Computer Animation
11-Kinematics and Physics
SS 19

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

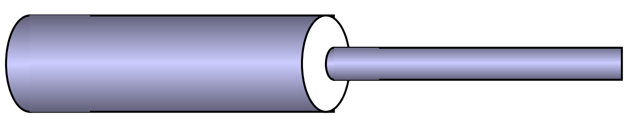
Hierarchical modeling

- Hierarchical modeling is placing constraints on objects organized in a tree like structure
- Examples can be:
  - A planet system
  - A robot arm
- The latter is quite common in graphics: it is constituted by objects connected end to end to form a multibody jointed chain
- These are called *articulated figures*

- They stem from robotics
- Robotics literature speaks with a different terminology:
  - Manipulator: the sequence of objects connected by joints
  - Links: the rigid objects making the chain
  - Effector: the free end of the chain
  - Frame: local coordinate system associated to each link

Hierarchical modeling

- In graphics, most of the links are revolute joints: here one link rotates around a fixed point of the other link
- The other interesting joint for graphics is the prismatic joint, where one link translates relative to the other
- Joints restrain the degree of freedom (DOF) of the links
- Joints with more than one degree of freedom are called *complex*
- Typically, when a joint has n>1 DOF it is modeled as a set of n one degree of freedom joints



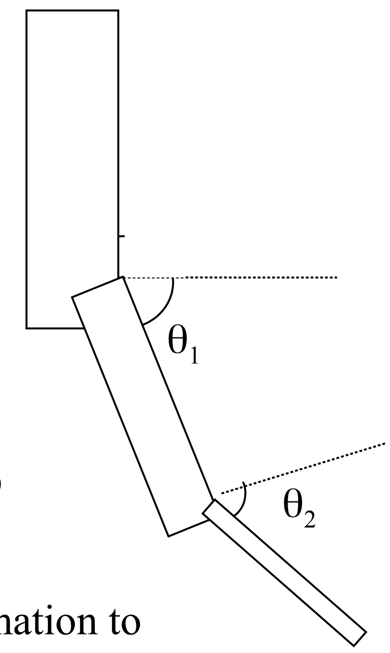
Hierarchical modeling

- Humans and animals can be modeled as hierarchical linkages
- These are represented as a tree structure of nodes connected by arcs
- The highest node of this structure is called the root node, and is the node that has position WRT the global coordinate system
- All other nodes have their position only as relative to the root node
- A node that has no child is called a leaf node
- Each node contains the info necessary to define the position of the corresponding part
- Two types of transformations are associated with an arc leading to a node:
  - Rotation and translation of the object to its position of attachment to the father link
  - Information responsible for the joint articulation

Hierarchical modeling

- How does this work?
- The idea is simple, store at each node
  - Info on the node geometry
  - The transformation (its rotation) with respect to the father node in the tree
- To obtain the position of the i-th node in the chain, one has to simply multiply the transformations to obtain the position of the current arc to be displayed
- The root node of course contains info of its absolute position and orientation in the global coord. system

T_0: transformation to rotate K_0 in WCS



T_1: transformation to rotate K_1 WRT K_0 = rotation by θ_1

T_2: transformation to rotate K_2 WRT K_1 = rotation by θ_2

- To obtain the position of K_2 in WCS, one will then have to multiply $T_0T_1T_2$

Forward kinematics

- Traversing the tree of the nodes produces the correct picture of the object
- Traversal is done depth first until a leaf is met
- Once the corresponding arc is evaluated, the tree is backtracked up until the first unexplored node is met
- This is repeated until there are no nodes left inexplored
- A stack of transforms is kept
- When tree is traversed down-wards, the corresponding trans-formation is added to the stack
- Moving up pops the transformation from the stack
- Current node position is generated through multiplying the current stack transforms

Forward kinematics

- To animate the whole, the rotation parameters are manipulated and the corresponding transforms are actualized
- A complete set of rotations on the whole arcs is called a *pose*
- A pose is obviously a vector of rotations

- Moving an object by positioning all its single arcs manually is called forward kinematics
- This is not so user-friendly
- Instead of specifying the whole links, the animator might want to specify the end position of the effector
- The computer computes then the position of the other links
- This is called *inverse kinematics*

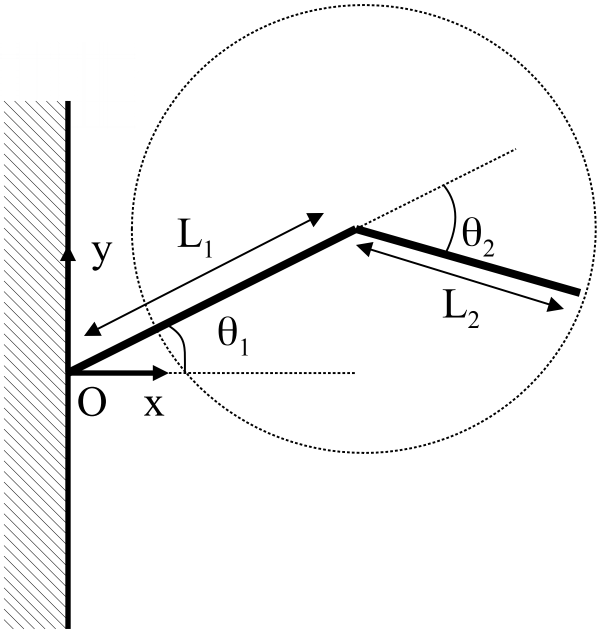
Inverse kinematics

- The user gives the position of the end effector and the computer computes the joint angles
- One can have zero, one or multiple solutions
  - No solution: overconstrained problem
  - Multiple solutions: underconstrained problem
  - Reachable workspace: volume that end effector can reach
  - Dextrous workspace: volume that end effector can reach in any orientation
- Computing the solution to the problem can at times be tricky
- If the mechanism is simple enough, then the solution can be computed analytically
- Given an initial and a final pose vector, the solution can be computed by interpolating the values of the pose vector
- If the solution cannot be computed analytically, then there is a method based on the jacobian to compute incrementally a solution

Inverse kinematics

- Consider the figure: the 2nd arm rotates aroound the end of the 1st arm.
- It is clear that all positions between $|L_1-L_2|$ and $|L_1+L_2|$ can be reached by the arm.
- Set the origin like in the drawing
- In inverse kinematics, the user gives the (X,Y) position of the end effector

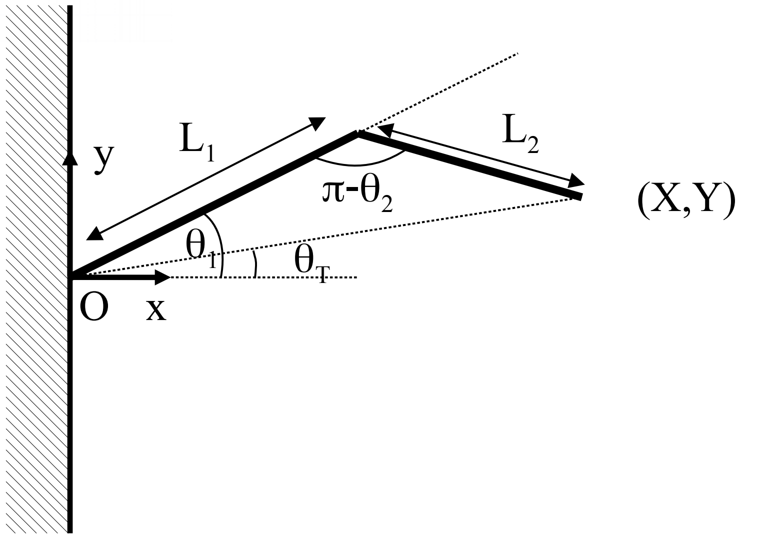
- Obviously there are only solutions if $|L_1-L_2| \leq \sqrt{X^2+Y^2} \leq |L_1+L_2|$



Inverse kinematics

- $\cos\theta_T = X/(X^2+Y^2)^{½}$
  $\Rightarrow \theta_T = \mathrm{acos}(X/(X^2+Y^2)^{½})$
- Because of the cosine rule we have also that
  $\cos(\theta_1-\theta_T)=$
  $(L_1^2+X^2+Y^2-L_2^2)/2L_1(X^2+Y^2)^{½}$
  and
  $\cos(\pi-\theta_2)=$
  $(L_1^2+L_2^2-(X^2+Y^2))/2L_1L_2$
  from which we have
  $\theta_1=\mathrm{acos}((L_1^2+X^2+Y^2-L_2^2)$
  $/2L_1Ö(X^2+Y^2)^{½})+\theta_T$
  and
  $\theta_2=\mathrm{acos}((L_1^2+L_2^2-(X^2+Y^2))/2L_1L_2)$

- Note that two solutions are possible, simmetric with respect to the line joining the origin and (X,Y)



Inverse kinematics

- In general, for the quite simple armatures used in robotics it is possible to implement such analytic solutions
- Unfortunately this works only for simple cases
- For more complicated armatures, the number of possible solutions there may be infinite solutions for a given effector location, and computations become so difficult to do that iterative numeric solution must be used

Using the Jacobian

- When the solution is not analytically computable, incremental methods converging to the solution are used
- To do this, the matrix of the partial derivatives has to be computed
- This is called the *Jacobian*

- Suppose you have six independent variables and you have a six unknowns that are functions of these variables

  $y_1=f_1(x_1,x_2,x_3,x_4,x_5,x_6)$
  $y_2=f_2(x_1,x_2,x_3,x_4,x_5,x_6)$
  $y_3=f_3(x_1,x_2,x_3,x_4,x_5,x_6)$
  $y_4=f_4(x_1,x_2,x_3,x_4,x_5,x_6)$
  $y_5=f_5(x_1,x_2,x_3,x_4,x_5,x_6)$
  $y_6=f_6(x_1,x_2,x_3,x_4,x_5,x_6)$

  or, in vector notation,

  $\underline{Y}=\underline{F}(\underline{X})$

Using the Jacobian

- What happens when the input variables change?
- The equations can be written in differential form:
  $\delta y_i = \partial f_i/\partial x_1\ \delta x_1 + \partial f_i/\partial x_2\ \delta x_2$
  $+ \partial f_i/\partial x_3\ \delta x_3 + \partial f_i/\partial x_4\ \delta x_4$
  $+ \partial f_i/\partial x_5\ \delta x_5 + \partial f_i/\partial x_6\ \delta x_6$
  or, in vector form
  $\delta \underline{Y} = \partial \underline{F}/\partial \underline{X}\ \delta \underline{X}$
- Given n equations in n variables, the matrix

$$J = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_1} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix}$$

  is called the Jacobian matrix of the system
- The Jacobian can be seen as a mapping of the velocities of $\underline{X}$ to velocities of $\underline{Y}$

Summary: articulated bodies

- Very useful for enforcing certain relationships among elements of an animation
- Allows animator to concentrate on effector forgetting the rest of the body
- Damn hard to do, to date not real in real time
- Adding control expressions can be tricky
- No physics considered. Only kinematics

Physics in animation

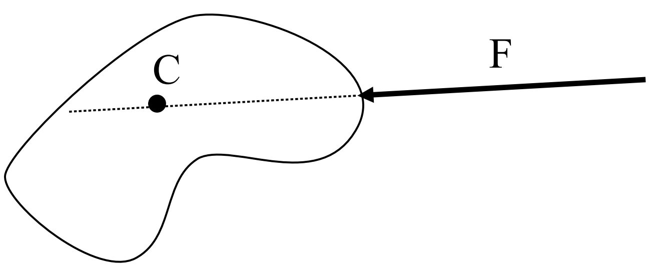
- While animating through kinematics may be interesting for plenty of applications, integrating physics is more difficult and a challenging problem
- The easiest way of integrating physics is rigid body simulation
- While physics is concerned with the exactness of the representation, animation is more interested in „credible“ effects, and in rendering frame by frame
- Having to deal with the system at discrete time samples creates numerical problems in the solution methods which are not simple to deal with

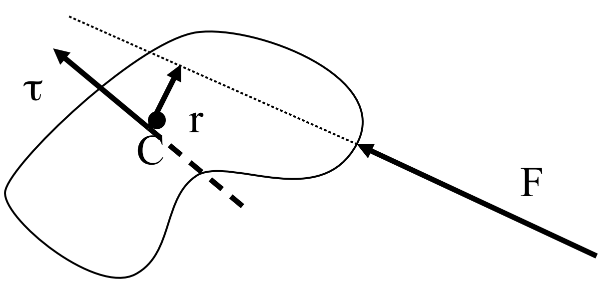
Recap on physics (physics 101)

- In the equations of motion, the following quantities play a role
  - Distance=speed · time
    time= frame# ·timeperframe
    averageVelocity= distancetraveled/time
- Linear motion
  - s=position
    v=velocity
    a=acceleration
  - $s(t)=v(t) \cdot t$
    $v(t)=a(t) \cdot t$
    $s(t)= ½ a(t) \cdot t^2$
- Circular motion
  - θ angular position
    ω angular velocity
    $\theta(t)= \omega(t) \cdot t$
  - For a body in circular motion, we have
    $a(t)=(-\omega)^2 \cdot r$
- Newton's law:
  - $\underline{F} = m \cdot \underline{a}$
  - A body continues its own motion therefore if the sum of the forces acting on it =0
    $\Sigma F_i=0$

Recap on physics (physics 101)

- Remember the definition of center of mass
  - The point at which the object is balanced in all directions
  - If an external force is applied to a body in line with its center of mass, then the body would move as if it was a point at the center of mass C
- Torque is the tendency of a force to produce circular motion
  - It is produced by a force off center to the center of mass
  - $\tau = r \times F$
  - Clearly, $\tau \perp F$ and $\tau \perp r$
- An object does not move if $\Sigma F_i = 0$ and $\Sigma \tau_i = 0$





Recap on physics (physics 101)

- Linear springs:
  - Hooke's law:
    $F= -k \cdot x,$
    where x is the change from the equilibrium length of the spring
- Friction:
  - Static:
    $F_s = s \cdot f_N$
    where F_s = frictional force
    s=static friction coefficient
    f_N = normal force
  - Kinetic:
    $F_k = k \cdot f_N$
    with similar coefficient definitions as in static friction
- Momentum: $m \cdot v$
  - In a closed system, total momentum does not vary
- Angular momentum:
  $L=r \times p$
  where
  r=vector from center of rotation
  p=momentum ($m \cdot v$)
  - Note that
    $\tau= dL/dt$
  - In a closed system, total angular momentm does not vary
- Inertia tensor: the resistence of an object to change its angular momentum

Rigid body simulation

- If one wants to simulate rigid bodies, many forces act on them
- Such forces vary in time conti-nuously and in a non linear way
- Therefore it is not enough to evaluate velocities and acce-lerations at fixed timesteps Δt
- Evaluating the velocities at t_0, $t_0+\Delta t$, $t_0+2\Delta t$ does not generate a correct movement, and slowly drifts away from the correct solution
- This solution method is an example of the Euler integration method
- The accuracy of the method is determined by the size of the time step
- Obviously the shorter the time step, the more computations are needed
- A better way of integrating the equations bases on the Runge Kutta method
  - In particular, often 2nd order Runge Kutta (midpoint method) is used
  - Remember, the order of the RK method is the magnitude of the error term
  - Even higher order ones, 4th or 5th ones are used

Motion equations for a rigid body

- To develop the equation of motion for a rigid body, we have to apply some of the physics presented before
- When a force is applied to a rigid body, the force and the relative torque are applied to the body
- To uniquely solve for the resulting motions of interacting bodies, linear and angular momentum have to be conserved
- Finally, to calculate the angular momentum the distribution of an object mass in space has to be characterized with its inertia tensor.

Orientation and rotational movement

- Similar to position, velocity and acceleration, 3D objects have
  - orientation,
  - angular velocity and
  - angular acceleration
- which vary in time
- Let R(t) represent the object rotation
- *Angular velocity* $\underline{\omega}(t)$ is the rate at which the object is rotated (independent from linear velocity)
- The direction of $\underline{\omega}(t)$ indicates the orientation of the axis about which the object is rotating
- The magnitude of $\underline{\omega}(t)$ gives the speed of rotation in revs per unit time

Center of mass

- The center of mass of a body is defined as the integral of the differential mass times its position in the object
- In a body with discrete masses, then the center of mass is *at* $q_i(t)$, the center of mass is at $x(t)=\Sigma m_i q_i(t)/\Sigma m_i$

Forces and torque

- A linear force applied to a mass gives rise to a linear acceleration

  $$F=ma \qquad \text{(Newton's law)}$$

- The various forces applied to a point sum up

  $$F(t)=\Sigma f_i(t)$$

- The torque arising from the application of forces acting on a point of an object is given by

  $$\tau_i(t)=(q(t)-x(t)) \times f_i(t)$$

  $$\tau(t)=\Sigma\tau_i(t)$$

Momentum

- The momentum of an object (= mass x velocity) is decomposed into
  - linear component: acts on center of mass
  - angular components: acts WRT center
- Both are preserved in a closed system
- Linear momentum p=m v
- Total linear momentum of a rigid body: $P(t)=\Sigma m_i \mathring{q}_i(t)$
- Deriving p=mv we obtain
  $\mathring{P}(t)=M\, v°(t)=F(t)$
- Angular momentum is a measure of the rotating mass weighted by the mass's distance from the axis of rotation
- $L(t)=$
  $\Sigma((q(t)-x(t)\times m_i(q°(t)-v(t)))$
  $=\Sigma(R(t)q\times m_i(\omega(t)\times(q(t)-x(t))))$
  $=\Sigma(m_i(r(t)q\times(\omega(\tau)\times R(t)q)))$
- Similar to linear momentum, torque equals the change in angular momentum
  $L°(t)=\tau(t)$
- Note that since angular momentum depends on distance to center of mass, to mantain constant angular momentum, the angular velocity increases if the distance of the mass decreases

Inertia tensor

- Angular momentum is related to angular velocity the same way linear momentum is related to linear velocity $P(t)=M\cdot v(t)$
- We have
  $L(t)=I(t)\cdot\omega(t)$
- The distrib. of mass of the obj. in space is defined through a matrix, the inertia tensor I(t)

$$I_{obj} = \begin{bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{yx} & I_{yy} & I_{yz} \\ I_{zx} & I_{zy} & I_{zz} \end{bmatrix}$$

  where the matrix terms are computed by integrating over the object, and I is symmetric
- In general, $I_{xx}=\iiint\rho(q)(q_y^2+q_z^2)dxdydz$ where ρ is the density of at an obj. point $q=(q_x,q_y,q_z)$
- In the case of discrete masses
  $I_{xx}=\Sigma m_i(y_i^2+z_i^2)$, $I_{xy}=\Sigma m_i x_i y_i$
  $I_{yy}=\Sigma m_i(x_i^2+z_i^2)$, $I_{xz}=\Sigma m_i x_i z_i$
  $I_{zz}=\Sigma m_i(x_i^2+y_i^2)$, $I_{yz}=\Sigma m_i y_i z_i$
- In a center of mass centered obj space, the intertia tensor of a transformed object depends on the obj orientation but not on its position, and therefore it depends on time
- It can be transformed with
  $I(t)=R(t)I_{obj}R(t)^T$

Motion equations

- The state of an object can be determined by the vector containing
  - Position
  - Orientation
  - Linear momentum
  - Angular momentum

$$S(t) = \begin{bmatrix} x(t) \\ R(t) \\ P(t) \\ L(t) \end{bmatrix}$$

- Object mass and its object space inertia tensor I_{obj} do not change in time
- At any time, the following quantities can be computed:
  Inertia tensor $I(t)=R(t)I_{obj}R(t)^T$
  Angular vel. $\omega(t)=I(t)^{-1}L(t)$
  Linear vel. $v(t)=P(t)/M$
- Now the time derivative can be formed:

$$\frac{d}{dt}S(t) = \frac{d}{dt}\begin{bmatrix} x(t) \\ R(t) \\ P(t) \\ L(t) \end{bmatrix} = \begin{bmatrix} v(t) \\ \omega(t)^{*} R(t) \\ F(t) \\ \tau(t) \end{bmatrix}$$

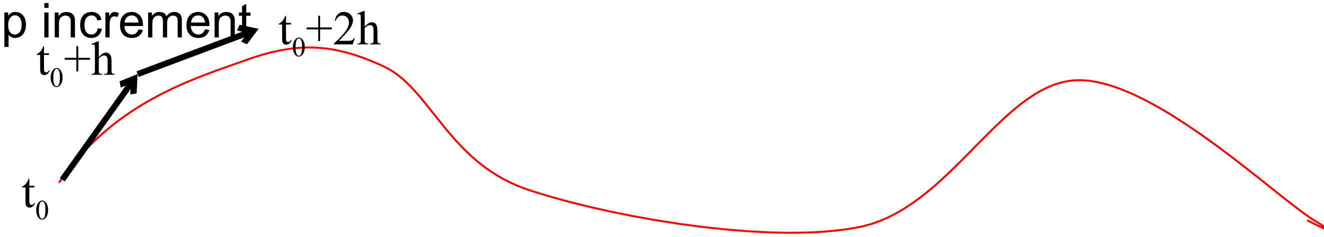
  This is enough to run a simulation
- A differential equation solver can now be used

Motion equations

- As the simplest solver, one can use Euler‘s method
  - The values of the state array are updated by multiplying their time derivatives by the length of the time step
- Euler methods assume that the solution at the time point t_0+h will can be approximated by the solution at t_0 plus the stepsize h multiplied by the derivative at the time t_0:

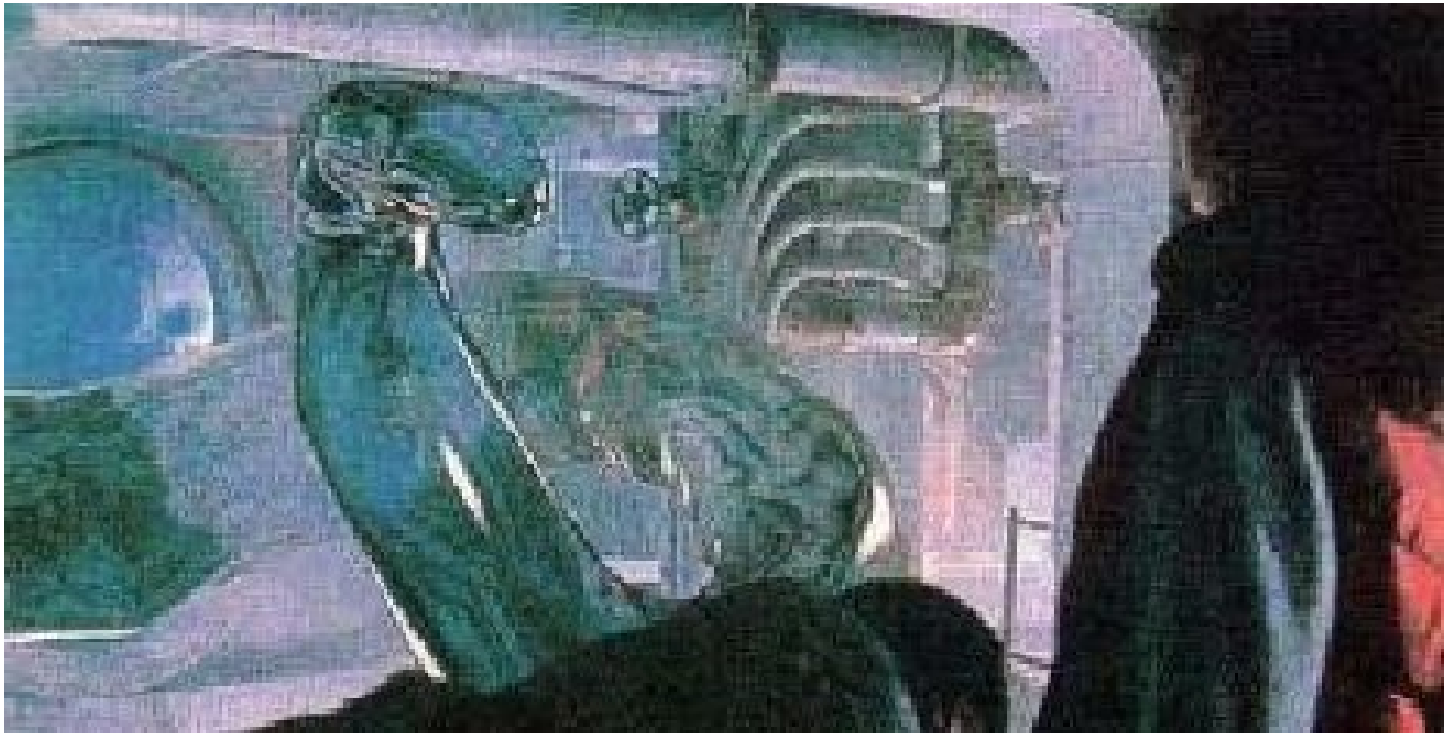
$$y(t_0+h) \approx y(t_0)+hf'(x(t_0),y(t_0))$$

- This is equivalent to saying: for a little while (h), approximating my movement with the tangent to the solution at the previous step is okay.
- Or in other words, since the derivative of space is velocity, it is the same as approximating the movement for a little while with the velocity times the timestep increment



- As long as the timestep h id small the resulting solutions are correct.
- There are, of course, more refined methods for solving differential equations, but explaining them would be beyond the purpose of this course.

End



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