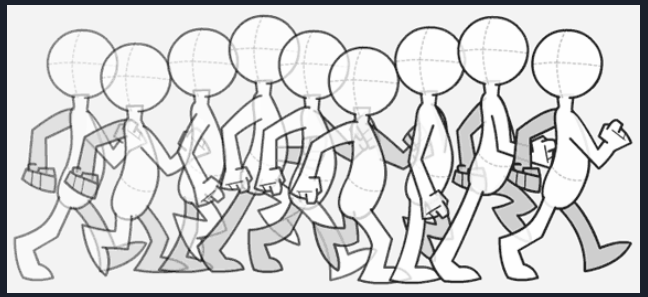
# Animation

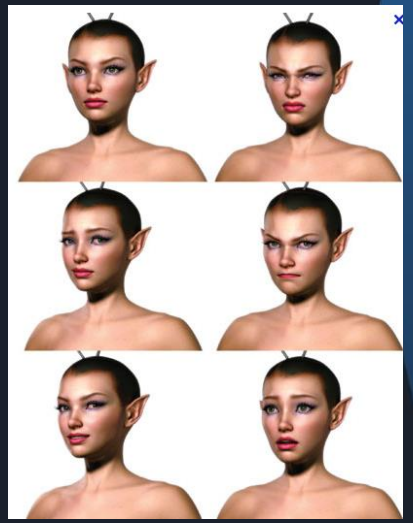
2-D and 3-D animation techniques

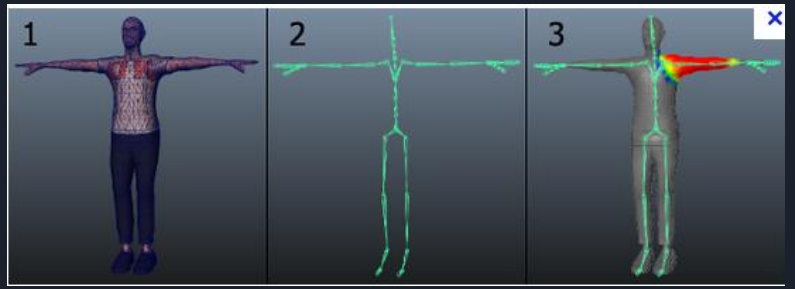
Programming – Computer Graphics

# Contents

- 2-D Animation
- Texture Animation
- Morphing
- Skeletal Animation





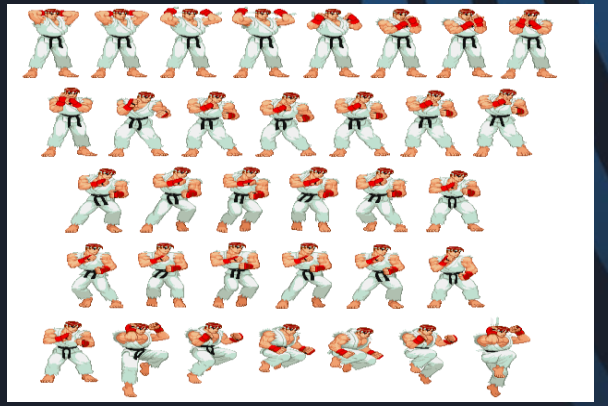


# Introduction

- Animation adds life to your games
- The Human brain is very good at detecting movement and analysing it
  - Minor imperfections in animation are very obvious
- Excellent animation data on poor models looks much better than poor animation data on excellent models
- Look at the credits of any Pixar movie to see how much importance they place on animation

# Traditional 2D Animation

- Animation data consists of a sequence of pre-drawn (or rendered) images called Key-Frames
  - Key Frames represent the state of an animated object at a specified time / frame



- Playing of animation is simply a matter of playing the correct frames in sequence at the correct frame rate

# UV Animation

- It is good to now mention a rather simple shader technique that can be used to achieve great results
  - Animated texture coordinates, commonly called UV Animation
- UV Animation is when texture coordinates are animated over time
  - The vertex data itself never changes, but rather time is used as a shader uniform variable to modify the texture coordinate
  - UV is the traditional name for 2-dimensional texture coordinates, XY or ST
- This causes the texture to either scroll across the surface or reference a different section of the texture
  - Like a 2D sprite animation using a sprite sheet
  - A water texture scrolling across a plane of water

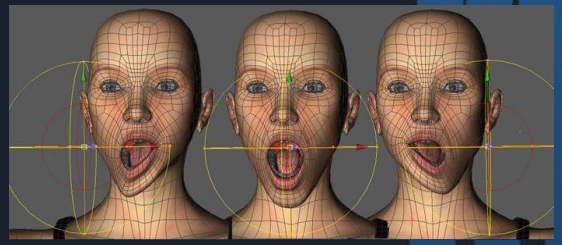
# UV Animation

- UV animation might not be advanced, but it can be a big performance saver!
- This technique can be used for many things:
  - 2D Sprite animation can be controlled with a single offset variable and a variable defining the size of a single frame of animation
  - Rippling flowing water or waterfalls
  - Particle FX such as lightning, fire, explosions, lasers, etc
  - Glowing lava
  - Lighting effects for moving spot lights
  - Full-screen effects like film grain
- Almost any technique where the size of the object doesn't change but its texture does could be set up as a UV animation

# UV Animation Examples




# 3D Animation Techniques

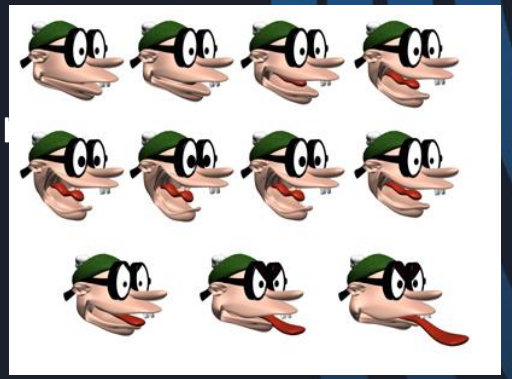


- Morph targets
  - Often used for facial expressions
  - Similar to traditional animation with a series of key frames, except that we can interpolate between key frames rather than use one specific frame
  - Very precise control of model, but data can be very large

- Skinning
  - The system used for most modern game animation
  - Model is rigged to a skeleton of bones
  - The skeleton is animated with key frames and the mesh wraps around the skeleton based off weighted values
  - Less precise control of model but data is much smaller

# Morphing aka Vertex Animation

- Morphing is the process of animating vertices by interpolating each vertex from one key frame to another vertex in another key frame
- Each key frame contains an entire set of every vertex within the mesh at their position for that key frame
  - Can be memory intensive for a model like a character with many vertices and many animations
- This form of animation was used in games like Quake, before Skinning was in common use



# Morphing Uses

- Morphing is useful in situations where a skeleton would be complicated to achieve fine detailed animation

- Can be combined with skinning to add an extra level of animation detail
  - For instance, a skinned character with a morphable face for detailed facial animations during cutscenes

# Other Morphing Uses

- Morphing can also be used on items such as cloth and flags
  - Flags can be given a far more detailed animation that has a greater level of realism than skeletal animation
  - Soft-body physics is increasingly used for these types of application

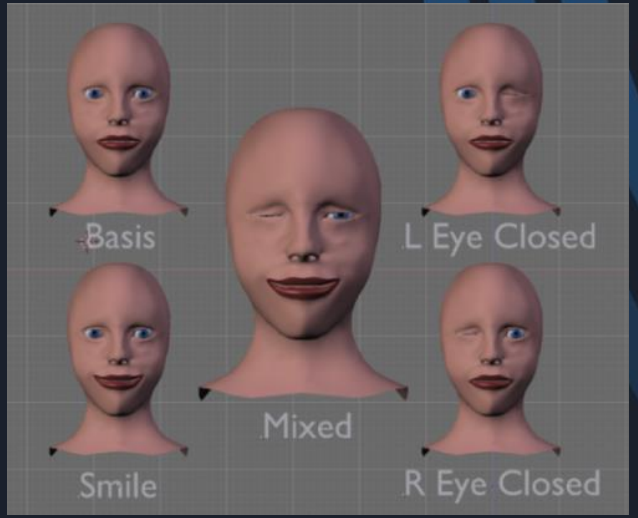
# Morphing Implementation

- Morphing requires a duplicate of the vertex data at each key frame
  - Positions, texture coordinates, normals, etc can all be animated
  - Only elements which change are needed
  - We find the two key frames that fall on either side of our current time within the animation
  - Interpolate the vertex data between the key frames using either CPU or GPU (preferred)

- To process morphing on the GPU you will need to send across multiple vertex buffers

# GPU Morphing Implementation

- When rendering an object we aren't limited to a single vertex buffer
  - Meshes can be composed of multiple vertex buffer streams
  - Each stream can have more or less data, whatever is relevant
- We send the streams for the two current key frames then interpolate using a time interval between them
- The Vertex program uses linear interpolation on each pair of vertices



# Non-Keyed Morphing

- Sometimes we might want to procedurally morph a mesh
  - Grass or trees blowing in the wind
  - Ocean waves rippling

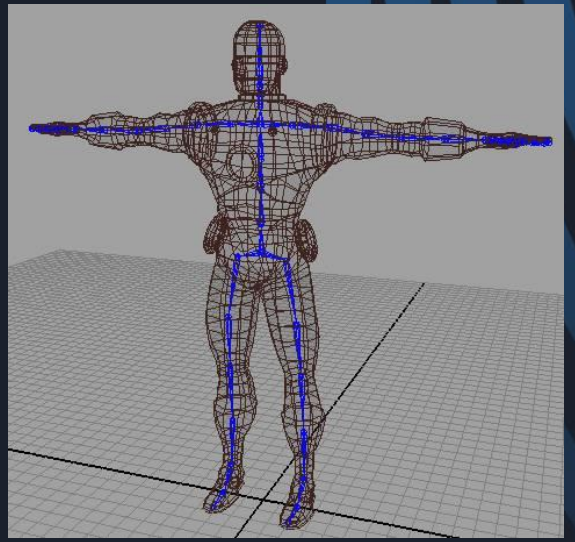
- In these cases artists don't need to create a buffer of vertex data, we can just animate it on the GPU in a vertex Program
  - This is where Sine and Cosine can come in handy

# Grass Example

- This can usually be achieved with a wind strength shader constant, that can be sent from the CPU and changed based on wind direction and speed etc
  - We also need to know how tall the grass is, or have this information baked into the vertex itself
- Then we apply the offset to the grass vertices, applying full strength at the top and zero strength at the bottom

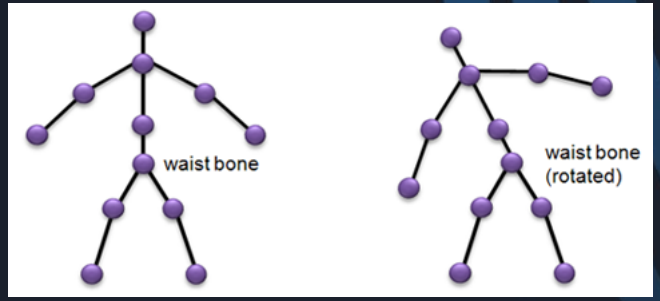
# Skinning aka Skeletal Animation

- Skeletal Animation is when we animate the vertices of a model based on weighted transforms
  - These transforms are commonly called Bones
- These bones are placed throughout a model and the vertices around it are influenced by any movement made by them
- The vertices that make up a mesh are known as the Skin
- Key frames contain movement of the bones, not the vertices, so less information is needed to be stored
  - Usually less than 100 bones, compared to thousands of vertices



# Skeletons

- A skeleton is a collection of transforms called bones
- Typically these bones are arranged in a hierarchy
  - If one bone moves then the bones attached to it also move, but don't require their own key frames to represent the movement
- Not all bones need to have key frames for particular animations
  - An animation of a head nodding doesn't require the legs to have key frames
  - This results in far less memory being required for animations



# Bones

- Bone transforms are concatenated with their parent bone to calculate their actual position
  - For example, the nod animation would rotate the neck bone but not the head bone, but concatenating the matrices would move the head accordingly
- Transforms are typically a Matrix
  - 4x4 Matrix storing rotation, scale and translation all in one
  - But as animation is stored as key frames we need to be able to interpolate between key frames, and with a Matrix we can't properly interpolate
- Key frame transforms are usually stored with separate properties
  - Position, Rotation and Scale are each separate
- Position and Scale can both be represented by 3 floats each
  - X, Y and Z, and X-Scale, Y-Scale and Z-Scale
- Rotation is stored as a Quaternion so that it can be interpolated properly

# Bones

- Bone Key Frames can then be interpolated easily to calculate the current transform for the bone
  - Using a interpolation value between 0.0 and 1.0 that represents the blend between Key Frame 0 and Key Frame 1…
    - Key Frame 0's position is interpolated with Key Frame 1's position
    - Key Frame 0's scale is interpolated with Key Frame 1's scale
    - Key Frame 0's rotation is interpolated with Key Frame 1's rotation
- We can then combine the 3 interpolated values into a single matrix to represent the bone's current transform that frame
  - Convert rotation Quaternion to a Matrix, scale the matrix and then translate it
  - We do this for every bone!

# Skinning

- Now that we have updated the skeleton bones and have their transforms we still need to skin the skeleton

- All Vertices have 2 additional properties
  - Bone Indices to specify which bones influence the vertex's movement
  - Blend Weights to specify how much each bone influences it as a percentage

- It is common for a Vertex to be influenced by 2-4 bones
  - The combined Blend Weights must add up to 100% and are typically values between 0.0 and 1.0, and thus must total 1.0
  - A Vertex might be heavily influenced by two bones (both with weights of 0.4) but slightly influenced by two more (both at 0.1)

# Skinning

- Before transforming a Vertex by its mesh's World Transform and the View / Projection Transforms we first transform it by the bones influencing
  - We can do this on the GPU by sending an array of matrices representing the bones as a uniform array of mat4 and by giving each vertex in the vertex buffer indices and weights

```
// sample skinning Vertex Shader
#version 410

in vec4 position;
in vec4 indices;   // allows us up to 4 bone influences
in vec4 weights;   // we can have less than 4 by using weights of 0

uniform mat4 projectionViewWorldMatrix;

const int BONE_COUNT = 92; // good to limit this
uniform mat4 bones[BONE_COUNT];

void main() {

     // need to convert indices to int from float
     ivec4 I = ivec4(indices);

     vec4 P = bones[ I.x ] * position * weights.x;
     P += bones[ I.y ] * position * weights.y;
     P += bones[ I.z ] * position * weights.z;
     P += bones[ I.w ] * position * weights.w;
     P.w = 1;

     gl_Position = projectionWiewWorldMatrix * P;
}
```

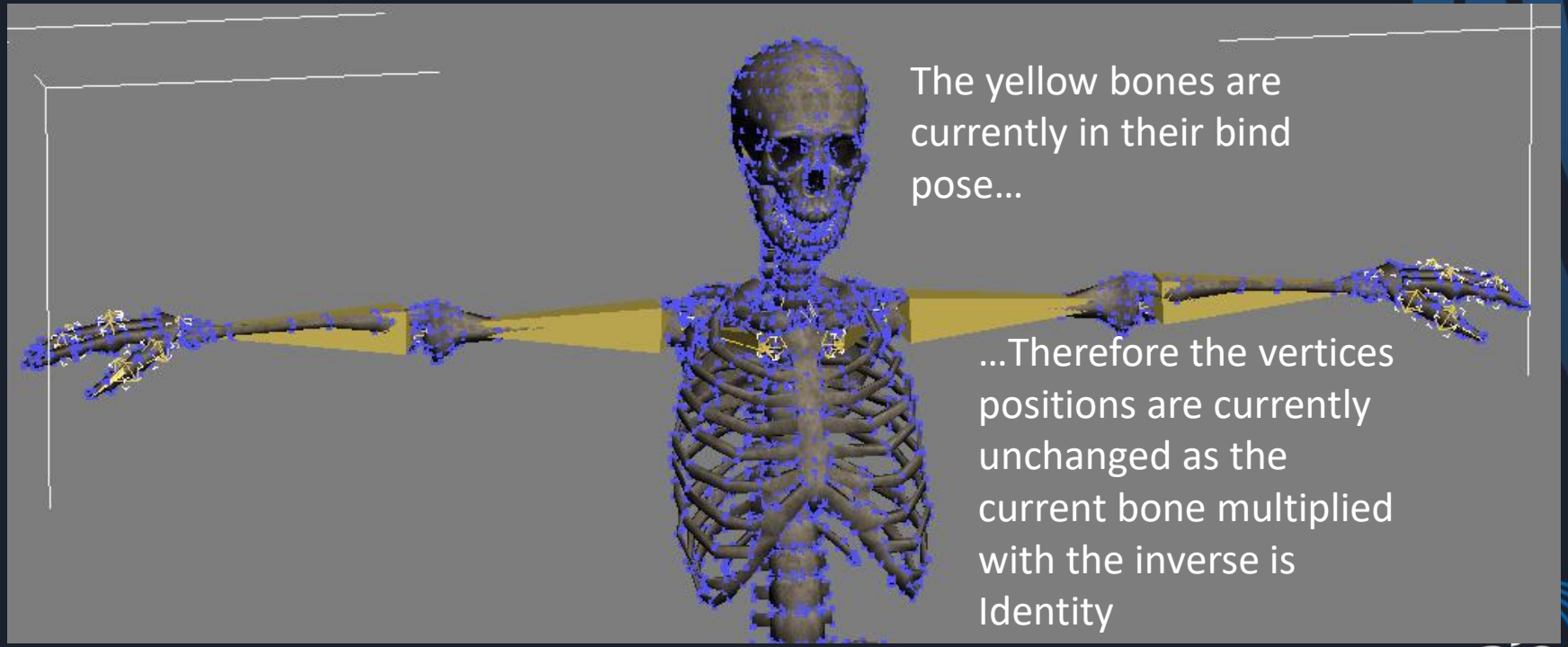
# Bind Pose

- There is just one step that we must do before we send the bones to the GPU
- We need the bones that we send to the GPU to be offsets from their initial starting location
  - This initial set of transforms is called the Bind Pose
- If they weren't offsets then the skin would be transformed incorrectly for some rather hilarious yet extremely broken results

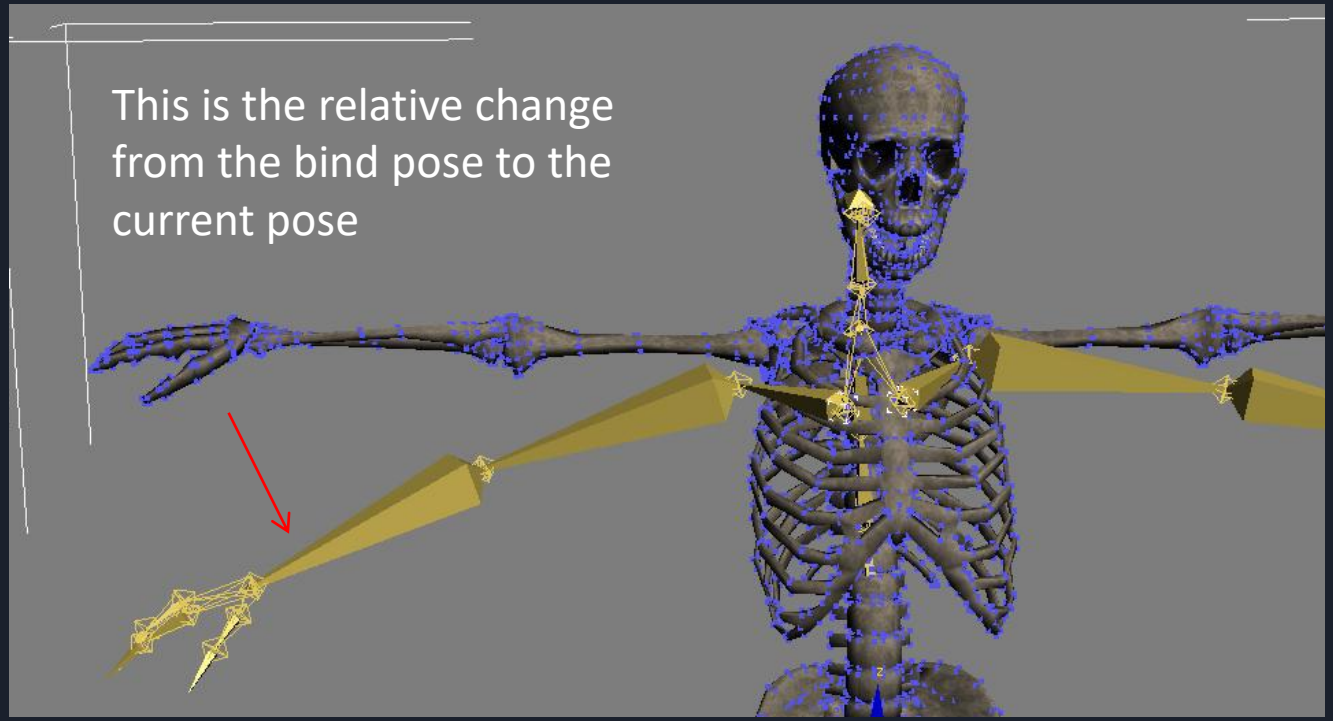
# Bind Pose

- To calculate the correct bone offsets we need the inverse transform of the Bind Pose
  - The Inverse being the reverse of the transforms
  - The Inverse multiplied with the Bind Pose results in the Identity matrix, but the Inverse multiplied against a bone that has moved results in just the offset movement from the Bind Pose
- We can use this inverse in two ways
  - Multiply the current bone transform by the Inverse Bind Pose to calculate the offset
  - Multiple the Vertex against the Inverse Bind Pose bones that effect it based off the Blend Weights
- The first way is often preferred as the mesh is still able to be correctly rendered without animation

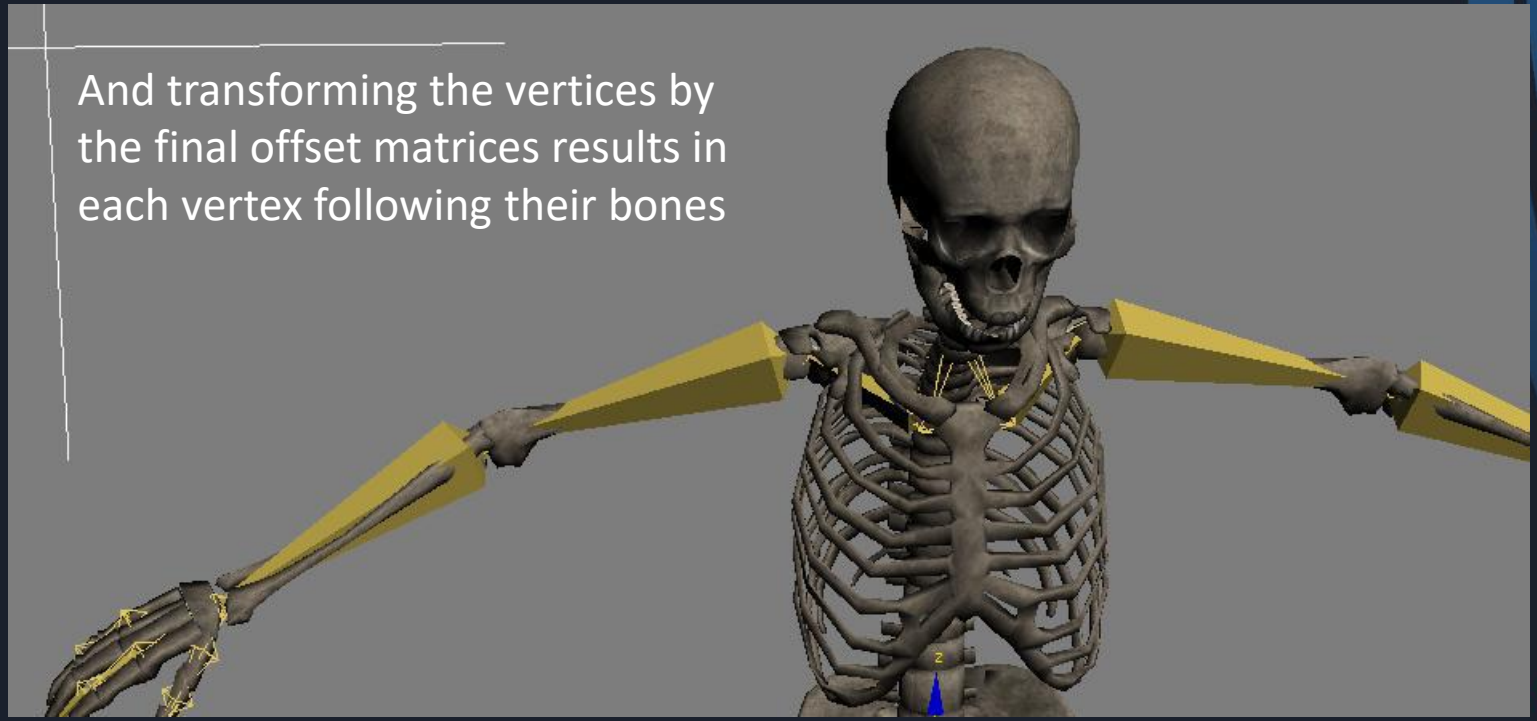
# Transformations



# Transformations



# And Finally…



# Advantages of Skinning over Morphing

- The same animation data can be played on multiple meshes
- Animation data can be acquired using motion capture then played on a range of models
- Animation data can be blended in different ways
  - In series: used for smooth transitions e.g. run to walk
  - In parallel: used for playing multiple animations simultaneously such as a run animation on the lower body and a combat animation on the upper

# Animation Blending

- A bonus of a skeleton hierarchy is that we can setup complex animation systems that allow us to blend animations together, either completely or in segments:
  - Waist-down from the “Run” animation combined with the Waist-up from the “Shoot” animation can give us a “Running-Shoot” animation without an artist having to make a 3rd animation
- We can also blend our own transforms on top of a bone:
  - Procedurally make a walking character look at an item by rotating it’s “head” bone to face the item

# Animation Blending

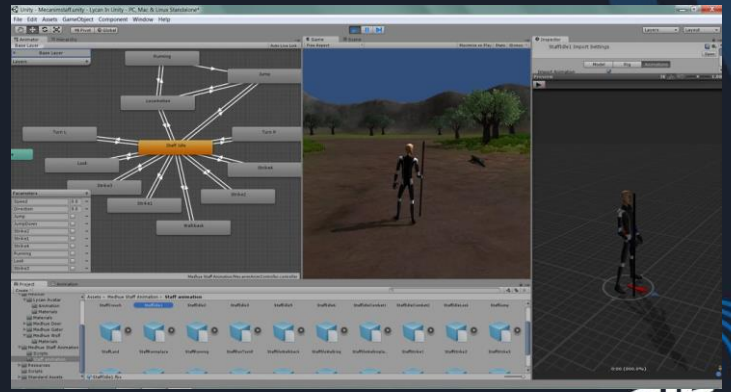
- We can also use skeletons on other models created with a different skeleton:
  - Bones are just arranged in an array, so as long as a skeleton has enough bones for the vertex bone indices it is usable
  - We could use a single set of character animations for all character models in a fighting game for example
  - We could also use a “dog” skeleton on a “horse” model for humorous effects

Warhammer Online’s animation system allowed skeletons to be applied to any model, including using an Orc’s skeleton for a wolf mount

# Animations in Engines

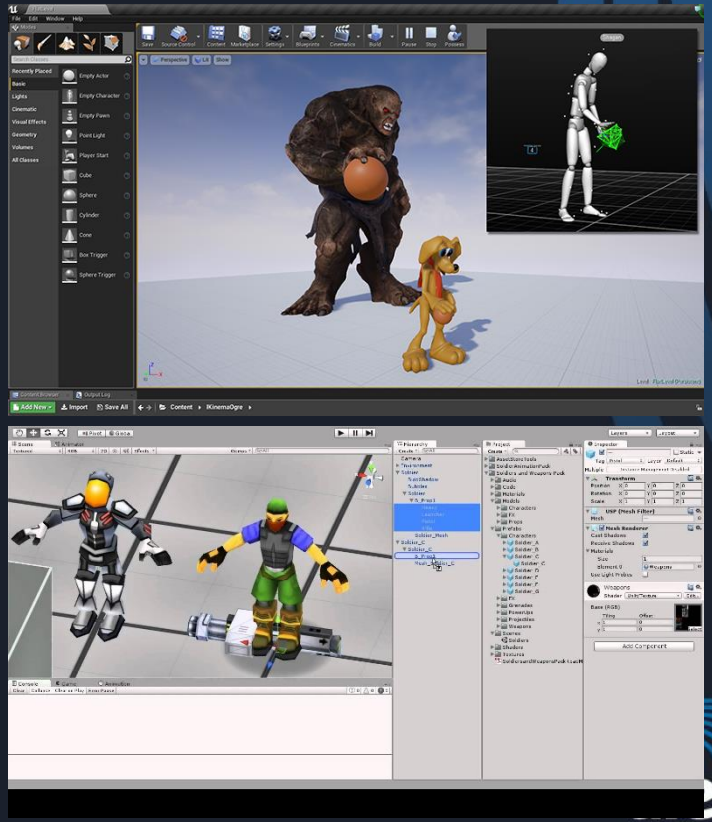
- All game engines have animation systems built-in
  - Shaders and update systems already taken care of for us
  - State graphs and trees control how animations blend between each other

- So if engines implement it all already, what is there for us to do?
  - Tie the gameplay to the graphs and trees and trigger blends
  - Add dynamic animation, such as head looks



# Animations in Engines

- Most engines also have tools for animation retargeting
  - Apply one mesh's skeleton to a different mesh so that they can reuse animations
- They also have tools to edit animation curves
  - Controlling how certain animatable properties interpolate



# Summary

- There are multiple methods for animation in 3D
  - Morphing acts on vertices blending with other vertices
  - Skinning treats the vertices as a “skin” covering a skeleton that blends between key frames
- To correctly interpolate skeleton bones we use Quaternions
  - Matrices can’t be easily interpolated because the rotation would not remain unit-length
  - Quaternions can be easily converted to Matrices, and vice-versa
- Skeletal animation is the primary form of animation in modern computer graphics

# Further Reading


- Akenine-Möller, T, Haines, E, 2008, *Real-Time Rendering*, 3rd Edition, A.K. Peters

- Gregory, J, 2014, *Game Engine Architecture*, 2nd Edition, CRC Press