Sketch-Based Path Design

James McCrae[*] Karan Singh[†]

Dynamic Graphics Project
University of Toronto

Abstract

We present *Drive*, a system for the conceptual layout of 3D path networks. Our sketch-based interface allows users to efficiently author path layouts with minimal instruction. Our system incorporates some new and noteworthy components. We present the break-out lens, a novel widget for interactive graphics, inspired by break-out views used in engineering visualization. We also make three contributions specific to path curve design: First, we extend our previous work to fit aesthetic paths to sketch strokes with constraints, using piecewise clothoid curves. Second, we determine the height of paths above the terrain using a constraint optimization formulation of the occlusion relationships between sketched strokes. Finally, we illustrate examples of terrain sensitive path construction in the context of road design: automatically removing foliage, building bridges and tunnels across topographic features and constructing road signs appropriate to the sketched paths.

Index Terms: I.3.3 [Computer Graphics]: Picture/Image Generation—Line and curve generation; I.3.6 [Computer Graphics]: Methodology and Techniques—Interaction techniques

1 Introduction

While path layouts are necessary for transportation design (roads, railways, nature trails), they are also important as motion paths for animation, navigation and visualization in games and virtual environments. Noted landscape architect Lawrence Halprin [16] points out that the design of such paths should emphasize the journey or *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. These shape modeling interfaces [18, 31], *Google SketchUp*, are not suitable for conceptual path layout, forcing path 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 often use road networks as a visual representation of the system (see Figure 1), our system can be adapted to paths representing railways, waterways, nature trails, pipe or power lines, graph networks, networks of surface patches or general 3D curve based modeling (see Figure 15).

2 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 [40, 18, 36, 31, 29, 10], camera motion along a path [17], spatial layout [1, 39], interface design [22], animation [11, 35] and the sketching of flora and fauna [2] and architectural environments, such as *Google SketchUp*. The area of sketch-based path design, however, is largely unexplored.

[*]e-mail: mccrae@dgp.toronto.edu
[†]e-mail: karan@dgp.toronto.edu

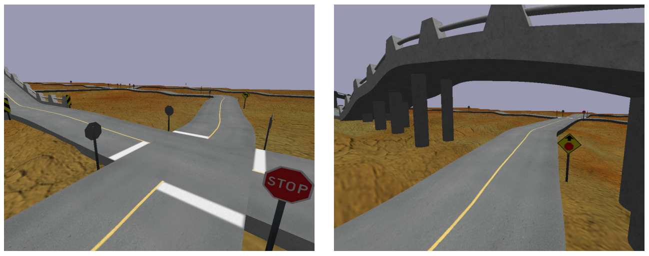

Figure 1: Views of a road network created within *Drive*.

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 [18, 29, 19] 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 [8, 20]. Multi-view approaches while mathematically straightforward are suboptimal for a user that 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 surrounding environment. While lenses for zooming have existed for some time [6, 24] in interactive visualization, we believe this the first approach to both view manipulation and editing via a lens using nonlinear projection [4, 9].

Occlusion has been exploited to disambiguate the depth of 3D organic shapes [10]. In [10] visible contours of 3D objects are processed to build 2D panels that are inflated into 3D shapes like Teddy [18]. 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 represent the spine of the path. We also satisfy occlusion constraints using an optimization formulation better suited to path design than 3D shape modeling [10] (see Figures 6, 7).

Interaction techniques commonly represent continuous curves as densely sampled polylines. Geometric properties for these curves, however, needs to be imposed by the curve creation and editing technique [14, 36]. 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 [26, 23] or a spline where every three consecutive points are fit with a parabola-like clothoid segment [37]. Discrete formulation of clothoid using nonlinear subdivision have also been proposed [15, 32]. The fairness properties of clothoids have been exploited to generate 2D curves that approximate sketched input strokes [25] with better fairness properties than the common fairing approaches of spline fitting [30] or discrete smoothing by iterative neighbour averaging [36]. This approach also 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 paths can be regenerated at arbitrary resolution. We build upon the framework of [25] in this paper to allow precise geometric constraints, necessary for local control when editing path networks.

Finally, our design 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* and *Microsoft Virtual Earth* provide compelling interfaces for visualizing terrain and other 3D environments, but do not represent paths with independent geometry. Path networks have also been used as an input for the procedural modeling of 3D geometry [7]. We handle the construction of the evolving environment by integrating it into the path design process. Creating a path automatically defines a cut-and-fill corridor on the terrain into which the path is integrated, removing foliage, constructing bridges, tunnels, path crossings and signage as dictated by the evolving landscape and path network.

The drive through is executed with a select and play action, allowing a user to animate the drive along a selected section of path viewed from a number of typical vantage points, or with interactive control during animation. There has been research on authoring virtual flythroughs in the context of scene visualization [38, 34] or product design [21, 5]. 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, functionality that is more precise and complex to control in animation systems like *Maya*.

3 Design Goals

Our goal of rapidly conceptualizing a 3D *driving experience* through sketching helps define a number of design principles.

1. 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.

2. This application is designed not only for game designers and transport landscape architects but also for novice users without exceptional artistic skills, for example, to build custom tracks to race on in a gaming environment, or 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 fun to play with, encouraging creative exploration.

3. 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.

4. 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.

5. 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 (as an example, see Figure 6).

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

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 14). Important advantages of the break-out lens are that its locality elegantly avoids view occlusion commonly caused by undulating terrain (see Figure 16) and sketching within the lens allows multi-view 3D curve editing, without the handicap of mentally resolving the 3D curve from disconnected views.

Clothoids: 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 [26, 37]. Recently [25] presented an approach to interactive sketch stroke filtering using clothoid curve fitting. We non-trivially adapt this approach to our application, by incorporating geometric constraints that allow us to interactively interpolate points, edit curves by oversketching and represent closed curves using piecewise clohoids.

Crossing paths: We handle arbitrarily complex path crossings by sketching occlusion as breaks in the curve path (see Figure 6). 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, while maximizing the variation of height needed to satisfy the occlusion relationships (see Figure 7).

Terrain sensitive sketching: Our implementation does not focus specifically on the construction of path layouts but also on the rapid conceptualization and preview of the entire driving experience. The terrain is more than a canvas on which to sketch paths, it is an evolving environment into which the paths integrate, with automatic construction of bridges, tunnels, road signs and changes in foliage (see Figure 1, 8, 9).

4 Path Creation and Editing

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 2), as is common in transportation design. The fitted curve comprising line, circular-arc and piecewise clothoid segments is discretely sampled and projected back to the 3D geometry to define the spine of the final path along which path geometry is deformed.

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). Selected regions of paths can be deleted using the lasso menu. A path edited by oversketching can be globally refitted using clothoids.

4.1 Clothoid Fitting with Geometric Constraints

The clothoid fitting approach of [25], while appropriate for the creation of smooth open paths as well as sharp corners in paths, is unsuitable for creating smooth closed curves or for local path editing. We accomplish this by using the end-point and end-tangent

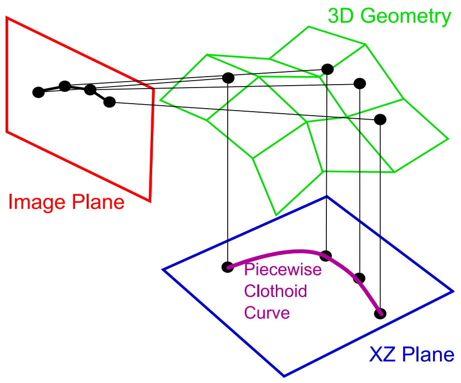

Figure 2: The sketched curve in the image plane is projected onto the terrain geometry, then projected onto the XZ plane where it is represented as a 2D model consisting of line, circular-arc and clothoid segments.

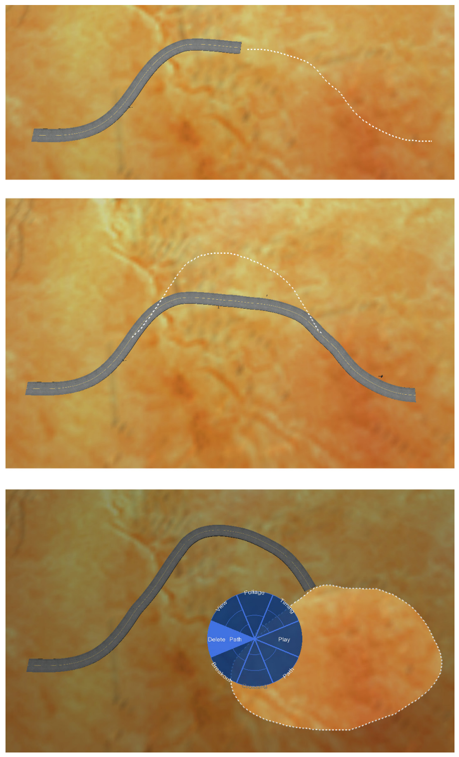

Figure 3: Extending a path (top) and editing it (middle) by oversketching. A selected portion is deleted (bottom) using the lasso-menu.

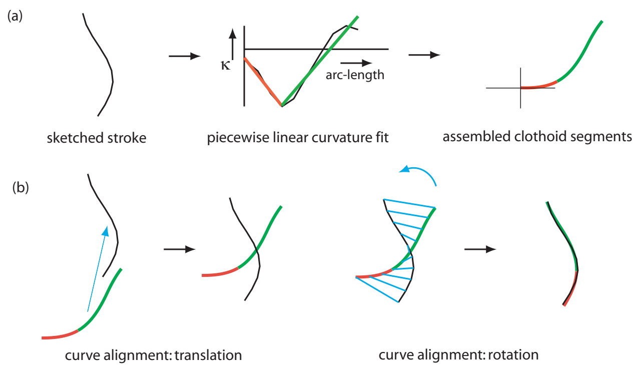
(a)
κ
arc-length
sketched stroke
piecewise linear curvature fit
assembled clothoid segments
(b)
curve alignment: translation
curve alignment: rotation

Figure 4: 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.

proximity of closed curves and local path edits as constraints to the clothoid fitting process.

The input to our algorithm is a 2D polyline stroke, and the output a smooth curve comprising line, circular-arc and clothoid segments.

We first provide a brief overview of the general clothoid fitting approach given in [25] (see Figure 4). The Frenet-Serret formula is used to compute discrete curvature [28] at vertices of the input polyline. A dynamic programming algorithm is then used to fit a piecewise linear approximation to the discrete curvature of the stroke as a function of arc-length. A user defined parameter controls the tradeoff between fitting error and the number of pieces of linear curvature. The start and end curvature values of each linear piece uniquely determine a line, circular-arc or clothoid curve segment. These segments 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. This transform can be computed efficiently by formulating the error as a weighted least squares optimization problem [27].

For each clothoid segment C_i, we have its curvature space end-points (x_i^P, y_i^P) and (x_{i+1}^P, y_{i+1}^P). These parameters uniquely map to a clothoid segment defined by B, that defines how tight the clothoid spiral is, and the start and end parameter values t_1 and t_2.

$$B = \sqrt{\frac{x_{i+1}^P - x_i^P}{\pi(y_{i+1}^P - y_i^P)}}, t_1 = y_i^P B \text{ and } t_2 = y_{i+1}^P B. \qquad (1)$$

Given a geometric constraint we thus determine one or more existing clothoid pieces whose parameters we vary to locally satisfy the given geometric constraints. For instance, to generate a G^2 transition when oversketching (see Figure 3) or a closed clothoid curve (see Figure 5) we need to enforce common end-point, end-tangent and end-curvature values.

To close a curve, our approach introduces an additional clothoid segment (which we call the *join-piece*) between the first and last clothoid segments, which provides 5 dimensional variability to satisfy the set of constraints. The 5 paramaters we operate on after adding the join-piece are: the curvature value shared between end-points of the first clothoid piece and the join-piece, the curvature value shared between end-points of the last clothoid piece and the join piece, and the arc-lengths of the first, last and join clothoid pieces. We apply gradient descent iteratively to vary these clothoid parameters, in order to minimize

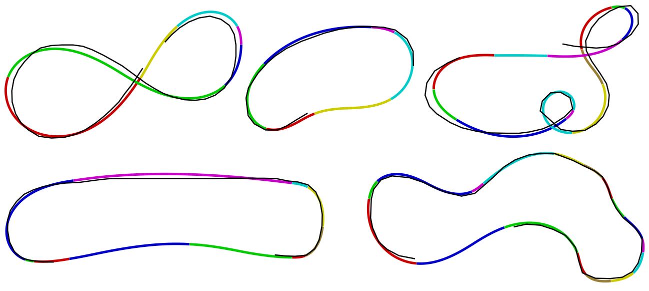
Figure 5: Clothoids with constraints are used to model almost-closed sketch strokes as closed G^2 paths.

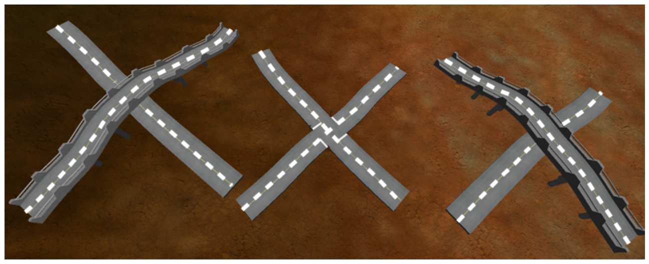
Figure 6: Breaks in the sketch stroke indicate the over/under occlusion relationship between intersecting paths. A small break in a path at a crossing indicates occlusion from above and makes the other path pass over it. Unbroken paths indicate an intersection.

$$|| \, endPoint - startPoint \, ||_2 + \tau \min_i (| \int \kappa - 2\pi i \, |). \quad (2)$$

The first error term measures continuity of position between *startPoint* and *endPoint*, and the second term is a measure of tangential continuity. The integral of the curvature κ along a continuous curve measures the change in tangent direction; tangents line up when the integral of curvature is an integer multiple of 2π (it will be $\pm 2\pi$ for a simple closed curve). Alternatively, the angular error between end-tangents may be used. Curvature continuity between the three clothoid pieces is maintained by definition. The scalar multiple τ of the second term is initialized at zero, and as convergence proceeds is incremented to 5.0. This results in the curve first bringing the end-points together, and then "straightening out" the tangents at the connected end-points. In practice we find this numerical approach to be stable and offering interactive performance, typically converging in at most 300 iterations.

We are thus able to locally edit piecewise clothoids with numerically precise point and tangent interpolation as well as represent almost-closed input strokes with a single closed G^2 curve.

4.2 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 6. The order in which the solid and broken path is sketched is not important. Two unbroken paths crossing each other indicate 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 relationship.

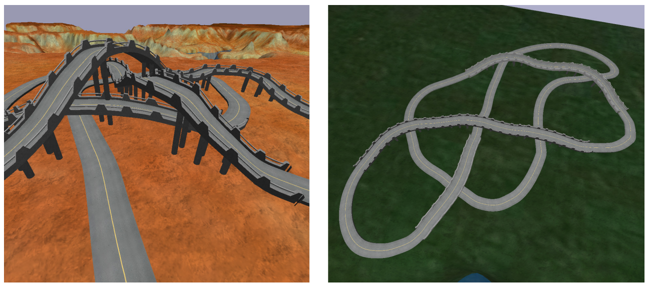
Figure 7: A complex multi-level crossing (left) and a closed path with many crossings (right).

4.3 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 7) require a more sophisticated approach to determining path height from the terrain. Our problem is to determine optimal height values at all points where crossing relationships exist, while minimizing overall height from the terrain.

The input to our path height optimizer is a set of n pairs of points in the path network $\{(a_0,b_0),\dots,(a_{n-1},b_{n-1})\}$ which have the over-under crossing relationship. This means that for each pair i, the vertical components of the points (denoted $y(a_i), y(b_i)$) are such that $y(a_i) > y(b_i)$.

We define a function for cost of path height from the terrain, *heightCost*, as

$$heightCost = \sum_{i=0}^{n-1} \left[(y(a_i) - T(a_i))^2 + (y(b_i) - T(b_i))^2 \right] \quad (3)$$

where $T(a_i), T(b_i)$ are the terrain heights at points a_i, b_i.

We also define a cost function, *relationCost*, that penalizes small vertical distances between each pair in an over-under relationship. This function is defined as

$$relationCost = \sum_{i=0}^{n-1} \begin{cases} \frac{1}{y(a_i)-y(b_i)} & \text{if } y(a_i) > y(b_i) \\ \infty & \text{otherwise} \end{cases} . \quad (4)$$

We minimize the objective function which is a weighted sum of the two above mentioned functions:

$$w_1 \cdot heightCost + w_2 \cdot relationCost. \quad (5)$$

We use the steepest descent method with backtracking line search to find a locally optimal solution for the set of vertical positions $\{y(a_0),\dots,y(a_{n-1}),y(b_0),\dots,y(b_{n-1})\}$. The gradient within this $2n$ dimensional space is estimated by applying finite differencing in each dimension independently.

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 (such as in Figure 7), reordering them using the lasso menu can be a problem as it is difficult to select a single pair of crossing paths.

4.4 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

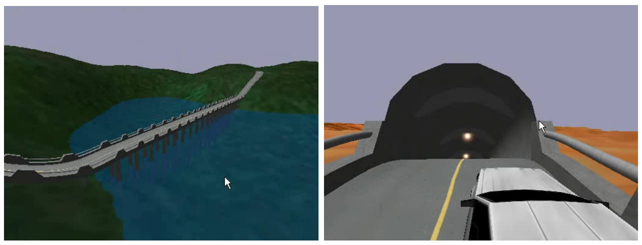

Figure 8: Bridges, support pillars (left) and tunnels (right) are automatically constructed to integrate paths into the evolving environment.

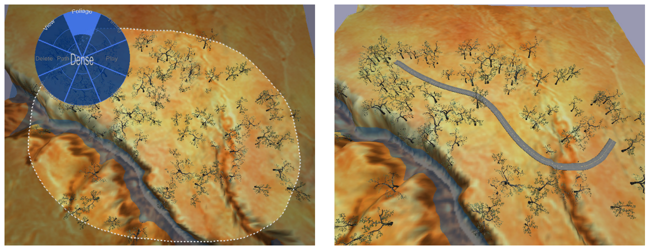

Figure 9: Foliage is added (left) or removed using the lasso-menu (right).

which they are sketched and appropriately add and remove geometric feature to alter the environment into which they are integrated.

Within *Drive*, a 2-lane road piece is defined parametrically that can be laid along any given path. Roads can be replaced by arbitrary geometry such as railway tracks, simply by replacing the parametric piece. Within the domain of roadway construction, as paths are generated, *Drive* will automatically determine and place an appropriate set of landmarks, such as signs for stop, stop ahead, sharp turn, bump, dip (see Figure 1). Appropriate road markings (such as at intersections) are also placed automatically.

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 8). Support pillars connecting the path to the terrain below, tunnel lights and bridges, like roads, are defined parametrically and can be readily customized.

We use foliage as an example of paths interacting with arbitrary terrain attributes. The creation of paths automatically removes any foliage on the paths. Optionally, foliage can be planted alongside 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, but will not be placed on top of paths or water bodies (see Figure 9).

5 Interaction and Visualization

The single essential instruction to a user is that sketching an open curve creates or edits paths and a closed (self-intersecting) curve selects a region of interest and invokes a radial menu [1] that is sensitive to the region's content (see Figure 10). Note that closed paths such as roundabouts can still be created, either as two or more open curves, or as a single curve that is nearly closed. This clean distinction between design and command space is trivial to learn, placing little cognitive overhead on the user.

5.1 Lasso Menu

As shown in Figure 10, our lasso-menu is an 8-item radial menu that is context-sensitive to the selection region. A number of menu items have up to 3 sub-options defined as concentric wedges of the

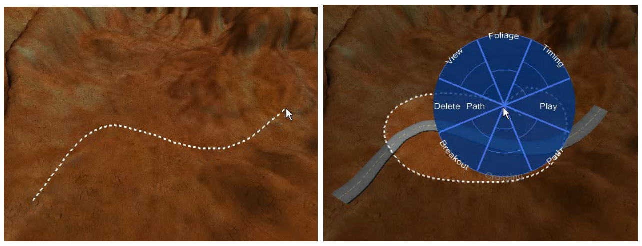

Figure 10: Open curves create and edit paths (left) while closed curves select a region and invoke a lasso-menu (right).

item. Hovering over a sub-option describes its text dynamically in the middle of the menu, avoiding clutter and text at awkward orientations. For many actions, such as path deletion or foliage change, hovering the cursor provides the user a preview of the action, improving the discoverability of the interface. The menu can also be invoked by a press-and-hold action as an alternative to sketching a closed curve.

5.2 Camera Control

As stated in our design goals, we would like promote single-view sketching and minimize time spent on view navigation, which is the single most frequently used control in typical 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 (see Figure 11) 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. Users can also explore alternate viewpoints within the context of a single view using the break-out view and lens.

Experienced users often expect typical tumble, pan and dolly camera tools. We additionally support camera controls as found in popular 3D systems such as *Maya*, even though persistent 3D camera control detracts from our modeless, single-view sketch design philosophy. The ALT key enables the camera, and the left, middle and right mouse buttons tumble, pan and zoom respectively.

5.3 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 [12]. We are motivated by these viewpoints, not only for rapid local 3D visualization of paths, but also as an interface element for performing path height editing.

5.3.1 Break-out View

The user selects a region of a path and invokes a break-out view from the lasso-menu (see Figure 12). We use the begining and end points of the selected path segment to determine a 3D segment, called the break-out axis. Intuitively, this axis defines a break-out view as if the user was standing by the side of the path facing perpendicular to the axis. Note that the break-out axis can be defined not just for planar paths, but those with height variation as well. The break-out view is generated and shown as an animated transition within a window the shape of the lasso selection, the viewing position within the break-out view translates to view the side of the path at a proper distance (see Figure 12). The screen-space position of the break-out view also moves across the centre of the screen such that the path segment of interest can be shown from both the original and break-out views simultaneously. The break-out view,

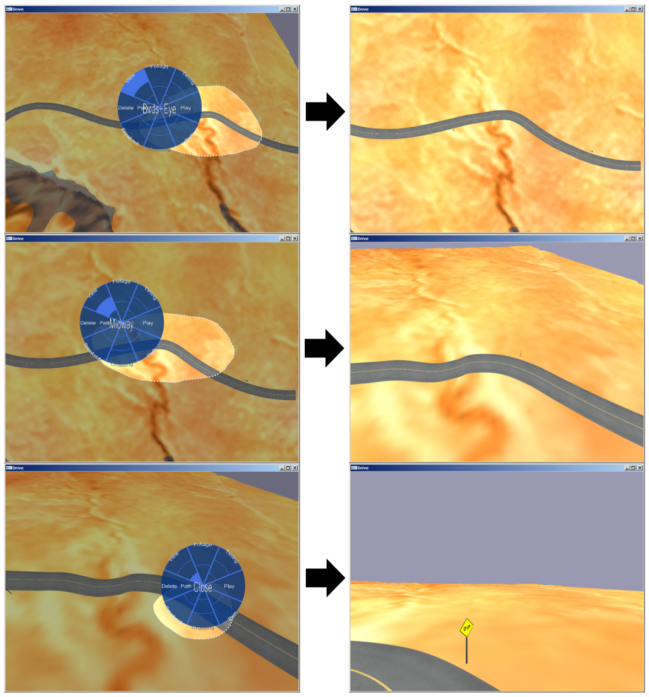
Figure 11: (Top row) The user lassos a region and selects the *Birds-eye* camera preset. (Middle row) The *Mid-way* preset. (Botton row) The *Close-up* preset.

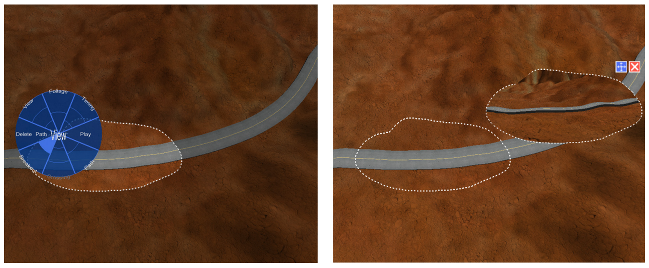
Figure 12: Users select a region of the path (left) and invoke a break-out view from the lasso-menu (right).

like any other GUI window, can be translated about screen space and destroyed as needed. Multiple break-out views can co-exist simultaneously.

We allow the user to oversketch within the view to edit path elevation in the same way paths are oversketched on the terrain (see Figure 13). To determine new path heights along the selected segment, points of the path and the user's oversketch stroke are projected onto an orthogonal plane whose axes are the break-out axis and the vertical (height) axis. For each point along the path whose projected position falls within the range of the projected sketch stroke along the break-out axis, we assign a new height interpolated from the two projected sketch stroke points which bound it.

5.3.2 Break-out Lens

Oversketching to edit path elevation in the break-out view is an example of multiview 3D curve sketching. While mathematically succinct, users find it difficult to draw 3D curves using multiple disconnected 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. However, we can address the

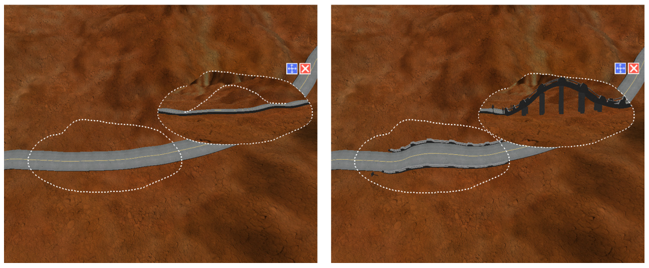
Figure 13: Oversketching within a breakout-view (left) allows the user to edit path elevation. (Right) The result of the oversketch.

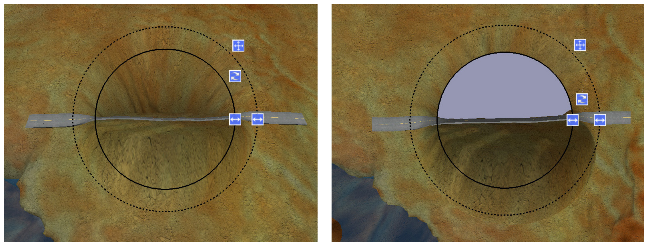
Figure 14: The break-out lens rendered without (left) or with (right) a background plane.

break-out view disconnect by reformulating it as a break-out lens (see Figure 14).

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 [9]. The lens has four controls: move the lens, control the inner and outer radii, and define the angle of rotation θ around the break-out axis.

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. We have implemented the view warping technique both in software and as a vertex shader to run on the GPU.

The break-out lens, like the break-out view, allows oversketching to edit path elevation. In this case, it is possible to perform oversketching using any setting for θ so long as the vertical axis is not coincident with the camera viewing direction. Intuitively, a rotation θ that provides a near-orthogonal viewing direction of the path will be more ideal for editing. 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 [33, 29] from a single view (see Figure 15).

Break-out lenses offer a number of appealing affordances over global view changes that make them a useful general tool for interactive 3D graphics (see Figure 15).

- Unconstrained 3D camera control can be notoriously cumbersome. Break-out lenses are domain specifically constrained by the scene geometry underneath to a simple view-angle control, rather than a free-form tumble/pan/zoom.

- The concept of lenses, such as zoom lenses, are common in interactive graphic applications, into which break-out lenses seamlessly integrate. Zoom lenses are popular as they provide fast alternate views of local regions within a global context,

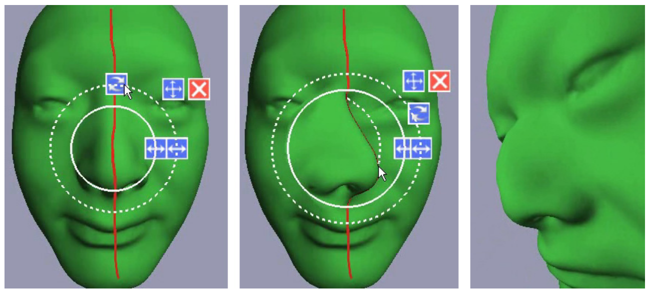
Figure 15: A curve is sketched and oversketched with a break-out lens to define a wire deformation of the nose.

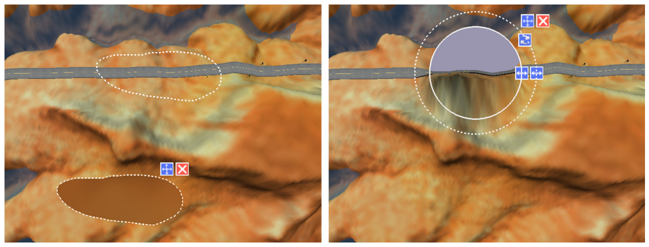
Figure 16: The break-out view of a path (left) can be occluded by scene geometry in the foreground. The break-out lens (right) avoids occlusion problems by deforming scene geometry locally.

while keeping user-focus on a primary task. Break-out lenses similarly allow continued sketching in a desired view with local view changes when needed, instead of bookmarking the view, temporarily changing it and then jumping back.

- Oversketching in a break-out lens is natural for path height editing and homogeneous with path editing on the terrain. In contrast, the same oversketch after a global view change can be ambiguously interpreted as a path height edit or a path edit on the terrain.

- Tumbling a birds-eye view to the ground often causes hilly terrain to occlude the sketched path/region. Users may be able to dolly the view-in past occluding features, or to manually adjust camera clip planes to solve this. Instead, the locality of the break-out lens solves this problem by design (see Figure 16).

5.3.3 Rendering the Background Plane

A major difference between the break-out view in Figure 12 and the break-out lens in Figure 14 is the lack of a horizon and background in the lens. This is because the terrain wraps around continuously obscuring the background plane. We can address this by rendering a background plane on the far side of the path from the camera to simulate a horizon and background. This plane, drawn strictly within the inner radius, makes the view within the lens more like a view from ground perspective, yet provides a continuous transition on the near side (see Figure 14). The opacity of the background plane is interpolated with the rotation of the lens to produce a continuous transition (the system uses values of 30 and 75 degrees to interpolate from transparent to opaque).

6 Timing and Playback

Visualizing the navigation experience along paths is an essential aspect of our system. Lassoing a path and selecting *Play* from the lasso-menu starts a vehicle navigating along it. The camera view during playback can be selected from a context sensitive lasso-menu (see video) from a set of meaninigful predefined views or controlled by the user in a freeform manner. If the user select portions of paths with 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 (our system depicts these timings as speed limit signs placed roadside).

7 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. Many of the terrains demonstrated are represented as grids whose resolutions range from 256x256 to 1024x1024. Path pieces and other environmental geometry primitives consist of approximately 100-200 triangles each. Complex scenes authored with our system consisting of hundreds of thousands of triangles can be displayed and interacted with in real-time. All aspects of creation and rendering in our system: sketching, clothoid fitting, path crossing resolution, view warping and editing operations perform in real-time.

8 Conclusion

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. We informally evaluated our system by providing a brief set of instructions and the application to half a dozen people with mixed computer graphics skill. In general, the response was positive and the surprise bonuses of terrain sensitive geometry kept the users engaged. Some were able to discover most of the functionality without being told. We also revaluated our application with the landscape architects who felt the system was instantly useable in conceptual transportation layout. 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 some users familiar with CG applications, persistently navigated the scene with the 3D camera controls by habit.

In summary, we have presented a novel system for the concept sketching of path layouts. Our system integrate a number of new ideas each of which is also applicable to a wider range of applications. Clothoid fitting is shown to be useful for generating fair curves from input sketch strokes. Arbitrarily complex self-occluding 3D curves can be effectively created using our optimization of crossing relationships. The break-out view and lens find use in general shape modeling and visualization (see Figure 15). 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 overall 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 applications in landscaping and urban planning.

References

[1] A. Agarawala and R. 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] F. Anastacio, M. C. Sousa, F. Samavati, and J. 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] E. A. Bier, M. C. Stone, K. Pier, K. Fishkin, T. Baudel, M. Conway, W. Buxton, and T. DeRose. Toolglass and magic lenses: the see-through interface. In *CHI '94: Conference companion on Human factors in computing systems*, pages 445–446, New York, NY, USA, 1994. ACM.

[5] N. Burtnyk, A. Khan, G. Fitzmaurice, R. Balakrishnan, and G. 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.

[6] M. 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.

[7] G. Chen, G. Esch, P. Wonka, P. Mueller, and E. Zhang. Interactive procedural street modeling. *ACM Trans. Graph.*, 27(3), 2008.

[8] J. M. Cohen, L. Markosian, R. C. Zeleznik, J. F. Hughes, and R. 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.

[9] P. Coleman and K. 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.

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

[11] R. C. Davis, B. Colwell, and J. 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.

[12] J. A. Dennison and C. D. Johnson. *Technical Illustration: Techniques and Applications*. Goodheart-Wilcox, 2003.

[13] G. 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] T. Grossman, R. Balakrishnan, and K. 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.

[15] L. Guiqing, L. Xianmin, and L. 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.

[16] L. Halprin. *Notebooks 1959–1971*. The MIT Press, 1972.

[17] T. Igarashi, R. Kadobayashi, K. Mase, and H. Tanaka. Path drawing for 3d walkthrough. In *ACM Symposium on User Interface Software and Technology*, pages 173–174, 1998.

[18] T. Igarashi, S. Matsuoka, and H. Tanaka. Teddy: a sketching interface for 3d freeform design. In *SIGGRAPH '07: ACM SIGGRAPH 2007 courses*, page 21, New York, NY, USA, 2007. ACM.

[19] L. B. Kara, K. Shimada, and S. D. Marmalefsky. Calligraphic interfaces: An evaluation of user experience with a sketch-based 3d modeling system. *Comput. Graph.*, 31(4):580–597, 2007.

[20] O. Karpenko, J. F. Hughes, and R. Raskar. Epipolar methods for multi-view sketching . In *Eurographics Workshop on Sketch-Based Interfaces and Modeling*, pages 167–173, Aug. 2004.

[21] A. Khan, B. Komalo, J. Stam, G. Fitzmaurice, and G. 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.

[22] R. 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.

[23] B. B. Kimia, I. Frankel, and A.-M. Popescu. Euler spiral for shape completion. *Int. J. Comput. Vision*, 54(1-3):157–180, 2003.

[24] F. D. F. M. S. T. Carpendale, D. J. Cowperthwaite. Multi-scale viewing. In *ACM SIGGRAPH 96 Visual Proceedings: The art and interdisciplinary programs of SIGGRAPH '96*, pages 149–149. ACM Press, 1996.

[25] J. McCrae and K. Singh. Sketching piecewise clothoid curves. In *Sketch-Based Interfaces and Modeling 2008*, pages 1–8. Eurographics Association, 2008.

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

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

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

[29] A. Nealen, T. Igarashi, O. Sorkine, and M. Alexa. Fibermesh: designing freeform surfaces with 3d curves. In *SIGGRAPH '07: ACM SIGGRAPH 2007 papers*, page 41, New York, NY, USA, 2007. ACM.

[30] V. 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.

[31] 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.

[32] R. Schneider and L. Kobbelt. Discrete fairing of curves and surfaces based on linear curvature distribution. In *In Curve and Surface Design: Saint-Malo*, pages 371–380. University Press, 1999.

[33] K. Singh and E. 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.

[34] D. S. Tan, G. G. Robertson, and M. 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.

[35] M. Thorne, D. Burke, and M. 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.

[36] S. Tsang, R. Balakrishnan, K. Singh, and A. 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.

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

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

[39] N. Watanabe, M. Washida, and T. 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.

[40] R. C. Zeleznik, K. P. Herndon, and J. F. Hughes. SKETCH: An interface for sketching 3D scenes. In H. Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, pages 163–170. Addison Wesley, 1996.