Visual Computing - Lecture notes week 12

- Author: Ruben Schenk
- Date: 14.12.2021
- Contact: ruben.schenk@inf.ethz.ch

8. Computer Animation

8.1 Describing Motion

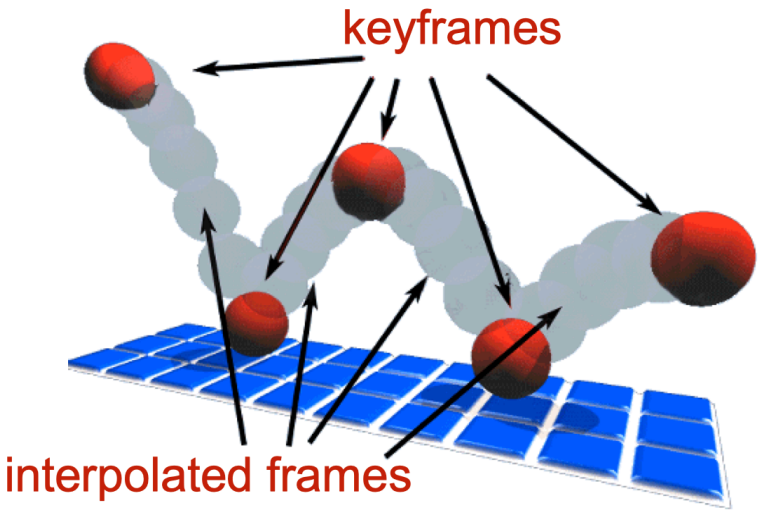
8.1.1 Introduction

How we describe motion on a computer is probably the most important question to answer when talking about motion in general. From a computer graphics perspective, there are three main techniques to that:

- Artist directed, such as keyframing
- Data-driven, such as motion capturing
- Procedural, such as simulations

8.1.2 Keyframing

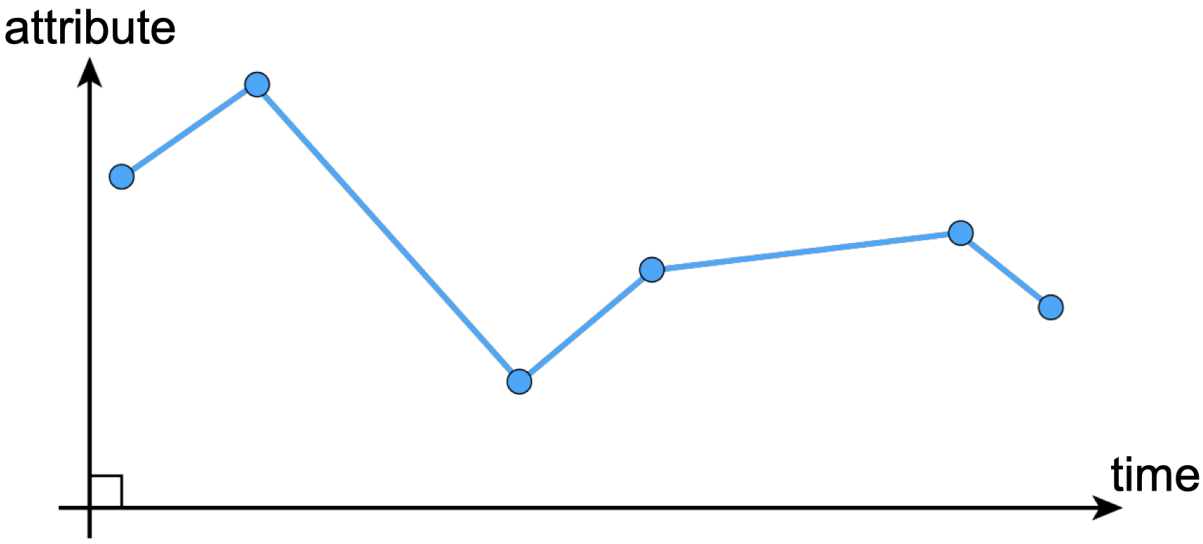
Keyframing is an important yet quite "simple" idea of describing motion. The basic idea is to specify important events in our motion only, and let the computer fill in the rest via interpolation or approximation.



8.2 Spline Interpolation

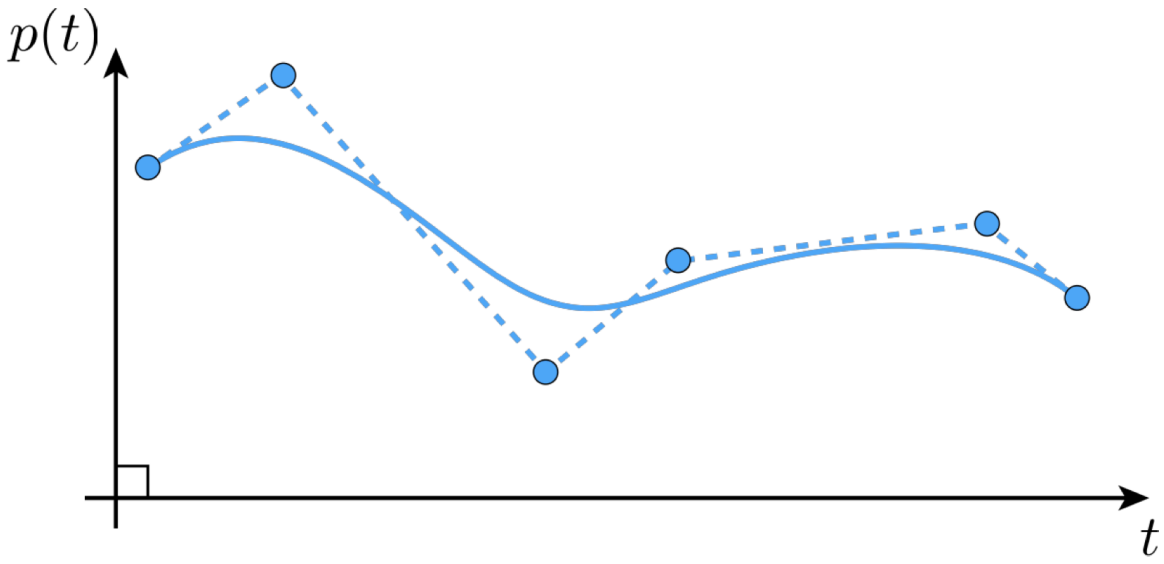
8.2.1 Interpolation

The basic idea behind **interpolation** data is to connect the dots, i.e. the given sample points. One such technique is *piecewise linear interpolation:*



It might be simple, but it yields a rather rough motion.

Another common interpolation technique is *piecewise polynomial interpolation:*



8.2.2 Splines

In general, a **spline** is any piecewise polynomial function. In 1D, splines interpolate data over the real line:

$$(t_i,\ f_i), \quad i = 0, \ldots, n$$

"*Interpolates*" in that case just means that the function exactly passes through those values, i.e.:

$$f(t_i) = f_i \quad \forall i$$

The only other condition is that the function is a *polynomial* when restricted to any interval between the knots:

$$\text{for } t_i \leq t \leq t_{i+1},\ f(t) = \sum_{j=1}^{d} c_i t^j =: p_i(t)$$

The splines most commonly used for interpolation are *cubic,* i.e. $d = 3$.

8.2.3 Fitting Cubic Polynomials To Endpoints

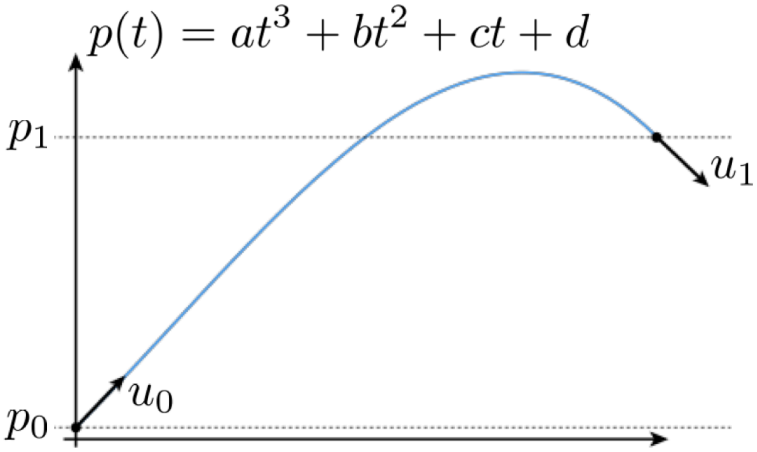
If we want to connect two end points with a cubic polynomial, there are many solutions! Cubic polynomials have four *degrees of freedom,* namely the four coefficients $(a,\ b,\ c,\ d)$ that we can manipulate and control.

However, we only need two degrees of freedom to specify the endpoints!

$$p(t) = at^3 + bt^2 + ct + d$$
$$p(0) = p_0 \Rightarrow d = p_0 p(1) = p_1 \Rightarrow a + b + c + d = p_1$$

This leaves us with four unknowns but only two equations, which is obviously not enough to determine the curve.

However, what if we also want to match the *derivatives* at the endpoints?

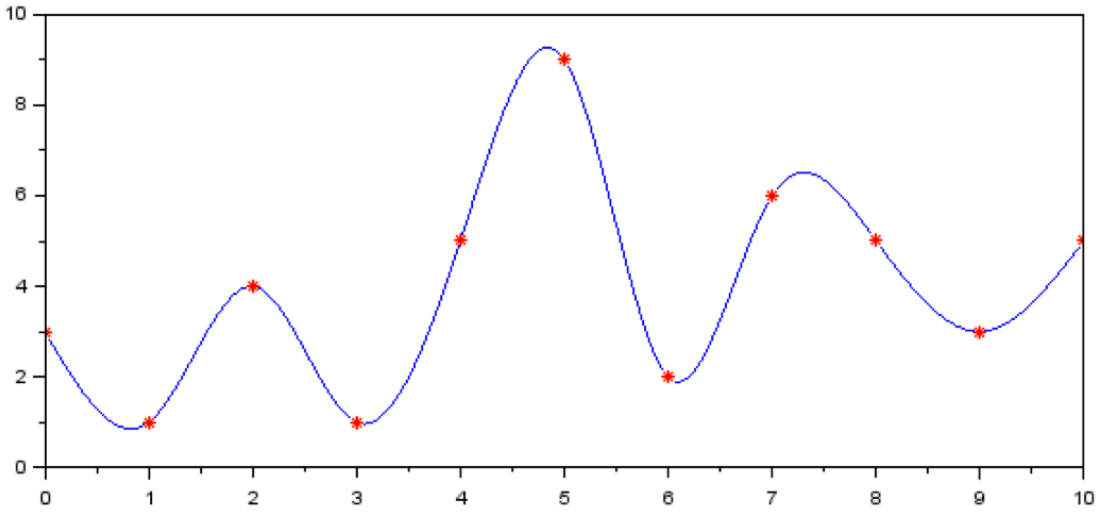


$$\begin{aligned} p(0) &= p_0 && \Rightarrow d = p_0 \\ p(1) &= p_1 && \Rightarrow a + b + c + d = p_1 \\ p'(0) &= u_0 && \Rightarrow c = u_0 \\ p'(1) &= u_1 && \Rightarrow 3a + 2b + c = u_1 \end{aligned}$$

Then we end up with 4 equations and 4 unknowns, which lets us uniquely identify the cubic polynomial we are looking for.

8.2.4 Natural Splines

Natural splines are piecewise splines made out f cubic polynomials p_i.



We want three conditions to hold for natural splines:

1. Interpolation at both endpoints:

$$p_i(t_i) = f_i,\ p_i(t_{i+1}) = f_{i+1}, \quad i = 0, \dots, n-1$$

2. Tangents agree at the endpoints, i.e. C^1 continuity:

$$p_i'(t_{i+1}) = p_{i+1}'(t_{i+1}), \quad i = 0, \dots, n-2$$

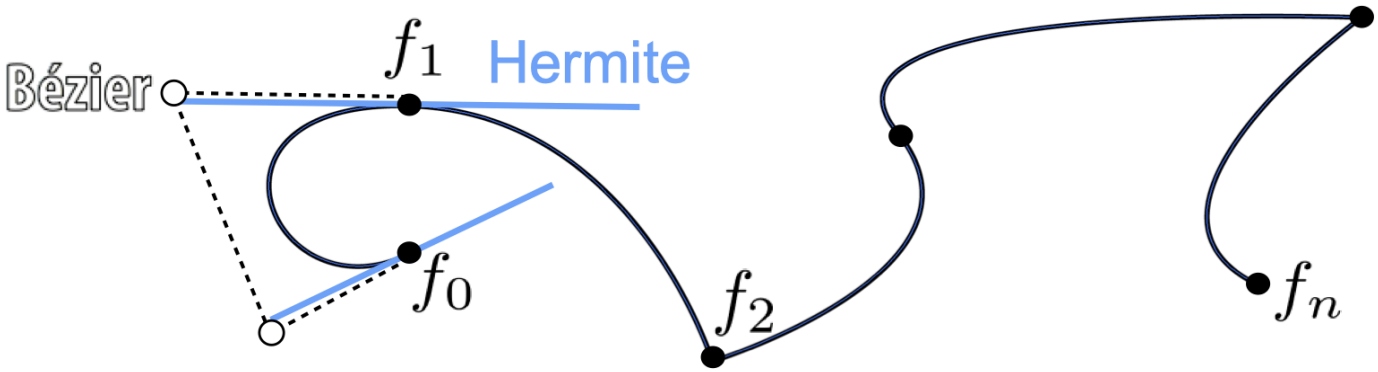
3. Curvatures agree at the endpoints, i.e. C^2 continuity:

$$p_i''(t_{i+1}) P p_{i+1}''(t_{i+1}), \quad i = 0, \dots, n-2$$

If we look at the picture above, we see that for $n+1$ points we have $4n$ DOFs, which leads us to $2n + (n-1) + (n-1) = 4n - 2$ equations. WE can pin down the remaining DOFs by setting the curvature to zero at the endpoints. This is what makes the curvature "*natural*".

8.2.5 Hermite/Bézier Splines

Hermite/Bézier splines are based on the idea that each cubic piece is specified by the endpoints and tangents, in contrast to natural splines where we define an additional point on which we have to exactly meet:



Hermite splines have the following properties:

1. Endpoints interpolate data:

$$p_i(t_i) = f_i,\ p_i(t_{i+1}) = f_{i+1}, \quad i = 0, \dots, n-1$$

2. Tangents interpolate some given data:

$$p_i'(t_i) = u_i,\ p_i'(t_{i+1}) = u_{i+1}, \quad i = 0, \ldots, n-1$$

8.3 Rigging

8.3.1 Introduction

Animation rigs are user-defined mappings between a few parameters and the deformations of a high-res mesh. Animations are simply time trajectories specified for rig parameters.

8.3.2 Blend Shape Rigs

Blend Shape rigs are simple rigs based on a set of meshes (the input). The output is a blended mesh obtained through interpolation. The splines (or keyframes) specify the blending weights over time.

Mathematically:

- Input: set of meshes M_i with vertices x_i^j and blending weights $\alpha = (\alpha_1, \ldots, \alpha_n)$
- Output: blended mesh M through linear combination: $M = \sum_i \alpha_i M_i$, i.e. $x^j = \sum_i \alpha_i x_i^j$

8.3.3 Cage-Base Deformers

The idea behind **cage-based deformers** is to embed the model in a coarse mesh (cage), then deform the coarse mesh and reconstruct the hi-res geometry. For example, we can model the vertices as weighted sums of the cage vertices:

$$v = \sum_{j=1}^{m} w_j(v) c_j$$

8.3.4 Skeletal Animation

Very often, shape implies the existence of a *skeleton,* and a skeleton imposes a lot of structure in ho a character can move.

The key idea behind **skeletal animation** is to animate just the skeleton (much less DOFs), and then have the mesh to follow automatically.

8.3.5 Forward Kinematics

We define *kinematic skeletons* as follows:

- Joints: local coordinate frames
- Bones: vectors between consecutive pairs of joints
- Each non-root bone defined in the frame of a unique parent
- Changes to parent frame affects all descendant bones
- Both skeleton and skin are designed in a rest pose

Assuming $n+1$ joints $0,\ 1, \ldots,\ n$, where joint 0 is the root, then each joint corresponds to a frame. $p(j)$ denotes the parent of joint j, and the frame of joint j is expressed w.r.t the frame of $p(j)$:

$$ {}_{p(j)}R_j = \begin{bmatrix} r_{11}(j) & r_{12}(j) & r_{13}(j) & t_1(j) \\ r_{21}(j) & r_{22}(j) & r_{23}(j) & t_2(j) \\ r_{31}(j) & r_{32}(j) & r_{33}(j) & t_3(j) \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} Rot(j) & t(j) \\ 0 & 1 \end{bmatrix}, $$

where $t(j)$ typically comes from a bind pose and $Rot(j)$ comes from some animation.

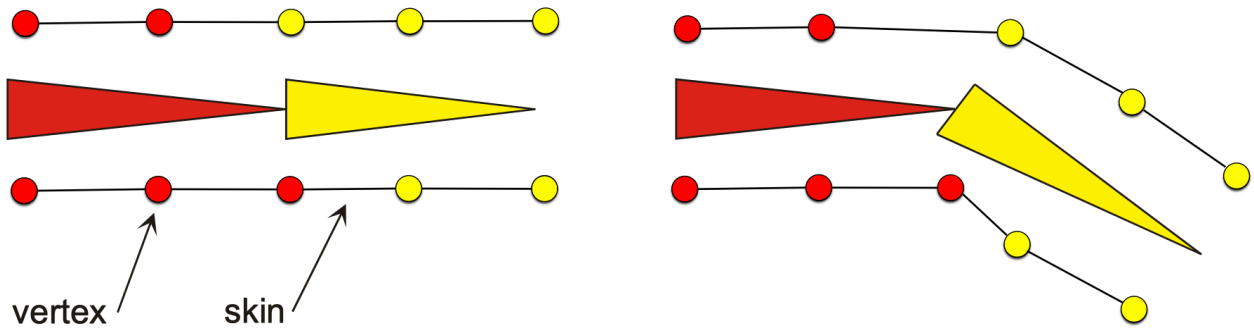
The transformation from frame j to world is then given by:

$$ \begin{aligned} {}_wR_j &= {}_wR_0 \cdots {}_{p(p(j))}R_{p(j)\ p(j)}R_j \\ &= T(0)Rot(0)...T(p(j))Rot(p(j))T(j)Rot(j) \end{aligned} $$

8.3.6 Skinning

The basic idea behind the *skinning process* is that we simply move the vertices of the skin along with the bones! In a first attempt, we might assign each vertex to the closest bone, compute the world coordinates according to the bone's transformation and move the skin vertices along with it:

$$ v = {}_wR_{j\ w}\bar{R}_j^{-1}v' $$

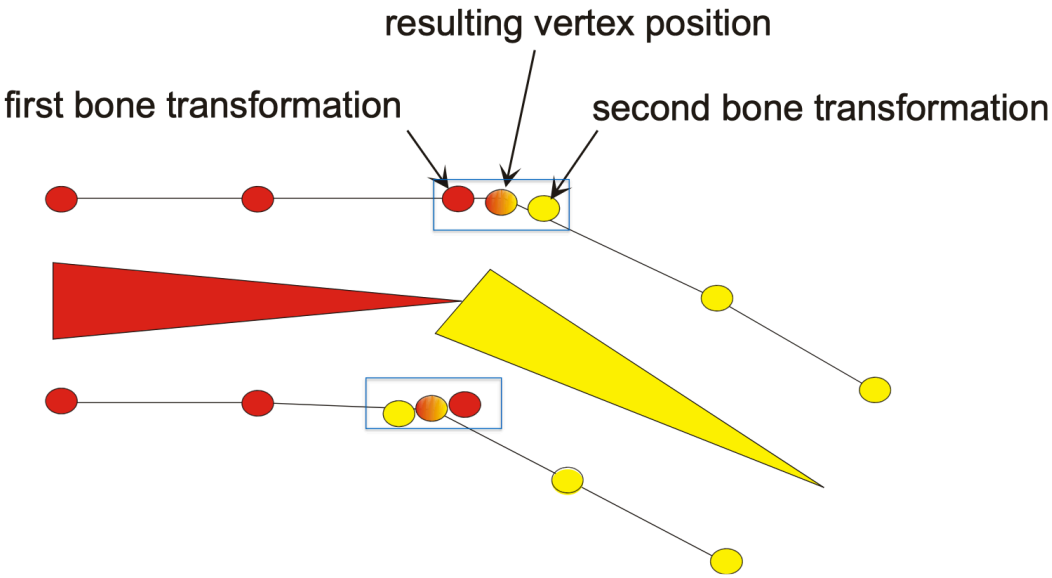


This process is also called **rigid skinning.**

Linear Blend Skinning

The attempt is similar to above, however we assign each vertex to multiple bones, and then compute the world coordinates as a convex combination. Weights define the influence of each bone on the vertex. This leads to an overall smoother deformation of the skin:

$$ v = \sum_j \alpha_{j\ w}R_{j\ \bar{w}}\bar{R}_j^{-1}v' $$



8.3.7 Inverse Kinematics

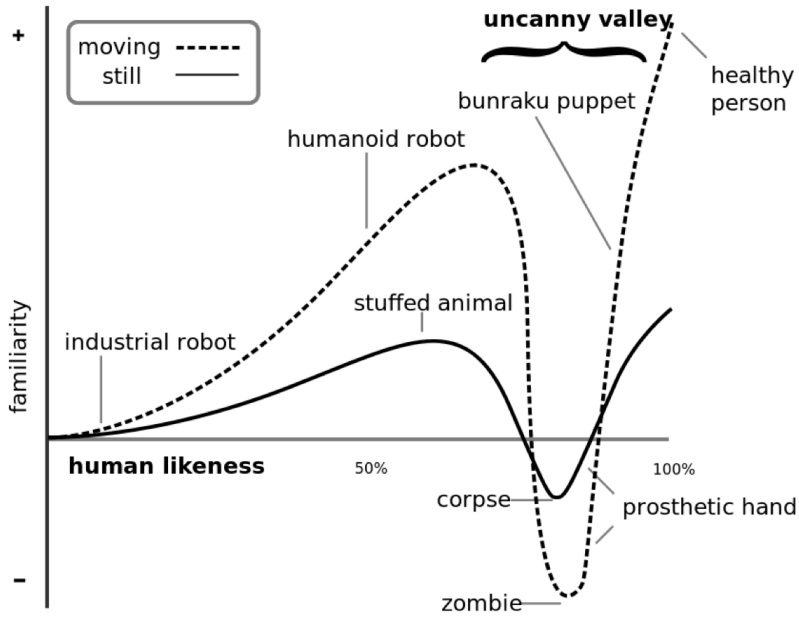
Inverse kinematics describes the problem that given some position of our character, compute the joint angles. This is one of the most fundamental techniques in animation and robotics!

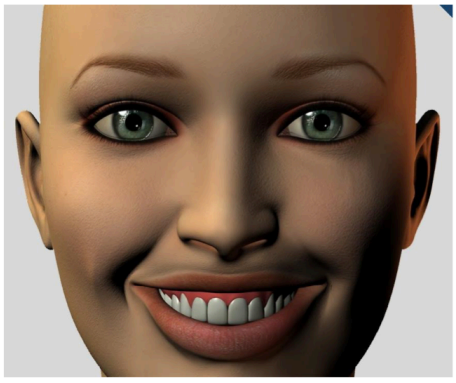
The basic idea behind an IK algorithm is as follows:

1. Write down the distance between the final point and the target and set up the objective
2. Compute the gradient with respect to angles
3. Go downhill from there

8.3.8 The Uncanny Valley

> The uncanny valley is a concept first introduced in the 1970s by Masahiro Mori, then a professor at the Tokyo Institute of Technology. Mori coined the term "uncanny valley" to describe his observation that as robots appear more humanlike, they become more appealing—but only up to a certain point. Upon reaching the uncanny valley, our affinity descends into a feeling of strangeness, a sense of unease, and a tendency to be scared or freaked out. So the uncanny valley can be defined as people's negative reaction to certain lifelike robots.





8.3.9 Motion Capture

Motion capture provides sparse signals, such as marker trajectories, from which a full body motion needs to be reconstructed.

This is a problem which is often-times posed as an optimization problem, for example in inverse kinematics.