Improved Interpolation

Interpolation plays an important role for motion compensation with improved fractional pixel accuracy. The more precise interpolated we get, the smaller our prediction residual becomes, and the more compression we get. How do we get improved and practical interpolation?

In Computer Graphics we have an analog problem of interpolating samples, but there it is samples given by a model or a designer, instead of the pixel grid of our image. Hence it makes sense to take a look at the approaches in Computer Graphics:

vorgegebene Punkte (z.B. unsere Kontrollpunkte)

interpolierte Kurve mit verschiedenen gewünschten Eigenschaften
Parallel to Nyquist-Sampling Theorem for reconstruction:

- Interpolation of samples.
- Tells us how to do the interpolation:

  With an ideal low pass filter, whose upper cutoff frequency is half the sampling rate. The samples are our control points.

Nyquist Reconstruction (or interpolation): Our samples correspond to our control points. The result of their low pass filterings is our interpolation.
Nyquist reconstruction assures that there are no frequency components above the Nyquist frequency, hence

→ minimum frequency of change, maximally smooth interpolated curve.

\[ \text{sinc}(t) = \frac{\sin(\pi t)}{\pi t} \]

Impulse response of our ideal low pass filter as interpolation function:

Sinc-Funktion

Tiefpassfilterung: **Faltung** mit dieser Funktion
If our samples are on integer sample points, the interpolation function of the other samples have a zero there, hence the interpolated curve has no influence of the other samples there, and the curve will go through the sample values.
Filtering is a convolution of our samples with the Sinc function:

\[ a(t) \ast b(t) = \int a(n) \cdot b(t-n) \cdot dn \]

\[ \sum_{n=0}^{\infty} x(n) \cdot \delta(t-n) \ast \text{sinc}(t) \]

Nyquist

Samples/Kontrollpunkte

Deltafunktion – (Kronecker Delta, Integral über Kronecker Delta ist 1)
\[ \delta(t-n) \ast \text{sinc}(t) = \text{sinc}(t-n) \]

\[ \left[ \sum_{n=0}^{\infty} x(n) \cdot \delta(t-n) \right] \ast \text{sinc}(t) \]

\[ = \sum_{n=0}^{\infty} x(n) \cdot \text{sinc}(t-n) \]
Sinc-Funktion ist unendliches Polynom, unendlich oft stetig differenzierbar

→ Interpolierte Kurve ist unendlich oft differenzierbar!
→ Besser als z.B. Kubische Splines, die nur bis zur 2. Ableitung stetig differenzierbar sind.
**Problem:** Sinc-Funktion ist unendlich lang, nicht so praktikabel.

**Alternative Möglichkeiten:** Faltung unserer Samples mit der sog. „Box-Funktion“ als Basisfunktion

**Resultat:** Wert wird beibehalten bis der Nächste kommt.
→ keine Stetigkeit

Weitere Möglichkeit zum Erhalt der Stetigkeit:

**Lineare Interpolation:**

Basisfunktion ist Box-Funktion mit sich selbst gefaltet.

\[
h_2(x) = \sum_{i=0}^{1} (*)
\]

Interpolation mit dieser Basisfunktion
Observe: The convolution in space or time is a multiplication in the frequency domain. Hence a convolution of 2 box functions becomes a squared sinc function in the frequency domain!

Beachte: Basisfunktion ist 0 bei den anderen Abtastwerten → Interpolierte Kurve geht durch unsere Abtastwerte.
→ C0 Stetigkeit, aber keine C1 Stetigkeit der ersten Ableitung

→ Nächster Schritt:
\[ h_4(x) = h_2(x) \ast h_2(x) = h_1(x) \ast h_1(x) \ast h_1(x) \ast h_1(x) \]

→ kubisches B-Spline

Observe: In the frequency domain, this B-Spline
is a sinc function to the 4th power! (sinc⁴)
(The “B” is for “Basis”)

z.B. am Punkt t=1
gilt:
(Gleiches gilt für t=2,3 …)

\[ Q_1(1) = Q_2(0) \quad \rightarrow \quad 2. \text{ Ableitung nach } t \]
\[ Q_1(1) = Q_2(0) \quad \rightarrow \quad 1. \text{ Ableitung nach } t \]
\[ Q_1(1) = Q_2(0) \quad \rightarrow \quad \text{ Funktionswerte} \]

\[ h_4(t) = \frac{4}{6} Q_1(u) \]

each interval is 0 ≤ u ≤ 1
has own polynomial

\[ \frac{d^4 Q(t)}{dt^4} = \delta(t) - 4 \delta(t-1) + 6 \delta(t-2) - 4 \delta(t-3) + \delta(t-4) \]

Observe that in this way we get a very compact support, which means our impulse response is
quite short (length only from 0 to 4), and it can be represented by polynomial pieces $Q_i(u)$ of third order. Both makes it quite efficient for an implementation, which is important for video coding purposes. The drawback is now that the B-Spline does not go through 0 at the other samples (at positions 1 and 3), such that an interpolation with a simple convolution will not go through the original sampling points/values. To fix this, we can pre-process the original signal (image) first, such that the original samples are again obtained after the convolution with the B-Splines, as we will see.

The values of the B-Spline and its derivates are:

$$
\frac{d Q_1}{du} \bigg|_{u=1} = \frac{1}{2} \quad \rightarrow \quad c_1 = \frac{1}{2}
$$

$$
\rightarrow \quad \frac{d Q_2}{du} = -\frac{3}{2} u^2 + u + \frac{1}{2}
$$

$$
\rightarrow \quad Q_2(u) = -\frac{1}{2} u^3 + \frac{u^2}{2} + \frac{1}{2} u + c_0
$$

$$
Q_2(1) = \frac{1}{6} \quad \rightarrow \quad c_0 = \frac{1}{6}
$$

$$
\rightarrow \quad \underline{Q_2(u)} = -\frac{1}{2} u^3 + \frac{u^2}{2} + \frac{1}{2} u + \frac{1}{6} = \frac{1}{6} (-3u^3 + 3u^2 + 3u + 1)
$$

$$
Q_2(0) = \frac{1}{6}, \quad Q_2(1) = -\frac{1}{2} + \frac{1}{2} + \frac{1}{6} = \frac{1}{2} + \frac{1}{6} = \frac{4}{6}
$$
Here we can see that our B-Spline has the value 1/6 at position 1 and 3 \( Q_2(0)=1/6 \) and it has the value of 4/6 at its center at position 2 \( Q_2(1)=4/6 \). Hence the centered sample will be multiplied (after convolution) with the factor of 4/6, and the 2 neighbouring samples with 1/6. This is also illustrated in the following picture (made with Octave)
The interpolation of the sequence 1,2,3 with splines as an example can be seen in this picture:

Here we have an interpolated grid of $1/10^{th}$ of a pixel. We can also see that the interpolation does not go through the original pixel values, even though it is a nice and smooth interpolation. How do we obtain a spline interpolation through the original pixels? To see that, we first need to analyse what happens with the original pixels when we interpolate with the B-Spline. Assume we already have a **pre-processing** of
our original image, which produces the pixel values $c(n)$ out of our image pixels $x(n)$. The question is, how do we produce the $c(n)$ such that the original pixel values $x(n)$ are maintained after the spline interpolation.

We would like to have the original, which means we would like to have:

$$x(n) = \frac{1}{6}c(n-1) + \frac{4}{6}c(n) + \frac{1}{6}c(n+1)$$

Here the factor $1/6$ comes from the spline value at the neighboring positions 1 and 3, and the factor $4/6$ comes from the spline value at its center position 2.

c comes from our pre-processed or “compensated” image (pre-processing with some filtering operation). So here we just look at the integer pixel positions.

Observe that this condition has to hold for all inter values of $n$! How do we obtain $c(n)$?

We need to apply the z-transform as a tool, because it is made for such recursive equations. Delays or advances can be formulated as multiplications with $z^{-1}$ or with $z$.

The above equation in the z-domain is:

$$X(z) = \frac{1}{6}z^{-1} \cdot C(z) + \frac{4}{6}C(z) + \frac{1}{6}z \cdot C(z)$$

This can be re-written as:
\[ X(z) = C(z) \cdot (1/6 \cdot z^{-1} + 4/6 + 1/6 \cdot z) \]

and we obtain following expression for \( C(z) \):

\[
C(z) = \frac{X(z)}{1/6 \cdot z^{-1} + 4/6 + 1/6 \cdot z} \quad (1)
\]

Now we can see that we obtain an IIR filter for the generation of \( C(z) \) or \( c(n) \)!

Note that we have a non-causal part \( z \) in our equation for \( c(n) \). In images we can solve this problem by separating a causal and an anti-causal filter, and then apply the anti-causal filter backwards over our image, as shown in following images.

First we go along the causal direction, from left to right or from up to down, and generate the intermediate signal \( p(n) \), with an IIR filter with pole at \( z_1 \):

\[
x(n) \quad p(n) = x(n) + z_1 \cdot p(n-1)
\]

Its transfer function in the \( z \)-domain hence becomes:

\[
P(z) = X(z) + z_1 \cdot z^{-1} \cdot P(z)
\]

\[
P(z)(1 - z_1 \cdot z^{-1}) = X(z)
\]
\[
\frac{P(z)}{X(z)} = \frac{1}{1 - z_1 \cdot z^{-1}}
\]

We see that it has indeed a pole at position \(z = z_1\).

Then we take the intermediate signal \(p(n)\) and filter it in the anti-causal direction, from right to left or from down to up, with an IIR filter with the anti causal pole (pole in \(z^{-1}\)) at \(z_2\), which is identical to a causal pole at \(1/z_2\):

\[
p(n) \quad c(n) = p(n) + z_2 \cdot c(n+1)
\]

Its transfer function becomes (using again the \(z\)-transform):

\[
C(z) = P(z) + z_2 \cdot z \cdot C(z)
\]

\[
C(z) \cdot (1 - z_2 \cdot z) = P(z)
\]

\[
\frac{C(z)}{P(z)} = \frac{1}{1 - z_2 \cdot z}
\]

This transfer function has indeed a pole at position \(z = 1/z_2\).

Our total transfer function of the causal and anti-causal structure in series becomes:

\[
\frac{C(z)}{X(z)} = \frac{P(z) \cdot C(z)}{X(z) \cdot P(z)} = \frac{1}{(1 - z_2 \cdot z) \cdot (1 - z_1 \cdot z^{-1})}
\]
\[
\frac{1}{-z_2 \cdot z (1 - z_2^{-1} \cdot z^{-1}) \cdot (1 - z_1 \cdot z^{-1})}
\]
Here we can see that the denominator has 2 pole sections. For this separation into the 2 poles in the denominator, we need to find the poles of our original denominator polynomial in (1). We can do it with Matlab or Octave: Search the roots of the denominator polynomial \([1/6, 4/6, 1/6]\) with:
```
roots([...]),
```
and obtain the roots at -0.26795 and -3.73205. Observe that -1/3.73205 = -0.26795.
And we obtain
\[
\frac{1}{6} \cdot z^{-1} + 4/6 + 1/6 \cdot z = z \cdot (1/6 \cdot z^{-2} + 4/6 \cdot z^{-1} + 1/6)
\]
\[
= \frac{z}{6} \cdot (1 + 0.26795 \cdot z^{-1}) (1 + 3.73205 \cdot z^{-1})
\]
Hence we can rewrite our desired compensation filter \(C(z)\) as
\[
C(z) = \frac{X(z)}{\frac{1}{6} \cdot (1 + 0.26795 \cdot z^{-1}) (z + 3.73205)}
\]
\[
= \frac{X(z)}{\frac{3.73205}{6} \cdot (1 + 0.26795 \cdot z^{-1}) (0.26795 \cdot z + 1)}
\]
Here we obtain one causal and one anti-causal pole at -0.26795.
This is exactly what the HHI proposal is:
\[
p[k] = s[k] + z_1 \cdot p[k - 1], \quad k = 1, \ldots, N - 1 \quad (3)
\]
\[
c[k] = z_1 \cdot (c[k + 1] - p[k]), \quad k = N - 2, \ldots, 0 \quad (4)
\]
where \( z_1 = -0.26795 \), and where the resulting \( c(k) \) are used for the spline interpolation (\( p_k \) is the result of the causal filter part, \( c(k) \) is our \( c(n) \), \( s(k) \) is our \( x(n) \)).

The resulting system is here:

![Diagram](image)

*Fig. 1. Motion-compensated prediction using generalized interpolation. The IIR filtering block in dashed lines constitutes the main difference to the standard techniques.*

The block IIR filter generates the \( c \)'s, and the FIR filter is the spline interpolator. “In-Loop processing” is anti-blocking filtering and similar.

(From: 28th Picture Coding Symposium, PCS2010, December 8-10, 2010, Nagoya, Japan, “FRACTIONAL-SAMPLE MOTION COMPENSATION USING GENERALIZED INTERPOLATION” Haricharan Lakshman, Benjamin Bross, Heiko Schwarz, and
Thomas Wiegand}