Computer Graphics: Animation - 1

Table of contents

Animation

- Animation consists in generating repeated renderings of a scene with smoothly changing parameters such as the viewpoint, the objects positions and shapes, quickly enough so that the illusion of motion is achieved;
- Animation is thought of as presenting a sequence of frames, rapidly enough to achieve the sense that the objects in the frames are moving smoothly.

Animation

- In the early days, objects were altered frame by frame to create animation. Examples: Walt Disney's picture movies, King Kong ...
- Systems with computer support for animation began to appear in the 70s.
  - The first computer generated 3D animated short film is Luxo Jr by Pixar in 1987.
  - The first computer generated 3D animated full length film is Toy Story by Pixar (1995).
- Despite completely created by computers, movies like Toy Story and more recent animation movies are still costly in human time.

Traditional animation

This is how animated picture movies were done (e.g. Walt Disney's movies). Animation was obtained by generating each frame of the animation by hand.

- Great control
- Tedious work

Various techniques introduced to further reduce burden on the artists:

- Usage of various layers (e.g. one layer for the background, one layer for a character, another layer for a different character, ...)
- Key frames drawn by a senior animator and in-between frames drawn by some stooge

Traditional animation principles

Main principles observed from the work of artists and animators and summarized in a paper by John Lasseter:

- Squash and stretch
- Staging
- Timing
- Anticipation
- Follow through
- Overlapping action
- Secondary action
- Straight-ahead vs pose-to-pose vs blocking
- Slow-in, slow-out
- Exageration
- Appeal
- Arcs
- Weight

Computer-assisted animation

Fully automatic animation does not exist yet, but several computer techniques have been developed. These help remove much of the tedious work of the animator. Basic approaches are:

- Physical models
- Procedural models
- Keyframing

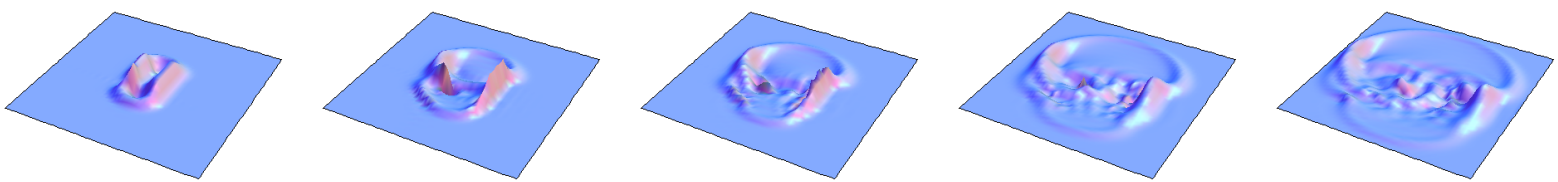
Physical modeling

Suitable for inanimate objects and effects where there is a numerically tractable physical model. Depending on the goal:

- Realism: numerical simulation from physical models. Involve solving some equations (ordinary differential equations, partial differential equations). Potentially expensive.
- Speed: simplification of physical models to get good looking results for a reasonable computational cost.

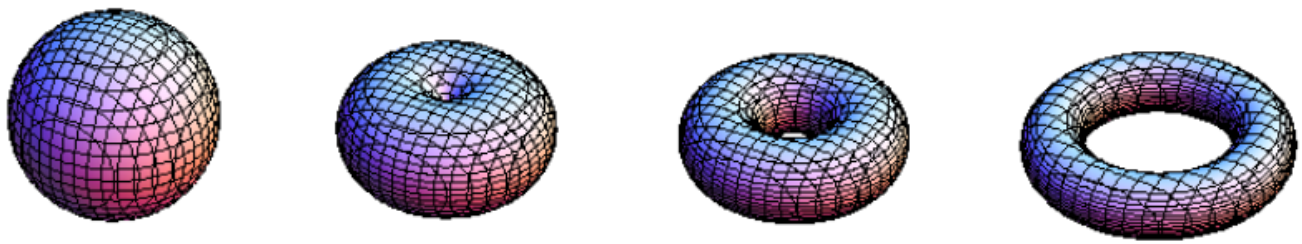
Examples:

- Particle systems for smoke, clouds, water
- Spring mass arrays for cloth simulation
- Partial differential equations for water simulation (shallow water wave equations, wave equation)



Procedural Approach

- The behaviour of an object is described by a procedure or a script
- Appropriate for easily defined behaviours of inanimate objects
- Example: crack propagation in glass or concrete (following some common pattern); a clock with second, minute and hour hands; a bouncing ball; metamorphosis between two shapes



Example: metamorphosis from a sphere to a torus

$$f(x, y, z, t) = (1 - t)f_1(x, y, z) + tf_2(x, y, z), t \in [0, 1]$$

with $f_1 = 0$ corresponds to the sphere, and $f_2 = 0$ to the torus.

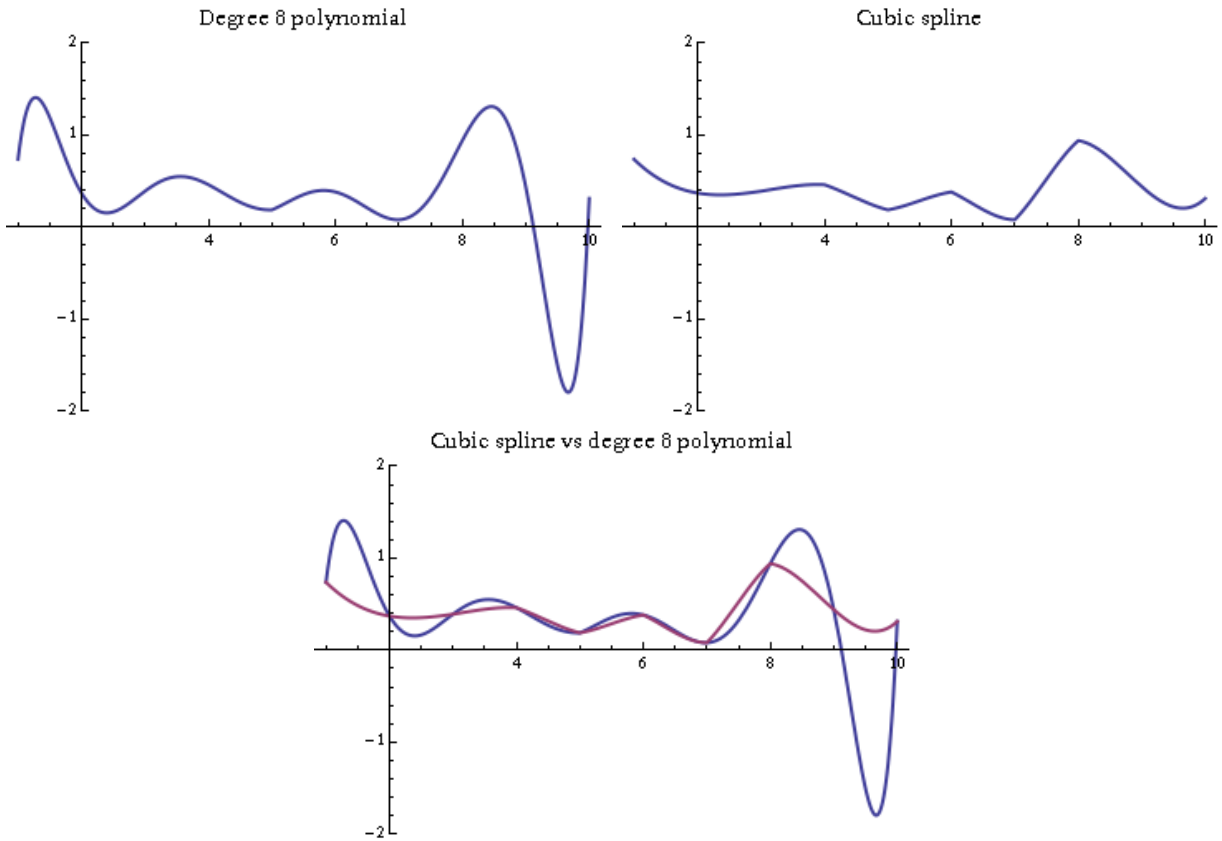
Keyframing

- Traditionally: senior animator drawing key frames at some time intervals; some stooge draws all the in-between frames
- Now: the computer calculates the in-between positions to simulate a smooth movement between the keyframes
- Usage of polynomial and spline curves to define path of motion for some particular points on a moving object

Interpolation

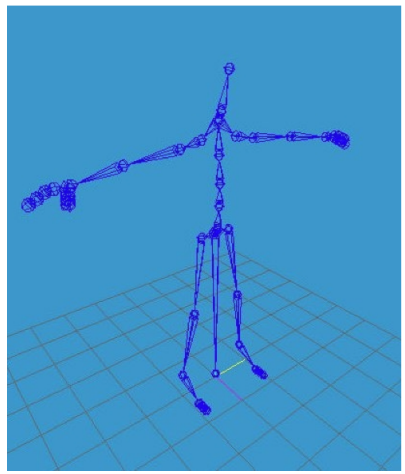
Polynomials are used for interpolation between the key frames.

- Lagrange polynomials,
- Cubic polynomials generally produce the nicest interpolation,
- High degree polynomials are too wigly.



Skeleton

- For animation of characters (humans, robots, animals), jointed skeletons are used.
- Skeletons are made of set of links
- Each link is rigid and movement are constrained by the degree of freedom at each joint
- Skeleton are then fleshed out (addition of skin) to create an animated figure



Skeleton

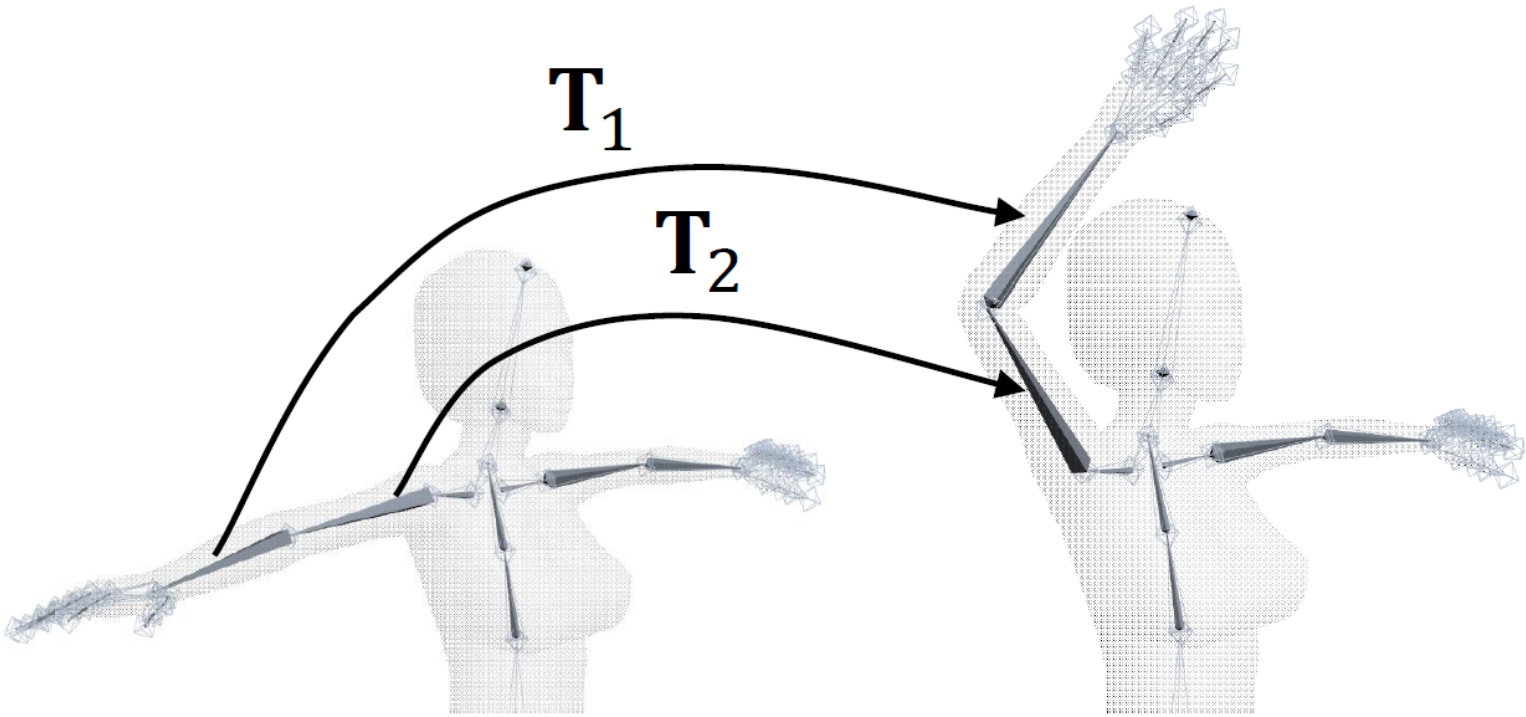


Figure: Skinning: Real-time Shape Deformation, Jacobson et al.

Complexity of motion

- Unfortunately, the motions of articulated creatures are very complex, making it difficult to specify for computer based animation systems
- Systematic approaches are needed for scripting skeleton motion:
  - Motion capture
  - Kinematics

Motion capture

- One approach to define motion is to use motion capture
- An actor wears a special suit and is marked with several identifiable spots that can be tracked by cameras
- Motion is registered in a computer and can be mapped to corresponding points on computer models
- Example of usage:
  - Motion capture was used in the Lord of the Rings films
  - In the remake of the movie King-Kong (for the gorilla)
  - Motion capture is used for animation of video games: soccer, basketball, ice-hockey, american football
- Motion capture, registration, mapping, usage is however laborious and difficult to automate

Motion capture

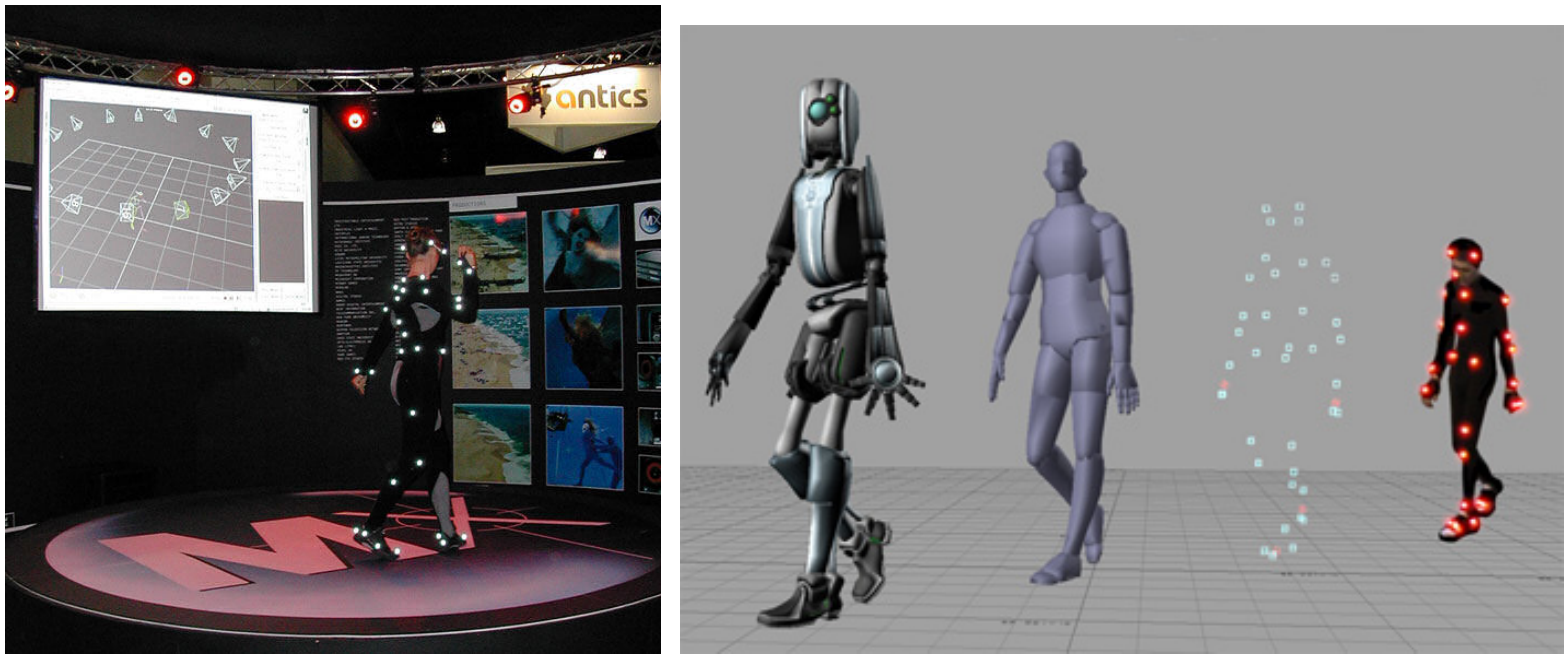
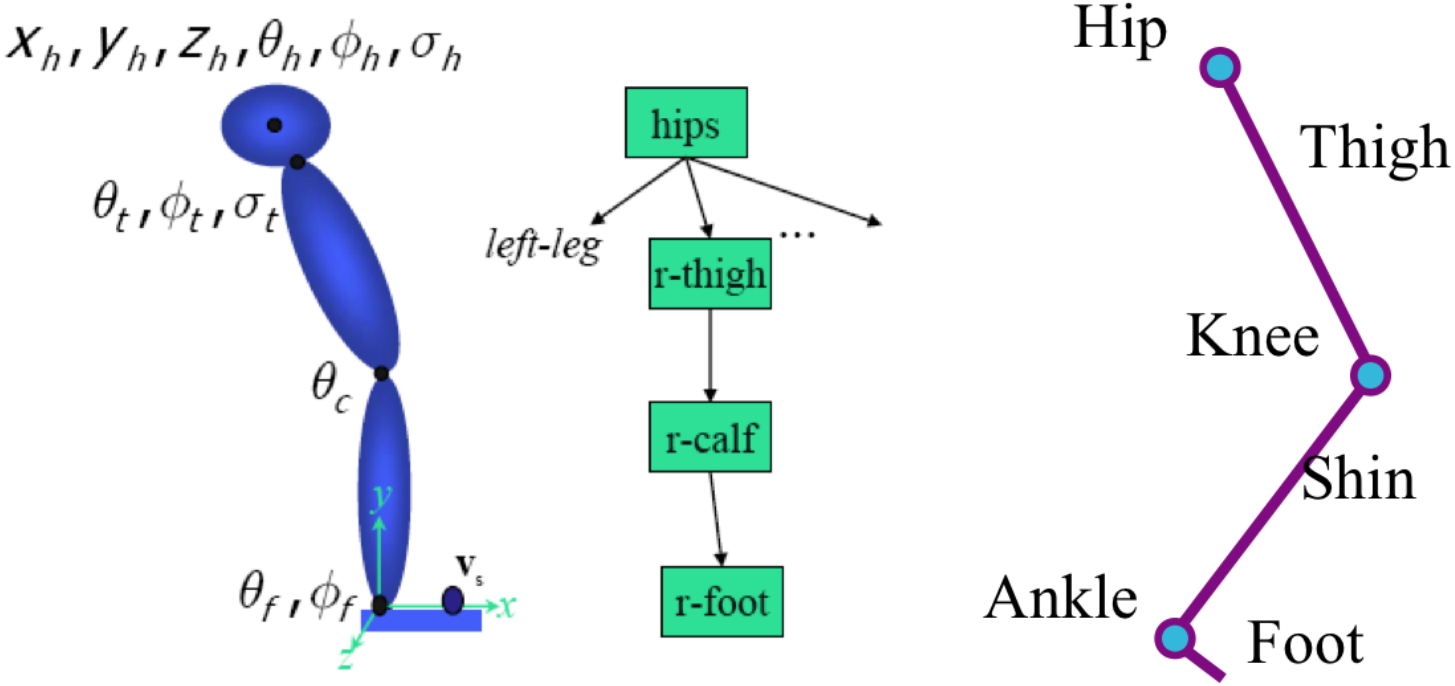


Figure: Image credit: Wikipedia

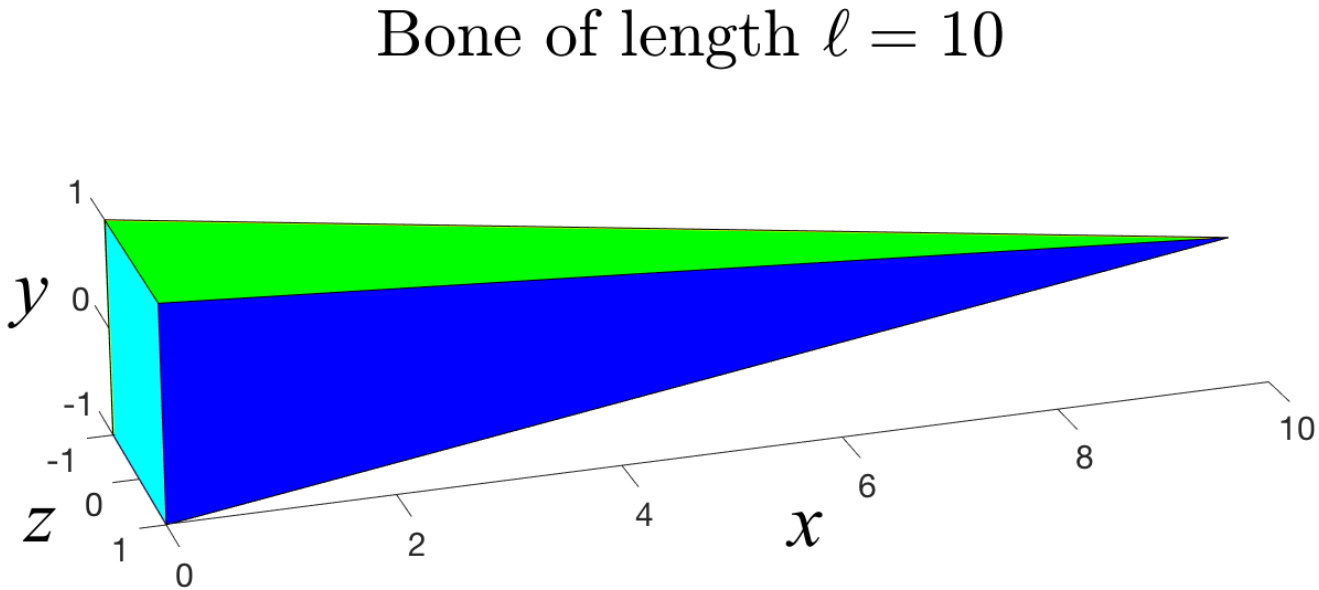
Kinematics

- A good approximation for articulated creatures is to treat them as rigid jointed skeletons (i.e. a collection of "bones")
- This allows to constrain their movement
- Study of the motion of articulated rigid chains started in mechanical engineering and in robotics



Canonical bone

Canonical bone (given its length L) is represented with its head at the origin $(0, 0, 0)$, and its end at $(L, 0, 0)$.
Often represented by a cone or a pyramid where the base stands for the head.



Twisting and bending

Transformations on the canonical bone correspond to rotations about the main axes:

- Twisting: around the x axis
- Bending: around the z axis

Local transformation of the canonical bone is a composition of these transformations in the following order: twisting - bending - twisting.
The corresponding angles are called: Euler angles. (Note: sometimes different conventions are used)

Euler angles

The Euler rotation matrix is then given by:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta_3 & -\sin\theta_3 & 0 \\ 0 & \sin\theta_3 & \cos\theta_3 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos\theta_2 & -\sin\theta_2 & 0 & 0 \\ \sin\theta_2 & \cos\theta_2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\cdots \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta_1 & -\sin\theta_1 & 0 \\ 0 & \sin\theta_1 & \cos\theta_1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Skeleton assembly (Rest position)

- To assemble the skeleton, one should map each bone from its canonical pose, to its position and orientation in the model at rest.
- Bones are rigid, thus for each bone we are given a transform $\hat{\mathbf{T}}$ that positions the head ($\mathbf{h}$) and end ($\mathbf{e}$) of the bone to their desired positions in the model:

$$\hat{\mathbf{h}} = \hat{\mathbf{T}}(0, 0, 0, 1)^t,$$

$$\hat{\mathbf{e}} = \hat{\mathbf{T}}(L, 0, 0, 1)^t.$$

- Furthermore, if the bone has a parent, the end of the parent and the head of the bone should be connected:

$$\hat{\mathbf{e}}_p = \hat{\mathbf{h}}$$

Pose bone

Moving a bone is done by specifying a mapping to a new position and orientation from its rest state.
Since bones are rigid, it means specifying a transformation $\mathbf{T}$ (made of a rotation and a translation) from rest position to pose position:

$$\mathbf{x} = \mathbf{T}\hat{\mathbf{x}}$$

$\hat{\mathbf{x}}$ corresponds to a point in rest position, $\mathbf{x}$ is a point in pose position.

Forward kinematics

- Given a specification of all the parameters, i.e. translation amounts and the Euler angles for each joint of an articulated creature, we can calculate the position and orientation of its end points
- This is done by combining the transformations by matrix multiplications to be applied on the vector characterizing the position of an end point

Forward kinematics

To compute the transformation $\mathbf{T}_i$ of bone i in a skeleton, aggregate the relative rotations $\mathbf{R}_i$ between the bone i and its parent p_i

$$\mathbf{T}_i = \mathbf{T}_{p_i} \hat{\mathbf{T}}_i \mathbf{R}_i \hat{\mathbf{T}}_i^{-1},$$

or

$$\mathbf{T}_i = \mathbf{T}_{p_i} \hat{\mathbf{T}}_i \mathbf{R}_x(\theta_3) \mathbf{R}_z(\theta_2) \mathbf{R}_x(\theta_1) \hat{\mathbf{T}}_i^{-1},$$

where $\hat{\mathbf{T}}_i$ is the canonical transformation of bone i, $\mathbf{R}_x, \mathbf{R}_z$ are the Euler rotation matrices and $\mathbf{T}_{p_i}$ is the transformation of the parent p_i.
This is a recursive definition that is usually implemented via a (matrix) stack.

Transformation hierarchy and matrix stack

- Each joint acts as a parent for the hierarchy below it.
- Given the local transformation at each joint, concatenation of the transformations (matrix multiplications) from the root to the joint gives the position of any joint in the world coordinate system.
- Evaluating the whole skeleton = depth-first traversal of the tree of joints. A transformation stack is used for this.
- While traversing down, the current matrix is pushed on the stack and a new one is created by multiplying the current matrix with the one stored at the joint.
- When backtracking to the parent, the transformation is undone by popping the matrix.

Interpolation of rotation (keyframe animation)

To create a long animation, parameters values (Euler angles) of the keyframes are specified. In-between values are then obtained by interpolation.
Problems:

- Interpolating Euler angles smoothly is difficult.
- Applying several rotations is difficult and unprecise.
- Gimbal lock. If two axes coincide, then we lose one degree of freedom.

Alternative: use and interpolate quaternions.

Quaternions

Quaternion space: four dimensional vector space with basis $\{1, i, j, k\}$ with the relationship

$$i^2 = j^2 = k^2 = ijk = -1.$$

Quaternions can be seen as extension of complex numbers and by analogy we write

$$q = a + b\,i + c\,j + d\,k.$$

a is the real part of the quaternion, and $b\,i + c\,j + d\,k$ the imaginary part.
Imaginary quaternions can be naturally used to represent vectors in $\mathbb{R}^3$

$$(x, y, z) \to x\,i + y\,j + z\,k.$$

Quaternions

Let $q = (s, \mathbf{v})$ be a quaternion with s the real part and $\mathbf{v}$ the imaginary part.
Addition: $q_1 + q_2 = (s_1 + s_2, \mathbf{v}_1 + \mathbf{v}_2)$.
Multiplication: $q_1 q_2 = (s_1 s_2 - \mathbf{v}_1 \mathbf{v}_2, s_1 \mathbf{v}_2 + s_2 \mathbf{v}_1 + \mathbf{v}_1 \times \mathbf{v}_2)$.
Conjugation: $\bar{q} = (s, -\mathbf{v})$.
Inversion: $q^{-1} = \bar{q}/(\bar{q}q)$.
Rotation about unit axis $\mathbf{n}$ by angle θ can be computed using the unit quaternion $q = (s, \mathbf{v}) = (\cos(\theta/2), \mathbf{n}\sin(\theta/2))$.
The point $p = (0, \mathbf{p})$ after rotation is then transformed into $p' = qpq^{-1} = qp\bar{q}$.

Spherical linear interpolation and arcball

Given two orientations A and B that we want to interpolate between. Let q_A and q_B the unit quaternions corresponding to these orientations,
$q = q_B q_A^{-1}$ is the quaternion corresponding to the rotation from A to B.
For $0 \leq t \leq 1$, $q^t q_A$ provides an homotopy between these orientations.
This is called a spherical linear interpolation (SLERP). This corresponds to a linear interpolation along a geodesic on the sphere SO(3).
A direct application of this is implementing an arcball.

Arcball

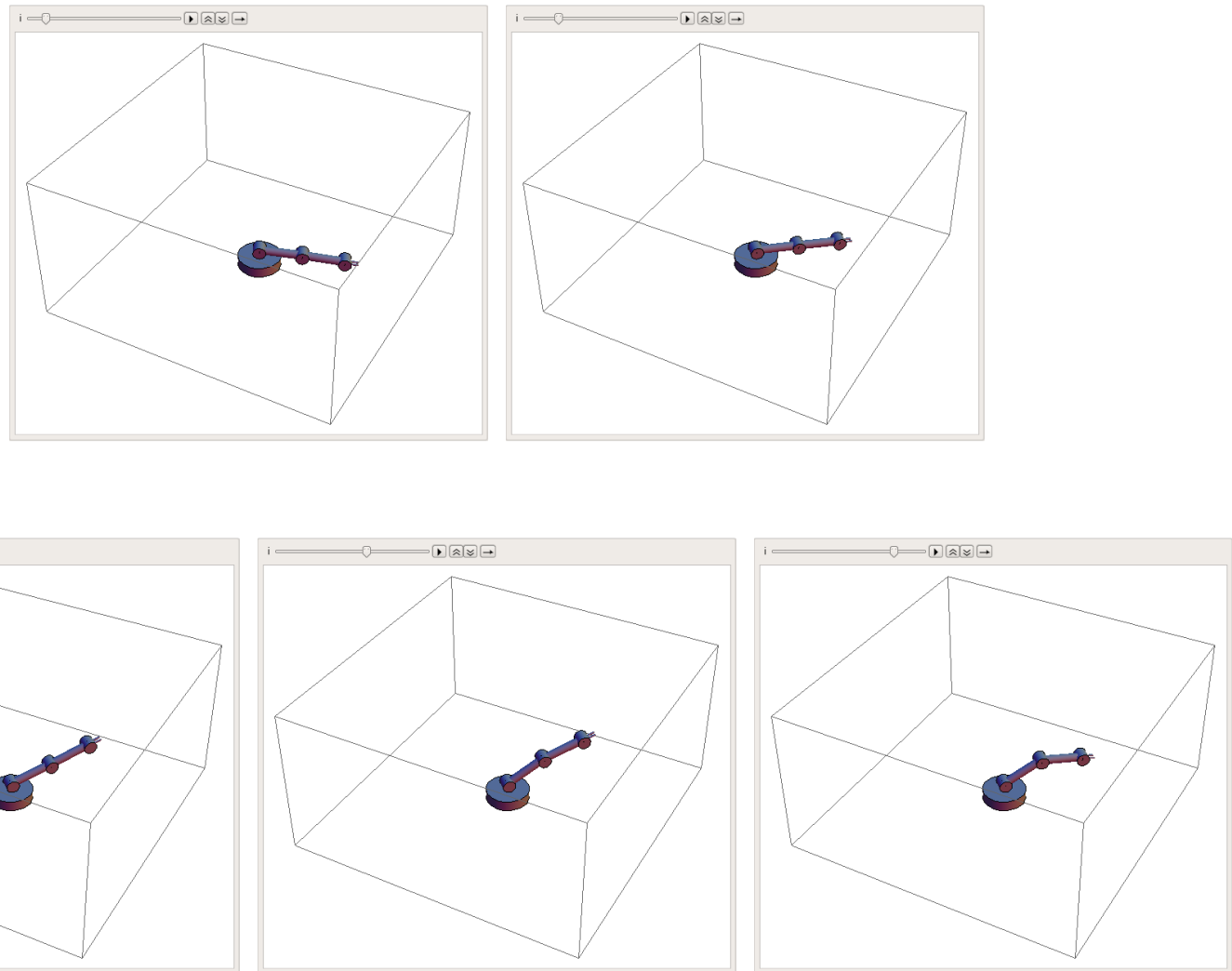
- Clicks are projected on a sphere around the object to be rotated,
- Dragging the mouse corresponds to an arc on that sphere,
- The object is rotated along that arc (by twice the angle of that arc),
- The composition of the rotations is done via quaternion multiplications.

Example of forward kinematics

- A robot arm is controlled by four joints, each parameterized by one angle
- We specify some intermediate values for these parameters
- Other values at each time-step are obtained by interpolation with cubic splines
- The robot arm position at each time-step is obtained by multiplying the transformation matrix of each joint by its position

Example of forward kinematics

Positions at various time-steps are illustrated below:



Inverse kinematics

- Forward kinematics: end points position determined by setting the value of all parameters (of the relative transformations). This is inconvenient.
- In practice, we usually want to do the opposite: i.e. specify some path for an end point of a chain and obtain the corresponding parameters values for the other links
- This is called inverse kinematics and is more difficult than forward kinematics:
  - This is an ill posed problem (there are many possible solutions for some chains)
  - It usually requires to minimize a constrained non-linear objective function
- In practice, we can specify a path (curve) for the end positions of a chain or we can specify end positions for different key frames and interpolate to get in-between positions. Finally, we have to solve a constrained non-linear problem to obtain the corresponding parameter values at each time step

Inverse kinematics

Let $\mathbf{x}$ be the position of the k end points and $\mathbf{p}$ the vector of parameters for specifying all internal joints along the chain (Euler angles).
The end point is obtained as: $\mathbf{x} = \mathbf{x}(\mathbf{p})$, with the concatenation of the transformations.
The goal is to minimize the function (or energy):

$$E(\mathbf{p}) = \sum_{i=1}^{k} \|\mathbf{x}_i(\mathbf{p}) - \mathbf{x}_{t,i}\|^2.$$

Furthermore, we usually have constraints on the parameters:

$$\min E(\mathbf{p}) \text{ such that } \mathbf{p}^{\min} \leq \mathbf{p} \leq \mathbf{p}^{\max}.$$

This is a constrained non-linear least squares problem.

Gradient descent

The minimization can be done by gradient descent

$$\mathbf{p} \leftarrow \mathbf{p} - \alpha \left(\frac{dE(\mathbf{x}(\mathbf{p}))}{d\mathbf{p}} \right)^t,$$

$$\mathbf{p} \leftarrow \mathbf{p} - \alpha \left(\frac{d\mathbf{x}(\mathbf{p})}{d\mathbf{p}} \right)^t \left(\frac{dE(\mathbf{x})}{d\mathbf{x}} \right)$$

Introducing the Jacobian of the transformation

$$\mathbf{J} = \frac{d\mathbf{x}}{d\mathbf{p}}.$$

We arrive at

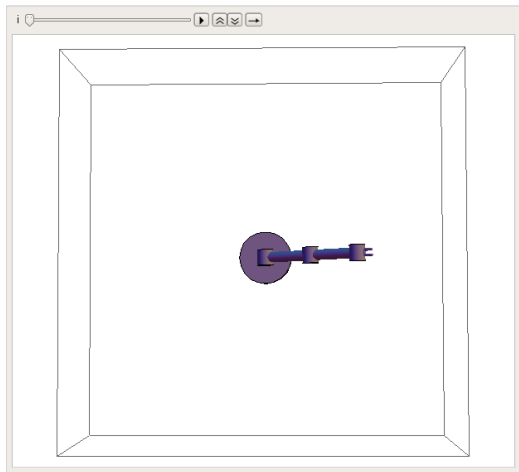
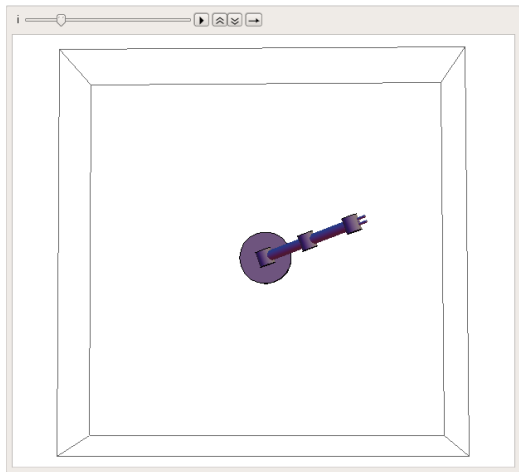
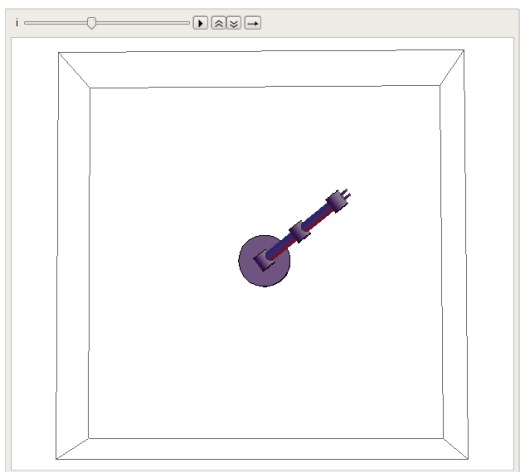
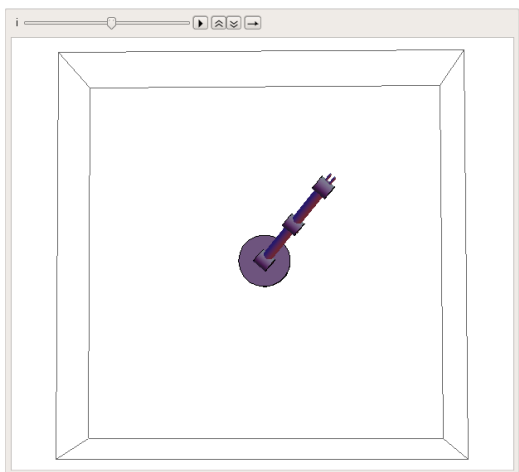
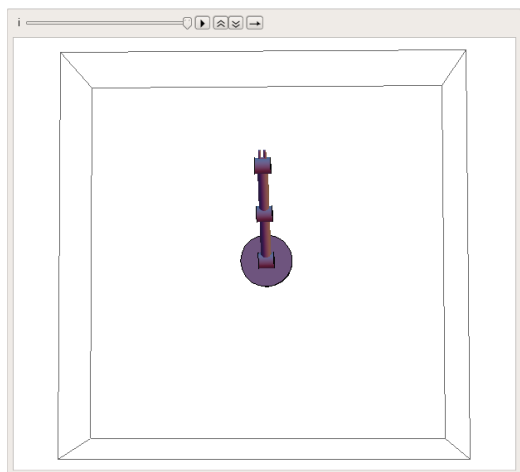
$$\mathbf{p} \leftarrow \mathbf{p} - \alpha \mathbf{J}^t \left(\frac{dE(\mathbf{x})}{d\mathbf{x}} \right).$$

Example of inverse kinematics

- A robot arm is controlled by four joints, each parameterized by one angle
- End point of the robot arm is moving along a circular arc
- At each time-step we know the position of the end point, we need to calculate the relative positions of all the joints
- This can be done by minimizing the distance between the end point and its target for the four parameters (function NMinimize of mathematica is used)
- Computed parameter values are then used to animate the robot arm (with forward kinematics)

Example of inverse kinematics

Some intermediate positions of the robot arm along a circular arc are illustrated below:

Skinning

The skeleton alone does not give a realistic appearance of the character to be animated.
Let us add some shape to the skeleton, such that the shape will be deformed with the animation of the skeleton.
More generally, skinning means controlling the deformations of an object using a set of deformation primitives and is not necessarily limited to skeletal animation.

Linear Blend Skinning

Assume:

- A rest pose shape for the character to be animated. Typically represented by a triangle mesh with vertices $\hat{\mathbf{v}}_i$ and fixed connectivity.
- Only the skeleton is animated via bone transformations: $T_1, ..., T_j, ..., T_m$ (the transformation matrices aligning the rest pose of bone j with the current animated pose).
- For each vertex $\hat{\mathbf{v}}_i$, we have a set of weights $w_{i,1}, ..., w_{i,m}$ expressing the influence of each of the bone j, $1 \leq j \leq m$ on the vertex $\hat{\mathbf{v}}_i$.

Upon deformation by a set of bone transformations, each vertex of the mesh is deformed as:

$$\mathbf{v}_i = \sum_{j=1}^{m} w_{i,j} T_j \hat{\mathbf{v}}_i$$

Skinning weights

Usual requirements: $w_{i,j} \geq 0$ (positivity) and $\sum_{j=1}^{m} w_{i,j} = 1$ (partition of unity). There are several techniques for automatically desigining skinning weights.

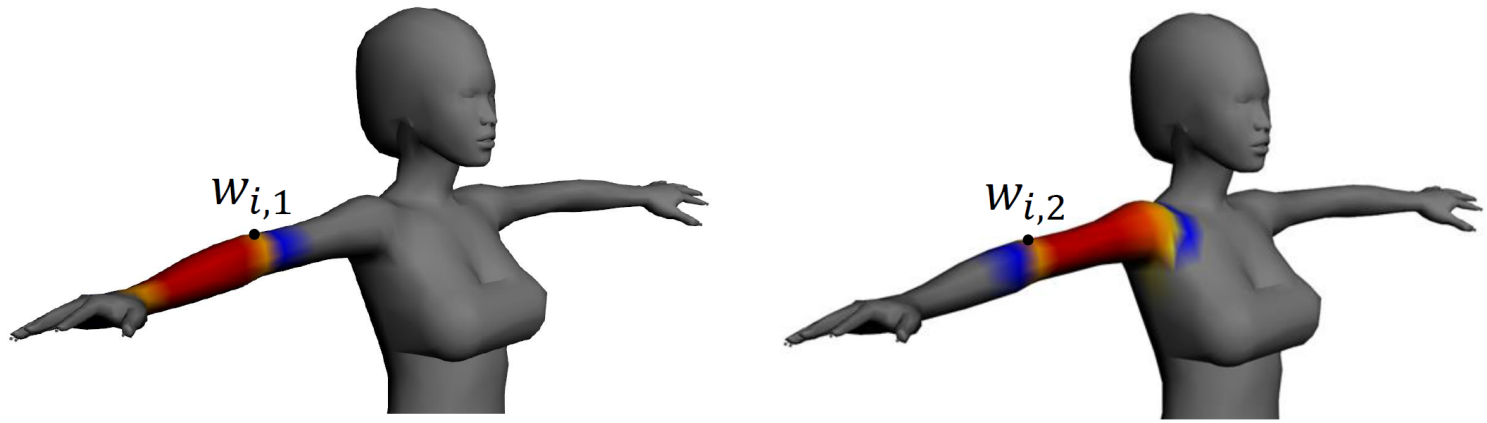


Figure: Skinning: Real-time Shape Deformation, Jacobson et al.

Animation control in OpenGL

In OpenGL animation can be controlled with a callback for the idle event. The callback function is used to update the model parameters, the scene and the display.
First, register the callback (e.g. HandleIdleEvent) with the function glutIdleFunc:

```
int main(int argc, char** argv) {
  // usual glut and gl initialization
  glutIdleFunc(HandleIdleEvent);
  // set other callbacks, call main loop
  return 0;
}
```

Animation control in OpenGL

The callback function needs then to be defined. Usually this function will update the model parameters (e.g. joint angles), the scene and the display:

```
static void HandleIdleEvents(void) {
  // Update model or display parameters
  // as a function of time

  // Force redisplay
  glutPostRedisplay();
}
```

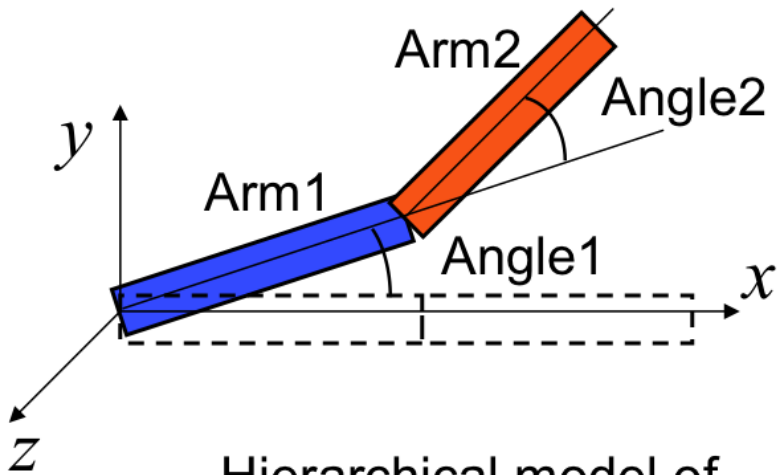
Animation control in OpenGL

We need to have control over the speed at which the parameters are updated. One solution is to use an empty while loop for some time duration corresponding to the time-step of the animation. The global variable g_previous_time saves the last time when the parameters were updated. And the global variable g_time_step_duration holds the time-step of the animation in milliseconds.

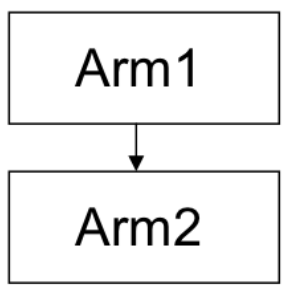
```
#include <time.h>
static void HandleIdleEvents(void) {
  while ((clock() - g_previous_time)/CLOCKS_PER_SEC
    < g_time_step_duration/1000) ;
  // update model or display parameters

  g_previous_time = clock();
  // Force redisplay
  glutPostRedisplay();
}
```

Forward kinematics example: animation of a two-part robot arm

Task: create an animation of an articulated model made of two parts. Forward kinematics is used, i.e. parameters values for the joints will be updated at each time step.



Hierarchical model of the robot arm:



Main function

```
int main(int argc, char** argv) {
  glutInit(&argc,argv);
  glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB);
  glutInitWindowSize(600,600);
  glutCreateWindow("Robot arm");

  InitGL();

  glutDisplayFunc(Display);
  glutReshapeFunc(Reshape);
  glutKeyboardFunc(HandleKeyboardEvents);
  glutIdleFunc(HandleIdleEvent);

  glutMainLoop();
  return 0;
}
```

Idle and keyboard events handling

```
static void HandleKeyboardEvents(unsigned char key,
    int x, int y) {
 if (key==27) exit(0);
}

static void HandleIdleEvent(void) {
  while ((clock() - g_previous_time)/CLOCKS_PER_SECS
    < g_time_step_duration/1000) ;

  // update model or display parameters
  g_angle_1 += 5;
  g_angle_2 += 5;

  g_previous_time = clock();
  // Force redisplay
  glutPostRedisplay();
}
```

Display function

```
static void Display(void) {
  glClear(GL_COLOR_BUFFER_BIT);
  glLoadIdentity();
  gluLookAt(3.0,4.0,6.0,
    0.5,0.0,0.0,
    0.0,1.0,0.0);

  glPushMatrix();
  // Rotation for arm 1 and 2
  glRotatef(g_angle_1,0.0,0.0,1.0);
  glTranslatef(1.0,0.0,0.0);

  glPushMatrix();
  // Geometry and local transform for arm 1
  glScalef(2.0,0.4,1.0);
  glutWireCube(1.0);
  // ...
}
```

Display function

```
static void Display(void) {
  // ...
  glPopMatrix();

  // Rotation for arm 2
  glTranslatef(1.0,0.0,0.0);
  glRotatef(g_angle_2,0.0,0.0,1.0);
  glTranslatef(0.5,0.0,0.0);

  glPushMatrix();
  // Geometry for arm 2
  glScalef(1.0,0.4,0.5);
  glutWireCube(1.0);
  glPopMatrix();

  glPopMatrix();
  glutSwapBuffers();
}
```

glPushMatrix/glPopMatrix

- glPushMatrix() pushes a copy of the matrix in the current mode (e.g. GL_MODELVIEW) to the corresponding stack.
- glPopMatrix() pops the matrix at the top of the current stack.
- These two functions allow to perform the depth first traversal of the tree of joints, when the mode is set to GL_MODELVIEW.
- The current mode is set with the function glMatrixMode(). Possible modes are GL_COLOR, GL_PROJECTION, GL_TEXTURE, and GL_MODELVIEW.