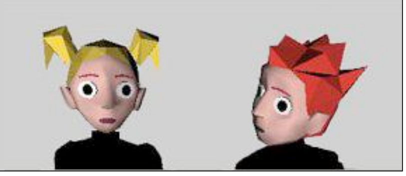
UCL

## Computer Animation

Tim Weyrich
March 2010

Heavily based on slides by Marco Gillies



## Computer Animation

- "Making things move"
- A key aspect of computer graphics
- Non-realtime for films
- Realtime for virtual worlds and games

## Computer Animation: Bools

- 2 books:
- "Real-time 3D Character Animation" Nik Lever – Focal Press
  - Very good, but only handles characters
- "Advanced Animation and Rendering Techniques: Theory and Practice" Alan Watt and Mark Watt – Addison-Wesley
  - More general but sometimes hard to follow
- Everything you need is in the slides

## Computer Animation

- This course will:
- Outline the major techniques used in animation
- Discuss general animation, character animation and physical simulation
- Go into detail on a few key methods

## Computer Animation

- 2 basic classes of computer animation:
- Keyframe animation
  - Data driven
  - Hand animation or performance driven
- Simulation
  - Procedural/algorithm driven
  - Particle systems, physics, artificial life

## Computer animation

- Keyframe animation relies on data
  - from animators or actors
- Simulation or procedural animation takes a more algorithmic approach
  - Animation is directly generated by an algorithm or calculation

## Course Outline

- Traditional animation
- Keyframe animation
- Character animation
  - Body
  - Face
  - Behaviour simulation
- Physical systems
  - Physics simulation
  - Particle systems
  - Integration techniques

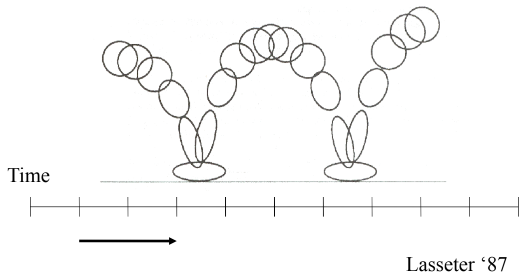
## Traditional Animation

## Flip Books

- The most basic form of animation is the flip book
- Presents a sequence of images in quick succession
- In film this becomes a sequence of frames

## The Time Line

- Animation is a sequence of frames (images) arranged along a time-line




## Frames

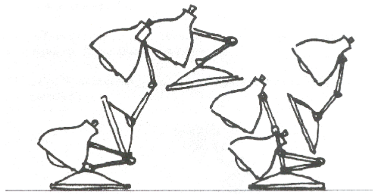
- Each frame is an image
- Traditionally each image had to be hand drawn individually
- This potentially requires vast amounts of work from a highly skilled animator

## Layers

- Have a background images that don't move
- Put foreground images on a transparent slide in front of it
- Only have to animate bits that move
- Next time you watch an animation notice that the background is always more detailed than the characters
- Japanese animation often uses camera pans across static images

## Keyframing

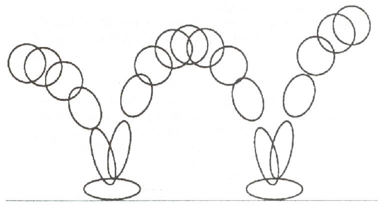
- The head animator draws the most important frames (Keyframes)
- An assistant draws the in-between frames (inbetweening)



Lasseter '87

## Other Techniques

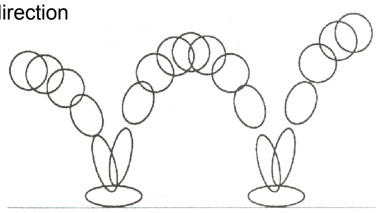
- Squash and stretch
- Slow in slow out



Lasseter '87

## Other Techniques

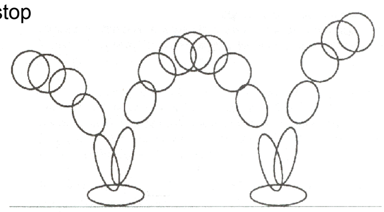
- Squash and stretch
  - Change the shape of an object to emphasise its motion
  - In particular stretch then squash when changing direction



Lasseter '87

## Other Techniques

- Slow in slow out
  - Control the speed of an animation to make it seem smoother
  - Start slow, speed up in the middle and slow to a stop



Lasseter '87

## Keyframe animation

## Keyframe animation

- Computer animation basics
- Computer based keyframing
- Interpolations methods
- Rotations and quaternions

## Keyframe Animation

- The starting point for computer animation is the automation of many of the techniques of traditional animation
- The labour savings can be greatly increased
- The following techniques are described for the 3-D case; 2-D is often even simpler

## Layering

- The essence of layering is that objects that move independently are animated independently, and only what is actually moving is animation
- This saving can be greatly increased with computer animation

## Properties of Objects

- In computer animation objects are now 3-D models rather than images
- Selected properties of these objects are animated rather than redefining the whole object in each frame
  - e.g. position, rotation, normal map, …
- Only changing properties need animation
  - e.g. you can rotate an object without having to do  anything to the texture

## Properties of Objects

- These properties are mostly numerical values
  - E.g. vectors for positions of objects, colour values, vertex positions for meshes
- Thus animation comes down to manipulating these values
  - directly, via GUI, or algorithmically

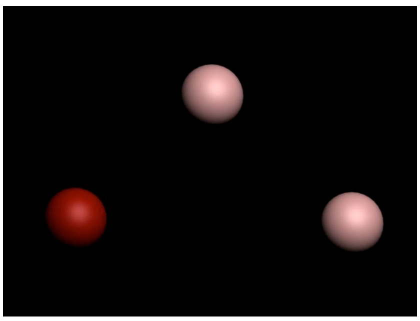
## Keyframing

- Keyframing is readily applicable to computer animations
- Keyframes are now settings for value at a given time (a tuple)

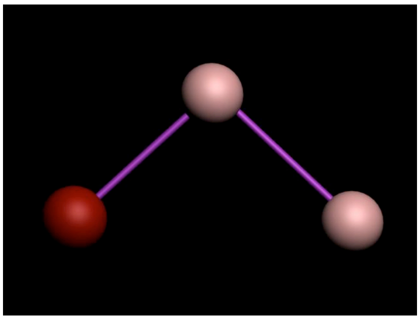
$$< time, value >$$

- The computer can do the inbetweening
- (For now I'll just talk about animating position)

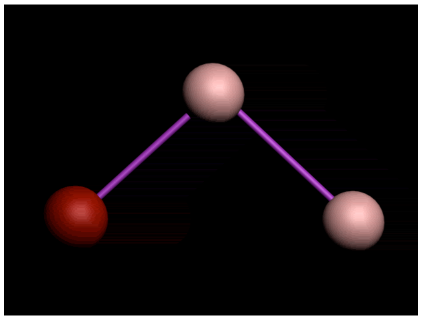
## Keyframing



## Linear Interpolation



## Linear Interpolation



## Linear Interpolation

- The position is interpolated linearly between keyframes

$$\mathbf{P}(t) = t\mathbf{P}(t_k) + (1-t)\mathbf{P}(t_{k-1})$$

## Linear Interpolation

- The position is interpolated linearly between keyframes

$$\mathbf{P}(t) = t\mathbf{P}(t_k) + (1-t)\mathbf{P}(t_{k-1})$$

$$\mathbf{P}(t) = \frac{t-t_{k-1}}{t_k - t_{k-1}}\mathbf{P}(t_k) + \left(1 - \frac{t-t_{k-1}}{t_k - t_{k-1}}\right)\mathbf{P}(t_{k-1})$$

## Linear Interpolation

- The position is interpolated linearly between keyframes

$$\mathbf{P}(t) = t\mathbf{P}(t_k) + (1-t)\mathbf{P}(t_{k-1})$$

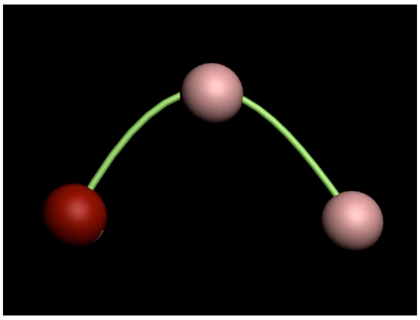
$$\mathbf{P}(t) = \frac{t-t_{k-1}}{t_k - t_{k-1}}\mathbf{P}(t_k) + \left(1 - \frac{t-t_{k-1}}{t_k - t_{k-1}}\right)\mathbf{P}(t_{k-1})$$

- The animation can be jerky
- Need some equivalent of slow-in slow-out

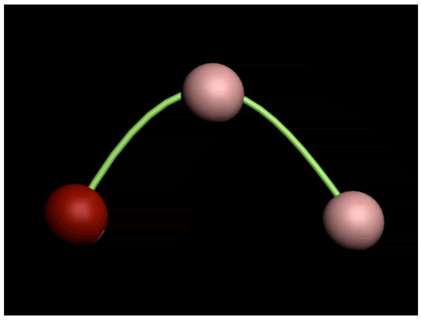
## Spline Interpolation

- Use smooth curves to interpolate positions
- Use curves similar to Bezier

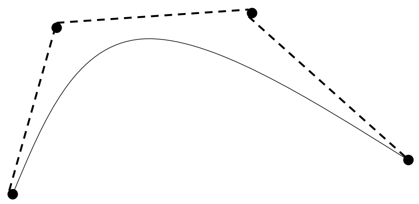
## Spline Interpolation



## Spline Interpolation

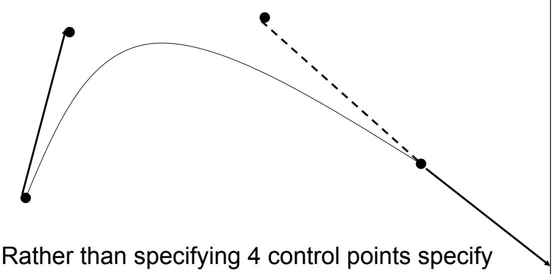


## Bezier Curves



- Smooth but don't go through all the control points, we need to go through all the keyframes

## Hermite Curves



- Rather than specifying 4 control points specify 2 end points and tangents at these end points
- In the case of interpolating positions the tangents are velocities

## Hermite Curves

- The maths is pretty much the same as Bezier Curves
- Bezier:

$$\mathbf{P}(t) = (1-t)^3\mathbf{P}_0 + 3t(1-t)^2\mathbf{P}_1 + 3t^2(1-t)\mathbf{P}_2 + t^3\mathbf{P}_3$$

- Mapping to Hermite:

$$\mathbf{P}_0 = \mathbf{P}_{B0}, \mathbf{T}_0 = 3(\mathbf{P}_{B0} - \mathbf{P}_{B1}), \mathbf{P}_1 = \mathbf{P}_{B3}, \mathbf{T}_1 = 3(\mathbf{P}_{B3} - \mathbf{P}_{B2})$$

- Hermite:

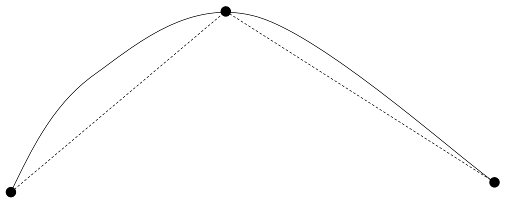
$$\mathbf{P}(t) = (2t^3 - 3t^2 + 1)\mathbf{P}_0 + (t^3 + 2t^2 + t)\mathbf{T}_0$$
$$+ (-2t^3 + 3t^2)\mathbf{P}_1 + (t^3 - t^2)\mathbf{T}_1$$

## Tangents

- That's fine, but where do we get the tangents (velocities) from?
- We could directly set them, they act as an extra control on the behaviour
- However often we want to generate them automatically

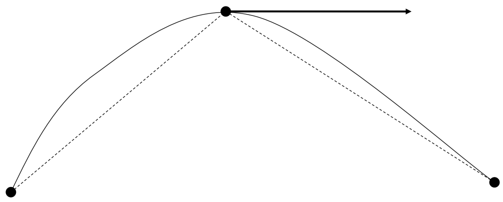
## Tangents

- Base the tangent as a keyframe on the previous and next keyframe



## Tangents

- Average the distance from the pervious keyframe and to the next one



## Tangents

- Average the distance from the previous keyframe and to the next one

$$\mathbf{T}_k = \frac{1}{2}(\mathbf{P}_k - \mathbf{P}_{k-1}) + \frac{1}{2}(\mathbf{P}_{k+1} - \mathbf{P}_k)$$
$$= \frac{1}{2}(\mathbf{P}_{k+1} - \mathbf{P}_{k-1})$$

- If you set the tangents at the first and last frame to zero you get slow in slow out

## Almost perfect…

- That’s pretty much it on keyframe animation
- But there’s one last problem:

## Almost perfect…

- That’s pretty much it on keyframe animation
- But there’s one last problem: Rotations

## Almost perfect…

- That’s pretty much it on keyframe animation
- But there’s one last problem: Rotations
- Rotations are used a lot on animation
- In fact human body animation is largely based on animating rotations rather than positions

## Rotations

- Rotations are very different from positions
- They are essentially spherical rather than linear

## Rotations

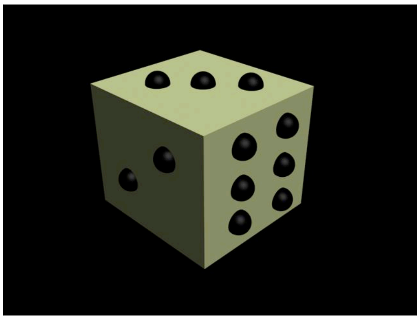
- Rotations are very different from positions
- They are essentially spherical rather than linear
- You can split them into rotations about the X,Y & Z axis (Euler angles), but:

## Rotations

- Rotations are very different from positions
- They are essentially spherical rather than linear
- You can split them into rotations about the X,Y & Z axis (Euler angles), but:
  - Then the order in which you do them changes in final rotation

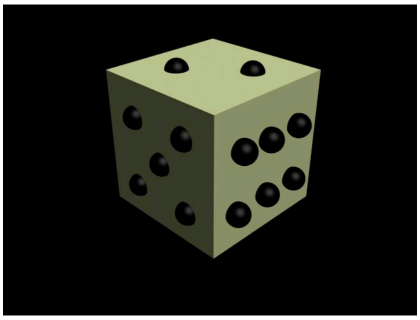
## Order of Rotations

- Rotating about XYZ



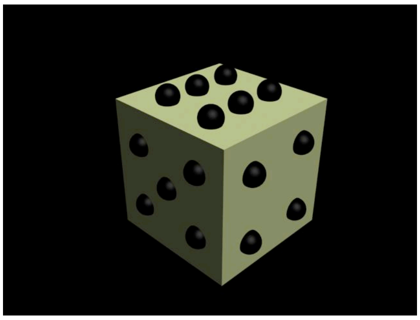
## Order of Rotations

- Rotate about X



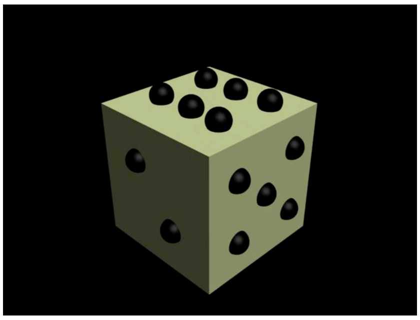
## Order of Rotations

- Rotate about Y



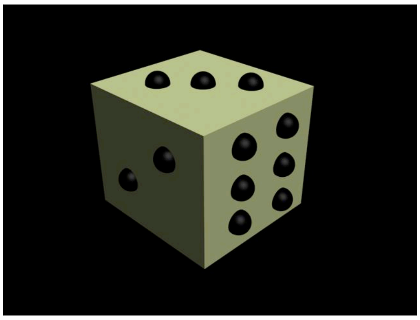
## Order of Rotations

- Rotate about Z



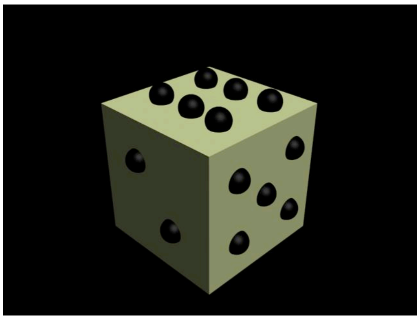
## Order of Rotations

- Start again



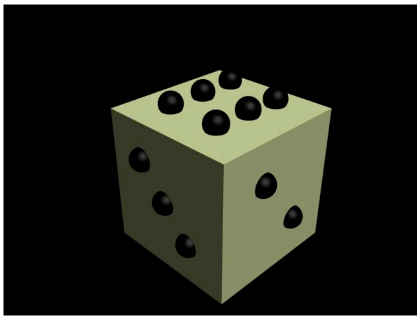
## Order of Rotations

- Rotate about Y



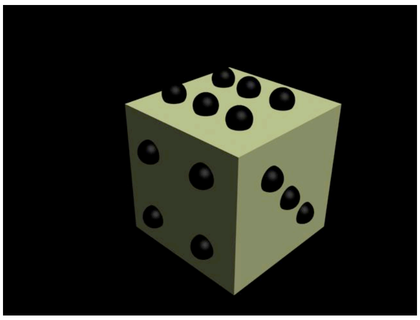
## Order of Rotations

- Rotate about X



## Order of Rotations

- Rotate about Z



## Rotations

- Rotations are very different from positions
- They are essentially spherical rather than linear
- You can split them into rotations about the X,Y & Z axis (Euler angles), but:
  - If you rotate about Y so that the Z axis is rotated onto the X axis you get stuck (Gimbal lock) and are in trouble

## Quaternions

- We need a representation of rotations that doesn't suffer these problems
- We use Quaternions
- Invented by William Rowan Hamilton in 1843
- Introduced into computer animation by Ken Shoemake
  - K. Shoemake, "Animating rotations with quaternion curves", ACM SIGGRAPH 1985 pp245-254

## Quaternions

- Quaternions are a 4D generalisation of complex numbers:

$$\mathbf{q} = w + xi + yj + zk$$

## Quaternions

- Quaternions are a 4D generalisation of complex numbers:

$$\mathbf{q} = w + v_x i + v_y j + v_z k$$

- The last three terms are the imaginary part and are often written as a vector:

$$\mathbf{q} = [w, \mathbf{v}]$$

## Quaternions

- The conjugate of a quaternion is defined as:

$$\bar{\mathbf{q}} = [w, -\mathbf{v}]$$

- And multiplication is defined as:

$$\mathbf{q}_1\mathbf{q}_2 = [w_1 w_2 - \mathbf{v}_1 \bullet \mathbf{v}_2, w_1\mathbf{v}_2 + w_2\mathbf{v}_1 + \mathbf{v}_1 \times \mathbf{v}_2]$$

## Quaternion Rotations

- A rotation of angle θ about an axis **V** is represented as a quaterion with:

$$w = \cos\left(\frac{\theta}{2}\right)$$

$$\mathbf{v} = \mathbf{V}\sin\left(\frac{\theta}{2}\right)$$

- All rotations are represented by unit quaternions (length 1)

## Quaternion Rotations

- A vector (**V**) is rotated by first converting it to a quaternion:

$$\mathbf{v} = [0, \mathbf{V}]$$

- Premultiplying by the rotation and postmultiplying by its inverse

$$\mathbf{q}\mathbf{v}\mathbf{q}^{-1} \qquad \mathbf{q}^{-1} = \bar{\mathbf{q}}$$

- And transforming back to a vector

## Quaternion Properties

- The arithmetic operators on quaternions don't have the same meaning as they do with vectors
- Concatenation and scale:
  - Vector addition -> multiplication
  - Vector subtraction -> multiplication by the inverse
  - Vector multiplication by a scalar -> multiply the angle

## Quaternion Properties

- Vector addition (of translations) means do 1 translation followed by another
- For Quaternions the equivalent operation is multiplication
  - q2*q1
- Order matters, q1 is performed first

## Quaternion Properties

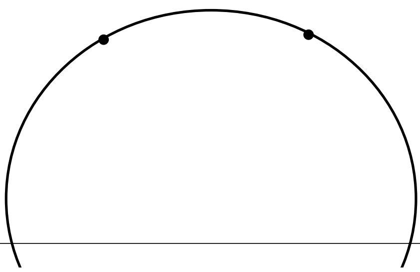
- Vector subtraction (of translations) means do the inverse of the first translation followed by the other
- For Quaternions the equivalent operation is multiplication by the inverse
  - $q2*q1^{-1}$
- Again order matters

## Quaternion Properties

- For translations multiplcation by a scalar means do more or less of the translation
- For Quaternions the equivalent operation is scaling the angle of rotation
  - Angle, axis = q.getAngleAxis)
  - q.setAngleAxis(t*Angle, axis)

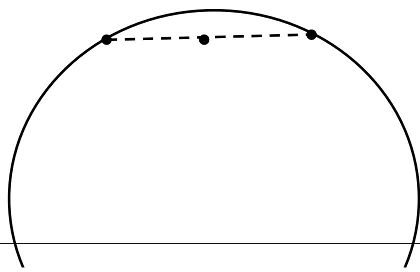
## Interpolating Quaternions

- As quaternions have unit length, they all lie on a sphere with centre on the origin:



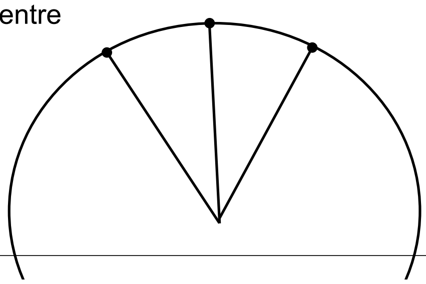
## Interpolating Quaternions

- Interpolating normally will result in a quaternion that is not unit length



## Interpolating Quaternions

- Shomake introduced Spherical Linear Interpolation (SLERP) which interpolates based on the angle at the centre



## SLERP

- Calculate the angle between the 2 (4D) quaternions q1 and q2 ($\theta$)
  - Use 4D dot product
- Interpolate using the sin of $\theta$:

$$q_1 \frac{\sin((1-t)\theta)}{\sin\theta} + q_2 \frac{\sin(t\theta)}{\sin\theta}$$

## SLERP

- The quaterions [*s, **v***] and [-*s*, -***v***] specify the same rotation
- So 2 quaternions on the opposite sides of the hypersphere are the same rotation
- Before doing SLERP we project the 2 quaternions onto the same side
- If $cos\theta < 0$ negate q2

## Interpolating Quaternions

- Shoemake also introduced a method of spline interpolation based on DeCasteljau's method of calculating Bezier splines