

# Modeling Solids and Surfaces with Sketches: an Empirical Evaluation

Manuel Oliveira

Vladimiro Colaço

Joaquim Jorge

Manuel Fonseca

Departamento de Engenharia Informática, IST/UTL  
Av. Rovisco Pais, 1049-001 Lisboa, Portugal

oliveira@syscog.pt

vladimiro.colaco@megamedia.pt

jorgej@acm.org

mjf@ist.utl.pt

## ABSTRACT

This paper presents and evaluates a simple editor for modeling solids and surfaces. The editor uses sketches and gestures as the main interaction paradigm. We want to show that sketch-based interaction for creating 3D scenes is more natural and intuitive than conventional approaches.

**Keywords:** 3D Scene Modeling, Sketch-based interaction, Usability testing

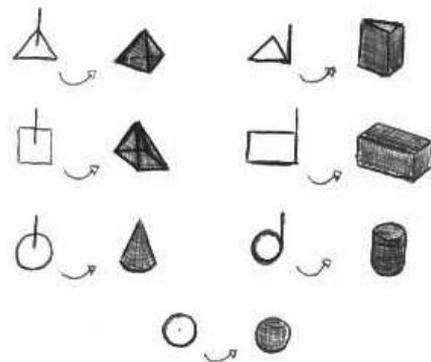
## 1. INTRODUCTION

Creating 3D scenes using conventional applications based on the WIMP (Windows, Icons, Mouse and Pointing) paradigm, is mostly a hand and unusual task. Nearly all direct-manipulation based applications use too many menus with too many options, making their use less intuitive and the interactions tedious and time-consuming. To overcome this problem we developed a 3D-scene editor based on sketches. The created scenes can be saved in Quake™ format. Our application uses the paper and pencil metaphor, offering a set of gestures as the principal way of interaction. Gestures are drawn using a pen and a digitizing tablet expressing graphic primitives (e.g. cones, spheres, etc.) or commands such as delete, copy, etc. We intend to show that interaction techniques based on sketches in drawing applications are faster and more intuitive than menu- and forms- driven interfaces. To this end we have conducted usability tests of our prototype comparing its performance with more conventional approaches.

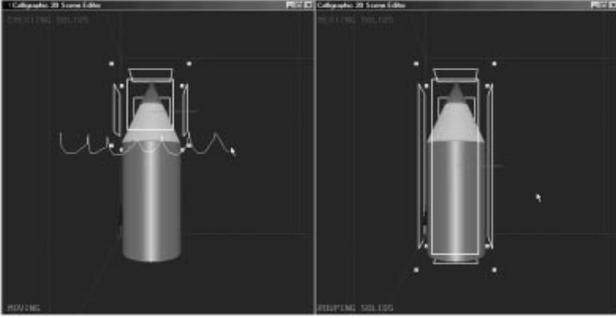
## 2. SKETCH-BASED INTERFACE

Sketch-based interfaces are organized around using gestures and drawings produced using a pen and a digitizing tablet. Individual gestures and drawing commands are identified using a shape recognizer [Fonse00]. Appendix A lists

the figures and gestures supported by the recognizer. Drawing commands create solid primitives as a combination of gestures with a specified order and semantics (e.g. to create a cylinder



der we draw a circle and then a line starting at the center of the circle). These define the syntax of a visual language used to create three-dimensional drawing primitives and executing commands, some of which are exemplified in Figure 1. The size of the shapes and the length of lines thus sketched define attributes of new primitives to be created (e.g. height, width, rotation, position, etc.) which translates to significant savings in commands and interactions as compared to more conventional approaches. We call this feature calligraphic 3D iconic input.

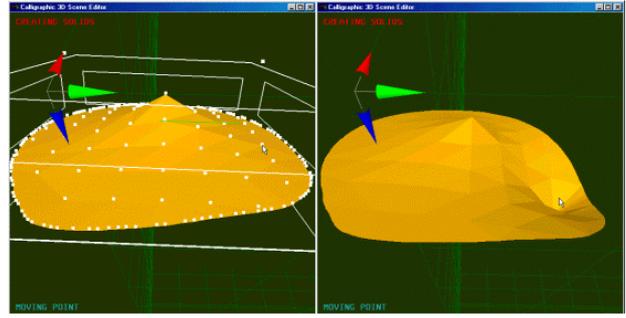


**Figure 2: Grouping solids using one gesture**

The editor supports the creation of simple solids, such as Cubes, Spheres, Cylinders, Cones, Prisms and Pyramids, just by sketching combinations of gestures. The main attributes of the solids are defined when they are created, without the need to invoke the usual boring menus. There are sets of operations that can be performed on solids, such as copying, deleting, changing color, cutting, applying textures, grouping, etc.

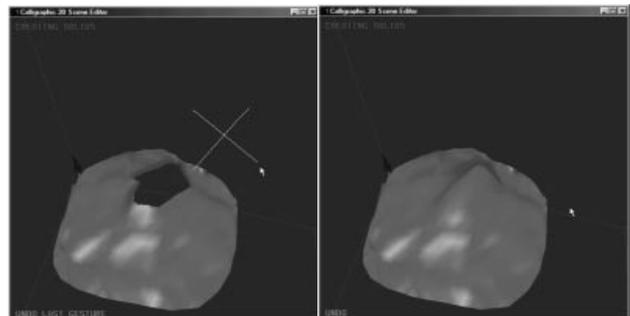
We use natural and intuitive gestures to perform these operations usually bearing a close mnemonic relation with the semantics of the operations we have in mind. For example, we use hand-drawn "C" to signify Copy command or a "WavyLine" representing a sewing operation to issue a Grouping command. This way, rather than wasting our time searching for options on menus or trying to remember shortcuts, we just have to sketch the commands. Furthermore, these sketch-based commands go a bit further than direct manipulation techniques, because they implicitly select the solid to apply the operation to, as we can see in Fig. 2. We can thus see that gestures are more expressive than conventional commands, because they can a) specify the action to be performed b) additional arguments such as geometry attributes and c) the object or objects upon which the action is to be performed in *one single interaction*. This expressiveness make gestures and sketches a better match for drawing-type applications than conventional direct manipulation environments, where the syntax of the interface gets in the way.

Our editor also allows creating three-dimensional surfaces from free-hand two-



**Figure 3: Creating and deforming a surface**

dimensional contours. After specifying the contour, we can modify surfaces and deform them by *pulling points* as depicted in Figure 3. We can also *cut* surfaces, by drawing a cutting line or even the application of a fractal roughing method to emulate "real" terrain. Finally, we can undo destructive operations applied to solids or surfaces, such as *delete*, *cut* or *roughing*, by drawing a cross over the object, as illustrated by Figure 4.



**Figure 4: Cutting surfaces and undoing**

### 3. IMPLEMENTATION DETAILS

Our editor was developed in C++ under Ms/Windows. We used the  $C_{al}$  library for recognizing commands and 2D primitives [Fonse99].

We use the OpenGL™ graphics library, and the OpenGL Utility Toolkit [Woo97, Kilg96] to create and display graphical objects and simplify a few direct-manipulation commands, e.g. picking and resizing three-dimensional objects.

Creating and deforming surfaces uses Delaunay triangulation to describe adjacencies and maintain surface consistency. We use the Super Delaunay Library [Kornmann00], to support these operations..

#### 4. USABILITY STUDY

To measure advantages and disadvantages of our editor we made a usability study involving nine users without any experience in this domain. We divided the users into two groups, one used a tablet with a built-in display and the other used a simple tablet. Both groups did the same experimental procedure. First, they answer a written inquiry where they have to define a vocabulary for a calligraphic domain. We want to notice that users did not have any previous contact with the application. A second phase was used to get users acquainted to our editor and to the conventional editor we used to compare. Third, we ask users to perform a set of steps that lead to clear values, allowing the withdrawal of results on which the conclusions are based. Finally, they answer a second written inquiry, in which we asked them to compare their model with the application's and to grade a set of items. This study allows the extraction of results of two different kinds, one is the number of errors and time spent per operation, and the other is users' feedback, based on their grades to a set of items such as intuitiveness and simplicity of the operations offered.

#### 5. RESULTS

The analysis of the first questionnaire revealed that users are quite unanimous about the way to represent simple solids. The test itself uncovered some interesting results. For instance, users made five times more errors with the simple tablet than with the other. Another interesting result is the gain of experience during the use of the editor. Users take less time to create and delete objects after some experience. We also noticed that the creation of surfaces using the tablet with display is far less time-consuming than the conventional methods. Comparing our editor with a map-building tool (BSP) shows that the latter is better for regular scenes oriented along the axes. Nevertheless, our editor has the user preference, as far as the global intuitiveness is concerned. The second inquiry revealed that users appreciated the generality of supported operations, as well as the simplicity of sketch-based edition. It also revealed that there are some problems to solve, such as, a

better navigation method for the camera and the implementation of spatial constraint satisfaction techniques to ease the process of combining objects within the scene.

#### 6. CONCLUSIONS

As seen, the results are quite good and the users revealed satisfied with the generality of the aspects concerning the calligraphic model and this editor, in particular. However, a few items are still blurring these results and these would naturally lead to desirable new functionality, like the inclusion of restriction satisfaction algorithms and the inclusion of several simultaneous views to solve the lack of three-dimensional assistance.

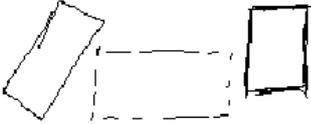
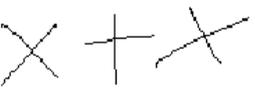
#### 7. ACKNOWLEDGMENTS

This work was supported in part by Portuguese Science Foundation (FCT).

#### REFERENCES

- [Fonse00] Fonseca, M. J. and Jorge, J. A. Using Fuzzy Logic to Recognize Geometric Shapes Interactively. In *Proc. of the FUZZ-IEEE 2000*, San Antonio USA, May 2000.
- [Fonse99] Fonseca, M.J., *CaI: A Software Library for Calligraphic Interfaces*, 1999, <http://immi.inesc.pt/~mjf/>
- [Igara99] Igarashi, T., Matsuoka, S., Tanaka, H., *Teddy: A Sketching Interface for 3D Freeform Design*, University of Tokyo, Tokyo Institute of Technology, SIGGRAPH 99, 1999.
- [Kilg96] Mark J. Kilgard, *The OpenGL Utility Toolkit (GLUT) – Programming Interface – API Version 3*, Silicon Graphics, Inc., Nov 13, 1996
- [Kornmann00] David Kornmann, “*The Super Delaunay (Indexed) Library*” (SDI), <http://www.iki.fi/~david>
- [Watt93] Alan Watt, *3D Computer Graphics*, second edition, Addison-Wesley, 1993
- [Woo97] Woo, M., Neider, J., Davis, T., *OpenGL® Programming Guide, Second Edition*, OpenGL Architecture Review Board, Addison-Wesley, 1997.

**APPENDIX A: 2D SHAPES AND GESTURES RECOGNIZED**

 <p>Line</p>	 <p>Arrow</p>	 <p>Rectangle</p>
 <p>Circle</p>	 <p>Ellipse</p>	 <p>Diamond</p>
 <p>Triangle</p>	 <p>Delete</p>	 <p>WavyLine</p>
 <p>Move</p>	 <p>Copy</p>	 <p>Cross</p>

**Figure 1: Figures and Commands Recognized by the C<sub>a</sub>I library**