Sketching

Jan Marten
jan.marten@rwth-aachen.de

Julian Meichsner
julian.meichsner@rwth-aachen.de

ABSTRACT

Sketching is a special kind of drawing used to quickly express ideas like the design of objects. While there are already systems available for drawing, these do not support the special affordances and requirements of sketching. In this paper we present some approaches for supporting the user in creating sketches. We focus on three different kinds of sketching: 2D sketching like we are used to with pencil and paper, sketching of 3D models and graph sketching. We evaluate each of the approaches and provide an outlook what problems still have to be solved to replace traditional sketching with digital sketching.

INTRODUCTION

Sketching is a technique which allows to quickly express ideas that someone has in mind. Classic sketching was almost always done with pencil and paper. The advantage of pencil and paper is that a sketch does not look finished which also usually is not the case because only a quick idea was visualized. Furthermore, sketching focuses on the main features of an idea or concept. Nowadays, in contrast to early sketching, computers are able to help humans with sketching. Thus, digital sketching is an alternative to pencil and paper sketching. To sketch with a computer there are different input devices available like graphics tablets and digital pens which allow users to draw as if they drew on paper.

With digital sketching users can be aided by the computer by for example making an intended circle a real circle or making an intended square a real square [3][11]. Furthermore, multi-strokes can be approximated to one curve. A sketching software can also display assisting lines or a grid on which all sketched lines are fitted to [2][3]. All in all a computer allows even non-experienced users to draw better sketches who would not be able to draw as well on paper [5].

Sketching programs have a long history, one of the first approaches for a sketching system and one of the first computer programs ever with a graphical user interface was *Sketchpad* by Sutherland et al. [13].

There are different sketching techniques. The best known technique is 2D sketching as this technique matches the pencil and paper like sketching. Therefore, these approaches really need to compete with traditional sketching. Another sketching technique is 3D sketching. With the help of a computer a user is able to create sketches in 3D. In 3D sketches a user is able to rotate and move the view, thus, the sketch can be seen from different sides. With classic pencil and paper one can only sketch a 3D object from one perspective. A last sketching technique we will discuss is graph sketching. Graph sketching is a technique where a graph is created, but without real raw data. These techniques are used for example to illustrate future development of a company's incomings.

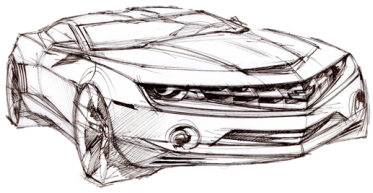

Figure 1. An example 2D sketch.[10]

This paper is structured as follows. In the first section approaches for 2D sketching will be explained. Following, in the next section approaches for 3D sketching will be explained. The next section will discuss approaches for sketching graphs. Finally, we will conclude the paper with a summary and a short outlook. At the end of every section we conclude each of the approaches.

2D SKETCHING

2D sketching is used by nearly everyone to quickly visualize an idea to others or to keep an idea in mind. But more general 2D sketching is mainly used by artists and designers. Figure 1 shows an example 2D sketch of a car. On the one hand designers and artists use sketches to express and remember ideas, but on the other hand sketches are often first versions of a final product. A sketch should usually not look finished, since only the concept should be expressed. Figure 1 is an example why a sketch should not look finished. The shape and concept of the car is clearly visible but it is not focused on the details like the mirror of the car. Thus, clients cannot argue about the details.

The ability to use digital 2D sketching to sketch ideas brings some advantages as well as some disadvantages. Digital sketches have the advantage that they are saved on storage whereas classic sketches have to be digitalized for instance with a scanner in order to preserve the sketch. Moreover, the quality of the sketch does not decrease. Pencil and paper

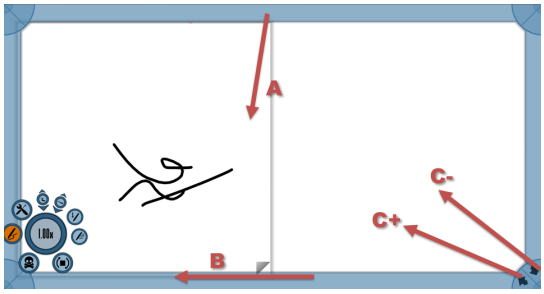

Figure 2. Interactions with the canvas. Translation (A), rotation (B) and scaling (C)

sketches may be crumbled or the ink may vanish. In addition to advantages which are not sketching specific there are also advantages which especially belong to digital sketching. One advantage is the aid the computer provides to the user. Lines can be approximated to straight lines, multi-stroke curves can be approximated to one curve. These advantages allow even non-experts to sketch things which they would not be able to sketch with pencil and paper.

But this also introduces a problem, because with such beautification a sketch might not look like a sketch anymore. Thus, by presenting such sketches, people might think that the idea is already finished.

Exploring Frame Gestures For Fluid Freehand Sketching

One major difference for artists and designers who are used to traditional sketching is the interaction. The sketchbook is the main interaction point. The sketchbook can be hold and rotated in any way according to the preferences of the artist or in order to support the natural hand movement.

Nijboer et al. [9] used the interactions which are done with a traditional sketchbook and tried to explore which interactions can be mapped to a sketching software. They explored an approach with a minimalistic user-interface which uses gestures. They focused predominantly on interactions with the drawing canvas and on stroke manipulation.

Early approaches to sketching software did not have a rotatable drawing canvas in mind, thus, artists were forced to draw in only one view. This forced view is unnatural and may even reduce the quality of a sketch because the user cannot use the natural hand movement for instance to draw a curve.

The system allow a user to rotate, translate and scale the drawing canvas. Since, Nijboer et al. tried to build a minimalistic user-interface, they use the frame of the interface as the most important interaction point. Figure 2 shows the three possible interactions with the canvas. Gesture A is used to translate the canvas. The stroke starts at the frame of the interface and is directed to the mid of the interface. Nijboer et al. explained the gesture as a mapping from interaction with paper, where a finger is placed at the border and is moved in the perpendicular direction to move the paper. Gesture B is used to rotate the canvas. The pen starts again at the border of the interface and is moved parallel to the border. Here they also tried to use a natural mapping of rotating a paper. Although the paper has to be hold in the center. The last gesture C is used to scale the canvas. Here two starting points are possible. Starting from the corner and moving the pen to the center of the interface can either be used to scale the canvas down (C-) or to scale the canvas up (C+). Scaling of a canvas is a real improvement over traditional sketching.

The same gestures can be applied if the user selected one or more strokes, since then a frame around the strokes is displayed which allows the exact same gestures. Drawing is done by using and pressing the pen and erasing with the eraser tip of the pen. To erase a whole stroke or a group of strokes further gestures are used. First of all a hold gesture with the eraser tip is required. After releasing a red circle is displayed. A scratch-out gesture starting from there with the eraser tip will delete all strokes which are touched.

The authors evaluated their system with ten people which are categorized into two classes. One group of expert users who were already familiar with other sketching and drawing programs and a group of five novice users who had little to no knowledge of digital sketching programs. The study was divided into two phases. In the first phase the users should experiment with the software and they were not given any explanation of the available gestures and the interface. After this phase the users got an introduction to the interface.

The novice users could easily draw with the basic functionality without introducing the features. Both groups could easily use the interface after the introduction. The experts could incorporate the possible interactions faster then the novice users. Overall they received positive feedback for the system.

With the help of the evaluation the authors were also able to see some flaws. One of them is a missing undo function. But they also received negative feedback for the zooming function since, as mentioned above, users are only able to zoom into the canvas by starting at the corner and moving the pen into the center of the interface. This is not a natural mapping, especially not since zooming out is using the same direction but only a different starting point. The same applies to the translation gesture. Starting the gesture at the left border it is not possible to translate the canvas to the left since there is no screen space left.

One major disadvantage of gesture based systems is that the user is not aware of all possible interactions, hence, users always need an introduction to use all features. The same applies here because the evaluation showed that novice users could draw but only with the basic functionalities.

iCanDraw?

As mentioned previously, digital sketching systems can assist the user in drawing and thus allow even users with little to no experience to sketch and draw. Dixon et al. [5] developed the system *iCanDraw?* which helps the majority of people who claim to be unable to draw well. They chose the

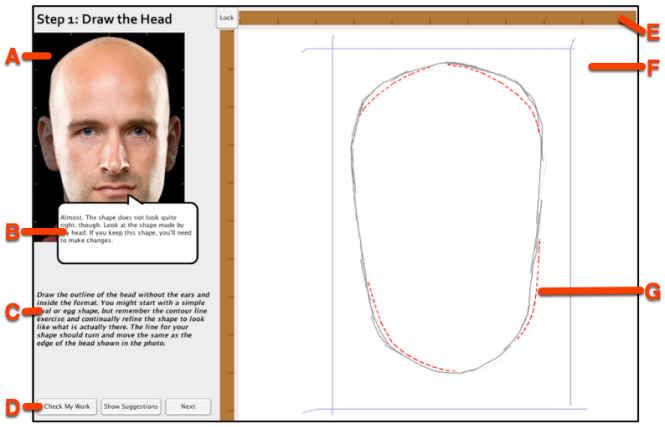

Figure 3. *iCanDraw?* interface: A: reference image, B: Text feedback, C: Task, D: Options, E: Straightedge tool, F: Canvas, G: Visual feedback [5]

task of drawing a human portrait faces which are provided as reference images through the whole drawing process.

One motivation for the creation of such an assisting system was that novice drawers usually draw what they know and not what they see. Novice users for example often draw eyes as two ellipses with two filled circles as the pupils. Thus, such a system helps the user to see where an error is made and to better perceive what should be drawn. The authors utilize a step-by-step construction of the final image to teach drawing a human face. In each step the current task is shown and after the user is finished with a step the system is asked to check the work. The system provides corrective feedback in a visual and textual way to the user. The user can then decide to improve the sketch or advance to the next step.

Figure 3 shows the interface of *iCanDraw?*. It is divided into the following parts: A shows the reference image the user should draw. Here, depending on the current task, different regions can be emphasized by dimming out other regions for example highlighting the nose. B provides textual feedback. C displays the current task in the different steps. E is a straightedge tool. F is the drawing canvas where the user draws the face. The red dotted lines at G provide the visual feedback.

In every step the system provides corrective feedback and the user may correct his work according to the feedback. In order to assist the user and provide corrective feedback the system needs to get some knowledge of the reference image and the sketched image. First of all the reference image is processed. The authors use a face recognition library to extract the most important facial features. For the reference image an example template is created which is used to compare the drawn sketch with the reference image. In each step the drawn strokes are recognized then classified and put into different pools depending on the similarity to other strokes (for instance all nose strokes). Then for each facial feature

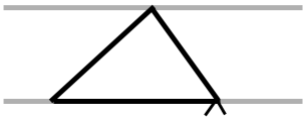

Figure 4. The caret snapping on an alignment line while dragging a triangle in *Snap-Dragging* [4].

the proportions and alignment of the strokes from one pool are compared to those of the example template.

The authors used two design iterations to create the interface and found several things in the first iterations which could be improved. Furthermore, they evaluated their system with five participants who claimed not to be able to draw well. The result was that every participant became better even with freehand drawing after gaining corrective feedback on the first examples which was also confirmed with a quantitative analysis of the drawn faces.

Such systems can really aid novice users in learning to draw. Nevertheless, drawing face images is just a small step towards the goal of learning to draw just with a computer, because it easy to extract important features from face images with known algorithms but very hard for other objects.

Snap-Dragging

An early approach on supporting the user to draw 2D objects consisting of straight lines is *Snap-Dragging* [4], which was introduced in 1986. The idea was, to use a ruler and compass (two tools usually used by draftsmen) metaphor in a computer program for making precise line drawings possible. The methods of Snap-Dragging were implemented in an illustrator system called *Gargoyle* which will not be further discussed here.

The user mainly interacts with the system with a so-called *caret*, a kind of marker which can be moved by the cursor and snaps to special points like control points, line endings or line crossings. When adding a new line to the drawing, the user specifies the start and end point of it by this caret. If he wants to connect the line with an existing line, the caret will automatically snap to the end point of that line by using a gravity function, and therefore supports the user in this task. While moving objects, the system displays alignment lines to which the caret will also snap, as shown in Figure 4. These alignment lines are placed related to other existing lines or points in the drawing. The drawing of a line between two clicks with the caret, as well as the option of enabling alignment lines, is what is meant with the ruler metaphor, because these are tasks a draftsman would usually draw with a ruler on his sketch board. Other functionalities of the system include moving, rotating and scaling drawn forms.

Drawing With Constraints

Gleicher et al. [7] created a system called *Briar* which allows the user to draw objects and then establish relations between them, the constraints. In contrast to other constraint based drawing systems where the user specifies relations be-

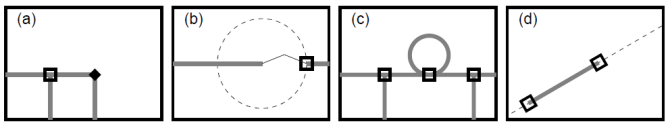

Figure 5. Constraints including their visual representations: a) point-on-object and point-on-point constraint b) distance between points c) points-aligned d) orientation. [7]

tween objects and afterwards the system has to solve the constraints which satisfy the relations, in *Briar* the user only creates constraints for relations which have been established before. Thus, there is always an initial solution available and therefore the system does not need to find a solution for constraints which come from arbitrary states.

The system employs a technique called *Augmented Snap-Dragging* which bases on the approach by Bier et al. Using augmented Snap-Dragging the user can create constraints with direct-manipulation which is also the only method in *Briar* to create constraints. Different types of constraints can be created with augmented Snap-Dragging including: positions, orientations and distances. Figure 5 shows some constraints for example distance b) and orientation d). A relation between objects can be created by dragging an object to another point or object. Then the system snaps the first object to the second object and creates a relationship. The user then can either choose to accept the new relationship, ignore it or if automatic constraint creation mode is turned on the constraint is automatically created and the user is able to reject the newly created constraint. In addition to dragging and snapping objects onto each other the user is also able to use the so called alignment objects to create constraints, for example an alignment circle, which were explained above. There might be ambiguity when a new relation should be established. Therefore, the user can cycle through possible objects which can be snapped to. To delete or edit a constraint *Briar* offers two methods. While dragging an object the user can press a key to "grab" that object "hard" which removes all constraint belonging to the point. The second method is to disable constraint maintenance. In this mode all constraints are disabled and the user may move objects around. After re-enabling constraint maintenance all violated constraints are then removed.

Briar provides the user with different feedback. If the cursor is snapped to an object or an edge the shape of the cursor changes and the color of the object changes as well. Constraints, which represent an additional state in the drawing, need to be visible to the user as well. Therefore, *Briar* uses an empty square for point-on-object constraints and a filled diamond for point-on-point constraints. Figure 5 a) shows the two basic constraints. Other constraints are visualized with the help of the two basic visualizations and additional alignment objects like in b) or d).

The authors also found some limitations of the system. First of all not all Snap-Dragging presented in [4] were implemented in *Briar* since it was only a research prototype. Further problems arise when the drawings become larger. One problem when the paper was written was scalability but as the authors already stated this is solved now because the processing power increased significantly. But a further problem with large drawings is clutter. With a large number of objects snapping may be problematic because there are many object which can be snapped to and the user might also cycle through a lot of objects to create the correct constraint.

Figure 6. *Pegasus*: Showing multiple candidate strokes

Interactive Beautification

A different approach to assist users in sketching was done by Igarashi et al. [8]. They focused on beautifying free strokes by also utilizing geometric constraints. This technique can be applied in 2D sketching but even more in sketching diagrams. The idea is that the user can draw free strokes without changing a mode, and the system infers what was intended to draw and then beautifying the stroke whilst taking into account different geometric constraints. Such a technique allows even users who are not able to draw well to create good looking sketches or diagrams but furthermore, this lets the user create these sketches or diagrams very fast and efficient.

To beautify the free strokes of a user the system infers geometric constraints from it, which depend on the strokes drawn before, and then solves these constraints displays the result or multiple candidates. Multiple candidates are shown because a stroke can often be ambiguous. There are several supported geometric relations by the systems like connecting strokes, parallelism, perpendicularity, alignment of strokes, congruency, symmetry and interval equality of strokes.

As already mentioned above, a drawn stroke may be ambiguous to the system. Thus, the system generates all possible candidates. The user can then choose the already selected primary candidate, another candidate by tapping on the candidate or redraw the stroke if the intended stroke is not displayed at all. Figure 6 shows the primary candidate stroke in red and two other possible candidates in magenta. To erase strokes are trimming parts of strokes the user can use the scratch-out gesture which is also used in the previous approaches.

The authors evaluated the system with 18 students where they focused on the metrics rapidness and precision. The students were asked to draw three different diagrams where they should be fast but also try to satisfy all geometric constraints with three different systems. They evaluated their *Pegasus* prototype software against a CAD software and an object orientation based drawing software. The result was that the users were faster with the *Pegasus* system and furthermore, they were more precise compared to the other drawing softwares.

This software also showed some limitations since it is lim-

ited to lines. Furthermore, it is difficult to select the desired candidate as there may be many candidates which may also overlap each other.

Sketch Based Interfaces

Sezgin et al. [11] created an approach which is along the same lines as the approach [8]. As before the authors only want the user to draw and not to deal with a large user interface with different drawing tools and menus, especially not for creating different geometric objects like curves and lines, since this would interfere with the interactions done with pencil and paper, and thus, again increasing the complexity.

The main focus in the approach of [11] is the approximation of strokes. The authors want to approximate the strokes with a more abstract representation. Therefore, the first step of approximating the strokes is to detect vertices which define the starting and end point of a segment. The authors use an interesting approach to detect the vertices which is speed and curvature data. The idea is that these are some main metrics which define a segment and furthermore, especially the beginning of a new segment. If, for example, the user draws a square, then the direction of the pen changes at the corners to start a new segment, thus, the curvature is high. But furthermore, usually also the drawing speed decreases. The authors use a combination of both metrics, minima in speed and maxima in curvature, to detect vertices because with only one of both metrics a new segment cannot be detected, since a user might for example slow down even on a straight line without intending to change direction. To approximate lines the authors use the least squares error to evaluate the quality of a fit and for approximating curves they use Bèzier curves with two end points and two control points.

The system also beautify the strokes after the detection and approximation phase like in *Interactive Beautification*, but the strokes are not as much beautified as in that approach.

The authors evaluated the system with thirteen participants which mainly were from the computer science field. They focused on the aspect how natural and efficient the system is. Therefore, they compared their system with Xfig which is a tool to create diagrams on Unix systems. Overall they received positive feedback and the result that their system is more natural. Moreover, all but one participant favor their system over Xfig. Additionally they evaluated how correct the approximations of the lines and curves were and state that 96% of the lines and curves were approximated correctly.

Live Paint

Live Paint [1] is a system for coloring dynamic planar map illustrations and was presented by Adobe Systems in 2007. Illustration systems based on planar maps do not have different layers of paths, but all paths lie unordered in a single plane. A well-known example for such a system is Adobe Flash. When filling coloring in this kind of illustration, a fill area is just bounded by the paths surrounding it. Therefore it is difficult to keep the coloring when changing paths of the image. Live Paint allows to edit the paths after coloring, and keeping the most reasonable coloring when paths move or are deleted.

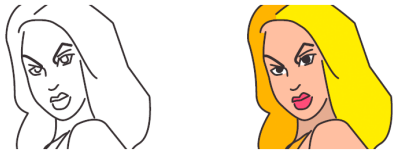

Figure 7. A sketch with gaps on the left, and the coloring done with *Live Paint* on the right [1].

When a path in the image is moved, Live Paint calculates the best solution to color the new image. Therefore it at first checks which regions changed, and which were untouched. When a path is just moved, without adding or deleting any regions, the color of those regions can just stay the same as it was. If a new path is created, or an old one is extended, so that it subdivides some regions, all new subregions will be colored in the same way as the whole region was before. Another simple operation is the deletion (or shortening) of a path: the new region gets the fill color of the largest of the subregions, that are now merged together. For more complex changes in the image, algorithms calculate the new coloring based on the so-called context of the regions. This context contains information about each path bordering the changed region, including the position of the path relative to the region, if a path is closed, whether the region is inside or outside the path, and so on.

The detection of gaps makes the system useful for freehand sketching. Gaps often occur in the outlines of a region when drawing them by hand. Live Paint detects those gaps, and regards the regions as closed when calculating the region coloring. An example of the coloring of a sketch including gaps is shown in Figure 7. The gap detection algorithm is called every time the image changes, because gaps can move around while editing the image, or new ones can be created.

Live Paint was not intended as a stand-alone system from Adobe Systems, but was developed to research algorithms and some heuristics for coloring regions. While not focused on hand-drawn images, the presented algorithms are also perfectly applicable in this area, because often digital 2D sketches are also based on the planar map metaphor, and the system offers a good gap detection, that often occur on this kind of images.

Evaluation and Comparison

We presented six different approaches for digital 2D sketching. Most of them had a different focus. In this section we want to evaluate the different systems.

The creation of the first system [9] focused mainly on the interactions with the canvas which should map to the traditional sketchbook. Although they were successful in creating a non-distracting user interface it is doubtable that the

interface is intuitive. Especially the small circular menu, where no description or tool-tip is available, does not communicate what settings are changed with the different bubbles, which was also a negative point in their evaluation. Furthermore, the translation and rotation and particularly the scaling of the canvas is not very intuitive as well. All in all the user really needs to learn the system to be able to interact with it and use all the possible features, thus, it is doubtable if an artist would prefer such a system over pencil and paper.

The second system, ICanDraw? [5], which provides the user with corrective feedback for their drawing and thereby tries to teach the user in drawing faces, is a good example for using a computer to help the user in sketching. This shows some real advantages of a computer over traditional pencil and paper sketching, because such a feedback could only be given by real teacher. The interface seems to be intuitive and only the gestures need to be learnt. The only shortage of the system is the evaluation because the system was only evaluated with five participants.

The third system [4] is an older system but it smoothed the way for further systems with their Snap-Dragging technique. Only precise lines can be created, thus, the main focus is sketching diagrams with it. The interactions are outdated nowadays but the snapping technique is used in many following approaches.

One of them is the fourth system [7] which uses constraints for creating drawings. Constraints can be helpful in creating sketches and drawings but they may also stop the rapid creation of sketches. However with constraints the user is able to easily edit a sketch or a drawing. But usually such system will only be used for sketching diagrams.

The fifth system [8] beautifies the drawn strokes very harsh. Thus, the result of a sketch drawn with such a system does not look like a sketch anymore. This might be problematic as explained before. But on the other hand this allows the user to create sketches or diagrams, which in this case only consist of lines, very rapidly. Mainly there are only two interactions: drawing and erasing with the scratch-out gesture. Therefore, interacting with the system is very easy. However, for instance a missing undo function is a shortfall of the system. They evaluated their system well since they used 18 participants and furthermore compared their system to other system which were representatives of different types of systems.

In the system by [11] the strokes are beautified only a little, but the system is very good at approximating the original strokes. The interactions with the system seem to be intuitive as well. The system was only evaluated with participants from the computer science field. These are not representatives of the main users, thus, the evaluation is questionable. Nevertheless, this system is a good system for understanding sketches and with some future work such systems may understand nearly the whole sketch and thus, they can support the users very well in creating sketches and diagrams.

3D SKETCHING

While 2D sketching may be made easier with the help of computer sketching systems, there exists a type of sketches that would not be possible without one: three dimensional sketches. With 2D sketches, the user works on a two dimensional plane (which may be a digital one, or just a piece of paper), and may compensate the lack of a third dimension by using vanishing points, shades, and other tricks that lead our brain to perceive a 3D object. But using this method, it is only possible to show a very limited amount of surfaces of an object. Concept arts of new cars usually show the front, the top and one side of the car. But if the designer also wants to show the rear, he has to start a new drawing next to the first one. If he wants to sketch a full 3D model of it, it is unavoidable to use a computer for this task.

The systems explained in this section come up with the logical next step to 2D sketching systems, namely 3D sketching. The user does not only create a two dimensional view on an object, but is able to create a three dimensional representation. Note that many techniques presented in the earlier section, like interactive beautification [8], can also partly be used for 3D sketching.

2D-to-3D Modeling

The first approach to present is not a complete 3D sketching tool on its own, but a system with which a user can create 3D models out of 2D sketches with the help of annotations placed on those [6]. When creating 3D models with software tools, the user normally has to rotate the image very often, for modeling every part of the object. The authors of the 2D-to-3D system want to avoid that by offering the user the ability to import any kind of sketch created with a computer or by hand with pen and paper, and create a 3D model out of that.

The first step to create a 3D model after importing the 2D sketch is to place primitives on the original image. Primitives are basic shapes that are used to create the different parts of the model, e.g. one for the head of a person, one for the body, and several for arms and legs. The system provides generalized cylinders and ellipsoids for this task. To create a cylinder type primitive, the user draws a simple line in the image. This generalized cylinder will then be placed around this line with a given width. In case of an ellipsoid type primitive, the user simply draws an ellipsoid. Both types of primitives are presented as 2D outlines in the drawing view above the original image. Each primitive type has different attributes specifying the behavior of it in 3D space, which can be changed by the user at any time. The generalized cylinder has the following handles for those attributes: out-of-image-plane tilt handle, cross-section scale handle, symmetry sheet handle, and end-cap handles.

The out-of-image-plane tilt handle allows to bend a primitive out of the image plane. By default, a primitive is perpendicular to the image plane. The change of this attribute is shown by using different colors for the areas dragged out of the plane, and a 3D preview is always shown in a small window, which can be fully rotated by dragging it. The cross-section

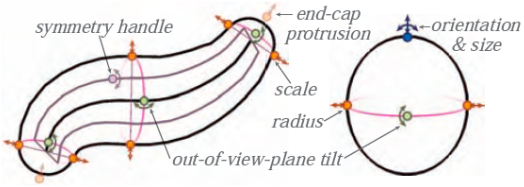

Figure 8. A generalized cylinder and an ellipsoid primitive with handles added to them in [6].

scale handle can be used to change the size of the cylinder. By clicking and dragging anywhere on the outline, the user can enlarge or scale down the primitive on that point. The symmetry sheet handle adjusts a plane inside the primitive, which is used e.g. for mirroring along it with appropriate annotations as explained later. With the end-cap handles, the user can adjust the protrusion of the caps at each end of the cylinder. The ellipsoid primitive also uses the out-of-image-plane handle, but does not provide any other attributes. In Figure 8 the different available handles are added to the two primitive types.

To complete a consistent 3D model, the user is able to create annotations in the image. These are used to adjust symmetry and consistency among the primitives. The connection curve annotation allows the user to connect two primitives. A mirror annotation can be used to create a copy of a primitive as a reflection along a symmetry sheet of another primitive. For example, this can be useful to create arms and legs of a character. With alignment annotations it is possible to align one or more primitives with respect to a symmetry sheet of another primitive. Same-tilt annotations are used to assign the same out-of-image-plane tilt to several primitives. The last annotation is the same-scale annotation, which marks two or more primitives to have the same scale. The primitive handlers and these annotations together are sufficient to create 3D models on top of existing 2D sketches.

A user study of the 2D-to-3D modeling system consisted only of seven people. Three of those already had 2D modeling experience, and five users had already worked with 3D model manipulation systems. As a result of the study, the authors describe that all users were able to create 3D models after 15 minutes of training, but all familiar to 3D manipulation software were very slow at the task and felt uncomfortable with the system. After time they got used to the new concepts.

Beside of the fact that a user study consisting of seven participants is not a meaningful one, the models created by the system and shown in the paper all have some shortcomings. Because a primitive has to be created for every part of the 3D model, it takes a lot of time to model details of the drawing. The presented models of people all just consist of the body, an arm, and two legs. Some have additional primitives for the feet and the hands, but details depicted in the original 2D drawing, like a face or clothes were not modeled in 3D. If a user wished to do that, he has to add and model many small primitives. Furthermore, the presented 3D models all look very round due to the fact, that the primitives are oval by default, and it is not really possible to make them rectangular.

ILoveSketch

ILoveSketch [2] is a 3D sketching system aimed at professional users, which tries to use the same affordances of creating 2D sketches using pen and paper, and to provide the user an interaction that is like using a physical sketchbook. The system supports 2D curve sketching, and offers a set of 3D curve creation tools. For both, 2D and 3D sketches, gestures may be used to execute basic commands, all of which can be performed using pen and digital graphic tablets.

At first we describe the 2D curve sketching techniques of ILoveSketch. When sketching a simple line, a designer does not usually draw a single line, but draws a very light curve first, and repeats to draw that line with several more strokes. ILoveSketch automatically shows an intermediate curve, that is the average of the hand drawn curves. This automatically created curve can be used by the designer to adjust the line he wants to draw by correcting it with some more strokes. After a time, all curves but the automatically created one vanish, leaving just one final line. This vanishing is made visible to the user by a change of the color of the curve. When the user starts a curve tangential to the previous one, the system connects them both to one single line. If that is not what the user intended, he can undo this action by a counterclockwise roll gesture with his pen on the screen. Normally when using pen and paper, the designer often rotates the paper in order to draw the curves more smoothly. ILoveSketch offers the functionality of rotating the canvas by pressing a button on the graphical tablet, and rotating the image with the active (drawing) hand around a fixed anchor point in the upper left of the screen. When dragging the canvas to or away from the anchor point while holding the button, the user can zoom in and out of the image. Using another button on the tablet, it is possible to pan the virtual sketchbook. Going a step further, the canvas will automatically rotate towards on optimal orientation of the last drawn curve after there has been a short pause after drawing it.

For providing navigation in the 3D space, an object-centric paradigm was chosen. When pressing a button on the tablet, the user can navigate through the 3D model by dragging anywhere on or near the object. There are several options to create 3D curves in ILoveSketch. The first one is to draw two symmetric lines on a given center plane from a single point of view. After the first one is drawn, two lines pointing to the vanishing point appear. The spatial relation of the two drawn curves together with the center plane define the position of them in the 3D space. A second possibility is to draw a curve from two different point of views. Some 3D sketching methods are based on sketch surfaces instead of the creation of two or more lines. A sketch surface is just a 2D plane in the 3D space on which the lines will be drawn. In order to define one, the system offers a context-sensitive axes widget. At first a lasso gesture is performed on a 3D curve, and the curve as well as the point where the gesture was performed get selected. A small axis widget is shown on that position (a small coordinate system, with an x, y and z

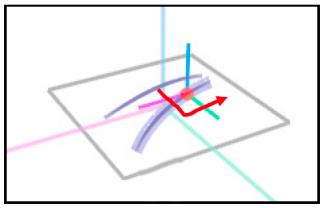

Figure 9. Performing a span gesture for creating an orthographic sketch surface in [2]. The small coordinate system around the red dot is the axis widget.

axis). Two delete it and return to the single-view sketching, the user has to perform a scratch-out gesture on the widget. A span gesture on two of the axes creates a sketching surface in the plane of these two axes as shown in Figure 9. This is also possible on the plane at the origin, in order to select the standard center plane. A flick gesture along one axis extrudes the selected curve along it. The last method is to perform a flick gesture crossing the origin of the axis widget defines a direction for an oblique extruded surface. The best view for drawing strokes on the sketch surface is the one where the surface has the largest visible projection. The systems provides feedback about the quality of the current view by changing the opacity of the current sketching surface. Similar to the auto-rotation after drawing a 2D stroke, ILoveSketch also rotates the 3D view to the best visibility of the sketch surface, after a new sketch surface has been chosen by the user.

The management of the sketches created by the user is based on a sketchbook metaphor as most parts of the system. In order to save a sketch the user performs a flipping-over gesture on the sketch, like one performs when flipping over a sheet on a real sketchbook. To delete a sketch, a tearing gesture is applied. The user can navigate through all of his sketches as if he would be navigating through a paper sketchbook.

The user study of ILoveSketch consisted only of one participant. Because the system is aimed at professionals, the authors invited a professional designer with more then ten years work experience. He created three 3D sketches in about five hours, where two of them were just modeled from existing designs, and the third was designed while working with the system. Because of the use of many pen-and-paper and sketchbook metaphors, the user was able to learn the system quite fast (about one hour), and found the navigation and creation of sketching surfaces quite easy after having worked with 3D graphics before. The part of the system he did not like at all was the automatic rotation of the model after adding strokes to the sketch.

EverybodyLovesSketch

EverybodyLovesSketch [3] was developed as an improvement over the previously presented *ILoveSketch* system. It is also a gesture-based 3D sketching system, but this time it aims not at professional users, but at beginners and amateurs. Although the system is based on a pen-and-paper like interface as its predecessor, it is build so that users without any sketching experience should be able to draw sketches with it. Therefore assisting lines are provided to guide the user with drawing in 3D space.

For navigating through the 3D space, the same navigation techniques as in ILoveSketch were implemented. The axis widget is also used here to select a sketch surface. The only difference is the change of the method to define the extrusion direction for an extruded sketch surface. In EverybodyLovesSketch it is defined by applying an angled flick gesture. The first part of the flick defines the angle between the x-axis and the vector on the horizontal plane of the axis widget. The second party defines the angle between this vector and the z-axis of the widget.

Another, simpler way to define sketch surfaces is the use of *ticks*. Ticks are usually used by professionals when creating perspective drawings to mark special points on the drawing, like the start and end position of a stroke to draw, or the desired intersection point of two lines. In EverybodyLovesSketch a horizontal sketch plane (in relation to the standard center plane) can be selected by simply applying one tick on a 3D curve. The created sketch surface then passes through the selected point. When creating two ticks on two different points on 3D curves, a new vertical plane (angle of 90 degrees to the center plane) passing through both of those ticks is created. By applying three ticks, an arbitrary sketch surface passing through all three points is constructed. In any created plane, the user can display a grid representing the plane by double tapping in the plane. When drawing a linear stroke on one of the grid line, a straight line along the grid line will be added. When drawing an elliptic stroke, the system will add a precise circle.

EverybodyLovesSketch introduces a method for selecting multiple lines by applying a lasso gesture on all of them. The user can then extrude the surface defined by the three lines with a flick gesture, and use this extruded surface to draw on. By applying a tick on one of the boundaries of the sketch surface to copy and paste the selected lines to the active sketch plane.

The system features a new crossing menu which is displayed when a button on the tablet is pushed. Functionality for saving and opening sketches is available in the menu, as well as a new symmetry function. When enabling symmetry, all drawn strokes will be mirrored along one center plane, making it easier to draw most parts of nearly symmetric objects, e.g. cars or aircrafts.

The user study of EverybodyLovesSketch involved 49 high school students attending a design class, and not specializing in art or design. 90% of the students with no experience in 3D sketching were able to draw meaningful 3D sketches within the first three using hours. In general the students liked the usage of the system and said that after working a while with it, it became quite natural to use it. A questionable user study result is that most students answered they were able to learn the system quite fast, while a polled design teacher said that it was not easy for his students to learn

the system.

Evaluation and Comparison

All three presented systems can be used to create three dimensional sketches and models, although the approaches are different, especially between the first and the last two systems. In this section, we want to evaluate and compare the systems.

The 2D-to-3D modeling system is not a real 3D sketching tool. It is designed just to create 3D models on top of 2D images, without any possibility to use sketching methods. By having a look at a primitive on which different handles were applied at several positions, it is hard to distinguish the different kinds of handles, making it difficult to get an idea of the 3D representation of that primitive. While the idea behind the system was to create 3D models out of 2D sketches, it is rather more the fact that the user creates a 3D model with the 2D image just as an underlying view. The 2D image is not really used in the modeling process at all.

As ILoveSketch is aimed at professional users, it is difficult to learn how to sketch in the 3-dimensional space. It is not very intuitive how to create sketch surfaces, and how to draw on them. Because no user with professional background tested the system, it is difficult to make a statement about the usability for non-professionals. When having no experience with the creation of 2D or 3D sketches, it may be hard to learn the different techniques necessary to use the system.

The successor of the system, EverybodyLovesSketch, was then aimed at non-professional users. By using ticks for the creation of sketch surfaces in the 3D space, it is a bit more easy to start 3D sketching. The introduction of the menu helped to be able to turn symmetry on and off, instead of defining that by different strokes. Still, it takes some time to get used to the 3D space in which the user creates the strokes, and it is difficult to distinguish the different surfaces one is sketching on. But nevertheless, EverybodyLovesSketch is a powerful and yet not too difficult system for creating 3D sketches, and remains the most intuitive system of the three presented systems, thanks to easy 3D navigation and the tick-based sketch plane definition.

GRAPH SKETCHING

Most sketch-based systems are aimed at sketching 2D or 3D free-form models as shown earlier. But sketch interfaces can also be used in other areas of application: some systems that aim to support sketching of quantitative graphs (based on scaled coordinate systems) and diagrams were introduced in the last few years.

While it is often the case, that exact graphs based on quantitative raw data are desired, another kind of quantitative graphs is also very important: conceptual quantitative graphs. They do not depend on raw data, but represent simplified theories about the discussed data. In reality they are more important than one might think, especially in economics, when the actual situation should be visualized without real data, or just the trend or progress of some data. In a book about Microeconomics analyzed by the authors of [12], about 20% of the pages actually contained conceptual quantitative graphs.

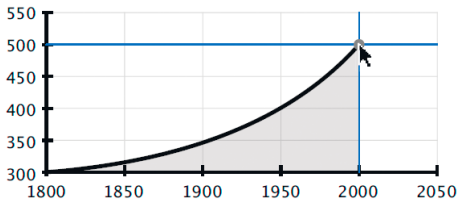

Figure 10. Snapping to the underlying grid when drawing a curve in *Graph Sketcher* [12].

There exist many systems for plotting raw data as graphs, but they do not support creating graphs without such data. A system that merges several methods and techniques mentioned earlier to support users with the illustration and sketching of graphs is *Graph Sketcher* [12]. It was presented in 2009 as shareware, and today is part of the *OmniGraphSketcher* software from The Omni Group. Graph Sketcher allows plotting graphs from data, sketching graphs by hand, and direct manipulation of the created graphs. The system uses already presented approaches, such as *Snap-Dragging* and *Live Paint*.

A quantitative concept diagram (QCD) consists of a set of lines, polygons, points and text labels in a two dimensional space. A line itself consists of points, connected by a straight or curved line, and polygons are filled areas bordered by points. For the basic drawing functionalities, Graph Sketcher uses several standard drawing tools, like a virtual pen, a polygon tool and an editing tool for selecting and dragging existing objects. A constraint-based layout system [7], as presented earlier, is used in the drawing interface of Graph Sketcher, to allow snapping to existing points, and to edit connected parts of the diagrams simultaneously. The coordinate system, in which the diagram will be drawn, is set up by the user. He can change the size of the grid and the labeling of the axes. When drawing new elements, the cursor will also snap to this grid as shown in Figure 10. Furthermore, the position of points, lines, labels and so on are stored in relation to the grid. When the user changes the coordinate system, the whole drawing of the diagram or graph is adjusted to the new underlying grid.

Graph Sketcher also extends the idea of planar map illustration [1]. A coloring is not saved as a new layer, but is stored with the boundaries of the lines that surround that area. This does only effect whole colored areas of the graph, not just filled polygons. Those are stacked like any other object in the This approach makes it much easier to keep the correct coloring after the user changes part of the diagram.

At the time of the publication of their paper, the software was purchased over 1000 times by mainly students, teachers and professional users. The authors have also performed a user study about their system, asking 300 users who downloaded to participate, and receiving responses from 31 of those. All

participants used the software for educational or professional purpose, like economics, engineering or natural science. The user created graphs and diagrams as well as a survey showed, that snapping of points to the coordinate system is one of the most important features of the program suitable for graph drawing.

CONCLUSION

In this paper we presented different systems for supporting digital sketching, including 2D, 3D and graph sketching.

2D sketching tries to catch the affordances of pencil and paper sketching on the computer. However this may lead to problems because often the interface is just gesture-based and thus, the user does not know the possible interactions. With real pencil and paper artists know how to erase with the eraser and know that lines are stronger if they push harder but these interactions are not yet possible with a computer. Hence, they need to learn the interactions with the system for instance to erase a line. This often may lead to the decision to use pencil and paper because it is available quicker and artists do not need to learn how to interact with it.

The beautification which can be made with sketching software can be positive and negative. On the one hand this may lead to quicker sketching because it is easier to prototype for instance diagrams with these beautifications. Furthermore, even users who are not able to draw very well may start to sketch with the help of beautifications. On the other hand beautifications may lead to a result which does not even look like a sketch anymore because it is too polished. Thus, others may think that the quickly expressed idea is already finished.

In the future 2D sketching needs to find a way to catch the affordances of pencil and paper sketching with a minimalistic interface but still making either the interactions so intuitive that the user can easily work with the system or by providing feedback to the user which interactions are possible. Furthermore, the border between a too polished looking sketch and helping the user will still be a difficult problem.

In the area of 3D sketching systems, there exist few good and usable systems, like the presented EverybodyLovesSketch. The main challenge is to make it easy to draw lines in 3D space, which is displayed on a 2D interface (the computer screen). Because 3D sketching is not possible with pen and paper, no affordances from real life can be used to support the user. Therefore, drawing of 3D curves is the most difficult part in all presented systems. A possible solution would be to use 3D environments, like a virtual reality workbench, in order to get rid of having to select a sketching plane on which to place the strokes. There is still a lot of research to be done in this area of application.

For supporting users in sketching of graphs and diagrams, the presented Graph Sketcher system is one of the few systems that were developed in this area of usage. It is very well suited for that task and provides everything one needs for creating quantitative concept diagram. The system was quite successful when it was available as shareware and the commercial OmniGraphSketcher software is build upon it.

REFERENCES

1. P. Asente, M. Schuster, and T. Pettit. Dynamic planar map illustration. *ACM Trans. Graph.*, 26(3):30, 2007.
2. S.-H. Bae, R. Balakrishnan, and K. Singh. Ilovesketch: as-natural-as-possible sketching system for creating 3d curve models. In *UIST '08: Proceedings of the 21st annual ACM symposium on User interface software and technology*, pages 151–160, New York, NY, USA, 2008. ACM.
3. S.-H. Bae, R. Balakrishnan, and K. Singh. Everybodylovessketch: 3d sketching for a broader audience. In *UIST '09: Proceedings of the 22nd annual ACM symposium on User interface software and technology*, pages 59–68, New York, NY, USA, 2009. ACM.
4. E. A. Bier and M. C. Stone. Snap-dragging. *SIGGRAPH Comput. Graph.*, 20(4):233–240, 1986.
5. D. Dixon, M. Prasad, and T. Hammond. icandraw: using sketch recognition and corrective feedback to assist a user in drawing human faces. In *CHI '10: Proceedings of the 28th international conference on Human factors in computing systems*, pages 897–906, New York, NY, USA, 2010. ACM.
6. Y. Gingold, T. Igarashi, and D. Zorin. Structured annotations for 2d-to-3d modeling. *ACM Trans. Graph.*, 28(5):1–9, 2009.
7. M. Gleicher and A. Witkin. Drawing with constraints. *The Visual Computer*, 11:39–51, 1994.
8. T. Igarashi, S. Matsuoka, S. Kawachiya, and H. Tanaka. Interactive beautification: a technique for rapid geometric design. In *Proceedings of the 10th annual ACM symposium on User interface software and technology*, UIST '97, pages 105–114, New York, NY, USA, 1997. ACM.
9. M. Nijboer, M. Gerl, and T. Isenberg. Exploring Frame Gestures for Fluid Freehand Sketching. 2010.
10. A. Panchal. *sketchisland.com*, 2010.
11. T. Sezgin, T. Stahovich, and R. Davis. Sketch based interfaces: Early processing for sketch understanding. In *ACM SIGGRAPH 2006 Courses*, page 22. ACM, 2006.
12. R. Stewart and m. schraefel. Graph sketcher: extending illustration to quantitative graphs. In *CHI '09: Proceedings of the 27th international conference on Human factors in computing systems*, pages 1113–1116, New York, NY, USA, 2009. ACM.
13. I. Sutherland. Sketch pad a man-machine graphical communication system. In *Proceedings of the SHARE design automation workshop*, pages 6–329. ACM, 1964.