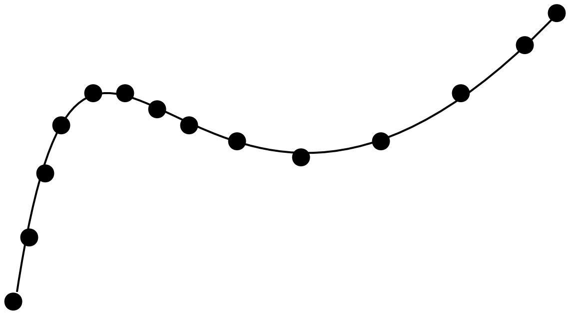
Computer Animation
10-Motion Control
SS 19

Prof. Dr. Charles A. Wüthrich,
Fakultät Medien, Medieninformatik
Bauhaus-Universität Weimar
caw AT medien.uni-weimar.de

Controlling motion along curves

- We all know now how to control the shape of the curve
- To an animator, it is equally important to know the speed at which a curve is traced by increasing parametric steps
- Obviously, since motion curves are of higher order, this relation is not straightforward
- Equal parameter intervals do not lead to arcs of equal length on the curve
- That is, speed is different at different points of the curve
- This can be overcome through a reparametrization of the curve



Computing arc length

- Suppose that we are moving along the curve

$$P(u)=U^TMB$$

- The relation between parameter and arc length is not linear.
- When a unit change in parameter results in a unit change in curve length the curve is said to be *parametrized by arc length*
- How do I establish the relationship between parameter and arc length?
- What we want is to know the function s=G(u) which computes the length of the curve from it starting point for all values of the paramenter u
- If we have G, then knowing G^{-1} allows us to compute the parameter values corresponding to a certain length

Arc length: Estimating through forward differences

- Suppose we have P(u).
- One can compute a table of the distance of P(u) from the point P(0) at regular intervals:

  P(0), P(Δu), P(2Δu),..,P(1)

  that is, containing

  P((i+1)Δu)-P(i Δu)

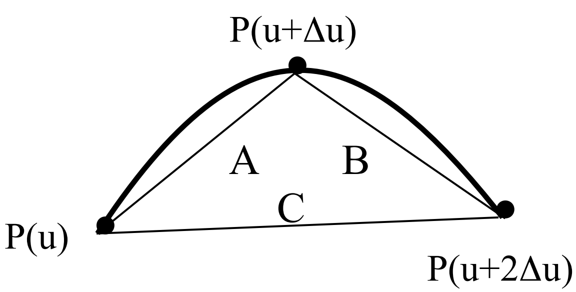
- One can interpolate these values first order (or higher order) to estimate the length of a segment in image space
- Conversely, one can use similar methods to deduce from the right hand column the corresponding value of u
- Main problem with this approach is controlling the error

0	|P(Δu)-P(0)|=G(Δu)
Δu	G(ΔU) +|P(2Δu)-P(Δu)|=G(2 Δu)
2 Δu	G(2ΔU) +|P(3Δu)-P(2Δu)|=G(3 Δu)
...	...

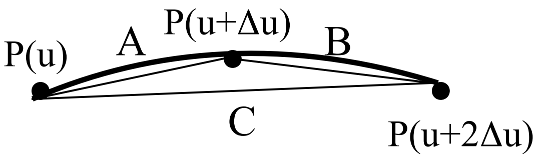
Adaptive forward differences

- Since the relations between the variation of the parameter and the length of the curve is non-linear, the method of the last slide has problems when there is a big error
  - i.e. When the polyline implicitly used to estimate the parameter values inbetwen table points is far from the actual curve
- This can be improved by computing the value of the midpoint of each interval between the table points.
  - if the sum of the sides A+B of the triangle is too different in length from the line joining the interval extremes C (over a threshold value), the midpoint is added to the list

Bad:

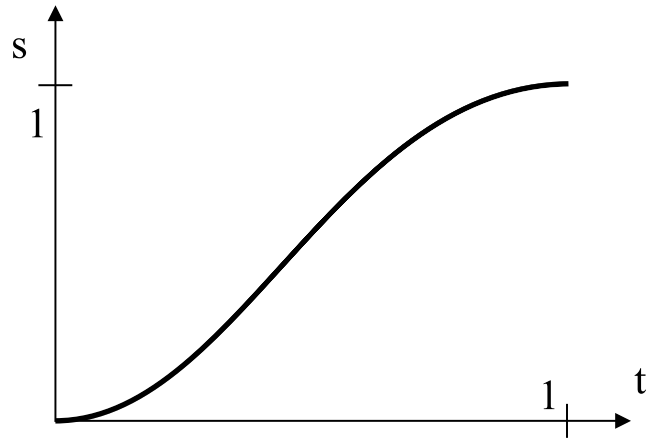


Better:



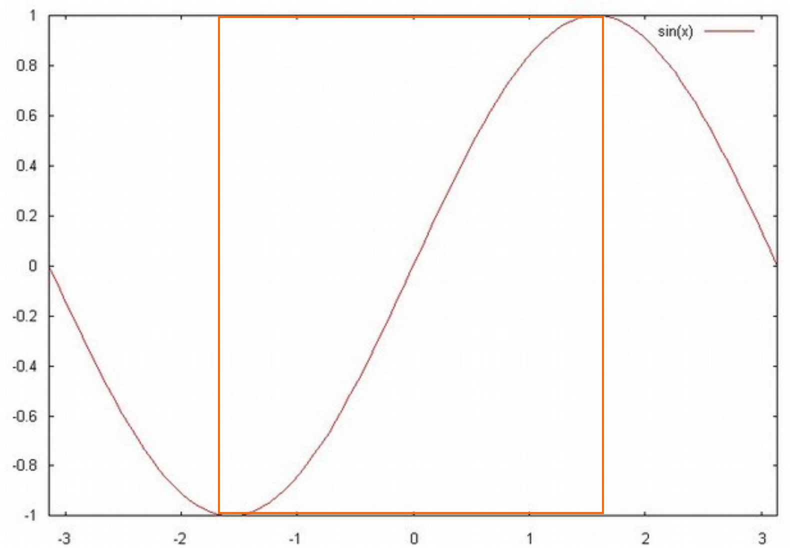
Speed control

- On a arc-length parametrized curve, it is possible to control speed
- Simplest (and dullest) control: constant speed (equal space *s* in equal time *t*)
- Easiest speed control is *ease-in/ease-out*:
  - From standstill, accelerate until maximum speed
  - Decelerates and stop
- Speed along a curve can be controlled by varying arc length at something else than a linear function of *t*.
- The speed variations are seeable in the distance-time curve, which plots the space traversed *s* against the time *t*.
- Here is an example of a distance-time curve for ease-in



Speed control: ease in/ease out

- There are different ways of mathematically achieve ease in/ ease out
- The first one is to use the sinus between $-\pi/2$ and $\pi/2$ and scaling the parameter to cover [0,1]
- $S(t)=(1/2)(\sin(\pi t-\pi/2)+1)$
- This curve can be split and joined with a straight line (take care of continuity at the splits) to add a period of constant speed



Speed control: constant acceleration

- The computational cost of the sinus function is high.
- A better method is to use physics for the calculations: $s=vt$, and $v=at$
- This obtains a parabolic ease-in function thus $s=at^2$
- Similarly for deceleration one can use a constant (limited) deceleration until the object stops
- To describe the distance-time function of such a movement the following equations are used

- In formulas:

$d=½t^2/2t_1 \quad 0<t<t_1$

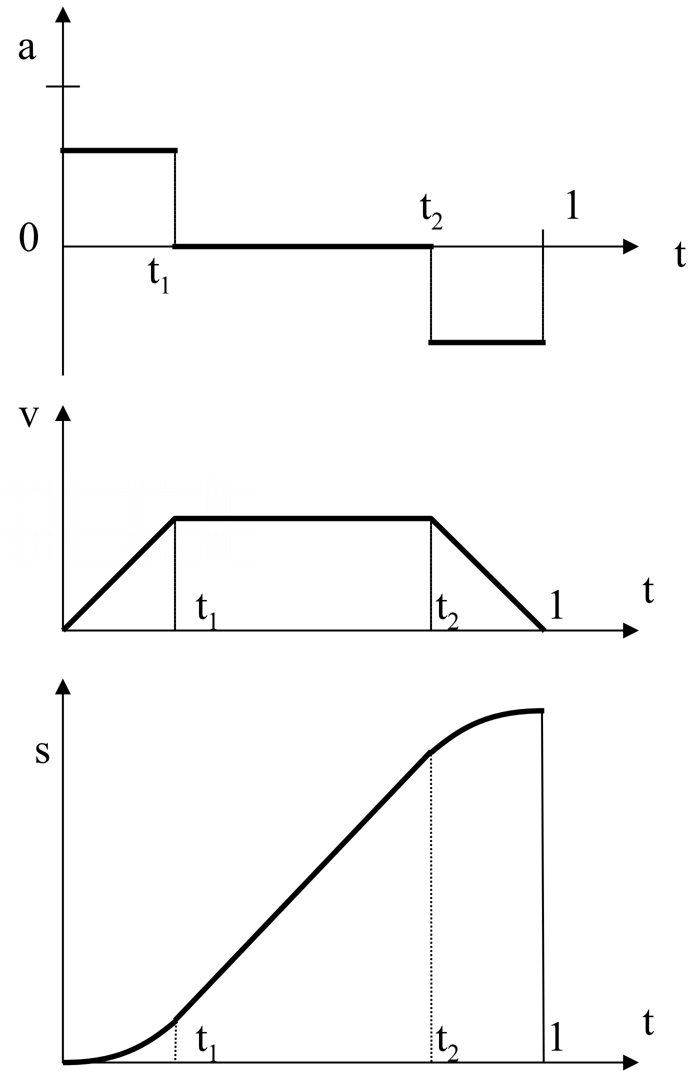
$d=½v_0t_1+v_0(t-t_1) \quad t_1<t<t_2$

$d=½v_0t_1+v_0(t-t_1) + (v_0-½(v_0(t-t_2)/1-t_2)(t-t_2) \quad t_2<t<1$

- Whereby v_0 is the velocity when acceleration ends

Speed control: constant acceleration

- $a=a_0$ $\quad 0<t<t_1$
  $a=0$ $\quad t_1<t<t_2$
  $a=-a_0$ $\quad t_2<t<1$
- $v=v_0 t/t_1$ $\quad 0<t<t_1$
  $v=v_0$ $\quad t_1<t<t_2$
  $a=v_0(1-(t-t_2)/(1-t_2)$ $\quad t_2<t<1$
- The formulas look really complicated, but there are different ways to plot this to make it understandable



General distance-time functions

- Many interesting aspects come up when allowing the user to control motion
- The more influence a user is given, the more problems come up
- Suppose the user defines some velocities at some points:
  - The rest of the velocity curve has to be fitted to these „fixed“ values
  - Sometimes leading to unwanted effects (reverse velocity to fit the time contraints)
- More intuitive is to control on the space-time curve
  - This because it allows to control velocities as a tangent, and to adapt the rest of the curve accordingly
- Motion control often requires specifying positions at specific times
  - The motion is specified as a series of constraints at a specific time, formally, a t-uple $<t_i, s_i, v_i, a_i, \ldots>$
  - higher order approximation is needed for smooth movement

Curve fitting

- If the animator specifies certain constraints then the time parametrized curve can be computed using these constraints as control points
- Suppose contraints are of the form (P_i, t_i) $(i=1,...,j)$
- It only requires to compute the curve passing through these points, i.e.

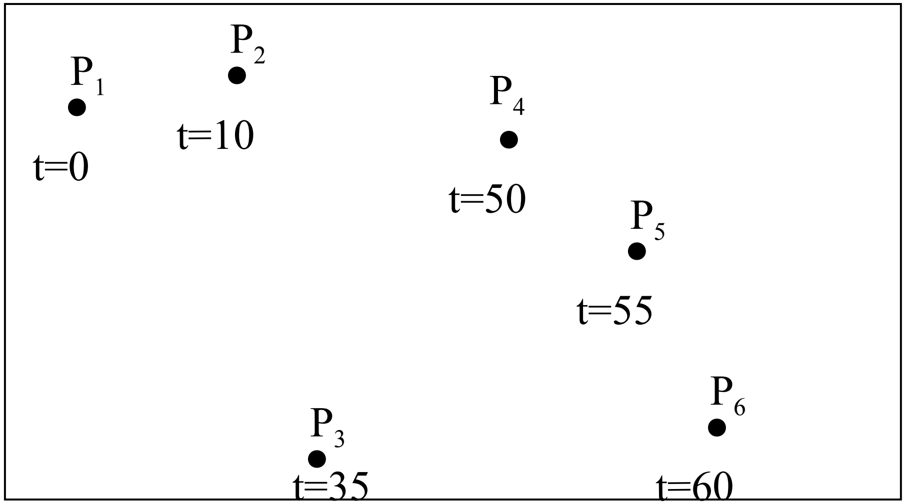
$$P(t)=\Sigma_{1,n} B_i N_{i,k}(t)$$

with $2 <= k <= n+1 <= j$

- In matrix form P=NB
- Inverting this equation leads to find the control point values for the curve

Curve Fitting to position-time pairs

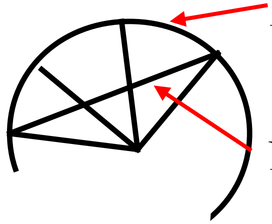
- Suppose the user gives the following positions and the corresponding times
- One can fit a B-spline curve to the values (P_i,t_i) $(i=1,...j)$:
  - That is, take the general eq. of B-splines and make it pass through points
  - Find corresp. control points.



- Computing the curve passing through these points means computing $P(t)=\Sigma_{1,n}B_iN_{i,k}(t)$ with $2 <= k <= n+1 <= j$
- In matrix form P=NB,
- Inverting this equation leads to find the control point values for the curve: $B=N^{-1}P$
- This is done through the pseudoinverse:
  $P=NB$
  $N^TP=N^TNB$
  $[N^TN]^{-1}N^TP=B$
- Remember the tradeoff: the higher the order, the higher the wiggling

Interpolation of quaternion rotations

- A major reason for choosing quaternions is that they can be easily interpolated, i.e. one canget even steps of rotation.
- Quaternion form can be interpolated to produce good intermediate orientations
- This does not work easily with direct interpolation
- Unit quaternions are used to represent orientation, and can be seen as point of on the unit sphere in 4-dimensional space



- To interpolate between two unit quaternions, one can linearly interpolate
- But this will not produce constant speed rotation, because a path on a sphere is not the same as a path on a plane (which is what linear interpol. follows)
- Equal speed interpolations can be computed by interpolating directly on the path on the 4-dimensional sphere

Following a path

- Animating an object to move along a path is quite natural and common
- Not only following the path is needed: also moving the orientation
- Typically, one would have a local coordinate system associated with the object
- Let the coordinates be (u,v,w), and suppose they are right handed
- Suppose the origin of the coordinate system follows the curve P(s), and that the movement of P(s) is specified
- Call POS the current position
- One can view the u,v,w coordinates as a view vector, an up vector and a vector perpendicular to u and v
- This is similar to camera definition in Computer Graphics

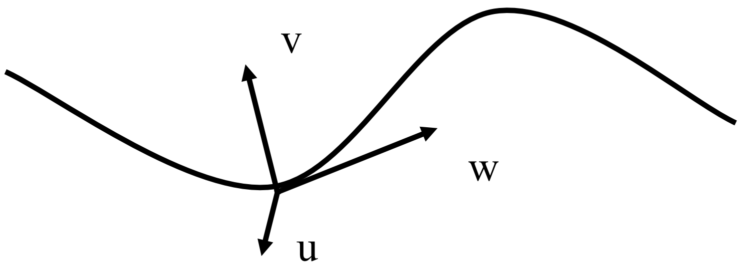
Following a path: Frenet Frame

- The orientation of the camera system can be made dependent from the properties of the curve P(s)
- A Frenet frame is given by the following axes definitions
  - w follows the tangent of the curve (its first derivative P´(s))
  - v is orthogonal to w and in the direction of the second order derivative (P´´(s))
  - u is the cross product of w and v
- In symbols:

  $w=P'(s)$

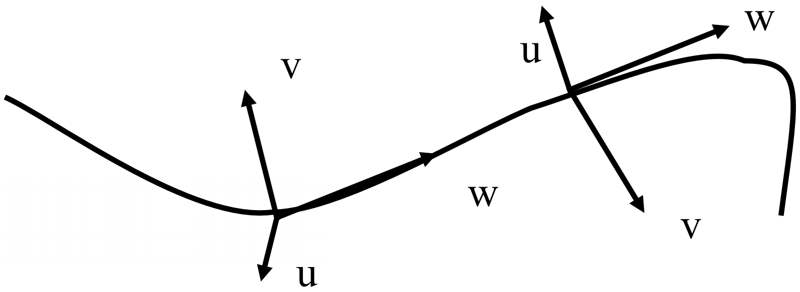
  $u=(P'(s) \times P''(s)$

  $v=w \times u$



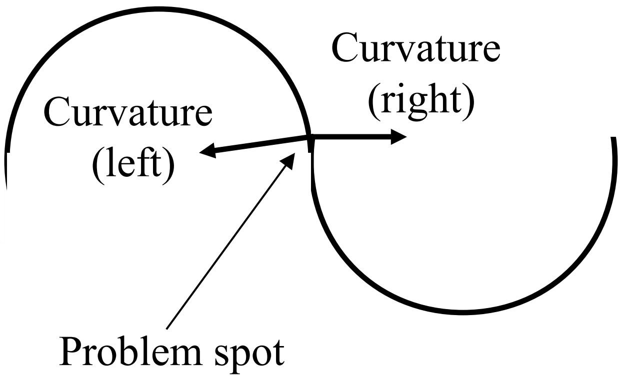
Following a path: Frenet Frame

- Frenet frames are quite nice, but bear some flaws
- When the curve has no curvature, its second order derivative is zero. Here the Frenet frame is undefined
  - This problem can be solved by interpolating the Frenet frames at the start and end of the rectilineal trait
  - Since the tangent vector must be the same at the extremities, it is only a rotation that has to be interpolated



Following a path: Frenet Frame

- A more complicated problem occurs at discontinuities in the curvature vector
- For example, when the path follows first a circle, and then a second circle
- At the problem point, the curvature will switch to pointing from one circle center to the other one
- Here, the Frenet frame is defined everywhere but is discontinuous
- Here, the object will rotate wildly along the path with „instant switches "



Following a path: Frenet Frame

- The worst problem is that the path following is not so natural:
  - when we view at something, we we do not look along the tangent
  - When we move, we anticipate curves
- Similar effect to your car light not following the road
- Also, one might want to make the object bend towards the interior to „anticipate the force“
- .... or, opposite, to let it bend out to give the effect of a force acting on the object

Camera Path Following: Center of Interest

- A more natural way of specifying the orientation of a camera is to use the center of interest (COI)
  - One can view towards a fixed point
  - Or alternatively the center of an object
- Good method for a camera circling some arena of action
- The center of interest is specified, and so the view vector w=COI-POS
- This leaves one degree of freedom in camera specification
- One simple way is to set the view vector v as viewing „up“, i.e. perpendicular to w and lying in the *wy* plane
  w=COI-POS
  $u = w \times y$
  $v = u \times w$
- This works quite well for a camera moving along a path and focussing to a single object.
- When it gets very close to the object, this results in drastic changes (fly-near effect)
- This is not always bad!!!

Camera Path Following: Center of Interest

- There are variations to specifying a fixed point
- One can for example specify various points on the camera path itself
- The up vector
  - is usually specified as lying in the *wy* plane
- But one can also allow the user to input
  - Either a tilting value with respect to the default up vector
  - Or the up vector on a whole
- Following a points on the path is relatively easy:
  - If P(s) describes the position on the curve, then P(s+δs), with δs >0, specifies its position in the future
  - It is advisable to choose points at equidistances on the curve, so as to make changes not that noticeable
  - Alternatively, one can take the baricenter of some future points to avoid too much hopping
- The real flaw of this method is the fact that camera views look jerky

Camera Path Following: Center of Interest

- A better method is to use instead of some function of the position path, a different function altogether for the POI
- Let P(s) be the curve of the camera path, and C(s) the curve of the COI (obviously the animator specifies this)
- Similarly, and up vector path must be specified U(s), so that the general up direction is U(s)-P(s)
- The resulting coordinates for the camera will then become

  $w = C(s) - P(s)$
  $u = w \times (U(s) - P(s))$
  $v = u \times w$

- This gives maximum control, but is also difficult to control.
- An easy way of specifying C(s) is to use fixed positions, with ease-in/ease-out moves between the different fixed points

Smoothing a path

- Sometimes the path I want to move along is obtained through a caption device.
- In this case, path data is often inexact, and there are errors in it.
- There are several ways to smooth a path if it has been generated by a sample process, such as a motion capturing system
- Motion capture is increasily used since it is becoming inexpensive: a Kinekt for the playstation costs around 150$.
- However, data here can be prone to noise or imprecision, depending on the input method

Courtesy Animazoo Ltd.

Smoothing paths: linear interpolation

- The simplest way of smoothing the data is to average neighbouring data point.
- Suppose we have the chain of points $\{P_i\}_{i=0,N}$
- In the simplest form, one averages P_i as the average itself and of P_{i-1} and P_{i+1}.

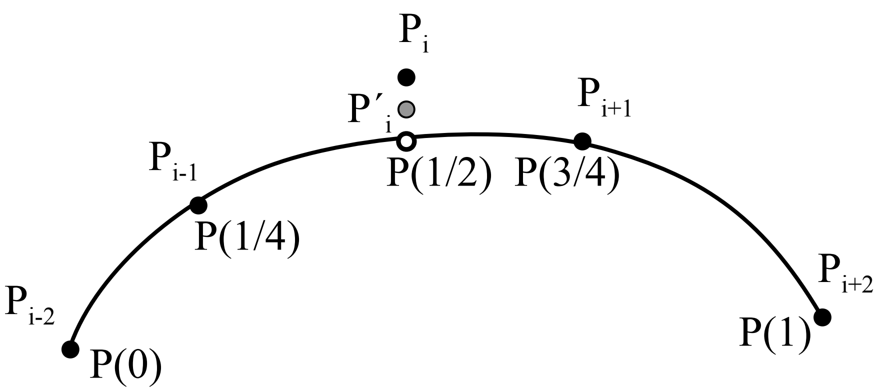
$$P'_i = \frac{P_i + \frac{P_{i-1} + P_{i+1}}{2}}{2} = \frac{1}{4} P_{i-1} + \frac{1}{2} P_i + \frac{1}{4} P_{i+1}$$

– Obviously, here the „spikes“ are flattened, so applying this method many times makes little sense

Smoothing paths: cubic interpolation

- A second method use the four adjacent points $P_{i-2}, P_{i-1}, P_{i+1}, P_{i+2}$ on either side to fit a cubic curve that is then evaluated at the midpoint.
- This midpoint is averaged with the original point to obtain the smoothed point
- Remembering that a 3rd order curve was $P(u)=au^3+bu^2+cu+d$

- One obtains
  $P_{i-2}=P(0)=d$
  $P_{i-1}=P(1/4)=$
  $a(1/64)+b(1/16)+c/4+d$
  $P_{i+1}=P(3/4)=$
  $a(27/64)+b(9/16)+3c/4+d$
  $P_{i+2}=P(1)=a+b+c+d$



Smoothing paths: cubic interpolation

- For the last points, a parabolic arc can be computed to fit the second and forelast points
- Notice that here the curve will be of the form au^2+bu+c , and the equation turns into

$$P'_1=P_2+1/3(P_0-P_3)$$

  and similarly for the last three points

Smoothing paths: B-spline approximation

- If the path does not necessarily have to pass through the sample points, one can use approximation methods we saw before
- Particularly B-splines are well adapted for the defining a path tacked from real data

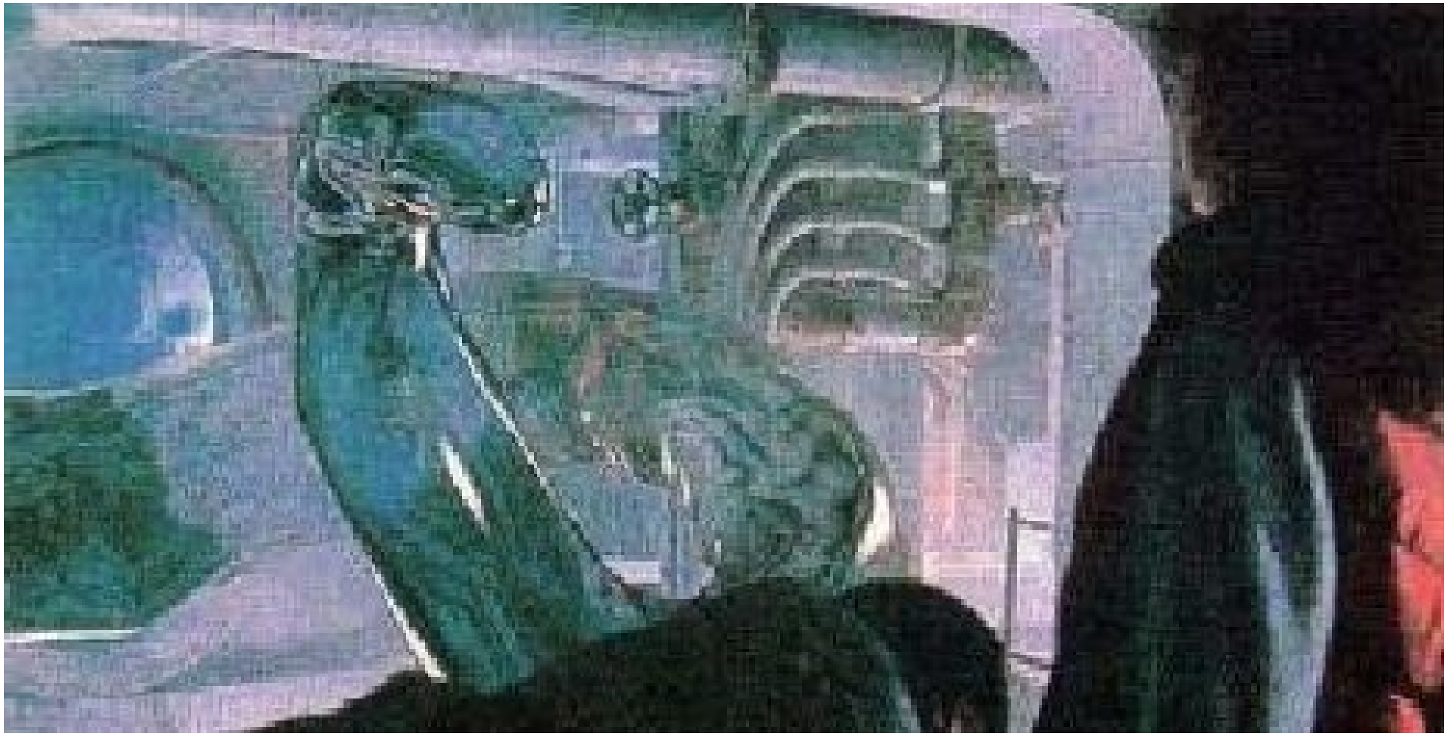
Animation languages

- In recent times, scripting languages have been developed to support animation systems
- Most animation languages are not easy to understand, and are close to hardcore programming
- A typical animation language is Renderman, or Alias/wavefront's MEL
- Their big advantage is control

Animation languages

- Some effort has been put to accomodate unskilled artistic animators without scripting capabilities
- Simpler scripting languages such as ANIMA II have been developed
- Recently, actor based languages have appeared
- This is a novel approach but still at its infancy
- The idea is to have objects (=actors) and the instantiation of their variables representing the moving parameters
- Finally, the development of avatars has generated the need for some form of interaction with the animated models.

End



+++ Ende - The end - Finis - Fin - Fine +++ Ende - The end - Finis - Fin - Fine +++