Sketch-Based Path Design

by

James Palmer McCrae

A thesis submitted in conformity with the requirements
for the degree of Master of Science
Graduate Department of Computer Science
University of Toronto

Abstract

Sketch-Based Path Design

James Palmer McCrae

Master of Science

Graduate Department of Computer Science

University of Toronto

2008

We first present a novel approach to sketching 2D curves with minimally varying curvature as piecewise clothoids. A stable and efficient algorithm fits a sketched piecewise linear curve using a number of clothoid segments with G^2 continuity based on a specified error tolerance.

We then present a system for conceptually sketching 3D layouts for road and other path networks. Our system makes four key contributions. First, we generate paths with piecewise linear curvature by fitting 2D clothoid curves to strokes sketched on a terrain. Second, the height of paths above the terrain is automatically determined using a new constraint optimization formulation of the occlusion relationships between sketched strokes. Third, we present the break-out lens, a novel widget inspired by break-out views used in engineering visualization, to facilitate the in-context and interactive manipulation of paths from alternate view points. Finally, our path construction is terrain sensitive.

Acknowledgements

I would like to acknowledge the efforts of my supervisor, Karan Singh, and thank him for his guidance over the duration of the Masters program. I learned much from him as a result of our meetings. I believe that his broad knowledge and talents as a supervisor brought out the best in me.

I would also like to acknowledge Arnold Rosenbloom, a senior lecturer at the University of Toronto Mississauga, who is an outstanding teacher. Without his influence, I am certain that I would not have pursued graduate studies.

Thank you to members of the DGP lab for their welcoming attitude, despite their critical responses to my work. In retrospect, interactions with them have always been helpful and in part inspired me to continue on with my education.

Finally, this opportunity would never have been made possible without the love and support of my parents. Thank you for everything.

Contents

List of Figures

Chapter 1

Introduction

"Two roads diverged in a wood,

And I took the one less traveled."

– Robert Frost

Curves are ubiquitous in Computer Graphics, as primitives to construct shape or define shape features, as strokes for sketch-based interaction and rendering or as paths for navigation and animation. Motivated originally by curve and surface design for engineering applications, complex shapes are typically represented in a piecewise manner, by smoothly joining primitive shapes (see Figure 1.1). Traditionally, research on curve primitives has focused on parametric polynomial representations defined using a set of geometric constraints, such as Bezier or NURBS curves [12]. Such curves have a compact, analytically smooth representation and possess many attractive properties for curve and surface design. Increased computing power, however, has made less efficient curve primitives like the clothoid a feasible alternative for interactive design. Dense piecewise linear representations of continuous curves have also become increasingly popular. Desirable geometric properties, however, are not intrinsically captured by these polylines but need to be imposed by the curve creation and editing techniques used [15, 44, 7].

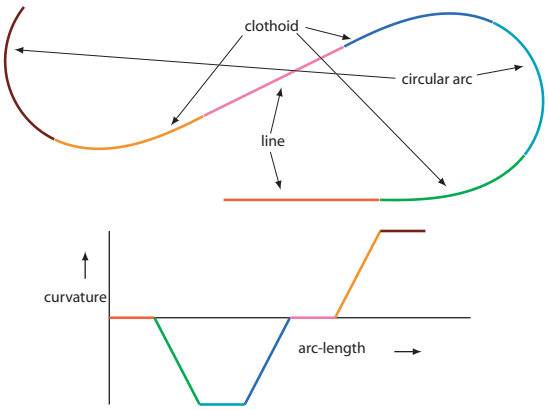

Figure 1.1: A curve composed of clothoids, line and circular-arc segments.

An important curve design property is *fairness* [13, 38, 31], which attempts to capture the visual aesthetic of a curve. Fairness is closely related to how little and how smoothly a curve bends [31] and for planar curves, described as curvature continuous curves with a small number of segments of almost piecewise linear curvature [13].

The family of curves whose curvature varies linearly with arc-length were described by Euler in 1774 in connection with a coiled spring held taut horizontally with a weight at its extremity. Studied in various contexts in science and engineering, such a curve is also referred to as an Euler spiral, Cornu spiral, linarc, lince or clothoid (see Figure 1.2). Clothoids are especially useful in transportation engineering, since they can be navigated at constant speed by linear steering and a constant rate of angular acceleration. Roller-coasters are frequently composed of sequences of clothoid loops. While intrinsic geometric splines like clothoids were introduced in computer aided design in 1972 [35] and subsequently developed as transition curves for road design [29, 45], they have had little recent exposure in mainstream Computer Graphics. In this paper, we exploit the fairness properties of clothoids to fit 2D strokes for sketch-based applications.

Mankind has been constructing paths in nature for millenia. While path layouts are necessary for transportation design (roads, railway tracks, nature trails), they are also important as motion paths for animation, navigation and visualization in virtual envi-

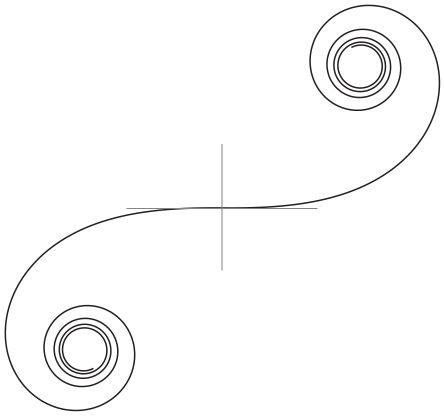

Figure 1.2: Clothoid: a curve whose curvature varies linearly with arc-length, also known as an Euler spiral, Cornu spiral or linarc. The above clothoid has a curvature range $[-1.15, 1.15]$ and arc-length 100 (or $t \in [-5.362, 5.362], B = 3.72$).

ronments. Noted landscape architect Lawrence Halprin [17] points out that the design of such paths should emphasize the driving experience. In other words, the spatio-temporal aesthetics of path design are as important as its engineering requirements. Unfortunately, unlike 3D shape design where concept sketching interfaces are now abundant, sketch-based path design is largely unexplored. Shape modeling interfaces like *Google SketchUp*, [20, 39], are not suitable for conceptual path layout, forcing designers to reluctantly work with engineering focused CAD tools such as *AutoCAD Civil3D*.

In this paper we present a coherent sketch-based system, *Drive*, specifically aimed at conceptual path design. While we use road networks to graphically illustrate the system (see Figure 1.3), our system is easily adapted to paths representing railroads, waterways, nature trails, pipe or power lines, graph networks, networks of surface patches and general 3D curve based modeling (see Figure 3.9).

Being a system focused on sketch-based interaction techniques, *Drive* differs greatly from applications conventionally used by landscape architects and civil engineers for path design. University of Toronto landscape architect John Danahy claims professional toolsets require "several minutes" to perform useful operations, whereas similar opera-

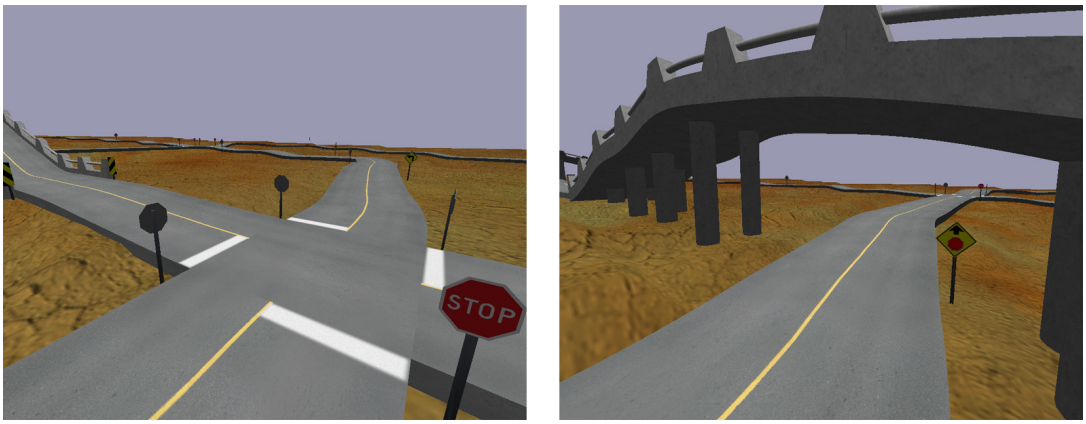

Figure 1.3: A road network with signs created in *Drive*.

tions can be performed within *Drive* in seconds by using simple gestures. The ability to produce prototypes rapidly that integrate themselves into the environment is a significant advantage. This advantage becomes especially apparent in the modelling of paths where exact precision is less of a requirement, e.g. a private driveway to a cottage, or a path traversed by labourers in logging and forestry.

The decision to create a sketch-based system is further justified by attempting to minimize the cognitive overhead that comes with working within the constraints of a particular modelling system. John Danahy notes that prevalent software toolsets available which perform path and other modelling tasks often force the user to alter their way of thinking about a problem. For example, the particular ordering of steps of a given task is ordained by the system and not the user, and may vary from what is conventional in the profession, which results in cognitive issues during use.

An important goal of *Drive* is that the task of modelling feel very natural and free-form and attempt to avoid or minimize any cognitive issues. This has motivated our approach of both interaction and visualization techniques. Considering interaction, as an example, University of Toronto landscape architect Robert Wright has said that the ability to sketch a path and embed it on arbitrary geometry is non-trivial and is itself a useful feature, that is not found in all commercial applications used in his profession.

Considering visualization, as an example, we have developed the break-out lens, a novel tool that provides an alternate viewpoint with significant contextual information, with the intention of reducing cognitive overhead.

Another significant benefit of *Drive* is the speed at which the system can be learned. All of the operations that can be performed within the system can be explained in some detail within minutes. Even better, the interface is such that most if not all of the functionality can be learned by the user just by giving the sole instruction: open curves create paths and closed curves correspond to a selection-action operation and invoke a menu. For all users who experimented with the system, this very brief instruction was enough to get them started and discover each of the individual system functions on their own. In contrast, conventional road design applications have a much steeper learning curve, often coupled by lengthy and complex technical manuals.

Designed and implemented within the context of this system, we draw attention to four novel components:

Clothoids: We interactively fit sketched strokes using a number of line, circular-arc and clothoid curve segments. Clothoids are the family of spirals whose curvature varies linearly with arc-length. They are widely used in transportation engineering, since they can be navigated at constant speed by linear steering and a constant rate of angular acceleration [29, 45]. Stroke filtering is important for almost all sketch-based applications and we show our curves to have smoothness properties superior to the common practice of spline fitting [37] or iterative point smoothing [44] (see Figure 2.1).

Crossing paths: We handle arbitrarily complex path crossings by sketching occlusion as breaks in the curve path (see Figure 3.4). We process the sketched strokes to implicitly join these breaks and define inequalities of path height. We then efficiently solve for height along the path as an optimization that minimizes the height of the path from the terrain, the grade of the path, and the overall variation of height needed to satisfy the occlusion relationships (see Figure 3.5).

Break-out lens: Inspired by break-out views used to indicate orthogonal viewpoints in engineering visualization, we develop an interactive lens that performs a continuous view warp to provide an in-context break-out view (see Figure 3.8). An important affordance of the break-out lens is that sketching within the lens allows multi-view 3D curve editing, without the handicap of mentally resolving the 3D curve from disconnected views.

Terrain sensitive sketching: An appealing aspect of our system is our focus on both the construction of path layouts and the rapid conceptualization and preview of the entire driving experience. The terrain thus is not just a canvas on which to sketch paths but an evolving environment into which the paths integrate, with automatic construction of bridges, tunnels, road signs and changes in foliage (see Figure 1.3, 3.10, 3.11).

Chapter 2

Sketching Piecewise Clothoid Curves

2.1 Problem statement

Polyline stroke data often needs to be denoised and processed into fair 2D curves for further use in many sketch-based applications. This is usually done using smoothing filters [44] or by cubic or high-order spline fitting [36, 37]. Iterative smoothing is best suited to removing high-frequency sketching noise and tends to produce low-frequency wiggles in the curve (local pockets of smooth curvature based on filter size). Spline fitting results, though visually smooth, frequently exhibit poor quality curvature plots (see Figure 2.1). We present a new approach to processing sketch strokes using clothoids, that intrinsically favour line and circular arc segments and result in holistically fair G^2 curves.

2.2 Overview of our approach

Our algorithm for fitting a sequence of G^2 clothoid segments to polyline stroke data is a two-step process (see Figure 2.2). We first fit a piecewise linear approximation to the discrete curvature of the stroke as a function of arc-length, with control over the tradeoff between fitting error and the number of linear pieces. The start and end

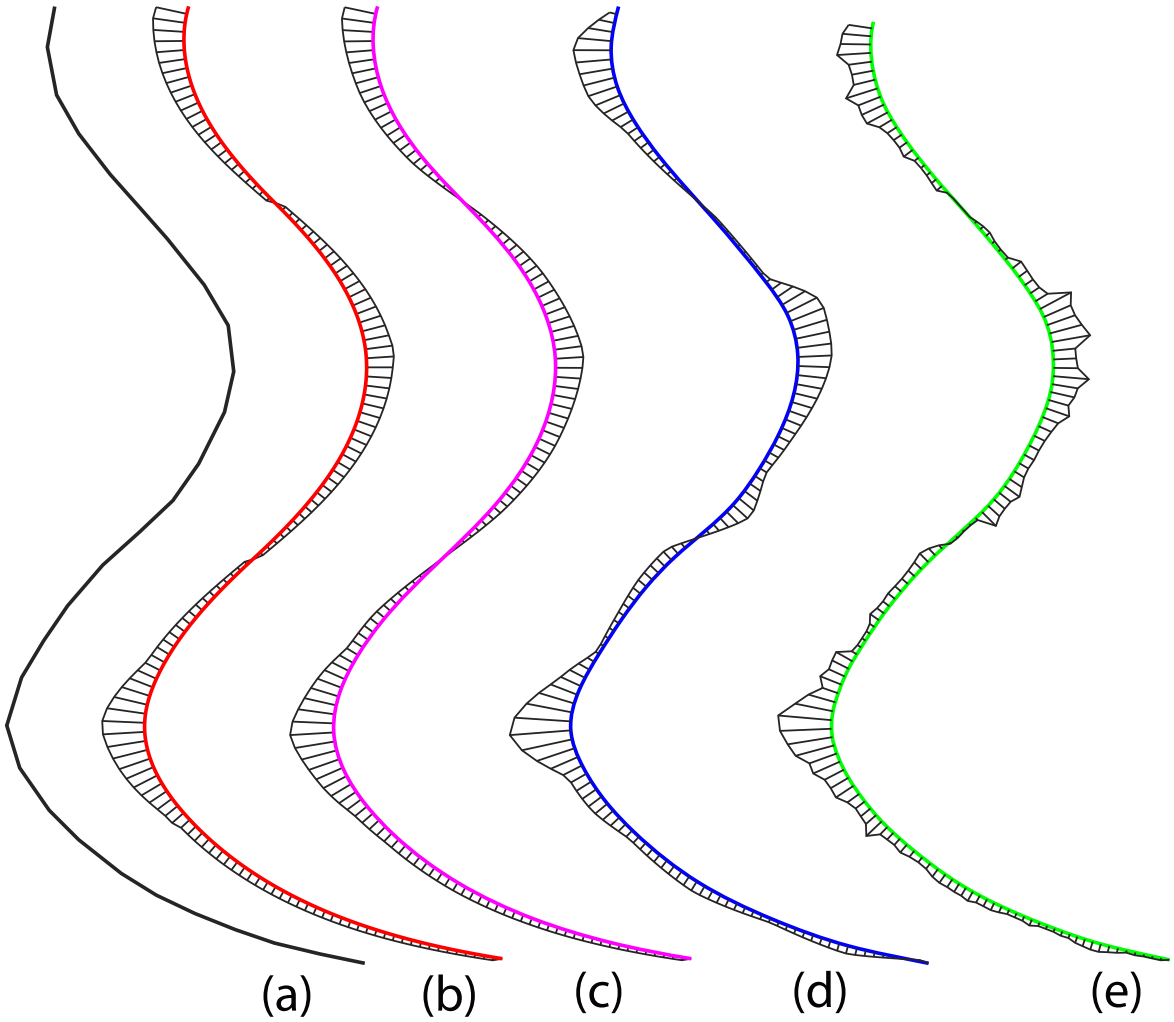

Figure 2.1: Stroke fairing: (a) A sketched stroke. (b) Clothoid fitting the stroke (a). (c) Cubic spline fitting the clothoid curves in (b). (d) Cubic spline fitting the stroke (a). (e) Laplacian smoothing (4 iterations at 10%) the stroke (a). Curvatures are plotted uncolored along the length of processed strokes (b-d) to evaluate smoothness.

curvature values of each linear piece uniquely determine a line, circular arc or clothoid curve segment. These segments further assemble together uniquely with G^2 continuity into a single composite curve. The next step involves determining a single $2D$ rigid transform that aligns this composite curve with the sketched stroke to minimize the error of the stroke from the transformed curve. We are able to solve for this transform efficiently by formulating the error as a weighted least squares optimization problem. While many sketch-based applications do not require precise interpolation of points and tangents, we show how this can be achieved by inserting or appending short spline segments to enforce interpolation (see Figure 2.10), if necessary. The resulting curve can also be made G^3 by linearly blending the adjacent clothoid segments locally (see Figure 2.9). Alternatively, sharp corners can be allowed by thresholding spikes in curvature to be segment boundaries and independently rotating these segments (see Figure 2.8).

2.3 Contributions

We develop a new formulation for efficiently fitting intrinsic spline primitives such as clothoids, to dense polyline data. While we focus on clothoids our algorithmic framework is applicable to any curve primitive with a characteristic curvature profile. The resulting curves are robust to sketching noise and are particularly well suited to sketch-based applications. We show a number of enhancements to the basic approach, including sharp corners, blended G^3 curves and point interpolation. Finally, we have implemented our results within a sketch-based application for track design (see Figure 2.3), where the clothoid segments provide not only aesthetically pleasing curves but are also required downstream, from an engineering standpoint.

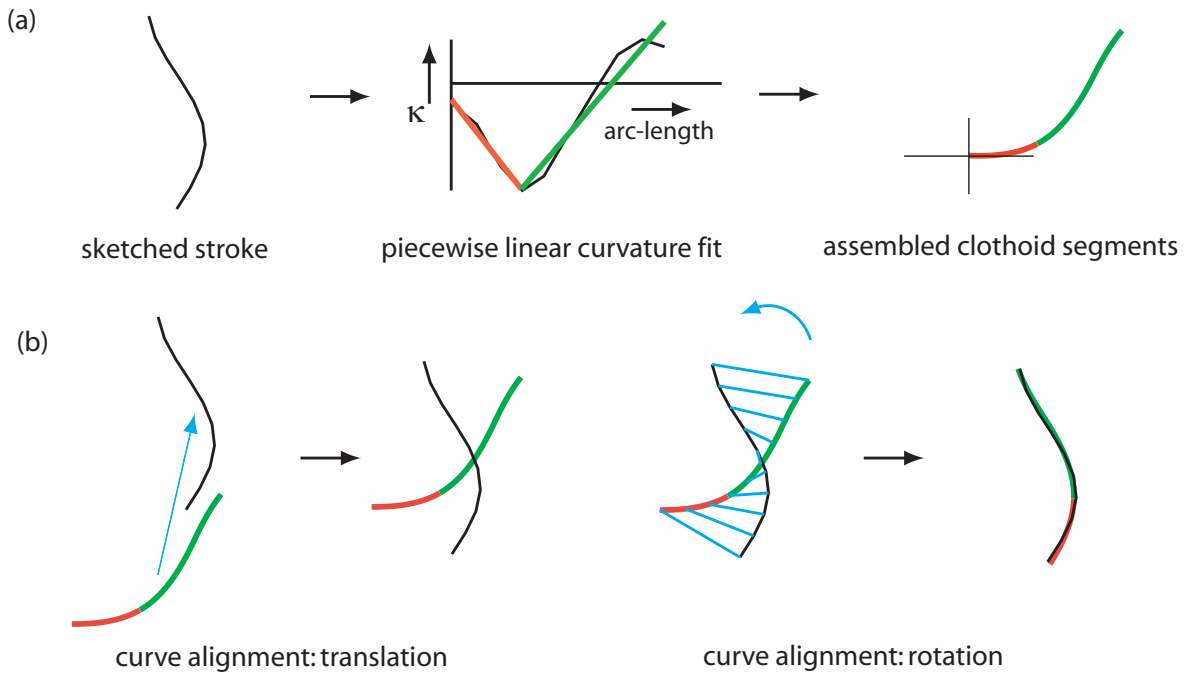

Figure 2.2: Clothoid fitting: (a) Discrete stroke curvature is approximated as a piecewise linear function uniquely defining clothoid segments. (b) A rigid 2D transform minimizes the weighted least squares error between the composite clothoid and the sketched stroke.

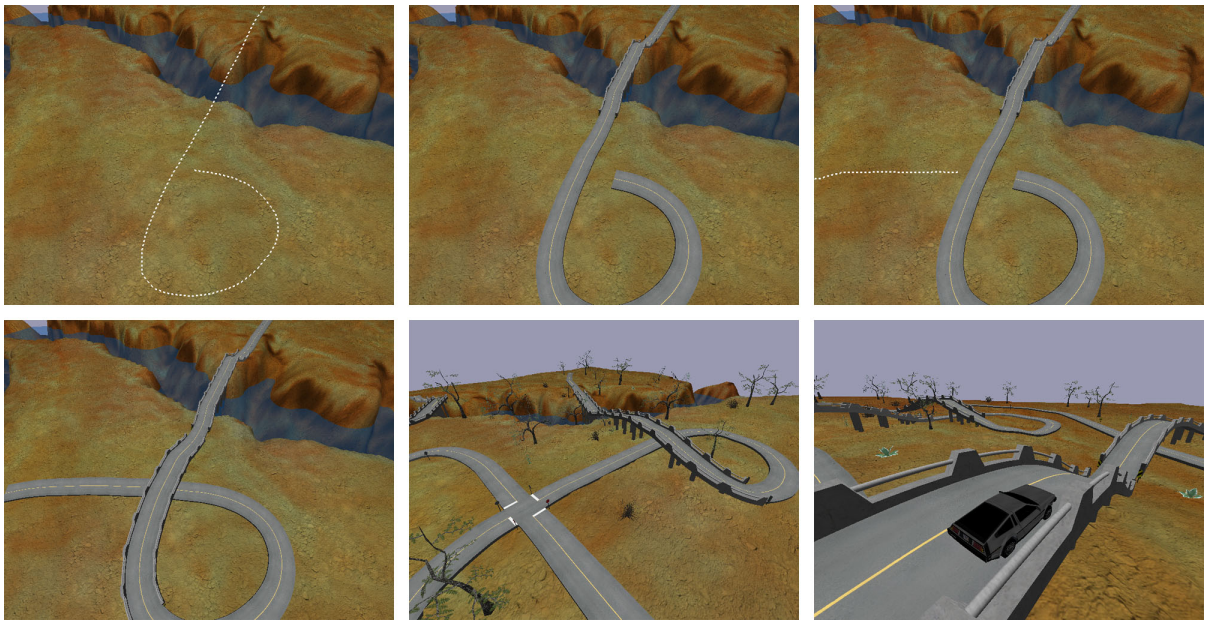

Figure 2.3: Sketching clothoid splines within *Drive*, a sketch-based track modeling system.

2.4 Related work

We now survey prior work specifically relating to curve and surface fairing in general and on clothoids in particular. A popular feature of cubic splines is that they provide a linear approximation to the minimum strain energy configuration of a thin-beam interpolating a set of points. While least squares spline fitting is robust and efficient [37], the curvature plot of the resulting spline can be highly variable (see Figure 2.1). Computing the actual minimum energy curve minimizes the overall bending of the curve [30]. Moreton and Sequin [31] show, however, that minimum variation curves provide a better fairness characteristic by minimizing the overall variation of curvature along the curve allowing natural shapes like circular arcs. These curves are typically computed by nonlinear optimization techniques. In contrast, we attempt to minimize overall variation in curvature along the curve by robustly approximating it using a number of piecewise linear segments. Our composite clothoid curve is thus an appealing alternative to minimum variation curves, particularly when precise interpolation of points is traded for precise

curvature control.

A more common, easy to implement, approach is to iteratively smooth the points of piecewise linear curves and surfaces directly [44]. Discrete filtering approaches vary from simple neighbour averaging to approaches that use a discrete curvature estimation to help guide the fairing process [33]. We similarly compute a discrete curvature estimate at points of the input polyline, but instead use these values to determine the segmentation of the curve into clothoid pieces. An additional advantage of fitting analytic curve segments like splines or clothoids over discrete methods is that the curve can be regenerated at arbitrary resolution.

Clothoids have been the subject of prior research in computer aided design. Motivated by transportation design, Meek and Walton have looked at conditions under which one or more clothoid segments can form a transition curve between two given curve segments [29]. They have also proposed a clothoid spline [45], where two clothoid pieces are used to form a parabola-like segment between every three consecutive points of a control polygon. While the resulting clothoid spline is G^2, the curve is forced through a point of zero curvature on every edge of the control polygon. A discrete formulation of clothoid using nonlinear subdivision has also been proposed [16]. Clothoids have also been used as a transition curve segment for computer vision applications of occluded contour completion and in-painting [26].

Originally motivated by a system for quickly sketching track layouts for game environments and road layout conceptualization by landscape architects, we find clothoids to be attractive curve primitives that qualitatively capture the natural curvature variations of human sketching well.

2.5 Clothoid Terminology

The clothoid spiral can be parameterized using the Fresnel integrals

$$C(t) = \int_0^t \cos \frac{\pi}{2} u^2 du, \tag{2.1}$$

$$S(t) = \int_0^t \sin \frac{\pi}{2} u^2 du, \tag{2.2}$$

as

$$\pi B \begin{pmatrix} C(t) \\ S(t) \end{pmatrix}, \tag{2.3}$$

where t is the arc length parameter, and πB is a positive scaling parameter that defines the slope of linear curvature variation of a family of spirals as seen in Figure 2.4.

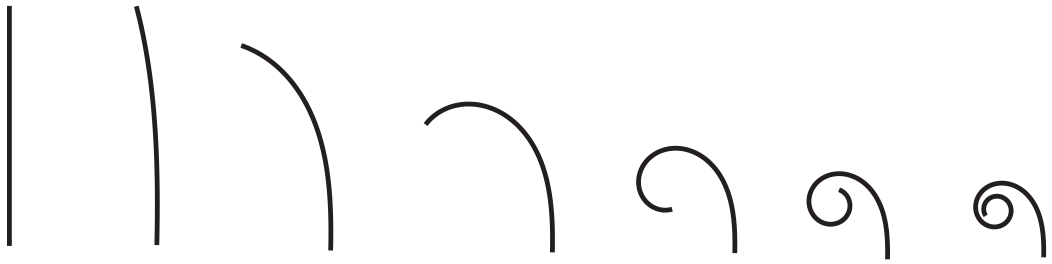

Figure 2.4: Fixing arc length and an initial curvature parameter, a family of clothoid segments is formed by decreasing parameter B near infinity (left) toward zero (right).

Clothoids can be expressed in a computationally efficient manner, using rational approximations for $C(t)$ and $S(t)$ given in [18]:

$$C(t) \approx \frac{1}{2} - R(t) \sin \left(\frac{1}{2} \pi (A(t) - t^2) \right), \tag{2.4}$$

$$S(t) \approx \frac{1}{2} - R(t) \cos \left(\frac{1}{2} \pi (A(t) - t^2) \right), \tag{2.5}$$

where

$$R(t) = \frac{0.506t + 1}{1.79t^2 + 2.054t + \sqrt{2}},$$
$$A(t) = \frac{1}{0.803t^3 + 1.886t^2 + 2.524t + 2}.$$

2.6 Curve fitting using clothoids

We now detail our approach to curve fitting using a sequence of clothoid, circular arc and line segments (see Figure 1.1,2.11). Note that while the steps below fit a polyline, they can be used to fit any curve representation that is discretely sampled at an appropriate resolution.

2.6.1 Discrete curvature estimation

Discrete curvature for planar curves can be estimated at a point using the circum-circle formed with its two adjacent points or the Frenet-Serret fomulae as shown in [33]. Given any three sequential points p_{i-1}, p_i, p_{i+1} of the input polyline, using the vectors $v_1 = p_i - p_{i-1}$, $v_2 = p_{i+1} - p_i$, the estimated curvature at p_i is given by

$$\kappa(p_i) = \frac{2\sin\left(\frac{\theta}{2}\right)}{\sqrt{||v_1|| \cdot ||v_2||}}, \tag{2.6}$$

where

$$\theta = \arccos\left(\frac{v_1}{||v_1||} \cdot \frac{v_2}{||v_2||}\right). \tag{2.7}$$

Robust statistical approaches to curvature computation that perform better in the presence of noise and irregular sampling [21] can also be used. The curvature for discretely sampled analytic curves may also directly sampled from the analytic curve.

Each point is now mapped into curvature space, where the horizontal axis denotes arc length and the vertical axis, curvature (see Figure 2.2a). We adopt (positive/negative) curvature to denote (right/left) turning in this space.

2.6.2 Piecewise linear curvature segmentation

We now segment the curve into a minimal sequence of pieces of linearly varying curvature. A dynamic programming algorithm finds a connected set of line segments which minimize both the number of line segments used, and the error in fit with the curvature space points. The number of pieces used is minimized by assigning a penalty E_{cost} for each linear piece. We populate a matrix M with values, in a bottom-up fashion, using the following:

$$M(a,b) = \min_{a<k<b} \left\{ M(a,k) + M(k,b), E_{fit}(a,b) + E_{cost} \right\}. \tag{2.8}$$

$M(a,b)$ denotes the minimal cost of a configuration of connected line segments from point a to b. $M(a,b)$ entries are calculated for all $a < b$, making M strictly upper triangular. $E_{fit}(a,b)$ denotes the vertical error resulting from linear regression with the points from a to b. Expressing the linear regression line using slope and y-intercept, denoting them l_{slope} and l_{yint} respectively, we can define E_{fit} precisely as

$$E_{fit}(a,b) = \sum_{i=a}^{b} \left| l_{yint} + l_{slope} \cdot arclength(p_i) - \kappa(p_i) \right|, \tag{2.9}$$

where $(arclength(p_i), \kappa(p_i))$ is the curvature-space point corresponding to p_i.

The solution, a set of connected line segments in curvature space, defines the set of connected clothoid segments that will be used to fit the input curve. Figure 2.5 shows the effect that different values of E_{cost} has on the generated solution.

2.6.3 Segment parameterization

For each clothoid segment, we have its curvature space endpoints (x_i^P, y_i^P) and (x_{i+1}^P, y_{i+1}^P). y_i^P and y_{i+1}^P specify the start and end curvatures of the segment, and the difference $x_{i+1}^P - x_i^P$ specifies the arc length. These parameters uniquely map to a clothoid segment defined by the scaling parameter B, and the start and end parameter values t_1 and t_2. Since the curvature of a clothoid is $\frac{t}{B}$:

$$t_1 = y_i^P B \text{ and } t_2 = y_{i+1}^P B. \tag{2.10}$$

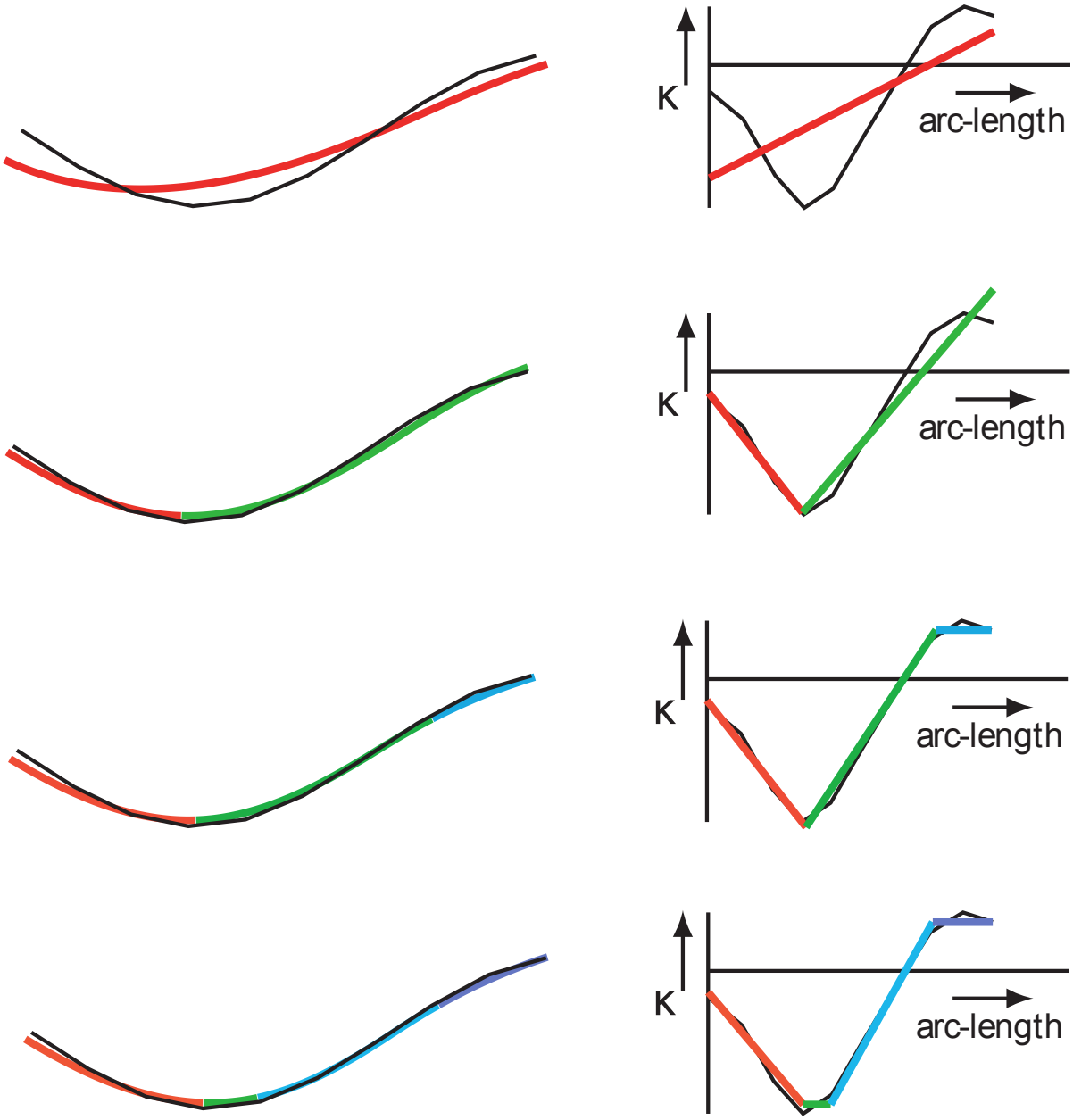

Figure 2.5: The effect of E_{cost} on the generated segmentation. As E_{cost} decreases, more segments are used.

B can be expressed using the formula for arc length:

$$\begin{aligned} x_{i+1}^P - x_i^P &= \pi B(t_2 - t_1) \\ &= \pi B(y_{i+1}^P B - y_i^P B) \quad \text{(using (2.10))} \\ &= B^2 \pi (y_{i+1}^P - y_i^P) \\ \frac{x_{i+1}^P - x_i^P}{\pi(y_{i+1}^P - y_i^P)} &= B^2 \end{aligned}$$

and since B must be positive,

$$B = \sqrt{\frac{x_{i+1}^P - x_i^P}{\pi(y_{i+1}^P - y_i^P)}}. \tag{2.11}$$

Each clothoid segment is translated and rotated to connect end points and align tangents to adjacent segments resulting in an overall G^2 curve (see Figure 2.2a).

2.6.4 2D rigid transformation

We now need to translate and rotate this overall curve, so as to minimize the fitting error to the input curve. We cast this as a weighted least squares minimization problem as follows: Sample a corresponding set of n points from the canonical clothoid spline, using the arc length positions from the input polyline. Define the set of corresponding n canonical points with an S superscript: $\left\{(x_0^S, y_0^S), \ldots, (x_{n-1}^S, y_{n-1}^S)\right\}$. Figure 2.6 shows the clothoid spline in its canonical form, with the corresponding set of n points sampled along it.

The goal is to minimize the sum of 2-norm distances between corresponding pairs of points with a rotation matrix R and translation vectors T and T^S:

$$\sum_{i=0}^{n-1} \left\| R\left(\begin{pmatrix} x_i^S \\ y_i^S \end{pmatrix} + T^S \right) + T - \begin{pmatrix} x_i \\ y_i \end{pmatrix} \right\|_2 . \tag{2.12}$$

Our approach is based on the solution for shape matching shown in [32]. The optimal

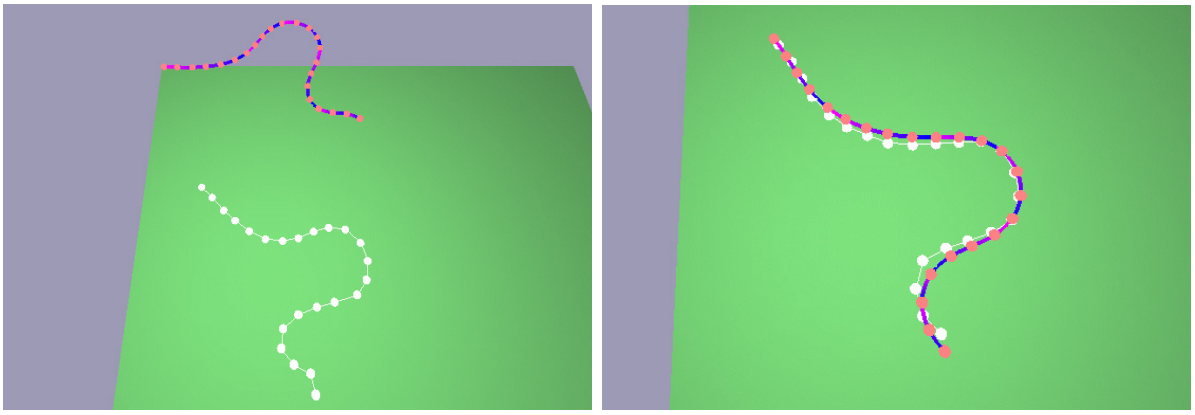

Figure 2.6: The points $\{(x_0^S, y_0^S), \ldots, (x_{n-1}^S, y_{n-1}^S)\}$ on the composite curve in pink must undergo a rigid 2D transformation to match the sketched input curve in white (left). The result of the transformation (right).

translation vector is given by aligning the weighted centroids of both sets of points:

$$T^S = \frac{1}{\sum_{i=0}^{n-1} w_i} \begin{pmatrix} \sum_{i=0}^{n-1} w_i x_i^S \\ \sum_{i=0}^{n-1} w_i y_i^S \end{pmatrix}, \tag{2.13}$$

$$T = \frac{1}{\sum_{i=0}^{n-1} w_i} \begin{pmatrix} \sum_{i=0}^{n-1} w_i x_i \\ \sum_{i=0}^{n-1} w_i y_i \end{pmatrix}, \tag{2.14}$$

where each weight w_i specifies the relative importance of the corresponding pair of points $(x_i, y_i), (x_i^S, y_i^S)$ in the fit (see Figure 2.2b,2.7).

Define sets of points which are the relative locations to the centroids $q_i = (x_i^S, y_i^S) - T^S$ and $p_i = (x_i, y_i) - T$. To determine the rotation matrix, the problem is relaxed to finding the optimal linear transformation A, where we want to minimize $\sum_{i=0}^{n-1} w_i (Aq_i - p_i)^2$. Setting the derivatives with respect to all coefficients of A to zero yields the optimal transformation

$$A = (\sum_{i=0}^{n-1} w_i p_i q_i^T)(\sum_{i=0}^{n-1} w_i q_i q_i^T)^{-1} = A_{pq} A_{qq}. \tag{2.15}$$

A_{qq} can be ignored as it is symmetric and does scaling only. The optimal rotation R is then the rotational part of A_{pq}, found by a polar decomposition $A_{pq} = RS$, where $S = \sqrt{A_{pq}^T A_{pq}}$, and so $R = A_{pq} S^{-1}$.

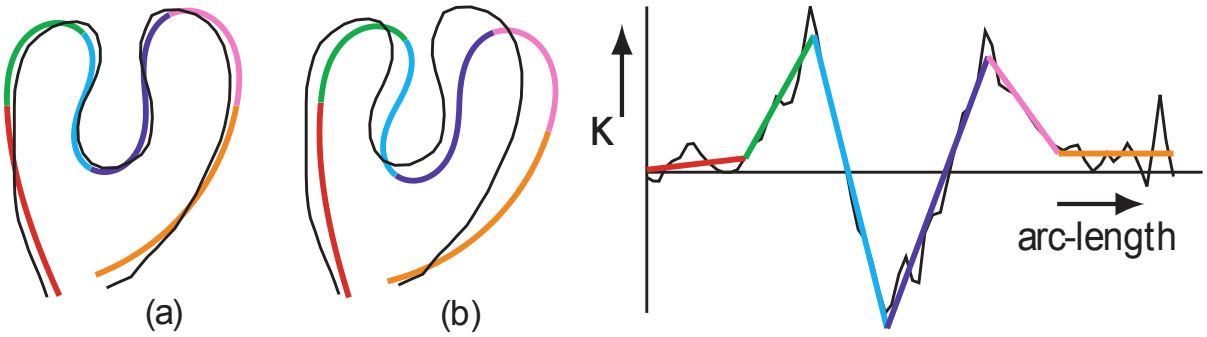

Figure 2.7: Equal weights for all sample points (a), and only weighting the end-points (b), result in different rigid transforms for the same composite curve segmentation on the right.

If the matrix $A_{pq}^T A_{pq}$ is near-singular, instead the vector from the start to end point of the sketched curve given by $(x_{n-1}, y_{n-1}) - (x_0, y_0)$ is used, and its arctangent provides an estimate of the best angle of rotation.

2.7 Fitting extensions

2.7.1 Sharp corners (G^1 discontinuity)

Many sketching applications require the user to only sketch smooth strokes and handle corners by requiring two separate smooth strokes to end at a corner. Such a restriction adds a cognitive burden on the user and can be disruptive to the sketching process. To automatically handle sharp corners in our framework, we first need to detect points of G^1 discontinuity in the sketched stroke. Observe that such sharp corners appear as large spikes in curvature space (see Figure 2.8). Statistical approaches to curvature estimation [21] are able to robustly filter out similar spikes that may arise from noise and outliers in the sketched stroke. Simple thresholding of points with both high curvature and high variation in curvature yields our set of sharp corners. We then force a segment break at all sharp corners and flatten the curvature spike from the set of curvature points so as

not to bias the subsequent fitting process. The final segmentation is a further refinement of the segments induced by the sharp corners. We now treat the composite curve as having limbs that articulate at the corners. We fit this curve by finding the optimal transformation for the first limb as in Section 2.6.4. The translation of each subsequent limb is now constrained but its optimal rotation may once again be solved as in Section 2.6.4. We use a higher weight for the corner points in this fitting to better match the user sketched corners. While a more globally optimal set of transformations may be sought, we find this greedy approach to work well in practice and the resulting curves closely match the input sketch.

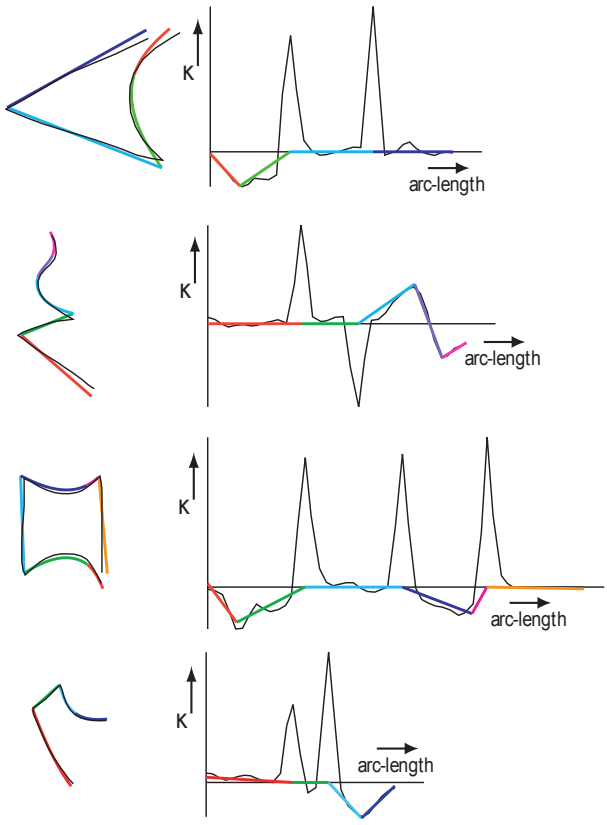

Figure 2.8: Curves with sharp corners or G^1 discontinuities are automatically handled by our fitting approach (curvature profiles on right).

2.7.2 G^3 continuity

It is also easy to extend the given piecewise construction to produce G^3 continuous curves. Following the curvature space linear segmentation step, between each pair of segments, we can round the corners in curvature space by performing a local linear blend. For each segmentation point (x_i^P, y_i^P) that is the endpoint of two segments, blending occurs within a window of distance d around x_i^P. A set of blended samples can be constructed for this window, sampling with a value s such that $0 < s < 1$, each sample point is given by

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x_i^P + d(2s-1) \\ y_i^P - m_1 d(s^2 + 2s - 1) + m_2 d s^2 \end{pmatrix} \tag{2.16}$$

where $m_1 = \frac{y_i^P - y_{i-1}^P}{x_i^P - x_{i-1}^P}$ and $m_2 = \frac{y_{i+1}^P - y_i^P}{x_{i+1}^P - x_i^P}$ are the slopes of the curvature space line segments.

Segmentation point (x_i^P, y_i^P) is then replaced by the generated set of blended samples. The samples in this region finitely approximate a quadratic function with a continuous derivative.

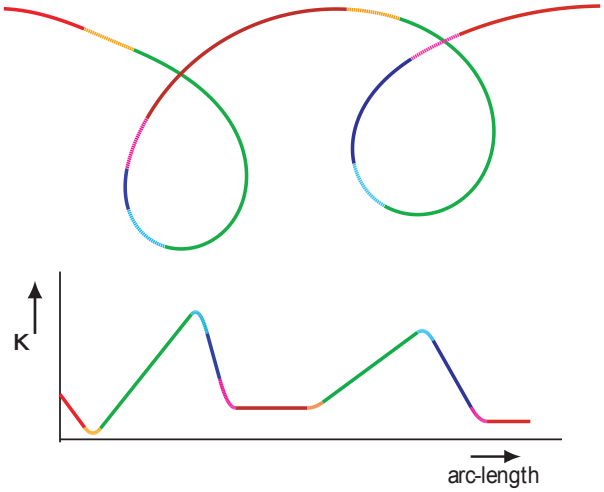

Figure 2.9: A G^3 continuous curve obtained by local linear blending of adjacent clothoid segments.

2.7.3 Geometric interpolation

While our approach is tailored towards constructing fair curves that approximate sketch strokes, it may be desirable to interpolate given geometric constraints. Performing such interpolation strictly using clothoids is sometimes impossible [29]. Instead within our system we simply use quintic Hermite splines that we locally blend into the curve generated by Section 4 to interpolate arbitrary points with G^2 continuity (see Figure 2.10). We note that while G^2, the use of Hermite splines can destroy the fairness properties of the overall curve.

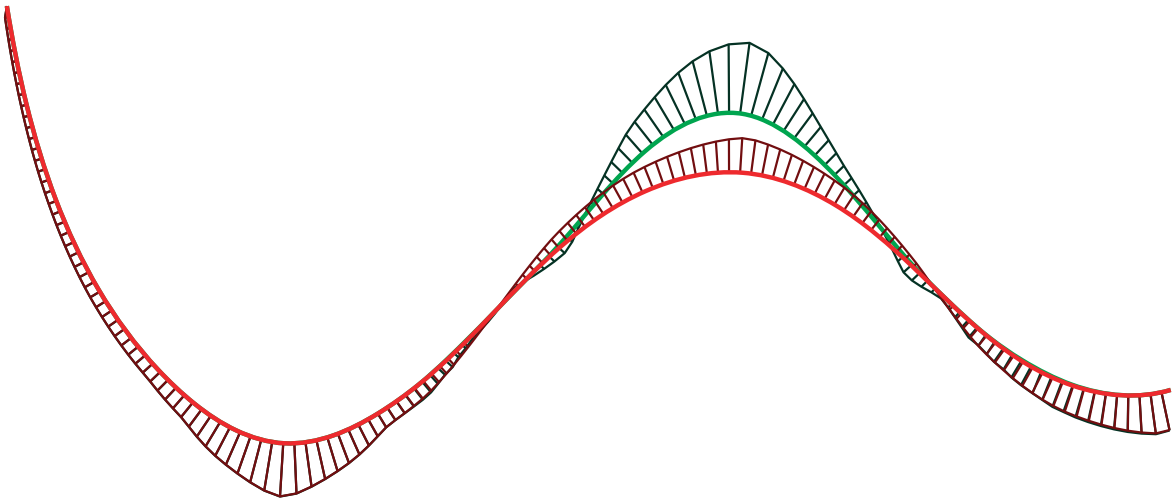

Figure 2.10: The curve composed of clothoid segments (red) in Figure 2.1 is edited in the middle using a quintic spline (green) with G^2 continuity but with degradation in quality of the curvature plot.

2.8 Sketching Applications

We have implemented our approach both as a simple sketching interface capable of generating a wide variety of aesthetic curves (see Figure 2.11) and as part of *Drive*, a comprehensive system for sketch-based road network design (see Figure 2.3). While *Drive*

has a number of sophisticated features specific to the conceptual sketching of a driving experience, it is built around a simple interface for sketching clothoid curves. The framework naturally favours lines, circular arcs and clothoids which are common in road design and also desirable from a steering standpoint. In our system users can prescribe a preference for more or less segments by directly specifying E_{cost}, or by specifying an error of fit, in which case the system iteratively uses a lower E_{cost}, if the error of fit is above the given tolerance (see Figure 2.5). Users can also oversketch parts of curves as one might expect, in which case the track is globally refitted or blended in locally using a spline (see Figures 2.10, 2.12).

A proof-of-concept demo application was implemented in C++ using OpenGL and GLUT. It was tested on 2 systems: an AMD Athlon64 3000+ 2GHz and an Intel Xeon 2.2GHz, both with 1 GB RAM, and in both cases curves consisting of hundreds of points are generated at interactive rates. The most computationally costly step in our approach is determining the curvature space segmentation. As a dynamic programming algorithm is used to find a global minimum solution, the number of points of the input polyline determine the number of rows and columns of the cost matrix M, leading to quadratic growth in the number computations required.

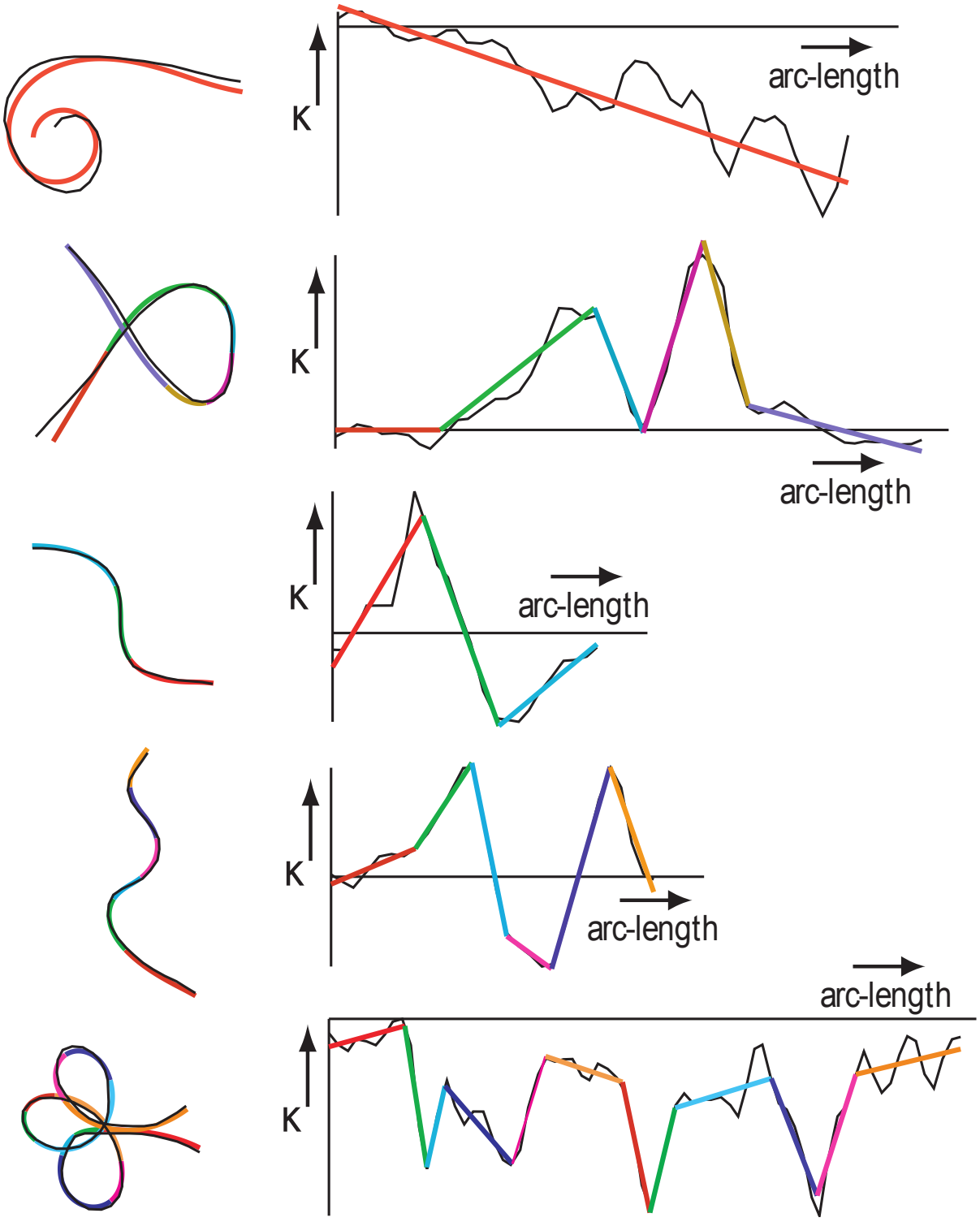

Figure 2.11: Gallery of curves sketched using our system (left) with corresponding curvature profiles (right).

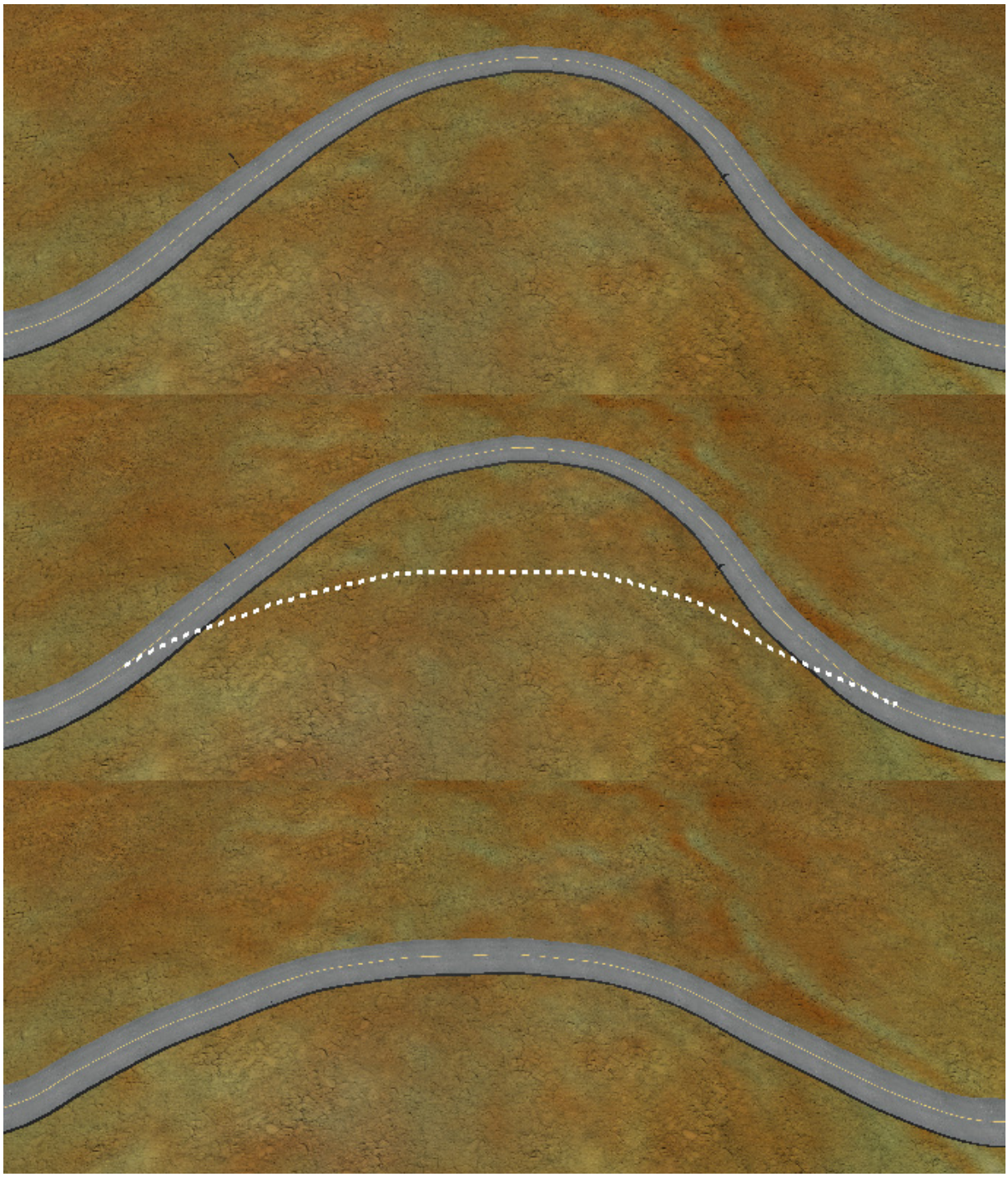

Figure 2.12: Oversketching to edit curves.

Chapter 3

Drive - A Comprehensive Road Design System

3.1 Related Work

Sketch-based interfaces generally have a quick-and-dirty feel to them that is well-suited to ideation and conceptual prototyping. Relevant to this paper a number of compelling systems have been proposed for 3D shape modeling [48, 20, 44, 39, 34, 9], camera motion along a path [19], spatial layout [1, 47], interface design [25], animation [10, 43], the sketching of flora and fauna [2] and architecture (*Google SketchUp*). The area of sketch-based path design, however, is largely unexplored.

Various aspects of our system draw upon existing research. Our interface is largely driven by a single lasso-menu and quick selection-action phrasing [1]. We draw curves directly projected onto a terrain much the same way that shape modeling systems [20, 34, 22] project a sketched stroke onto underlying geometry.

Lifting these curves in 3D off the projected geometry or creating non-planar curves in general is a difficult problem since sketched strokes are inherently 2D in the view-plane. Common solutions to this problem is to resolve in 3D, strokes sketched in multiple views

[6, 23]. Multi-view approaches while mathematically straightforward are suboptimal for a user who has to mentally deconstruct a 3D curve into multiple disconnected views. We alleviate this problem using a break-out lens that provides the capability of multiview editing but in context of the current curve and its surrounding environment. While lenses for zooming have existed for some time [5] in interactive visualization, we believe this is the first approach to both view manipulation and editing via a lens using nonlinear projection [8].

Occlusion has been exploited to disambiguate the depth of 3D organic shapes [9]. In [9] visible contours of 3D objects are processed to build 2D panels that are inflated into 3D shapes like Teddy [20]. The occluded contours result in overlapping panels whose depth is solved for by optimizing curvature, orientation and distance of a panel axis from the sketch plane. Since our paths are the same width, we simplify drawing by using a single stroke to represent the spine of the path. We also satisfy occlusion constraints by using an optimization formulation better suited to path design than 3D shape modeling [9] (see Figures 3.4, 3.5).

Interaction techniques commonly represent continuous curves as densely sampled polylines. Geometric properties for these curves, however, need to be imposed by the curve creation and editing technique [15, 44]. One such geometric property is *fairness* [13], that attempts to capture the visual aesthetic of a curve. Fairness is closely related to how little and how smoothly a curve bends. For planar curves it has been described as curvature continuity G^2, with a small number of segments of almost piecewise linear curvature [13]. The family of curves whose curvature varies linearly with arc-length are known as clothoids. Clothoids have been the subject of prior research in CAD and transportation design as transition curves smoothly connecting two curve segments [29, 26] or a spline where every three consecutive points are fit with a parabola-like clothoid segment [45]. A discrete formulation of clothoid using nonlinear subdivision has also been proposed [16]. In this paper, we exploit the fairness properties of clothoids to fit 2D

strokes for sketch-based applications. Our approach automatically favours precise line and circular-arc segments which are desirable for transport path layout. A further advantage of fitting analytic curve segments like splines or clothoids over discrete smoothing methods is that the curve can be regenerated at arbitrary resolution.

Finally, our goal is not simply path network creation but a conceptualization of the entire driving experience, in which paths are an integrated part of the environment on which they are laid out. Programs such as *Google Earth* provide compelling interfaces for visualizing 3D environments. We handle the construction of this evolving environment intelligently: creating a path automatically defines a cut-and-fill corridor on the terrain, removes foliage on the path, and constructs bridges, tunnels, path crossings and signage as dictated by the evolving landscape and path networks.

The actual drive through is accomplished with a select-and-play action, allowing a user to animate the drive of a selected section of path. There has been research on authoring virtual flythroughs in the context of scene visualization [46, 42] or product design [24, 4]. We also provide a simple select-and-time action to alter the pacing of a drive along the path. We are thus able to quickly author the rough timing of objects or cameras animating along motion paths. Such functionality is more precise and complex to control in animation systems like *Maya*.

3.2 Design Goals

Our overreaching goal of rapidly conceptualizing a 3D driving experience through sketching helps define a number of smaller design guidelines.

- A general problem with sketch-based 3D applications is a philosophical disconnect in that view navigation for 3D scene understanding is critical while traditional sketching is inherently a 2D task from a fixed viewpoint. We aim towards maintaining the focus on sketching and minimizing interactive view navigation.

- In addition to transport layout designers, we design this application for novice users without exceptional artistic skills: for building custom tracks to race on in a gaming environment or for exploring options in designing a path through a thicket from their cottage to a nearby lake. The system should thus be easy to use with limited instruction and generally fun to play with.

- While most sketch-based applications attempt to leverage the many degrees of freedom of a pen and tablet, we aim to design an application that is equally operable by a mouse, with minimal use of buttons or mode switching.

- As with most concept design applications we would like to keep user focus on design with a maximal sketching surface and without any distractions or cognitive overhead from the UI.

- Finally given the conceptual nature and specific domain of our design problem, we would like to make many intelligent inferences from sketched strokes in the context of the evolving environment, to maximize the visual impact of each sketched stroke (see Figure 1.3).

3.3 Basic Interaction and Visualization

The single essential instruction to a user is that sketching open curves creates or edits paths, and a closed (self-intersecting) curve lassos a region of selection as well as invokes a radial menu [1] that is sensitive to the lasso selection (see Figure 3.1). This design decision is made on the simple observation that closed curves are rare in transportation path design (barring roundabouts). Most users figure out themselves that roundabouts or self-intersecting paths can be easily created anyway by connecting two or more open curves. This clean distinction between design and command space is trivial to learn and places virtually no cognitive overhead on a user.

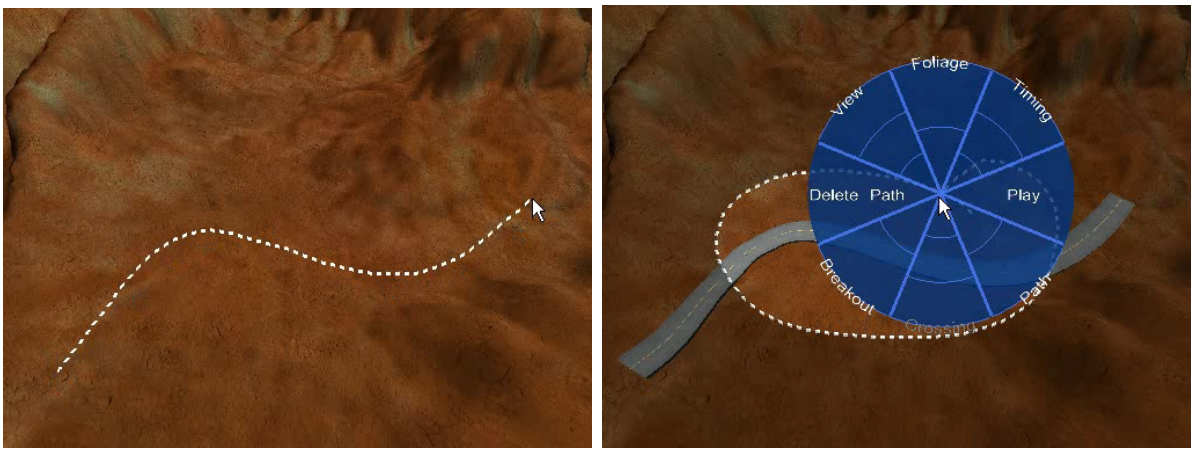

Figure 3.1: Open curves create and edit paths (left) while closed curves select and invoke a lasso-menu (right).

3.3.1 Lasso-menu

As described and shown in Figure 3.1, our lasso-menu is an 8 item radial menu that is context sensitive to the lasso selection. A number of menu items have up to 3 sub-options defined as concentric wedges of the item. Hovering over a sub-option describes its text dynamically in the middle of the menu avoiding menu clutter or text at awkward orientations. For many actions such as path deletion or foliage change, hover also provides the user a preview of the action. The lasso-menu can also be invoked by a press-and-hold action.

3.3.2 Camera control

As stated in our design goals we would like promote single view sketching and minimize the time spent on view navigation, which is the single most frequently used control in most 3D applications. Fortunately terrains are typically height-fields and largely visible from a birds-eye view, reducing the need for constant view manipulation while sketching. Given the large scale of terrains, however, it is necessary to pan and zoom to access regions of the terrain at an appropriate resolution. Users can frame a region by lassoing it and

selecting one of Birds-eye, Mid-way or Close-up sub-options from the lasso-menu, that are hand-crafted to not only magnify but also tumble the camera so that the Close-up view from the ground is almost orthogonal to the aerial Birds-eye view.

Interestingly enough, newer generations of users familiar with 3D applications sub-consciously expect typical tumble, pan and dolly camera tools. We thus also support the camera controls found in popular 3D systems such as *Maya*, where the ALT key enables the camera, and the left, middle and right mouse buttons tumble, pan and zoom respectively.

3.4 Path Creation and Editing

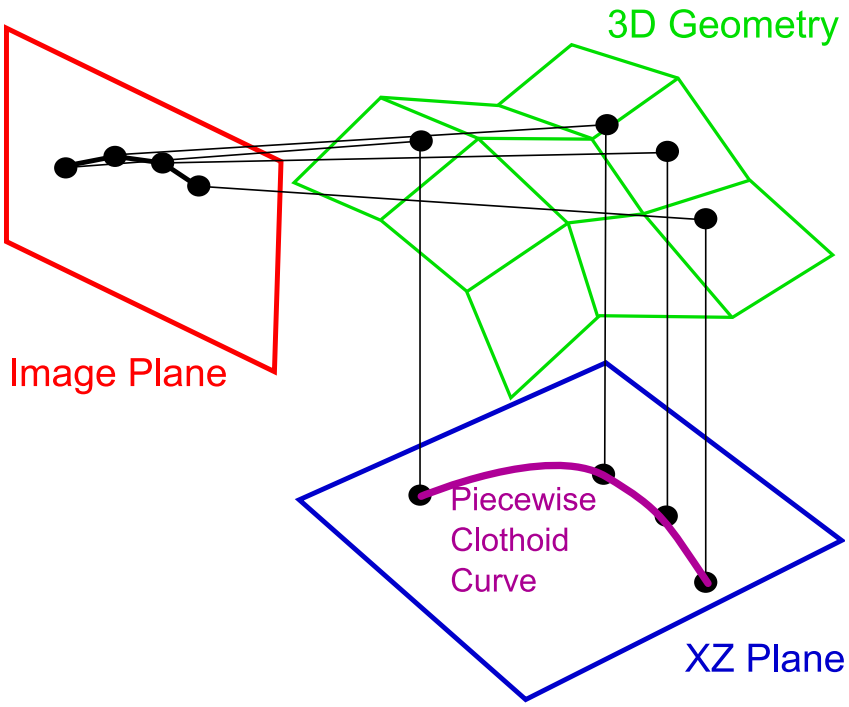

Figure 3.2: The sketched curve in the image plane is projected onto the terrain geometry, then projected onto the XZ plane where it is fit in 2D using line, circular-arc and clothoid segments.

Open curves are used for path creation and editing. Users sketch strokes as 2D

polylines in the view plane. Each of these points is then projected onto a 3D heightfield terrain representation, to create an unfiltered 3D curve representing a path. We resample the 3D curve on the terrain by inserting and deleting points to ensure a reasonably even arc-length sampling. This 3D curve is then fit using clothoids in 2D in the XZ-plane, ignoring the height component Y (see Figure 3.2). The fitted curve comprising line, circular-arc and piecewise clothoid segments is ideally suited to path design. This curve is discretely sampled and projected back to the 3D geometry to define the spine of the final path that we represent using a Catmull-Rom spline along which path geometry is deformed.

3.4.1 Clothoid fitting

The input to our clothoid fitting is a 2D polyline and the output is a sequence of line, circular-arc and clothoid segments that best approximate the input curve. As mentioned earlier, clothoids are characterized by a curvature that changes linearly with arc-length, forming a spiral shape. We thus first fit a piecewise linear approximation to the discrete curvature of the stroke as a function of arc-length, with control over the tradeoff between fitting error and the number of linear pieces (see Figure 2.2(a)). The start and end curvature values of each linear piece uniquely determine a line, circular-arc or clothoid curve segment. These segments further assemble together uniquely with G^2 continuity into a single composite curve. The next step involves determining a single 2D rigid transform that aligns this composite curve with the sketched stroke to minimize the error of the stroke from the transformed curve (see Figure 2.2(b)). We are able to solve for this transform efficiently by formulating the error as a weighted least squares optimization problem.

3.4.2 Path editing

Paths may be extended or edited by oversketching [3]. End-point proximity and end-tangent alignment of the oversketched path to existing paths is used to infer whether a new path is created or if an existing path is extended or edited (see Figure 3.3). Clothoid refitting is applied to edited paths. Selected regions of paths can be deleted using the lasso-menu.

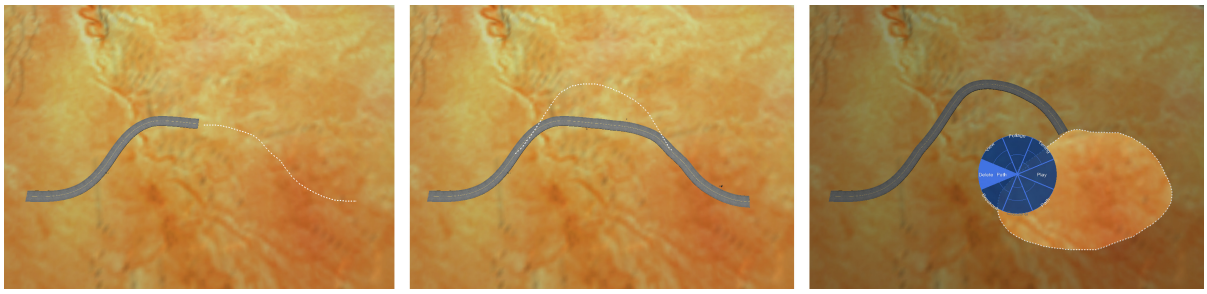

Figure 3.3: Users extend (left) or edit (middle) a path by oversketching, and delete segments (right) using a lasso-menu.

3.4.3 Crossing relationships

Crossing relationships between sketched paths are automatically determined based on the observation that an overpass occludes paths under it from an aerial view. Users can specify such occlusion by a small break in one of the sketched paths at a crossing (see Figure 3.4). The order in which the solid and broken path is sketched is not important. Two unbroken paths crossing each other indicates an intersection. Distinguishing a path that is broken at a crossing from two different paths is performed in a manner similar to path extension using end-point proximity and end-tangent alignment. Users can always change the crossing relationship by selecting it with a lasso and choosing between the over, under and intersect crossing sub-menu options. Hovering over the suboptions previews the new crossing.

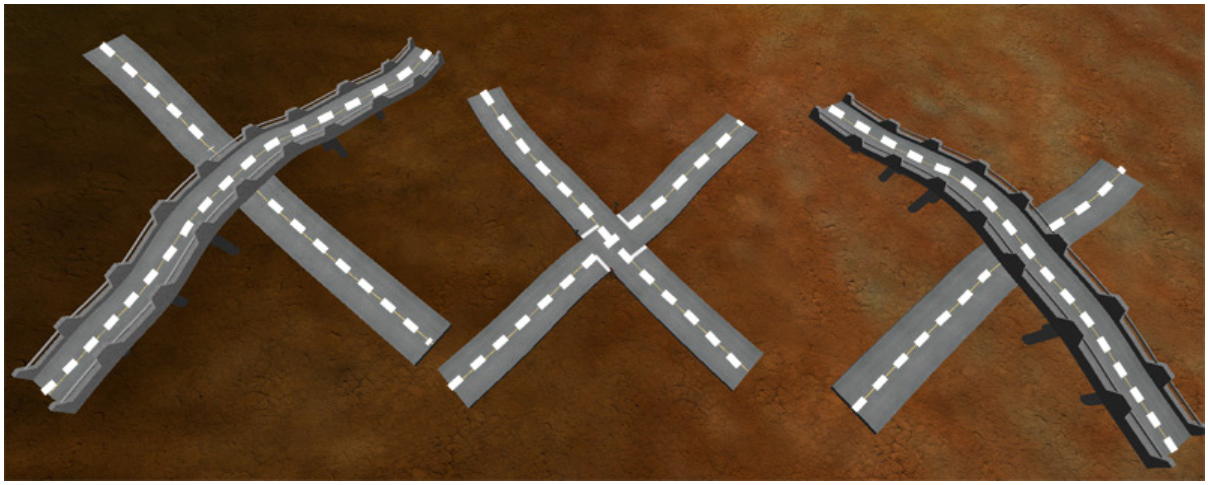

Figure 3.4: Breaks in the sketch stroke indicate the over/under occlusion relationship between intersecting paths. A small break in a path at a crossing indicates occlusion from above and makes the other path pass over it. Unbroken paths indicate an intersection.

3.4.4 Complex crossings

A simple overpass is easily handled using a typical height clearance and grade for path elevation. Overpasses close to each other on the same road or more complex crossing relationships (see Figure 3.5) require a more sophisticated approach to determining the height of path from the terrain. Our problem is to define a height value for a number of points along a path where a crossing relationship has to be satisfied. To further constrain the problem we minimize the overall height from the terrain and the variation of height (path grade) in real-time using nonlinear energy optimization.

The input to our path height optimizer are a set of n crossing points with height y_i relative to the terrain at increasing arc-lengths l_i. We also have k height inequality relationships that index into the set of crossing points: $\{(a_0, b_0), \ldots, (a_{k-1}, b_{k-1})\}$, where $\forall i \in \{0, \ldots, k-1\}$, $a_i, b_i \in \{0, \ldots, n-1\}$, $a_i \neq b_i$ and $y_{a_i} > y_{b_i}$.

We define objective functions for height from the terrain as

$$heightCost = \sum_{i=0}^{n-1} ||y_i||^2, \tag{3.1}$$

and path grade as

$$gradeCost = \sum_{i=0}^{n-1} \left[\frac{|y_{i+1} - y_i|}{l_{i+1} - l_i} \right]^2 . \tag{3.2}$$

For points in a height inequality relationship, their objective function is inversely proportional to their difference in height and infinite for a constraint violation. In other words,

$$relationCost = \sum_{j=0}^{k-1} \begin{cases} \left[\frac{1}{y_{a_j} - y_{b_j}} \right]^2 & \text{if } y_{a_j} > y_{b_j} \\ \infty & \text{otherwise} \end{cases} . \tag{3.3}$$

We minimize the cost function $heightCost+gradeCost+relationCost$ using a steepest descent method with backtracking line search. The gradient of this function is found using finite differencing with respect to each dimension. Once we have normalized height values defined at crossing points we scale them by a specified clearance and solve for height along the path using Catmull-Rom spline interpolation of crossing point heights.

We are thus able to sketch and handle an arbitrarily complex ordering of crossing relationships. Note however, that if the relationships are close to a common intersection as in Figure 3.5 (left), reordering them using the lasso-menu is currently a problem since it is difficult to select a single pair of crossing paths.

We have now defined a simple path creation and editing formulation with automatic determination of height from the terrain, based on path crossing relationships. We have yet to address how the elevation profile of these roads can be conveniently visualized and furthermore edited.

3.5 Break-outs

In traditional concept sketching and engineering visualization, static alternate viewpoints known as break-out views, are used to illustrate local parts of a scene [11]. We are motivated by these views, not only for rapid local 3D visualization of paths but also path height editing.

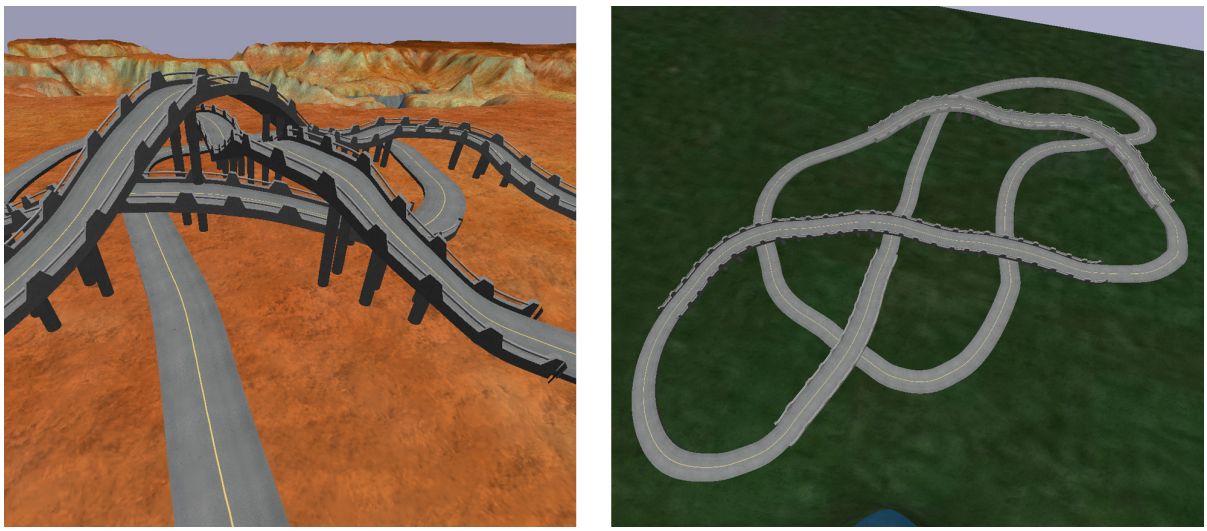

Figure 3.5: A complex multi-level crossing (left) and a closed path with many crossings (right).

3.5.1 Break-out view

The user selects a region of a path and invokes a break-out view from the lasso-menu (see Figure 3.6). We use the start and end points of the selected path segment to determine a 3D segment, called the break-out axis. This axis determines the break-out view as if the user was standing by the side of the path facing perpendicular to the break-out axis. The break-out view is then generated by an animated transition within a window the shape of the lasso selection, that is translated to a default location by the side of the path (see Figure 3.6). The break-out view, like any other GUI window can be moved around in the system and killed as needed. Multiple break-out views can also co-exist.

We now transcend the use of break-out views for rapid visualization by allowing a user to oversketch within the break-out view, and edit path elevation in the same fashion they oversketch to edit paths on the terrain. The oversketched path must first be rotated, such that the break-out view plane becomes perpendicular to the ground-plane (see Figure 3.7).

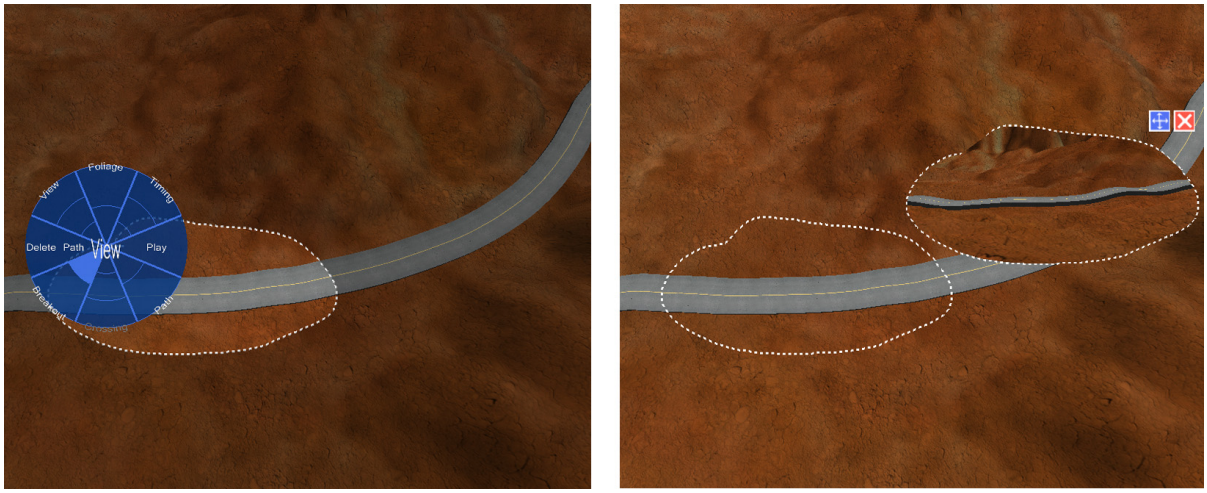

Figure 3.6: Users select a region of the path and invoke a break-out view from the lasso-menu.

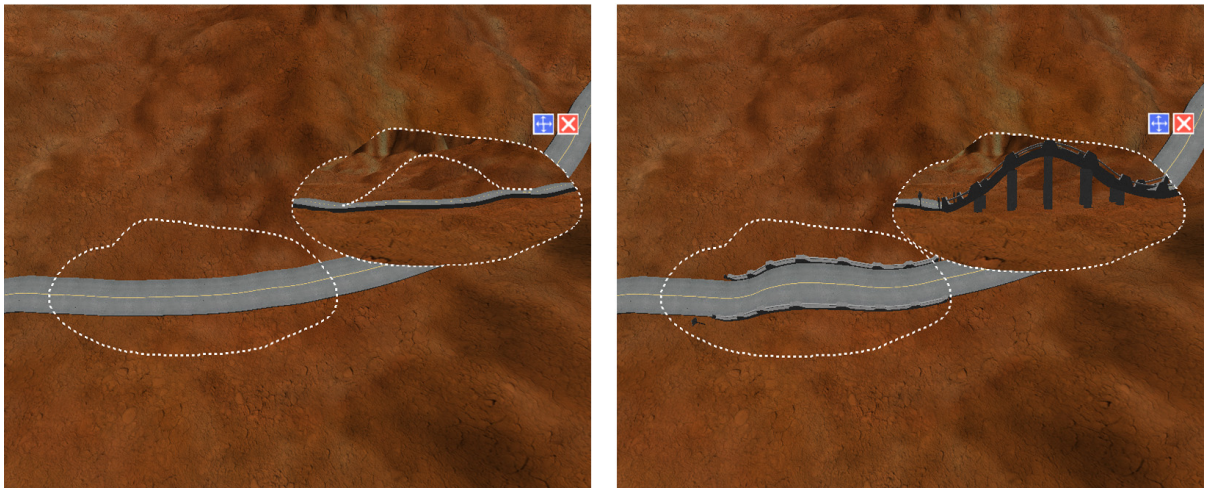

Figure 3.7: Oversketching to edit path elevation in a break-out view.

3.5.2 Break-out lens

The oversketching to edit path elevation in the break-out view is an example of multiview 3D curve sketching. While this approach is mathematically succinct, users find it difficult to draw 3D curves in general using disconnected multiple views. This is less evident in our case, since path curves have a clear dimensionality separation as 2D curves on a ground-plane with a path elevation. We can, however, address the break-out view disconnect by formulating it as a break-out lens (see Figure 3.8).

The break-out lens is similarly invoked with a selected path from the lasso-menu. The lens consists of two concentric circles. The interior of the inner circle behaves like a break-out view and the region between the two circles provides a continuous view change between the current camera view and the interior break-out view [8]. The lens has four controls, to move the lens, control the inner and outer radii, and to define the angle of rotation θ around the break-out axis, of the break-out view from the current camera view. If the current camera view matrix is C and break-out view rotation is R_θ, the view warp is accomplished by deforming points inside the inner circle by $R_\theta C^{-1}$. The deformation of points in between the two circles $(R_\theta C^{-1})^t$ smoothly decays radially to the current view as t goes from 1 at the inner radius to 0 at the outer radius.

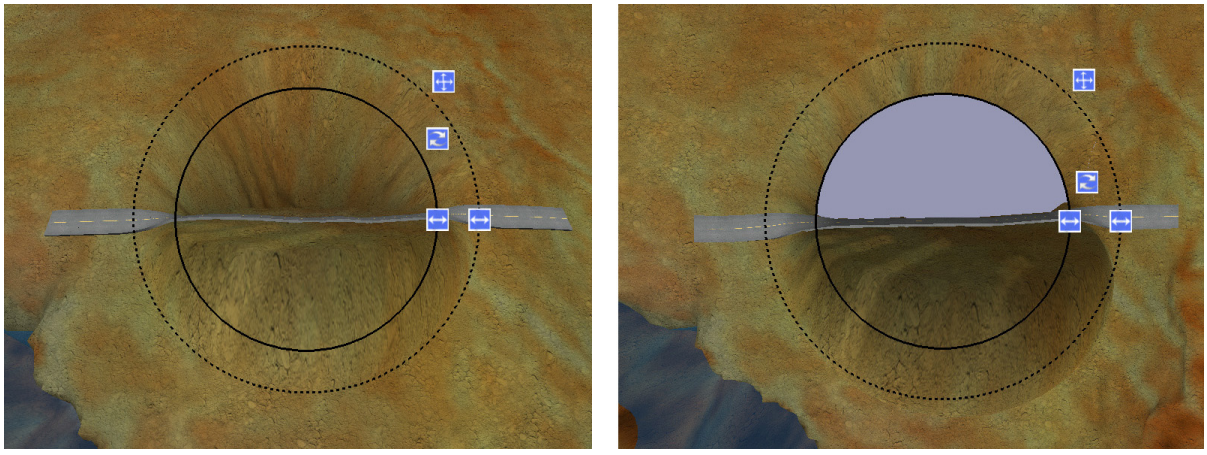

Figure 3.8: The break-out lens provides a continuous local transition to a break-out view (left) rendered with a background plane (right).

The break-out lens also allows oversketching to edit path elevation. Here too the oversketched path should be rotated by $90^o - \theta$ prior to the edit. The continuous context of the break-out lens improves the usability of multi-view sketching and can be used to sketch curves and perform silhouette based 3D deformations [41, 34] from a single view (see Figure 3.9).

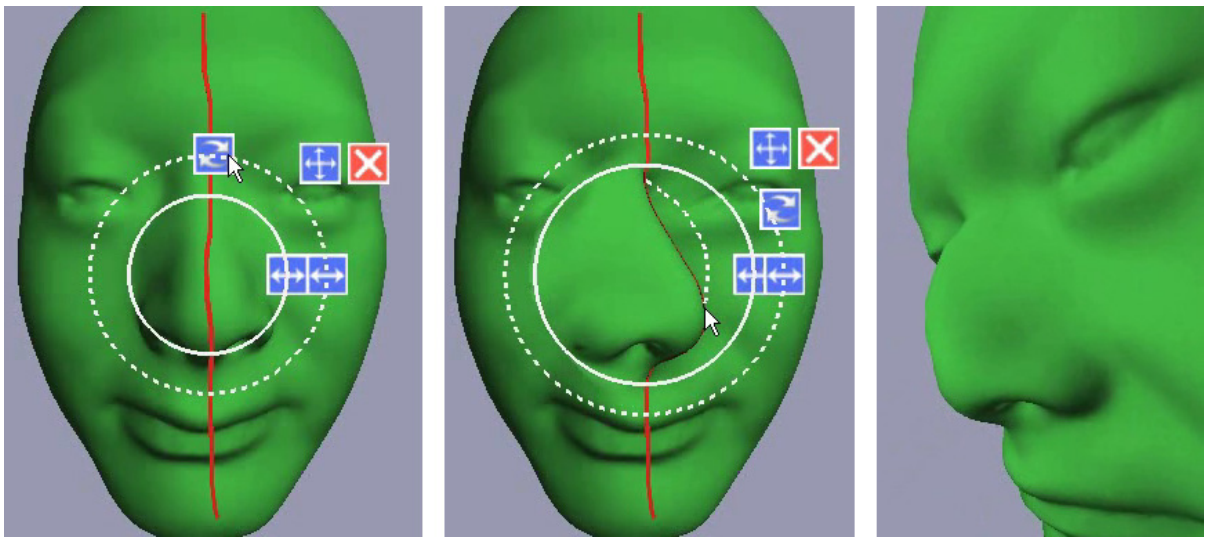

Figure 3.9: A curve is sketched and oversketched with a break-out lens to define a wire deformation of the nose.

3.5.3 Rendering the background plane

A major difference between the break-out view in Figure 3.6 and the break-out lens in Figure 3.8 (left), is the lack of a horizon and background in the lens. This is because the terrain wraps around continuously obscuring the background. We can address this by rendering a background plane on the far side of the path to simulate a horizon and background. This plane, drawn strictly within the inner radius makes the lens more like a true break-out view on the far side of the path and provides a continuous transition of view on the near side (see Figure 3.8 (right)). The opacity of the background plane is smoothly interpolated with the rotation value θ for a continuous transition (the system

uses values of 30 and 75 degrees to interpolate from transparent to opaque).

3.6 Terrain Sensitive Sketching

The terrain in our system is not just a canvas on which to sketch paths but an evolving environment into which sketched paths are integrated. The paths we create are thus sensitive to the terrain over which they are sketched and appropriately add and remove geometric features to alter the environment.

3.6.1 Roads

In our system a 2-lane road (see Figure 1.3) piece is defined parametrically that can be laid along any given path. One can thus easily replace roads with railway tracks or ribbons (see Figure 3.9), simply by replacing the parametric piece. The terrain under a road is edited to match the road capturing the cut-and-fill needed for road engineering.

3.6.2 Road signs

The current path layout automatically determines an appropriate set of landmarks to place along the paths. Examples of signs automatically generated by our system for a given road network include stop, stop ahead, sharp left/right turn ahead, bump or dip (see Figure 1.3). In addition, for intersections, road markings are generated at stop signs by using a textured road piece.

3.6.3 Bridges, tunnels and support pillars

When a user sketches a path traversing water, edits a path to cut through a terrain or elevate it above a terrain, bridges, tunnels and support pillars are automatically constructed to add visual realism (see Figure 3.10). Support pillars connecting the path to

the terrain below, tunnel lights and bridges, like roads, are defined parametrically and can be readily customized.

Figure 3.10: Bridges, support pillars (left) and tunnels (right) are automatically constructed to integrate paths into the evolving environment.

3.6.4 Foliage

We use foliage as an example of paths interacting with arbitrary terrain attributes. The creation of paths automatically removes any foliage on the paths. At the same time the foliage is often planted by the side of paths to improve the driving experience. Foliage in any selected region may be made more or less dense using a random distribution through the lasso-menu, while avoiding any paths or water bodies (see Figure 3.11).

3.7 Timing and Playback

Visualizing the navigation experience along paths is an essential aspect of our system. Selecting a portion of a path and invoking Play from the lasso-menu navigates a vehicle down the selected path. The camera view during playback can be selected from a context sensitive lasso-menu (see video) to choose from a set of meaninigful predefined views or controlled by the user in a freeform manner. If the user select portions of paths with

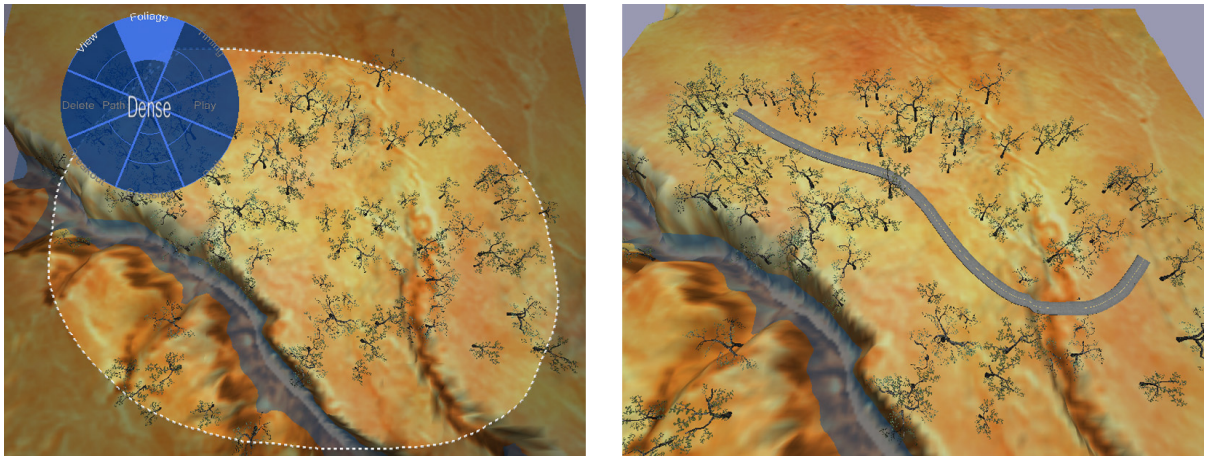

Figure 3.11: Foliage may be added or removed using the lasso-menu.

intersections, the straightest option is followed. Navigation speed is a constant by default. The user can alter the timing along the path by selecting a region of the path and choosing a speed from the timing menu item. Timing control causes speed limit signs to be posted appropriately, and the speed between different regions is automatically interpolated.

3.8 Implementation

Our path design system *Drive* is implemented in C++ using OpenGL and GLUT. It was tested on 2 systems: an AMD Athlon64 3000+ 2GHz and an Intel Xeon 2.2GHz, both with 1 GB RAM. Terrains are represented as 1024x1024 textured grids rendered with level of detail. Path pieces and environmental geometry primitives comprise 100-200 triangles and typical scenes in the video have tens of thousands of faces representing geometry authored within our system.

All aspects of creation and rendering in our system: sketching, clothoid fitting, path crossing resolution, view warping and path editing are performed in real-time.

3.9 Discussion

Our system was initially aimed at custom path design for virtual game environments. Early conversations with practicing landscape architects, however, made us realize both the broader application space of conceptual path design and the lack of any existing systems that address it. As landscape architects have substantial training in engineering related fields such as road design [28, 14, 27], these early conversations and the meetings which followed were valuable, to both gain knowledge and understanding of toolsets currently in use in their profession, as well as to have people who design professionally experiment with and provide input on the system.

We first evaluated our system informally by providing a brief set of instructions and the system to six people with mixed computer graphics skill. In general, the response was positive and the surprise bonuses of terrain sensitive geometry kept the users engaged. Users were able to discover most of the functionality without being told, within 5 to 10 minutes of use. No comment was made about the lasso-menu which we see as a sign of a good inconspicuous UI. The feedback with respect to all our design goals was positive, though users familiar with CG applications, persistently navigated the scene with the 3D camera controls by habit, rather than using the menu camera controls. Half of the individuals who used the system did not use the over/intersect/under sketching convention, instead drawing all crossings as intersections and then modifying them using the crossing menu commands.

We also evaluated our application with University of Toronto landscape architects John Danahy and Robert Wright, who felt the system had potential for real use in conceptual transportation layout, despite some criticisms. The first criticism was the use of 32-bit floating point values, in professional applications where numerical precision is of paramount importance 64-bit values are used. Also, in practice, heightmaps are not stored in conventional image formats, since a discrete-valued 8-bit channel of an image is less accurate than a floating point value. They suggested the inclusion of controls to

manipulate the centreline of created paths in a precise manner, and additional terrain attributes that define the constituents of the earth. The type of earth constrains how cutting and filling is performed, and defines the maximum angle at which the terrain can taper downward at the sides of the path in order to reliably support it, and is therefore an important aspect of road design which the system does not address. Finally, they suggested the inclusion of the commonly-used formulas that govern road design, e.g. minimum radius of curvature for a specific speed, maximum acceptable grade.

Chapter 4

Conclusion

We have presented an approach to fitting sketched strokes with a sequence of line, circular-arc and clothoid segments. We empirically find that clothoids tend to capture sketched strokes well and usually only a few (less than five) clothoid segments can capture a stroke with screen resolution fidelity. Figure 2.1 also shows our fitting approach to be an appealing alternative to current approaches to stroke fairing such as Laplacian smoothing or cubic spline fitting, particularly when a good approximation is more desirable than precise interpolation of any given point. If cubic splines are necessary for downstream use, we find that fitting the clothoid curves provides better fairness than directly fitting the input stroke. Designers often work with characteristic shape palettes defined by French curves [40], or predefined pieces of track. In the future we hope to explore the use of intrinsic splines such as clothoids for both palette representation and shape editing.

We have also presented a novel system for the concept sketching of path layouts. Our system integrates a number of new ideas, each of which is also applicable to a wider range of applications. Clothoid fitting is shown to be good for creating fair curves from sketched strokes, especially when curve smoothness takes precedence over precise point interpolation. Arbitrarily complex self-occluding 3D curves can be effectively created using our energy optimization of crossing relationships. The break-out view and lens

are likely to find use in general shape modeling and visualization (see Figure 3.9). The locality of the break-out lens in particular, guarantees that any region can be seen from an arbitrary viewpoint without being obscured by other parts of the scene. Terrain sensitive sketching adds to the appeal of our system allowing users to very quickly author engaging environments. In the future we hope to extend terrain sensitive sketching from curve layouts to area layouts with application to landscaping and urban planning.

Bibliography

[1] Anand Agarawala and Ravin Balakrishnan. Keepin' it real: pushing the desktop metaphor with physics, piles and the pen. In *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 1283–1292. ACM, 2006.

[2] Fabricio Anastacio, Mario Costa Sousa, Faramarz Samavati, and Joaquim A. Jorge. Modeling plant structures using concept sketches. In *NPAR '06: Proceedings of the 4th international symposium on Non-photorealistic animation and rendering*, pages 105–113, New York, NY, USA, 2006. ACM.

[3] T. Baudel. A mark-based interaction paradigm for free-hand drawing. In *Proceedings of UIST 1994*, pages 185–192. ACM, 1994.

[4] Nicholas Burtnyk, Azam Khan, George Fitzmaurice, Ravin Balakrishnan, and Gordon Kurtenbach. Stylecam: interactive stylized 3d navigation using integrated spatial & temporal controls. In *UIST '02: Proceedings of the 15th annual ACM symposium on User interface software and technology*, pages 101–110. ACM, 2002.

[5] M.S.T. Carpendale. Viewing transformations: Perspective, distortion and deformation. In *SIGGRAPH03 Course Notes; Theory and Practice of Non-Photorealistic Graphics: Algorithms, Methods, and Production Systems Presentation*. ACM, 2003.

[6] Jonathan M. Cohen, Lee Markosian, Robert C. Zeleznik, John F. Hughes, and Ronen Barzel. An interface for sketching 3d curves. In *I3D '99: Proceedings of the 1999*

symposium on Interactive 3D graphics, pages 17–21, New York, NY, USA, 1999. ACM.

[7] Patrick Coleman and Karan Singh. Cords: keyframe control of curves with physical properties. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Sketches*, page 60, New York, NY, USA, 2004. ACM.

[8] Patrick Coleman and Karan Singh. Ryan: rendering your animation nonlinearly projected. In *NPAR '04: Proceedings of the 3rd international symposium on Non-photorealistic animation and rendering*, pages 129–156, New York, NY, USA, 2004. ACM.

[9] Frederic Cordier and Hyewon Seo. Free-form sketching of self-occluding objects. *IEEE Comput. Graph. Appl.*, 27(1):50–59, 2007.

[10] Richard C. Davis, Brien Colwell, and James A. Landay. K-sketch: A "kinetic" sketch pad for novice animators. In *CHI '08: Proceedings of the SIGCHI conference on Human Factors in computing systems*. ACM, 2008.

[11] John A. Dennison and Charles D. Johnson. *Technical Illustration: Techniques and Applications*. Goodheart-Wilcox, 2003.

[12] G. Farin. *Curves and surfaces for computer aided geometric design*. Academic Press Professional, Inc., San Diego, CA, USA, 1990.

[13] Gerald Farin, G. Rein, N. Sapidis, and A. J. Worsey. Fairing cubic b-spline curves. *Comput. Aided Geom. Des.*, 4(1-2):91–103, 1987.

[14] Elizabeth E. Fischer, Heidi Hohmann, and P. Daniel Marriott. Roadways and the land: The landscape architect's role. http://www.tfhrc.gov/pubrds/marapr00/landarch.htm.

[15] Tovi Grossman, Ravin Balakrishnan, and Karan Singh. An interface for creating and manipulating curves using a high degree-of-freedom curve input device. In *CHI '03: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 185–192, New York, NY, USA, 2003. ACM.

[16] Li Guiqing, Li Xianmin, and Li Hua. 3d discrete clothoid splines. In *CGI '01: Proceedings of the International Conference on Computer Graphics*, page 321, Washington, DC, USA, 2001. IEEE Computer Society.

[17] Lawrence Halprin. *Notebooks 1959–1971.* The MIT Press, 1972.

[18] M. A. Heald. Rational approximations for the fresnel integrals. *Mathematics of Computation*, 44(170):459–461, 1985.

[19] Takeo Igarashi, Rieko Kadobayashi, Kenji Mase, and Hidehiko Tanaka. Path drawing for 3d walkthrough. In *ACM Symposium on User Interface Software and Technology*, pages 173–174, 1998.

[20] Takeo Igarashi, Satoshi Matsuoka, and Hidehiko Tanaka. Teddy: a sketching interface for 3d freeform design. In *SIGGRAPH '07: ACM SIGGRAPH 2007 courses*, page 21, New York, NY, USA, 2007. ACM.

[21] Evangelos Kalogerakis, Patricio Simari, Derek Nowrouzezahrai, and Karan Singh. Robust statistical estimation of curvature on discretized surfaces. In *SGP '07: Proceedings of the fifth Eurographics symposium on Geometry processing*, pages 13–22, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association.

[22] Levent Burak Kara, Kenji Shimada, and Sarah D. Marmalefsky. Calligraphic interfaces: An evaluation of user experience with a sketch-based 3d modeling system. *Comput. Graph.*, 31(4):580–597, 2007.

[23] Olga Karpenko, John F. Hughes, and Ramesh Raskar. Epipolar methods for multi-view sketching . In *Eurographics Workshop on Sketch-Based Interfaces and Modeling*, pages 167–173, August 2004.

[24] Azam Khan, Ben Komalo, Jos Stam, George Fitzmaurice, and Gordon Kurtenbach. Hovercam: interactive 3d navigation for proximal object inspection. In *I3D '05: Proceedings of the 2005 symposium on Interactive 3D graphics and games*, pages 73–80. ACM, 2005.

[25] Richard A. Kilgore. Silk, java and object-oriented simulation. In *WSC '00: Proceedings of the 32nd conference on Winter simulation*, pages 246–252, San Diego, CA, USA, 2000. Society for Computer Simulation International.

[26] Benjamin B. Kimia, Ilana Frankel, and Ana-Maria Popescu. Euler spiral for shape completion. *Int. J. Comput. Vision*, 54(1-3):157–180, 2003.

[27] Andropogon Associates Ltd. King's College Circle Precinct. http://www.toronto.ca/auda/2005_8_honourable_places_kings.htm.

[28] Jones & Jones Architects & Landscape Architects Ltd. Paris Lexington Road. http://www.asla.org/meetings/awards/awds02/parislexroad.html.

[29] D.S. Meek and D.J. Walton. Clothoid spline transition spirals. *Mathematics of Computation*, 59(199):117–133, July 1992.

[30] Even Melhum. Nonlinear splines. *Computer Aided Geometric Design*, pages 173–207, 1974.

[31] Henry P. Moreton and Carlo H. Séquin. Functional optimization for fair surface design. *SIGGRAPH Comput. Graph.*, 26(2):167–176, 1992.

[32] Matthias Müller, Bruno Heidelberger, Matthias Teschner, and Markus Gross. Meshless deformations based on shape matching. *ACM Trans. Graph.*, 24(3):471–478, 2005.

[33] Glen Mullineux and Sebastian T. Robinson. Fairing point sets using curvature. *Comput. Aided Des.*, 39(1):27–34, 2007.

[34] Andrew Nealen, Takeo Igarashi, Olga Sorkine, and Marc Alexa. Fibermesh: designing freeform surfaces with 3d curves. In *SIGGRAPH '07: ACM SIGGRAPH 2007 papers*, page 41, New York, NY, USA, 2007. ACM.

[35] A. W. Nutbourne, P. M. McLellan, and R. M. L. Kensit. Curvature profiles for plane curves. *Comput. Aided Des.*, pages 176–184, 1972.

[36] Theodosios Pavlidis. Curve fitting with conic splines. *ACM Trans. Graph.*, 2(1):1–31, 1983.

[37] Vaughan Pratt. Direct least-squares fitting of algebraic surfaces. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 145–152, New York, NY, USA, 1987. ACM.

[38] Bu qing Su and Ding zhe Liu. *Computational geometry: curve and surface modeling.* Academic Press Professional, Inc., San Diego, CA, USA, 1989.

[39] R. Schmidt, B. Wyvill, M. C. Sousa, and J. A. Jorge. Shapeshop: sketch-based solid modeling with blobtrees. In *SIGGRAPH '07: ACM SIGGRAPH 2007 courses*, page 43, New York, NY, USA, 2007. ACM.

[40] Karan Singh. Interactive curve design using digital french curves. In *I3D '99: Proceedings of the 1999 symposium on Interactive 3D graphics*, pages 23–30, New York, NY, USA, 1999. ACM.

[41] Karan Singh and Eugene Fiume. Wires: a geometric deformation technique. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 405–414, New York, NY, USA, 1998. ACM.

[42] Desney S. Tan, George G. Robertson, and Mary Czerwinski. Exploring 3d navigation: combining speed-coupled flying with orbiting. In *CHI '01: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 418–425, New York, NY, USA, 2001. ACM.

[43] Matthew Thorne, David Burke, and Michiel van de Panne. Motion doodles: an interface for sketching character motion. In *SIGGRAPH '07: ACM SIGGRAPH 2007 courses*, page 24, New York, NY, USA, 2007. ACM.

[44] Steve Tsang, Ravin Balakrishnan, Karan Singh, and Abhishek Ranjan. A suggestive interface for image guided 3d sketching. In *CHI '04: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 591–598, New York, NY, USA, 2004. ACM.

[45] D.J. Walton and D.S. Meek. A controlled clothoid spline. *Computers & Graphics 29*, pages 353–363, 2005.

[46] Colin Ware and Steven Osborne. Exploration and virtual camera control in virtual three dimensional environments. *SIGGRAPH Comput. Graph.*, 24(2):175–183, 1990.

[47] Nayuko Watanabe, Motoi Washida, and Takeo Igarashi. Bubble clusters: an interface for manipulating spatial aggregation of graphical objects. In *UIST '07: Proceedings of the 20th annual ACM symposium on User interface software and technology*, pages 173–182, New York, NY, USA, 2007. ACM.

[48] Robert C. Zeleznik, Kenneth P. Herndon, and John F. Hughes. SKETCH: An interface for sketching 3D scenes. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, pages 163–170. Addison Wesley, 1996.