Workshop on Virtual Reality Interaction and Physical Simulation VRIPHYS (2010)
J. Bender, K. Erleben, and M. Teschner (Editors)

3D Sketch Recognition for Interaction in Virtual Environments

D. Rausch[1] and I. Assenmacher[1] and T. Kuhlen[1]

[1]Virtual Reality Group, RWTH Aachen University, Germany

Abstract

We present a comprehensive 3D sketch recognition framework for interaction within Virtual Environments that allows to trigger commands by drawing symbols, which are recognized by a multi-level analysis. It proceeds in three steps: The segmentation partitions each input line into meaningful segments, which are then recognized as a primitive shape, and finally analyzed as a whole sketch by a symbol matching step. The whole framework is configurable over well-defined interfaces, utilizing a fuzzy logic algorithm for primitive shape learning and a textual description language to define compound symbols. It allows an individualized interaction approach that can be used without much training and provides a good balance between abstraction and intuition. We show the real-time applicability of our approach by performance measurements.

Categories and Subject Descriptors (according to ACM CCS): Computer Graphics [I.3.6]: Methodology and Techniques—Interaction Techniques Computer Graphics [I.3.7]: Three-Dimensional Graphics and Realism—Virtual Reality; Pattern Recognition [I.5.5]: Implementation—Interactive Systems;

1. Introduction

The development of suitable interfaces for immersive environments is an ongoing challenge. In contrast to 2D environments on desktop systems, no commonly accepted interface approach like the WIMP metaphor exists. Especially for constructive and scientific applications, a good mixture between intuitive, natural interaction and artificial guidance and controlling structures is required. A pen-based interface is a suitable approach for this, since a drawing can quickly communicate ideas, hold contextual information and instructions for the reader. For example, in an architectural context, it is quite common to sketch ideas and construction hints on plans or newly build walls.

This proposes the use of a *sketch-based interface* where the user can draw a *sketch*, which is then recognized and an associated command is executed. The command is supplied with additional information from the drawing (see figure 1), so that for example new objects can be created such that their the location, rotation, and scale matches the size, position, and orientation of the sketch. Since drawing is a natural and intuitive task, sketches allow for an easy operation, while their lack of precision complies to the inaccurate nature of initial design phases and review sessions. One major advantage is the non-intrusiveness of the approach, since the sketch-based interface is fully passive until the user starts to draw, and it only requires a small input device with a single button. This way, it is especially suited for applications where the user only performs commands now and then, while focusing on other tasks like exploration.

To use sketch-based interaction in a Virtual Environment (VE), a recognizer is required to interactively analyze and interprete the input drawings. For more complex sketches consiting of several strokes, this is a challenging task. We present a multi-level recognition approach for three-dimensional sketches that can reliably match moderately complex symbols in real-time, and enables a fluent and non-intrusive interaction.

The remainder of this paper is structured as follows. We will start with an overview of related work in the field. The general concepts and design requirements of the sketch recognition system are presented in section 3. Its three main components are then explained in more detail in the following sections. An enhanced stroke segmentation algorithm splits lines into meaningful segments (see section 4), which

DOI: 10.2312/PE/vriphys/vriphys10/115-124

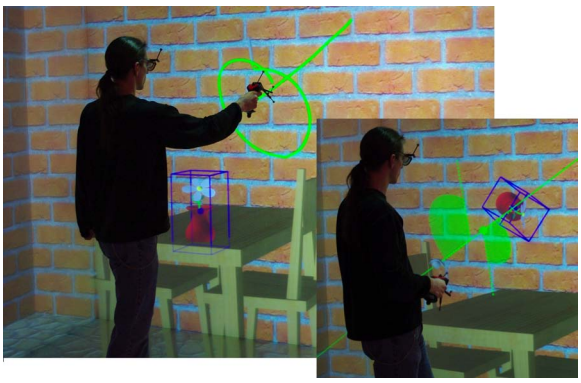

Figure 1: *The sketch-based interface allows to draw predefined symnols that are recognized and translated into commands. The example shows a sketched symbol (left) to rotate an object around the drawn axis (right).*

are then classified using a fuzzy logic-based primitive shape recognizer (section 5). After that, we describe an algorithm for the efficient matching of compound sketches, again using fuzzy logic (see section 6). A discussion of our findings, including a preliminary evaluation and performance analysis, concludes this paper.

2. Related Work

A commonly used method for interaction in a VE is the recognition of hand postures and gestures where the pose and movement of the user's hand is interpreted to interact with the system [LaV99]. While this aproach is very natural, the high number of degrees of freedom of the hand make a reliable recognition difficult. Furthermore, recording the state of the hand requires either a camera array and a computationally expensive image analysis, or worn hardware like an instrumented glove that poses ergonomic problems. Pen-based input can be seen as a simplified form of hand gestures, where instead of the whole hand only a single point – the tip of a finger or a pen – is examined.

For 2D interfaces, pen-based input methods have been used for some time now. An early representative is *SKETCH* [ZHH96] which uses simple drawn symbols to control a modeling application. Another popular sketching application is *Teddy* [IMT99], where the user can draw a silhouette to create roundish free-form objects, like stuffed animals. This application combines free-form sketch interpretation with a basic gestural interface, and thus only requires a minimal GUI for system control.

For simple pen gestures, *fuzzy logic* is a frequently used tool for fast and efficient recognition [Rub91, EBSC99, QWJ00,FPJ02], while other approaches like Hidden Markov Models are sometimes used for more specialized tasks [KEA03]. These can detect simple gestures – often consisting of only a single stroke – and allow for only a limited set of different shapes. A comparative evaluation of some pen gesture recognizers is presented by Schmieder et al. [SPB09].

Recently, algorithms for sketch recognition have been developed that can recognize not only single symbols but complex sketches consisting of many different symbols. These can for example be used to analyze hand-drawn electric circuit diagrams or relation diagrams in order to create rectified drawings of them. Due to the increased complexity of the recognition, artificial intelligence approaches like Bayesian Networks [AD04, SD07], Hidden Markov Models [SD04, SD05] or combinatorial models [HPW07] are used. However, while these complex sketch recognition systems are capable of parsing large drawings, they are usually not fast enough for real-time interaction.

Some Virtual Reality (VR) applications use 2D pen gestures to enrich their interfaces. However, recognition of truly three-dimensinal pen gestures or sketches is rarely used. Encarnação et al. [EBSC99] present a fuzzy logic-based recognizer for 3D single-stroke pen gestures. Using this recognizer, they developed a sketch recognition architecture for VEs [BES00] that can be combined with additional modalities like speech input. Compound sketches consist of several single-stroke gestures and are combined using a context free grammar. However, the grammar only specifies the number of occurring gestures, so that ambiguity between symbols with the same set of primitives still needs to be resolved. For this, it appears that hand-tailored decisions are used for a specific set of symbols, and is probably not easily extensible. Additionally, no segmentation is used so that each primitive has to be drawn with an own stroke. Another limitation arises from the gesture recognizer that uses a fixed reference frame for its computation. While this works for 2D sketches or environments where the user does not move, it fails as soon as the user can walk around and draw sketches in arbitrary directions. Thus, the presented applications of their framework use a handheld pad as drawing plane, effectively reducing the gestures to two dimensions.

3. Sketch Recognition Framework

The overall goal of the presented recognition system is to analyze input sketches of a user and translate them into a corresponding command. An overview over the general recognition process is shown in figure 2. As a basis of input, movement trajectories are given as *strokes* to the *recognizer*. A stroke can contain an arbitrary number of points collected by the tracking hardware, and recording is triggered application-side, for example by a button press. The recognizer then proceeds in several steps. First, the *segmentation* analyzes the input stroke to detect corner points, which are then used to split the stroke into smaller *segments*. Afterwards, the *primitive shape recognition* calculates a matching of primitives for each of the segments. Both stages are per-

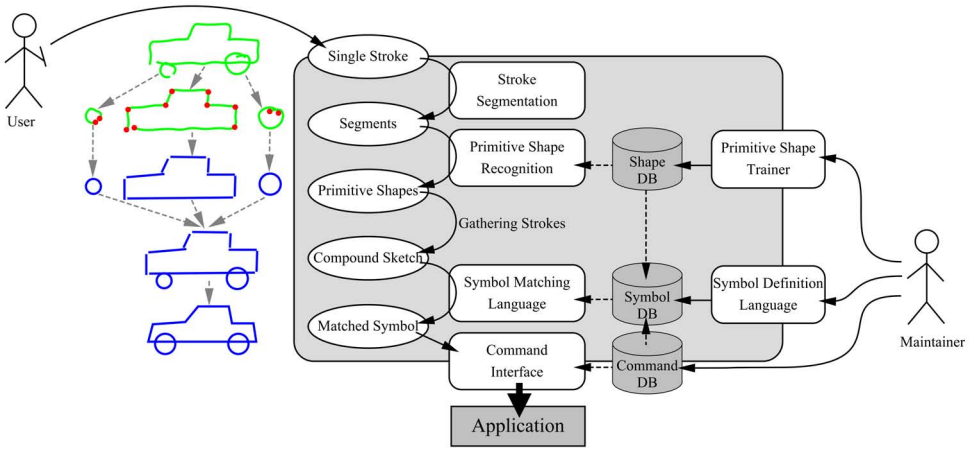

Figure 2: *Overview of the stages of the sketch recognition process. Continuous arrows represent the data flow through the system, while dashed arrows represent the use of additional resources. The gray area defines the boundary of the sketch recognition framework, and users and maintainers only access it over well-defined interfaces. The processing of an example sketch is shown on the left, indicated by dashed grey arrows.*

formed once a new stroke is added, and the results are stored together with earlier strokes, forming a *compound sketch*. This sketch is analyzed by the *symbol matching*, which compares it to predefined symbols in a database, trying to find a match. If the compound sketch matches a predefined symbol, the according command is called at the *command interface* and contextual information about the sketch is provided as parameter. The sketch recognition framework can be configured and adjusted to a specific domain of application using abstract interfaces, the *primitive shape trainer*, *symbol definition language*, and *command interface*.

3.1. 3D Sketching

At first, three-dimensional sketches seem to be inferior to two-dimensional ones. Since drawing is usually performed on paper and thus two-dimensional, 3D sketching is less natural and intuitive. Furthermore, both the visual sense and the physiology of the hand make sketching along the depth direction rather imprecise, which is made even worse by the lack of a supporting drawing plane. For these reasons, it is most common to use 2D sketch recognition not only for 2D interface, but also for applications in VR by either using a carried pad to draw on, or by projecting the sketch onto a 2D plane.

Despite the problems of unsupported 3D sketches, we feel that they have benefits encouraging their use for VR interfaces. First of all, one of the main problems, namely the lack of precision when drawing unsupported 3D sketches, can be accepted since a sketch is not meant to stay, but only triggers a command, and therefore the imprecision only matters for the recognizer and the extracted parameters. Additionally, 3D sketches only require a 3DOF or 6DOF input device, which is generally available in a VE and is easy to handle. This way, the user can freely move around in a VE, and does not have to wear bulky input hardware. In contrast, a supporting plane would either require a fixed position of the user, as in a fishtank VE, or a carried pad that he has to hold accordingly, requiring additional and uncomfortable hardware that either has to be carried around or requires a place to put down between uses. Furthermore, one of the main advantages of sketches is that they do not just trigger commands, but also provide additional parameters extracted from the drawing, like the position and size of a new object. These parameters are directly available in 3D, while the use of 2D sketches would require additional methods to resolve the missing degrees of freedom.

All in all, we feel that the use of 3D sketches to control an interface is a good option. While GUI-based interfaces are in general faster to use and do not require much learning, the sketch-based approach has the advantage of being fully passive when not being used, therefore maintaining the immersion. This makes this interface suitable for tasks that are not continuously performed, like architectural modifications during an exploration of the scenery.

3.2. Design Aspects

While designing the sketch recognition framework, we have found several important aspects that needed special consideration, resulting from the special requirements of a sketch-based interface. As a summary, we pose the following requirements.

- **3D Drawing.** The sketch interface is supposed to handle truly three-dimensional sketches that are drawn unsup-

ported and free-handed. Therefore, the recognition system needs to handle the additional degrees of freedom, and must be tolerant to increased drawing inaccuracies.

- **Compound Sketches.** In order to provide reliable recognition results, it is mandatory that symbols are sufficiently diverse. For pen gestures consisting of only a single drawn stroke, this obviously limits the number of available commands symbols. Therefore, we support *compound sketches* that describe more complex shapes assembled from several primitive elements. However, complex symbols usually take longer to be drawn by the user, making the interface slower. Additionally, despite the better iconography, complex symbols may be more difficult to memorize. Still, we feel that the advantages of compound sketches for multi-domain applicability justify these shortcomings, since more different symbols can be created and can better match a desired prototype iconography.
- **Customizability.** Depending on the domain of application, different sets of commands and their corresponding symbols are required. Clearly, the symbols to control an architectural construction session are different to those of a scientific visualization application, and even each user should be able to define her own set of symbols and semantics for every application. This appeals to the iconographic memory and associative thinking of individual users, but raises the need for suitable software interfaces. It is necessary to provide an easy way to customize the symbol database and the respective triggered commands. We developed interfaces that allow to specify the sketch recognition interface by defining *primitive shapes*, *sketch symbols*, and the *commands* to be executed.
- **Drawing Styles.** A problem that is also known from the recognition of hand gestures and speech is the variation between individual users. Since each person draws differently, the resulting sketches may be quite different for the same symbol. This problem is often overcome by training the system to each new user to adopt to her style. However, these training sessions are often bothersome when using the system for different users, and thus we instead tried to tailor the recognition process to be general and variable enough to handle the drawing styles of different users. To achieve this, we decided to use a variable symbol definition language, a stroke segmentation algorithm, and a primitive shape recognition based on global features only.
- **Real-Time Constraints.** An important aspect of VR interfaces is their real-time constraint. When triggering a command by sketching a symbol, the response should occur without noticeable delay, and therefore the recognizer has to provide its results fast enough.

3.3. Symbol Definition

In general, there are different approaches to create symbol definitions. One can represent them by a fixed set with hand-tailored parameters, but this is obviously hard to maintain and adapt to new use-cases. A more versatile approach is the use of training sets, where examples are provided to a learning system that then extracts a symbol representation. This is a suitable approach for simple gestures, but often fails for more complex ones. Additionally, the created definitions are often user-specific and thus require special training for each new person. A way to define more complex symbols is the use of a descriptive definition language that allows a maintainer to easily create new symbols. A prominent example is LADDER [HD05], which allows to create 2D symbols from basic shapes.

Our recognition framework detects low-level primitives and compound sketches in separate modules, so that it also uses individual strategies to define the objects. For the primitive shapes, we developed an optimization-based parameter learning algorithm described in section 5. While we expect the primitive shape trainer to be used seldom since the number of basic element types is limited, the definition of high-level compound symbols is performed more frequently. For this, we developed a textual definition language similar to LADDER, where symbols can be assembled from several *elements* and *constraints*. Elements define the primitive shapes in the symbol, like straight lines or circles, while constraints describe how these elements are assembled by restricting their relative size, position, or orientation. Typical examples for constraints are `PARALLEL`, `SAME_LENGTH` or `MEET`, and each constraint is prefixed with a qualifier – `SLIGHTLY`, `MOSTLY`, or `VERY` – that specifies how strongly it has to be fulfilled, allowing for a finer control over the symbol's final shape. Furthermore, constraints can be inverted or marked as optional. By allowing optional elements and different variants for a symbol, it can be defined to be drawable in different ways, which helps to compensate for different drawing styles of users.

In addition to the symbol definition, means to execute a command after a successful recognition are needed. For this, a software interface is defined, which passes a recognized sketch as well as parameters extracted from the drawing. The commands are free to interpret the properties for execution either globally on sketch level, or locally on the level of individual segments. For example, a command can use the orientation of the sketch as a whole, or the orientation of a single element in the sketch as a rotation axis.

3.4. Fuzzy Logic

We decided to use *fuzzy logic* for both the primitive shape and symbol recognition steps. Fuzzy logic is an extension of standard logic that allows to handle uncertainty. In traditional boolean logic, expressions are either true or false, while fuzzy logic has a continuous range of truth values that may lie anywhere between zero (false) and one (true). This helps to deal with the imprecise nature of gestures and sketches. *Fuzzy sets* define a sliding *degree of membership*

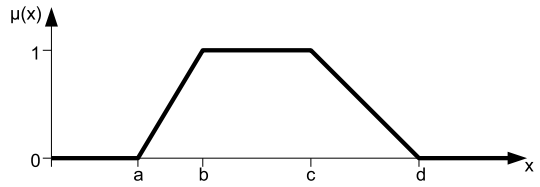

Figure 3: *An example for a piecewise linear membership function $\mu(x)$ described by a fuzzy range $\pi = [a,b,c,d]$.*

(DoM) in the range between zero and one, which gives a definition of how strongly a value belongs to the set. The DoM of a value x for a given fuzzy set can be calculated by a *membership function* $\mu(x)$ that maps each input parameter to a value between zero and one. A common approach is the use of a piecewise linear function forming a trapezoid, which we define by a *fuzzy range* using four values $\pi = [a,b,c,d]$ (see figure 3). This function can be stored efficiently, and its easy evaluation is beneficial under real-time constraints.

4. Stroke Segmentation

When assembling a complex sketch from several primitive shapes, it is necessary to find parts of the sketch that correspond to individual primitives. Some sketch recognition approaches simply assume that the whole sketch consists of only one stroke, or that each basic element is drawn with an individual line. However, this strongly limits the drawing styles and comfort, as users tend to create several primitive shapes in a single stroke. As a consequence, we feel that it is necessary to segment each input stroke before performing the actual recognition.

The segmentation detects *feature points* that partition a stroke into meaningful parts, and uses them to split the stroke into *segments*. For example, it is then possible to draw a square with one stroke forming all four sides at once, or with multiple strokes each containing one or more segments. This gives the user individual freedom when drawing symbols, and makes the whole recognition less susceptible to variation in drawing styles. However, this does not allow feature accentuation, for example dashing or over-sketching, as supported by [FPJ02]. For sketch-based interaction, this is not necessary because the drawing are supposed to execute commands instead of modeling visual effects.

The segmentation step is preceded by smoothing the input point set with a simple averaging filter to reduce the effects of noise. Additionally, information like curvature, speed, and arc length is computed for each point. Then, the corner detection finds features in the processed input. As an initial step, we use a corner detection algorithm based on speed and curvature information, similar to existing approaches [QWJ00, FMRU03, SD06]. Intuitively, corners have a high curvature, and speed serves as an additional

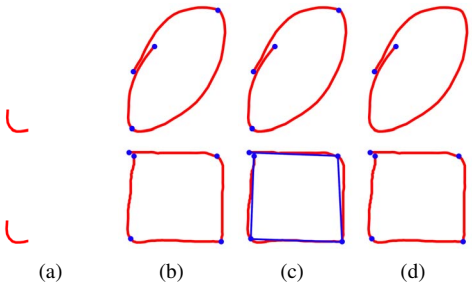

Figure 4: *When using only local information* (a) *it is hard to decide if a corner point should be set. Despite differences in the global shape, either both or none of the corners can be found (b). By using linearity as global information about the line (c), a reliable feature point detection can be achieved (d).*

measure since humans usually draw corners more carefully and thus slower. For each point $\mathbf{p}_i$ in the line, a quality measure $q(\mathbf{p}_i)$ is calculated from its curvature $c(\mathbf{p}_i)$ and speed $s(\mathbf{p}_i)$:

$$q(\mathbf{p}_i) = \frac{c(\mathbf{p}_i) - m_c}{\bar{c}} + i_s \cdot \frac{m_s - s(\mathbf{p}_i)}{\bar{s}}$$

where $\bar{c}$ and $\bar{s}$ are the average values of speed and curvature over the line, i_s is the relative influence of speed onto the result, and m_c and m_s are threshold values for curvature and speed. If the quality measure of a point is above zero and also has the highest quality in its local neighborhood, it can be accepted as a corner point.

However, this basic approach turned out to be insufficient. Only local information is used, but the decision to accept a corner point might depend on the global shape of a curve (see figure 4). *Linearity* of the final segments provides such global information, which we use to improve the quality of the segmentation. Intuitively, a corner is more likely to appear at the end of a straight segment than on a curved one, so that we prefer feature points that are neighbored by linear segments.

Thus, we enhanced the corner point detection to utilize the linearity. Initially, a set of candidate points $\{\mathbf{c}_i\}$ is computed based on speed and curvature values, where the parameters $i_s = 0.1$, $m_s = 0.8\bar{s}$, and $m_c = 33°$ are empirically chosen. Each pair of adjacent candidates $\mathbf{c}_i$ and $\mathbf{c}_{i+1}$ enclose a line segment l_i, for which the linearity is computed from the arc length $arc(l_i)$ and the endpoint distance by $lin(l_i) = \frac{arc(l_i)}{|\mathbf{c}_{i+1} - \mathbf{c}_i|}$. A straight line evaluates to $lin(l_i) = 1$, while larger values propose a curved shape. Since we want to prefer candidates on linear segments, we calculate a bonus factor $b_i = 1 - \frac{lin(l_i)-1}{max\_lin-1}$ for the segment if $lin(l_i)$ is smaller than a maximal value max_lin, otherwise b_i is set to zero. We empirically set $max\_lin = 1.3$ for good results.

Based on these linearity values, an artificial curvature c_a is calculated for each candidate point:

$$c_a(\mathbf{c}_i) = c(\mathbf{c}_i) + (b_{i-1} + b_i) \cdot q_1 + b_{i-1} \cdot b_i \cdot q_2$$

Here, $q_1 = 30^\circ$ and $q_2 = 70^\circ$ are bonus curvature values that are applied if the line segments l_i and l_{i+1} adjacent to the candidate $\mathbf{c}_i$ are linear. There are two different curvature bonuses, a smaller one that is given based the individual linearity of each neighbor, and a larger one that only applies if both adjacent segments are linear.

The artificial curvature values are then used to re-compute the quality measure $q(\mathbf{c}_i)$ of each candidate point, but this time using a higher curvature threshold $m_c = 90^\circ$. This way, only points that initially had a high enough curvature or that received a sufficient linearity bonuses are kept, forming the new set of feature points. If a candidate point is discarded this way, the linearity bonus of adjacent candidates no longer depends on the discarded point and has to be reevaluated. Therefore, the corner point detection is repeated with the reduced set of candidate points until no more points are discarded. The final set is accepted as corner points and is used to split the input lines into segments.

The extension of the approach significantly increases the reliability of the segmentation over the speed-and-curvature-based version. Points on curves that only show a high local curvature are reliably discarded, while sharp breaks between two curved segments are still detected.

5. Primitive Shape Recognition

Once meaningful segments are identified, they are classified by the primitive shape recognition step that compares them to predefined shape prototypes. A set of DoMs is computed for each segment, storing the matching result for every shape. Thus, a segment is matched to different primitive shapes simultaneous, allowing it for example to count as both a circle and an ellipse.

5.1. Shape Classification Features

To differentiate between shapes, it is necessary to use specific classification features. A main design goal of the recognizer is to allow usage by different users without training it individually. For this reason, we only use parameters that are universal enough to describe the general form of the shapes, and not the way they are drawn. Therefore, we observe global geometric properties of the stroke, for example the start- and endpoint, center of gravity, or bounding box.

In an immersive VE it is necessary to determine a suitable reference frame to calculate the global properties. Using the same fixed reference frame, like the world coordinate system, for all segments does not work because transformations of a stroke relative to the global coordinate system result in different geometric properties. For example, two identical shapes rotated by 90° to each other would be recognized as different. It is thus necessary to define a frame of reference for each segment that is independent from its global transformation. For this, we compute the Oriented Bounding Box (OBB) [Got00] of each segment, which forms the local reference frame for the calculation of the geometric primitives. OBBs are well-suited for this since they align to the drawn segments and are independent of the global transformation, and thus enable the recognition to work in 3D environments.

Still, some of the properties describe absolute values that are not directly usable, like the x-extent of the OBB. To create relative *features*, we combine different properties, for example instead of using the absolute x-extent, we divide it by the OBB's diagonal extent to receive a ratio. A total of 24 features is used for the classification:

- *Size* of the three axes of the OBB, normalized by its diagonal
- *Size ratios* measuring the OBB's x-y, x-z, and y-z ratios
- *Arc length* of the segment, normalized by the OBB diagonal
- *Relative movement* along the OBB axes
- *Endpoint distance* normalized by the OBB diagonal
- *x-, y-, and z-coordinates* of the startpoint, endpoint, and center of gravity, relative to the corresponding OBB axis
- *Distance* of the startpoint, endpoint, and center of gravity to the OBB center, normalized by the OBB's diagonal
- *Average distance from center of gravity*, normalized by the diagonal size of the OBB

For an input stroke, these values are stored in a *feature vector* $\mathbf{x}$ of size 24, which can then be used to classify the stroke.

5.2. Shape Recognition

Given the feature vector $\mathbf{x}$ of an input segment, we want to classify this stroke by comparing it to a primitive shape type. For this, a shape is defined by its property definition $\mathbf{p} = (\pi_i, r_i)_{i \in \{1,\ldots,24\}}$, where for each feature i, $\pi_i = [a_i, b_i, c_i, d_i]$ is the *fuzzy range* that defines the membership function $\mu_i(x)$ of the corresponding fuzzy set, and the *relevance value* $r_i \in [0,1]$ is used to determine how strongly the specific parameter influences the recognition result. This relevance value allows a modeling of dominant features for sketch classification, so that only those properties that actually help in distinguishing a shape influence the recognition result. Since five values are stored for each feature, $\mathbf{p}$ is of dimension 120.

For the input features $\mathbf{x}$, we can now calculate the DoM $f_{\mathbf{p}}(\mathbf{x})$ that describes how well the segment resembles a primitive shape defined by $\mathbf{p}$. A local matching degree for each feature $\mathbf{x}_i$ is calculated from $\mu_i(\mathbf{x}_i)$, and weighted by r_i. These local degrees are multiplied to find the total DoM $f_{\mathbf{p}}$.

$$f_{\mathbf{p}}(\mathbf{x}) = \prod_{i=1}^{24} 1 - r_i \cdot (1 - \mu_i(\mathbf{x}_i))$$

5.3. Shape Learning

While it is now possible to recognize a segment, we still need a way to create primitive shape definitions. We started off using a fixed set of basic shapes, where ranges and relevance values were determined manually, but this did not allow to individualize the system further. Thus, we introduce an approach to define new shapes using an example-based learning algorithm.

The system learns new shapes by providing it several example strokes, from which it derives a suitable description automatically. The set of examples S includes both positives and negatives, i.e., strokes that should be recognized as the desired shape and those that should not. Partial matches can also be given to the system by specifying a numeric DoM, for example to describe that an elliptic shape is somewhat similar to a circle, but does not fully match.

An *example* $s \in S$ consists of its geometric features $\mathbf{x}_s$ and its desired DoM d_s. Initially, a rough estimate for the shape is created by setting each fuzzy set to include all parameters of the positive examples, and each relevance value is set to 0.5. An iterative optimization algorithm then refines the values of the shape description by optimizing an error function $\phi(\mathbf{p})$ that measures the agreement of the current shape with the set of examples.

$$\phi(\mathbf{p}) = \sum_{s \in S} \left(d_s - f_{\mathbf{p}}(\mathbf{x}_s)\right)^2$$

This function is minimized using the method of steepest descent. For this, the gradient of the error function $\nabla\phi(\mathbf{p}) = \left(\frac{\partial\phi(\mathbf{p})}{\partial\mathbf{p}_1}, \ldots, \frac{\partial\phi(\mathbf{p})}{\partial\mathbf{p}_{120}}\right)$ is needed. We calculate $\nabla\phi(\mathbf{p})$ using a finite difference to approximate the derivatives $\frac{\partial\phi(\mathbf{p})}{\partial\mathbf{p}_i} \approx \frac{\phi(\mathbf{p}+\varepsilon\cdot\mathbf{e}_i)-\phi(\mathbf{p})}{\varepsilon}$ with a small offset ε along the unit vector e_i. Once $\nabla\phi(\mathbf{p})$ is calculated, it is used to update the shape definition to create a better approximation.

$$\mathbf{p} \leftarrow \mathbf{p} - \alpha \cdot \nabla\phi(\mathbf{p})$$

The step width $\alpha \in (0,1]$ is determined by linear search along the gradient's direction. During iterations, $\phi(\mathbf{p})$ converges toward a (local) minimum that defines the properties of the shape description.

However, a simple application of this approach does not succeed since the used membership functions $\mu_i(x)$ have large areas with a derivative of zero. To overcome this problem, we use a modified membership function for the optimization, where an exponential fall-off on the edges and small linear slopes in the center ensure that its derivative is always non-zero. Additionally, the optimization may produce invalid shape description by creating a relevance value outside of $[0,1]$ or a fuzzy range that does not conform the condition $a < b < c < d$. Therefore, after every update the validity of the shape description is checked and, if necessary, reestablished by clamping to legal values.

A small set of examples – normally about 10 – is usually sufficient to calculate a shape description. The optimizer automatically determines important features for classification, which are usually only a few (2 to 8), while most other relevance values are zero. In most cases, the presented learning algorithm manages to find a good result. However, the method of steepest descent only converges to a local minimum, so that it may happen that no good approximation can be found. In our experience, this only occurs rarely, and in these cases it is usually sufficient to just provide more examples. Global optimization algorithms, like simulated annealing, could be used instead to find a globally optimal solution at the expense of a higher computation time.

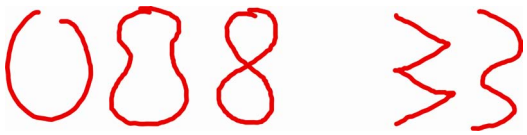

Figure 5: *The shape recognizer can distinguish even similar shapes like an ellipse, an eight, and the body of a violin* (left). *However, problems occur if the differences are only visible in local features, as for a zig-zag and a wave* (right).

With the shape learning and the classification features as presented in section 5.1, a variety of shapes can be separated. The depicted approach can reliably distinguish even rather similar gestures, like an ellipse, eight, and violin body. However, since only the global geometry of a stroke is examined, difficulties arise when shapes only vary by local features such as corners (see figure 5). The preceding segmentation step splits input lines with local feature points into several segments, which in turn can be represented by a compound symbol, so this does not pose a problem.

6. Symbol Matching

When a new stoke has been processed, it is added to a compound sketch. Now, the symbol matching performs a high-level recognition and computes a vector of DoMs that describes the similarity of the sketch with all predefined symbols in the database.

6.1. Matching Algorithm

The actual matching algorithm is performed separately for each symbol in the database and computes a DoM for a set of classified segments. Initially, it tests if the number of segments is valid, and evaluates global constraints based on the OBB enclosing the sketch. This allows to reject unsuited sketches quickly without much computational effort.

Afterwards, the algorithm assigns the segments of the sketch to the symbol elements and checks the matching. However, this assignment is not trivial since the recognition should be successful independent of different drawing

```
function CalculateSymbolDoM( segments )
  if invalid number of segments return 0;
  dom = 1;
  for each global constraint c
    dom *= EvaluateConstraint( c );
  if dom < prune_threshold return 0;
  MatchingRecursion( 0, segments, dom );
  return LoadBestConfiguration();
end

function MatchingRecursion( recursion_depth,
                            available_segments, dom )
  if( available_segments == {} ) do
    prune_threshold = dom;
    StoreBestConfiguration();
  end
  element = GetElement( recursion_depth );
  for each segment s in available_segments do
    local_dom = dom;
    element.AssignSegment( s );
    local_dom *= EvaluateElement( element );
    for each constraint c of element
      local_dom *= EvaluateConstraint( c );
    if( local_dom > prune_threshold )
      MatchingRecursion( recursion_depth + 1,
              available_segments \ {s},
              local_dom );
  end
end
```

Figure 6: *Pseudocode of the symbol matching algorithm.*

styles and temporal drawing orders. To achieve this, all possible permutations of segments to elements have to be tested. For this, we propose a recursive algorithm that is depicted as pseudocode in figure 6. The symbol's DoM starts with a value of 1 and is updated whenever an element or constraint is evaluated, as described in section 6.2, by multiplying it with the local evaluation result. At each call of the recursion, the algorithm examines the first unused element of the symbol, and assigns and tests each of the available sketch segments. It thus processes one element after the other. The recursion ends when all elements have an assigned segment, as a valid configuration is found. Its matching quality is described by the local DoM.

This recursive algorithm constructs a permutation tree of the possible segment combination, which may become very large. For performance reasons, sub-trees are pruned (see figure 7) as soon as their local DoM drops below a threshold value. For this, we use a dynamic cutoff threshold that is updated whenever the algorithm encounters a leaf node, since now configurations with a lower DoM than the one of the leaf can be discarded. This pruning significantly reduced the share of the tree that actually needs to be evaluated.

6.2. Evaluation of Elements and Constraints

After each assignment of a segment to an element, the match of the new configuration has to be checked. This requires the evaluation of the corresponding constraints and element type. As the DoM for each primitive is known from the shape recognition step (see section 5), it can simply be looked up.

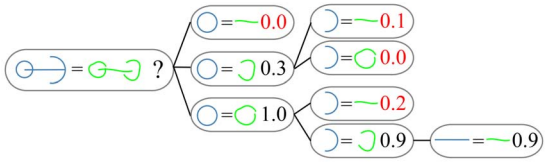

Figure 7: *To compare a set of sketched segments (right) with a symbol definition (left), a permutation tree is assembled. At each node, the algorithm assigns one of the available segments to the next element of the symbol and updates the current DoM (numbers). If this value falls below a threshold, the following sub-tree is pruned (dashed nodes).*

Depending on the constraint type, a specific geometric parameter is used to calculate the matching degree of a constraint, e.g., the angle between two segments. Each constraint has a predefined membership function that depends on its type and strength and allows to calculate the DoM from the respective parameter.

6.3. Sub-Sketch Recognition

In an early version of the framework, the recognition process always examined a sketch as a whole so that a symbol had to be formed by all contained segments. This is bothersome in many cases, as a sketch has to be empty when a user starts drawing. We expanded the algorithm to perform sub-sketch recognition, which inspects subsets of the drawn strokes.

Since one can assume that all segments of a symbol are drawn in succession and that the last stroke is contained – since otherwise it would have been recognized before already – potential sub-sketches consists of all segments from a varying start point until the last added stroke. Each of these sets of segments is individually matched by the recognizer, yielding a separate matching degree. In cases where more than one sub-sketch matches a symbol, accepting the most complete sub-sketch is a good heuristic, but this choice is generally application-dependent.

7. Results

The presented framework was implemented in C++ using the ViSTA VR Toolkit [AK08] and was utilized for a use in a CAVE-like environment. To draw sketches, a 6DOF input device was used, which provided position and orientation data at 60Hz using ART optical tracking.

7.1. Preliminary Evaluation

Using the sketch recognition framework, we implemented an interface for architectural modifications [RA08]. It provides a set of 28 example commands and corresponding symbols for the modification of an architectural scenery, including the translation, rotation, or automatic positioning of existing

objects or the creation of new elements like doors or windows. Using this application, we performed a preliminary user study. The uses were given an introduction to the interface and then had to perform specific tasks. Examples of the required gestures were shown to the user initially, but no further hints about correct drawings or – in case of misrecognitions – of the cause of errors were given. The study was performed with eight subjects, of which four were architects or students of architecture.

All sketches drawn by the users were recorded to determine the quality of the sketch recognition system. In total, 782 sketches were drawn during the user studies, of which 74.8% were recognized correctly. While this rate is rather low, it includes several recognition failures that are not due to errors of the recognizer. 9.6% of the sketches were not recognized because the drawn sketch was too different from the intended symbol, usually because the user was not experienced enough with the system and drew a wrong symbol. With more training, the users got better when drawing the symbols, and recognition rates got better. Another frequent cause for recognition failures were deficient symbol definitions, causing 10.1% of the sketches to not be recognized. Here, it was possible to distinguished symbols that were usually drawn correctly, and those that frequently caused problems. This shows that the symbols in the database have to be designed carefully and adjusted to different drawing styles of different users. Especially some user-specific variations in the symbol's drawing styles were not included in the definition, and led to repeated errors.

The remaining 5.5% of the errors were caused by the recognizer. Here, a main error source was the corner point detection, which caused 2.9% of the sketches to not be recognized. During the study, we still used a simpler version of the corner point detection, which we enhanced as a result of these finding, leading to the method presented in section 4.

In general, the evaluation showed that the sketch-based interface approach is suitable for interaction in 3D, even for novice users. When suitable feedback about recognition success or failure is provided, one quickly learns to interact with the system using the available symbols. After a short learning phase a reliable recognition can be achieved, allowing for a fluent interaction.

7.2. Performance Benchmarks

A variety of factors can influence the running time of the recognizer, most notably the symbols to be recognized. We have run performance benchmarks for characteristic cases in order to show the real-time capability of the algorithm. Stroke segmentation and primitive shape recognition are performed only once for each input stroke, and their complexity is linear in the number of points of the stroke and the number of segments, respectively. The symbol matching works on the whole sketch and has a general complexity of $\mathcal{O}(n!)$ in the number of segments, but the introduced pruning strategy enables interactive recognition rates for a reasonable amount of segments. We tested the performance for two different scenarios: normal application and extreme cases. We used a mobile PC with a Core 2 Duo 2GHz and 2GB RAM running on Windows XP to test at vocabulary of 10 primitive shapes and 28 symbols as they are used in an actual application. The recognizer is able to classify most sketches in less than $1ms$ for this set-up, while complex sketches consisting of up to 10 segments require at most $2.5ms$. Since the segmentation and shape recognition only depend on the size of the input line, their computation time is relatively constant and ranges from 50 to $200\mu s$ each. The symbol matching step varies more strongly, as the run-time largely depends on the number of segments, and ranges from 50 to $2000\mu s$.

	Underconstrained			Constrained		
Elements	8	12	16	8	12	16
Avg. (ms)	2.2	4.2	764	1.0	3.2	43.4
Dev. (ms)	0.6	4.3	1638	0.3	2.1	43.7
Max. (ms)	3.2	16.0	5420	1.4	5.9	99.3

Table 1: *Results of the run-time measurement for test symbols with varying number of elements of the same type. The symbols have a normal (constrained) or minimal (underconstraind) amount of constraints.*

The main performance bottleneck is the symbol matching step. While the number of elements in a symbol determine its general complexity, element types and constraints also affect the efficiency of pruning. Symbols that contain many elements of the same type have a high computation time since the type cannot be used for pruning, and thus a larger portion of permutations has to be tested. Thus, we constructed artificial symbols consisting of 8, 12, and 16 straight line elements. To test a normal case, those lines were arranged to a complex symbol using several constraints. Additionally, an extremal case was model with a minimal number of constraints so that a large part of the permutation tree has to be constructed. We randomized the order of the segments in the sketch to prevent influences from drawing regularities.

The results of the performance tests are shown in Table 1. As expected, the computation times increases rapidly when more elements are added. In the worst case, the symbol matching can take several seconds, but for smaller symbols interactive recognition rates are achieved. The results also show that for symbols with a reasonable amount of constraints, the pruning works a lot better and allows to use a sufficient number of elements. To ensure a fast recognition, the amount of elements in a symbol should not exceed a value of about $12-14$, especially if the elements are of the same type. While this limits the amount of possible symbols for an application, it is more of a theoretical margin because symbols of such complexity are hard to memorize by the user, and take too long to draw for quick interaction.

7.3. Summary

Sketch-based interaction is a useful approach for 3D interaction as it is intuitive and abstract. It avoids cumbersome input devices or cluttering of the display. It only uses hand movement trajectories with a decent sampling rate and one explicit trigger to start sketching. The metaphor is non-intrusive and thus can be used in fully immersive VEs and in combination with other metaphors.

In this paper we described the structure and algorithms of a 3D sketching framework and reflected on important aspects we encountered. By using an initial segmentation step, input lines are divided into segments that each describe a primitive shape, thereby providing more freedom when drawing. Since current feature point detection algorithms were not able to achieve sufficient results, we developed an approach using linearity as global information, and thereby could significantly improve the recognition results. For the recognition of primitive shapes, we utilized oriented bounding boxes to allow an orientation-independent recognition of 3D gestures using a fuzzy logic approach. We propose an automatic learning system that determines suitable recognition parameters for a new shape from examples. Based on these shapes, a user-definable and application specific set of complex 3D symbols can be created using a description language. All in all, this system provides a customizable, easy-to-use interaction approach that performs recognition of moderately complex sketches in real-time.

In the future, the recognition system can be enhanced by further improving the stroke segmentation, which still is the main source of misrecognitions. Additionally, global optimization strategies, like simulated annealing or genetic algorithms, can be utilized to enhance the primitive shape learner. Furthermore, we consider performing a user study to compare the benefits of truly three-dimensional sketches over 2D gestures drawn onto a supporting surface.

Acknowledgement. We would like to thank the German Research Foundation (DFG) for funding of this research in context of the project Ku1132/5-Vo600/3.

References

[AD04] ALVARADO C., DAVIS R.: SketchREAD: A Multi-Domain Sketch Recognition Engine. In *UIST: Proceedings of the 17th Annual ACM Symposium on User Interface Software and Technology* (2004), pp. 23–32. 2

[AK08] ASSENMACHER I., KUHLEN T.: The ViSTA Virtual Reality Toolkit. In *SEARIS Workshop on IEEE VR 2008* (2008). 8

[BES00] BIMBER O., ENCARNAÇÃO L. M., STORK A.: A Multi-Layered Architecture for Sketch-Based Interaction within Virtual Environments. *Computers and Graphics 24*, 6 (2000), 851–867. 2

[EBSC99] ENCARNAÇÃO L. M., BIMBER O., SCHMALSTIEG D., CHANDLER S.: A Translucent Sketchpad for the Virtual Table Exploring Motion-based Gesture Recognition. *Computer Graphics Forum 18* (1999), 277–286(11). 2

[FMRU03] FIORENTINO M., MONNO G., RENZULLI P. A., UVA A. E.: 3D Sketch Stroke Segmentation and Fitting in Virtual Reality. In *International Conference on the Computer Graphics and Vision* (2003). 5

[FPJ02] FONSECA M. J., PIMENTEL C., JORGE J. A.: CALI: An Online Scribble Recognizer for Calligraphic Interfaces. In *Sketch Understanding, Papers from the 2002 AAAI Spring Symposium* (2002), pp. 51–58. 2, 5

[Got00] GOTTSCHALK S.: *Collision Queries using Oriented Bounding Boxes*. PhD thesis, University of North Carolina at Chapel Hill, 2000. 6

[HD05] HAMMOND T., DAVIS R.: LADDER, a Sketching Language for User Interface Developers. *Computers & Graphics 29* (2005), 518–532. 4

[HPW07] HALL A., POMM C., WIDMAYER P.: A Combinatorial Approach to Multi-Domain Sketch Recognition. In *Proceedings of the 4th Eurographics Workshop on Sketch-Based Interfaces and Modeling (SBIM07)* (2007), pp. 7–14. 2

[IMT99] IGARASHI T., MATSUOKA S., TANAKA H.: Teddy: A Sketching Interface for 3D Freeform Design. In *SIGGRAPH '99: Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques* (1999), pp. 409–416. 2

[KEA03] KESKIN C., ERKAN A., AKARUN L.: Real Time Hand Tracking and 3D Gesture Recognition for Interactive Interfaces using HMM. *ICANN/ICONIPP* (2003), 26–29. 2

[LaV99] LAVIOLA J. J.: *A Survey of Hand Posture and Gesture Recognition Techniques and Technology*. Tech. Rep. CS-99-11, Department of Computer Science, Brown University, 1999. 2

[QWJ00] QIN S. F., WRIGHT D. K., JORDANOV I. N.: From On-line Sketching to 2D and 3D Geometry: A System Based on Fuzzy Knowledge. *Computer-Aided Design 32*, 14 (2000), 851–866. 2, 5

[RA08] RAUSCH D., ASSENMACHER I.: A Sketch-Based Interface for Architectural Modifications in Virtual Environments. In *5th GI workshop VR/AR, Magdeburg* (2008). 8

[Rub91] RUBINE D.: Specifying Gestures by Example. *Computer Graphics 25*, 4 (1991), 329–337. 2

[SD04] SIMHON S., DUDEK G.: Pen Stroke Extraction and Refinement using Learned Models. In *Sketch Based Interfaces and Modeling* (Grenoble, France, 2004), Jorge J. A. P., Hughes J. F., (Eds.), Eurographics Association, pp. 73–79. 2

[SD05] SEZGIN T. M., DAVIS R.: HMM-Based Efficient Sketch Recognition. In *Intelligent User Interfaces* (2005), pp. 281–283. 2

[SD06] SEZGIN T. M., DAVIS R.: Scale-Space Based Feature Point Detection for Digital Ink. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Courses* (2006). 5

[SD07] SEZGIN T. M., DAVIS R.: Sketch Interpretation Using Multiscale Models of Temporal Patterns. *IEEE Computer Graphics and Applications 27*, 1 (2007), 28–37. 2

[SPB09] SCHMIEDER P., PLIMMER B., BLAGOJEVIC R.: Automatic Evaluation of Sketch Recognizers. In *SBIM '09: Proceedings of the 6th Eurographics Symposium on Sketch-Based Interfaces and Modeling* (New York, NY, USA, 2009), ACM, pp. 85–92. 2

[ZHH96] ZELEZNIK R. C., HERNDON K. P., HUGHES J. F.: SKETCH: An Interface for Sketching 3D Scenes. In *SIGGRAPH '96: Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques* (1996), pp. 163–170. 2