Three-dimensional shape modeling, or simply modeling, creates a computer representation of a 3D shape. The 3D shape could be constructed from scratch, from existing 2D shapes, or by deforming preexisting 3D shapes. Modeling has applications in CAD, creative design, entertainment (video games and movies), ideation (concept shapes), and communication of shapes in scientific disciplines.

Unfortunately, computer interfaces for modeling are difficult to learn and take long to master. Even highly trained modelers might spend hours designing a shape. They often prefer building the initial concept shapes using physical (non-digital) stand-ins such as clay maquettes. Owing to this high learning curve, modeling has stayed out of reach for nonexperts. Much academic research has gone into simplifying the modeling interface to let novices create 3D shapes quickly and easily.

One category of research in simplifying modeling from scratch is curve-based modeling, which creates shapes through pen-based strokes. A curve-based-modeling interface is easier than the traditional modeling interface because it relies on skills most people learn when very young: drawing a curve on a flat surface with a pen-like tool. Hardware for pen-based computing has improved to the point at which most professional designers and artists heavily use a tablet-and-stylus interface (for example, Wacom’s Intuos pen tablets and Cintiq pen displays). Even many popular consumer devices such as smartphones and handheld computers can employ a stylus-based interface for creative design. All these factors have generated much excitement and research in curve-based modeling, which could make 3D modeling accessible to novices.

Here, I assume that you want to create a 3D shape from scratch. Toward that end, I discuss three rough categories of curve-based modeling (in increasing order of difficulty):

- extruding 2D shapes,
- inflating 2D shapes, and
- drawing 3D curves.

This tutorial isn’t a comprehensive survey or a state-of-the-art report; rather, it aims to provide a starting point for someone new to this area. The methods I describe aren’t necessarily the best or ideal, but they’re representative examples that yield positive results while also exposing some issues related to curve-based modeling. I usually cite the more recent academic publications that best illustrate the ideas and provide good starting points, not necessarily the papers that first introduced the idea or are the most frequently cited.

**Extruding 2D Shapes**

A common, well-established approach for modeling from curves is to start from 2D curves and extrude them to produce 3D shapes. Several off-the-shelf programs (for example, Adobe Photoshop, Google SketchUp, Autodesk Maya and 3ds Max, Maxon Cinema 4D, and AutoDesSys bonzaI3d) provide extensive capability for extruding shapes.

Although this approach is relatively simple, it’s worth considering here because it lets you model a range of real-world objects (especially human-made objects) as sweeps of complex 2D curves.
Most user interaction time is spent on drawing the 2D curves. After that, the area bounded by them can be immediately extruded along the sweep paths, using only a handful of parameters.

Drawing 2D Curves

The field of 2D-curve drawing is mature, and off-the-shelf programs such as Adobe Illustrator, Autodesk Sketchbook Pro, Corel Draw, or Inkscape can produce 2D curves. Traditional methods for 2D-curve drawing offer high-fidelity control over the curve positions and tangents (usually by representing the curves as 2D cubic Bezier curves). Drawing curves using these methods usually takes time and precision, but such precise curves are necessary to produce precise 3D content.

Recent methods for 2D-curve drawing try to enable a more casual, spontaneous sketching interface, in which the user experience is similar to that of drawing on paper. Sketch-based interfaces trade precise output for a spontaneous, seamless user experience—one more conducive to creativity and ideation. Many of the off-the-shelf curve-drawing programs I mention offer a sketching interface (usually called a pencil tool). These sketching programs come close to mimicking drawing on paper with a pencil. However, using sketch-based interfaces to edit curves needs further development and is an area of active research.

For example, a sketch-based curve-editing interface might aim to implement three features: merging, overstroking, and blending (see Figure 1). To implement them, it must distinguish a stroke (a single, continuous mark by the user) from a curve (the geometric representation of a component of the intended artwork). A significant research question is how the system can differentiate between drawing strokes intended to create marks on the virtual canvas and the gestural strokes intended to issue commands to the editing interface. For discussions on such curve-based editing interfaces, see “ILoveSketch: As-Natural-as-Possible Sketching System for Creating 3D Curve Models”¹ and “Improving the Sketch-Based Interface: Forming Curves from Many Small Strokes.”²

Figure 1. Sketch-based-interface tasks. (a) Merging melds multiple overlapping or disjoint curves into a single curve. (b) In overstroking, a stroke near an existing curve deforms part of the curve to more closely follow the stroke. (c) Blending combines small user strokes to form a single, smooth curve. (Source: Cindy Grimm; used with permission.)

Sweeping 2D Curves

A sweep moves a 2D cross section along an arbitrary 3D space curve or sweep path. The cross section’s boundary traces the final surface as it moves along the path.

The simplest form of a sweep is when the sweep path is a straight line; the cross section is translated orthogonally to this line without any other transformations. Such sweeps are typically simply called extrusions. They’re most commonly used for text effects and composing models from relatively simple human-made objects, as in Google SketchUp and Sketch³ (see Figure 2).

You can edit the sweep path or transform the cross section as it moves along the sweep path (see Figure 3). The parameters for editing the path and transformations are usually few (significantly fewer than the number of points used to draw the 2D curves). So, editing the path and transformation usually is an interactive process with rapid feedback. That is, once you’ve created the 2D curves, sweeping them to produce
3D shapes is relatively easy. As Youngihn Kho and Michael Garland show, similarly to having a sketch-based interface for editing 2D curves, you can have a sketch-based interface for editing the sweep path.4 Also, Joseph Cherlin and his colleagues showed how to construct general-purpose shapes (even smooth, organic shapes) using sweeps in a sophisticated user interface.5

A commonly used nonlinear sweep is lathing or a surface of revolution (see Figure 3c), in which the sweep path is a circle and the cross section is swept perpendicularly to that circle. More generally, you can edit the straight sweep path into a nonplanar curve to produce nonlinear sweeps, surfaces of revolution, and twisted sweeps. In most of these cases, and especially for surfaces of revolution, the key challenge is to parameterize the sweep path’s shape using extremely few (usually fewer than five) parameters while maintaining a large range of expressibility.

You can make many surfaces by sweeping a single, closed contour. However, more complex surfaces, such as in Figure 4, require splitting the 2D drawing into disjoint regions each constituting a well-defined sweep cross section. Such a cross section is necessary to provide “end caps” for the sweep.

To construct a well-defined cross section, you must analyze the intersections and hierarchy in the input curves to extract disjoint areas that might intersect only at boundary points. That is, a robust implementation of sweeping 2D curves needs a planar-mapping pass to break down the collection of input curves into vertices (including intersection points), edges, and faces. Implementing planar mapping is nontrivial and fraught with numerical issues. So, you must allocate sufficient time for this implementation if it’s necessary. CGAL (the Computational Geometry Algorithms Library; www.cgal.org) provides an open source planar-map implementation.

Inflating 2D Shapes

These methods inflate the area bounded by the input 2D curves (see Figure 5). Although inflation is relatively more difficult than sweeps, it’s easier to use for modeling certain types of shapes. Owing to recent research advances, current inflation-based methods range from being very simple to use (but limited in scope) to being very powerful (but more difficult to use).

As Figure 5 shows, a typical inflation implementation specifies the region to be inflated by drawing a curve, tessellates that region (usually using triangles), and specifies how to lift the interior vertices from the plane. Among the first applications to demonstrate this technique was Teddy.6 Teddy places a chordal axis (an approximation to the medial axis) in the interior and connects it to the boundary with triangles. It then lifts the axis to produce a rounded cross section. You can also use Teddy to create rounded extrusions (resembling stuffed-toy limbs) and to create sharp edges.
by cutting out parts of the inflated model. Teddy remains popular because of its simplicity and computational speed. However, it’s limited to relatively simple, rounded shapes.

The FiberMesh project (see Figure 6) added the ability to draw curves on the existing inflated model. The inflated surface exactly (and sharply, if desired) passes through the drawn internal curves. Moreover, you can change the internal curves’ position while obtaining the new inflated surface interactively. That is, the internal curves serve as handles over the inflated model, which lets you create more complex shapes than if you were just using inflation. FiberMesh implements inflation by approximately minimizing the variation of curvature subject to position constraints specified by the curve. This produces smoother shapes (in general) than Teddy does. The disadvantage of this energy minimization is a lack of stability, even for some simple tasks.

James Andrews and his colleagues mitigated FiberMesh’s instability while adding more control for a surface near the boundary and curve handles. They developed a linear approximation of the nonlinear energy minimization used in FiberMesh, resulting in a faster, more stable system. They also let you control not just the position but also higher-order shape features (such as the surface normal and curvature). The increased stability and scalability along with the improved surface control produce a powerful, general-purpose modeling tool. However, compared to the simple, intuitive user interface in Teddy or FiberMesh, the user interface is more complex and difficult to use.

In Adobe Photoshop CS5 Extended, the Repoussé feature implements part of this method. As with sweeps, if several curves intersect in the 2D sketch, you need a planar-mapping pass over the input curves to identify individual regions to inflate. After computing those regions, you must tessellate them, usually as triangles. The triangulation approximates the smooth, continuous inflation surface as piecewise linear triangular pieces. The tessellation quality (shape of the elements) affects the inflated surface’s smoothness. So, unlike with sweeps, the triangles must be as close to equilateral as possible. In practice, this is interpreted as needing a Delaunay triangulation. To generate the triangulation, I recommend Jonathan Shewchuk’s Triangle triangulator, although you could use other open-source libraries such as CGAL. I also recommend having uniform-size triangles. Otherwise, rendering artifacts (namely, normal discontinuities) across edges that share triangles of different sizes become obvious.

Figure 5. Typically, inflation-based modeling takes a closed region bounded by the input curves, (a) tessellates that region, and (b) inflates it.

Figure 6. Local surface control in curve-based modeling. James Andrews and his colleagues added high-order shape control near the curve handles for added expressibility.

**Drawing 3D Curves**

These methods create surface models from drawn 3D curves. Although 2D curves provide a good starting point for creating 3D shapes, they’re not very useful if the curves themselves are to represent the shape being designed. For shape ideation (for example, creating concept shapes), 3D curves are more useful because they’re a sparse representation that can quickly convey the shape’s design without containing distracting aspects such as surface materials or shading effects.

Drawing 3D curves is, generally, much more difficult than drawing 2D curves. So, few off-the-shelf tools allow 3D-curve drawing. Professional tools such as Maya and 3ds Max let you place NURBS (nonuniform rational basis spline) curves directly in 3D space and manipulate their locations interactively. However, navigating to edit the
curves is difficult and time-consuming. Therefore, researchers are developing improved versions of 3D-curve drawing.

**Piecewise Planar Curves**

One strategy for producing 3D curves is to specify a 2D drawing plane on which you can draw piecewise planar 2D curves. To draw curves on a 2D plane, you can use either the faster sketch-based interfaces or the slower, precise methods I describe in this tutorial. This strategy builds on the maturity and ease of use of 2D-curve-drawing interfaces.

---

**One strategy for producing 3D curves is to specify a 2D drawing plane on which you can draw piecewise planar 2D curves.**

---

For this strategy, the key technical challenge is letting artists easily specify the drawing plane. The specification must be seamless so that it doesn't interfere with the creative process and so that it mimics, as much as possible, the experience of drawing on paper. ILoveSketch chooses a drawing plane parallel to the view plane and lets you rotate the camera (thereby changing the view plane) intuitively.\(^1\) It also lets you specify an extruded drawing plane by choosing a curve on the plane. A different approach lets you specify beforehand the planes that will form the canvas on which the curves are drawn. These scaffold planes can be specified intuitively, as Ryan Schmidt and his colleagues showed.\(^10\) Once you've started drawing curves, the subsequent drawing planes can be deduced automatically as you change the viewpoint parameters, as Adrien Bernhardt and his colleagues showed.\(^11\)

**Curves on Surfaces**

Another strategy for easily producing 3D curves is to draw curves on a curved canvas. That is, you could import a 3D surface as a stand-in canvas and project the 2D points from the view plane onto the canvas, obtaining 3D curves. Methods such as Teddy, FiberMesh, and ILoveSketch use this strategy. You can discard the canvas surface at any time or replace it with another surface to continue drawing the curve.

Drawing curves on a surface or series of surfaces lets you create complex 3D curves. However, you must decompose the desired curve into a series of curves drawn on different surfaces (which isn't intuitive). Also, unlike drawing curves on a plane, drawing curves on a curved surface isn't yet intuitive or easy.

To address this second drawback, Andrews and his colleagues introduced inactive constraints, which combine the ease of drawing 2D curves with the ability to deform a planar surface patch.\(^5\) You draw 2D curves on a flat surface that's then inflated or curved in some way. The 2D curves bend with the surface, becoming 3D curves. If the surface deformation is smooth (usually the case with inflation), the result is smoothly curved 3D curves. This method is suitable for a small category of 3D curves—those that can be considered smooth deformations of 2D curves. Of course, you can “activate” those 3D curves at any time to turn them into handles for deforming the surface.\(^9\)

**Surfacing 3D Curves**

Some applications might require converting a collection of 3D curves into a 3D surface that passes through those curves. This process, called surfacing, is useful for getting a better visualization of the shape implied by the 3D curves. It’s also necessary when the 3D-curve model is complex and dense. Surfacing lets you hide the display of the curves that would have been occluded by a surface, thereby making the model easier to understand.

One strategy for surfacing is to decompose the regions bounded by sections of the intersecting 3D curves into surface patches. That is, from the collection of 3D curves, you must extract patch boundaries. This is the 3D equivalent (and harder version) of planar mapping. For most 3D-curve models, finding appropriate patch boundaries is an ill-posed problem with no unique solution. So, such patch boundaries must be specified manually. Fatemeh Abbasinejad and her colleagues’ recent method (see Figure 7) is a good start for automatically obtaining a default specification of patch boundaries.\(^12\) However, even this method requires minimal user interaction for tweaking the automatically found boundaries.

After extracting the patch’s boundaries, you must extract its surface. You can segment the patch boundaries into four groups and map a quadrilateral patch representation (for example, a Bezier patch or Coons patch) onto the given boundaries. This works well for patches that are “almost” four-sided (whose boundaries can be decomposed coherently into four sides) but not for all patches. Finding the surface for an arbitrarily nonplanar n-sided patch, where n can be arbitrarily large, is difficult. One method obtains a rough approximation of the surface from the boundary curves and uses
nonlinear curvature-based optimization to obtain the final surface. Getting a reasonable initial approximation is often the difficult part for highly nonplanar boundary paths.

Alternately, you could express the entire 3D-curve collection as contours of an implicitly defined volume. You then obtain the desired 3D surface as that volume's zero-set surface. Variational radial basis functions usually provide a smooth implicit surface, as Emilio Brazil and his colleagues demonstrated. This method lets you easily make topological changes as the 3D curves are modified. However, it can't handle arbitrarily complex 3D-curve models without first decomposing them into simpler subsets. Also, the implementation and computational costs are relatively higher, especially with sharp features such as sharp edges or boundaries.

I hope this tutorial helps you build a solid foundation in your knowledge of this exciting new field, so that you can experiment with your own novel ideas.

References

Pushkar Joshi is a senior research engineer at Motorola Mobility. His research interests are geometric modeling, user interfaces, and visualization. Joshi has a PhD in computer science from the University of California, Berkeley. Contact him at pushkarj@siggraph.org.

Figure 7. Fatemeh Abbasinejad and her colleagues developed a method that takes (a) space curve models as input. It then extracts (b) boundaries of patches inferred by the input models.12