts: 15.0 seconds computation

Why It Matters: This enables interactive applications where motion must be generated on-the-fly, like video games.

3.2 Broader Applications

3.2.1 Interactive Character Control

The paper enables real-time control where user input (keyboard, joystick) directs character movement. By precomputing optimal first edges for different directions from each node, characters can respond instantly to control changes.

3.2.2 AI Integration

Motion graphs serve as a ”motion dump” backend: AI systems plan high-level actions and paths, motion graphs fill in the realistic movement details.

3.2.3 Crowd Simulation

By adding randomness to the search (random starting points, random selection among near-optimal solutions), different characters following similar paths will move differently—essential for believable crowds.

3.3 Scientific Contributions

1. **Automatic Transition Detection:** First system to automatically find and create transitions at scale
2. **Graph-Based Motion Representation:** Formalized motion synthesis as graph search
3. **Practical Motion Synthesis:** Demonstrated controllable motion generation from small databases
4. **Application Framework:** Showed how to apply graph search to specific problems (path following)

4 Limitations and Subsequent Advances

4.1 Limitations of Motion Graphs

4.1.1 Manual Threshold Setting

The Problem: Different motion types need different quality thresholds for transitions. Walking requires very smooth transitions (people notice even small errors), but ballet might tolerate more variation. Users must manually set these thresholds for each motion type pair.

Why It's Limiting: For large databases with many motion types, this becomes tedious. The paper acknowledges this and suggests automating it as future work.

4.1.2 Database Dependency

The Problem: The system can only generate what's implicitly in the database. If no captured motion involves sharp left turns, the system can't generate them convincingly.

Simple Explanation: Motion graphs are like making new sentences by cutting and pasting words from existing sentences. You can create many variations, but you can't create words that aren't in your source material.

Consequence: Achieving a truly diverse movement repertoire requires large motion capture databases, which are expensive to create.

4.1.3 Quadratic Preprocessing

The Problem: Finding transitions requires comparing all pairs of frames: $O(F^2)$ complexity.

Practical Impact: For very large databases (hundreds of thousands of frames), preprocessing becomes time-consuming. The paper reports 25 minutes for 6000 frames—scaling to 100,000 frames could take hours.

4.1.4 Limited Motion Editing

The Problem: Once the graph is built, you can't easily modify the captured motions (make a character taller, adjust style, etc.). The system splices existing motion but doesn't reshape it.

Trade-off: This limitation is actually by design—preserving original motion maintains quality. But it means less flexibility than systems that actively modify motion.

4.1.5 Foot Skating

The Problem: Linear blending in transitions can violate constraints, causing "foot skating" where a planted foot slides along the ground.

Solution in Paper: Use post-processing constraint enforcement (inverse kinematics) to fix planted feet.

Why It's Not Ideal: This is a patch rather than preventing the problem. More sophisticated transition methods could maintain constraints inherently.

4.2 Concurrent and Subsequent Work

The motion graphs paper appeared in 2002 alongside several similar approaches. Here are the major advances since:

4.2.1 Parametric Motion Graphs (2003-2005)

Key Innovation: Instead of fixed motion clips, nodes contain parameterizable motion (motion blended from multiple examples).

Advantage: Greater flexibility—the character can smoothly adjust speed, step width, etc., not just pick discrete motion clips.

Example System: ”Real-Time Motion Synthesis Using Motion Graphs with Adaptive Sampling” by Zhao and Safonova (2008).

4.2.2 Motion Matching (2016+)

Key Innovation: Rather than precomputing a graph, search the database in real-time for the best next few frames given current state and future trajectory.

Major Paper: ”Motion Matching and The Road to Next-Gen Animation” by Clavet (2016, GDC talk) and ”Learned Motion Matching” by Holden et al. (2020).

Advantages:

- No preprocessing required
- Naturally handles large databases
- More responsive to user input
- Better handles database updates (just add new clips)

Why It Succeeds Motion Graphs: Motion matching is now the industry standard for game character animation (used in The Last of Us Part II, many other AAA games). It provides similar quality with better flexibility.

4.2.3 Deep Learning Approaches (2017+)

Neural Networks for Motion Synthesis: Systems like ”Phase-Functioned Neural Networks” (Holden et al., 2017) and ”Learned Motion Matching” learn to synthesize motion from examples.

Advantages:

- Can generate subtle variations not in database
- Compact representation (network weights vs. full database)
- Potentially handles novel situations better

Trade-offs:

- Requires large training datasets
- Less guaranteed quality than using original motion
- ”Black box” behavior harder to control

4.2.4 Motion Graphs with Reinforcement Learning (2018+)

Key Innovation: Use reinforcement learning to learn policies for walking the motion graph, rather than hand-designed search algorithms.

Example: ”DeepMimic” (Peng et al., 2018) learns physics-based controllers from motion capture.

Advantage: Automatically learns complex control strategies, can adapt to new situations.

4.3 Modern Context

Legacy: Motion graphs established core ideas still used today:

- Preserving original motion capture quality
- Automatic transition detection
- Formulating motion synthesis as optimization
- Separating motion database from control algorithm

Evolution: Modern systems often combine multiple approaches:

- Motion matching (real-time database search) + neural networks (smooth blending)
- Motion graphs (structure) + learning (control policies)
- Traditional animation + data-driven synthesis

5 Conclusion

The Motion Graphs paper made several lasting contributions to computer animation:

1. **Automatic Motion Structure:** Showed how to automatically analyze and structure motion capture data for reuse
2. **Quality-Preserving Synthesis:** Demonstrated that using original data with minimal modification produces superior results
3. **Practical Framework:** Provided a complete system from preprocessing to motion generation
4. **Application Versatility:** Showed applications from interactive control to path following to crowds

While subsequent techniques like motion matching and deep learning have superseded motion graphs in many applications, the fundamental insights remain relevant. The paper's emphasis on preserving motion quality, automatic transition detection, and casting synthesis as optimization continue to influence modern animation systems.

For students and researchers, motion graphs provide an excellent introduction to data-driven animation: the mathematics is tractable, the algorithms are understandable, and the results are impressive. It demonstrates that careful engineering and clever algorithms can extract remarkable capabilities from relatively simple motion databases.

The paper's 20+ year legacy in both academic research and industry practice confirms its significance as a foundational work in computer animation and motion synthesis.