Versatile Geometrical Sweeps

Monica Tang
Carlo H. Séquin

Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2024-13
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2024/EECS-2024-13.html

March 23, 2024

Versatile Geometrical Sweeps

Monica Tang and Carlo H. Séquin
University of California, Berkeley

Abstract

Our sweep generator is a generalization of the basic geometrical extrusion process, allowing an arbitrary 2D cross-section to move along a complex 3D space curve. During this sweep process, the cross-section can be rotated, scaled non-uniformly, and even be morphed into other profile shapes. Sweeps can serve as the back-bone of modeling many intricate geometrical shapes.

Introduction

Most CAD programs [1] [6] [7] have some extrusion operator. In a long-term evolutionary process [8] [2] [3] [5], we have enhanced and augmented this extrusion process into a powerful, versatile sweep operator that can follow an arbitrary sweep path through 3D Euclidean space and also change its cross-section while doing so. We discuss the main parameters of this sweep process and describe how the resulting sweep surface is constructed so that a "watertight" boundary representation can readily be used to make 3D-prints.

The Basic Sweep Operation

In the final implementation, the sweep-path is always a piecewise linear polyline embedded in 3D space. This piece-wise linear path can be spelled out explicitly as a sequence of *nodes*, or it can be generated as a finely sampled spline curve, defined by a coarser control polygon. The *profile* or *cross-section* swept along this sweep-path is a 2D piecewise-linear polyline defined in a separate x-y-coordinate system. Conceptually, this cross section is centered with the origin of its defining x-y-coordinate system placed on the sweep-path and oriented at a right angle to the sweep-path. Because of the discrete nature of the sweep-path, the resulting sweep surface will be a sequence of properly mitered prismatic segments. To construct these segments so that they all have the proper cross-sectional profile, a first instance of the cross-section is placed perpendicularly at the start of the sweep path. At all intermediate nodes of the sweep poly-line, that cross-section is placed into the angle-dividing plane of the incoming and outgoing line segments. In addition, the profile is non-uniformly stretched by $1/\cos(\beta/2)$, where β is the bending angle at that node in the sweep polyline. This keeps the profile of the prismatic segments between two subsequent nodes constant in the shape of the specified cross-section. The last cross-section is placed at the end of the sweep path, perpendicularly to the last polyline segment. Corresponding points on subsequent instances of the cross-sections placed at the sweep-path nodes are then connected to form the (possibly warped) quadrilateral facets of the mitered prism segments (Fig.1). These facets can also be split into two triangles, e.g. for the purpose of rendering. But there are two options to make this split, and different considerations may prefer one or the other option. The quad prism faces were kept as a good default case, because thay also provide a good starting point for CC-subdivision [10] .

The cross-section profile can be a closed polygon; this will result in a tubular surface topologically equivalent to a cylinder or an annulus. But it can also be an open-ended polyline or a simple line segment; the result is then an open surface. The "cross-section" can even be a single point – typically offset from the sweep-path – and the result will then be a single poly-line, which may possibly spiral around the sweep-path, and which then can be used as the sweep-path for yet another sweep.

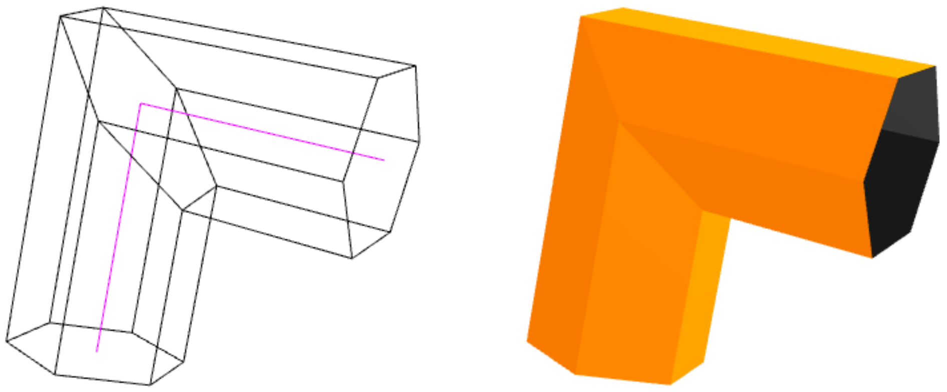

(a) (b)

Figure 1: *Basic sweep construct: (a) the purple sweep path, the perpendicular starting and ending cross-sections, and an intermediate stretched cross-section; (b) the resulting sweep surface.*

There is an additional degree of freedom that specifies how the cross-section may be oriented around the sweep path. This rotation angle is called *azimuth*. There are two principal ways in which this azimuth angle can be determined at every node of the sweep polyline.

In the *intrinsic* mode, the x-axis of the specified cross-sectional profile is aimed in the direction towards the center of local curvature of the sweep-path. For a piece-wise linear polyline, this coincides with the angle-bisecting vector between the incoming and outgoing segments at that node. If the cross-section has a crescent shape that is open in the direction of the negative x-axis, a saddle-shaped surface with negative Gaussian curvature is formed wherever the sweep-path makes a bend.

A second mode aims to *minimize torsion*. In this mode, all the vertices of the profile polygon are projected forward, parallel to the next polyline segment, until they intersect with the angle-dividing plane at the next node of the sweep-path, where they will form the next profile-polyline. To avoid any build-up of projection errors, the program does not do any actual projections. Instead, it will simply calculate how much the Frenet frame [4] of the sweep-path changes from one node to the next one, and it adds the negative amount of this rotation-angle into the calculation of the actual rotation-angle before stretching it and placing it at the next node. The *minimize torsion* mode (Fig.2a) avoids the hour-glass geometries (Fig.2b) that result where the sweep path has inflection points.

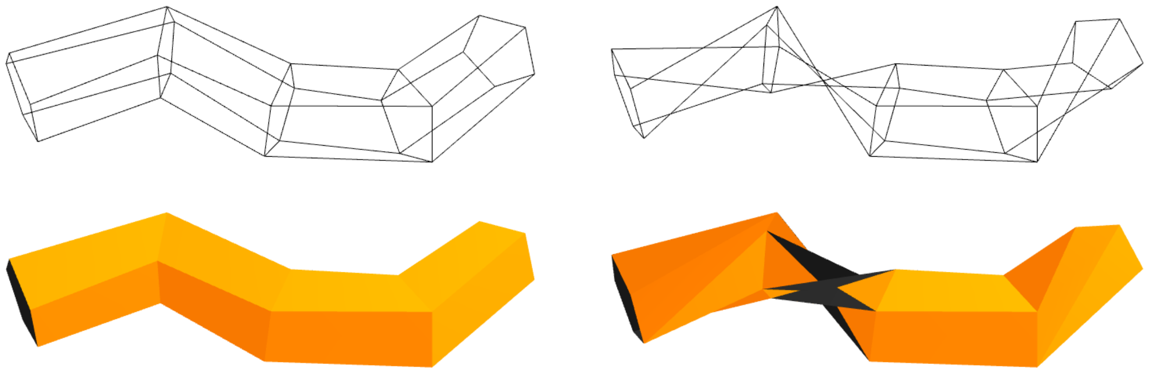
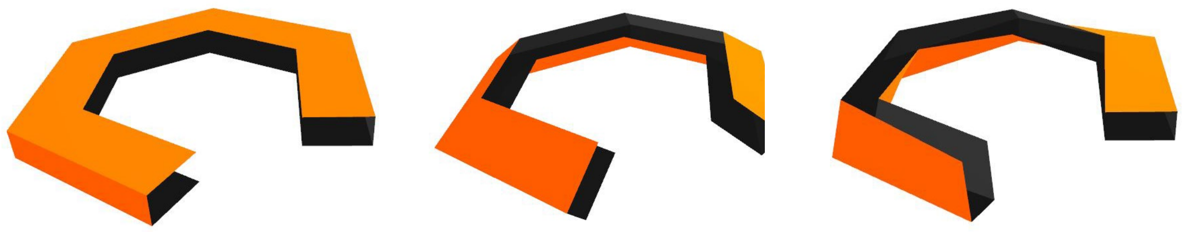

(a) (b)

Figure 2: *Cross-section orientation: (a) "minimize torsion" mode. (b) "intrinsic" mode,*

In either mode, the azimuth orientation at the beginning of the sweep curve is poorly specified. By default, the program turns the cross-section described in its defining x-y-plane in the simplest way to become orthogonal to the first sweep-path segment. Since this may not be the most desirable orientation, we give the designer two additional high-level parameters: *azimuth* and *twist*.

Azimuth is an additional, constant rotation angle through which the cross-sections at all sweep-path nodes are rotated. For a toroidal sweep, this would perform a motion as seen in a smoke ring that rotates through itself (Fig.3b). This parameter allows the user to set the proper orientation of the starting cross-section.

Twist provides an additional rotation to the many cross-sectional profiles placed at subsequent nodes. The twist value starts at zero at the beginning of the sweep and increases in small incremental steps from node to node, so that the cross-section at the end of the sweep undergoes an additional rotation as specified by the *twist* parameter (Fig.3c).



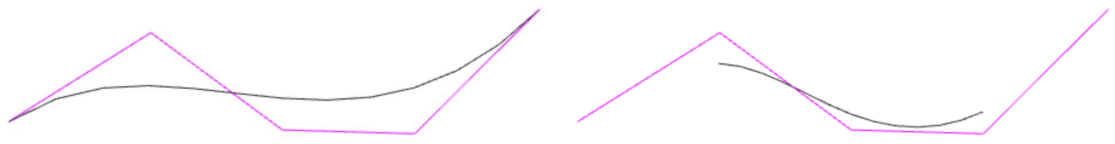
(a) (b) (c)

Figure 3: *(a) Basic sweep: azimuth =0, twist =0; (b) azimuth =40, twist =0; (c) azimuth = 0, twist =90.*

In the *intrinsic* mode, there is an additional problem at nodes where there is no distinct curvature because the incoming and outgoing segments of the sweep path are colinear. At all such nodes the *azimuth* value is taken from the previous node in the sweep-path. This still leaves one or more problem nodes at the beginning of the sweep that have zero curvature. To obtain a reasonable *azimuth* value for these nodes, the program looks for the first node in the sweep with a non-zero curvature, where it can set the *azimuth* value unambiguously. It then propagates this *azimuth* value to all the previous nodes back to the start of the sweep.

Smooth Sweep Curves

Often the designer wants to have a smooth-looking sweep-path. A natural way to define such a path is to use some spline, like a Bezier curve or a cubic B-spline. These types of curves are defined by a piecewise linear control polygon and by a parameter *segs* that specifies by how many individual segments the desired spline curve should be approximated. The ideal spline curve is then sampled by a corresponding number of points and turned into a piecewise linear polyline. The *sweep* program then does its work on this discrete polyline.



(a) (b)

Figure 4: *Control polygons (in purple): (a) for a Bezier curve. (b) for a cubic B-spline.*

Begincap and Endcap

If the cross-section is a closed polygon, the sweep operation described so far generates an open-ended tube. To close off this tube at either end, for instance, to generate a closed, “water-tight” boundary representation of some curvy rod that can be 3D printed, the generation of the two polygons at the ends of the sweep is initiated with the flags *begincap* and *endcap*. If the cross-section is not a closed polygon, perhaps just a single line-segment, or even just a single point, no endcaps will be constructed.

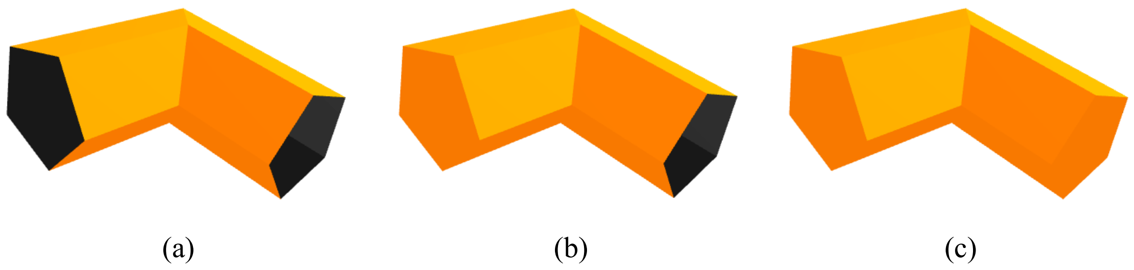


Figure 5: *(a) open-ended tube. (b) with begincap face. (c) with begincap and endcap faces.*

Sweep Ends and Closed Loops

Often a designer may want to create a sweep that closes onto itself to form a toroidal loop or a model of a mathematical knot. This requires that the sweep-path is a closed curve. But it is not good enough to just let the first and last point of the sweep-polyline coincide; the shared cross-sectional polygon at that location must be tilted properly to fall into the angle-dividing plane between the first and last sweep-path segment. To make this process easier for the designers, we provide the parameter *closed* in the sweep-path specification; the necessary construction will then be carried out automatically. The redundant, shared closing node need not be be listed in the sweep polyline; the closing procedure will automatically insert a closing segment. If the closing node is already repeated, it will be merged with the starting node. Moreover, depending on the shape of the sweep path, *twist* will have to be adjusted to get proper closure of the tubular surface. This twist-adjustment is <u>not</u> yet provided automatically (Figs.6a,b).

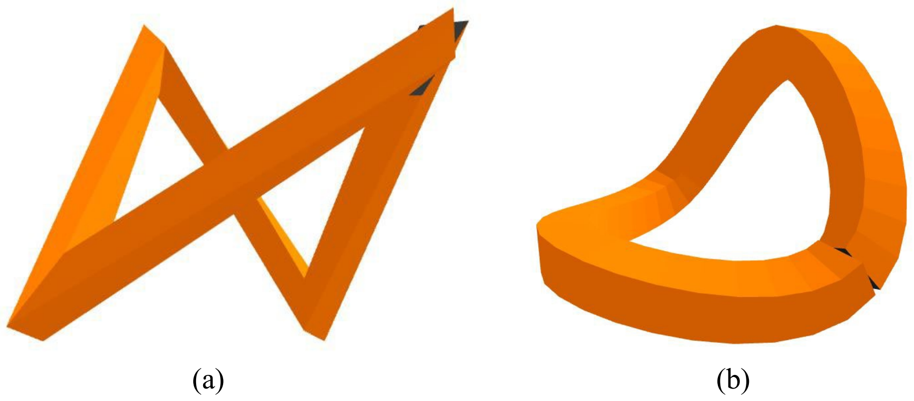


Figure 6: *(a) Closing a piecewise linear sweep path. (b) Closing a cubic B-spline curve.*

If the sweep path is defined by the control polygon of a cubic B-spline, generating a smooth closure of the sweep surface takes some additional work. To form a smoothly closing cubic B-spline curve, the first three nodes and the last three nodes at the end of the control polygon must be the same. By sampling the closed smooth spline-curve, we obtain a closed, piece-wise linear polyline (Fig.6b), which can then produce a properly closed sweep surface, once the *twist* value is properly adjusted. The three redundant, overlapping control nodes could be generated automatically if the cubic B-spline path carries the *closed* flag. At this time (March 2024), this option has not been implemented in the code, because there are several different options of what must be done when the designer has already included one or more overlapping or shared vertices at the ends of the the B-spline sweep path. For Bezier curves, a different set of point constructions will have to be worked out.

Mitred Sweep Terminations

Sometimes, a lengthy sweep tube may have to be 3D-printed in several partial stretches. These individual pieces need to have properly mitered ends, so that they readily fit together and form the desired composite result. To make it easy to model such a partial sweep surface with properly mitered ends, we have introduced the parameters *cutbegin* and *cutend*. They will suppress the output of the first and/or last prismatic segments of the sweep surface, but will maintain the properly mitred end-profiles at these cutoff locations (Fig.7a,b). Thus each partial sweep path can simply include the second node in the adjacent continuing sweep piece to generate the proper cross section geometry at the junction node. Any *begincap* or *endcap* statements can also be used at the cutoff nodes to construct proper, "water-tight" boundary representations needed to make 3D-prints.

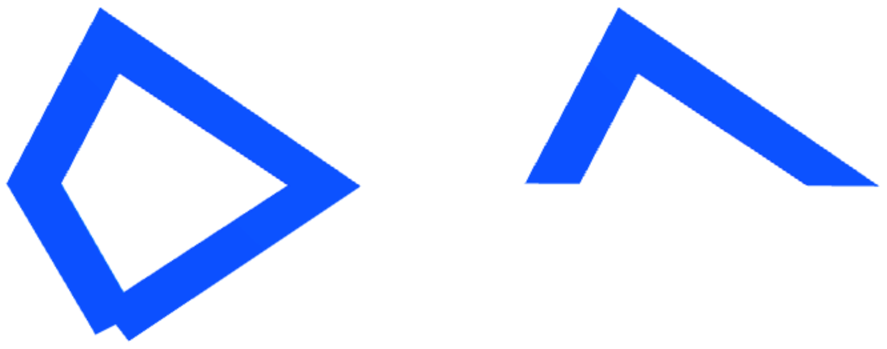


Figure 7*: (a) Sweep starting/ending at bottom mode. (b) Mitered sweep using cutbegin and cutend flags.*

The Complete Basic Sweep Command

The JIPCAD commands for a polyline, for a cubic B-spline, and for a complete sweep are shown below:

polyline id (point_idlist) [closed] endpolyline

bspline id order 4 (point_idlist) segs {*segs*} endbspline

sweep id
  crosssection crosssection_id [begincap] [endcap] [reverse] endcrosssection
  path path_id [mintorsion] [azimuth {*a_angle*}] [twist {*t_angle*}] [cutbegin] [cutend] endpath
endsweep

Properly Mitred Sweep Terminations in Splines

Things are more complicated when we want to generate properly mitred endings on sweeps along smooth spline curves. The geometry for the end-profiles on a sweep also depends on the sampling density along the spline curve. The orientation of these endfaces should be based on the geometry of the ideal spline curve; they should be perpendicular to the tangent at the end of the spline, and they should be non-uniformly stretched based on the local curvature. This information must be extracted from the information contained in the control polygon. The proper plane for the endface is the angle-dividing plane between the last two segments of the finely sampled smooth B-spline. The local stretching of that face in the osculating plane is determined by the angle between the last sampled line segment and the tangent direction of the spline.

With this adjustment, two separate sweeps along smooth spline curves that join with curvature continuity (guaranteed with a 3-node overlap between the two splines) will have properly matching mitred end-geometries if they both also have the same number of sample spots per segment of the control polygon.

Figure 8 shows the proper way of joining sweeps based on polylines (Fig.8a,b) and on cubic B-splines (Fig.8b,c). The sweep path for the blue polyline sweep adds at either end the next node in the green segment, but then removes the redundant segments with *cutbegin* and *cutend*, and vice versa. The sweep path for the cyan B-spline sweep adds at either end the next nodes in the control polygon of the B-spline sweep shown in yellow, and vice versa (Fig.8c).

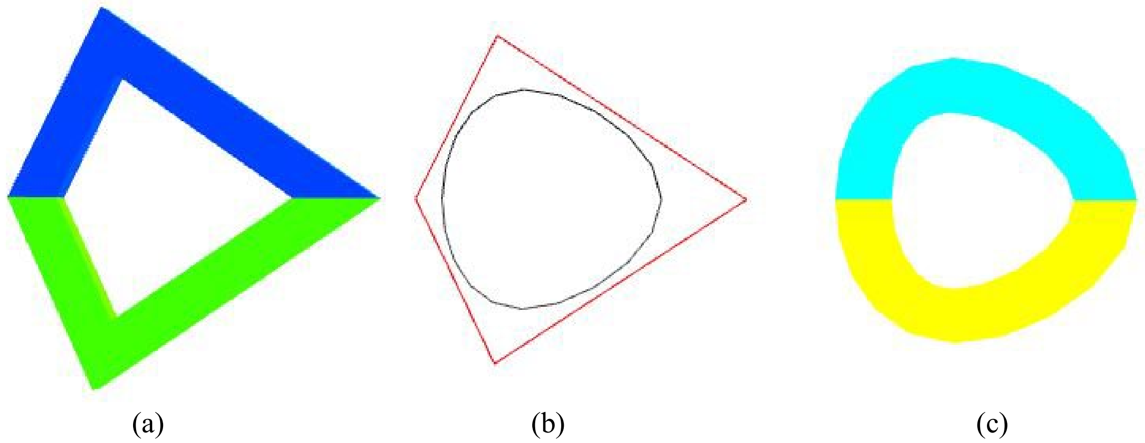


Figure 8: *Properly mitered and joined components: (a) two polylines, (b) underlying sweep paths, (c) two cubic B-splines.*

Sweeps with Local Control

There are additional handles to fine-tune the shape of the resulting sweep surface. By introducing *controlpoints*, the designer has the ability to locally fine-tune the twisting behavior of the sweep and to scale the cross-section in a non-uniform manner as it travels along the sweep-path.

Control-Points in Polylines

Some of the nodes of the poly-line sweep path can be designated as *controlpoints*. At these control-points, two scale-factors can be introduced to scale the basic cross-section in the x- and y-directions. Furthermore, the cross section at a control point can be given an additional amount of azimuth rotation (around the sweep path, i.e., the "z-axis"). This is the complete command:

controlpoint id **point** p_id [scale ({*sx*} {*sy*} 1)] [rotate (0 0 {*rz*})] [reverse] endcontrolpoint

For its location in the sweep path, a controlpoint {id} references a previously defined "ordinary" point {p_id}. The *scale* variable has three components. The first two define the nonuniform scaling of the basic cross-section (Fig.9). The third component (*sz*) should just be set to "1". This field is provided here in case we want to implement a future version of *sweep* with non-planar cross-sections. Correspondingly, in the *rotate* variable, the first two components should be set to "0" to keep the profile at that node in the angle-dividing plane; only *rz* is relevant to adjust the azimuth angle (Fig.10).

All the above adjustments affect only the cross-section located at that controlpoint; all other nodes will use the basic, unaltered cross-section. When differently scaled cross-sections are connected, the resulting prismatic elements in between will take on the shape of a conic frustum.

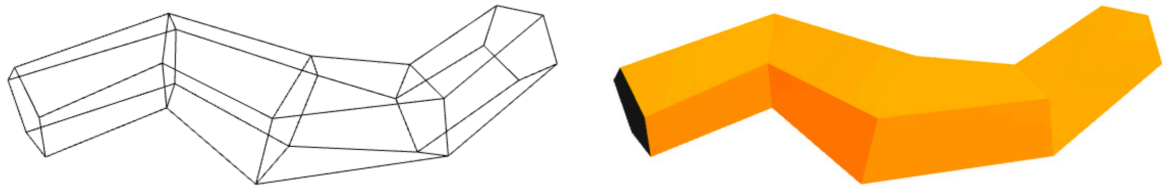


Figure 9: *Sweep along a 4-segment piecewise-linear path, with an enlarged cross-section in the middle.*

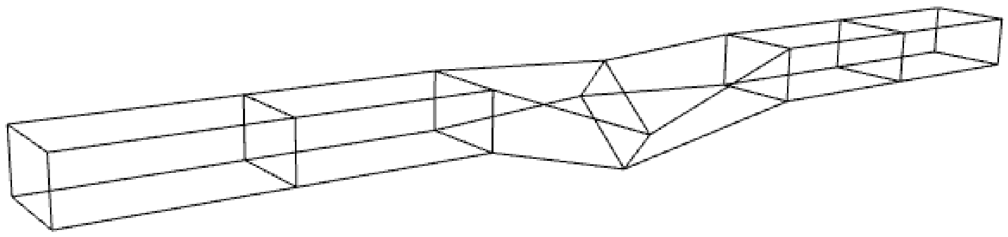


Figure 10: *Sweep along a 6-segment piecewise-linear path, with azimuth warp in the middle.*

Control Points in Cubic B-splines and Bezier Curves

Control points that affect the cross-section locally can also be introduced into the control polygons of spline curves. However, since each segment in the control polygon can be sampled into many segments to form a smooth curve, we want to have the added fine-tuning also affect the neighbourhood of the curve in a smooth manner. Thus we use the same polynomial that is used to interpolate the x-y-z-coordinates of the spline curve to also blend the adjustment terms introduced through controlpoints. As in the basic piecewise linear sweep, points that are not specifically designated as *controlpoints* will use the basic cross-section in its original form in this interpolation process. With all the profiles defined at the nodes of the control polygon, the scaling and rotation of the profiles at all the spline sample-points will be carried out with the same basic cubic polynomial that defines the spline sweep path.

Here is the key part of the JIPCAD code that creates Figure 11:

```
polyline  xsect   { define a closed regular heptagon }   endpolyline
bspline  bspath   order 4   ( p00  p0 p1  CP2  p3 p4 p44 )  segs 12   endbspline
sweep Fig11
    crosssection  xsect  endcrosssection
    path  bspath  mintorsion  endpath
endsweep
```

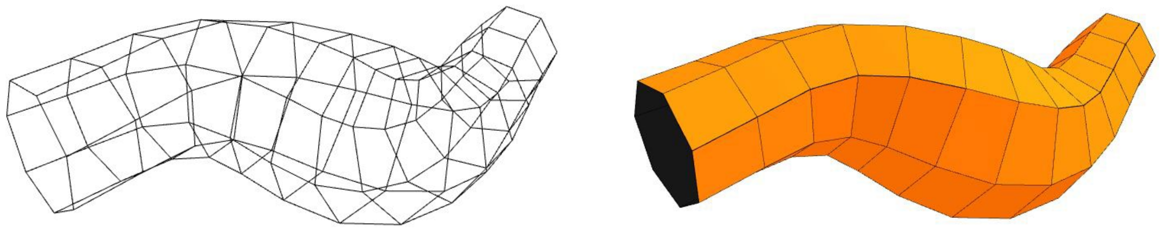


Figure 11: *Cubic B-spline sweep with a 6-segment control polygon, uniformly fattened in the middle.*

And here is part of the JIPCAD code that creates Figure 12:

polyline xsect { define a closed regular hexagon } **endpolyline**

beziercurve bezpath (p0 p1 CP2 p3 p4) **segs** 12 **endbeziercurve**

sweep Fig12

  crosssection xsect **endcrosssection**

  path bezpath mintorsion **endpath**

endsweep

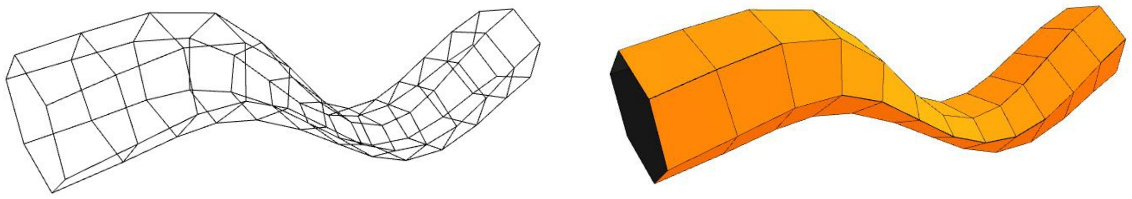


Figure 12: *Bezier sweep with a 4-segment control polygon, non-uniformly narrowed in the middle.*

Sweep-Path Reversals

Reverse is an additional flag that can be associated with any *controlpoint*. It should be associated only with nodes where the sweep-path turns back on itself. At every cross-section that has the *reverse* flag (and only there!), the sweep should look like a "sock turned inside-out" or like the "mouth" in the classical rendering of a Klein bottle. This means that a "reverse" cross-section joins together a normal prism shell with one that is turned inside-out.

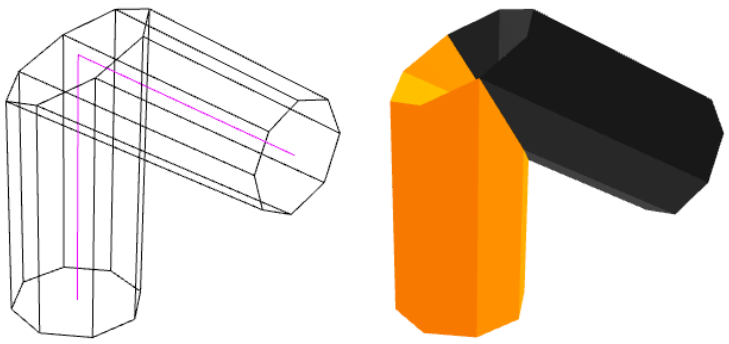
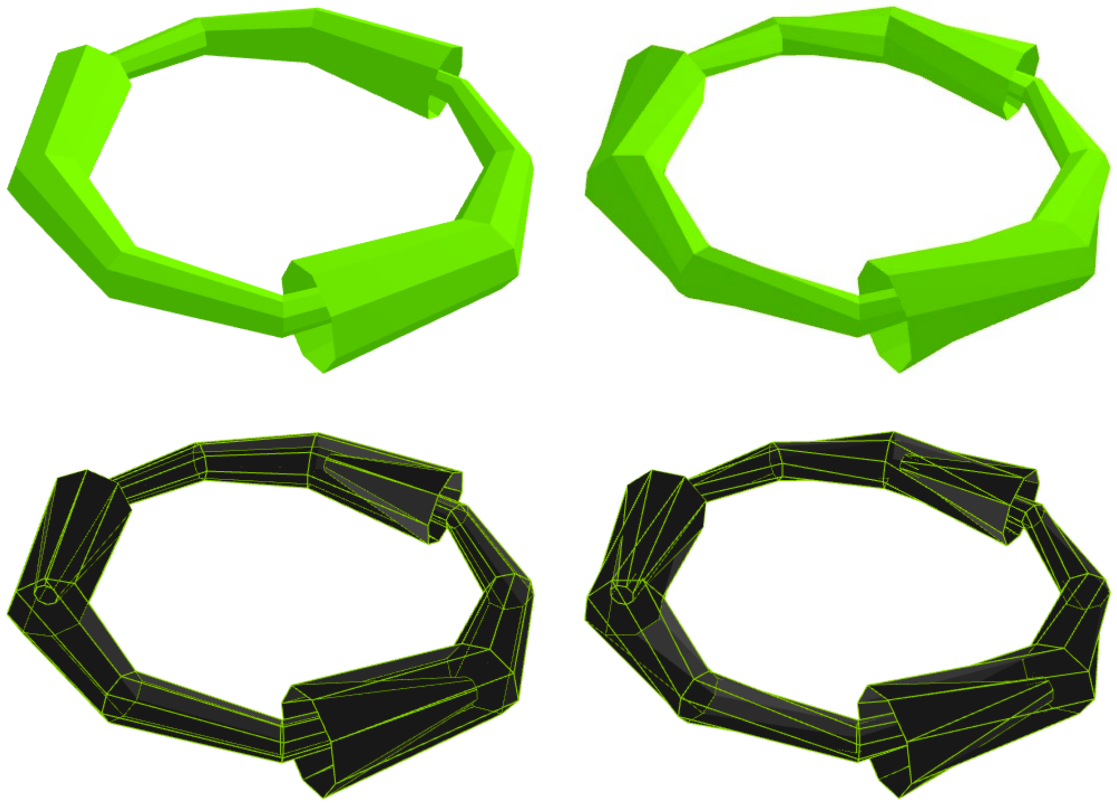


Figure 13: *Sweep with a reversal at the middle node.*

If the incoming and outgoing sweep-curve segments at a reversal-node are not perfectly aligned, then the generated reversal cross-section will lie in a plane that is perpendicular to the angle-divider vector between the two segments (Fig.13). The non-uniform stretching factor for this cross section, 1/cos(β/2), is based on the angle between the two sweep-curve segments. Moreover, on the reversed segments of the sweep tube, the rotations associated with *azimuth* and with *twist* must turn in the reversed directions. (There should be no sudden self-intersections emerging when azimuth or twist are non-zero).

We can make "zig-zag inside-out" shapes by having more than one *reverse* flag. If we place such flags at nodes B and C in a sweep that has (among others) nodes: A … B … C … D, an "inside-out" tube segment is generated that runs from nodes B to C. Figure 14 shows three such double-reversals in a loop.



(a) (b)

Figure 14: *Circular sweeps with six reversals: (a) without any twist, (b) with 720° of twist.*

In an ordinary sweep, this reverse flag could also be used at the cross-section definition within the sweep command to turn the whole sweep tube inside-out (Fig.15).

Any *begincap* or *endcap* will be oriented to be compatible with the tube segment to which they are attached so that their face normals are pointing in the same topological direction as the prismatic facets in the end-segments of the sweep (Fig.15).

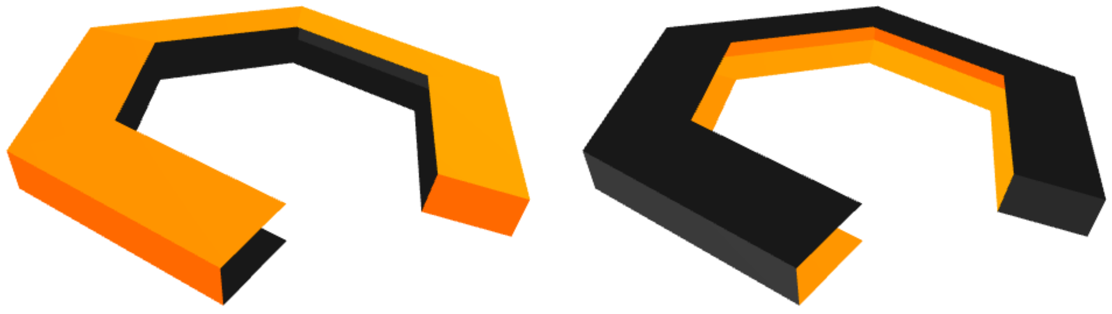

(a) (b)

Figure 15*: (a) Sweep, (b) reversed sweep, -- shown with endcaps at the right end.*

Sweep Morphs

Sweepmorphs are a further enhancement of the basic *sweep* generator. They allow the cross-sectional profiles to be modified in a more substantial way. There is one additional (optional) parameter string in the enhanced *controlpoint* of a sweep morph; it refers to a predefined cross-section that should be applied at this node location. Different cross-sections can then transform into one another, as long as they all are defined with the same number of vertices. These shape transformations occur gradually from one specified cross-sectional polygon at a certain *control-node* to the next one.

Except for additional blending operations between different profile geometries, *sweepmorphs* use all the same algorithms and computations as *sweeps*. Nevertheless, we decided to make *sweepmorphs* available to the users as a separately named generator. Since the additional blending operations involve a lot of extra computations, the users should be fully aware that they are invoking these additional computations. “Under the hood” both generators refer to the same program.

Enhanced Control-Points

The complete syntax for the enhanced *controlpoint* to be used in *sweepmorphs* has an additonal parameter string referring to a particular predefined cross-section:

controlpoint id **point** p_id [scale (sx sy 1)] [rotate (0 0 rz)] [reverse] [**crosssection** x_id] **endcontrolpoint**

When constructing a *sweepmorph,* all cross-sections used must have the same number of vertices. Then, <u>two</u> interpolating blending processes are carried out consecutively. If two different cross-sections are specified *n* nodes apart in the control polygon, a first blend calculates linearly interpolated cross-sections for all intermediate nodes. This is done by linearly moving each vertex in the cross-sectional profile from their position in the first cross-section to their position in the next one. If the starting node and the termination node of the *sweepmorph* are not explicitly defined by an enhanced *controlpoint*, it is assumed that their cross-sections are the same as the closest explicitly defined cross-sections, respectively. This defines the shape-morphed cross-sections for every node of the control polygon.

If the sweep path is a cubic B-spline curve, the regular <u>cubic</u> blending, used for creating the “smooth” sweep path from its control polygon, should be used to calculate the profiles for all the sample nodes introduced along the spline curve, but this has not yet been implemented; currently *sweepmorph* uses linear blending regardless of the type of the sweep curve.

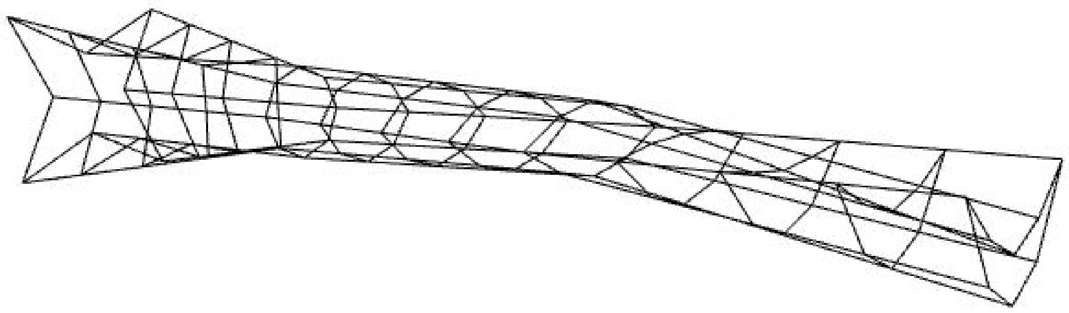


Figure 16: *Sweepmorph with changing cross-sections: Star – Octagon – Pointed Oval – Crescent.*

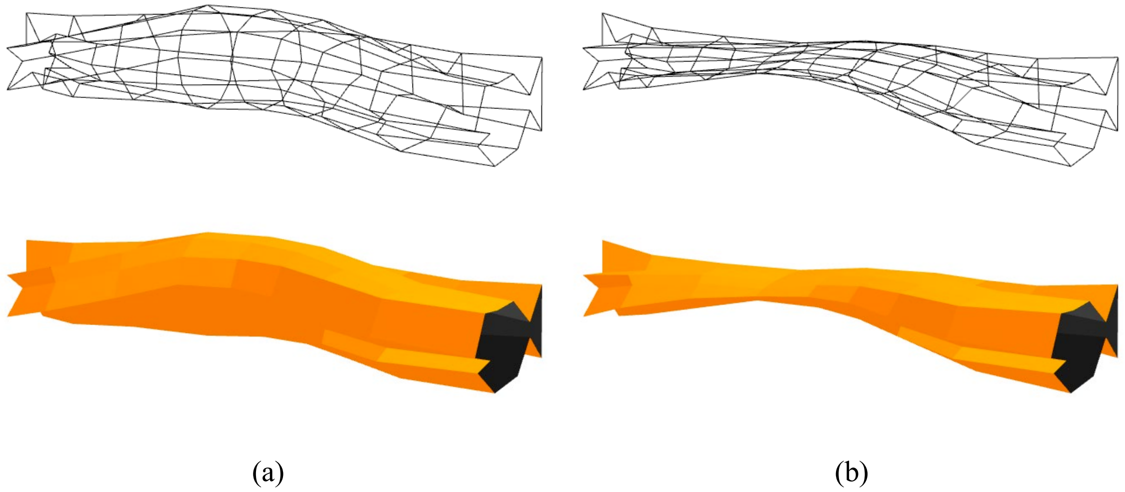
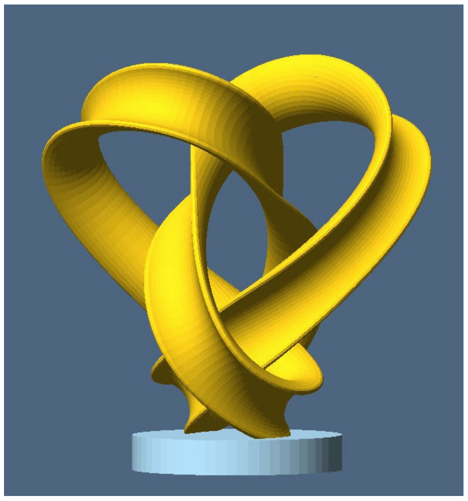


Figure 17*:* *(a) Sweepmorph with changing cross-sections: Star – Circle – Fish;*
(b) non-uniform scaling applied to the Circle cross-section in the middle.

Artistic Examples

Figure 18 shows some artistic geometrical models created with the versatile *sweep* construct described above. Figure 18a has been generated by sweeping a crescent-shaped cross-section along a knotted path forming a mathematical, 4-crossing Figure-8 knot. The sweep was done in *intrinsic* mode, so that the concave part of the crescent is pointing outwards in all the bends of the sweep path, – thereby forming saddle shapes with negative Gaussian curvature.

Figure 18b is based on a different version of the Figure-8 knot. It uses a small, square cross-section that is offset from the origin of the defining coordinate system. It then sweeps this cross-section four times along the complete Figure-8 sweep path, with a total amount of 360 degrees of twist. Thus, in each pass, the little square is rotated 90 degrees around the sweep path. The four passes have been colored differently to make the construction easier to understand. The result is equivalent of sweeping a "4-strand cable" along the basic Figure-8 knot and connecting it to itself after a 90-degree twist. Overall, this forms a much more complicated knot with 67 crossings.


(a)

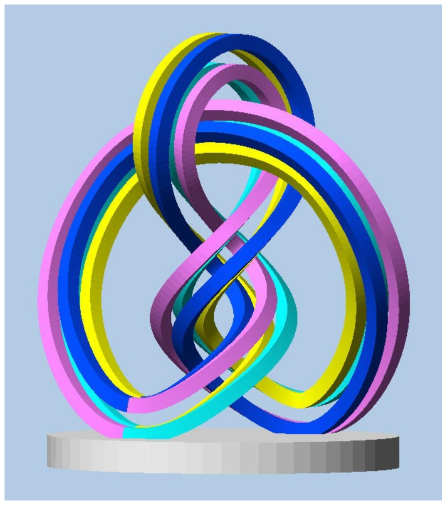

(b)

Figure 18: *Artistic Examples: (a) Figure8-knot swept with a crescent cross-section. (b) Figure8-knot swept with a 4-strand cable with 90° twist.*

Summary and Conclusions

Many of the capabilities described in this report had already been realized in some programs in the Berkeley SLIDE system [8] and in earlier versions of NOME [2] and JIPCAD [3]. Now they have been combined into a single powerful and versatile sweep program. We have used sweeps primarily to make fancy models of mathematical knots and links. But sweeps are also a convenient modeling tool for mechanical systems, for air-conditioning ducts in building designs, for decorative legs and arm-rests in furniture, or for limbs in robotic creatures.

There are some additional enhancements that could be made to the sweep construct. For instance, an option to have multiple sweeps with different cross-sections following along the same sweep path, could be implemented by allowing the "cross-section" referenced in the sweep statement to be a compound *group* that combines instances of all listed cross-section geometries, possibly rendered in different colors.

We have not yet made this an integrated feature in the current sweep program. Once a sweep path has been defined, possibly with many control points and localized adjustments to the scaling and rotation of the cross section, it is easy to repeat the few statements that define a sweep surface with a modified call to another cross-section. As another example, if the goal is to make a 3-sided prismatic sweep where all three sides are of different colors, we can define a cross-section consisting of a single side of an equilateral triangle and then define three separate sweeps rendered in different colors, where all sweeps use that same cross-section but have azimuth angles that differ by 120 degrees.

Acknowledgements

We acknowledge the contributions made by many URAP (Undergraduate Research Apprentice Program) students during the last several years to figure out the parameters and general behavior of a useful sweep program and in implementing and integrating corresponding code into the JIPCAD environment.

References

[1] Autodesk. "2D and 3D Computer-Aided Design." – https://www.autodesk.com/solutions/cad-software

[2] G. Dieppedalle. "Interactive CAD Software for the Design of 2-manifold Free-form Surfaces." 2018. – https://www2.eecs.berkeley.edu/Pubs/TechRpts/2018/EECS-2018-48.pdf

[3] R. Fan, C. H. Séquin, R. Ng. "Joining Interactive Graphics and Procedural Modeling for Precise Free-Form Designs." Tech-Report (EECS-2021-125) May 14, 2021. – https://www2.eecs.berkeley.edu/Pubs/TechRpts/2021/EECS-2021-125.html

[4] Frenet Serret Formulas. – https://en.wikipedia.org/wiki/Frenet%E2%80%93Serret_formulas

[5] JIPCAD Language Reference. – https://tinyurl.com/9ds62ee2

[6] OpenSCAD Documentation. – https://openscad.org/documentation.html

[7] Python Blender Documentation. – https://docs.blender.org/api/current/

[8] J. Smith. "SLIDE design environment." (2003). – https://people.eecs.berkeley.edu/~ug/slide/

[9] Ultimaker. "Reliable 3D printers." – https://ultimaker.com/3d-printers

[10] Wikipedia. "Catmull–Clark subdivision surface." – https://en.wikipedia.org/wiki/Catmull%E2%80%93Clark_subdivision_surface

+++