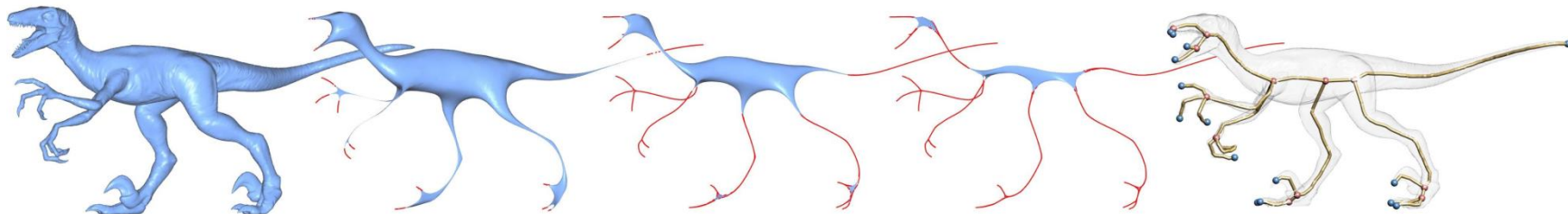


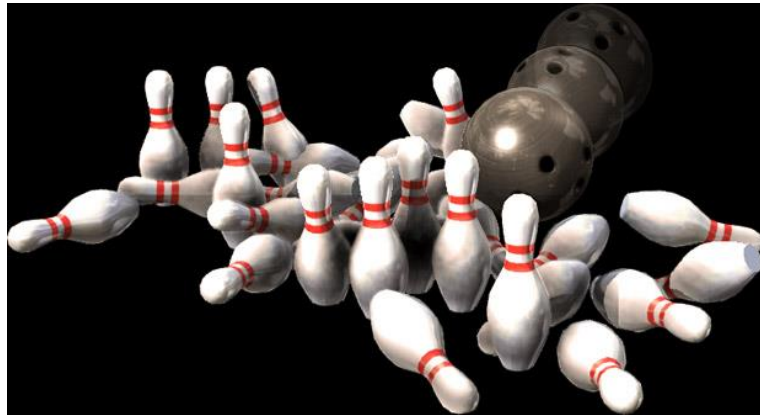
Hierarchical Modeling & Digital Animation



[Au et al. 2008]

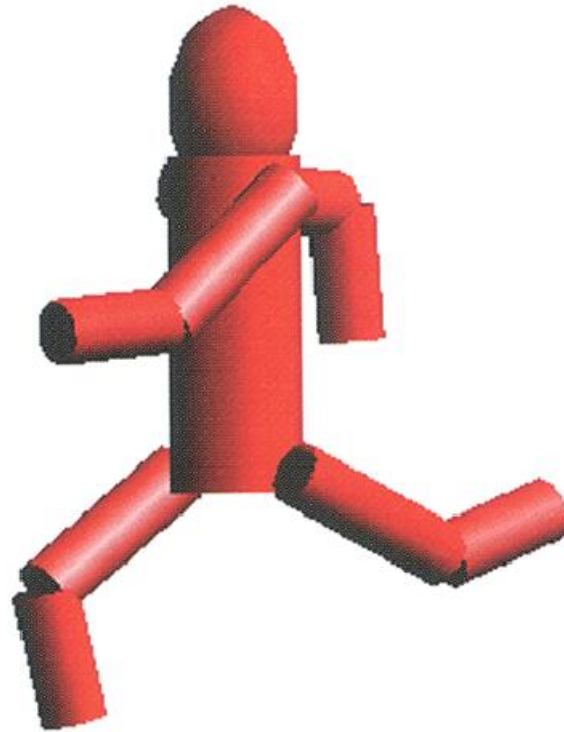
Types of animated objects

- Rigid objects animation
 - A rigid object is one that is non-deformable



- Free-form deformation
- Articulated figure animation
 - objects made of rigid sub-parts

Model and animate articulated figure



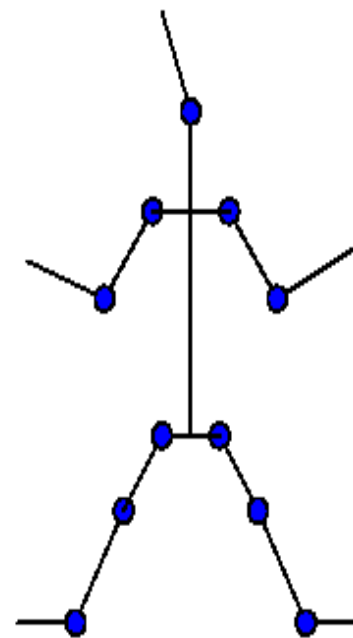
Articulated figure:

Rigid parts (BONES) connected by joint

Animate articulated figure

Applications:

- Robotics
- 3D virtual actors CA,
- Where the dynamic behavior of the objects is characterized by relations between the various constituent parts of the model.



joint
●
Bones
/link



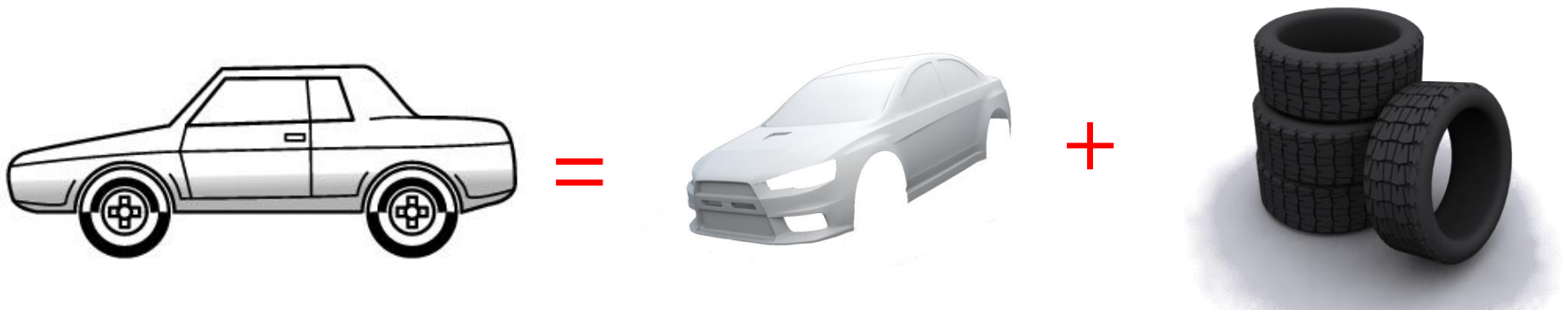
skeleton

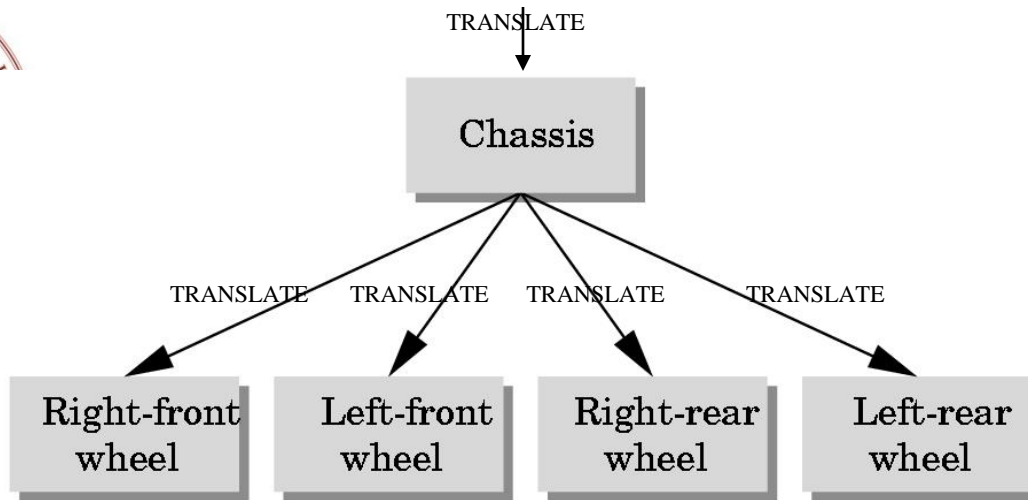
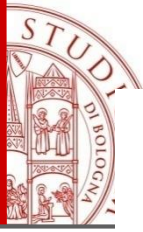
**Modeling articulated figures
through hierarchical modeling:**
Represent parts of an object and the
connecting joints between the parties

Hierarchical modeling of articulated figures

**A complex object is a set of more elementary objects
in the hierarchical relationship among them.**

- Suppose you want to draw and animate a car
- We can design the body and 4 wheels





Representation tree

The children inherit the transformations applied to the parent node

Invoking a transformation in OpenGL modifies the current matrix Modelview that automatically affects all primitives drawn after that point

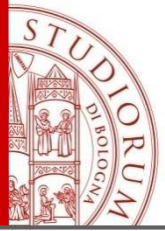
It's quite natural to implement inheritance transformations in OpenGL

```
glTranslatef(..);  
Draw_chassis();
```

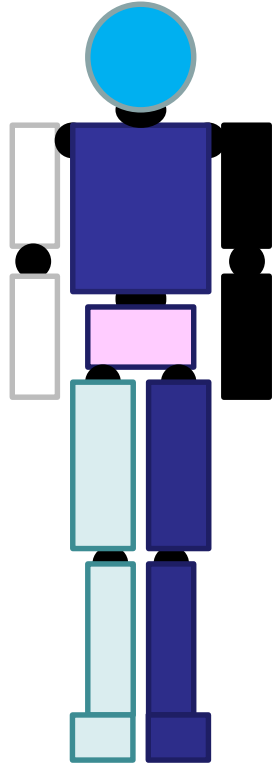
```
glTranslatef(...);  
Draw_wheel();
```

```
glTranslatef(...);  
Draw_wheel();
```

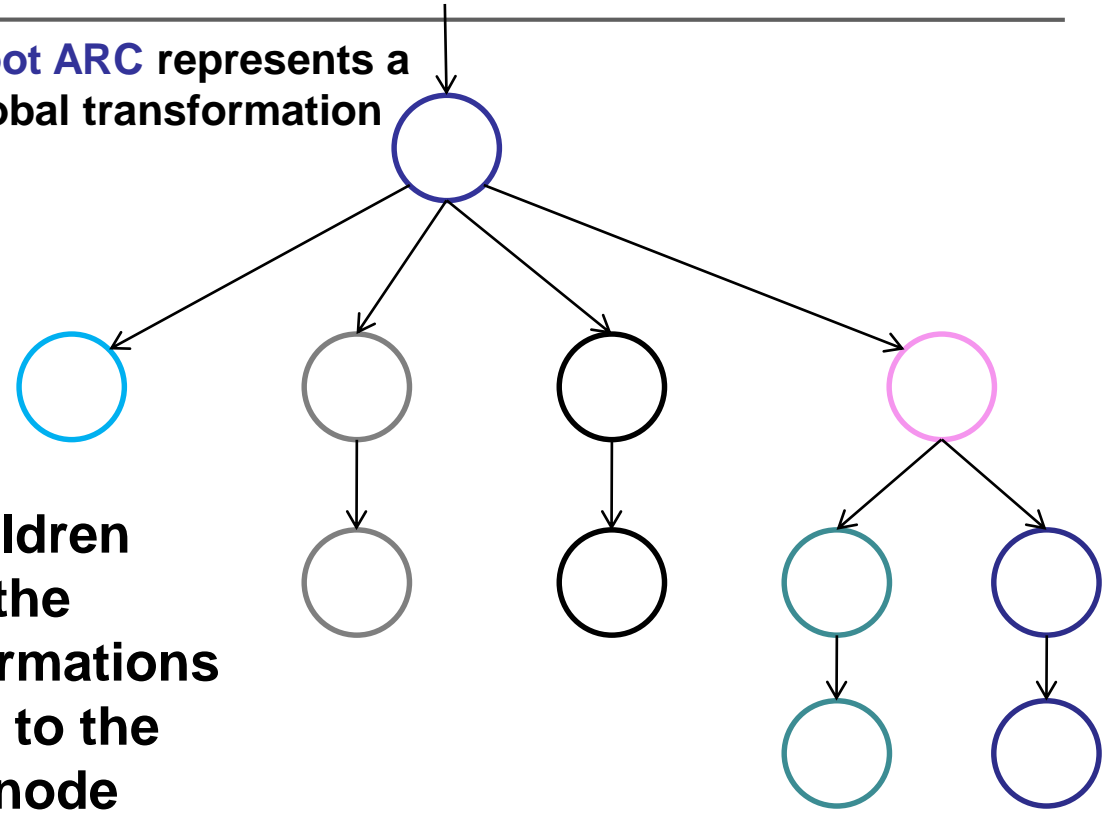
.....




Tree representation




Root ARC represents a global transformation



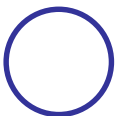
The children inherit the transformations applied to the parent node

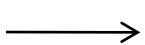
 **Link**

 **Joint**

To each LINK of the model we associate a NODE

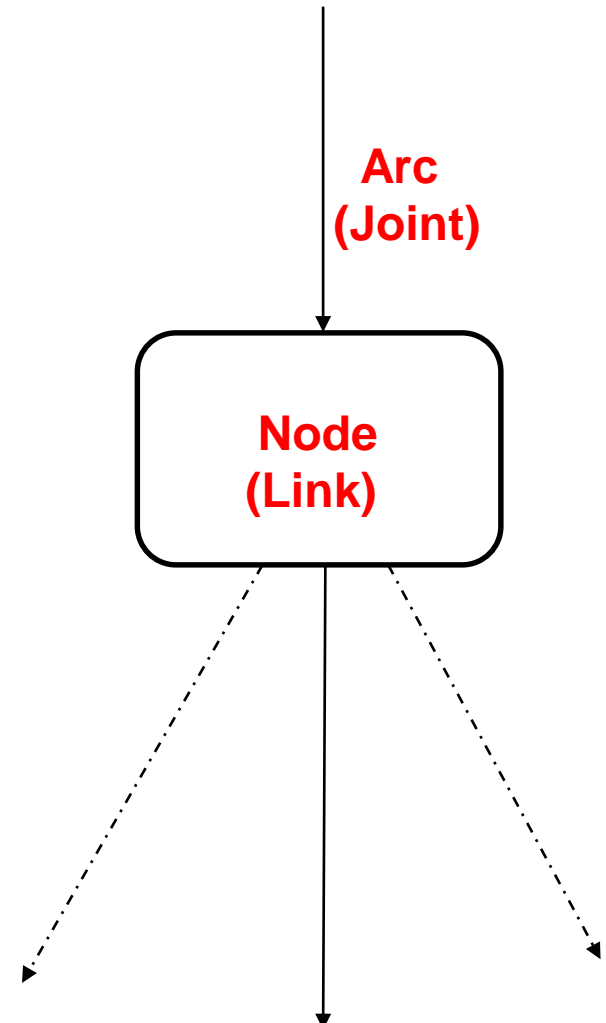
To each JOINT of the model we associate a ARC

 **Node**

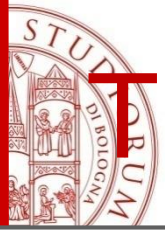
 **Arc**

Tree representation

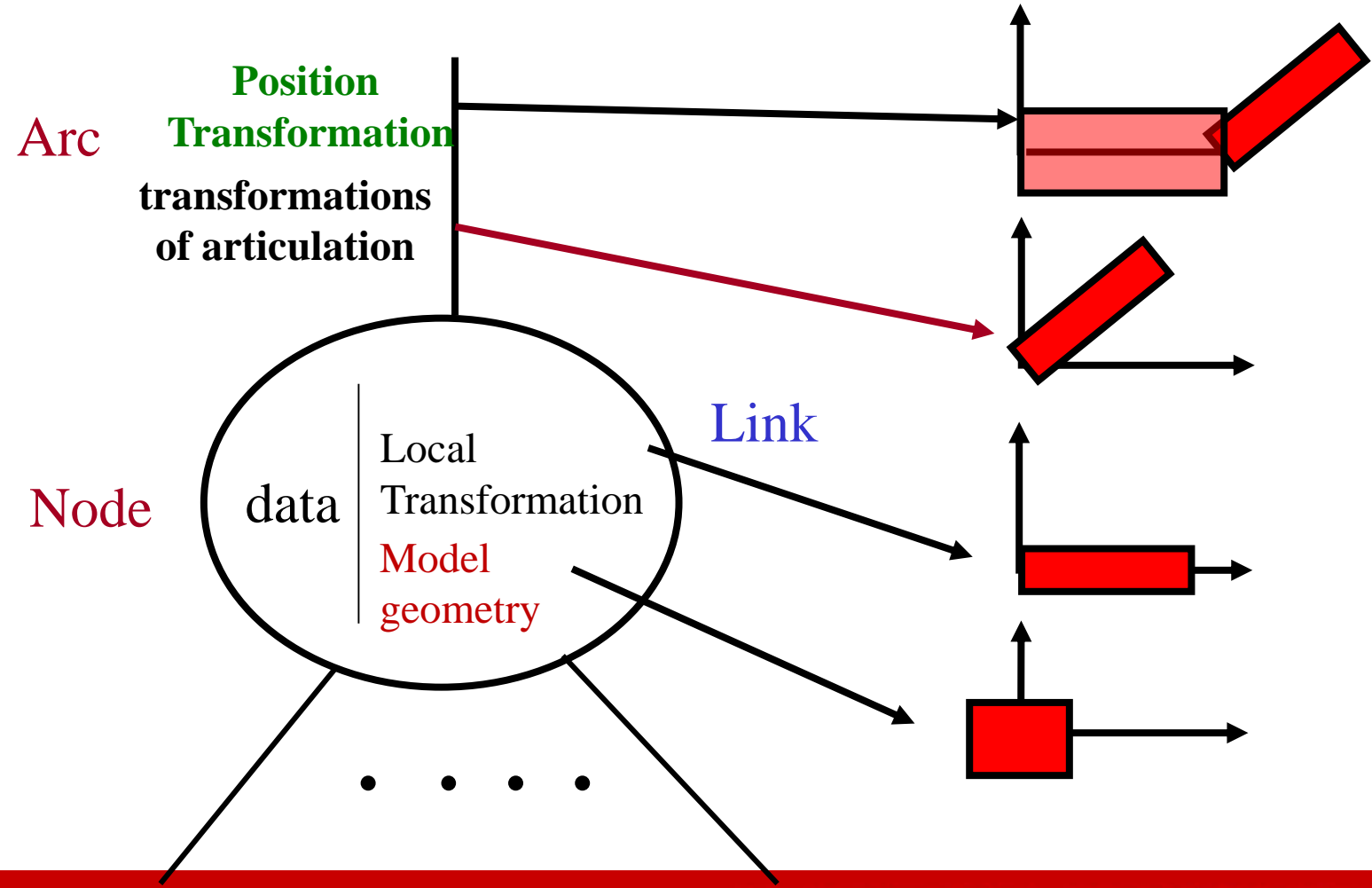
- **Arc (joint)** contains:
 - **matrix M_{pos}** transformation to place a link related to the link parent
 - **matrix M_{art}** transformation of articulation of the link
- **Node (link)** contains:
 - Draw Function
 - **matrix M_{loc}** transformation of the link in a position ready to be articulated



A part of an object can have more joints

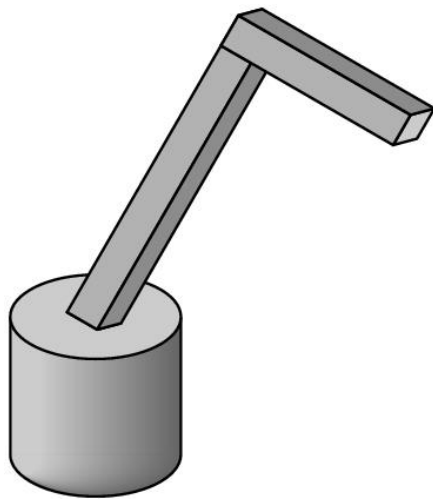


Tree representation: arc and node

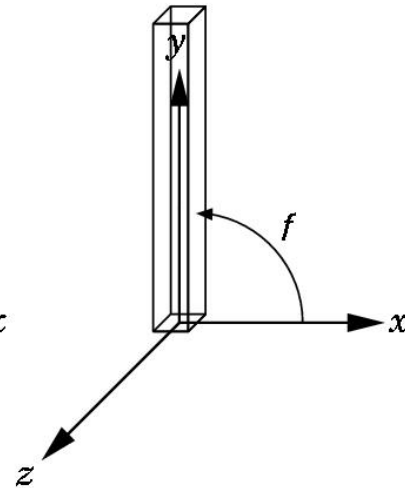
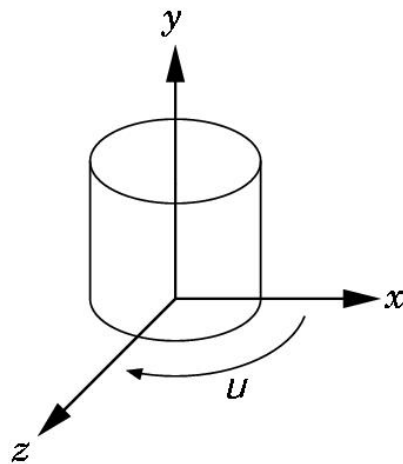


Example 1: robot arm

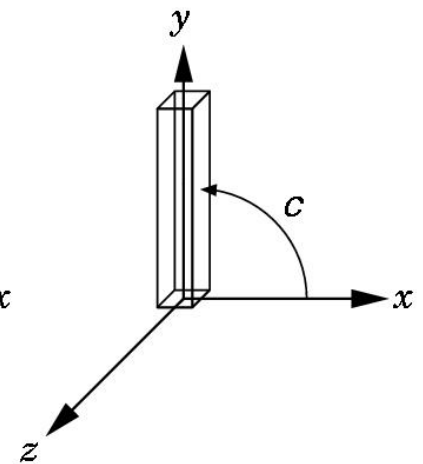
Consider a robot arm with 3 parts:



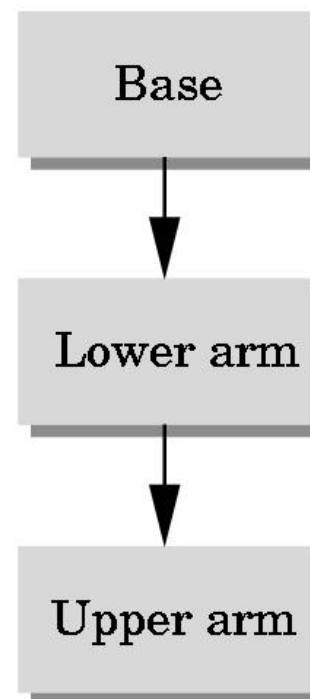
(a)



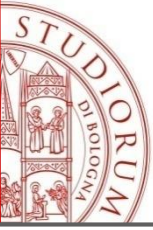
(b)



Tree representation



- We can change the shape of the robot by changing only the functions that draw the three parties.
- This makes it possible to write programs to draw the separate components and to animate them.
- Draw an articulated figure requires a traversal of the tree.



Visualizing the static model

```
display()
```

```
{
```

```
  base();
```

```
  glTranslatef(0.0, h1, 0.0);
```

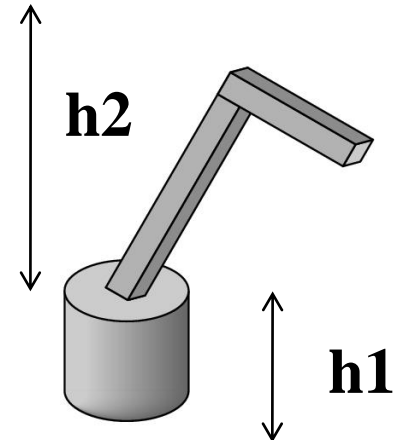
```
  lower_arm();
```

```
  glTranslatef(0.0, h2, 0.0);
```

```
  upper_arm();
```

```
}
```

M_{pos}
transformations
to place the link
related to the
parent node





Control the movement

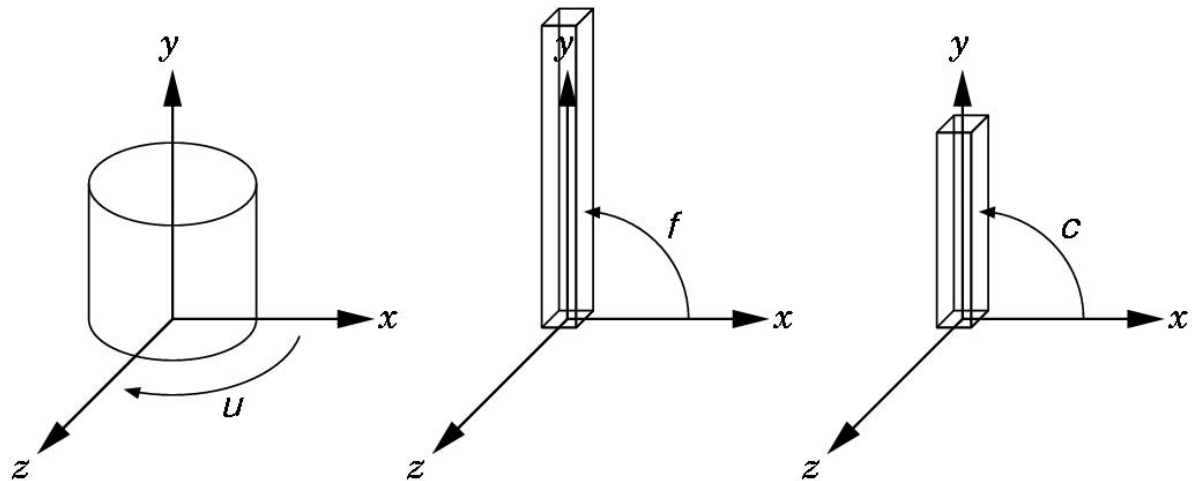
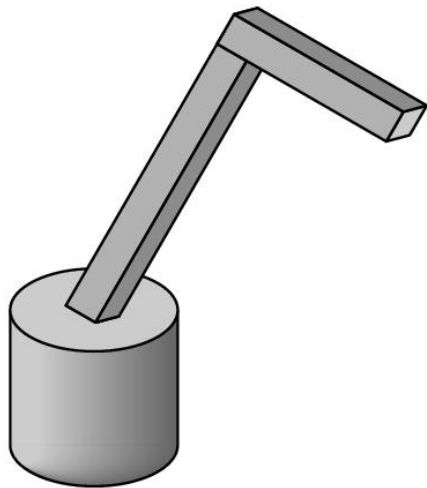
3 degrees of freedom (3 JOINT):

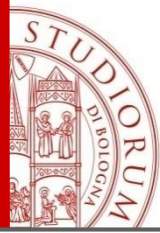
u - rotation of the base

f - rotation of the long arm

c - rotation of the short arm

Each joint determines a local transformation of the component relative to the component to which it is attached in the reference system of origin.





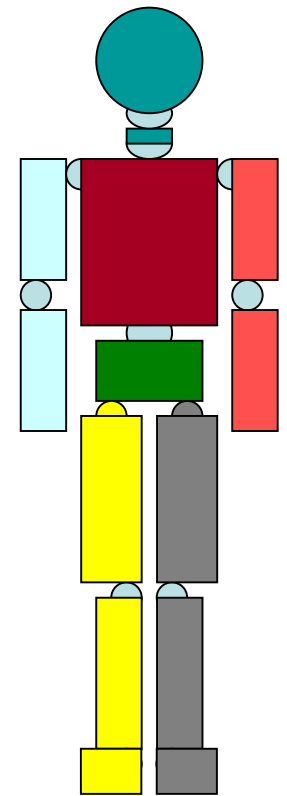
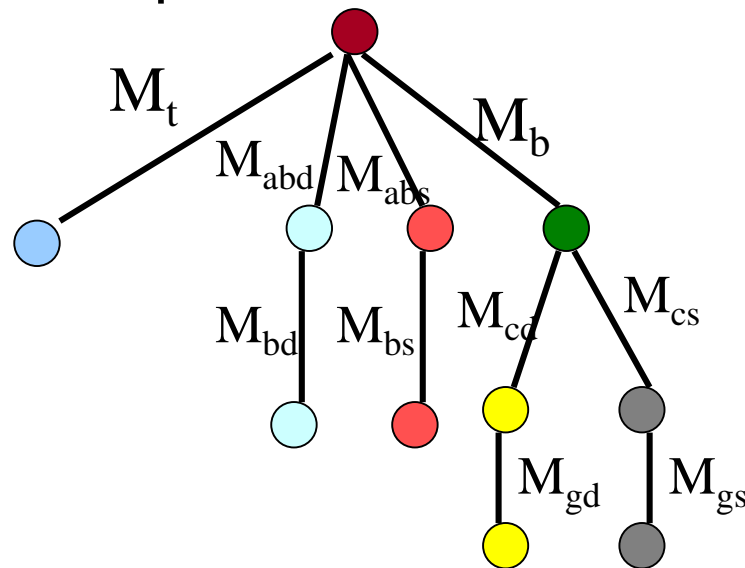
Visualizing the dynamic model

```
display()
{
  M_art → glRotatef(u, 0.0, 1.0, 0.0); Ry(u)
          base();
          glTranslatef(0.0, h1, 0.0);
          → glRotatef(f, 0.0, 0.0, 1.0); Rz(f)
          lower_arm();
          glTranslatef(0.0, h2, 0.0);
          → glRotatef(c, 0.0, 0.0, 1.0); Rz(c)
          upper_arm();
}
```

Each rotation is centered at the origin

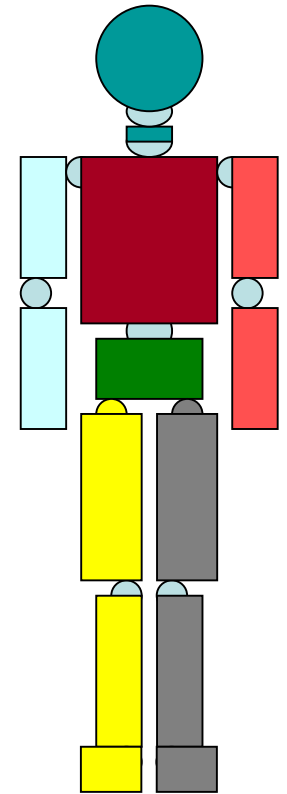
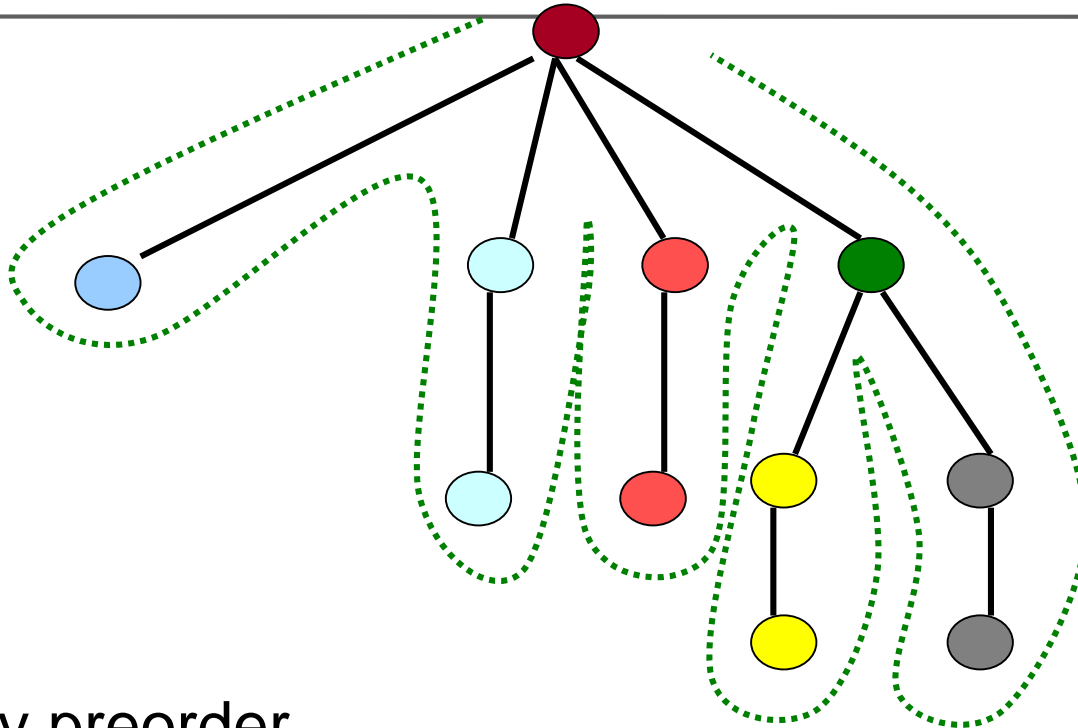
Example 2: human body

Tree Representation



- Assign matrix M at each joint, it is specified exactly how to draw the robot.
- Skeleton of a humanoid:
at least 22-24 bones (typically) reasonable: ~40 bones.

The problem is how to traverse the tree



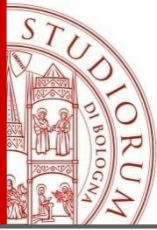
Typically preorder

This can be done in two ways:

1) **Explicitly (stack-based)**

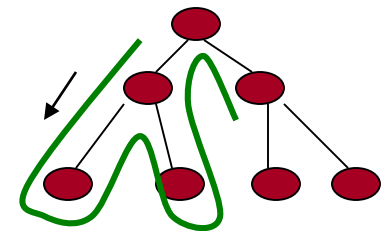
using the stack to store attributes and matrices

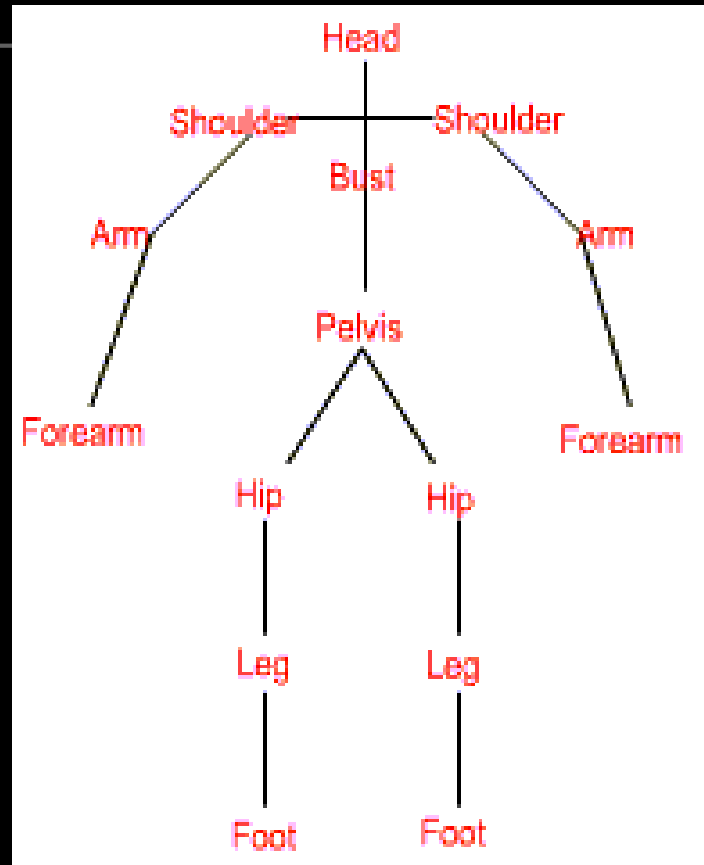
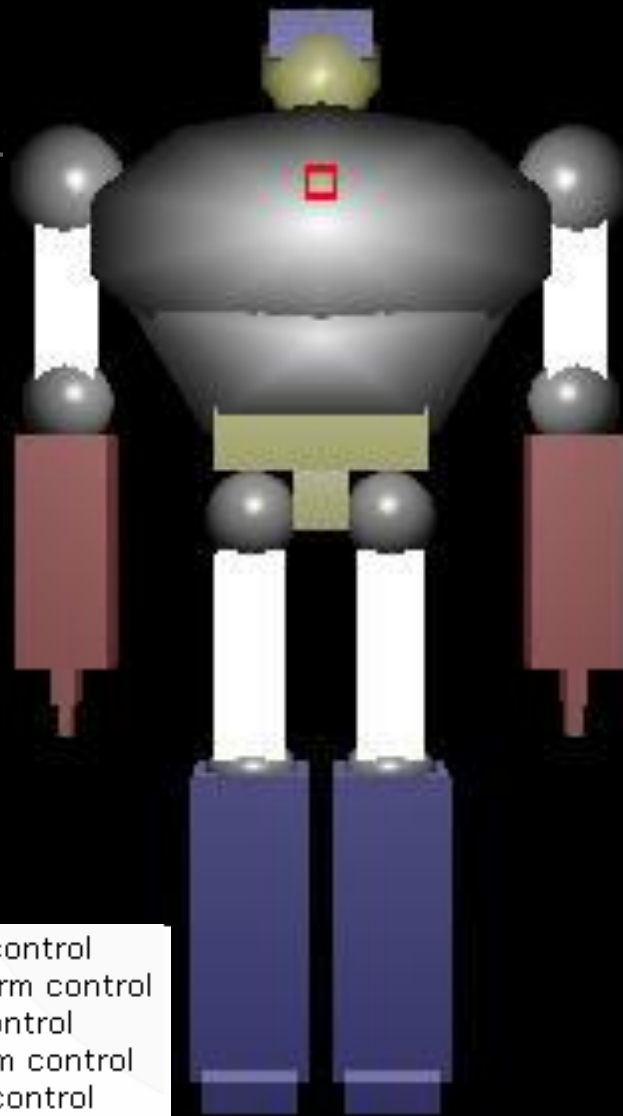
2) **Recursively (tree-based)**



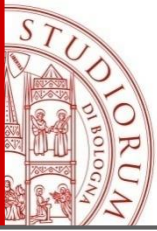
1) Traverse the tree: Stack-Based

- Mechanism **glPushMatrix** and **glPopMatrix** to traverse the tree.
- Each call **glPushMatrix** duplicates the current model-view matrix
- **DOWNWARD** crossing of an arc: **Push** the transformation of arc concatenates with the current transformation
- **UPWARD** crossing of an arc: **Pop** to restore the original matrix saved
- Similar mechanism **glPushAttrib** and **glPopAttrib**

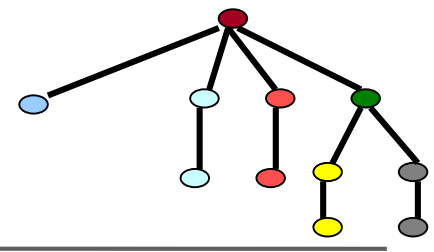




- '1' & 'Q' right arm control
- '2' & 'W' right forearm control
- '3' & 'E' left arm control
- '4' & 'R' left forearm control
- '5' & 'T' left thigh control
- '6' & 'Y' left leg control
- '7' & 'U' right thigh control
- '8' & 'I' right leg control
- KEY ARROWS rotate model
- PGUP & PGDOWN light control
- 'V' 'B' 'N' 'M' camera control

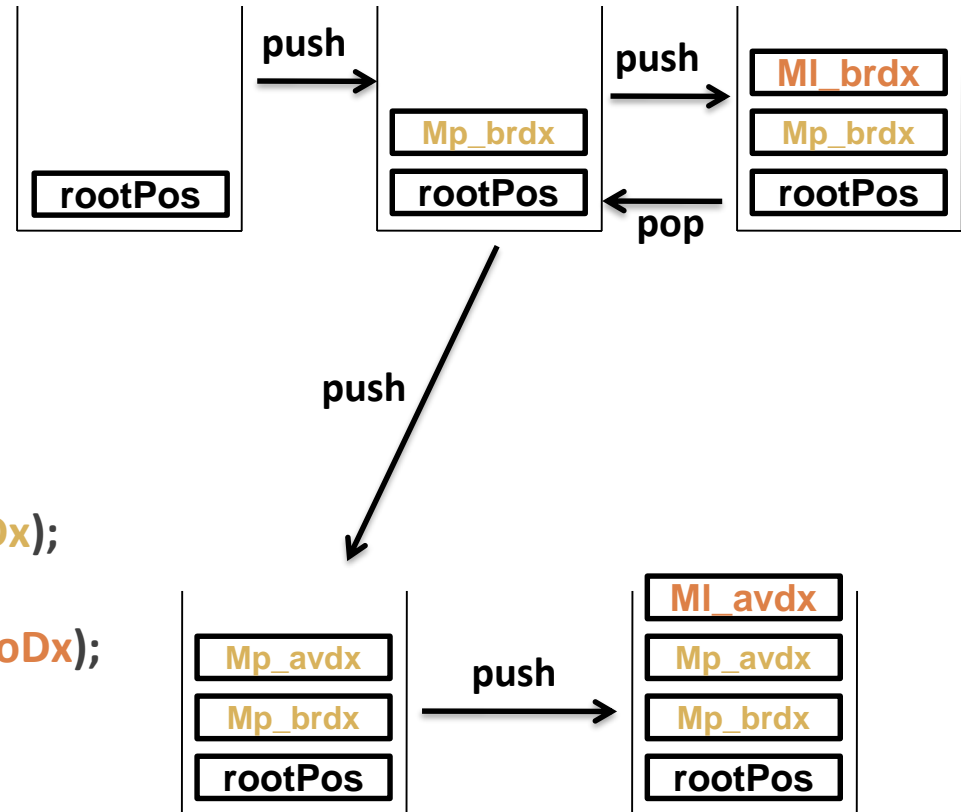


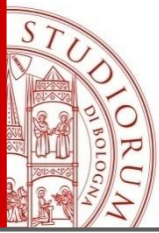
1) Traverse the tree: Stack-Based



```
void drawHuman()
```

```
{  
  ...  
  → glPushMatrix();  
  → glMultMatrixf(Mpos_braccioDx);  
  → glPushMatrix();  
  → glMultMatrixf(Mloc_braccioDx);  
  → glCallList(braccioDx);  
  → glPopMatrix();  
  → glPushMatrix();  
  → glMultMatrixf(Mpos_avBraccioDx);  
  → glPushMatrix();  
  → glMultMatrixf(Mloc_avBraccioDx);  
  → glCallList(avambraccioDx);  
  → glPopMatrix();  
  → glPopMatrix();  
  → glPopMatrix();  
  ...  
}
```

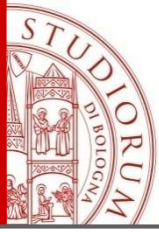




2) Traverse the tree: Tree-Based

Using a tree data structure to represent the hierarchy we can draw the object with a traversal algorithm independent of the model.

We create such a data structure for the recursive traversal in **depth-first**



2) Traverse the tree: Tree-Based

Each node has the following information:

1. A pointer to the **draw ()** function that draws the object
2. A **matrix M** which combines the transformations of position and articulation.
3. **Pointers** to the children / neighbors of the node.

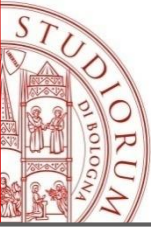
```
typedef struct treenode
{
    GLfloat M[16];
    void (*draw)();
    struct treenode *sibling;    // right-sibling
    struct treenode *child;    // leftmost-child
}
```



Draw Procedure (display())

Recursive traversal of the tree:

```
void traverse (treenode * node)
{
    if(node == NULL) return
    glPushMatrix();           // downward
    glMultMatrix( node->M ); //static transformation of articulation
    node->draw();           //local transformation and draw
    if( node->child!= NULL )
        traverse( node->child ); //pointer to the children node
    glPopMatrix();           // upward
    if( node->sibling != NULL )
        traverse( node->sibling ); //pointer to the sibling node
}
```



Set-up of the tree (init())

.....

```
glLoadIdentity();
```

```
glTranslatef(0.0, TORSO_HEIGHT, 0.0);
```

```
glGetFloatv(GL_MODELVIEW_MATRIX, head_node.M);
```

```
head_node.draw = draw_head;
```

```
head_node.sibling = &braccioSx_node;
```

```
head_node.child = NULL;
```

**Position
Transform.**

.....

```
void draw_head()
```

```
{
```

```
    glPushMatrix();
```

```
    glTranslatef(0.0, 0.5*HEAD_HEIGHT, 0.0);
```

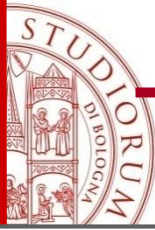
```
    glScalef(HEAD_RADIUS, HEAD_HEIGHT, HEAD_RADIUS);
```

```
    gluSphere(h, 1.0, 10, 10);
```

```
    glPopMatrix();
```

```
}
```

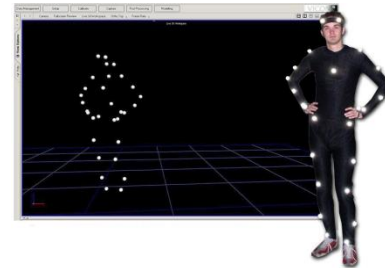
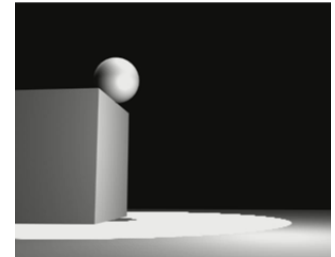
**Local
Transform.**

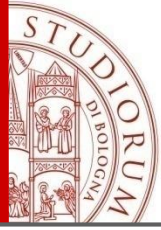


Types of Animations

Building 3D models and their animation controls is a major component of every animation pipeline. Building the controls is called “**rigging**”.

- Physically-based/
procedural animation
- Data Driven
(Motion Capture)
- Keyframing





Physics-based Animation

Realistic but difficult to control..

Ideally suited for:

- Large volumes of objects – wind effects, liquids, ...
- Cloth animation/draping

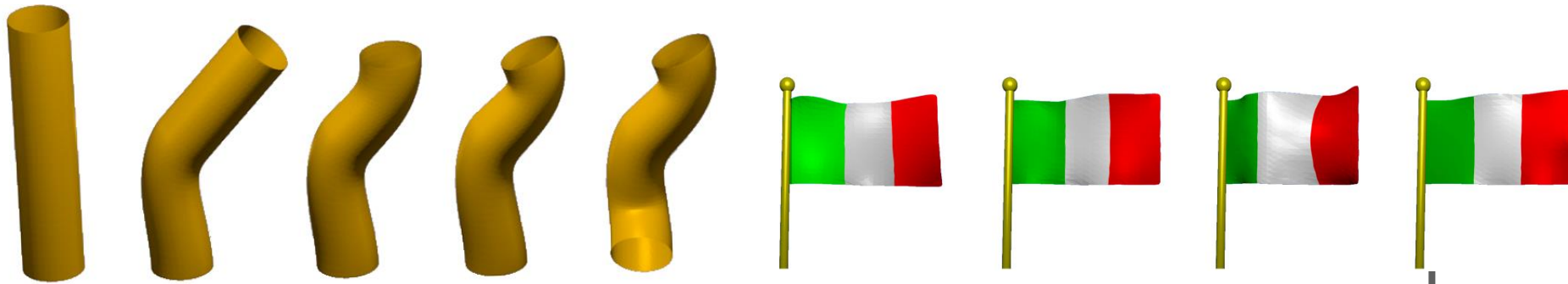
Underlying mechanisms are usually:

- **Particle systems**
- Mass-spring systems

Typically solve ordinary or **partial differential equations** using iterative methods with some initial/ending boundary values and constraints on conservation of mass/energy/angular momentum

Types of Dynamics

- Point (particle systems)
- Rigid Body simulation
- Deformable body/Physics Simulations
(include clothes, fluids, smoke, etc.)

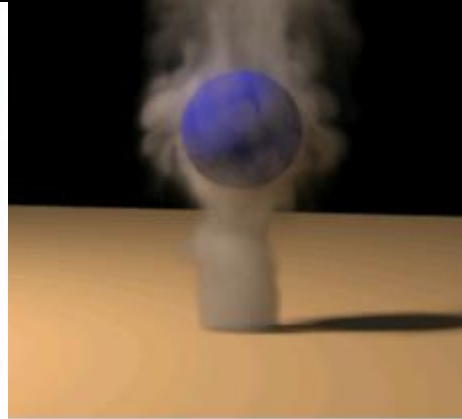


Rest Position

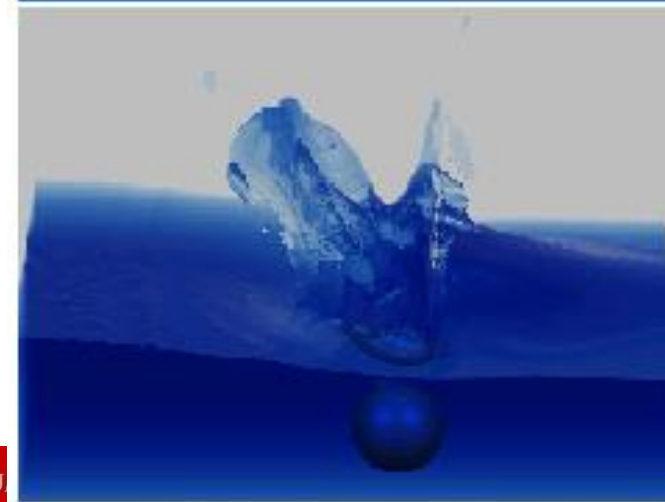
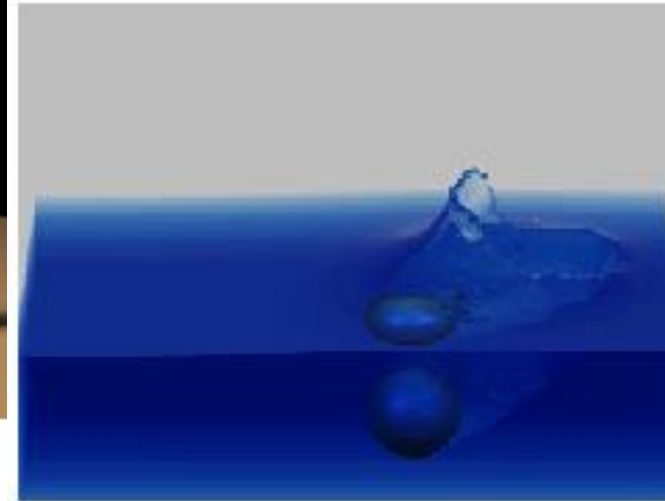
Forces \equiv Constraints

Examples

Arnauld Lamorlette and Nick Foster (PDI/DreamWorks), SIGGRAPH 2002

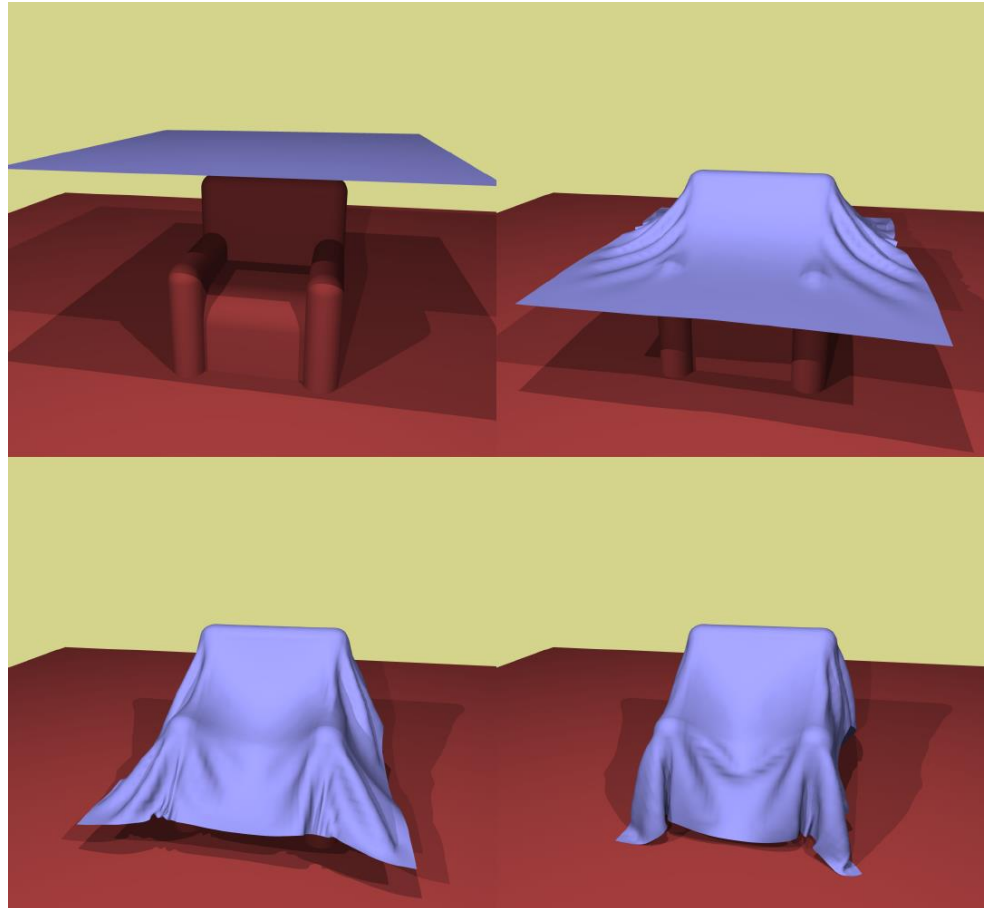


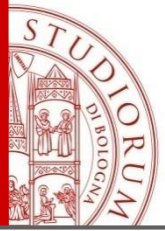
Images from Fedkiw, Stam, Jensen, SIGGRAPH 2001



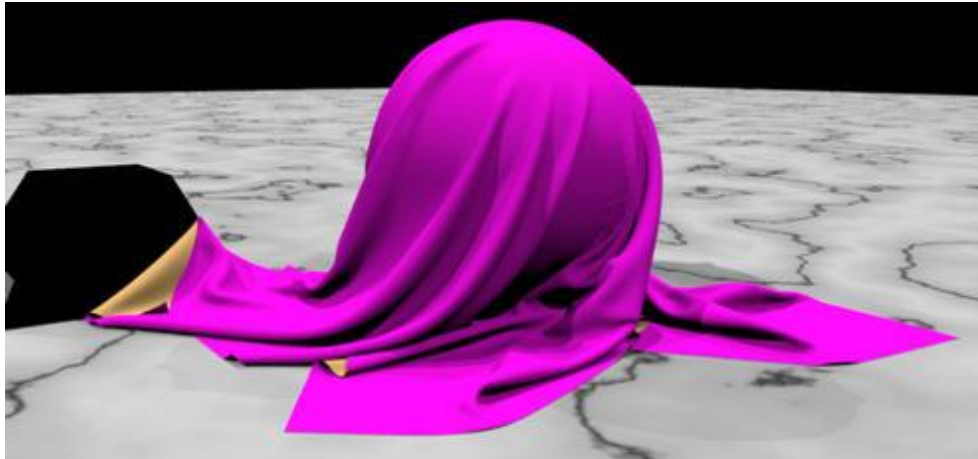


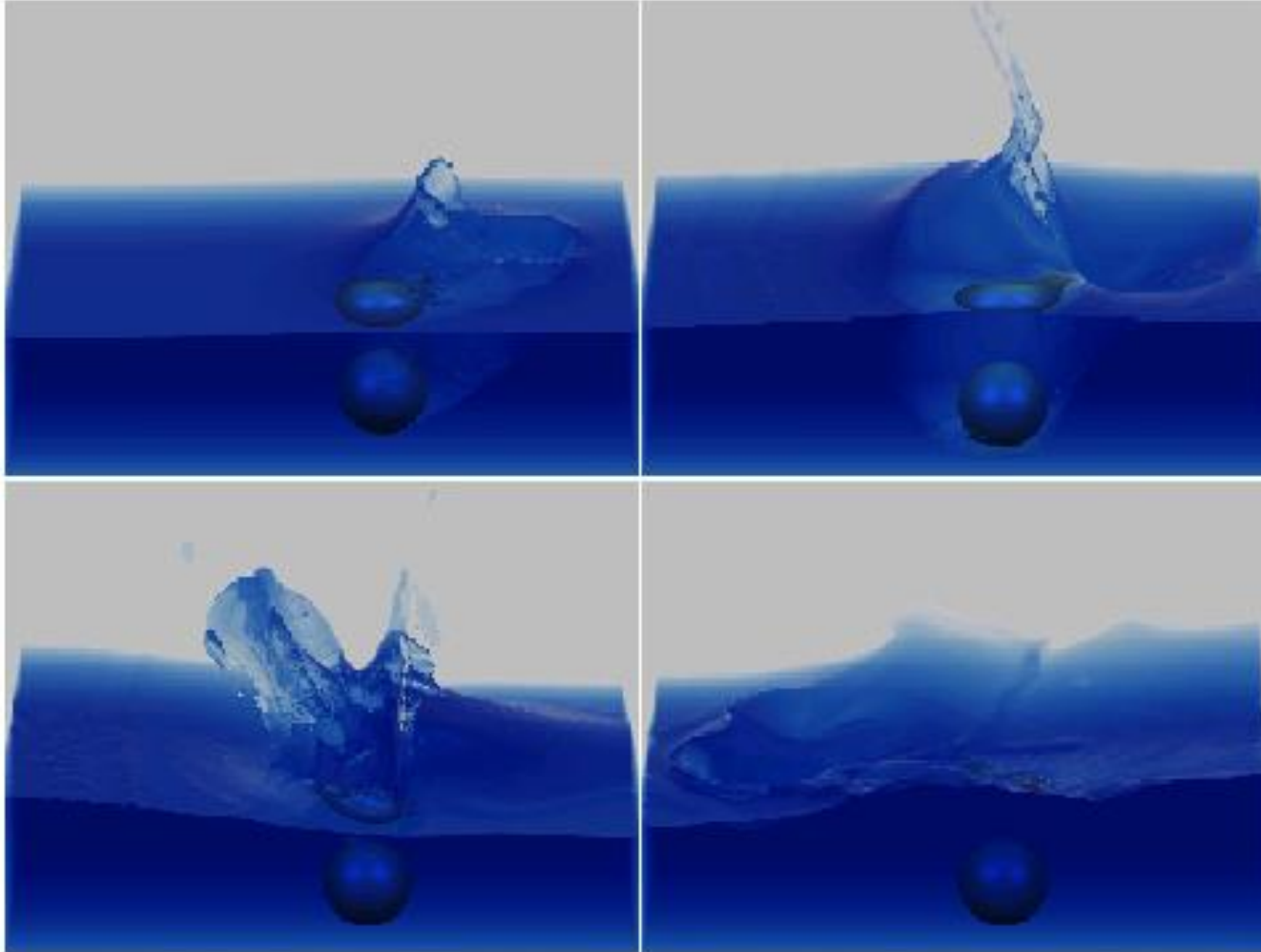
Cloth Modeling



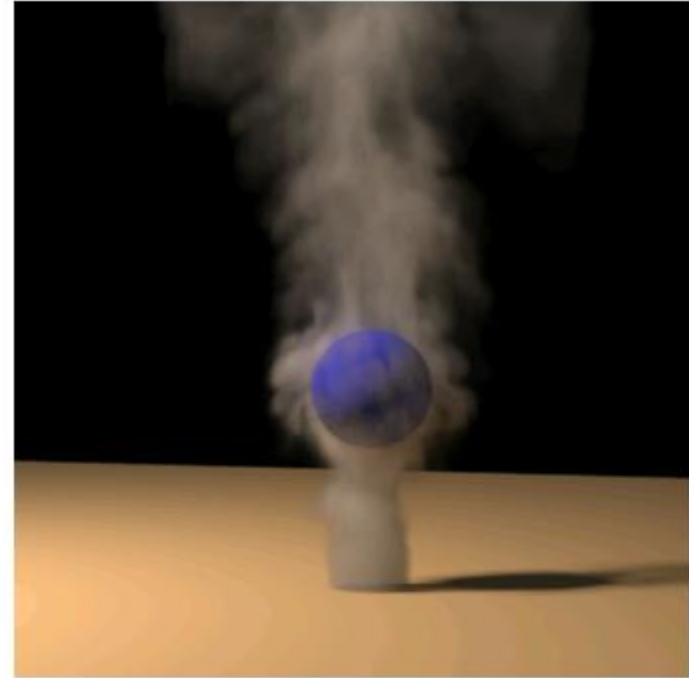


Visual Effects Simulation

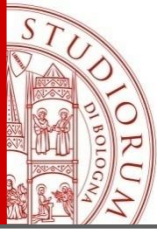




Images from Foster & Fedkiw
SIGGRAPH 2001



Images from Fedkiw, Stam, Jensen, SIGGRAPH 2001



Physically-based animation

Modeling a real movement with:

kinematics

Study of motion without considering the forces that have caused.

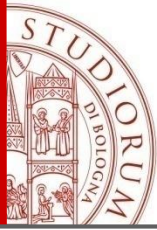
Law of evolution: the equation of motion in time-controlled functions of position, velocity or acceleration.

dynamics

simulation of rigid bodies. Using forces and moments to control the movement of the characters.

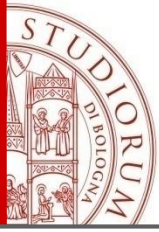
Physical reaction of rigid bodies subjected to forces (gravity, viscosity, friction, ..)

Ex: particle system (animate multiple objects)



What is a Particle System?

- Collection of many small simple pointlike things
 - Described by their current state: position, velocity, age, color, etc.
- Particle motion influenced by external force fields and internal forces between particles
- Particles created by **generators** or **emitters**
 - With some randomness
- Particles often have lifetimes
- Particles are often independent
- Treat as points for dynamics, but rendered as anything you want



Particle systems (Reeves 1983)

Array of

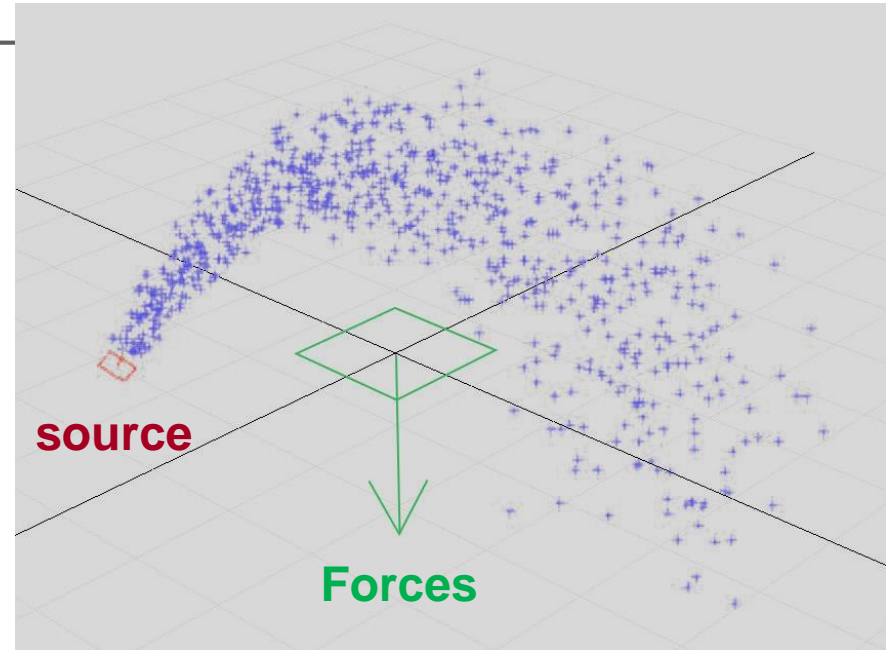
Particle Attributes

Position,
velocity,
accumulated forces,
mass

Shape (if any)

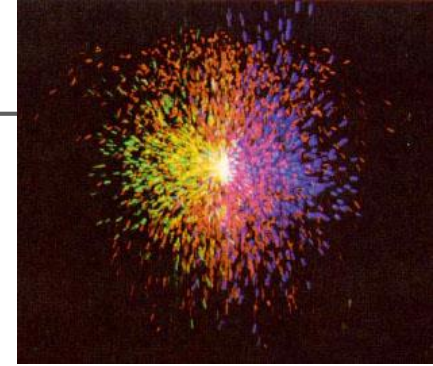
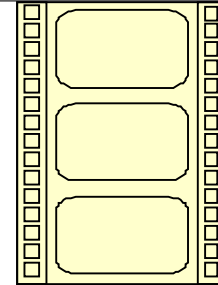
Display attributes (e.g. color, transparency)

Life expectancy (pseudo random).

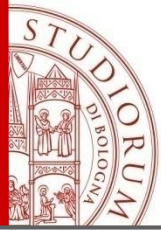


<http://processing.org/learning/topics/simpleparticlesystem.html>

One frame of motion



1. **Generate** new particles
2. Assign **attributes** to each particle
3. **Terminate** any particle with expired life span
4. **Animate** remaining particles
5. **Change** shading parameters
6. **Render** the particles



Newtonian Mechanics

- Basic governing equation $\vec{f} = m \vec{a} = m \frac{d^2 \vec{x}}{dt^2}$

- We know f and m , want to solve for x

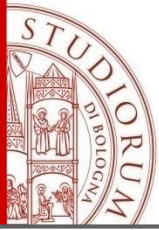
$$\frac{d^2 x}{dt^2} = \vec{f} / m$$

- Corresponds to system of first order ODEs

$$\begin{cases} \frac{d}{dt} \vec{x} = \vec{v} \\ \frac{d}{dt} \vec{v} = \frac{\vec{f}}{m} \end{cases}$$

- Let's stack the pair (x,v) into a state vector \mathbf{X}

$$\mathbf{X} = \begin{bmatrix} \vec{x} \\ \vec{v} \end{bmatrix} \rightarrow \frac{d}{dt} \mathbf{X} = \mathbf{F}(\mathbf{X}, t) = \begin{bmatrix} \vec{v} \\ \vec{f}(x, v) / m \end{bmatrix}$$

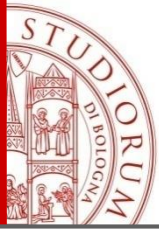


Now, Many Particles

- We have N point masses
- Let's just stack all \mathbf{x} s and \mathbf{v} s in a big vector of length $6N$
- \mathbf{f}_i denotes the force on particle i
- When particles don't interact, \mathbf{f}_i only depends on \mathbf{x}_i and \mathbf{v}_i .

$$X = \begin{bmatrix} x_1 \\ v_1 \\ \cdot \\ \cdot \\ x_N \\ v_N \end{bmatrix} \quad F(X, t) = \begin{bmatrix} v_1 \\ f_1(X, t) \\ \cdot \\ \cdot \\ v_N \\ f_N(X, t) \end{bmatrix}$$

$$\frac{d}{dt} X = F(X, t)$$

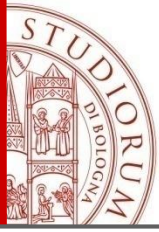


Numerical solution of ODEs

Discretize the time interval, $\Delta t = \textit{time step}$

$$\begin{aligned} t_{n+1} &= t_n + \Delta t \\ (*) \quad X_{n+1} &= X_n + \Delta t F(X_n, t_n) \end{aligned}$$

```
While (1) {  
    Compute force  $F_n$  at  $t_n$ ;  
    Compute pos.  $X_{n+1}$ , vel.  $V_{n+1}$  by solving (*)  
    Update particle positions  $X$ ;  
    Update time  $t_{n+1} = t_n + \Delta t$ ;  
}
```

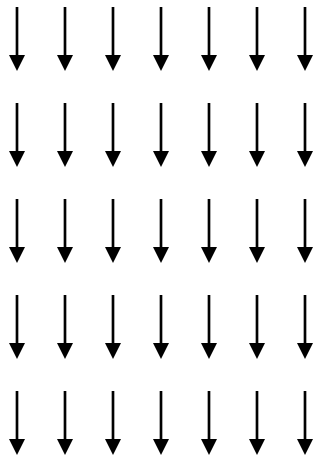


What is a Force?

- A force changes the motion of the system
- For point masses, forces are vectors
 - I.e., to get total force, take vector sum of everything
 - Gravity
 - Simple drag
 - Particle interactions: particles mutually attract and/or repel (es. spring forces)
 - Internal Forces
 - Wind forces
 - User interaction

External Forces

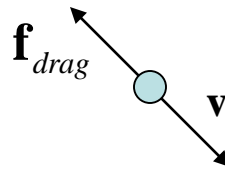
Forces that depend on the single particle



Gravity

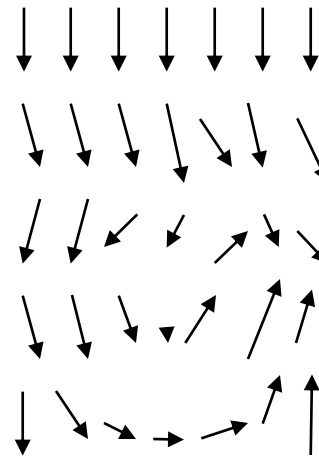
$$\mathbf{f} = m \mathbf{g}$$

$$g = 9.81 \text{ m/s}^2$$



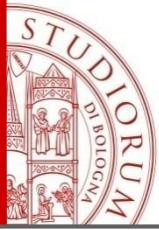
Viscosity/
Dampening

$$\mathbf{f}_{drag} = -k_d \mathbf{v}$$



Wind Fields

$$\mathbf{f} = k \mathbf{v}_{wind}$$



Internal Forces

$$m \ddot{x} = f_{internal} + f_{external} = -\frac{\partial E}{\partial x} + \sum_i f_i$$

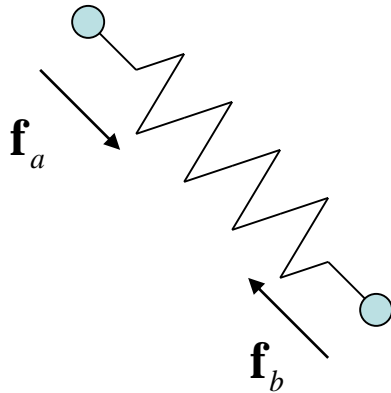
- The internal forces originate from deformations
- Deformations -> changes in potential energy
Stretch / Shear / Bending
- Derivative of the potential energy = force

$$E = E_{stretch} + E_{shear} + E_{bending}$$

$$m_i \ddot{x}_i = -\frac{\partial E}{\partial x} \Big|_{x=x_i} = -\left[\frac{\partial E_{stretch}}{\partial x} + \frac{\partial E_{shear}}{\partial x} + \frac{\partial E_{bending}}{\partial x} \right]_{x=x_i}$$

External/Internal Forces

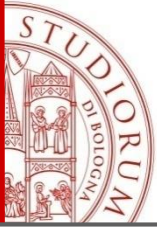
Forces that depend on two particles



$$\mathbf{f}_a = -k_s (|\mathbf{x}_a - \mathbf{x}_b| - l_0) \frac{\mathbf{x}_a - \mathbf{x}_b}{|\mathbf{x}_a - \mathbf{x}_b|} - k_d \left(\frac{(\mathbf{v}_a - \mathbf{v}_b) \cdot (\mathbf{x}_a - \mathbf{x}_b)}{|\mathbf{x}_a - \mathbf{x}_b|} \right) \frac{\mathbf{x}_a - \mathbf{x}_b}{|\mathbf{x}_a - \mathbf{x}_b|}$$

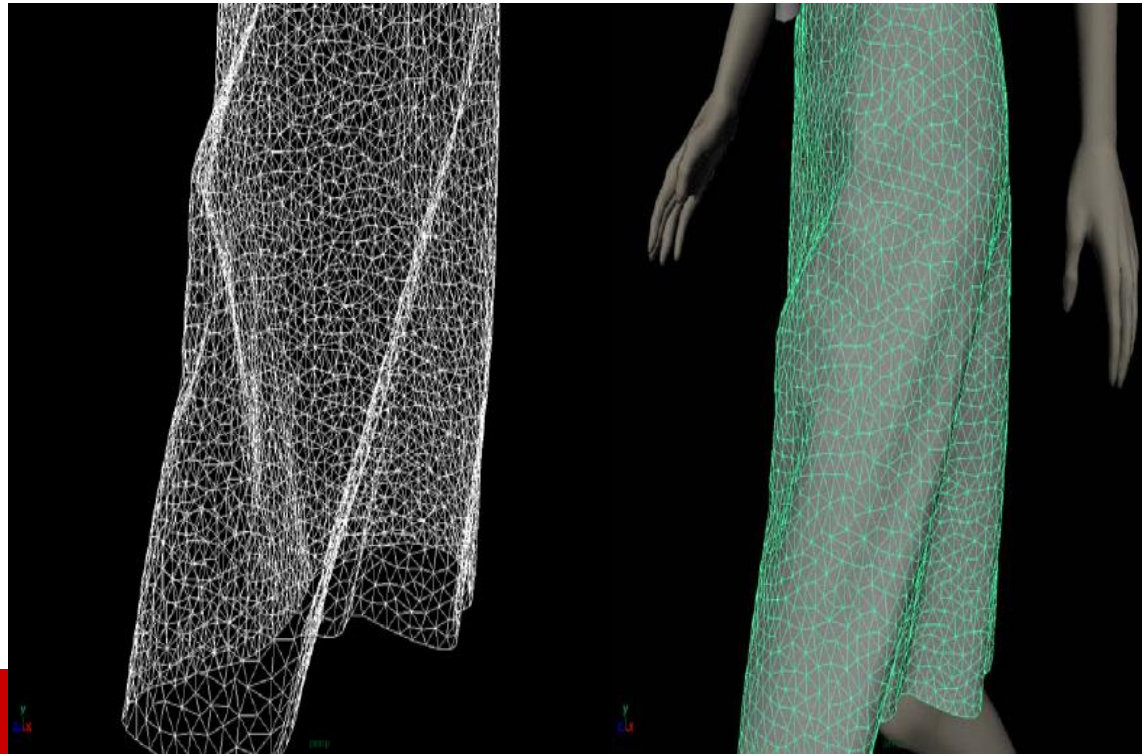
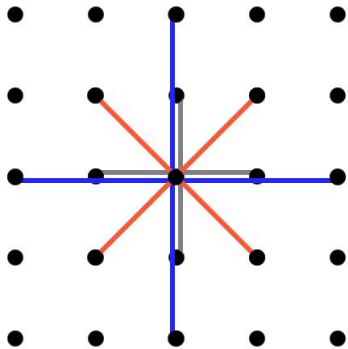
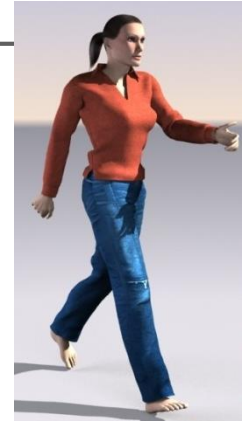
Springs

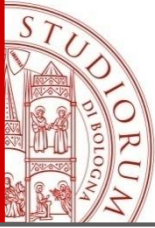
Generalizes to forces that depend on more particles



Clothes as particle system

- clothes=mesh
- Each **vertex** of the geometry of the cloth is a particle with mass.
- Each particle is linked to 12 particles by bonds elastic.





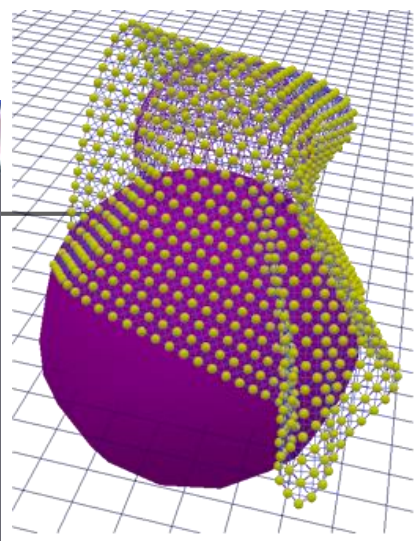
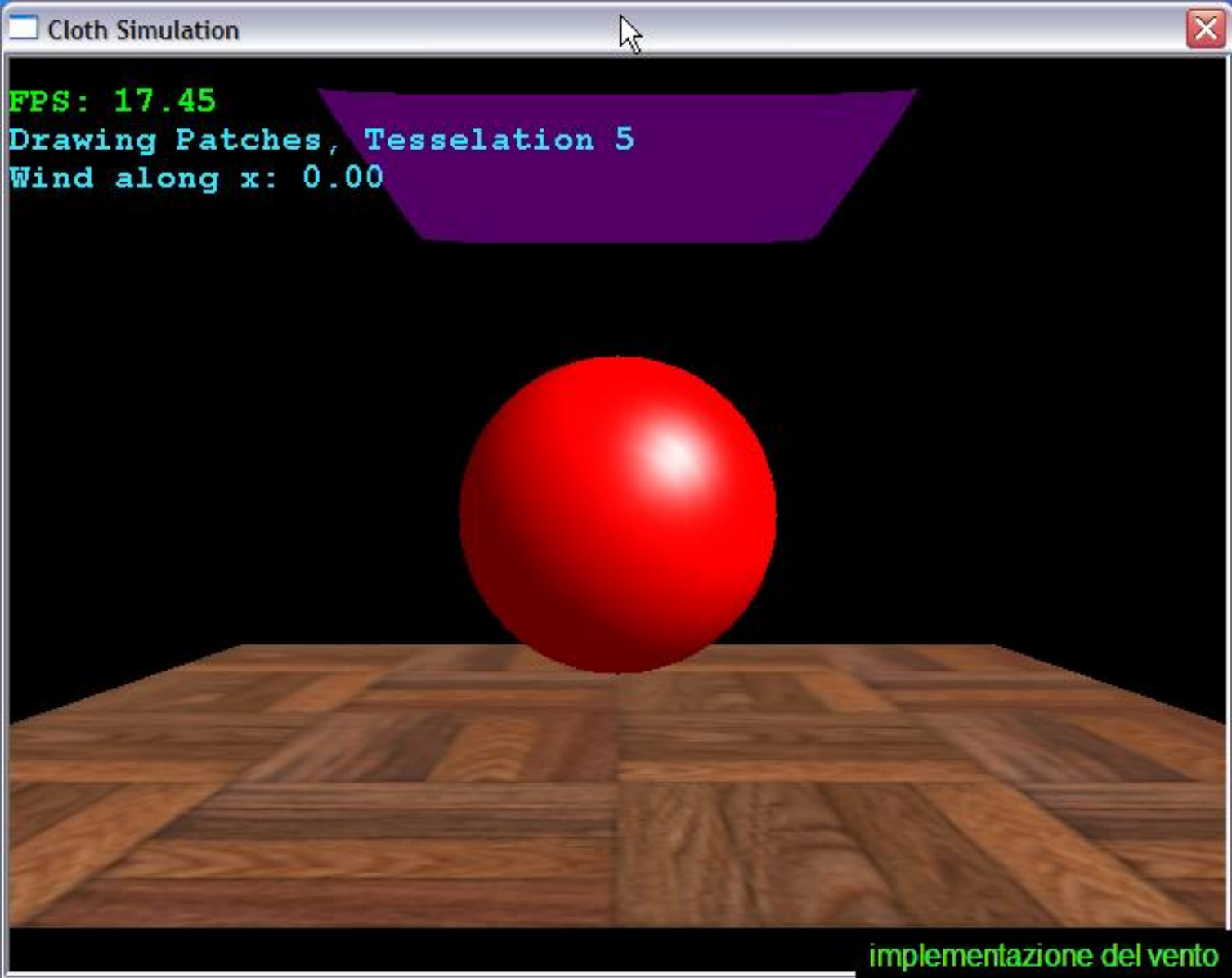
Clothes as particle system

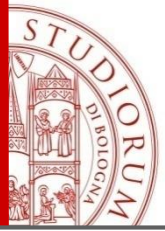
Dynamic Simulation

The behavior of the tissue is described by the **position** and **velocity** of the particles in each frame of the animation.

Repeat

1. Update position of particle
2. Draw the mesh from particles (vertices)





FA Motion + cloth motion



forces cloth to intersect both itself and the body. (d) Without GIA, a cloth-cloth inter-
the sleeve. (e) The same frame as (d), but using GIA, the cloth doesn't snag as the arm

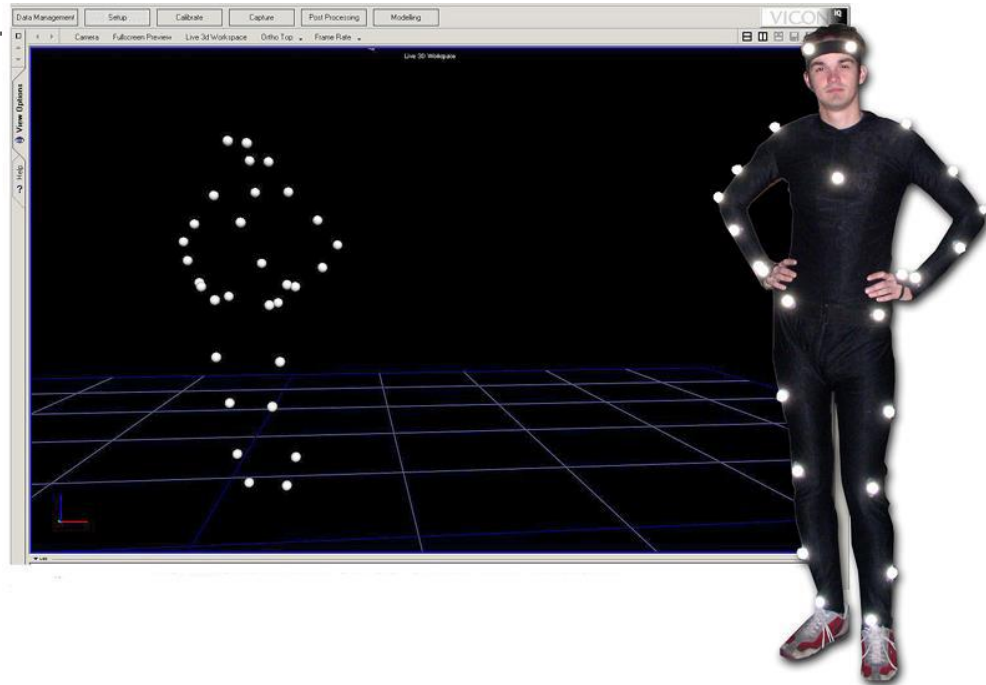
Further reading:

- [Witkin, Baraff, Kass: Physically-based Modeling Course Notes, SIGGRAPH 2001](#)
- [William Reeves: Particle systems—a technique for modeling a class of fuzzy objects, Proc. SIGGRAPH 1983](#) (The original paper on particle systems)
- particlesystems.org
- ParticleFX (Max Payne Particle Editor) You can download it (for Windows) and easily create your own particle systems.

Motion Capture (MOCAP)



Andy Serkis in digital Character "Gollum" in "The lord of the rings"

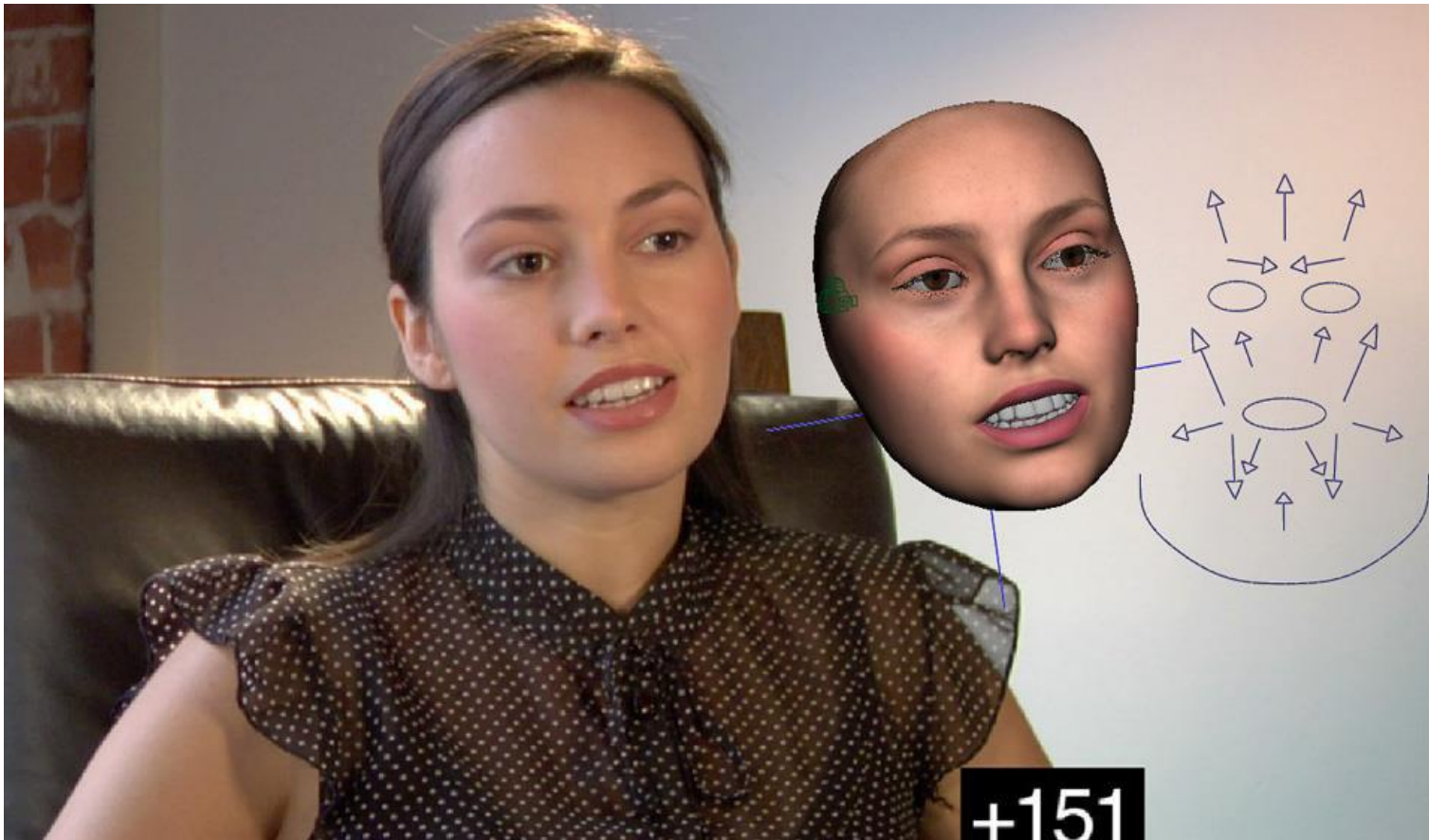


MOCAP is an effective 3D animation tool for realistically capturing human motion

<https://www.youtube.com/watch?v=zQPfxcQKr0Q>

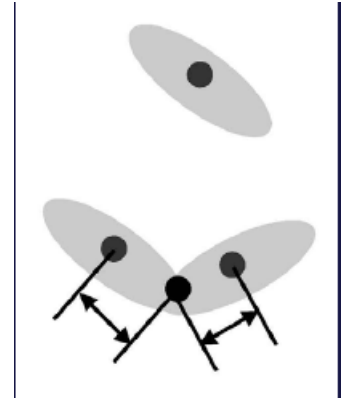
Facial mocap

<https://www.youtube.com/watch?v=piJ4Zke7EUw>



The MOCAP Pipeline

- Calibration
- Capture
 - Sampling reference points in time
- 3D Position Reconstruction
 - Convert them in JOINT ANGLES
- Fitting to the Skeleton
 - Use the joints to control the virtual model (IK)
- Post Processing



Animators could use more than 750 controls to create Shrek's performance. Some controlled one joint or muscle, others controlled groups of several.

Mocap: technologies

Optical

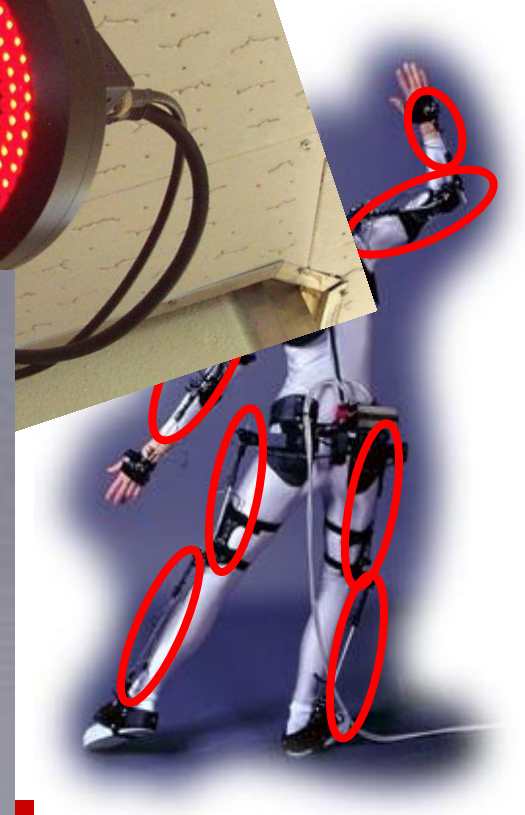
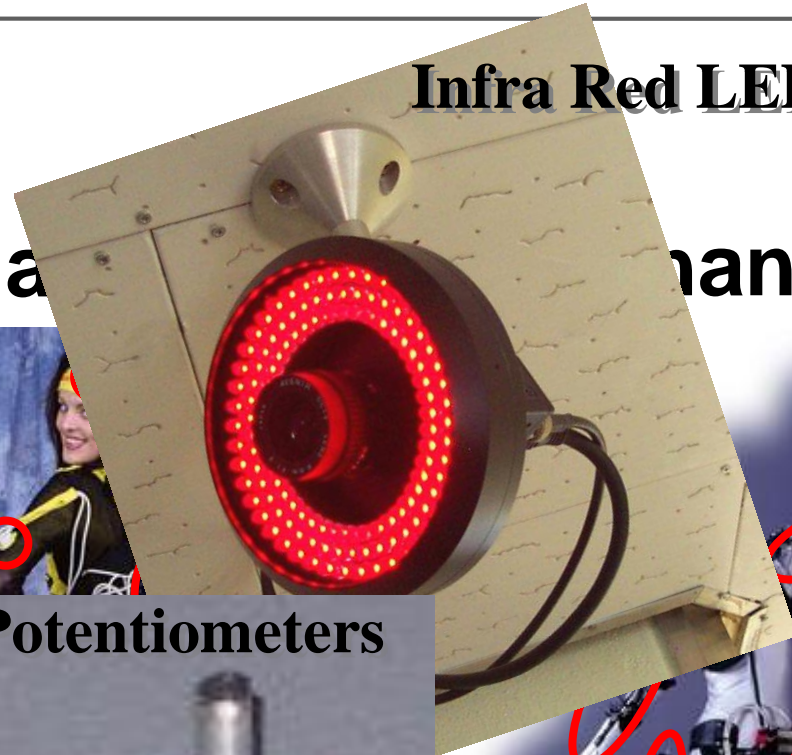
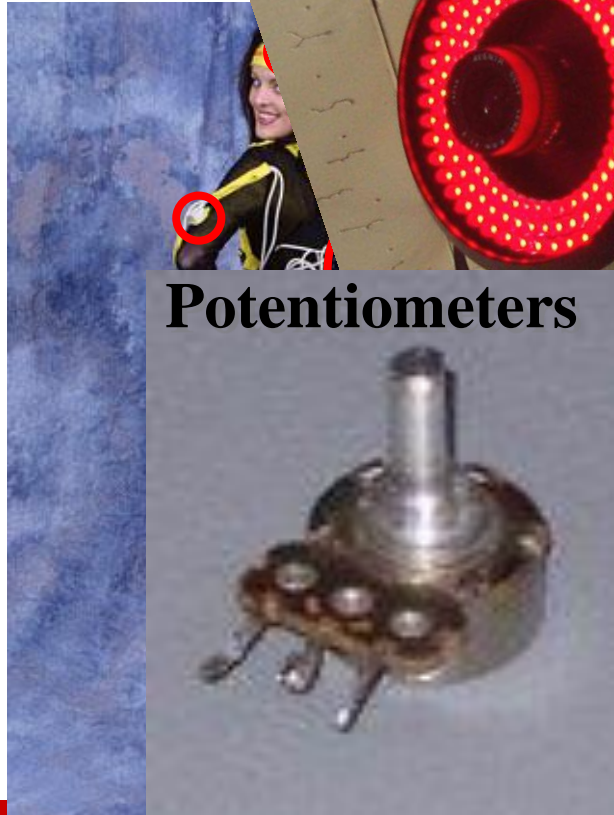
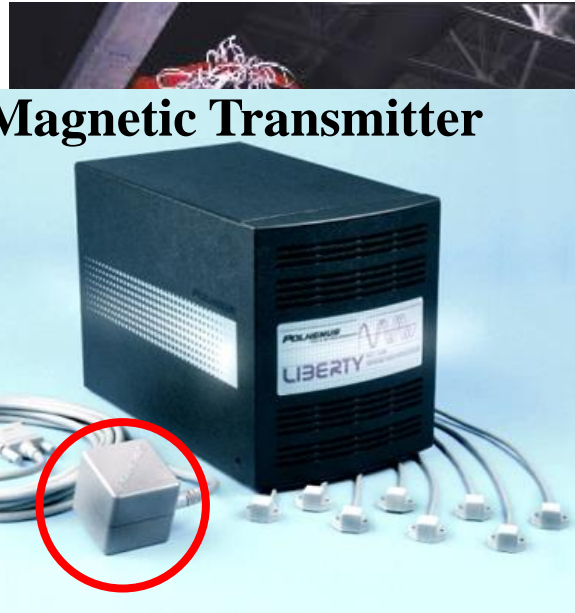
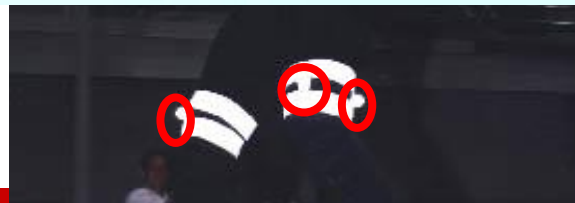
Magnetic

Infra Red LEDs

Mechanical

Magnetic Transmitter

Potentiometers





1) Technology: electromagnetic

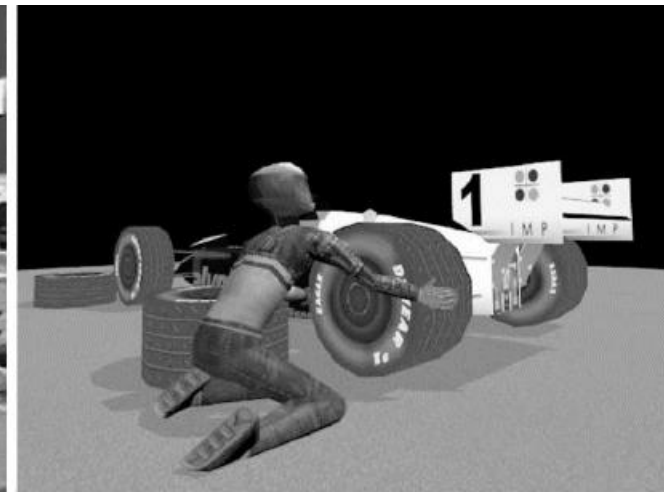
Provide the data directly:

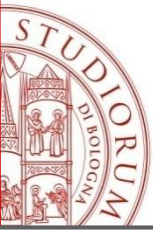
- Position / orientation sensors on the object
- Sensors transmit data to CPU
- Cable or wireless

Cons:

- Specialized equipment
- Limitations in the motion
- Expensive
- Magnetic field distortions

Real time systems: fast





2) Technology: Optical

More convenient for the user

No restriction in motion

Cheaper

More complex processing

Image processing

identify markers

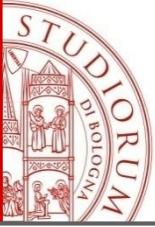
Computer vision techniques

get marker positions

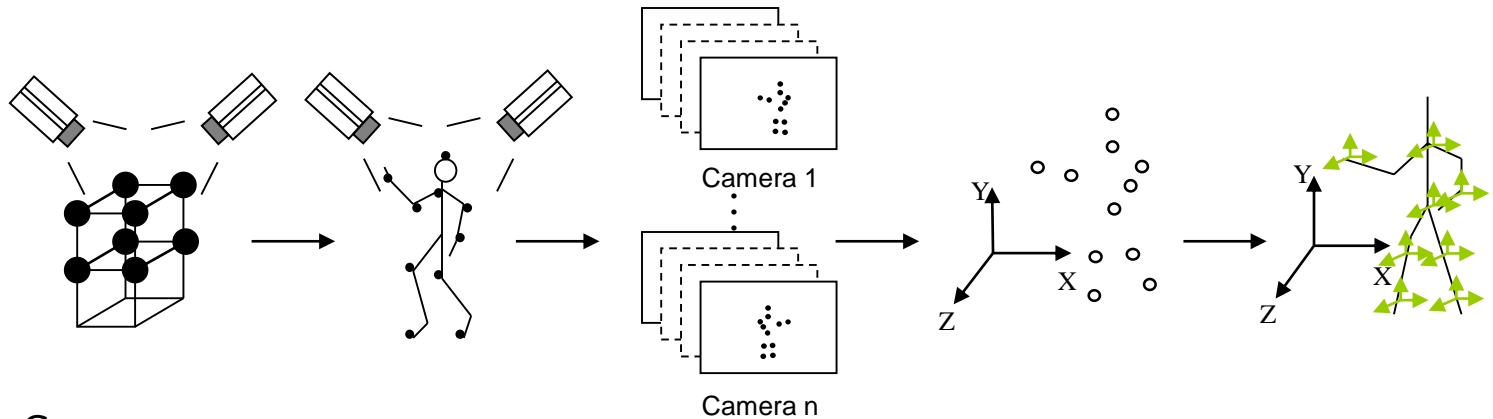


Vicon

MotionAnalysis



Optical Mocup: pipeline



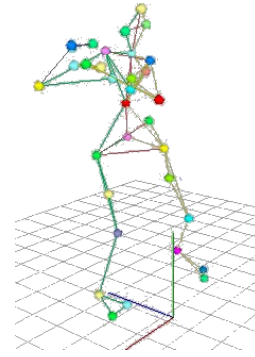
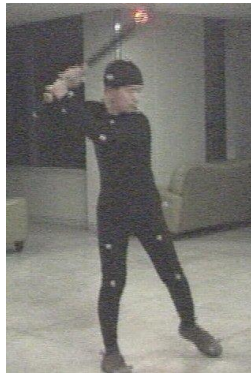
Camera calibration

Capture

Image acquisition

Matching & tracking

Post-processing



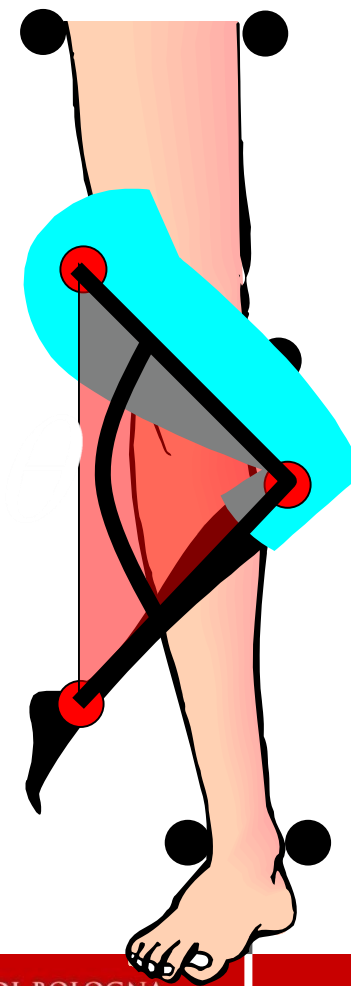
obtain the location of each camera in WCS

Fitting to the Skeleton

Give the skeleton that represents the virtual character, assign the motion of each marker acquired

Utopia approach: 1 marker --- 1 joint

- Markers on both sides of each joint
- Joint Displacement
- Use Rotation Angles Only

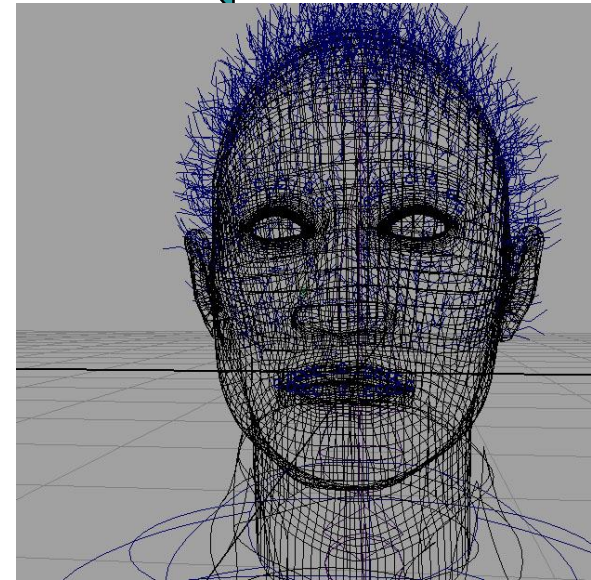
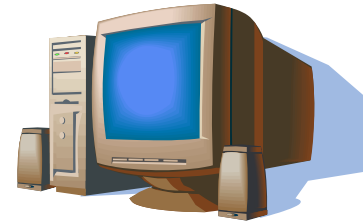
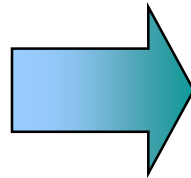
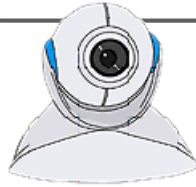
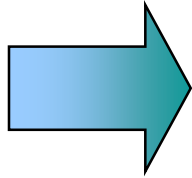


Post Processing

- **Motion Editing**
 - Cut, Copy, Paste
- **Motion Warping**
 - Speed up or Slow Down
 - Rotate, Scale or Translate
- **Motion Signal Processing**
 - Smoother Motions

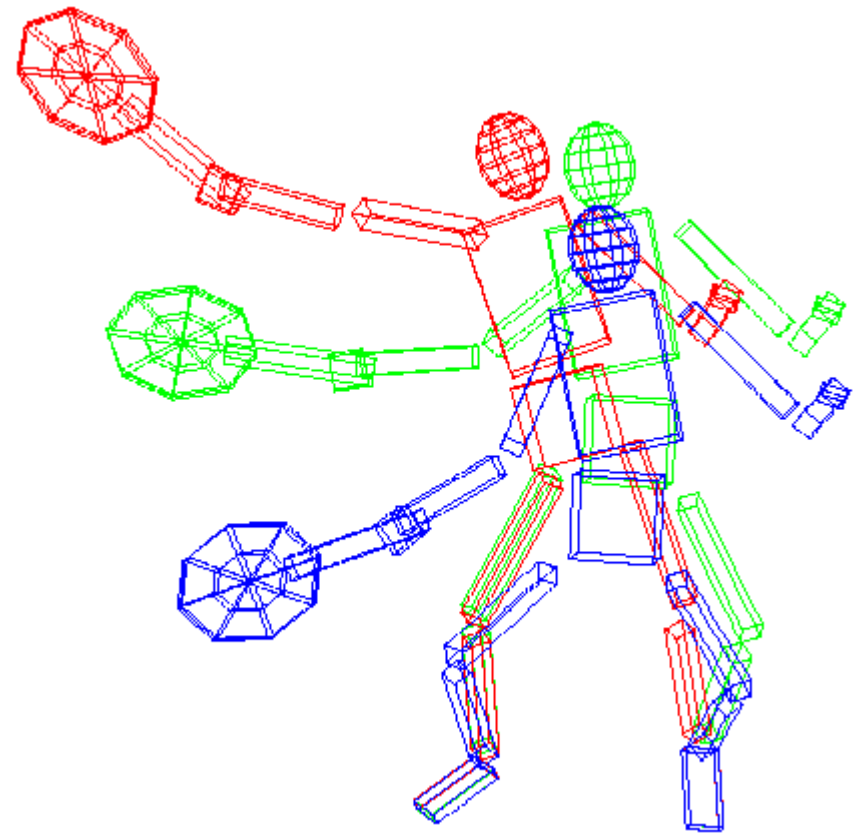


Motion Capture



Keyframe Animation

- Basic idea:
 - specify important events only, computer fills in the rest via interpolation/approximation



Generating Hand-Drawn Animation

- Senior artist draws *keyframes*
- Assistant draws *inbetweens*

keyframe



keyframe



keyframe

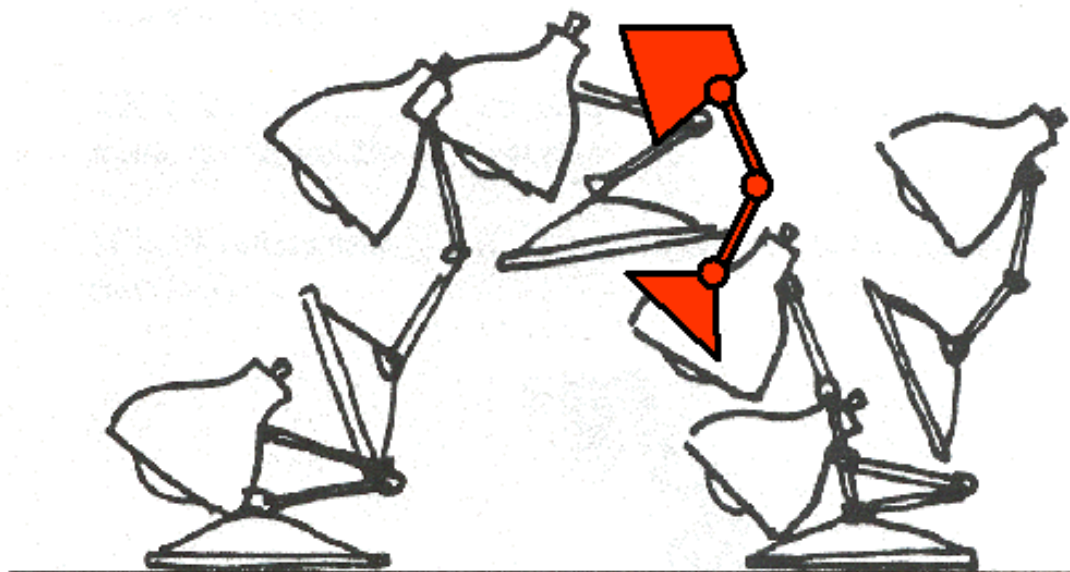


***inbetweens* ("tweening")**

Keyframe

- **Keyframe:**

describe the motion of objects over time starting from a set of key positions of the object



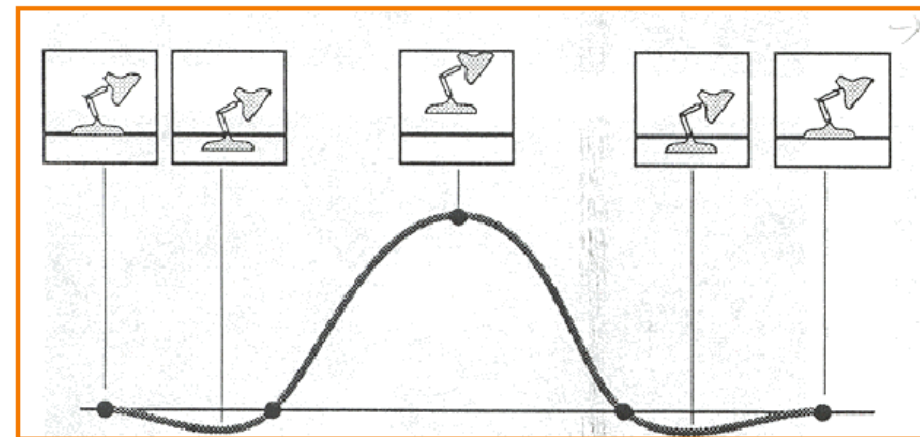
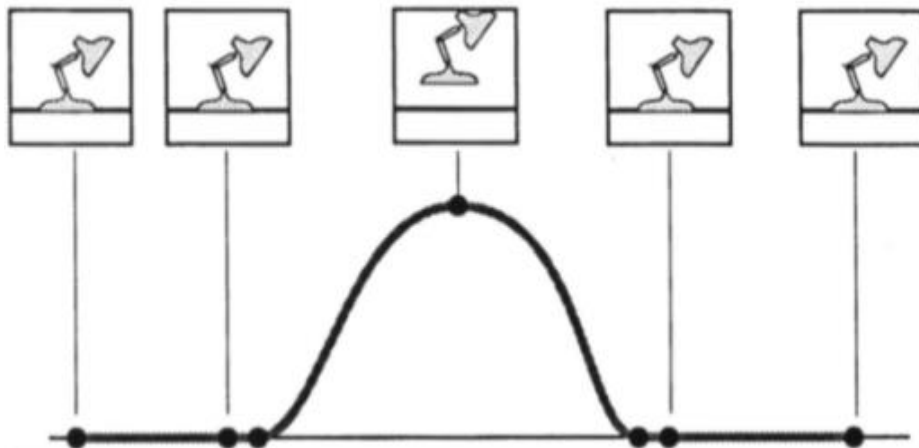
- **Key and inbetween:**

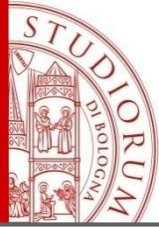
- Define the **key** frame,
- Compute the **inbetween** (frames) to get motion continuity

Keyframe : ALGORITHM

- assign values to the parameters characterizing the model for each key frame
- **Inbetweening**: interpolate values to obtain the intermediate frames. For position values use linear interpolation, cubic spline curve.

The curve parameterization controls the speed of animation





Parameters for Keyframe

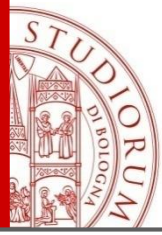
What exactly are we interpolating?

Position, orientation, material properties (color / texture), brightness, shape (for non-rigid objects)

How to get the values of the parameters?

- Set by the animator;
- Captured (mocap)
- Procedural

The quality of the motion depends heavily on the skill of the animator



Animating an articulated figure

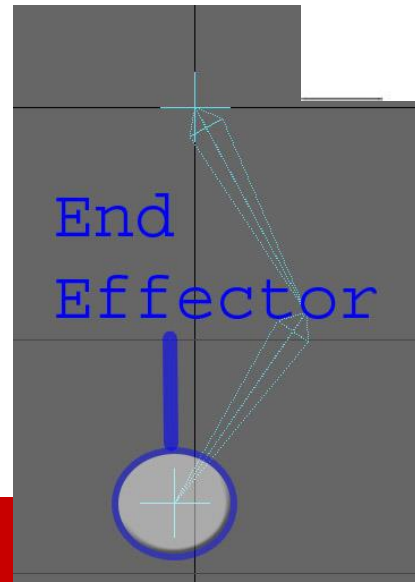
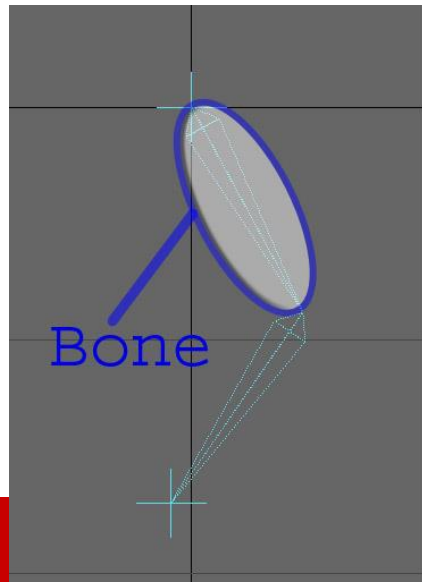
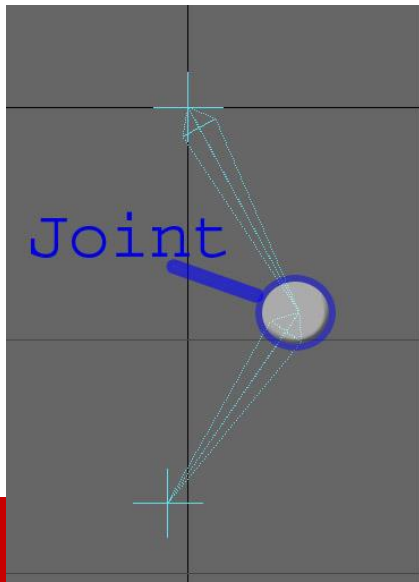
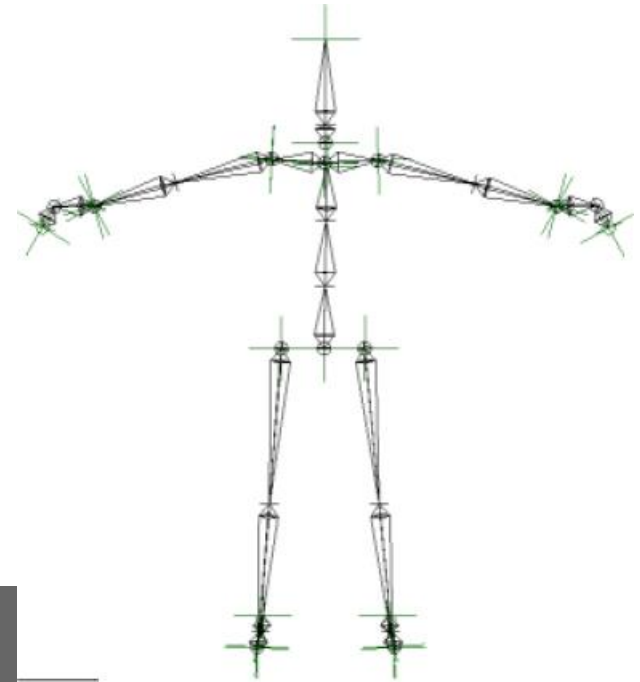
Animate the skeleton, consisting of:

Rigid parts BONES (**LINK**)

connected by **JOINT**

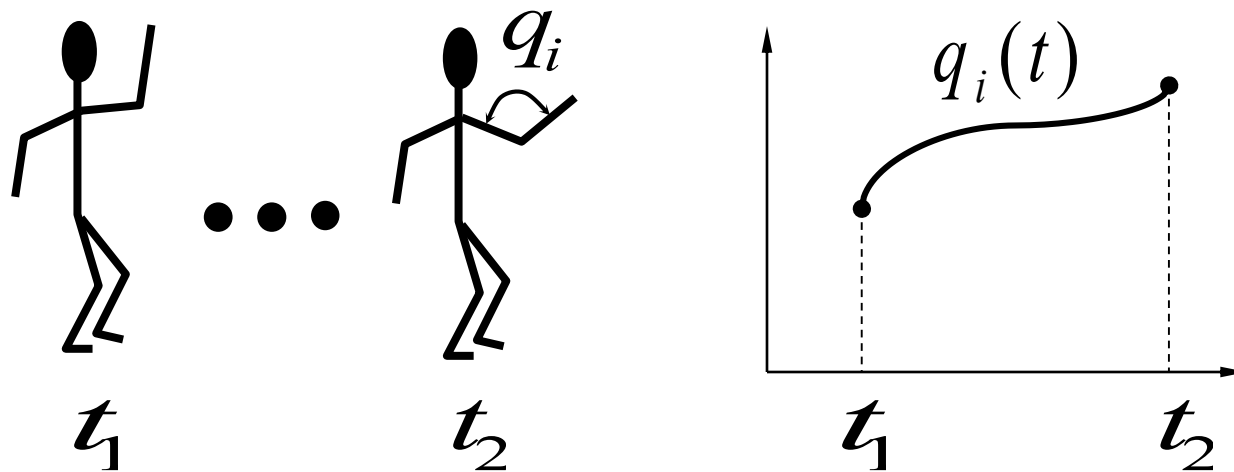
Ends of the chain joint and link

are called **END-EFFECTOR**

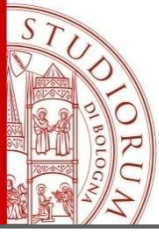


Animating an articulated figure

The evaluation of the hierarchical model, by means of tree traversal, produces the figure in a position that reflects a determined setting of the parameters of the joint. The set of parameters is called the **pose**. A pose is specified by a vector containing a value for each joint.



To animate an articulated figure, by changing its pose, we can modify the joint as function of time.

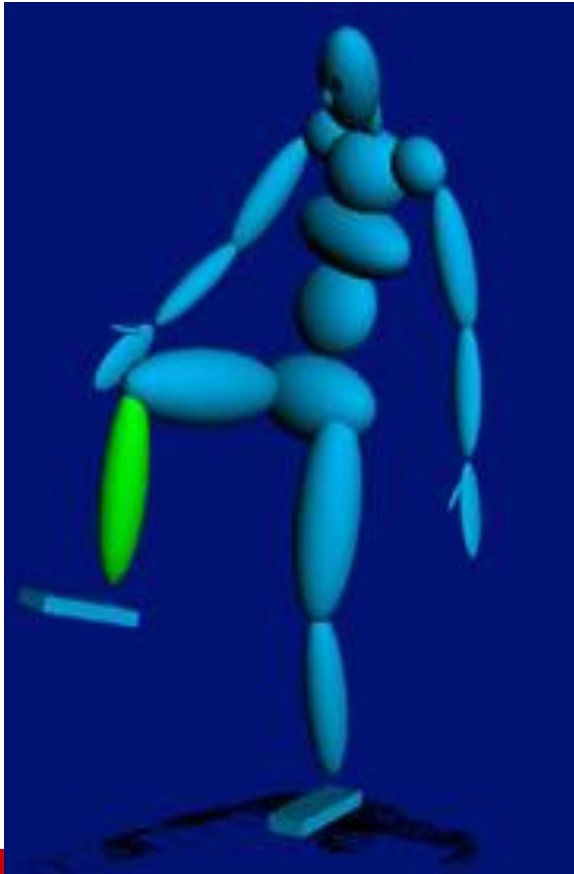


DOF in human model

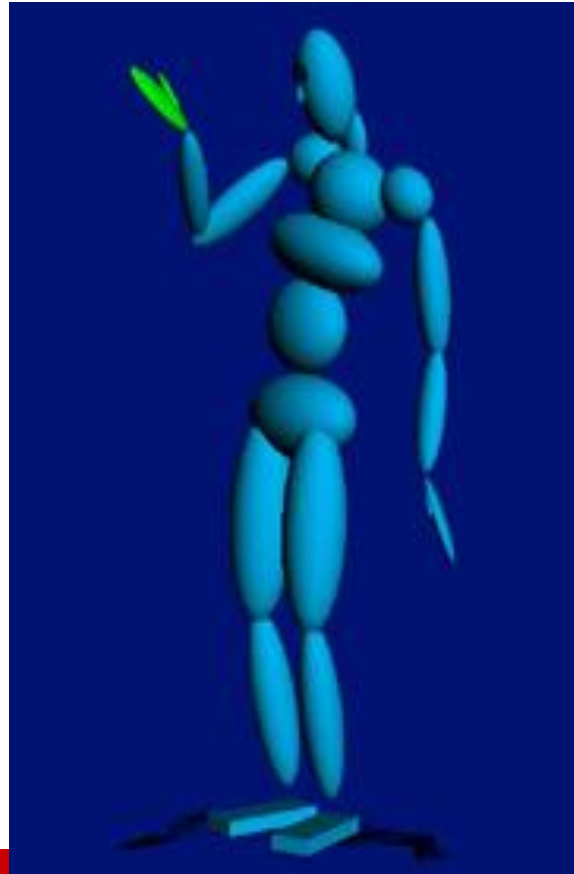
Each joint can have up to 6 DOFs

3 translation, 3 rotation $\theta=(x,y,z,\mu,\phi,\Phi)$.

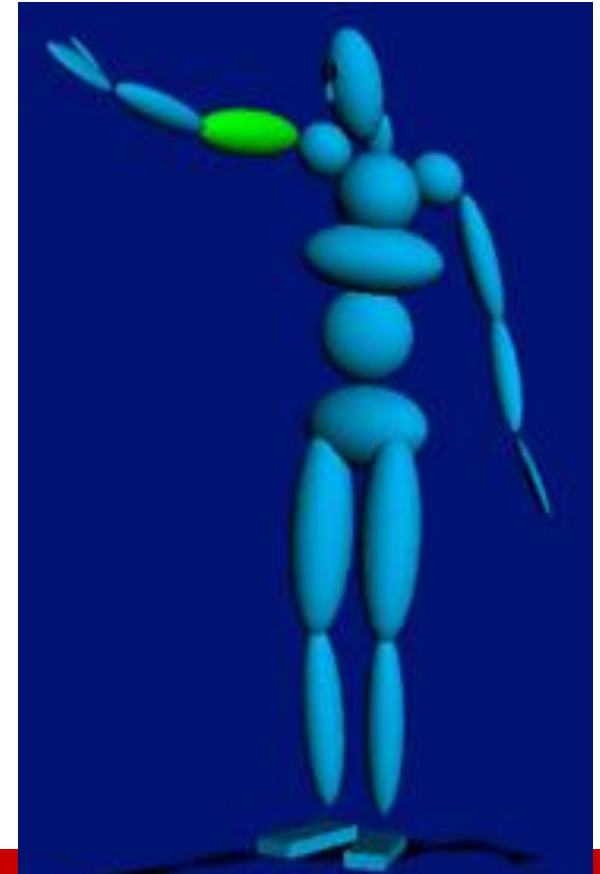
1 DOF: knee

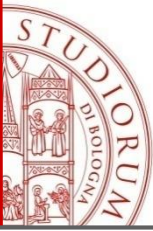


2 DOF: wrist



3 DOF: arm





How to define KEYFRAME?

How to change the parameters over time to achieve the desired motion?

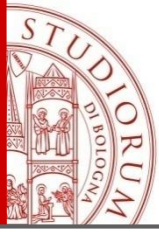
There are two ways to modify a pose of an articulated figure: direct and inverse kinematics

Direct Kinematics (FK)

The animator has complete control over the entire chain but must assign (turn) manually each joint.

Inverse Kinematics (IK)

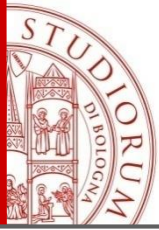
The animator controls only the last term of the chain (end-effector) and delegates to the software responsible for placing the remaining joints to reach the final pose.



How to generate the Inbetween?

By Interpolation!

- In order to “move things”, we need both translation and rotation
- Interpolating the translation is easy, but what about rotations?
- Finding the most natural and compact way to represent rotation and orientations



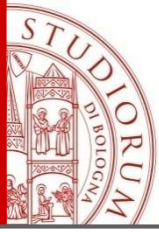
Representation and Interpolation of orientation

- orientation = rotation matrix 4x4
- How about interpolating each entry of the rotation matrix?

Given the matrices M_i and M_{i+1} at time t_i , and t_{i+1}
compute $M(t)$ for t in $[t_i, t_{i+1}]$ such that

$$M(t_i) = M_i \text{ and } M(t_{i+1}) = M_{i+1}$$

- $M(t)$ is an intermediate orientation given by interpolating M_i and M_{i+1}



Interpolation: KO!

Example: M_0 identity and M_1 is 90° around x

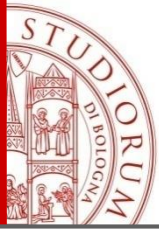
$$\text{Interpolate} \left(\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{bmatrix} \right) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0.5 & 0.5 \\ 0 & -0.5 & 0.5 \end{bmatrix}$$

Is the resulting matrix a rotational matrix?

NO:

RR^T is not the identity matrix.

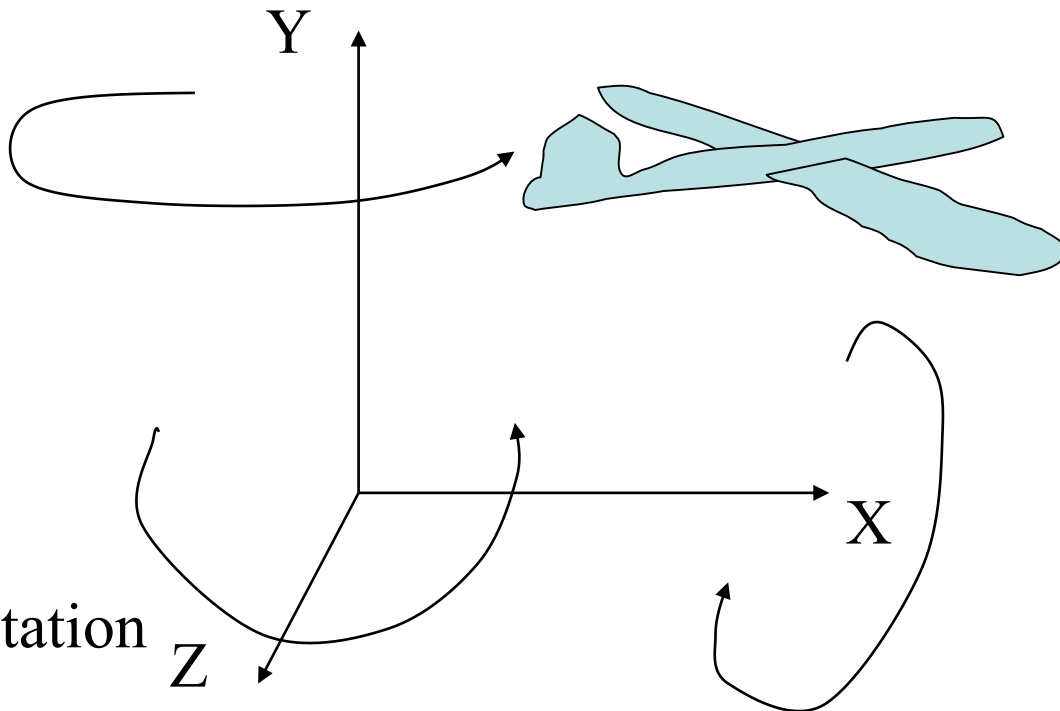
The interpolated matrix might no longer be orthonormal, leading to nonsense for the in-between rotations



Fixed Angle Representation

Angles used to rotate about fixed global axes.

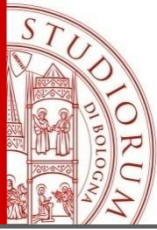
Y-axis rotation



X-axis
rotation

Z-axis rotation

Orientations are specified by a set of 3 ordered parameters that represent 3 ordered rotations about fixed axes, i.e. first about x, then y, then z



Fixed Angle Representation

$$(\theta_x, \theta_y, \theta_z) = R_z R_y R_x$$

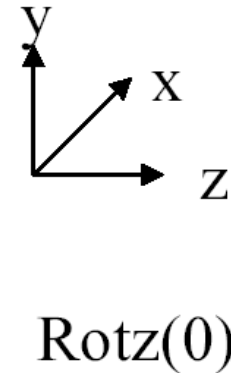
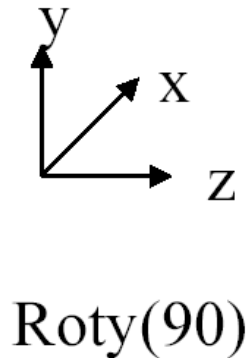
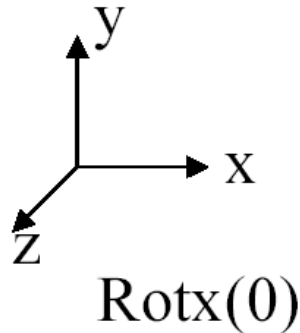
- Rotate θ_x wrt x
- Rotate θ_y wrt y
- Rotate θ_z wrt z

Many possible orderings, don't have to use all 3 axes

- $(y, z, x), (x, z, y), (z, y, x) \dots$

Gimbal Lock

Problem occurs when two of the axes of rotation line up on top of each other. This is called “Gimbal Lock”

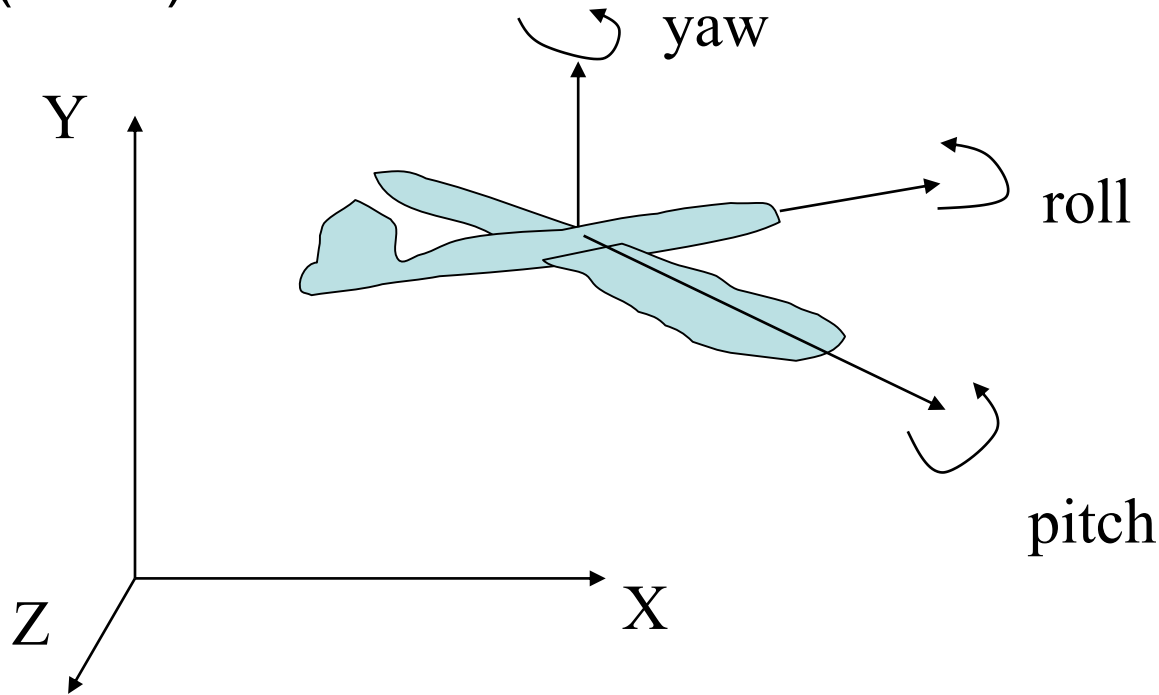


- A 90 degree rotation about the y axis essentially makes the first axis of rotation align with the third.
- Incremental changes in x,z produce the same results – you’ve lost a degree of freedom

Euler Angles

Same as fixed angles, except now the axes move with the object (Local)

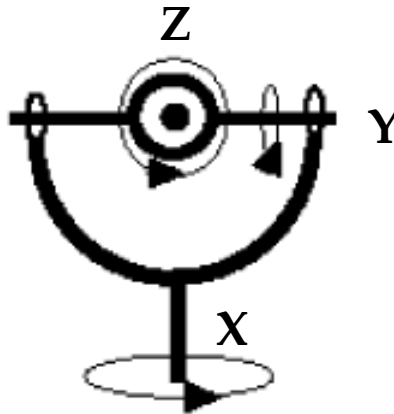
Roll
Pitch
Yaw

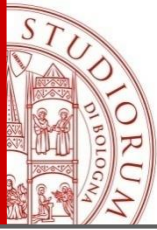


A general rotation is a combination of three elementary rotations: around the x-axis (x-roll), around the y-axis (y-roll) and around the z-axis (z-roll).

Euler angle vs. fixed angle

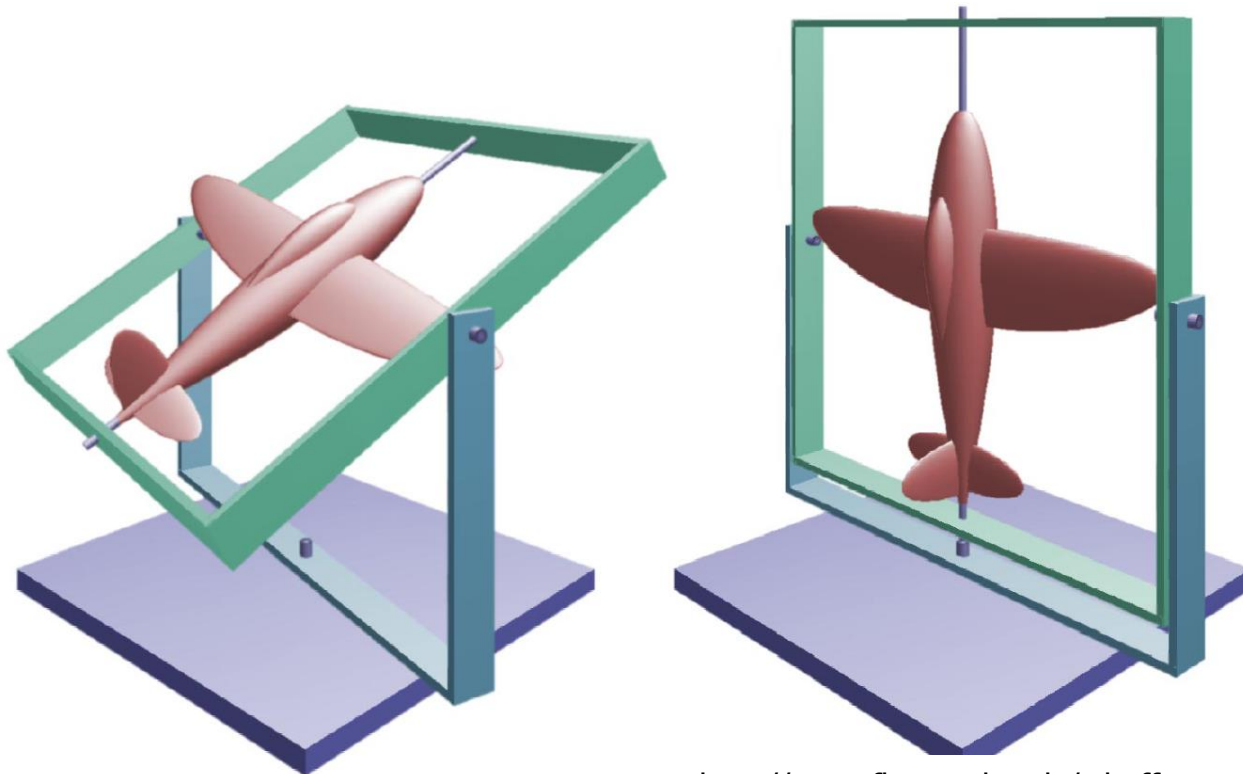
- $\mathbf{R}_z(90)\mathbf{R}_y(60)\mathbf{R}_x(30) = \mathbf{E}_x(30)\mathbf{E}_y(60)\mathbf{E}_z(90)$
- Euler angle rotations about moving axes written in reverse order are the same as the fixed axis rotations





Gimbal Lock (again!)

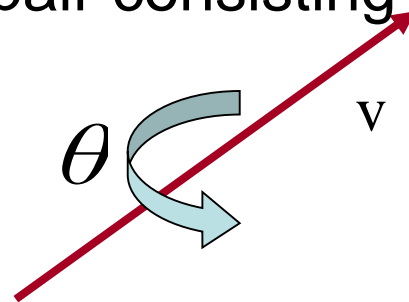
- When two rotational axis of an object point in the same direction, the rotation ends up losing one degree of freedom
- Rotation by 90° causes a loss of a degree of freedom



Quaternion

Any orientation can be represented by a 4-tuple of real numbers: $q=[w,x,y,z]$ or, better, by a pair consisting of the real value and the 3D vector

$$q = \left[\cos\left(\frac{\theta}{2}\right), v\left(\sin\left(\frac{\theta}{2}\right)\right) \right]$$



Given

Rotation axis $\rightarrow v=(ax, ay, az)$ /* unit vector */
angle \rightarrow theta (radianti)

Then

$$w = \cos(\theta/2)$$

$$x = ax * \sin(\theta/2)$$

$$y = ay * \sin(\theta/2)$$

$$z = az * \sin(\theta/2)$$



Quaternion q

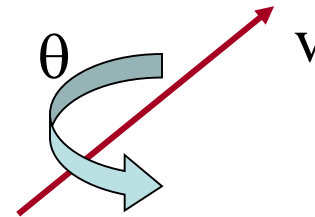
Example $q = [\cos(\pi/2), \sin(\pi/2)(1,0,0)]$ Rotate π around x axes

$$q = [0, v]$$

Represent a v vector

Given a quaternion

$$q = [w, x, y, z]$$



then

Rotation axes $v = (ax, ay, az)$
Angle \rightarrow theta (radianti)

$$\text{theta} = 2 * \text{acos}(w)$$

$$\text{ax} = x / \text{scale}$$

$$\text{ay} = y / \text{scale}$$

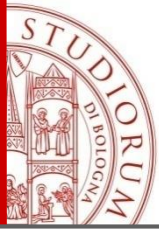
$$\text{az} = z / \text{scale}$$

where $\text{scale} = x^2 + y^2 + z^2$

If scale = 0 \rightarrow no rotation

Thus:

Set the rotation axes to an arbitrary unit vector and rotation angle 0.



Quaternion Math

$$[s_1, v_1] \cdot [s_2, v_2] = [s_1 s_2 - v_1 \cdot v_2, s_1 v_2 + s_2 v_1 + v_1 \times v_2]$$

Quaternions are non commutative

$$[1, (0, 0, 0)] \quad \text{multiplicative identity}$$

$$q^{-1} = \left(\frac{1}{\|q\|} \right)^2 \cdot [s, -v] \quad \|q\| = \sqrt{s^2 + x^2 + y^2 + z^2}$$

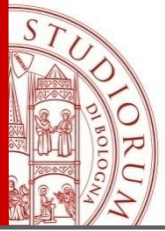
$$q \cdot q^{-1} = [1, (0, 0, 0)] \quad \text{same angle, same vector, opposite direction}$$

To rotate a vector, **v** using quaternion math

- represent the vector as $[0, \mathbf{v}]$
- represent the rotation as a quaternion, q

$$q = \text{Rot}_{\theta, (x, y, z)} = \left[\cos(\theta/2), \sin(\theta/2) \cdot (x, y, z) \right]$$

$$v' = \text{Rot}(v) = q \cdot v \cdot q^{-1}$$

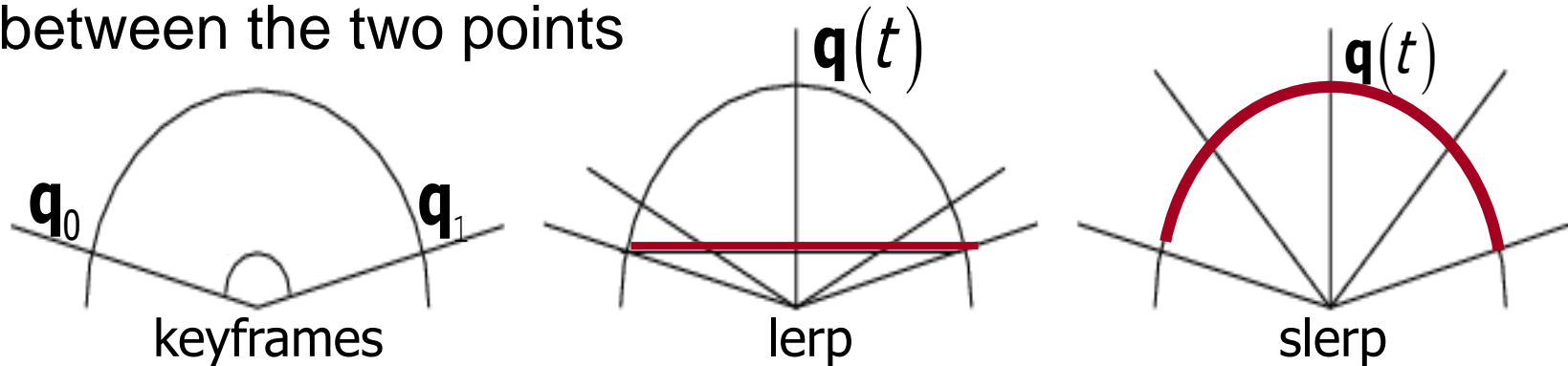


Interpolation between quaternions: Lerping vs. Slerping

linearly interpolated (lerp) intermediate points are not uniformly spaced when projected onto the circle

$$\Rightarrow \text{lerp}(\mathbf{q}_0, \mathbf{q}_1, t) = \mathbf{q}(t) = \mathbf{q}_0(1-t) + \mathbf{q}_1 t$$

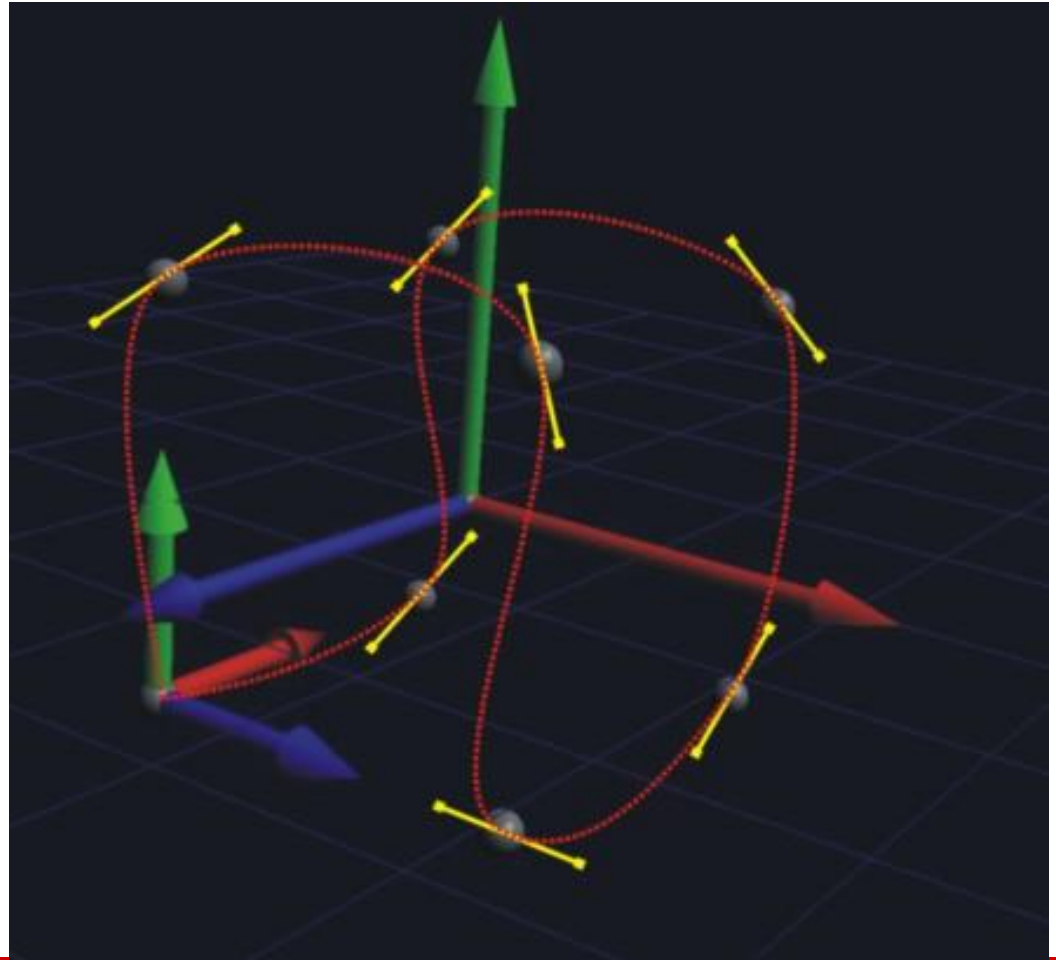
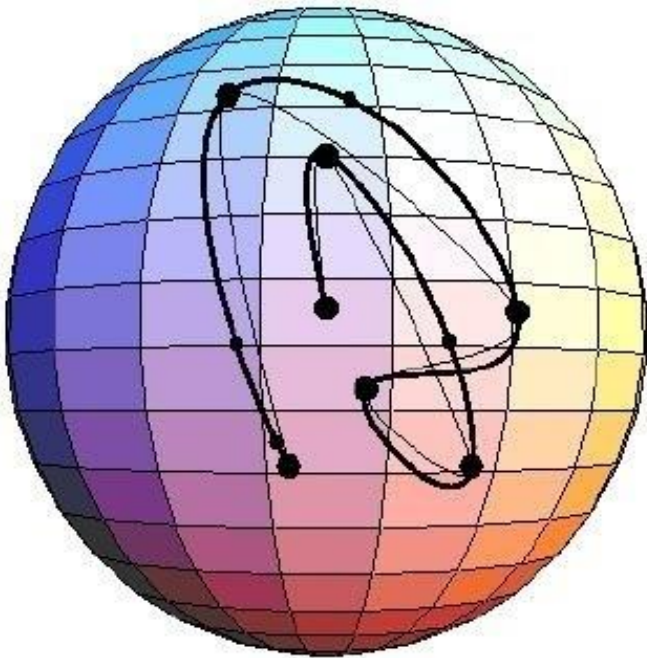
move with constant angular velocity along the great circle between the two points

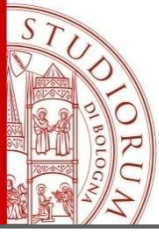


Solution: Spherical linear interpolation (slerp):
interpolate the circular arcs.

$$\text{slerp}(\mathbf{q}_0, \mathbf{q}_1, t) = \mathbf{q}(t) = \frac{\mathbf{q}_0 \sin((1-t)\omega) + \mathbf{q}_1 \sin(t\omega)}{\sin(\omega)},$$

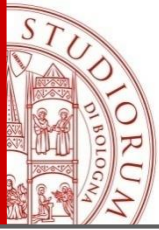
where $\omega = \cos^{-1}(\mathbf{q}_0 \cdot \mathbf{q}_1)$





Keyframe with quaternions

- Each key represented as a **rotation matrix**
- Convert a sequence of matrices in a sequence of **quaternions**
- **Interpolation** between quaternion keys
- Convert the quaternion sequence inbetween in **sequence of rotation matrices**

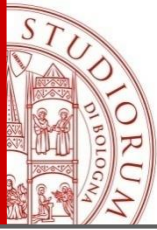


Rotations in Reality

- It's easiest to express rotations in Euler angles or Axis/angle
- We can convert to/from any of these representations
- Choose the best representation for the task
 - input: Euler angles
 - interpolation: quaternions
 - composing rotations: quaternions

Rotate using q_1 and then using q_2 is like rotation using $q_2 \cdot q_1$

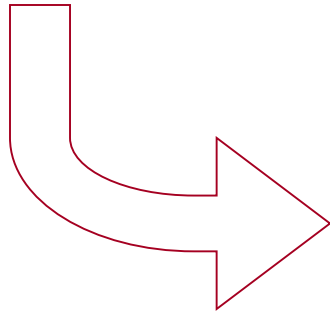
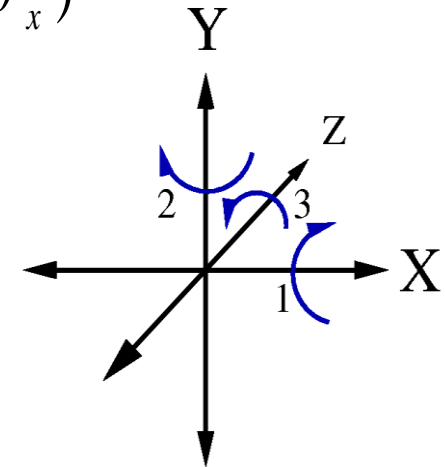
$$\begin{aligned} q_2 \cdot (q_1 \cdot P \cdot q_1^{-1}) \cdot q_2^{-1} &= (q_2 \cdot q_1) \cdot P \cdot (q_1^{-1} \cdot q_2^{-1}) \\ &= (q_2 \cdot q_1) \cdot P \cdot (q_2 \cdot q_1)^{-1} \end{aligned}$$



$$E(\theta_x, \theta_y, \theta_z) = R(z, \theta_z) \cdot R(y, \theta_y) \cdot R(x, \theta_x)$$

Euler Angle

```
glRotatef( angleX, 1, 0, 0)
glRotatef( angleY, 0, 1, 0)
glRotatef( angleZ, 0, 0, 1)
// translate
```



Quaternion

```
// convert Euler in quaternion
// multiply quaternions
// convert resulting quaternion in
// axis/angle
glRotate(theta, ax, ay, az)
// translate
```



Convert

→ Quaternion to Matrix (rotation)

$$q = (w, x, y, z)$$

$$M = \begin{bmatrix} 1 - 2y^2 - 2z^2 & 2xy - 2zw & 2xz + 2yw & 0 \\ 2xy + 2zw & 1 - 2x^2 - 2z^2 & 2yz - 2xw & 0 \\ 2xz - 2yw & 2yz + 2xw & 1 - 2x^2 - 2y^2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Convert

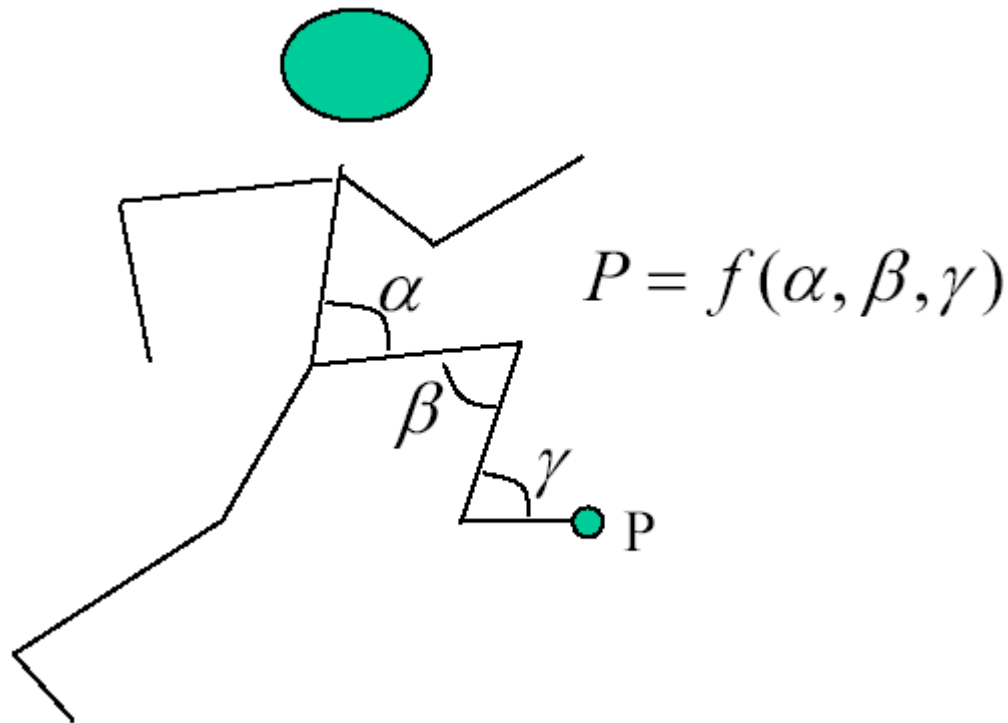
→ Matrix (rotation) to Quaternion

$$w = \frac{1}{2} \sqrt{M_{00} + M_{11} + M_{22} + M_{33}}$$

$$x = \frac{M_{21} - M_{12}}{4w} \quad y = \frac{M_{02} - M_{20}}{4w} \quad z = \frac{M_{10} - M_{01}}{4w}$$

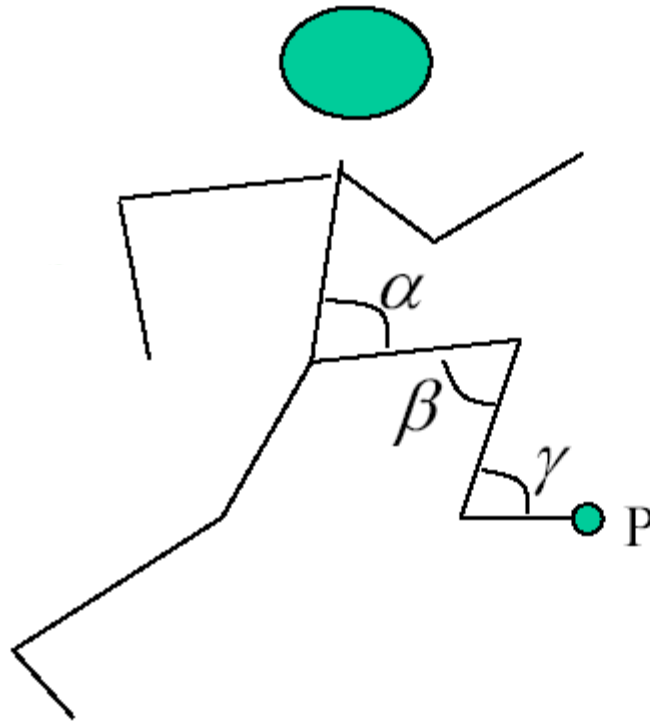
Direct kinematics

Describes the position of each part of the articulated structure by assigning values to the parameters (joint angle) for each key position and interpolating the joint between the key positions.



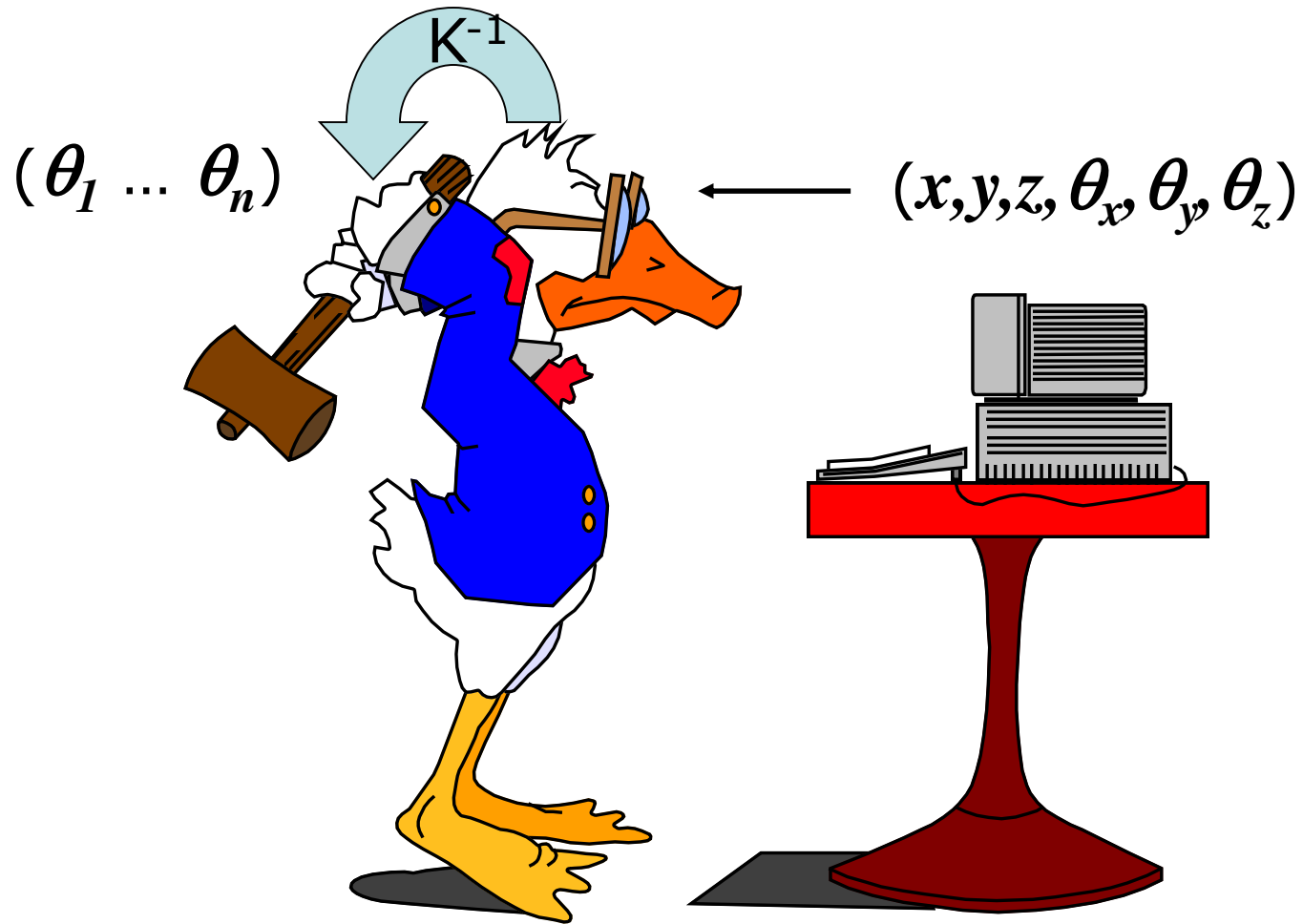
Inverse Kinematics

Specifies the position of the terminal part of the structure. The animator does not indicate how each separate part of the structure must move, all joint angles are then calculated so that the end effector be positioned as required.



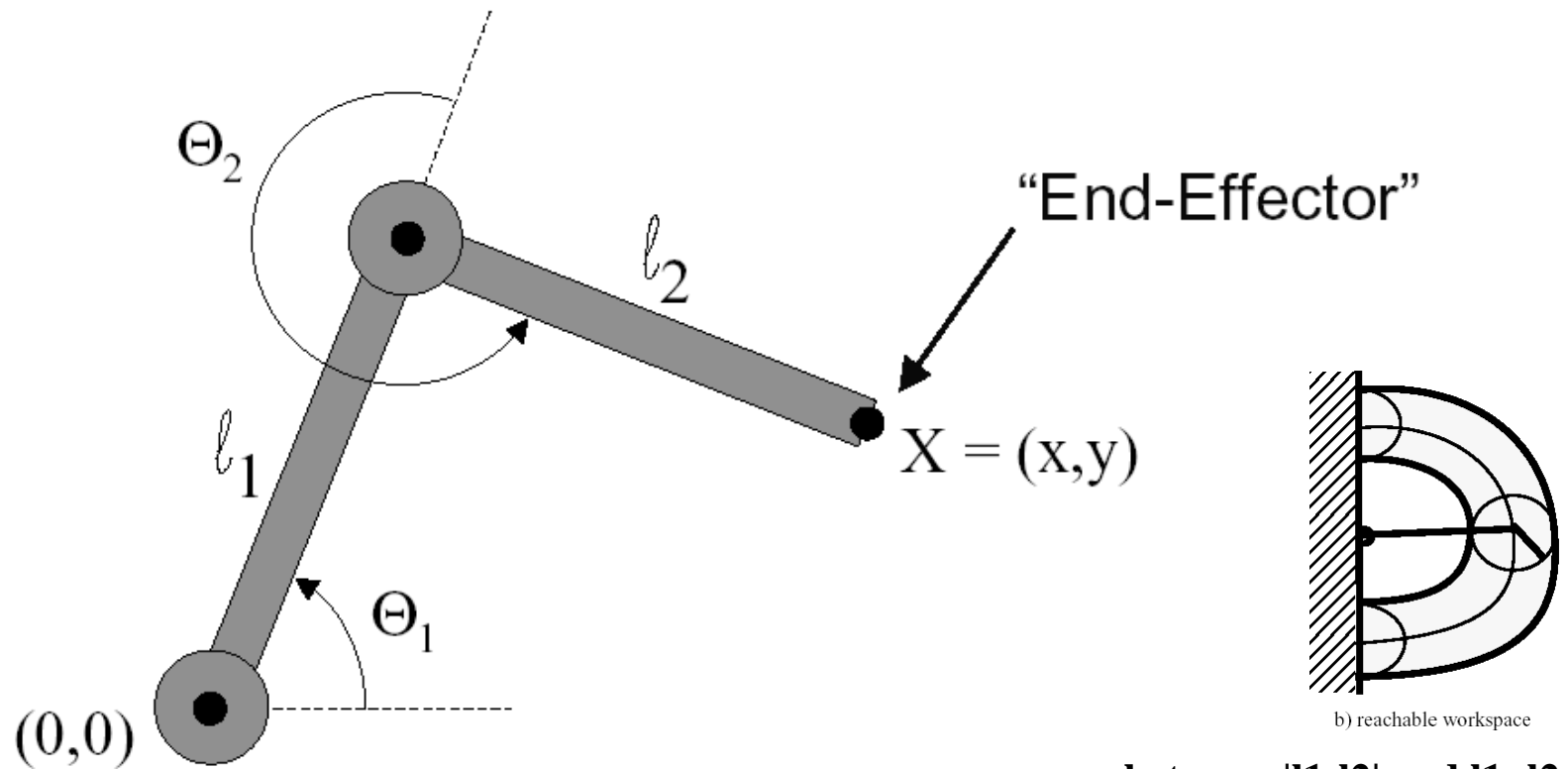
$$\alpha, \beta, \gamma = f^{-1}(P)$$

Inverse Kinematics



Example: 2-Link Structure

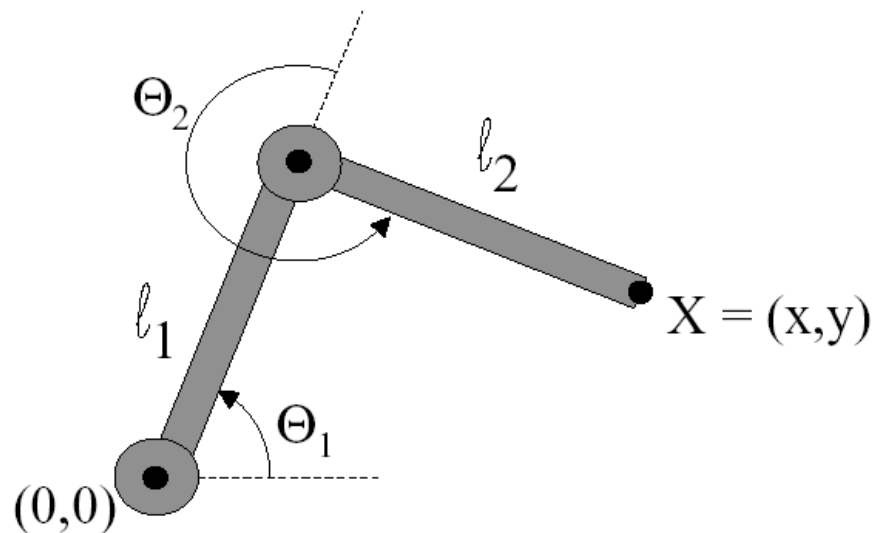
- Two links connected by rotational joints



between $|l_1 - l_2|$ and $l_1 + l_2$

Forward Kinematics

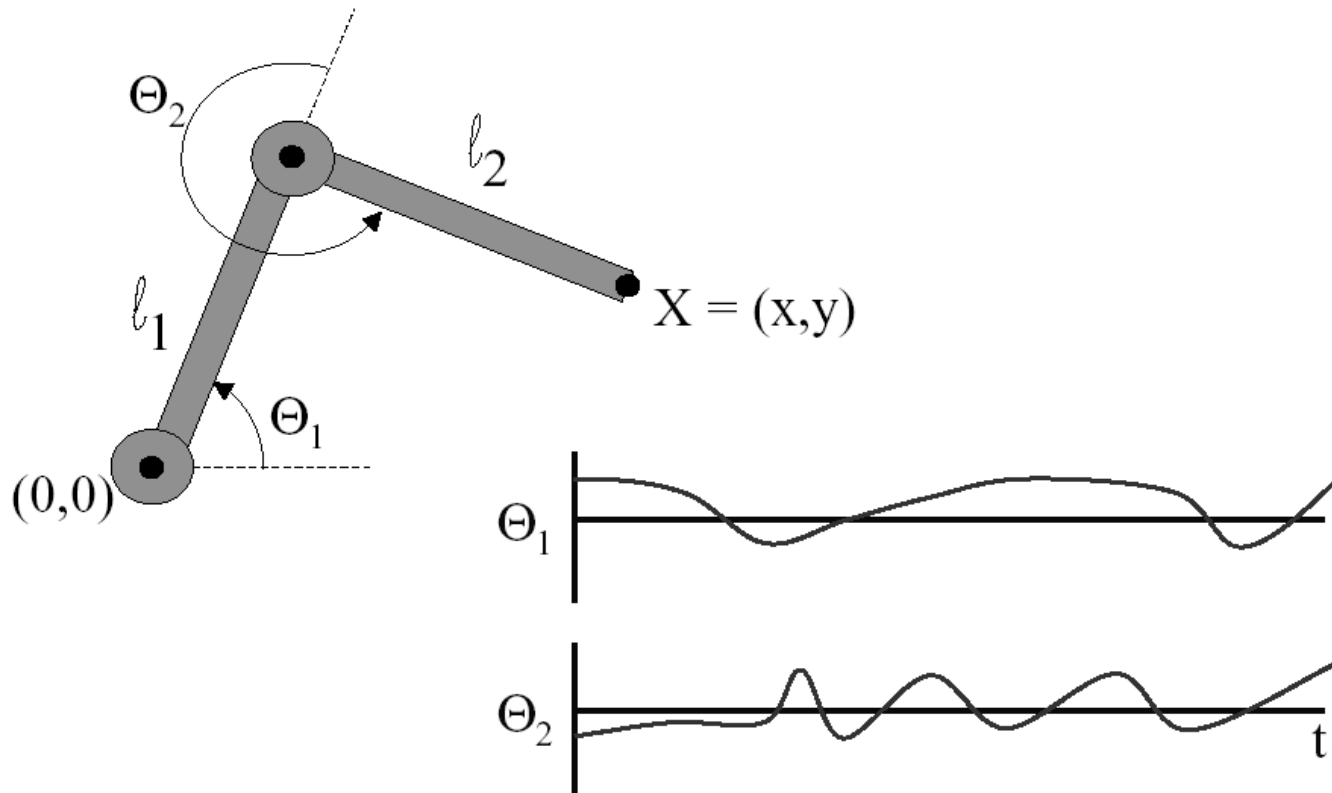
- Animator specifies joint angles: Θ_1 and Θ_2
- Computer finds positions of end-effector: X



$$X = (l_1 \cos \Theta_1 + l_2 \cos(\Theta_1 + \Theta_2), l_1 \sin \Theta_1 + l_2 \sin(\Theta_1 + \Theta_2))$$

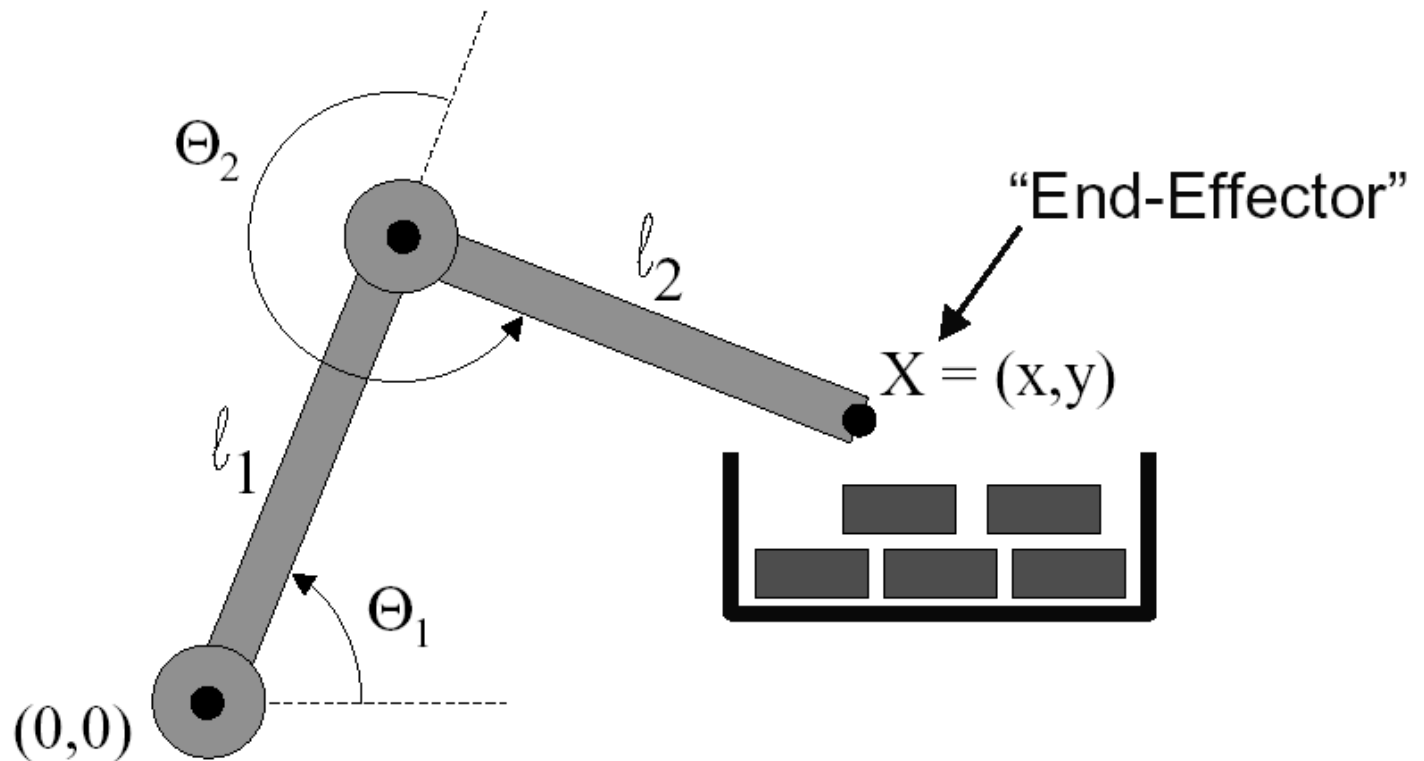
Forward Kinematics

- Joint motions can be specified by spline curves



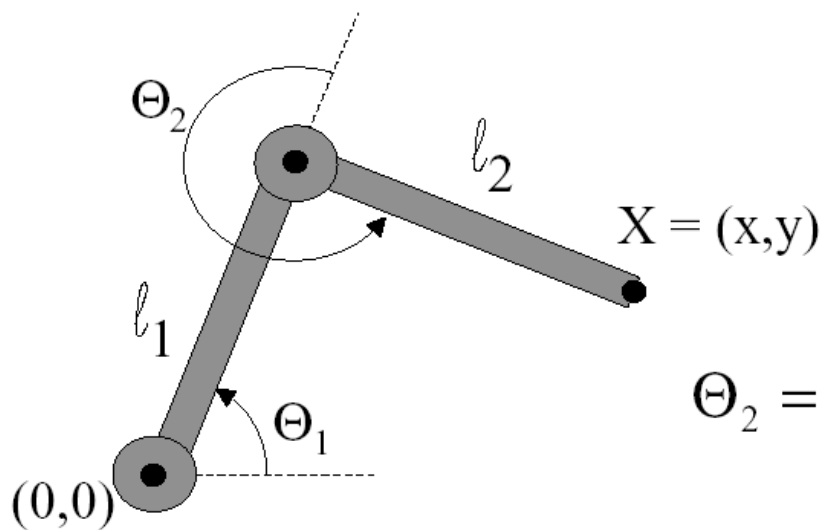
Inverse Kinematics

- What if animator knows position of “end-effector”



Inverse Kinematics

- Animator specifies end-effector positions: X
- Computer finds joint angles: Θ_1 and Θ_2 :



$$\theta_1 = f^{-1}(x, y)$$

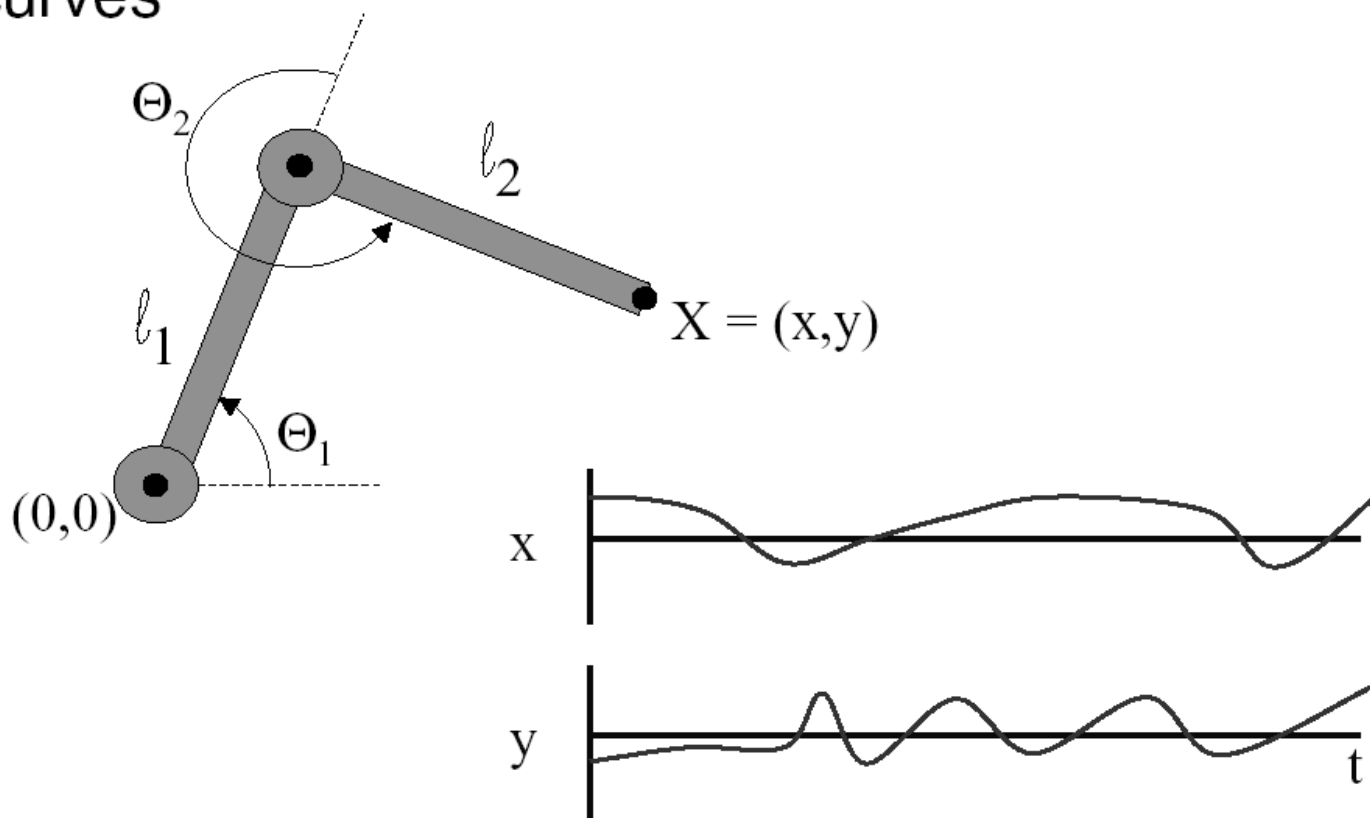
$$\theta_2 = f^{-1}(x, y)$$

$$\Theta_2 = \cos^{-1} \left(\frac{x^2 + y^2 - l_1^2 - l_2^2}{2l_1l_2} \right)$$

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

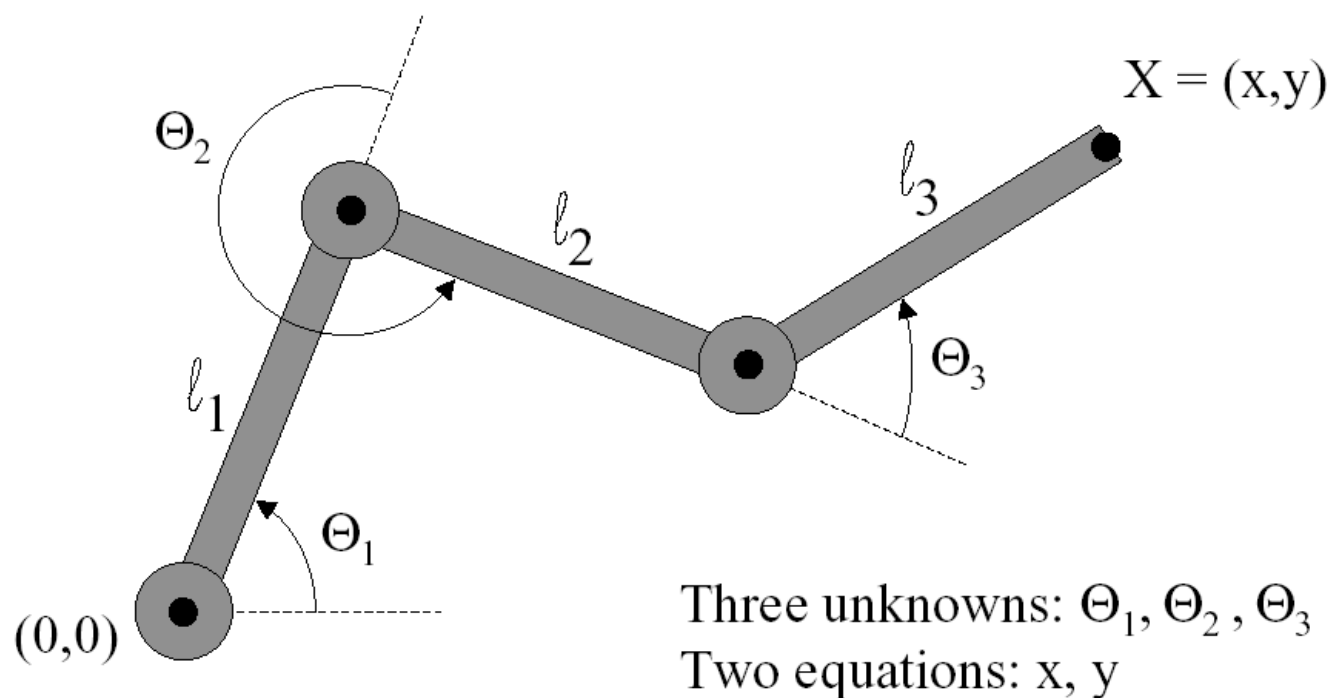
Inverse Kinematics

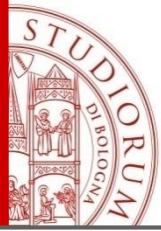
- End-effector positions can be specified by spline curves



Problems with IK

- Problem for more complex structures
 - System of equations is usually under-defined
 - Multiple solutions



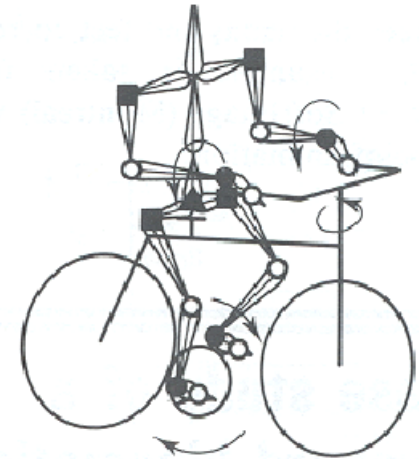


Inverse Kinematics (IK)

- Analytical Solutions
- Iterative Methods
(optimization-based methods)
- Incremental Constructions
(matrix inversion techniques)

Summary

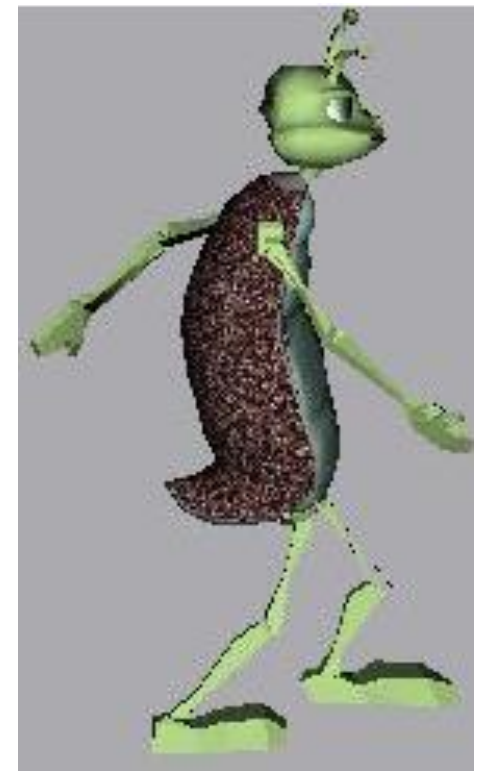
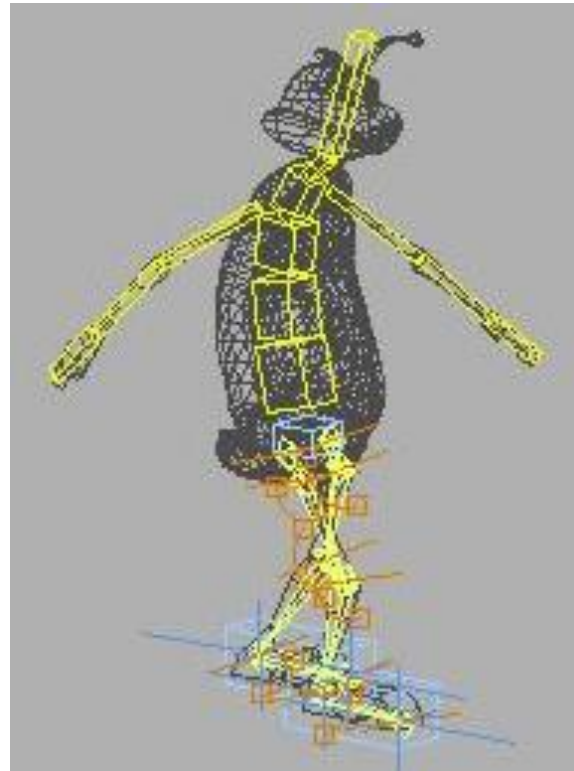
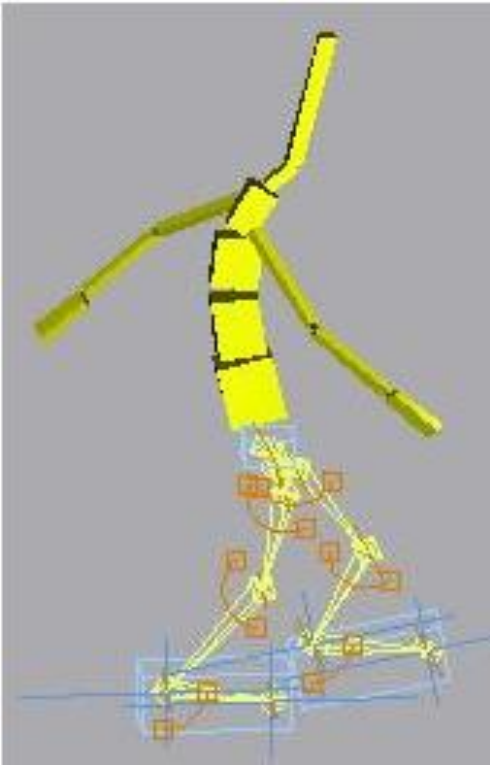
- Forward kinematics
 - Specify conditions (joint angles)
 - Compute positions of end-effectors
- Inverse kinematics
 - “Goal-directed” motion
 - Specify goal positions of end effectors
 - Compute conditions required to achieve goals



Inverse kinematics provides easier specification for many animation tasks, but it is computationally more difficult

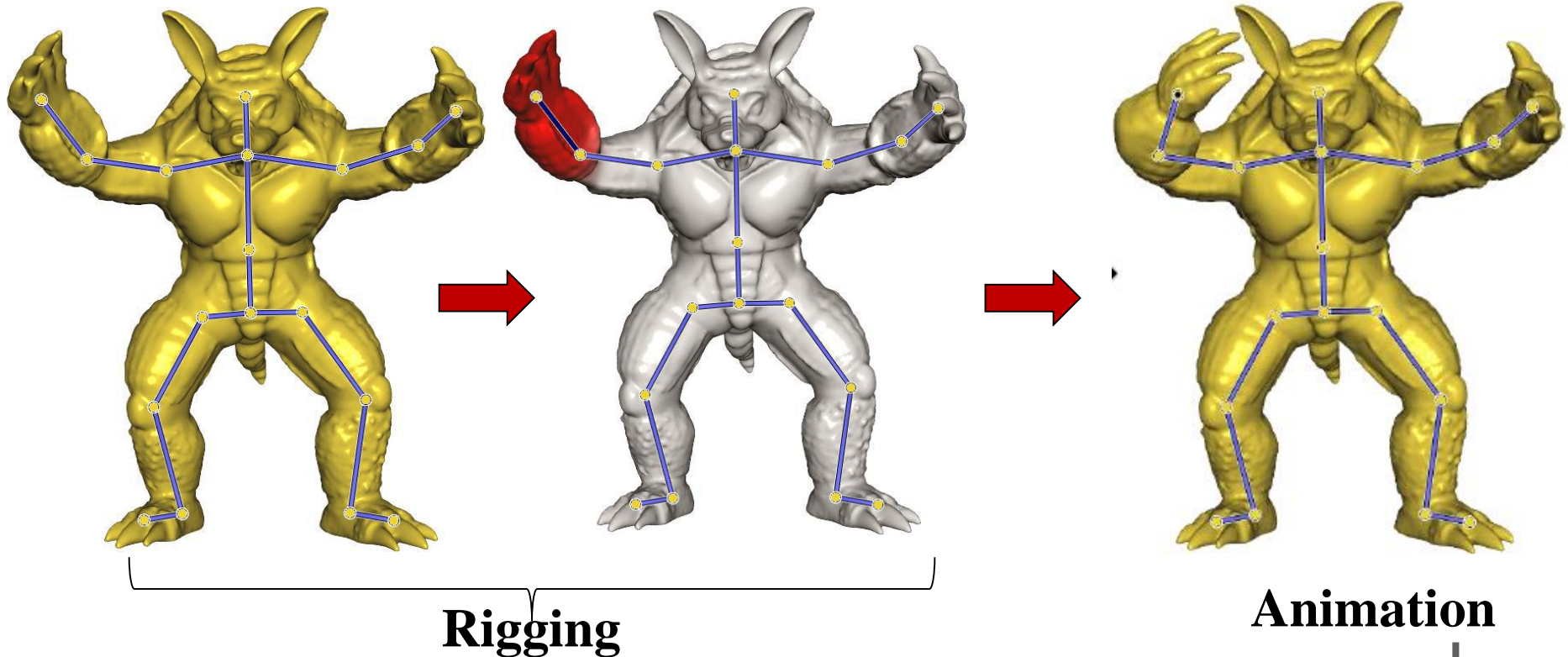
Skeleton and skinned mesh

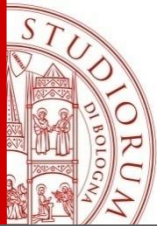
- Embed a skeleton into a detailed character mesh



Skeleton (Rigging) –
authoring of a rig , defining
the skeleton

Skinning
“paint” of (weighted) links
between vertices and bones





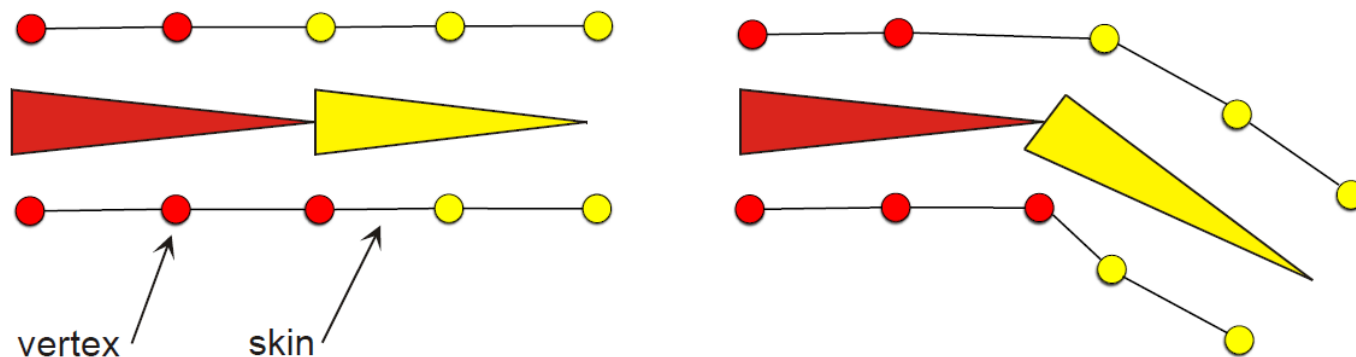
Skinned mesh

- A skinned mesh is a mesh animated by a bone system (Skeleton of an articulated figure)
- The movement of the skeleton imposes a movement of the skinned mesh

PROBLEM1: Bind the skeleton to the mesh

- Attempt 1: assign each vertex to closest bone

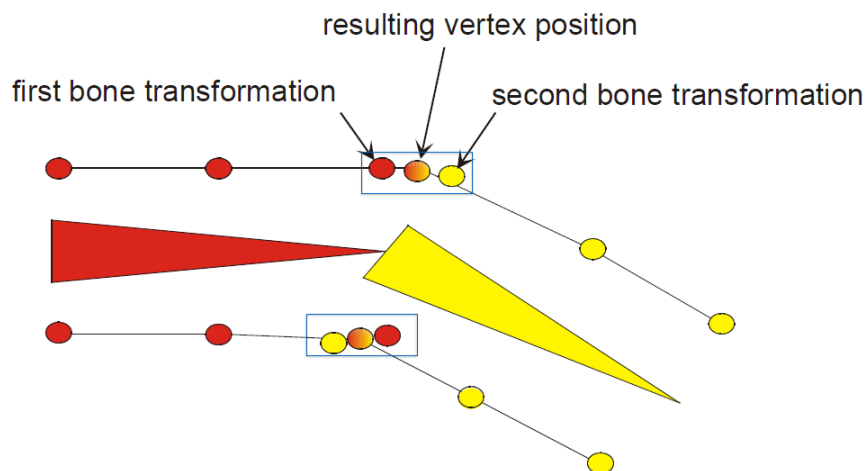
PROBLEM 2: Provide a natural way to deform the mesh vertices along with the bones!



Linear blend Skinning

Attempt 2: assign each vertex to multiple bones, compute world coordinates as *convex combination*

- Weights: influence of each bone on the vertex
- Leads to smoother deformations of the skin



- Moving the skeleton, each joint produces a movement to the vertex, the final vertex position is an average of the computed positions



Thank you!