ISSN 2464-4617 (print)
ISSN 2464-4625 (DVD)
Computer Science Research Notes
CSRN 2901
WSCG Proceedings Part I

Cubic Spline Interpolation in Real-Time Applications using Three Control Points

Jens Ogniewski
Linköping University
Information Coding Group
Department of Electrical Engineering
581 83 Linköping, Sweden
jens.ogniewski@liu.se

ABSTRACT

Spline interpolation is widely used in many different applications like computer graphics, animations and robotics. Many of these applications are run in real-time with constraints on computational complexity, thus fueling the need for computational inexpensive, real-time, continuous and loop-free data interpolation techniques. Often Catmull-Rom splines are used, which use four control-points: the two points between which to interpolate as well as the point directly before and the one directly after. If interpolating over time, this last point will ly in the future. However, in real-time applications future values may not be known in advance, meaning that Catmull-Rom splines are not applicable. In this paper we introduce another family of interpolation splines (dubbed Three-Point-Splines) which show the same characteristics as Catmull-Rom, but which use only three control-points, omitting the one "in the future". Therefore they can generate smooth interpolation curves even in applications which do not have knowledge of future points, without the need for more computational complex methods. The generated curves are more rigid than Catmull-Rom, and because of that the Three-Point-Splines will not generate self-intersections within an interpolated curve segment, a property that has to be introduced to Catmull-Rom by careful parameterization. Thus, the Three-Point-Splines allow for greater freedom in parameterization, and can therefore be adapted to the application at hand, e.g. to a requested curvature or limitations on acceleration/deceleration. We will also show a method that allows to change the control-points during an ongoing interpolation, both with Thee-Point-Splines as well as with Catmull-Rom splines.

Keywords

Spline Interpolation, Parameterization, Real-Time, Animation, Catmull-Rom Splines

1 INTRODUCTION

Spline interpolation is widely used for the creation of smooth paths e.g. in computer graphics or for the motion of robots. These splines have typically to fulfill four mathematical properties:

1. they have to be (at least) C^1 continuous,
2. they have to be interpolating splines, i.e. pass through all of their control points,
3. they have to be affine invariant, i.e. the sum of all blending polynomials is always 1, for all values of the interpolation variable, and
4. they must allow for local control, i.e. each control-point influences only a small, compact part of the overall generated curve.

Probably the most used interpolating splines are Catmull-Rom splines (CR), which fulfill all of the above properties, are fast to calculate and simple to use. CR use 4 control points - the two defining the current interpolation segment as well as the one directly before and the one directly after; if interpolating over time the latter point will ly in the future. However, in real-time applications limited information is available about future points, especially there might be no future information available apart from the next control point, rendering CR and all other conventional interpolation splines unsuitable. The spline in this paper, called Three-Point-Splines (TPS), exhibits the same properties as CR, but does not rely on the knowledge of any future points, and is thus applicable even in these scenarios.

In some cases the path to the next control point might be blocked, e.g. by a moving object, normally neces-

sitating motion replanning and probably stopping the current motion. As alternative, a method is introduced enabling to change the next control-point during an ongoing interpolation without loosing any of their properties, for both TPS and CR. This can be used e.g. in animations for computer games or for instant reactions of robots to a changed environment, thus enabling the robots to perform safer and more efficiently, a critical factor for their success in real-world applications [ET14].
Furthermore, TPS exhibit a very rigid behavior, and less variation as comparable splines (like e.g. CR). Because of this the curves generated by TPS are naturally self-intersection-free, a property that has to be introduced to CR by careful parameterization. In contrast, TPS allow for a range of different parameterizations that all are guaranteed to be self-intersection-free, and can therefore be adapted to meet the requirements of the application like e.g. limits on speed or acceleration/deceleration. Typically, reparameterizations are needed to ensure that these limits are kept, which introduces computational overhead. Also, not all reparameterizations have an analytical solution (like e.g. arc-length parameterization).
Finally, it is shown that the TPS are of lower computational complexity than CR which is important for real-time systems with limited computational capabilities.
TPS have only been described once earlier, in a conference poster by the author [Ogn13]. This paper extends the earlier work by describing the exact properties of TPS and by introducing a recursive evaluation form and parameterization scheme, needed for artifact-reduction. The change of the next control-point during an ongoing interpolation has not been described earlier.

1.1 Related Work

TPS are asymmetric splines that use three control-points. Asymmetric interpolation, in the sense of using a different number of points before (i.e. "in the past" of) and after (i.e. "in the furture" of) the current interpolation segment, was first proposed in [Yua96], albeit in a more theoretical-stochastic fashion, without a discussion of specific splines or their mathematical properties. Interpolation using three points was first explored in [Dod97]. However, in their work only functions of second degree were considered, and thus it was not possible to construct an interpolating spline and reach C^1 continuity at the same time. Furthermore, their interpolation scheme is symmetric, and uses a point in the past for the first half of the interpolation segment and a point in the future for the second half. An interpolating spline which does not use future but reaches a higher continuity than C^0 has to be asymmetric by definition.
Edwin Catmull and Raphael Rom introduced their splines already in [CR74], and later publications built on this initial approach, for example [DB88] or [KB84] which explored the basic properties of the splines further. The latter one even introduced an additional relaxation term, with which C^1 continuity can no longer be guaranteed though. Both did not explore more advanced parameterization techniques, which were made possible by the introduction of recursive evaluation forms in [JAW67]. Correct parameterization has been proven to be a vital part of spline interpolation, since uniform parameterizations can lead to unwanted behavior like loops. Thus, a number of different publications have examined this issue, often based on the findings of [IDS99], where for the first time a scheme was proposed to include non-uniform parameterization in CR and similar splines. Two of the most popular schemes are the centripetal [Lee89] and the chordal [ML88] scheme, which have been applied to CR in [NDH09]. The work described in [CYK11] built on this further by applying methods introduced in [DM96] and [Flo08] to prove that the centripetal parameterization is the only scheme that guarantees no self-intersections in CR.

1.2 Definitions

In this paper piecewise polynonmial interpolation is considered over a number of control-points $\mathbf{P}_i$, where the current interpolation segment $\mathbf{r}$ is interpolating between the control-points $\mathbf{P}_r$ and $\mathbf{P}_{r+1}$. Although only the properties of this segment are considered, its properties apply to all other segments as well, and especially the continuation between segments is guaranteed if not stated otherwise.
The following definitions are used:

$$\mathbf{D}_{a,b} = \mathbf{P}_a - \mathbf{P}_b$$

for the difference vector (between two points), and

$$\mathbf{l}_{(t)} = t\mathbf{D}_{r+1,r} + \mathbf{P}_r, t \in \mathbb{R}$$

for the line spawned by the two control-points of the interpolation segment.
The rest of this paper is organized as follows: section 2 introduces the uniform form of TPS interpolation, section 3 its recursive version, section 4 shows how control-points can be changed during an ongoing interpolation (for both TPS and for CR), and section 5 concludes the paper. Proofs and Derivations can be found in the appendix.

ISBN 978-80-86943-37-4

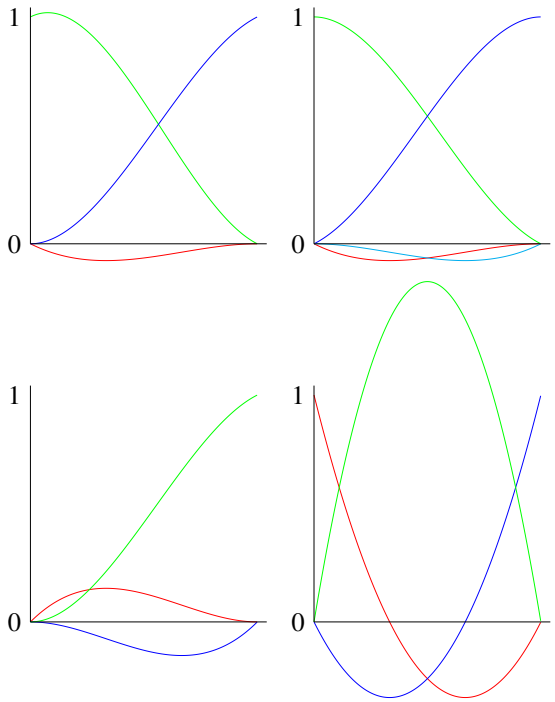

Figure 1: Blending polynomials:
Upper row: Three-Point-Spline (left) and Catmull-Rom (right): Red: polynomial for $\mathbf{P}_{r-1}$, green: polynomial for $\mathbf{P}_r$ and blue: polynomial for $\mathbf{P}_{r+1}$; Catmull-Rom includes an additional polynomial (cyan) for $\mathbf{P}_{r+2}$; all polynomials use an α of 0.5
Lower row: the polynomials for the difference vector form (left) and their derivatives (right):
$u(u-1)^2$ (red), $u^2(u-1)$ (blue) and $u^2(3-2u)$ (green)

2 DERIVATION OF THE UNIFORM FORM

Taking the aforementioned mathematical properties as constraints, the interpolation equations are set up accordingly and lead to the following equation, with u as interpolation variable, $0 \leq u \leq 1$:

$$\begin{aligned}\mathbf{F}_{TPS(u)} = \quad & (-\alpha u^3 + 2\alpha u^2 - \alpha u)\mathbf{P}_{r-1} \\ + \quad & (2u^3 - (3+\alpha)u^2 + \alpha u + 1)\mathbf{P}_r \\ + \quad & ((\alpha-2)u^3 + (3-\alpha)u^2)\mathbf{P}_{r+1}\end{aligned} \tag{1}$$

with α a parameterization value that can be chosen freely. Figure 1 shows the blending polynomials, with the ones from CR for comparison.
These equation can be rewritten to be based on the differences of points rather than based on points:

$$\begin{aligned}\mathbf{F}_{TPS(u)} = \quad & \alpha u(u-1)^2\mathbf{D}_{r,r-1} \\ + \quad & \alpha u^2(u-1)\mathbf{D}_{r+1,r} \\ + \quad & u^2(3-2u)\mathbf{D}_{r+1,r} \\ + \quad & \mathbf{P}_r\end{aligned} \tag{2}$$

where the first two terms blend the derivatives at points $\mathbf{P}_r$ and $\mathbf{P}_{r+1}$ respectively, while being 0 at both endpoints. The rest of the equation interpolates between the two points of the segment, while having a zero derivative at the endpoints (see also figure 1). Thus, this form gives a good overview of how the spline will behave (a similar form was already used in [KB84] albeit not for TPS). For comparison, uniform CR can be written in a similar way:

$$\begin{aligned}\mathbf{F}_{CR(u)} = \quad & \alpha u(u-1)^2\mathbf{D}_{r+1,r-1} \\ + \quad & \alpha u^2(u-1)\mathbf{D}_{r+2,r} \\ + \quad & u^2(3-2u)\mathbf{D}_{r+1,r} \\ + \quad & \mathbf{P}_r\end{aligned} \tag{3}$$

Note that CR includes an additional "future" point $\mathbf{P}_{r+2}$. The main difference between CR and TPS is which difference vectors are used for the derivatives, which has a high impact on how the splines behave. Especially, using the same difference vector in the second and the third term is exactly what gives the TPS their rigid behavior.
Incidentally, the α values for the different segments do not need to be the same, as long as they are the same where the same segment is concerned, i.e. the α value from the first term (in equation 2) needs to be the same α value used for the second term in the previous segment, or otherwise the interpolation will only be G^1 continuous, which is true for both CR and TPS.
Of special interest is of course the maximal distance $d_{\max}$ of the curve segment generated by TPS to the line $\mathbf{l}_{(t)}$, as already discussed in [Flo08] and [CYK11]. Also, this distance is needed to prove that TPS will never generate loops (for parameterizations inside a sensible bound), but might lead to overshots.
We define $\alpha a_{(u)} = \alpha u(u-1)^2$, blending $\mathbf{D}_{r,r-1}$, and $b_{(u)} = u^2((\alpha-2)u + (3-\alpha))$ (combining the second and the third blending polynomials of equation 2), blending $\mathbf{D}_{r+1,r}$. Inserting these functions into the point-line-distance equation yields (see also the appendix 7.1.2):

$$d_{(u)} = ||\mathbf{D}_{r,r-1}||\alpha a_{(u)}(1-\cos^2\theta)^{\frac{1}{2}} \tag{4}$$

with θ the angle between $\mathbf{D}_{r,r-1}$ and $\mathbf{D}_{r+1,r}$.
From the derivative $d'_{(u)}$ of equation 4 it is clear that only one extreme point exists for $0 < u < 1$, a maximum which lies at $u = \frac{1}{3}$, i.e. where the derivative of $a_{(u)}$ becomes zero. Thus, the position of the maximal distance of the curve segment from the line $\mathbf{l}_{(t)}$ lies at the same u-value in each segment and the maximum distance can therefore be easily calculated.
The three points used for the current interpolation segment ($\mathbf{P_{r-1}}$, $\mathbf{P_r}$, $\mathbf{P_{r+1}}$) span a plane on which all points of the generated curve for the current segment lie (the special case where all lie on the line $\mathbf{l}_{(t)}$ will be discussed later). The plane also contains the line $\mathbf{l}_{(t)}$. As was shown earlier, the distance to this line depends

ISBN 978-80-86943-37-4

only on the a-part of the interpolation function $\mathbf{F}_{CR(u)}$ (i.e. the b-part of the function lies completely on the $\mathbf{l}_{(t)}$), which is non-zero for $0 < u < 1$. Since a has no zero-point in the interpolation interval all points of the generated curve-segment lie on the same side of the line.

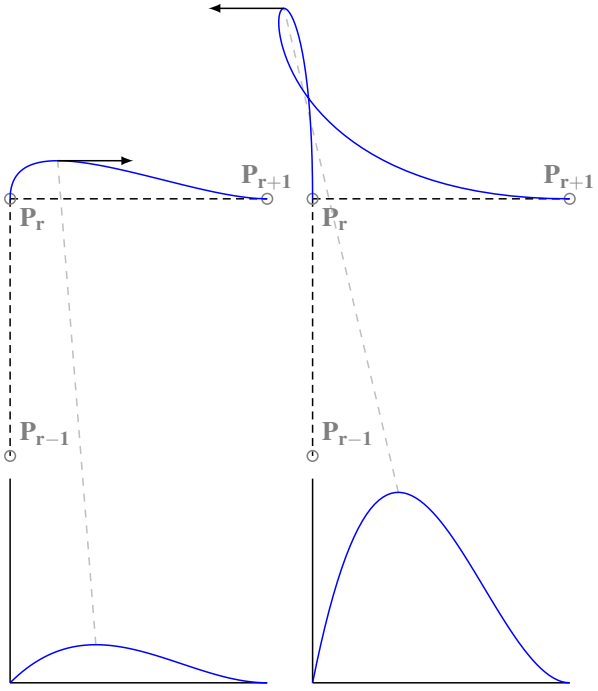

Figure 2: the different behavior of the three-point-spline with different αs: The interpolating curves (upper row) and the distance to the line $\mathbf{l}_{(t)}$ (lower row)
a) (left) with $\alpha = 1$, representing the case of $0 < \alpha < 3$
c) (right) $\alpha = 5$, representing the case of $\alpha >= 4$
The direction of the tangents in the respective extreme-points (i.e. the maximum distances to $\mathbf{l}_{(t)}$) are visualized by arrows; note that they are not in the same scale as the curves for better visualization

The interpolation function can be divided into two parts: a part e whose tangents are parallel to the line (i.e. which blend the vector $\mathbf{D}_{r+1,r}$) and a part f whose tangents are perpendicular to the line. While the f function-part only depends on the a-part of the interpolation function (as was shown earlier), the e function-part might depend on both a and b since $\mathbf{D}_{r,r-1}$ is not necessarily perpendicular to $\mathbf{D}_{r+1,r}$.
If the generated curve has a loop, both function-parts e and f change the signs of their tangents at some point in the loop (i.e. move into one direction first, than into the other, see also figure 2), which means that the loop has to contain an extreme point for both e and f, i.e. both function-parts need to have an extreme point in the interval $0 < u < 1$. It was already shown that the a-function-part has a maximum, but since it is only depending on one vector (i.e. all points generated by a lie on a segment of the line through $\mathbf{P_{r-1}}$ and $\mathbf{P_r}$) this is not enough to cause a loop, and whether a loop exist or not depends on the extreme-points of b. Looking at the derivative of b it becomes obvious that it contains one at $u = 0$ (which it does by definition of the blending functions), and another one that depends on the chosen value for α:

$\alpha = 0$: at $u = 1$
$0 < \alpha < 2$: at $u > 1$
$\alpha = 2$: at $u = 0$
$2 < \alpha < 3$: at $u < 0$
$\alpha = 3$: at $u = 0$
$\alpha > 3$: at $0 < u < 1$

This means that for $0 <= \alpha <= 3$ b does not have an extreme point in the interpolation interval and thus the generated curve will not contain a loop.
For $\alpha > 3$, the first derivative of the b-function-part will be negative in the beginning of the interpolation segment, and become positive after the extreme-point. If this extreme point lies after or at the extreme-point of a (which is located at $u = 1/3$), the generated curve will contain a loop. Inserting $u = 1/3$ int b' yields:

$$b'_{(\frac{1}{3})} = (\tfrac{4}{3} - \tfrac{1}{3}\alpha), \tag{6}$$

Which means that the generated curve will contain a loop for $\alpha >= 4$. The interval $3 < \alpha < 4$ would need further analysis, but this is omitted here since the interval $0 <= \alpha <= 3$ is deemed to be large enough, since larger αs lead to curves with a large maximal distance to $\mathbf{l}_{(t)}$, and are therefore of limited use. Also, note that in CR freedom of loops is much harder to introduce, in fact only centripetal CR fulfills it [CYK11]; the reason for the comparable large range of possible parameterizations of TPS which are guaranteed to be free from loops lies in the rigid behavior of TPS.
Unfortunately, the case where all points of the current segment lie on the line $\mathbf{l}_{(t)}$ is harder to examine. For $\mathbf{D}_{r+1,r}$ and $\mathbf{D}_{r,r-1}$ pointing in exact opposite directions the generated line will contain a segment where the interpolation function moves into one direction first, than in the other. If $\mathbf{D}_{r+1,r}$ and $\mathbf{D}_{r,r-1}$ are pointing in the exact same direction this depends on the length of these two vectors, and on the value chosen for α. On a side-note, setting $\alpha = 1$ means that the function will behave like a linear blending function for $\mathbf{D}_{r,r-1} = \mathbf{D}_{r+1,r}$ (as with Catmull-Rom for $\alpha = 0.5$ and $\mathbf{D}_{r,r-1} = \mathbf{D}_{r+1,r} = \mathbf{D}_{r+2,r+1}$), which means it will interpolate with a constant first derivative i.e. it will move with a constant speed if interpolating over time.
Since the distance to $\mathbf{l}_{(t)}$ depends on α, setting α to a low value will lead to a curve that deviates only slightly from $\mathbf{l}_{(t)}$. However, this will also result in large changes of the length of the tangent vectors (i.e. changes in the

ISBN 978-80-86943-37-4

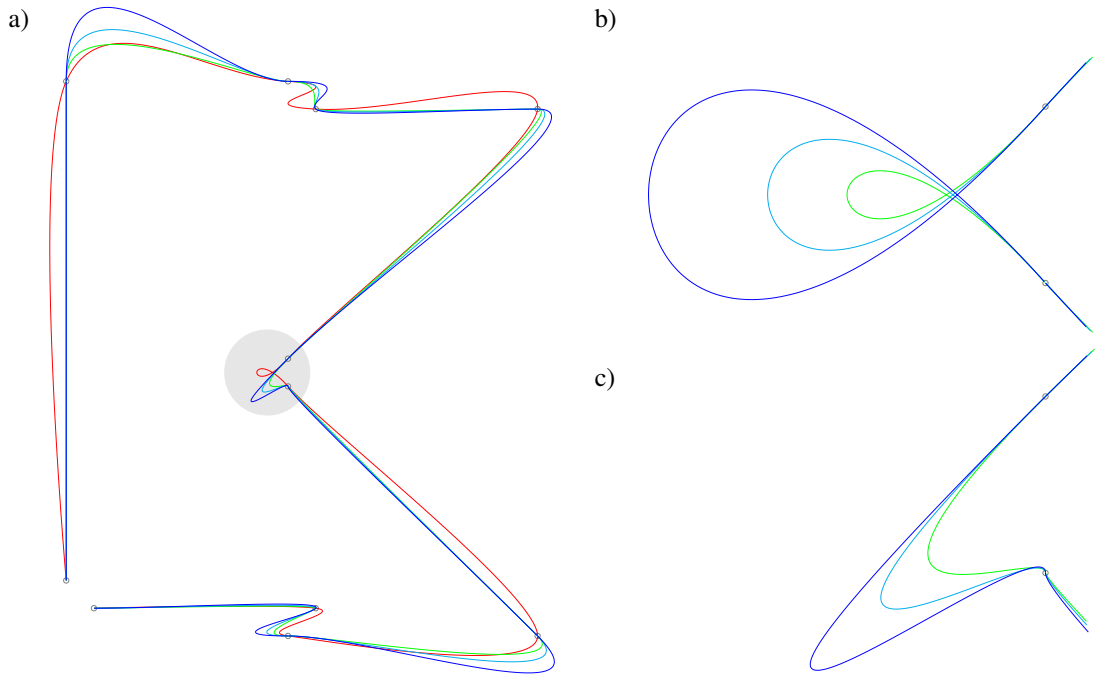

Figure 3: Comparison of CR and TPS with different parameterizations:
a) Left: CR with $\alpha = 0.5$ (red), TPS with $\alpha = 0.5$ (green), $\alpha = 0.7$ (cyan) and $\alpha = 1$ (blue)
Right: zoomed in on the light-gray circle:
b) Top right: CR and c) Bottom right: TPS, both with parameterizations of $\alpha = 0.5$ (green), $\alpha = 0.7$ (cyan) and $\alpha = 1$ (blue)

speed if used for animations or robot movement), since a small α results in short tangents in the control-points, but not all terms are bound by α (see also equation 2 and figure 1, to the right). Thus the tangents in the middle of the curve segments would be comparably large. Depending on the applications, limits can be determined on how far each curve segment is allowed to stray from $\mathbf{l}_{(t)}$, and how large changes of the lengths of the tangent-vectors are allowed to be, and thus a good value for α can be determined.
It is interesting to look at the derivative of TPS for $\alpha = 1.0$:

$$\begin{aligned} \mathbf{F}'_{TPS(u)} &= (3u^2 - 4u + 1)\mathbf{D}_{r,r-1} + (4u - 3u^2)\mathbf{D}_{r+1,r} \\ &= (4u - 3u^2)(\mathbf{D}_{r+1,r} - \mathbf{D}_{r,r-1}) + \mathbf{D}_{r,r-1} \end{aligned}$$

Thus, the derivative becomes an interpolation itself, between the derivatives at the point $\mathbf{P}_r$ and $\mathbf{P}_{r+1}$.
A positive α value means also that $\mathbf{P}_{r-1}$ and the point with the maximum distance to $\mathbf{l}_{(t)}$ will ly on opposite sides of $\mathbf{l}_{(t)}$. This means that if $\mathbf{P}_{r+2}$ lies between $\mathbf{l}_{(t)}$ and the generated curve segment, the next curve segment will cross the current one, thus generating a self-intersection in consecutive segments. For more than two dimensions, the points $\mathbf{P}_{r-1}$, $\mathbf{P}_r$, $\mathbf{P}_{r+1}$ and $\mathbf{P}_{r+2}$ need to lie in the same plane for this self-intersection to happen, since each curve segment of course lies completely in the plane spanned by the three points it uses for the interpolation. The reason for these possible self-intersections lies in the fact that it does not include a term containing the next interpolation segment (e.g. $\mathbf{D}_{r+2,r}$). Without this information, it is of course impossible to construct a curve that is guaranteed to be free of self-intersections in consecutive segments. Note that this applies even to the recursive form shown in the next section.
A comparison between different parameterizations of TPS and CR is given in figure 3. Although TPS do not lead to loops like CR, it introduces other artifacts in the form of overshots. Taking a look at the distance of the generated curve to the line $\mathbf{l}_{(t)}$ (equation 4), it becomes clear that it depends on the length of the last segment $\mathbf{D}_{r,r-1}$ rather than on the length of the current segment $\mathbf{D}_{r+1,r}$. Thus, if a long line segment is followed by a much shorter line segment the interpolation will unfortunately always overshoot in the shorter segment. A better parameterization to conquer this issue will be developed in the next section.

3 RECURSIVE EVALUATION FORM AND PARAMETERIZATION

Using TPS, one might be tempted to simply vary the different α values depending on the length of the

ISBN 978-80-86943-37-4

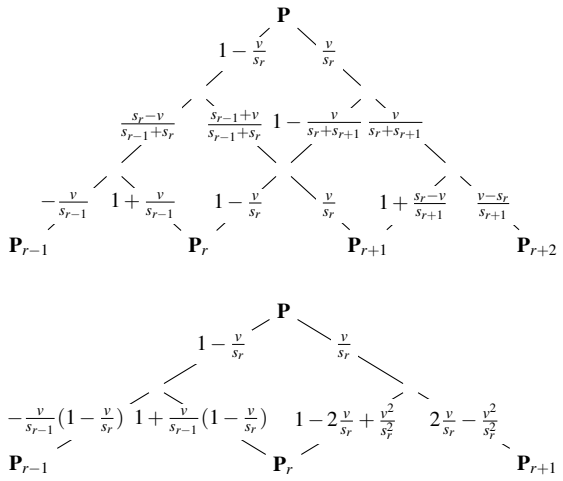

Figure 4: Different recursive formulations, with $s_r = ||D_{r+1,r}||^\beta$: a) (top) CR, b) (bottom) TPS

segment, but this can lead to unwanted side-effects, especially it would no longer be guaranteed that the α values would be inside the bounds established earlier. Instead, here one of the main ideas behind chordal, centripetal and other related parameterizations is used, namely varying the interpolation interval with the distance between the different points, using the interpolation variable v:

$$0 \leq v \leq s_r = ||\mathbf{D}_{r+1,r}||^\beta, \tag{7}$$

Conventionally, recursive CR-formulations are defined regarding to the sum of the lengths of the interpolation segments:

$$s'_0 = 0$$
$$s'_r = \sum_{i=0}^{r-1} s'_i + ||\mathbf{D}_{r,r-1}||^\beta \tag{8}$$

However, this more common formulation for recursive CR can be easily received from the one given here, and vice versa.

Using the formulation of (equation 7) reveals that the derivative of recursive CR in point $\mathbf{P}_r$ is no longer a simple scalar product of $\mathbf{D}_{r+1,r-1}$ multiplied by the interpolation parameter α but becomes:

$$\mathbf{F}'_{CR,rec(v=0)} = \frac{\frac{s_r}{s_{r-1}}\mathbf{D}_{r,r-1}+\frac{s_{r-1}}{s_r}\mathbf{D}_{r+1,r}}{s_{r-1}+s_r} \tag{9}$$

(similar for $v = s_r$ since it is still C^1 continuous.)

The centripetal and similar parameterizations of CR are facilitated by the fact that they can be expressed by a recursive evaluation form. Thus, to be able to use similar parameterizations of TPS they are first expressed in a recursive form as well, which is given in figure 4b), with the corresponding form of CR in figure 4a), and example interpolations in figure 5.

The function of the distance to the line $\mathbf{l}_{(t)}$ (equation 4) becomes for recursive TPS:

$$d_{(v)} = ||\mathbf{D}_{r+1,r}||^\beta ||\mathbf{D}_{r,r-1}||^{1-\beta} a'_{(v)} \left(1-\cos^2\theta\right)^{\frac{1}{2}}$$

with $a'_{(v)} = \frac{v}{s_r}(\frac{v}{s_r}-1)^2$ (10). (See also the appendix 7.2.1 for a more detailed derivation.)

Thus, β can be seen as a blending factor of how much the length of the current segment influences the distance to the infinite line $\mathbf{l}_{(t)}$, and how much the segment before it influences this distance. For $\beta = 1$ the distance to $\mathbf{l}_{(t)}$ is only depending on the length of the current segment, and not at all on the length of the segment before. For $\beta = 0$ it behaves exactly like equation 2 for $\alpha = 1$, i.e. it becomes a uniform parameterization. In the same way, the recursive formulation of CR becomes the uniform parameterization with $\alpha = 0.5$ by setting β to 0.

Since the distance to $\mathbf{l}_{(t)}$ is now bounded by the length of the current segment the generated curves do not overshoot anymore while interpolating comparably small segments, as can be seen in figure 5, which shows CR and TPS with different parameterizations. Also, an additional curve was added (in magenta) which uses different values for β, depending on the length of the segment. The idea behind this is to minimize the $||\mathbf{D}_{r+1,r}||^\beta$ and $||\mathbf{D}_{r,r-1}||^{1-\beta}$ factors (and thus the distance to $\mathbf{l}_{(t)}$)as much as possible.

Again it is possible to use different β values for different segments, just as it is to use different α valuess in the uniform versions, but again the same β value has to be used for the same segment, i.e. the β value used to blend $\mathbf{D}_{r,r-1}$ has to be equal to the β value used to blend $\mathbf{D}_{r+1,r}$ during the interpolation of the last segment, to maintain C^1 continuity. This applies to both TPS and CR, however for CR only the centripetal ($\beta = 0.5$) parameterization is guaranteed to be free from loops. For TPS β can be chosen freely, without risk for self-intersections (in the same interpolation segment), as will be shown next.

Similar as in section 2 the places of the extreme points can be found of the distance function to $\mathbf{l}_{(t)}$ for recursive TPS, and again only one extreme point exists, a maximum at $v = \frac{1}{3s_r}$, i.e. again after $\frac{1}{3}$ of the interpolation interval, which is not surprising taking the similar derivatives of the blending weights for $\mathbf{D}_{r,r-1}$ into account. From the derivative it becomes obvious that the blending function b'_v for $\mathbf{D}_{r+1,r}$ is always positive in this point, i.e. the interpolated curve will never contain any loops, independent of which value β has (See also the appendix 7.2.1 for a more detailed derivation.). It should be pointed out however that whereas centripetal CR is also free from self-intersections in neighboring segments, this property cannot be introduced to the TPS, due to the fact that it is missing a term for the next segment in its

ISBN 978-80-86943-37-4

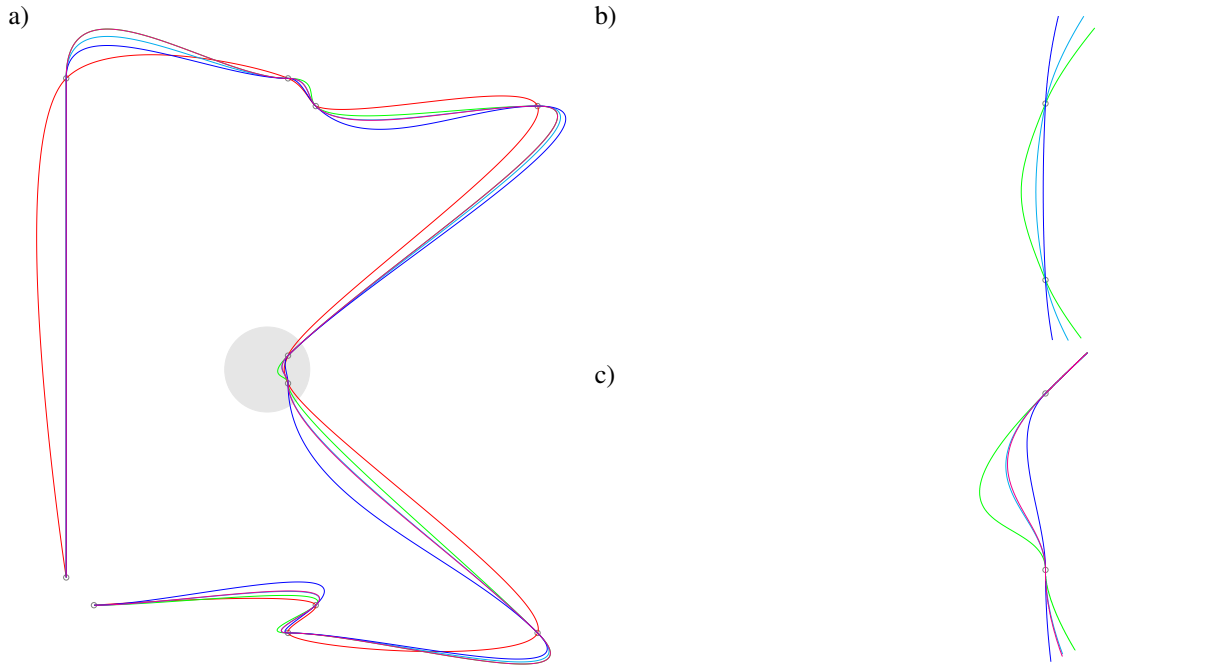

Figure 5: Comparison of centripetal CR and the recursive TPS with different parameterizations:
Left: centripetal CR (red), TPS with $\beta = 0.5$ (green), $\beta = 0.7$ (cyan) and $\beta = 1$ (blue)
a) Right: zoomed in on the light-gray circle:
b) Top right: CR and c) Bottom right: TPS, both with parameterizations of $\beta = 0.5$ (green), $\beta = 0.7$ (cyan) and $\beta = 1$ (blue)
An additional curve was added for a TPS with a variable parameterization, which uses $\beta = 0.5$ for the long segments and $\beta = 1$ for the short segment. Note that it in this example follows (and overlaps) very closely either the curve created by the TPS with $\beta = 0.5$ or the one with $\beta = 0.7$, depending on the segment.

equation, as already discussed in section 2. Thus, TPS will generate a self-intersection in two neighboring segments if the next control-point after the current segment lies in between $\mathbf{l}_{(t)}$ and the interpolation curve of the current segment and on the plane spanned by the three control-points used during the current segment.
Finally, since the recursive formulation of TPS behaves similar as the uniform version with $\alpha = 1$, the derivative of the recursive formulation of TPS is a blending function itself, of the derivatives at point $\mathbf{P}_r$ and $\mathbf{P}_{r+1}$ (see also the appendix 7.2.2):

$$\mathbf{F}'_{TPS,rec(v)} = (1 - h_{(v)}) \frac{\mathbf{D}_{r,r-1}}{||\mathbf{D}_{r,r-1}||^{\beta}} + h_{(v)} \frac{\mathbf{D}_{r+1,r}}{||\mathbf{D}_{r+1,r}||^{\beta}}$$
$$\text{with } h_{(v)} = 4\frac{v}{s_r} - 3(\frac{v}{s_r})^2 \qquad (11)$$

For a comparison in computation time both the uniform and the recursive version of both TPS and CR were implemented. For a fair comparison, each was implemented in different, hand-optimized ways and the respective fastest version was used; the reached optimizations should be comparable. The same $\alpha = \beta = 0.5$ were used for all the different interpolation methods and the same 1 000 000 randomly chosen control-points between which 100 intermediate points for each segment were interpolated. This was done on an Intel Core i7-6700 CPU running at 3.40GHz with 16 GBytes of RAM and gcc 5.4.0 with full optimization. The results are given in table 1.
It is counter-intuitive that the recursive version of TPS is faster than the uniform version, especially since it involves square roots - however with modern computers and compilers square roots can be computed very efficiently (in fact it took less than 4.5 s to calculate all the s values needed during the comparison) and need to be calculated only once for each interpolation segment. The remaining computation of the recursive version of TPS contains very few operations, less than needed for the uniform version. However, for the uniform version the minimal number of operations needed depends highly on the value chosen for α, and in fact for $\alpha = 1$ the uniform version needs fewer operations and thus performs marginally faster than the recursive version.

Computation time	uniform CR	uniform TPS	centripetal CR	recursive TPS
in seconds	1295	908	2009	699
in percent	65	45	100	35

Table 1: Timing results for the different methods.

ISBN 978-80-86943-37-4

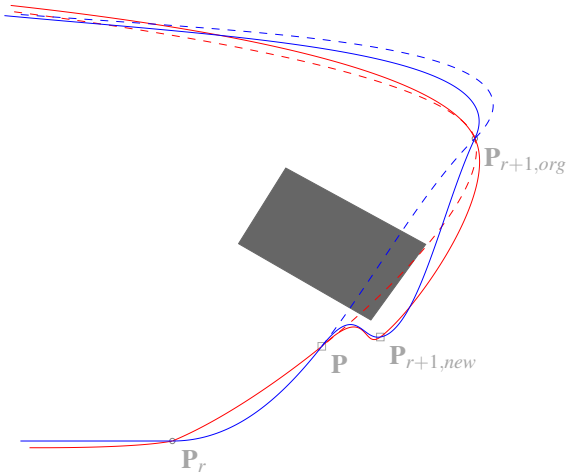

Figure 6: Example of a change of control-points during an ongoing interpolation:
A course is planned for a robot with centripetal CR (red) and recursive TPS (with $\beta = 0.5$, in blue) through a number of points given by the circles. However, during its run the robot notices a new obstacle (visualized by the gray rectangle) in point $\mathbf{P}$. The robot plots an evasion course through point $\mathbf{P}_{r+1,new}$ and thus is able to reach its destination without the need to stop for a replanning. The course it actually took is shown with solid lines, the planed one with dotted lines (note that it is partly overlapping with the robots actual course.)

4 CHANGE OF CONTROL POINTS DURING AN ONGOING INTERPOLATION

Here, it will be shown how it is possible to change the next control-point ($\mathbf{P}_{r+1}$) during an ongoing interpolation, without losing any of the mathematical properties, for both TPS and CR. This is useful in many real-time scenarios, an example might be a robot which encounters a previously unknown obstacle that it needs to circumvent.
The basic idea is to stop the ongoing interpolation, and start a new one in the current point $\mathbf{P}$, which thus becomes $\mathbf{P}_{r,new}$. The derivative in this point $\mathbf{P}' = \mathbf{P}'_{r,new}$ is calculated using the current interpolation interval, and a new new destination point $\mathbf{P}_{r+1,new}$ is chosen. The derivative of the recursive formulation of TPS in point $v = 0$ is:

$$\mathbf{F}'_{TPS,rec(0)} = \frac{1}{||\mathbf{D}_{0,r-1}||^{\beta_0}} \mathbf{D}_{0,r-1} \tag{12}$$

Thus, setting $\mathbf{D}_{0,r-1} = \mathbf{P}'$ and $\beta_0 = 0$ for the new interpolation segment maintains C^1 continuity. (Alternatively, if β_0 is chosen freely the generated curve will be G^1 continuous in point $\mathbf{P}$.)
In a similar way the next control-point can be changed during an ongoing interpolation using recursive CR. Equation 9 describes the derivative for CR and $v = 0$. We set again the β_0 for segment $\mathbf{D}_{r,r-1}$ to zero, as well as equation 9 $\mathbf{F}'_{CR,rec(0)} = \mathbf{P}'$, and solve for $\mathbf{D}_{r,r-1}$:

$$\mathbf{D}_{r,r-1} = \frac{(1+s_r)\mathbf{P}' - \frac{\mathbf{D}_{r+1,r}}{s_r}}{s_r} \tag{13}$$

Note that in contrast to TPS, in CR setting β_0 to another value but zero will not lead to G^1 continuity, but will rather have no higher continuity than C^0.
Setting $\beta_0 = 0$ however means using an uniform parameterization, albeit for only one segment, which might lead to unwanted behavior (especially loops in the case of CR). However, although no proof for the existence or non-existence of such unwanted behavior can be presented, none were found during the experiments with neither CR nor TPS (i.e. no large overshoots either). An example of this method can be found in figure 6.
Of course, if the necessity to change the control-points becomes known beforehand (i.e. before entering the segment containing an afflicted control-point), no measures need to be taken in case of TPS. For CR, this is only the case if both the current and the next segment are not affected.

5 CONCLUSION / FUTURE WORK

Three-Point-Splines (TPS) have been introduced which have similar mathematical characteristics as Catmull-Rom (CR). However, in contrast to CR and similar splines TPS do not rely on the knowledge of future points. Thus TPS enable spline interpolation even in real-time scenarios where no future values are known. Also, it was shown that TPS can be computed faster than CR.
TPS are more rigid than CR and because of that are trivially free of self-intersection (inside the interpolation segment) without the need for a particular parameterization to avoid this. Thus the parameterization of TPS can be adapted to the application, but ideal values for different applications still need to be found. For CR, only one parameterization, centripetal, fulfills this property. However, centripetal CR is also free from self-intersections in directly consecutive segments, a property that cannot be introduced to TPS due to the fact that they do not include knowledge of future points.
Also, a method was presented how to change the next control-point during an ongoing interpolation, for both TPS and CR, rather than having to do a computational expensive replanning. However, although no such behavior was observed in the experiments, it is not completely clear yet if this can lead to unwanted behavior like self-intersections (in the case of CR) or overshots (in the case of TPS).

ISBN 978-80-86943-37-4

6 ACKNOWLEDGMENTS

My special gratitude to Jan-Åke Larsson, Per-Erik Forssén, Ingmar Ragnemalm and Robert Forchheimer for their valuable help in writing this paper.

7 APPENDIX - PROOFS AND DERIVATIONS

In the following excessive use is made of the fact that $||\mathbf{P}|| = \sqrt{\mathbf{PP}}$ (note that since this describes a norm always the positive root has to be selected), and especially $(\frac{\mathbf{P}}{||\mathbf{P}||})^2 = \frac{\mathbf{PP}}{(\sqrt{\mathbf{PP}})^2} = 1$.

7.1 Properties of the Uniform Version

7.1.1 Formulation and Derivative

Separating equation 2 in two terms, one blending $\mathbf{D}_{r,r-1}$ and one blending $\mathbf{D}_{r+1,r}$ yields (with $0 <= u <= 1$):

$$\begin{aligned}\mathbf{F}_{CR(u)} &= \alpha u(u-1)^2\mathbf{D}_{r,r-1} \\ &+ u^2((\alpha-2)u+(3-\alpha))\mathbf{D}_{r+1,r}+\mathbf{P}_r \\ &= \alpha a_{(u)}\mathbf{D}_{r,r-1}+b_{(u)}\mathbf{D}_{r+1,r}+\mathbf{P}_r\end{aligned}$$

With the derivative:

$$\begin{aligned}\mathbf{F}'_{CR(u)} &= \alpha(3u^2-4u+1)\mathbf{D}_{r,r-1} \\ &+ (3(\alpha-2)u^2+2(3-\alpha)u)\mathbf{D}_{r+1,r} \quad (14) \\ &= \alpha a'_{(u)}\mathbf{D}_{r,r-1}+b'_{(u)}\mathbf{D}_{r+1,r}\end{aligned}$$

7.1.2 Distance to $l_{(t)}$

Inserting $\mathbf{F}_{CR(u)}$ into the point-line-distance function becomes:

$$\begin{aligned}d_{(u)} &= ||(\mathbf{F}_{CR(u)}-\mathbf{P}_r)-((\mathbf{F}_{CR(u)}-\mathbf{P}_r)\mathbf{D}_{r+1,r})\mathbf{D}_{r+1,r}|| \\ &= || \quad (\alpha a_{(u)}\mathbf{D}_{r,r-1}+b_{(u)}\mathbf{D}_{r+1,r}) \\ &\quad - ((\alpha a_{(u)}\mathbf{D}_{r,r-1}+b_{(u)}\mathbf{D}_{r+1,r})\tfrac{\mathbf{D}_{r+1,r}}{||\mathbf{D}_{r+1,r}||}) \\ &\quad \cdot \tfrac{\mathbf{D}_{r+1,r}}{||\mathbf{D}_{r+1,r}||}||\end{aligned}$$

Simplifying this equation leads to:

$$\begin{aligned}d_{(u)} &= || \quad \alpha a_{(u)}\mathbf{D}_{r,r-1}+b_{(u)}\mathbf{D}_{r+1,r} \\ &\quad - \alpha a_{(u)}\tfrac{\mathbf{D}_{r,r-1}\cdot\mathbf{D}_{r+1,r}}{\mathbf{D}_{r+1,r}\cdot\mathbf{D}_{r+1,r}}\mathbf{D}_{r+1,r} \\ &\quad - b_{(u)}\tfrac{\mathbf{D}_{r+1,r}\cdot\mathbf{D}_{r+1,r}}{||\mathbf{D}_{r+1,r}||\,||\mathbf{D}_{r+1,r}||}\mathbf{D}_{r+1,r}|| \\ &= \alpha a_{(u)}||\mathbf{D}_{r,r-1}-\tfrac{\mathbf{D}_{r,r-1}\cdot\mathbf{D}_{r+1,r}}{||\mathbf{D}_{r+1,r}||\,||\mathbf{D}_{r+1,r}||}\mathbf{D}_{r+1,r}|| \\ &= ||\mathbf{D}_{r,r-1}||\alpha a_{(u)} \\ &\quad \cdot ||\tfrac{\mathbf{D}_{r,r-1}}{||\mathbf{D}_{r,r-1}||}-\tfrac{\mathbf{D}_{r+1,r}\cdot\mathbf{D}_{r,r-1}}{||\mathbf{D}_{r+1,r}||\,||\mathbf{D}_{r,r-1}||}\tfrac{\mathbf{D}_{r+1,r}}{||\mathbf{D}_{r+1,r}||}|| \\ &= ||\mathbf{D}_{r,r-1}||\alpha a_{(u)}\,(\tfrac{\mathbf{D}_{r,r-1}\cdot\mathbf{D}_{r,r-1}}{||\mathbf{D}_{r,r-1}||\,||\mathbf{D}_{r,r-1}||} \\ &\quad - 2\tfrac{\mathbf{D}_{r+1,r}\cdot\mathbf{D}_{r,r-1}}{||\mathbf{D}_{r+1,r}||\,||\mathbf{D}_{r,r-1}||}\tfrac{\mathbf{D_{r+1,r}}}{||\mathbf{D_{r+1,r}}||}\tfrac{\mathbf{D_{r,r-1}}}{||\mathbf{D_{r,r-1}}||} \\ &\quad + (\tfrac{\mathbf{D}_{r+1,r}\cdot\mathbf{D}_{r,r-1}}{||\mathbf{D}_{r+1,r}||\,||\mathbf{D}_{r,r-1}||})^2\tfrac{\mathbf{D}_{r+1,r}\cdot\mathbf{D}_{r+1,r}}{||\mathbf{D}_{r+1,r}||\,||\mathbf{D}_{r+1,r}||})^{\frac{1}{2}} \\ &= ||\mathbf{D}_{r,r-1}||\alpha a_{(u)}(1-\cos^2\theta)^{\frac{1}{2}}\end{aligned}$$

with θ the angle between $\mathbf{D}_{r,r-1}$ and $\mathbf{D}_{r+1,r}$ (4)

7.2 Properties of the Recursive Version

7.2.1 Formulation and Derivative

From figure 4 the equation for the recursive version of TPS can be derived, for $0 \le v \le s_r = ||\mathbf{D}_{r+1,r}||^\beta$:

$$\begin{aligned}\mathbf{F}_{TPS,rec(v)} &= -\tfrac{v}{s_{r-1}}(1-\tfrac{v}{s_r})^2\mathbf{P}_{r-1}+\tfrac{v}{s_{r-1}}(1-\tfrac{v}{s_r})^2\mathbf{P}_r \\ &\quad + (1-\tfrac{v}{s_r})\mathbf{P}_r+\tfrac{v}{s_r}\mathbf{P}_r \\ &\quad - \tfrac{v}{s_r}(2\tfrac{v}{s_r}-\tfrac{v^2}{s_r^2})\mathbf{P}_r+\tfrac{v}{s_r}(2\tfrac{v}{s_r}-\tfrac{v^2}{s_r^2})\mathbf{P}_{r+1} \\ &= \tfrac{v}{s_{r-1}}(1-\tfrac{v}{s_r})^2(\mathbf{D}_{r,r-1}) \\ &\quad + \tfrac{v}{s_r}(2\tfrac{v}{s_r}-\tfrac{v^2}{s_r^2})(\mathbf{D}_{r+1,r})+\mathbf{P}_r \\ &= s_r((\tfrac{v}{s_r}-2(\tfrac{v}{s_r})^2+(\tfrac{v}{s_r})^3)\tfrac{\mathbf{D}_{r,r-1}}{s_{r-1}} \\ &\quad + (2(\tfrac{v}{s_r})^2-(\tfrac{v}{s_r})^3)\tfrac{\mathbf{D}_{r+1,r}}{s_r}+\mathbf{P}_r\end{aligned}$$

And from that the derivative:

$$\begin{aligned}\mathbf{F}'_{TPS,rec(v)} &= (1-4\tfrac{v}{s_r}+3(\tfrac{v}{s_r})^2)\tfrac{\mathbf{D}_{r,r-1}}{s_{r-1}} \\ &\quad + (4\tfrac{v}{s_r}-3(\tfrac{v}{s_r})^2)\tfrac{\mathbf{D}_{r+1,r}}{s_r} \\ &= (4\tfrac{v}{s_r}-3(\tfrac{v}{s_r})^2)(\tfrac{\mathbf{D}_{r+1,r}}{s_r}-\tfrac{\mathbf{D}_{r,r-1}}{s_{r-1}}) \\ &\quad + \tfrac{\mathbf{D}_{r,r-1}}{s_{r-1}}\end{aligned}$$

7.2.2 Distance to $l_{(t)}$

Inserting $\mathbf{F}_{TPS,rec(v)}$ in equation 4 (with $a_{(v)} = \frac{s_r}{s_{r-1}}(\frac{v}{s_r}(\frac{v}{s_r}-1)^2)$) leads to:

ISBN 978-80-86943-37-4

$$d_{(v)} = ||\mathbf{D}_{r,r-1}|| \frac{||\mathbf{D}_{r+1,r}||^{\beta}}{||\mathbf{D}_{r,r-1}||^{\beta}} (\frac{v}{s_r}(\frac{v}{s_r}-1)^2)(1-\cos^2\theta)^{\frac{1}{2}}$$
$$= ||\mathbf{D}_{r+1,r}||^{\beta} ||\mathbf{D}_{r,r-1}||^{1-\beta} a'_{(v)} (1-\cos^2\theta)^{\frac{1}{2}}$$
with $0 \leq a'_{(v)} = \frac{v}{s_r}(\frac{v}{s_r}-1)^2 \leq 1$

7.3 Equation and Derivative of recursive Catmull-Rom

The following is needed to be able to change the control-points during an ongoing interpolation as described in section 4. Again we use $0 \leq v \leq s_r = ||\mathbf{D}_{r+1,r}||^{\beta}$.

7.3.1 Formulation

$$\begin{aligned}\mathbf{F}_{CR(v)} = & \frac{s_1^2 v - 2 s_r v^2 + v^3}{s_{r-1} s_r (s_{r-1}+s_r)} \mathbf{D}_{r,r-1} \\ & + (\frac{s_{r-1} s_r v + 2 s_r v^2 - v^3}{s_r s_r (s_{r-1}+s_r)} + \frac{s_r v^2 - v^3}{s_r s_r (s_r + s_{r+1})}) \mathbf{D}_{r+1,r} \\ & + \frac{v^3 - s_r v^2}{s_r s_{r+1} (s_r + s_{r+1})} \mathbf{D}_{r+2,r+1} \\ & + \mathbf{P}_r\end{aligned}$$

7.3.2 First Derivative

$$\begin{aligned}\mathbf{F'}_{CR(v)} = & \frac{s_r^2 - 4 s_r v + 3 v^2}{s_{r-1} s_r (s_{r-1}+s_r)} \mathbf{D}_{r,r-1} \\ & + (\frac{s_{r-1} s_r + 4 s_r v - 3 v^2}{s_r s_r (s_{r-1}+s_r)} + \frac{2 s_r v - 3 v^2}{s_r s_r (s_r + s_{r+1})}) \mathbf{D}_{r+1,r} \\ & + \frac{3 v^2 - s_r 2 v}{s_r s_{r+1} (s_r + s_{r+1})} \mathbf{D}_{r+2,r+1}\end{aligned}$$

REFERENCES

[CR74] E. Catmull and R. Rom. A class of local interpolating splines. In *Computer Aided Geometric Design*, 1974.

[CYK11] S. Schaefer C. Yuksel and J. Keyser. Parameterization and applications of catmull-rom curves. In *Computer-Aided Design*, volume 43, pages 747–755, 2011.

[DB88] T. DeRose and B.A. Barsky. Geometric continuity, shape parameters, and geometric constructions for catmull-rom splines. In *ACM Transactions on Graphics*, volume 7, 1988.

[DM96] F. Canny D. Manocha. Detecting cusps and inflection points in curves. In *Computer-Aided Geometric Design*, volume 12, 1996.

[Dod97] N.A. Dodgson. Quadratic interpolation for image resampling. In *IEEE Transactions on Image Processing*, volume 6, 1997.

[ET14] P. Englert and M. Toussaint. Reactive phase and task space adaption for robust motion execution. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014.

[Flo08] M.S. Floater. On the deviation of a parametric cubic spline interpolant from its data polygon. In *Computer-Aided Geometric Design*, volume 25, 2008.

[IDS99] I. Guskov I. Daubechies and W. Sweldens. Regularity of irregular subdivision. In *Constructive Approximation*, volume 15, 1999.

[JAW67] E.N. Nilson J.H. Ahlberg and J.L. Walsh. *The Theory of Splines and Their Applications*. Academic Press, 1967.

[KB84] D. H. U. Kochanek and R. H. Bartels. Interpolating splines with local tension, continuity, and bias control. In *ACM SIGGRAPH*, volume 18, 1984.

[Lee89] E.T.Y. Lee. Choosing nodes in parametric curve interpolation. In *Computer-Aided Geometric Design*, volume 21, 1989.

[ML88] R. Moore and J. Lopes. A recursive evaluation algorithm for a class of catmull-rom splines. In *Proceedings of the 15th annual conference on Computer graphics and interactive techniques (SIGGRAPH)*, volume 22, 1988.

[NDH09] M.S. Floater N. Dyn and K. Hormann. Four-point curve subdivision based on iterated chrordal and centripetal parameterization. In *Computer-Aided Geometric Design*, volume 26, 2009.

[Ogn13] Jens Ogniewski. C1-continuous, low-complex splines using 3 control points. In *Motion in Games*, 2013.

[Yua96] J.T. Yuan. Asymmetric interpolation lattice. In *IEEE Transactions on Signal Processing*, volume 44, 1996.