Game Architecture
Animation

Bringing things to life.

Today’s Agenda

- What is animation?
- Types of animation
- Skinned animation
- Joints and skeletons
- Math!
- Clips
- Blending

Animation

- What is it?
- Cel / Sprite
- Rigid Hierarchical
- Per-Vertex
- Skinned

Animation

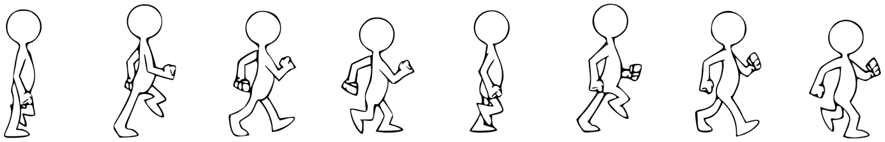
- **What is it?**
- Cel / Sprite
- Rigid Hierarchical
- Per-Vertex
- Skinned

Let’s ask Wikipedia!

Animation is the process of making the illusion of motion and the illusion of change by means of the rapid display of a sequence of images that minimally differ from each other.

Animation

- What is it?
- **Cel / Sprite**
- Rigid Hierarchical
- Per-Vertex
- Skinned



Great for 2D, not really great for 3D.

Animation

- What is it?
- Cel / Sprite
- **Rigid Hierarchical**
- Per-Vertex
- Skinned

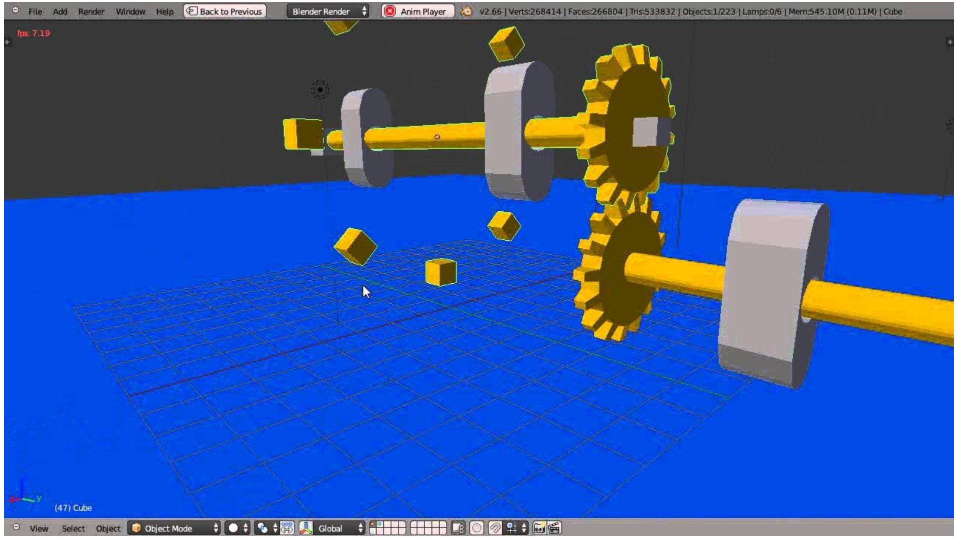


Image courtesy of Sci Fi Animator via Youtube..

Animation

- What is it?
- Cel / Sprite
- Rigid Hierarchical
- **Per-Vertex**
- Skinned

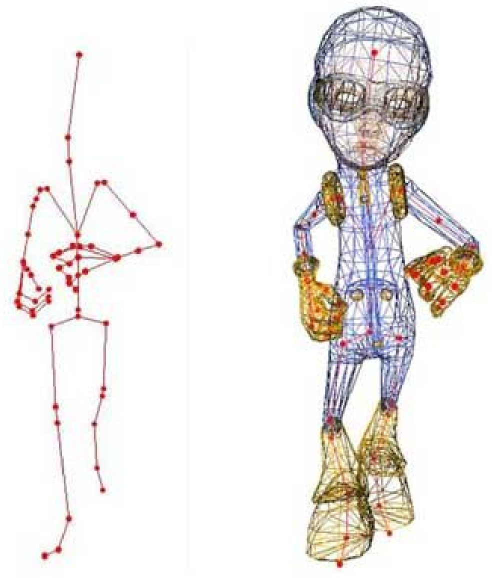
30 frame animation for 3000 verts with 12-byte position per vert per frame == 1.0 MB

Doesn’t scale well.

Image courtesy of DeletionQuality forums...

Animation

- What is it?
- Cel / Sprite
- Rigid Hierarchical
- Per-Vertex
- **Skinned**



- Standard used by film and video game industry.
- Treats vertices as the skin, and animation is defined as transformations of the skeleton.

Image courtesy of waZim via wazim.com..

Skinned Animation

- Also known as **skeletal subspace deformation**.
- A **skeleton** consisting of **joints** is moved in order to drive deformation of a model's vertices, or **skin**.
    - Much like how our skeletal and muscular structure works.
    - Our skin covers the muscle and moves with it.

Skeletons

- Joints
- Hierarchy
- Representation
- Update
- T-pose

Skeletons

- **Joints**
- Hierarchy
- Representation
- Update
- T-pose

- Sometimes referred to as bones.
- Define a coordinate space within a model.

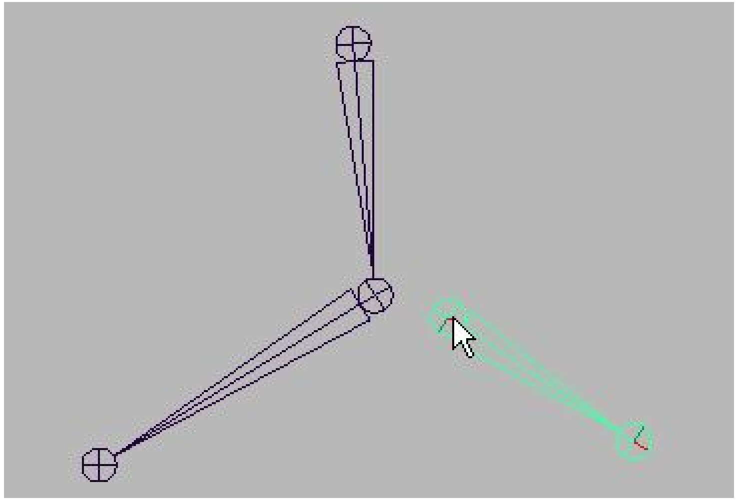


Image courtesy of Autodesk.

Skeletons

- Joints
- **Hierarchy**
- Representation
- Update
- T-pose

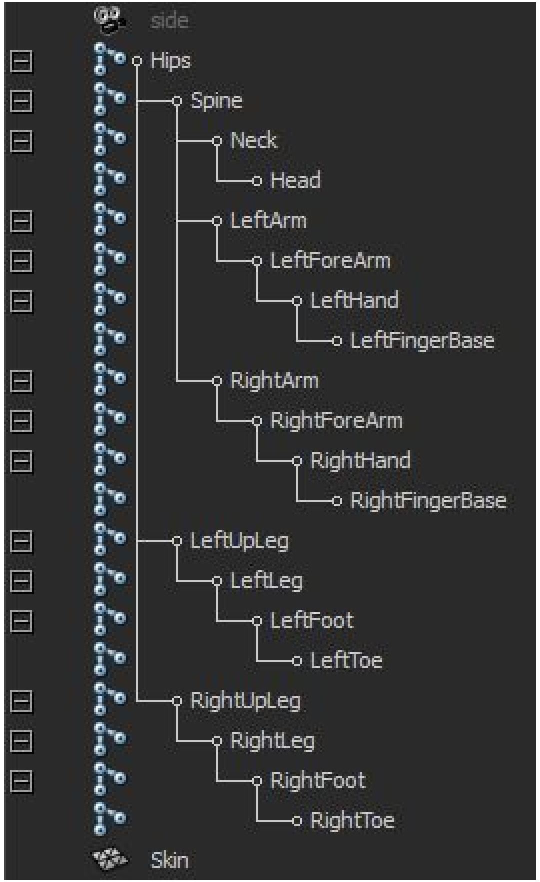


Image courtesy of Carleton University.

Skeletons

- Joints
- Hierarchy
- **Representation**
- Update
- T-pose

```
struct Joint
{
    Matrix4x3 m_invBind;
    const char* m_name;
    u8 m_parent;
};

struct ga_joint
{
     const char _name[32];
     ga_mat4f _world;
     ga_mat4f _inv_bind;
     ga_mat4f _skin;
     uint32_t _parent;
};
```

Skeletons

- Joints
- Hierarchy
- Representation
- **Update**
- T-pose

For each joint in the hierarchy:

- Update the joint’s local transform from animation data.
- Multiply by parent transform to obtain world space transform.
- Multiply by inverse bind matrix to obtain skinning matrix.

That last bit will make more sense soon.

Skeletons

- Joints
- Hierarchy
- Representation
- Update
- **T-pose**

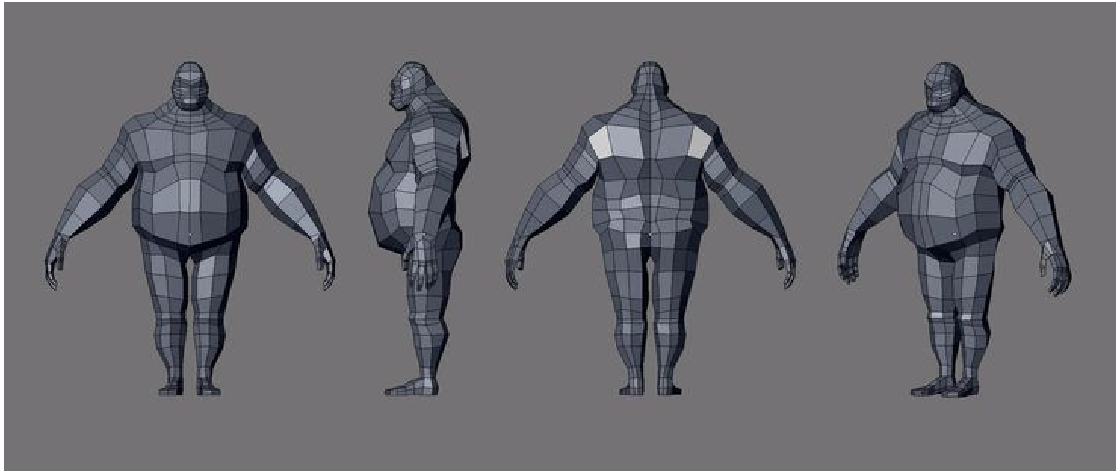


Image courtesy of Pintrerest..

Skinning

- Ok, so how do we get the skin to follow the skeleton?
- Vertices of a model are **bound** to the skeleton.
  - Each vertex has some small number, often four, of joints that influence it.
  - The amount of influence is described by a **weight**, from 0 to 1.
  - The process of assigning joint weights to vertices is called **weight painting**.
- To move the skin, we move each vertex using the influencing joints.

Some Terms...

Before we dive in, you're going to want to know some terms.

- **Binding space** describes positions in the T-pose skeleton.
    - Vertex positions are described in binding space.
- **Pose space** describes positions in animated model.
    - A **pose** is a particular configuration of the skeleton.

And remember…

- **Model space** is the coordinate space relative to the model's origin.
    - Vertex positions are described not only in binding space, but also in model space.

Mathematics

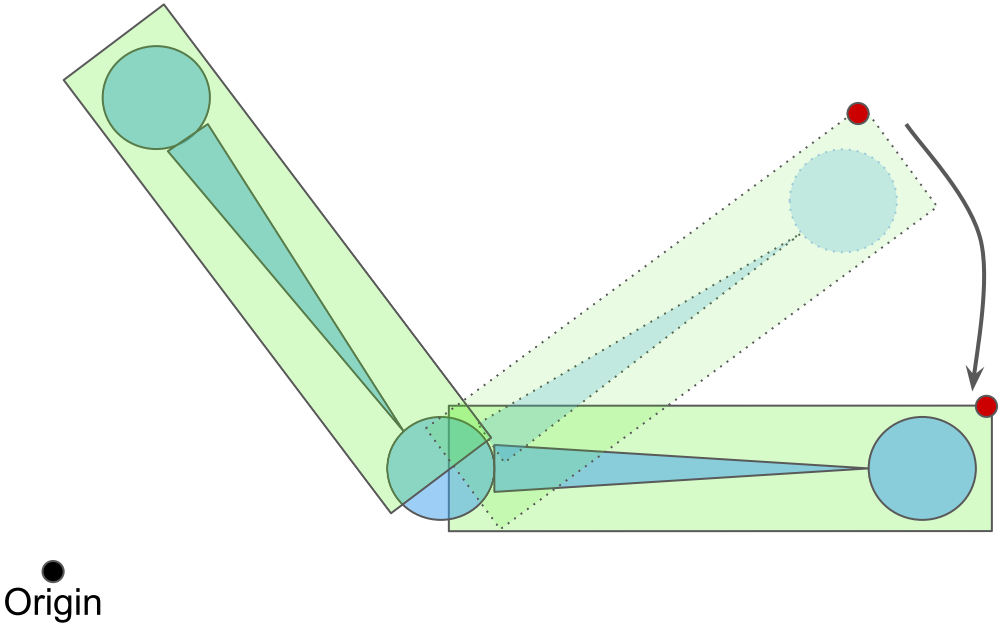
- **Concept**
- Equations
- Algorithm

- We need to transform a vertex from binding space to pose space.
- Vertex position is always constant relative to joint influencing it.
- Transform to local joint space, pose, then transform back to model space.

Mathematics

- **Concept**
- Equations
- Algorithm

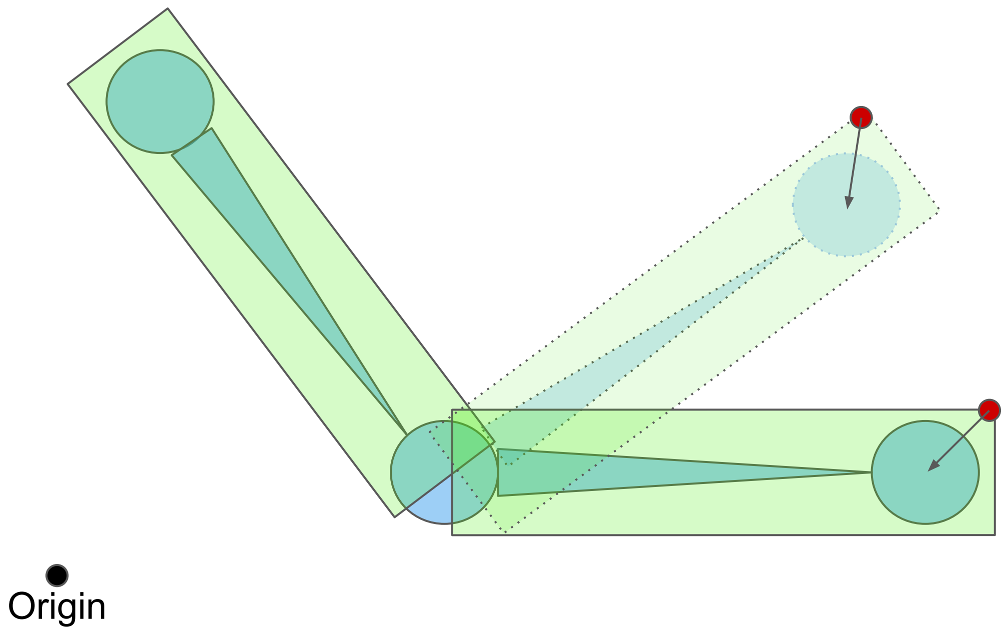
- **We need to transform a vertex from binding space to pose space.**
- Vertex position is always constant relative to joint influencing it.
- Transform to local joint space, pose, then transform back to model space.

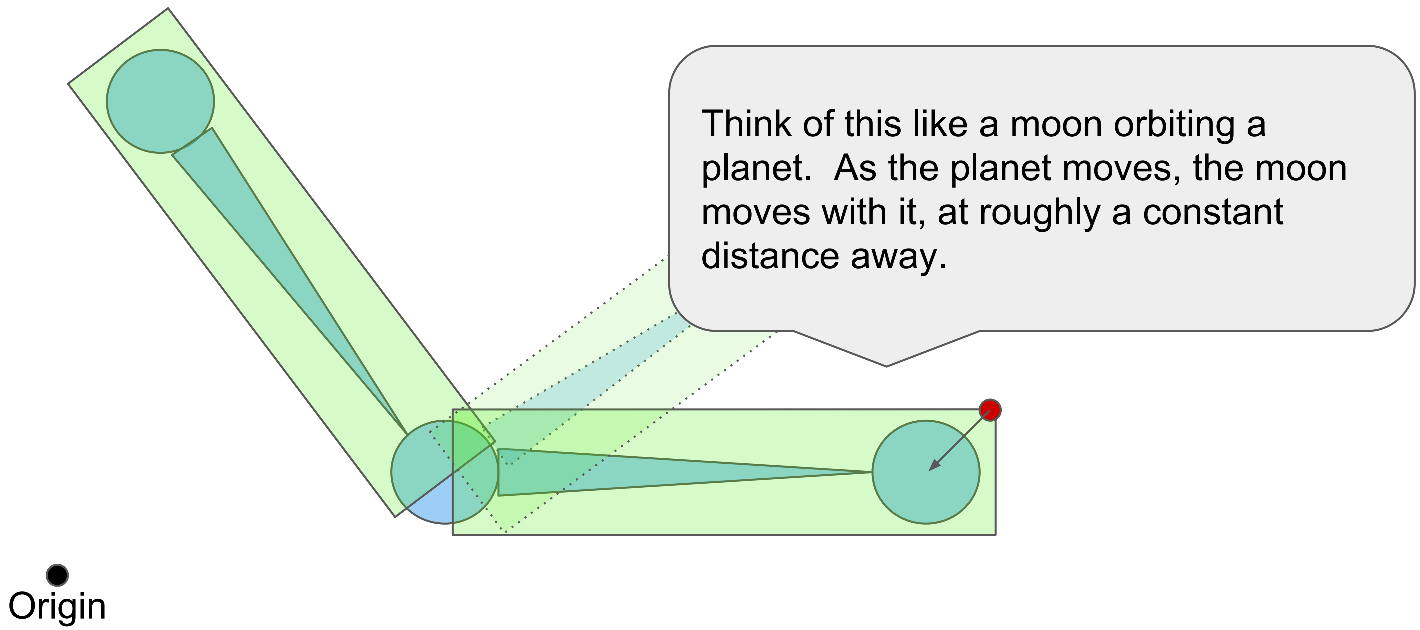

Origin

Mathematics

- **Concept**
- Equations
- Algorithm

- We need to transform a vertex from binding space to pose space.
- **Vertex position is always constant relative to joint influencing it.**
- Transform to local joint space, transform into the pose, then transform back to model space.

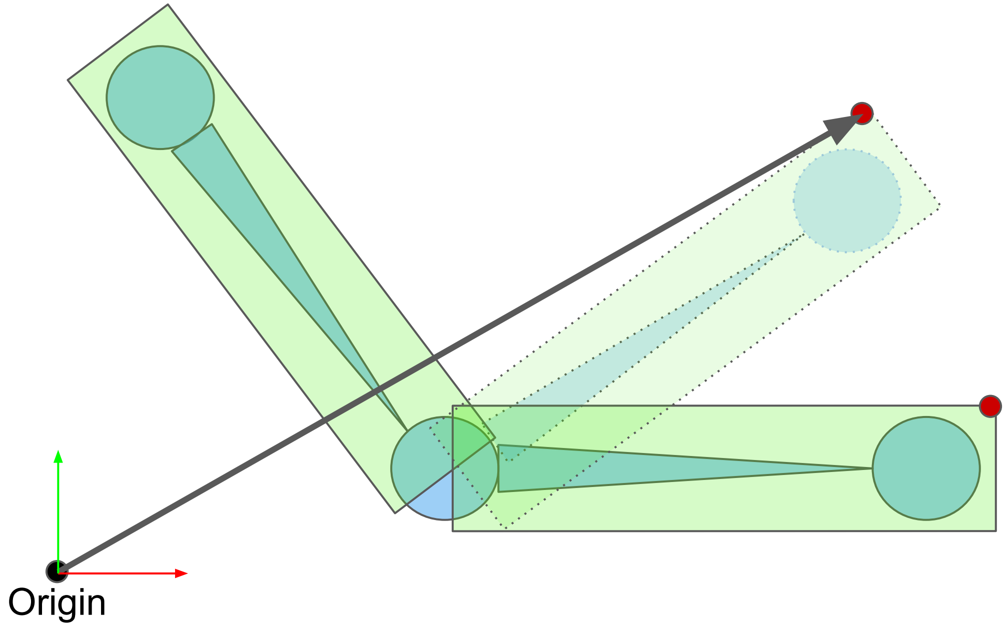

Origin


Think of this like a moon orbiting a planet. As the planet moves, the moon moves with it, at roughly a constant distance away.
Origin

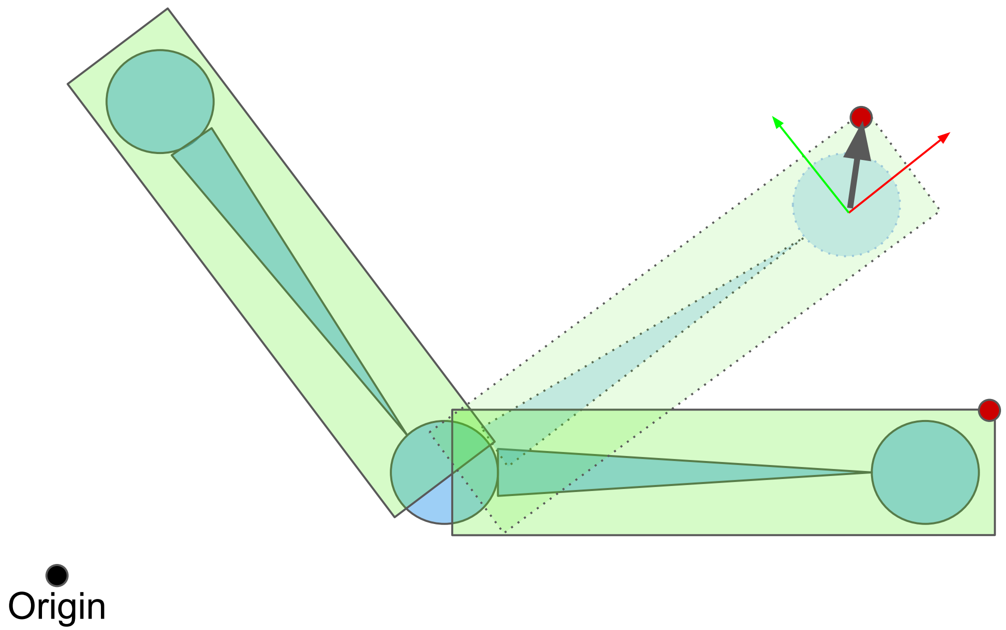
Mathematics

- **Concept**
- Equations
- Algorithm

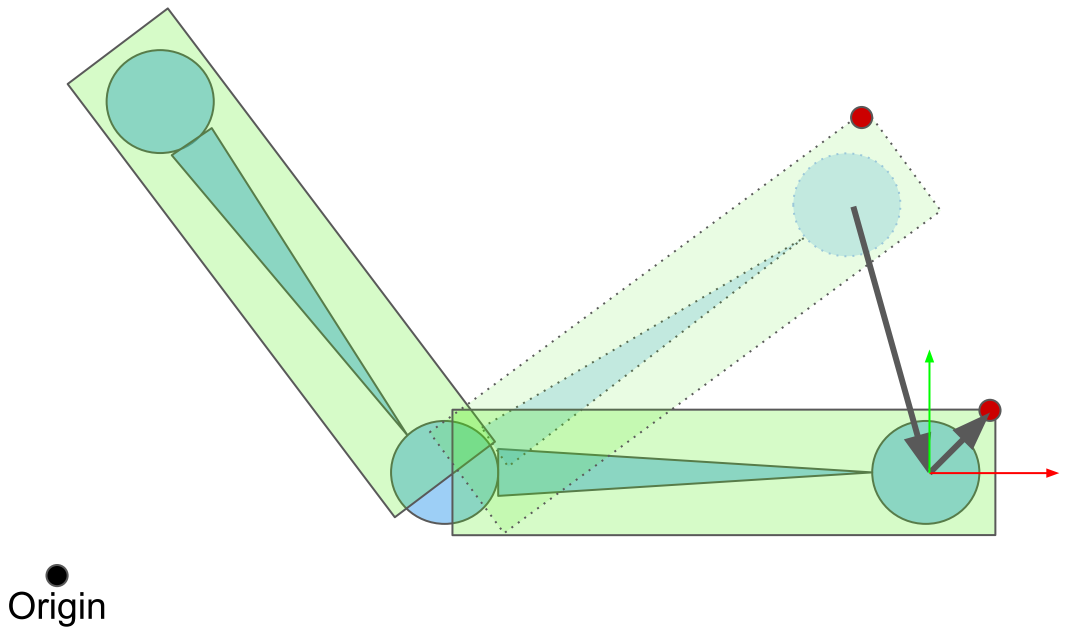
- We need to transform a vertex from binding space to pose space.
- Vertex position is always constant relative to joint influencing it.
- **Transform to local joint space, pose, then transform back to model space.**



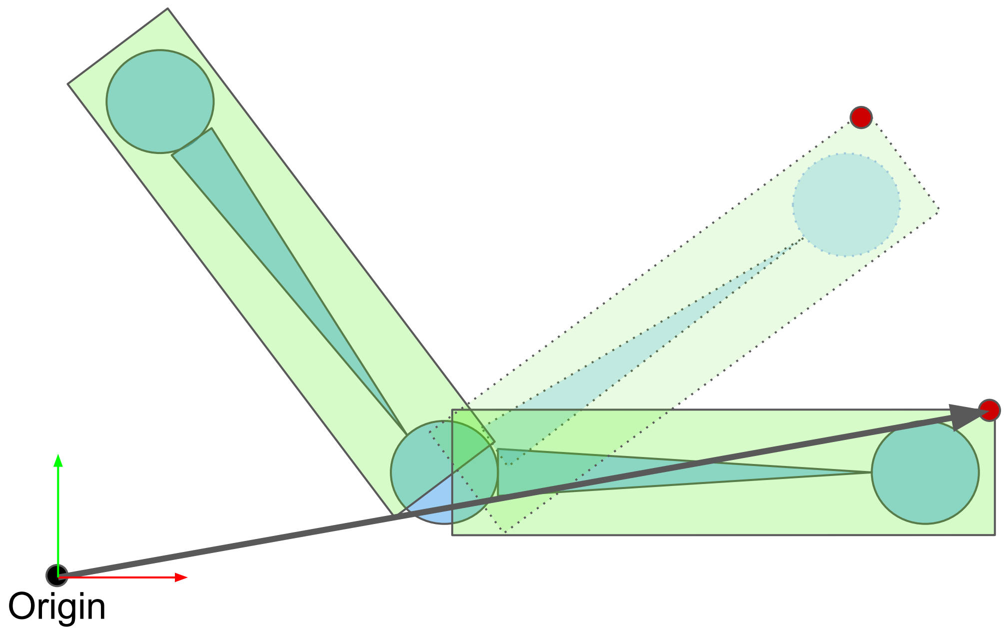
Model (binding) space.



Joint space.



Pose space.



Model (pose) space.

The Binding Matrix

The **binding matrix** describes a joint's transform in **model space**.

- Or in other words, the binding matrix will transform a point relative to a joint into model space.

Which means…

- The **inverse binding matrix** will transform a point in model space to a point relative to a joint.

Mathematics

- Concept
- **Equations**
- Algorithm

First obtain a vertex's position relative to a joint:

- Given $\mathbf{B}_{j \to M}$, the bind matrix of joint *j*, and vertex **v** in the bind position (denoted by superscript *B*), multiply **v** by the inverse binding matrix:

$$\mathbf{v}_j = \mathbf{v}_M^B \mathbf{B}_{M \to j} = \mathbf{v}_M^B (\mathbf{B}_{j \to M})^{-1}$$

Mathematics

- Concept
- **Equations**
- Algorithm

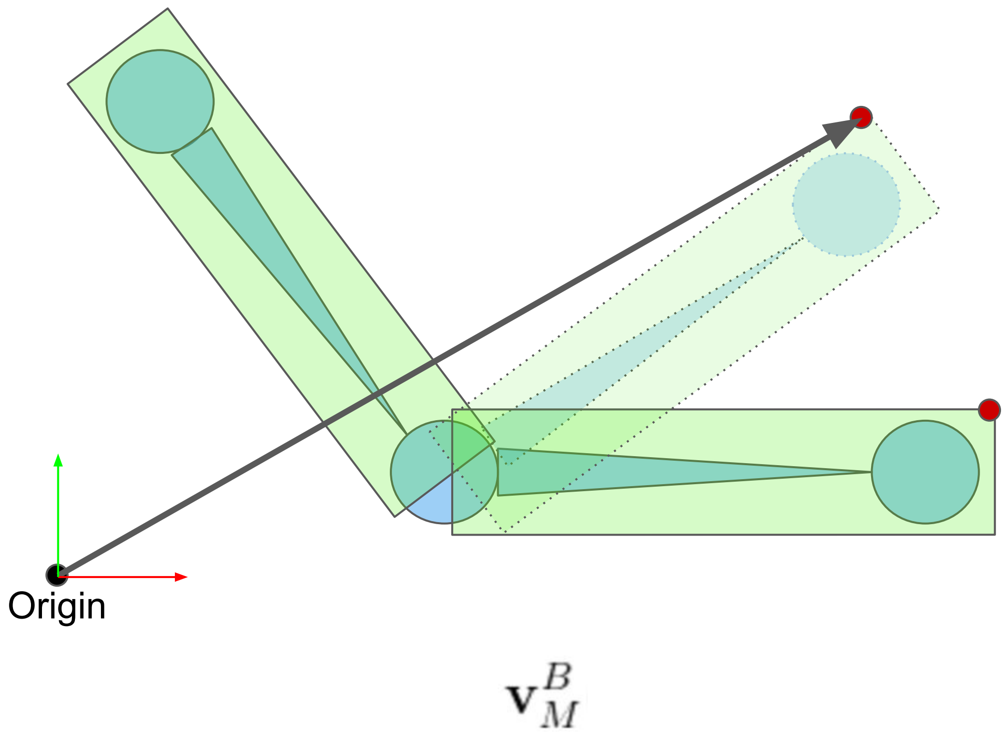
Given $\mathbf{C}_{j\to M}$, the current pose of joint *j*, we can convert **v** into the current pose by multiplying it by **C**:

$$\mathbf{v}_M^C = \mathbf{v}_j \mathbf{C}_{j\to M}$$

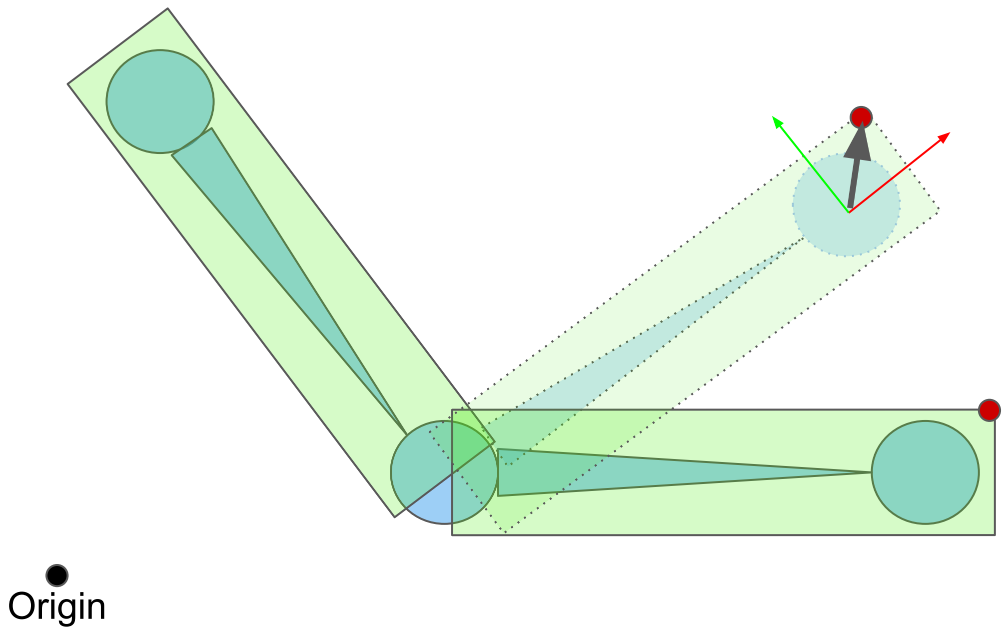
So…

$$\mathbf{v}_M^C = \mathbf{v}_j \mathbf{C}_{j\to M} = \mathbf{v}_M^B (\mathbf{B}_{j\to M})^{-1} \mathbf{C}_{j\to M} = \mathbf{v}_M^B \mathbf{K}_j$$

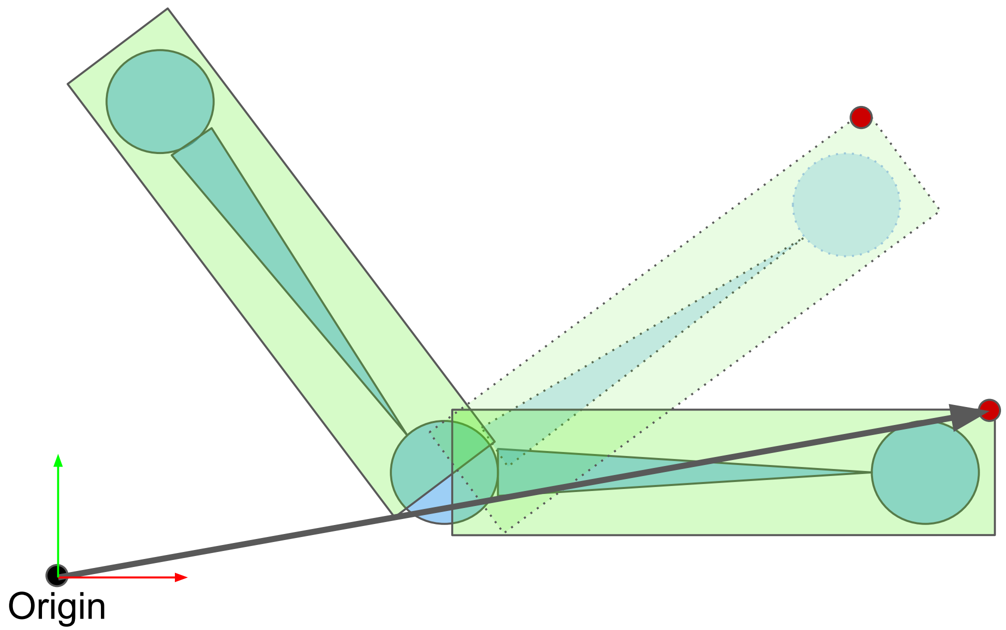
Where $\mathbf{K}_j$ is known as the skinning matrix.



Origin
$\mathbf{v}_M^B$



$$\mathbf{v}_M^B(\mathbf{B}_{j\to M})^{-1}$$



$$\mathbf{v}_M^B(\mathbf{B}_{j\to M})^{-1}\mathbf{C}_{j\to M} = \mathbf{v}_M^B\mathbf{K}_j$$

Mathematics

- Concept
- **Equations**
- Algorithm

For multiple joints, average the positions of the vertex skinned by each influencing joint:

$$\mathbf{v}_M^C = \sum_{i=0}^{N-1} w_i \mathbf{v}_M^B \mathbf{K}_i$$

Where w_i is the weight of joint i.

Mathematics

- Concept
- Equations
- **Algorithm**

Commonly, skinned animation is done by:

- Updating skeletons on the CPU.
- Pushing joint data to the GPU.
- Performing skinning calculations in the vertex shader.

The skinned vertex position is then multiplied by the model-view-projection matrix to obtain its final clip space coordinates.

Animation Clips

- A game is driven by user input, and is therefore unpredictable.
- Instead of lengthy animated sequences, short animation clips are made ready to be played depending on the player’s input.
- Examples might include…
    - Walk
    - Run
    - Jump
    - Land

Animation Clips

- **Timelines**
- Creation
- Playback
- Interpolation

Image courtesy of animationmethods.wordpress.com.

Animation Clips

- Timelines
- **Creation**
- Playback
- Interpolation

- **Keyframes** are points on the timeline where a model’s pose is recorded.
- Poses are interpolated to fill the time between keyframes.
- The more keyframes, the more control being exerted over the animation.

Animation Clips

- Timelines
- Creation
- **Playback**
- Interpolation

There are three common types of playback:

- One time
- Looping
- Ping-pong

A playing animation can be considered to have its own local timeline.

- t = 0 is the beginning, t = n is the end where n is the animation length.
- t can be scaled for slow or fast motion

Animation Clips

- Timelines
- Creation
- Playback
- **Interpolation**

A game doesn’t run at the exact rate an animation is defined, and an animation is not keyframed at regular intervals.

- Thus, we need to interpolate joint transforms between keyframes based on *t*.

Best method depends on transform:

- Lerp for translation.
- Slerp for orientation.

Blending

- To create believable movement, multiple animations can be played simultaneously and **blended** together.
- For example, a character may be transitioning from walking to running.
    - In order to avoid a very visible switch from a walk cycle to a run cycle, the game may execute a smooth blend between the two.

Blending

- The smooth transition from one animation to another is known as **cross-fading**.
    - Over a period of time *t*, animation A starts at 100 percent influence and ends at 0 percent, and animation B starts at 0 and ends at 100.

Blending

- Another common type of blending is known as **additive blending**.
- In this scenario, the motion from one animation is added on top of another.
    - For example, a character may be running, and at the same time a “look at” animation is added on top to make them look at a target.
    - Or, a character may crouch and a “pick up” animation may be blended over it.
- Care must be taken to blend animations that do not have heavy contention over joints.
- **Animation layers** offer a way for animations to be restricted to portions of a skeleton.
    - For example, one layer may target legs, while another targets torso.
    - Clever use of animation layers can lead to very complex, believable motion.

Blending

- Mathematically, blending can be achieved by lerping between two target transforms, similar to lerping between frames within an animation.
- So, given pose $\mathbf{M}_{Ai}$ for joint *i* in animation A, and pose $\mathbf{M}_{Bi}$ for joint *i* in animation B:

$$\mathbf{M}_{ABi} = LERP(\mathbf{M}_{Ai}, \mathbf{M}_{Bi})$$

State Machines

- At the highest level, it's common to have an **animation state machine**.
    - A finite state machine controlling animation state.
- Responsible for smooth transitions between states.
- Individual states may be a single animation clip or a collection of blended clips.
- Serves as a simple interface for external systems to control character motion.

Want to learn more?

- Dual Quaternion skinning:
  - https://www.cs.utah.edu/~ladislav/kavan08geometric/kavan08geometric.pdf
- Unity’s Animation manual:
  - https://docs.unity3d.com/Manual/AnimationSection.html
- Keep an eye out for this talk:
  - http://schedule.gdconf.com/session/huddle-up-making-the-spoiler-of-inside

End of Lecture

- Homework 5 is due Thursday, 3/9, at 11:59pm eastern.
- Homework 6 is due Monday, 3/20, at 11:59pm eastern.