Report Number: WUCSE-2012-5

2012

Just Draw It! A 3D Sketching System

Cindy Grimm and Pushkar Joshi

We present a system for sketching in 2D to create 3D curves. The interface is light-weight, pen-based, and based on observations of how artists sketch on paper.

2012-5

Just Draw It! A 3D Sketching System

Authors: Cindy Grimm and Pushkar Joshi

Corresponding Author: cmg@wustl.edu

Abstract: We present a system for sketching in 2D to create 3D curves. The interface is light-weight, pen-based, and based on observations of how artists sketch on paper.

Type of Report: Other

EUROGRAPHICS Symposium on Sketch-Based Interfaces and Modeling (2012), pp. 1–14
L. B. Kara and K. Singh (Editors)

Just DrawIt: a 3D sketching system

Cindy Grimm[1] and Pushkar Joshi[2] †

[1]Washington University in St. Louis
[2]Motorola Mobility

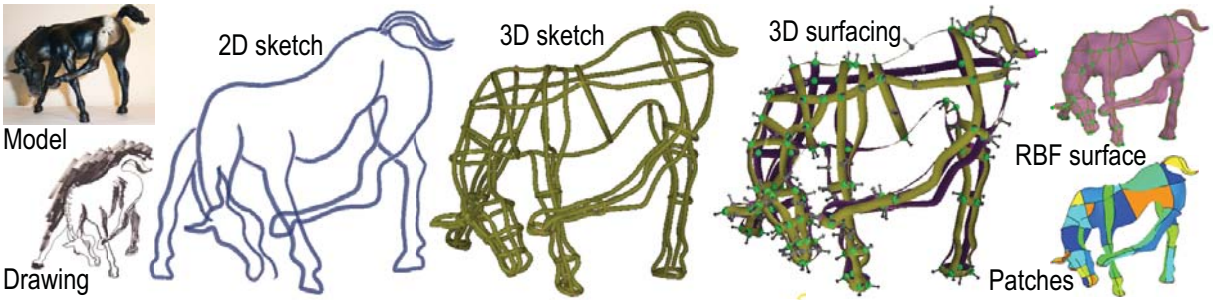

Figure 1: *A 2D sketch and 3D sketch created using JustDrawIt (in approximately four hours) inspired by a traditional artist's drawing of the horse model (shown on left). JustDrawIt was also used for 3D surfacing: "snapping" the curve network together and specifying normals where needed in order to create surfaces (upper right) or patches (lower right).*

Abstract

We present "JustDrawIt", a sketch-based system for creating 3D curves suitable for surfacing. The user can sketch in a free-form manner from any view at any time, and the system infers how those sketch strokes should be added to the drawing. Specifically, existing curves are projected to 2D and analyzed to see if the stroke edits or extends an existing curve, or if stroke should make a new curve. In the former case the 2D stroke is promoted to 3D using information from the existing curve, then joined to that curve. In the latter case, we use additional information (temporary 3D surfaces) to create a new curve in 3D. All non-sketching interactions are based on unintrusive context-aware, in-screen pie menus designed for rapid pen-based input. We also provide novel rendering styles and aides for interpreting and working with 3D sketches. Finally, we support "snapping" together curve networks and specifying normals in order to create consistent curves from which surface models can be generated.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation—Line and curve generation

1. Introduction

We present "JustDrawIt" — a sketch-based curve editing system for creating 3D curve-network models on the computer. The system's primary design goal is to mimic the experience of drawing on paper as closely as possible. Towards that goal, we support a free-form, natural 2D sketching user experience developed by a study on how artists draw [Gri11b, Gri11a]. We augment the natural user experience with an in-screen, contextual pen-based, menu system for issuing editing and non-drawing commands. This menu system scales better than gestures for multiple tasks and tablet input. We provide several visualization aides (shadows, depth-based shading, 3D geometry) to better place the curve network in 3D. Finally, we provide several options for specifying depth values along non-planar curves.

JustDrawIt is built as a judicious combination of exist-

† Research conducted at, and funded by, Adobe Systems Inc.

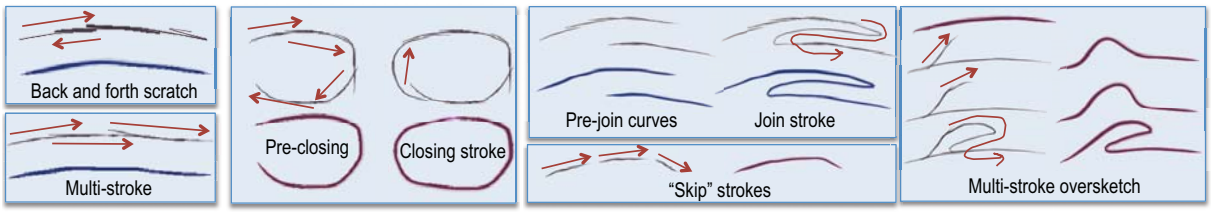

Figure 2: *Examples of strokes joined into curves. Back and forth scratches (upper left) are first turned into smooth strokes before joining.*

ing and novel sketch-based drawing techniques. A drawing in JustDrawIt is represented as a collection of 3D curves, any of which can be edited at any time, and from any view. The core drawing system analyzes input 2D strokes and uses them to edit existing curves or to create new curves. JustDrawIt supports advanced 3D curve editing by offering 3D sketching surfaces (drawing planes, extrusion and inflation surfaces), direct manipulation (dragging) portions of curves in 3D, and editing curves by oversketching them from other views. JustDrawIt can create 3D curve networks that define a consistent, unambiguous surface by "snapping together" curves and defining local surface normals.

From an implementation point of view, JustDrawIt can be viewed as three different systems, each with increasing levels of functionality. First, we have a complete, stand-alone 2D sketching system. To this, we can add advanced 3D curve editing functionality. Finally, we can add technology for creating a consistent and unambiguous curve network suitable for surfacing.

JustDrawIt incorporates and extends many of the excellent sketching and tablet interface ideas that exist already [OSSJ09]. Integrating several of these techniques into one system is challenging, more so because we want to support interactive drawing and editing from any view in a natural way. The style of 2D drawing we support was inspired by a user study that analyzed artists drawing on paper [Gri11b, Gri11a]. This study showed that a single "curve" can be created in a variety of ways, from one long stroke to multiple, disjoint strokes (see Figure 2). Additionally, artists often switch back and forth between drawing new curves and editing existing curves, may edit curves in random order, and may edit the same curve repeatedly at different times of the drawing process. Our 2D sketching system supports this free-form, "sketch-anywhere-anytime" approach (Section 4). Specifically, we incorporate multi-stroke sketching [BBS08, OS10], both for creating curves *and* for oversketching [BBS08] existing curves. We add to this the ability to scratch back and forth [OSJ11] and to leave small gaps between strokes (see "skip" strokes in Figure 2). There is no notion of a selected curve — instead, the system is continually inferring which existing curve the user stroke should modify (if any).

For 3D curve creation we support the traditional drawing plane [BPCB08, BBS08] and extrusion surface [BBS08] approaches, as well as introduce a new paradigm we call an *inflation surface* (Section 5.1). This approach was motivated by the interior "contour" strokes we saw in our artist's drawings (see Figure 11). With two quick strokes the user specifies a 3D, non-planar surface that they can then draw on. This is similar in concept to inflation surfaces such as those used by Teddy [IMT99], FiberMesh [NISA07] and Repoussé [JC08], except we do not require a closed, planar contour to specify the surface.

We do not use epipolar constraints [KHR04, BBS08] to specify depth values along non-planar curves. Instead, we treat the problem as one of oversketching [CMZ*99]. It is notoriously difficult for a user to envision what a curve would look like from two different views, so instead we *always* create a 3D curve. The user can then change the view and oversketch or continue that curve from the new view. We use a novel depth interpolation and extrapolation technique to make the new stroke consistent (in depth) with the existing curve (Section 4.4).

For 3D surface creation we provide visualization and interface support for automatically and semi-automatically snapping curves together and orienting them. In particular, we use a novel ribbon rendering method which makes visualizing and editing the curve orientation (which direction is "out") easier.

We next walk through what the user sees and does for several common tasks in the interface. We follow this with previous work. The implementation details are laid out in the same order as the layers: 2D stroke analysis (Section 4), 3D curve creation (Section 5), and 3D surfacing (Section 6).

2. User's view

We describe how the user interacts with the JustDrawIt system at various drawing stages and for specific tasks. We include complete instructions in the supplemental materials as well as an accompanying video. In order to support the ability to support a wide variety of input device (tablet) configurations, we assume only pen 2D positional input (no

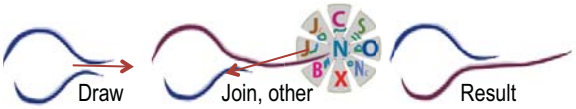

Figure 3: *The user drew a stroke that the system assigned to the wrong curve. The user over-rides that choice by tapping on the end of the stroke, which brings up the stroke menu. The user can pen-down on the "join" option and draw to the desired curve. Menu options, from top clock-wise: Combine two curves, Smooth the join, Oversketch, New closed curve, Delete, Back-and-forth scratch, Join, Join and close. Center option is for New curve.*

keyboard modifiers, pen proximity, pressure, or tilt information). While JustDrawIt can be used with a mouse, we expect an optimal user experience with a pen-like stylus. We have three drawing modes which the user can toggle between at will: 2D stroke-rendering, 3D tube-rendering, and 3D ribbon-rendering, which map conceptually to 2D drawing, 3D curve drawing, and 3D surfacing. All curves are always 3D; by default, the curves are created on the view plane, which is initially centered at the origin in the x, y plane.

Strokes: To start, the user simply starts drawing by placing the pen down on the drawing surface, dragging it, and lifting the pen up. We call the mark created in this continuous motion a *stroke*. If a stroke starts (or ends) near an existing curve, that stroke will be added to the curve. If the system picks the wrong option then the user can over-ride that decision, and optionally indicate which existing curve to add the stroke to (see Figure 3). The system doesn't have a notion of a "selected" (or un-selected) curve since a stroke can be added to any curve at any time. In order to ignore an existing curve, the user instead changes the curve into a "ghost" curve. Ghost curves are faintly visible and have access to the curve menu described below (just like any other curve), but are meant to be ignored by the computation that determines which curve to edit by the stroke.

The user can perform more traditional curve operations (dragging, scaling, rotating, smoothing, erasing some or all) by clicking on a curve, which brings up the curve menu (see Figure 4). Additional information (e.g. how much of the curve) is indicated by selecting the relevant option and then "scrubbing" (repeatedly moving back and forth) over the curve. Camera motions (pan, zoom, center) are similarly invoked by clicking on the background, away from any curve, to bring up the camera menu. In our experience, such heads-up-display menus are less intrusive than a standard menu bar, less ambiguous than gestures , and do not interfere with the creative drawing process [KB94, RJ02, MZL09].

3D curves: Once the user has drawn a few curves they can begin to change the depth (in 3D) of points along those

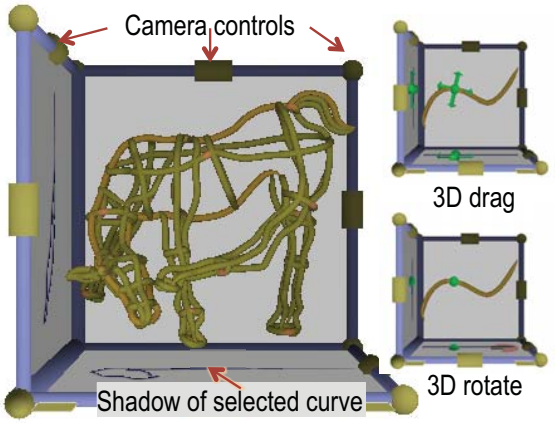

Figure 5: *The shadow box provides 3D visualization cues (shadows), camera controls, and 3D affine transformations (right). The 3D versions are identical to the 2D, except they are constrained to the view and right directions (floor) and view and up (right wall).*

curves, and to sketch curves that are not in the initial drawing plane. Perhaps the simplest (but non-sketching) method for moving the curves out of the drawing plane is 3D dragging: the user can drag all (or part) of a curve in the view direction. The shadow box [GH98]and 3D tube rendering (Figure 5) provide both 3D manipulation tools and better 3D visualizations. For example, to bring the back leg of the horse forward, the user selected the leg curve up to the hip, then grabbed the hoof and pulled it forward using a drag, creating a smooth depth change from the hoof to the hip.

Dragging is useful for large-scale 3D changes, but is not very useful for precisely shaping sections of curves. A more useful approach is to simply oversketch the curve from a different view direction (Figure 6). In this case, the user rotates the camera to the new view direction, oversketches, then rotates back and continues oversketching if desired.

Once a few curves are in place the user can define drawing planes and extrusion surfaces based on the current curves (see Figure 10). For example, to pick a drawing plane the user clicks on the curve, clicks on the "A" option, then clicks on one of the three arrows to pick the plane direction. Extrusion surfaces are created in a similar manner. To simplify creating cross-section contours, the user can draw a line to a second curve instead of picking a plane direction. This creates a plane that is as orthogonal as possible to both curves, and passes through the selection points.

Once the rough silhouette of the shape is drawn, the user can also make a temporary non-planar surface on which to draw interior curves. They draw the left and right boundaries of the surface simply by sketching over the existing curves.

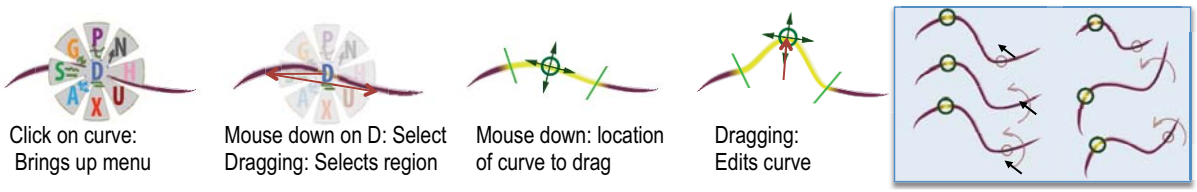

Figure 4: *Dragging a part of a curve. The user selects the curve (one click) then selects the drag option and the region of the drag on the curve (pen-down in option, drag over curve region). The can then grab and drag the curve in the plane, or in and out of the plane (clicking on top or bottom of drag arrow). Right: Translating, scaling, and rotating the curve are also invoked from the menu, with the specific operation determined by where the cursor is with respect to the drag icon (circle) and curve (above or below).*

They can then draw non-planar curves on the resulting sweep surface between the left and right boundaries (see Figure 11).

As the user builds up the curve network they can "snap" the curve network together by using the Pin option on the curve menu. They drag from the Pin menu option to the desired snap point on the opposite curve; the system automatically finds the closest pair of points. This also creates a normal at the pin point, which the user can grab and manipulate (see Figure 12). The user can place additional normal constraints to control the orientation of the curves, as visualized in the ribbon rendering mode.

To create a surface the user makes sure all of the curves are snapped together and the normals oriented. The system shows which curves are close, but not snapped — the user can fix these by clicking on them. Generating a surface takes a few seconds to a minute depending on the desired resolution and curve complexity.

3. Previous work

Olsen et. al [OSSJ09] provides an excellent survey that covers the 3D sketching and gesture-based modeling field. As stated in the introduction, we share the goals of many existing systems. We discuss here differences with specific systems. We will not touch on the extensive work on recovering models from drawings; our system is designed to be interactive, with the users explicitly creating curves in 3D, rather than a system for inferring 3D from a static drawing.

ILoveSketch [BBS08] is probably the most similar in spirit to our system. Like ILoveSketch, we focus on creating 3D curves as an end-goal in and of itself. We incorporate many of the interface elements of ILoveSketch (camera control, extrusion planes, curve-based selection of planes). We differ in four areas. First, our 2D sketching is designed to be more free-form and paint-system like. Strokes can be applied to any curve, not just the currently active one, from any view. We support over-stroking, merging with strokes, and scratch-style input, not just building a curve from multi-strokes. Also, we retain the user's original strokes, instead of fitting curves to them. Second, we use over-stroking from arbitrary views, rather than explicit epipolar one or two-view sketching, to create 3D curves. Third, we provide rendering cues and shaders to help disambiguate the 3D curve drawings. Fourth, we have explicit support for turning curves into a consistent curve network — this was not a goal of ILoveSketch.

We are not the first to merge strokes into curves. One approach is to treat the strokes as an image and perform image processing techniques to the finished drawing [OS10]. A second is to treat multiple strokes as a curve fitting problem [OSJ11, BBS08]. We are most similar to the latter approach, except we perform the analysis on the fly against *all curves*. We also do not rely on curve fitting to "glue" our strokes together, but instead work directly with the input strokes. Obviously, we can always perform smoothing or curve-fitting afterwards, but keeping the original data allows the user to create arbitrarily bendy curves with sharp corners.

There are several approaches for using drawings from two different views to create a 3D, non-planar curve [CMZ*99, KHR04, BBS08]. We share the same basic idea as these approaches, but treat this as an *editing* problem, not a construction one. The first stroke *always* creates a 3D curve (usually on the drawing plane). The user then moves the camera and edits the curve (either by oversketching or extending it with another stroke), rather than explicitly drawing a new curve from that second view. This leads to a slightly different formulation of the 3D reconstruction problem (see Section 4.4) and gets around the cognitive problem of trying to figure out which two curves to sketch.

Probably the most successful 3D sketching paradigm to-date is the inflation surface one, originally described in Teddy [IMT99] and since expanded on in a variety of ways and with different technologies [JC08, NISA07, OS10, SWSJ05]. Unlike these systems, we push surface creation to the end of modeling process, rather than starting with it. This has the advantage of not limiting the types of surfaces we can create to ones that are easily described by planar curves, but the disadvantages of 1) not having a surface to visualize

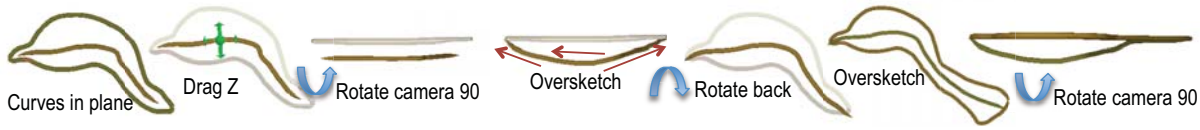

Figure 6: *Using a drag plus oversketching from different views to create a 3D, non-planar curve for the side of the fish.*

(and edit) from the start, and 2) of making the surfacing task much more difficult. We can, of course, at any time create a surface or part of one (see Section 5.1) and then use that surface to draw new curves on (which can then be edited). In that sense, inflation surfaces can be seen as a subset of the types of surfaces our system can create.

Similar to inflation surfaces, it is possible to sketch silhouette and cross-section contours and construct a surface from those [CSSJ05, RDI10, AS11]. A more general version of this allows the user to explicitly build up a "scaffold" of orthogonal planes for sketching curves on (no surface is built) [SKSK09]. The user can mimic this type of construction in our system, albeit not from a single view or as quickly, by drawing the silhouette curve, then explicitly placing planes for the contours.

We use the Hermite RBF formulation [BMS*10] to create surfaces from the curve network. We provide a more extensive and complete curve editing system, and a mix of interactive and automatic approaches (as opposed to purely automatic) for establishing the normal orientations. We have also automatically found patches [AJA11], demonstrating that our curve networks could also be used in other techniques requiring consistent curve networks [RSW*07,LD09, SWZ04].

4. Sketching (stroke inference engine)

In this section we describe how we process strokes into curves. *Strokes* are 2D curves, made by a single pen down, draw, pen up action. *Curves* are 3D entities with a defined normal direction, and are built up out of one or more processed strokes. The goal is to mimic, as best as possible, the freedom of pencil and paper while still supporting the creation of well-behaved curves from the user's individual strokes. When the user draws a stroke, it is analyzed to determine if it should create a new curve, be added to an existing curve (extending or oversketching), or join together two existing curves. We break this analysis up into the following steps: (pre-process) If the user scratches back and forth, first convert this to a smooth stroke (Section 4.1), 1) For each curve, determine if it makes sense to apply the stroke to the curve, and if so, how (merge, oversketch, close). 2) From all candidate curves, pick the best option (including creating a new curve, or combining two curves together with the stroke). 3) Actually apply the stroke to the 3D curve by first promoting the stroke to 3D, then merging the result. See Figure 7 for a flowchart of the steps. We discuss each of these steps in turn.

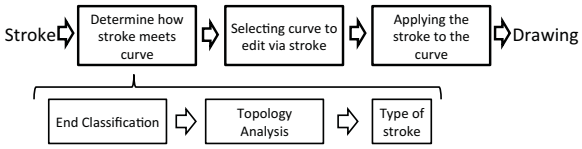

Figure 7: *Steps for incorporating stroke into existing drawing*

Note that the analysis is primarily in 2D — from the user's point of view they are sketching on a 3D drawing projected on the image plane using the current view direction, so all decisions about joining should be made with respect to the projected curves. We only use 3D information in order to avoid adding strokes to curves that are largely perpendicular to the current view. The other place we use 3D is merging the stroke into an existing curve; in this case, the 3D information for the stroke is gleaned from the 3D curve. Only after the stroke is promoted to 3D do we actually join it to the curve.

Parameters: Devices differ in how accurate they are; window size relative to tablet size can also play a role. We provide the user with two intuitive controls for specifying the behavior of they inference system. The first is a 2D distance threshold d, defined in pixels (see yellow circle in Figure 9). The second is a smoothing parameter N which determines how much smoothing is applied. All other parameters are empirically derived from d and N by experimentation; essentially, draw "good" and "bad" edits, look at the values, and set thresholds.

Representation: All of our curves, both 2D and 3D, are stored as just a list of points — we do not fit curves to them. We do apply some amount of smoothing (based on N) and re-sampling (to ensure at least 3 points per d interval). We assume our strokes and curves are arc-length parameterized on the range $[0,1]$.

4.1. Preprocessing for back and forth scratching

Before applying the stroke to the curve we first need to see if the stroke itself needs to be processed because it was made using a back and forth scratch motion. This can be detected by seeing if the stroke folds back on itself. Note that we only

Figure 8: *From left to right: Dividing the stroke into sections then extending each section using the projection operator. Simple averaging leaves "scallops" and does not allow for intentional corners. Strokes made from blends can be combined with other strokes. Strokes that overlap substantially are also treated as blends.*

do this check if the user had enabled it. If the stroke does fold back on itself, we convert it to a non-folding stroke. Unlike [OSJ11], we do not use curve fitting, but instead break the stroke into pieces and then "glue" the pieces together. We define a fold-over as a section of a stroke that has corners at either end and is within a distance d of the another part of the stroke (or falls off the end). This distinguishes a fold-over from a corner in the curve (see Figure 8). To find fold-overs we first use the Short Straw [WEH08] algorithm to find corners, then break the stroke into pieces at the corners. We then check each section to see if lies on top of the previous section; if not, i.e., it was a corner and not a fold-over, we join that section back up to the previous one. Note that a section is allowed to extend past the previous section.

Once we have broken up the stroke into its fold-over sections, we need to construct a single, non-folded stroke from the pieces. It is tempting to simply apply some sort of weighted averaging to the points, but this tends to result in a scalloped look (Figure 8, middle top) because the section ends often "stick up". Also, averaging everywhere loses the flavor of the original stroke and flattens it out. Instead, we extend each section at either end, then slowly morph the sections towards each other using a projection operator, moving the ends more than the middles. Once the sections are in agreement, we can sort the points topologically and downsample to reduce the number of points.

More specifically, we define an averaging function that takes in a set of sections and a point. The point is projected to each of the sections, and only those points that project to the interior of the section are kept. The section points are weighted by their distance to the input point and averaged.

To initially extend a section, construct a Hermite curve that joins the end point to the average curve. One end of the Hermite curve is the end point and tangent of the section end. The other end is found by taking a point d distance along the tangent and projecting it using the averaging function above (see Figure 8, middle). If the tangent extends past the end of the average curve, just use the point along the tangent. The distance d is the selection distance size. The tangent at the second point is the average of the tangents at the projection point. Both tangents are scaled to be length d. Sample the Hermite curve with 6 points and add those points to the section.

To blend a section with the average curve, we move each point on the section towards the projection point. The middle 50% of the curve (by arc length) is moved 0.25 towards the projected point, the ends are moved by 0.75 (the movement amount is linearly interpolated). Each section is also smoothed before averaging. We apply the projection operator 3 times before extracting the final curve.

This fold-over processing is used in one other place – if the current stroke overlaps more than 90% with the last *stroke* (not curve). If this is the case, we walk back in the list of strokes until we have a complete list of all of the overlapping strokes. Then we merge all of the strokes into a single, non-folding stroke and then proceed as before (see far right of Figure 8). This is preferable to treating these strokes as overlaps because the new stroke should not over-ride the last one, but should be blended with it.

4.2. Determining how the stroke meets the curve

The goal of this section is to determine if it makes sense to apply a stroke to a curve, and if so, how. We break the decision-making into three steps: End-classification, Topology, and Type (see Figure 9). The output of this analysis is the type. In the end-classification step we determine if one, or both, of the end-points of the stroke meet the curve smoothly. In the second stage we rule out cases where both ends of the stroke meet the curve, but not in any way that makes sense. In the final step we determine what type the stroke is, based on the end-classifications, whether it meets the curve once or twice, where (ends or middle) and whether or not it overhangs the end of the curve. Recall that we are working with projected 3D curves, and a 2D stroke, so all equations are in 2D.

End-classification: We support two types of stroke-curve meetings. The first is a *merge*: The end of the stroke starts near the curve, then travels along it for some distance without back-tracking. The second is a *join*: The end of the stroke does not overlap the curve, but the stroke and the curve ends can be joined with a short "nice" arc (see Figure 9). We remind the reader that for all of the following, d is a screen-based selection distance specified by the user (yellow circle in the figure).

The merge test: Define the end of the stroke s_e as 1/3 the length of the stroke, or $3d$ along the stroke, whichever is smaller. Let $s_s \subset s_e$ be the largest contiguous region that is 1) within distance $1.5d$ of the curve, 2) does not fold back

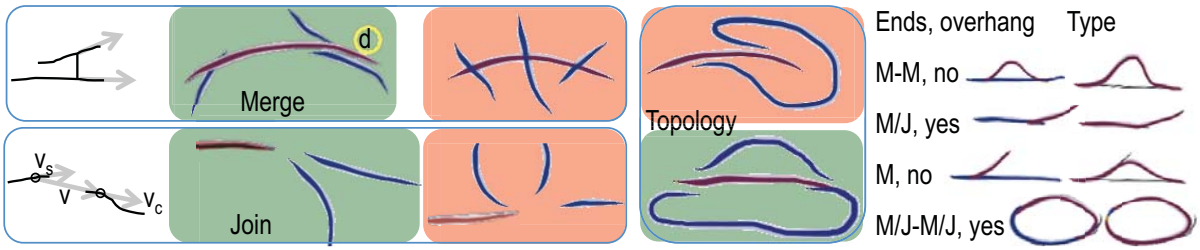

Figure 9: *Left: Variables for defining a merge and a join. Middle: Examples of good and bad merges, joins, and topology. The purple mark is the curve, the blue marks are the incoming strokes. Red indicates there is not a join or merge, green means potentially there is. Right: Determining type based on end conditions and overhangs.*

on the curve, 3) does not project off of the end of the curve, 4) who's angle with the closest point on the curve is less than $3/4\pi$. is within distance d of the curve. Let s_c be the corresponding part of the curve the stroke projects to, d_a be the average distance, and α_a be the average angle. Then it is *not* a merge if any of the following are true:

$$\begin{aligned} ||s_s|| &< (1/4)d \text{ and } d_a < 0.1d \\ ||s_s|| &< (3/4)d \text{ and } d_a < 0.2d \\ d_a &> 1.1d \text{ or } \alpha_a > 3/4\pi \\ ||s_s|| &= 0 || ||s_s|| < ||s_c||/2 \end{aligned} \quad (1)$$

The first three tests rule out strokes that meet at right angles or are too far away, the last test makes sure the stroke is not projecting to different parts of the curve.

The join test: Since strokes and curves can have small "hooks" at the end we actually search for the best join between end of the stroke and the end of the curve (up to $2d$ in from the end). Given a point p_s on the end of the stroke and p_c on the end of the curve, we define a good join as follows. Let $d_j = ||p_s - p_c||$, $v = (p_c - p_s)/d_j$, and $v_{s,c}$ be the unit tangent at p_s and p_c respectively. Using $\alpha_{s,c} =< v, v_{s,c} >$ define two terms: α_d is the how well-balanced the two angles are and α_t measures the total angle. All of the following must be true for a valid join:

$$\begin{aligned} \alpha_d = (|\alpha_c - (\alpha_c + \alpha_s)| + |\alpha_s - (\alpha_c + \alpha_s)|)/2 &< 0.2 \\ \alpha_t = (\alpha_c + \alpha_s) &< 0.3 \\ 0 < d_j &< 8d \end{aligned} \quad (2)$$

From all of the valid joins we pick the one with the best score $0.2\alpha_d + 0.8\alpha_t$. Additionally, any join with $\alpha_c < \pi/2$ out-scores one with $\alpha_c > \pi/2$.

Topology: It is possible for both ends of the stroke to meet the curve well from a geometric stand-point, but still not be valid. Orient the stroke so that the tangents at the start of the stroke line up with the curve. If the *other* end of the stroke is now pointed in the opposite direction with respect to the curve, then we rule out that stroke.

Type: To analyze how the stroke meets the curve we need one more piece of information — if the stroke overhangs the end of the curve. This is true if, while walking along the stroke, you never back track or fold-over with respect to the curve, and at some point you walk off of the end of the curve.

We assume here that the stroke is oriented in the same direction as the curve. The second and third types are special cases of the first two:

- Oversketch: An over-sketch exists if both stroke ends merge with the curve *and* those merge directions are the same with respect to the curve *and* the start of the stroke merges with the curve before the end of the stroke (no overhangs). If the curve is already closed then the latter check is not needed (the stroke over-strokes where the curve ends meet).
- Extend: One end of the stroke either merges and overhangs or joins with the curve; the other end does not merge or join.
- Partial over-sketch: One stroke end merges with the curve but does *not* overhang the end of the curve.
- Extend and close: The second end *does* merge or join with the curve, but at the other end of the curve and there is an overhang for both ends.

4.3. Select curve to edit

In the previous section we applied the stroke to an individual curve; in this section we determine, out of all of the possibilities, which is the best. In addition to a stroke applied to a single curve there are a couple of other possibilities we check: 1) The stroke forms a new closed curve (check if each end of the stroke merges or joins with the other end). 2) The stroke combines two curves into one. 3) The stroke extends the previous *stroke* (if the last action was a stroke). 4) The stroke overlaps the previous *stroke* by at least 90%; in this case we blend the stroke to the previous one(s) to create a stroke first (Section 4.1). We use 3) for creating oversketch strokes from multiple strokes; in this case, we merge the strokes together before applying the resulting merged stroke to the curve as an oversketch.

Essentially, we score each valid stroke-curve possibility

(see below) and choose the one with the lowest score. If there were no valid possibilities, or the best scoring possibility is a closed new stroke, we create a new curve. We add a few exceptions to this. If the stroke can be applied to the last edited curve (or stroke), we always do that. If a stroke combines two curves, we do that next. We also rule out any curves behind the drawing plane (if it is visible). After that, we pick the join with the lowest score. We do not use the score to rule out possibilities, but we will use it to control how much smoothing is done to the join.

To create the score for each match we use a combination of the end merge and join information (q_m) and the depth values of the projected curve (q_d). In general, we prefer matches with curves that do *not* extend backwards along the view direction at the join point. The score for a merge is $q_m = 0.7d_a/d + 0.3\alpha_a/(3/4\pi)$ (see Eqn 1). The score for a join is based on how far apart the ends are. If they are really close or really far apart the score goes up. Specifically (see Eqn 2): Let $q_a = (\alpha_d/0.2)/4 + 3(\alpha_a/0.3)/4$ and $q_l = d_j/8d$. Then we have two scores for q_m, depending on how big q_l is:

$$q_l > 3/4 \quad use \quad q_m = (1 + (q_l - 3/4)/(1/4))q_a \tag{3}$$

$$q_l < 1/4 \quad use \quad q_m = (1 + q_l/(1/4))q_a \tag{4}$$

$$1/4 \leq q_l \leq 3/4 \quad use \quad q_a \tag{5}$$

The depth score q_d is based on how far back the curve is relative to the depth of all curves (the further back, the worse the score). For merges and joins we also add in a term that increases as the depth change increases. Let Z_m and Z_M be the minimum and maximum depth values of the entire 3D curve network, and z_m and z_M be the corresponding depth values for the curve. Then

$$q_d = \frac{1}{4}\frac{(z_m + z_M)/2 - Z_m}{Z_M - Z_m} + \frac{3}{4}\frac{z_M - z_m}{Z_M - Z_m} \tag{6}$$

We use just the first term for scoring over-sketches. If this is a join, not a merge, we double q_d.

The final score for a match is $q = 1/4q_d + 3/4q_m$ (averaging the merge or join scores if there is two).

4.4. Applying stroke to the drawing

Once we have determined what to do with the stroke, we need to actually apply it to the curve (or create a new curve). First, we add depth values to the stroke to create a 3D stroke. Second, we cut and paste the 3D stroke into the curve. We have three goals. The first is that the stroke blend smoothly into the curve both in the film plane and in depth. The second is that the 3D stroke, when projected back to the film plane, looks the same as the 2D stroke. Third, where the stroke edits the curve (as opposed to extending it), it only edits it in the direction of the film plane, i.e., if you looked at the curve from the top (or bottom) it would look the same after the edit.

Traditionally, the problem of merging 2D curves from different views has been treated as follows: Try to find a 3D point so that, for each pair of corresponding points on the two curves, the 3D point projects to the 2D points. We keep the spirit of this approach, but formulate it differently because one of our curves is already 3D (refer to Figure 6).

During the stroke processing we project the stroke end onto the curve, finding for each stroke point in s_e a matching point on the curve. We use these overlap regions to assign depth values to the stroke, interpolating or extrapolating to the remainder of the stroke.

4.4.1. Adding depth values to the stroke

Specifically, position a plane in the scene (usually the film plane, but it can be the draw plane as well). Assign a depth value to each point on the 3D curve as follows. Cast a ray from the camera through both the 3D point on the curve and the plane. The (signed) depth value is the distance between those two points. Now take a point on the 2D stroke. Find the depth value of the corresponding point on the 3D curve. Cast a ray from the camera through the 2D stroke point and plane. Walk out from the plane by that depth value.

Exactly how depth values are added to the stroke depends on the type of operation. Essentially, where the stroke overlaps or projects onto the curve we use the curve's depth values. Where the stroke falls off of the curve, we either extrapolate the curve's depth values (no draw plane visible) or use zero, essentially placing the stroke on the draw plane (after first moving the draw plane so that it intersects the end of the curve).

To extrapolate the depth values, we use the average depth change per unit step in the film plane. To establish the correspondence we either use the closest point between the stroke and the projected curve, or, if the stroke folds over with respect to the curve, we use the arc-length parameterization of the curve. Once we get the depth values, we filter them several times before reconstructing the curve.

4.4.2. Joining curves

Once the 2D stroke is promoted to 3D, it needs to be merged or joined to the exisiting curve. The basic idea here is to search for a 3D Hermite curve that blends smoothly with the original curve and the stroke. This is done in 3D to make sure that the join is smooth in all dimensions. The optimization function is $0.2\alpha_d + 0.8\alpha_t$ for a join, and $0.8\alpha_d + 0.2\alpha_t$) for a merge (see Eqn 2) The tangent lengths of the Hermite curve are set to 1.5 of the length of the join, or 0.5 if the curve will "zig zag", ie, $< v \times v_s, v \times v_c < 0$ (see Figure 9).

The search region depends on the selection distance d and how good the merge or join is, scaling from d to $4d$ for a bad

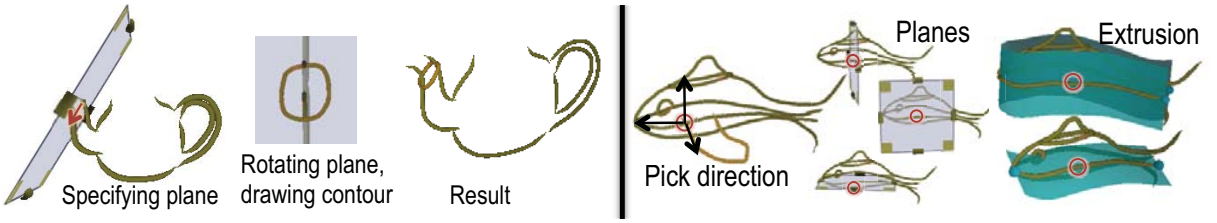

Figure 10: *Creating a cross-section curve by using the curve menu to define a plane (click on the first curve, pick option A, then drag to a second curve. Then click on the middle plane handle to automatically rotate to look down the plane normal. Draw the cross section. The user can also create a drawing plane or extrusion surface (right).*

merge score. The starting point for the search is the middle of the region where they overlap (merge) or from the endpoint of the curve (join).

4.5. Implementation details and parameters

Recall that all curves and strokes are stored simply as lists of 3D or 2D points. Because different devices have different amounts of noise and sampling rates, we need some mechanism for controlling the quality of the samples and their frequency. The user supplies two parameters (set via sliders): The selection distance d as a number of pixels, and a smoothing rate N. As the strokes are processed, we perform the following three actions: 1) inserting samples to ensure the sampling rate is at least 10 points per screen distance d (in case the pen input "skips", 2) smoothing some amount based on both N and the quality of the data, and 3) re-sampling to ensure even-spacing in 3D.

Smoothing is applied in three places: 1) To the ends of the stroke before computing merge and join information, 2) to the projected depth values while promoting the stroke to 3D, and 3) to a region around the joins. The system applies N rounds of smoothing to the stroke before processing, $2+N$ rounds of smoothing for the depth, and $1+N+40q$, where q is the merge or join score from Section 4.3 for the join (q is typically in the range 0.01 to 0.1). One round of smoothing moves the point 0.2 towards the average of the neighbors. For the depth smoothing, we construct the 3D points, average, then re-project the point to the ray, rather than filtering the depth values directly.

We re-sample the strokes after promoting them to 3D. d is a screen-based sampling rate; by projecting a segment of length d onto a plane perpendicular to the view, centered in the middle of the curves, we can convert d to a 3D absolute distance d_3. When re-sampling, we ensure the are spaced $0.1d_3$ apart. Since d reflects, in some sense, how "accurate" the 2D device is, and how close is close enough, it is a good measure for how accurately to sample the data.

To speed up the stroke to curve calculations we keep a 3D bounding box for each curve. If the stroke lies a distance d outside of the projected bounding box then we ignore that curve.

5. 3D sketching

Creating a curve in 3D, especially one that does not lie on a single plane, is challenging both conceptually and from an interface standpoint. The tablet input is 2D, and the display only shows a 2D projection of the 3D curves, which can creates ambiguity in the depth dimension. The challenge is to support methods of 3D curve creation that are both controllable (ie, do not surprise the user) and flexible (ie, make it possible to create the curve you want). Fundamentally, there are two basic methods for creating 3D curves. The first is to project the 2D curve onto a 3D surface (typically a plane, but any surface works). The second is to sketch the curve from multiple directions and merge the result into a 3D curve that, ideally, projects to each of the drawings. In practice, this is nearly impossible because people do not create consistent drawings. Sketching on surfaces is usually fairly predictable, but often limits the types and complexity of the curves.

Our approach is to 1) provide a couple of methods for quickly creating and placing drawing surfaces, 2) use over stroking to *edit* curves from additional view points (rather than explicitly creating curves from different view points and trying to merge them) 3) support standard affine transformations (rotation, scale, translation), both in-plane and out of plane. The general work flow is to start with a handful of curves in a single plane, then use those curves to define surfaces to make initial curves that are *not* in the original plane, then use over stroking to further edit them. The user can also use transformations to pull curves out of the plane, and then over stroke to get the actual shape they want.

Editing by over sketching or extending is described in the previous section. The key observation is that we use the existing curve to construct depth values relative to a plane (usually the view plane) then apply those depth values to the stroke. This preserves, as best as possible, what the curve looks like when viewed from the `up` vector of the current view.

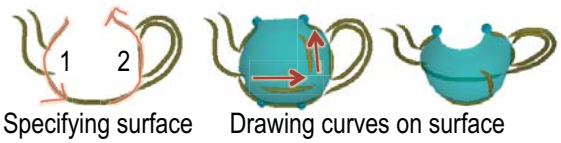

Figure 11: *Creating an inflation surface and drawing on it.*

5.1. Drawing surfaces

We discuss three methods of creating drawing surfaces: Planes, extrusion surfaces, and inflation surfaces.

As has been done many times in the past, we have a draw plane which can be explicitly positioned in the scene (see Figure 5). We use a direct edit approach (grab and move) to position the plane. The plane can be slid back and forth along its normal (squares in the corners), rotated about the plane's two axes (grab middle of the edge and pull towards or away from the plane center) or spun about (grab middle of the edge and pull *along* the edge). Tapping on the hot spots snaps the camera view to the plane. One simple method of creating curves is to move the draw plane forward in small increments, drawing cross-sections each time. This is, however, a vary tedious method for creating surfaces, and in general moving the plane around by hand is painful.

We support two more useful methods of positioning the draw plane. The first is to pick a point on a curve and a direction (tangent, normal, bi-normal at that point). This snaps the plane to that point, with the normal pointing in the second direction. The second is to pick *two* points in the scene (usually from two different curves, but not always). The plane is positioned so that it passes through those points and is as perpendicular as possible to the two tangent directions of the curves. This makes it simple to add cross sections to two silhouette curves (see Figure 10). Both actions are off of the curve menu (tap on the curve, tap on the Align option, select the direction, OR, tap on the curve, drag from the align option to the second point), usually followed by a tap on the draw plane controls to align the view with the draw plane, or a rotate of the camera.

The extrusion and inflation surfaces are mode-based (the user explicitly says they are creating one). The extrusion surface works by extruding a curve in a given direction chosen by the user [BBS08]. Once the surface is up, the user can draw as many curves on it as they want, or click elsewhere to pick another surface.

The inflation surface is a simplified version of the surface created by inflation-based methods [IMT99, NISA07, JC08]. We implemented the inflation surface to support contour lines typically drawn by artists (see Figure 11) in the interior of a round object. The user strokes where they want the left side to be, then the right side, then the system creates a surface that joins those two curves, bulging towards the viewer in the middle. Note that our quick inflation surface approximation has several advantages to the inflation-based methods cited above. While we expect existing curves to be underpart of the left and right user strokes, they do not need to correspond to *specific* curves in the scene. The strokes can even pass over gaps between the existing curves (eg, the spout) – the depth values will be interpolated in this region. The curves do not even need to be planar. These lenient input requirements allow us to create inflation surfaces at will for arbitrarily complex curves.

Inflation Surface Implementation: The system builds a temporary 3D curve for each stroke (super-imposed on the left- and right-side existing curves), then joins pairs of points on the curves with a half-circle. The radius of the half-circle is one-half the distance between the two points, with starting tangents in the view direction. Arc-length parameterization is used to determine which pairs of points to use. The series of half-circles are stitched together to form a ruled, triangular mesh surface. For reasonable curves this results in a non self-intersecting surface; we do not enforce this condition, though.

The temporary curves are built using a variation of the depth assignment in Section 4.4. For each stroke point, we cast a ray into the scene and find the closest curve point in depth. The selection is "fat" (within $2d$). If no intersection is found the depth values are interpolated or extrapolated from nearby depth values. The result is a 3D curve that "tracks" the curves under the stroke.

6. Surfacing

We have experimented with two methods of surfacing a model. The first requires a curve network with no "dangling" curves [AJA11], the second is an implicit Hermite formulation [BMS*10]. In both cases, curves that cross near each other need to be snapped together, and in the latter case, we also want to know the desired surface normal at sampled points. We provide tools for explicitly snapping curves together and for searching for potential intersections, which can be fixed by clicking on them. We also provide tools for explicitly setting the normal at points along the curve (see Figure 12). In general, the Frenet frame is *not* the best way to define the normal; besides being notoriously unstable, we actually want a normal that represents the desired surface normal at that point, not the curve normal.

We define default normals as follows. If the curve lies in a plane than we define the normal by taking the cross product of the tangent and the plane normal. This gives the direction; for orientation we take the direction that points "out" from the centroid, as measured at the mid-point of the curve. For non-planar curves we take the centroid of the curve and find points on the curve where the tangent is perpendicular to the vector v from the point to the centroid. We then define the normal at that point as v. In addition, whenever the user

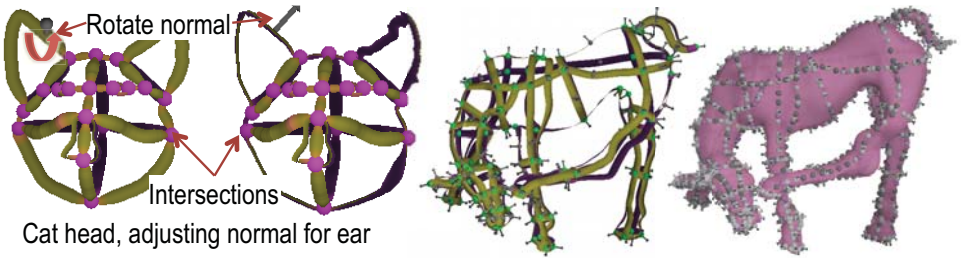

Figure 12: *Left: Purple dots indicate snapped intersections. The user can place a normal constraint on the curve (ear) and rotate to change the local surface normal. Right: Green dots are snapped intersections with normal constraints, green arrows without dots are additional normal constraints. Far right: All of the point, tangent (dark gray) and normal (arrow) constraints used to generate the RBF surface.*

snaps two curves together we also define a normal at that point. We could use the cross product of the tangents at that point, but this can be unstable when the curves don't meet orthogonally. Instead, we take the 10 points on each curve around the intersection point and fit a plane to those points. The user is free to grab and move this normal in any direction; the actual normal used at the curve is found by taking the rejection with respect to the tangent.

To propagate normals along the curve we do not linearly interpolate; instead, we do a spherical interpolation of the normals (then ensure they are perpendicular to the tangent).

Once the curves are snapped together and the normals specified the user can generate a surface using the Hermite RBF formulation [BMS*10]. We use three constraints: point, normal, and tangent [Mac11] (see far right of Figure 12). We generate point samples at least every 6 point samples, or if the skipped samples deviate by more than the average sample distance from a line fitted to the points. We generate a tangent constraint whenever the dot product of the current tangent and the last one differs by more than 0.4, or every 12 point samples. We generate normal constraints similarly, except we use 0.1 for the maximum allowable normal difference. This always results in a water-tight, manifold surface, even for dangling curves, although the topology may not be correct and the surface may "bulge" unexpectedly. Surface generation takes between a few seconds and two minutes for our examples, depending both on the number of curves and the size of the marching cubes used to generate the surface.

We have also patched [AJA11] the curve networks that do not have dangling curves (the horse and the dragon head).

7. Menus

The overall interface goal was to minimize the amount of time spent doing non-sketching tasks (menu selection, mode changes, pen taps, etc). We use a variation of in-screen pie-menus [MZL09, KB94] for changing the behavior of the last stroke, curve editing, and camera control. Unlike traditional pie menus, we combine menu selection with Cross-y [AG05] style selection to minimize pen taps. For example, to select a region of a curve and drag it, the user 1) clicks on the curve to bring up the curve menu 2) pen-down on the drag option 3) drags over the part of the curve to drag then 4) releases the pen. At this point they can drag the curve selection using the drag icon (see Figure 4). Additionally, doing a pen-down on the top of the drag icon produces an in-out drag, while a pen-down elsewhere performs an in-plane drag. This is a total of three pen actions for selecting a curve, selecting the region on the curve to edit, and whether they want to edit in the plane or out of plane.

In general we support three menu selection and edit techniques, which can result in different behaviors. The first action is to simply pass the cursor out of the menu over the desired selection. This typically does the most global option, such as centering the camera on all the curves or snapping the camera to the closest view. The second action is to tap on the menu option. This can (optionally) bring up additional controls, such as the selection arrows for the plane (see Figure 10). The third action is a pen down and drag. This is used for direct manipulation of the camera and for selection, for example, selecting which curve to center the camera on.

We support one other menu action, which is a version of the lasso technique. Similar to a pen-drag, the user does a pen-down on the center curve option, but then drags over a set of curves, and releases the pen close to the original menu item. This selects all of the curves the pen crossed over, not just the latest one.

We have three menus in total; a stroke menu invoked from selection circles at the end of the last stroke, a curve menu invoked by clicking anywhere on a curve (which also selects the curve), and a camera menu, invoked by clicking on the background. See the instructions supplemental materials for details on the menu options.

Figure 13: *Examples made with the system. Surfaces are made using the Hermite RBF formulation.*

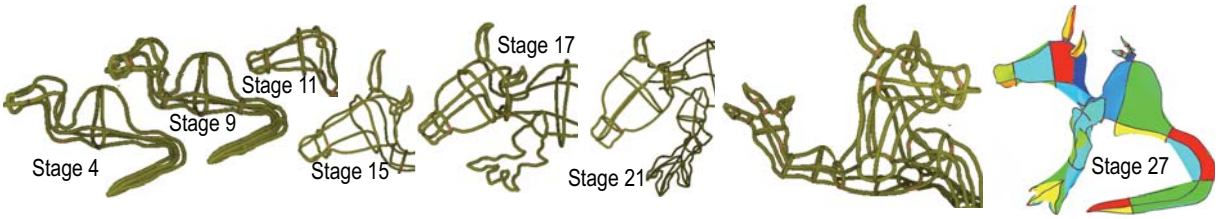

Figure 14: *Example construction sequence (3-4 hours editing time).*

8. Visuals and interface elements

We have created three distinct visual styles, one each for stage: 2D drawing, 3D curve editing, and 3D curve network snapping. The 2D drawing style is simply view-facing strokes with a texture. The 3D curve editing style has both warm-cool shading (the curves are rendered as tubes) and depth-based shading. The normalization stage uses ribbons — basically the top part of a very wide tube passing through the curve and oriented with the tangent plane. The rendering style here is yellow on the front, purple on the back, with a bit of warm-cool shading.

To provide a sense of 3D we have a *shadow box*, which is three sides of a box centered around the current selection. The shadow box provides some depth cues (the curve shadows are drawn on the left and bottom sides) and spacing cues (through optional grids on the sides), and spatial cues on the location of the drawing plane. We also use the shadow box to provide in-screen control of the camera (separate from the camera menu). The user can do axis-aligned and trackball rotation using the hotspots on the edges and corners, as well as zoom (lower left). There are also two hotspots (lower left and lower right) for snapping the draw plane to the center of the box and drawing or undrawing the draw plane.

9. Results and discussion

We have not performed a formal user study. However, we have shown the system extensively to four experienced users (and allowed them to experiment with the system as they wish). Informal feedback suggests that the 2D drawing aspect is particularly compelling. One user was able to make a simple 3D head-shape after a few minutes; comments from this user (and others) on the 3D portion of the system is that they wanted a handful of "quick-starts" (curve networks in default configurations) plus basic curve transformations (which were not implemented at that time).

We are fairly confident that the 2D drawing aspect of JustDrawIt is easy to use and intuitive for traditional artists. We expect that the 3D aspect of JustDrawIt may have a slightly steeper learning curve. The benefit of this steeper leaning curve is that our 3D curve drawing system allows complete control over the sketched curve in order to make careful, detailed edits, which is crucial in order to support workflows of discerning artists. However, we believe that as future work, we can make the 3D drawing experience even simpler for the novice user by incorporating some of the work on single-view sketching and supplying some standard "quick-start" curve networks.

All the examples in this paper were made by one user with a four-year degree in art, and the system was tuned to optimize that user's experience. As a next step, we would like to conduct a user study to gather feedback from a large number of users with a wide range of artistic abilities. The system as a whole could then be fine-tuned to optimize the experience for most users. During the course of the user study, we plan to track when users reject or select a different option. We can then apply machine learning to this information to learn better thresholds and parameters for example, for the end-classification in Section 4.2).

References

[AG05] APITZ G., GUIMBRETIÈRE F.: Crossy: a crossing-based drawing application. In *ACM SIGGRAPH 2005 Papers* (New York, NY, USA, 2005), SIGGRAPH '05, ACM, pp. 930–930. URL: http://doi.acm.org/10.1145/1186822.1073286, doi:http://doi.acm.org/10.1145/1186822.1073286. 11

[AJA11] ABBASINEJAD F., JOSHI P., AMENTA N.: Surface patches from unorganized space curves. *Comput. Graph. Forum 30*, 5 (2011), 1379–1387. 5, 10, 11

[AS11] ANDRE A., SAITO S.: Single-view sketch based modeling. In *Proceedings of the Eighth Eurographics Symposium on Sketch-Based Interfaces and Modeling* (New York, NY, USA, 2011), SBIM '11, ACM, pp. 133–140. URL: http://doi.acm.org/10.1145/2021164.2021189, doi:http://doi.acm.org/10.1145/2021164.2021189. 5

[BBS08] BAE S.-H., BALAKRISHNAN R., SINGH K.: Ilovesketch: as-natural-as-possible sketching system for creating 3d curve models. In *Proceedings of the 21st annual ACM symposium on User interface software and technology* (New York, NY, USA, 2008), UIST '08, ACM, pp. 151–160. URL: http://doi.acm.org/10.1145/1449715.1449740, doi:http://doi.acm.org/10.1145/1449715.1449740. 2, 4, 10

[BMS*10] BRAZIL E. V., MACEDO I., SOUSA M. C., DE FIGUEIREDO L. H., VELHO L.: Sketching variational hermite-rbf implicits. In *Proceedings of the Seventh Sketch-Based Interfaces and Modeling Symposium* (Aire-la-Ville, Switzerland, Switzerland, 2010), SBIM '10, Eurographics Association, pp. 1–8. URL: http://dl.acm.org/citation.cfm?id=1923363.1923365. 5, 10, 11

[BPCB08] BERNHARDT A., PIHUIT A., CANI M.-P., BARTHE L.: Matisse: Painting 2D regions for modeling free-form shapes. In *EUROGRAPHICS Workshop on Sketch-Based Interfaces and Modeling, SBIM 2008, June, 2008* (Annecy, France, June 2008), Alvarado C., Cani M.-P., (Eds.), pp. 57–64. 2

[CMZ*99] COHEN J. M., MARKOSIAN L., ZELEZNIK R. C., HUGHES J. F., BARZEL R.: An interface for sketching 3d curves. In *Proceedings of the 1999 symposium on Interactive 3D graphics* (New York, NY, USA, 1999), I3D '99, ACM, pp. 17–21. URL: http://doi.acm.org/10.1145/300523.300655, doi:http://doi.acm.org/10.1145/300523.300655. 2, 4

[CSSJ05] CHERLIN J. J., SAMAVATI F., SOUSA M. C., JORGE J. A.: Sketch-based modeling with few strokes. In *Proceedings of the 21st spring conference on Computer graphics* (New York, NY, USA, 2005), SCCG '05, ACM, pp. 137–145. URL: http://doi.acm.org/10.1145/1090122.1090145, doi:http://doi.acm.org/10.1145/1090122.1090145. 5

[GH98] GRIMM C., HUGHES J.: Implicit generalized cylinders using profile curves. In *Implicit Surfaces* (June 1998), pp. 33–41. Creating sweep surfaces using sketching. 3

[Gri11a] GRIMM C.: *Results of an observational study on sketching*. Tech. Rep. WUCSE-2011-57, Washington University in St. Louis, June 2011. 1, 2

[Gri11b] GRIMM C.: Results of an observational study on sketching (poster). In *SBIM '11* (2011). 1, 2

[IMT99] IGARASHI T., MATSUOKA S., TANAKA H.: Teddy: a sketching interface for 3d freeform design. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1999), SIGGRAPH '99, ACM Press/Addison-Wesley Publishing Co., pp. 409–416. URL: http://dx.doi.org/10.1145/311535.311602, doi:http://dx.doi.org/10.1145/311535.311602. 2, 4, 10

[JC08] JOSHI P., CARR N. A.: Repoussé: Automatic inflation of 2d artwork. In *SBM* (2008), pp. 49–55. 2, 4, 10

[KB94] KURTENBACH G., BUXTON W.: User learning and performance with marking menus. In *Proceedings of the SIGCHI conference on Human factors in computing systems: celebrating interdependence* (New York, NY, USA, 1994), CHI '94, ACM, pp. 258–264. URL: http://doi.acm.org/10.1145/191666.191759, doi:http://doi.acm.org/10.1145/191666.191759. 3, 11

[KHR04] KARPENKO O., HUGHES J. F., RASKAR R.: Epipolar methods for multi-view sketching. In *Proceedings of Eurographics workshop on Sketch-based Interfaces and Modeling (SBIM '04)* (Grenoble, France, 2004), Jorge J. A. P., Galin E., Hughes J. F., (Eds.), Eurographics Association, pp. 167–173. URL: http://diglib.eg.org/EG/DL/WS/SBM/SBM04/167-173.pdf. 2, 4

[LD09] LAVOUÉ G., DUPONT F.: Technical section: Semi-sharp subdivision surface fitting based on feature lines approximation. *Comput. Graph. 33* (April 2009), 151–161. URL: http://dl.acm.org/citation.cfm?id=1528932.1529113, doi:10.1016/j.cag.2009.01.004. 5

[Mac11] MACEDO I.: *Generalized interpolation of implicit surfaces using radial basis functions*. PhD thesis, IMPA, january 2011. 11

[MZL09] MARINKAS D., ZELEZNIK R. C., LAVIOLA JR. J. J.: Shadow buttons: exposing wimp functionality while preserving the inking surface in sketch-based interfaces. In *Proceedings of the 6th Eurographics Symposium on Sketch-Based Interfaces and Modeling* (New York, NY, USA, 2009), SBIM '09, ACM, pp. 159–164. URL: http://doi.acm.org/10.1145/1572741.1572768, doi:http://doi.acm.org/10.1145/1572741.1572768. 3, 11

[NISA07] NEALEN A., IGARASHI T., SORKINE O., ALEXA M.: Fibermesh: designing freeform surfaces with 3d curves. In *ACM SIGGRAPH 2007 papers* (New York, NY, USA, 2007), SIGGRAPH '07, ACM. URL: http://doi.acm.org/10.1145/1275808.1276429, doi:http://doi.acm.org/10.1145/1275808.1276429. 2, 4, 10

[OS10] OLSEN L., SAMAVATI F. F.: Stroke extraction and classification for mesh inflation. In *Proceedings of the Seventh Sketch-Based Interfaces and Modeling Symposium* (Aire-la-Ville, Switzerland, Switzerland, 2010), SBIM '10, Eurographics Association, pp. 9–16. URL: http://dl.acm.org/citation.cfm?id=1923363.1923366. 2, 4

[OSJ11] OLSEN L., SAMAVATI F., JORGE J.: Naturasketch: Modeling from images and natural sketches. *IEEE Comput. Graph. Appl. 31* (Nov. 2011), 24–34. URL: http://dx.doi.org/10.1109/MCG.2011.84, doi:http://dx.doi.org/10.1109/MCG.2011.84. 2, 4, 6

[OSSJ09] OLSEN L., SAMAVATI F. F., SOUSA M. C., JORGE J. A.: Sketch-based modeling: A survey. *Computers & Graphics 33*, 1 (2009), 85 – 103. URL: http://www.sciencedirect.com/science/article/pii/S0097849308001295, doi:10.1016/j.cag.2008.09.013. 2, 4

[RDI10] RIVERS A., DURAND F., IGARASHI T.: 3d modeling with silhouettes. In *ACM SIGGRAPH 2010 papers* (New York, NY, USA, 2010), SIGGRAPH '10, ACM, pp. 109:1–109:8. URL: http://doi.acm.org/10.1145/1833349.1778846, doi:10.1145/1833349.1778846. 5

[RJ02] RUBIO J. M., JANECEK P.: Floating pie menus : Enhancing the functionality of contextual tools. *Learning* (2002), 39–40. URL: http://www.acm.org/uist/archive/adjunct/2002/pdf/demos/p39-rubio.pdf. 3

[RSW*07] ROSE K., SHEFFER A., WITHER J., CANI M.-P., THIBERT B.: Developable surfaces from arbitrary sketched boundaries. In *Proceedings of the fifth Eurographics symposium on Geometry processing* (Aire-la-Ville, Switzerland, Switzerland, 2007), Eurographics Association, pp. 163–172. URL: http://dl.acm.org/citation.cfm?id=1281991.1282014. 5

[SKSK09] SCHMIDT R., KHAN A., SINGH K., KURTENBACH G.: Analytic drawing of 3d scaffolds. In *ACM SIGGRAPH Asia 2009 papers* (New York, NY, USA, 2009), SIGGRAPH Asia '09, ACM, pp. 149:1–149:10. URL: http://doi.acm.org/10.1145/1661412.1618495, doi:http://doi.acm.org/10.1145/1661412.1618495. 5

[SWSJ05] SCHMIDT R., WYVILL B., SOUSA M. C., JORGE J. A.: Shapeshop: Sketch-based solid modeling with blobtrees. In *Proceedings of Eurographics workshop on Sketch-based Interfaces and Modeling (SBIM '05)* (Dublin, Ireland, 2005), Jorge J. A. P., Igarashi T., (Eds.), Eurographics Association, pp. 53–62. URL: http://doi.acm.org/10.1145/1281500.1281554, doi:10.2312/SBM/SBM05/053-062. 4

[SWZ04] SCHAEFER S., WARREN J., ZORIN D.: Lofting curve networks using subdivision surfaces. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing* (New York, NY, USA, 2004), SGP '04, ACM, pp. 103–114. URL: http://doi.acm.org/10.1145/1057432.1057447, doi:http://doi.acm.org/10.1145/1057432.1057447. 5

[WEH08] WOLIN A., EOFF B., HAMMOND T.: *Shortstraw: A simple and effective corner finder for polylines*. 2008, p. 33Ð40. URL: http://www.eecs.ucf.edu/courses/cap6938/fall2008/penui/handouts/asgn2.pdf. 6