A Sketch-Based Interface for Architectural Modification in Virtual Environments

Dominik Rausch Ingo Assenmacher

Virtual Reality Group
RWTH Aachen University
e-mail: dominik.rausch@rwth-aachen.de

Abstract: This paper presents a sketch-based interface for interactive modification of architectural design prototypes inside a virtual environment. The user can move around in the scenery and create line drawings to add annotations. In order to modify or extend the scenery, she can sketch command symbols, which are then recognized by the application to trigger the application defined commands. This allows the user to modify existing objects in the scene as well as to create new ones. This sketch based interface is supposed to act as a front-end application for architects in order to have a tool for the interactive reconfiguration of rooms and interior in a visual-acoustic virtual environment.

Keywords: Sketching, Interaction, Gesture Recognition, Virtual Reality, Virtual Prototyping

1 Introduction

There are different approaches for designing architectural prototypes, like pencil sketches, physical prototypes, and virtual 3D models. The latter have the advantage that they can be examined in a virtual environment (VE), where flaws of the model may be visible that are imperceptible in 2D images or small-scale physical prototypes. But usually, there is no way to directly change these flaws inside the VE. This often requires to leave the VE, use a specialized modeling tool to apply the modifications, and reload the model.

This paper describes a prototype application that provides an interface which allows a user to perform modifications of a room's interior directly inside the VE. She can add new objects – like chairs or doors – to the scenery. Alternatively, the user can change the state of existing ones, e.g. by moving a table. In some cases, the user may want to perform complicated modifications that are not supported or feasible in the VE. In this case, the system supports the annotation on virtual objects which can be used as a note-taking tool to later use these notes and apply the desired changes in an external and more specialized application [AHC+06].

It is not the purpose of this application to offer full modeling or CAD support inside the VE, but instead to provide means to have a high level interface for the modification of rooms and their interior, like building, moving, and changing materials of walls, or adding, moving, or deleting furniture. With this tool, users are able to parametrize rooms for a real-time acoustic-visual immersive simulation that is part of the VirKopf system [LSVA07]. The idea is to create a prototyping system for buildings that need to have special acoustic properties, for example concert halls, lecture rooms or the evaluation of alarming installations in factory halls.

To meet the requirements of VEs, the interface should be non-intrusive. Therefore, we want to avoid the use of WIMP-like graphical interface elements because virtual menus or buttons may occlude important parts of the scenery or are too cumbersome to be used in a virtual reality setting. Additionally, in CAVE-like environments, users are usually standing and can freely move around, while at the same time they have to carry input devices all the time. Thus, the devices used should be small, light-weight and non-intrusive. For example hands-free devices like the SmartWand [HKAK07] are an interesting option, as it provides two buttons for explicit interaction and six degrees of freedom (6DOF) tracking data.

A sketch-based interface seems to be a promising approach to account for the above mentioned requirements and was implemented in our prototype. By controlling a pen-like cursor, the user sketches lines into space or onto objects. These lines can either be used to create annotations or to sketch special command symbols that allow modifications of the scenery. Since sketching with pen and paper is a natural interaction – especially for architects – this offers an intuitive and simple interface approach. Sketches can be performed quickly with a single pen-like input device and do not require any graphical interface elements apart from the sketched lines. Thus, a sketch-based interface seems well-fit for the use in VEs.

The remainder of this paper is structured as follows. First, we will give a short overview over the related work that we based our work on for the development of the sketch-based interface. After that, we will describe the currently possible sketch drawing interactions within the VE with a special emphasis on the requirements of the gesture recognition module that is used to detect the meaning of the symbols drawn by the user. Finally, we will describe the possible interaction with scene objects that we currently offer to modify rooms and their interior within a VE.

2 Related Work

Sketching for desktop applications is an active research topic. Unlike static gesture recognition where the posture of a hand is used to control an interface [KAHF05], sketching is

dynamic and examines the movement of the input device.

Some approaches use sketches to trigger commands, for example SKETCH [ZHH96], while others interpret sketches to create 3D Models, like Teddy [IMT99] that allows to create roundish models from two-dimensional strokes only and inspired a lot of follow-up work. Specifically for the creation of architectural models, programs like [JL04] analyze blueprint-like sketches to create 3D sceneries. For Virtual Reality, sketching is often use to model free-form curves and surfaces [WS01] [FdAMS02] [BLMR02]. CavePainting [KFM+01] uses physical props like brushes and buckets to create artistic drawings.

For the recognition of 2D sketches and gestures, numerous different algorithms exist. While many of these are designed for one specific application, SketchREAD [AD04] and LADDER [HD05] provide domain-independent approaches. [EBSC99] present a recognition algorithm for sketching in VEs based on fuzzy logic. [BES00] extend this approach towards a more configurable architecture and include speech recognition and proposed possible applications.

3 Overview

Our prototype interface consists of three main modules: *Line Drawing*, *Gesture Recognition*, and *Scenery Modification*. The first module allows to sketch lines, both in space and onto surfaces. There are two modes for creating lines: The *Annotation Mode* creates drawings that can be used to add notes to the scenery. Lines sketched in *Command Mode* are interpreted in order to trigger commands which are predefined by the application. Each of these modes is initiated by applying explicit interaction, e.g. the press of two distinct buttons. Apart from these two buttons and the 6DOF positional data, no further input is required to control the interface. After drawing a sketch in command mode, the lines are handed to the gesture recognizer were they are compared with the set of available command symbols to find a best match. The symbol's corresponding command is then executed to create new objects or to modify existing ones, thus modifying the scenery.

4 Line Drawing

To draw freehand lines for annotation in free space, it suffices to simply accumulate the positional data from the input device and connect them virtually. On the contrary, drawing on the surface of objects is more challenging since VEs usually provide no haptic feedback. To compensate or this, a snap function is used to aid the user by moving the cursor onto the surface of an object when coming close enough. To find the closest surface point of an object, an AABB hierarchy collision structure of that object is used to determine candidate

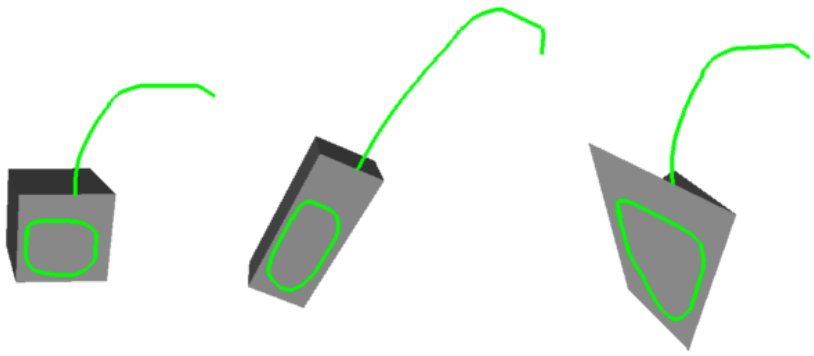

Figure 1: *Lines drawn on the surface of objects* (left) *adapt if the parent transforms* (center) *or deforms* (right)

faces, which are then examined with an exact distance test. If the nearest point that is found this way is close enough to the cursor, the pen is moved. As experimental studies have shown, the pointing accuracy in three dimensions is lower along the view direction [ZMR97] [FMRU03], for which reason an elliptic snap volume is used to determine if a point is close enough.

When drawing on a surface, the new line becomes attached to the object so that it follows transformations of the parent. Similarly, when the surface deforms, the line matches its shape (Fig. 1). By storing a line relative to it's parent, it automatically follows transformations of an object. To handle deformations, the line registers as an observer of the geometry. When getting notified of a deformation, the line recalculates surface points from their barycentric coordinates. It is also possible to connect two objects with a line so that transforming one of the objects changes the shape of the connecting stroke. In this case, the intermediate line is morphed to adjust changes of one of the end points.

5 Gesture Recognition

After sketching lines in command mode, they are handed to the gesture recognizer in order to find a corresponding command symbol. The recognition algorithm works in three steps. First, each input line is processed and corner points are determined in order to split the line into segments. These segments are then classified by comparing them to basic element shapes. Afterwards, the symbol matching tries do find the gesture hat describes the sketch best.

Our approach uses *Fuzzy Logic* and calculates a degree of truth for an object property. This value ranges between 0 and 1 and describes how well a property is fulfilled. The advantage of fuzzy logic is that it does not deliver a boolean value, but the degree of truth gives an

estimate on how similar they are. For example, if a sketch resembles two different symbols, the two values give a hint about which symbol matches better.

The set of symbols defines an application specific vocabulary, which is different for different application purposes. In order to have the tool applicable to different application domains and even different users of the same application, we modularized the definition of that vocabulary by providing a symbol definition language.

5.1 Symbol Definition Language

To allow the user to define her own command symbols without requiring access to source code and without the need for training sessions, a descriptive language has been implemented. Each symbol is stored in an external database as a human readable definition file which defines one or more gestures which describe the shape that is to be sketched in order to recognize the symbol. The use of multiple gestures per symbol allows to recognize variations of the desired shape, for example a set of different kinds of arrows for a move command.

Each gesture definition consists of *elements* and *constraints*. Elements describe the basic shape – like lines or circles – that make up the symbol. Constraints define the relations between elements to define their relative position, size, and orientation in the sketch.

5.2 Stroke Processing

To prepare lines for the recognition processing, they first are reduced and smoothed, and speed and curvature data is calculated for every point. This data is then used to detect corner points in the line. By using the algorithm described in [QWJ01], a set of corner point candidates is determined by looking for vertices with high local curvature and low drawing speed. This approach only uses local information – speed and curvature – to classify corner points, which can lead to bad detection results if the local data is similar, but the global line describes a different shape. To solve this problems, we extended the algorithm by using an additional, global criterion: the linearity of the line between two adjacent corner candidates. Now, corner points at linear segments, like the corners of a square, can be accepted, and candidates on curved lines like an ellipse are discarded (Fig. 2).

The found corners are used to split the line into *segments*, which are thereafter treated individually. This decomposes the input lines into several basic shapes, which are independent of the way the original lines were drawn. Thus, the basic segments can either be drawn in a single line or with several lines, which makes the recognition less dependent on the drawing style of a user.

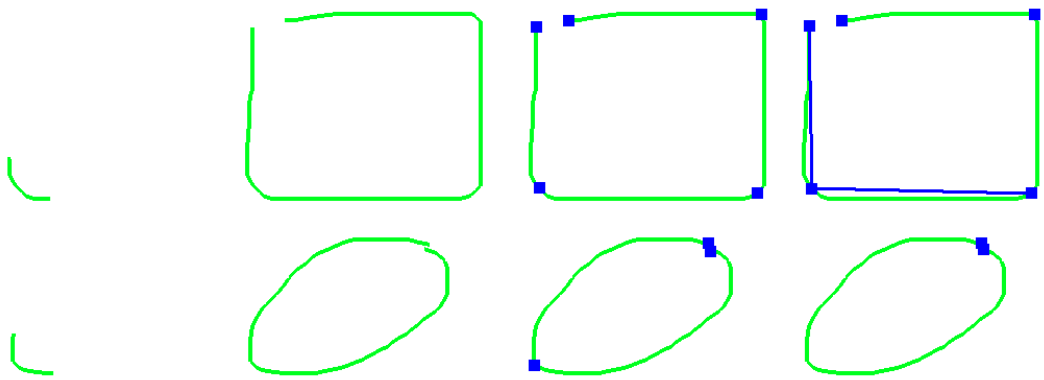

Figure 2: *When using only local information – speed and curvature – it may be impossible to distinguish between the corner of a square and the angular point of an ellipse. By using linearity as a global information, these cases can be distinguished.*

5.3 Primitive Recognition

After splitting the input lines, the resulting segments are classified by comparing them to a set of predefined primitives like lines, circles, or waves. The used algorithm is similar to the one described in [EBSC99]: For each of the input segments, a set of geometric properties is calculated, for example linearity, aspect ratios, or relative movement along axes. Each primitive type is defined by some of these properties and corresponding intervals that give the allowed values for these properties. To find the degree of matching of the segment to the element type, the fuzzy value of each property is calculated.

It is important that the classification works independently from the segment's orientation. While the screen defines a fixed coordinate system for 2D sketch recognition, the user can turn into an arbitrary direction when sketching in VEs. Therefore, an oriented bounding box [Got00] is calculated for each input segment. This box defines a local coordinate system and allows calculation of the geometric properties independent of the global orientation.

5.4 Symbol Matching

To find the best match between the sketch and the defined symbols, the classified segments are compared to the gesture definitions. A recursive algorithm is used to assign the input segments to the elements of the symbol definition. To ensure that the recognition works independent of the drawing order, all segment permutations have to be tested. By multiplying the degrees of truth of the elements and constraints of the symbol definition, a total degree of truth is gained.

At each step, the algorithm examines the next free element and tries each of the unused seg-

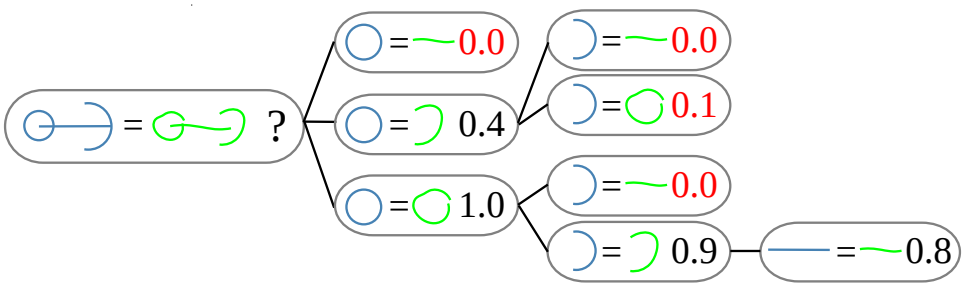

Figure 3: *An example for the permutation tree of the symbol matching step to compare the sketched segments (green, right) with a symbol definition (blue, left). At each node, the algorithm assigns one of the available segments to the next element of the symbol and updates the symbol matching degree (numbers). If this degree falls below a minimum, the following subtree is pruned (red numbers).*

ments. It then tests how well the segment matches the element type, and the corresponding degree to truth is multiplied with the symbol matching degree. Similarly, the constraints belonging to this element are checked. When reaching a leaf, all elements have been assigned a segment, and the degree of truth for this combination has been computed. After finding all leaves, the best one is chosen and its matching degree is used as the total truth value of the symbol.

This algorithm constructs a permutation tree of all segments, which becomes complex for larger symbols. Thus, it is important to prune subtrees as early as possible. After every change of the degree of truth, its value is compared to a cutoff threshold, and if it falls below, the calculation of the current subtree is aborted. The threshold starts at a predefined value, but is updated whenever a leaf node is reached. Figure 3 depicts the tree with their respective truth degree.

6 Scenery Modification

The previous sections described how sketches are created and analyzed to find a symbol. These symbols are now used to execute commands. For the visual-acoustic scenario, we inspected two classes of commands: *creation* and *modification* of objects. Their nature will be discussed in the following section.

6.1 Object Creation

When designing architectural sceneries, it is often useful to have a configurable set of objects available. This can be used to reflect personal preferences or to adapt the models to a

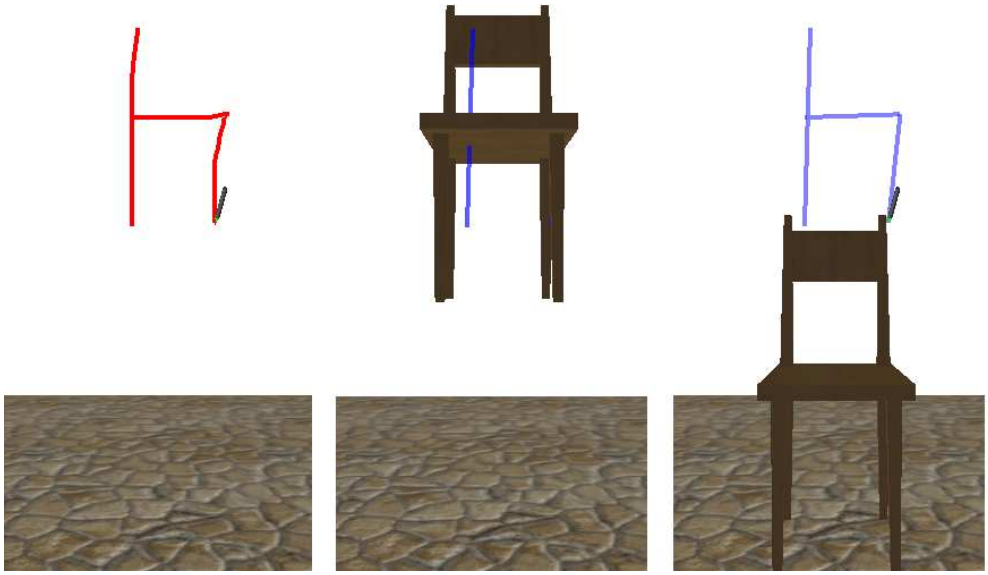

Figure 4: *After sketching the symbol for a chair* (left) *it is created at the position of the sketch* (center)*, but then drops down onto the floor* (right)*.*

specific domain – for example when designing an auditorium a different set of chairs would be used than for a living room. Therefore, the application allows to define *Object Classes* in an external database which link to a geometric model that can be created with desktop modeling applications.

It is also possible to link several models into one object class, e.g. to give a choice between several different types of chairs. The classes are given a unique name which is used to link them to symbol definitions. After sketching the related gesture, a new instance of the object class is created. Usually, the object is fit to the position, size, and orientation of the sketch, but these parameters can be adjusted, e.g. to always create the new object at a fixed size and upright. Additionally, automatic placement options can be specified, for example to define that a new chair automatically drops onto the floor (Fig. 4), or a window is placed on the wall behind the sketch.

If an object class contains more than one model type, it is necessary to choose one of the alternatives. For this purpose a circular selection menu appears and allows the user to preview the different models before picking one (Fig. 5).

6.2 Modification Commands

For the modification of interior, rooms and their wall material we predefined the following operations: *moving, scaling, rotating, switching type,* and *deleting.* Before applying a modification, a target object has to be determined explicitly. To do so, one can simply draw the command symbol onto an object, but this is difficult if the object is small or has a complex

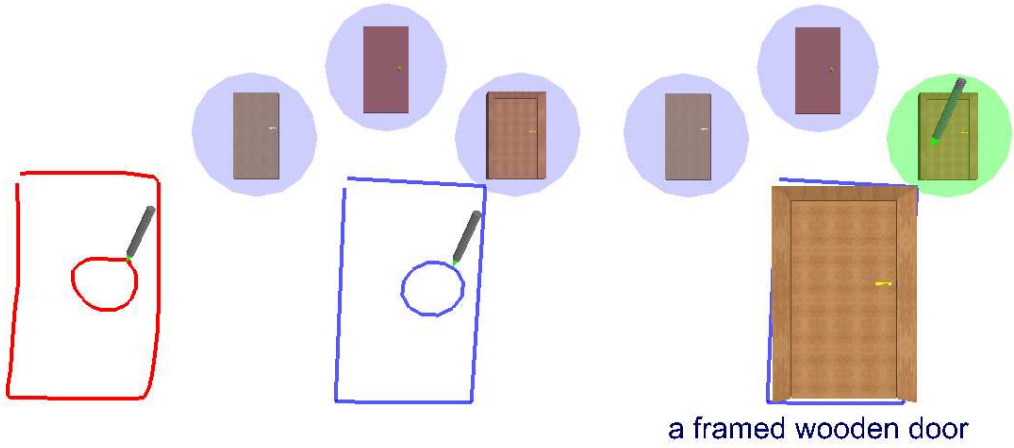

Figure 5: *After sketching the door symbol* (left) *a pop up menu appears* (center). *Here, the user can choose between different doors* (right). *All doors belong to a unique object class that is configured before application run.*

shape. Alternatively, one can explicitly select an object by clicking on or near it, what also allows to mark a set of targets for the modification. To select an object that is further away, one can move the cursor over it and press the button slightly longer. To highlight selected objects, they are surrounded by a blue frame and the selection's center point is shown.

To move an object, there are both *direct* and *modal* options. The direct modification determines the movement parameters from the sketch, for example by moving an object to the tip of an arrow or along the direction of its shaft. Alternatively, the application can switch into a mode where clicking and dragging the cursor moves the object. While the direct movement commands can be performed faster, they are not very precise, whereas the modal movement is slower, but allows to place the objects more accurately. Whenever the application switches to a special interaction mode, the cursor is changed to indicate the new mode. To scale or rotate an object, the application again switches to a special mode where the user can apply the modification with the cursor.

In these modes, visual *helpers* are displayed to aid the user (Fig. 6). For example, when scaling, a local coordinate system is displayed to indicate the direction for growing or shrinking the selection, and the object's original is shown as reference. Rotations for example are constrained, and can either be performed around the object center, an arbitrary point, or an axis depending on the sketched command.

As described earlier, there are object classes with several different models, so it may be useful to change an object's model type after it was created. After sketching the appropriate gesture, a selection menu as for the creation appears. Here, the user can change the object to another model type while keeping the original position and size. Finally, one can also draw an erasure symbol to remove selected objects from the scenery.

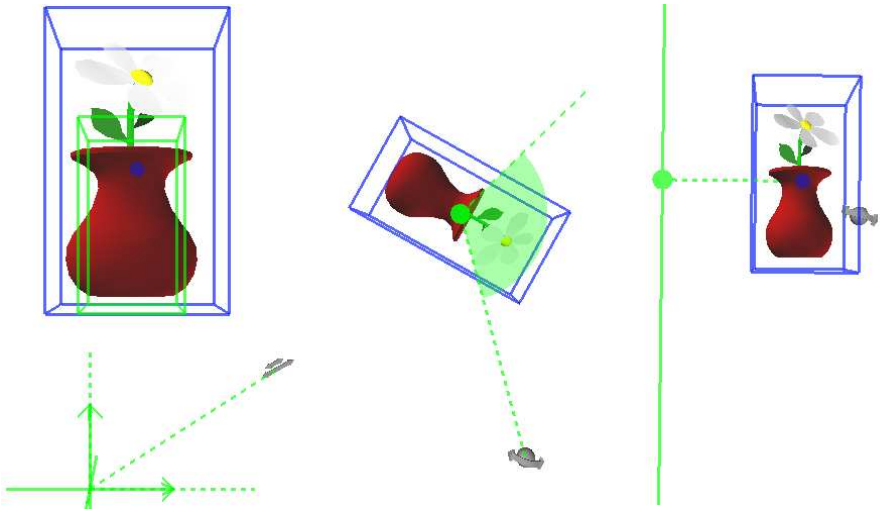

Figure 6: *Modification Modes: Scaling* (left), *Rotation around center* (center), *Rotation around axis* (right)

7 Conclusion and Future Work

The presented prototype provides a sketch-based interface which allows to annotate, modify and extend an architectural scenery inside a VE. Here, the small input device and absence of graphical interface elements helps to maintain the immersion. By using external databases, our prototype allows to define custom gesture symbols and object classes to adapt to different application domains or personal preferences. In the further course of this work, user studies will be performed to find the strengths and weaknesses of this interface approach. Additionally, the gesture recognizer still needs some optimizations, especially the corner point detection still causes errors. This is especially a problem when sketching in 3D without haptic feedback, as users tend to snatch the input device at the beginning or end of a stroke, which results in line segments that may confuse the gesture recognition.

In the future, several additions could be made to enhance the prototype. Currently, new objects like windows or walls are simply moved onto a wall, but it would be nice to cut a hole into the wall to create a real passage. Currently, the automatic placement for objects on other objects only works on bounding box level. This can produce strange results, for example, dropping a vase onto a table will align it with the plate, but when dropping it onto a chair, the vase would stop at the height of the back instead of moving further down onto the seat. Replacing this procedure with a fine grained collision detection algorithm will create better results. And finally, there could be specific interaction methods for different objects, for example to allow to open and close a door. The latter is very important for room design, as doors and windows act as portals that allow or forbid acoustic transmission to or from neighboring rooms.

References

[AD04] Christine Alvarado and Randall Davis, *SketchREAD: A Multi-Domain Sketch Recognition Engine*, UIST (Steven Feiner and James A. Landay, eds.), ACM, 2004, pp. 23–32.

[AHC+06] Ingo Assenmacher, Bernd Hentschel, Ni Cheng, Torsten Kuhlen, and Bischof Christian, *Interactive Data Annotation in Virtual Environments*, Proceedings of the Eurographics Symposium on Virtual Environments, Lissabon (Roger Hubbold and Ming Lin, eds.), ACM SIGGRAPH, 2006, pp. 119–126.

[BES00] Oliver Bimber, L. Miguel Encarnação, and André Stork, *A multi-layered architecture for sketch-based interaction within virtual environments*, Computers and Graphics **24** (2000), no. 6, pp. 851–867.

[BLMR02] F. Bruno, M. L. Luchi, M. Muzzupappa, and S. Rizzuti, *A Virtual Reality Desktop Configuration for Free-Form Surface Sketching.*, Proceedings of XIV Congreso Internacional de Ingeniera Grfica, 2002.

[EBSC99] L. M. Encarnacao, O. Bimber, D. Schmalstieg, and S.D. Chandler, *A Translucent Sketchpad for the Virtual Table Exploring Motion-based Gesture Recognition*, Computer Graphics Forum **18** (September 1999), pp. 277–286(11).

[FdAMS02] Michele Fiorentino, Raffaele de Amicis, Giuseppe Monno, and Andre Stork, *Spacedesign: A Mixed Reality Workspace for Aesthetic Industrial Design*, Proceedings of the International Symposium on Mixed and Augmented Reality, 2002.

[FMRU03] Michele Fiorentino, Giuseppe Monno, Pietro A. Renzulli, and Antonio E. Uva, *3D Pointing in Virtual Reality: Experimental Study*, XIII ADM - XV INGEGRAF International Conference on Tools and Methods Evolution in Engineering Design, 2003.

[Got00] Stefan Gottschalk, *Collision Queries using Oriented Bounding Boxes*, Ph.D. thesis, University of North Carolina at Chapel Hill, 2000.

[HD05] Tracy Hammond and Randall Davis, *LADDER, a Sketching Language for User Interface Developers*, Computers & Graphics **29** (2005), no. 4, pp. 518–532.

[HKAK07] Bernd Hentschel, Johannes Künne, Ingo Assenmacher, and Torsten Kuhlen, *Evaluation of a Hands-Free 3D Interaction Device for Virtual Environments,*, Workshop GI VR/AR, 2007.

[IMT99] Takeo Igarashi, Satoshi Matsuoka, and Hidehiko Tanaka, *Teddy: A Sketching Interface for 3D Freeform Design*, SIGGRAPH '99: Proceedings of the 26th

Annual Conference on Computer Graphics and Interactive Techniques (New York, NY, USA), ACM Press/Addison-Wesley Publishing Co., 1999, pp. 409–416.

[JL04] Roland Juchmes and Pierre Leclercq, *A Multi-Agent System for the Interpretation of Architectural Sketches*, Proceedings of the 2004 Eurographics Workshop on Sketch-Based Interfaces and Modeling (SBM-04) (Aire-la-Ville) (J. F. Hughes and J. A. Jorge, eds.), Eurographics Association, August 30–31 2004, pp. 53–62.

[KAHF05] Hyosun Kim, Georgia Albuquerque, Sven Havemann, and Dieter W. Fellner, *Tangible 3D: Hand gesture interaction for immersive 3D modeling*, 9th Int. Workshop on Immersive Projection Technology, 11th Eurographics Workshop on Virtual Enviroments (Aalborg, Denmark) (Erik Kjems and Roland Blach, eds.), Eurographics Association, 2005, pp. 191–199.

[KFM+01] Daniel F. Keefe, Daniel Acevedo Feliz, Tomer Moscovich, David H. Laidlaw, and Joseph J. LaViola Jr, *CavePainting: a Fully Immersive 3D Artistic Medium and Interactive Experience*, SI3D, 2001, pp. 85–93.

[LSVA07] Tobias Lentz, Dirk Schröder, Michael Vorländer, and Ingo Assenmacher, *Virtual Reality System with Integrated Sound Field Simulation and Reproduction*, EURASIP Journal on Applied Signal Processing (2007).

[QWJ01] Sheng-Feng Qin, David K. Wright, and Ivan N. Jordanov, *On-line Segmentation of Freehand Sketches by Knowledge-Based Nonlinear Thresholding Operations*, Pattern Recognition **34** (2001), pp. 1885–1893.

[WS01] Gerold Wesche and Hans-Peter Seidel, *FreeDrawer: A Free-Form Sketching System on the Responsive Workbench*, VRST '01: Proceedings of the ACM Symposium on Virtual Reality Software and Technology (New York, NY, USA), ACM, 2001, pp. 167–174.

[ZHH96] Robert C. Zeleznik, Kenneth P. Herndon, and John F. Hughes, *SKETCH: An Interface for Sketching 3D Scenes*, SIGGRAPH '96: Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques, 1996, pp. 163–170.

[ZMR97] Shumin Zhai, Paul Milgram, and Anu Rastogi, *Anisotropic Human Performance in Six Degree-of-Freedom Tracking: An Evaluation of 3D Display and Control Interfaces*, IEEE Transactions on Systems, Man, And Cybernetics - Part A: Systems and Humans **27** (1997), pp. 518–528.