Hochschule für Angewandte Wissenschaften Hamburg
*Hamburg University of Applied Sciences*

# AW 2 Ausarbeitung

Dennis Kilan

Related Work: Sketch Recognition

*Fakultät Technik und Informatik*
*Department Informatik*

*Faculty of Engineering and Computer Science*
*Department of Computer Science*

# Inhaltsverzeichnis



# Abbildungsverzeichnis

# 1 Einführung

## 1.1 Motivation

Die Modellierung ist ein wichtiger Schritt beim Entwurf von Software Systemen. In der Modellierungsphase entstehen eine Vielzahl an verschiedenen Modellen. Alle Modelle haben jedoch etwas gemeinsam: Sie werden in der Regel immer in 2 verschiedenen Ansätzen gezeichnet.

Im ersten Ansatz werden die Modelle per Hand auf Medien wie Papier oder Whiteboards gezeichnet. Das Modellieren ist sehr einfach und intuitiv. Der Modellierer kann Ideen sofort zeichnen, ohne dass sein kreativer Gedankenfluß unterbrochen wird. Dieser Ansatz eignet sich hervorragend für kollaboratives Arbeiten.

Dieser Ansatz birgt jedoch auch viele Nachteile. Gezeichnete Elemente lassen sich nicht mehr verändern. Ein falsch gezeichnetes Element muss als ungültig markiert und neu gezeichnet werden. Damit die entstandenen Modelle gespeichert oder verteilt werden können, müssen sie nach der Sitzung manuell auf den Rechner übertragen werden. Das Zeichnen mit der Hand ist unsauber und die Modelle werden schnell unübersichtlich. Das Übertragen auf den Rechner ist also nicht nur zeitaufwändig sondern auch fehleranfällig.

Im zweitem Ansatz erstellen die Modellierer die Modell direkt in CASE Tools am Rechner. In diesem Ansatz liegen die Modelle direkt digital vor und müssen nicht aufwändig neu gezeichnet werden. Die erstellten Modellelemente lassen sich einfach ändern oder löschen.

Auch wenn dieser Ansatz viele Probleme des ersten Ansatzes löst bringt er auch einige Nachteile mit sich. Die Bedienung dieser Tools ist weniger intuitiv als das Zeichnen per Hand. Der Anwender muss erst den Umgang mit diesen Tools lernen. Das Erstellen von Zeichnungen über das WIMP[1] Konzept lenkt von der eigentlichen Aufgabe des Modellierens ab, wodurch der kreative Arbeitsfluss gebremst werden kann. CASE Tools eignen sich schlecht für das kollaborative Arbeiten.

Sketch Recognition kombiniert die Vorteile beider Ansätze. In der Sketch Recognition werden die Modelle mit den Fingern oder speziellen Stiften auf touchfähigen Oberflächen gezeichnet. Dadurch ist die Sketch Recognition so intuitiv wie der erste Ansatz. Der Rechner erkennt die gezeichneten Elemente und wandelt sie in die Elemente des Modells um. Somit lässt sich das Modell, wie bei einem CASE Tool, einfach verändern,speichern und verteilen.

[1] Windows Icons Menus und Pointer

## 1.2 Einführung in die Sketch Recognition

Um den Einstieg in das Thema Sketch Recognition zu erleichtern stellt dieses Kapitel Related Work vor. Die meisten Arbeiten im Bereich Sketch Recognition haben eine ähnliche Architektur(1), die in der Regel aus 3 Komponenten besteht.

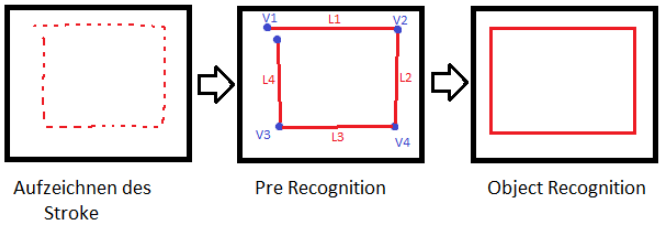


Abbildung 1: Sketch Recognition Vorgehen

In der ersten Komponente wird der Stroke von dem Rechner aufgezeichnet. Dazu merkt er sich in bestimmten Zeitintervallen die aktuelle Position des Eingabegeräts. Dadurch entsteht ein Stroke, der intern eine Liste von Punkten ist.

Ein Pre Recognizer kann nun diesen Stroke analysieren (2), um daraus einfache Formen zu erkennen. Einfache Formen sind zum Beispiel Linien oder Kreise. Ein Stroke besteht meist aus mehreren einfachen Formen, der Stroke muss dann in mehrere Substrokes aufgeteilt werden. Dazu kann der Pre Recognizer zum Beispiel im Stroke nach Ecken suchen. Eine Ecke dient dann als Trennpunkt zwischen den einfachen Formen.

In der Object Recognition wird versucht aus diesen einfachen Formen komplexe Formen zu erkennen. Der Object Recognizer hat dafür eine Beschreibung der komplexen Formen und versucht die einfachen Formen dieser Beschreibung zuzordnen. Es wird überprüft ob alle benötigten Formen für die komplexen Elemente vorhanden sind und ob die Beziehungen (Constraints) zwischen diesen einfachen Formen eingehalten werden. So besteht zum Beispiel ein Rechteck aus 4 Linien, wobei jeweils die gegenüberlienden Seiten parallel zueinander sein müssen und sich der Startpunkt der ersten Linie in der Nähe des Endpunkts der vierten Linie befinden muss.

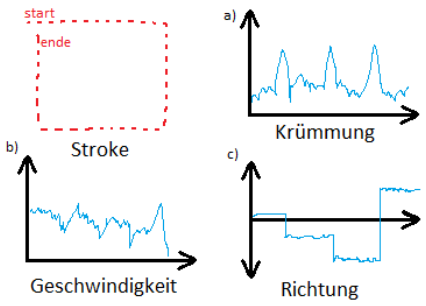


Abbildung 2: Analyse eines Stroke

# 2 Pre Recognition

## 2.1 Sketch Based Interfaces(2001)

2001 veröffentlichte Tevfik Metin Sezgin und Thomas Stahovich vom MIT[2] Sketch Based Interfaces zusammen mit Randall Davis von der CMU[3]. Die Motivation dieses Papers ist die Entwicklung eines Systems welches das direkte Zeichnen auf Rechnern erlaubt. Das Zeichnen soll so einfach sein wie das Zeichnen auf Papier. Die Besonderheit von Sketch Based Interfaces ist, dass es nicht vorschreibt, wie ein Element gezeichnet werden soll. Dadurch soll ein möglichst natürliche Art des Zeichnens erreicht werden.

Der erste Teil dieser Arbeit nennt sich „Approximation“ und befasst sich mit dem Finden von Geraden und Kurven aus dem aufgezeichneten Stroke. Weiterhin beschreibt das Paper noch die beiden Phasen „Beautification“ und „Basic Recognition“. Beautification wandelt den erkannten Stroke in eine sauberere Form. Wird zum Beispiel der Stroke als Gerade interpretiert, so ersetzt die Beautification den unsauberen Stroke durch eine Gerade. Die Basic Recognition versucht aus den Geraden und Kurven mit Hilfe von Templates komplexere Formen zu erkennen. Das Hauptthema dieses Papers ist die Approximation, die beiden anderen Gebiete werden in Sketch Based Interfaces nur kurz angerissen und sind somit kein Bestandteil dieser Arbeit.

Der erste Schritt in der Approximation ist das Finden von Ecken in dem Stroke. Sketch Based Interfaces nutzt die Krümmungs- (2 a) ) und die Geschwindigkeitsdaten(2 b) ) eines Strokes. Bei den Krümmungsdaten weist eine hohe Krümmung auf eine Ecke hin. Wie die Krümmung berechnet wird, wird in diesem Paper nicht erwähnt. Bei den Geschwindigkeitsdaten weist eine niedrige Geschwindigkeit auf Ecken hin, da beim Zeichnen einer Ecke die

[2] Massachusets Institute of Technology
[3] Carnegie Mellon University

Geschwindigkeit des Zeichnens abnimmt.

Wie in der Abbildung 2 zu sehen sind beide Daten sehr noisy, was zu vielen False Positives führt. Die erste Maßnahme dieser Arbeit ist das Filtern dieser Daten mit einem „Average Based“ Filter. Durch den Filter werden nur die Daten die über (Krümmung) bzw. unter (Geschwindigkeit) einem variablem Threshold (Mittelwert der Daten) liegen betrachtet. Da dieser Filter die False Positives nicht ausreichend herausfiltert, wird in einem zweitem Schritt aus beiden Daten hybride Fits erstellt. Dazu wird zuerst ein initialer Fit $F_0$ aus der Schnittmenge beider Datensätze gebildet. Aus den verbleibenden Daten werden in mehreren Zyklen jeweils der beste Kandidat für eine Ecke herausgesucht, der nicht in $F_0$ steckt, herausgesucht. Mit diesen beiden Kandidaten werden 2 neue Fits $F_i'$ ($F_0$ mit besten Kandidaten der Krümmungsdaten) und $F_i''$ ($F_0$ mit besten Kandidaten der Geschwindigkeitsdaten) generiert. Diese Fits werden mittels der mittleren quadratischen Abweichung vom Originalstroke miteinander verglichen. Der Fit mit der geringsten Abweichung wird der neue Fit $F_i$. Dieses geschieht solange bis es keine weiteren Kandidaten mehr gibt.

Mit diesen Vorgehen sollen Formen die nur aus Linien bestehen sehr gut gefunden werden. Dieser Ansatz ist jedoch zum Finden von Kurven so noch nicht geeignet. Deshalb wird nach dem Erstellen des hybriden Fits $F_n$ ein Kurventest durchgeführt. Dazu wird die euklidische Distanz $l_1$ mit der Bogenlänge $l_2$ zwischen 2 aufeinanderfolgenden Ecken verglichen. Wie die Bogenlänge berechnet wird, steht nicht in dem Paper. Das Verhältnis $l_2/l_1$ ist bei geraden Stücken ungefähr gleich 1. Bei einer Kurve ist dieses Verhältnis größer. In diesem Fall wird zwischen den beiden Ecken eine Bézier Kurve mit 2 Kontrollpunkte gezeichnet. Zur Bewertung dieser Kurve wird sie in lineare Teilstücke zerlegt und die mittlere quadratische Abweichung berechnet.

Die Evaluierung dieses Papers erfolgte über einen Usability Test mit 13 Personen die jeweils nach einer Einarbeitungsphase 10 Formen mit Sketch Based Interfaces und xFig, einem klassischem Modellierungstool für Unix zeichnen sollten. Diese 13 Personen empfanden das Zeichnen mit Sketch Based Interfaces einfacher und intuitiver als mit xFig. Laut dem Paper wurde in diesem Test 96% aller Ecken korrekt gefunden.

## 2.2 A Domain Independent System for Sketch Recognition(2003)

Die Arbeit von Bu Yi und Shijey Cai von der Nanjing Universität ist dem Ansatz von Sezgin sehr ähnlich. Dieses System stellt auch keine Restriktion an die Art wie eine Form gezeichnet werden soll. Der größte Unterschied zum Ansatz von Sezgin ist, dass das Finden der Ecken nicht als separater Schritt betrachtet wird, da es implizit in dem Finden der einfachen Formen steckt. Diese Arbeit besteht aus 2 Phasen: „Imprecise Stroke Recognition“ und „Post Process“.

Imprecise Stroke Recognition entspricht dabei der Pre Recognition, also dem Finden von einfachen Elementen aus dem Stroke. Die Post Process Phase ersetzt die unsauberen Strokes durch saubere Formen und versucht aus den gefundenen einfachen Elementen komplexere zu bilden. Diese Phase entspricht also den Phasen Beautification und Basic Recognition im Ansatz von Sezgin. Imprecise Stroke Recognition bildet den Hauptteil dieser Arbeit.

In der Imprecise Stroke Recognition werden zunächst die Richtungs- sowie die Krümmungsdaten eines Stroke berechnet. Die Richtung berechnet sich als $arctan\left(\frac{y_{n+1}-y_n}{x_{n+1}-x_n}\right)$ und die Krümmung ist das Verhältnis der Richtungsänderung in einem bestimmten Intervall zu der echten Pfadlänge des Stroke. Außerdem berechnet die Imprecise Stroke Recognition die sogenannte „Feature Area“ zu einem Referenzobjekt. Ein Referenzobjekt ist zum Beispiel eine Linie oder ein Punkt. Die Feature Area berechet sich durch den Unterschied des echten Stroke zu dem Referenzobjekt (siehe Abbildung 3). Die Geschwindigkeitsdaten werden nicht berechnet, da sie nicht zu den visuellen Features eines Stroke gehören und sie durch ihren Noise die Erkennung stören können.

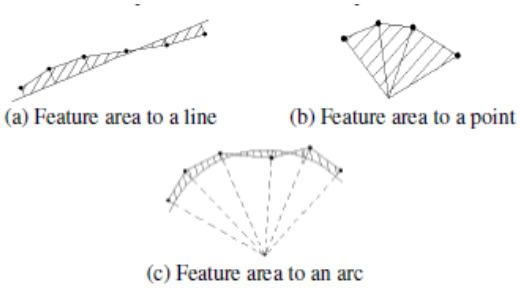


**Figure 2: The examples of feature area.**

Quelle:Yu und Cai (2003)

Abbildung 3: Feature Area

Nach der Berechnung der Richtungs- und Krümmungsdaten versucht dieser Ansatz den Stroke durch ein einfaches Element zu ersetzen. Schlägt dieser Versuch fehl, so wird der Stroke am Punkt mit der höchsten Krümmung geteilt und es werden diese beide neuen Strokes analysiert. Dieses geschieht solange bis jeder Stroke einem einfachen Element zugeordnet werden kann.

Als erstes wird versucht den Stroke durch eine Linie zu ersetzen. Dazu kann vom Start- bis zum Endpunkt des Stroke eine Gerade gelegt werden und die mittlere quadratische Abweichung zum Originalstroke berechnet werden. Liegt diese Abweichung unter einem Threshold, so wird eine Linie angenommen. Diese Hypothese wird dadurch abgesichert, dass man die Fläche dieser Geraden (Länge x Breite des Stroke) mit der Feature Area (siehe Abbildung 3(a)) vergleicht. Bei Geraden sollte dieses Verhältnis ungefähr gleich sein.

Schlägt der Linientest fehl, so wird zunächst nach Ellipsen im Stroke gesucht. Man kann diese Formen gut im Richtungsgraph erkennen. Wie in Abbildung 4 zu sehen haben Ellipsen im Richtungsgraph eine Steigung von ca. $2\pi/n$, wobei n die Anzahl der Datenpunkte ist. Bögen liegen zwischen Linien und Kreisen. Ihr Richtungsgraph hat eine geringere Steigung als eine Ellipse aber eine höhere als eine Linie.

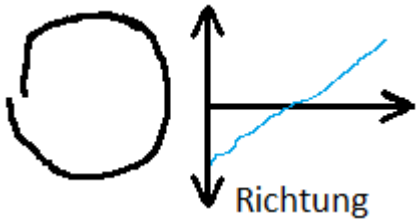


Abbildung 4: Konzepte des Ansatzes von Yi und Cai

Für die Evaluierung wurde ein Usability Test mit 10 zufällig ausgewählten Studenten durchgeführt. Im Test waren sowohl einfache Formen als auch zusammengesetzte Formen. Die einfachen Formen hatten eine Erkennungsrate von 98%, die hybriden Formen eine Erkennungsrate von 90%. Dieser Ansatz hat eine schlechtere Erkennungsrate als der Ansatz von Sezgin u. a. (2006), die Authoren schreiben das ihr System jedoch mehr einfache Formen erkennen kann und ihr Pre Recognizer mehr Informationen für die höheren Ebenen bereitstellt.

## 2.3 Short Straw: A simple and effective Corner Finder for Polylines(2008)

Dieser von Aaron Wolin (Texas A&M University) in 2008 entwickelte Ansatz befasst sich nur mit dem Finden von Ecken. Die Motivation dieser Arbeit ist, dass das Finden von Ecken mit den bisherigen Ansätzen viel zu komplex ist. Short Straw will das deutlich vereinfachen und zugleich bessere Ergebnisse liefern.

Die Grundidee hinter Short Straw sind die Straws. Ein Straw ist die euklidische Distanz von 2 nicht benachbarten Punkten $p_n$ und $p_{n+x}$ eines Stroke. Eine Ecke kann nun gefunden werden in dem der Straw mit der echten Pfadlänge, also $\Delta(p_n, p_{n+1}) + \Delta(p_{n+1}, p_{n+2}) + \ldots + \Delta(p_{n+x-1}, p_{n+x})$, verglichen wird. Ist die Pfadlänge länger als der Straw, so wurde eine Ecke gefunden (siehe 5). Da durch verschiedene Zeichengeschwindigkeiten die Distanz zwischen den Punkten nicht gleich ist, müssen die Punkte vorher so resampelt werden, dass sie äquidistant sind.

Nachdem die Ecken gefunden wurden, führt Shortstraw noch einen False Positive und einen False Negative Test durch. Für den False Positive Test wird die euklidische Distanz zwischen 2 gefundenen Ecken mit der echten Pfadlänge des Stroke zwischen diesen beiden Ecken ins Verhältnis gesetzt. Bei einer Linie ist dieses Verhältnis nahe bei 1, ist es kleiner als ein fixer Threshold so wird der mittlere Punkt zwischen diesen beiden Punkten als Ecke hinzugefügt. Für den False Positive Test wird zwischen 2 gefundenen Ecken $c_n$ und $c_{n+2}$ eine Gerade gezogen und überprüft ob die Ecke $c_{n+1}$ auf dieser Geraden liegt. Ist dieses der Fall wird die Ecke $c_{n+1}$ aus der Liste der gefundenen Ecken entfernt.

Shortstraw wurde ebenfalls in einem Usability Test evaluiert. Dafür haben 6 Studenten 11 verschiedene Symbole jeweils 4 mal gezeichnet. Die Ergebnisse dieses Test wurden mit dem Ansatz von Sezgin und einem weiterem Ansatz verglichen. In dem Test wurde gemessen ob alle Ecken gefunden wurden und ob die minimale Anzahl an Ecken (ohne False Positives) gefunden wurde. In dem ersten Fall erreichte Shortstraw eine Genauigkeit von 97%, der Ansatz von Sezgin nur 82%. In dem zweitem Fall sind die Ergebnisse 74% für Shortstraw und nur 28% für Sezgin.

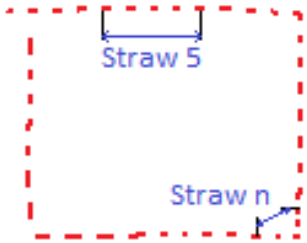


Abbildung 5: Shortstraw Prinzip

# 3 Object Recognition

## 3.1 LADDER (2007)

2007 veröffentlichten Tracy Hammond und Randall Davis vom MIT LADDER[4]. LADDER ist eine domänenunabhängige Sprache zur Beschreibung von Formen (Shapes). Der Anwender gibt in LADDER an, aus welchen einfachen Formen der zusammengesetzte Shape besteht und wie die Beziehungen (Constraints) zwischen diesen einzelnen Formen sein muss. Eine Beschreibung eines Rechtecks besteht zum Beispiel aus der Definition, dass es aus 4 Linien

[4] **la**nguage to describe **d**rawing, **d**isplay and **e**diting in sketch **r**ecognition

besteht, wobei Linie 1 und Linie 3 sowie Linie 2 und Linie 4 parallel zueinander sein müssen, sowie weiteren Constraints. Einfache Formen wie Linien oder Kreise müssen nicht definiert werden, da sie in LADDER bereits enthalten sind.

Eine Beschreibung in LADDER enthält noch viel mehr. Zusätzlich kann der Anwender noch das Editierverhalten festlegen. So kann der Anwender beispielsweise festlegen, dass ein Pfeil gestreckt wird, wenn er am Kopf gezogen wird und verschoben wird, wenn er am Schaft bewegt wird. Außerdem kann der Anwender auch noch festlegen wie die erkannte Form in LADDER dargestellt werden soll. Der Anwender kann zum Beispiel festlegen, dass Form A so dargestellt werden soll, wie sie gezeichnet wurde oder das sie durch eine „sauberere“ (geglätte Linien, rechte Winkel exakt 90°, usw.) Form ersetzt wird. Eine Beispiel Beschreibung ist in Abbildung 6 zu sehen.

Einige Formen sind Spezialfälle von abstrakteren Formen. So ist zum Beispiel ein Pfeil mit geschlossener Pfeilspitze ein Spezialfall des allgemeinen Pfeils mit offener Pfeilspitze. Beide Pfeile können das gleiche Verhalten aufweisen, der einzige Unterschied ist, dass der geschlossene Pfeil eine Linie mehr besitzt. Für solche Fälle lassen sich in LADDER Formen auch hierarchisch definieren. Außerdem lassen sich Shapes in LADDER gruppieren, falls alle Shapes fest zusammengehören. Ein Beispiel für solch eine Gruppe ist die „implements“ Beziehung in UML Klassendiagrammen, die aus 2 Rechtecken (Klasse und Interface) und einem Pfeil (implements Konnektor) bestehen. Mit dieser Gruppierung kann nun festgelegt werden, dass immer alle Formen zusammen bewegt werden.

Intern funktioniert LADDER über ein JESS Regelsystem[5]. Die vom Anwender geschriebene Beschreibung wird automatisch in einen „shape recognizer“ zum Erkennen der Shapes, einen „editing recognizer“ für die Editierkommandos und einem „shape exhibitor“ für die Darstellung des Shape übersetzt. Diese 3 Dinge entsprechen den JESS Regeln. Beim Zeichnen durchläuft der Stroke diese Regeln. Trifft eine Regel zu, so wird die gezeichnete Form erkannt. Treffen mehrere Regeln zu, so wird die genommen, die am besten passt (nach Ockhams Rasiermesser).

Für die Evaluierung von LADDER wurden Beschreibungen für viele verschiedene Domänen, wie UML Klassendiagramme oder Flowcharts, erstellt. Somit enstanden über 100 verschiedene Beschreibungen. Diese wurden automatisch in eine Zeichenoberfläche zum Zeichnen und Editieren übersetzt. Leider steht in dem Paper nichts davon wie erfolgreich LADDER diese Beschreibungen erkannt hat und wie der Test genau durchgeführt wurde.

[5] Ein Regelbasiertes System in Java

```
(define shape Arrow
    (comment "An arrow with open head")
    (components
        (Line shaft)
        (Line head1)
        (Line head2))
    (constraints
        (coincident shaft.p1 head1.p1)
        (coincident shaft.p1 head2.p1)
        (equalLength head1 head2)
        (acuteMeet head1 shaft)
        (acuteMeet shaft head2))
    (editing
        ((trigger (holdDrag head))
            (action (rubberBand this tail head))
        ((trigger (holdDrag tail))
            (action (rubberBand this head tail))
        ((trigger (holdDrag head))
            (action (move this)))
    (display
        (originalStrokes shaft)
        (cleanedStrokes head1 head2)
        (color red))
)
```

Abbildung 6: Beschreibung in LADDER

## 3.2 SketchML (2009)

Im Jahr 2009 veröffentlichten Danilo Avola („La Sapienza“ Universität, Rom), Andrea Del Buono (National Research Center, Rom), Giorgio Gianforme und Stefan Paolozzie (jeweils „Rome 3“ Universität) sowie Rui Wang (Saarland Universität) das Paper „SketchML a Representation Language for novel sketch recognition approach“. Die Motivation dieses Papers ist sowohl die Steuerung von Geräten über 2D Gesten sowie das Zeichnen von Modellen auf geeigneten Oberflächen. SketchML ist flexibel genug um in mehreren Domänenen arbeiten zu können. SketchML ist eine XML basierte Sprache um optimalen Datenaustausch mit eventuell zu steuernden Geräten zu ermöglichen.

SketchML besteht aus 2 Hauptelementen: die Library Editor Umgebung (LEE) zum Erstellen von Libraries und die Sketch Recognition Umgebung (SRE) zum Erkennen der gezeichneten Symbole. Eine Library ist dabei die Menge der zu erkennenden Symbole für einen bestimmten Bereich (z.B. Petrinetze).

Zum Erstellen neuer Symbole für eine Library zeichnet der Anwender das neue Symbol in der LEE. Das gezeichnete Symbol wird zunächst in der First Layer Segmentation in Closed Regions (komplett von Linien umschlossener Bereiche) und Polylinien zerlegt. Sind diese Elemente gefunden, so werden diese in der Second Layer Segmentation in Ovale, Bögen und Linien zerlegt. Aus diesen einfachen Elementen kann die Zeichnung in das SVG Format umgewandelt werden, welches ebenfalls eine XML basierte Sprache ist. Im letzten Schritt werden noch die Constraints der gefundenen einfachen Elemente untersucht. Constraints sind zum Beispiel, der Winkel zwischen 2 Elementen oder die Orientierung eines Elements. Mit den einfachen Elementen im SVG Format und den Constraints kann nun eine Beschreibung im SketchML Format erstellt werden. In der Sketch Recognition Umgebung passiert zunächst dasselbe (First Layer Segmentation,Second Layer Segmentation, usw.) wie in der LEE. Zusätzlich muss die so aufgezeichnete Zeichnung mit den Library Symbolen verglichen werden. Das Vergleichen geht sehr einfach, da sowohl die Zeichnung vom Anwender als auch das

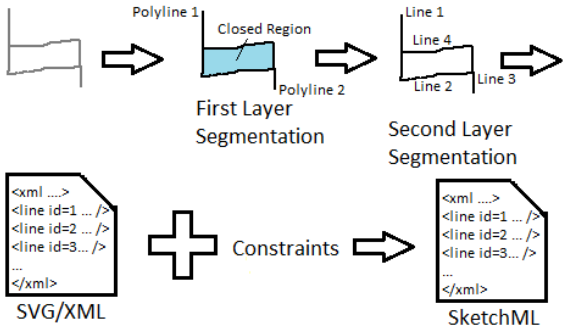


Abbildung 7: Erstellen eines neuen Symbols

Library Symbol im SketchML Format vorliegen. Die einzelnen Objekte der Zeichnung können wegen diesem XML Format einfach mit allen Objekten der Library verglichen werden.

Für die Evaluierung wurden 154 verschiedene Symbole entworfen. Diese Symbole wurden von 80 verschiedenen Personen verschiedener Altersklassen mehrfach gezeichnet. Insgesamt wurde jedes Symbol insgesamt 75 mal gezeichnet. Die Authoren behaupten eine Erkennungsrate von 98% erreicht zu haben. Probleme bereiten Symbole, die aus mehr als 7 Elementen bestehen, da die Constraints dieser Symbole mehrdeutig sind.

# 4 Abgrenzung

Der Ansatz von Yu(2.2) und Sketch Based Interfaces (2.1) sind sehr ähnlich. In beiden Ansätzen wird zunächst nach Ecken im Stroke gesucht, auch wenn es im Ansatz von Yu nur implizit passiert. Dieses Vorgehen ist sinnvoll um den Stroke in die einfachen Formen einteilen zu können, weshalb ich diesen Schritt übernehmen will. Leider lieferte meine Implementierung weitaus schlechtere Ergebnisse als die angebenen Ergebnisse in den beiden Arbeiten.

Shortstraw(2.3) lieferte bessere Ergebnisse. Jedoch eignet sich Shortstraw nur für Zeichnungen ohne Kurven und Ovalen, da in einer kurvenreichen Region alle Straws sehr kurz sind und demnach viele False Positives gemeldet werden können. Eine gute und sinnvolle Idee von Shortstraw ist das Resamplen von Punkten um die verschiedene Geschwindigkeiten beim Zeichnen auszugleichen.

Mit dem Resampeln von Shortstraw und der Formel zur Berechnung der Richtung aus dem Ansatz von Yu konnte ich letzten Endes dennoch einen Ansatz entwickeln der gute Ergebnisse liefert. Eine Ecke kann mit resampelten Punkten über die Änderung der Richtung gefunden werden. Eine komplexe Berechnung der Krümmung entfällt damit. Um einfache Formen zu finden können die Ansätze von Yu verwendet werden, jedoch habe ich sie bisher noch nicht getestet und kann somit keine Aussage über diese Algorithmen treffen.

Den Bereich Object Recognition habe ich im Projekt noch nicht weiter untersucht. Jedoch klingt LADDER(3.1) vielversprechender als SketchML (3.2). In SketchML werden die Formen in einer Trainingsphase gelernt und in LADDER textuell beschrieben. Da ich mich im Bereich der Modellierung mit eindeutig beschreibaren Elementen bewege, ist eine textuelle Beschreibung ausreichend und der große Aufwand einer Trainingsphase entfällt.

## Literatur

[Avola u. a. 2009] AVOLA, Danilo ; DEL BUONO, Andrea ; GIORGIO, Gianfor ; PAOLOZZI, Stefano ; WANG, Rui: SketchML a representation language for novel sketch recognition approach. In: *Proceedings of the 2nd International Conference on PErvasive Technologies Related to Assistive Environments*. New York, NY, USA : ACM, 2009 (PETRA '09), S. 31:1–31:8. – URL http://doi.acm.org/10.1145/1579114.1579145. – ISBN 978-1-60558-409-6

[Hammond und Davis 2007] HAMMOND, Tracy ; DAVIS, Randall: LADDER, a sketching language for user interface developers. In: *ACM SIGGRAPH 2007 courses*. New York, NY, USA : ACM, 2007 (SIGGRAPH '07). – URL http://doi.acm.org/10.1145/1281500.1281546

[Sezgin u. a. 2006] SEZGIN, Tevfik M. ; STAHOVICH, Thomas ; DAVIS, Randall: Sketch based interfaces: early processing for sketch understanding. In: *ACM SIGGRAPH 2006 Courses*. New York, NY, USA : ACM, 2006 (SIGGRAPH '06). – URL http://doi.acm.org/10.1145/1185657.1185783. – ISBN 1-59593-364-6

[Wolin u. a. 2008] WOLIN, Aaron ; B.EOFF ; HAMMON, Tracy: ShortStraw: A Simple and Effective Corner Finder for Polylines. In: *EUROGRAPHICS Workshop on Sketch-Based Interfaces and Modeling (2008)*, URL http://srl.csdl.tamu.edu/courses/SR2008/papers/03_corner_finding/ShortStraw.pdf, 2008

[Yu und Cai 2003] YU, Bo ; CAI, Shijie: A domain-independent system for sketch recognition. In: *Proceedings of the 1st international conference on Computer graphics and interactive techniques in Australasia and South East Asia*. New York, NY, USA : ACM, 2003 (GRAPHITE '03), S. 141–146. – URL http://doi.acm.org/10.1145/604471.604499. – ISBN 1-58113-578-5

## Glossar

**CASE Tool** CASE (Computer Aided Software Engineering) Tools helfen bei dem Entwurf von Softwaresystemen zum Beispiel als Editor zum Erstellen von Modellen.

**Fit** Ein Fit ist ein mögliche Passform für eine bestimmte Form. Ein Fit ist besser, desto weniger er von dieser bestimmten Form abweicht.

**Multitouch** Eine Bedienungsfeld eines Rechners, welches die Eingabe über mehrere gleichzeitige (Multi-) Berührungen (Touch) erlaubt.

**Ockhams Rasiermesser** Wissenschaftliches Prinzip, das sagt, dass bei mehreren möglichen Erklärungen die 'einfachste'(minimale Hypothesen und Variablen) gewählt werden soll

**Smartboard** Ein intelligentes Whiteboard von Smart Technologies. Smartboards erkennen das Zeichnen mit speziellen Stiften auf der Whiteboard Oberfläche, und kann somit wie ein Touchscreen bedient werden.

**Stroke** Ein Stroke ist der aufgezeichnete Strich, den der Rechner von Berührung der Zeichenoberfläche bis zum Verlassen der Zeichenoberfläche aufzeichnet, in dem sich der Rechner in bestimmten Zeitintervallen die Position des Eingabegerätes merkt.