Chapter 9
Animation System
Humorous Phases of Funny Faces

©April 6, 1906
The Vitagraph Co. of America
In 1972 Ed Catmull and Fred Parke created the world's first 3D rendered movie, an animated version of Ed's left hand.

This clip was eventually discovered by a Hollywood producer and incorporated into the 1976 movie Futureworld.
9.1 Types of Character Animation
Cel animation is a specific type of traditional animation. A cel is a transparent sheet of plastic on which images can be painted or drawn.

An animated sequence of cels can be placed on top of a fixed background painting or drawing to produce the illusion of motion without having to redraw the static background over and over.

Figure 11.1. The sequence of sprite bitmaps used in most Intellivision games.
The earliest approach to 3D character animation is a technique known as rigid hierarchical animation. In this approach, a character is modeled as a collection of rigid pieces.

A typical break-down for a humanoid character might be pelvis, torso, upper arms, lower arms, upper legs, lower legs, hands, feet, and head.

The rigid pieces are constrained to one another in a hierarchical fashion, analogous to the manner in which a mammal’s bones are connected at the joints.
Rigid Hierarchical Animation

Pelvis
Torso
  UpperRightArm
  LowerRightArm
  RightHand
  UpperLeftArm
  UpperLeftArm
  LeftHand
Head
UpperRightLeg
LowerRightLeg
RightFoot
UpperLeftLeg
UpperLeftLeg
LeftFoot
Per-vertex animation. In this approach, the vertices of the mesh are animated by an artist, and motion data is exported which tells the game engine how to move each vertex at runtime.

This technique can produce any mesh deformation imaginable (limited only by the tessellation of the surface). However, it is a data-intensive technique, since time-varying motion information must be stored for each vertex of the mesh. For this reason, it has little application to real-time games.
Per-Vertex Animation and Morph Targets
A variation on this technique known as **morph target animation** is used in some real-time games. In this approach, the vertices of a mesh are moved by an animator to create a relatively small set of fixed, extreme poses.

Animations are produced by blending between two or more of these fixed poses at runtime.

The position of each vertex is calculated using a simple linear interpolation (LERP) between the vertex’s positions in each of the extreme poses.
Per-Vertex Animation and Morph Targets
As the capabilities of game hardware improved further, an animation technology known as skinned animation was developed.

This technique has many of the benefits of per-vertex and morph target animation—permitting the triangles of an animated mesh to deform. But it also enjoys the much more-efficient performance and memory usage characteristics of rigid hierarchical animation. It is capable of producing reasonably realistic approximations to the movement of skin and clothing.
Skinned Animation
In skinned animation, a skeleton is constructed from rigid “bones,” just as in rigid hierarchical animation. However, instead of rendering the rigid pieces on-screen, they remain hidden.

A smooth continuous triangle mesh called a skin is bound to the joints of the skeleton; its vertices track the movements of the joints.

Each vertex of the skin mesh can be weighted to multiple joints, so the skin can stretch in a natural way as the joints move.
When considering the trade-offs between various animation techniques, it can be helpful to think of them as compression methods, analogous in many respects to video compression techniques.

We should generally aim to select the animation method that provides the best compression without producing unacceptable visual artifacts. Skeletal animation provides the best compression when the motion of a single joint is magnified into the motions of many vertices. A character’s limbs act like rigid bodies for the most part, so they can be moved very efficiently with a skeleton.
9.2 Skeletons
A skeleton is comprised of a hierarchy of rigid pieces known as joints. In the game industry, we often use the terms “joint” and “bone” interchangeably, but the term bone is actually a misnomer.

Technically speaking, the joints are the objects that are directly manipulated by the animator, while the bones are simply the empty spaces between the joints.

As an example, consider the pelvis joint in the Crank the Weasel character model. It is a single joint, but because it connects to four other joints (the tail, the spine, and the left and right hip joints), this one joint appears to have four bones sticking out of it.
Figure 11.5. The pelvis joint of this character connects to four other joints (tail, spine, and two legs), and so it produces four bones.
The Skeletal Hierarchy
9.3 Poses
This is the pose of the 3D mesh prior to being bound to the skeleton (hence the name). In other words, it is the pose that the mesh would assume if it were rendered as a regular, unskinned triangle mesh, without any skeleton at all.

The bind pose is also called the T-pose because the character is usually standing with his feet slightly apart and his arms outstretched in the shape of the letter T.
Figure II.6. Two different poses of the same skeleton. The pose on the left is the special pose known as *bind pose*.
Figure 11.7. Every joint in a skeletal hierarchy defines a set of local coordinate space axes, known as joint space.
Figure 11.8. A global pose can be calculated by walking the hierarchy from the joint in question towards the root and model space origin, concatenating the child-to-parent (local) transforms of each joint as we go.
The model-space pose of joint J2 can therefore be written as follows:

\[
P_{2\rightarrow M} = P_{2\rightarrow 1} P_{1\rightarrow 0} P_{0\rightarrow M}
\]

Likewise, the model-space pose of joint J5 is just

\[
P_{5\rightarrow M} = P_{5\rightarrow 4} P_{4\rightarrow 3} P_{3\rightarrow 0} P_{3\rightarrow M}.
\]
9.4 Clips
The Local Time Line

Animation A: Local Time

$t = 0$  \hspace{1cm}  t = (0.4)T  \hspace{1cm}  t = (0.8)T  \hspace{1cm}  t = T$
Figure 11.10. An animator creates a relatively small number of key poses, and the engine fills in the rest of the poses via interpolation.
Time Units

Typical frame durations are 1/30 or 1/60 of a second for game animation. However, it’s important not to make the mistake of defining your time variable $t$ as an integer that counts whole frames.

No matter which time units are selected, $t$ should be a real (floating-point) quantity, a fixed-point number, or an integer that measures subframe time intervals. The goal is to have sufficient resolution in your time measurements for doing things like “tweening” between frames or scaling an animation’s play-back speed.
Unfortunately, the term *frame* has more than one common meaning in the game industry. This can lead to a great deal of confusion. Sometimes a frame is taken to be a *period of time* that is 1/30 or 1/60 of a second in duration.

But in other contexts, the term frame is applied to a *single point in time* (e.g., we might speak of the pose of the character “at frame 42”).

![Figure 11.11. A one-second animation sampled at 30 frames per second is 30 frames in duration and consists of 31 samples.](image-url)
Frames, Samples and Looping Clips

- If a clip is *non-looping*, an $N$-frame animation will have $N + 1$ unique samples.
- If a clip is *looping*, then the last sample is redundant, so an $N$-frame animation will have $N$ unique samples.

Figure 11.12. The last sample of a looping clip coincides in time with its first sample and is, therefore, redundant.
It is sometimes convenient to employ a normalized time unit $u$, such that $u = 0$ at the start of the animation, and $u = 1$ at the end, no matter what its duration $T$ may be.

We sometimes refer to normalized time as the *phase* of the animation clip, because $u$ acts like the phase of a sine wave when the animation is looped.

![Normalized Local Time Diagram](image)

*Figure 11.13. An animation clip, showing normalized time units.*
Figure 11.14. Playing animation clip A starting at a global time of 102 seconds.
As we saw above, playing a looping animation is like laying down an infinite number of back-to-front copies of the clip onto the global time line.

We can also imagine looping an animation a finite number of times, which corresponds to laying down a finite number of copies of the clip.
Figure 11.16. Playing an animation at twice the speed corresponds to scaling its local time line by a factor of $\frac{1}{2}$. 
Figure 11.17. Playing a clip in reverse corresponds to a time scale of -1.
Comparison of Local and Global Clocks
Synchronizing Animations with a Local Clock

Figure 11.18. The order of execution of disparate gameplay systems can introduce animation synchronization problems when local clocks are used.
Synchronizing Animations with a Global Clock

Figure 11.19. A global clock approach can alleviate animation synchronization problems.
A Simple Animation Data Format
Figure 11.20. An uncompressed animation clip contains 10 channels of floating-point data per sample, per joint.
Figure 2: Secondary Relative. *Moving the green external target affects the reach pose and the red spline, but not the rest pose.*

http://chrishecker.com/Real-time_Motion_Retargeting_to_Highly_Varied_User-Created_Morphologies
Continuous Channel Functions

Figure 11.21. The animation samples in a clip define continuous functions over time.

Figure 11.22. Many game engines use a piece-wise linear approximation when interpolating channel functions.
Figure 11.23. A special event trigger channel can be added to an animation clip in order to synchronize sound effects, particle effects, and other game events with an animation.
End of Chapter 9