Precise offsetting of quadratic Bézier curves

Fabian Yzerman

https://blend2d.com/

Based on a scientific article - Revision 1.1

January 18, 2020

Abstract

An iterative algorithm will be developed to generate an approximate offset path for planar quadratic Bézier curves within a specified maximum deviation. The offset path consists of one or multiple quadratic Bézier curves and retains G^1 continuity at all junctures. Finally, it will be compared against existing methods in terms of quality and performance.

1 Introduction

Parallel curves in 2D computer graphics are called offset curves. They are needed when drawing uniformly thick curves which is commonly referred to as stroking. Many different techniques already exist, two of which will be discussed in the following.
The first method is introduced in *Adaptive Subdivision of Bezier Curves* and originates from Anti-Grain Geometry (AGG), a C++ 2D graphics framework that has been broadly adopted for both open-source and commercial uses [1]. It works by recursively flattening the original curve to a polyline approximation. Then each point will be offset to both sides of the curve so that the output is, again, a polyline. One drawback of this approach is variable quality as it generates the same number of line segments for each offset path, i.e. usually the inner side has more line segments than needed while the outer has less. Moreover, the quality decreases with greater offset widths since no more line segments will be generated without a change of parameters.
Thomas F. Hain's *Fast, Precise Flattening of Cubic Bézier Segment Offset Curves* suggests computing an iterative circular approximation for each side separately [2].

Any offset path may have a different number of line segments and the quality criterion is closely met. However, in certain cases there are visible defects that are not handled. Another issue with both strategies is that they do not perform optimally under affine transformations. They are only able to anticipate the dimension with the biggest scale resulting in more line segments than needed.

In this paper we suggest using a spline of approximated Bézier curves in the offset paths which will retain more information about the curve's behavior, such as continuity, than a polyline. They are easily transformed afterwards and flattening could happen right before rasterisation so that the desired quality is met without having too many line segments, even after extreme transformations.
A similar approach has already been discussed in *Quadratic bezier offsetting with selective subdivision* by Gabriel Suchowolski, yet the author does not specify an error bound [3]. We will estimate this approximation error and introduce an iterative method to offset quadratic Bézier curves within a given quality parameter.

2 Definitions

A Bézier curve of order n is defined by

$$C_n(t) = \sum_{i=0}^{n} \binom{n}{i} t^i (1-t)^{n-i} P_i$$

with $0 \leq t \leq 1$. From now on we simply refer to Bézier curves as curves. So for quadratic and cubic curves it gives

$$C_2(t) = (1-t)^2 P_0 + 2t(1-t)P_1 + t^2 P_2$$
$$C_3(t) = (1-t)^3 P_0 + 3t(1-t)^2 P_1 + 3t^2(1-t)P_2 + t^3 P_3$$

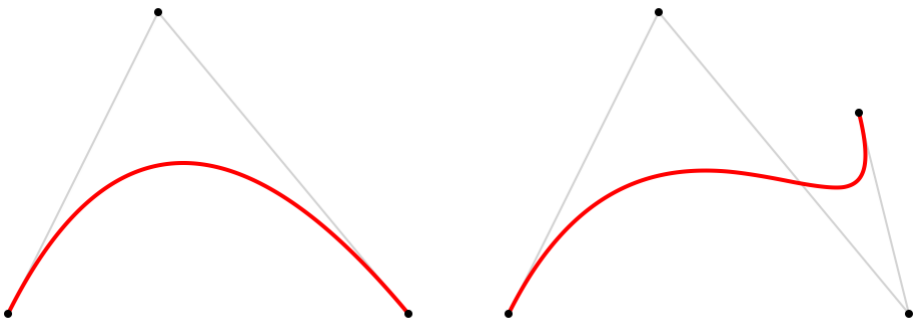


Figure 1: Quadratic curve (left) and cubic curve (right)

We can also write them in polynomial canonical form:

$$C_2(t) = (P_2 - 2P_1 + P_0)t^2 + (2P_1 - 2P_0)t + P_0$$
$$C_3(t) = (P_3 - 3P_2 + 3P_1 - P_0)t^3 + (3P_2 - 6P_1 + 3P_0)t^2 + (3P_1 - 3P_0)t + P_0$$

An offset curve of $C(t)$ is described as $\bar{C}(t) = C(t) + \bar{d}\vec{n}(t)$ where $\bar{d}$ is the offset distance and $\vec{n}(t)$ is the unit normal of the derivative $C'(t)$ such that:

$$\vec{n}_x(t) = +\frac{C'_y(t)}{\sqrt{C'_x(t)^2 + C'_y(t)^2}} \qquad \vec{n}_y(t) = -\frac{C'_x(t)}{\sqrt{C'_x(t)^2 + C'_y(t)^2}} \tag{1}$$

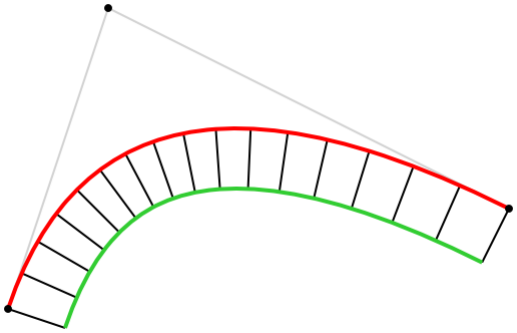


Figure 2: Quadratic curve (red) and offset curve (green)

The equations in (1) cannot be trivially expressed by another polynomial curve of the same order. Thus the offset curve has to be approximated instead.

3 Offsetting of quadratic Bézier curves

Let $C(t)$ be a quadratic curve. We want to approximate the left or right offset path for $C(t)$ by using one or multiple quadratic curves. The proposed algorithm will split $C(t)$ into a spline of curves which can be offset by translating the edges of their control polygon while staying within a specified maximum deviation from the actual offset.

3.1 Translating the control polygon

Let $\bar{C}(t)$ be an offset curve of $C(t)$. Then the curves are defined as:

$$C(t) = (P_2 - 2P_1 + P_0)t^2 + (2P_1 - 2P_0)t + P_0$$
$$\bar{C}(t) = (\bar{P}_2 - 2\bar{P}_1 + \bar{P}_0)t^2 + (2\bar{P}_1 - 2\bar{P}_0)t + \bar{P}_0$$

We want to calculate $\bar{C}(t)$ so that at least G^1 continuity is ensured at the endpoints $\bar{P}_0$ and $\bar{P}_2$. So the idea is to translate $\overline{P_1P_0}$ and $\overline{P_2P_1}$ perpendicular to their tangent yielding an approximated offset of $C(t)$. The actual order of continuity is close to G^2 because the curvatures of $C(t)$ and $\bar{C}(t)$ are approximately equal at their endpoints.

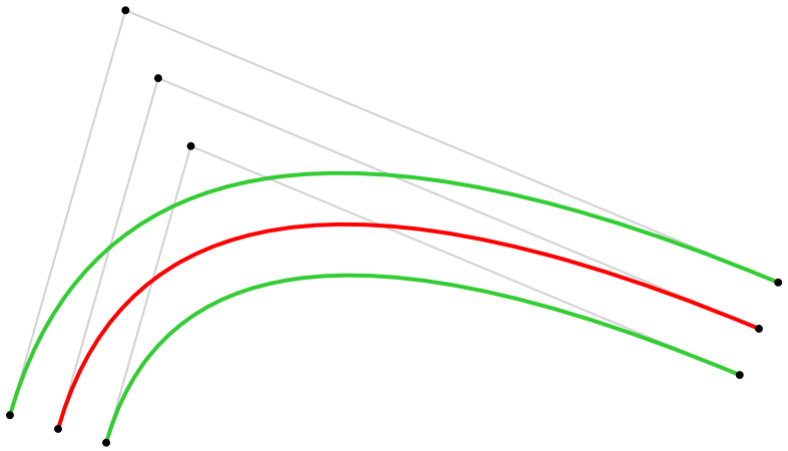


Figure 3: Offsetting of $\overline{P_1P_0}$ and $\overline{P_2P_1}$ to both sides

Thus the control point $\bar{P}_1$ is the intersection of those tangents at $t = \{0, 1\}$. With the offset distance $\bar{d}$ we define $\bar{P}_0$, $\bar{P}_1$ and $\bar{P}_2$ for left and right offsets by

$$\vec{n}_0 = \vec{n}(0) \qquad \vec{n}_1 = \vec{n}(1) \qquad \vec{n} = \vec{n}_0 + \vec{n}_1$$

$$\bar{P}_0 = P_0 \pm \bar{d}\vec{n}_0 \qquad \bar{P}_2 = P_2 \pm \bar{d}\vec{n}_1 \qquad \bar{P}_1 = P_1 \pm \frac{2\bar{d}\vec{n}}{\vec{n} \cdot \vec{n}}$$

Remark. The denominator of the translation vector in $\bar{P}_1$ is a dot product of $\vec{n}$. The full derivation is explained in *Quadratic bezier offsetting with selective subdivision* [3].

3.2 Approximation error

The approximation of $\bar{C}(t)$ introduces an error which depends on the curvature of $C(t)$. We define the distance function of the two curves as

$$d(t) = \|\bar{C}(t) - C(t)\|$$

and the absolute approximation error:

$$\epsilon(t) = |\bar{d} - d(t)|$$

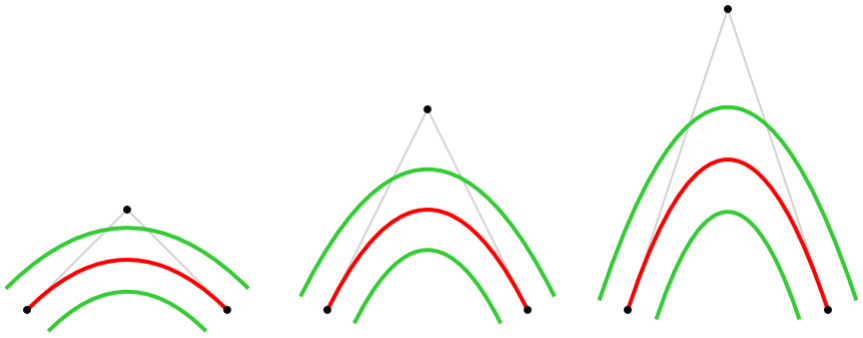


Figure 4: Deviation of offset curves calculated from a single curve

It should be mentioned that $d(t)$ differs from the actual distance $d(t_1, t_2) = \|\bar{C}(t_2) - C(t_1)\|$ where t_1 and t_2 is chosen as such the normal of the slope at $C(t_1)$ will intersect at $\bar{C}(t_2)$. This difference happens because $\bar{C}(t)$ is an approximation of the actual offset and thus the parameter t does not run evenly through both curves. However, $d(t)$ has some interesting properties that will help specifying the approximation error.

Theorem. *The distance function $d(t) = \|\bar{C}(t) - C(t)\|$ of a non-degenerate curve $C(t)$ and its offset curve $\bar{C}(t)$ has the following properties:*

i) The maximum is at $t_{max} = \frac{1}{2}$ and the two minima are at $t = 0$ and $t = 1$.

ii) If the distance $\bar{d}$ and the angle φ between $\overline{P_1P_0}$ and $\overline{P_2P_1}$ of $C(t)$ are constant then the value of $d(t_{max})$ will also be constant.

Proof.

i) We parameterize the curve $C(t)$, along with its offset curve $\bar{C}(t)$, so that we would be able to generate all possible curves with the help of affine transformations:

$$P_0 = \begin{pmatrix} -1 \\ 0 \end{pmatrix} \qquad P_1 = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \qquad P_2 = \begin{pmatrix} P_{2x} \\ P_{2y} \end{pmatrix}$$

Then solving $d'(t) = 0$ yields $t = \{0, \frac{1}{2}, 1\}$ with arbitrary $\bar{d}$ and P_2. We take a look at the second derivative to specify the extrema:

$$d''(0) = d''(1) = \frac{2|\bar{d}|{P_{2y}}^2({P_{2x}}^2 + {P_{2y}}^2 - P_{2x}\sqrt{{P_{2x}}^2 + {P_{2y}}^2})}{({P_{2y}}^2 + P_{2x}(P_{2x} + \sqrt{{P_{2x}}^2 + {P_{2y}}^2}))^2}$$

$$\begin{cases} > 0, & \text{if } P_{2y} \neq 0 \text{ and } \bar{d} \neq 0 \\ = 0, & \text{otherwise} \end{cases}$$

Similarly $d''(\frac{1}{2}) < 0$ if $P_{2y} \neq 0$ and $\bar{d} \neq 0$. So we have got the maximum at $t_{max} = \frac{1}{2}$ and two minima at $t = \{0, 1\}$.

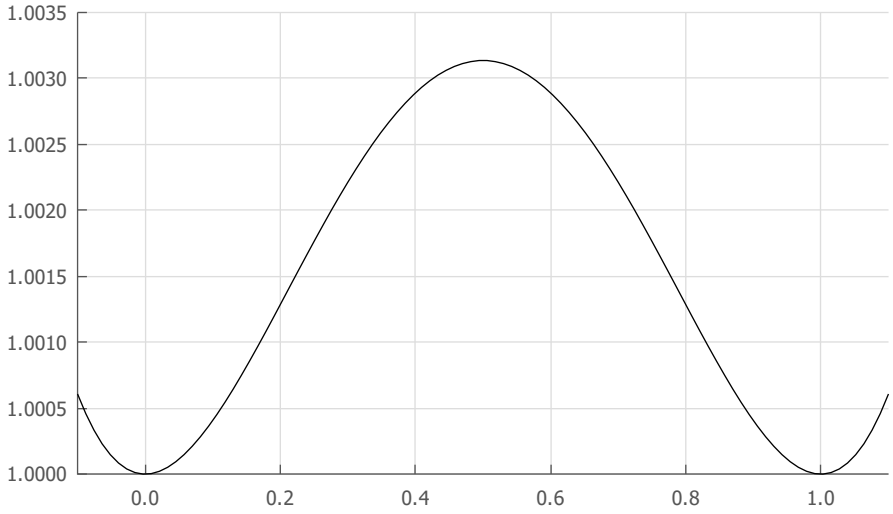


Figure 5: Graph of $d(t)$ with $\bar{d} = p_{2x} = p_{2y} = 1$

Remark. If $P_{2y} = 0$ or $\bar{d} = 0$ there are no extrema due to $d'(t) = 0$ such that $d(t) = |\bar{d}|$. With $P_{2y} = 0$ and $P_{2x} \geq 0$ the curve $C(t)$ is basically a line. However, if $P_{2y} = 0$ and $P_{2x} < 0$ we get a degenerate curve. The latter is a special case and has to be handled separately by replacing the curve with two lines.

ii) Now we parameterize the curves such that

$$P_0 = \begin{pmatrix} -1 \\ 0 \end{pmatrix} \qquad P_1 = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \qquad P_2 = \begin{pmatrix} r\cos(\varphi) \\ r\sin(\varphi) \end{pmatrix}$$

and look at the function $d_{max}(\varphi) = d(t_{max}, \varphi)$ for $0 \leq \varphi < \pi$:

$$d_{max}(\varphi) = \frac{\bar{d}}{4}(3 + \cos(\varphi)) \sec\left(\frac{\varphi}{2}\right) \tag{2}$$

It is obvious that $d_{max}(\varphi) \sim \bar{d}$ so the maximum distance grows linearly with $\bar{d}$. Additionally r vanishes from the equation. This shows that if φ and $\bar{d}$ are constant, $d(t_{max})$ will be constant, too.

□

We have confirmed that $d(t_{max}) \geq d(t)$ for $0 \leq t \leq 1$. Furthermore, $d_{max}(\varphi) \geq \bar{d}$ holds true for $0 \leq \varphi < \pi$ which means that the maximum possible excursion of $\bar{C}(t)$ always lies outside the theoretical offset curve.

Therefore the maximum approximation error is given:

$$\epsilon_{max} = |\bar{d} - d_{max}| \qquad \text{(absolute error)}$$

$$\eta_{max} = |1 - \frac{d_{max}}{\bar{d}}| \qquad \text{(relative error)}$$

Since r vanished from equation (2) and $d_{max}(\varphi) \sim \bar{d}$ we can define the function of the maximum relative error in dependency of only φ as

$$\eta_{max}(\varphi) = 1 - \frac{1}{4}(3 + \cos(\varphi)) \sec\left(\frac{\varphi}{2}\right)$$

or alternatively:

$$\eta_{max}(\varphi) = 2\sec\left(\frac{\varphi}{2}\right)\sin^4\left(\frac{\varphi}{4}\right) = \frac{2\sin^4\left(\frac{\varphi}{4}\right)}{\cos\left(\frac{\varphi}{2}\right)} \tag{3}$$

In the plot of $\eta_{max}(\varphi)$ the maximum relative error starts at zero and grows with φ.

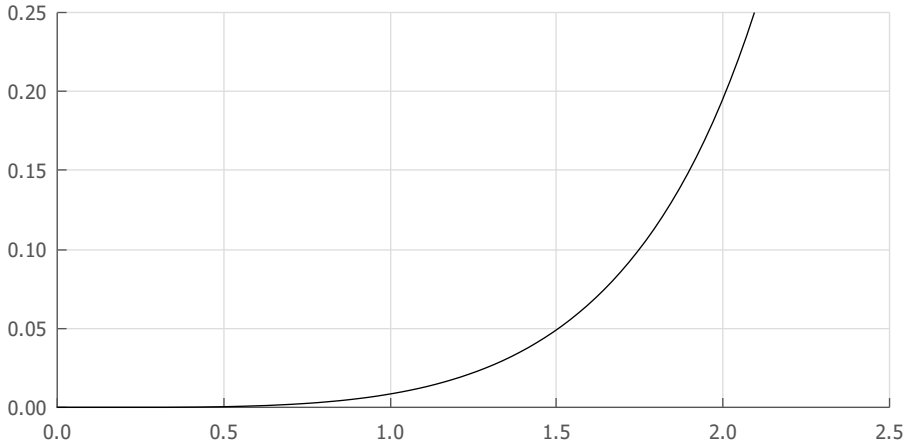


Figure 6: Graph of $\eta_{max}(\varphi)$

Remark. Mike Kamermans showed that by approximating circles with quadratic curves the maximum error depends on a specific angle [4]. Interestingly, this is the same angle from equation (2). He parameterized symmetric curves so that they have its maximum deviation at $t = \frac{1}{2}$. The error distance $\epsilon(\varphi)$ of the unit circle is described as

$$\epsilon(\varphi) = 2\sin^4\left(\frac{\varphi}{4}\right)\sqrt{\frac{1}{\cos^2\left(\frac{\varphi}{2}\right)}}$$

which is basically the same as equation (3).
He also solved the function for φ to calculate the angle within a given error:

$$\varphi(\epsilon) = 4\arccos\left(\frac{\sqrt{2 + \epsilon - \sqrt{\epsilon(2+\epsilon)}}}{\sqrt{2}}\right)$$

3.3 Split curve by angle

With knowledge of equation (3) we are able to restrict the maximum approximation error for arbitrary quadratic curve segments by splitting $C(t)$ at the angle φ.
For convenience the curve components are written as $C'(t) = at + b$ so that a and b are the associated coefficients of the derived polynomial form. To split a curve at the parameter t_s where the angle condition is satisfied, we take the slope

$$m_0 = \frac{C'_y(0)}{C'_x(0)} = \frac{b_y}{b_x}$$

at $t = 0$ and the slope

$$m_t = \frac{C'_y(t_s)}{C'_x(t_s)} = \frac{a_y t_s + b_y}{a_x t_s + b_x}$$

at t_s. Thus the angle φ between m_0 and m_t is

$$\tan\varphi = m = \left| \frac{m_0 - m_t}{1 + m_0 m_t} \right|$$

If we plug in m_0 and m_t, then solve for t_s it gives

$$t_s = \frac{m(b_x b_x + b_y b_y)}{|a_x b_y - a_y b_x| - m(a_x b_x + a_y b_y)}$$

We split the curve at the parameter t_s while ensuring the maximum deviation with the angle φ. The rest of the remaining curve can be processed iteratively as long as $0 < t_s < 1$. The final segment is reached when $t_s \geq 1$ so that further splitting is not needed.

3.4 Handling cusps

If an offset curve is generated and its local curvature radius falls below the offset width a cusp will appear on the inner path. This usually happens for curves with sharp turns and there may be critical points where they are not differentiable because direction vector suddenly reverses. Quadratic curves are not able to represent that kind of shape.

Figure 7: Cusps appearing in offset curves

These specific cases have to be handled separately because the splitting method that was introduced in section 3.3 does not take them into account.
We are able to calculate the critical points by equating the curvature radius $R(t)$ of the planar curve $C(t)$ with the offset distance $\bar{d}$.

$$R(t) = \frac{1}{\kappa(t)} = \frac{(C'_x(t)^2 + C'_y(t)^2)^{3/2}}{C'_x(t)C''_y(t) - C'_y(t)C''_x(t)} \stackrel{!}{=} \bar{d}$$

For quadratic curves we are able to solve this equation for t with a and b being the coefficients of $C'(t) = at + b$.

$$t_{1,2} = \frac{-(a_x b_x + a_y b_y) \pm \sqrt{(a_x b_x + a_y b_y)^2 - (a_x^2 + a_y^2)(b_x^2 + b_y^2 - \sqrt[3]{\bar{d}^2(b_x a_y - a_x b_y)^2})}}{a_x^2 + a_y^2}$$

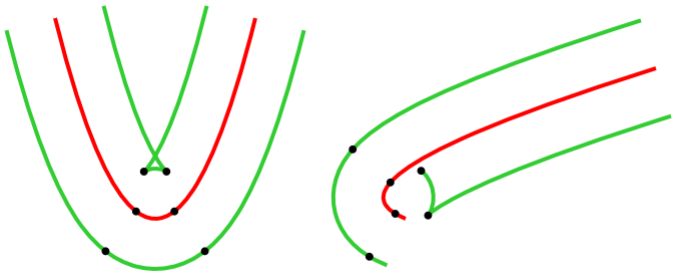


Figure 8: Critical points of $C(t)$ and $\bar{C}(t)$

t_1 and t_2 are both locations of the critical points. So if we conditionally split the curves at $0 < t_1 < 1$ and $0 < t_2 < 1$ the cusps will appear at $C(t_{1,2}) + \bar{d}\vec{n}(t_{1,2})$ for that specific distance. The offset curves are more precise because the cusps will emerge at the splitting point of the curve and not between its interval.

3.5 Algorithm

Finally, all parts of the algorithm will be put together and presented in the following pseudocode. It requires an input curve, the offset distance and the angle condition to generate the output path which will consist of one or multiple quadratic curves.

procedure OffsetQuadraticCurve
  Set c to input curve
  Set offset distance $\bar{d}$
  Set angle condition m
  Calculate parameters t_1 and t_2 where curvature radius equals $\bar{d}$
  if $0 < t_1 < 1$ **then**
    Split c at t_1
  end if
  if $0 < t_2 < 1$ **then**
    Split c at t_2
  end if
  for all curve segments of c **do**
    Set cc to current curve segment
    loop
      Set t_s to parameter where angle condition m of cc is met
      if $0 < t_s < 1$ **then**
        Split cc at t_s into cc_1 and cc_2
        Translate control polygon of cc_1 by $\bar{d}$ and add to output path
        Set cc to cc_2
      else
        Break loop
      end if
    end loop
    Translate control polygon of cc by $\bar{d}$ and add to output path
  end for
end procedure

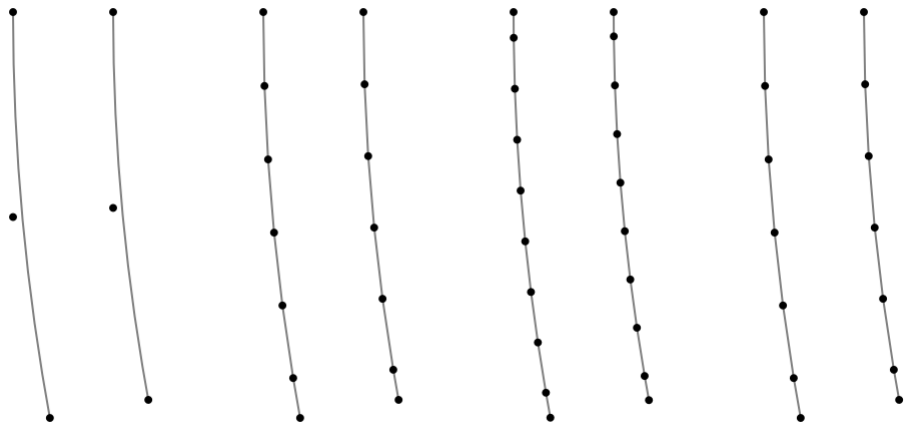
4 Quality comparison

In the following section we take a comparative look at several approaches, stroking a single quadratic curve to both sides at half offset width $\bar{d} = \frac{\bar{w}}{2}$ each. We desire to meet

the quality criterion closely while generating few vertices.
The first method to compare will be the new one keeping the generated offset path as curves. The second one will also be the new approach but the curves are flattened to illustrate the final output and to make the visual comparison easier. For that the iterative flattening technique of Thomas Hain's *Fast, Precise Flattening of Cubic Bézier Segment Offset Curves* (adapted for quadratic curves) is used [5]. The third one is the method from AGG and the last one an adaption of Hain's offset algorithm that works with quadratic curves. To obtain the relative approximation error for *New* and *NewFlatten* we choose $\varphi = 22.5$ degrees so that $\eta_{max}(\varphi) \approx 0.000188$ (i.e. 0.188 pixel for $\bar{d} = 1000$). The absolute flattening error for *Hain* and *NewFlatten* will be 0.15 pixel. *AGG* will use the default parameter as proposed by the author [1].
For every case the vertex counts of both offsets path will be compared. We estimate that *Hain* meets the quality criterion closest because in the original (cubic) version 94% of all vertices fall within 20% of the quality condition [2]. Thus we will have some kind of guideline to assess the number of vertices from *NewFlattened* and *AGG*.

4.1 Slightly curved

First we parameterize the curve so that *New* generates a single offset curve segment.



New	**NewFlatten**	**AGG**	**Hain**
$3 + 3$ vertices	$7 + 7$ vertices	$10 + 10$ vertices	$7 + 7$ vertices

Both *NewFlatten* and *Hain* achieve optimal results by meeting the quality condition closely. We notice that the vertices of those two look identical which arises from using a very similar method of iterative flattening. *AGG* is not able to meet the quality

criterion as closely because it uses a recursive approach for splitting and thus needs more vertices than necessary to offset the curve.

4.2 Bent

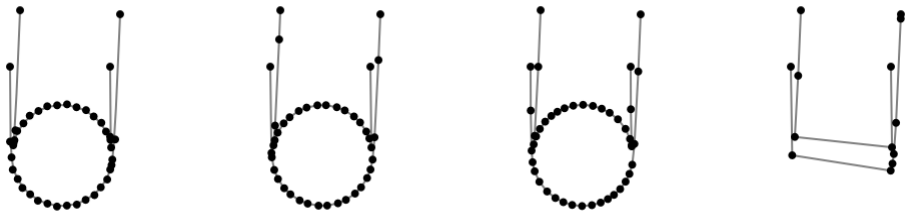
The next curve is close to a right angle which results in 4 offset curve segments for *New*.

New	**NewFlatten**	**AGG**	**Hain**
$9 + 9$ vertices	$17 + 14$ vertices	$18 + 18$ vertices	$16 + 13$ vertices

In this case *Hain* achieves the best result of the flattened techniques. There is one vertex less in each path than *NewFlatten*. Furthermore, we notice that the vertex distribution of *Hain* is smooth while *NewFlatten* has a few irregular spots due to the splitting. The quality of *AGG* is appropriate but, again, both paths have more vertices than needed.

4.3 Sharp turn

The last curve produces a circle and narrow cusps around the turning point.



New	**NewFlatten**	**AGG**	**Hain**
$21 + 21$ vertices	$21 + 20$ vertices	$22 + 22$ vertices	$(8) + (5)$ vertices

New produces 10 quadratic curve segments, 8 due to the approximation of almost 180 degrees and another 2 because of the cusp splits. Yet with many quadratic curve segments *NewFlatten* manages to have slightly less vertices than *AGG*. *Hain* cannot contribute a meaningful result because the algorithm fails with handling the cusps and thus is not stable enough for arbitrary input curves.

5 Run-time performance

Finally, we will test the run-time performance with different parameters. Time is measured from generating both offset paths of a single curve. For this we will denote the half offset width $\bar{d} = \frac{\bar{w}}{2}$. The tests were done with BenchmarkDotNet (.NET Core 2.1) and run on an AMD Ryzen 2600X CPU under Windows 10 (1803).

5.1 Angle

In the first test we will compare the timings depending on the angle φ in degrees and parameterize curves such that

$$P_0 = \begin{pmatrix} -50 \\ 0 \end{pmatrix} \qquad P_1 = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \qquad P_2 = \begin{pmatrix} 100\cos(\varphi) \\ 100\sin(\varphi) \end{pmatrix}$$

with a constant offset width $\bar{w} = 1$.

Method	**Angle**	**Mean**	**Error**	**StdDev**
New	20	141.4 ns	0.2746 ns	0.2435 ns
NewFlatten	20	604.5 ns	2.8405 ns	2.6570 ns
AGG	20	657.1 ns	1.1076 ns	1.0360 ns
Hain	20	632.3 ns	1.3111 ns	1.0948 ns
New	60	270.6 ns	0.2831 ns	0.2364 ns
NewFlatten	60	1039.7 ns	1.3011 ns	1.2171 ns
AGG	60	955.6 ns	4.5149 ns	4.0023 ns
Hain	60	987.0 ns	1.9502 ns	1.7288 ns
New	100	422.1 ns	1.3179 ns	1.1682 ns
NewFlatten	100	1372.9 ns	2.6274 ns	2.4577 ns
AGG	100	1132.6 ns	3.8131 ns	3.3802 ns
Hain	100	1157.1 ns	0.8279 ns	0.7339 ns

We notice that the performance costs rise with increasing angles. Although *NewFlatten*

has a little advantage at $\varphi = 20$ the overhead of splitting the curve quickly adds up and it falls behind at $\varphi = 100$. The timings of *AGG* and *Hain* are closely together.

5.2 Offset width

Now the performance impact of the offset width $\bar{w}$ is measured. The parameterization is similar to the previous test:

$$P_0 = \begin{pmatrix} -50 \\ 0 \end{pmatrix} \qquad P_1 = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \qquad P_2 = \begin{pmatrix} 100\cos(20°) \\ 100\sin(20°) \end{pmatrix}$$

With such a small angle we ensure that *Hain* will not fail because of emerging cusps.

Method	**Width**	**Mean**	**Error**	**StdDev**
New	1	141.1 ns	0.1054 ns	0.0934 ns
NewFlatten	1	607.8 ns	3.1337 ns	2.6168 ns
AGG	1	636.0 ns	2.7962 ns	2.6156 ns
Hain	1	626.8 ns	0.2242 ns	0.2097 ns
New	25	140.8 ns	0.0914 ns	0.0855 ns
NewFlatten	25	600.8 ns	2.9803 ns	2.4887 ns
AGG	25	634.0 ns	1.5795 ns	1.3190 ns
Hain	25	628.9 ns	2.9617 ns	2.7704 ns
New	50	141.4 ns	0.3959 ns	0.3703 ns
NewFlatten	50	605.6 ns	2.6276 ns	2.4579 ns
AGG	50	635.5 ns	2.7409 ns	2.4297 ns
Hain	50	626.1 ns	0.5769 ns	0.4817 ns

The timings show that the performance hardly scales with different widths. Theoretically, AGG is never affected by width at all and *New* only will if a cusp induces further splits which is not the case here. *NewFlatten* and *Hain* will have variable execution times if the arc length of the generated paths differ. Although the outer path gets longer with increasing offset widths the inner path usually gets shorter.
If we compare the methods with flattening there is no distinct winner. It is however interesting to notice that flattening is the dominant part in terms of run-time and *New* is able to delay those costs. If we included transformations and clipping it would be possible to only flatten the parts that are needed by the viewport while also closely meeting the quality condition. This is not easily possible with other techniques and could result in an overall speedup for specific cases.

5.3 Size

Finally, we will compare the timings for randomly distributed curves with offset width $\bar{w} = 1$. All points of the input curve lie inside a square with the side length set to the size parameter. Although we are not able to guarantee that *Hain* will not break early, especially if the size is small, we may still take those figures as a guideline.

Method	**Size**	**Mean**	**Error**	**StdDev**
New	10	278.3 us	0.2710 us	0.2402 us
NewFlatten	10	515.6 us	0.3019 us	0.2677 us
AGG	10	881.2 us	0.8265 us	0.7327 us
Hain	10	124.4 us	0.1139 us	0.0951 us
New	100	254.9 us	0.6605 us	0.6178 us
NewFlatten	100	622.1 us	1.1586 us	1.0271 us
AGG	100	778.5 us	1.4239 us	1.1890 us
Hain	100	380.5 us	0.2225 us	0.2082 us
New	1000	244.1 us	0.4501 us	0.3990 us
NewFlatten	1000	1147.7 us	3.0755 us	2.8768 us
AGG	1000	1309.2 us	1.1140 us	0.9302 us
Hain	1000	1117.0 us	1.5324 us	1.3584 us

In general all timings of the methods with flattening get closer to each other for increasing size with *Hain* always being the fastest one. *NewFlatten* is faster than *AGG* for all sizes and gets really close to *Hain* for the $size = 1000$.

6 Conclusion

A new algorithm to calculate an offset path from a quadratic curve has been introduced. The approach works independently of resolution and affine transformations. Furthermore, it ensures G^1 continuity at all junctures and closely meets the specified quality criterion. The method will also be precise and robust if the offset path has cusps.

Although run-time performance with flattening is slightly slower than other techniques in some cases, it is much faster without flattening most of the time. It could prove especially useful if a path received additional transformations after the offsetting.

Bibliography

[1] Maxim Shemanarev. Adaptive Subdivision of Bezier Curves. `http://antigrain.com/research/adaptive_bezier/`, 2005.

[2] Thomas F. Hain, Sri Venkat R. Racherla, David D. Langan. Fast, Precise Flattening of Cubic Bezier Segment Offset Curves. *17th Brazilian Symposium on Computer Graphics and Image Processing*, pages 244–249, 2004.

[3] Gabriel Suchowolski. Quadratic bezier offsetting with selective subdivision, 2012.

[4] Mike Kamermans. A Primer on Bézier Curves. `https://pomax.github.io/bezierinfo/#circles`, 2013.

[5] Thomas F. Hain, Athar L. Ahmad, David D. Langan. Precise Flattening of Cubic Bézier Segments. *Canadian Conference on Computational Geometry*, pages 180–183, 2004.