Computer Graphics

Lecture 12: animation

Ágoston Sipos
siposagoston@inf.elte.hu

Eötvös Loránd University
Faculty of Informatics

2025-2026. Spring semester

Table of contents

Animation

- Our scenes are rarely static, especially in case of interactive application
- Animation: a sequence of images
- If they follow each other quickly enough, a sense of continuous movement develops (see Critical Flicker Frequency earlier)

Traditional animation methods

- Each frame is drawn by hand.
  - We can precisely control everything
  - Extremely labor intensive (and expensive)
- Cel animation
  - Use of layers, some parts of the color space on reusable cels
  - Keyframes are drawn by the main draftsmen, the junior artists are responsible for the transitions (inbetween) (in accordance with the instructions given)
  - But: if everything is moving in the background, this is also incredibly labor-intensive (and expensive: e.g. The Wings of Honneamise)...


Image from Krtek a zápalky
© Zdeněk Miler / Krátký Film Praha 1974


Image from Krtek a zápalky
© Zdeněk Miler / Krátký Film Praha 1974

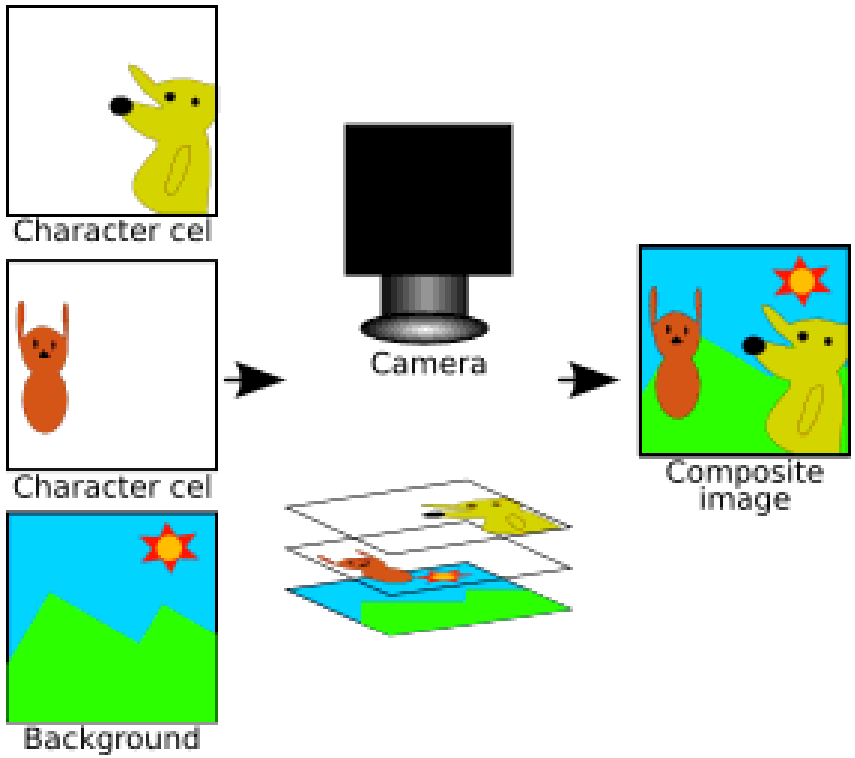

Image from Krtek a zápalky
© Zdeněk Miler / Krátký Film Praha 1974


Image from Krtek a zápalky
© Zdeněk Miler / Krátký Film Praha 1974


Image from Krtek a zápalky
© Zdeněk Miler / Krátký Film Praha 1974


Image from Krtek a zápalky
© Zdeněk Miler / Krátký Film Praha 1974

Traditional animation methods



More interesting facts

- "Principles of Traditional Animation Applied to 3D Computer", SIGGRAPH'87, pp. 35-44.
- Multiplane Camera
  `https://www.youtube.com/watch?v=YdHTlUGN1zw`

Computer-assisted animation

- Keyframe based animation
  - The intermediate animation phases are automated: "Only" the keyframe needs to be created and the parameters for the desired transition
  - Flexible, less labor-intensive but still requires serious training
- Procedural animation
  - Algorithmic description of motion and changes
  - For example, the height of a bouncing ball as a function of time (t): $m(t) = |\sin(\omega t + \theta_0)|e^{-kt}$

Computer-assisted animation

- Physics based animation
  - We provide the objects of our scene with physical properties (mass, forces, flexibility, etc.)
  - Computer animation based on the solutions of equations derived from physical laws.
  - Realistic (if done right) but difficult to control
- Motion capture
  - Using instruments to record someone/something performs the desired movements and then transferring this to a digital model



What can we animate?

- Essentially, anything that affects the displayed image of the scene (position, orientation, color of models, properties related to representation, BRDF, etc.)
- We primarily look at model and camera transformations now
- Our task is to make these parameters dependent on time

Animation synthesis

- For each object o, let the model (world) transformation be $M_o \in \mathbb{R}^{4\times4}$
- Let the view (camera) transformation be: $V \in \mathbb{R}^{4\times4}$
- *Remark: if V is the View matrix, then it only contains the position and orientation of the camera; if V is View and Projection combined, then we can also adjust the field of view (angle)*
- Let both matrices be a function of time!
- $M_o \in \mathbb{R} \to \mathbb{R}^{4\times4}$
- $V \in \mathbb{R} \to \mathbb{R}^{4\times4}$

Outline of a generic animation program

```
while keep_running:
    t = get_time()
    for o in objects:
        M_o = M_o(t)
    V = V(t)
    render_scene()
```

Animation synthesis

- Realtime/interactive:
  - we display the frames immediately
  - the calculation must be fast enough to give the appearance of continuity
  - needs to react to user inputs
  - $\Rightarrow$ the scene can be so detailed that it can still be displayed this way
  - $\Rightarrow$ we use incremental image synthesis

Outline of a realtime animation program

```
def update():
  # FrameUpdate() / UpdateScene()
  t = get_time()
  for o in objects:
    M_o = M_o(t)
  V = V(t)

def render():
  # Render() / DrawScene()
  for o in objects:
    render_object(o)
```

Animation synthesis

- Not realtime/*offline*:
  - It "doesn't matter" how long it takes to calculate a frame
  - Synthesis and playback are separated
    - First we save the frames
    - Then it can be played back as a video
  - $\Rightarrow$ the user can't affect the animation
  - $\Rightarrow$ we use as detailed a scene and as complicated algorithms as we want, as long as we have the patience
  - However, the budget sets bounds: Final Fantasy: Spirits Within (2001), was a 4-year work of a team of 200 people (approx. 120 man-years in total), for the production of which 960 workstations were used. The film consisted of 141964 frames in the end (15TB), one frame took an average of 90 minutes to render!

Offline rendering

```
# render
Δt = 1/FPS
for (t = t_start; t < t_end; t += Δt):
    for o in objects:
        M_o = M_o(t)
    V = V(t)
    render_scene_to_disk()

# display
for (t = t_start; t < t_end; t += Δt):
    draw_frame(t)
    wait(Δt)
```

Realtime animation – incorrectly

- What's the easiest way to mess up the animation calculation?
- By not considering how much time has passed between two frames.
- E.g.: We want to rotate the object around its center with a constant angular velocity.

```
model = rotate(phi, 0,1,0); phi += phi_step;
```

- The object will rotate as fast as the frequency with which this code snippet is called.
- More times on a faster machine, less often on a slower machine – and nothing guarantees that the same amount of time would pass between two calls on the same machine!

Realtime animation – correctly

- What's the easiest way to prevent it?
- We never store how much to change, but what the *speed* of the change is.
- Before each calculation, we query how much time has passed since the previous frame and multiply the speed by this.
- E.g.: Instead of `phi += phi_step;` we use `phi += get_time_since_last_frame() * phi_speed;`
- On a faster machine, less time passes between two frames, on a slower machine, more time passes between two frames $\Rightarrow$ on a fast machine, we move with smaller steps, on a slower machine with bigger steps – and it doesn't matter if the time between frames is not consistent

Camera animation

- Camera properties:
  - camera position (*eye*),
  - a point to look at (*center*),
  - vector of the upwards direction (*up*),
  - ratio of the screen/window sides (*aspect*),
  - field of view (*fovy*).
- These can all be changed individually to create the animation.
- For example, the eye position can be specified with the parametric curves seen in previous lessons, where the parameter is time! E.g. from $\mathbf{a}$ into $\mathbf{b}$ in 5 seconds $\mathbf{p}(t) = (1 - \frac{t}{5})\mathbf{a} + \frac{t}{5}\mathbf{b}$, if t is in seconds and we start with $t = 0$ from $\mathbf{a}$.

Position and orientation

- Position: *"Where is the object?"*
- Orientation: *"What direction is the object facing?"*
- We mainly want to change these two. (Rigid-body transformations)
- $M_o(t)$ gives both.
- Normally

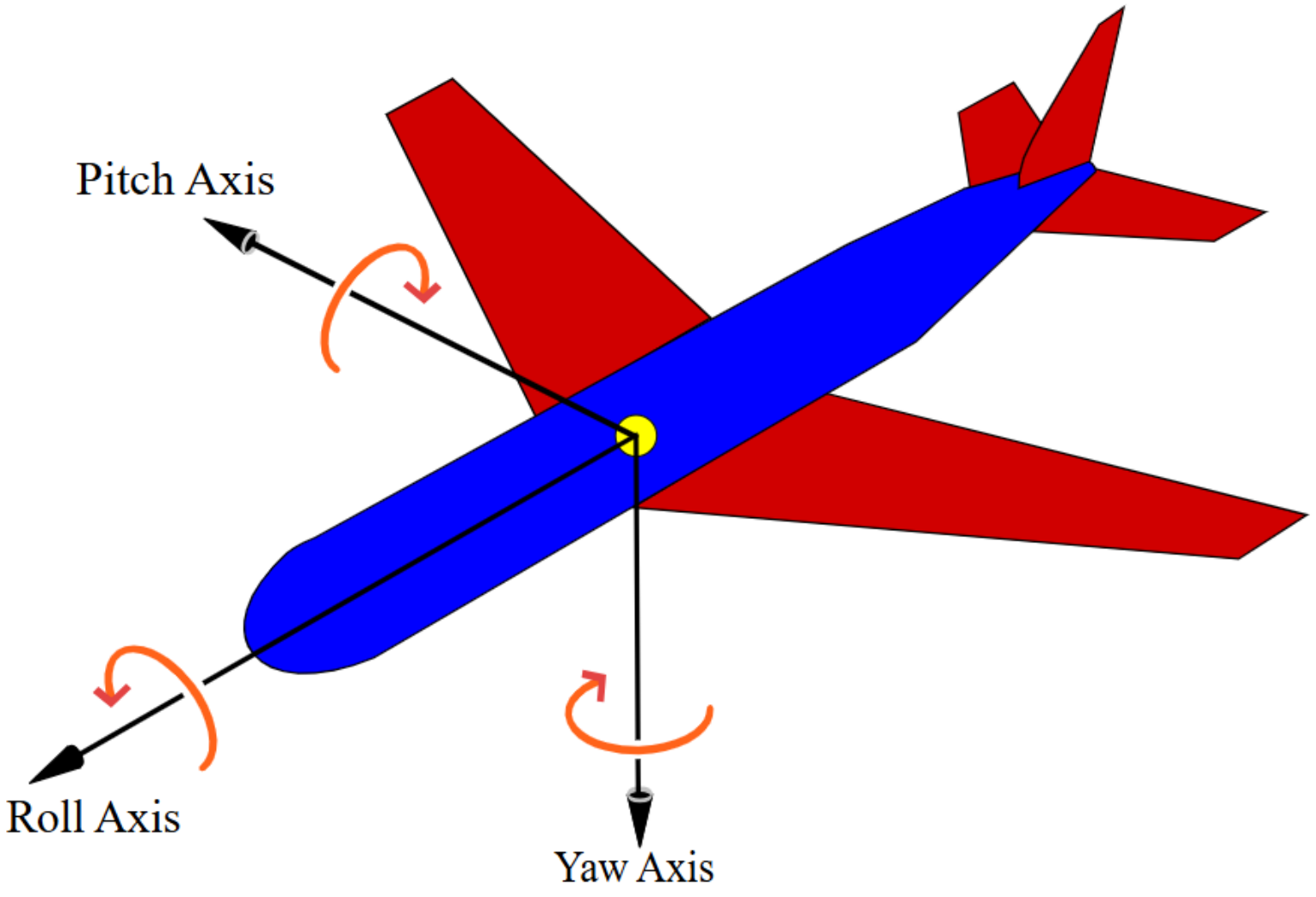
$$M_o(t) = \begin{bmatrix} A_{11} & A_{12} & A_{13} & p_x \\ A_{21} & A_{22} & A_{23} & p_y \\ A_{31} & A_{32} & A_{33} & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

- $\mathbf{p} = (p_x, p_y, p_z)$ is the position.
- The **A** matrix contains the orientation.

Orientation parameters

- Make **p** and **A** time dependent!
- We can represent members of **p** with different functions.
- *E.g.: If something is falling changing p_y is enough.*
- Members of **A** are connected, and we only care about the orientation (not interested in: scaling, shearing)
- The orientation can be given by three rotation along the axes $\rightarrow$ with three independent functions.

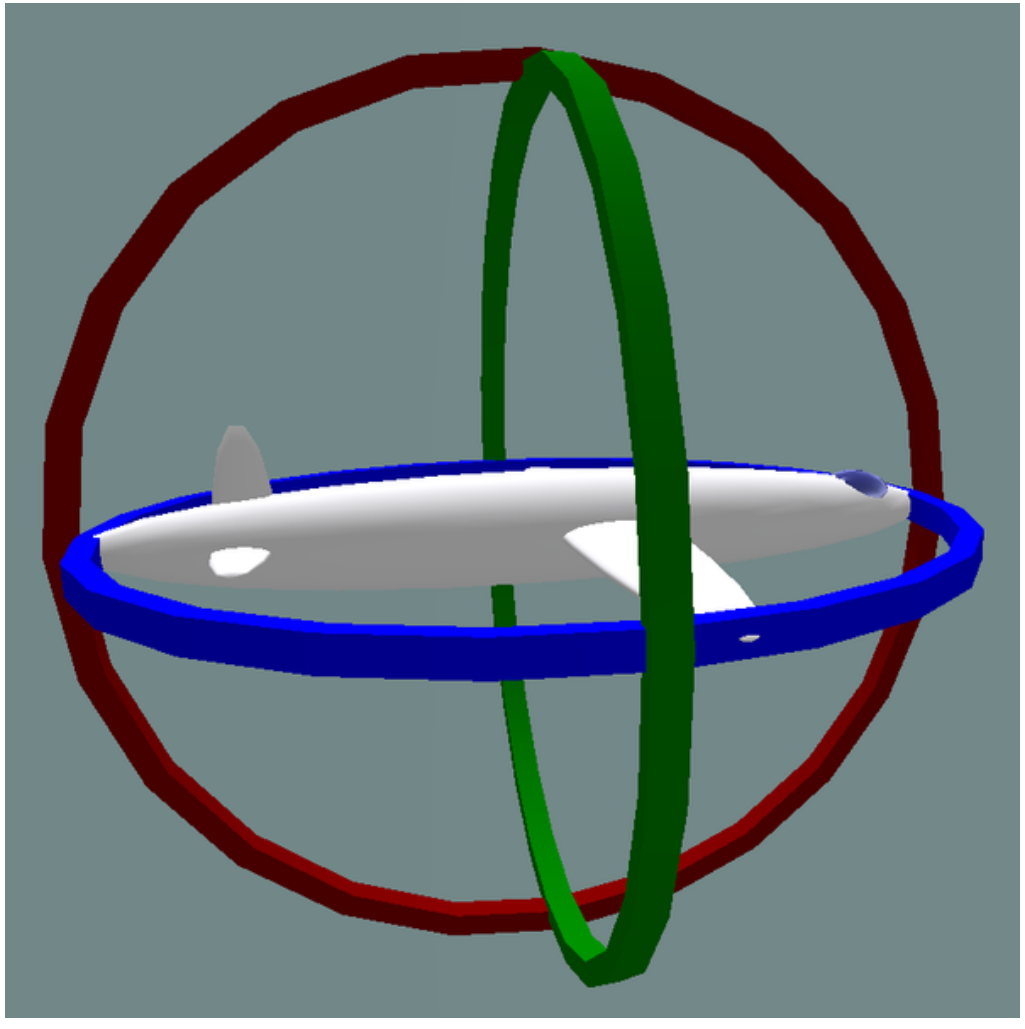
Yaw, pitch, roll



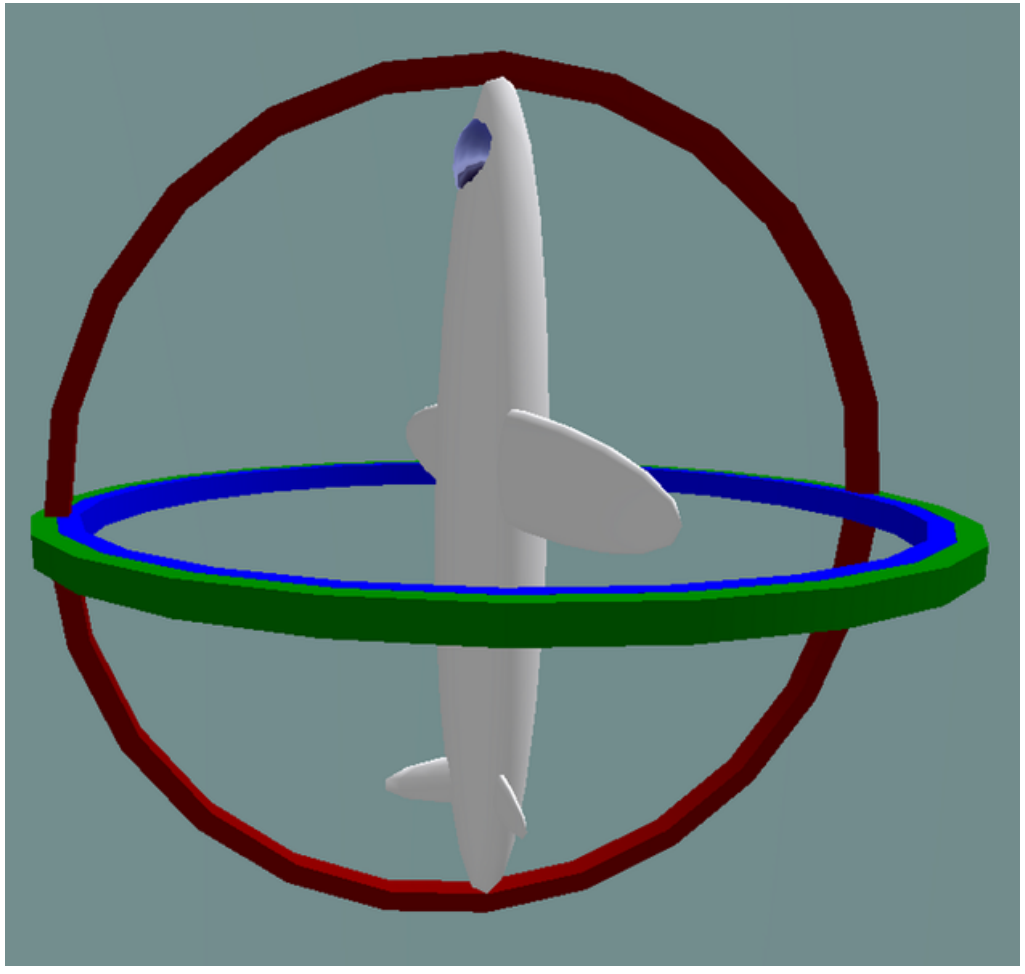
Yaw, pitch, roll

- We specify the vertical (*yaw*), lateral (*pitch*) and longitudinal (*roll*) rotations at the same time.
- We already used it, it can be described with a 3×3 matrix, product of the three rotations.
- It can be specified with the angle of rotations along three axes $\Rightarrow$ it gives the orientation.
- Arbitrary orientation can be described with it, but be careful when animating...

"How about sending me a fourth gimbal for Christmas?" – Mike Collins, pilot of Apollo 11

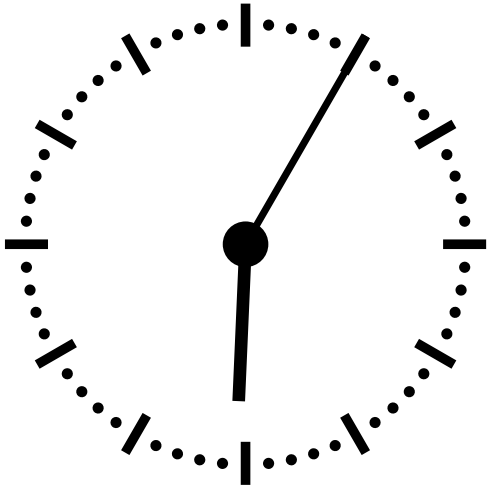


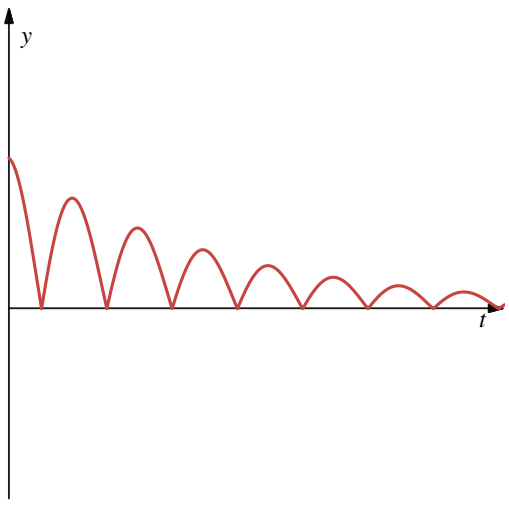
Gimbal lock



Formula based animation

- The change of a given property is described by a corresponding function.
- E.g.: Clock hands
  - Minute hand: $yaw(t) = t/10$
  - Hour hand: $yaw(t) = t/120$
  - Is t is in seconds and the rotation in degrees.
- E.g.: Bouncing ball
  - $p_y(t) = |\sin(\omega t + \theta_0)| \cdot e^{-kt}$





Keyframe animation

- It would be difficult to give a formula for a complicated movement.
- Rather, we describe what we want to see at certain timeframes.
- These are the keyframes.
- A property between two keyframes is calculated by *interpolation*.

Keyframe animation

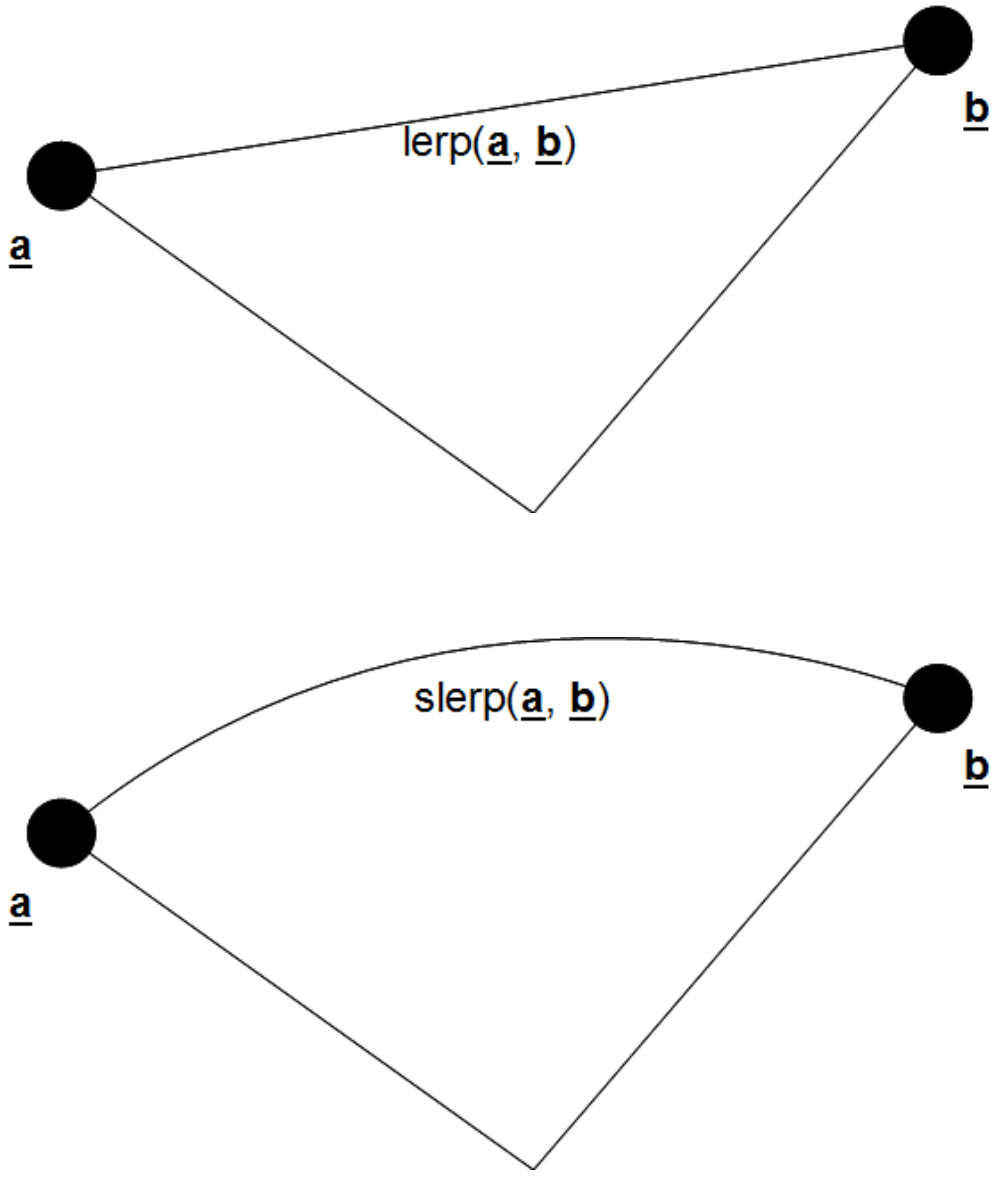
- With interpolation, we fit a continuous curve to some parameters of the object.
- During the playback/saving of the animation, the program evaluates the object's parameter functions with the corresponding t value in each frame.
- It calculates the transformation matrices from these.
- It produces the image using the matrices.

Linear interpolation (reminder)

- Let the timestamps of our keyframes be t_0 and t_1.
- Let the interpolated property be g.
- With linear interpolation $\forall t \in [t_0, t_1]$ we get

$$g(t) = \left(1 - \frac{t - t_0}{t_1 - t_0}\right) g(t_0) + \frac{t - t_0}{t_1 - t_0} g(t_1)$$

Linear and spherical linear interpolation



Interpolation between keyframes

Q: What else is wrong with linear interpolation?

A: It rarely looks natural. During animation, the speed is constant, before and after it is zero.

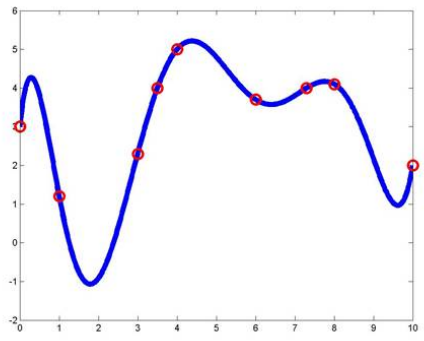
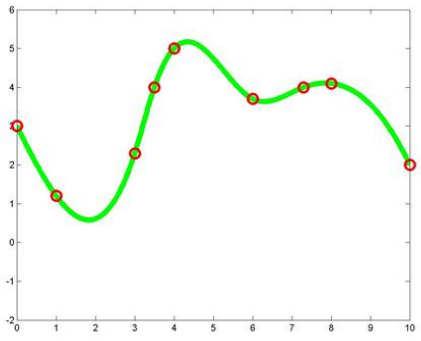
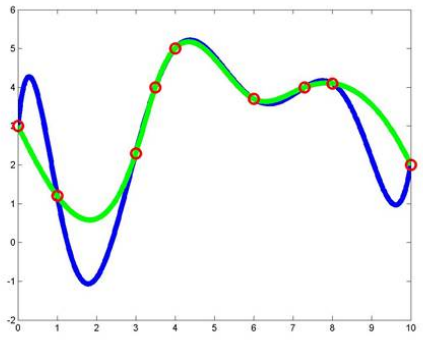
- Rolled ball: continuously decelerates.
- Falling Piano: Accelerates continuously.
- Rocket from Earth to Mars: accelerates, moves, decelerates.

- For interpolation like this, we can use:
  - Square root function.
  - Quadratic function.
  - The interpolation methods we learned
  - ...
- In practice: Bézier curves, splines

Polynomial interpolation

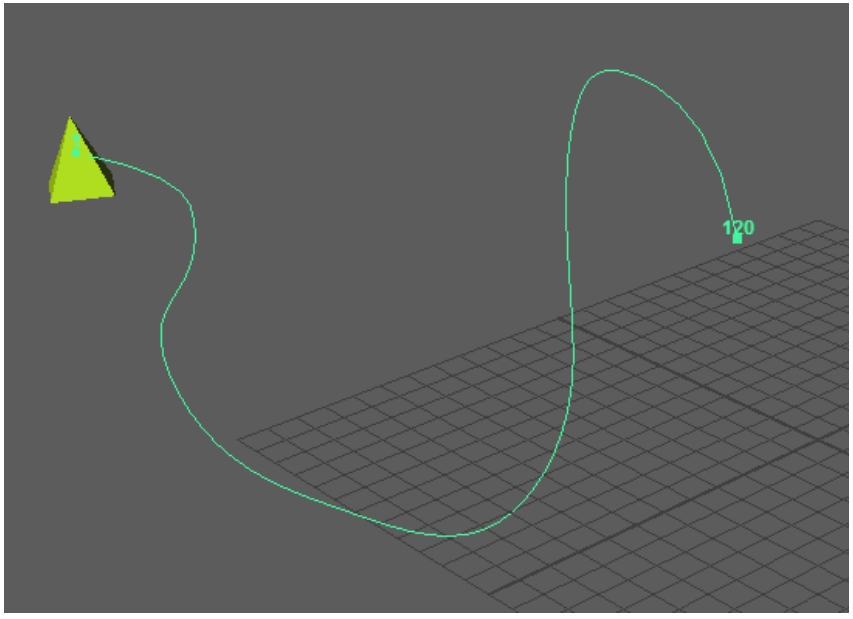
- We can write $n-1$ degree polynomial for n key points.
- Interpolation polynomial: in every keyframe it gives back the prescribed value
- Coefficients can be calculated with Lagrange interpolation as well (but: large degree – large "swings" even between data points!).
- Linear interpolation is a special case of Lagrange interpolation where $n = 2$

Spline interpolation

- In case of a function from a high degree polynomial interpolation, it “fluctuates” between adjacent points, thus messing up the animation.
- Spline interpolation: use several connected low degree polynomials for interpolation!

Path animation



- The movement of an object can also be specified by specifying the path to be traversed.
- The path is given by a 3D parametric curve.
- The model moves along this curve.

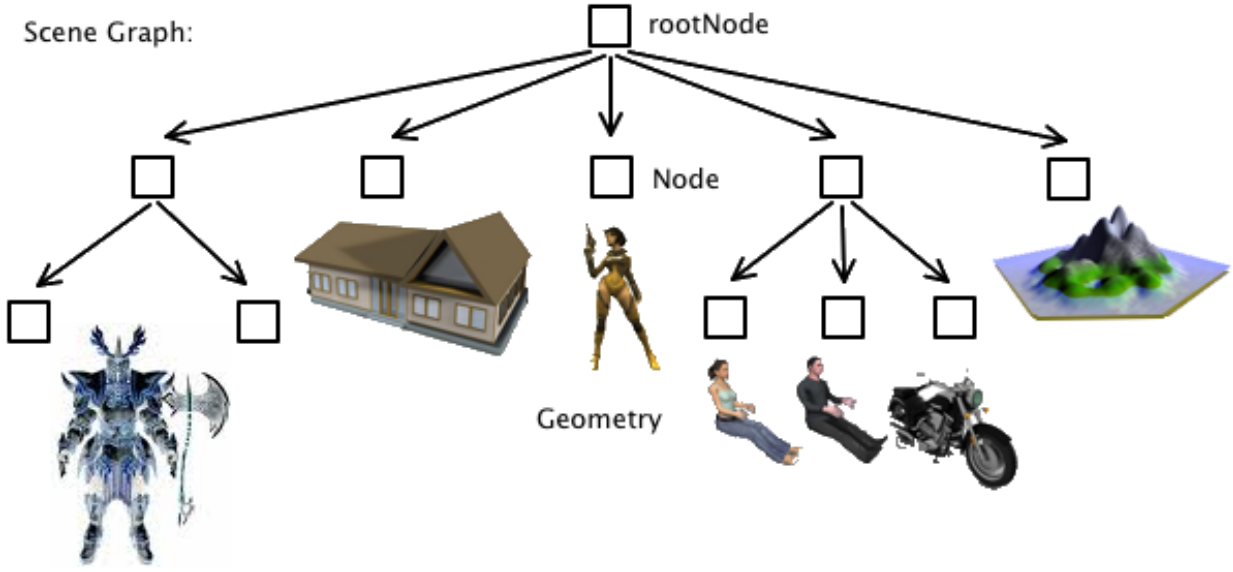
Specifying the orientation

- How do we specify the orientation of our object?
- A *forward* and an *up* direction clearly define this.
- *Remark: in case of our camera, these are the center–eye vector and up vector*
- If the path's curve is differentiable, it gives the velocity vector at each instant of time.
- The velocity vector always points forward.

Specifying the orientation

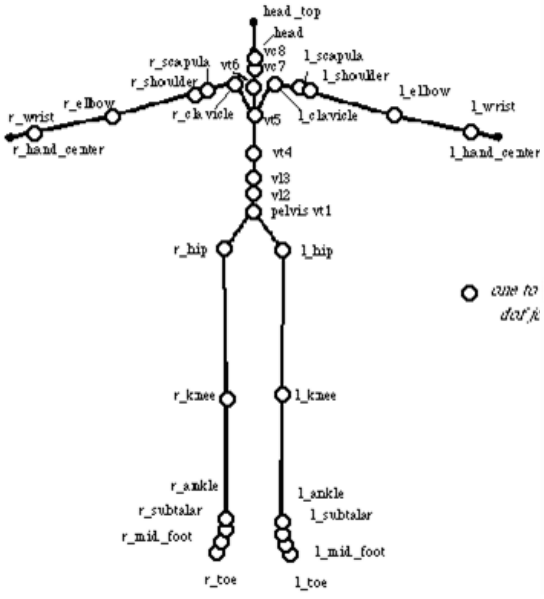
- We have two options for specifying the *up* direction.
- If there is a natural *up*, we use it. (For everything that doesn't tilt in the corners.)
- If this direction also changes, then this is the same as the direction of acceleration, i.e. the direction of the second derivative of the path's curve.

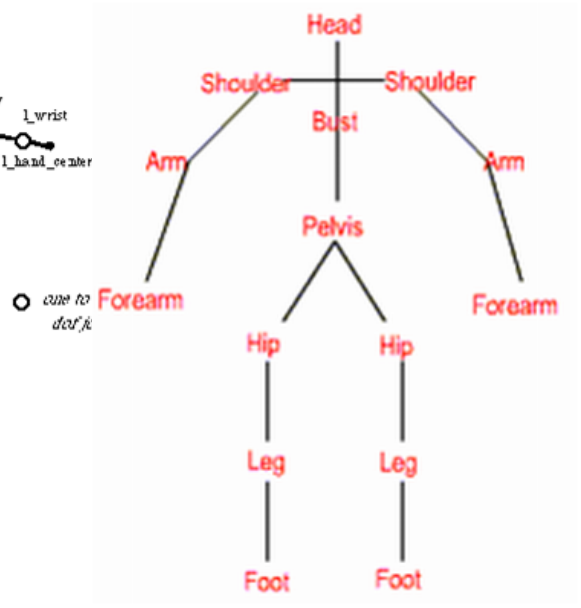
Hierarchical systems

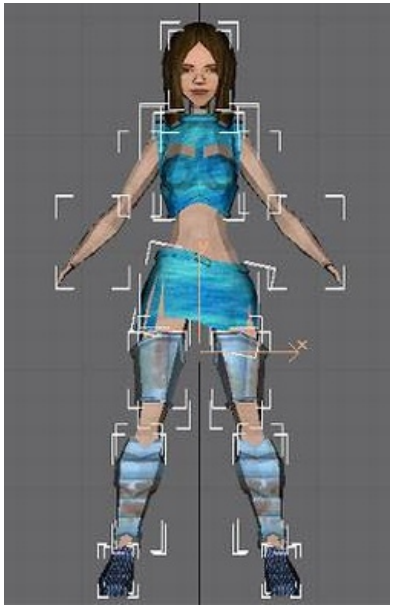


- Based on scene-graph.
- We specify the movement of a child object relative to the parent.
- Children can have more children, etc.
- We get a hierarchical system – a tree.

Example: Human body







Constraints

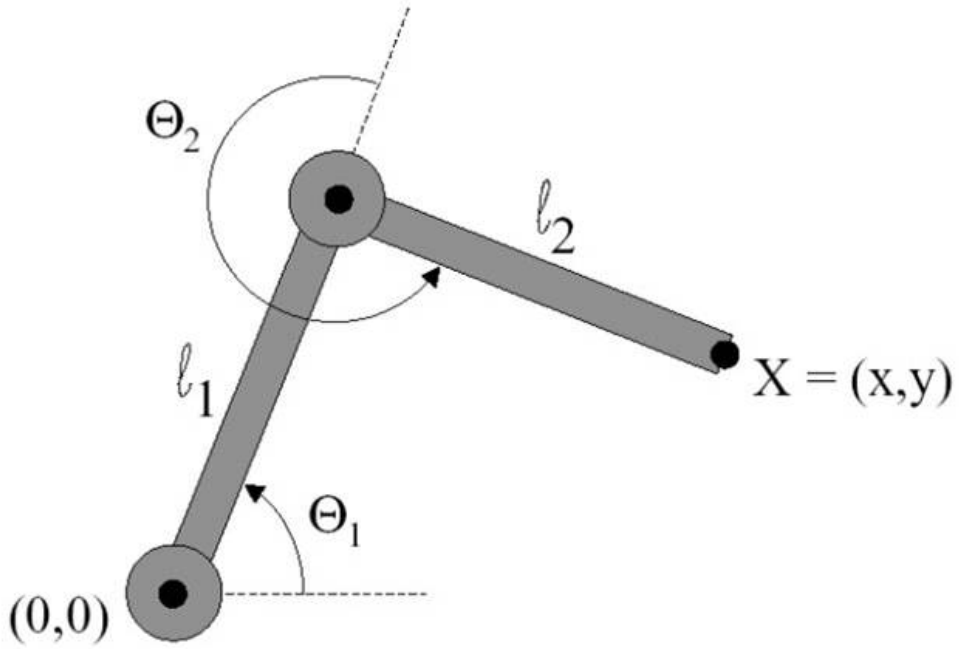
- We do not want to allow all movement relative to the parent.
- We call these limits *constraints*.
- We can limit the degrees of freedom: e.g. the elbow can only rotate along one axis, but the wrist can rotate along two
- Or the ranges: few can stand it if their head turns more than 90°.

Forward kinematics

- We define an end state as a function of state variables.
- It can be used well for simulations.
- For each element, we specify the corresponding transformation relative to the parent object.
- We evaluate them from top to bottom in the hierarchy.
- The transformation belonging to the given element is the product of all ancestors and its own transformation.

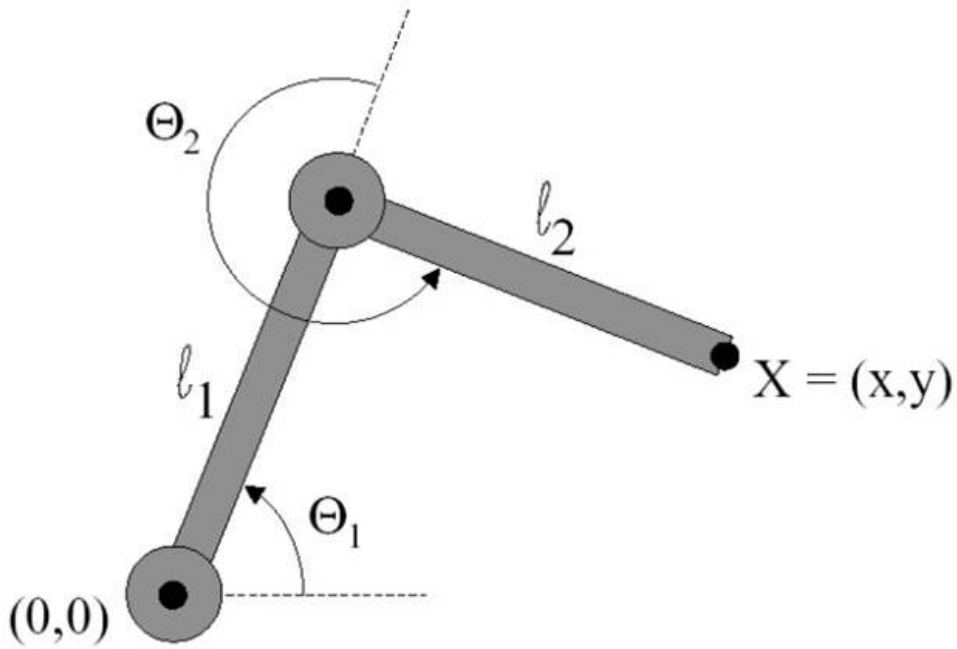
Example

- A system with two degrees of freedom containing rotational joints.
- The joints only rotate around the Z axis



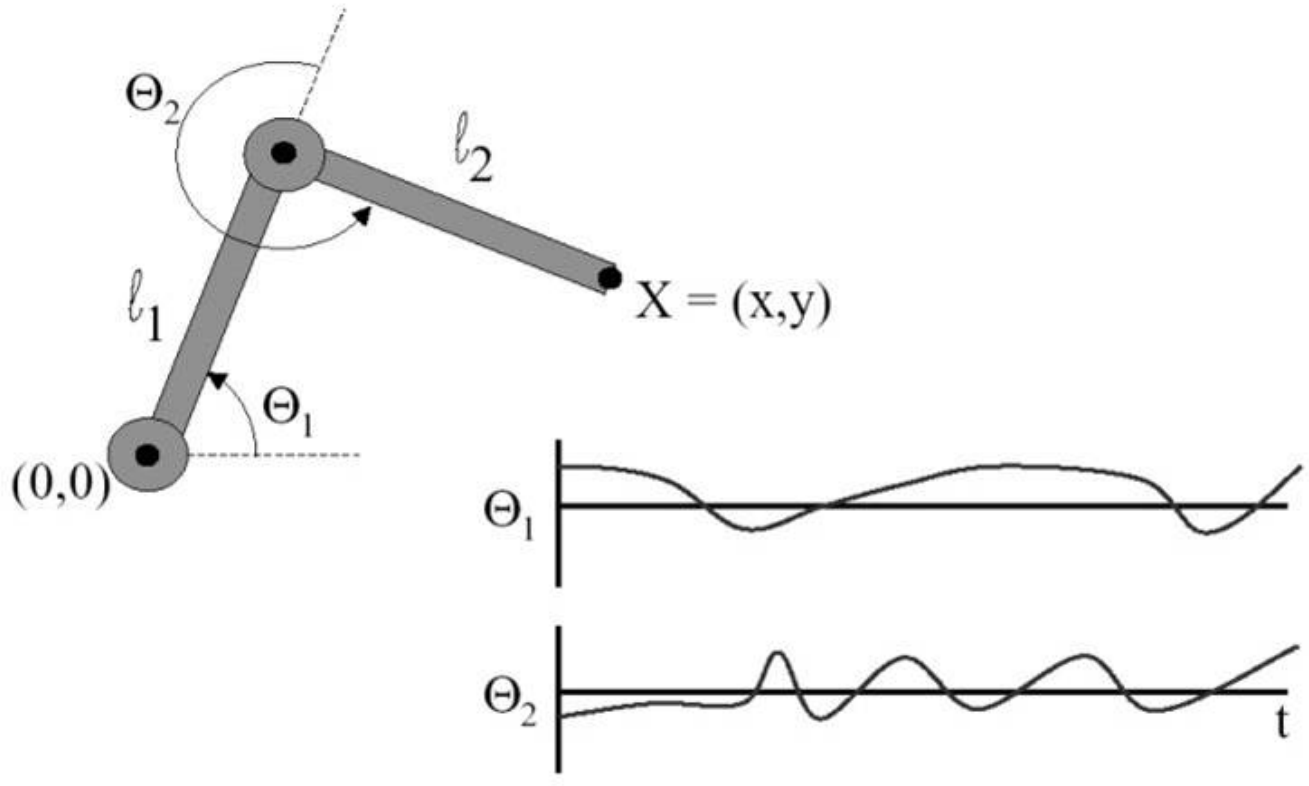
Example – continued

- State Variables: Θ_1, Θ_2
- We calculate the position of the end effector (X)
- $X = (\ell_1 \cos\Theta_1 + \ell_2 \cos(\Theta_1 + \Theta_2), \ell_1 \sin\Theta_1 + \ell_2 \sin(\Theta_1 + \Theta_2))$



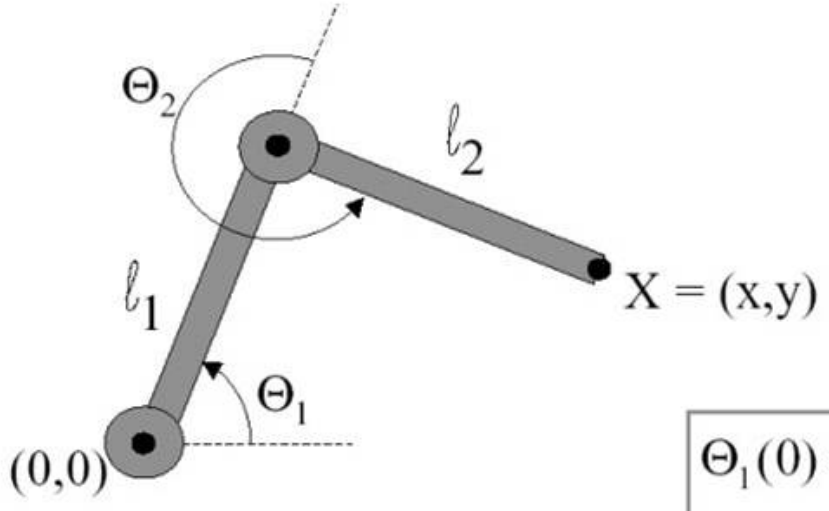
Example – continued

- State variables can be specified with a function (e.g. spline).



Example – continued

- State variables can be specified with an initial value and speed.



$$\Theta_1(0) = 60^\circ \qquad \Theta_2(0) = 250^\circ$$

$$\frac{d\Theta_1}{dt} = 1.2 \qquad \frac{d\Theta_2}{dt} = -0.1$$

What can't forward kinematics do?

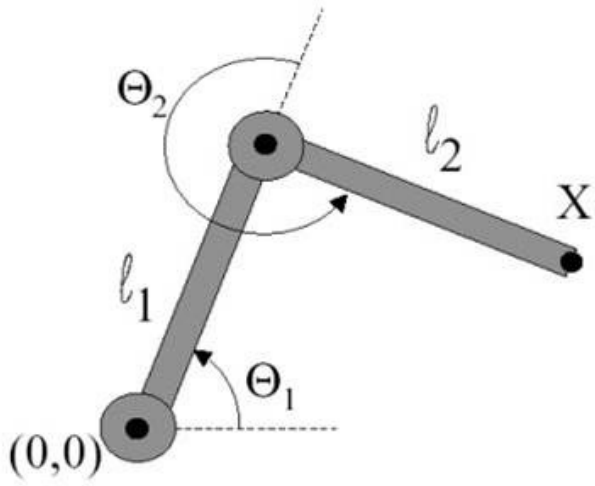
- Forward kinematics cannot be used if the structural relationship is highly non-linear
- Even though we interpolate evenly in the space of the state, the effector can wildly swing between key points
- Problematic cases:
  - Foot movement on the ground
  - The final state is good, but parts of the equipment may go through each other while interpolating.

Inverse kinematics

- Inverse kinematics interpolates the position of the effector, then calculates the value of the state variables from the interpolated position of the effector.
- Another name for inverse kinematics is goal-oriented animation.
- *"I want to grab this, how do I rotate my joints?"*

Example

- We calculate the value of the state variables from the position of the effector.



$$\Theta_2 = \cos^{-1}\left(\frac{x^2 + y^2 - l_1^2 - l_2^2}{2 l_1 l_2}\right)$$

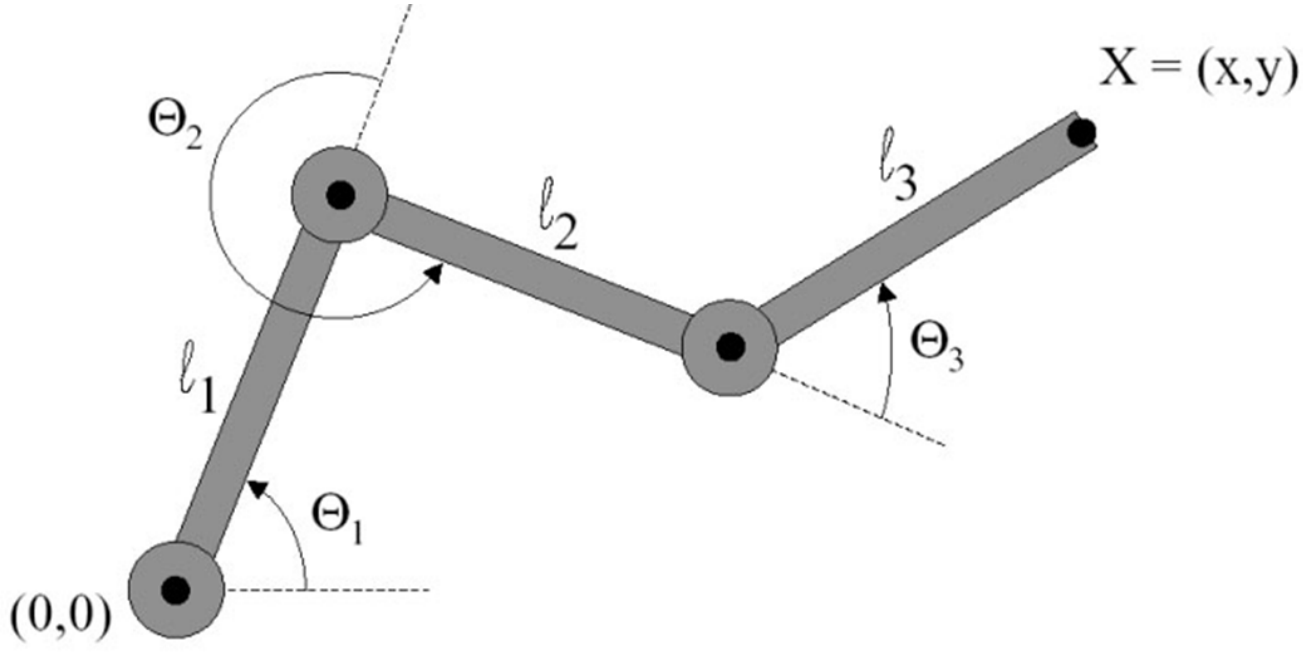
$$\Theta_1 = \frac{-(l_2 \sin(\Theta_2))x + (l_1 + l_2 \cos(\Theta_2))y}{(l_2 \sin(\Theta_2))y + (l_1 + l_2 \cos(\Theta_2))x}$$

Problems

- It is difficult to describe a “natural-looking” movement with it.
- Calculating the inverse function is not trivial,
- ... and the solution may not be unique (under-determined),
- ... or there is no solution at all (over-determined).

Example

- Number of equations: 2, number of unknown variables: 3 $\Rightarrow$ Infinite number of solutions!
- or no solution, e.g. if $\|X\| > \ell_1 + \ell_2 + \ell_3$
- degree of freedom – DOF
- System DOF $>$ effector DOF
- Human skeleton approx. 70 DOF!



Example

Example for multiple and non-existent solutions