Lecture 08

Animation System

Basics of Animation Technology

WANG XI

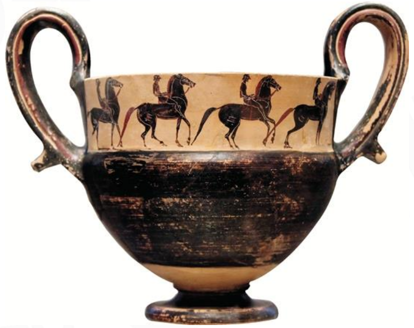
GAMES 104

2022

Humans have been trying to represent object in motion

Stone age mural in the Cave of Altamira

Painted Pottery of Majiayao Culture

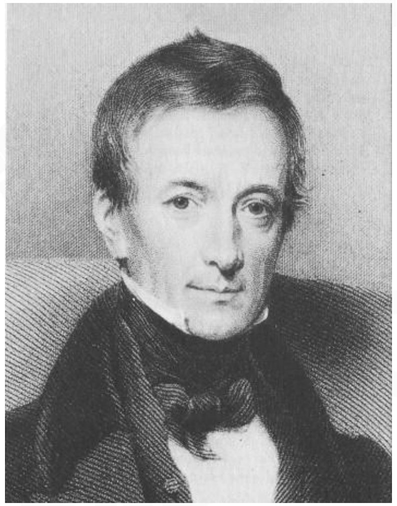


Ancient Greek Pottery

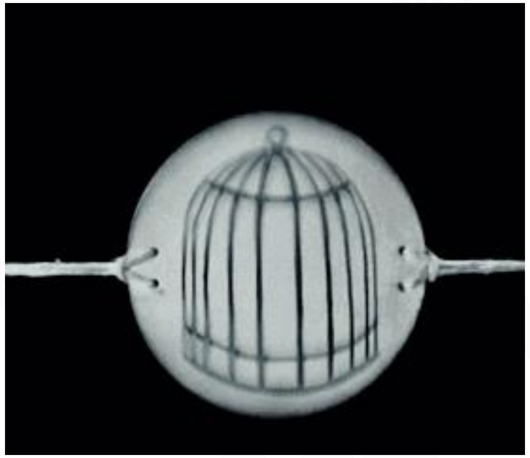
Temple of Isis

Humans have been trying to represent object in motion

- The persistence of vision



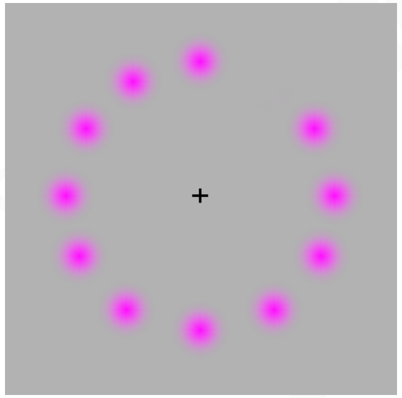
Discoverer of "*the persistence of vision*"
Peter Mark Roget(1779-1869)



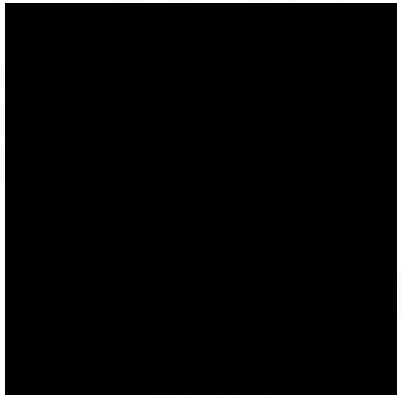
Thaumatrope

- Illusory motion

Discoverer of "*phi phenomenon*"
Max Wertheimer (1880-1943)

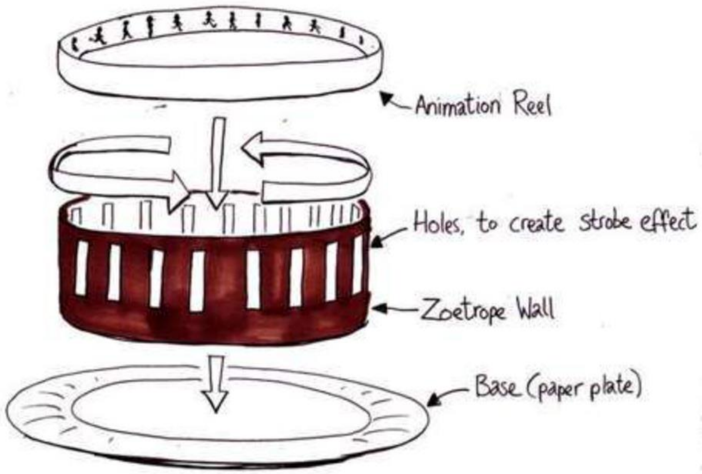


Phi phenomenon



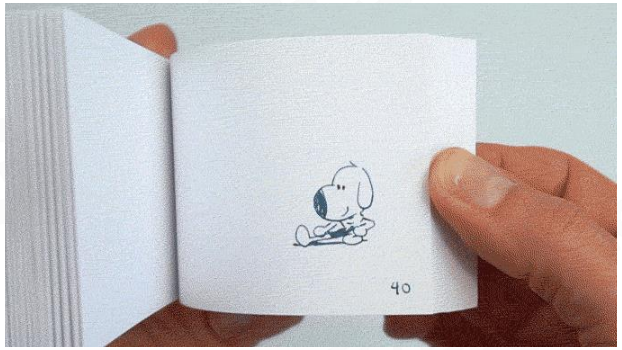
Beta movement

Humans have been trying to represent object in motion



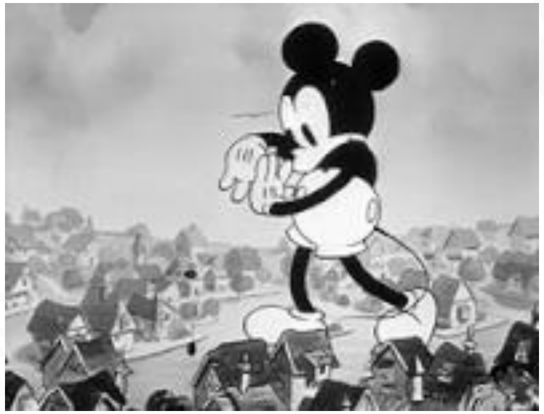
Zoetrope

Praxinoscope



Flip Book

Animation Techniques in Film



Gulliver Mickey(1934)

Westworld(1973)

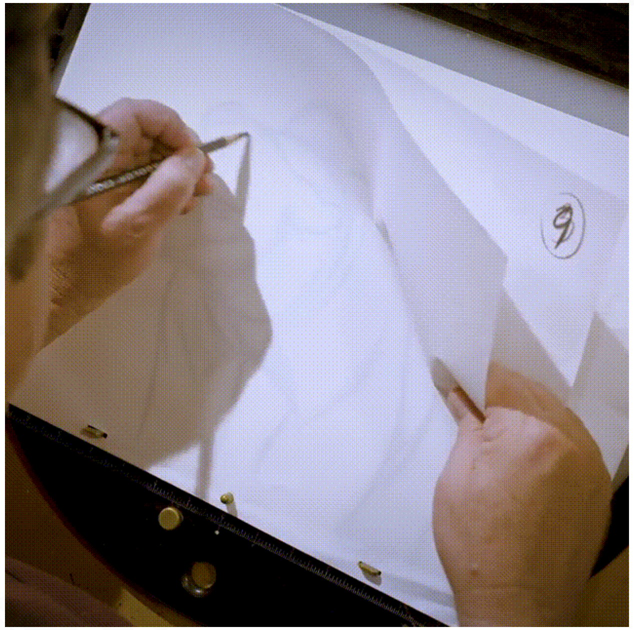
Futureworld(1976)

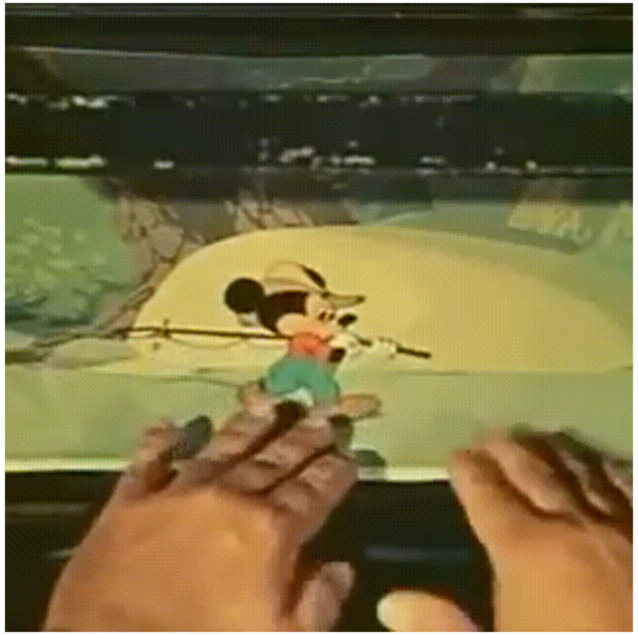
Jurassic Park(1993)

Avatar(2009)

Zafari(2018)

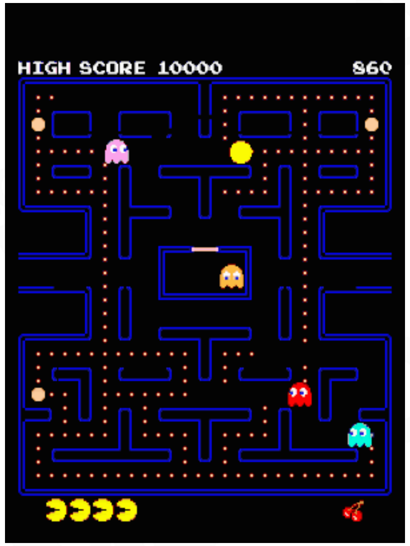
Animation Techniques in Film


Hand Draw Animation


Cel Animation

Computer Animation

Animation Techniques in Game



Sprite Animation in *Pac-Man*(1980)

Sprite Animation Based on Rotoscoping in *Prince of Persia*(1989)

Multi-view Sprite Animation in *Doom*(1993)

Animation Techniques in Game

Rigid Hierarchy Animation in *Resident Evil*(1996)

Soft Skinned Animation in *Half-life*(1998)

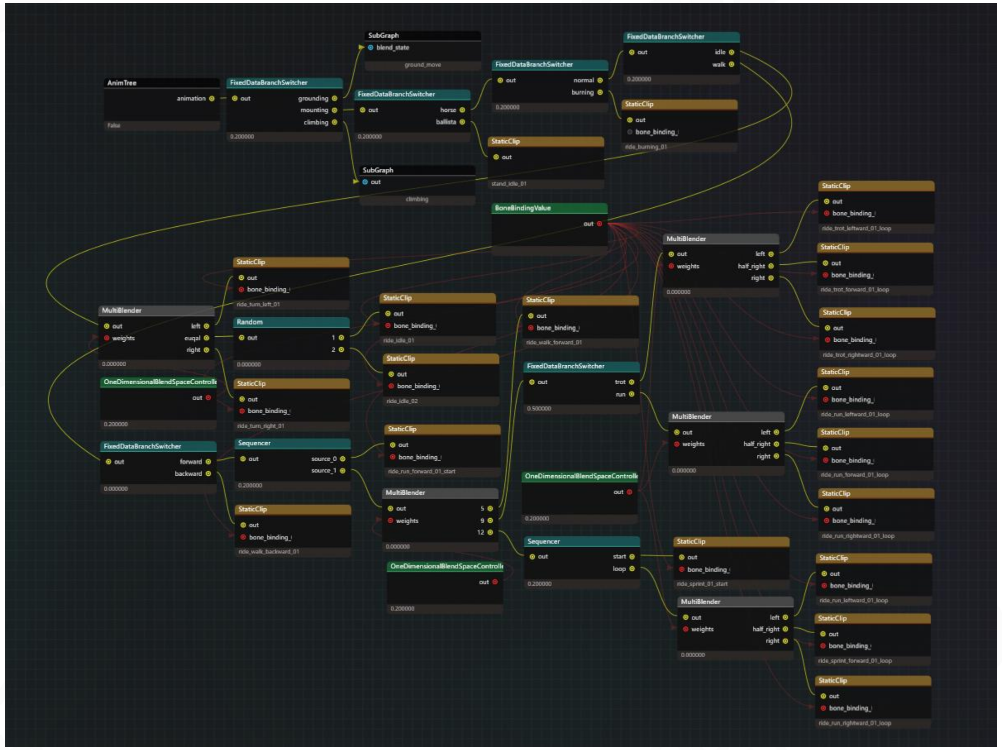
Physics Animation in *Uncharted 4*(2016)

Challenges in Game Animation (1/3)

Interactive and dynamic animation

- Vary according to the interaction
- Cooperate with other gameplay systems
- Make adjustments in complex environments

Climbing in *Uncharted 4*

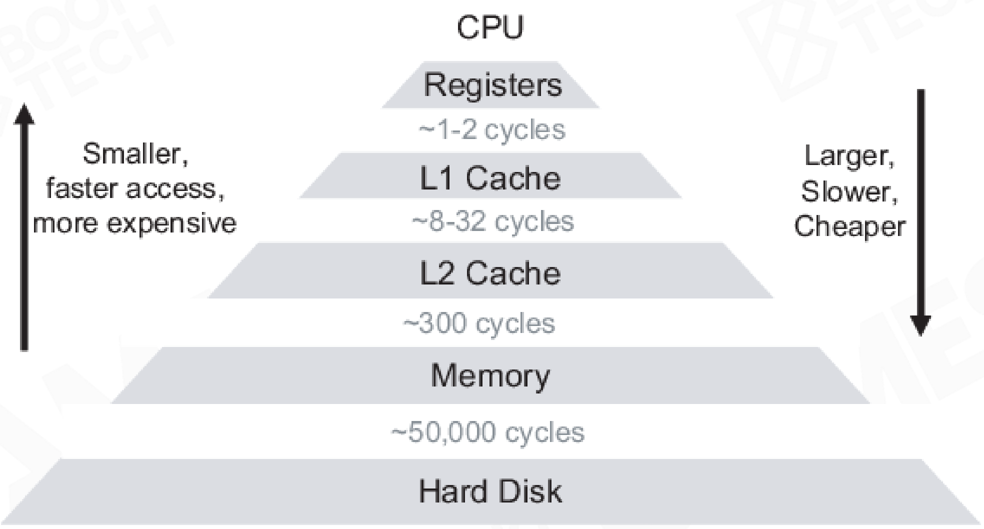


Animation tree example

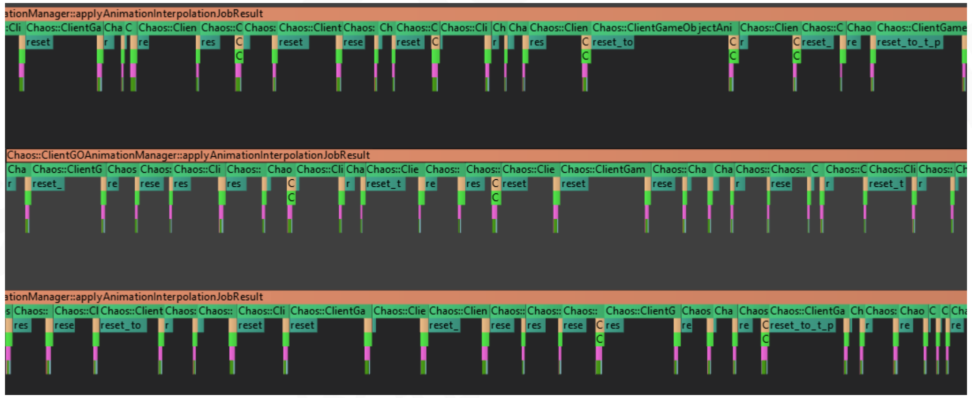
Challenges in Game Animation (2/3)

Real-time

- Compute per frame
- Massive animation data(Disk and memory)



Memory hierarchy



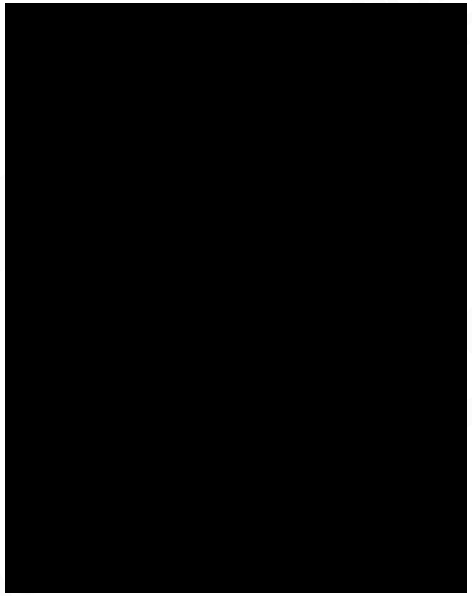
Profiling Data of Animation System

Challenges in Game Animation (3/3)

Realism

- More vivid expression
- More authentic experience

Facial Animation in *Witcher 3*



Ragdoll Physics in *Uncharted 4*

Motion Matching in *For Honor*

Outline of Animation System

01. Basics of Animation Technology

- 2D Animation
- 3D Animation
- Skinned Animation Implementation
- Animation Compression
- Animation DCC

02. Advanced Animation Technology

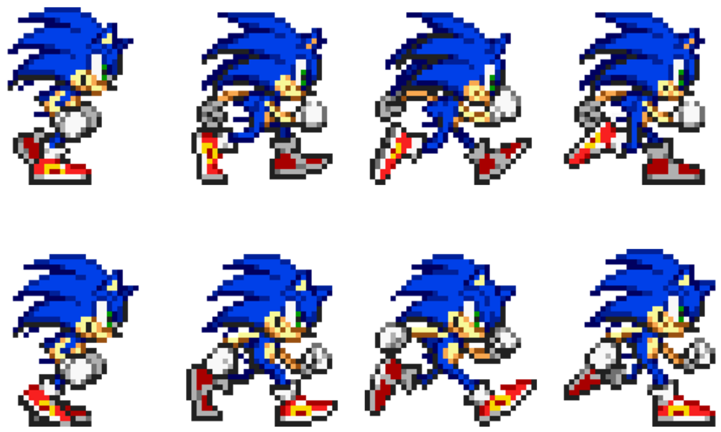
- Animation Blend
- Inverse Kinematics
- Animation Pipeline
- Animation Graph
- Facial Animation
- Retargeting

2D Animation Techniques in Games

2D Animation - Sprite animation

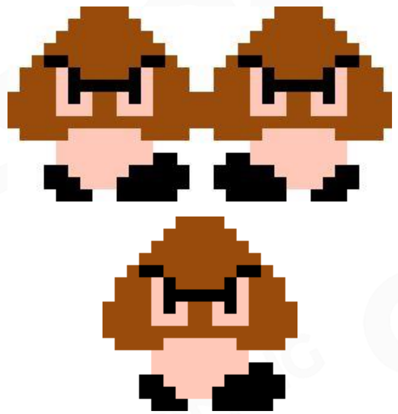
The electronic equivalent to cel animation

- A sprite is a small bitmap that can be overlaid on top of a background image without disrupting it
- The sequence of frames was designed so that it animates smoothly even when it is repeated indefinitely



Sprite Sequence

Loop



The mushroom in *Super Mario Bros.*

The Sprite-like animation technique in pseudo-3D game

A sprite sheet in *Doom*

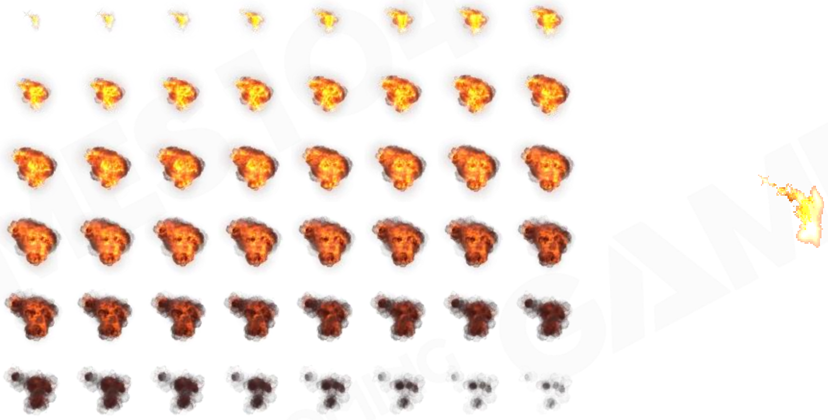
Multiple views of the sprite animation

Sprite animation in *Doom*

Sprite Animation in Modern Game

Application

- 2D character
  - Sprite on 2D background image
  - Sprite on top of 3D rendered environment
- Game effect
  - Sprite sheet texture for particles



Sprite animation for particle effect

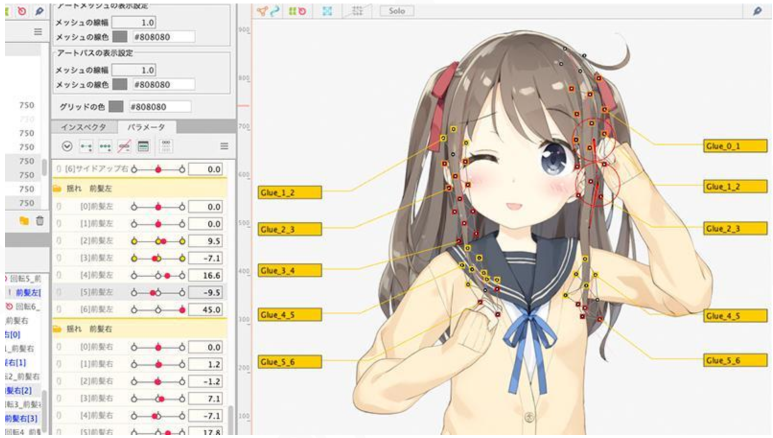
2D character with 3D environment in *Octopath Traveler*

2D game *To the Moon*

Live2D

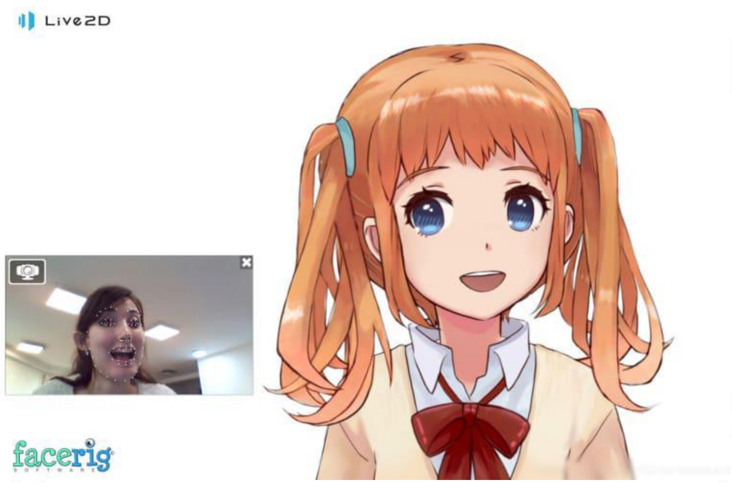
A technology to generate 2D animation without 3D model

- Usually refer to eponymous software series epmloying the technology created by Live2D Ltd.
- Could develop dynamic character, especially anime-style character without a 3D model.



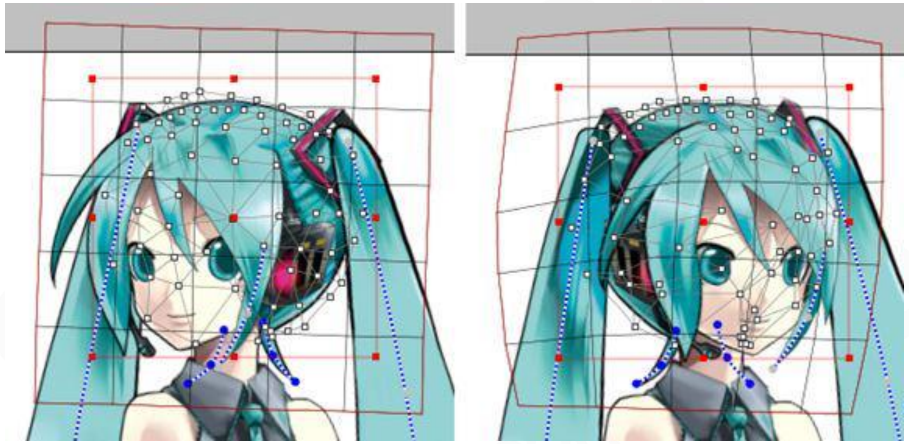
Live2D

- By applying translation, rotation and transformation to different parts and layers of image.
- Combined with real-time motion capture, could be used for vtubing



Combined with motion capture


Divide origin image into parts

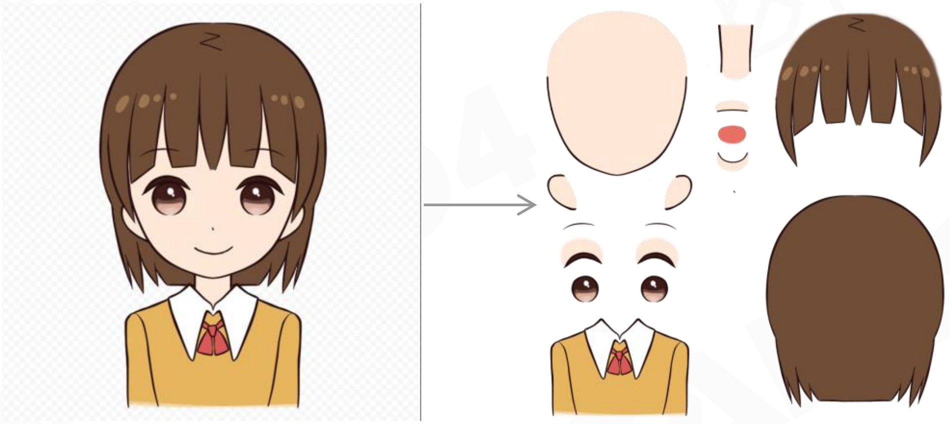

Transform of image

Live2D

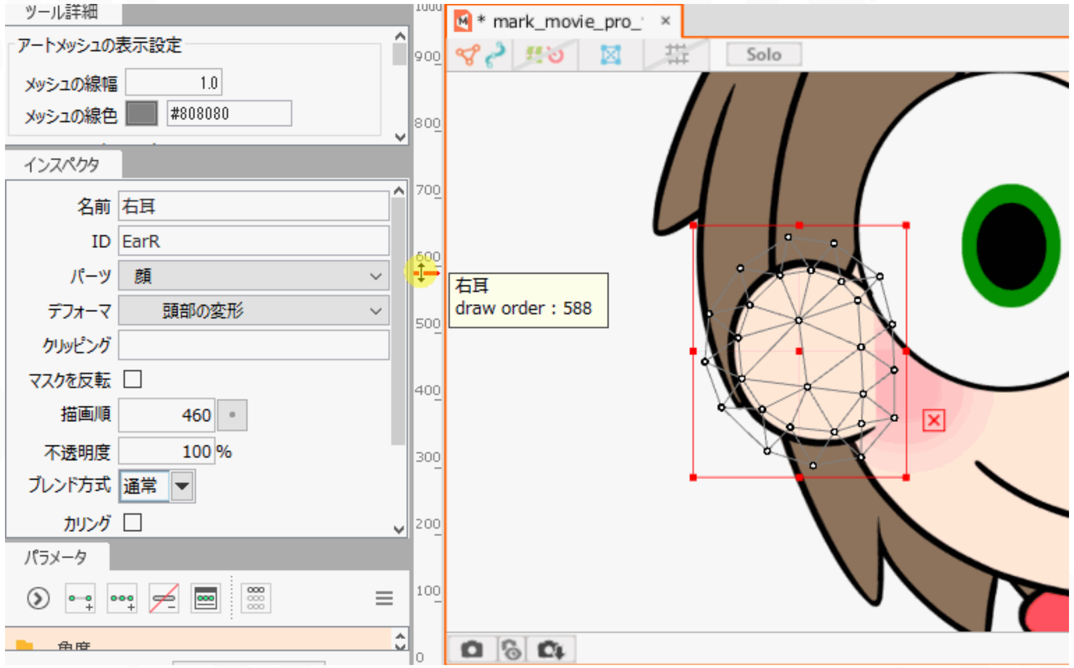
Make a Live2D animation

Prepare resources

- Dividing origin character image into different parts
- Set "draw order" to each parts for further use



Divide Image into parts



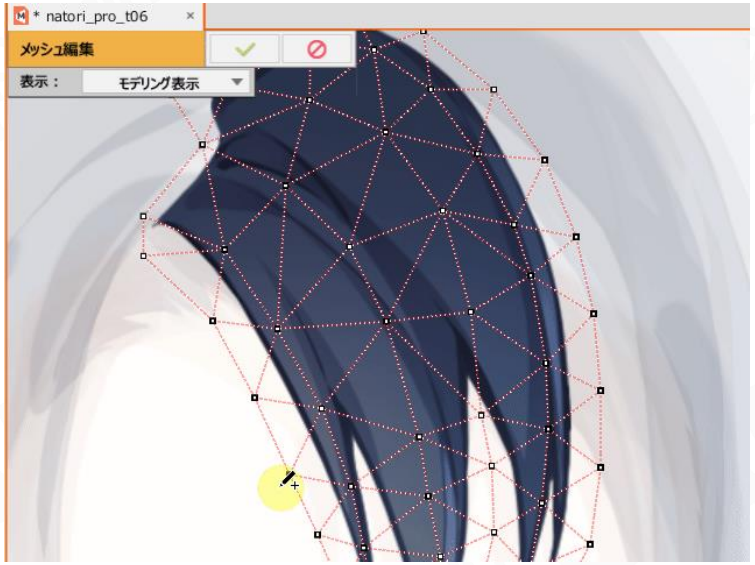
Determine "draw order"

Live2D

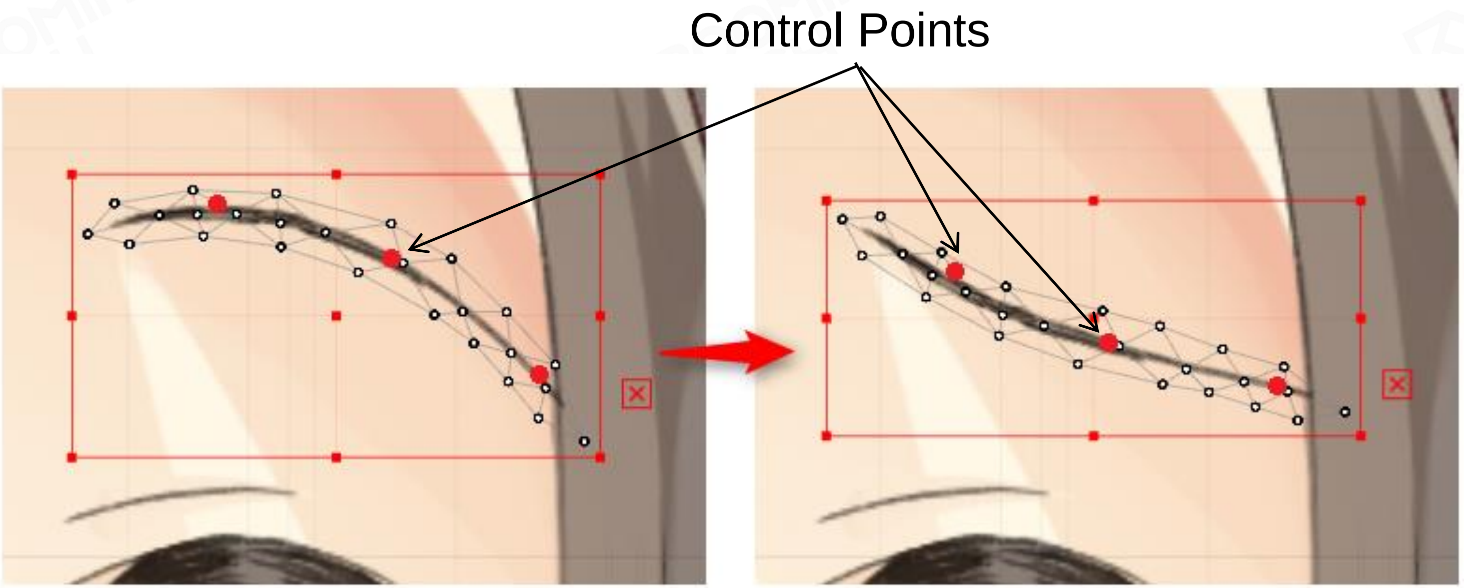
Make a Live2D animation

Transform image by using control points for parts

- "ArtMesh" could be automately generated for each parts, which defined by vertices, edges and polygons
- Control points could be used to help transforming "ArtMesh"



Adjust automately generated "ArtMesh"



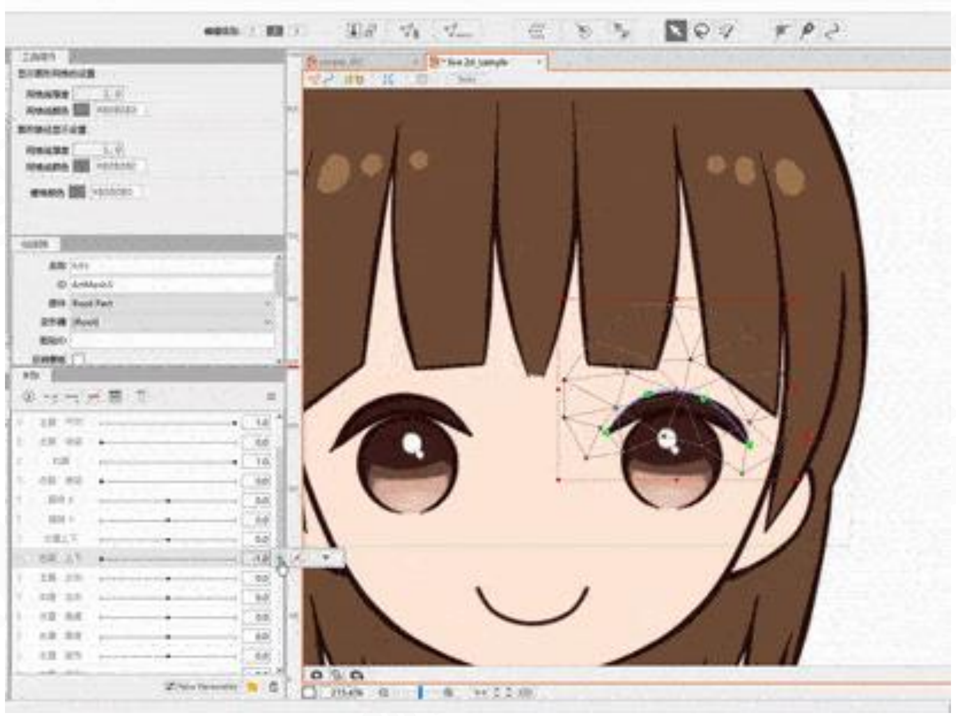
Use control point to change "ArtMesh"

Live2D

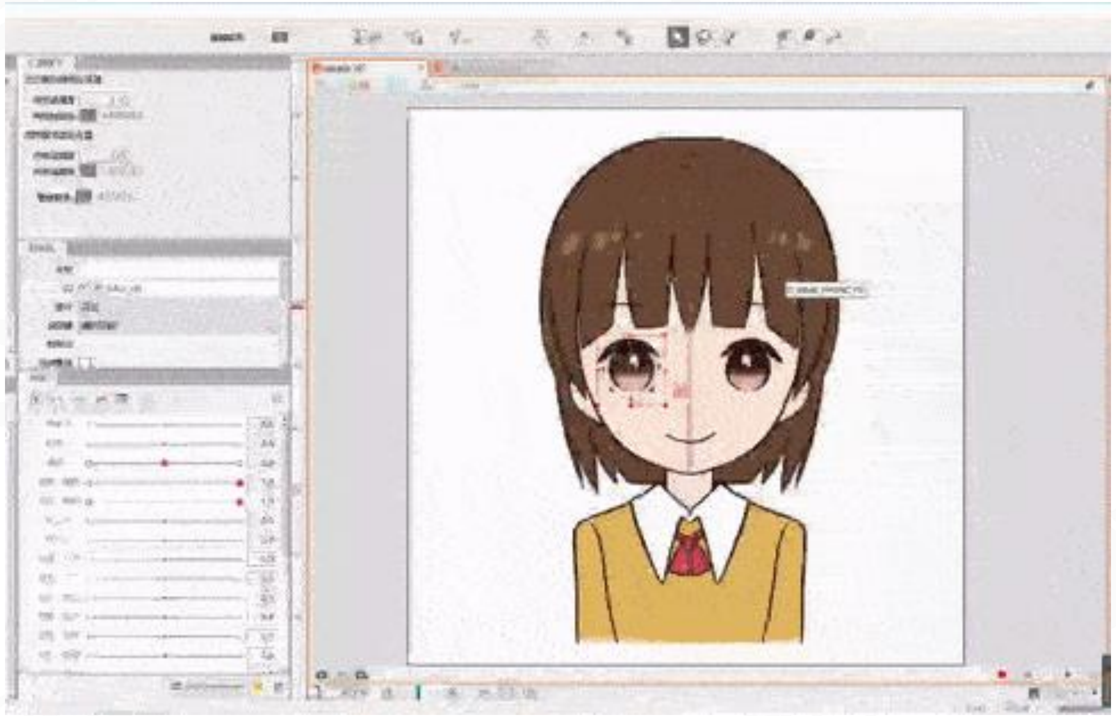
Make a Live2D animation

Set animation "key frame"

- Set "key frame" to help animation interpolation



Set "key frame" of animation

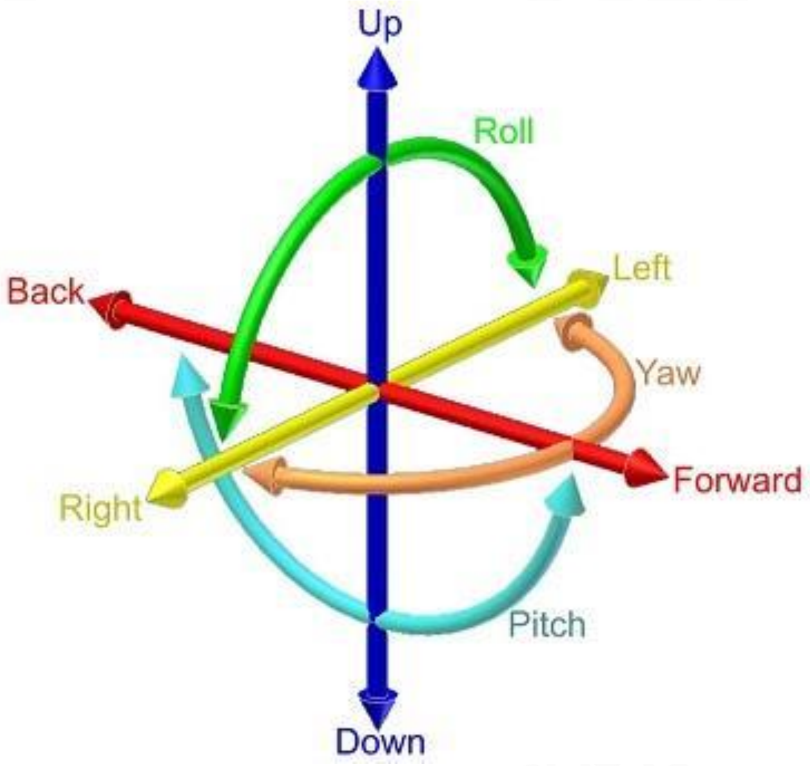


Final animation

3D Animation Techniques in Games

DoF(Degrees of Freedom)

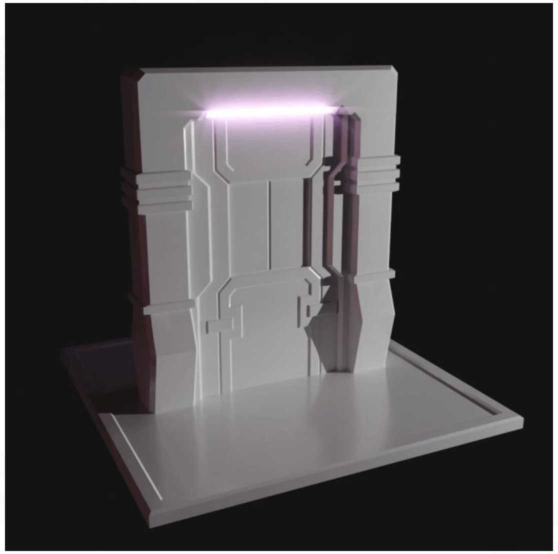
- refers to the number of independent variables or parameters of a system



6 DoFS

DoF For rigid objects

- 6 DoFs per object or sub-part


Translation of door

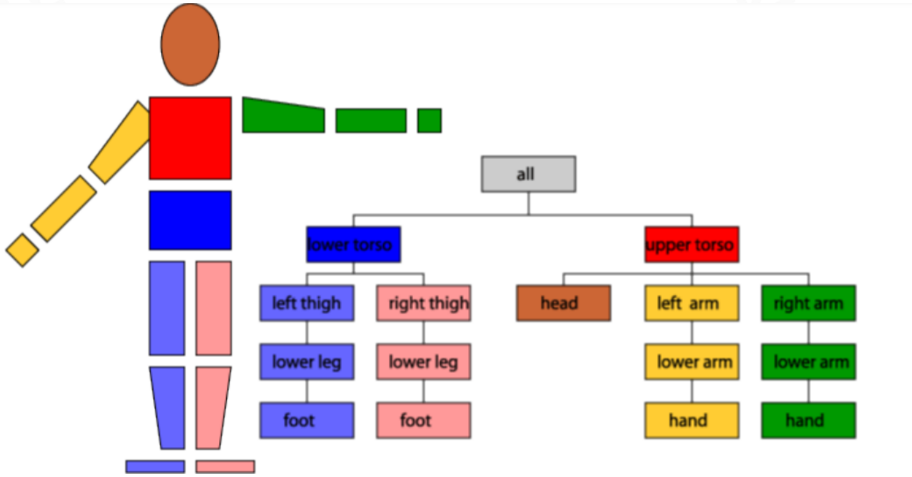

Rotation of windmill


Translation and rotation of tyre

Rigid Hierarchical Animation

- The earliest approach to 3D character animation
- A character is modeled as a collection of rigid pieces
- The rigid pieces are constrained to one another in a hierarchical fashion

Chinese leather-silhouette show



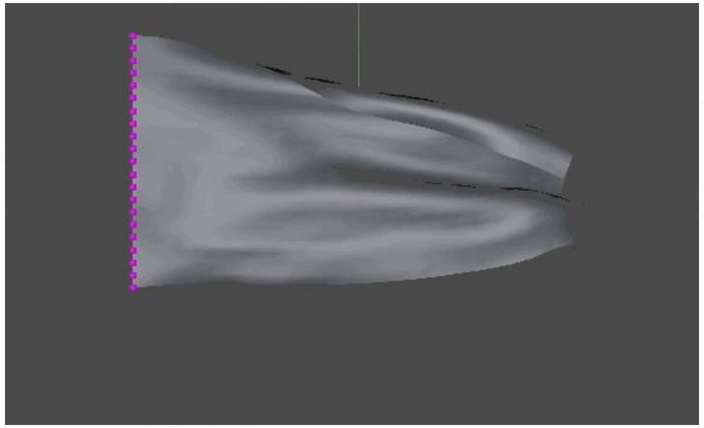
General rigid hierarchy

Rigid hierarchical animation in *Resident Evil*(1996)

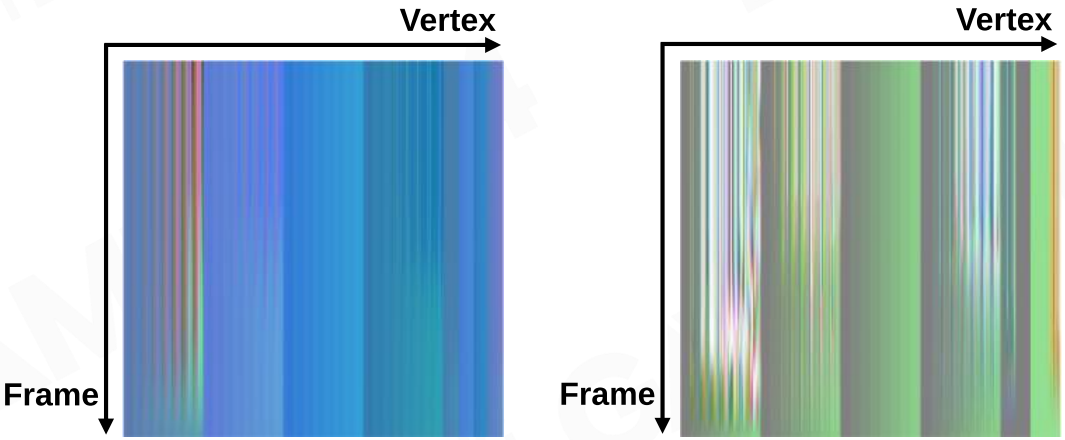
Cracking at the joints

Per-vertex Animation

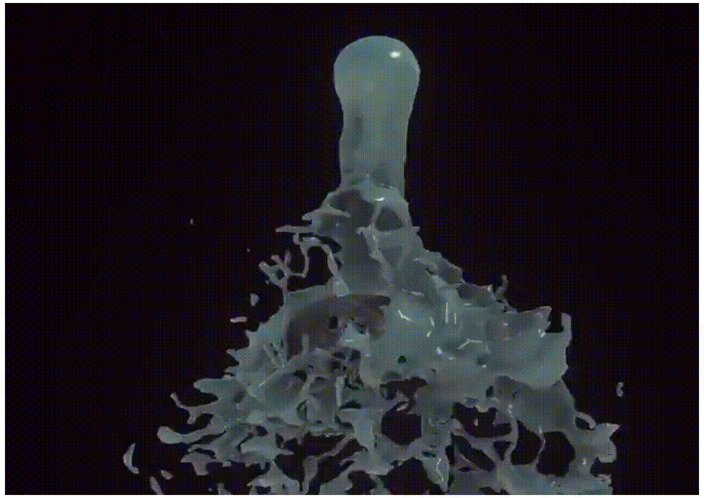
- Most flexible(3 DoFs per vertex)
- Mostly implemented by Vertex Animation Texture(VAT)
- Suitable for complex morphing
- Need massive data



Cloth Vertex Animation



An example of VAT: Translation Texture(left) and Rotation Texture(right), e.g. the texel at (5,8) stores the data at the 9th frame for vertex with index 5



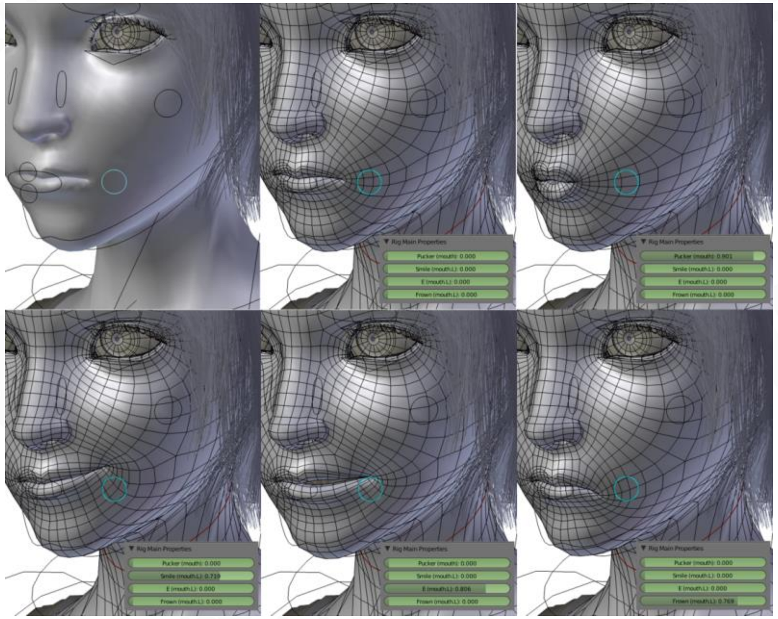
Fluid Vertex Animation

Morph Target Animation

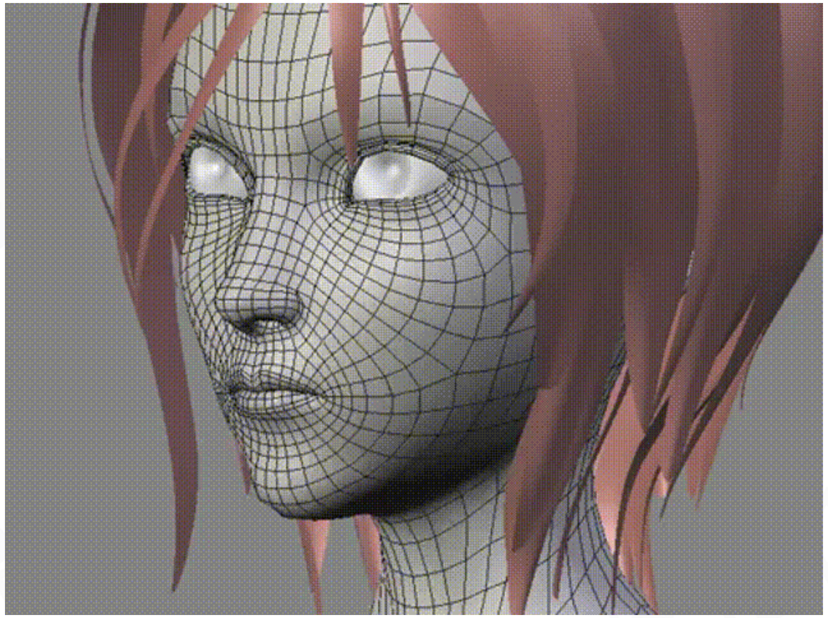
- A variation on Per-vertex Animation
  - Use key frames with LERP instead of sequence frames(e.g. 30 frames per second)
- Suitable for facial expression

+ Key Frame LERP

Per-Vertex Animation → **Morph Target Animation**



Key Poses



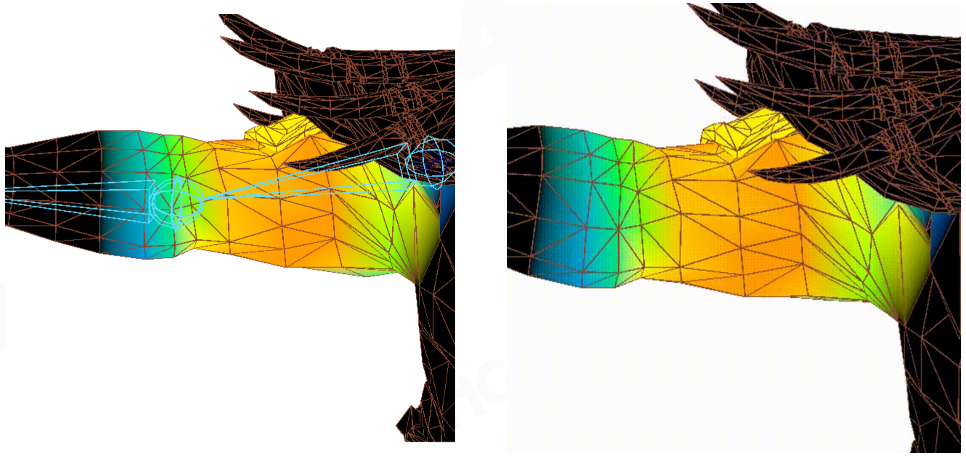
Play with LERP

3D Skinned Animation

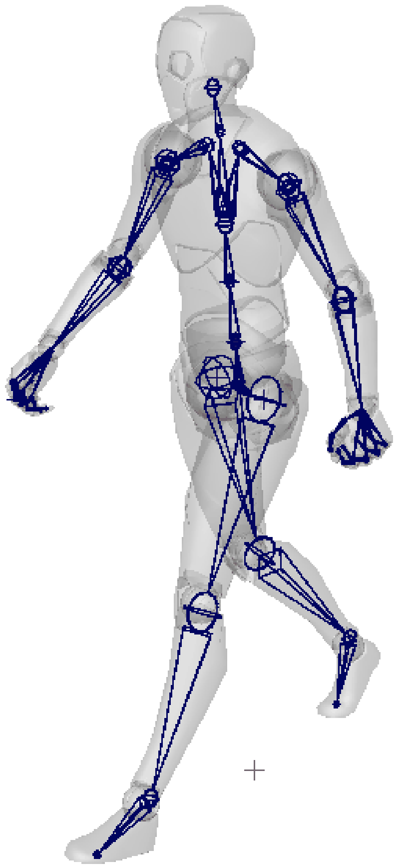
- Mesh(or skin) is bound to the joints of the skeleton
- Each vertex can be weighted to multiple joints

Advantages

- Need less data than per-vertex animation
- Mesh can be animated in a natural way (like human "skin")



Animation is continuous and smooth at joints



A Skinned Animation of Walking

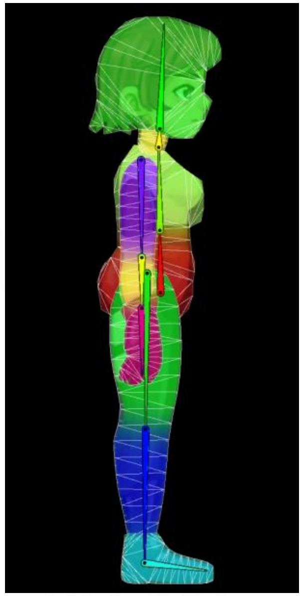
2D Skinned Animation

Derived from 3D skinned animation

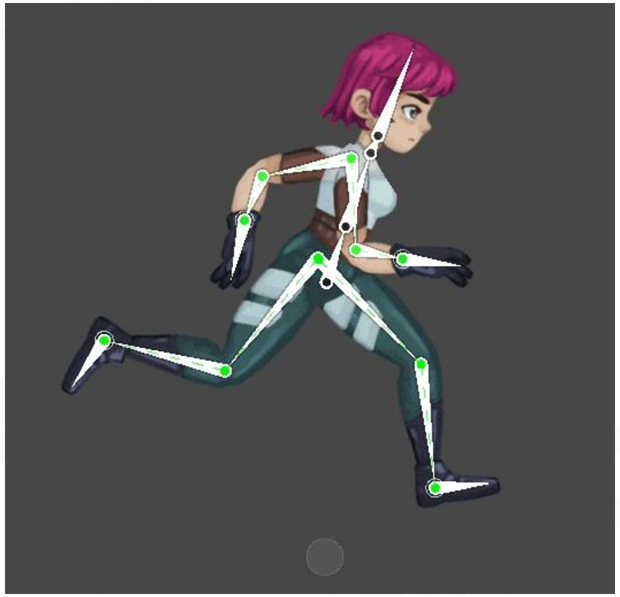
- Break up character into various body parts
- Create body part meshes and piece them together
- Rigging, skinning and animation

Body parts

Combine parts



Rigging and Skinning



Animation

Physics-based Animation

- Ragdoll
- Cloth and Fluid simulation
- Inverse Kinematics(IK)

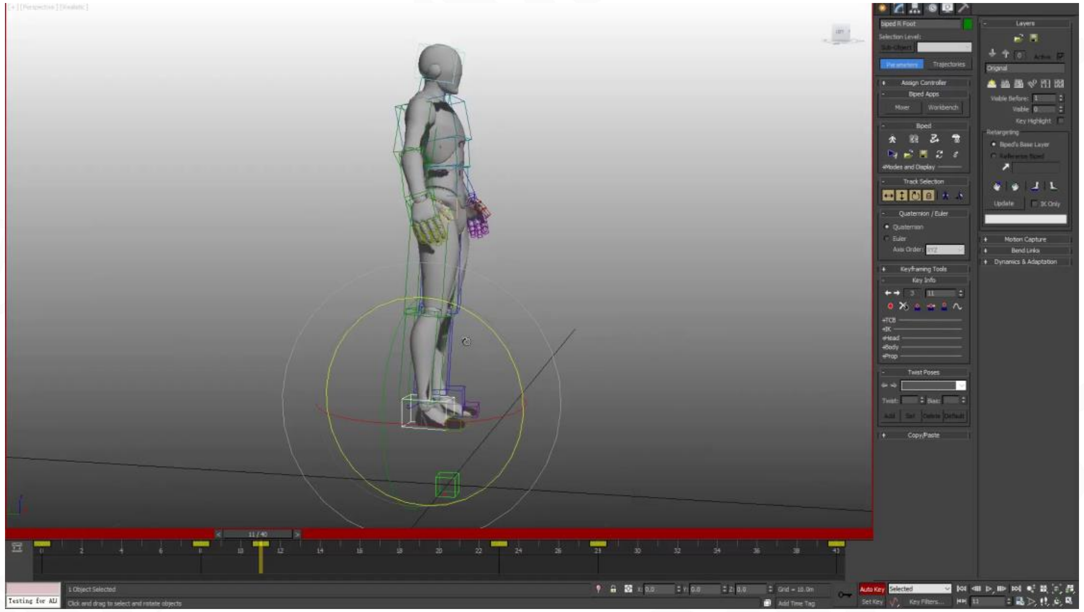
Ragdoll in *GTA 5*

Cloth in Valley of The Ancient from UE5

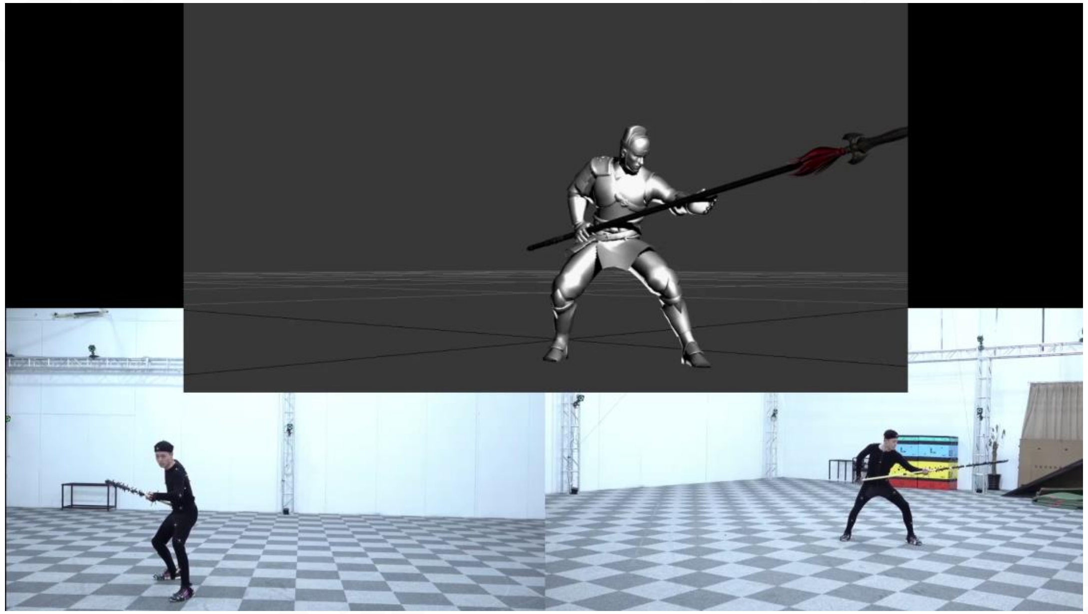
Animation with IK in *Uncharted 4*

Animation Content Creation

- Digital Content Creator + Animator
- Motion Capture



Animation from DCC



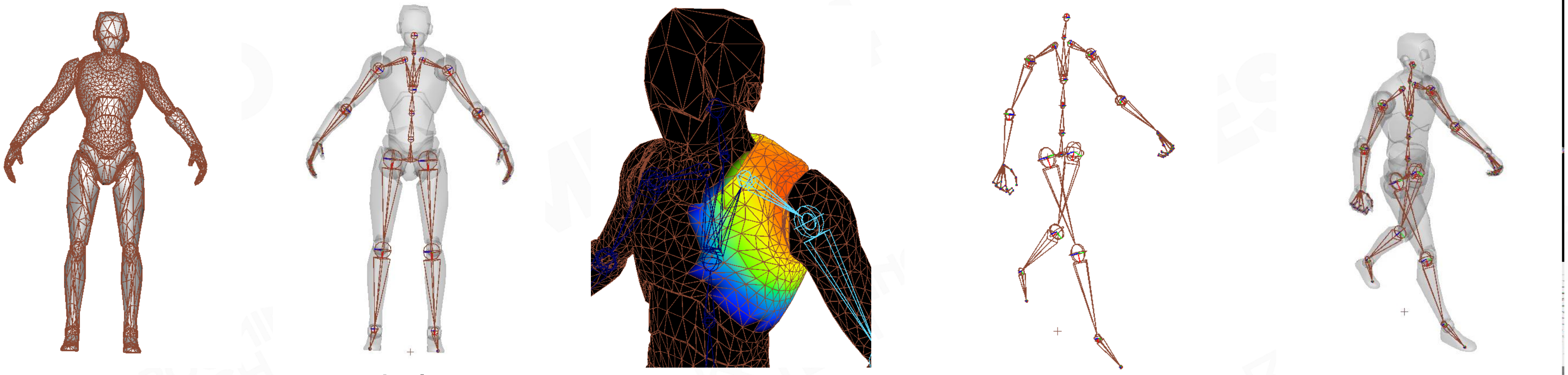
Animation from Motion Capture

Skinned Animation Implementation

How to Animate a Mesh

1. Create mesh for a binding pose
2. Create a binding skeleton for the mesh
3. “Paint” per-vertices skinning weights to related skeleton
4. Animate skeleton to desired pose
5. Animate skinned vertices by skeleton and skinning weights

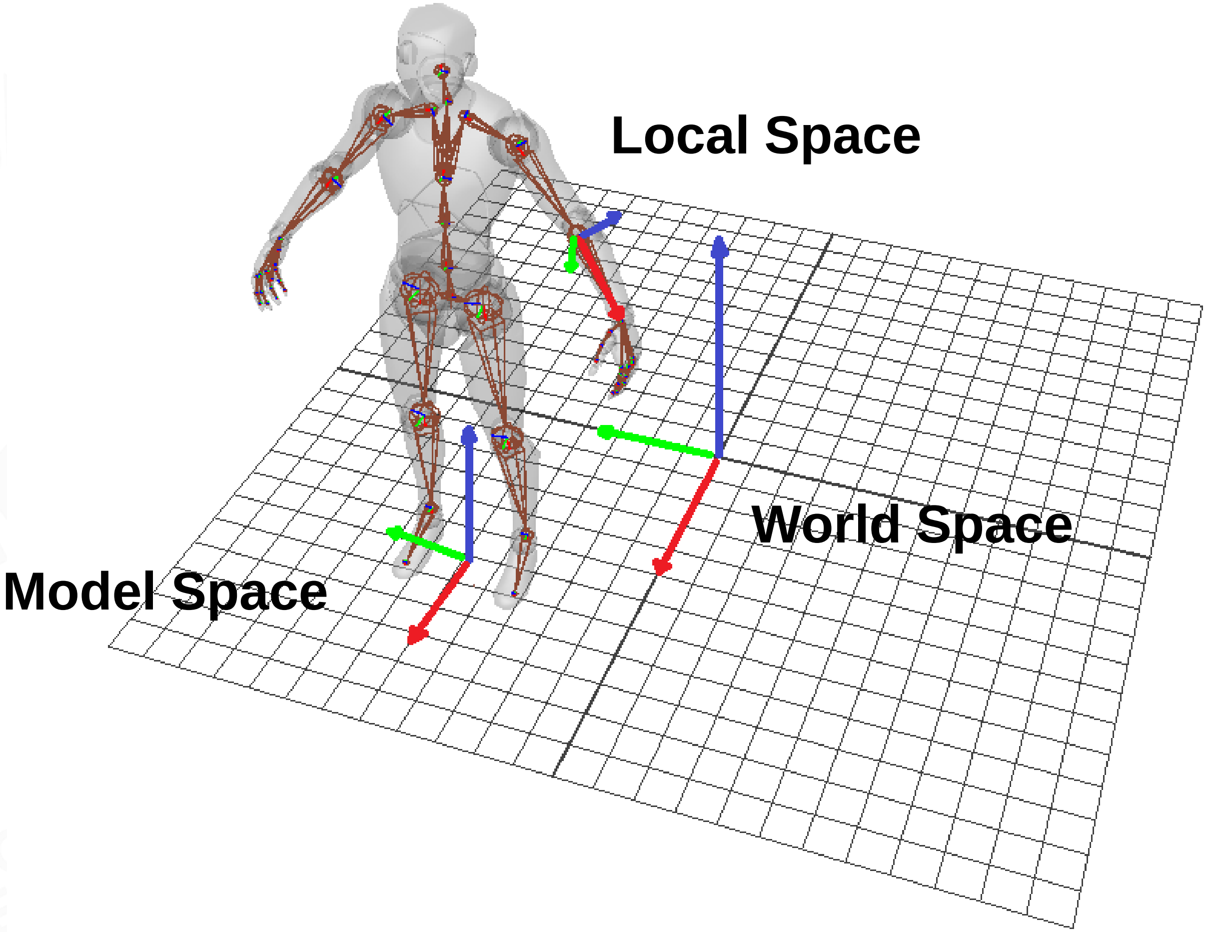
Sounds Easy but Not Simple



Mesh Skeleton Skinning/Rig Pose Animation

Different Spaces

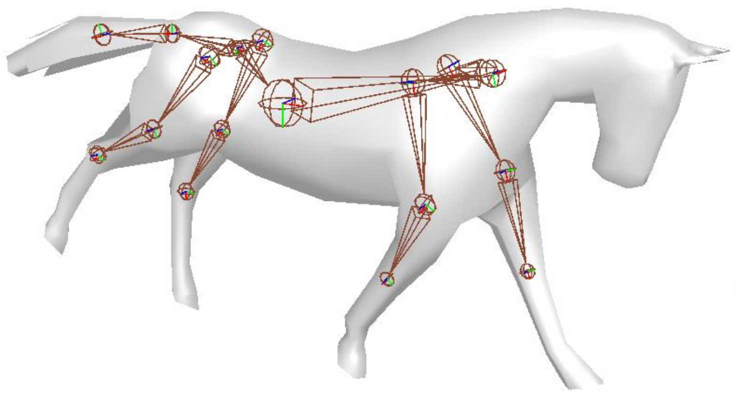
- Local space , Model space and World space



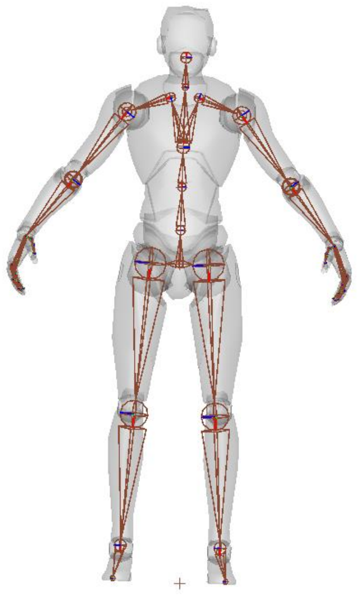
Skeleton for Creatures

Comprised of a hierarchy of rigid pieces known as joints

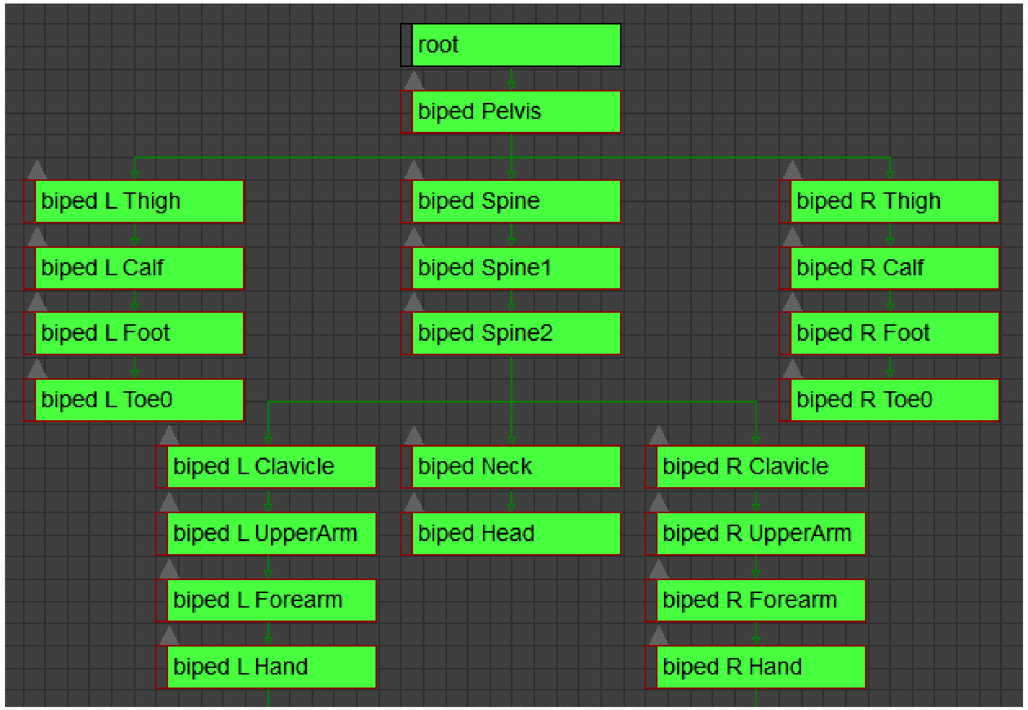
- One joint is selected as the root
- Every joint has a parent joint except the root



Non-humanoid Skeleton



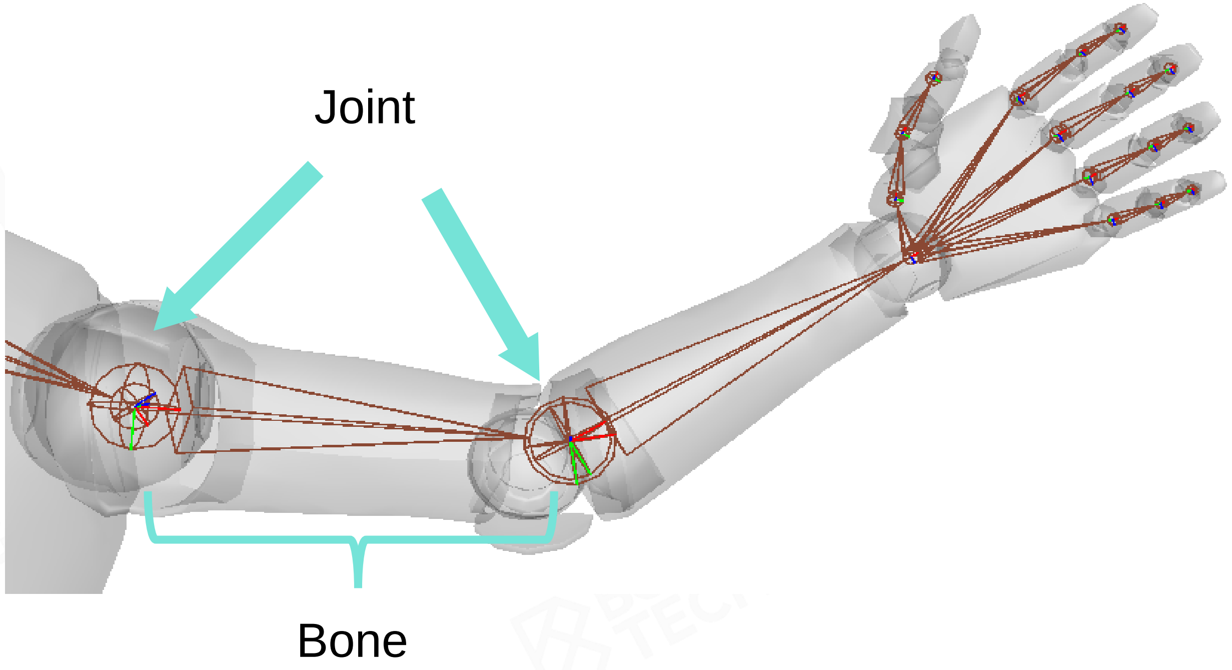
Humanoid Skeleton



Skeleton Hierarchy

Joint vs. Bone

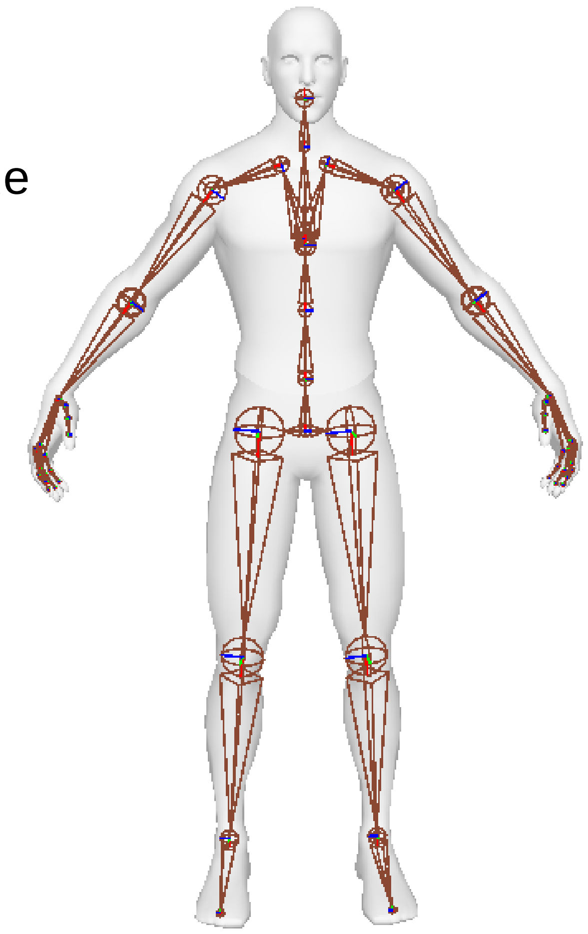
- The joints are the objects directly manipulated by the animator to control motion
- The bones are the empty space between the joints



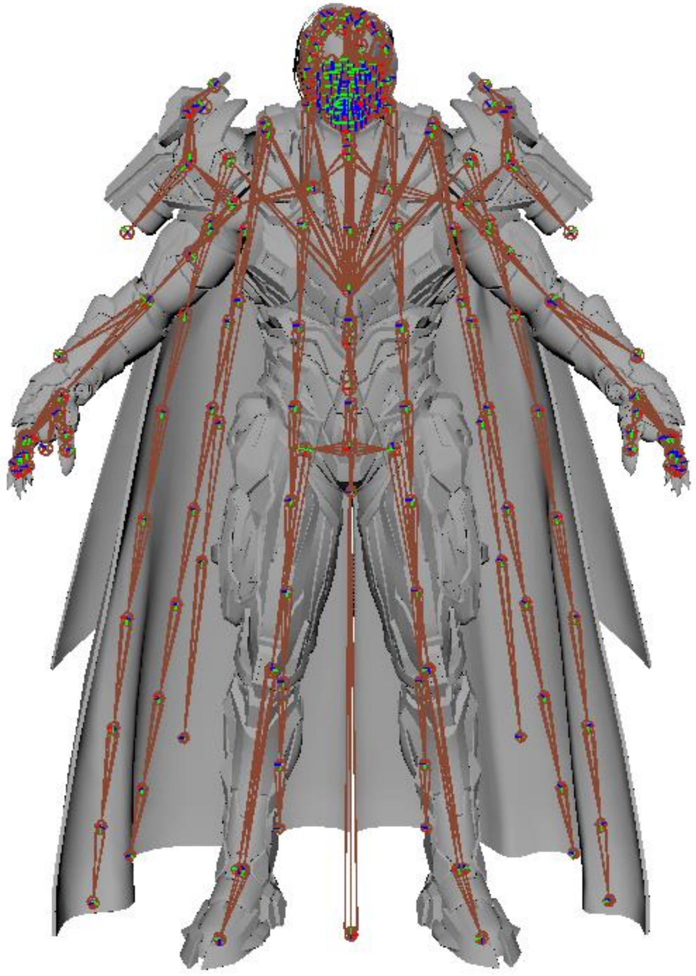
Humaniod Skeleton in Real Game

Number of joints in a humaniod skeleton

- Normal : 50~100 joints
- May more than 300+ joints include facial joints and gameplay joints



58 joints

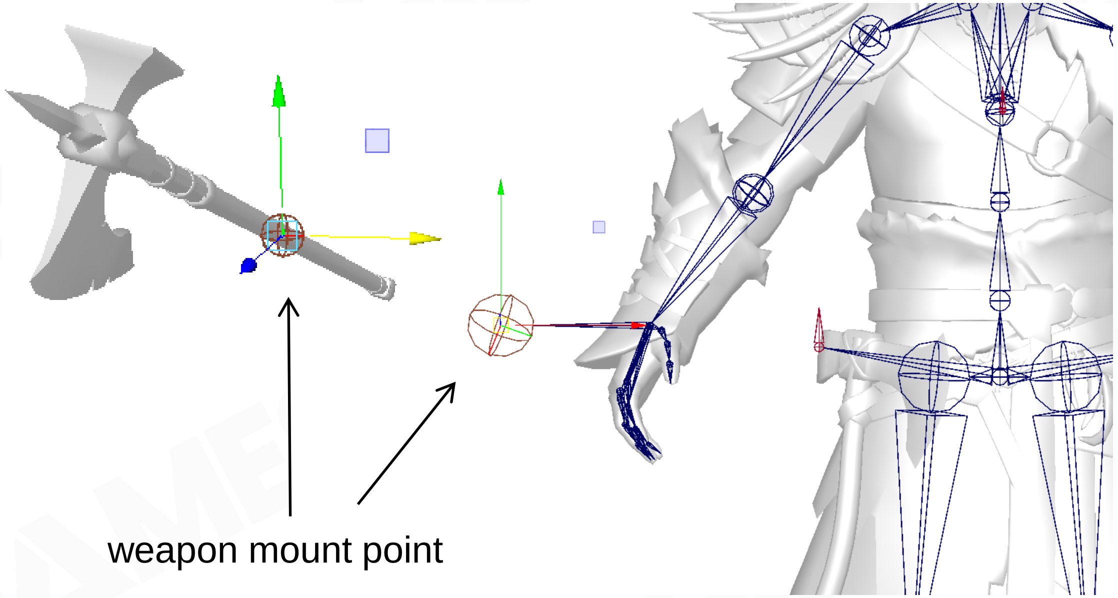


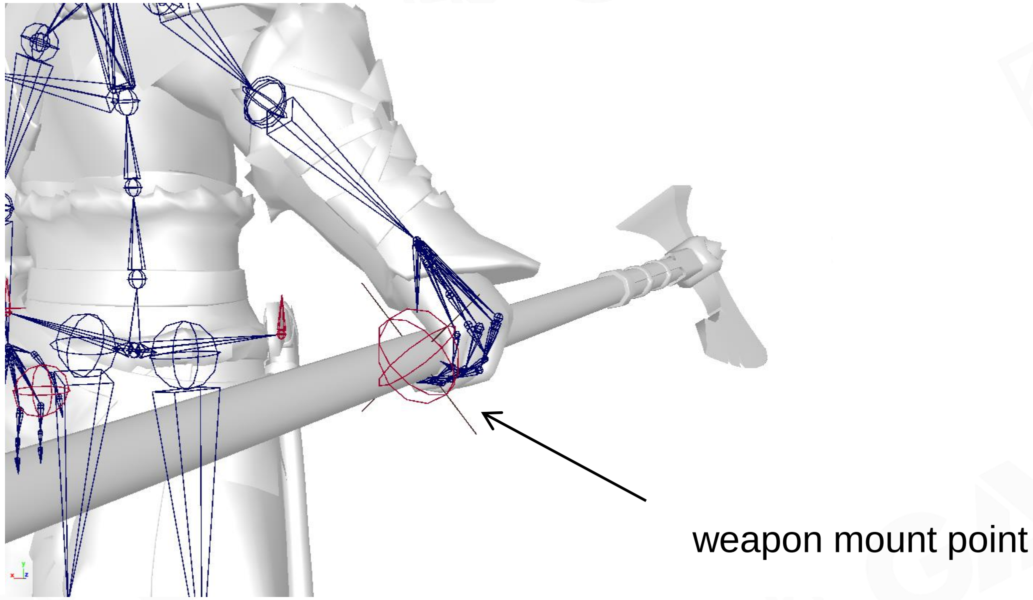
320 joints

Joints for Game Play

Additional joints

- Weapon joint
- Mount joint





attach weapon to the mount point

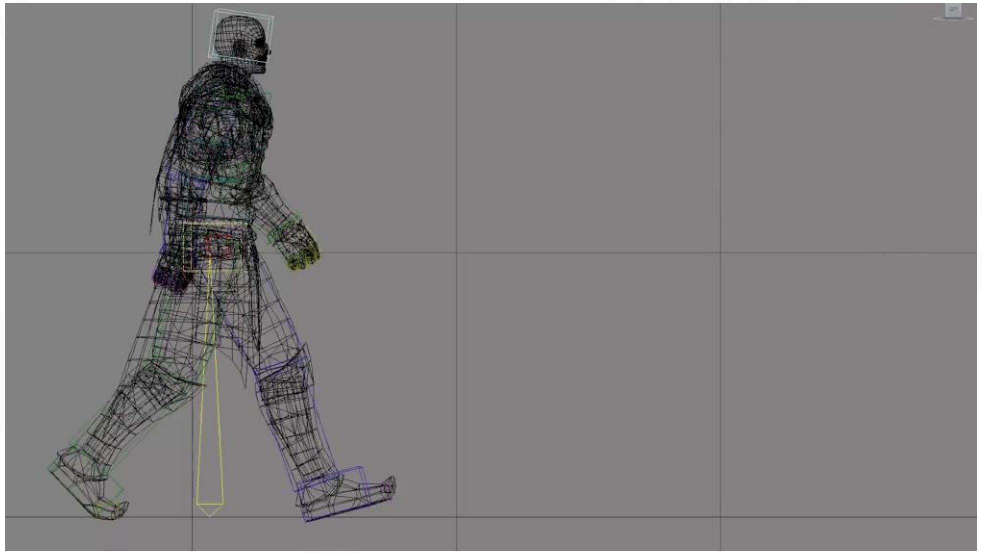
Where to Start the Skeleton – Root Joint

Root joint

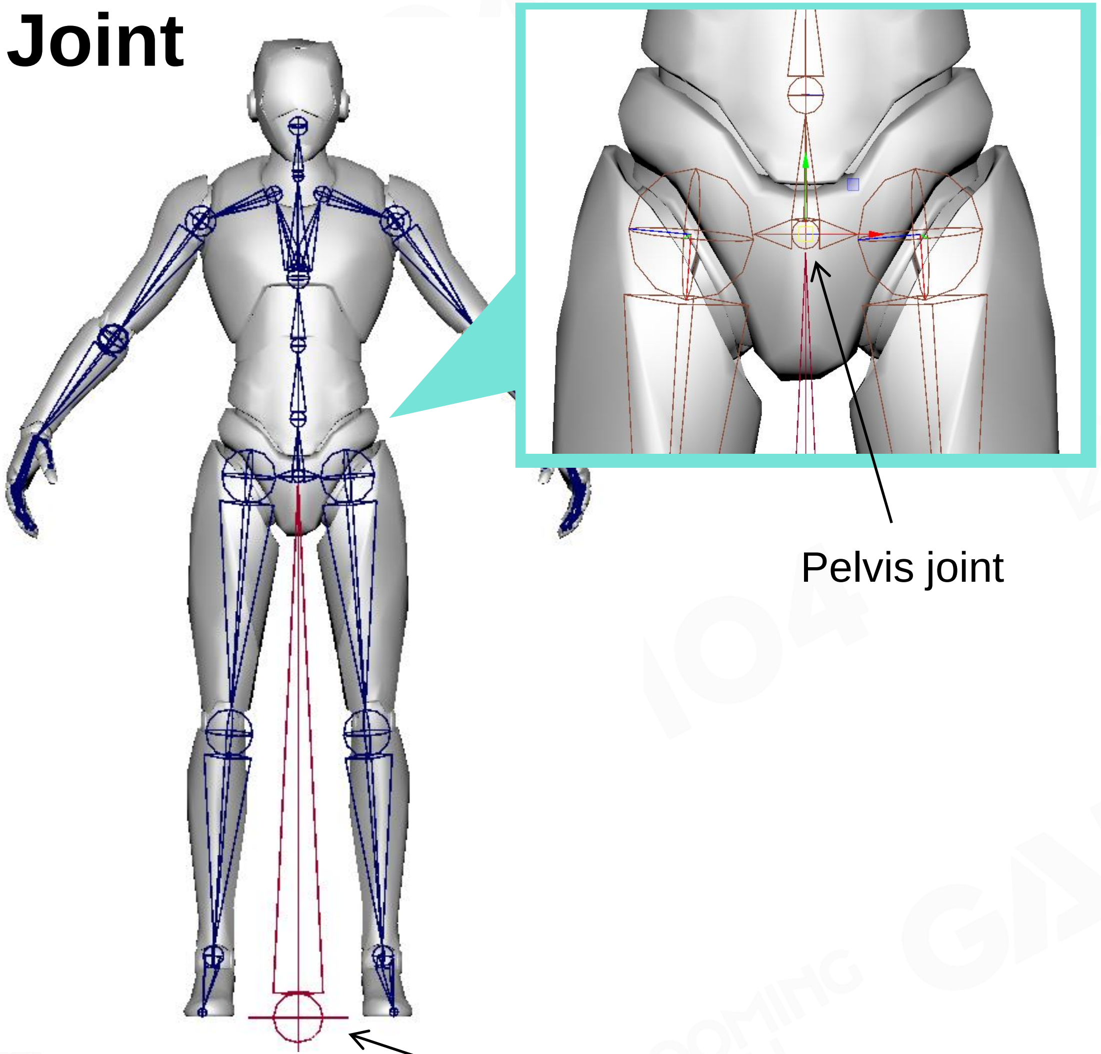
- The center of the feet
- Convenient to touch the ground

Pelvis joint

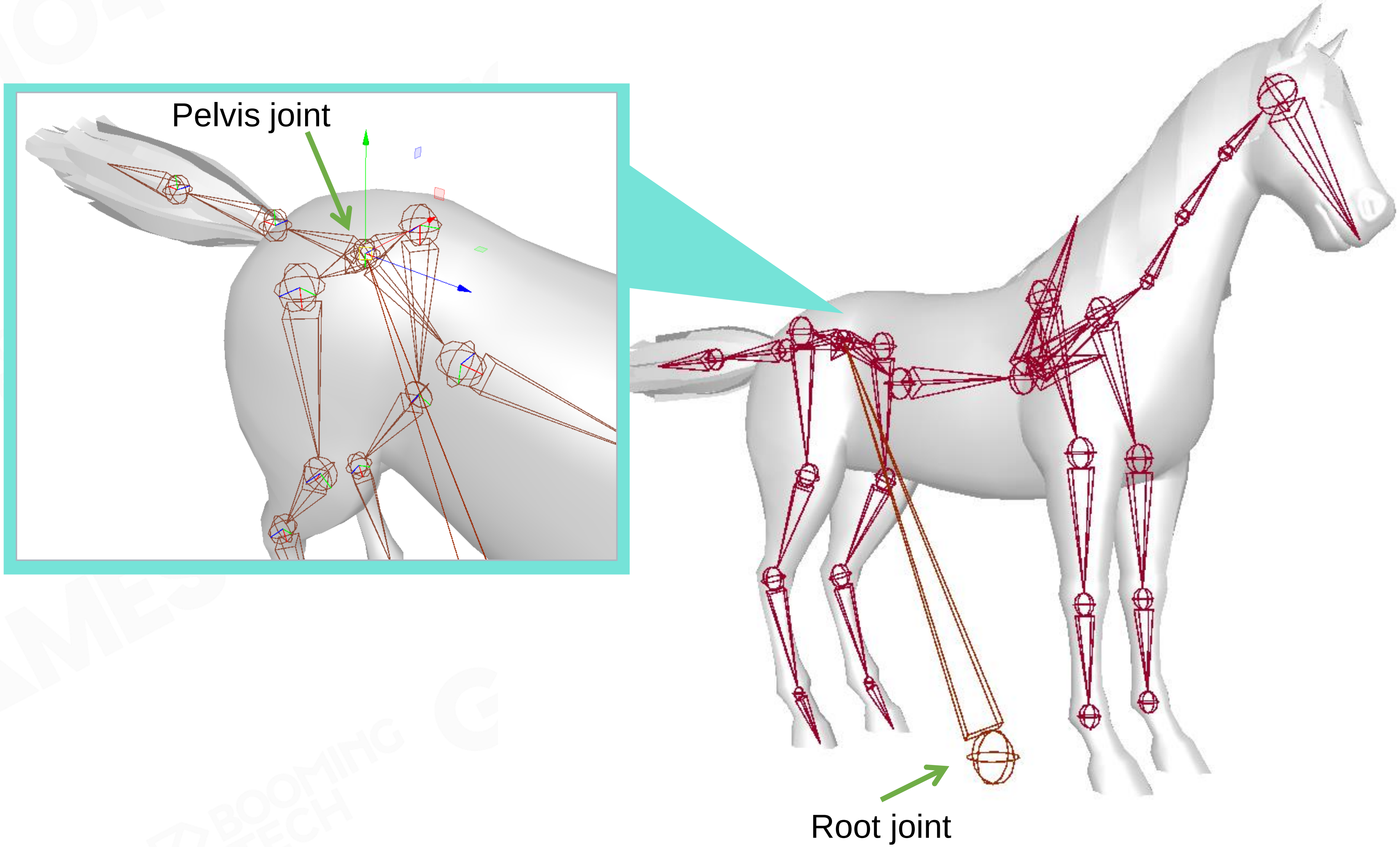
- The first child joint of the root joint
- Human upper and lower body separation



Root joint move

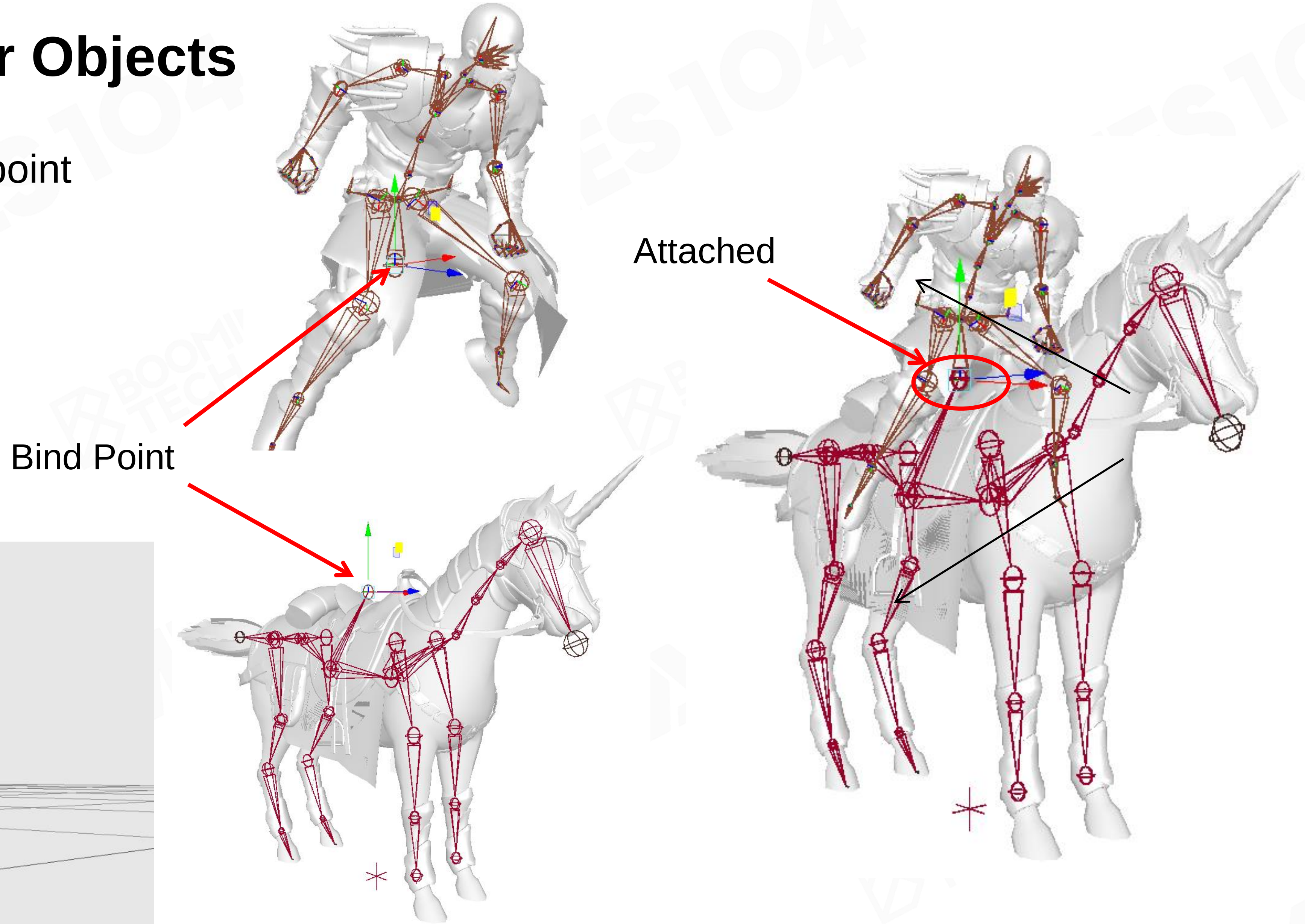


How to Handle Horse Skeleton?



Bind Animation for Objects

Attach two skeleton's bind point



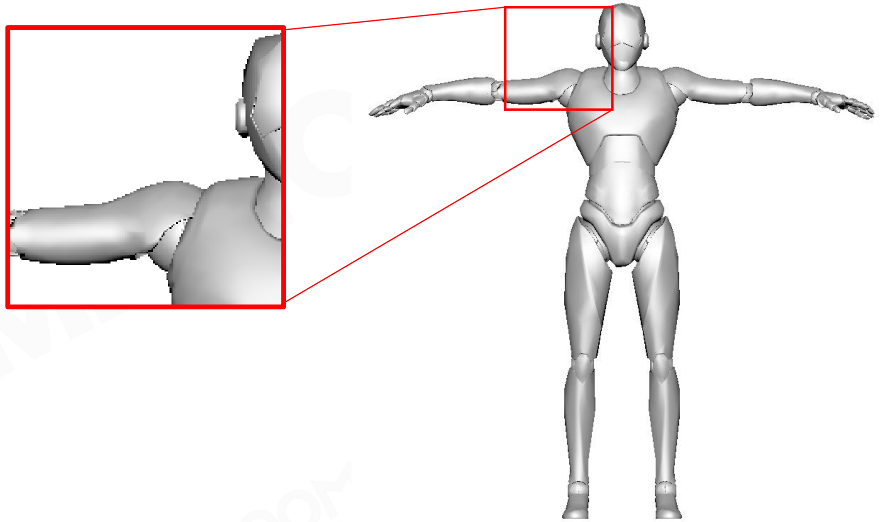
Bind Pose – T-pose vs. A-pose

The pose of the 3D mesh prior to being bound to the skeleton

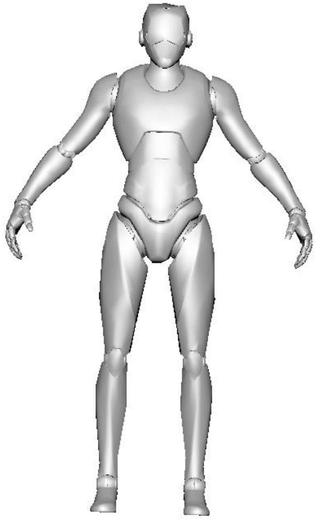
- Keep the limbs away from the body and each other, making the process of binding the vertices to the joints easier
- Usually close to natural pose

T-pose vs A-pose

- Shoulders in A-pose are more relaxed
- Easy to defarmating in A-pose



T-pose



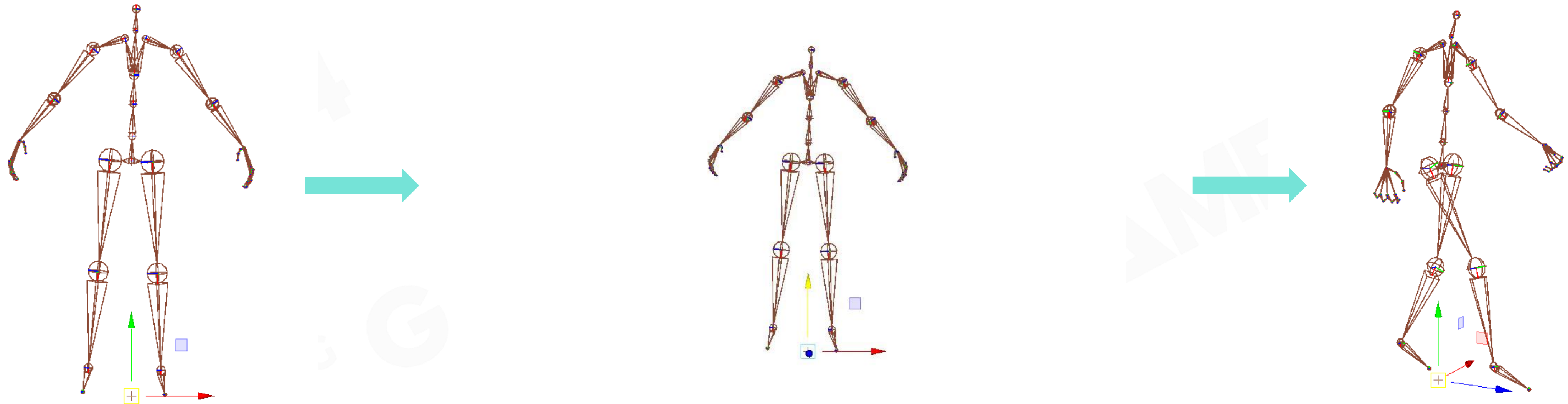
A-pose

Skeleton Pose

Skeleton Pose : A skeleton is posed by transform its joints from the bind pose

Joint Pose (9 DoFs)

- Orientation (3 DoFs)
- Position (3 DoFs)
- Scale (3 DoFs)



Bind Pose

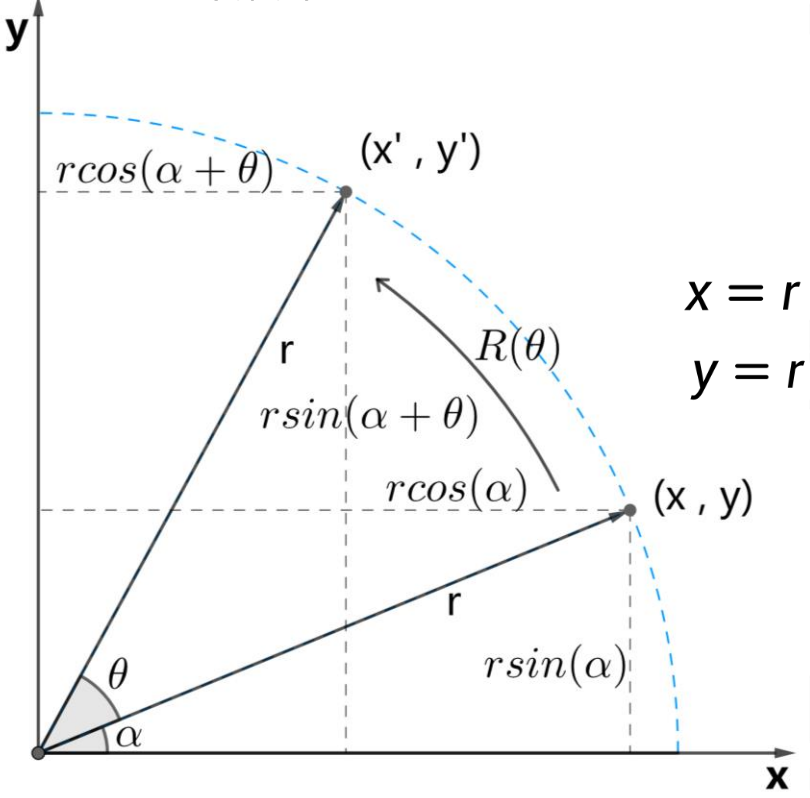
Posed by Transform Joints

The Pose of Walking

Math of 3D Rotation

2D Orientation Math

- 2D-Rotation



$$x = r \cdot cos(a)$$
$$y = r \cdot sin(a)$$

$$x' = r \cdot cos(a + \theta)$$
$$= r \cdot (cos(a)cos(\theta) - sin(a)sin(\theta))$$
$$= r \cdot cos(a)cos(\theta) - r \cdot sin(a)sin(\theta)$$

$$y' = r \cdot sin(a + \theta)$$
$$= r \cdot (sin(a)cos(\theta) + cos(a)sin(\theta))$$
$$= r \cdot sin(a)cos(\theta) + r \cdot cos(a)sin(\theta)$$

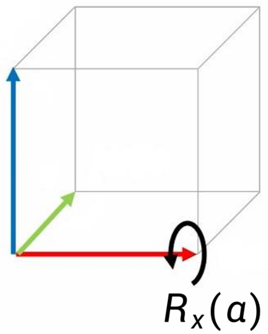
$$x' = x \cdot cos(\theta) - y \cdot sin(\theta)$$
$$y' = x \cdot sin(\theta) + y \cdot cos(\theta)$$

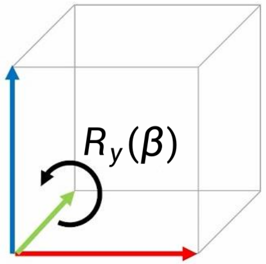
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \qquad R(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$$

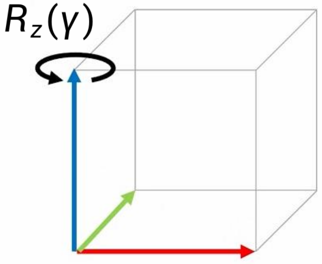
3D Orientation Math

Euler Angle

- 3D-Rotation by single axis

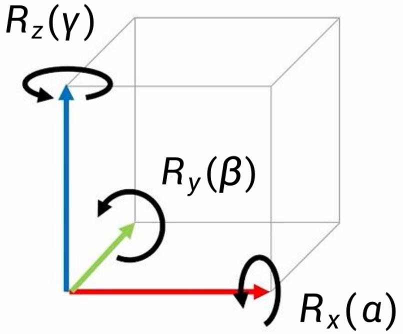






$$R_x(\alpha)=\begin{bmatrix}1 & 0 & 0\\ 0 & \cos(\alpha) & -\sin(\alpha)\\ 0 & \sin(\alpha) & \cos(\alpha)\end{bmatrix}\quad R_y(\beta)=\begin{bmatrix}\cos(\beta) & 0 & \sin(\beta)\\ 0 & 1 & 0\\ -\sin(\beta) & 0 & \cos(\beta)\end{bmatrix}\quad R_z(\gamma)=\begin{bmatrix}\cos(\gamma) & -\sin(\gamma) & 0\\ \sin(\gamma) & \cos(\gamma) & 0\\ 0 & 0 & 1\end{bmatrix}$$

- 3D-Rotation combined by x, y, z axis **sequentially**



$$R=R_x(\alpha)R_y(\beta)R_z(\gamma)$$

$$=\begin{bmatrix}\cos(\alpha)\cos(\beta) & \cos(\alpha)\sin(\beta)\sin(\gamma)-\sin(\alpha)\cos(\gamma) & \cos(\alpha)\sin(\beta)\cos(\gamma)+\sin(\alpha)\sin(\gamma)\\ \sin(\alpha)\cos(\beta) & \sin(\alpha)\sin(\beta)\sin(\gamma)+\cos(\alpha)\cos(\gamma) & \sin(\alpha)\sin(\beta)\cos(\gamma)-\cos(\alpha)\sin(\gamma)\\ -\sin(\beta) & \cos(\beta)\sin(\gamma) & \cos(\beta)\cos(\gamma)\end{bmatrix}$$

Euler Angle

Euler Angle provides a brief description of 3D rotation and is widely used in many fields

Roll-Pitch-Yaw Angles

Yaw angle: ψ

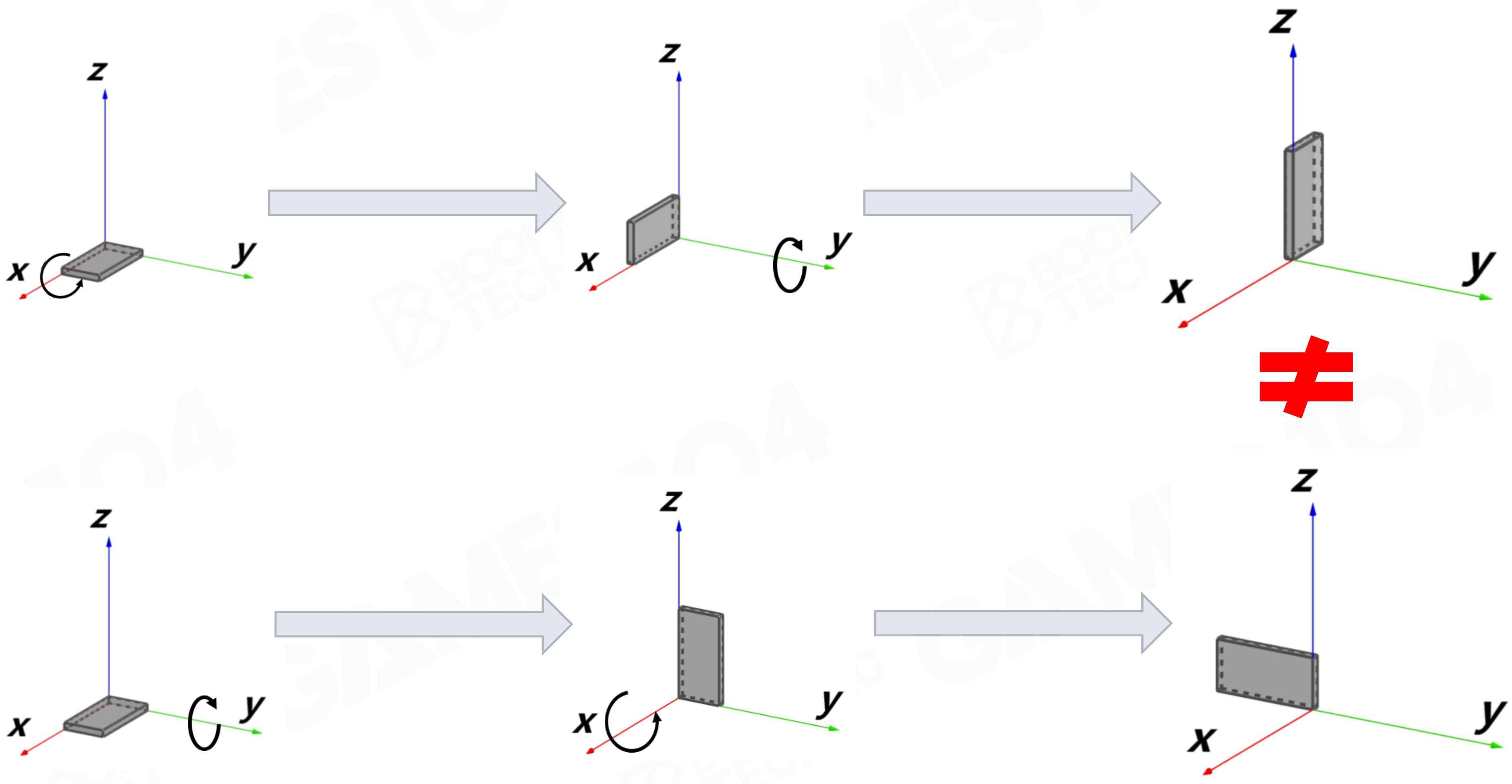
Pitch angle: θ

Roll angle : ϕ

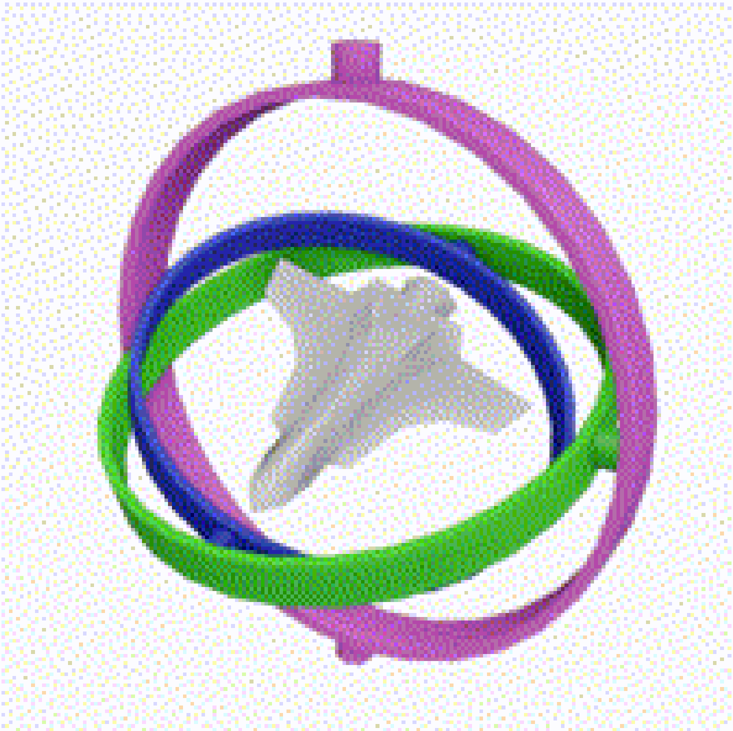
Order Dependence on Euler Angle

Path A:

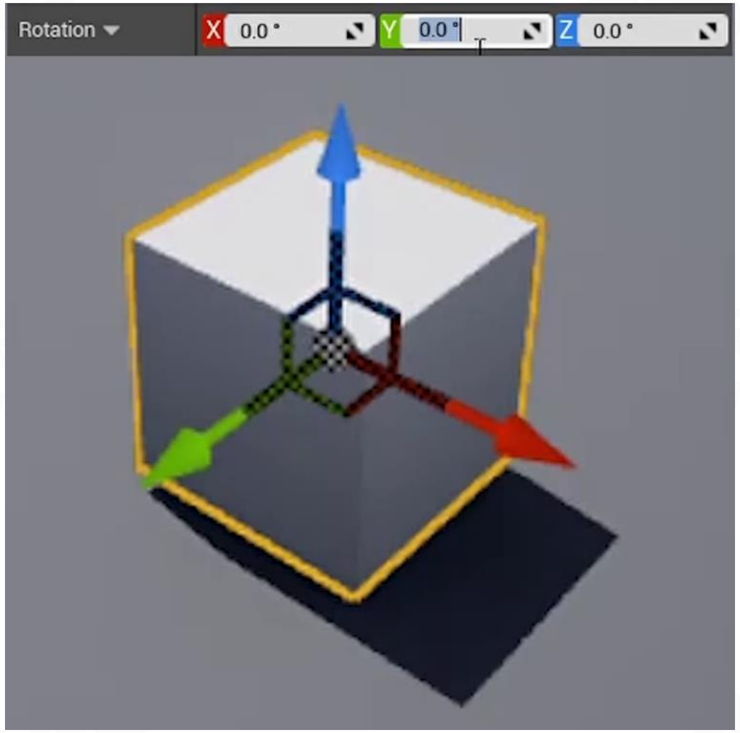
Path B:



Gimbal Lock



Gimbal lock



Gimbal lock in Game Editor

Gyroscope

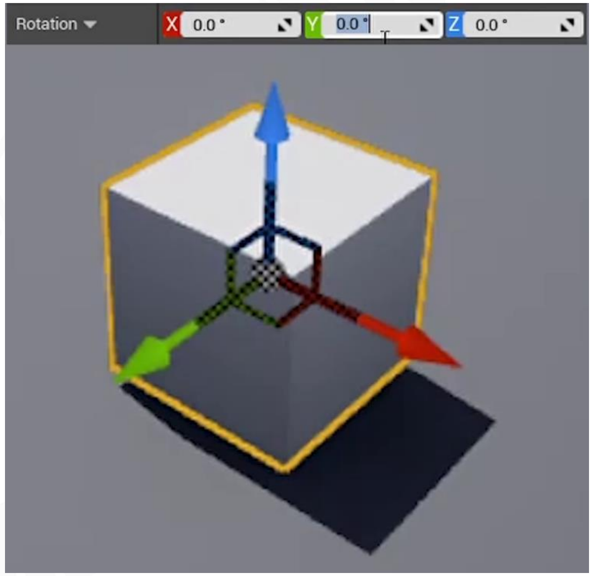
Degeneration of Euler Angle

$$R = R_x(\alpha)R_y(\beta)R_z(\gamma) = \begin{bmatrix} \cos(\alpha)\cos(\beta) & \cos(\alpha)\sin(\beta)\sin(\gamma) - \sin(\alpha)\cos(\gamma) & \cos(\alpha)\sin(\beta)\cos(\gamma) + \sin(\alpha)\sin(\gamma) \\ \sin(\alpha)\cos(\beta) & \sin(\alpha)\sin(\beta)\sin(\gamma) + \cos(\alpha)\cos(\gamma) & \sin(\alpha)\sin(\beta)\cos(\gamma) - \cos(\alpha)\sin(\gamma) \\ -\sin(\beta) & \cos(\beta)\sin(\gamma) & \cos(\beta)\cos(\gamma) \end{bmatrix}$$

$$\Big\downarrow \quad \beta = \frac{\pi}{2}$$

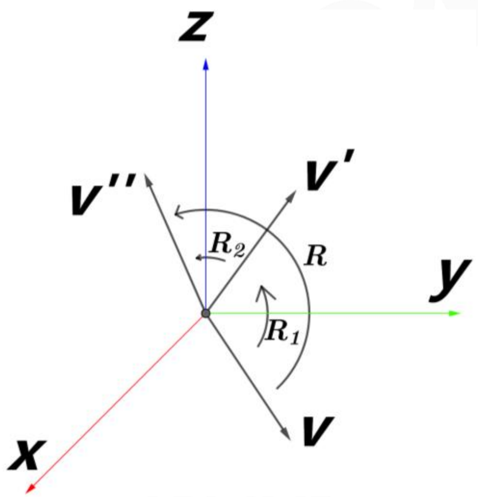
$$R = R_x(\alpha)R_y(\tfrac{\pi}{2})R_z(\gamma) = \begin{bmatrix} 0 & \cos(\alpha)\sin(\gamma) - \sin(\alpha)\cos(\gamma) & \cos(\alpha)\cos(\gamma) + \sin(\alpha)\sin(\gamma) \\ 0 & \sin(\alpha)\sin(\gamma) + \cos(\alpha)\cos(\gamma) & \sin(\alpha)\cos(\gamma) - \cos(\alpha)\sin(\gamma) \\ -1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & -\sin(\alpha-\gamma) & \cos(\alpha-\gamma) \\ 0 & \cos(\alpha-\gamma) & \sin(\alpha-\gamma) \\ -1 & 0 & 0 \end{bmatrix}$$

Let α - γ = *t*, we reduce the final DoF to 1

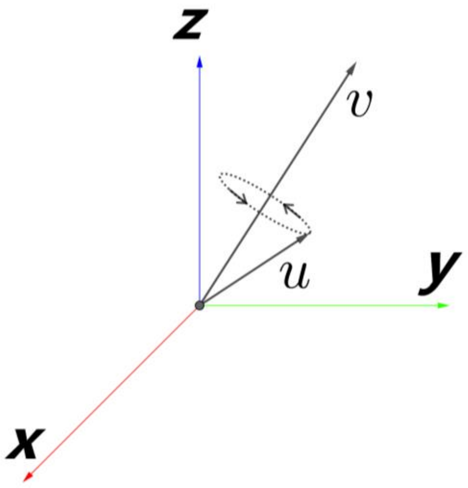


Problems of Euler Angle

- **Gimbal Lock**
  Gimbal Lock occurs because of the loss of one DoF
- **Hard to interpolate**
  Singularity problem make it hard to interpolate
- **Difficult for rotation combination**
  Rotation combination need rotation matrix
- **Hard to rotate by certain axis**
  Easy to rotate by x,y,z axis but hard to others



Rotation Combination



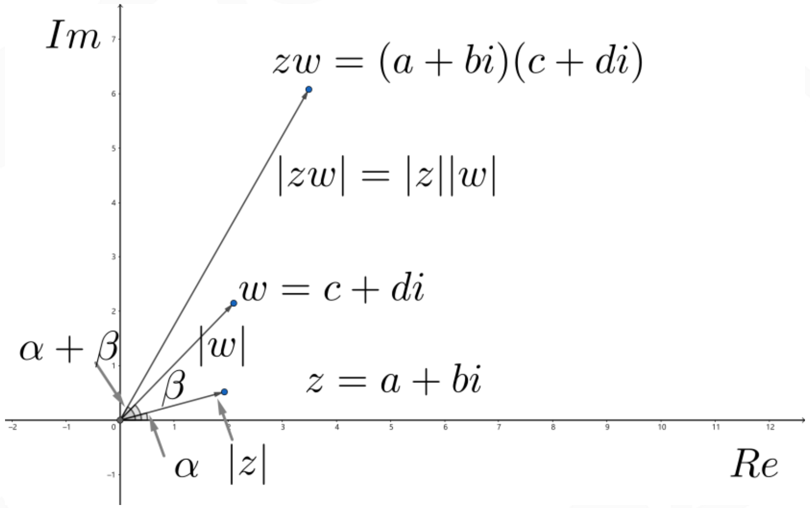
Rotation By Certain Axis

Quaternion

Every morning in the early part of October 1843, on my coming down to breakfast, your brother William Edwin and yourself used to ask me: "Well, Papa, can you multiply triples?" Whereto I was always obliged to reply, with a sad shake of the head, "No, I can only add and subtract them.

Sir William Rowan Hamilton
Irish mathematician

Complex Number and 2D Rotation



Sir William Rowan Hamilton
Irish mathematician

Complex Number

- Definition

$$c = a + bi \quad (a,b \in R)$$
$$i^2 = -1$$

- Represent as Vector

$$c = \begin{bmatrix} a \\ b \end{bmatrix}$$

- Product

$$c_1 = a + bi \qquad c_1 c_2 = \begin{bmatrix} a & -b \\ b & a \end{bmatrix} \begin{bmatrix} c \\ d \end{bmatrix}$$
$$c_2 = c + di$$

Quaternion

Quaternion

- Definition

$$q = a + bi + cj + dk \quad (a,b,c,d \in R)$$

$$i^2 = j^2 = k^2 = ijk = -1$$

- Represent as two parts pair (real number and vector)

$$q = (a,v) \quad (v = \begin{bmatrix} b \\ c \\ d \end{bmatrix}, a,b,c,d \in R)$$

- Product

$$q_1 = a + bi + cj + dk$$
$$q_2 = e + fi + gj + hk$$

$$q_1 q_2 = \begin{bmatrix} a & -b & -c & -d \\ b & a & -d & c \\ c & d & a & -b \\ d & -c & b & a \end{bmatrix} \begin{bmatrix} e \\ f \\ g \\ h \end{bmatrix}$$

- Norm

$$||q|| = \sqrt{a^2 + b^2 + c^2 + d^2}$$

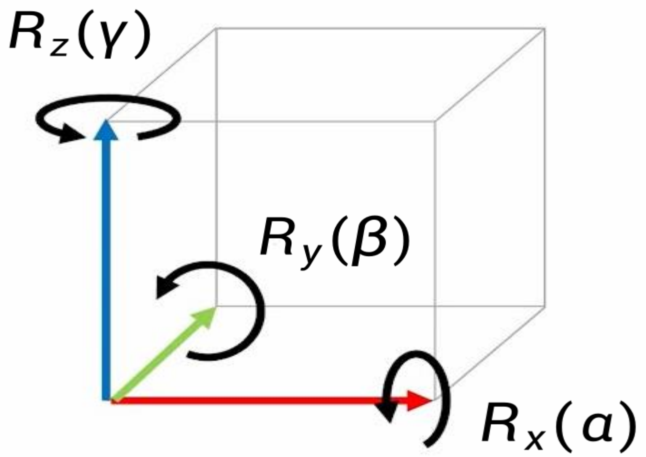
- Conjugate

$$q^* = a - bi - cj - dk$$

- Inverse

$$q^{-1}q = qq^{-1} = 1$$

Euler Angle to Quaternion



$$q = \begin{bmatrix} 1 & i & j & k \end{bmatrix} \begin{bmatrix} \cos(\gamma/2)\cos(\beta/2)\cos(\alpha/2) + \sin(\gamma/2)\sin(\beta/2)\sin(\alpha/2) \\ \sin(\gamma/2)\cos(\beta/2)\cos(\alpha/2) - \cos(\gamma/2)\sin(\beta/2)\sin(\alpha/2) \\ \cos(\gamma/2)\sin(\beta/2)\cos(\alpha/2) + \sin(\gamma/2)\cos(\beta/2)\sin(\alpha/2) \\ \cos(\gamma/2)\cos(\beta/2)\sin(\alpha/2) - \sin(\gamma/2)\sin(\beta/2)\cos(\alpha/2) \end{bmatrix}$$

Rotation by Quaternion

Quaternion

- Vector to quaternion
  - A 3D vector ***v*** could be written in quaternion format as follow:

$$v_q = (0, v) = bi + cj + dk \quad v = \begin{bmatrix} b \\ c \\ d \end{bmatrix}$$

- Rotation

$$v_q' = q v_q q^* = q v_q q^{-1}$$

$$q^* = a - bi - cj - dk$$

$$q_1 = a + bi + cj + dk$$
$$q_2 = e + fi + gj + hk$$

$$q_1 q_2 = \begin{bmatrix} a & -b & -c & -d \\ b & a & -d & c \\ c & d & a & -b \\ d & -c & b & a \end{bmatrix} \begin{bmatrix} e \\ f \\ g \\ h \end{bmatrix}$$

Quaternion to Rotation Matrix

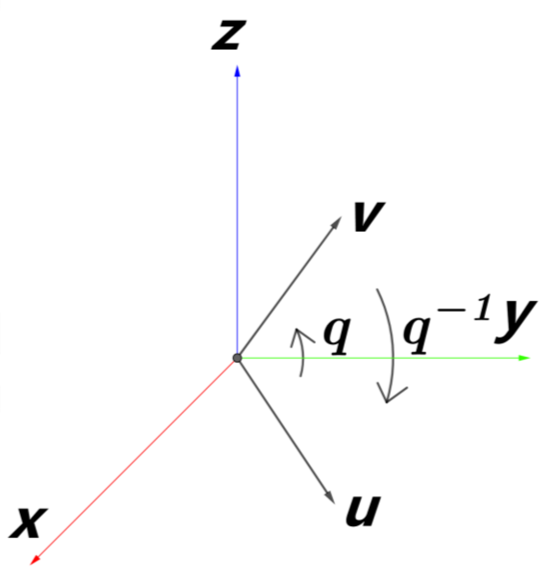
$$q = (a,b,c,d) \qquad ||q|| = 1$$

$$v' = \begin{bmatrix} 1-2c^2-2d^2 & 2bc-2ad & 2ac+2bd \\ 2bc+2ad & 1-2b^2-2d^2 & 2cd-2ab \\ 2bd-2ac & 2ab+2cd & 1-2b^2-2c^2 \end{bmatrix} v$$

Rotation Math by Quaternion

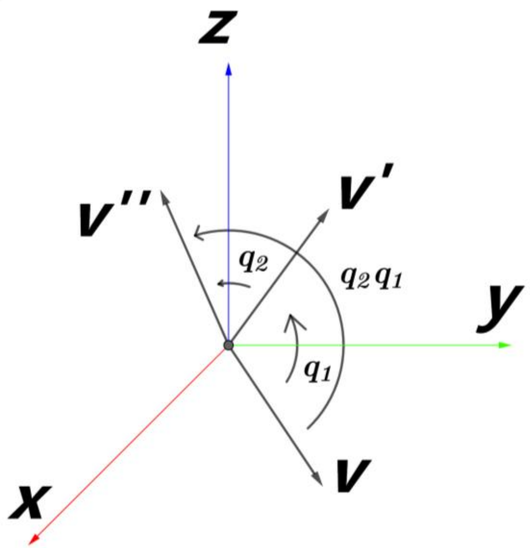
Inverse Resolving

$$q^{-1} = \frac{q^*}{||q||^2}$$



Rotation Combination

$$\begin{aligned} q_1^* q_2^* &= (q_2 q_1)^* \\ v' &= q_1 v q_1^* \\ v'' &= q_2 v' q_2^* \\ &= q_2 q_1 v q_1^* q_2^* \\ &= (q_2 q_1) v (q_2 q_1)^* \end{aligned}$$

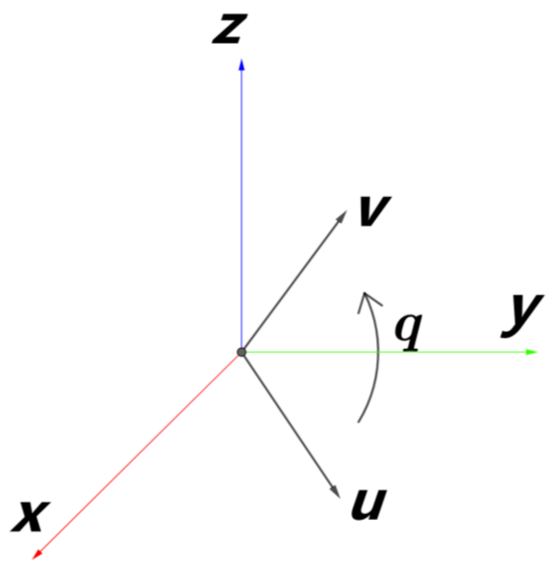


Quaternion between Two Unit Vectors

$$w = u \times v$$

$$q = [u \cdot v + \sqrt{(w \cdot w) + (u \cdot v)^2}, w]$$

$$(||u|| = ||v|| = 1)$$



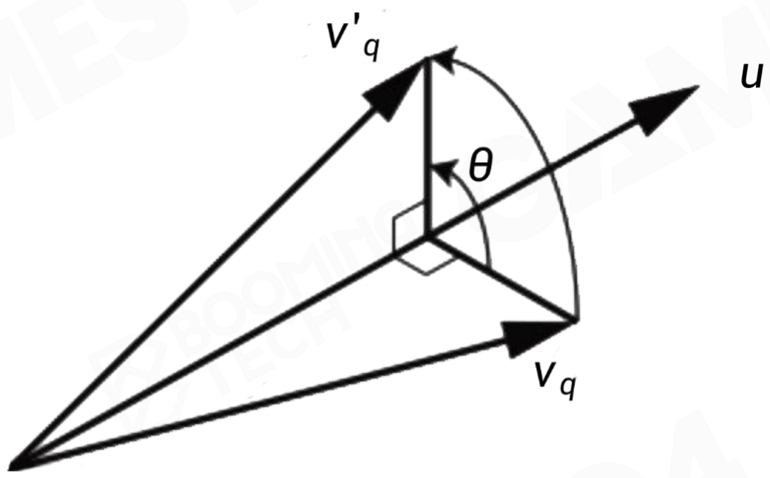
Given Axis Rotation by Quaternion

Quaternion

- Vector to quaternion
  - A 3D vector ***v*** could be written in quaternion format as follow:

$$v_q = (0, v) = bi + cj + dk \quad v = \begin{bmatrix} b \\ c \\ d \end{bmatrix}$$

- Rotation
  - For **vector** v, rotated by **unit axis** u *of* **angle** θ, the **result vector** v_q'



$$q = \left(\cos\left(\frac{\theta}{2}\right),\ \sin\left(\frac{\theta}{2}\right)x_u,\ \sin\left(\frac{\theta}{2}\right)y_u,\ \sin\left(\frac{\theta}{2}\right)z_u\right)$$

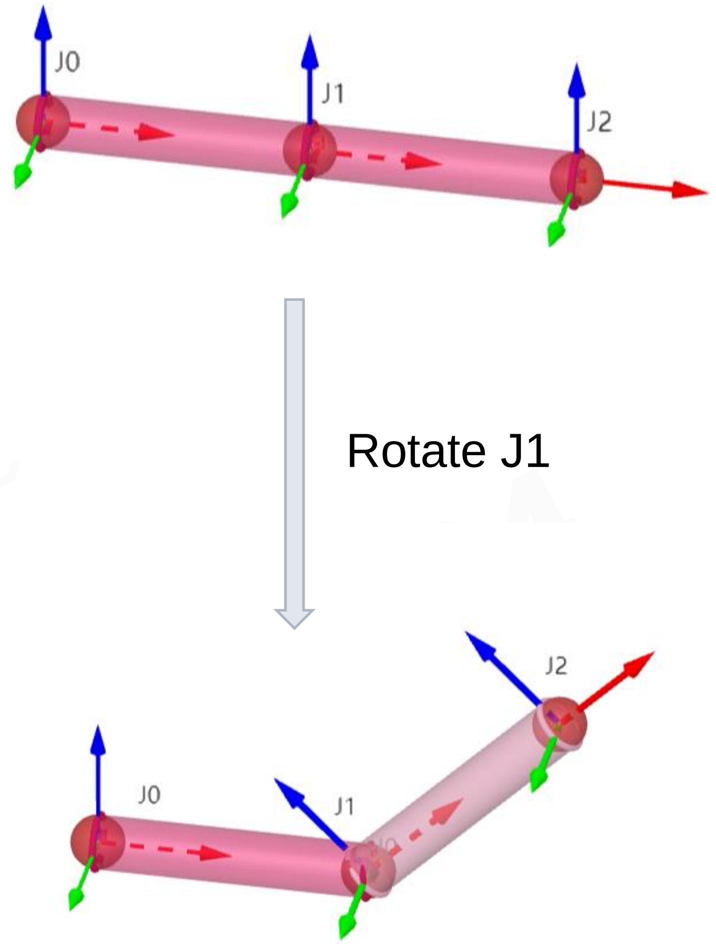
$u = \begin{bmatrix} x_u \\ y_u \\ z_u \end{bmatrix}$ is a **unit vector** represents rotation axis

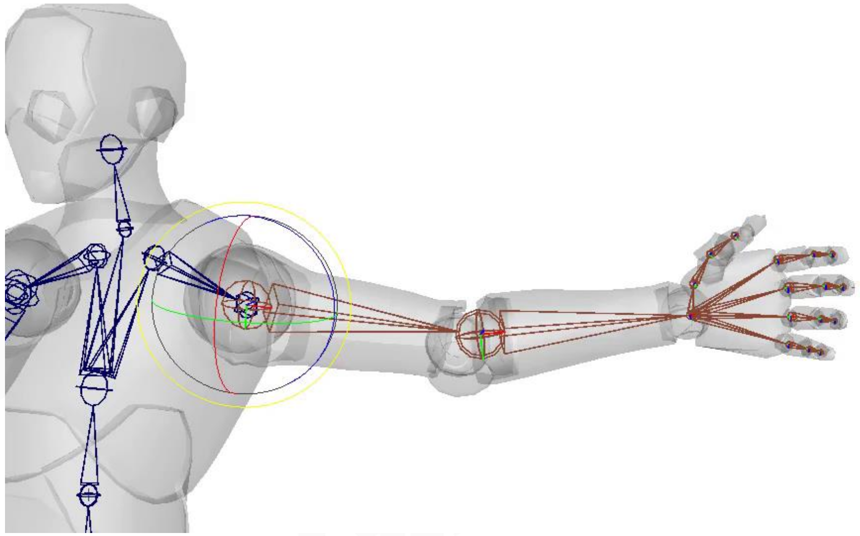
$$v_q' = q v_q q^* = q v_q q^{-1}$$

Joint Pose

Joint Pose - Orientation

- Rotation -> Change the Orientation of joints
- Most skeleton poses change orientations of joints only





Rotate the forearm

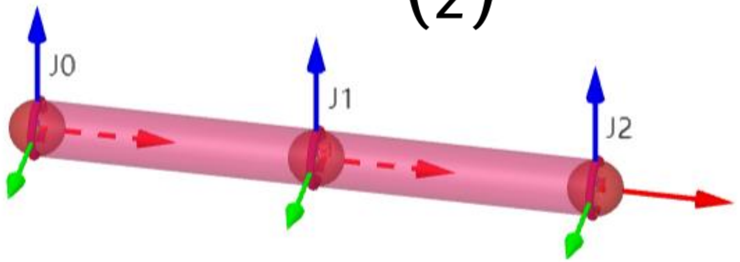
Joint Pose - Position

- Translate -> change postion
- Translate point P to point P' by vector T

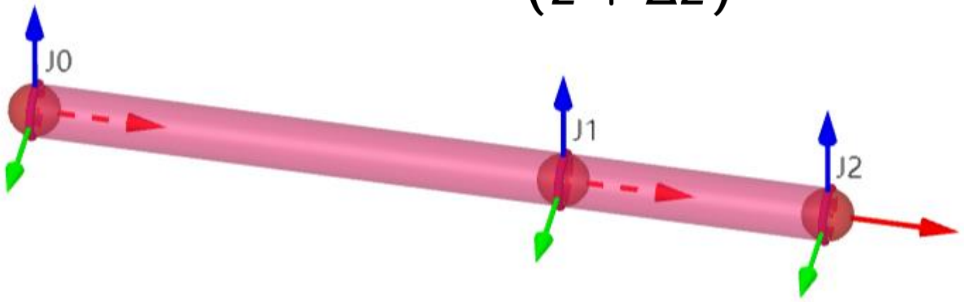
$$P = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

$$T = \begin{pmatrix} \Delta x \\ \Delta y \\ \Delta z \end{pmatrix}$$

$$P' = P + T = \begin{pmatrix} x + \Delta x \\ y + \Delta y \\ z + \Delta z \end{pmatrix}$$

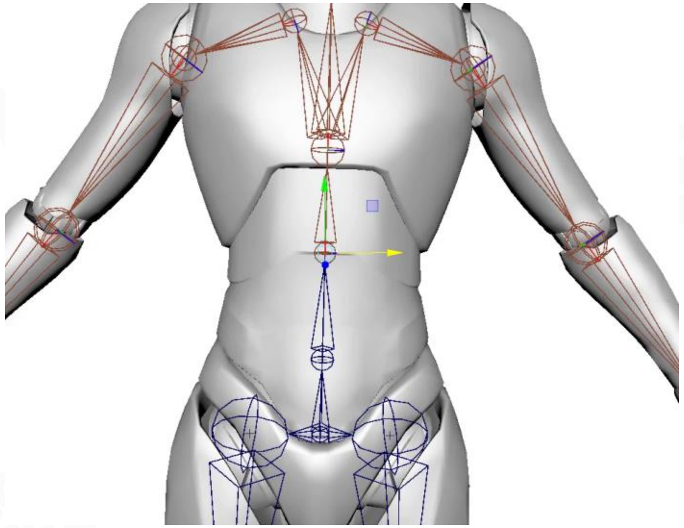


Tanslate J1

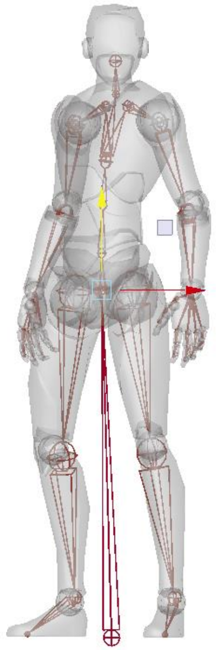


Joint Pose - Position

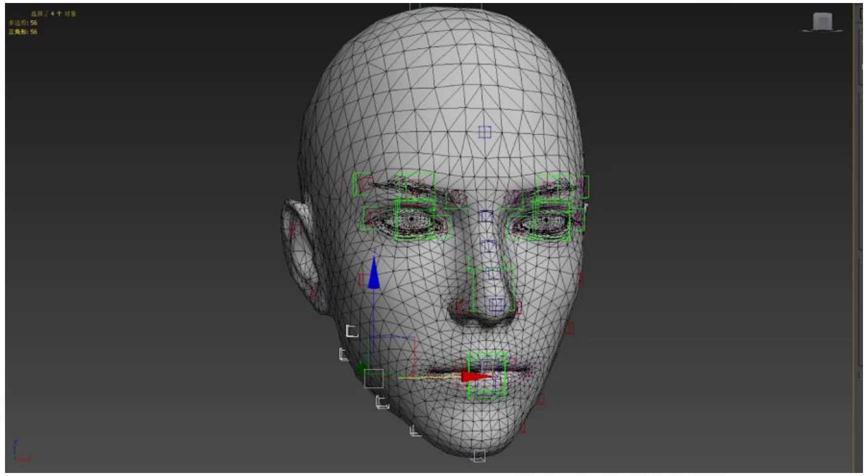
- Usually not changed in humanoid skeleton except the pelvis, facial joint and other special joints
- Used for stretching models



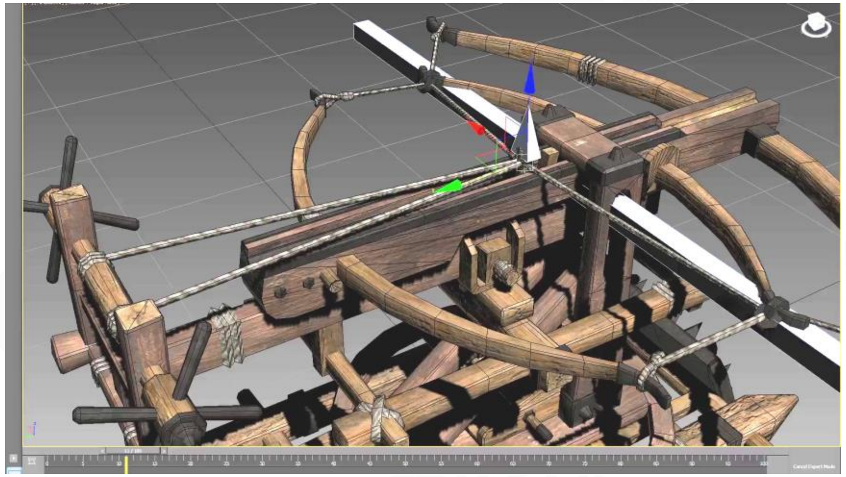
Example of error



Character Movement



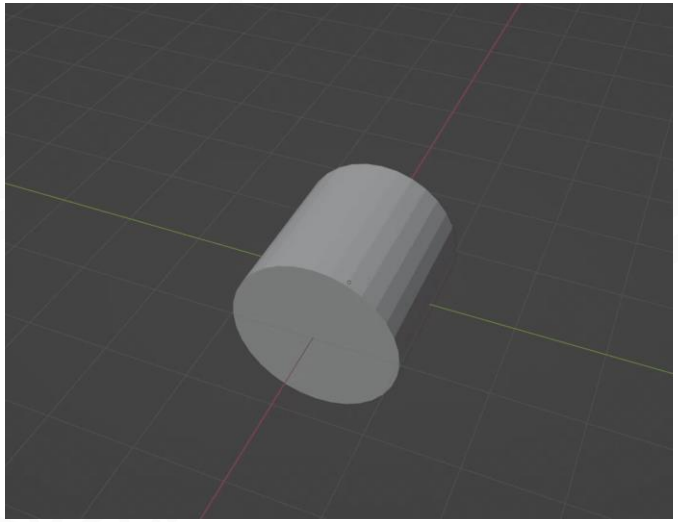
Facical Joint Translation



Draw the Bow

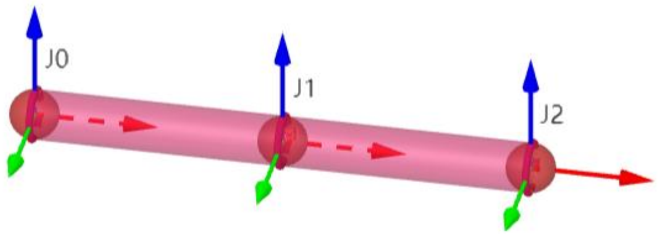
Joint Pose - Scale

- Scale -> change the size of the model
- Uniform vs. Non-uniform Scale



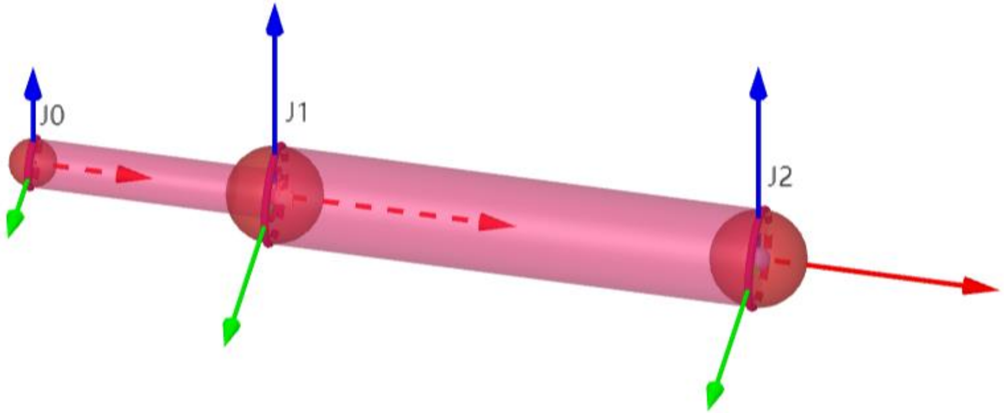
Non-uniform Scale

$$V = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \qquad S = \begin{pmatrix} a \\ b \\ c \end{pmatrix} \qquad P_1' = \begin{pmatrix} ax \\ by \\ cz \end{pmatrix}$$



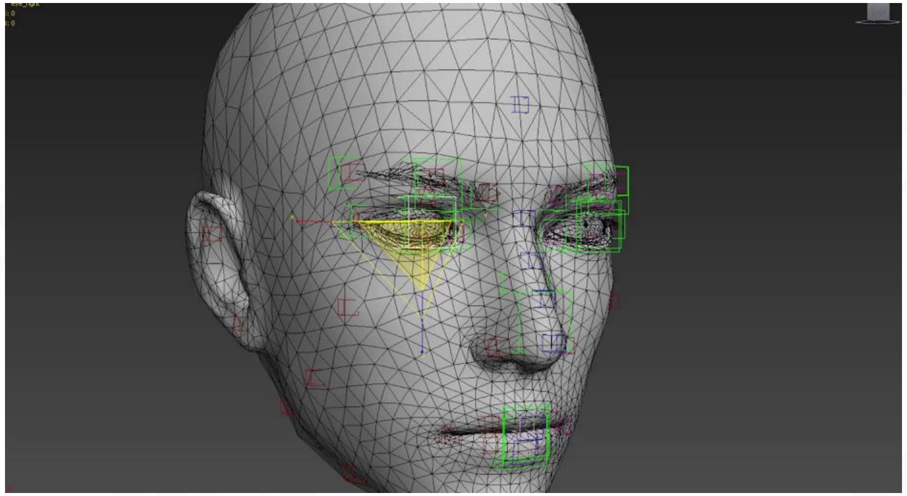
(Uniform Scale: a = b = c)

Scale J1

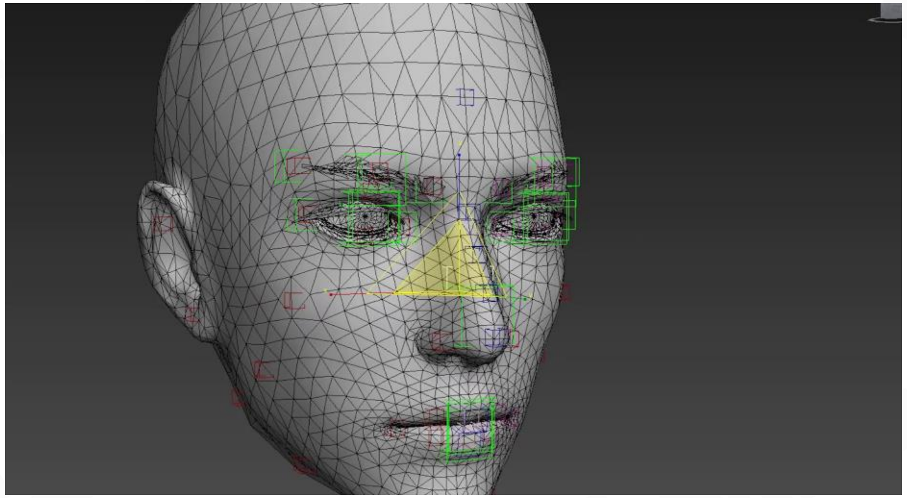


Joint Pose - Scale

- Widely used in facial animation
- Uniform and non-uniform scale facial joints



Uniform Scale Facial joints



Non-uniform Scale Facial joints

Joint Pose - Affine Matrix

Rotation

$$Q = \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} \Rightarrow R = \begin{bmatrix} 1-2c^2-2d^2 & 2bc-2ad & 2ac+2bd \\ 2bc+2ad & 1-2b^2-2d^2 & 2cd-2ab \\ 2bd-2ac & 2ab+2cd & 1-2b^2-2c^2 \end{bmatrix} \Rightarrow$$

$$R_{HM} = \begin{bmatrix} 1-2c^2-2d^2 & 2bc-2ad & 2ac+2bd & 0 \\ 2bc+2ad & 1-2b^2-2d^2 & 2cd-2ab & 0 \\ 2bd-2ac & 2ab+2cd & 1-2b^2-2c^2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Translation

$$T_v = \begin{pmatrix} \Delta x \\ \Delta y \\ \Delta z \end{pmatrix} \Rightarrow T_{HM} = \begin{bmatrix} 1 & 0 & 0 & \Delta x \\ 0 & 1 & 0 & \Delta y \\ 0 & 0 & 1 & \Delta z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Scale

$$S_v = \begin{pmatrix} a_x \\ b_y \\ c_z \end{pmatrix} \Rightarrow S = \begin{bmatrix} a_x & 0 & 0 \\ 0 & b_y & 0 \\ 0 & 0 & c_z \end{bmatrix} \Rightarrow S_{HM} = \begin{bmatrix} a_x & 0 & 0 & 0 \\ 0 & b_y & 0 & 0 \\ 0 & 0 & c_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Affine Matrix

$$M = R_{HM} T_{HM} S_{HM} = \begin{bmatrix} SR & T \\ 0 & 1 \end{bmatrix}$$

Joint Pose - Local Space to Model Space

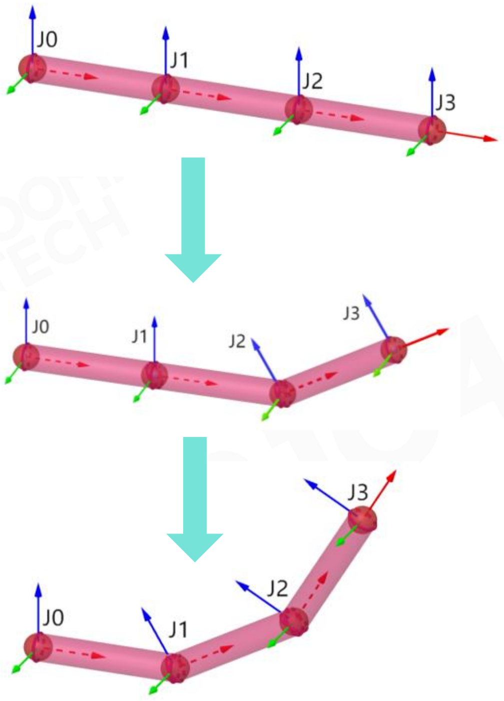
For a joint ***j*** in a skinned mesh

- ***p(j)*** : Joint ***j***'s parent joint
- $M^l_{p(j)}$: Joint ***j***'s parent joint pose in local space

M^m_J : joint ***J***'s pose in model space

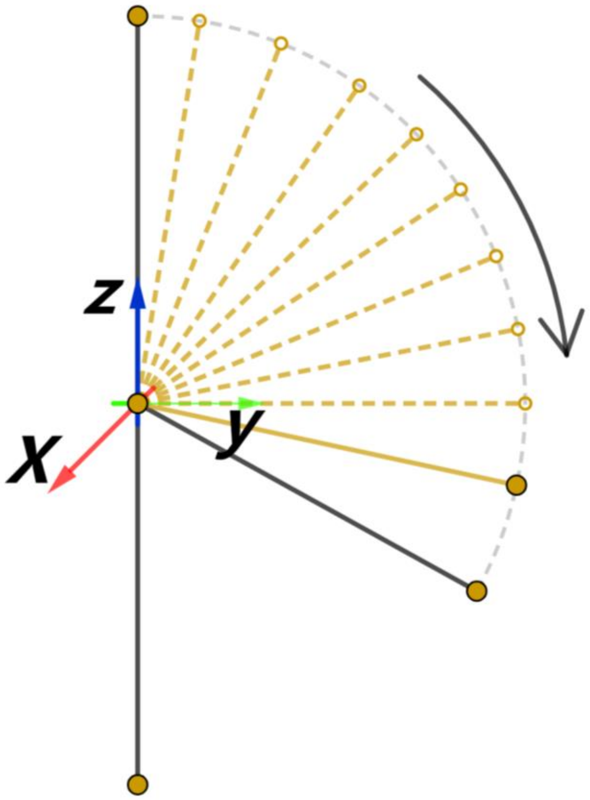
- Walking the skeletal hierarchy from ***J*** to the root:

$$M^m_J = \prod_{j=J}^{0} M^l_{p(j)}$$

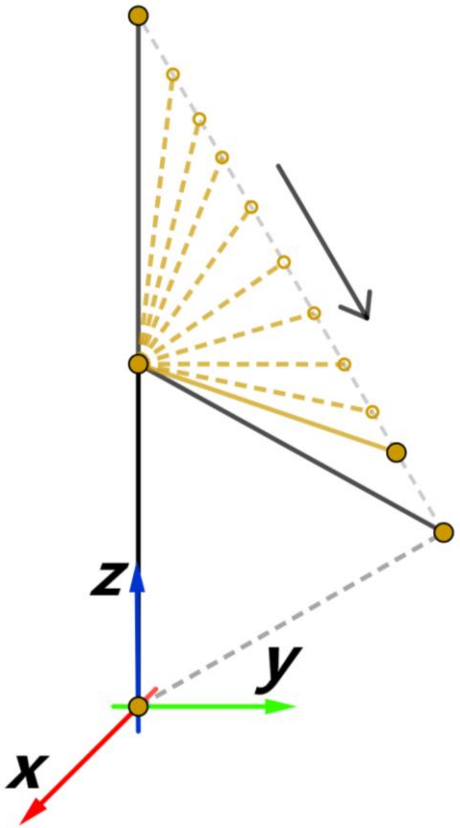


Joint Pose Interpolation - Local Space vs. Model Space

Local Space
- Less data with delta transform
- Convenient for interpolation or blend



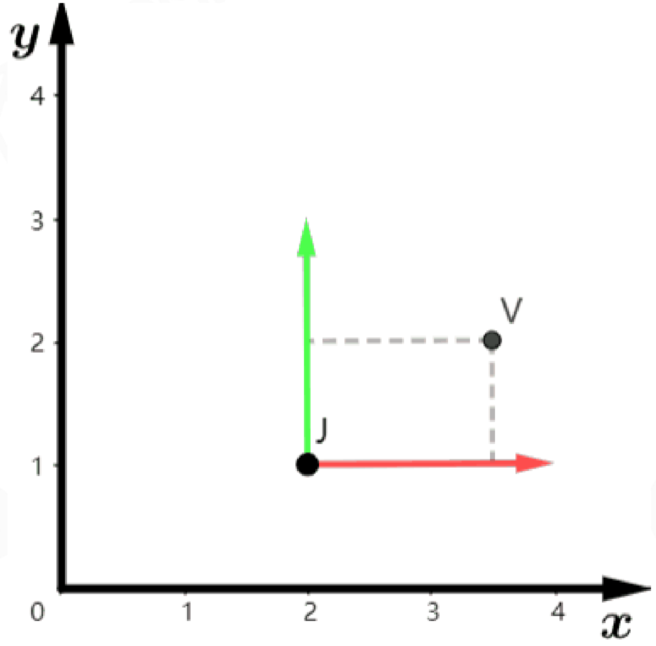
Model Space
- Incorrect for interploation



Single Joint Skinning

Attach the vertices of a mesh to a posed skeleton

- Each vertex can be bound to one or more joints with a weight parameter
- The vertex position in each bound joint's local space is fixed



Vertex ***V***'s position in joint ***J***'s local space is fixed

Skinning Matrix

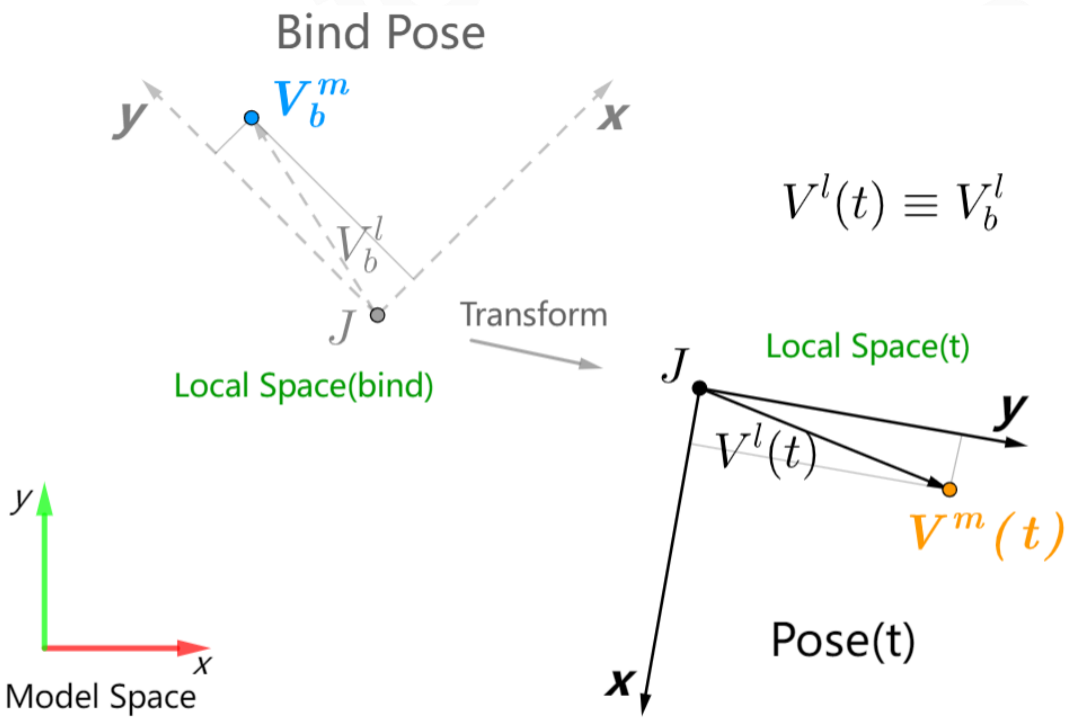
Bind Pose: the skeleton pose for binding

For a mesh vertex ***V*** which is bound to a joint ***J***

- V_b^m : ***V***'s position in **model space** within **bind pose**
- V_b^l : ***V***'s position in **local space** within **bind pose**
- $M_{b(J)}^m$: ***J***'s pose in **model space** within **bind pose**

V's position in **local space** at any time ***t*** is fixed as

$$V^l(t) \equiv V_b^l = (M_{b(J)}^m)^{-1} \cdot V_b^m$$



Skinning Matrix

$M_J^m(t)$: joint ***J***'s pose in **model space** at time ***t***

$$M_J^m(t) = \prod_{j=J}^{0} M_{p(j)}^l(t)$$

$V^m(t)$: ***V***'s position in **model space** at time ***t***

Inverse Bind Pose Matrix

$$V^m(t) = M_J^m(t) \cdot V_J^l = \boxed{M_J^m(t) \cdot (M_{b(J)}^m)^{-1}} \cdot V_b^m$$

$$\Downarrow$$

Skinning Matrix: $K_J = \boxed{M_J^m(t) \cdot (M_{b(J)}^m)^{-1}}$

Representing a Skeleton in Memory

- The name of the joint, either as a string or a hashed 32-bit string id
- The index of the joint's parent within the skeleton
- The inverse bind pose transform is the inverse of the product of the translation, rotation and scale

```
struct Joint
{
    const String   m_joint_name;                  // the name of joint
    UInt8          m_parent_joint_index;          // the index of parent joint or 0xFF if root
    Translation    m_bind_pose_translation;       // bind pose:translation
    Rotation       m_bind_pose_rotation;          // bind pose:rotation
    Scale          m_bind_pose_scale;             // bind pose:scale
    Matrix4X3      m_inverse_bind_pose_transform; // inverse bind pose
};
```

Inverse Bind Pose Matrix

```
struct Skeleton
{
    UInt           m_joint_count;    // number of joints
    Joint          m_joints[];       // array of joints
};
```

Skinning Matrix Palette

An array of skinning matrices for each joint

- To be used by GPU in shaders
- Optimization: count the transform matrix M^w for **model space** to **world space**

  The optimized **skinning matrix** of joint ***J*** is

$$K'_J = M^w \cdot M_J^m(t) \cdot (M_{b(J)}^m)^{-1}$$

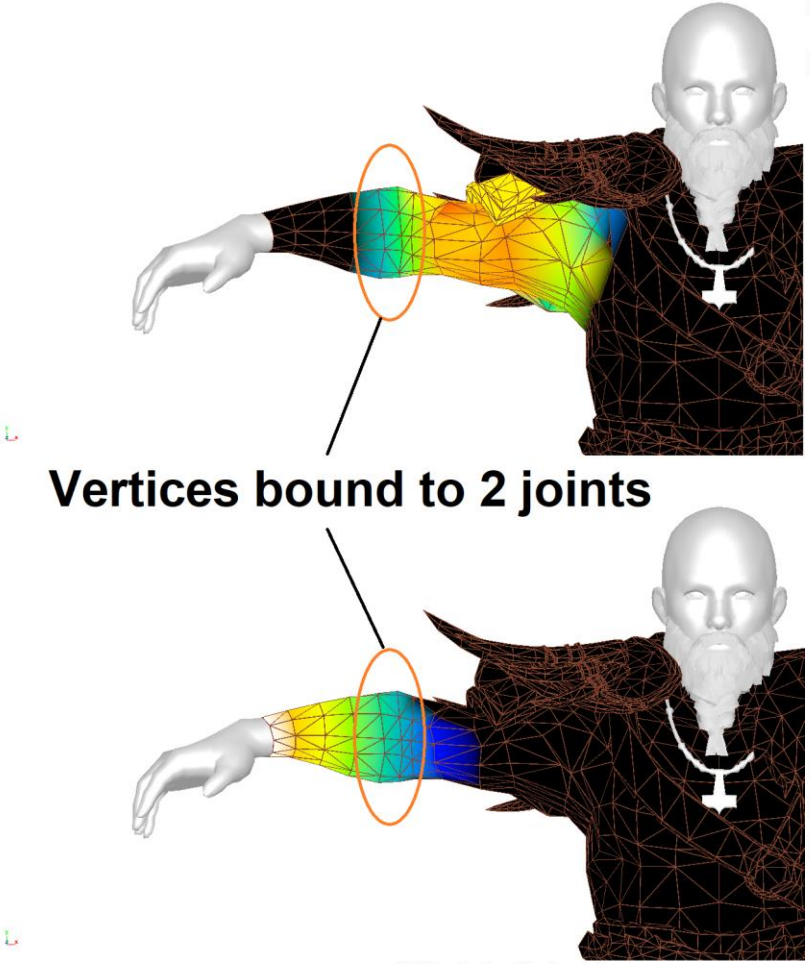
Weighted Skinning with Multi-joints

For a mesh vertex ***V*** which is bound to ***N*** joints

- W_i : the **Skinning Weight** of the ***i***-th bound joint

$$\sum_{i=0}^{N-1} W_i = 1$$

Skinning weight of many different joints



Skinning weight of the upper arm and the fore arm

Weighted Skinning Blend

For a vertex ***V*** which is bound to ***N*** joints J_0 to J_{N-1}

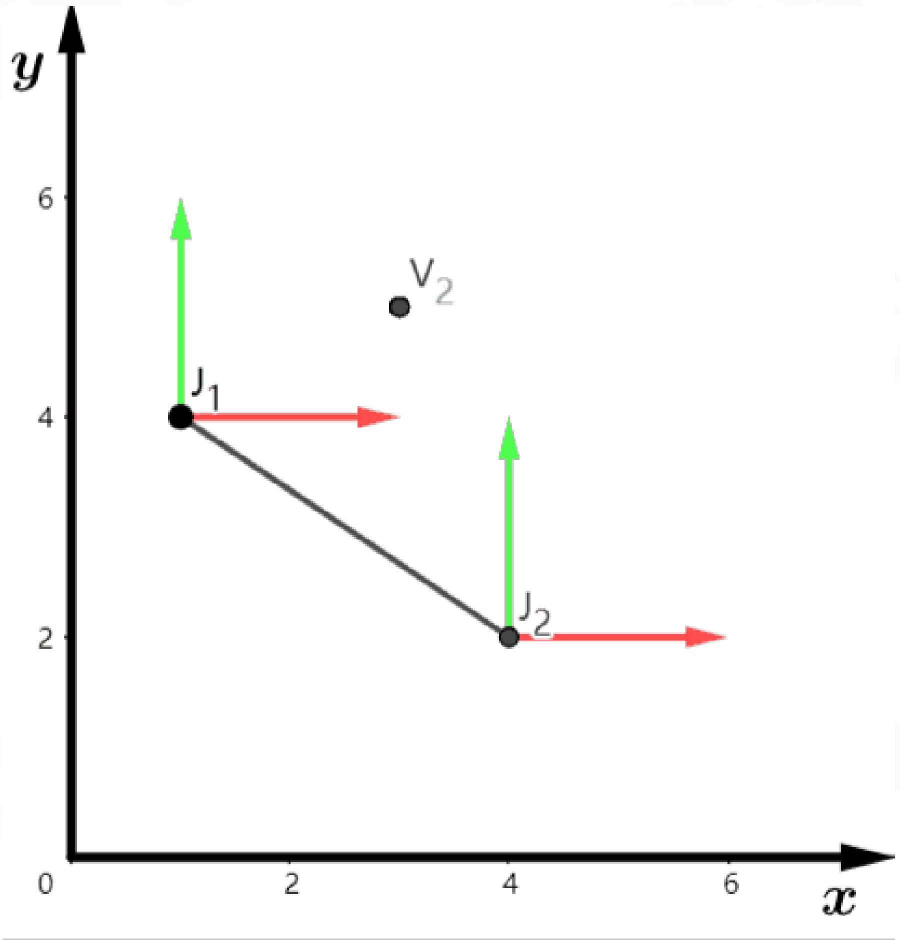
- $K_{J_i}(t)$: the skinning matrix of joint J_i at time t

Transform ***V***'s position in joint J_i's **local space** to **model space**:

$$V_{J_i}^m(t) = K_{J_i}(t) \cdot V_{b(J_i)}^m$$

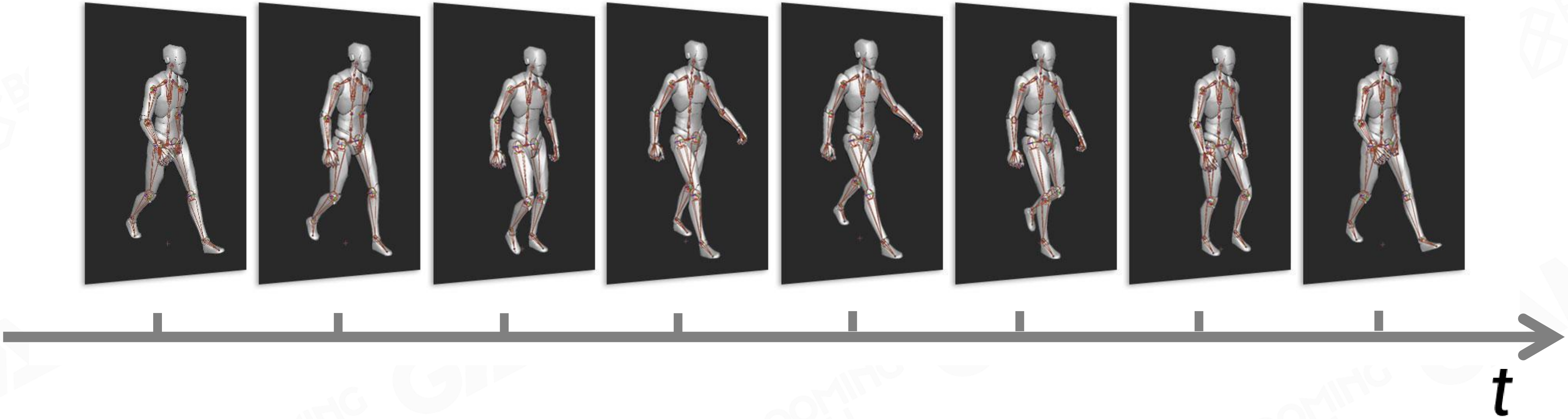
V's position in **model space**:

$$V^m(t) = \sum_{i=0}^{N-1} W_i \cdot V_{J_i}^m(t)$$



Clip

A sequence of skeleton poses



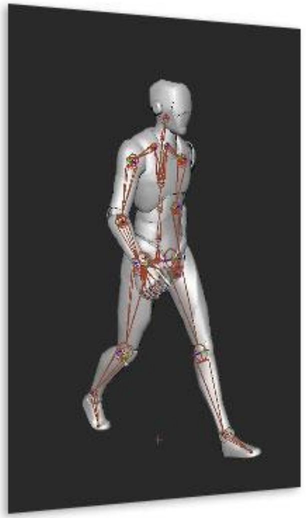
Interpolation between Poses

- Animation's timeline is continuous

Pose 1

... ? ...

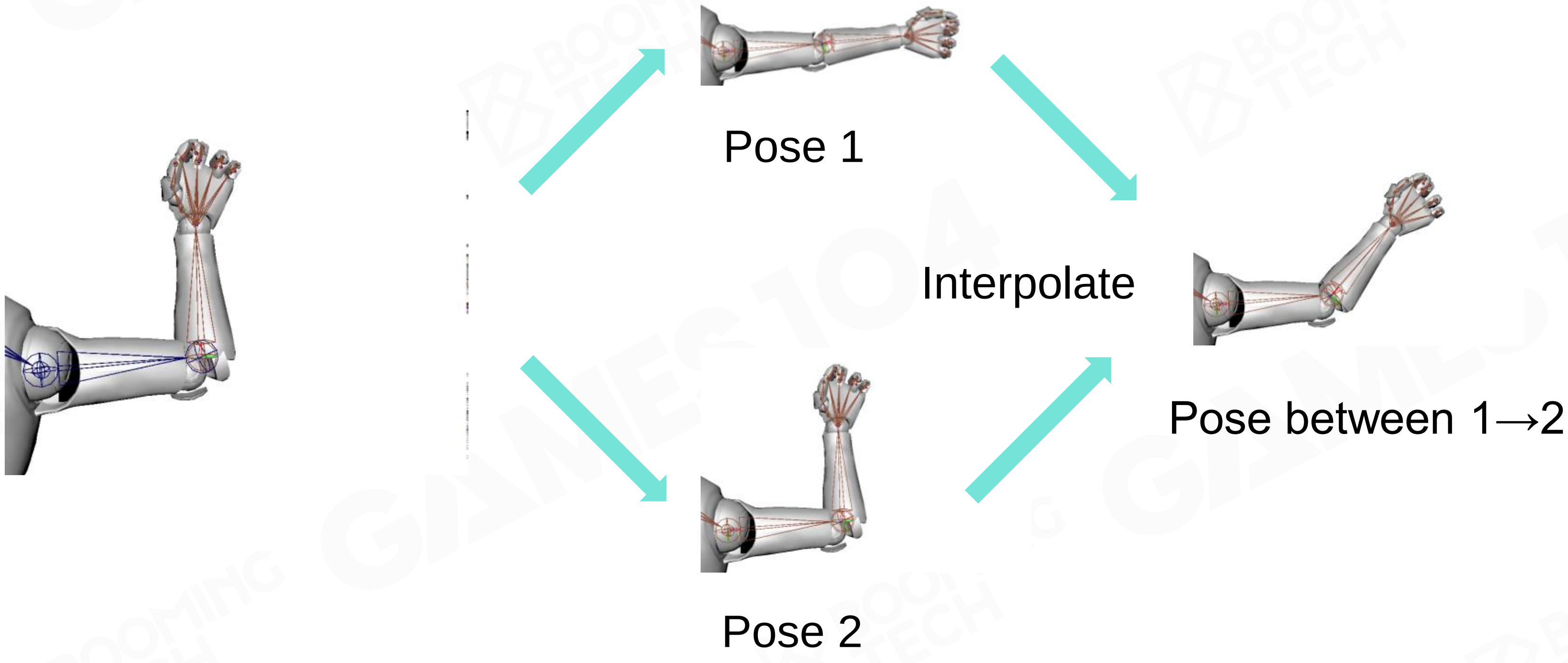
... Pose 1.5 ...



Pose 2

Interpolation

- Calculate the pose between key poses

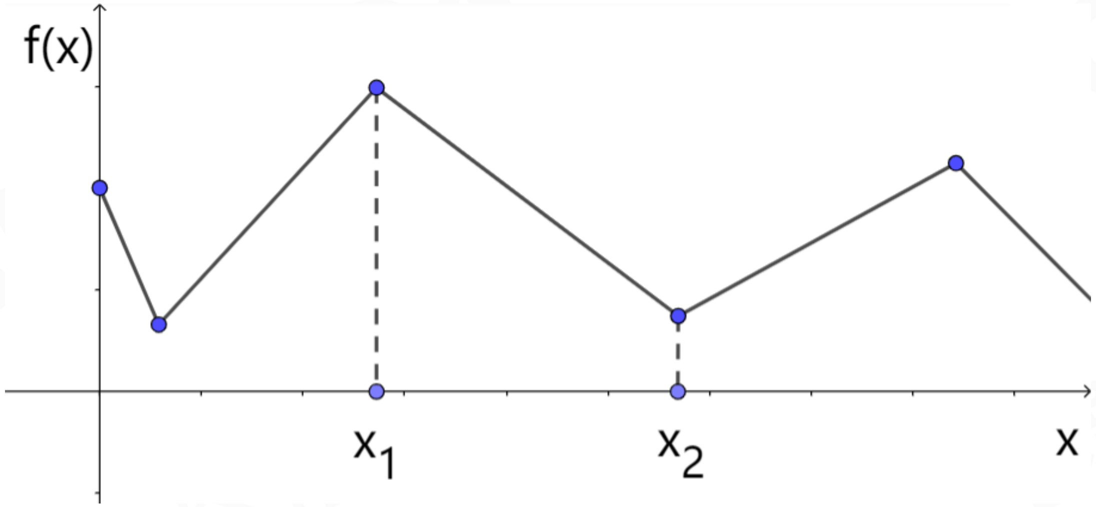


Simple Interpolation of Translation and Scale

- **Linear interpolation (LERP)**

$$f(x) = (1-a)f(x_1) + af(x_2)$$

$$a = \frac{x - x_1}{x_2 - x_1}, x_1 < x_2, x \in [x_1, x_2]$$



Translation:

$$T(t) = (1-a)T(t_1) + aT(t_2)$$

Scale:

$$S(t) = (1-a)S(t_1) + aS(t_2)$$

Quaternion Interpolation of Rotation

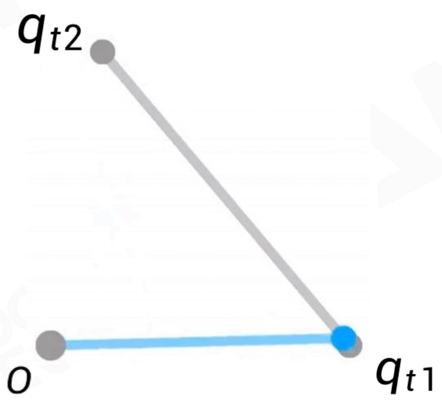
- **NLERP for quaternion**
  - Liner Interpolation

$$q_t = Lerp(q_{t1}, q_{t2}, t) = (1-a)q_{t1} + aq_{t2}$$

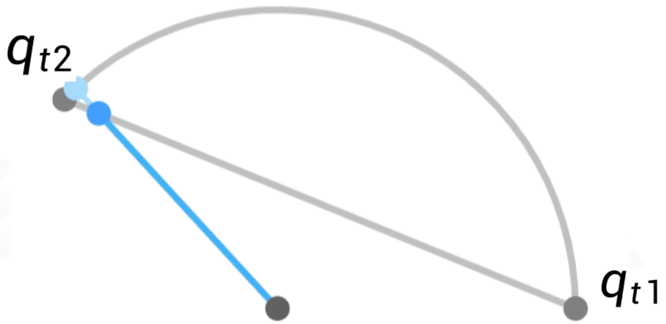
$$a = \frac{t - t_1}{t_2 - t_1}, t_1 < t_2, t \in [t_1, t_2]$$

  - Normalization

$$q_t' = Nlerp(q_{t1}, q_{t2}, t) = \frac{(1-a)q_{t1} + aq_{t2}}{||(1-a)q_{t1} + aq_{t2}||}$$



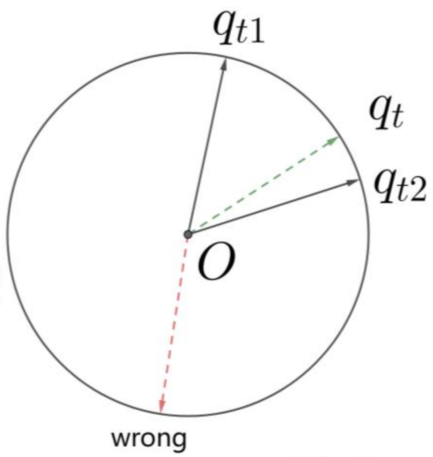
LERP



NLERP

Shortest Path Fixing of NLERP

- The shortest path



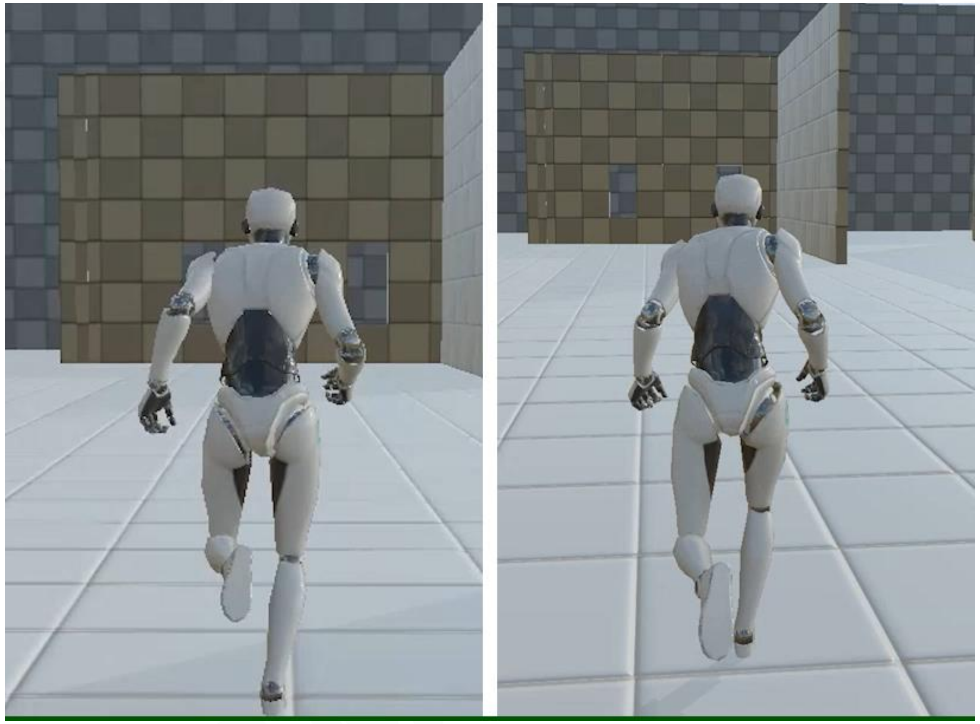
$$q_t = \begin{cases} \dfrac{(1-a)q_{t1} + aq_{t2}}{||(1-a)q_{t1} + aq_{t2}||} & q_{t1} \cdot q_{t2} \geq 0 \\ \dfrac{(1-a)q_{t1} - aq_{t2}}{||(1-a)q_{t1} - aq_{t2}||} & q_{t1} \cdot q_{t2} < 0 \end{cases}$$

$$q_{t1} = a + bi + cj + dk$$

$$q_{t2} = e + fi + gj + hk$$

$$q_{t1} \cdot q_{t2} = ae + bf + cg + dh$$

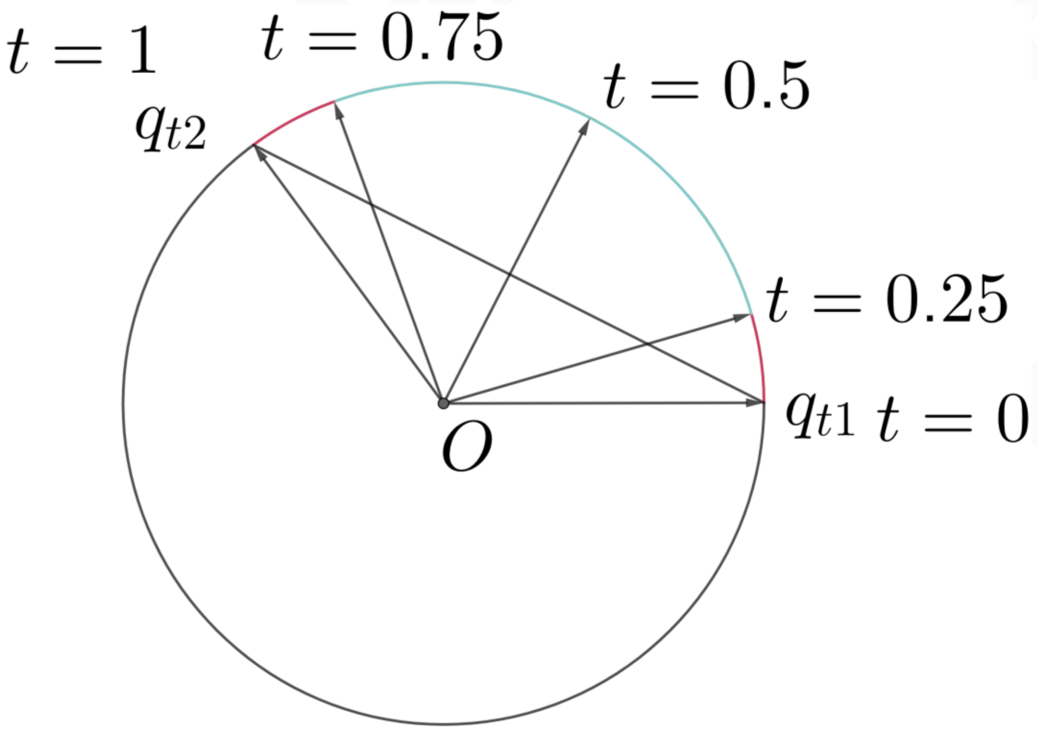
$$= cos(\theta)||q_{t1}||||q_{t2}||$$

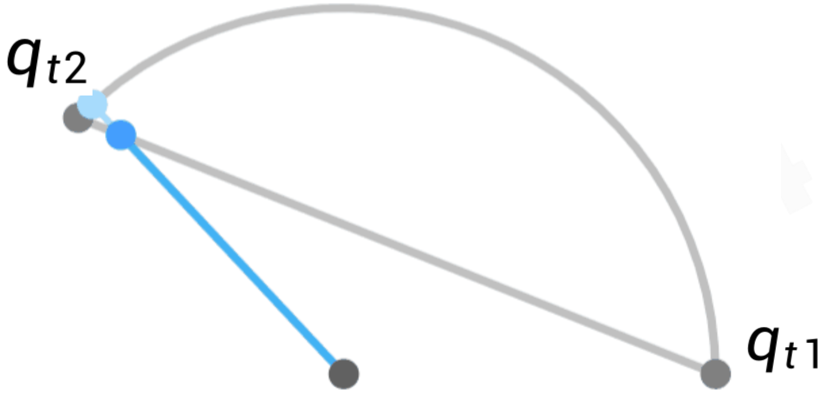


Wrong Path Right Path

Problem of NLERP

- Non-constant angular speed of NLERP



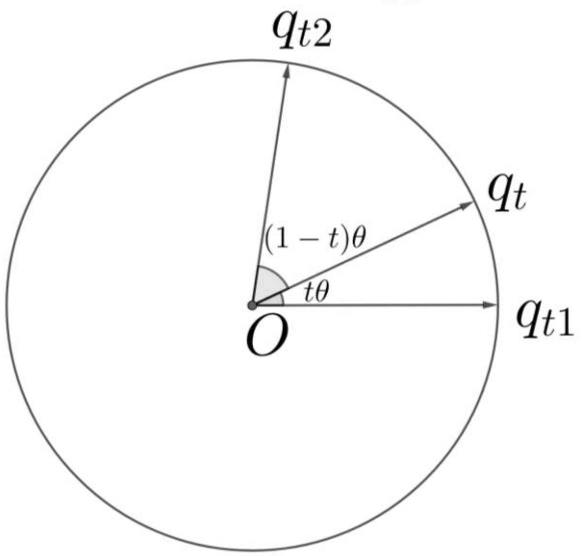


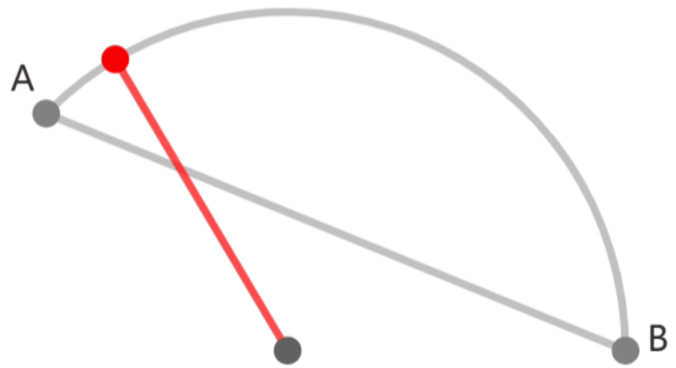
SLERP : Uniform Rotation Interpolation

- SLERP for quaternion

$$q_t = Slerp(q_{t1}, q_{t2}, t) = \frac{\sin((1-t)\theta)}{\sin(\theta)} \cdot q_{t1} + \frac{\sin(t\theta)}{\sin(\theta)} \cdot q_{t2}$$

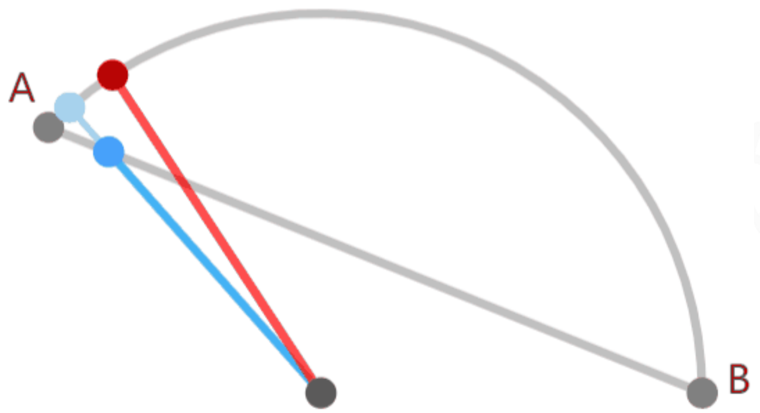
$$\theta = arccos(q_{t1} \cdot q_{t2})$$



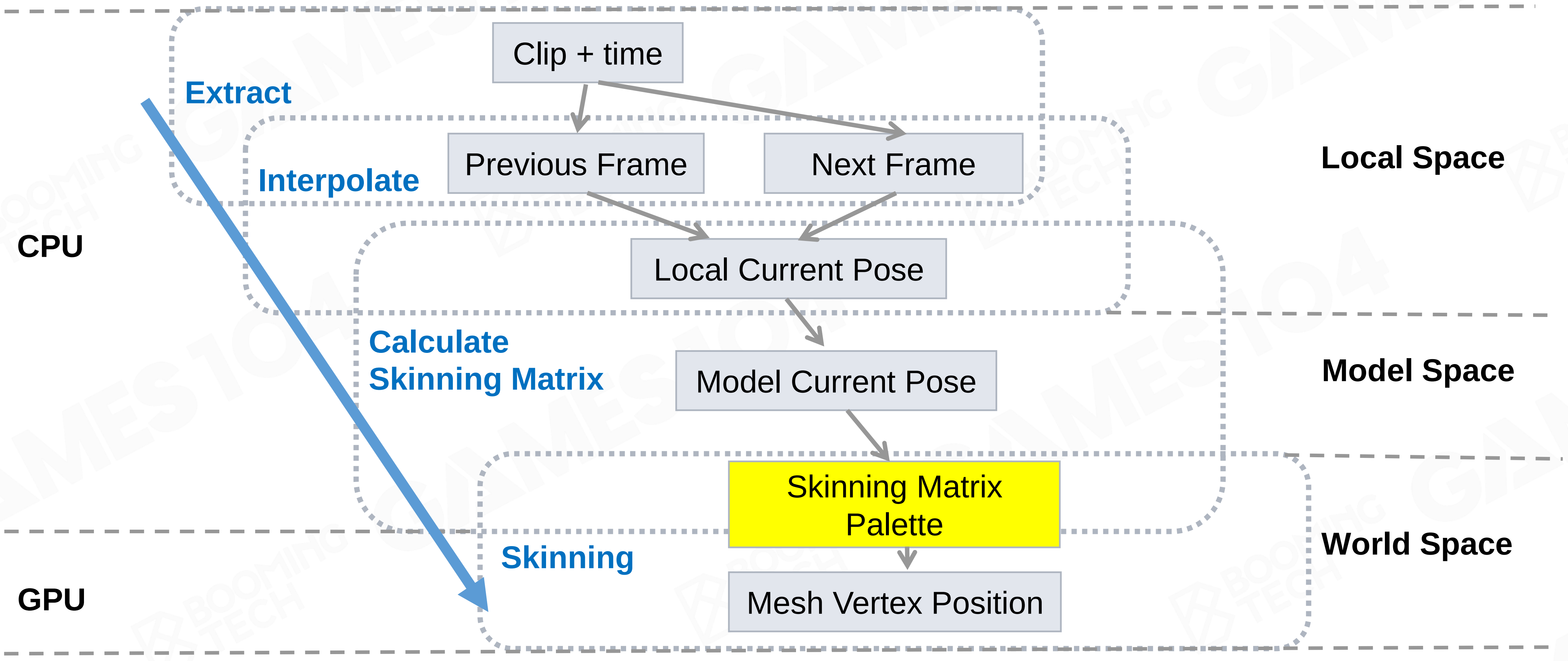


NLERP vs. SLERP

- **NLERP**
  - Non-constant angular speed
  - Almost constant angular speed when θ is small
- **SLERP**
  - Constant angular speed
  - May have zero-divide problem when θ is small
- **Combination**
  - Widely used in 3A-game development
  - Use SLERP when θ is large, and NLERP when θ is almost zero



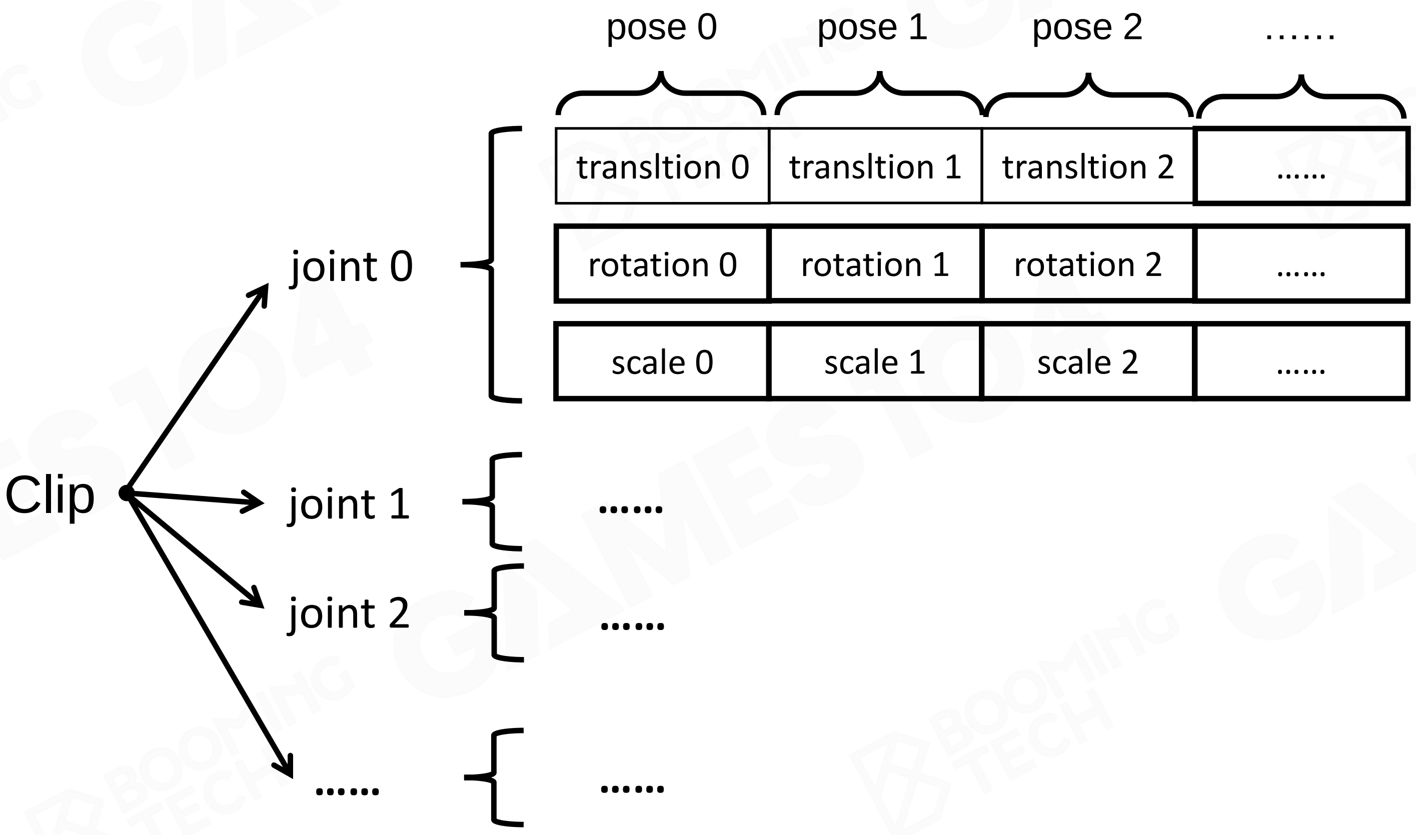
Simple Animation Runtime Pipeline



Animation Compression

Animation Clip Storage

- Animation clip split to seperated joint pose sequences
- Joint pose sequence splits to seperated translation, rotation and scale tracks



Animation Data Size

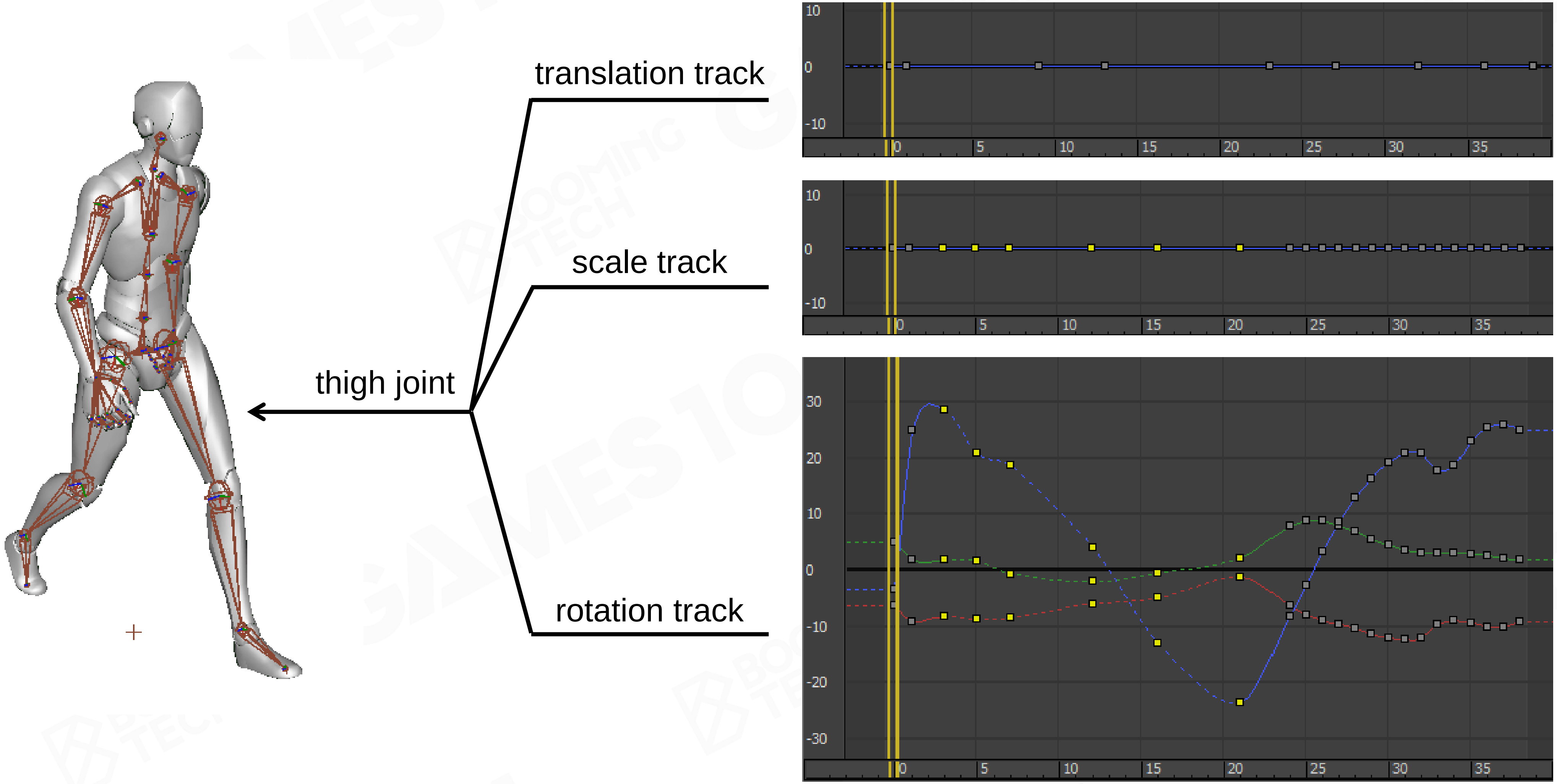
- Single clip size estimation

Frame rate	*	Storage needed per frame	*	Joint count per character	=	Storage needed per clip second
		scale 3x float translation 3x float rotation 4x float				
30 frame/s		40 Byte/frame		50~100		58.59~117.19KB/s

e.g. To develop a game containing over 150 unique character,
like League of Legends,
each of these has 30 clips with length about 5s,
It takes about: 1.26~2.51GB

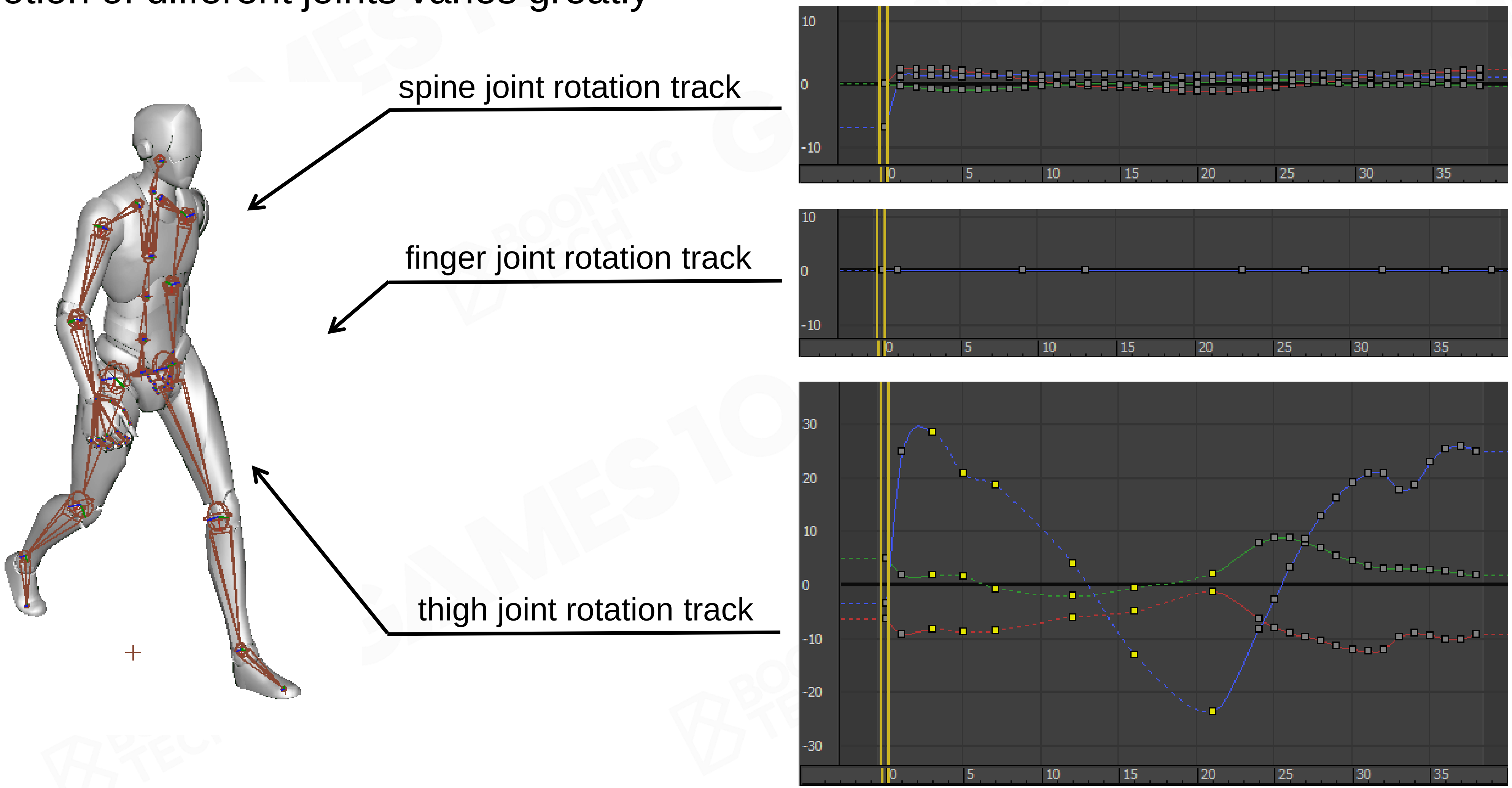
Distinction among Animation Tracks

For the same joint, rotation, translation and scale changes vary greatly



Distinction among Joints

The motion of different joints varies greatly

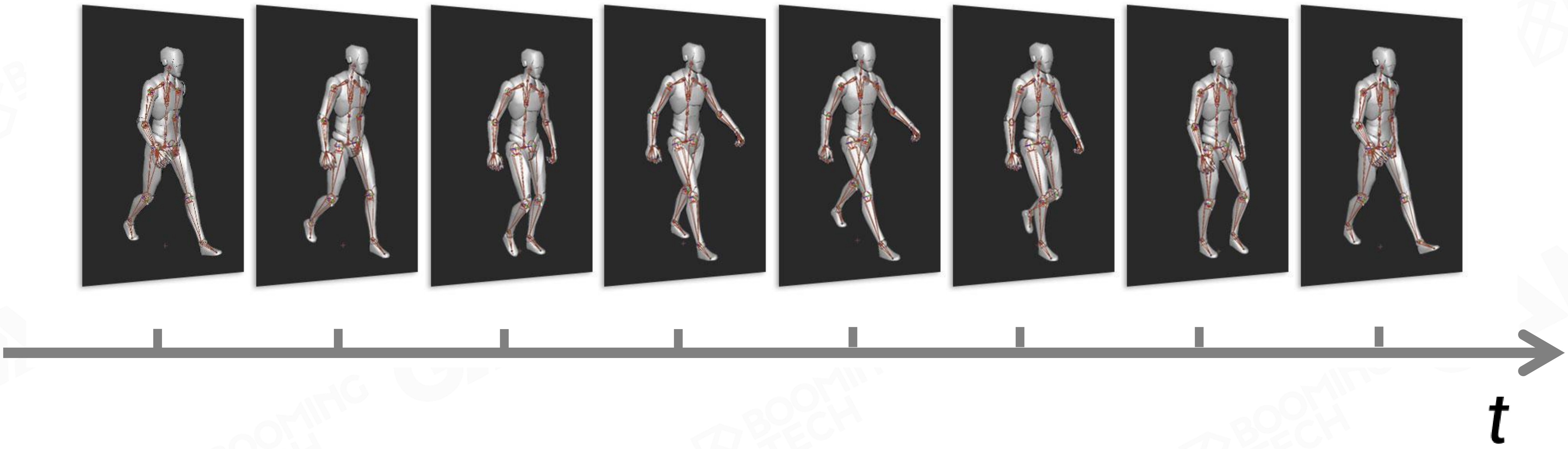


Simplest Compression - DoF Reduction

- Scale
  - Discard scale track (Usually not changed in humanoid skeleton except facial joints)
- Translate
  - Discard translate track (Usually not changed in humanoid skeleton except the pelvis, facial joint and other special joints)

Keyframe

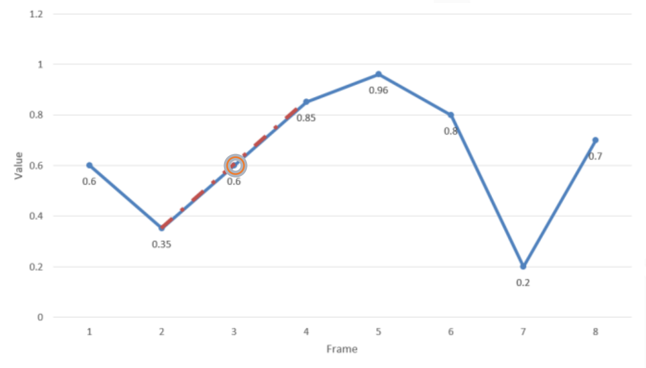
A key frame (or keyframe) in animation and filmmaking is a drawing or shot that defines the starting and ending points of any smooth transition



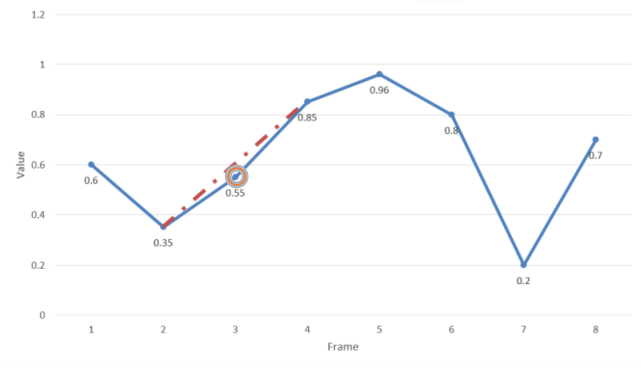
Keyframe Extraction - Linear Keys Reduction

Remove those frames which can be fitted by linear interpolation of adjacent frames

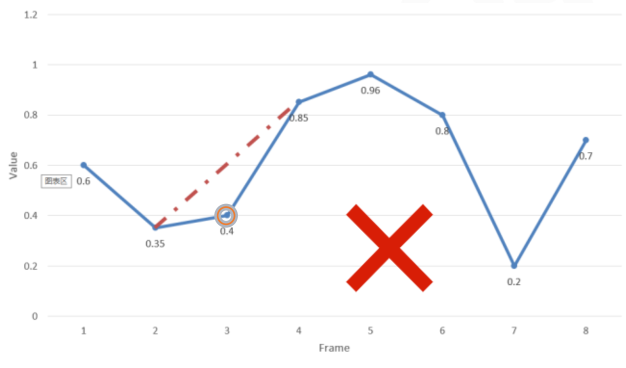
```
KeyFrame = {}
for i = 1 to n-1 do
    frame_interp = Lerp(frame[i-1], frame[i+1])
    error = Diff(frame[i], frame_interp)
    if isNotAcceptable(error) then
        KeyFrame.insert(frame[i])
    end
end
```



Accurate Case



Acceptable Case



Not Acceptable Case

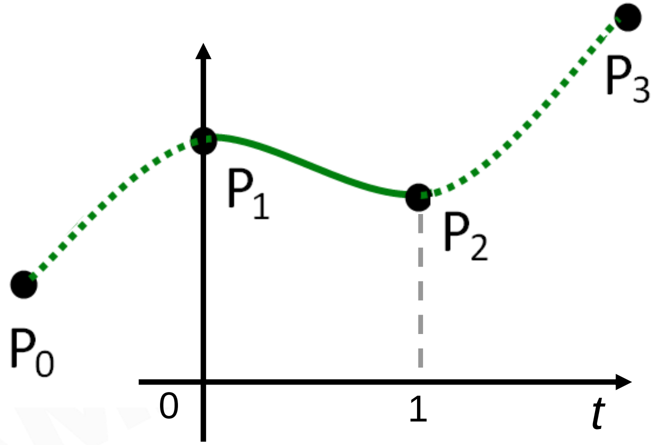
Catmull-Rom Spline

Four control points P_0, P_1, P_2, P_3 will make a curve from P_1 to P_2

- α : affects how sharply the curve bends at control points(usually $\alpha = 0.5$)
- t : the interpolation coefficient

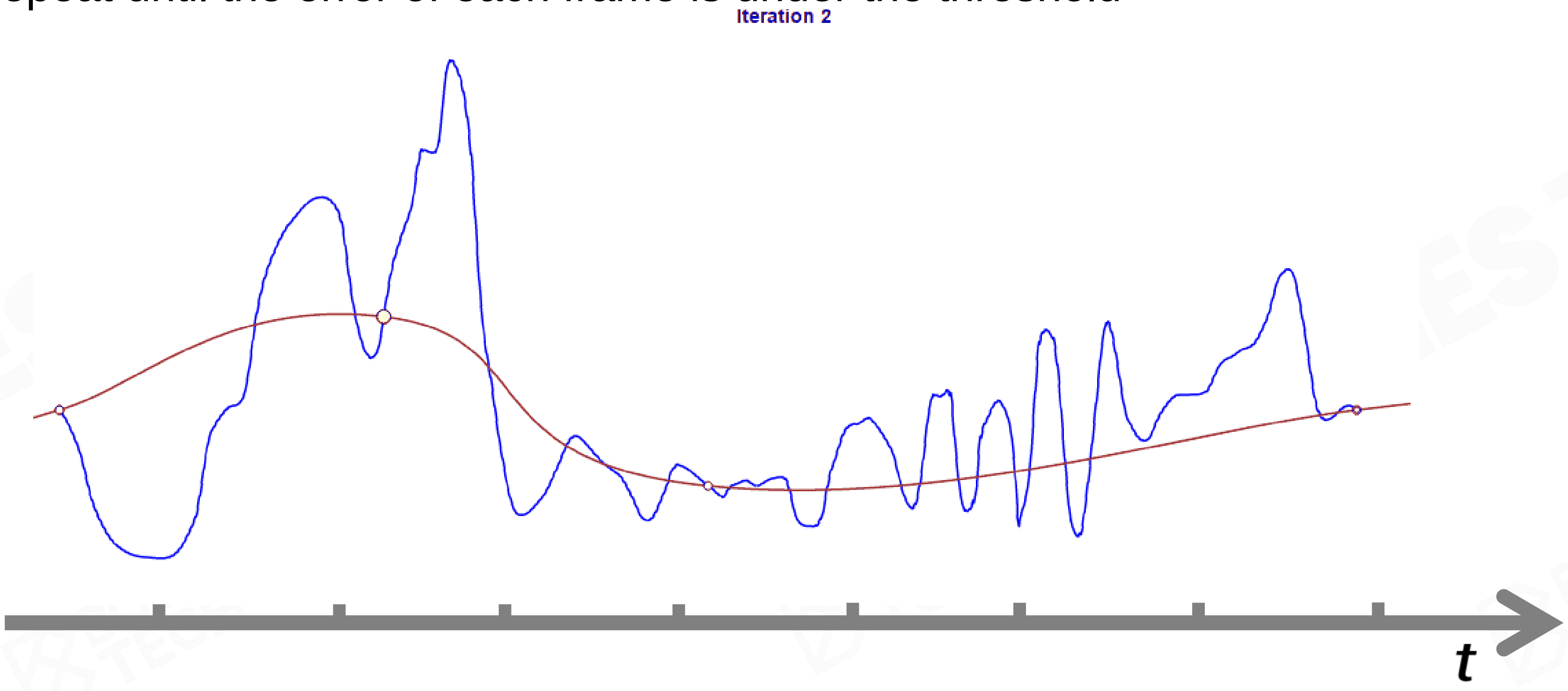
Interpolate on the curve with t in range (0, 1)

$$P(t) = \begin{bmatrix} 1 & t & t^2 & t^3 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 & 0 \\ -a & 0 & a & 0 \\ 2a & a-3 & 3-2a & -a \\ -a & 2-a & a-2 & a \end{bmatrix} \begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{bmatrix}$$



Catmull-Rom Spline

- Fitting Process
  - Make a Catmull-Rom spline with the middle 2 control points at both ends of the original curve
  - Iteratively add control points like binary search
  - Calculate inner curve by the most closet 4 points
  - Repeat until the error of each frame is under the threshold



Float Quantization

Use less bits integer to represent limited range and accuracy float value

$$DesiredBits = \lceil log_2 \frac{Range}{Accuracy} \rceil$$

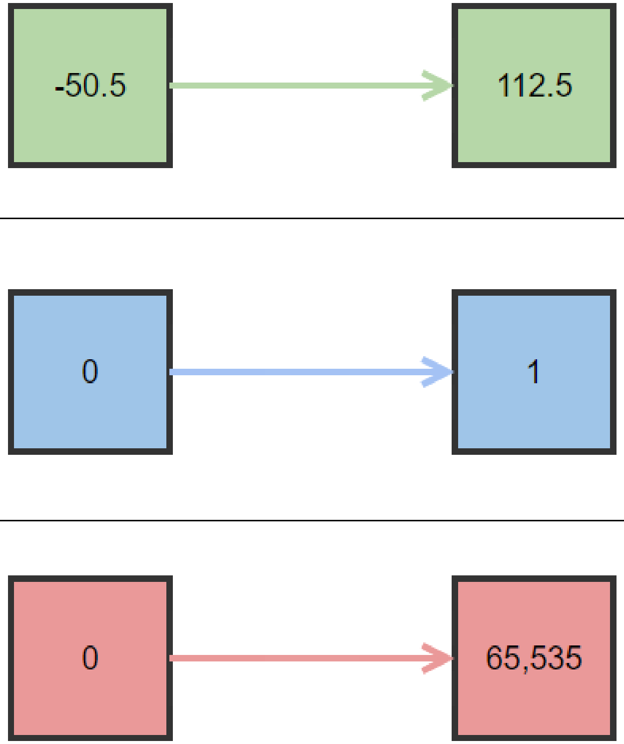
Example
Translation Range [0, 10], Accuracy 0.001m:

$$DesiredBits = \lceil log_2 \frac{10}{0.001} \rceil = 14bits$$

In general, 16bits can cover pose's float range and accuracy requirements in the game engine

Float Quantization

Example of linear quantizing a 32bit float to a 16bit unsigned integer.



Our original value is in a fixed range [-50.5, 112.5]

We normalize the original value

We then remap the [0,1] range to [0,65535]

Each value represents a step of 0.00001525...

An example of float value quantization

Quaternion Quantization

- 3 numbers is enough to represent a **unit quaternion** like $q = (a,b,c,\sqrt{1-(a^2+b^2+c^2)})$
- The range of the rest 3 numbers could be limited in $[-\frac{1}{\sqrt{2}},\frac{1}{\sqrt{2}}]$, if always omitting the number with largest absolute value

$$a^2+b^2+c^2+d^2=1, |a| \geq max(|b|,|c|,|d|)$$

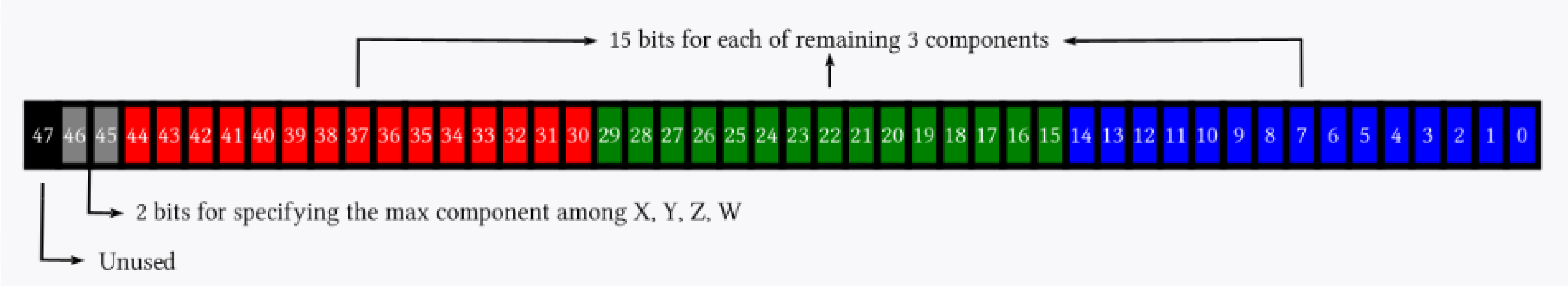
$$2b^2 < a^2+b^2 < a^2+b^2+c^2+d^2=1$$

$$b^2 \leq \frac{1}{2} \Rightarrow -\frac{1}{\sqrt{2}} \leq b \leq \frac{1}{\sqrt{2}}$$

Similarly, $-\frac{1}{\sqrt{2}} \leq c \leq \frac{1}{\sqrt{2}}$ and $-\frac{1}{\sqrt{2}} \leq d \leq \frac{1}{\sqrt{2}}$

Quaternion Quantization

- Use 2bits to represent which number is discard
- Use 15bits storage for each number, ranged in $[-\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}]$, precision $\sqrt{2}/32767 \approx 0.000043$
- Finally a quaternion could store in a 48bits storage, cutting down from 128bits



Size Reduction from Quantization

- The key point of quantization is to find a proper error threshould, and use as less bits of storage as possible
- Keyframe and quantization can use together to obtain a better compression ratio

Joint Pose Per Frame	
scale	1x float
translation	3x float
rotation	4x float

↓

32x bytes

Before

Joint Pose Per Frame	
scale	16x bits
translation	48x bits
rotation	48x bits

↓

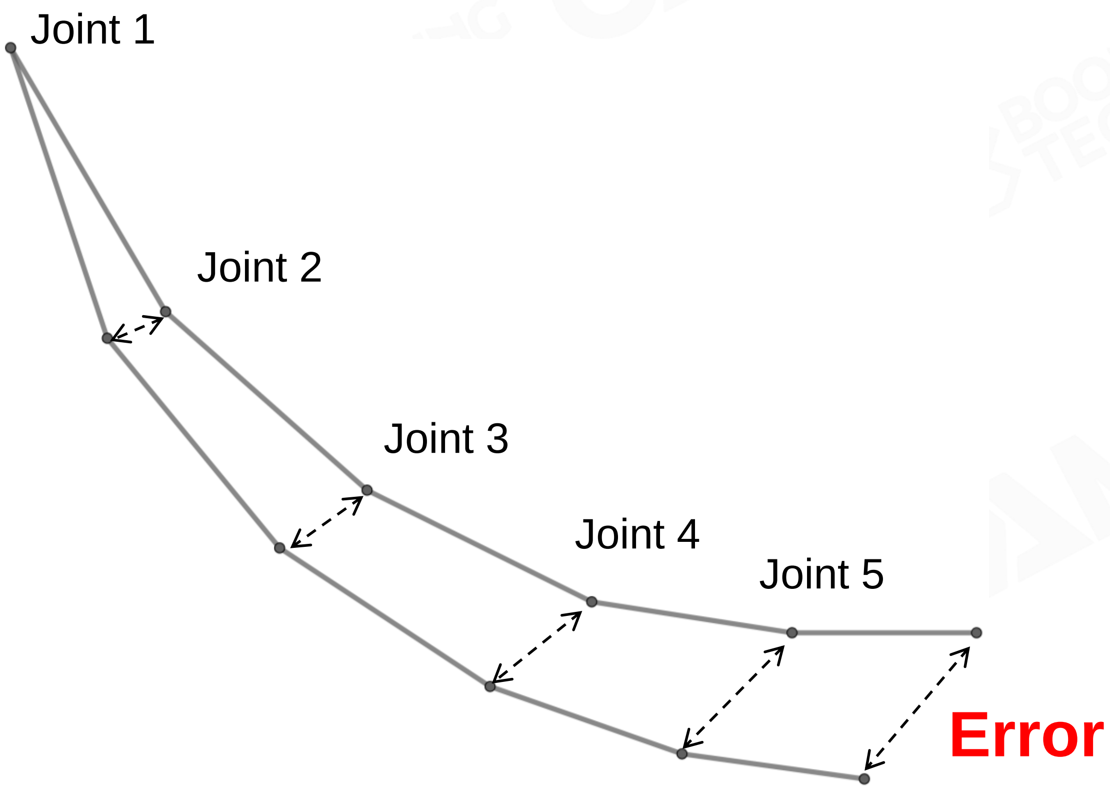
14x bytes

After

Error Propagation

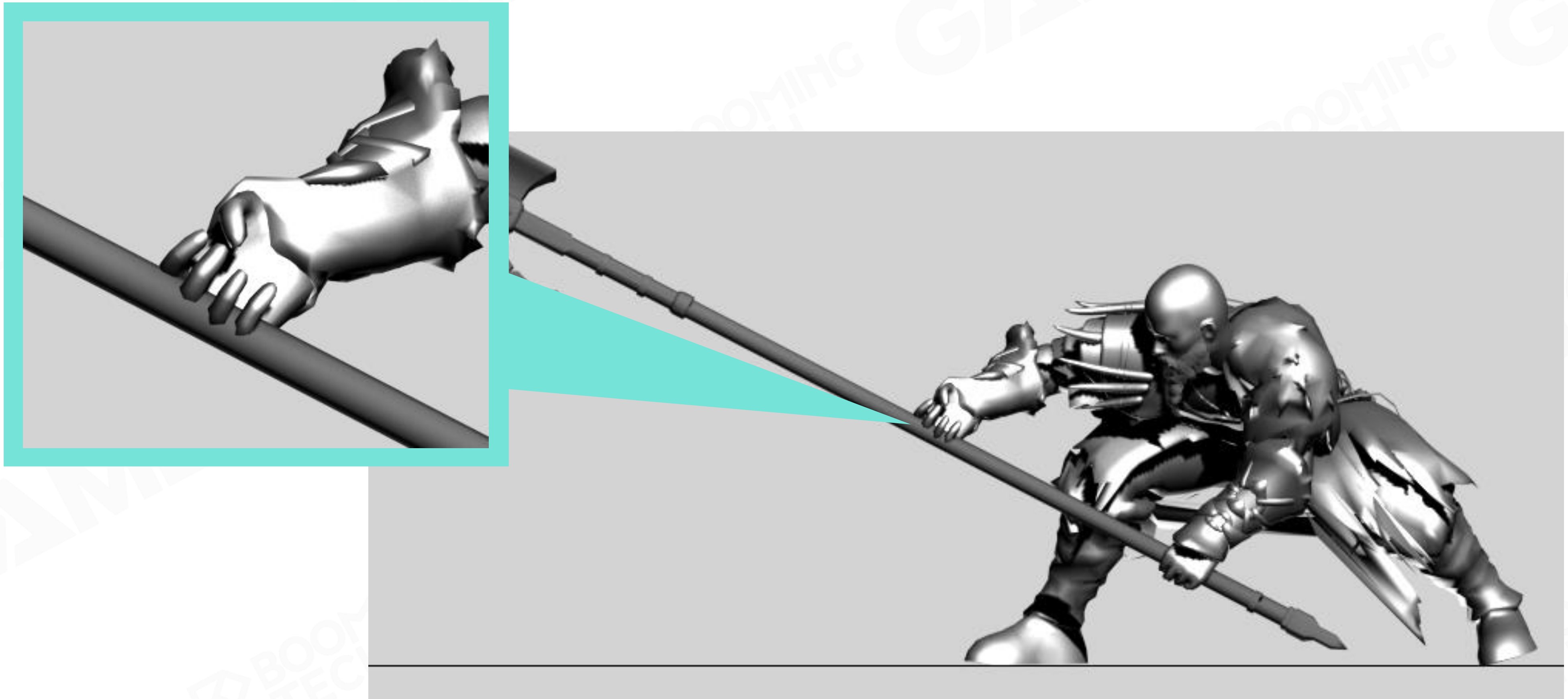
Error cuased by compression of animation data will accumulate between bones

- Bones store local space transformation data
- Bones are organized by hierachy



Joint Sensitivity to Error

Some special parts require high precision animation.



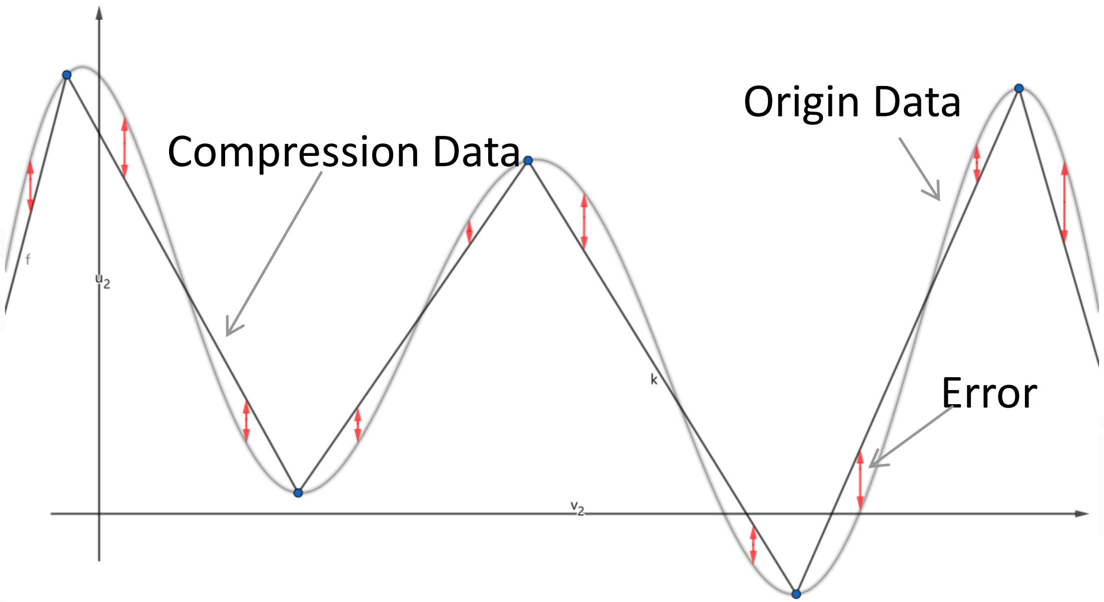
Measuring Accuracy - Data Error

Calculate the error with the diff between the interpolated transform data and the origin

Translation Error: $||T_1 - T_2||$

Rotation Error: $||R_1 - R_2||$

Scale Error: $||S_1 - S_2||$

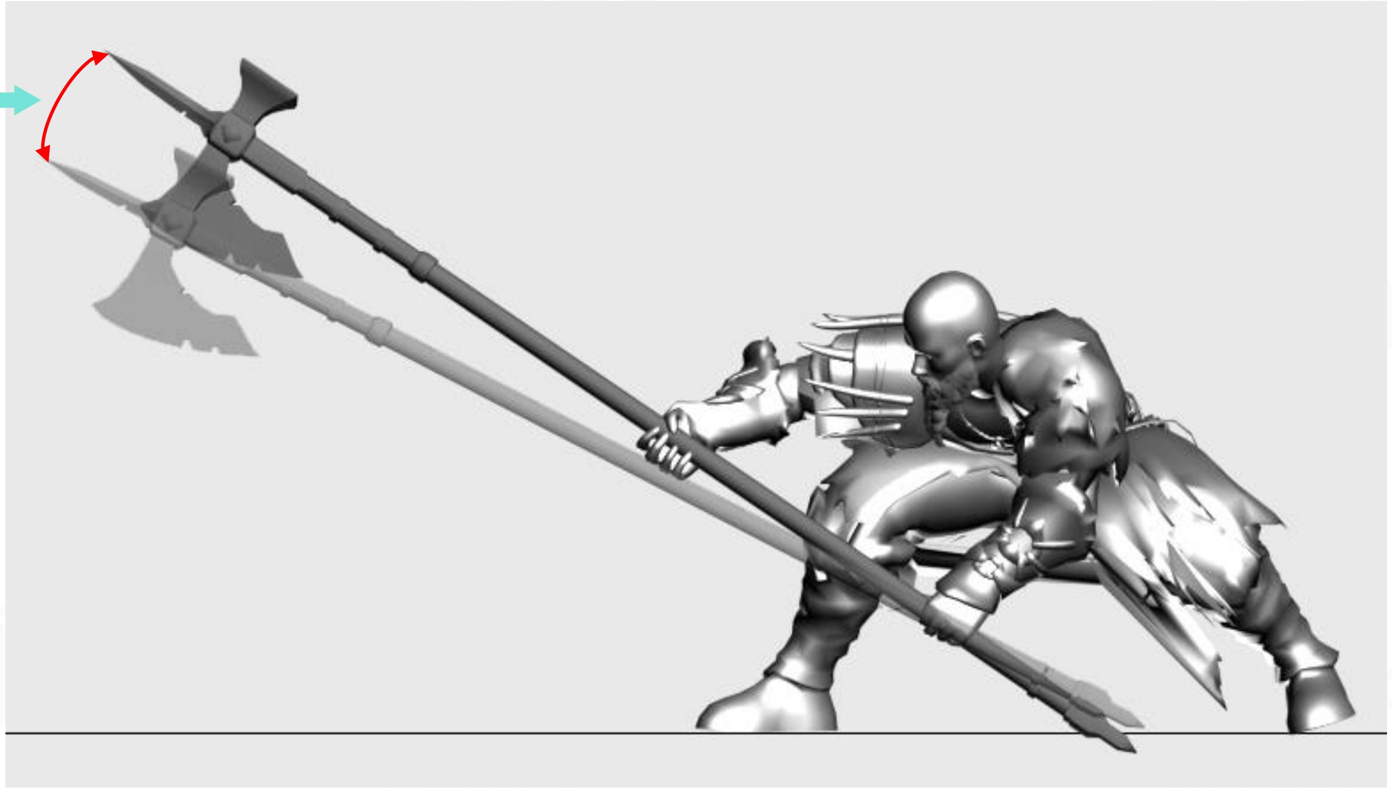


The Error Caused By Compression

Measuring Accuracy - Visual Error

Calculate the error with the diff betweent the interpolated vertex and the desired vertex

visual error



Measuring Accuracy - Visual Error

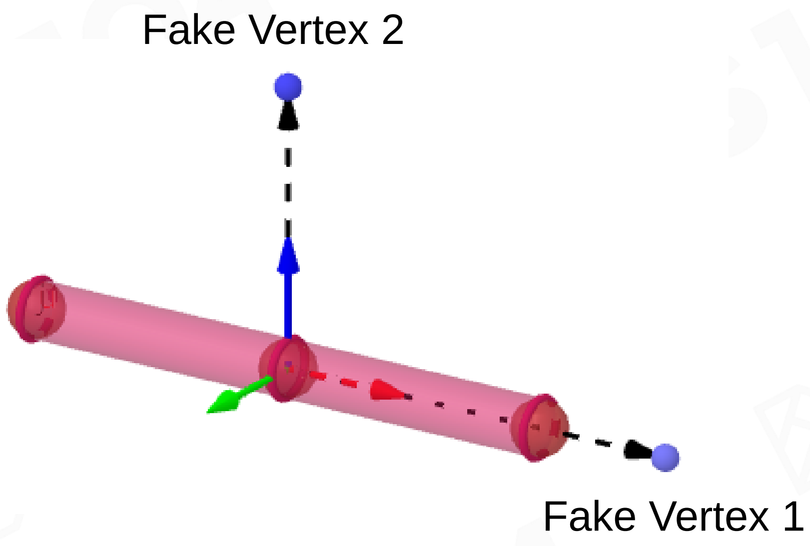
Hard to cacluate the visual error of every vertex

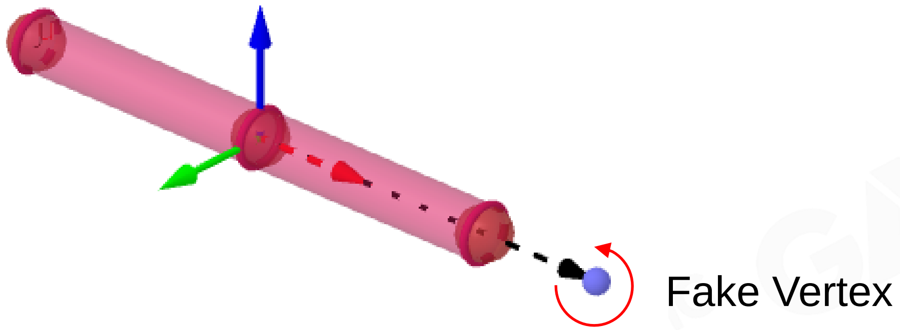
- Great amount of calculation

Estimate the visual error

- Fake vertex： Two orthogonal virtual vertices at a fixed distance from the joint （not co-linear with the joint rotation axis）

- Fake Vertex Distance Approximation
  - character bones 2 ~ 10 cm
  - large animated objects 1 ~ 10 m



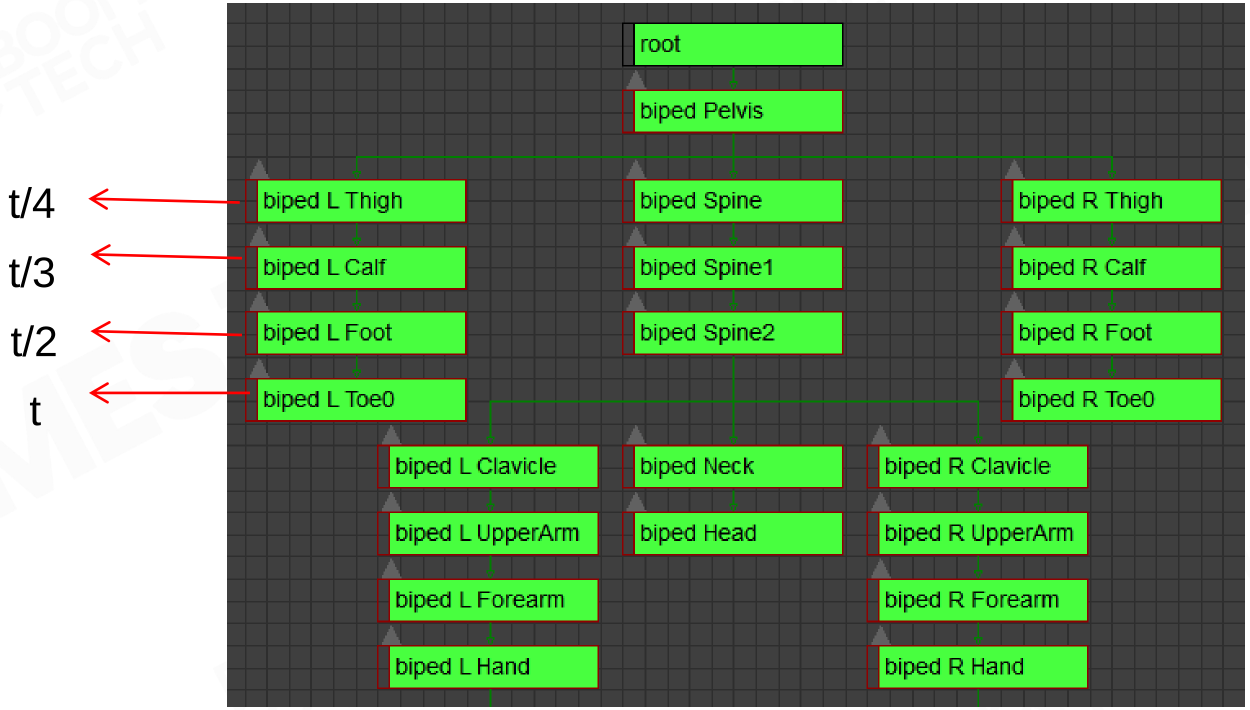


One fake vertex can't estimate error in some situation

Error Compensation - Adaptive Error Margins

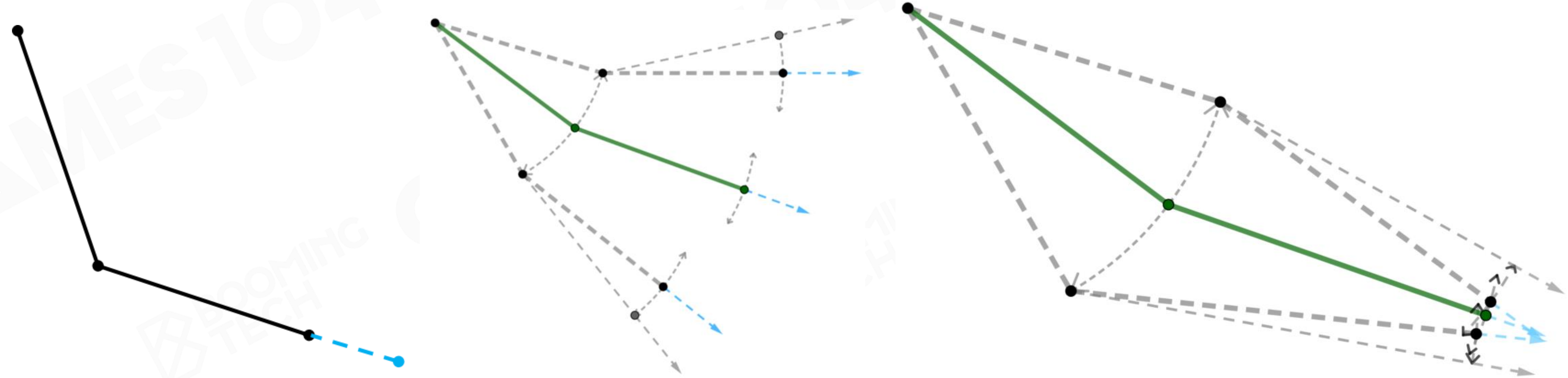
- Adaptive Error Margins
  - Use differenct accuracy threshold for different joint from end to root, in order to reduce error caused by parent joint

Max Error Threshold = t



Error Compensation - In Place Correction

- **Process**
  - Select a point on every bone except root
  - Compute the rotation of every compressed bone from root that takes the tagged point closet to its actual position in model space
  - Add the rotation to transform of compressed data
- **Pros**
  - No Overhead during decompression period since all data already computed during compression
- **Cons**
  - May produce memory overhead because of possible modification to ConstantTrack
  - May produce NoiseTrack because it directly changes keyframe data
  - Compression may cost more time



Animation DCC Process

Animation DCC Process

In general, the Animation DCC(Digital Content Creating) process includes:

- Mesh
- Skeleton binding
- Skinning
- Animation creation
- Exporting

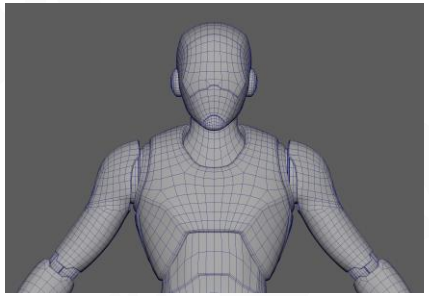
Mesh building

- Blockout Stage: Create a rough outline of the character
- Highpoly Stage:Improve the building precision
- Lowpoly Stage:Devide the surface into meshes
- Texture Stage:Add texture to character

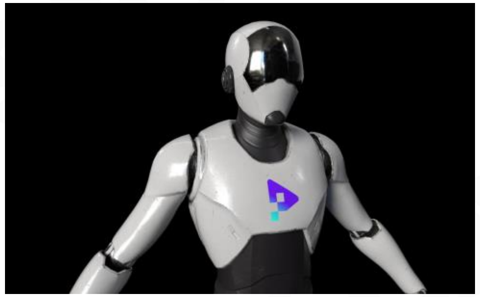
animation related

Blockout Stage

Highpoly Stage



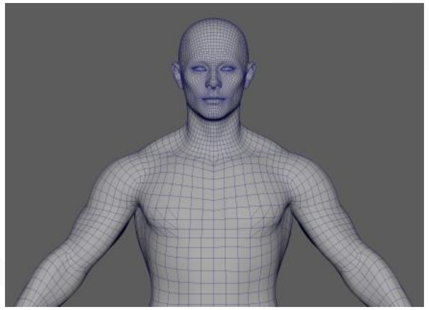
Lowpoly Stage



Texture Stage

Mesh Adjustment for Animation

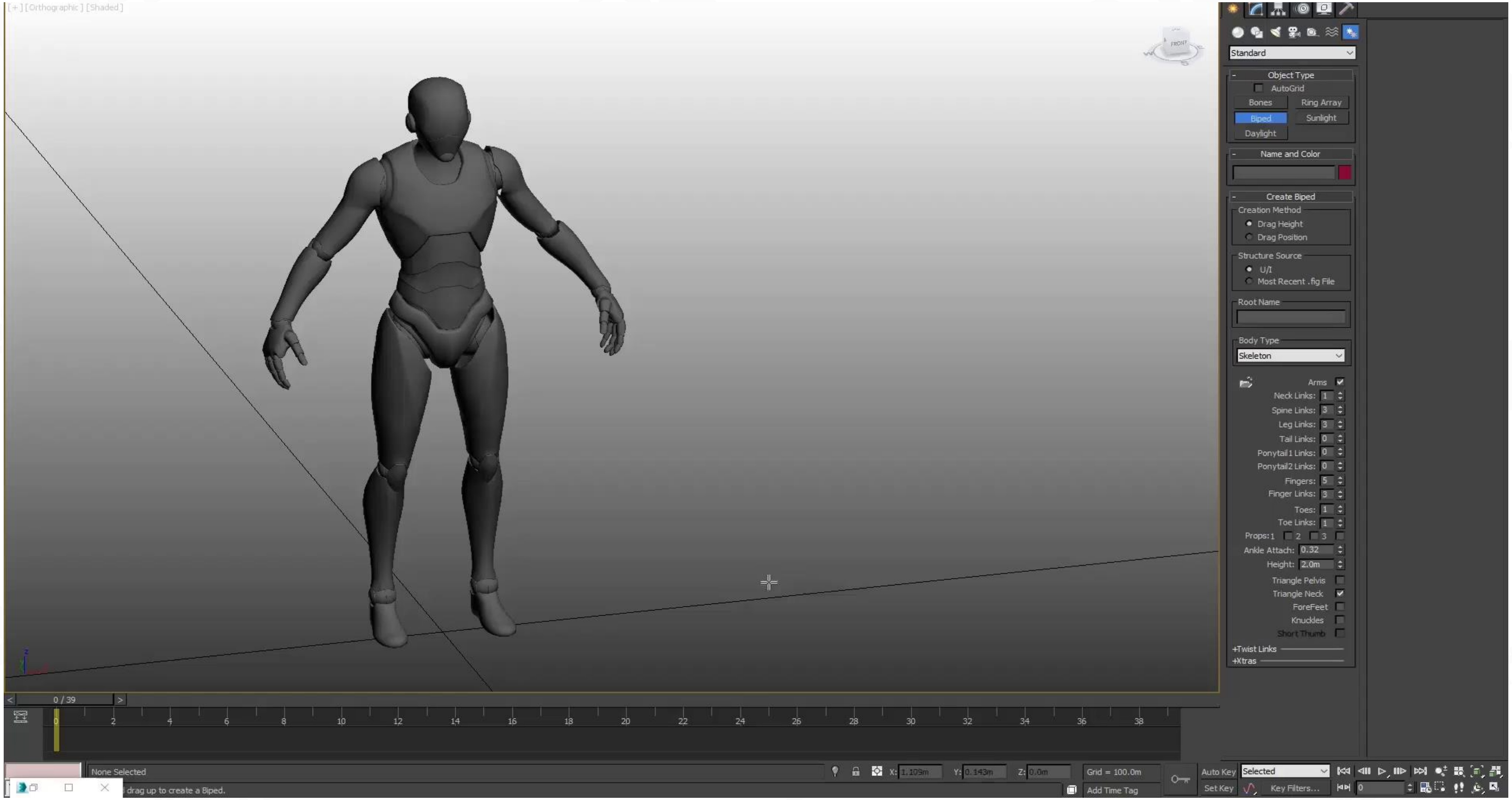
- Mesh dividing is vital for animation creating, it defines how the skin curves
- Animation turns to be weird if the meshes are too sparse
- Dense meshes cause a performance overhead



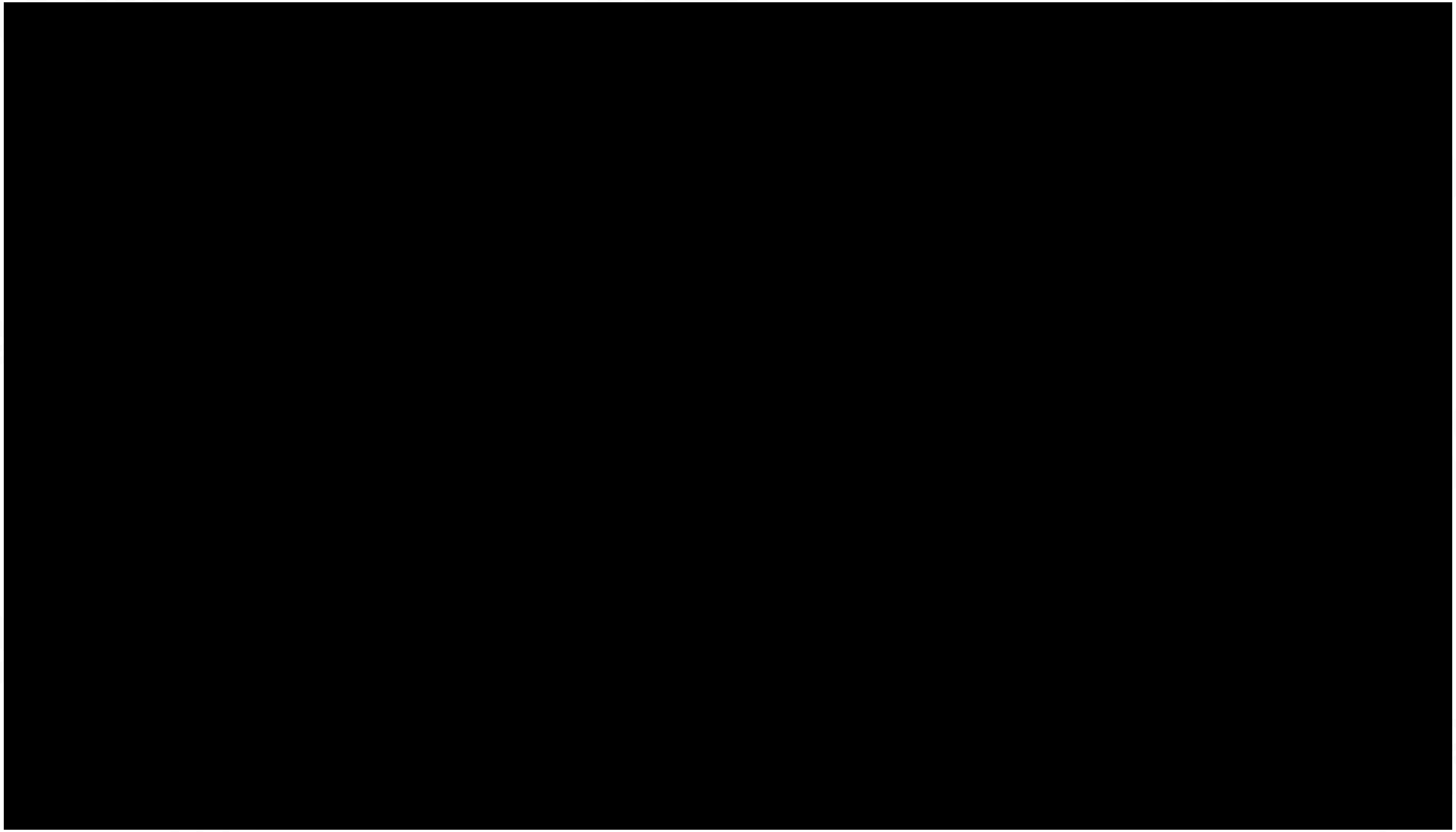
Lowpoly Stage

Mesh Dividing

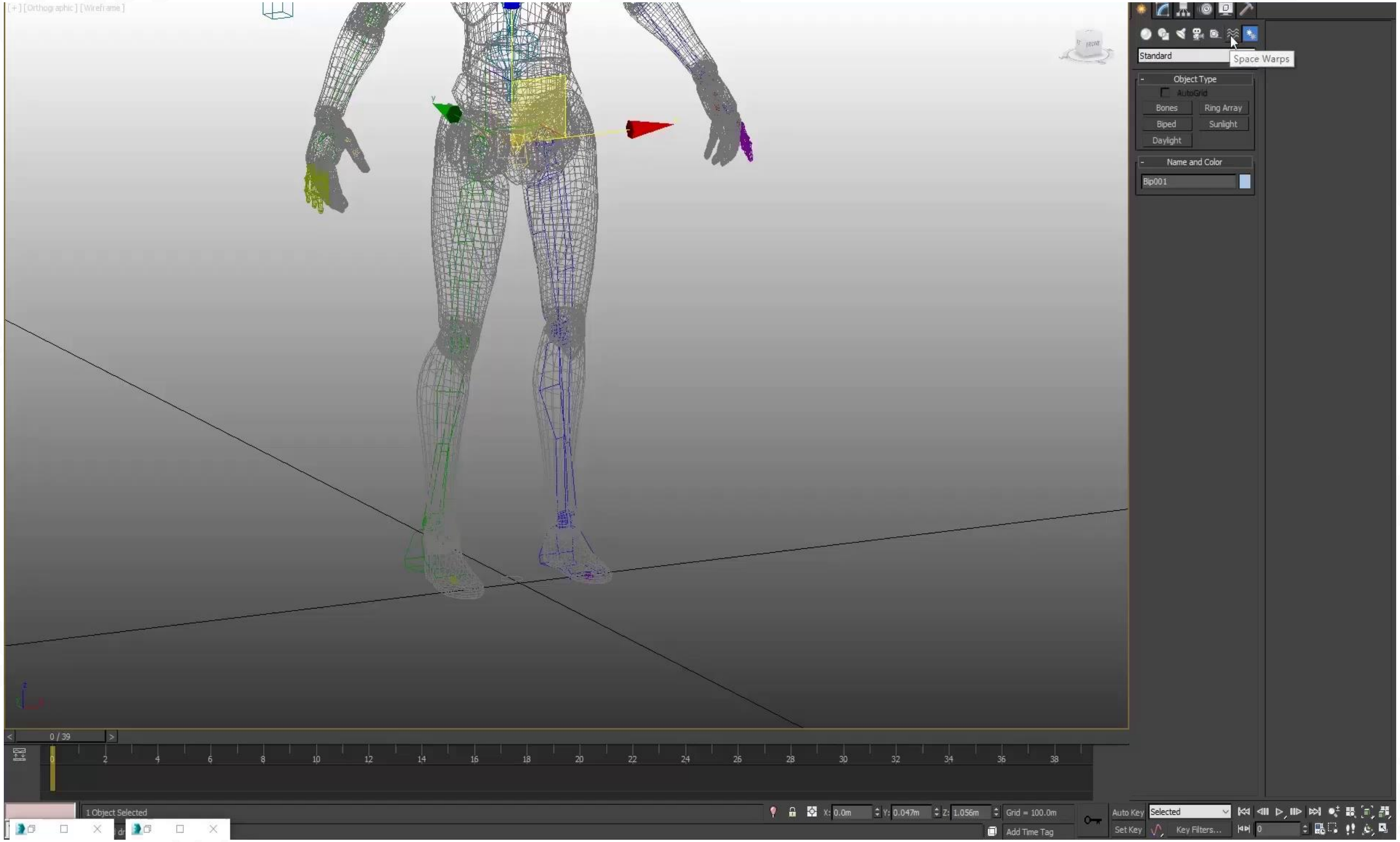
Skeleton Binding



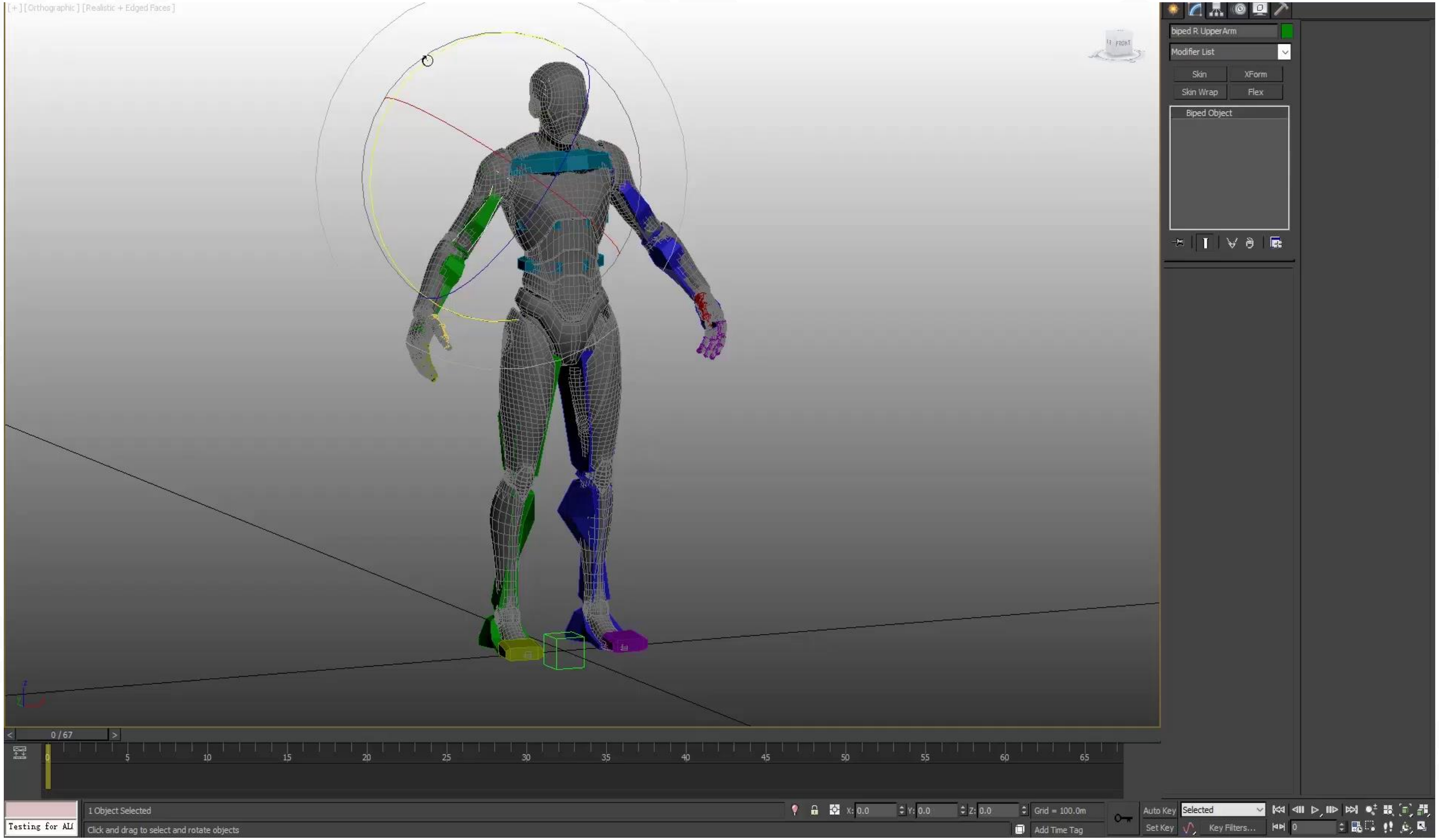
Skeleton Binding – Add Game Play Joints



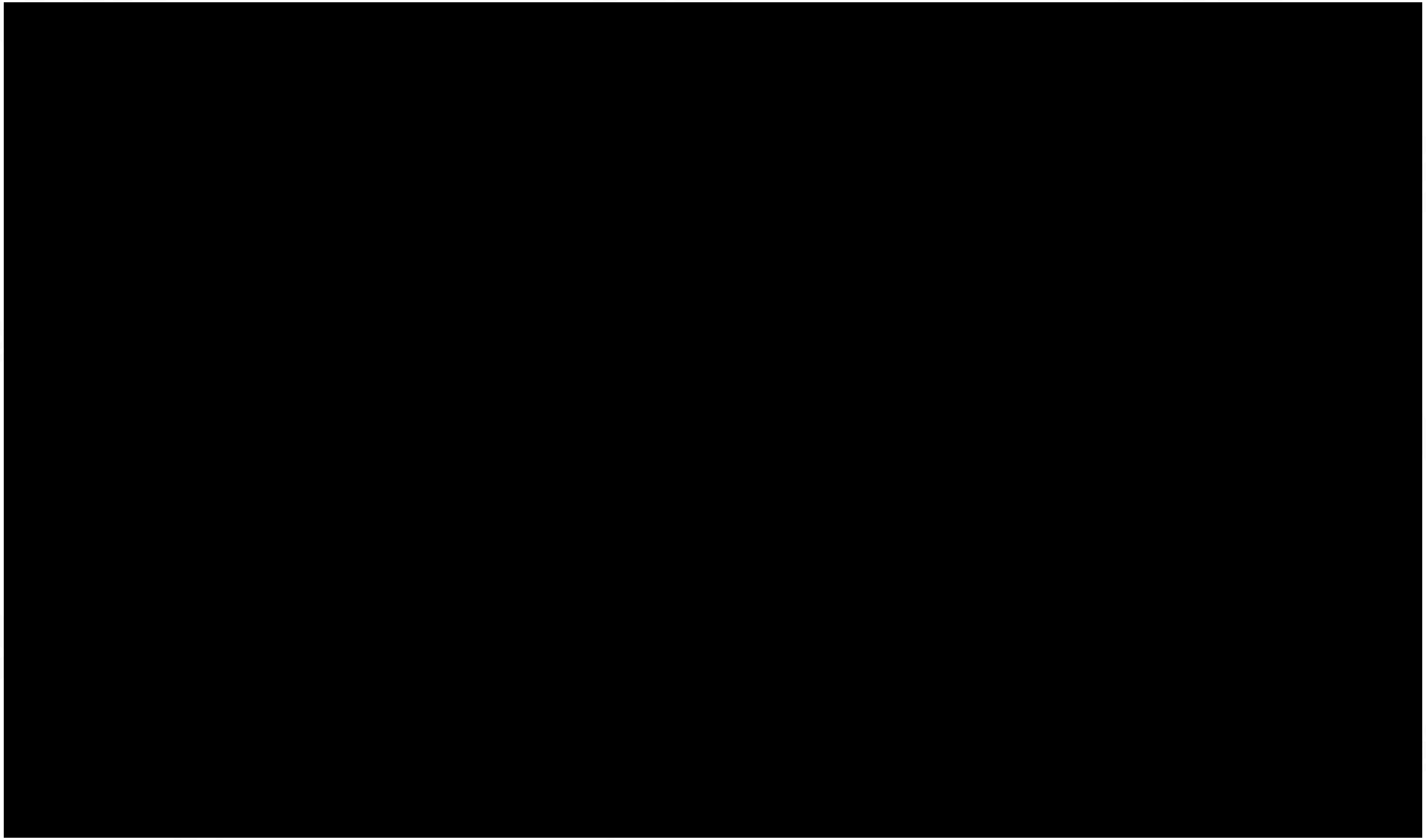
Skeleton Binding – Add Root Joint



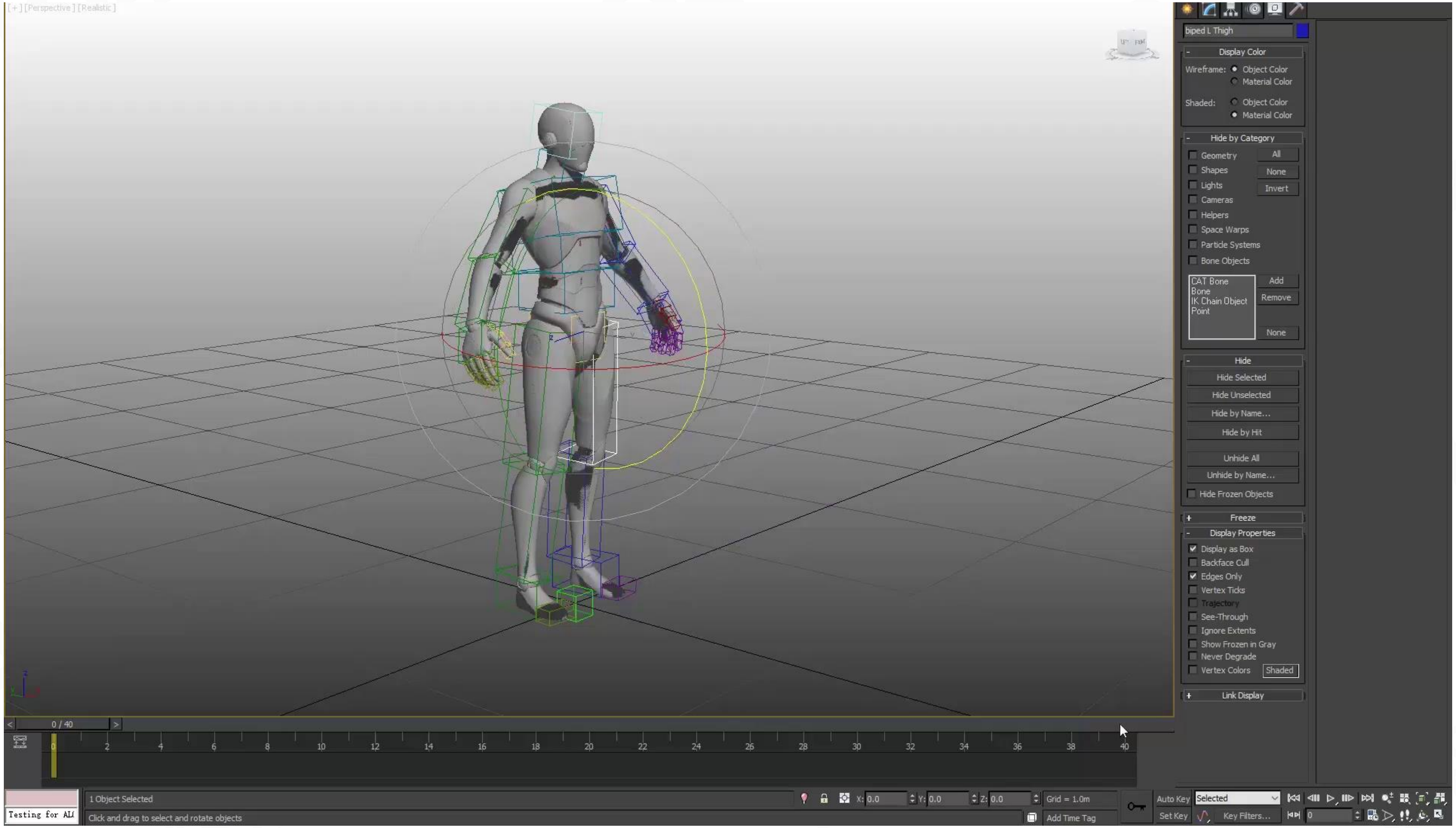
Skinning – Auto Calculation



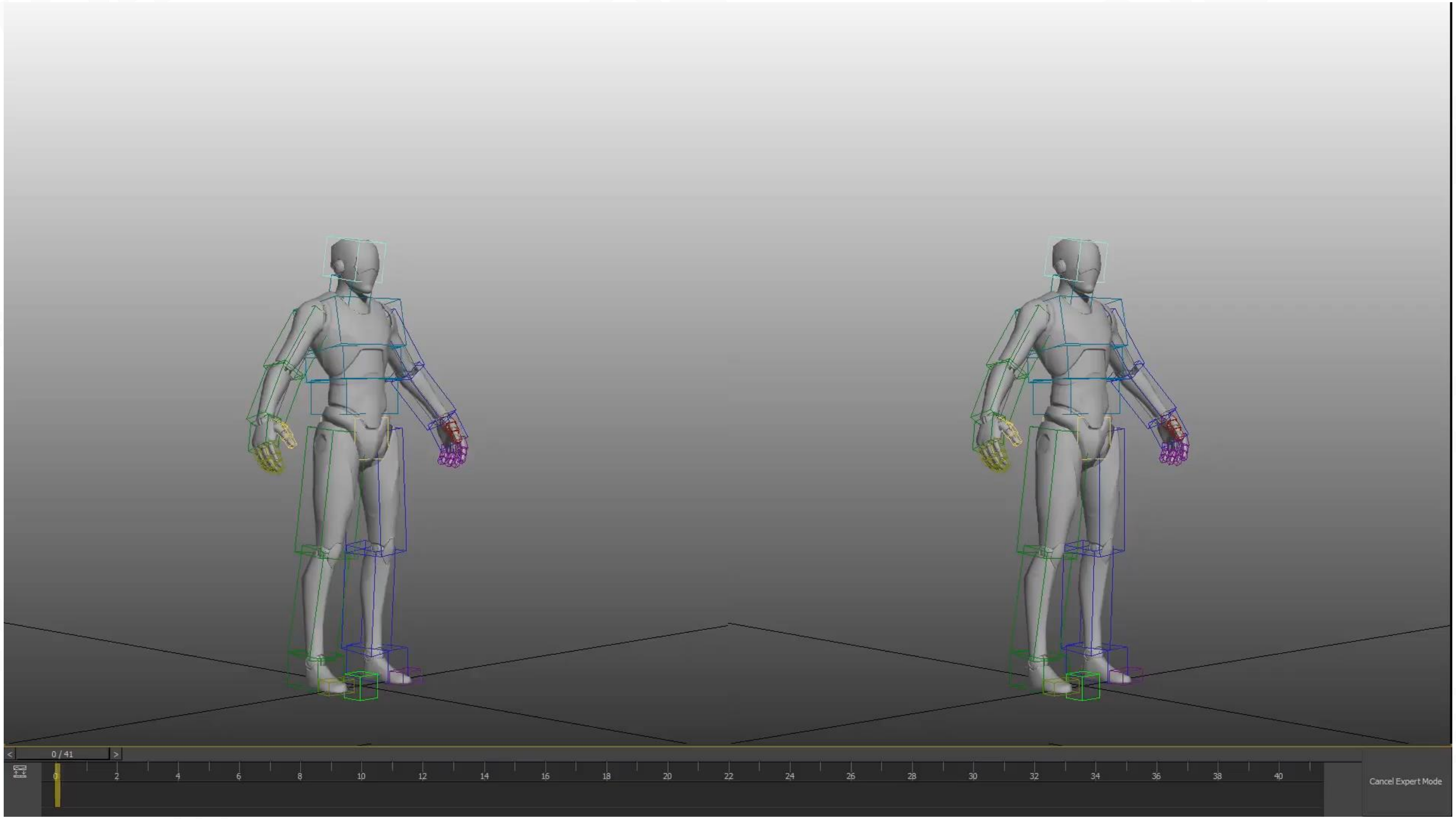
Skinning – Manual Fixing



Animation Creation



Exporting


0 / 41
0
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
32
34
36
38
40
Cancel Expert Mode

Exporting

Model mesh

skeleton

skinning data

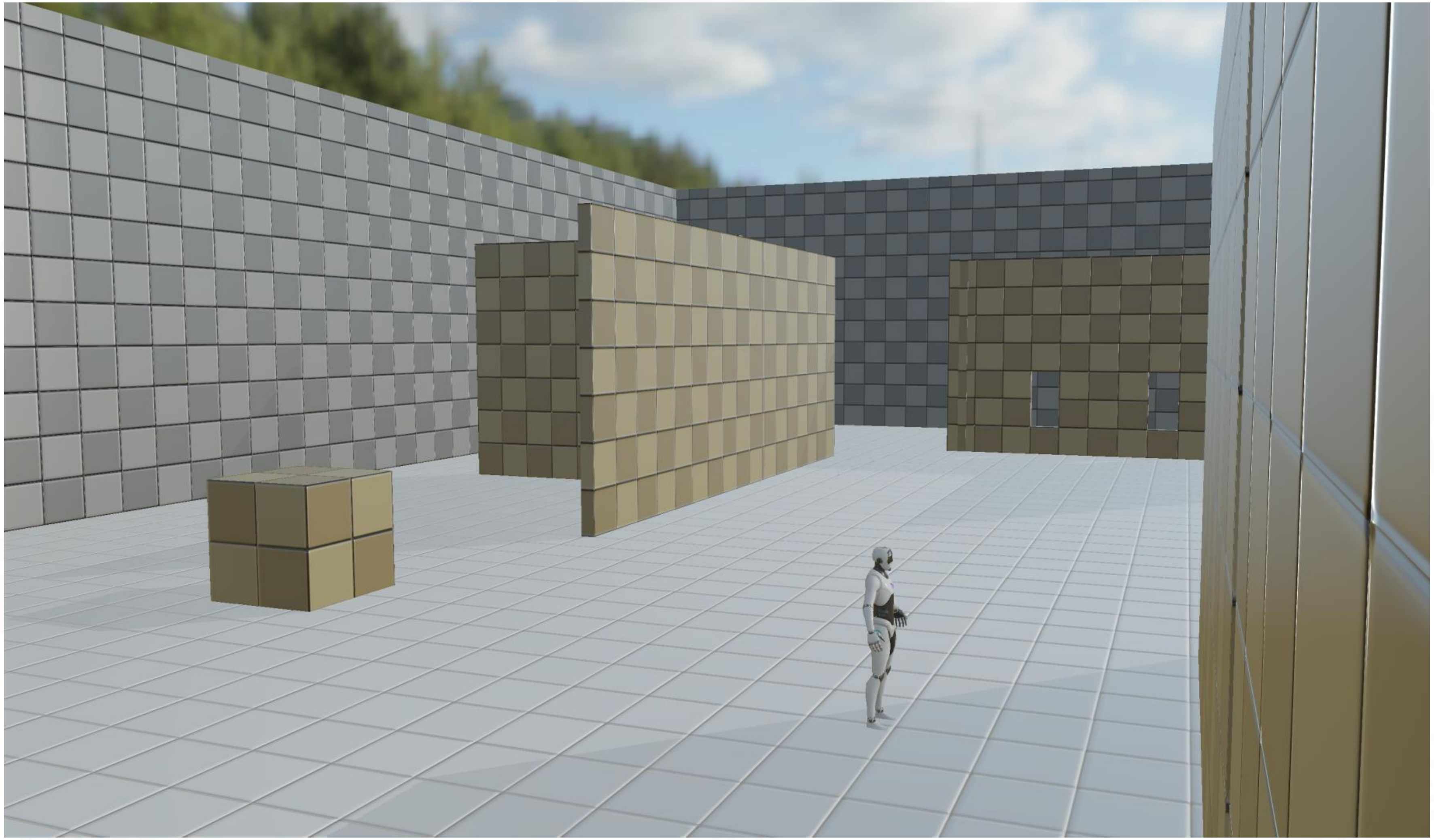
animation clip

FBX file

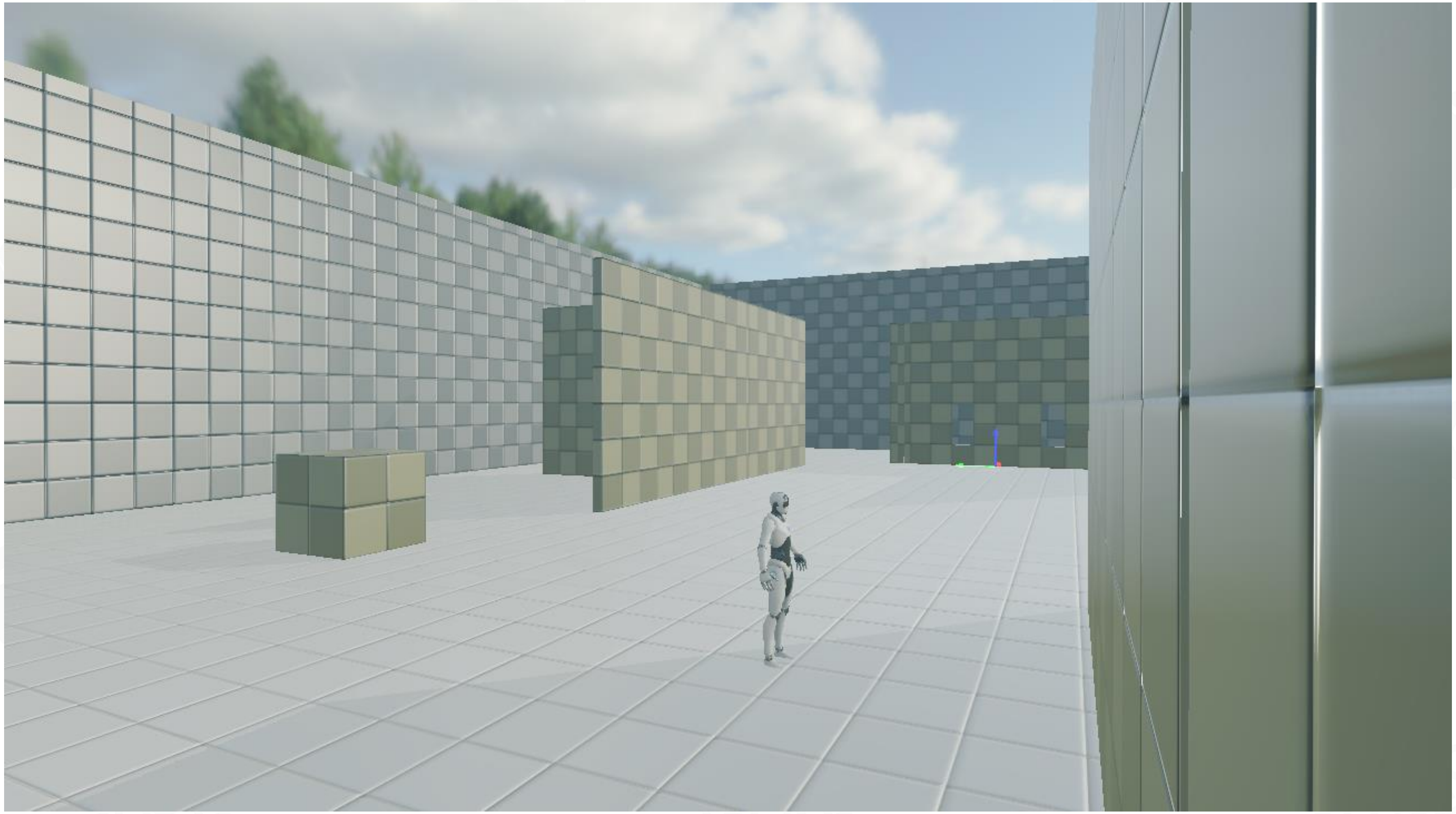
FBX: the industry-standard 3D asset exchange file format for games. It is developed by Autodesk as a proprietary format.

Exporting Dialog: Select Asset

Pilot Without Color Grading

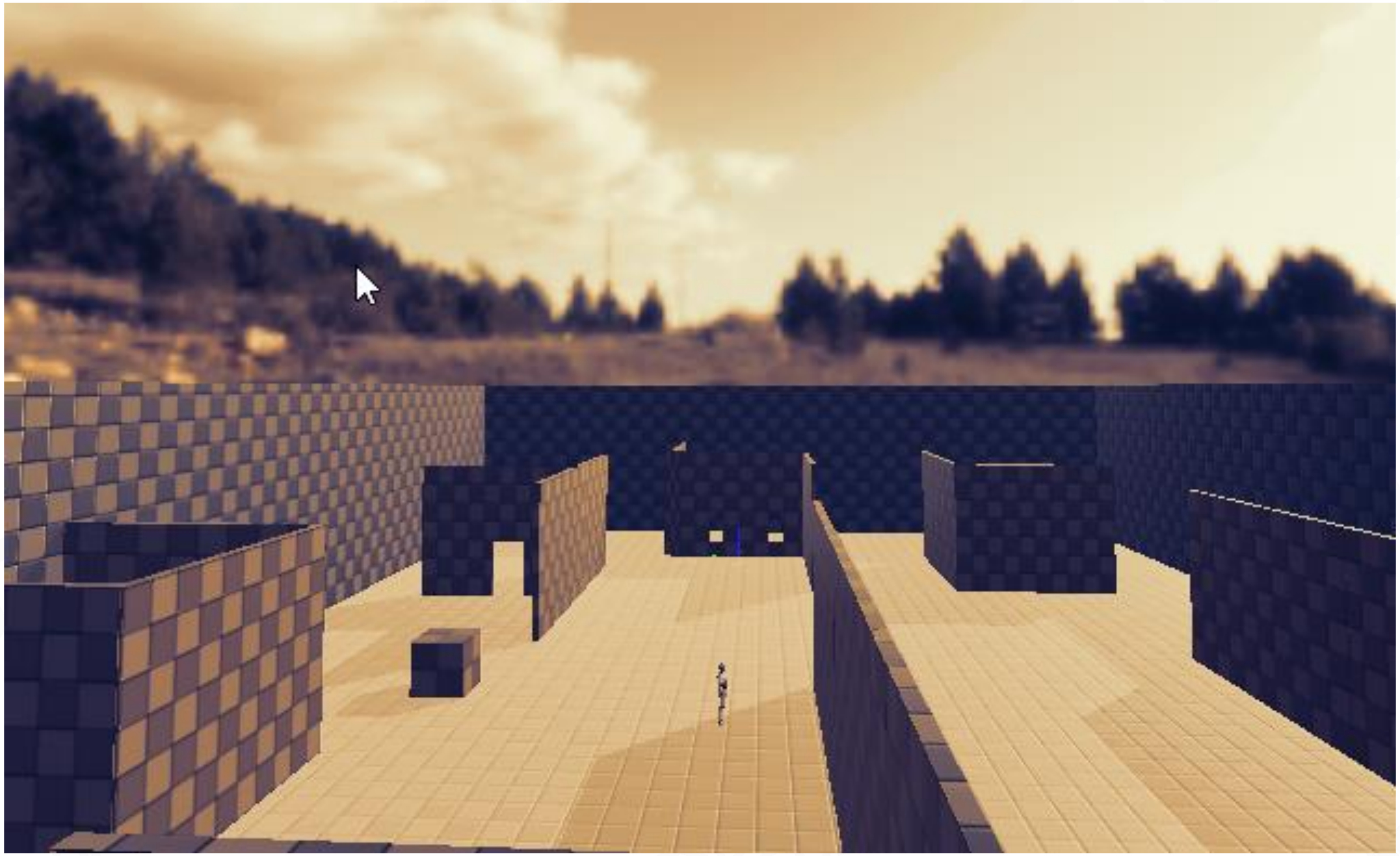


Interesting Submissions



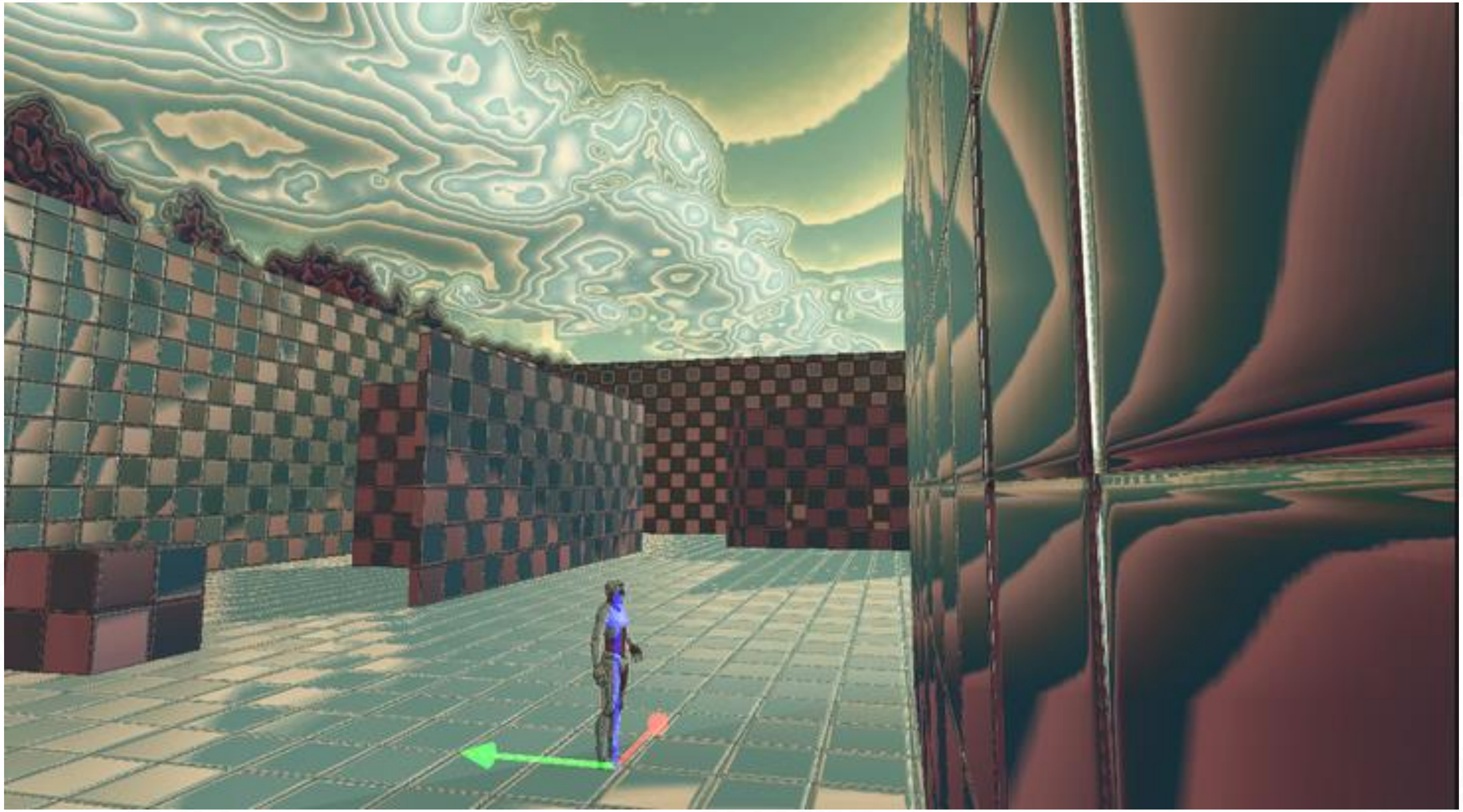
@无所谓

Interesting Submissions



@fwzhuang

Interesting Submissions



@Comarpers CMXXll Leo

Find a Better Name for Mini Engine

Due date:
May 12th 20:00

Rewards:
10 contributors will get a souvenir with the name of Mini Engine

Please scan the QR code to submit your naming

Lecture 08 Contributor

- 一将
- 喵小君
- 灰灰
- 蓑笠翁
- 小老弟
- 建辉
- Hoya
- 爵爷
- Jason
- 砚书
- BOOK
- MANDY
- 乐酱
- 灰灰
- 金大壮
- Leon
- 梨叔
- Shine
- 浩洋
- Judy
- 乐酱
- QIUU
- C佬
- 阿乐
- 阿熊
- CC
- 大喷
- 大金

Q&A

Enjoy ;) Coding

Course Wechat

Follow us for further information