Elasticurves: Exploiting Stroke Dynamics and Inertia for the Real-time Neatening of Sketched 2D Curves

Yannick Thiel, Karan Singh, Ravin Balakrishnan
Department of Computer Science
University of Toronto
{ythiel, karan, ravin}@dgp.toronto.edu

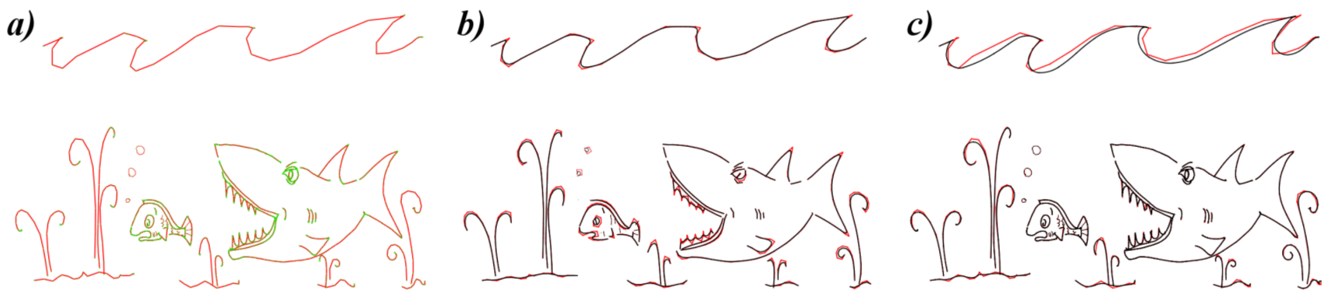

Figure 1: Input strokes are drawn in red, with drawing speed indicated by the spacing of green input points (a). The input stroke in (a) is neatened using Laplacian smoothing with fixed-distance sampling (b), and using elasticurves (c). Note the sharp corners and smooth arcs on the waves and teeth in (c), compared to the featureless smoothing in (b).

ABSTRACT

Elasticurves present a novel approach to neaten sketches in real-time, resulting in curves that combine smoothness with user-intended detail. Inspired by natural variations in stroke speed when drawing quickly or with precision, we exploit stroke dynamics to distinguish intentional fine detail from stroke noise. Combining inertia and stroke dynamics, elasticurves can be imagined as the trace of a pen attached to the user by an oscillation-free elastic band. Sketched quickly, the elasticurve spatially lags behind the stroke, smoothing over stroke detail, but catches up and matches the input stroke at slower speeds. *Connectors*, such as lines or circular-arcs link the evolving elasticurve to the next input point, growing the curve by a *responsiveness* fraction along the connector. Responsiveness is calibrated, to reflect drawing skill or device noise. Elasticurves are theoretically sound and robust to variations in stroke sampling. Practically, they neaten digital strokes in real-time while retaining the modeless and visceral feel of pen on paper.

Author Keywords
Sketching, Stroke-based interfaces, Fair curve design.

ACM Classification Keywords
H.5.2 Graphical User Interfaces, Input Devices, and Strategies;D.2.2User Interfaces;I.3.6 Interaction Techniques

General Terms
Algorithms, Design, Human Factors.

INTRODUCTION

Sketching has been used throughout history as a primitive mode of expression and visual communication. Sketching is also an increasingly viable medium of interaction with devices ranging in size from small tablets to large displays. Sketch strokes are used in a variety of computing scenarios: as curves representing visual content from simple 2D cartoons to complex 3D product designs [4, 15, 16, 23], as motion paths for animation [13, 30] and as general gestural input to invoke commands [3, 17, 22, 24, 25, 34].

An important area of ongoing research deals with attempting to model and eliminate the difference or error between the stroke a user mentally imagines and the one that is drawn using a digital device. In this paper, we refer to this problem as stroke *neatening*.

Stroke neatening can be addressed in two ways: first, by attempting to model the characteristics of the differences between sketched strokes and the resulting curves; second, by using priors that describe desirable properties of curves resulting from user strokes. Perhaps the most common prior is smoothness (Figure 1b) since high-frequency jitter is usually the result of device noise or an unsteady hand and further, smooth or fair [8, 19] curves are generally desirable. Indeed for many applications such as 3D design curves, paths for navigation or spatial curves for visualization, smoothing is sufficient for stroke neatening. There are, however, applications such as 2D cartoon drawing or motion paths for performance-based animation and interactive tracking, where the desired result of a sketch is a mix of smoothness and high-frequency detail in different parts of the neatened output curve (Figure 1c).

In general, the extent to which a user intends the resulting curve to precisely track any portion of a sketched stroke is a directive that must be explicitly defined by the user. While the intended precision along parts of a stroke could be specified after its completion using a variety of interfaces [19, 26], not only would such an approach impede the fluidity of a sketching workflow, it comes too late for real-time applications like drawing or performance animation, where the stroke must be neatened as the user sketches. It would thus be ideal if a user could specify intended precision along the stroke *while* sketching, using an affordance of the drawing tool such as finger pressure or pen tilt [21, 31]. Unfortunately, there is no evidence indicating that there exists a natural relationship between these device affordances and intended precision. We have observed, however, that there is a relationship between drawing speed and intended precision. In accordance with the speed-accuracy trade-off common to human activity [28], users instinctively slow down when drawing parts of a curve where they desire precision and speed up over regions that are smoother or less precise. We note also that drawing speed is a user controlled variable independent of the input device used, be it a finger, mouse, or stylus. Thus, using speed as a mechanism to control precision simply builds on users' inherent sketching behaviour.

A number of approaches that neaten a stroke after its completion [4, 16, 19, 23] typically fit a curve primitive such as a cubic spline [4] or optimize a criterion such as variation of curvature [19], over the entire stroke. Most of these approaches can be further improved by additionally exploiting the precision intent conveyed via stroke speed.

These approaches, however, remain ill-suited to real-time applications, where a neatened curve must be incrementally committed while the user draws. This was especially noted by in-between and clean-up artists sketching over scanned drawings, where the lack of commitment of any part of the neatened curve until a stroke was completed was visually frustrating and often required sketching the same stroke multiple times without guarantee of success. A similar frustration was voiced by animators wishing to lasso-select objects in regions contained within neatened strokes.

To perform real-time neatening, however, the evolving end of a committed curve must differ at times from the current end of the stroke, which we refer to as stroke inertia or spatial lag (see waves in Figure 1c and video). We draw inspiration from the traditional tape drawing technique [5] used by designers where curves are created by rolling out tape with one hand and fastening it with the other. Metaphorically, the hand rolling out the tape defines the stroke and the hand fastening the tape defines the committed curve. The tape in-between the two hands is the spatial lag. In a one-handed sketch-based version of tape drawing [11], the spatial lag has a fixed length and the committed curve can be thought of as being drawn by a pen attached to the user's hand by an invisible rod. The smaller the lag (the shorter the rod), the more closely the committed curve tracks the sketched stroke, and larger lags result in smoother curves with less detail. This is captured in spirit by Dynadraw [12], simulating a pen with mass and friction being physically pulled across the paper. Given that controlling the amount of lag enables the creation of curves with different smoothness and detailed variation characteristics, we propose a novel approach whereby the lag is directly modulated in real-time by the stroke speed. The user can select from different curve primitives such as lines or circular-arcs to model the lag segment, allowing them to generate near perfect lines or arcs despite drawing quickly. At the same time, slowly drawn parts of a curve are tracked precisely without any explicit mode changes.

There exists a continuum of intended curves ranging from completely free-hand to precise geometric primitives like lines and circles [10]. A good real-time stroke neatening algorithm would allow users to move freely within this continuum over the course of a single stroke. We believe elasticurves are the first real-time stroke neatening approach to possess this property.

RELATED WORK

There has been much research in the area of stroke processing. Broadly one branch looks at the symbolic processing of strokes for handwriting and other gestural recognition [3, 17, 22, 24, 25, 34]. Here, the stroke is classified as an instance of a known set of symbols. This is typically done by looking for structure within the stroke in terms of geometric features such as corners or inflections and then by matching these features to corresponding sets of examples for each known symbol. The actual geometry of such strokes serves only to classify and distinguish them.

The second and more relevant branch addresses the neatening of strokes. An essential aspect of stroke neatening is determining which parts of the stroke to neaten. The majority of approaches [4, 19] simply neaten the entire stroke based on the assumption that smooth curves are desirable and that sharp corners or high-frequency detail will explicitly be created by concatenating multiple smooth strokes [4]. While this is perfectly acceptable for many applications, sketches such as that in Figure 1c would be cumbersome to create and require a large number of tiny strokes. Some approaches relax this assumption by breaking the stroke into a number of smooth segments connected at sharp corners [26].

In other approaches users explicitly describe the intended shape of the curve using templates [10] or French curves [27] allowing the creation of very precise curve shapes. The two disadvantages of such approaches are that it is still difficult to transition through different neatening directives within the same stroke, and the external template needs to be invoked explicitly by the user, breaking the desirable flow of pure modeless sketching [14].

There are many different ways of achieving a desired stroke neatening directive. For simple stroke smoothing a variety of techniques exist including Laplacian smoothing [32], cubic spline fitting [4], clothoid fitting [19] and one-handed tape drawing [11]. Piecewise clothoid fitting [19] has been shown to create curves with the most appealing curvature properties but is hard to combine with point, tangent or other precise constraints. Some of these techniques such as Laplacian smoothing and one-handed tape drawing can be used to smooth and commit a stroke as it is drawn, whereas the remainder of the approaches are global in nature, based on optimization or fitting, and require a complete stroke.

Ours is the first approach to propose the principled and explicit use of stroke dynamics as a neatening directive and perform this neatening in real-time as the stroke is sketched. We draw inspiration for this affordance from the kinematics of drawing [18, 28] and research that relates drawing speed to curve features such as cusps and corners [25].

There are a number of other approaches to curve creation and control that are relevant to this work. Fiume [9] introduced arc-length as a control parameter in conjunction with typical Bezier constraints. Using physical forces and dynamics as a control methodology has also been used extensively [6, 12, 29]. Dynadraw [12], aimed at creating calligraphic strokes, indirectly correlates stroke speed and lag by physically simulating a pen pulled across paper. Cords [7] are 3D curves which wrap around scene objects. Cords are procedurally generated from user-defined guide curves and a stiffness parameter that models their pliability. Our elasticurve framework has a similar mathematical formulation.

Variants of the above research exist in commercial software such as Sketchbook-Pro [2], Illustrator [1] and Windows Journal [20]. We discuss these in relation to Elasticurves in the comparisons section.

PROBLEM STATEMENT

Given an input stroke segment *Q*, compute a neatened curve *P* that continuously changes from precisely *Q* to a smooth approximation of *Q* with increasing drawing speed (Figure 3). The curve construction must also be incremental: i.e. if *Q* is a sub-stroke of a longer stroke *Q*', its neatened curve *P* is the corresponding sub-stroke of the longer neatened curve *P'* (Figure 2).

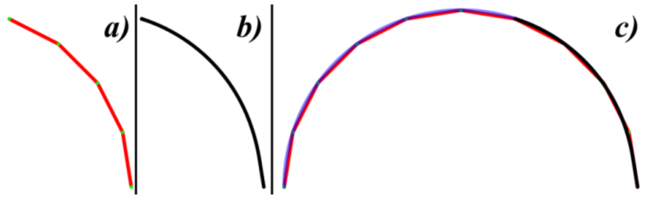

Figure 2: Incremental Elasticurve construction: (a) A partial segment *Q* of an input stroke. (b) The elasticurve segment *P* corresponding to *Q*. (c) The continued stroke *Q'* and elasticurve *P'* (in blue). Once commited, *P* is invariant to subsequent input.

ELASTICURVES ALGORITHM

The elasticurve framework is a "pure" sketching interface, in that all information related to stroke input and neatening is provided by the user in the stroke itself. We describe a minimal number of parameters that allow additional control over the generated curves but in practice users can create their desired curves predictably with the default settings.

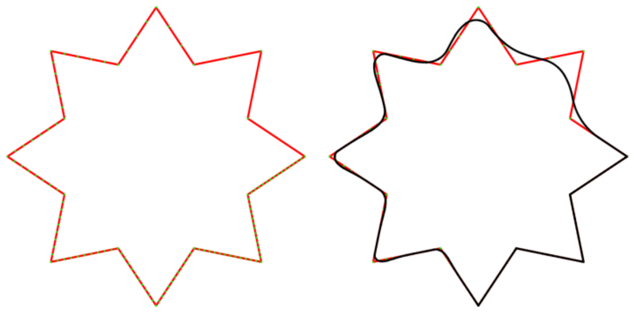

Figure 3: The input stroke (left) is parameterized by time: the spacing of the green input points indicates stroke speed. The elasticurve (right) varies with increasing speed from a precise replica to a smooth approximation of the input.

Input Stroke

Input strokes from current sketching devices are typically a sequence of 2D points that are sampled at a small and regular time interval *dt* ms (see Appendix A). In practice this simply parameterizes the input stroke such that the distance between adjacent point samples is a measure of stroke speed (Figure 3). We denote this input stroke as *Q*, and the i^{th} point on it as q_i. Elasticurves grow as a fraction (called responsiveness) of the spatial lag between the current elasticurve and stroke. Therefore, in a discrete setting, they only ever get infinitesimally close to the input stroke if subsequent points on the stroke are at the same position. In practice we can replace this converging progression of elasticurve points with an analytic curve segment as long as we can detect such a *paused* stroke state. Note that while the elasticurve will inertially lag and catch-up to the stroke as the user draws, the paused state can be thought of as an explicit catch-up of the elasticurve where the stroke inertia or lag is reset to zero. In practice, the curve often enters the paused state at sharp corners and upon stroke completion. We detect a pause in a stroke at a point where there is no movement for dt_{pause} milliseconds.

Curve Generation

We define an inertial responsiveness parameter *r*, which controls the mapping from stroke speed to the neatness of the curve. Users typically calibrate *r* to reflect their drawing skill or the noise and ergonomic inaccuracy of the input device ($0<r<=1$, $r=0.5$ by default). The elasticurve precisely matches the input stroke for $r=1$. Lowering *r* increases stroke inertia (for $r=0$ the elasticurve is a stationary point) resulting in fairer curves (Figure 6).

We will denote the generated elasticurve as *P* and its i^{th} point as p_i. While we can use the metaphor of an oscillation-free zero-rest-length elastic band to describe the inertia between p_i and q_i, this is more to describe its visual behaviour than to accurately model its dynamics. Indeed in our case, the points are generated by a generalization of the formulation in [7], subject to $p_0=q_0$.

$$p_{i+1} = p_i + f(r, p_i, q_{i+1}, \dots) \qquad (1)$$

We will refer to the function f as being the *connector* between the evolving elasticurve and the input stroke. The connector locally controls the shape of the elasticurve and thus models prior knowledge of desirable curve shapes (such as lines or circles) that would connect p_i and q_{i+1}. The elasticurve segment between p_i and p_{i+1} is simply the parametric fraction r of this connector (Figure 5). The overall evolution of the elasticurve for linear and circular-arc connectors is shown in Figure 4. We explored various connector shapes and present the mathematical details of lines and circular-arcs in *Appendix A*.

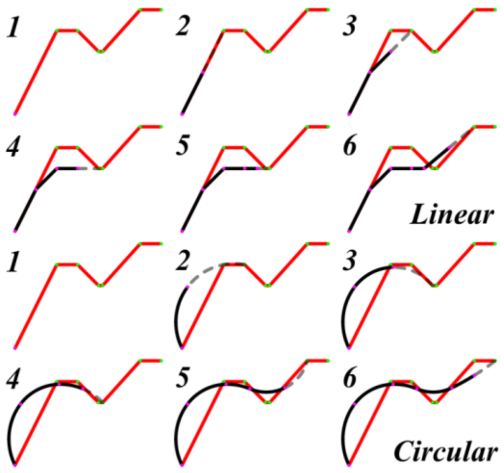

Figure 4: Elasticurve construction for 6 points of an input stroke using linear (top) and circular (bottom) connectors: the elasticurve evolves over six steps. At each step the curve grows by a fraction (r=0.5) along the connector shown by a dashed shape.

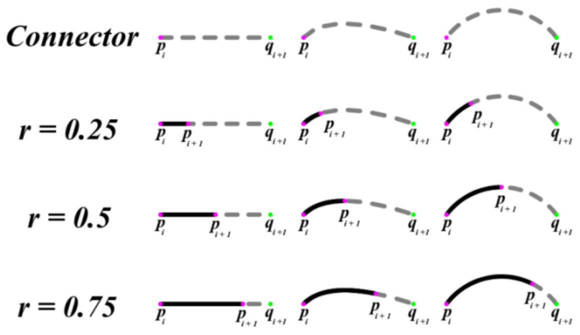

Figure 5: Connector shapes: lines, parabolas, arcs (left to right) with increasing *r* (top to bottom).

Linear Connectors

Linear elasticurves favour a *linear* interpretation of the input stroke. Intuitively, if the user were to provide input points which were all collinear, the generated curve should be a straight line. Deviations from this linearity in the input stroke result in similar deviations in the elasticurve, albeit dampened by a factor of r. The formulation used is:

$$f(r, p_i, q_{i+1}) = r(q_{i+1} - p_i)$$

Leading to the following recursive formula:

$$p_{i+1} = p_i + r(q_{i+1} - p_i)$$

This allows deviations from linearity in Q to be attenuated in P. As a consequence, P will always be more linear than Q especially when drawn rapidly (Figure 6), making the linear formulation ideal for sketching straight lines quickly. Further, while linear elasticurves seem to be only discrete polylines, they converge with finer sampling of an input stroke to a limit curve with the same degree of continuity as the input stroke (*Appendix A*).

Circular-arc Connectors

Circles are also a common shape prior that can be captured as a connector. Circular-arc connectors are defined by the circle passing through q_{i+1}, p_i with the tangent at p_i being the same as the tangent of P at p_i. Using this circle, we take a fraction r of the smaller-arc between p_i and q_{i+1} to find p_{i+1}. (Figure 5). The first circular-arc connector is defined by the smaller circular-arc connecting q_0,q_1,q_2. Like linear elasticurves, circular-arc elasticurves also converge to a limit curve but are G^1 continuous (a sequence of tangent continuous circular-arcs) even in the discrete setting. Circular-arcs can also represent lines (arcs of infinite radius) and are thus our default choice of connector.

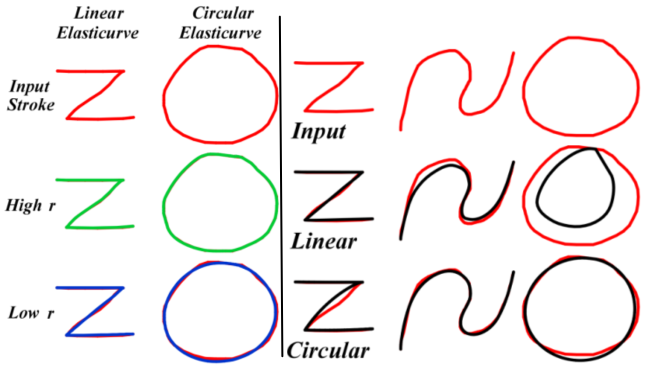

Figure 6: Responsiveness: At high *r*, elasticurves track the input regardless of connector. The impact of the connector shape is evident at lower *r* or when drawing quickly (left). A comparison between linear and circular-arcs for the same input at low *r* shows that circular-arcs handle curved regions better and can also capture line segments (right).

Alternate Connectors

While we present linear and circular connectors in detail, the elasticurve framework can accommodate any parametric connector function *f*. In particular, cubic Beziers can be used if curvature continuous elasticurves are desired. Curves with desired arc-lengths [9] can also be created using a tangent continuous parabolic connector passing through p_i with an arc length given by $r * \|q_{i+1} - p_i\|$.

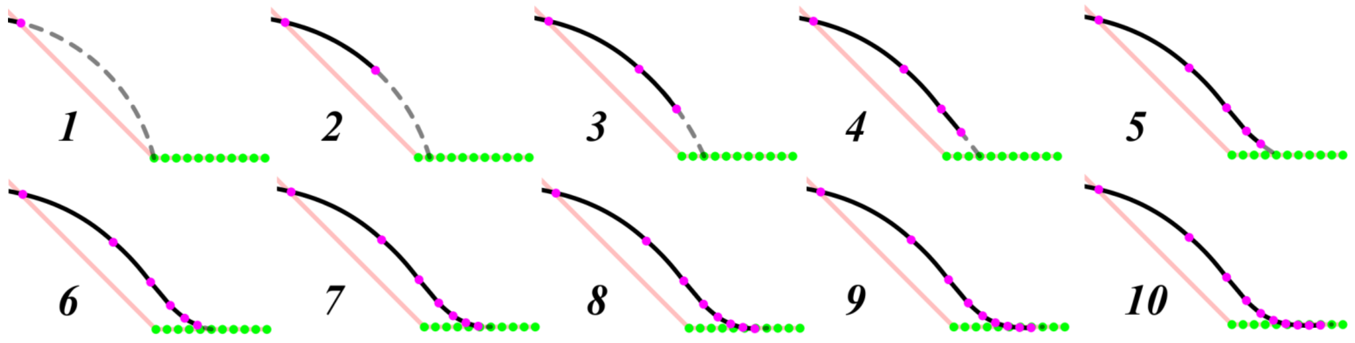

Figure 7: The elasticurve (connector is shown dashed) naturally catches-up to the input in a slowly drawn section.

Elasticurve Properties

Explicit use of drawing speed

An important property of elasticurves is their embodiment of stroke dynamics. Regardless of the connector or the responsiveness, elasticurves *naturally* match the input stroke when the user draws slowly and approximates quickly drawn portions of the stroke, in keeping with the speed-accuracy trade-off seen in human input actions [28].

Figure 7 shows a circular-arc elasticurve approaching a slowly drawn section of the input stroke, indicated by the many input points (sampled at equal time intervals) close to each other. Since the elasticurve grows by a fraction of the connector it gets progressively closer to the input where points are repeated or are close to each other. Within a few such iterations, the elasticurve becomes visually indistinguishable from that of the input stroke until an increase in stroke speed creates visually discernable spatial lag. If multiple points of an elasticurve are built using the same input point repeated over time, the elasticurve converges to the point as an infinite geometric progression of the responsiveness fraction. We identify this condition as a pause in sketching to complete the connector. These pauses occur naturally when drawing sharp corners, cusps and upon stroke completion.

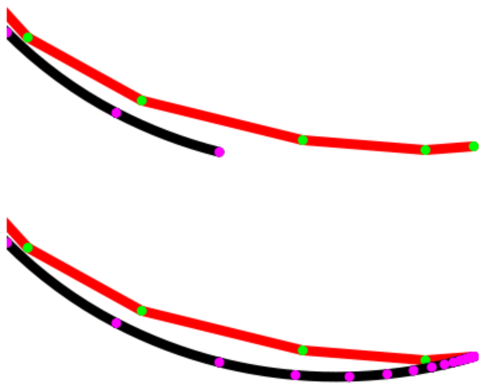
Figure 8: Curve completion: The elasticurve typically lags the end-point of a stroke (top). A paused state causes the curve to grow to the end-point (bottom).

Curve Completion

Once the input stroke is in a *paused* state, which is to say the cursor has not changed position for dt_{pause} ms or the stroke is completed, we complete the curve along its last connector. Conceptually, this is equivalent to creating an infinite number of input points at the given end-point. We implement this by repeating end-points (till the elasticurve is within a distance threshold to the curve) and then snap to the end-point, to keep the point distribution of the generated elasticurves consistent (Figure 8). Using the *paused* state in this manner also allows for the easy creation of cusps and tangent discontinuities, where natural pauses in drawing allow the elasticurve to catch up to the input at corners.

Incremental generation

The incremental construction of elasticurves makes it suitable to real-time curve neatening. In contrast most curve neatening algorithms [4, 19, 26] use a “global” fit to the current stroke, implying that the overall curve is never finalized until the input stroke is completed.

While the “global” fit algorithms can produce curves with nicer mathematical properties such as linear variation in curvature [19], it comes at the cost of losing the immediacy of curve creation, a critical affordance in certain scenarios.

Elasticurve Parameters

Elasticurves have three meaningful parameters: sampling interval (dt), time interval for a pause state to be detected (dt_{pause}), and responsiveness (r). dt is a function of the spatio-temporal resolution of the input device and dt_{pause} a matter of user agility and drawing skill. While we set these manually in our implementation, both are easy to calibrate: dt can be set such that the difference between the user input and the poly-line stroke is below a desired threshold. dt_{pause} can be inferred by asking users to draw a few sharp corners. Responsiveness is the one free parameter that relates to the inertial feel of the curve. While we could attempt to learn a user-specific default setting for r, we find that users grasp its behaviour quickly and adjust it often while sketching.

SPATIAL LAG RELATIONSHIPS

As mentioned above, elasticurves possess an inertial relationship to their input stroke. This spatial lag is a direct consequence of real-time neatening, where the curve can drift from the stroke to smoothen the input but must adhere to it when precision is intended. The spatial lag model used impacts the nature of curves produced. We classify a few existing models: *stick* lag connects the evolving curve and current input by a conceptual stick, akin to one-handed tape drawing [11]. *String* lag models this link as a piece of string. Points are added to the input stroke only when the string is taut, creating curves similar to one-handed tape drawing but allowing sharp corners, by letting the string go slack to abruptly change directions. *Spring* lag, akin to Dynadraw [12], models the link with a zero-rest-length spring, creating physically plausible trajectories that are often pleasing but can have undesirable oscillations. Elasticurves model what can be termed as *speed* lag, where the spatial inertia is directly related to stroke speed. Note that connectors and lag models are complementary and can be arbitrarily paired.

HUMAN ABILITY TO CONTROL SKETCHING SPEED

An obvious question that follows from wanting to use dynamics to indicate stroke precision while drawing is: how naturally and precisely can a user control sketching speed?

Attempting to determine if users had any control over their drawing speed, we asked 5 participants (aged 24-30, 4 right-handed, 1 left-handed), to continuously draw the same shape repeatedly, but with decreasing speed. For example, a participant would begin by drawing a line as fast as they could, and subsequently draw that same line with monotonically decreasing speed. This experiment was performed with lines, Bezier arcs and circles.

The results (Figure 9) illustrate that users do in fact possess reasonable control over their drawing speed, in that they can gradually slow down or speed up. The results, like other sensory controls, tend to follow Weber's law [33], in that a user can more aptly distinguish between slower speeds than fast ones. While a formal investigation of human drawing speed is worthwhile, this experiment suggests that users do possess adequate stroke speed control to utilize elasticurves effectively.

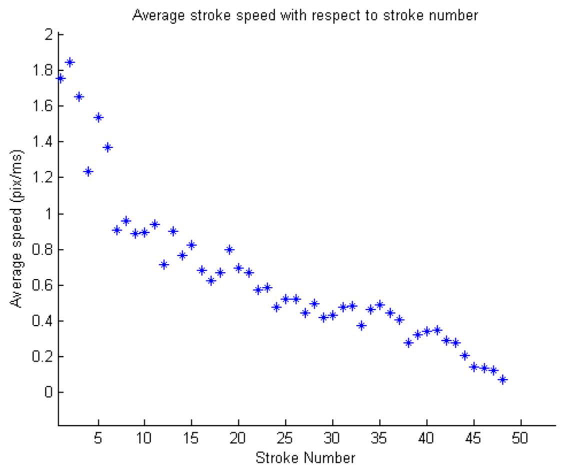

Figure 9: Speed control experiment: plotting average speed (in pix/ms) versus stroke index (decreasing speed).

CURVATURE AND ELASTICURVES

A well-documented perceptual factor affecting sketching speed is curvature. As was noted in [18], stroke speed follows a power law relationship with respect to curvature, namely:

$$v(t) = C * \kappa(t)^{-1/3}$$

Where $v(t)$ is tangential end-point speed, $\kappa(t)$ is the instantaneous curvature of the path, and C is a constant.

This formula entails that sketching speed is lower when attempting to draw areas of high curvature. However, this behaviour arises instinctively; there is no active decision to do so. In other words one might expect that a user intends to draw a curve with the same degree of smoothing but the unconscious drop in speed in regions of high curvature cause them to be drawn more precisely than regions of low curvature. Elasticurves can be modified to compensate for this by computing two speeds: $v_{measured}$, the observed speed of the input stroke, and $v_{expected}$, the speed given by the power law relationship. Discrete curvature is computed at points by looking one input ahead and computing the angle between the neighbours. The difference $v_{actual} = v_{measured} - v_{expected}$ thus captures any conscious variation in stroke speed by the user. We then map v_{actual} to responsiveness.

While the compensation method described above accounts for the perceptual effect curvature has on sketching speed, we found it to have little effect in practice, since the instant visual feedback from the elasticurve lets users compensate for this effect in their drawing directly.

USER FEEDBACK

We evaluated our system by distributing it to 6 users and asking them to try it out and send us their creations. We did not specify which input method they should use; two used a tablet computer, one a trackpad and three a mouse. Each user was aged between 20 and 40, had a familiarity with computers and drawing ability varying from professional to completely inexperienced. Notably, the experienced users used tablets, while the inexperienced ones used a mouse or trackpad as their input device. Their period of usage ranged from one to eight hours. Figures 1 and 10 illustrate sketches created by a user of above-average drawing skill on a pen and tablet interface. Of particular note in these images is how elasticurves satisfactorily handle both precise strokes (such as the arm and book of the character, as well as the fish and shark) and rapid strokes where precision is of no concern (such as the waves, ground, and tree foliage).

Figure 11, on the other hand, illustrates how elasticurves can be used to improve sketching ability. The image was created by a user with no sketching experience using a mouse and a background image to trace over.

The ability of elasticurves to allow users to create desirable primitive shapes (Figures 11-13), with a mouse or trackpad and little drawing experience, is particularly noteworthy.

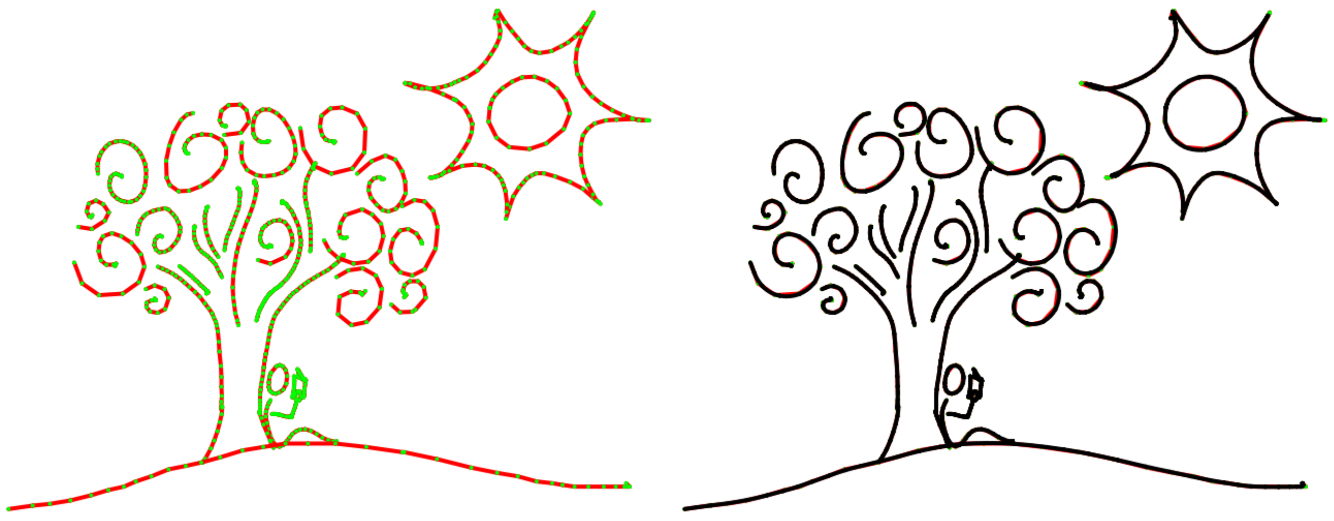
Figure 10: Elasticurve sketch created using pen and tablet interface. Stroke input (left), overlaid elasticurves (right).

An experienced user also mentioned how she felt she had to draw “extra slow” for it to match her exact movements. We feel this is indicative of how, with experience, skilled artists will become proficient at making precise strokes quickly. While increasing responsiveness easily remedies this problem, it does indicate that parameter calibration is recommended (if not required) before extended use. Yet another user complained that the sampling frequency setting was too coarse for their device, once again hinting that calibration may be required on different input devices. One user also complained that the stroke inertia of elasticurves was distracting. The user felt they had to mentally predict the curve’s reaction to their future motion. Given that other users were comfortable with spatial lag, something professional tape-drawing artists live with, we believe that experience with the tool would mitigate such discomfort.

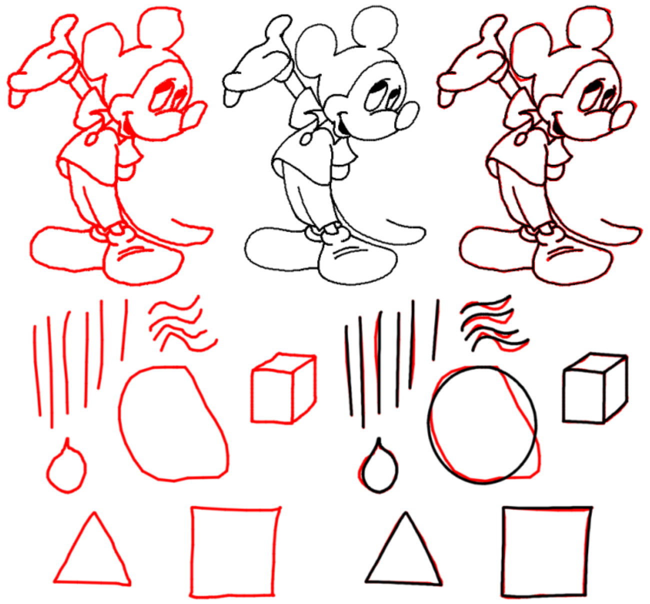
Figure 11: Sketches created by a novice user. Tracing over a background image (top).

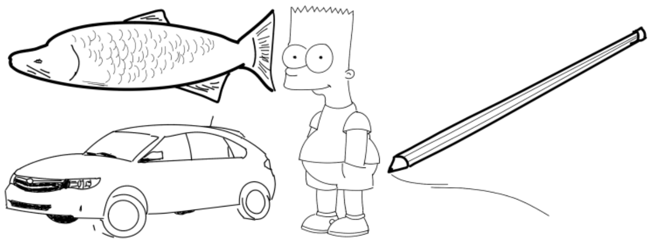
Figure 12: Mouse sketches by an intermediate user.

Users described the interface as “cool”, “fluid” or “physical” and remarked that the system was different from any sketching applications with which they had prior experience. One user who sketched with a mouse mentioned how when using our tool he “drew in a different way than [he] would using something unassisted” because “with [something else], [he] would try to draw things as straight (or circular) as possible, and probably fail, whereas with [elasticurves], [he] would guide the mouse in the general direction knowing roughly how the program was going to correct it”.

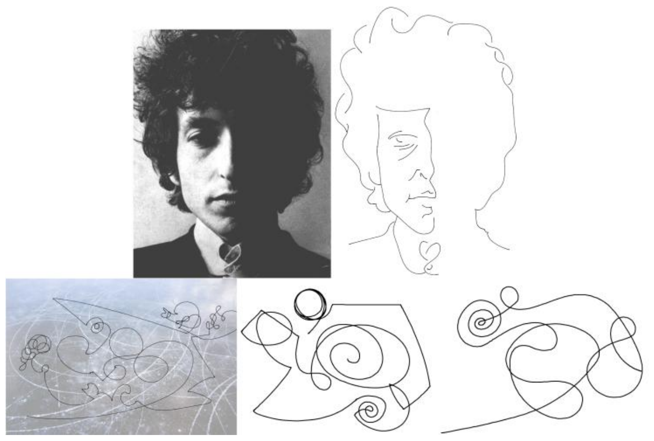
Figure 13: Sketches created using a trackpad by an intermediate user. An oversketch (top). Using the trackpad as a virtual skating rink (bottom).

Another user mentioned that it felt like "drawing with hair on bathroom tiles", which we believe is an allusion to the inertial and smooth nature of elasticurves.

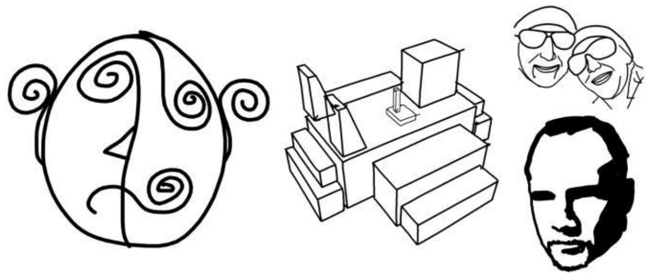
Figure 14: Additional sketches created by various users of intermediate drawing experience.

APPLICATIONS

We believe the affordances provided by an elasticurve sketching system are applicable in many fields. Notable here are: *a front-end to gesture-based interfaces* [3, 17, 22, 24, 25, 34], where the real-time mix of smoothness and sharp corners can provide shape recognizers with improved input; *image tracing for vectorization and cel animation*, where the incremental nature of elasticurves allows visual evaluation of the result while the user draws; *interactive trajectories for performance animation* [30], where in addition to real-time neatening, the timing along the path is encoded in elasticurve parameterization.

In general sketching scenarios, responsiveness provides users a single parameter to control a level of drawing assistance and to compensate for device and motor noise.

Drawing Assistance

Elasticurves were in part developed as a means to assist users lacking the control and practice of a professional artist (Figure 11). Since the responsiveness parameter controls how closely the input stroke is tracked (Figure 6), it provides all users with a manner by which to adjust elasticurves to appropriately augment their personal sketching ability, in particular for drawing near perfect lines or circular arcs.

Device and motor ability compensation

The strong approximations elasticurves with low responsiveness can induce on input strokes make them viable candidates for sketching tasks on commonly used devices that are not designed for drawing such as mice, trackpads, touchscreens and trackballs (Figures 11-13).

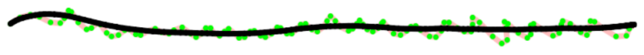
Figure 15: An input stroke with jitter is attenuated using a low responsiveness elasticurve.

A similar argument holds for users lacking fine motor control. As shown in Figure 15, by lowering responsiveness to an appropriate level, jitter and noisy input can be attenuated to create visually appealing strokes.

ELASTICURVE COMPARISONS

The neatening of sketches is a long standing problem with a large body of research. To evaluate elasticurves relative to existing research we compare elasticurves to three popular commercial systems: Windows7 Journal [20], Illustrator CS5 [1] and Sketchbook-Pro 2010 [2].

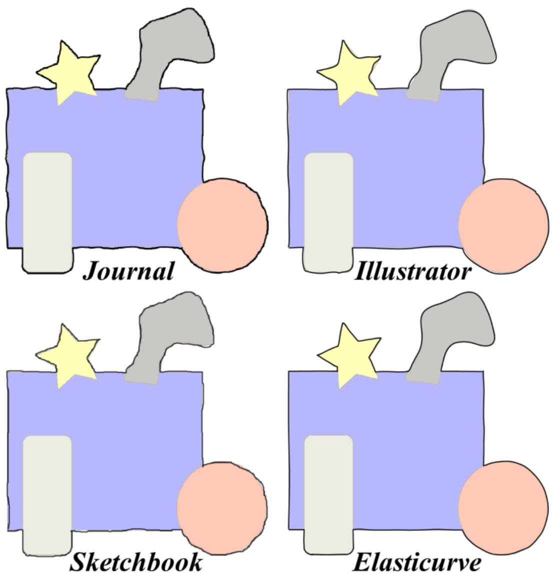

Figure 16: Sketch neatening technique comparison using a trackpad on an image tracing task. The visually neatest result of 7 trials for each technique by an intermediate user are shown.

Sketch neatening in Journal and Illustrator are based on global fitting after stroke completion. Sketchbook-Pro implements a real-time variant of Dynadraw [12]. Theoretically, a fair comparison with elasticurves is difficult. Journal and Illustrator have the advantage of globally optimizing an entire stroke over the real-time local approach of Sketchbook and Elasticurves. Conversely, the latter two use speed as a neatening directive. Elasticurves enable further user control via responsiveness. Despite these issues, to gain some practical insight, we asked an intermediate user to trace the outline of a background image using all four techniques. After a few strokes to gain familiarity with all systems, the user traced over the image with a single stroke. The four techniques were used in turn and repeated overall 7 times. The neatest visual result of each technique using a trackpad is shown in Figure 16. Qualitatively, the neatening in Journal is more localized than Illustrator, resulting in sharp corners but an overall noisier sketch. Illustrator conversely produces a globally smooth but wiggly curve that tends to round subtle corners (the concave corners of the star in Figure 16). Sketchbook is optimized for local real-time neatening and produces noisy results visually similar to Journal. Elasticurves, with low responsiveness, produce the most pleasing result: Corners are precisely created at pauses in the stroke and the circular-arc connector captures lines, arcs and smooth curves with ease. The same task done with a pen produced differences that were subtler but noticeably similar to the trackpad.

Within the genre of elasticurves we further explored viable alternatives to the problem of real-time curve neatening. In both cases we re-parameterized the curve by arc-length and then used stroke speed to modulate a smoothing approach.

Speed modulated Laplacian Smoothing

Simple neighbour averaging is a popular approach to stroke smoothing that is trivial to implement. Conceptually, correlating smoothing strength to stroke speed should result in neatened curves. In practice (Figure 1b), the locality of neighbour averaging can create wiggles or pockets of curvature in curves and varying smoothing strength along the curve can cause irregular spacing of curve points. It also lacks the tangent continuity of circular-arc connectors.

Speed modulated responsiveness

Stroke speed can also be inversely related to the responsiveness of an arc-length parameterized elasticurve. This provides better results than Laplacian smoothing (see Figure 17 and video), but both approaches suffer from re-parameterization artifacts and do not have the convergence guarantees of elasticurves shown in *Appendix A*.

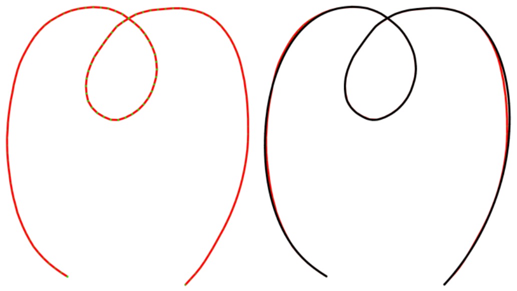

Figure 17: Speed modulated responsiveness built using input points 30 pixels apart. The brightness of the green input points indicates speed.

CONCLUSION

We explored stroke neatening in real-time, and argued that in a general scenario, neatening intent along a stroke should be provided by the user. Motivated by the ergonomics of sketching, we proposed using stroke speed as a neatening directive. An experiment where users drew sequences of strokes while consciously controlling drawing speed led further credence to this choice

We then developed elasticurves, a stroke neatening framework explicitly driven by stroke speed. We capture desirable shapes like lines and circles as connectors along which the elasticurve evolves. We analyzed the geometric properties of elasticurves and showed them to be mathematically stable, robust and convergent with increasing sampling resolution of the input. They are also capable of representing precise shapes like lines and circles.

Our evaluation of elasticurves was two-fold. First a free-form user study with 6 users of varying skills show elasticurves to be an effective and promising solution to the real-time neatening of sketch input. Second, we favourably compared our results with those of three commercial systems for an image tracing task.

Avenues for improvement and future work on elasticurves include the automatic adaption of responsiveness to estimated sketch noise, curvature continuous connectors and an extension of elasticurves to 3D surface modeling.

REFERENCES

1. Adobe Systems Inc. (2010). Adobe Illustrator CS 5. *http://www.adobe.com/products/illustrator.html*
2. Autodesk Inc. (2010). Autodesk Sketchbook Pro 2010. *http://area.autodesk.com/sketchbook*
3. Anderson, D., Bailey, C., & Skubic, M. (2004). Hidden Markov Model Symbol Recognition for Sketch-Based Interfaces. *AAAI Fall Symposium* (pp. 15-21). Menlo Park, CA: AAAI Press.
4. Bae, S.-H., Balakrishnan, R., & Singh, K. (2008). ILoveSketch:As-Natural-As-Possible System for Creating 3D Curve Models. *Proc. UIST* , 151-160.
5. Balakrishnan, R., Fitzmaurice, G., Kurtenbach, G., & Buxton, W. (1999). Digital Tape Drawing. *Proc. UIST*, 161-169.
6. Barzel, R. (1997). Faking Dynamics of Ropes and Strings. *IEEE CGA, 3*, pp. 31-39.
7. Coleman, P., & Singh, K. (2006). Cords: Geometric Curve Primitives for Modeling Contact. *IEEE CGA, 3*, pp. 72-79.
8. Farin, G., Rein, G., Sapidis, N., & Worsey, A. (1987). Fairing Cubic B-Spline Curves. *Computer Aided Geometric Design* , 91-103.
9. Fiume, E. (1995). Isometric Piecewise Polynomial Curves. *Computer Graphics Forum , 1*, pp. 47-58.
10. Fung, R., Lank, E., Terry, M., & Latulipe, C. (2008). Kinematic Templates: End-User Tools for Content-Relative Cursor Manipulations. *Proc. UIST* , 47-56.
11. Grossman, T., Balakrishnan, R., Kurtenbach, G., Fitzmaurice, G., Khan, A., & Buxton, B. (2002). Creating Principal 3D Curves with Digital Tape Drawing. *Proc. CHI* , 121-128.
12. Haeberli, P. (1989). DynaDraw. *Silicon Graphics Corporation*. Mountain View, California, USA. *http://www.graficaobscura.com/dyna/index.html*
13. Igarashi, T., Kadobayashi, R., Mase, K., & Tanaka, H. (1998). Path Drawing for 3D Walkthrough. *Proc. UIST*, (pp. 173-174).
14. Igarashi, T., Kawachiya, S., Matsuoka, S., & Tanaka, H. (1997). In Search for an Ideal Computer-Assisted Drawing System. *INTERACT*, (pp. 104-111).
15. Igarashi, T., Matsuoka, S., & Tanaka, H. (1999). Teddy: A Sketching Interface for 3D Freeform Design. *SIGGRAPH*, (pp. 409-416).
16. Igarashi, T., Matsuoka, S., Kawachiya, S., & Tanaka, H. (1997). Interactive Beautification: A Technique for Rapid Geometric Design. *Proc. UIST*, (pp. 105-114).

17. Labahn, G., MacLean, S., Marzouk, M., Rutherford, I., & Tausky, D. (2006). MathBrush: An Experimental Pen-Based Math System. *Dagstuhl Seminar Proceedings, Challenges in Symbolic Computation*.

18. Lacquaniti, F., Terzuolo, C., & Viviani, P. (1983). The law relating the kinematics and figural aspects of drawing movements. *Acta Psychologica* , pp. 115-130.

19. McCrae, J., & Singh, K. (2008). Sketching Piecewise Clothoid Curves. *SBIM*, pp. 1-8.

20. Microsoft Corporation (2009). Windows 7 Journal.

21. Ramos, G., Boulos, M., & Balakrishnan, R. (2004). Pressure Widgets. *Proc. CHI*, (pp. 487-494).

22. Rubine, D. (1991). Specifying gestures by example. *Proc. SIGGRAPH*, (pp. 329-337).

23. Schmidt, R., Khan, A., Singh, K., & Kurtenbach, G. (2010). Analytic Drawing of 3D Scaffolds. *Proc. SIGGRAPH ASIA (to appear)*.

24. Sezgin, T., & Davis, R. (2005). HMM-based efficient sketch recognition. *Proc. IUI* , 281-283.

25. Sezgin, T., Stahovich, T., & Davis, R. (2001). Sketch Based Interfaces: Early Processing for Sketch Understanding. *Proc. PUI* .

26. Shao, L., & Zhou, H. (1996). Curve Fitting with Bezier Cubics. *Graphical Models and Image Processing , 3*, pp. 223-232.

27. Singh, K. (1999). Interactive Curve Design using Digital French Curves. *Proc. I3D*, 23-30.

28. Soukoreff, R., & MacKenzie, I. (2009). An informatic rationale for the speed-accuracy trade-off. *Proc. IEEE SMC*, (pp. 2969-2975).

29. Terzopoulos, D., & Qin, H. (1994). Dynamic NURBS with Geometric Constraints for Interactive Sculpting. *ACM TOG, 2*, pp. 103-136.

30. Thorne, M., Burke, D., & van de Panne, M. (2004). Motion Doodles: An Interface for Sketching Character Motion. *ACM TOG, v.23 n.3.*

31. Tian, F., Ao, X., Hongan, W., Setlur, V., & Dai, G. (2008). Tilt menu: using the 3D orientation information of pen devices to extend the selection capability of pen-based user interfaces. *Proc. CHI*, (pp. 1371-1380).

32. Tsang, S., Balakrishnan, R., Singh, K., & Ranjan, A. (2004). A suggestive interface for image guided 3d sketching. *Proc. CHI*, (pp. 591-598).

33. Weber, E. (1846). Der Tastsinn und das Gemeingefühl. In Wagner, *Handlewörterbuch der Physiologie* (Vol. iii).

34. Wobbrock, J., Wilson, A., & Li., Y. (2007). Gestures without Libraries, Toolkits or Training: a 1$ Recognizer for User Interface Prototypes. *Proc. UIST*. (pp 159-168).

APPENDIX A

We present the computation of linear and circular-arc connectors. We also show convergence to a continuous limit elasticurve with increased sampling frequency

Linear Elasticurves

$$f(r, p_i, q_{i+1}) = r(q_{i+1} - p_i)$$

And substituting back into (1), we get:

$$p_{i+1} = p_i + r(q_{i+1} - p_i)$$

Writing the responsiveness r as *s*dt*, where we fix s to be constant and *dt* is the sampling interval of time we get:

$$dp/dt = s(q(t) - p(t))$$

which is a linear first order differential equation, whose solution is the elasticurve *p(t)* as *dt→0.*

Circular-arc Elasticurves

Figure 18 shows the circle passing through q_{i+1}, p_i with tangent $\boldsymbol{t_i}$ at p_i, where $\boldsymbol{t_i}$ and $\boldsymbol{n_i}$ are the tangent and normal of the elasticurve at p_i.

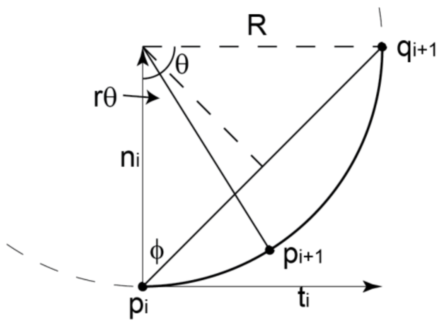

Figure 18: Computing the circular-arc connector.

$$f(r, p_i, q_{i+1}) = R(1 - \cos(r\theta))\boldsymbol{n_i} + R\sin(r\theta)\ \boldsymbol{t_i}$$

The circle radius R = $||q_{i+1} - p_i||/(2\cos\phi)$ and the connector angle $\theta = \pi - 2\phi$, where

$$\cos\phi = (q_{i+1} - p_i).\boldsymbol{n_i}/||q_{i+1} - p_i||.$$

As before writing r as *s*dt* we get:

$$dp/dt = R/dt[(1 - \cos(sdt\theta))\boldsymbol{n_i} + \sin(sdt\theta)\,\boldsymbol{t_i}]$$

In the limit as *dt→0*, $dp/dt = Rs\theta\boldsymbol{t_i}$

which verifies that the curve is tangent continuous and shows that *p(t)* converges to the solution of the above equations as *dt→0.* It is important to note here that the vectors $\boldsymbol{t_i}$ and $\boldsymbol{n_i}$ are updated by a rotation of $r\theta$.