EUROGRAPHICS Symposium on Sketch-Based Interfaces and Modeling (2009)
C. Grimm and J. J. LaViola Jr. (Editors)

Sketching Subdivision Surfaces

M. Bein[1] and S. Havemann [2] and A. Stork[1] and D. Fellner[1,2]

[1]GRIS, TU Darmstadt & Fraunhofer IGD, Germany
[2]Institut für ComputerGraphik & WissensVisualisierung (CGV), TU Graz, Austria

Abstract

We describe a 3D modeling system that combines subdivision surfaces with sketch-based modeling in order to meet two conflicting goals: ease of use and fine-grained shape control. For the excellent control, low-poly modeling is still the method of choice for creating high-quality 3D models, e.g., in the games industry. However, direct mesh editing can be very tedious and time consuming. Our idea is to include also stroke-based techniques for rapidly modeling regular surface parts. We propose a simple and efficient algorithm for converting a 2D stroke to a control polygon suitable for Catmull/Clark subdivision surfaces. We have realized a small but reasonably rich set of interactive modeling tools to assess the expressiveness of stroke-based mesh design with a number of examples.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Modeling packages I.3.6 [Computer Graphics]: Methodology and Techniques—Interaction techniques

1. Introduction

The design of high-quality 3D shapes from scratch still is a difficult problem. A search for *"3d modeling tutorial"* on www.youtube.com returns about 27500 hits (April 2009). A quick evaluation of the first 60 video tutorials (ranked by relevance) yields the surprising result that about 80% of them are concerned with direct editing of low-polygon meshes. Although this is of course not a rigorous empirical study, it nevertheless indicates that there is a demand for efficient mesh editing methods.

It is an interesting research question why polygonal modeling is still so predominant despite the many alternative methods for shape design. NURBS, implicit surfaces, space carving, point clouds, Z-painting, and many others, all have their specific user community. But apparently, many shape professionals prefer stitching together surfaces manually, typically quad by quad. The obvious advantage of this approach is ***controllability***: Manipulations are very local, typically either single vertices are dragged or faces are split or joined by editing individual edges. Like shape design in general, polygonal modeling usually proceeds in a coarse-to-fine manner. Another observation is that most often, quads are used and not triangles. Regular quad meshes are compatible with parametric (e.g., tensor product) surfaces, and they are apparently well suited for man-made shapes. However, sometimes more flexibility is required than a regular grid can provide. In such cases, irregular vertices (valence $\neq 4$) or faces (triangles, pentagons, n-gons) are used.

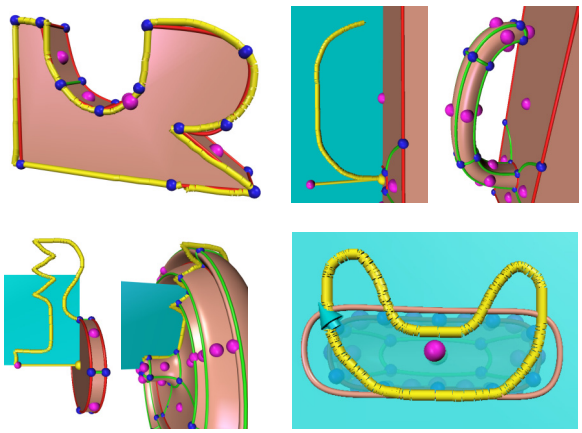

Figure 1: *Our system uses strokes for (a) sketching initial faces, (b) path sweeping, (c) extrusions, and (d) lofting.*

Sketch-Based Interfaces and Modeling 2009, New Orleans, LA, August 1–2, 2009.

This was the motivation for us to investigate how rapid stroke-based design can be integrated into a mesh modeling framework. Sketching should be used in such a way that primarily quad faces are produced. Furthermore, our system uses Catmull/Clark subdivison surfaces. This reflects a trend in professional polygonal modeling: Catmull/Clark surfaces are compatible with low-poly modeling; they are well suited for quadrangles; and they allow to get free-form surfaces "for free", as basically every manifold (*water-tight*) low-poly mesh is suitable as control mesh for Catmull/Clark.

2. Related Work

Sketching is often perceived as a method for the early conceptual design phase, focusing on rapid model creation. Accurate and detailed geometric model construction and manipulation still is the domain of high-end modeling systems like Maya, Rhino, SolidWorks, etc., with WIMP-style (windows, icons, menu, pointer) user interfaces. The challenge of *sketch-based interfaces for modeling* (SBIM) is to bring together the best of the two worlds.

Methods for stroke-based shape editing can be roughly divided into *semi-discrete* and *semi-continuous* methods. Semi-continuous methods operate directly on the surface, e.g., on densely sampled triangle meshes or point clouds. Examples include the pioneering *Teddy* application [IMT99] or recent inflation-based systems like Matisse [BPCB08] or Répousse [PC08]. Another class of approaches uses silhouette strokes or contour strokes, mainly to edit existing surfaces, for example the work from Nealen, Alexa et al., e.g., FiberMesh [NSACO05, ZNA08, NISA07].

Our system belongs to the class of semi-discrete methods, where strokes control only discrete proxy objects, which in turn manipulate the surface. One example are curve-based models: The FreeDrawer system from Wesche et al. [WD00, WS01] allows drawing curve networks between which surfaces are suspended; Schkolne et al. [Sch06] draw surface strips with free hands. Hui et al. generate surfaces from profile curves [HL07], and Wang et al., decompose suitable 3D objects into tubular pieces whose axis can be controlled with strokes. Skeletons can also be used to control implicit surfaces, as shown by Sugihara et al. [SdGWS08].

Very few sketch-based design systems are concerned with editing low-poly models, or with sketching control meshes for subdivision surfaces. Much early work was on recognizing "clean" polygonal objects from 2D sketches. However, the challenge in editing polygonal surfaces with strokes is that strokes have a continuous nature; so at some point, information is lost in the conversion. – The survey from Olsen et al. [OSSJ09] mentions five major open problems of SBIM. One is *Model Quality*, and they point out that bridging the gap between surface quality and ease of use is a major challenge. Another problem is *Precision*, where the control-point paradigm is usually perceived as being superior to semi-continuous editing. We want to address these two problems.

3. Stroke Processing

The key functionality of our system is the conversion of a sketched stroke to a reasonable control polygon. It shall describe a curve that reflects "best" the shape of the sketched stroke, furthermore

- the control polygon should have as few control points as possible,
- control points should be tagged as *smooth* or *sharp* (see sec. 4).

Since Catmull/Clark is a generalization of uniform bi-cubic B-spline surfaces [CE78], we can use a B-spline approximation scheme, e.g., least squares minimization:

$$\sum_{i=1}^{n} |d_i - c(t_i)|^2 \rightarrow min$$

The d_i are the stroke data points, t_i are the parameters assigned to the data points, and $c(t_i)$ is the B-spline evaluated at t_i. The minimization has a unique solution which can be found by solving a system of linear equations. The problem remaining is to assign suitable parameter values t_i for the points d_i of the sketch to solve for. Keep in mind that we use uniform B-splines and that assigning parameters to points automatically induces to segment the stroke. To solve that task we use this simple but effective greedy algorithm:

Initialization. The algorithm starts with a single segment for the whole input curve, control points are the first and last data point.

Parameters. As the B-spline is uniform, parameters must be assigned to data points such that every segment has the same length. We use a simple chord length mapping.

Approximation. Solving the minimization problem yields the control polygon

Evaluation. If all data points are closer to the corresponding curve points than the threshold, the algorithm terminates. If not, a new B-spline knot is inserted at the data point ***with the largest deviation*** adding another B-spline segment and control point.
In case the data point with the largest deviation is a knot, or very close to a knot, this knot is tagged as *sharp* instead of, e.g., increasing the knot multiplicity. This typically forces a tangent discontinuity.
In any case, another iteration is done (goto *Parameters*).

The algorithm is illustrated in Fig. 2. The accuracy can be controlled by the distance threshold which terminates the algorithm. The simple rule to insert the data point with the largest deviation as new knot does not always lead to the best approximation in the mathematical sense. But this heuristic has proven to work very well in the test cases because users quickly understand how it works. Especially the corner selection works reliably and transparently, even without any explicit corner detection scheme being applied (for an overview of corner detection methods see [Ham09]).

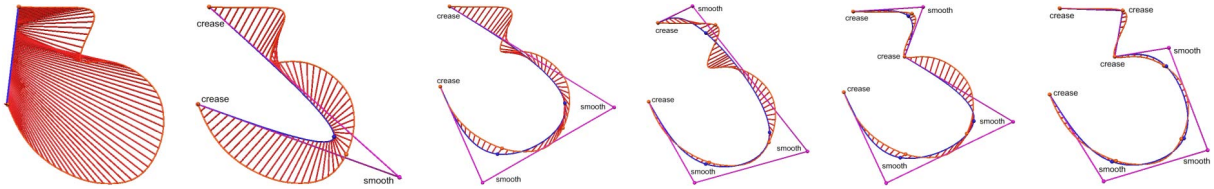

Figure 2: *Iterative Approximation of a sketched curve. The left image shows the initial curve approximation with only two knots, the line segment between first and last data point. The distances between the sketch (orange) and the B-spline curve (blue) are shown in red. The next images show successive iterations, with new knots being inserted at the data points with the largest deviation. The last picture shows the seventh iteration, where 5 knots were added and 2 knots were tagged as sharp.*

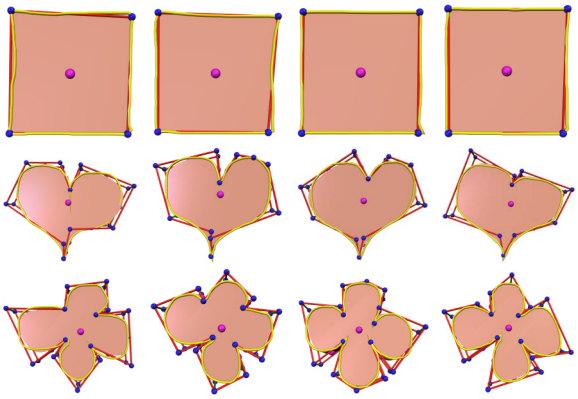

Figure 3: *Given similar strokes, the sketch processing yields very similar control polygons. Users perceive the method therefore as very* ***predictable****, and experience not much frustration from unexpected results.*

To assess the usability of the method we have urged users to sketch repeatedly some simple shapes (Fig. 3). Due to the uniform knot spacings, the control vertices are also quite evenly spaced along the curve. Although the number of control points may "jump", their spacing remains stable. – The result of the stroke processing is a control polygon, i.e., a list of tagged points (smooth or sharp). The knots are uniform, so no explicit knot vector is necessary. Thus, the control polygon can be directly used with Catmull/Clark.

4. Underlying Shape Representation

The system integrates Catmull/Clark surfaces with standard B-reps to combine free-form with polygonal modeling. Without restrictions the edges of the B-rep can be tagged as *smooth* (green) or *sharp* (red). When all edges are sharp, the surface is rendered as standard polygonal mesh (*polygonal faces*); face planarity is not enforced, though. Each face with at least one smooth edge is rendered as Catmull/Clark surface (*smooth faces*). Finally, *sharp faces* are flat like polygonal faces but can have B-Spline boundaries and are used at the transition between smooth and polygonal surface parts.

The control mesh is a boundary representation (B-rep) implemented in C++ as half-edge data structure. The resulting shape representation is therefore called *combined B-rep* or short cB-rep. For rendering, cB-reps use an optimized computation scheme that allows gap-free tesselation-on-the-fly as well as adjusting the level-of-detail for each face in every frame in a view dependent way.

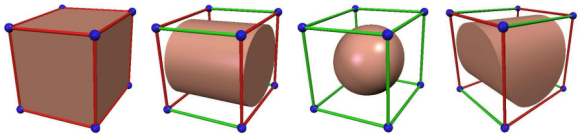

Figure 4: *The same control mesh with different sharpness flags. Sharp edges are red, smooth edges green.*

cB-reps are efficient for rendering but provide no modeling functionality. This is contributed by another layer, finally resulting in *progressive combined B-reps* (pcB-reps) [HF08]. They act as a middleware allowing geometry modification exclusively through *Euler operators*. This is a small but provably closed and complete set of operations for manifold meshes, for example makeEF to insert a diagonal to a face, or makeEV to split up a vertex in two vertices connected by an edge. There are 10 Euler operators in total:

makeVEFS	makeEV	makeEF	makeEkillR	makeFkillR
killVEFS	killEV	killEF	killEmakeR	killFmakeR

The pcB-reps keep a log of all operations applied to the mesh, the *Euler sequence*. Euler operators are invertible. This is a highly desirable property for interactive shape design: It allows to undo the result of a previous modeling operations, and to replace it by another, e.g., when dragging interactively a control vertex, or adjusting when an extrusion. So undo/redo is provided on the middleware layer.

Euler operators are typically not directly exposed to end-users. Instead, they are a basis for creating higher-level modeling operations, e.g., extrusion or lofting, as described next.

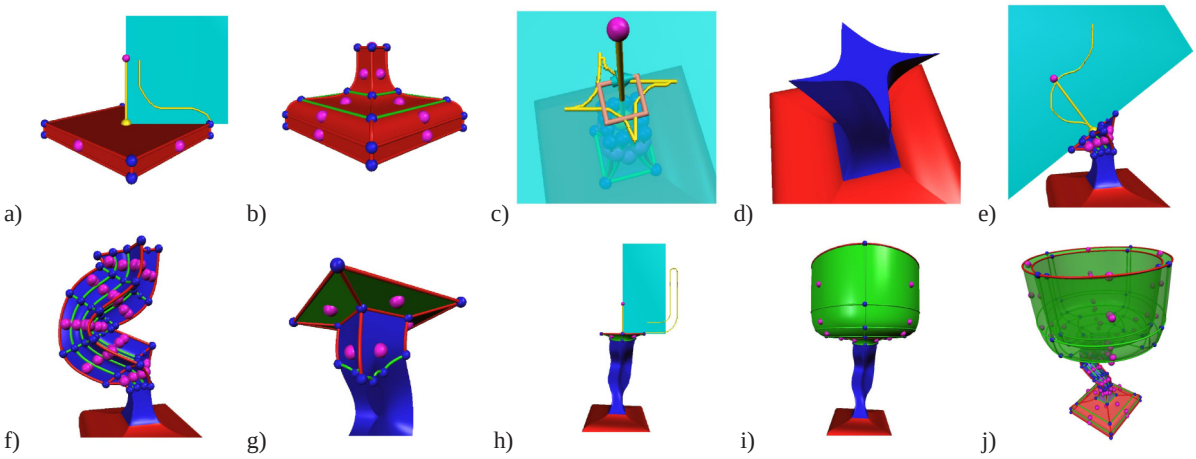

Figure 5: *Example of a modeling session to create a fancy drinking mug. First the base is created by rotational extrusion of the selected square (a,b). Then the square face is transformed to a star shaped face by lofting (c,d). This face is slightly rotated to the side, and a path extrusion is sketched (e) which results in the glass stem shown in (f). A square face is formed out of the star shaped stem using low level modeling operations (g). The glass is completed with a rotational extrusion of the square face on top of the stem (h). Switching the sharpness of edges to smooth finally rounds the surface of the glass (i,j).*

5. Modeling Tools

Our sketching tool uses the gizmo approach. A *gizmo* is a 3D object that stands for an operation that can be applied. Our most common gizmos are sticks for edge-based operations and balls for vertices and faces. Operations are typically issued by clicking on a gizmo and dragging. Since there are many modeling operations, all three mouse buttons had to be seized for almost all gizmos. Advantage of the gizmo approach is that it is very immediate and visual compared to, e.g., menu selection or keyboard shortcuts. Our system needs *no* conventional 2D GUI elements *at all*, not even as head-up displays; everything is displayed in 3D. This makes it suitable for non-GUI environment, e.g., modeling in a web browser, in a presentation environment, or in a CAVE.

The disadvantage of the gizmo approach is that it is mostly non-textual, i.e., not very descriptive at first. However, users tend to learn the 3D toolkit faster by trial-and-error than by reading the manual, which is some indication that the method is intuitive. Conventional non-3D menu selection and shortcuts are used in our system for activating switching between one of three available *operation modes*:

1. **Sketching**: Stroke-based modeling operations
2. **Drag and flag**: Moving components, sharpness edits
3. **Insert and remove** of vertices or edges

Each of these modes provides in fact its own assignment for each gizmo of functions to mouse buttons. I.e., when clicking on a vertex ball, the left mouse button behaves differently depending on which operation mode is active. This is nonetheless intuititive since the operation models group similar functionalities together.

5.1. First Operation Mode: Sketching

Creation of a new component with a sketched silhouette (Fig. 6). The user draws the silhouette of the intended shape onto a drawing plane. When releasing the right mouse button, a two-sided face is created and extruded.

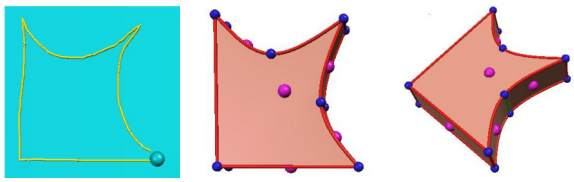

Figure 6: *Creating a new object by sketching a face.*

Rotational extrusion of a face with a sketched silhouette (Fig. 7). The user selects a face and holds down the mouse button on a vertex to initialize the rotational extrusion. This brings up a drawing plane for the user to sketch the extrusion. The plane contains the vertex, face normal and face midpoint. When the mouse button is released, the face is extruded in normal direction, and scaled, according to the silhouette. To keep the tool simple, the center of the rotational extrusion always lies on the ray in normal direction emanating from the face midpoint. – As shown in Fig. 7, rotational extrusions can be copied to other faces.

Lofting a face with a sketched silhouette (Fig. 8). By selecting a face and clicking on the normal stick the user brings up a drawing plane. It is parallel to the selected face and can be slided up or down. Once it is fixed, a closed curve can be drawn. When the mouse button is released, a lofted face in the shape of the curve is created and connected

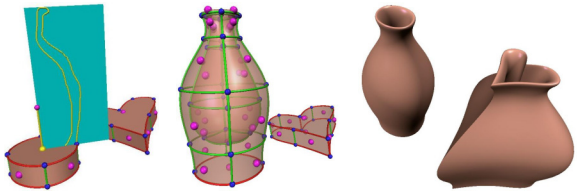

Figure 7: *Sketch of a rotational extrusion, the result with transparent material, and the result of copying the extrusion to the heart-shaped object.*

(with quads) to the previous face. The lofted face must have the same number of vertices as the selected face, which restricts the quality of the approximation.

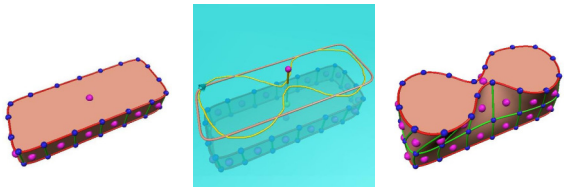

Figure 8: *A rectangle shape, a lofting sketch (yellow) and the original silhouette (brown) on a plane above the shape. Right: the result of the lofting operation.*

Path Extrusion (Sweeping) (Fig. 9). Holding the right mouse button on a face midpoint brings up a drawing plane. It contains the face normal, face midpoint, and the cross product of the viewing direction and face normal. The user can sketch the path of the extrusion on the plane. When releasing the mouse button, copies of the selected face are placed all along the path and connected.

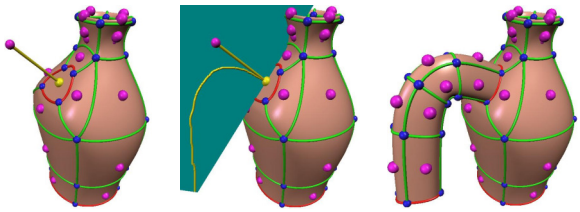

Figure 9: *A model of a vase, a sketch of a extrusion path and the result of the sketching operation.*

Face Path Insertion along a path sketched from one vertex to another vertex of the same face (Fig. 10). The face is split in two by inserting a path of valence 2 vertices.

5.2. The two other Operation Modes: Drag-and-Flag, Insert-and-Remove

These modes are for direct low-level manipulations of the control mesh components. Every sketch operation, which is

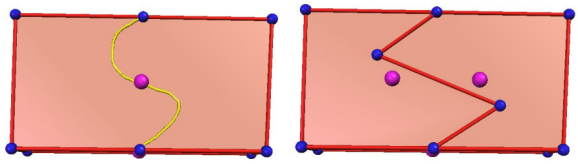

Figure 10: *A sketch on the surface inside a face and the inserted edges which split the face along the sketched path.*

quickly done, is typically followed by tedious fine-grained shape editing, where many action sequences of the form

selecting – adjusting – view change

are issued. The possible adjustments are:

- **Drag a vertex** parallel to the view plane
- **Drag a face** parallel to the view plane
- **Rotate a face** around the face center
- **Switch the sharpness** of a vertex, edge and face
- **Insert a new vertex** on an edge
- **Insert a new edge** between two vertices
- **Remove** individual vertices or edges
- **Connect** two faces of the same degree

Additional operations that are issued less frequently:

- Rotate a face to the direction of the mouse position
- Copy a rotational extrusion to another face (Fig. 7)
- Change the material of a face
- Undo the last mesh operation

Figure 5 shows an exemplaric modeling session that illustrates what is possible through the combination of the tools.

6. GML Scripting

Our sketch-based 3D modeling toolkit is implemented in GML. The *Generative Modeling Language* from Havemann [Hav05] is a very simple stack-based programming language, syntactically very similar to PostScript. GML provides operators for geometric calculations in 3D, and it provides an operator layer on top of the pcB-rep meshes (section 4). Euler operators are available for modeling on the half-edge level, as well as higher-level modeling tools.

GML scripting yields the necessary flexibility to try out quickly several different approaches to sketch-based modeling. The integration of the sketching functionality was not difficult, it was essentially sufficient to add only two new GML operators (which are implemented in C++):

$$[\ p_0 \ldots p_n\ ] \quad \text{gaussian-filter} \quad \longrightarrow \quad [\ q_0 \ldots q_n\ ]$$
$$[\ a_0 \ldots a_n\ ]\ i_{\min}\ i_{\max}\ \varepsilon \quad \text{approximate} \quad \longrightarrow \quad [\ m_i\ ]\ [\ b_i\ ]$$

The gaussian-filter operator smoothes out the sequence p_i of stroke input points, pushing as result on the stack an array q_i of 3D points that has the same size as the input. This operation can be easily iterated. The approximate operator

takes as input the (possibly smoothed) stroke points a_i. It performs i iterations, $i_{\min} \leq i \leq i_{\max}$, of the algorithm described in section 3, with ε as accuracy threshold (distance stroke/B-spline). The result is the sequence b_i of B-spline control vertices, and an array m_i of (integer) flags indicating which nodes are sharp (crease) or smooth.

```
1  dict {
2    mouseButton 1 eq {
3      /pt pNear pFar (0,0,1) 0.0
4           intersect_lineplane pop
5      def
6      mouseEvent 1 eq { /points [ pt ] def      } if
7      mouseEvent 2 eq { points    pt    append } if
8      mouseEvent 3 eq {
9        /point points 4 { gaussian−filter } repeat def
10       points 4 40 0.2 approximate
11       /cps exch def pop
12       deleteallmacros
13       cps 5 poly2doubleface
14       (0,0.01,5) extrude
15       pop
16       bns−clear
17       0.02 /stdBlue bns−style−stick
18       points 0 0 bns−poly
19       0.02 /stdRed bns−style−stick
20       0.04 /stdRed bns−style−ball
21       cps 0 0 bns−poly
22       cps { bns−ball pop } forall
23     } if
24   } { ioremoveall } ifelse
25 } iocapturemouse
```

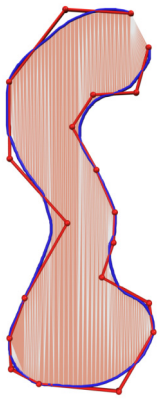

Figure 11: *GML code to create a double-sided polygon from an input sketch. It registers a mouse callback that keeps adding points as long as the left-button is dragged. On release it produces a model similar to the one shown.*

The example in Fig. 11 shows the use of these operators. The iocapturemouse operator (line 25) registers a mouse callback function. The left button activates a drag operation (lines 3-23), any other button unregisters the callback (line 24). First of all, the mouse pick ray from point p_{near} to p_{far} is intersected with the (x,y)-plane, producing point pt (lines 3-5). A pick (left-push) initiates a new stroke (line 6), drag keeps adding points (line 7). When the button is release, a new 3D object is created: The stroke is filtered 4 times (line 9), and then the control polygon is computed and stored as cps. The multiplicities are discarded (lines 10/11).

Line 12 destroys all existing Euler sequences, line 13 creates a double-sided face from the control polygon, leaving a halfedge on the stack. It is consumed by the extrude operator (line 14), which extrudes the face by 0.01 in vertical direction without shrinking it; the 5 is a mode flag, just as with poly2doubleface. The halfedge produced by extrude is discarded in line 15. Lines 16-22 finally just show the face boundary in blue and the control polygon in red (tube radius 0.02). The image clearly shows the effect of the required approximation quality of only 0.2. – This example shows that reasonable results can be obtained quickly with GML.

7. Results and Discussion

The sketching tool does not yet enjoy a large user base, which impedes a rigorous empirical evaluation. Nonetheless good results have been produced and important conclusions can be drawn. Although the sketching functions modify the cB-Rep control mesh automatically it is still important for the user to understand the mechanism of the control mesh. Knowing which modifications have to be done to obtain a specific result, or to prepare the control mesh for subsequent sketching operations can very much improve the quality of the resulting model. To have a construction process in mind is very helpful to create a model effiently making good use of the sketching functions.

Both the cB-Rep mechanism and procedural thinking can be learned by using the sketching tool. As the interaction is done intuitively in 3D, and the number of operations is small, the whole sketching tool is easy to learn. Explaining the tool to test users and giving a short introduction takes only about 15 minutes. Another 15 minutes are sufficient for the user to exercise the functions, basic principles, and to collect first experiences. Then the user is ready to create a first model. Figure 12 shows the first results of two users who never created a 3D-model before in their lives.

The test user of the table lamp almost started right away with creating models. After 5 minutes of experiencing the tool the user started to create the table lamp and successfully finished the construction within a few minutes. The user of the alien like creature spent more time to experience the sketching tool. Several low level mesh modifications like inserting edges and moving vertices where needed on the

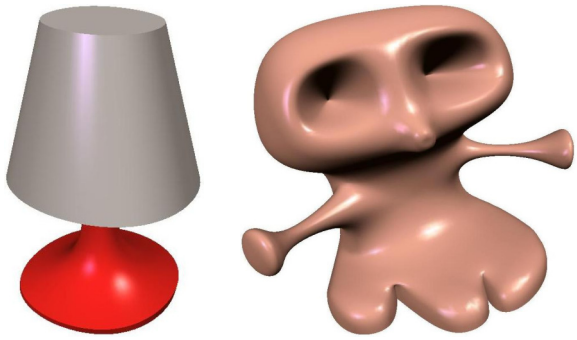

Figure 12: *A table lamp and a not classified alien like creature (maybe a paperweight) created by two test users.*

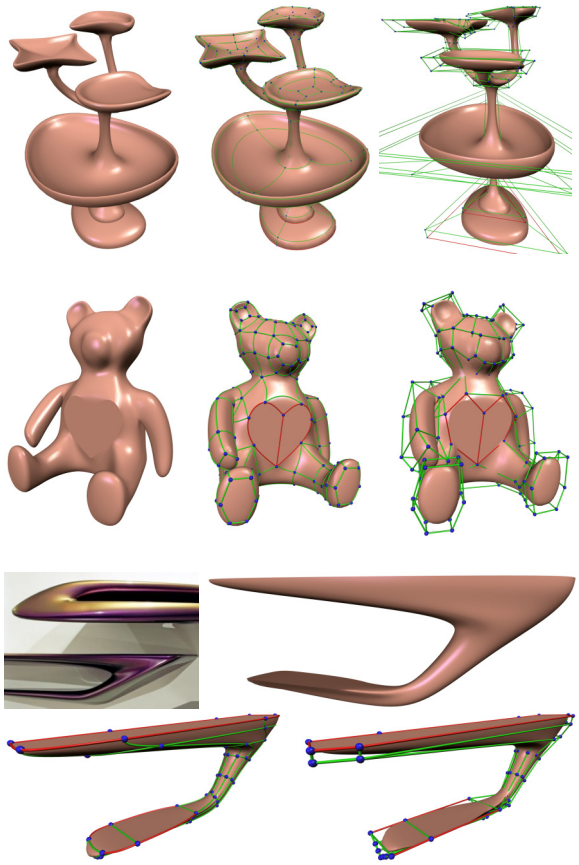

Figure 13: *High-end results. Fountain (30min), Teddy (60min) and a shelf reconstructed from a picture (10min). The bottom object is inspired by furniture from Zaha Hadid.*

model which took about half of the 20 minutes construction time. With some more time spent experiencing the sketching tool we created many models, see Figures 14 and 13.

In some situations the sketching tool does not behave as intended by the user. Row 1 in figure 15 shows the sweeping operation creating self intersections because of a large base face. Row 2 shows the lofting operation, which requires enough control points to be available for approximating the sketch. But in this case the four available control points allow to approximate the sketched silhouette only as a circle. Row 3 shows the creation of a new face which works as intended, but trying to create the table lamp this way is tedious.

In our test cases the sketching tool proved to successfully create freeform models fast. The sketching operations encourage a procedural modeling style, and the non-sketching operations allow changing models intuitively and precisely. The tool allows creating any possible manifold surface in GML without the need to resort to coding the construction in a stack-based language, for which GML is often used.

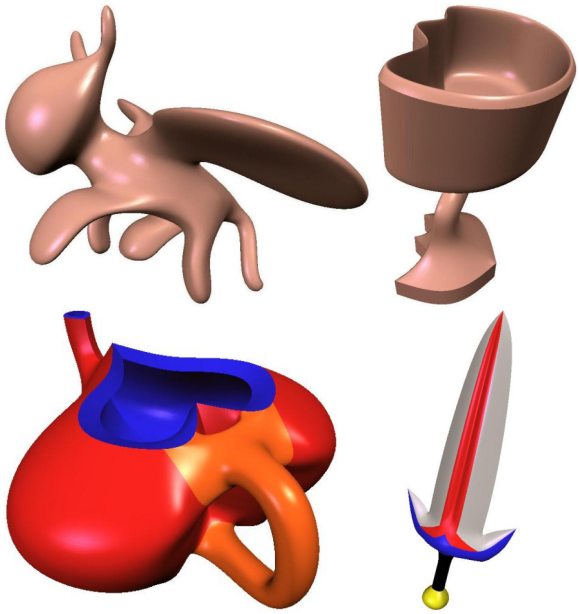

Figure 14: *Models created with GML sketching. Top: a mythical creature (25min) and a goblet (6min). Bottom: a heart-shaped teapot (25min) and a sword (20min).*

8. Conclusions and Future Work

Our experience so far indicates that the following extensions would be very useful:

- Oversketching to edit a sketch afterwards
- Life update of the control polygon while skeching
- Grid snapping for the control mesh, or for the control polygon of a sketch; alignment to existing geometry
- Symmetry planes for mirroring objects and operations
- Surface diagnosis to support control point placement
- Advanced sketching operations: Variable width sweeping, rotational extrusion with variable center and normal
- Moderate head-up display to improve overall usability

Since our sketching tool is realized in GML, a language for procedural modeling, the obvious next step is to find ways of ***combining sketching with procedural modeling***. Only a first step in this direction is copying a rotation extrusion to other faces. A key task for realizing more involved procedural operations is to extract rules out of the user actions; or to give the user to possibility to specify rules explicitly. All the user's selections have certain relations to each other. For instance, in our heart shaped teapot (figure 14 (c)), the handle and the nozzle are on opposite sides. This relation can not be captured automatically when the user just selects arbitrary faces. Without the rule *"'opposite to each other"'* it is not possible to apply the construction process to a different basic shape, like a circle, to create a round teapot. Translating selections into rules is essential for more involved pro-

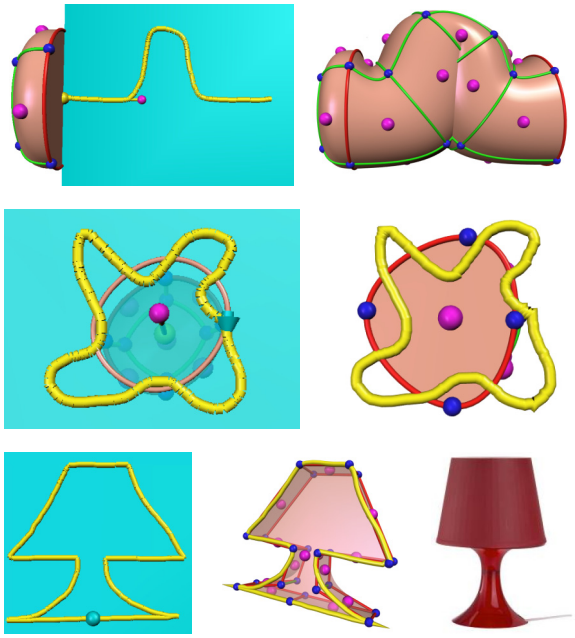

Figure 15: *Examples of confusing results.*

cedural models. Another promising challenge is including flow control like if-else conditions or loops. One way to realize these features is implementing an intelligent user interface that visualizes procedural rules and dependencies, and offers gizmos for entering rules, attaching parameters, creating functions, and for flow control.

Another promising area for extensions is ***high-quality surface reconstruction***. In many industrial applications, e.g., car design, the NURBS model must be "better" than the scanned surface, which is possible only if the reconstruction is controlled by an experienced human operator. With current technology, building a freeform model on top of a scan is very, very tedious. With specialized sketch modeling functions it may be possible to generate a subdivision control mesh interactively from a given triangulated surface. The user might sketch on the surface of the object while the approximation generates control polygons and meshes. The benefit would be a great reduction of surface complexity and the generation of semantically meaningful meshes. The goal is to obtain a coarse but accurate control mesh, because fewer polygons would increase the re-usability of a mesh and ease its further processing.

Acknowledgement

We gratefully acknowledge the generous support from the European Commission for the research project **3D-COFORM** (**3D CO**llection **FORM**ation, 3D-coform.eu) under grant number FP7 ICT 231809.

References

[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)* (Annecy, France, june 2008), Alvarado C., Cani M.-P., (Eds.), pp. 57–64.

[CE78] CATMULL E. C. J.: Recursively generated b-spline surfaces on arbitrary topological meshes. *Computer Aided Design 10* (1978), 350–355.

[Ham09] HAMMOND T.: Introduction to sketch recognition. In *Eurographics 2009 Tutorial* (April 2009), Eurographics Association.

[Hav05] HAVEMANN S.: *Generative Mesh Modeling.* PhD thesis, Braunschweig Technical University, Germany, November 2005.

[HF08] HAVEMANN S., FELLNER D.: Progressive combined b-reps - multi-resolution meshes for interactive real-time shape design. *Journal of WSCG 16*, 1-3 (2008), 121–135.

[HL07] HUI K. C., LAI Y. H.: Generating subdivision surfaces from profile curves. *Comput. Aided Des. 39*, 9 (2007), 783–793.

[IMT99] IGARASHI T., MATSUOKA S., TANAKA H.: Teddy: A sketching interface for 3d freeform design. In *Proceedings of SIGGRAPH 99* (Aug. 1999), Computer Graphics Proceedings, Annual Conference Series, pp. 409–416.

[NISA07] NEALEN A., IGARASHI T., SORKINE O., ALEXA M.: Fibermesh: designing freeform surfaces with 3d curves. In *SIGGRAPH '07: ACM SIGGRAPH 2007 papers* (New York, NY, USA, 2007), ACM, p. 41.

[NSACO05] NEALEN A., SORKINE O., ALEXA M., COHEN-OR D.: A sketch-based interface for detail-preserving mesh editing. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers* (New York, NY, USA, 2005), ACM, pp. 1142–1147.

[OSSJ09] OLSEN L., SAMAVATI F. F., SOUSA M. C., JORGE J. A.: Technical section: Sketch-based modeling: A survey. *Comput. Graph. 33*, 1 (2009), 85–103.

[PC08] PUSHKAR J., CARR N. A.: Répousse: Automatic inflation of 3d artwork. In *Eurographics Workshop on Sketch-Based Interfaces and Modeling (SBIM)* (Annecy, France, june 2008), Alvarado C., Cani M.-P., (Eds.), pp. 49–55.

[Sch06] SCHKOLNE S.: Making digital shapes by hand. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Courses* (New York, NY, USA, 2006), ACM, pp. 84–93.

[SdGWS08] SUGIHARA M., DE GROOT E., WYVILL B., SCHMIDT R.: A sketch-based method to control deformation in a skeletal implicit surface modeler. In *Eurographics Workshop on Sketch-Based Interfaces and Modeling (SBIM)* (Annecy, France, june 2008), Alvarado C., Cani M.-P., (Eds.), pp. 65–72.

[WD00] WESCHE G., DROSKE M.: Conceptual free-form styling on the responsive workbench. In *VRST '00: Proceedings of the ACM symposium on Virtual reality software and technology* (New York, NY, USA, 2000), ACM, pp. 83–91.

[WS01] WESCHE G., SEIDEL H.-P.: Freedrawer: a free-form sketching system on the responsive workbench. In *VRST '01: Proceedings of the ACM symposium on Virtual reality software and technology* (New York, NY, USA, 2001), ACM, pp. 167–174.

[ZNA08] ZIMMERMANN J., NEALEN A., ALEXA M.: Sketch-based interfaces: Sketching contours. *Comput. Graph. 32*, 5 (2008), 486–499.