Data Driven Computer Animation

HKU COMP 7508

Tutorial 3 - Animation Processing and Scripting

Prof. Taku Komura

TA: Zhouyingcheng Liao (zliao@connect.hku.hk)

Section 1A, 2024

Review on Assignment 1

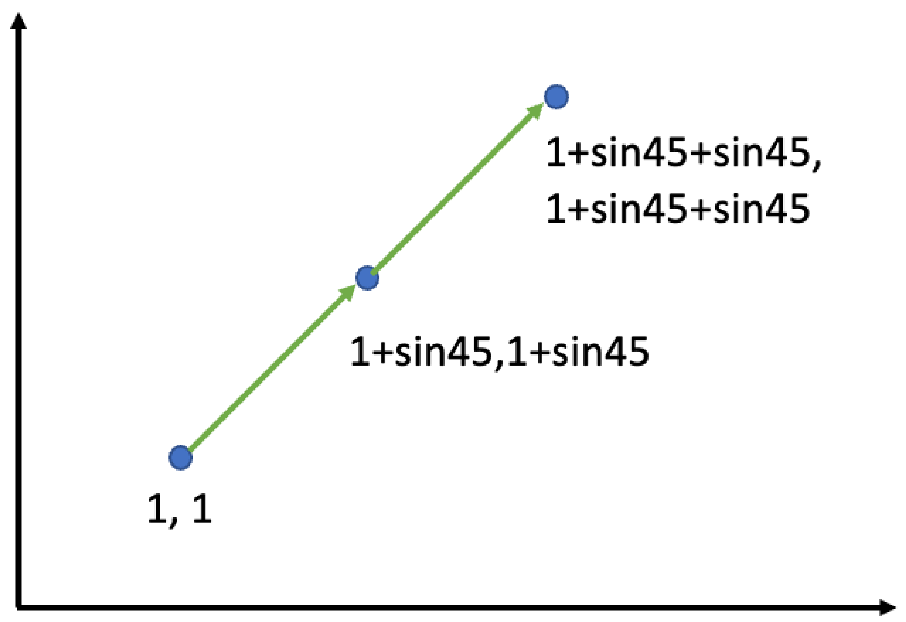
T-Pose

```
if parent_idx == -1:
    global_joint_position[joint_idx] = joint_offsets[joint_idx]
else:
    global_joint_position[joint_idx] = global_joint_position[parent_idx] + joint_offsets[joint_idx]
```

-> The vector between two joints is known
-> make the vector align to its parent joint position

Review on Assignment 1

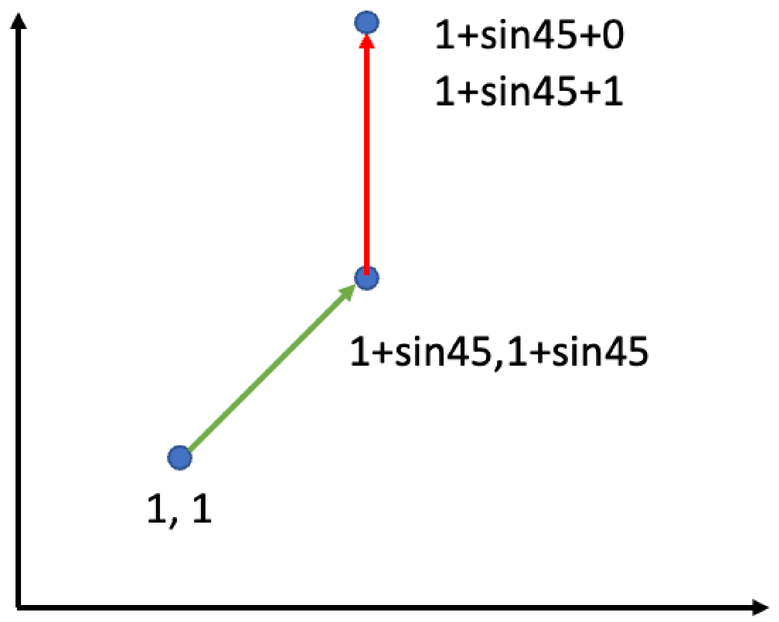
Method 1: Consider each rotation



The first rotation

new_bone0 = bone0 @ 45 degree = (sin45, sin45)
new_bone1 = bone1 @ 45 degree = (sin45, sin45)

Joint 1: joint0.position + new_bone0
Joint 2: joint1.position(new) + new_bone1



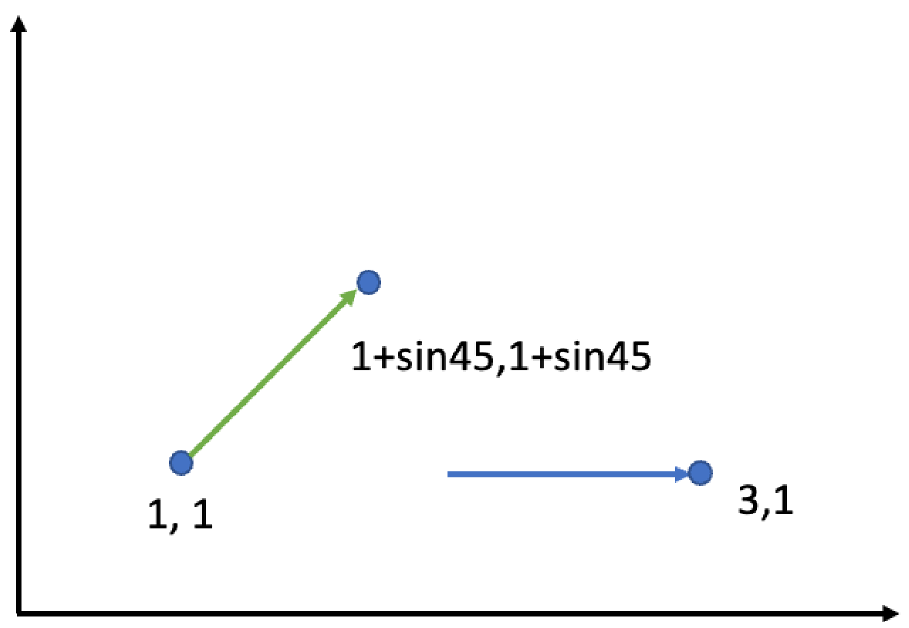
The second rotation

new_new_bone1 = new_bone1 @ 45 degree
= (0, 1)

Joint 2: joint1.position(new) + new_new_bone1

The location of joints will be changed n·log(n) times

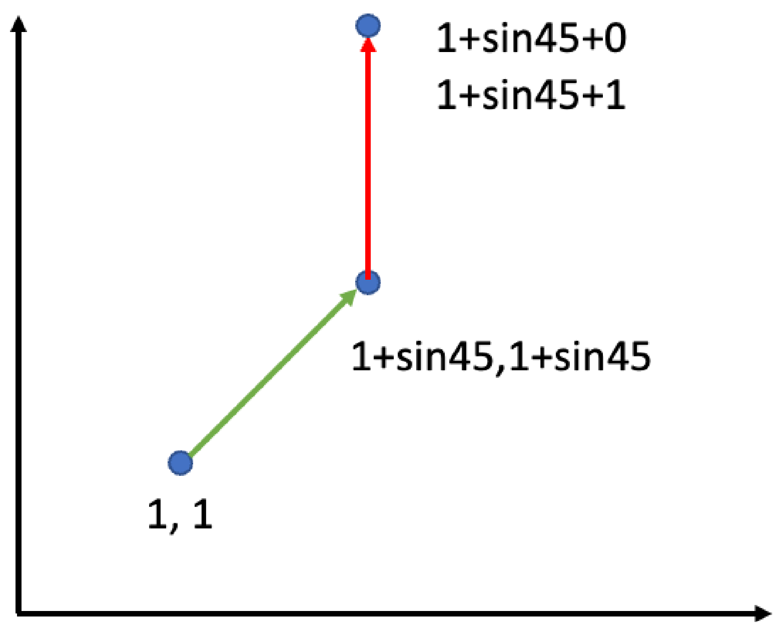
Method 2: Consider each bone



The first joint updated

new_bone0 = bone0 @ 45 degree = (sin45, sin45)

Joint 1: joint0.position + new_bone0
= 1+sin45,1+sin45



The second joint updated

new_joint1_rotation = 45 + 45 = 90
new_bone1 = bone1 @ 90 degree
= (0, 1)

Joint 2: joint1.position(new) + new_bone1

The location of joints will be changed n times

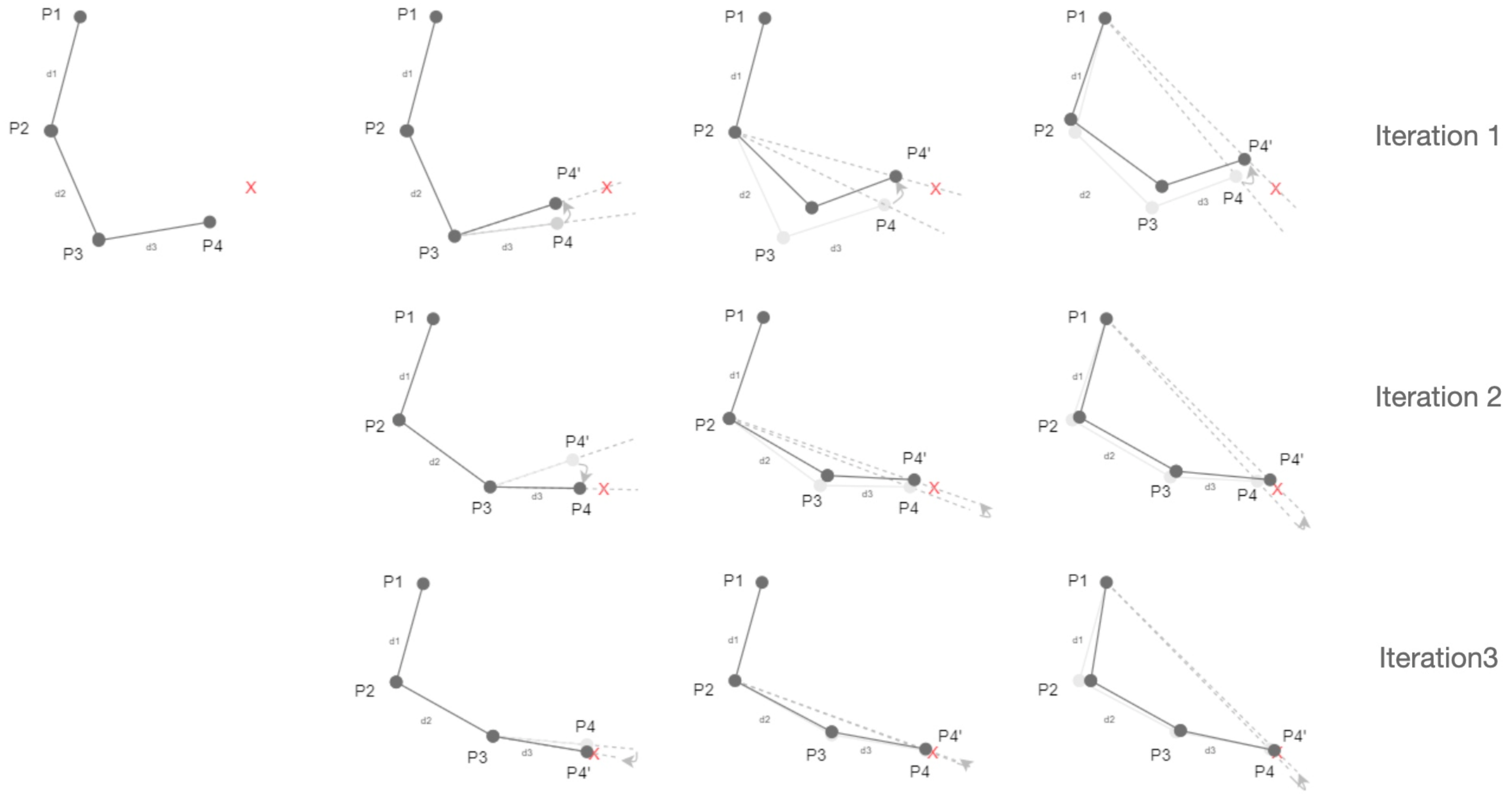
FK

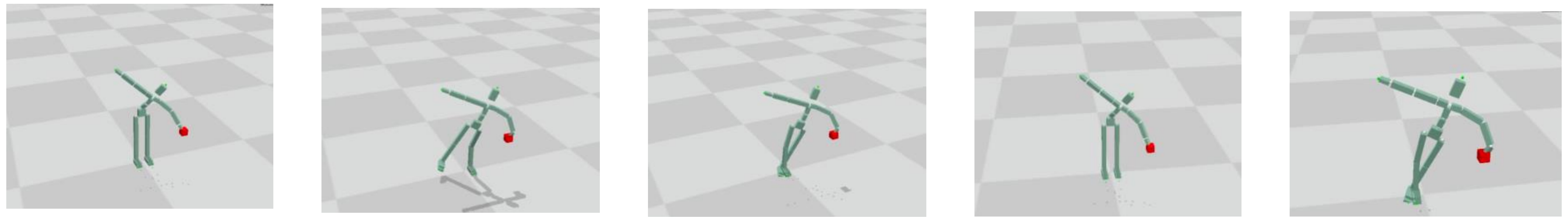
```python
for joint_idx, parent_idx in enumerate(joint_parents):
    parent_orientation = R.from_quat(global_joint_orientations[:, parent_idx, :])
    rotated_vector = parent_orientation.apply(joint_positions[:, joint_idx, :])
    global_joint_positions[:, joint_idx, :] = global_joint_positions[:, parent_idx, :] + rotated_vector
    global_joint_orientations[:, joint_idx, :] = (parent_orientation * R.from_quat(joint_rotations[:, joint_idx, :])).as_quat()
```

Review on Assignment 1

CCD-IK

```python
vec_cur2end = norm(chain_positions[end_idx] - chain_positions[current_idx])
vec_cur2tar = norm(target_pose - chain_positions[current_idx])

axis = norm(np.cross(vec_cur2end, vec_cur2tar))
rot = np.arccos(np.vdot(vec_cur2end, vec_cur2tar))

if np.isnan(rot):
    continue

rotate_vector = R.from_rotvec(rot * axis)
chain_orientations[current_idx] = rotate_vector * chain_orientations[current_idx]
```





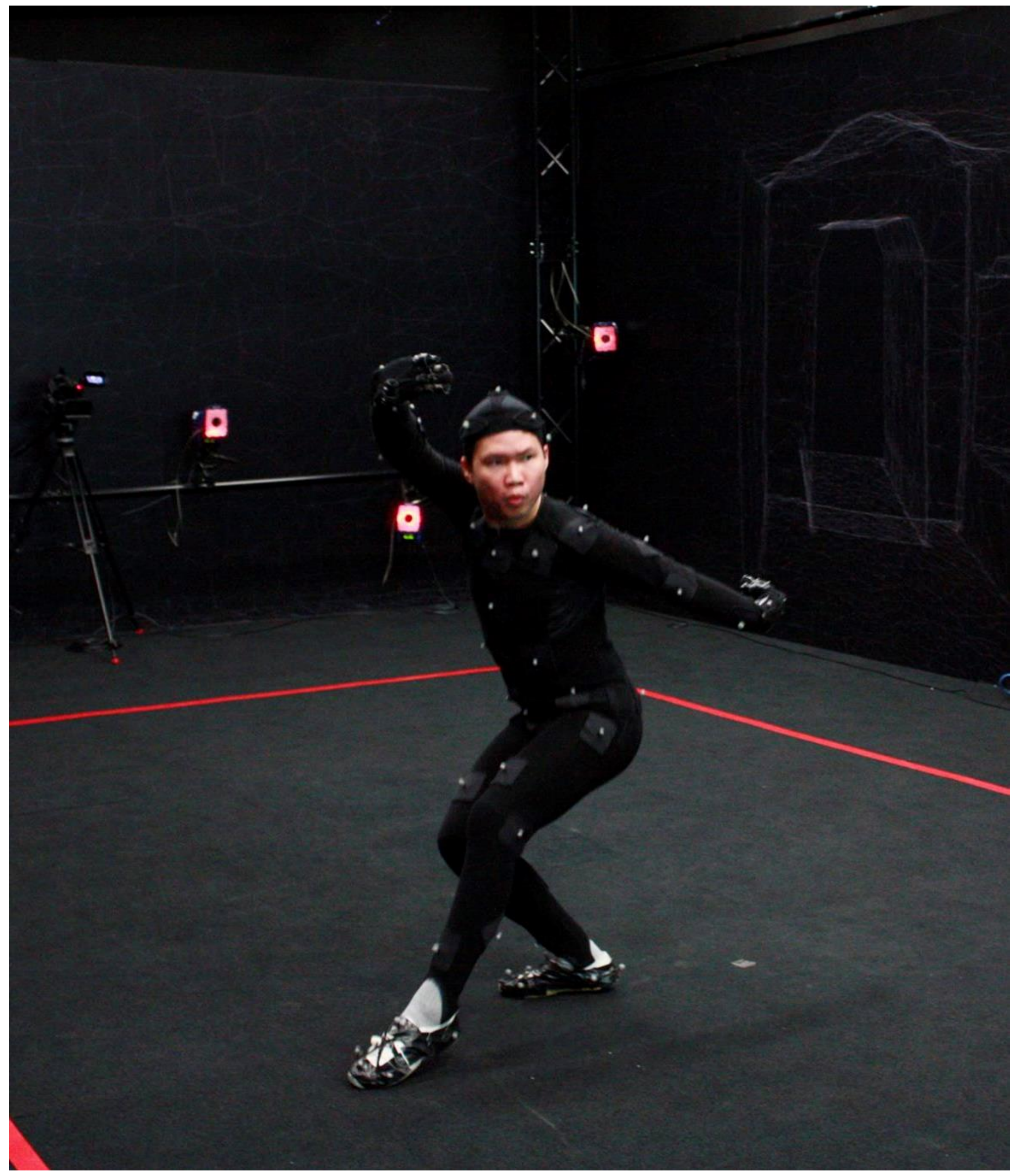
Review on Assignment 1

By the experiment of Keyframe Animation and IK, you might find:

1. Some poses look unrealistic

2. It requires lots of experiences to produce high quality animation

3. IK makes problems also (sometimes)

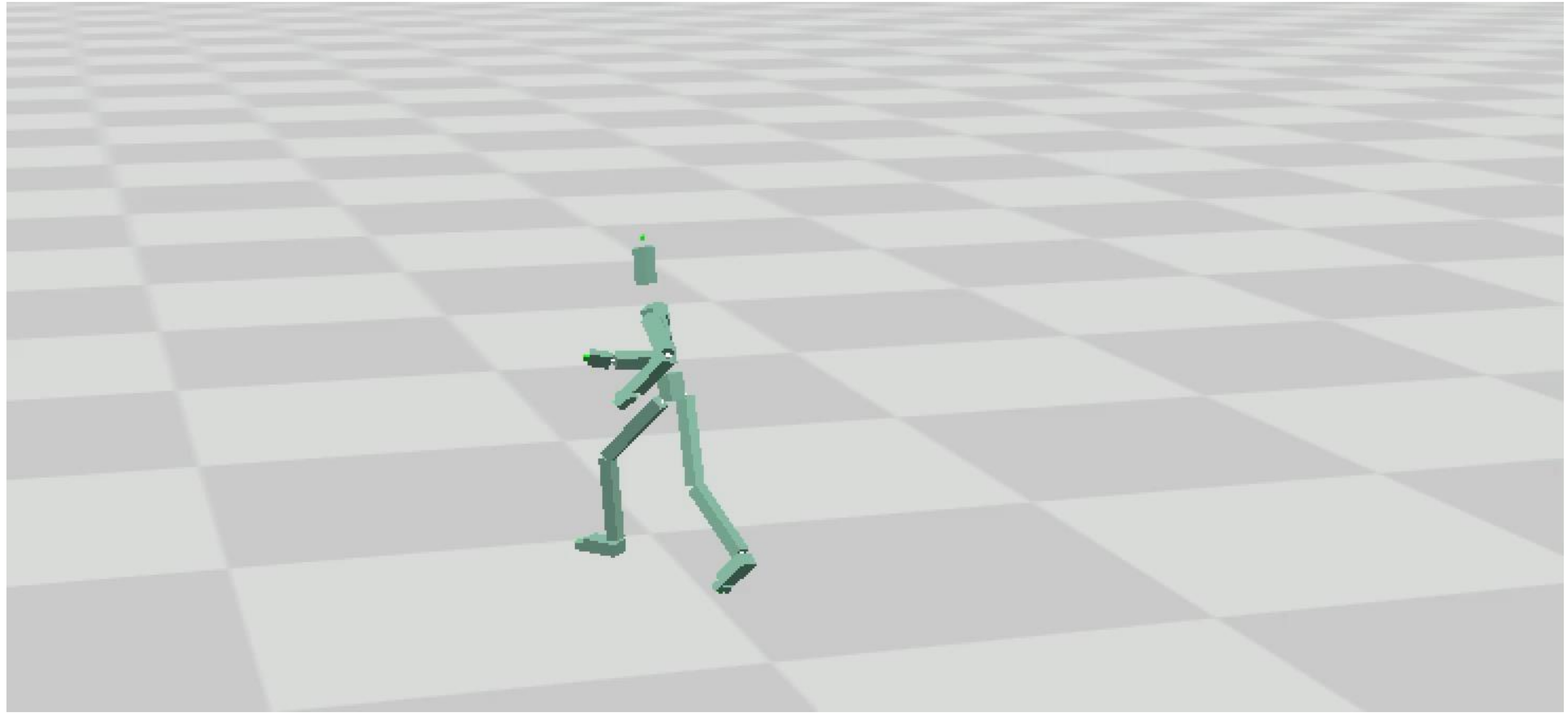


We are trying to improve it

We still need Motion Capture

Tutorial 3 - Agenda

- Basic properties for the motion data
- Implement the Temporal Editing of motion data (30%)
- Implement a simple Motion Blending (35%)



(Under python and panda3D environment)

Motion Features

- Frametime = 1 / fps
- Joint Velocity: delta_(jointPosition) / delta_(t)
- Joint Angular Velocity delta_(jointRotation) / delta_(t)
- Predicted(future) joint position/rotation: current_position + velocity * time

AS2 Task 1: Temporal Editing

- Given a motion data with 120 frames, 60 fps (2s)
- Downsampling: take frame 0, 2, 4, 6… 120. Keep 15 fps, 4s
- Upsampling: use 120 frames to do the interpolation for generating 240 frames, keep same time duration -> 60 fps
- Change the frame number, but use different fps

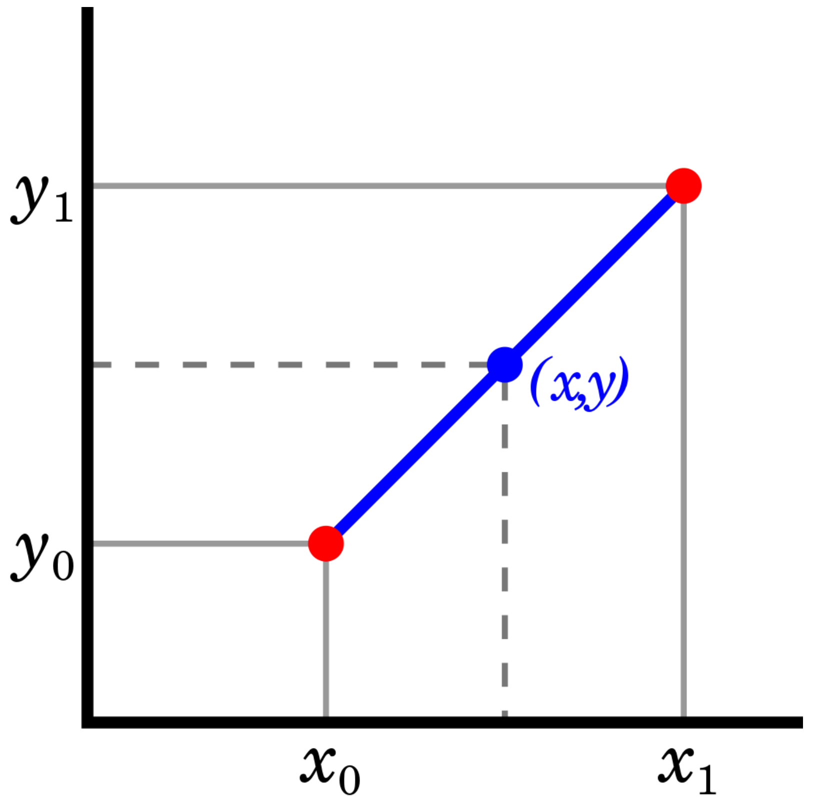
What is Interpolation

- 1, 2, 3, 4, 5, 6, ?, ?, ?, 10, 11 can we fill it?
- 1, ?, ?, ?, ?, ?, ? ?, ?, ?, 11 can we fill it?

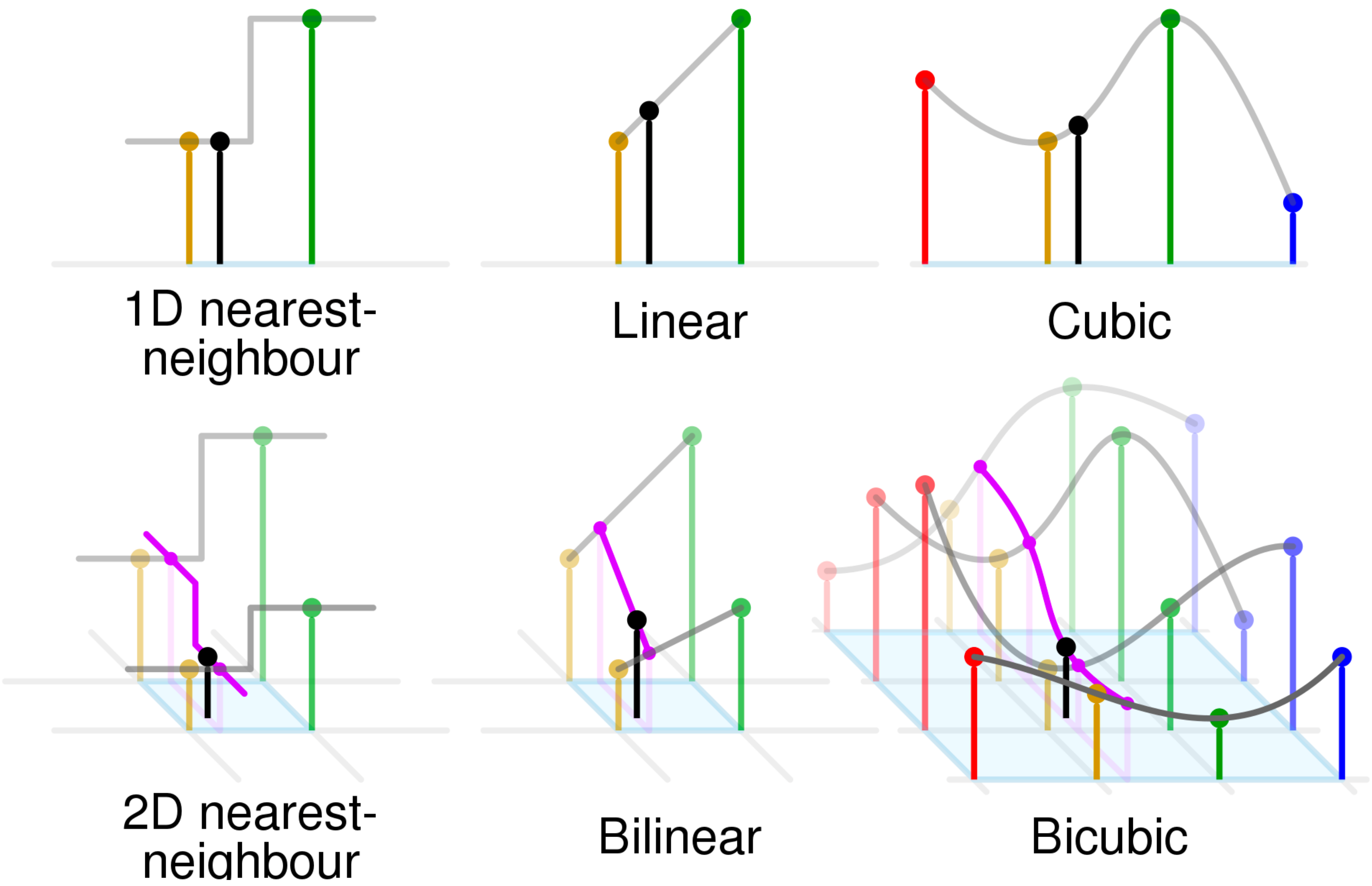
offset = 11 - 1 = 10

per_item_offset = 10 / 10 = 1

results = [1 + i*1] for i in range(0, 10)

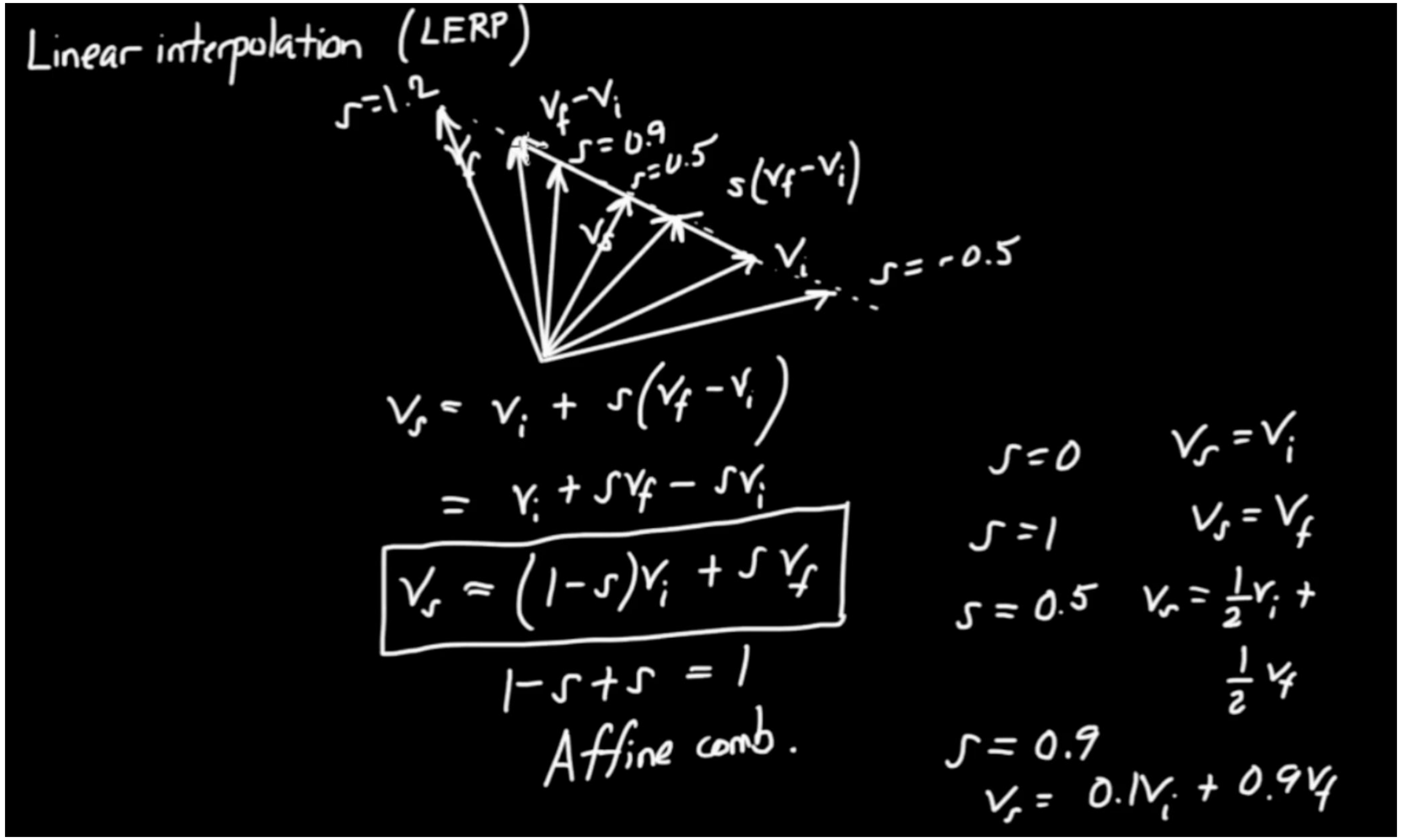


Linear interpolation, refer to [wiki]



The zoo of interpolation method

Interpolation for vector



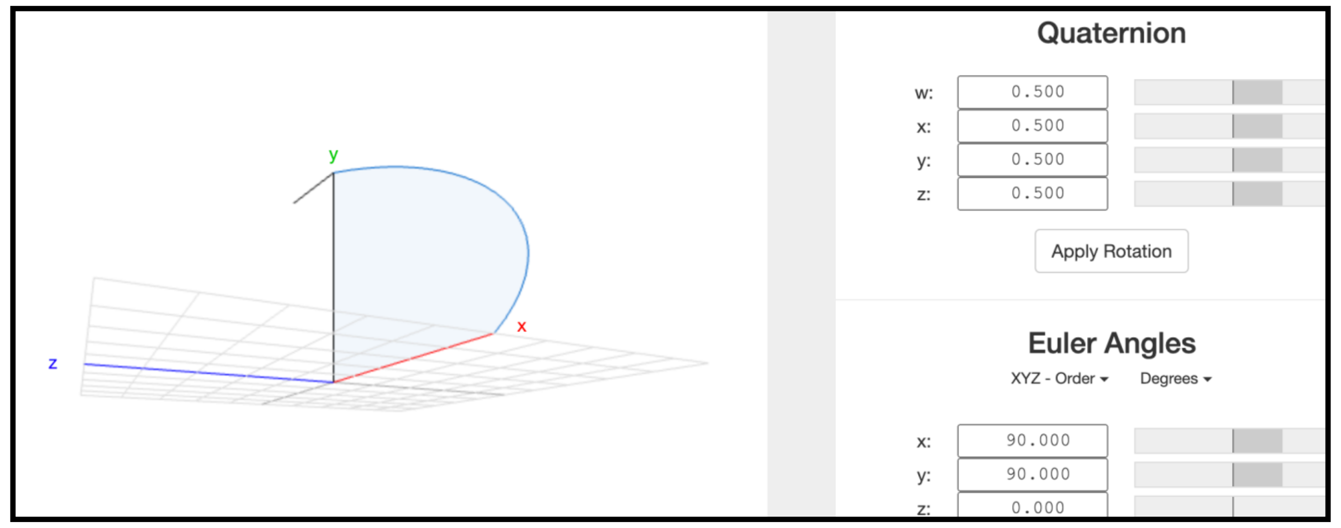
https://www.youtube.com/watch?v=ibkT5ao8kGY

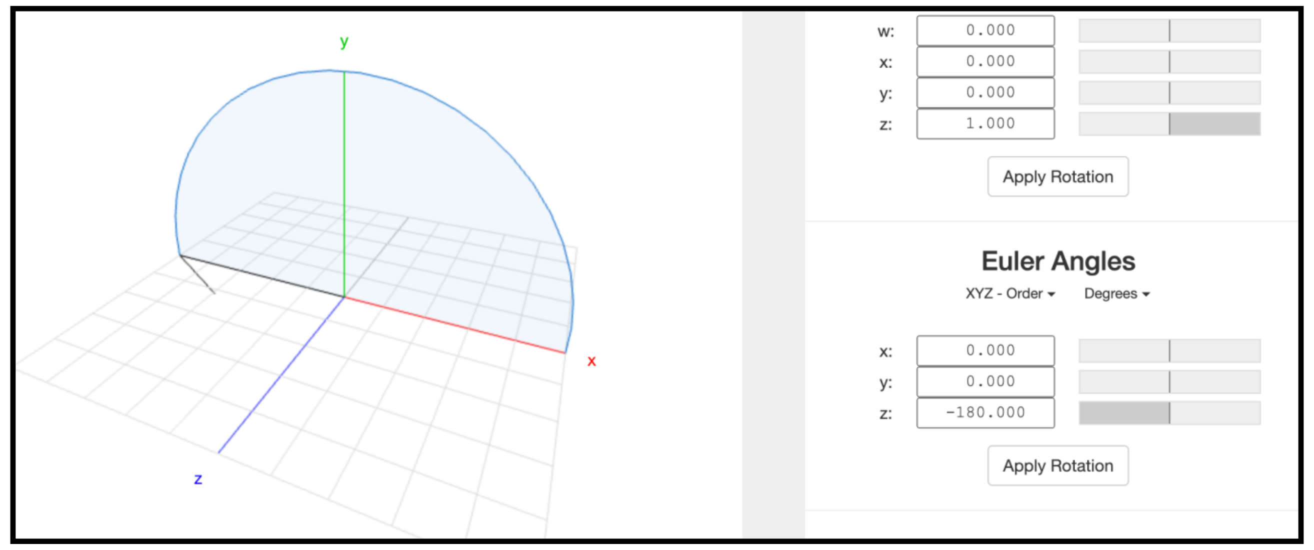
What is Interpolation

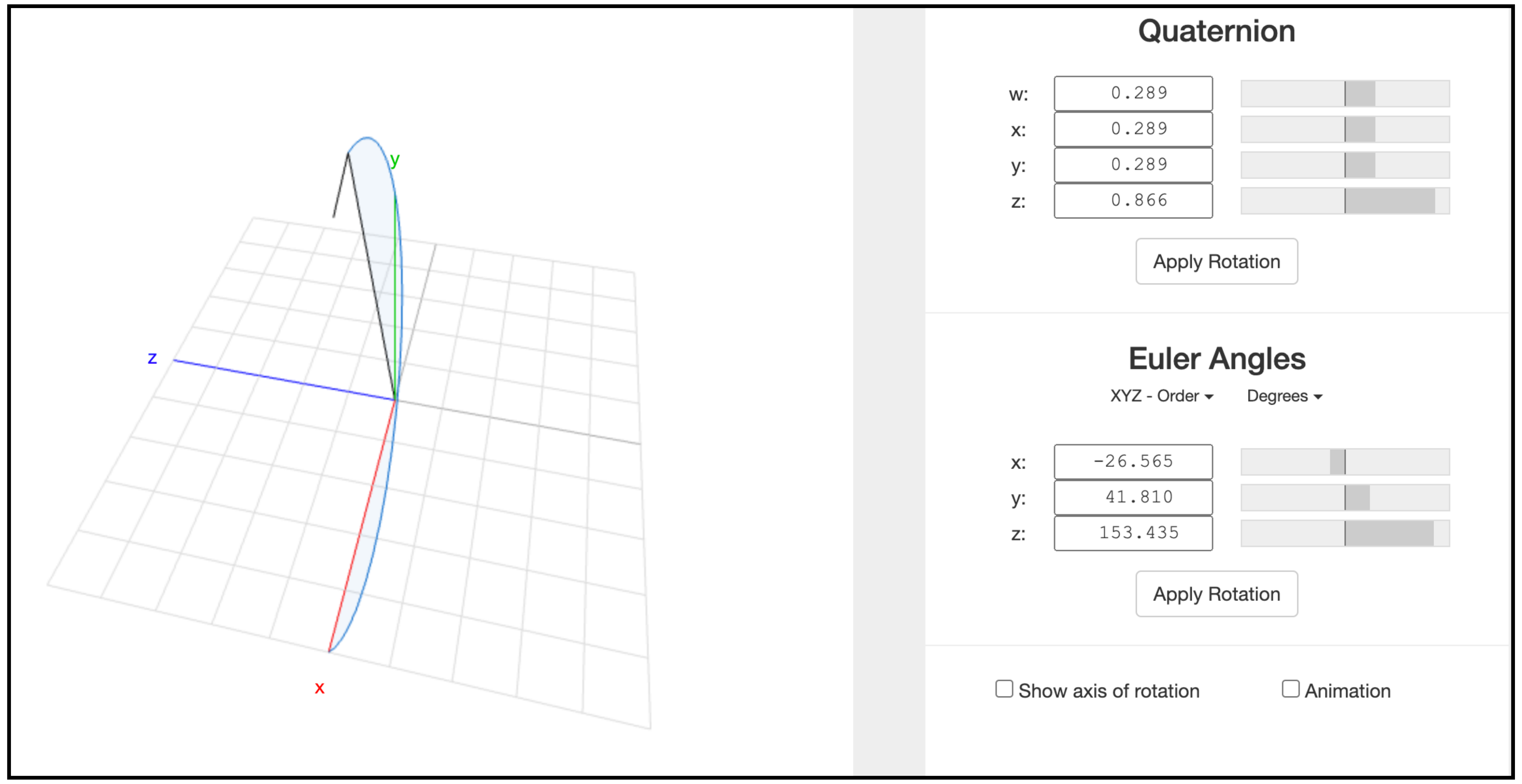
Pose = local_joint_position (root) + local_joint_rotation

We can apply the linear on the joint position, but how about the rotation?

For example, we have [0.5, 0.5, 0.5, 0.5] and [0, 0, 0, 1]







A weird result by linear interpolation

Slerp of Rotation

Slerp: spherical linear interpolation

$$\mathrm{Slerp}(q_1, q_2; u) = q_1 \left(q_1{}^{-1} q_2\right)^u$$

$$q = \cos\left(\frac{\theta}{2}\right) + \sin\left(\frac{\theta}{2}\right)(xi + yj + zk)$$

$$q^x = \cos\left(\frac{x\theta}{2}\right) + \sin\left(\frac{x\theta}{2}\right)(xi + yj + zk)$$

A quaternion q to the power of x means its rotation axis stays the same, but its rotation angle is multiplied by x

Derivation of Slerp

Before looking at the general case $\operatorname{Slerp}(q_0, q_1; t)$, which interpolates from q_0 to q_1, let' s look at the much simpler case of interpolating from the identity $\mathbf{1}$ to some unit quaternion q.

$$\mathbf{1} = (1, (0, 0, 0))$$
$$q = \left(\cos\frac{\alpha}{2}, \vec{n}\sin\frac{\alpha}{2}\right)$$

To move along the great arc from $\mathbf{1}$ to q, we simply have to change the angle from 0 to α while the rotation axis $\vec{n}$ stays unchanged.

$$\operatorname{Slerp}(\mathbf{1}, q; t) = \left(\cos\frac{\alpha t}{2}, \vec{n}\sin\frac{\alpha t}{2}\right) = q^t, \text{ where } 0 \le t \le 1$$

To generalize this to the great arc from q_0 to q_1, we can start with q_0 and left-multiply an appropriate Slerp using the relative rotation (global frame) $q_{0,1}$:

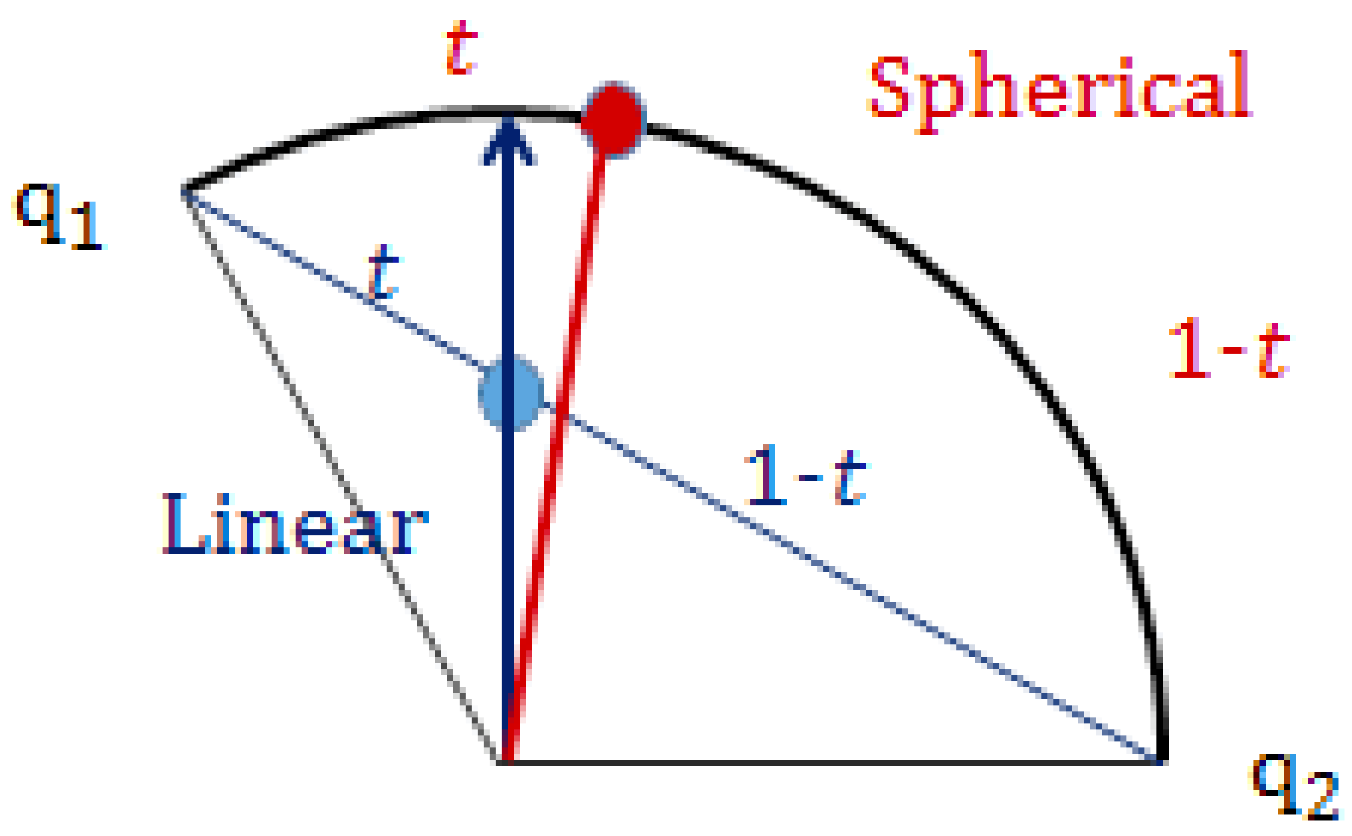
$$\operatorname{Slerp}(q_0, q_1; t) = \operatorname{Slerp}(\mathbf{1}, q_{0,1}; t)\, q_0$$

Inserting $q_{0,1} = q_1 {q_0}^{-1}$, we get:

$$\operatorname{Slerp}(q_0, q_1; t) = \left(q_1 {q_0}^{-1}\right)^t q_0$$

Spherical Linear Interpolation (Slerp) — splines, version 0.3.2-1-g58e77e1

Slerp of Rotation



More refer to wiki

We use Scipy to implement it; there are q1, q2:

```
from scipy.spatial.transform import Slerp

key_quaternions = R.from_quat(q1), …(q2)
keys = [0 , 1]
slerp_function = Slerp(keys, key_quaternions)
new_keys = np.linspace(0, 1, 10)        ->       0, 0.1, 0.2 … 1
interp_quaternions = slerp_function(new_keys)
```

Assignment 2 - Part 1

Your goal:

For a given motion with 100 frames, take the keyframes per 10 frames.

Then you have 10 keyframes.

Then you use these keyframes, to produce the interpolation between each pairs

For example:

each pair -> 10 new poses (same as original motion)

each pair -> 20 new poses (more poses, so the same fps will yield longer time duration)

each pair -> 5 new poses (more poses, so the same fps will yield shorter time duration)

Assignment 2 - Part 1

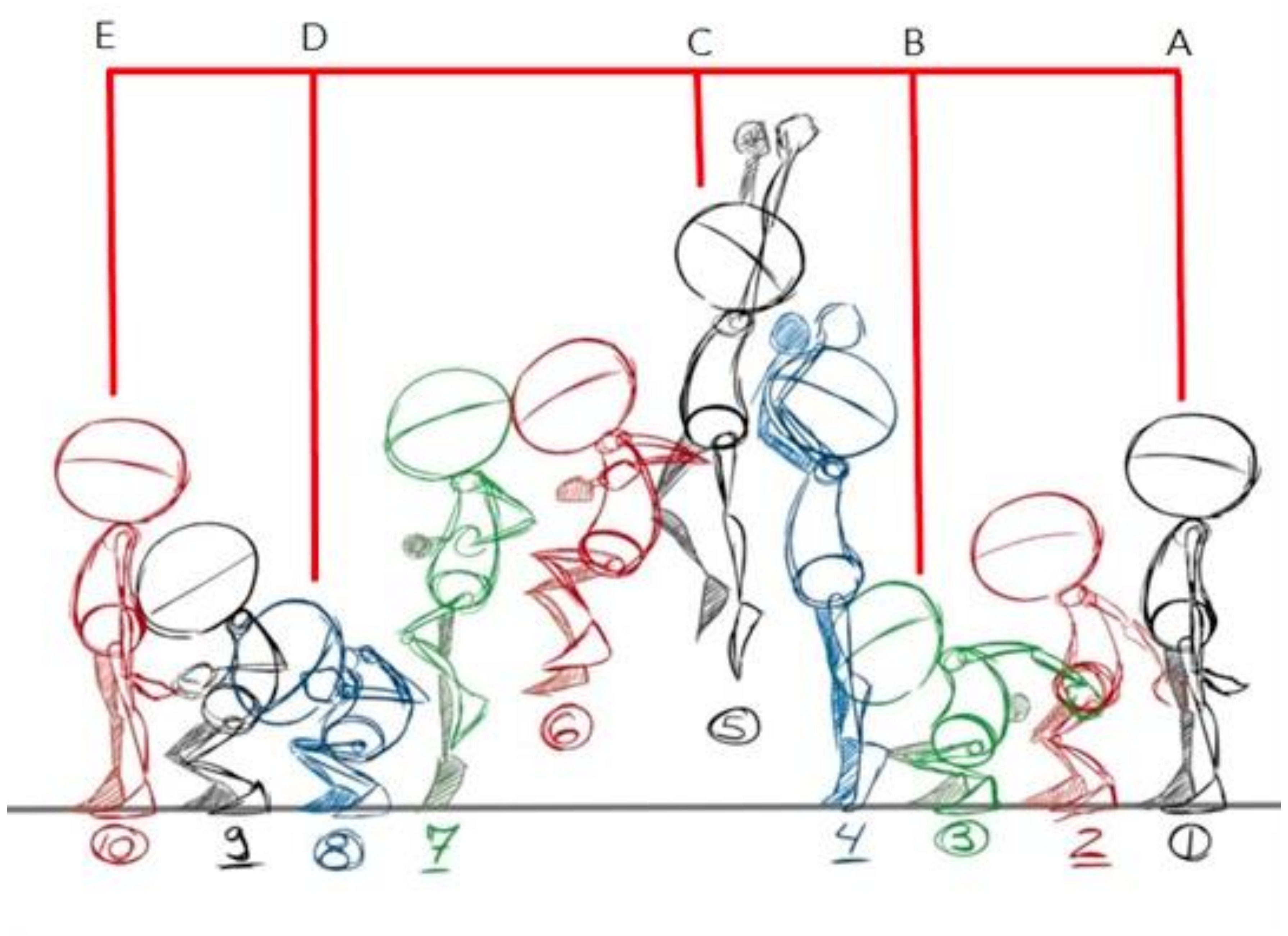
```python
def part1_key_framing(viewer, time_step, target_step):
    motion = BVHMotion('data/motion_walking.bvh')

    motio_length = motion.local_joint_positions.shape[0]
    keyframes = np.arange(0, motio_length, time_step)

    new_motion_local_positions, new_motion_local_rotations = [], []

    previous_frame_idx = 0
    for current_frame_idx in keyframes[1:]:
        between_local_pos = interpolation(motion.local_joint_positions[previous_frame_idx],
                                          motion.local_joint_positions[current_frame_idx],
                                          target_step - 1, 'linear')
        between_local_rot = interpolation(motion.local_joint_rotations[previous_frame_idx],
                                          motion.local_joint_rotations[current_frame_idx],
                                          target_step - 1, 'slerp')
        new_motion_local_positions.append(between_local_pos)
        new_motion_local_rotations.append(between_local_rot)
        previous_frame_idx = current_frame_idx
```

```python
########## Code Start ############
if method == 'linear':


    return res
elif method == 'slerp':


    return res
########## Code End ############
```

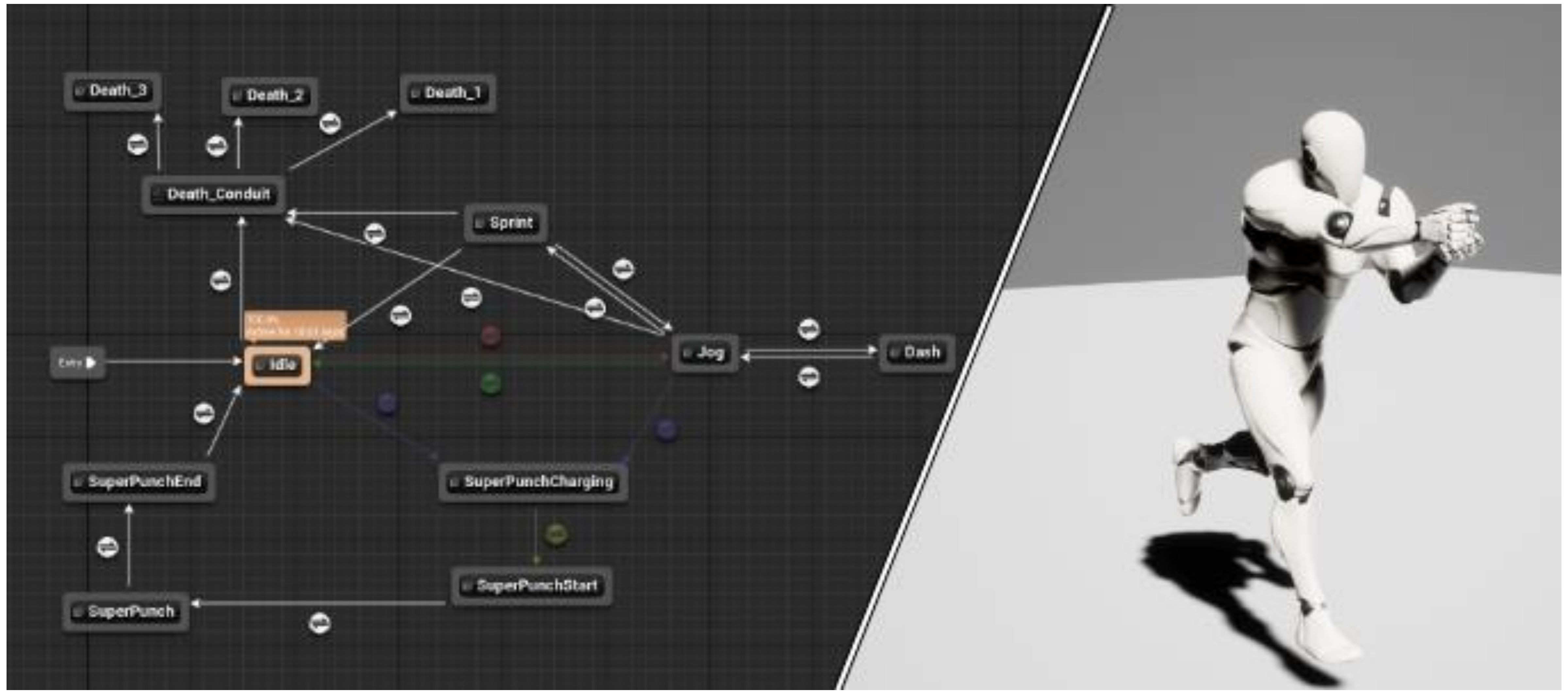
```python
# part1_key_framing(viewer, 10, 10)
# part1_key_framing(viewer, 10, 5)
# part1_key_framing(viewer, 10, 20)
# part1_key_framing(viewer, 10, 30)
```

- Linear interpolation (10%)
- Slerp Interpolation (15%)
- Report the different performance by giving different numbers (5%)

When will we use the interpolation?

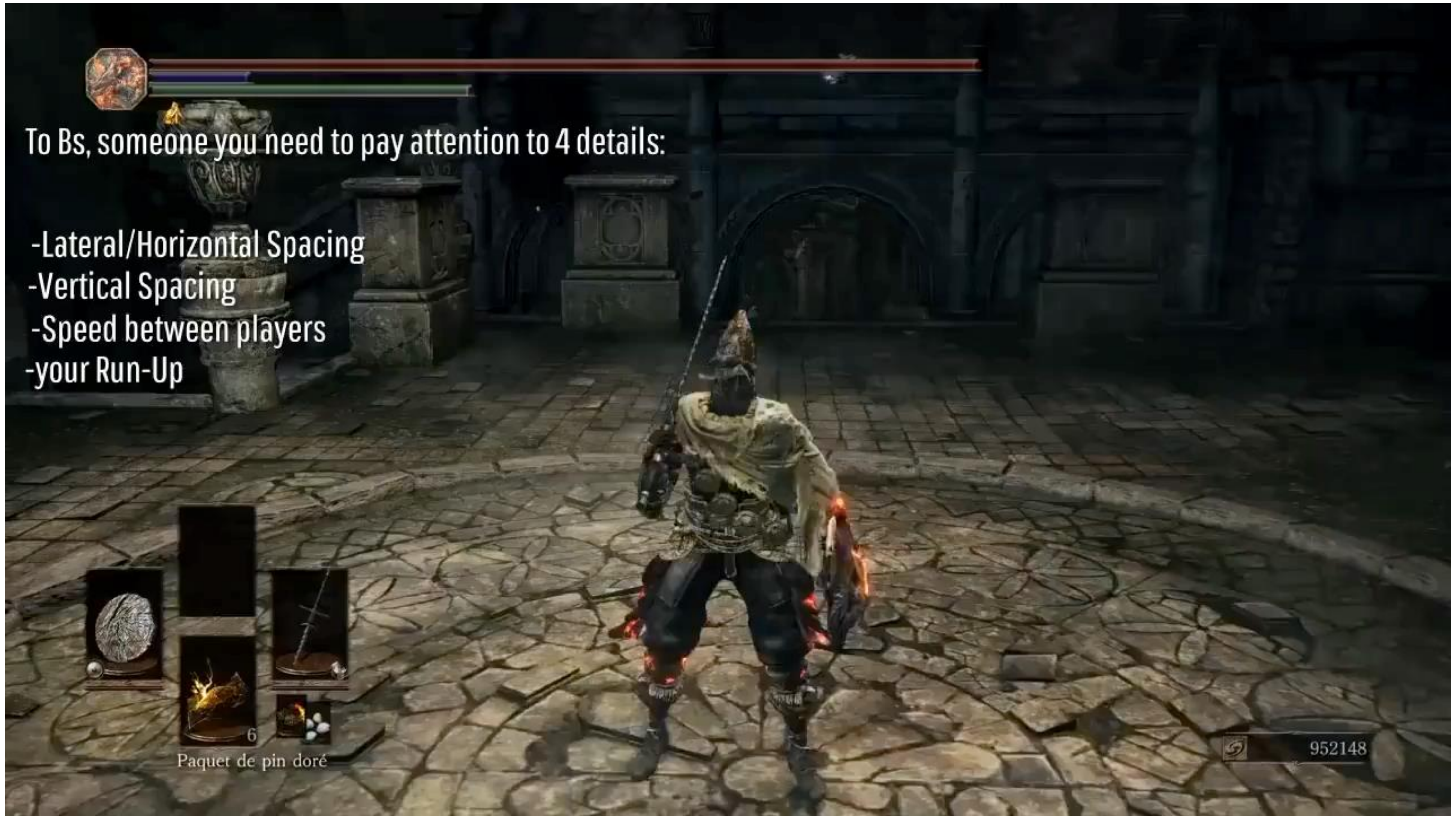


Offline production

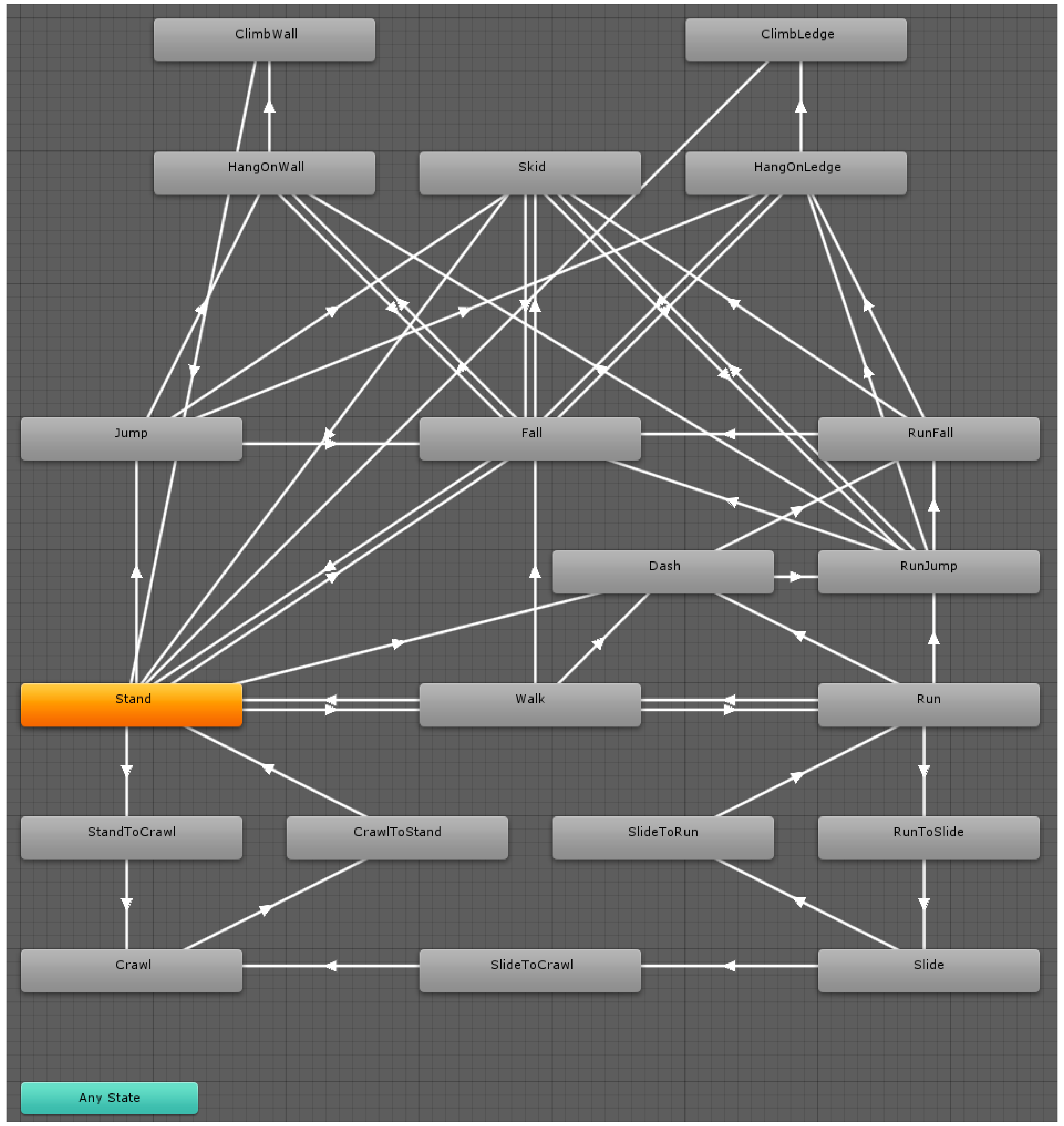


Real-time Animation

Have you played any digital Games?



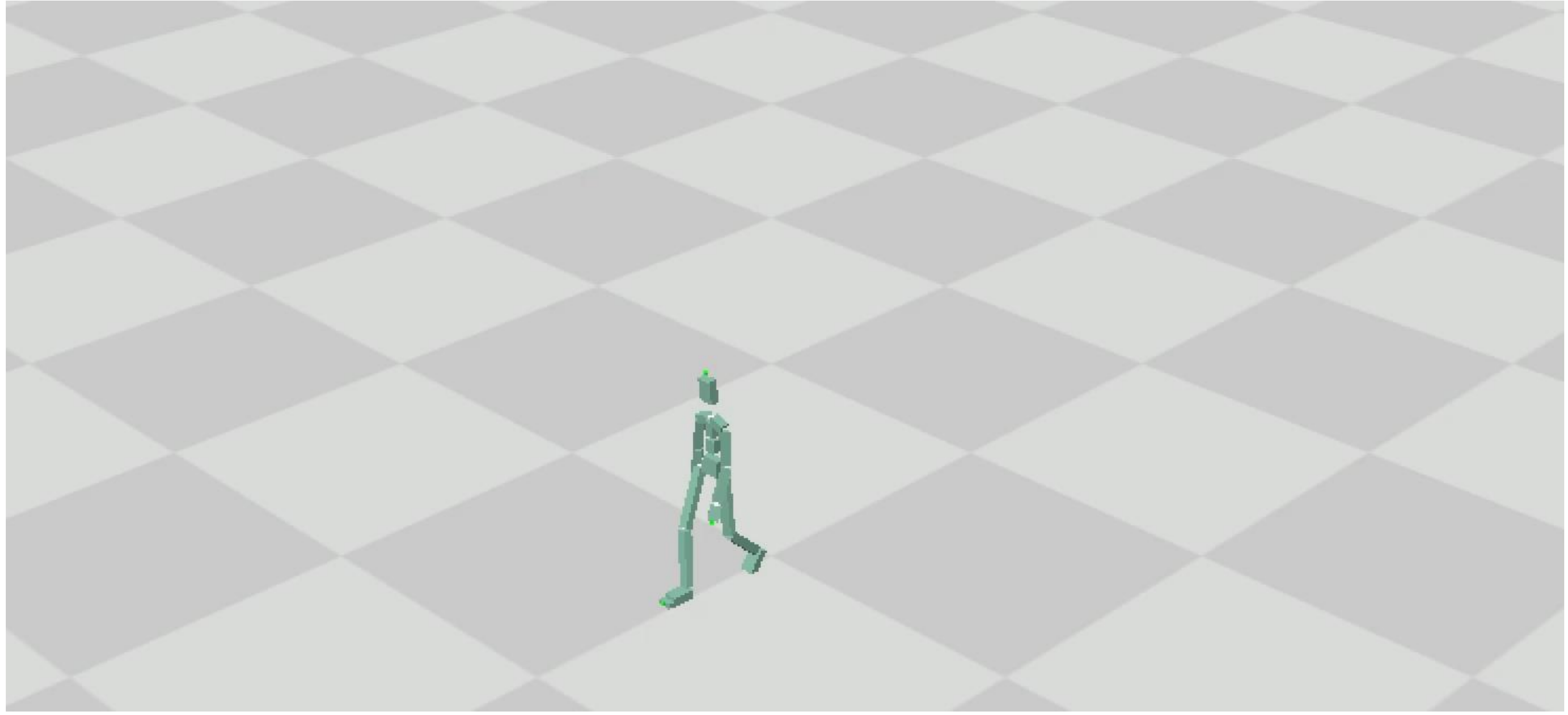
Controlling system is a combination for lots of motion clips



Character State Machine

Motion Concatenation / Transition

Given two different motions, how can we concatenate them into one?



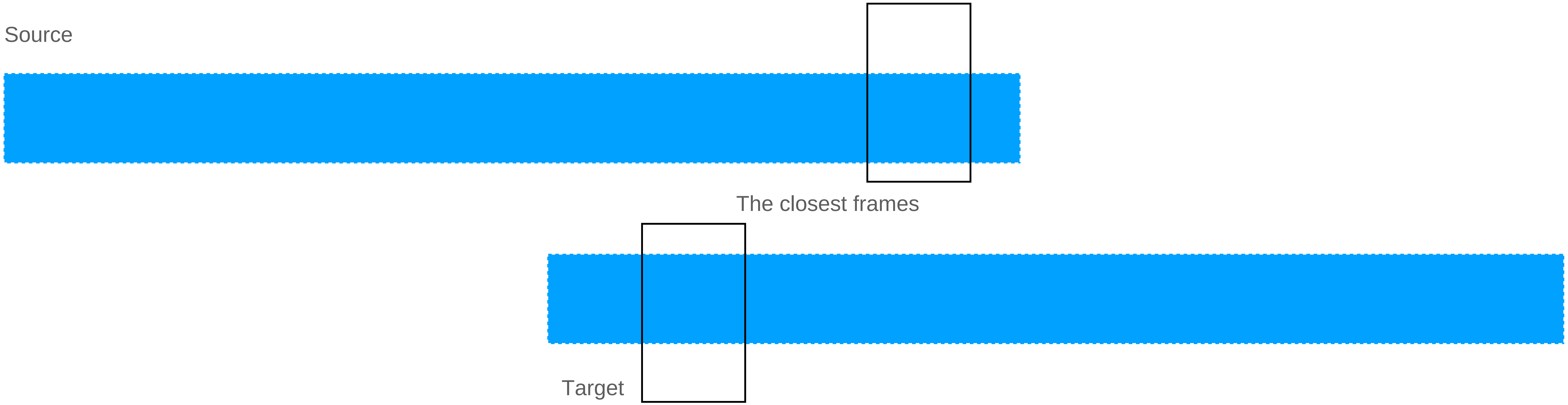
An idea: The interpolation between two frame in these two motion clips

(a simple solution, there are more advanced way in industry)

The basic Idea

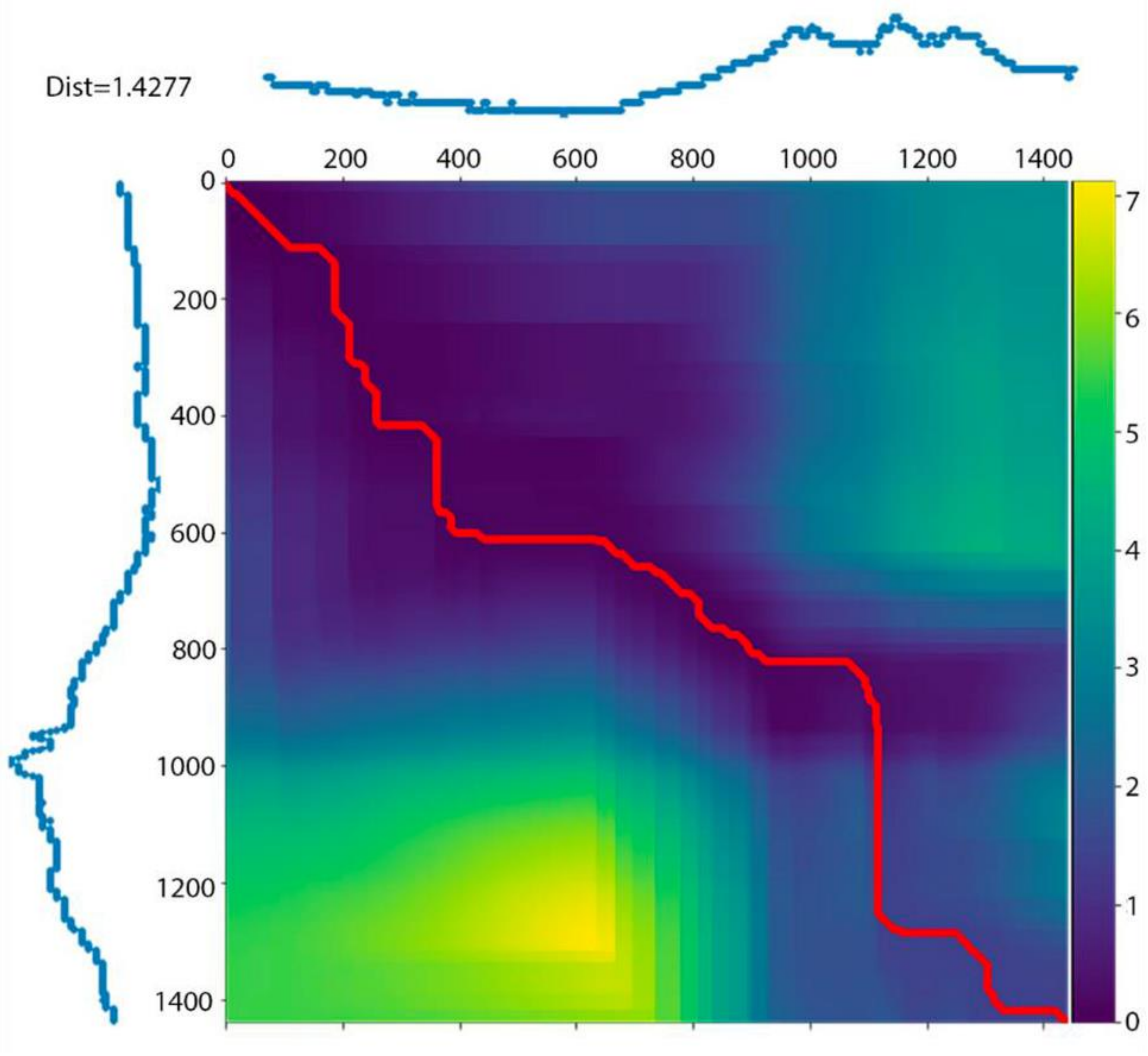
- Find a frame_i in motion 1
- Find a frame_j in motion 2
- Generate the between frames for pose_i and pose_j
- Make a new motion by motion1[:frame_i] + between + motion2[frame_j:]

Idea



1. Find the closest frames between two motions
2. Get the interpolation by these two frames as betweening poses
3. Concatenate the motion 1, betokening and motion 2 to produce a new one

How to find the closest frames?



Motion 1: 40 frames
Motion 2: 40 frames

We can calculate a similarity matrix with shape (40*40)

i -> the frame index of motion 1
j -> the frame index of motion 2

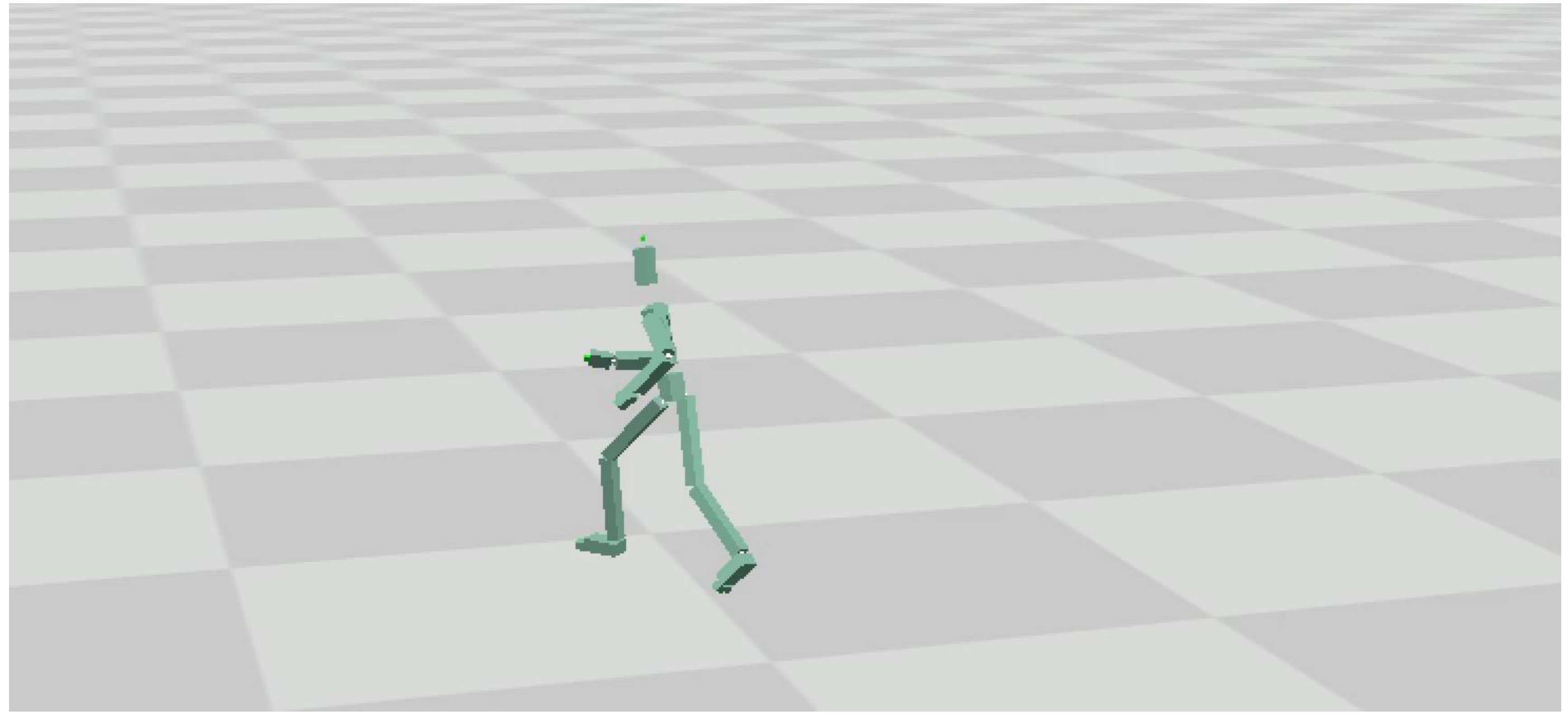
sim_m[i, j] -> the difference between frame i in motion_1 and frame j in motion_2

Then find the index of min value by np.argmin

(The similarity matrix can do lots of things, but we only choose the min value here)

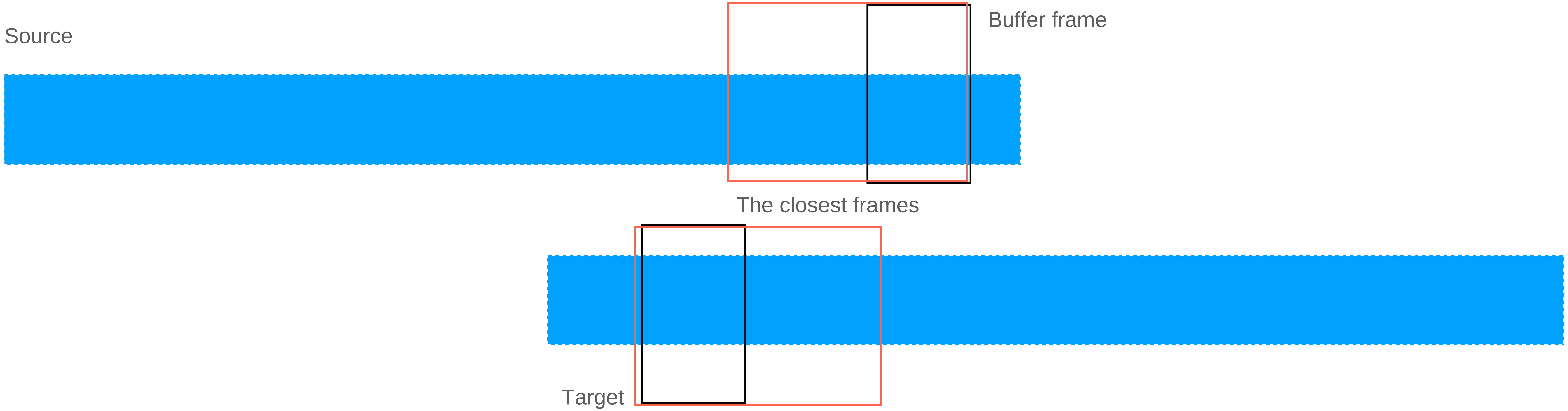
More details

- For the rotations data
  - Find the closest frame, and do the interpolation will be enough
- For the root position data
  - Shift the motion 2 to the motion 1
  - Then do the interpolation between two frames



More details (bonus)

- The velocities of motion1 and motion2 are different If we concatenate them together, it’s fine - but not the perfect because of the sharp change of velocity
- Can we use some buffer frame to shift the velocity difference?
- For example, find an average velocity of source_v and target_v, then do the interpolation



Bonus - part 2 is an open problem

- There are N between_frames, but the root position in these frames are not considered
- The velocity of two motions are not the same, it can give different weight to the two motions interpolation
- The inertialization method provides the best results
  - ref:https://theorangeduck.com/page/spring-roll-call#inertialization
- Any way to produce more smooth and natural transitions

- Thinking about the above questions will help you in part 3 of assignment 2.

Assignment 2 - Part 2

```
walk_forward = BVHMotion('data/motion_walking.bvh')
run_forward = BVHMotion('data/motion_running.bvh')
run_forward.adjust_joint_name(walk_forward.joint_name)

last_frame_index = 40
start_frame_indx = 0

if not example:
    motion = concatenate_two_motions(walk_forward, run_forward, last_frame_index, start_frame_indx, between_frames, method='interpolation')
```

```
########## Code Start ############
# search_win1 =
# search_win2 =

# sim_matrix =
# min_idx =
# i, j = min_idx // sim_matrix.shape[1], min_idx % sim_matrix.shape[1]
# real_i, real_j =

# between_local_pos =
# between_local_rot =

########## Code End ############
```

- Define the search window (10%)
- Calculate the sim_matrix (10%)
- Find the real_i and real_j (10%)
- The shifting on the root joint position (5)

https://github.com/LamWS/COMP7508_Data_Driven_Animation/assignment_2/task1_motion_editing.py

Overview

- part1_key_framing (30%)
    - - Linear interpolation (10%); Slerp Interpolation (15%)
    - - Report the different performance by giving different numbers (5%)
- part2_concatenate (35%)
    - - Define the search window (10%); Calculate the sim_matrix (10%); Find the real_i and real_j (10%); The shifting on the root joint position (5)
    - - Any improvements will get bonus

https://github.com/LamWS/COMP7508_Data_Driven_Animation/assignment_2/task1_motion_editing.py