

Bartosz Fabianowski · John Dingliana

Sketching Complex Generalized Cylinder Spines

Abstract Generalized cylinders are a versatile class of objects commonly constructed from a spine and cross-sections orthogonal to it. We propose a novel method for the intuitive sketch-based specification of arbitrarily complex spines, including those that loop over and under themselves. A two-dimensional sketch of the spine is drawn using a pen and a graphics tablet first. This is surrounded by a swept-sphere bounding volume representing the generalized cylinder. Any overlapping sections are automatically offset perpendicular to the sketch plane, their ordering controlled by pen pressure. The user may adjust the resulting shape by oversketching or rotating the view and dragging points in the spine. All user input is processed by an optimization that generates a smooth, non-intersecting shape at interactive speeds.

Keywords Sketch-based modeling · Generalized cylinders · Space curves · Pen pressure

CR Subject Classification I.3.3 [Computer Graphics]: Line and curve generation · I.3.5 [Computer Graphics]: Modeling packages · I.3.6 [Methodology and Techniques]: Interaction techniques

1 Introduction

Sketch-based modeling allows three-dimensional objects to be built using planar strokes. Initial techniques focused on reconstructing rectilinear models from orthographic sketches [15]. Follow-up work introduced new modeling paradigms, most notably the inflation of sketched silhouettes into “puffy” shapes by Teddy [12]. For generalized cylinders, the prevalent construction method is extrusion of two-dimensional cross-sections along a spine.

While planar cross-sections are easy to sketch, spines are more difficult to handle. Current systems often rely on a single stroke for the spine [37], allowing a limited range of shapes to be drawn very quickly. Multi-stroke techniques are more flexible, but complex spines remain difficult or even

impossible to express [19]. We introduce a novel method for quickly and intuitively sketching arbitrarily complex spines, including those that loop over and under themselves. Our approach additionally helps to ensure that the resulting generalized cylinder will be free of self-intersections.

The user first draws an overhead view of the spine using pressure-sensitive pen and graphics tablet. This fully specifies the x and y coordinates of all its points, leaving only z unknown. Additionally, whenever two sections of the spine lie so close together that the surrounding generalized cylinder overlaps, pen pressure indicates which section should be placed above the other. An optimization automatically computes z values that vary as smoothly as possible while respecting the non-intersection constraints. In a second step, the user may interactively adjust z values by rotating the view and dragging points in the spine. Optimization is repeated after each change to ensure that the spine follows the user’s input smoothly.

All calculations are performed on a simplified bounding volume representation. The spine is modeled as a polyline and each segment surrounded by a capsule, the envelope of a sweep along it with a sphere. Efficient collision detection between capsules [22] allows the optimization to run at interactive speeds. Save for two hemispheres at the ends, the space occupied by the capsules corresponds to a generalized cylinder with constant circular cross-section. This bounding volume is well-suited to many modeling applications, including knots of strings or rope and animal limbs, whose cross-sections are often close to circular.

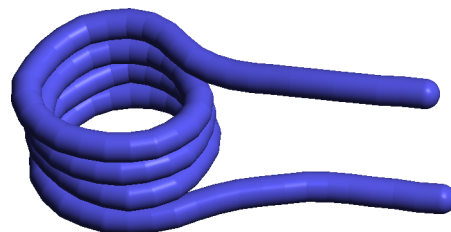


Fig. 1 A coil spring sketched using our system in mere seconds

2 Related work

Sketch-based modeling can follow two main approaches. Reconstructive techniques build three-dimensional objects from two-dimensional sketches by reversing the projection. This works for CAD-like models exhibiting symmetry and 90° angles [15], and also some curved objects [8, 16, 24].

Gestural methods interpret input strokes by instantiating and modifying building blocks instead. These may be domain-specific templates [25, 36], simple primitives such as cubes and cylinders [30, 37], reconstructions from sketches [23, 26], or members of object families parametrized by the strokes. In *Teddy* [12] and follow-up work [1, 7, 18, 29], the user obtains “puffy” shapes by inflation of their sketched silhouettes. Other families of objects often parametrized by strokes include surfaces of revolution [5, 32], ruled surfaces [9], and generalized cylinders.

The latter are commonly limited to sweeps along a planar stroke [30, 32, 37], although some methods exist that can provide three-dimensional spines. In [19], two sketches from different perspectives are fused to obtain a three-dimensional curve, while [6] evaluates sketches of a curve and its shadow. Neither technique is able to produce complex, looping spines. A three-dimensional spine may also be derived from a single stroke by projecting it onto auxiliary geometry [17] or using domain-specific heuristics to fill in the additional degree of freedom [28] at the cost of reduced control over the shape. Similarly, [7] internally builds a three-dimensional spine but gives the user no direct control over its depth coordinate. Arbitrarily complex spines can currently only be described by directly inputting three-dimensional coordinates using appropriate hardware [10, 21, 31] or a combination of multiple input devices [14].

Our approach computes the spine’s shape from planar input strokes by solving a constrained, nonlinear optimization problem. We use a numerical method [27] for ease of implementation and adaptability. After each iteration, intersections of the generalized cylinder with itself must be found and resolved. These have been formally analyzed [11], but collision detection for deformable objects remains computationally expensive [34]. Approximation of the generalized cylinder by a bounding volume made of capsules [22] allows for fast collision detection [20, 33]. Similar techniques may be found in surgical simulation [4], albeit with the very different goal of simulating highly domain-specific interaction, not general-purpose modeling.

3 Interface and interaction

Construction begins with the user drawing an overhead view of the desired spine using a pressure-sensitive pen and a graphics tablet. The spine’s coordinate system is aligned with this initial viewport, its x and y axes spanning the sketch plane and z pointing along the view direction. An orthogonal projection ensures that the sketch fully defines the (x, y) coordinates of each point in the spine, leaving z unknown. The

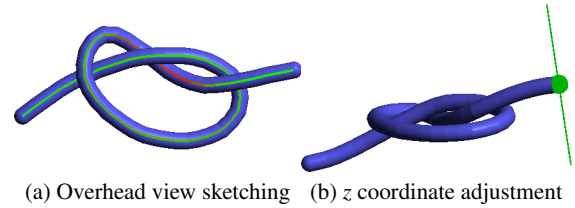


Fig. 2 Visualization during the two construction steps

input is sampled at short intervals and the spine modeled as a polyline. To accelerate computations, the surrounding generalized cylinder is represented by a bounding volume composed of capsules around the spine’s segments. Instant feedback is provided by visualizing the stroke together with a live preview of the generalized cylinder (figure 2a).

The spine nodes’ z coordinates are computed by an optimization that varies them as smoothly as possible while preventing the capsules from intersecting each other. Any two sections of the cylinder’s representation that overlap in the overhead view must be moved apart in the z direction. The desired ordering is indicated by pen pressure – the section whose spine was drawn with less pressure is placed above the other. Because drawing with very low pressure is difficult on some tablets, the comparison is omitted and the newer section always placed above the other if its pressure does not exceed 60% of the possible maximum.

Mistakes can be corrected by oversketching [2]. Any subsequent stroke beginning and ending on the generalized cylinder replaces the section of the spine between its endpoints. If the stroke ends on a blank section of the canvas, the spine is extended instead. Oversketching may be used to both correct minor mistakes such as the wrong pressure used for a short stretch of the spine or to significantly alter the spine. After every oversketching stroke, the capsules are regenerated to follow the spine’s new shape and optimization is rerun.

By sketching an overhead view, the user fully specifies the (x, y) positions of all nodes in the spine. The corresponding z coordinates initially are automatic guesses obtained by optimization. These may be adjusted freely in a second construction step by rotating the view and dragging nodes using the pen. Since only a single coordinate is to be modified, the dragging motion is projected onto the z axis, avoiding any ambiguity in its interpretation. The direction of adjustment is visually indicated to the user (figure 2b). A z coordinate may manually be specified for any number of nodes, giving the user complete control over the three-dimensional shape of the spine. To reduce the amount of input required and prevent self-intersections, optimization continues to compute z coordinates for all other nodes. Manually positioned nodes are highlighted when then pen hovers over the tablet and may be returned to automatic computation of their z values by depressing the pen without dragging. The first node of a new spine by default is positioned at $z = 0$ to provide an absolute reference point.

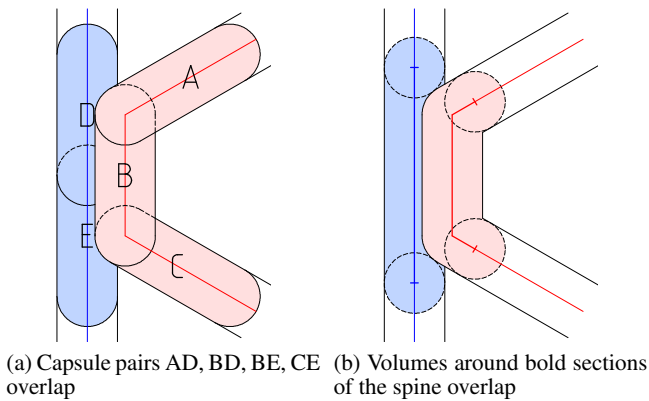


Fig. 3 The two notations used for an overlap

Spine construction is completed by computing tangential coordinate frames at the nodes that allow planar cross-sections to be positioned. We provide the user with a choice between the propagation of an initial frame using incremental rotation and Sloane’s technique [3]. Our current implementation covers construction of the spine only. The final generalized cylinder may be obtained by exporting the spine to another modeling application, such as Blender, and adding custom cross-sections there.

4 Preprocessing

Overlaps in the generalized cylinder are located by preprocessing the sketched overhead view. Any other information that will accelerate the subsequent optimization is also calculated at this stage. Processing occurs progressively as an initial stroke is being drawn and is then selectively repeated for parts of the spine affected by oversketching.

Our representation for the generalized cylinder is a series of capsules, which together form a swept-sphere volume around the spine. Every overlap may therefore be expressed in two ways: as a list of overlapping capsule pairs (figure 3a), or the exact sections of the spine whose surrounding vol-

umes overlap (figure 3b). Both notations are useful and serve complementary purposes.

A list of overlapping capsule pairs is assembled by testing each newly constructed capsule against its predecessors. This directly accelerates optimization as only pairs on the list need to be checked for correct ordering and potential collisions. To obtain the second notation, overlaps between capsule pairs are expressed using their actual starting and ending points, then recursively merged whenever consecutive stretches of the spine are affected. It is important to note that not all overlaps are relevant. As seen in figure 3a, a capsule always overlaps its immediate predecessor because the two share a point in the spine. Such spurious local overlaps are a property of the capsule representation and should be ignored. We therefore discard any overlap involving two consecutive capsules overlapping each other.

While the first notation is used to accelerate optimization, the second is required to determine the desired ordering. If pressure was compared independently for each overlapping capsule pair, D in figure 3a may for example be placed above A but below B, leading to a break in the generalized cylinder. By comparing the average pressures of the two merged segments in figure 3b instead, capsules A, B and C are all placed on the same side of D and E.

A special case occurs if the generalized cylinder repeatedly loops over itself, such as when sketching the coils of a spring. In figure 4a, the red and blue sections of a generalized cylinder overlap. As the user continues to draw a spiral, the sections grow by merging to eventually reach figure 4b. At this point, there still is a single overlap between red and blue. However, any further extension of the stroke would result in corruption with both the red and blue sections overlapping their own starts and part of the spine belonging to both sections at once. To handle such cases, we detect a loop whenever the end of a section reaches its own start. The section is counted as a full winding and a new one begun, shown green in figure 4c. Should this section again reach its own start, another winding would be detected. The entire arrangement continues to be treated as a single large overlap. This prevents breaks in the generalized cylinder as

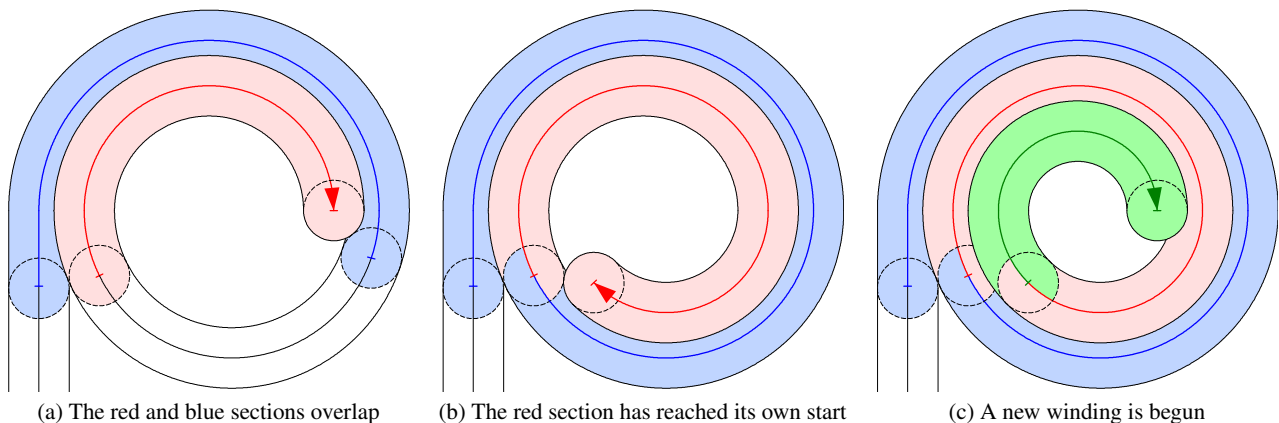


Fig. 4 Overlap handling for the special case of a generalized cylinder repeatedly looping over itself

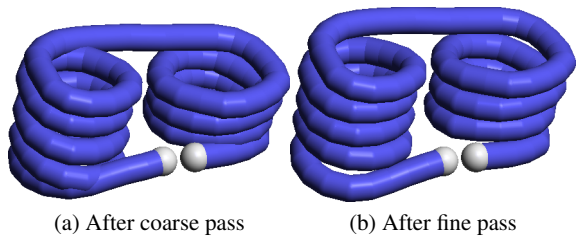


Fig. 5 Optimization results for a heavily looping spine

all windings are arranged in the same order, determined by comparing the average pressures of the first and last one.

5 Optimization

The need to determine z coordinates for all nodes in the spine gives rise to a constrained, nonlinear optimization problem. To produce a visually plausible shape, the coordinates should vary smoothly while arranging overlaps in the correct order, preventing self-intersections and respecting manually set z values. Any algorithm capable of performing this optimization could be used. We describe here our numerical technique. Additional details are given in the appendix. In each iteration, internal forces drive the spine toward a smoother shape while penalty forces correct constraint violations. Internal forces are attenuated as optimization progresses to prevent oscillation. Although the capsule is an efficient bounding volume, computational effort is dominated by inter-capsule collision detection and resolution. A multiresolution approach accelerates the optimization by approximating the spine's shape in a coarse pass and reducing the number of iterations in which collision detection is required.

5.1 Coarse pass

This initial pass operates on the level of entire overlapping sections (figure 3b), calculating a single z value for each, regardless of its length and the number of capsules spanned. For every section i passing over another one j , nodes are placed at their centers, initialized to $z_i = -r$ and $z_j = r$, with r the radius of the swept sphere. Instead of costly collision detection between the two sections, the simple constraint $z_i \geq z_j + 2r$ is used to ensure that one passes roughly over the other. For the special case of two sections separated by w windings, initial values $\pm wr$ and the constraint $z_i \geq z_j + 2wr$ are used instead. Additional immovable nodes are placed wherever the user has manually set a z coordinate by dragging the spine. This ensures that the internal forces strive to interpolate manually set positions.

In each iteration, the internal forces apply a Laplacian smoothing, directing every movable node toward the linear interpolation of its neighbors. Before displacing nodes, corrective forces are computed for each node pair violating a constraint. After 1000 iterations, the results of the coarse

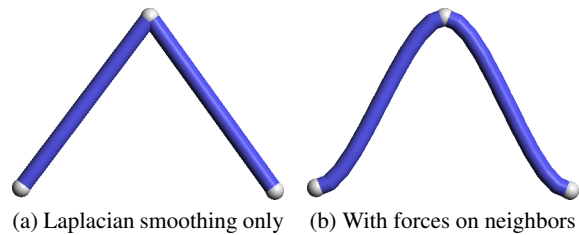


Fig. 6 Laplacian smoothing of a spine with three manually set z values

pass are transferred to the actual spine nodes by interpolating between them. We have found Catmull-Rom and simple linear interpolation to work equally well. Figure 5 illustrates the approximation of a final shape produced by the coarse pass for a heavily looping spine.

5.2 Fine pass

In the fine pass, z coordinates at the actual spine nodes are optimized. Internal forces again apply a Laplacian smoothing. Because nodes with manually set z coordinates are never moved, this alone would make the spine linearly interpolate between them (figure 6a). A smoother shape is obtained with each node exerting an additional force of opposite sign and half the magnitude on its neighbors. The total internal force is the sum of these influences, clamped to the strongest individual one (figure 6b).

Before advancing to the next iteration, each overlapping capsule pair is checked for constraint violations. If two capsules intersect or have incorrect ordering, the z displacement is computed that will arrange them in the right order without intersections again. This is preceded by a simple bounding box test to eliminate unnecessary calculations. Half the displacement is applied to either capsule in the form of penalty forces acting on the nodes at its ends. Penalty forces are accumulated for each node, clamped to the strongest individual influence to prevent excessive displacement from multiple forces acting in the same direction. This technique is guaranteed to resolve an isolated collision in a single iteration. Interactions between multiple capsules evolve over several iterations, a total 1000 of which are made.

6 Results

We introduced a small number of users to our implementation of the technique. All were immediately able to model spines. Pressure-sensitive ordering of overlapping sections was perceived as intuitive and prompted creative exploration. Users were often sketching looping shapes that are difficult or impossible to achieve using other approaches. Overdrawing support resolved the two most common problems. First, users would sometimes draw with too little or too much pressure, leading to an incorrect ordering of overlapping sections. This could quickly be fixed by retracing the relevant section with corrected pressure. Second, when attempting to

draw with very low pressure, the pen sometimes was accidentally lifted, prematurely ending the stroke. Overdrawing allowed the stroke to be continued from this point. Manual adjustment of z coordinates was only used when tasked with constructing a specific three-dimensional shape. We observed overall modeling times on the order of seconds, never exceeding a minute per spine. The spine shape was updated at interactive speeds by our C++ implementation running on a Pentium D 3.73 GHz under FreeBSD. As optimization time is dominated by collision detection and resolution, update times ranged from 2 ms for spines without any overlaps to 120 ms for the tightly packed windings of figure 5.

Figure 7 demonstrates some of the spine shapes that were easily constructed using our technique. Examples 7a to 7c illustrate the utility of pressure-based ordering. Spines looping over and under themselves were specified using a single stroke each. Shown in figure 7d is a shape similar to the wire trailing behind Pixar’s famous *Luxo, Jr.*, modeled after an example in [19]. In our system, the desired spine was obtained using an initial stroke and four interactively added z coordinates. To better illustrate its three-dimensional shape, the result has been raytraced in Blender. The next two examples demonstrate that our technique can simplify a modeling task even if the desired generalized cylinder does not fit the bounding volume. Instead of constructing it fully in Blender, the trumpet’s spine was quickly sketched using our system. While the system could in this case not automatically ensure sufficient space would remain around the spine, the specification of a looping space curve via its sketching interface remained much faster than traditional modeling tools. The octopus illustrates how our system can be used to model and adjust a character’s pose. Each arm was sketched as a separate stroke in an overhead view and then interactively transformed into the desired shape by dragging on the spines.

The final example in figure 8 shows how a field beyond generic three-dimensional modeling can benefit from our technique. When designing neon signs, an arrangement of the fluorescent tube is required that spells out a given writing without self-intersecting. Our system can quickly produce this from sketched input.

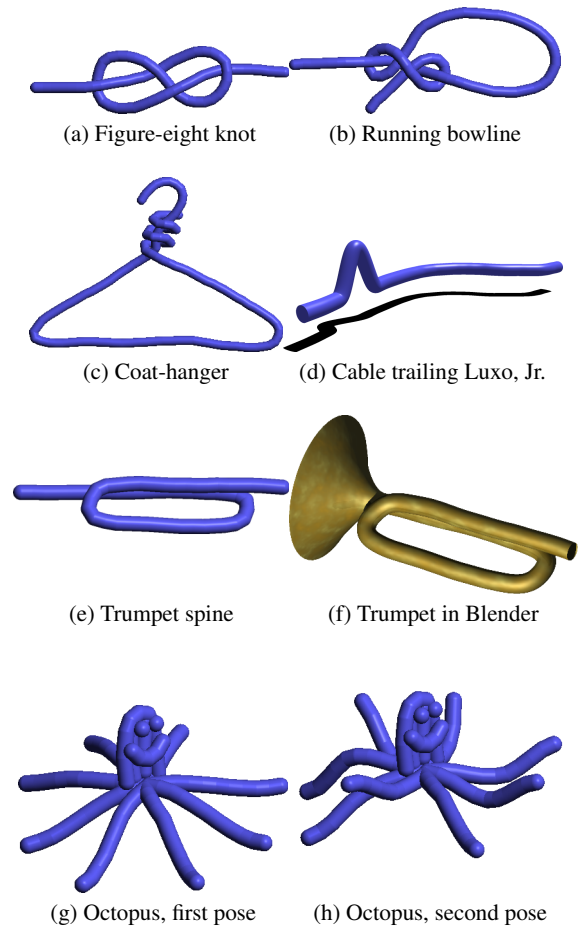


Fig. 7 Models sketched using our system

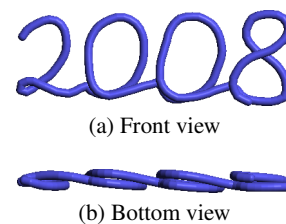


Fig. 8 Neon sign prototyped in our system

7 Conclusions and future work

We have introduced a novel method for sketching generalized cylinder spines. Arrangement of overlapping sections based on pen pressure allows complex, looping spines to intuitively be described using a single stroke. Computation of z coordinates by optimization produces smooth spines and prevents the surrounding generalized cylinder from self-intersecting. Adjustments to the proposed spine shape via oversketching or the dragging of nodes result in renewed optimization at interactive speeds.

Our technique provides several benefits for the construction of generalized cylinders. Existing methods for sketching space curves can be used to build a spine [6, 19]. However, with the spine treated on its own, no provision is made

to ensure sufficient space around it remains for the generalized cylinder. Also, curve complexity is limited in these approaches by the need to find a mapping between multiple strokes. An alternative is the construction of generalized cylinders using methods based on Teddy, which inflate them from sketched silhouettes. Self-occluding objects are possible by automatically inferring hidden silhouettes [7, 18] and using a paneling technique [35] to arrange overlapping sections. The process is not interactive, however, and no control is provided over z coordinates. Contours may also be connected wrongly, resulting in incorrect objects.

In our approach, the ordering of overlapping sections is controlled by the user via pen pressure. This can be prob-

lematic when multiple overlaps coincide and a very precise choice of pressure is required to stack them in the desired order. We would like to give the user additional graphical feedback about the current position in the stacking order, adapting the idea from [13]. It would also be desirable to verify whether the ordering constraints specified for all overlaps can simultaneously be fulfilled. The mathematical treatment required is complicated by the fact that the generalized cylinder's capsule structure must be taken into account. It is currently also possible to force self-intersections by dragging nodes so that insufficient space remains between them. While a formal treatment again would be complicated, we want to experiment with rejecting a dragging motion if it leads to the optimization being unable to find a valid shape.

The limitation we want to address first is the constant circular cross-section of our bounding volume representation. Direct use of custom cross-sections would lead to prohibitive computational costs for collision detection and resolution. However, we can improve the tightness of fit by allowing a wider range of bounding volumes to be used. Sweeping a sphere of varying radius is our first goal. Next, we want to investigate nonuniform scaling of the sphere, allowing eccentric cross-sections to better be approximated. The tightest bounding volumes can automatically be computed if we extend our system to let the user sketch cross-sections. This will also eliminate the need to export spines into Blender as all the final model can then be assembled by our system.

Abstracting from our particular implementation or even generalized cylinders, we believe that the implicitness with which our users accepted pressure as an additional degree of freedom shows its great potential for sketch-based modeling. The use of pressure is common throughout painting and drawing applications such as Corel Painter or Adobe Illustrator. Techniques aiming to adapt the sketching metaphor to three dimensions, however, generally neglect it. To closely emulate the feeling of pen and paper, a graphics tablet is required. And with a tablet, pressure information is inherently available. We are therefore very interested in investigating the potential uses of pressure throughout sketch-based modeling. The other degrees of freedom offered by some tablets, pen tilt and rotation, may extend this research in the future.

Acknowledgements We thank the anonymous reviewers for their helpful comments. This work was supported by the Irish Research Council for Science, Engineering and Technology under the Embark Initiative.

References

- de Araújo, B., Jorge, J.: BlobMaker: Free form modelling with variational implicit surfaces. In: Encontro Português de Computação Gráfica EPCG 2003, pp. 17–26 (2003)
- Baudel, T.: A mark-based interaction paradigm for free-hand drawing. In: ACM Symposium on User Interface Software and Technology UIST 1994, pp. 185–192 (1994)
- Bloomenthal, J.: Graphics Gems, chap. Calculation of Reference Frames Along a Space Curve. Academic Press, San Diego, CA, USA (1990)
- Brown, J., Latombe, J.C., Montgomery, K.: Real-time knot tying simulation. *The Visual Computer* **20**(2–3), 165–179 (2004)
- Cherlin, J., Samavati, F., Sousa, M., Jorge, J.: Sketch-based modeling with few strokes. In: Spring Conference on Computer Graphics SCCG 2005, pp. 137–145 (2005)
- Cohen, J., Markosian, L., Zeleznik, R., Hughes, J., Barzel, R.: An interface for sketching 3D curves. In: Symposium on Interactive 3D Graphics I3D 1999, pp. 17–21 (1999)
- Cordier, F., Seo, H.: Free-form sketching of self-occluding objects. *IEEE Computer Graphics and Applications* **27**(1), 50–59 (2007)
- Das, K., Diaz-Gutierrez, P., Gopi, M.: Sketching free-form surfaces using network of curves. In: Eurographics Workshop on Sketch-Based Interfaces and Modeling SBIM 2005, pp. 127–134 (2005)
- Eggli, L., Hsu, C.Y., Brüderlin, B., Elber, G.: Inferring 3D models from freehand sketches and constraints. *Computer-Aided Design* **29**(2), 101–112 (1997)
- Fleisch, T., Rechel, F., Santos, P., Stork, A.: Constraint stroke-based oversketching for 3D curves. In: Eurographics Workshop on Sketch-Based Interfaces and Modeling SBIM 2004, pp. 161–165 (2004)
- Gansca, I., Bronsvoort, W., Coman, G., Tambulea, L.: Self-intersection avoidance and integral properties of generalized cylinders. *Computer Aided Geometric Design* **19**(9), 695–707 (2002)
- Igarashi, T., Matsuoka, S., Tanaka, H.: Teddy: A sketching interface for 3D freeform design. In: ACM SIGGRAPH 1999, pp. 409–416 (1999)
- Ju, T., Zhou, Q.Y., Hu, S.M.: Editing the topology of 3D models by sketching. In: ACM SIGGRAPH 2007, pp. 42:1–42:9 (2007)
- Kallio, K.: 3D6B editor: Projective 3D sketching with line-based rendering. In: Eurographics Workshop on Sketch-Based Interfaces and Modeling SBIM 2005, pp. 73–79 (2005)
- Kanungo, T., Haralick, R., Dori, D.: Understanding engineering drawings: A survey. In: IAPR International Workshop on Graphics Recognition GREC 1995, pp. 119–130 (1995)
- Kaplan, M., Cohen, E.: Producing models from drawings of curved surfaces. In: Eurographics Workshop on Sketch-Based Interfaces and Modeling SBIM 2006, pp. 51–58 (2006)
- Kara, L., Shimada, K.: Construction and modification of 3D geometry using a sketch-based interface. In: Eurographics Workshop on Sketch-Based Interfaces and Modeling SBIM 2006, pp. 59–66 (2006)
- Karpenko, O., Hughes, J.: SmoothSketch: 3D free-form shapes from complex sketches. *ACM Transactions on Graphics* **25**(3), 589–598 (2006)
- Karpenko, O., Hughes, J., Raskar, R.: Epipolar methods for multi-view sketching. In: Eurographics Workshop on Sketch-Based Interfaces and Modeling SBIM 2004, pp. 167–173 (2004)
- Lam, D.: Tokamak physics engine. Website (2007). <http://www.tokamakphysics.com/>
- Lapides, P., Sharlin, E., Sousa, M., Streit, L.: The 3D tractus: A three-dimensional drawing board. In: IEEE International Workshop on Horizontal Interactive Human-Computer Systems Table-Top 2006, pp. 169–176 (2006)
- Larsen, E., Gottschalk, S., Lin, M., Manocha, D.: Fast proximity queries with swept sphere volumes. Tech. Rep. TR99-018, UNC Chapel Hill (1999)
- Liu, W., Kondo, K., Mitani, J.: An interactive sketch-based modeling system using a topology library and subdivision methods. In: Eurographics Workshop on Sketch-Based Interfaces and Modeling SBIM 2005, pp. 89–98 (2005)
- Masry, M., Lipson, H.: A sketch-based interface for iterative design and analysis of 3D objects. In: Eurographics Workshop on Sketch-Based Interfaces and Modeling SBIM 2005, pp. 109–118 (2005)
- Murakawa, J., Yoon, I., Hong, T., Lank, E.: Parts, image, and sketch based 3D modeling method. In: Eurographics Workshop on Sketch-Based Interfaces and Modeling SBIM 2006, pp. 67–74 (2006)

26. Naya, F., Conesa, J., Contero, M., Company, P., Jorge, J.: Smart sketch system for 3D reconstruction based modeling. In: International Symposium on Smart Graphics SG 2003, pp. 58–68 (2003)
27. Nocedal, J., Wright, S.: Numerical Optimization, second edn. Springer, New York, NY, USA (2006)
28. Okabe, M., Owada, S., Igarashi, T.: Interactive design of botanical trees using freehand sketches and example-based editing. Computer Graphics Forum **24**(3), 487–496 (2005)
29. Owada, S., Nielsen, F., Nakazawa, K., Igarashi, T.: A sketching interface for modeling the internal structures of 3D shapes. In: International Symposium on Smart Graphics SG 2003, pp. 49–57 (2003)
30. Pereira, J., Jorge, J., Branco, V., Ferreira, F.: Calligraphic interfaces: Mixed metaphors for design. In: International Workshop on Design, Specification and Verification of Interactive Systems DSV-IS 2003, pp. 154–170 (2003)
31. Sachs, E., Roberts, A., Stoops, D.: 3-Draw: A tool for designing 3D shapes. IEEE Computer Graphics and Applications **11**(6), 18–26 (1991)
32. Schmidt, R., Wyvill, B., Sousa, M., Jorge, J.: ShapeShop: Sketch-based solid modeling with BlobTrees. In: Eurographics Workshop on Sketch-Based Interfaces and Modeling SBIM 2005, pp. 53–62 (2005)
33. Smith, R.: Open dynamics engine. Website (2007). <http://www.ode.org/>
34. Teschner, M., Kimmerle, S., Heidelberger, B., Zachmann, G., Raghupathi, L., Fuhrmann, A., Cani, M.P., Faure, F., Magnetat-Thalman, N., Strasser, W., Volino, P.: Collision detection for deformable objects. Computer Graphics Forum **24**(1), 61–81 (2005)
35. Williams, L.: Perceptual completion of occluded surfaces. Ph.D. thesis, University of Massachusetts Amherst (1994)
36. Yang, C., Sharon, D., van de Panne, M.: Sketch-based modeling of parameterized objects. In: Eurographics Workshop on Sketch-Based Interfaces and Modeling SBIM 2005, pp. 63–72 (2005)
37. Zeleznik, R., Herndon, K., Hughes, J.: SKETCH: An interface for sketching 3D scenes. In: ACM SIGGRAPH 1996, pp. 163–170 (1996)

A Implementation details

In the coarse optimization pass, two nodes are constructed for each pair of overlapping sections. The nodes are ordered by their positions p_i along the sketched overhead view of the spine, so that $p_i < p_{i+1}$. An internal force then drives each node's z coordinate toward linear interpolation between its neighbors:

$$F_i = \frac{1}{a} \left(\frac{(p_{i+1} - p_i)z_{i-1} + (p_i - p_{i-1})z_{i+1}}{p_{i+1} - p_{i-1}} - z_i \right)$$

The nodes are displaced by adding these forces to their current z coordinates. To prevent oscillation, the attenuation factor a is exponentially increased over the course of the optimization. Whenever the new z coordinates would violate the constraint $(z_i + F_i) \geq (z_j + F_j) + 2r$ for a pair of nodes, corrective forces are applied. Node i experiences a force proportional to F_j and vice versa. This corresponds to two sections of the spine colliding and moving together in the direction of the stronger force. The resulting total forces acting on the nodes are:

$$c = 1 - \frac{z_i - z_j - 2r}{F_j - F_i}$$

$$\tilde{F}_i = F_i + cF_j$$

$$\tilde{F}_j = F_j + cF_i$$

During the fine pass, the z coordinates of the actual spine nodes are optimized. The internal force is calculated as above, but each node is additionally affected by the internal forces at its neighbors:

$$\tilde{F}_i = F_i - 0.5(F_{i-1} + F_{i+1})$$

Corrective forces are determined by collision detection. When two capsules collide or are incorrectly ordered, the minimal displacement d is computed that will arrange them in the correct order again. This can efficiently be done by expressing each capsule as two spheres and a truncated cylinder. The displacement required is computed separately for each pair of primitives in the two capsules and the largest value used. A corrective force of $F_c = 0.5d$ is applied to the nodes at the ends of one capsule, $F_c' = -0.5d$ at the other. Corrective forces from multiple collisions are added. To prevent excessive displacement, every sum of forces is clamped to the strongest individual influence acting in its direction. During both optimization passes, nodes whose z value has manually been set by the user are present, but immovable with the forces acting on them always zero.