INFT3960 – Game Production

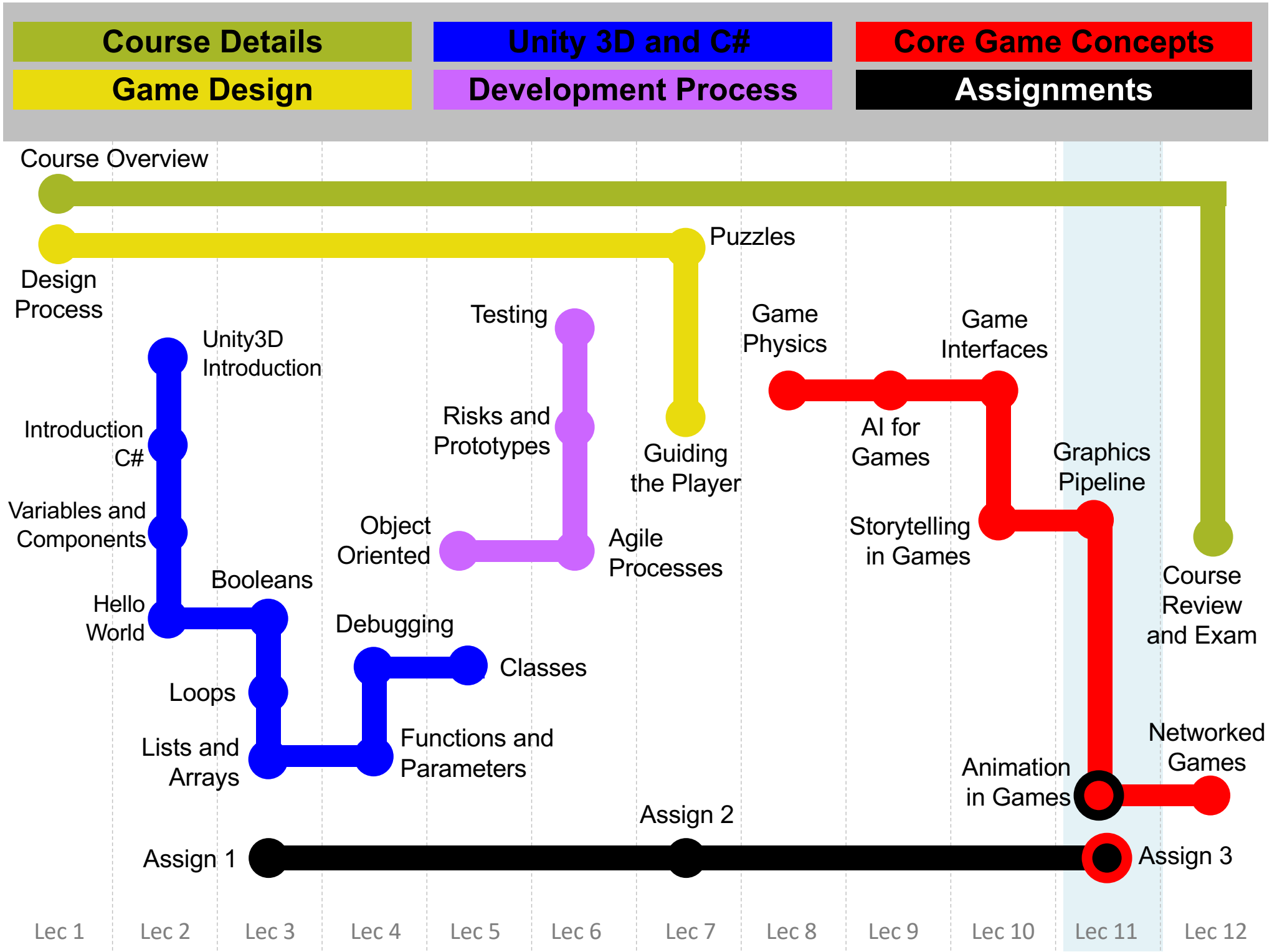
Week 11

Module 11.2

Animation in Games

Course Overview

Lec	Start Week	Modules	Topics	Assignments
1	3 Aug	Mod 1.1, 1.2	Course Overview, Design Process	
2	10 Aug	Mod 2.1, 2.2, 2.3, 2.4	Unity3D Introduction, Introduction C#, Variables and Components, Hello World	
3	17 Aug	Mod 3.1, 3.2, 3.3	Booleans, Loops, Lists and Arrays	Assign 1 21 Aug, 11:00 pm
4	24 Aug	Mod 4.1, 4.2	Functions and Parameters, Debugging	
5	31 Aug	Mod 5.1, 5.2	Classes, Object Oriented	
6	7 Sep	Mod 6.1, 6.2, 6.3	Agile Processes, Risks and Prototypes, Testing	
7	14 Sep	Mod 7.1, 7.2	Puzzles, Guiding the Player	Assign 2 18 Sep, 11:00 pm
8	21 Sep	Mod 8.1	Game Physics	
9	12 Sep	Mod 9.1	AI for Games	
10	19 Oct	Mod 10.1, 10.2	Game Interface, Storytelling in Games	
11	26 Oct	Mod 11.1, 11.2	Graphics Pipeline, Animation in Games	Assign 3 1 Nov, 11:00pm
12	2 Nov	Mod 12.1, 12.2	Networked Games, Course Review	


Course Details
Unity 3D and C#
Core Game Concepts
Game Design
Development Process
Assignments
Course Overview
Puzzles
Design Process
Testing
Game Physics
Game Interfaces
Unity3D Introduction
Risks and Prototypes
Guiding the Player
AI for Games
Graphics Pipeline
Introduction C#
Variables and Components
Object Oriented
Agile Processes
Storytelling in Games
Course Review and Exam
Booleans
Hello World
Debugging
Classes
Loops
Functions and Parameters
Lists and Arrays
Networked Games
Animation in Games
Assign 2
Assign 1
Assign 3
Lec 1
Lec 2
Lec 3
Lec 4
Lec 5
Lec 6
Lec 7
Lec 8
Lec 9
Lec 10
Lec 11
Lec 12

Animation in Games – Topics

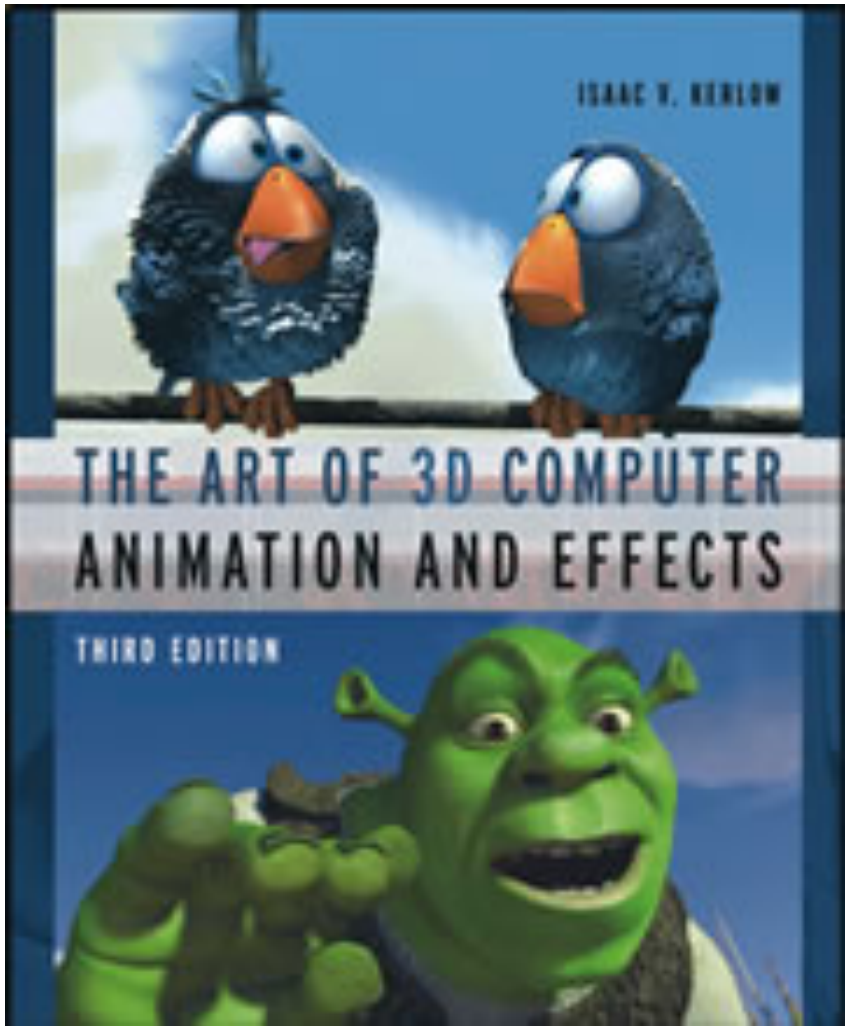
- Frame rates
- 2D Animation
- Tiling
- 3D Animation
- Motion Capture
- Morphing
- Skeletal Animation

Useful References

Kerlow , I. (2003).

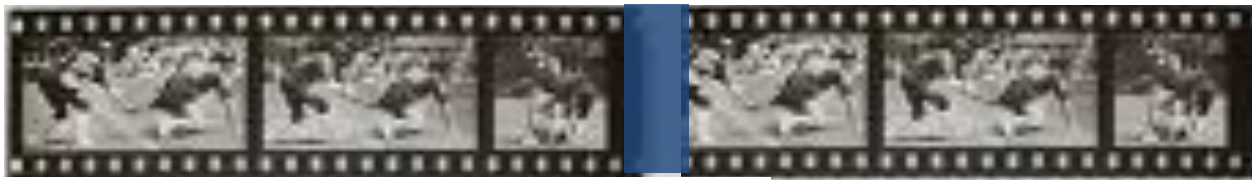
The Art of 3D Computer Animation and Effects.

(3rd edition), Wiley



Background

Computer animations use a sequence of images in order to create an illusion of change.



Often the change involves some kind of movement in position or orientation.

But other properties may also change such as.. colour, lighting, shape,....

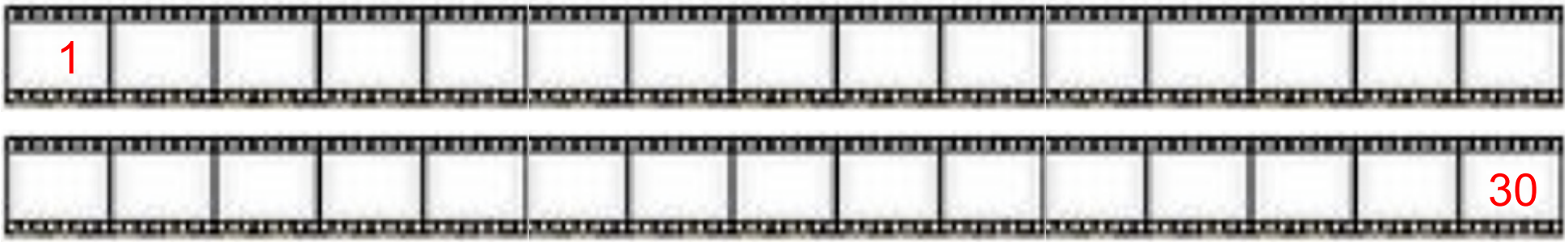
Frame Rate

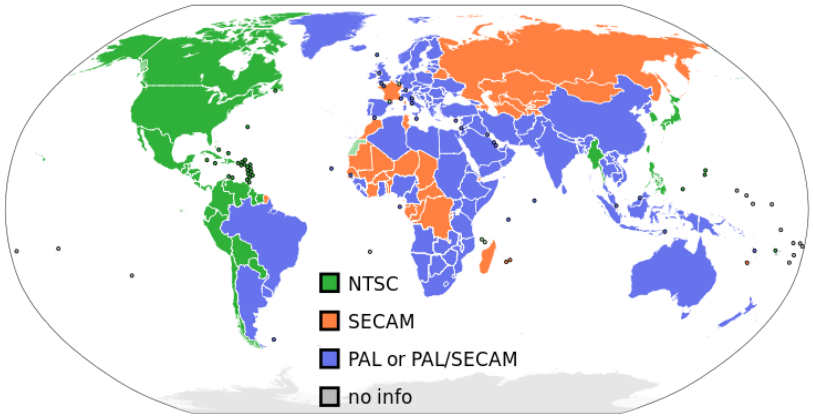
The images need to change at a rate of at least 12 frames per second or changes may appear disjointed and the illusion broken.

Hand-drawn animated characters typically use about 15 frames per second. It is cheaper to produce a lower number of hand-drawn frames and often realism is not the primary goal.

Realistic Frame Rates

Motion picture standard frame rate is 24 frames per second (fps). Television typically runs at 25-30 frames per second. The illusion of movement is more realistic with faster rates.





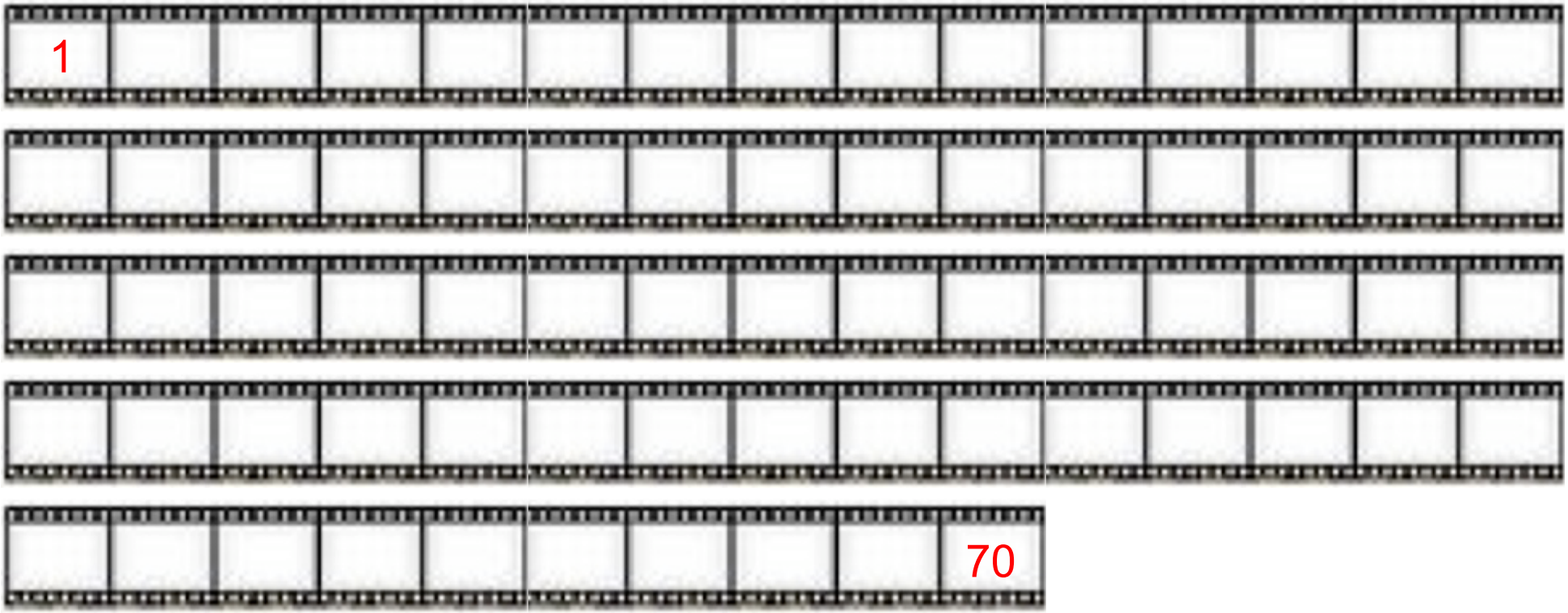
PAL – 25 fps

NTSC – 30 fps

SECAM – 25 fps

Realistic Frame Rates

After about 70 frames per second there is no further improvement due to the physical limitations of human perception.

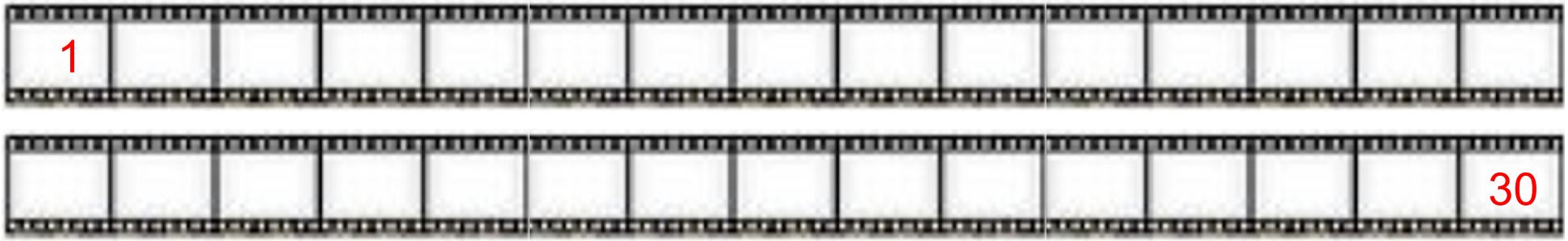


Interactive Frame Rates

In interactive games the frame rate will depend on the other functions that need to be carried out by the game engine.

A typical goal is 60 fps – although this can be difficult

A rate of 30 updates to the game world per second is desirable - especially if realism is required.



Interactive Frame Rates

Below 12 frames will produce a noticeable lag in gameplay that is disjointing for the player and may contribute to breaking the player's immersion in the game.

Controlling Frame Rate

The appropriate frame rate may depend on the type of game and the current situation in that game.

It can be difficult to control frame rates as it may be hardware dependent.

One question you need to consider is the use of time-based or frame-based animation

Controlling Frame Rate

Time-based versus frame-based animation

It is possible to use *elapsed time* rather than *frame rate* as a parameter in the animation.

This can keep the scene consistent (eg moving objects) but with slow frame rates – any animated changes may appear jumpy rather than smooth.

Quality versus speed

It is also possible to reduce the quality of rendered frames to ensure a reasonable frame rate.

2d Animation

In 2D games, animated scenes can be created by using vector graphics (models) or directly using bitmap images.

For each frame the position of the vector object can be moved to create the animation effect.

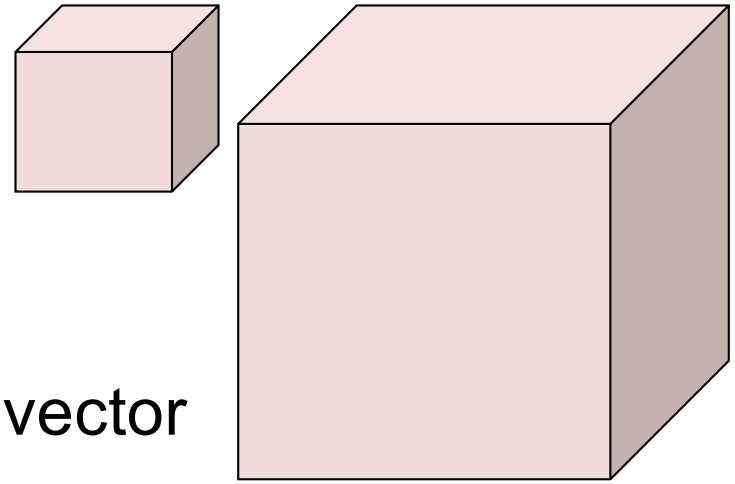
Other properties such as colour, size etc can also be changed.

Objects can be also added or removed as required in the scene.

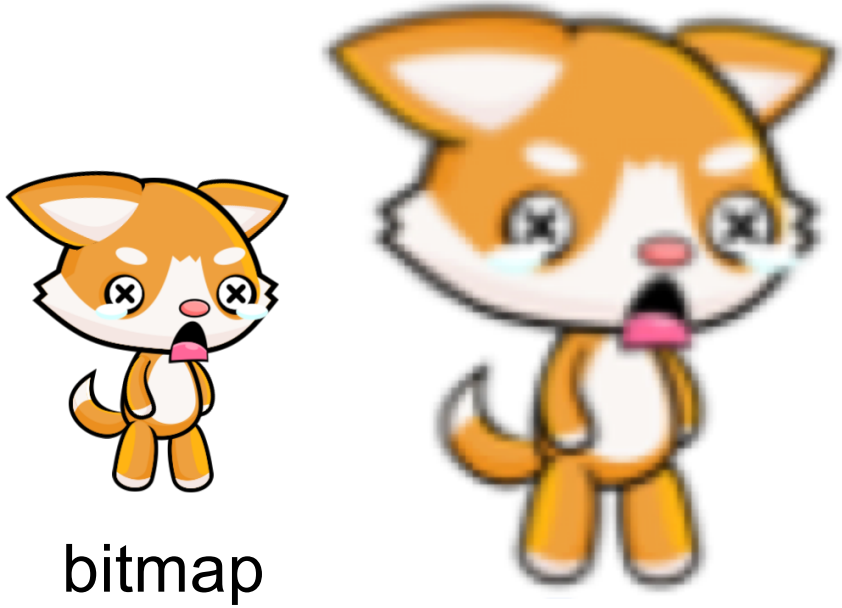
Vector Graphics

Objects described by vector graphics have some advantages over bitmap images.

They can be scaled to any size without loosing quality (resolution).



vector



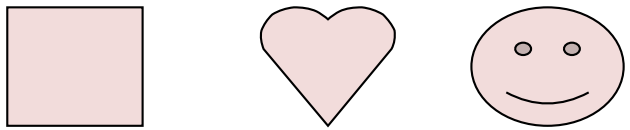
bitmap

Vector Graphics

Vector objects are described by a set of parameters e.g. a circle of radius 3 with colour blue at position (3,4).

So no information is lost in scaling as can happen when scaling images.

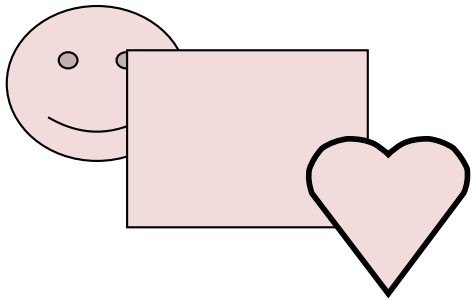
Since vector objects are completely described by their parameters this is all that needs to be stored. So they don't require much space in memory.



The graphical objects in powerpoint are an example of vector graphics

Vector Graphics

However vector objects may need to be redrawn, positioned, outlined, filled, etc on each update of the scene. When objects overlaps with other objects, these occlusions must be accounted for as well.



Even relatively simple scenes described in vector graphics can require a lot of calculations for each frame. Hence they can be too slow too draw in games at interactive rates.

Layered Bitmaps

The traditional approach in 2D games has been to use bitmap graphics layered over each other.

For example, a small character image is overlaid on a background scene to create a single image.

This approach takes advantage of fast hardware operations to support rapid scene rendering. (Blitting allow images to be overlaid together directly in memory)

Background image with character sprite overlaid

Animated Actions

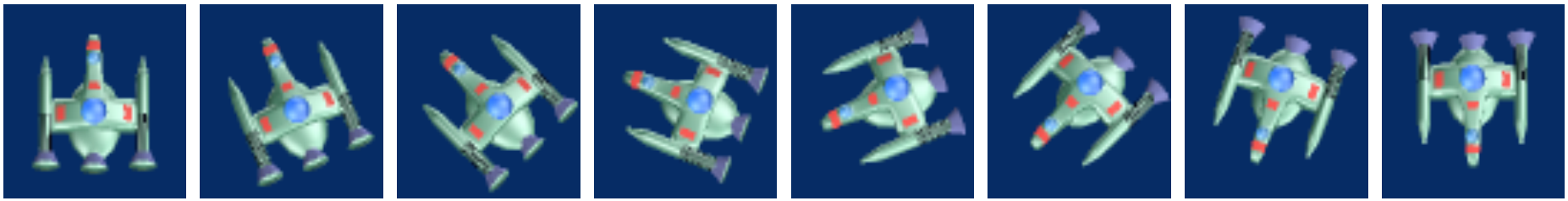
An animation sequence typically has to be produced for each action an object or character can undertake in the game

e.g. running, jumping, waving,....

Animated Actions

An animated sequence might also be used to signify an event (e.g. an explosion, space ship turning,..).

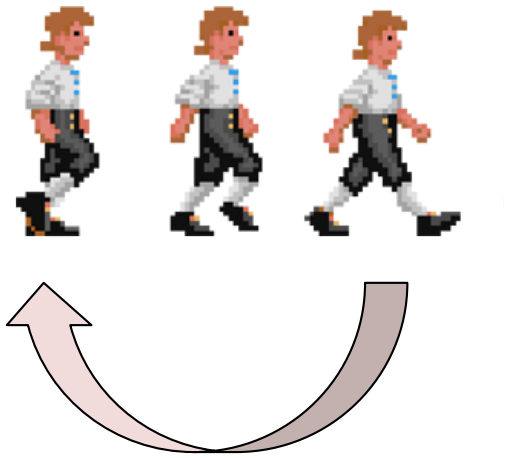
In this case the animation runs over a few frames, the image being changed at each iteration of the game loop until the sequence is completed.



A sequence of space ship sprites which could be used to show the ship turning.

Looping Actions

For more repetitive actions the sequence can keep playing in a loop.

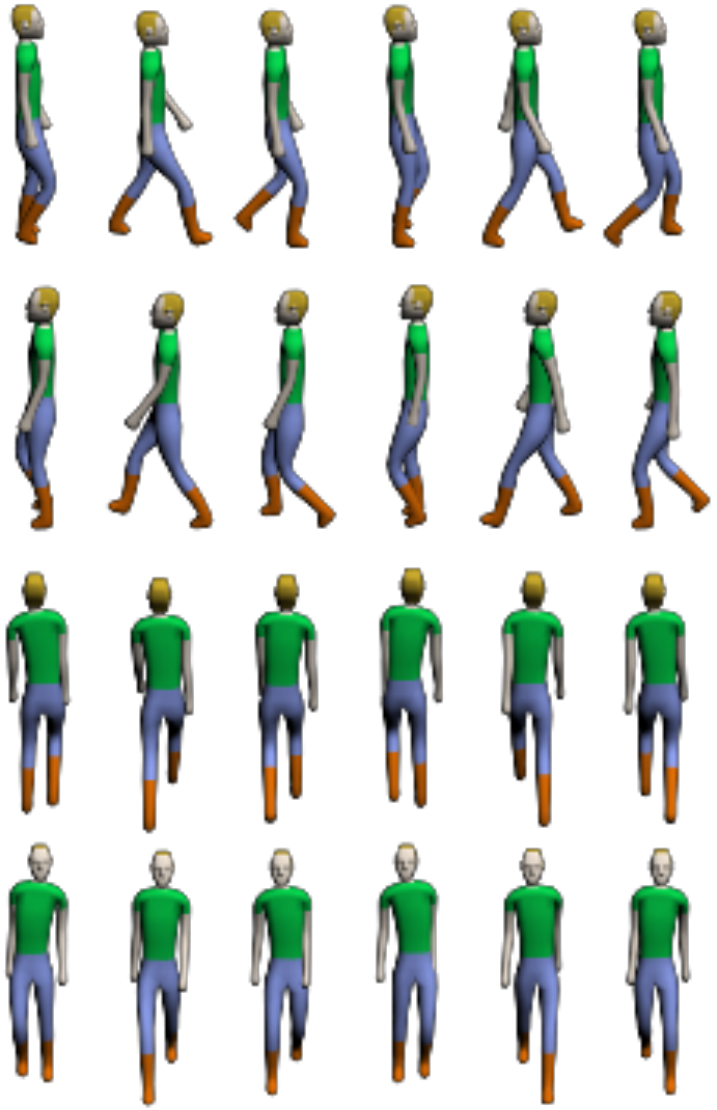


Looping Actions

In this example, the six images would be repeated while the character is in motion.

(Of course for actions like walking the position of the image also needs to be changed at each frame.)

Looping Actions



4 sprite sequences (each with 6 images) used to animate a character walking in four different directions

Sprites

In 2D games, small bitmaps were traditionally moved around a static background and became known as sprites.

Moving or changing sprites is still a very rapid and common technique for animating characters in 2D computer games

Sprites

Sprites can be overlaid on a background bitmap very quickly using direct memory transfers in hardware (blitting).

The hardware also takes care of a number of other effects like flipping, alpha-blending and simple transformations like rotation and scaling.

Creating Sprites

Since sprites are just images so they can be created in any tool that supports the creation of images.

In the past sprites were typically square images between 8x8 and 64x64 pixels in size.

Some sprite sizes may be more efficient on some platforms. (Although this is less of an issue with the graphics capabilities of modern hardware)

Creating Sprites

You also needed to consider the colour depth of the image (e.g. 8 bit versus 256 bit, etc).

Since sprites can be scaled you should also avoid lossy compressed formats such as jpg.

Gif images use non-lossy compression and because sprites often have large blocks of colour this compression format works well.

Sprite Sheets

The Prince of Persia:
The Shadow and the Flame
Prince Sprites

Colour Keying

Sprites take advantage of *colour keying* to exclude some pixels in the image. For example in the following example the colour pink would not be overlaid in the final image.

The colour used for keying is usually one that doesn't appear in most images - but it will depend on the software written for the game and can usually be set in the game engine.

sprite

final appearance

Sprites in 3D Games

The term 3D sprite is used in 3D games, but the images are still 2D - they are simply overlaid on a single 2D polygon called a billboard. Once again the image may change to depict an animation.

Rendering images onto a 2D plane (Billboard) is still a very fast operation in 3D games. Often faster than using a 3D model with texture mapped polygons.

Sprites in 3D Games

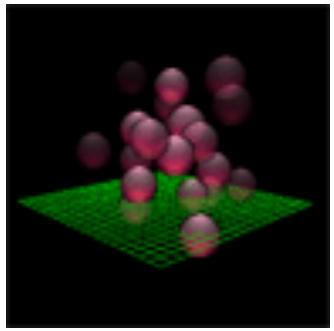
In a 3D game a flat polygon usually called a "billboard" is positioned to always face the camera (player).

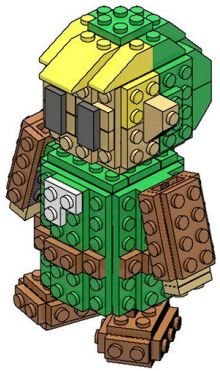
The 2D image is then directly overlaid on this billboard. Again colour keying can be used so that only the required part of the image is rendered.

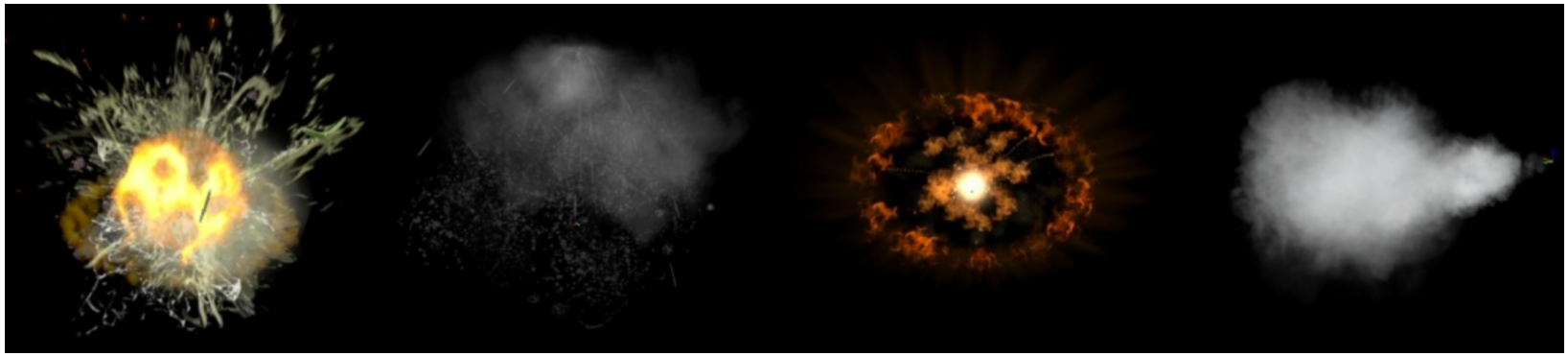
The illusion works well when the viewer doesn't notice that the sprite is flat and always turned to face them.

Sprites in 3D Games

3D sprites are particular useful for some difficult to create illusions such as fire, smoke, small objects, small plants (grass), symmetrical pickups and also for different particle effects.







Using 3D Sprites

3D Sprites are usually most effective when:

- Sprite image depicts a three dimensional object
- Animation is constantly changing or depicts rotation
- Sprite exists only shortly
- Object has a similar appearance from many common viewing angles (rotational symmetry e.g. sphere)
- Viewer accepts that the depicted object only has one perspective (such as small plants or leaves)

Tiling

Although tiling is not really a feature of animation, we discuss it here as it is an important concept in building game worlds.

Tiling is predominantly associated with 2D games but is also used in some 3D games. Most 2D scrolling platform games (vertical and horizontal) use this technique.

Note that this term is also used in a slightly different way when related to texture mapping. In texture mapping the texture image may be smaller than the surface it needs to cover - in this case the texture image may be "tiled" or repeated until it covers the complete surface.

Tiling

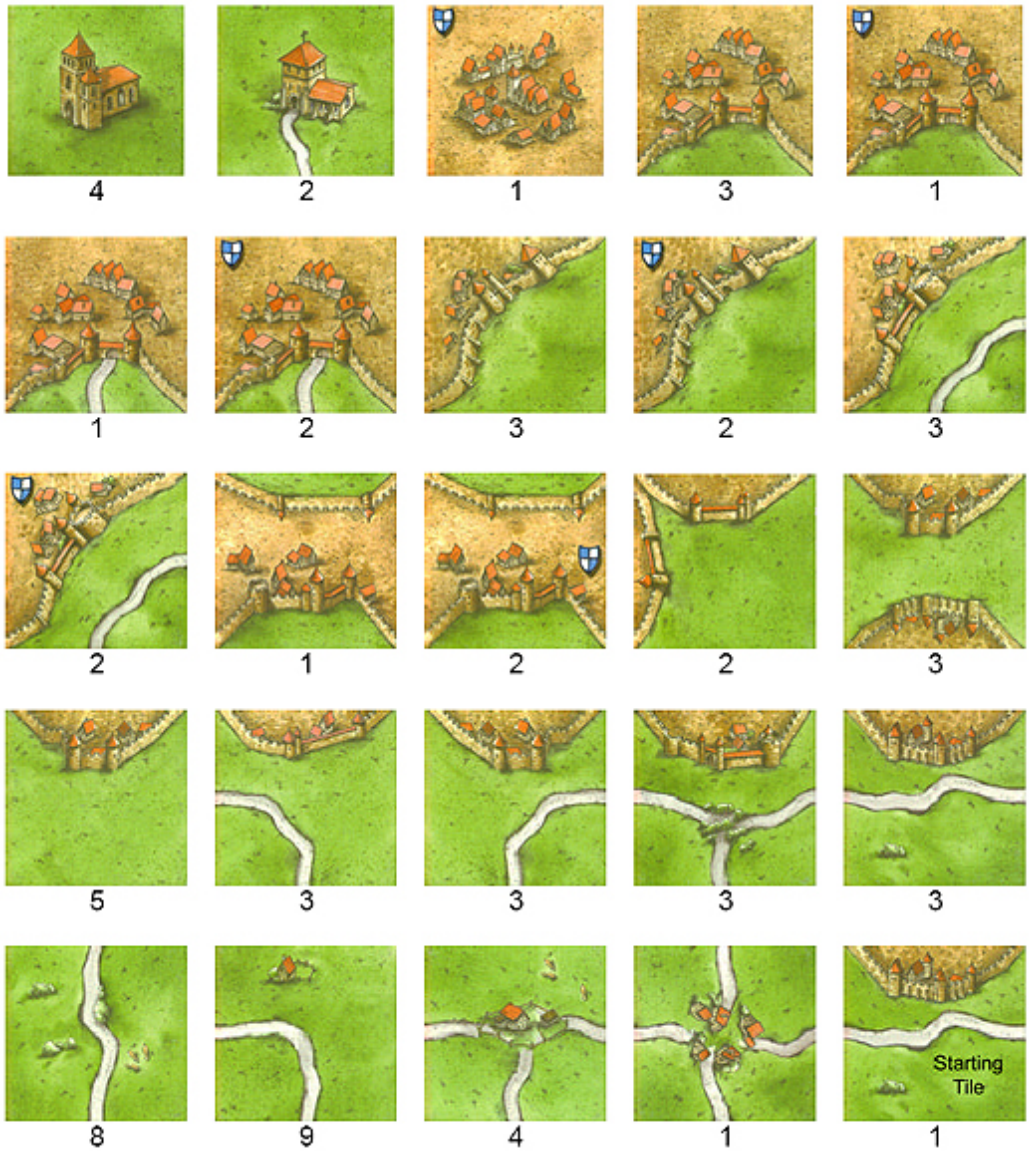
In 2D games the game world is often specified as a group of tiles. These tiles can be configured together to create many different scenes while only requiring a limited number of tiles. Using a map editor each new level of the game could simply use a new arrangement of tiles. This means less art resources need to be created.

Tiling

A tiled background can be composed from a matrix of tiles. Each tile contains an index to a square pixel bitmap known as the tile. So what ends up on screen is a matrix of tiles or bitmaps.

Tiling

This approach has the advantage that if a very large world is made of tiles then only the tiles that make up the part of the world currently in view need to be drawn.



3D Animation

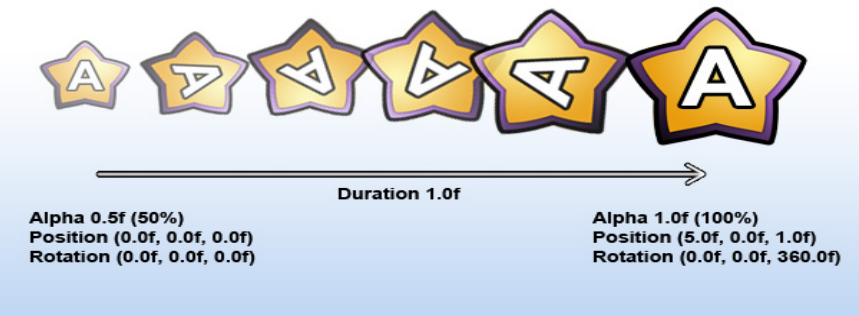
Apart from the use of animated sprites in 3D game worlds, most animation is achieved using either :

- 3D models designed to move directly by an animator
- automatic techniques based on physics.

Like in 2D animation any attribute of the model can be animated, including position, rotation, size, shape, etc

Tweening

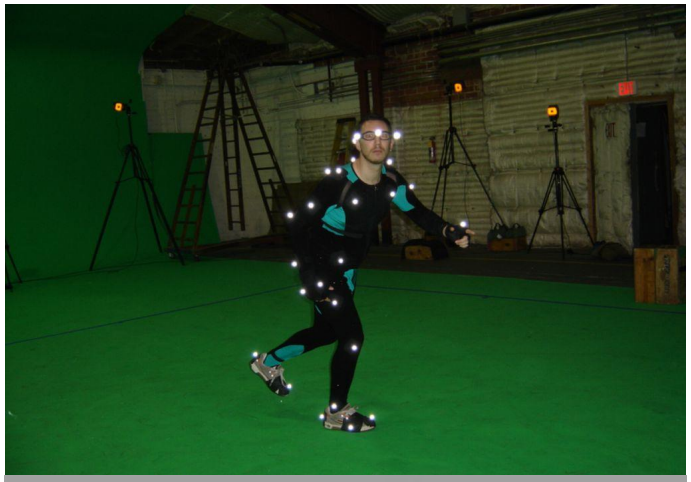
Often in animation modelling only keyframes are directly specified and the inbetween frames are then calculated automatically (tweening).

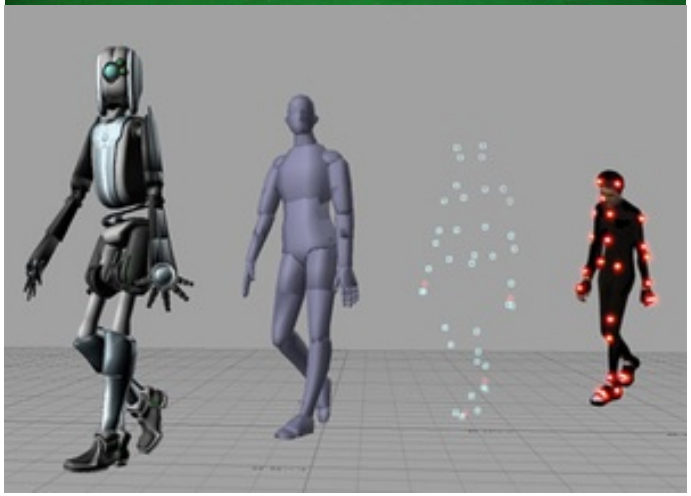


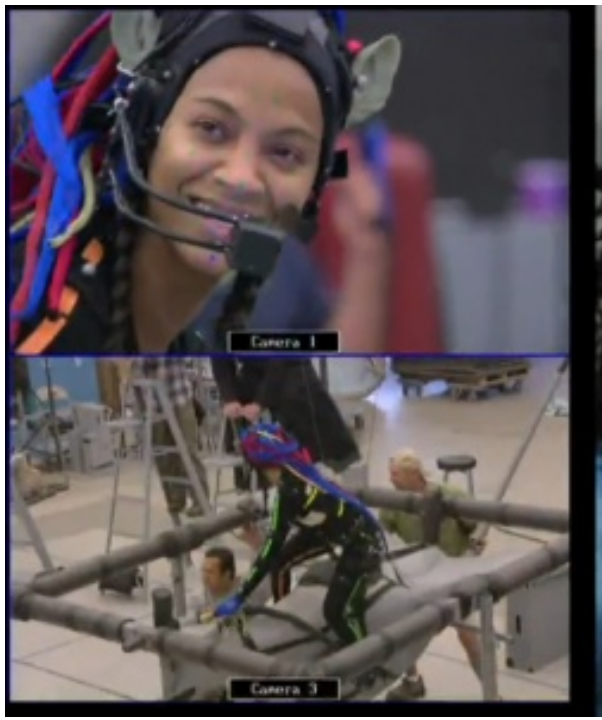
For translation, scaling and rotation this can be very effective and often only a start and end condition needs to be specified.

3D Animation

Another alternative is to use motion capture equipment to capture the position of key parts on an object.







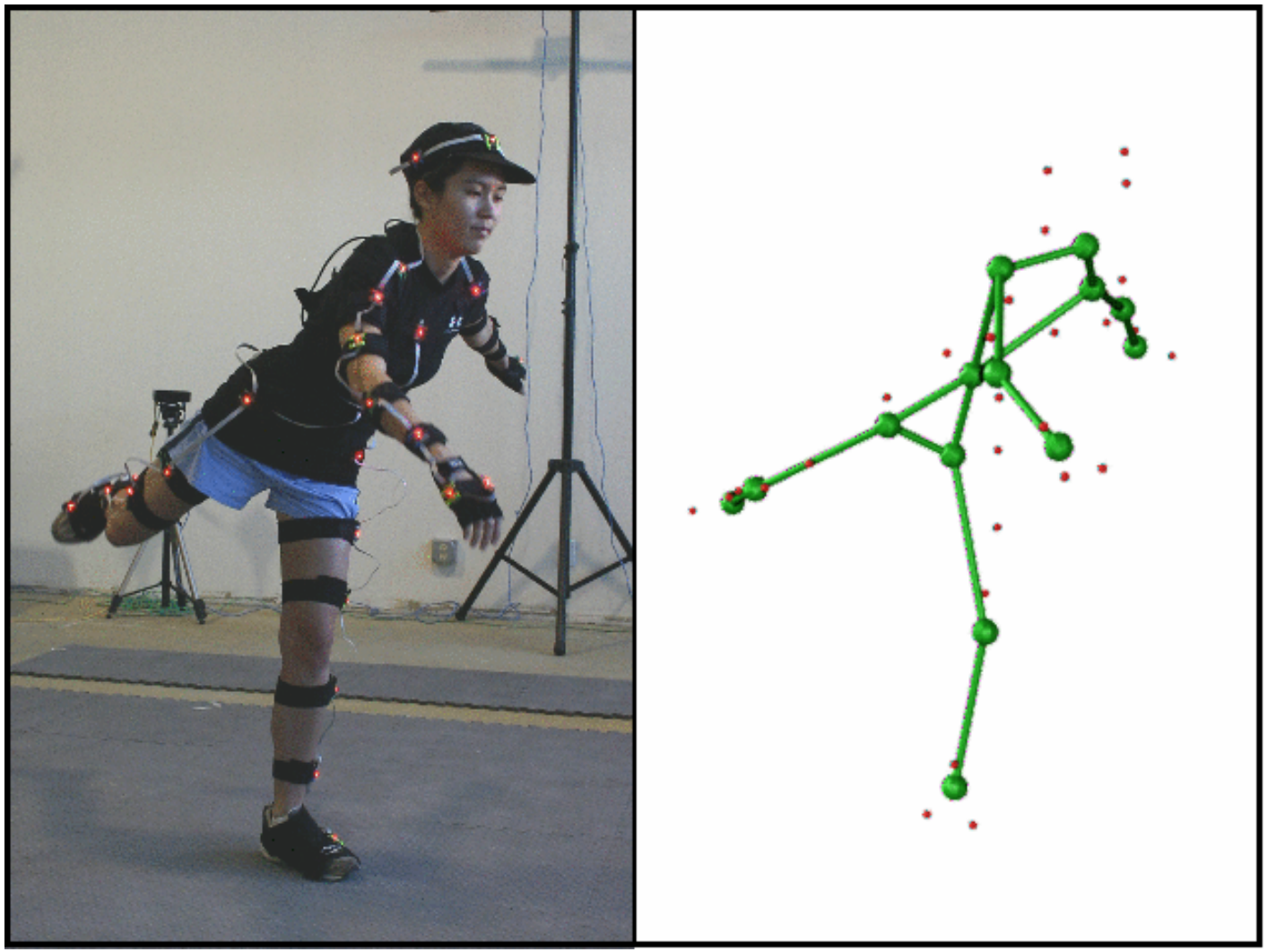
Motion Capture System

A motion capture system allows you to capture the key position of various points of moving objects (usually people).

This can be a fast way to develop complex (and very subtle) character animations. Because the key skeletal (or marker) positions are captured from real objects such as an actor) the movement will also appear very natural.

Motion capture systems can capture very detailed motion as well as larger body motion.

Motion Capture System



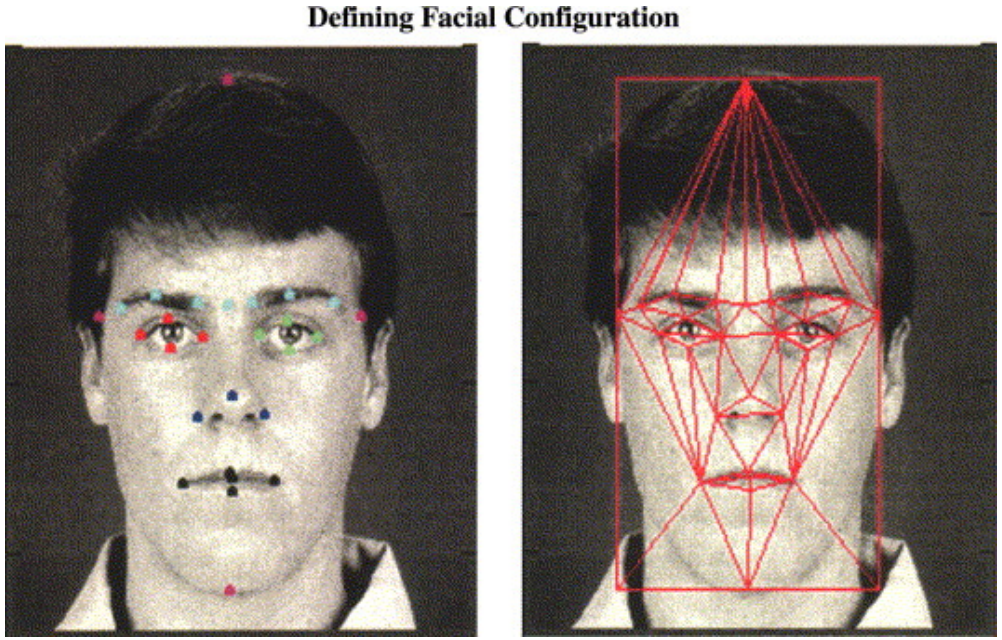
Motion Capture System

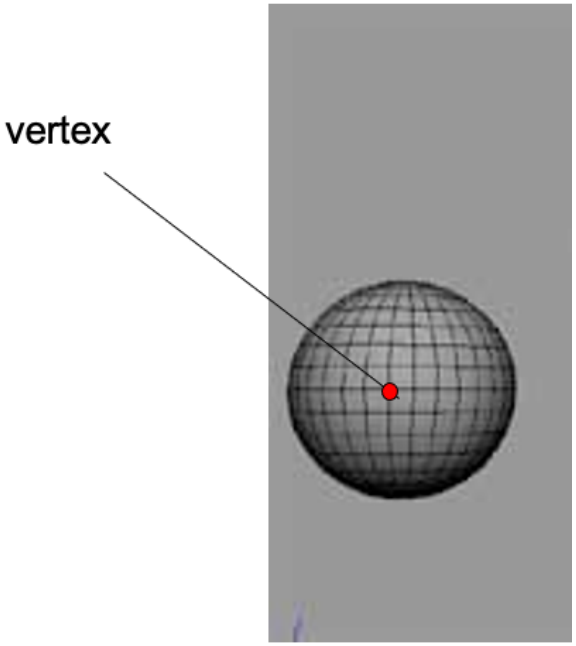
Motion capture systems should enable accurate measurement of the positions and orientation of the key points such as joints

Positions usually need to be recorded in real time so that they can be used to animate each frame of the animation.

Depending on the points captured with the motion system this approach can be used for either morphing or skeletal animation techniques.

Morphing



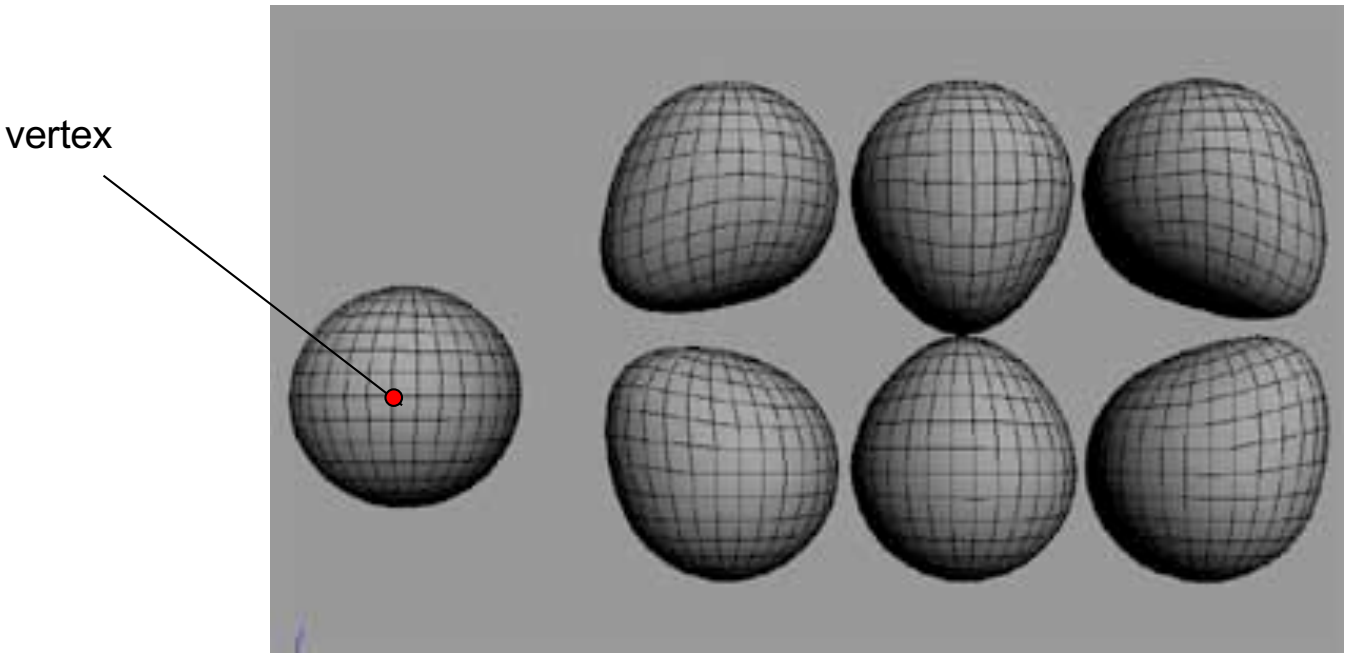


Morphing is a complex operation that can be used to help create animations by considering how object vertices change

Morph Target Animations

Morph target animation uses a series of stored locations for each point (vertex) in the polygon mesh.

In each keyframe the points (vertices) are moved to their new position. This is also called per-vertex animation.



Morphing

Compared to skeletal animation (discussed shortly) the artist has more control over the movements because they can directly define the individual positions of the vertices within a keyframe, rather than being constrained by skeletons.

However, there can be a lot of points to move so this is often a more time-consuming process than skeletal animation.

(Tweening might also be used to automate this)

Morphing

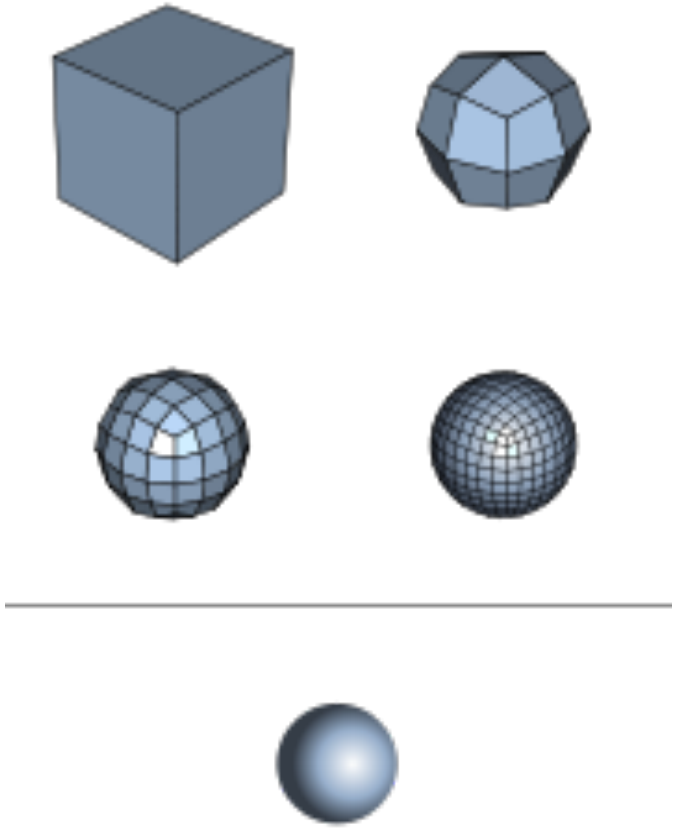
This is useful for things which move in a way that is not well described by the internal bone structure required for skeletal animation. (e.g. cloth, skin, facial expressions)

Source: nvidia.com

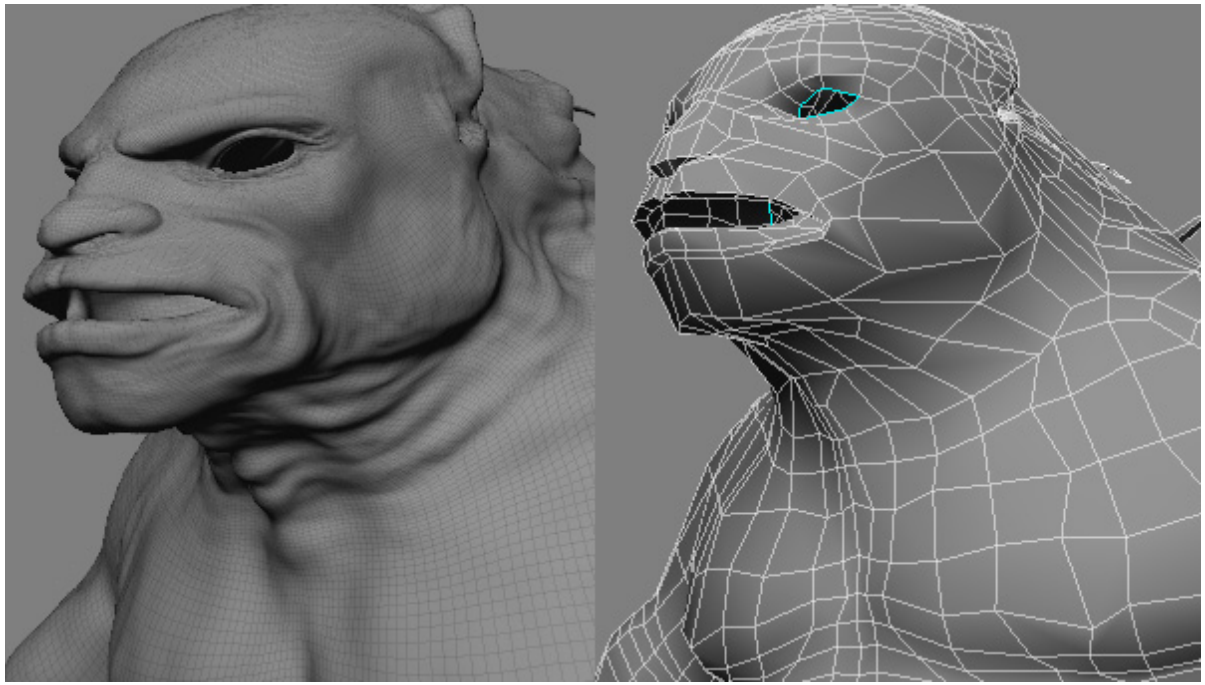
Animations Steps

The first step of animation in 3D is to create the 3D models themselves.

As with all 3D models the objects are described by a mesh of polygons.



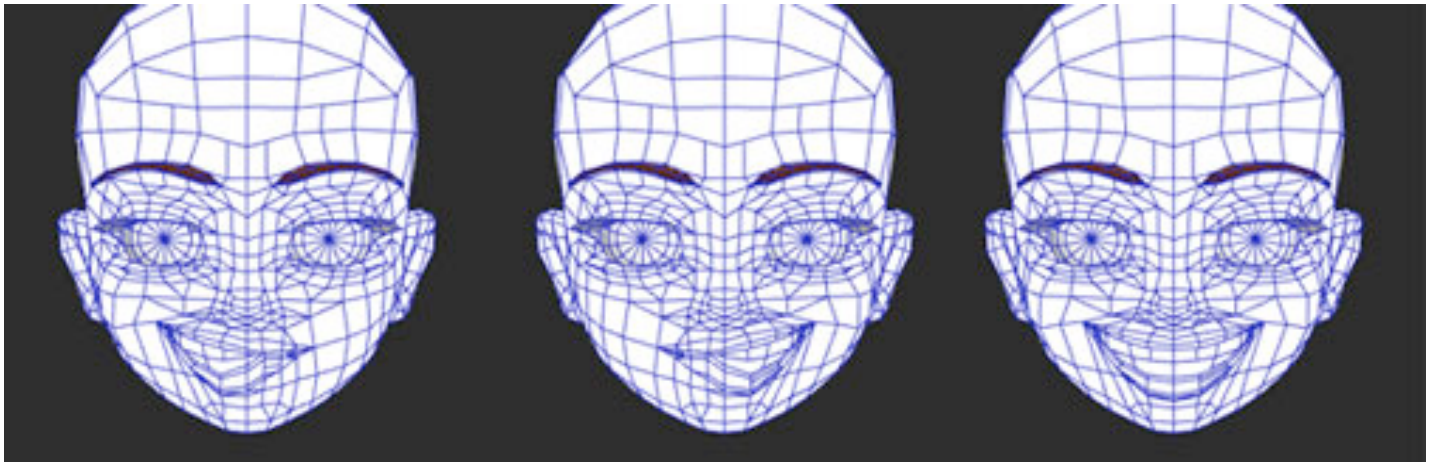
Animations Steps



This mesh of polygons is usually kept as simple as possible, with extra detail added using textures overlaid on the polygonal mesh.

With characters this mesh is often called the "skin".

Movement

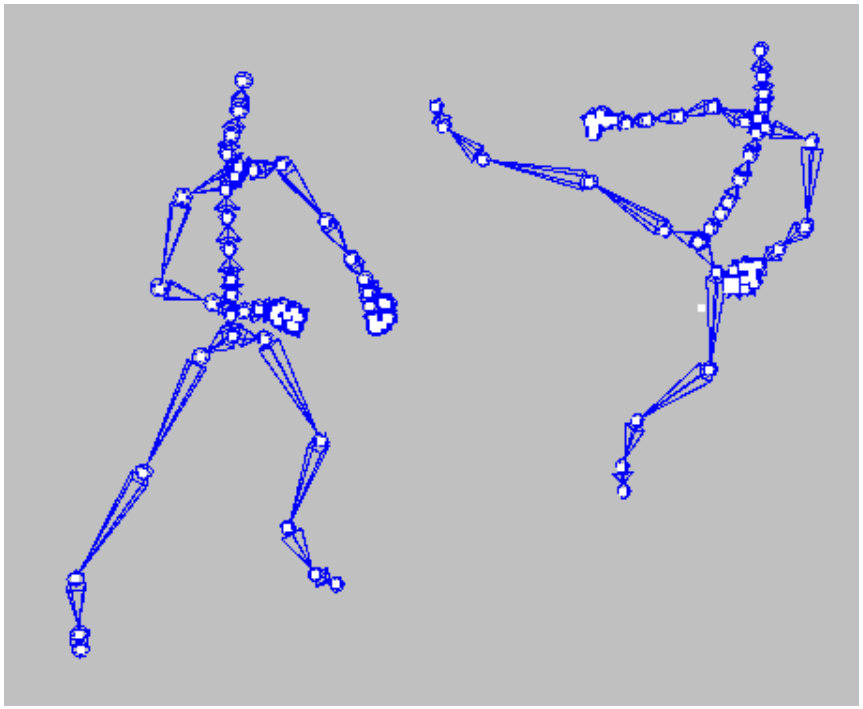


Parts of the model or indeed the whole mesh that represents the object can then be animated –

e.g. points in each polygon could be moved directly or the whole object could be rotated.

Skin and Bones

Often the polygonal mesh of an object requires some extra modelling e.g. a skeleton ("bones") or rigging to help simplify the animation process.



Skeletal Animation

Skeletal animation is a technique often used for animating animals or people. (Although mechanical things like cranes also work well).

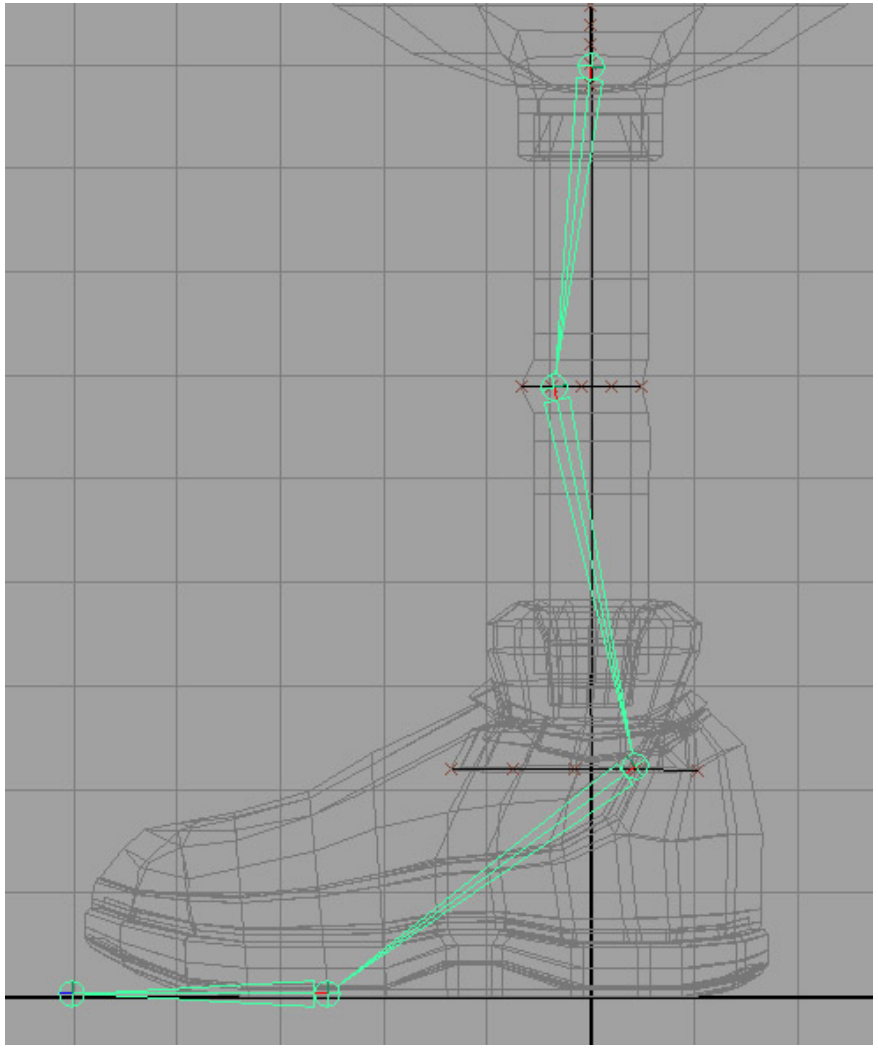
For example, a character is created in two parts: a surface representation of meshes and textures that are used to draw the character (called the skin) and a set of bones used to do the animation (called the skeleton).

Skin and Bones

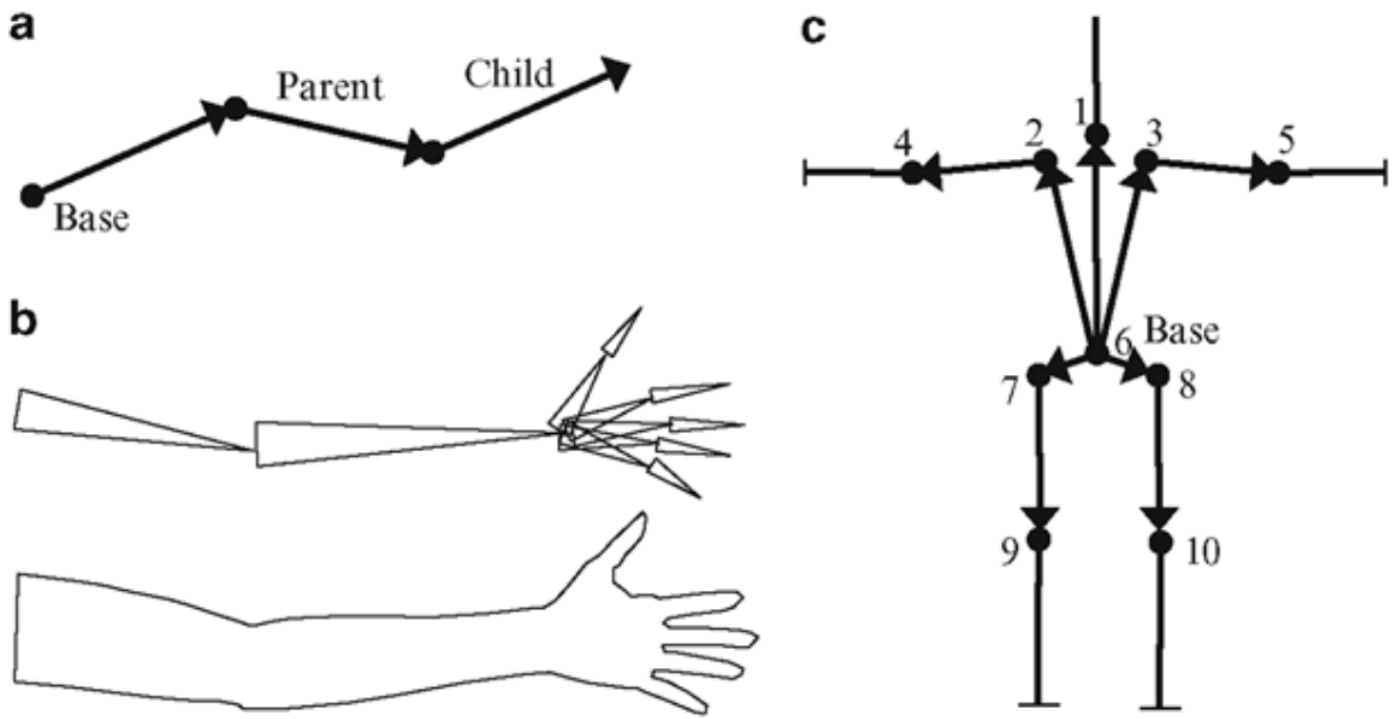
The skeleton of the character is constructed from bones and joints.

It can be specified in a hierarchical way so that a bone can have a parent bone.

So for example, the forearm is the parent of the hand. Therefore moving the forearm (parent) will also move the hand (child).

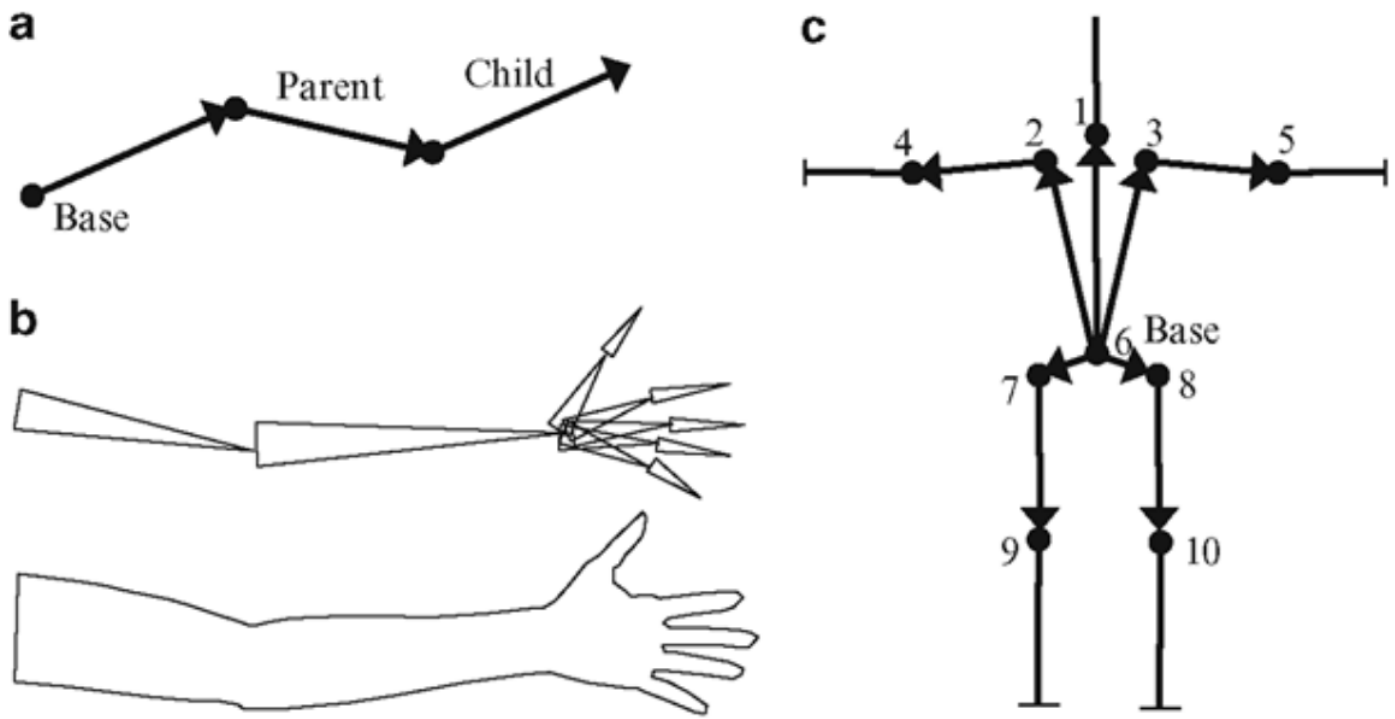


Skin and Bones



The movement at the joints of each bone is created by defining a three dimensional transformation for each bone. (This includes its position, scale and orientation).

Skin and Bones

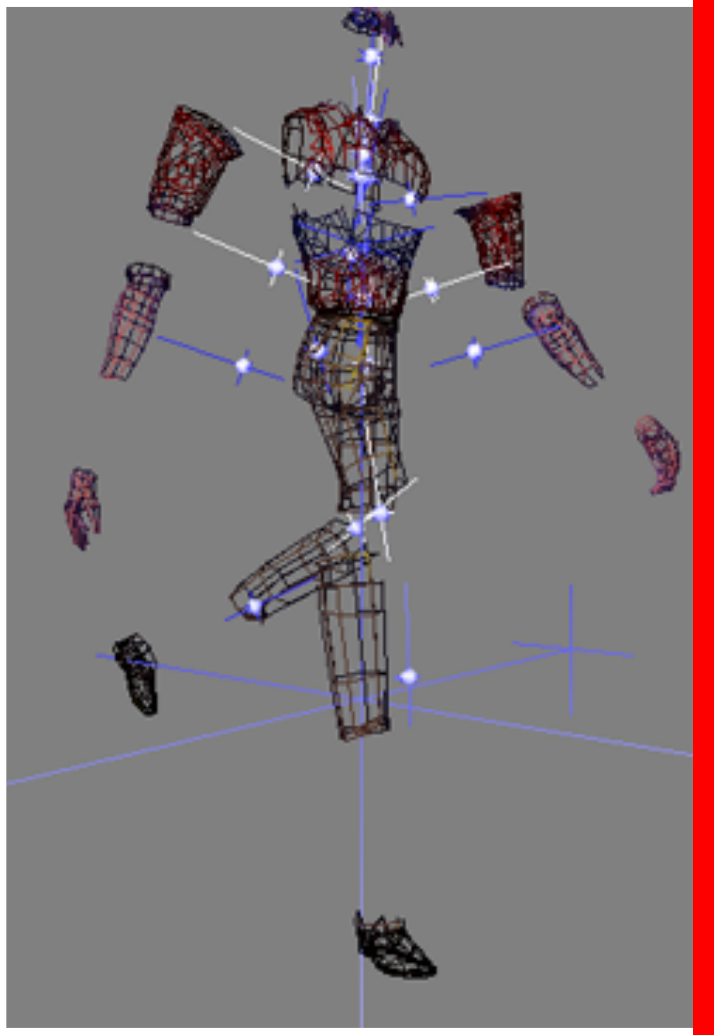


The character is animated as the position and orientation of the bones change over time (the skin can also move in relation to the bones).

Skin and Bones

Each bone in the skeleton is associated with a part of the character's visual representation e.g. polygons of the forearm are associated with the bone for the forearm.

It may be a more complex relation as some of the skin polygons near a joint may be under the influence of more than one bone. In this case the influence of each bone on a vertex is described by a weighting.



Pros and Cons

The major advantage of this technique is that there are less moving parts for the animator to control and so the focus is on large scale movement.

There are also some automated techniques which can be used to create animations of rigged characters.

These techniques can even be applied during the game loop.

Pros and Cons

Skeletal animation is an approximation of human movement.

This technique does not produce realistic simulation of the underlying muscles and skin movements.

If an animator is required to fine tune the animations for specific game actions it can be expensive.

Although this level of realism is not usually a problem for games.

Automatic Skeletons

There are also some useful techniques based on physics which can simplify the animation process of skeletal models. We will briefly discuss:

- Ragdoll physics
- Inverse Kinematics
- Forward Kinematics

These approaches can also be used for real-time animation of characters during a game. These is often desirable in games as they can be configured to respond correctly to the current game event and current scene.

(e.g. character falls onto a table or onto the ground, or a park bench)

Ragdoll Physics

This approach was traditionally used in early games for creating context dependent death scenes of characters.

The character animation is based on the underlying bone structure. To make the animation occur in real time, the solving techniques make use of approximations. Such as assuming no stiffness or rotation in the joints.

Ragdoll Physics

Thus in these type of animations a character is usually seen collapsing into unusual position like a toy rag doll.

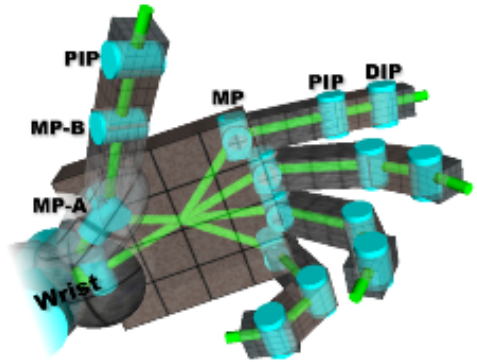
These animations although somewhat unrealistic can be done in real time and hence configured for the current state of the world in a way that prerecorded animations cannot.

Kinematics

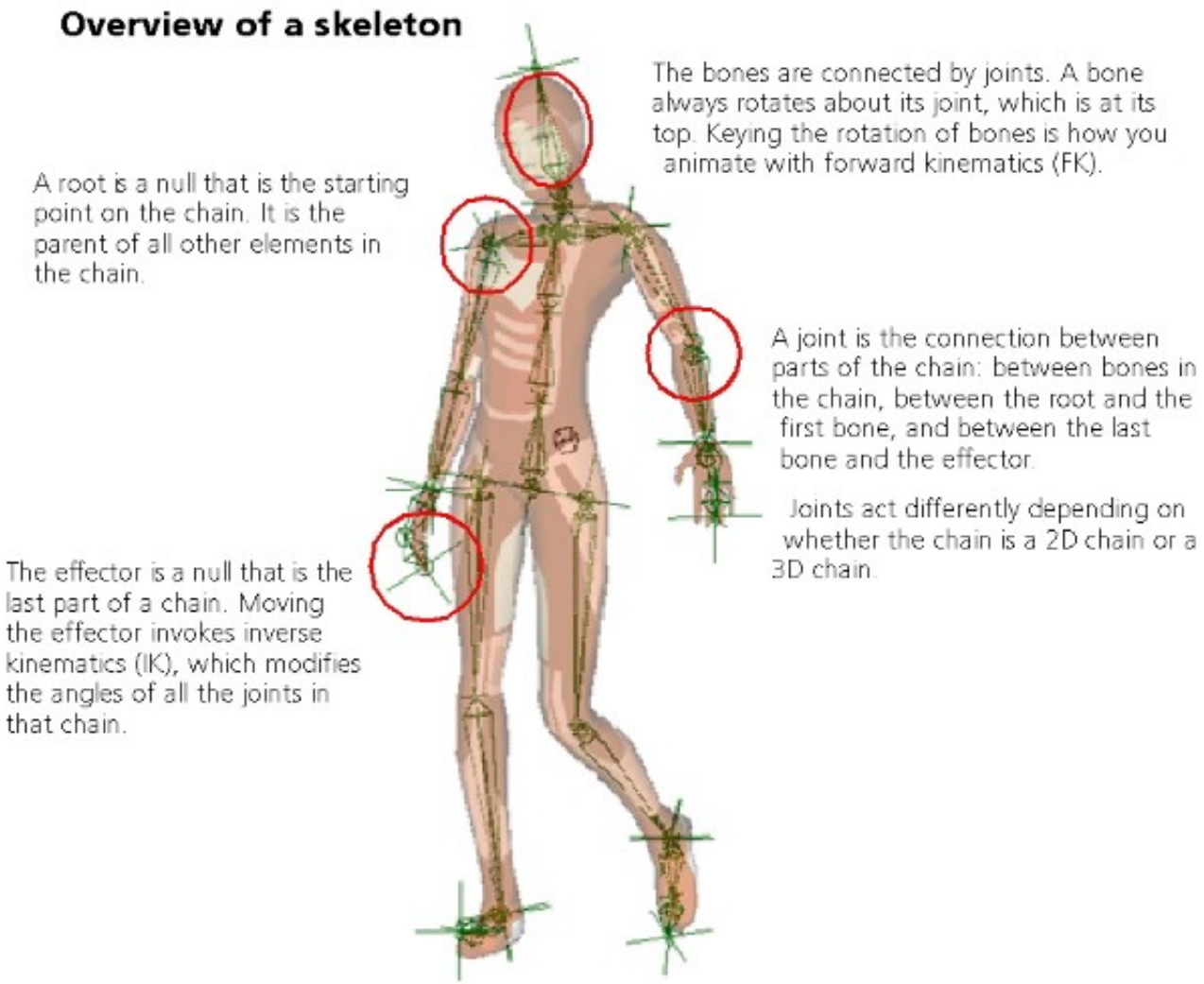
The bones and joints in a skeletal model form what is called a kinematic chain.

The joints are flexible and can undergo certain transformations that allow the bones to be moved and rotated in 3D. (There may be limitations to movements at some joints)

If you wish to move the end part of the chain (e.g. the hand or foot) then you need to calculate the appropriate transformations for each joint for each frame of the animation.



Kinematics



softimage.wiki.avid.com/ xsidocs/skel.htm

Inverse Kinematics

One approach to animating a kinematic chain is to specify the final end position and orientation of the last part of the chain.

Automated solvers can then calculate the required transitions and joint positions of the other bones in the chain. This is called inverse kinematics. However it is a very difficult problem to solve and there is no unique solution.

This can be a time consuming problem for the animator, who has to specify all the movements at each joint. But it ensures that the end parts of the chain will end in the correct position. e.g. that the characters feet land on the floor as they walk.

Forward Kinematics

Forward kinematic animation starts at the beginning of the chain and calculates the movement of all the links (bones) until it calculates the the position of the last bone of the chain.

For example to move the thumb, first the shoulder would be moved, followed by the elbow, wrist and finally the base of the thumb. All these transformations in the parent bones would specify the final position at the end of the thumb.

Creating this type of animation can be achieved automatically using a forward kinematics solver. The calculation is simpler than the inverse kinematics problem (but may not end at the correct position).

3D Animation Summary

We have discussed a few 3D animation approaches in detail :

- Skeletal animation (ragdoll, kinematics)
- Motion capture
- Morphing

Note that these many of these techniques are also used in film making (CGI). However in films there is no particular speed limit or resource limitation for rendering each frame. Once again, game animations must not interfere with the need for interactive rendering rates.