Computer Graphics
Practical 4: Procedural animation

INSA Fourth Year - 2023/2024

Maud Marchal, Glenn Kerbiriou

1 Objectives

Procedural animation, also known as descriptive animation, keyframed animation or forward kinematic, consists in describing the motion of an object by specifying key postures of this object. The motion between the key postures, also called keyframes, is interpolated.

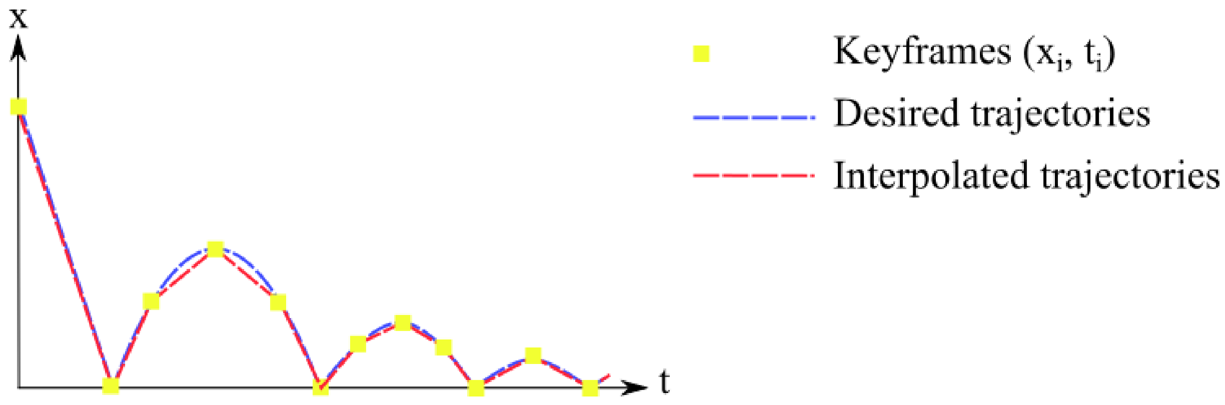
In Computer Graphics, any parameters of an object can be keyframed to create an animation, e.g. position, orientation, scale, colors and visibility.

The focus of this practical is to be able to keyframe the transformation matrices of a renderable in order to animate it.

1.1 About keyframe animation

Below, we quickly illustrate the keyframe process for a 1D point described by a x coordinates.

Let's imagine that you want to animate a bouncing ball. You have in mind the trajectory of this ball (the blue line) but you do not want to specify manually the position of the ball at each frame. Instead, you discretize the trajectory into keyframes (the yellow squares), which means that you specify the position of your object at specific times. By linearly interpolating the position of your object between the keyframes, you can approximate the trajectory you wanted (red line). The more keyframes you have, the more control you have (but also more work you have to do!).



For a time $t \in [t_i, t_{i+1}]$, the position x of the point is linearly interpolated between the corresponding keyframed positions $\{x_i, x_{i+1}\}$:

$$x(t) = \frac{t_{i+1} - t}{t_{i+1} - t_i} x_{t_i} + \frac{t - t_i}{t_{i+1} - t_i} x_{t_{i+1}} = (1 - f)x_{t_i} + f x_{t_{i+1}} \tag{1}$$

with $f = \frac{t-t_i}{t_{i+1}-t_i} \in [0, 1]$.
Such interpolations are quite efficient since they are linear. This is why we use as much as possible linear interpolation to achieve real-time or interactive performance.

1.2 About geometric transformation interpolation

In this practical, we propose you a keyframe system that can be applied to the local and parent transformations of `HierarchicalRenderable`. This means that at specific times, the transformation matrices are defined and the system interpolates between those keyframes to compute transformation matrices at any time. This will be used to animate our renderables.

How can we interpolate between two transformation matrices?

1.2.1 Linear interpolation of matrices is not enough

Let's have a look to the two following matrices. One is the identity matrix and the other one represents a rotation of 90 degrees around the X axis:

$$M_1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \text{ and } M_2 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Now, suppose we want to interpolate between these two matrices, so we create a animated rotation. Half the way, we expect the transformation to be a 45 degrees rotation around the axis X, something like this:

$$M = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} & 0 \\ 0 & \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

However, if we naively use linear interpolation, we get:

$$M = 0.5M_1 + 0.5M_2 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0.5 & -0.5 & 0 \\ 0 & 0.5 & 0.5 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

1.2.2 The solution: decomposition

To correctly interpolate geometric transformations, a commonly used solution is to decompose the transformations into three components (translation, scale and rotation) and interpolate each of them separately.

A transformation matrix M can be composed as follow:

$$M = \begin{pmatrix} R_{xx} \times s_x & R_{xy} \times s_y & R_{xz} \times s_z & t_x \\ R_{yx} \times s_x & R_{yy} \times s_y & R_{yz} \times s_z & t_y \\ R_{zx} \times s_x & R_{zy} \times s_y & R_{zz} \times s_z & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

with :

$T = (t_x t_y t_z)$ the translation, $S = (s_x s_y s_z)$ the scale and $R = \begin{pmatrix} R_{xx} & R_{xy} & R_{xz} \\ R_{yx} & R_{yy} & R_{yz} \\ R_{zx} & R_{zy} & R_{zz} \end{pmatrix}$ the rotation.

Thus, the rotation component can be converted into a quaternion which is better suited for interpolation. For this practical, no deep understanding of the quaternions is required. Just remember that they alleviate the matrix interpolation issue and must be of unit length to represent a valid rotation.

2 Exercise 1: complete the interpolation framework

Recall that most of the provided renderables inherit from `KeyframedHierarchicalRenderable`.

- Read carefully `KeyframedHierarchicalRenderable.hpp/cpp` and `KeyframeCollection.hpp/cpp` to understand how the keyframe logic is (partially) implemented in the framework.
- Read the todos markups and implement the function `KeyframeCollection::interpolateTransformation()`.
- Read and complete the file `practical4.cpp` in `sampleProject` folder to understand how to build a keyframed animation with the framework.

3 Exercise 2: Project-related assignment

Once the framework completed and the given examples running, you should be ready to start working on the animations in your project.