Easydrawer - A 2D Geometric Construction Tool

Thomas Weiner*
534 / 0525190
Bachelor Thesis

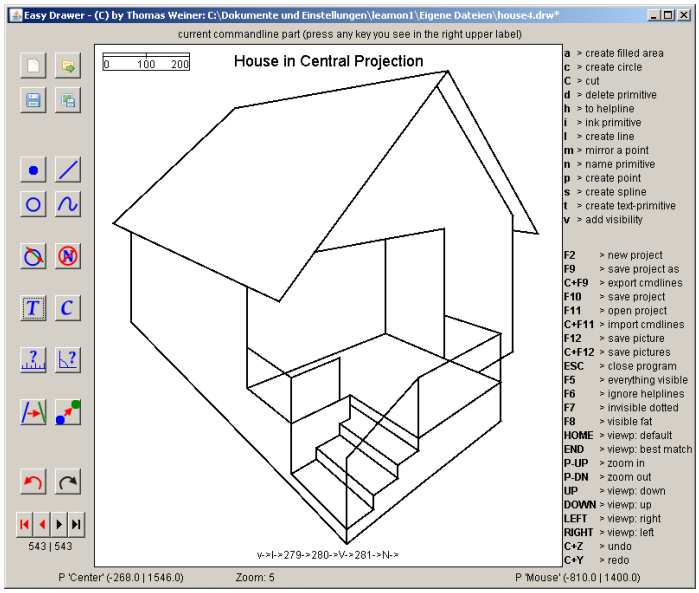

Figure 1: Easydrawer - User-Interface Demo.

Abstract

Easydrawer is a graphical computer application which helps users to draw 2D constructions as they would do on a sheet of paper using a pair of compasses, a ruler and a pen. Easydrawer can significantly increase the productivity of geometrical drawers and help in making the construction process transparent to persons who are currently trained in technical drawing.

The application comes with some features which specially qualify it for educational use. They are mainly: colorization of primitives, gradually playing back creation-history, entering author's comments for each construction step, hiding or highlighting construction parts and giving objects a name. Users are able to directly use any paper-construction know-how with this tool. This means that users will not forget how to draw on a sheet of paper.

In addition, Easydrawer supports geometric drawers when it comes to quality (exactness in distance and angle measurement, calculating intersections and completely dirt-free constructions), working speed and simplifying their job (blending-out uninteresting parts of the construction or coloring primitives, for example).

To cover all constructions which are possible on a sheet of paper, the following primitives and operations are integrated:

- Primitives: points, lines, circles, text (as a primitive or as description of other primitives), splines (for free-hand parts as they are needed to draw ellipses or parables, for example) and areas (for shadow simulation).
- Operations: intersection of two lines, line and circle or two circles, changing visibility mode of primitives and coloring primitives.

Keywords: geometrical drawing tool, ruler-and-compass construction, point, line and circle drawing, 2D constructions, intersecting lines and circles

1 Introduction

Almost everyone begins to train his power of three-dimensional imagination with the help of a sheet of paper and a pen. In the modern computer age, geometrical drawing on a sheet of paper will, of course, play a more and more tangential role except in some niches; for example studying or constructing fast drafts.

However, in technical drawing on a sheet of paper, various problems can occur, especially after many construction steps. Geometers have developed solutions to solve or reduce such problems. Some of them are listed in Table 1.

Easydrawer exactly comes in at learning of technical drawing and solves all problems listed in Table 1 (except for very large constructions, which are not in the scope of this application). This software is a result to the following conclusions:

- Some of the stated problems will never be completely solved for ruler-and-compass constructions on a sheet of paper. This applies to "confusing line clutter" and "dirt on paper" which are impossible to prevent when the construction reaches some complexity. "Loss of exactness" is not neccessarily given when the geometer works exactly.

*e-mail: tw@attingo.com

Table 1: Ruler-and-compass construction problems.

Problem	Solution / Reduction
confusing line clutter	construct thin help-lines, highlight important parts, name primitives
dirt on paper	draw only really essential stuff continually clean rulers, fingers cont. blow dirt off the paper
loss of exactness	be very exact use drawing boards use sharpened pencils

- All of these problems can be solved by state-of-the-art computer software. But this existing software is either not as low level centered as is needed or it is not easy to learn or not comfortable enough to work with.
- Geometrical drawing by hand should not be eliminated because it develops skills with ruler, compass, pencil and paper which would be irreplaceably undeveloped otherwise.

The main goal of this application is to solve all these problems, to be easy to learn, comfortable to use and to be low level centered in order to stay exchangeable to hand-made constructions. This software does not attempt to replace drawing on a sheet of paper. It tries to extend it.

Handling Easydrawer is easy, the learning curve in the use of this application is short and the course of construction is not abstracted. This is why the differences to hand-made drawing, when compared to the way of construction, are negligible. As a result, users can apply their given know-how from drawing on a sheet of paper and simply port it to Easydrawer and vice versa.

Furthermore, constructing in Easydrawer is, except for some simple examples, completely reduced to keyboard. The application introduces a command-line-configuration-file which allows users to flexibly administer the dynamic command-lines.

The possibility to define help-lines and to switch between different visibility modes allows users to get distinct views of the construction with different parts highlighted. Other abstraction layers known from common graphics packages, like blendable layers, are not included. They would easily remove construction complexity in very large projects. On one hand, Easydrawer is not developed to handle large projects. On the other hand, this would bring in additional complexity to the user interface and have a bad influence on the learning-curve. Furthermore, when someone is new to technical drawing, he/she will most likely not start out with extremely large projects. This is why the missing possibility to handle increasing complexity is not relevant.

The results of the user-test uncovered two main things:

1. The participants reach an extreme improvement of construction efficiency in comparison to the beginning of their work with Easydrawer.
2. They reach a completely new level of exactness compared to hand-made drawings when provided the same amount of time.

An additional advantage of Easydrawer is being platform independent and runable either without an installation phase or by providing a very simple one. The current version of Easydrawer needs no installation. Special installation of additional packages is avoided.

Easydrawer uses Java because it is best suited for the need of platform independency and provides a built-in graphics library (AWT) which perfectly fits for managing the graphics output. AWT is neither as fast, nor are the resulting images comparable toDirectX or OpenGL renderings because of missing features like direct use of graphics hardware or anti-aliasing. This is no real problem because Easydrawer is not intended to be used in a professional environment where beautiful and aesthetic results need to be presented.

It is worth mentioning that being platform independent and having no installation phase was vital for realizing the user-test. The school computers did not allow installation of any software.

2 Theory

This section covers most of the theory which was needed to implement Easydrawer's functionality. The first part of this section, "Ruler-and-Compass Construction", deals with general information about technical drawing which directly leads to Easydrawer's functionality range and special look&feel. The next part illuminates "Easydrawer's Visualization Support", covering colored primitives, help-lines, line thickness, areas and splines in detail. The theory section concludes with the "Keyboard Input System" part which clarifies Easydrawers command-line input interface.

2.1 Ruler-and-Compass Construction

Definition of "constructing" in technical drawing introduced by Euclid:

Drawing an exact image of a figure using suitable instruments under specified conditions.

In a more detailed form, suitable instruments are: the idealized ruler (being of infinite length, having no markings on it and only one edge) and a pair of compasses only. Exact means no construction step is valid where anything is "measured" or "compared" only by eye. Comparing, for example, a length or an angle from reference objects (like a ruler with markings on it would be) is not as exact as needed and, because of this, does not count as solution.

Why do we need these restrictions? They separate the field of exact technical drawing, which is also used to prove mathematical theorems, from the wide and unprecise field of painting which considers viewers satisfaction as the main goal. These restrictions lead to the following basic constructions (exactly taken from [Wikipedia 2010]) also shown in Figure 2:

- Creating the line through two existing points.
- Creating the circle through one point with centre another point.
- Creating the point which is the intersection of two existing, non-parallel lines.
- Creating the one or two points in the intersection of a line and a circle (if they intersect).
- Creating the one or two points in the intersection of two circles (if they intersect).

As a prerequisite for these constructions, Easydrawer needs to implement its most basic type, the point. This leads to the following additional operation, which is needed to perform any of the previous constructions:

- Creating the point at any position on the plane.

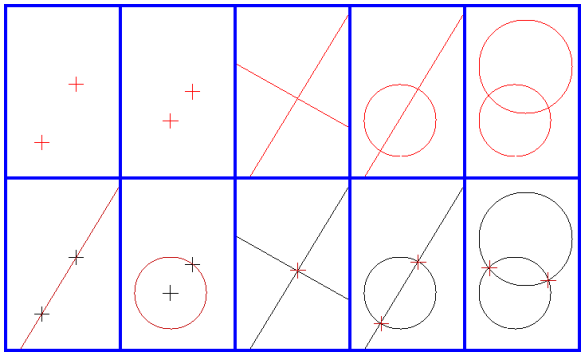

Figure 2: Basic constructions [Wikipedia 2006].

This point construction does not soften the rules described above as long as:

- unlimited point creation is only used at initialisation phase of the construction (following the particular problem specification) or
- when just "a" point is needed to go on with construction (this point's position must be freely selectable)
  - compare pointing anywhere on the plane with closed eyes (allowing some simple restrictions as "the resulting point must not lie on a specified line/circle").

These construction methods enable users to do everything except:

- "squaring the circle", "doubling the cube", "trisecting the angle" or other famous tasks proven to be impossible to be solved when sticking to strict ruler-and-compass construction,
- colorizing constructed areas (shadow casts),
- interpolating a curve running through a set of predefined points (splines),
- giving primitives a name or writing text anywhere on the sheet,
- specifying different thickness of primitives, and
- coloring the primitives.

To be able to evaluate constructions, the following two methods are implemented. The existence of these methods does not weaken the construction process unless they are only used to evaluate lengths and angles:

- Measure the distance between two existing, non-equal points.
- Measure the angle between two existing, non-parallel lines.

The following additional constructions stick to strict ruler-and-compass construction but make life much easier for users:

- Creating the line specified by the starting point and its direction is orthogonal/parallel to another existing line.
- Creating the line specified by the starting point and its direction is parallel to x-axis or y-axis.
- Creating the circle specified by the center point and another circle's radius as this circle's radius.
- Creating the circle specified by the center point and the distance of two existing, non-equal points as radius.
- Creating the circle specified by the center point and the distance of an existing line and a point, which does not lie on this line, as radius.
- Creating the mirror of an existing point corresponding to project origin, x-/y-axis, another existing point or the existing line which does not go through the specified point.

Even the old Greeks encountered problems, which now are known to be unsolveable, when sticking to these basic constructions. Taking into account that a modern software needs to enable users to get around these restrictions, certain additional constructions are introduced to Easydrawer. However, changing the construction's contents after creation using the "move point" and "edit command-line" tools produce the best results when users stick to ruler-and-compass constructions in as many cases as possible.

All following additional constructions soften the possibility to create exact geometrical images, which is guaranteed by compass and straight-edge constructions only:

- Creating the line specified by the starting point and the angle (radians or degree) either from horizon or from another line.
- Creating the circle specified by the center point and the radius.
- Creating the spline[1] (Catmull-Rom) from at least three different existing points.

The following features complete the applications functionality range:

- Creating the text-primitive at a specified position.
- Creating the area-primitive enclosed by a specified border (See section 2.2.1).
- Specifying visibility (hidden, normal, visible) for all primitives and for all parts of a primitive (See section 2.2).
- Naming a previously created primitive.
- Converting a previously created primitive to a help-line or backwards to a normal primitive.
- Inking a previously created primitive.
- Deleting a previously created primitive.
- Giving the project a title.
- Adding an author comment to the current screen.
- Toggling the visibility modes.
- Enabling/Disabling visible names and length legend.
- Moving and zooming the viewport.

Meta functionality, partly known from other graphics applications, is provided by the following commands:

- Saving/Opening projects.
- Saving the current screen to a picture or saving all screens to continuous pictures.
- Exporting/Importing command-lines.
- Browsing the creation history (screens).
- Undo/Redo functionality.

[1] See section 2.2.2 for more details on splines.

2.2 Easydrawer's Visualization Support

In general, a geometer's work can be split into two main parts:

1. There is the construction part, where all the line and circle constructions and calculation of intersection points are performed. For more information on this topic, see section 2.1 (page 2).
2. They need to keep a visually meaningful end-result in mind since they most often want to make some issues clear to the viewer. This section deals with Easydrawer's visualization support for geometers.

The lowest level of visualization geometers use in hand-made drawings are different line thickness and style. When they need a help-line to be able to get forward to the next construction step, they will draw this help-line very thinly in order to not disturb the resulting image in the end. Another example is when technical drawers give an architectural overview. They will either simply ignore all edges on the backside or draw them dotted to illustrate where they are. Simultaneously, edges in the foreground are painted in a thicker style to make clear that they are part of the end result. Easydrawer fully supports these basic visualization concepts.

A fundamental feature to achieve these thin construction lines and reduce complexity is the help-line concept. Each primitive in Easydrawer can be toggled between being a help-line or a normal-style primitive. Normal-style primitives in general are black or have a userdefined color. Help-line primitives are displayed in gray, which tends to look thinner than black and indicate less importance. Help-lines are only drawn when Easydrawer is in the "everything visible" mode (available by pressing `F5`). In the other visibility modes, help-lines are simply ignored. This feature can be used to reduce complexity by showing only what is actually needed in the work progress. See Figure 3 for a demonstration of the help-line feature.

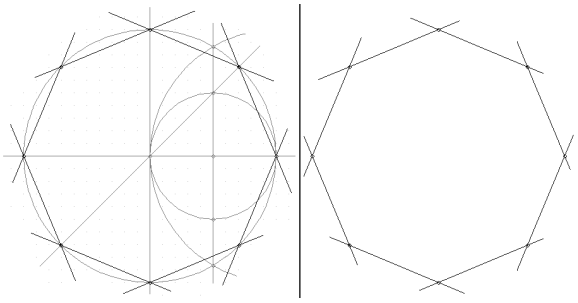

Figure 3: Help-line demo - left: "everything visible"-visibility mode, right: "ignore help-lines"-visibility mode.

Another basic visualization feature comes with Easydrawer's visibility function. Each primitive can be shown in a specific visibility - visible, hidden and normal. Primitives which are marked to be shown visible appear thick when Easydrawer is in the "invisible dotted"- (available by pressing `F7`) or "visible fat"- mode (available by pressing `F8`). Primitives which are marked to be shown hidden appear dotted when Easydrawer is in the "invisible dotted" mode and are not drawn when in "visible fat" mode. Primitives which are marked to be shown normal (they are marked to be shown normal per default) are not shown in "invisible dotted"- and "visible fat" mode. All non-help-line primitives are shown in normal style in the "everything visible"- and the "ignore help-lines" modes. Additionally, it is possible to only mark parts of lines, circles and splines to be visible, invisible or normal. The parts can be separated by points which must lie on the surface of the primitive. You can see some Easydrawer constructions which make use of this visibility concept in Figure 4.

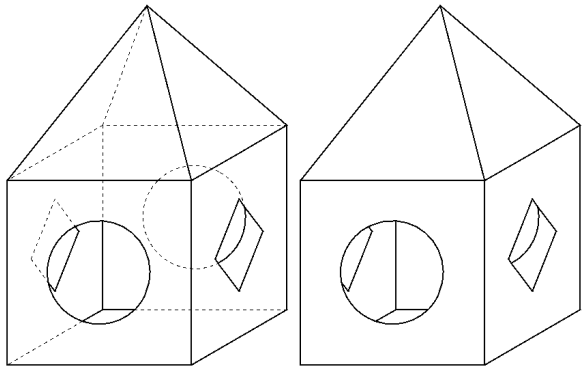

Figure 4: Visibility demo - left: "invisible dotted", right: "visible fat".

Geometers sometimes use colors to put even more importance on certain construction steps. Easydrawer implements this feature and allows users to paint all primitives in either

- predefined colors, defined in the "`config.properties`"-file or
- userdefined colors, entered by hexvalues.

The default colors specified in the "`config.properties`"-file are red, green, blue, black, white, gray, yellow, cyan, magenta, orange and pink. Userdefined colors are entered in hexadecimal style with a preceeding `#`-symbol. For example, either the text "`#00ff00`" or "`green`" changes the given primitive's color to green.

Users are free to enter new predefined colors in the "`config.properties`"-file in the following style:

```
colors.colorname = R,G,B.
```

`colorname` can be replaced by any name which is not already defined in the properties-file. `R`, `G` and `B` specify the red-, green- and blue parts of a color and each value is allowed to range from 0-255. The yellow color definition, for example, looks this way:

```
colors.yellow = 255,255,0.
```

2.2.1 Areas

A more complex primitive, which is only needed for visualization purposes, is the area. Areas mainly enable users to ink regions which are surrounded by other primitives. Geometers specially need this functionality for tasks like constructing images of 3D scenes with shadow casts, along with other reasons.

In Easydrawer, an area can be surrounded by lines, circles, splines and a combination of them. These surrounding primitives are called `border primitives`. Area definition is programmed in a very flexible way and therefore supports (nearly[2]) all possible combinations of border primitives. The only prerequisites are: each border primitive and each intersection point between lines $\cap$ circles and two circles must already exist when area creation is initialized.

A triangle is the simplest possible area because it has only lines as border primitives. Lines do not need additional information to form the border. See Figure 5 for a triangle example.

[2] see page 6 for the limitations.

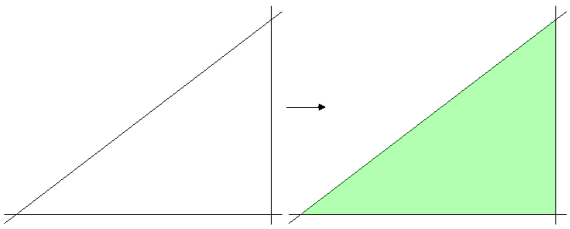

Figure 5: Area demo - Triangle.

The situation gets a little more complicated when creating an area with more than three border lines. There are two ways an area creation could be interpreted in a "four border line"-situation, as Figure 6 reveals. Situation **A** shows that the whole area can be meant. This interpretation leads to a concave polygon. Situation **B** shows that the smaller area can be meant as well and therefore leads to a convex polygon. Easydrawer solves this problem by taking the order of the border primitive selection into account. This way Easydrawer elegantly (but not perfectly performing) gets around the area limitations most softwares have, as Hearn and Baker state on page 123 [Hearn and Baker 2004, 1994, 1986]. It does not care about convex or concave polygons, neither is it limited to "standard-" or "simple polygons" as Hearn and Baker call "polygons with no crossings" on page 124 [Hearn and Baker 2004, 1994, 1986]. For any curved line segment between two points, after constructing an appropriate Catmull-Rom spline polynom, the corresponding y-value for each x-coordinate step is calculated. Finally these created points are connected with straight lines that are usually so small that they are not visible to the user. The arrows in Figure 6 describe the different order of border primitive selection which lead to situation **A** or **B**.

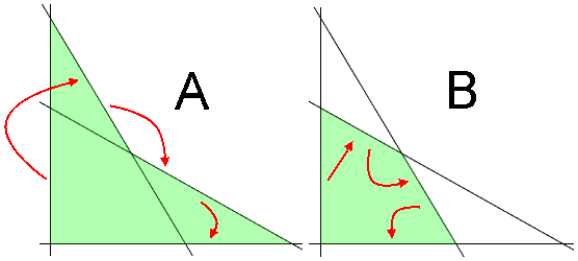

Figure 6: Area demo - 4 lines.

The border primitive combinations for area creation can be classified in the following groups:

- no intersection - circle alone,
- maximum 1 intersection - line ∩ line,
- maximum 2 intersections - circle ∩ line and circle ∩ circle,
- maximum can be more than 2 intersections - spline ∩ any other border primitive.

Figure 7 explains how this classification is obtained. It shows all the border primitive combination possibilities to form an area out of one or two primitives. One can see the circle is the only primitive in Easydrawer, which encloses an area all alone. At least three lines are needed to form a valid area. Furthermore, splines can form a valid area when at least one additional primitive is used to close the border. Finally, Figure 7 shows how many intersections (red numbers) can occur when intersecting these primitives. The number of intersections and the shape of the affected primitives influences how many different area possibilities can appear. For example, the circle-only situation (C) has no intersection and therefore only one area possibility exists. Circle ∩ Line (CL) and Circle ∩ Circle (CC) may have two intersections and therefore the CL-case has two area possibilities (compare to Figure 8) and the CC-case has seven area possibilities (left part only, right part only, center part only, left and right part, left and center part, center and right part, left and center and right part). Splines may have more intersections depending on the number and position of their control points and the relative position to the other primitive. The SC-situation for example has 4 intersections and produces 15 different area possibilities (a, b, c, d, ab, ac, ad, bc, bd, cd, abc, abd (this is the circle only), acd, bcd and abcd).

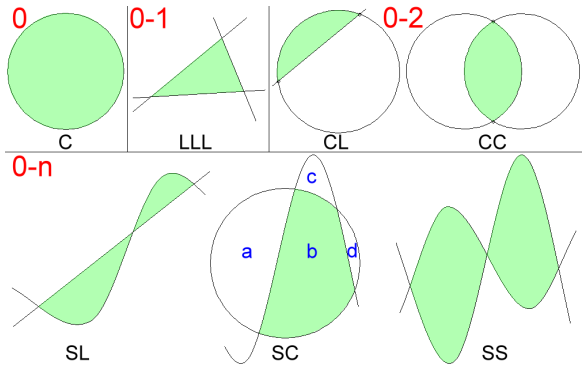

Figure 7: Border primitive combination possibilities to form an area out of one or two primitives.

When only having lines as area borders, it is sufficient to take their specification order into account to get a valid area. When allowing circles to be part of the area border, the specified area is underdetermined. For example, take a look at the "circle ∩ line"-situation shown in Figure 8. Let's assume that the specification order would be adequate. Easydrawer would get the following information: "line" → "circle". So what does the user mean: CLa or CLb? It is impossible to tell with so little information.

Easydrawer solves this problem by asking which direction to go on at a specified intersection point. So the following information is obtained: "line" → "circle" → "first intersection point A" → "second intersection point B" → "clockwise". This information is interpreted in the following way: the area border is defined by the "line" until the "first intersection point A" is reached, from there the circle is the border and is followed in "clockwise" order until the "second intersection point B" is reached. This interpretation leads to situation CLb. This definition is able to completely describe the border of any area with circles partly making up their border.

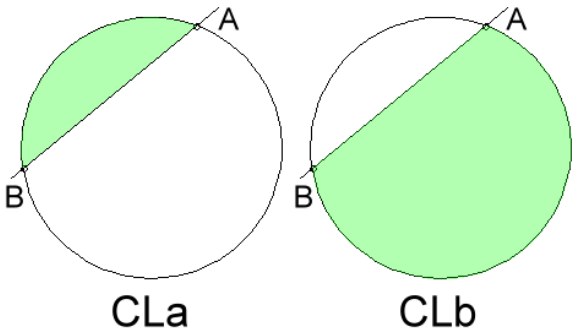

Figure 8: Circle ∩ line area definition.

- CLa could be defined by:
  - "line" → "circle" → "B" → "A" → "clockwise".
  - "circle" → "B" → "A" → "clockwise" → "line".
  - "line" → "circle" → "A" → "B" → "¬clockwise".
  - "circle" → "A" → "B" → "¬clockwise" → "line".
- CLb could be defined by:
  - "line" → "circle" → "A" → "B" → "clockwise".
  - "circle" → "A" → "B" → "clockwise" → "line".
  - "line" → "circle" → "B" → "A" → "¬clockwise".
  - "circle" → "B" → "A" → "¬clockwise" → "line".

When two or more circles are shown, everything works similarly. An example which leads to situation CC shown in Figure 7 is: "left circle" → "lower intersection point" → "upper intersection point" → "¬clockwise" → "right circle" → "¬clockwise".

Working with splines[3] requires a different approach to get all the information neccessary to build the border of a valid area. Splines are, same as areas, intended to be only visual helpers. However, an intersection of primitives is definitely regarded as a construction step. This is one reason, besides the question of sence, why splines can not be intersected with other primitives. An exception to that rule is when an intersection of two primitives, with one or two of them having the type spline, is needed to create an area which is a visual-only primitive as well. As a result, splines can only be intersected with other primitives when it comes to area definition.

Since a spline can have more than just two intersections with any other object, the calculation as well as the selection of the correct intersection point is more difficult than when intersecting the other primitives. Easydrawer makes use of the nature of Catmull-Rom splines to perform this task. They are functions which never have more than one y-value for a given x-value. This implies that the calculated intersection points can be counted from left to right. At first, Easydrawer calculates all intersections in $\mathbb{R}$ and sorts them by x-value. Then, by turning over the correct number of the intersection, the user needs to specify which one is meant.

See Figure 9 for an example of a spline intersecting a line. The red intersection point numbers are respectively used to build up the area. SLa, for example, could have been created in the following way: "spline" → "1" → "2" → "line". SLc on the other hand could have been created like this: "line" → "spline" → "1" → "3".

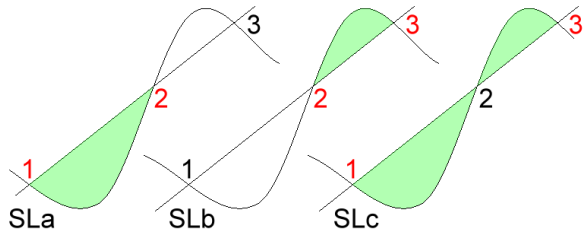

Figure 9: Spline ∩ line area definition.

Areas built up by two or more splines (alone) follow the same rules as spline ∩ line areas. See Figure 7 part SL for an example. "Spline and Circle"-areas however, on one hand need the information about the circle's clockwise or counter clockwise orientation from one intersection point to the next and the spline's intersection numbers, on the other hand. See Figure 7 part SC for an example.

Figure 10 shows a more complex area example to demonstrate the power of this tool. The yellow area border is built up of 5 circle parts, 27 lines and 2 splines.

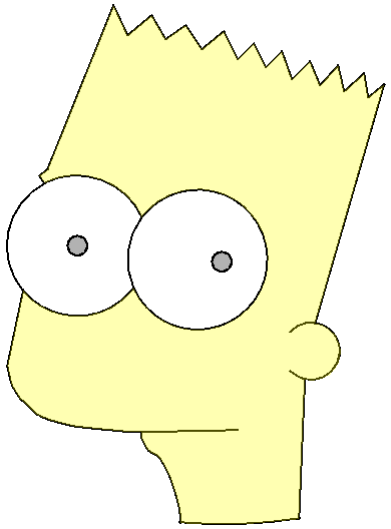

Figure 10: Complex area demonstration - image created with Easydrawer with the help of a drawing of Bart Simpson (`http://images.foren-city.de/images/uploads/78667/simpsons_212.jpg`) at `http://the-simpsons-fan-forum.foren-city.de/`.

Although Easydrawer has a very flexible area definition method, not everything is possible this way. An example for a limitation of this area definition method is shown in Figure 11. Any area with a hole in the middle can not be created naturally. A workaround solution is shown on the right side of Figure 11. When the area is split up into two parts (along the line, in this example) it is possible to create an area for each of the two sides. The downside of this approach is that the result must be handled as two different areas.

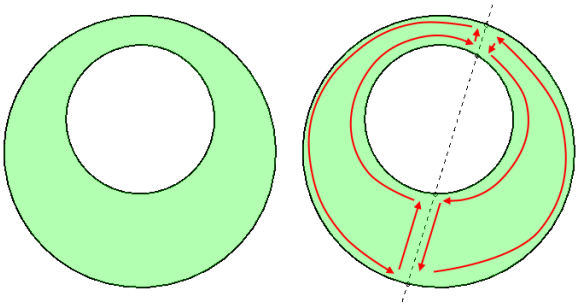

Figure 11: Area with hole in the middle and solution.

2.2.2 Catmull-Rom Splines

Being able to construct circles, lines, areas (for shadow casts) and by naming and coloring these primitives, Easydrawer supports everything neccessary for completing each design process. Specially all parts being widely regarded as technical drawing. In practise, geometers and architects often draw ellipses, other

[3] Details about Catmull-Rom-Splines: Section 2.2.2 (page 6).

conic sections or more general curved surfaces. Figure 12's leftmost graphic shows an ellipse with the constructed helper-circles ("Scheitelkrümmungskreise" in German) and the ellipse-surface points in between. After having constructed enough surface points, geometers connect these points by approximating the ellipsis curvature. Because of this demand, users need a possibility to connect points in a natural looking way. From this point of view, this action can be called "creation of an effect" on top of the final construction. Easydrawer implements Catmull-Rom splines to support this operation, since they have attributes which enable them to create best modulated results. As Figure 12 shows, in the middle and the right section, this type of spline fits quite well for this operation.

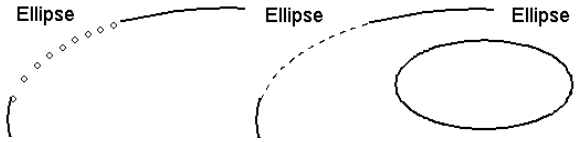

Figure 12: Construction of an ellipse using splines to form the shape.

Splines were initially developed to fulfill requirements of aircraft and shipbuilding industries. In the late 1950ś and early 1960ś this idea was picked up by several automobile body designers; de Casteljau, Pierre Bézier or de Boor, to mention just a few. All of them refined the spline idea, published papers and created their own spline-sub-types which had better attributes for their specific purposes. De Boor, for example, wrote an article in 1962 on "Bicubic Spline Interpolation" and therefore introduced the "B-Splines" [de Boor 1962]. In 1966 Pierre Bézier wrote an article on "Définition numerique de courbes et surfaces I" which gave the "Bézier Curves" its name [Bézier 1966]. "Bézier Courves" were originally developed by Paul de Casteljau in 1959.

2D Splines in general can be considered to be a mathematical way of describing a natural-looking connection of a given set of points. The most common spline-variation is the cubic spline. Cubic splines are made up of cubic functions which are connected at the given control points. For a specified number of n control points we need n-1 functions to connect all control points. Cubic splines are not adequate for this application because they tend to oscillate too much when switching from one control point to the next. As the ellipse example displays, this application often needs to connect points which lie on a curved segment which has the same curvature sign over more than one or even all control points.

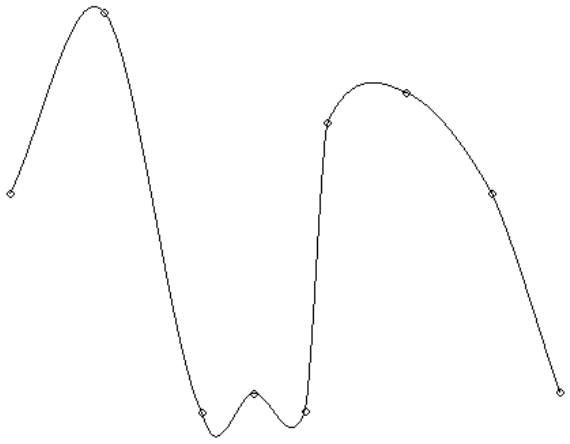

Figure 13: Random points connected using a Catmull-Rom spline.

Finally, Catmull-Rom splines turned out to be a good choice for the given task. They were developed by Edwin Catmull and Raphael Rom and are a subtype of the cubic interpolation splines family. Therefore, their function-parts between each control point are interpreted by a cubic polynom. They have C^1 continuity and therefore guarantee to exactly hit each control point and to be continuous regarding the tangent by doing that as Figure 13 uncovers. The tangent of each control point is calculated by taking the previous and the next point into account. Catmull-Rom splines introduce the parameter τ, often called tension as seen in the tangent calculation formula: $\tau(p_{i+1} - p_{i-1})$. The geometry matrix after [Twigg 2003] looks like this:

$$p(s) = \begin{bmatrix} 1 & u & u^2 & u^3 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 & 0 \\ -\tau & 0 & \tau & 0 \\ 2\tau & \tau-3 & 3-2\tau & -\tau \\ -\tau & 2-\tau & \tau-2 & \tau \end{bmatrix} \begin{bmatrix} p_{i-2} \\ p_{i-1} \\ p_i \\ p_{i+1} \end{bmatrix}$$

See Figure 14 for the effect τ has on a Catmull-Rom spline. The best result for τ, tested on an ellipse part, was 0.5.

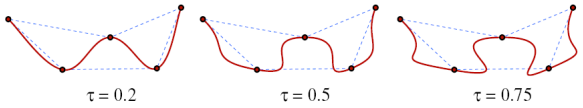

Figure 14: The effect of τ on a Catmull-Rom spline (taken from [Twigg 2003].

In Easydrawer splines can not be intersected with other primitives because they only represent ONE possible solution for a natural looking connection between points. Therefore, no spline alternative can claim to be the real solution for this task. From a geometers point of view, an intersection with a curve which had been approximated by hand will never be regarded as exact or as legal construction part. When creating an area, the intersection of splines and other primitives is automatically calculated because they are needed to form a valid area. These intersections are not returned as points and therefore do not allow drawers to use them in their subsequent construction steps.

2.3 Keyboard Input System

A fundamental feature of Easydrawer is its flexible primitive input and manipulation method. In general users prefer mouse input in graphics software over keyboard usage. The problem in mouse input is on one hand the missing exactness required for a constructing tool and on the other hand the need of a large and complex menu to be able to cover each needed task.

Some softwares solve the problem of exactness with the help of a zoom function. Others use an additional keyboard editing mode. Both solutions are not pure because they need additional steps to fulfill the actually simple task of primitive creation.

Each software suitable for real constructing purposes comes with an enormous menu to trigger all mouse related tasks. This is not easy to learn at first and complicates merely simple construction steps on the long run. Toolboxes with icons and tool-tip-tags or keyboard mnemonics combined with menu elements allow professional users to increase their productivity a little bit after they managed to learn the software.

For these main reasons Easydrawer introduces another concept for all the main construction tasks. In general it only uses keyboard

strokes to trigger actions. Except when it comes to selecting already existing primitives, the mouse is used to ease this task (although it is possible to even do this per keyboard). This reduces the usage of the mouse to a minimum and allows users to mainly concentrate on the construction and not on "where in the menu do I find the desired behaviour". To enable new users to learn this keyboard input system, a small but always visible help dialogue is displayed on the right side, showing "what key can be pressed next". See Figure 15 for a view at this help dialogue without any key pressed yet.

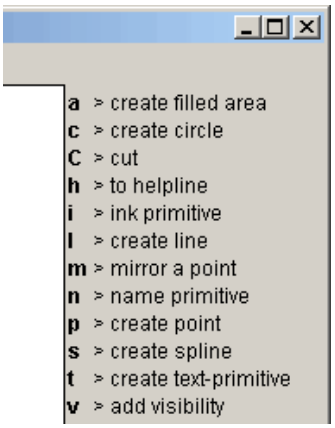

Figure 15: Easydrawer screen - Help Dialogue.

This keyboard input system uses the same key for each primitive or action. For example, one can see in Figure 15 that `p` is used for point creation, `l` is used for line creation and `c` is used for circle creation. Whenever a point can be entered the `p` key is reused. The same applies to the other primitives. A line creation example is:

$$l \rightarrow p \rightarrow 50 \rightarrow -50 \rightarrow p \rightarrow s \rightarrow \textit{point selection with mouse}$$

The `l` initializes a line creation. Then `p` initiates the starting point creation of this line. `50` and `-50` are the coordinates of this starting point. `p` and then `s` tells the system to wait for a point selection by the user. After the user selected the point the line is created.

Since point, line and circle creation happens remarkably often, users rapidly learn Easydrawer's main features. So they are able to automatically remember the keystrokes and therefore increase their productivity enormously. Another positive effect, which is related to this simplified input system, is that users can concentrate on the construction. This direct input system creates a feeling like drawing on a sheet of paper. Users can even see the command-line which resulted in the current screen on the bottom of the panel. See Figure 16 for an example. The next section takes a look behind the scenes of this keyboard input system.

2.3.1 Dynamical Command-Line Construction

The sum of all keyboard interaction possibilities forms a system of command-lines which can be ordered in a tree. Figure 17 shows a not complete part of the line creation command-lines of this tree. All tree elements symbolize a keystroke or a mouse selection. In order to be flexible enough these command-lines are managed and stored in an extra textfile, the `commandlinestrings.txt` file situated in the `img/` directory. A typical entry in this file looks like:

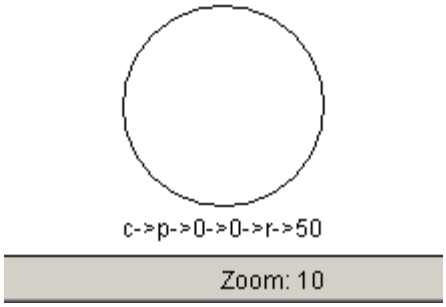

Figure 16: Command-line which resulted in the current screen.

$$\textit{keyboard input} := \textit{function calls} \; ? \; \textit{help dialogue info}.$$

An example for the "keyboard input"-part of an entry in this textfile is:

$$l- > p- > x0- > y0- > p- > s- > P\_ID$$

The corresponding "function calls"-part looks this way:

$$p0 = CADPoint(x0, y0);$$
$$p1 = this.getObjectById(P\_ID);$$
$$CADLine(p0, p1);$$

And finally the particular "help dialogue info"-part contains the following text:

$$\textit{create line}- > \textit{start point}- >$$
$$\textit{specify x}- \textit{coordinate or click on the canvas}- >$$
$$\textit{specify y}- \textit{coordinate (press enter to finalize)}- >$$
$$\textit{end point}- > \textit{enable end point selection}- >$$
$$\textit{select end point}$$

At first readers may recognize that the count of `->` in the "keyboard input"-part exactly matches the count of `->` in the "help dialogue info"-part. This is a prerequisite because the user must be provided a help text at each step of this command-line. So when splitting the "keyboard input"- and the "help dialogue info"-part along the `->` we get pairs looking like that: "`l`" and "`create line`", "`p`" and "`start point`", "`x0`" and "`specify x-coordinate or click on the canvas`" and so on.

Further the combination of "keyboard input"- and "function calls"-part makes use of a simple variables concept. `x0`, `y0` and `P_ID` are variables, which are internally ported from the "keyboard input"- to the "function calls"-part. `p0=CADPoint(x0,y0)` is able to interpret the variables `x0`, `y0` and create a new variable `p0` for storing the result of this function. This uncovers a major difference between the two parts. The "keyboard input"-part can only make use of a fix amount of variables whereas the "function calls"-part is able to create completely new variables. The predefined variable names listed in the `commandlinestrings.txt` file are shown in Table 2.

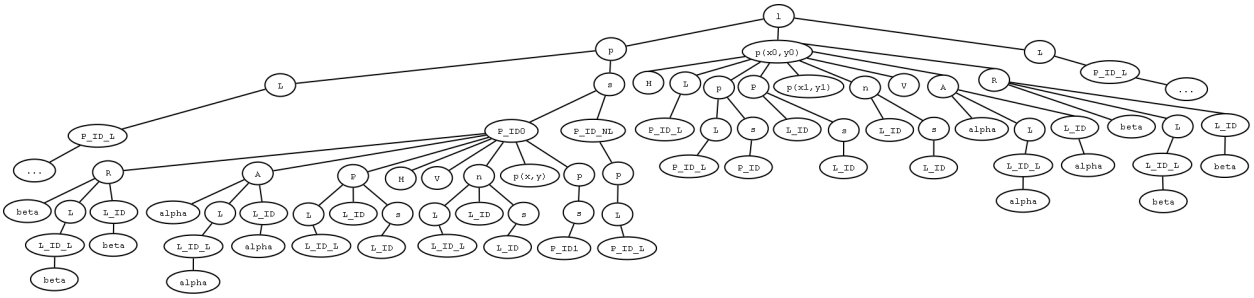

Figure 17: Easydrawer command-lines tree.

Variables noted with lowercase letters specify numerical or text values the user can freely enter. For example, `x0` in `p->x0->y0` can be substituded by any double value specifying the x-coordinate of the new point. Variables containing a `_ID` are noted in uppercase letters and specify primitive ids. Users can only use ids from objects which already are present in the current project. Because of that they are called non-free variables.

Suffices `_L` or `_NL` in non-free variables stand for the last created object of the respective type or explicitely not the last created object of the specified type. The support of command-lines with the `_L` suffix is mandatory to enable users to get in a fast work flow. The following example shows the creation of a horizontal line, with the last created point as start point: `l->L->P_ID_L->H`. In this example the user only needs to press the following keys: "l", "L" and "H".

Variables with suffices `j/J` or `k/K` are reserved for use within loops. Loops are urgent to realize spline-, area- and visibility constructions because they have a variable amount of member-primitives or have to change a primitive's visibility at each point which lies on this primitive's surface. A spline[4] can have three or more member points. An area's[5] border can be built-up from one or more border primitives. A visibility definition can contain more than one surface points where the visibility type may changes. The following command-line shows a simplified loop intended to create a spline:

$$
\begin{aligned}
s->\{&P\_IDJ \;:=\; pj = this.getObjectById(P\_IDJ);\\
&this.userPointsCollector.add(pj);|\\
p->xj->yj \;&:=\; pj = CADPoint(xj,yj);\\
&this.userPointsCollector.add(pj);\}\\
&:=\; CADSpline(this.userPointsCollector);
\end{aligned}
$$

`s` needs to be pressed to initiate the spline creation process. The following `{` indicates where the loop part starts. Then the user can either select an existing point (`P_IDJ`) or create a new point (`p->xj->yj`). The different possibilities are separated by a single `|`. `}` quits the loop mode. Readers possibly recognize that the corresponding "function calls"-part is inside the loop, again separated by `:=`. The loop is repeated until "Return" is pressed. Then the last part is accessed.

Each new command-line is added to a list where all already entered command-lines reside. When the panel needs a redraw, all command-lines from this list are interpreted again, in order to recreate a valid image. An Easydrawer project file contains all command-lines and additional stuff like project title or where the current project center is. When opening a project file, all command-lines stored in this file are simply added to the list.

Undo- and redo-functionality make use of the same issue. The undo function simply removes the last command-line from the list and puts it on a temporary stack. Then a refresh of the painting plane is initiated. The redo function on the other hand pops the object on the top of the stack and adds it to the list again. Afterwards the plane is refreshed as well.

The re-interpretation of the command-lines-array allows to simply edit an already existing primitive and therefore automatically change a complete construction. Figure 18 shows how the "move point" feature can be used to change a finished construction. In this example only point `A` has been changed and depending on this action everything else changes as well. This is a good way to demonstrate how the points in a triangle change when one of the specification points changes. This feature only works when the construction sticks to ruler and compass construction[6] because otherwhise the primitives created after the changed point, do not rely on it.

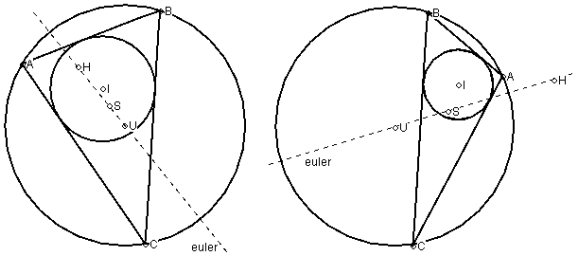

Figure 18: Demonstration of the "move point" tool on a triangle.

When users need to change other parameters of the command-line than just the coordinates of a free point (free points are no results of an intersection) they can use the "change command-line" functionality. This function allows users to select any type of primitive with the mouse. When the primitive is selected, an input box with the command-line, which created this primitive, appears. It allows users to directly change any part of the command-line, according to the following rules:

- the first letter of the command-line stays the same,
- the count of created primitives by this command-line stays the same and
- the count of created primitives by all following command-lines stays the same.

[4] See section 2.2.2 (page 6) for more details on Catmull-Rom splines.

[5] See section 2.2.1 (page 4) for more details on areas.

[6] See section 2.1 (page 2) for more details on Ruler and compass construction.

Table 2: Predefined variables in `commandlinestrings.txt`.

x,y	double
z	double && > 0
x0,x1	double && x0 $\neq$ x1
y0,y1	double && y0 $\neq$ y1
xj,yj	double //for loop variables
alpha	double && $>$ -360 && $<$ 360 //angle: degree
beta	double && $>$ -2*PI && $<$ 2*PI //angle: radian
text	string
intj,intk	integer values in loops
P_ID	any point-ID
P_ID0,P_ID1	any point-ID && P_ID0 $\neq$ P_ID1
P_IDJ,P_IDK	any point-ID
P_ID_L	the point-ID of the last created point
P_ID_NL	any point-ID BUT P_ID_NL $\neq$ P_ID_L
L_ID	any line-ID
L_ID0,L_ID1	any line-ID BUT L_ID0 $\neq$ L_ID1
L_ID_L	the line-ID of the last created line.
L_ID_NL	any line-ID BUT L_ID_NL $\neq$ L_ID_L
L_IDJ	any line-ID
C_ID	any circle-ID
C_ID0,C_ID1	any circle-ID BUT C_ID0 $\neq$ C_ID1
C_ID_L	the circle-ID of the last created circle.
C_ID_NL	any circle-ID BUT C_ID_NL $\neq$ C_ID_L
C_IDJ	any circle-ID
S_ID	any spline-ID
S_ID_L	the spline-ID of the last created spline.
S_IDJ	any spline-ID
T_ID	any text-ID
T_ID_L	the text-ID of the last created text.
F_ID	any filled area-ID
F_ID_L	last created filled area-ID
O_ID_L	last created cutable object (line/circle)
O_ID_NL	not last created cuttable object (line/circle)
V_ID_L	last created object with visibility property
B_IDJ	any object which is a CADAreaBorder
A_ID_L	last created object (ANY type)

The first rule ensures that the return type of the created object is not changed. The second rule checks if the user, for example, changes the command-line:

$$l \rightarrow p \rightarrow 50 \rightarrow -60 \rightarrow p \rightarrow 0 \rightarrow 0$$

to

$$l \rightarrow p \rightarrow 50 \rightarrow -60 \rightarrow X.$$

The first command-line internally creates three primitives. Two points and the line. The second command-line creates only two primitives and therefore changes the total count of primitives in the project. Any command-line which is interpreted later may use the second point which could not exist any longer if the second rule is not executed.

The third rule checks for logical errors which may appear later when users change a command-line. See Figure 19 for a simple example. The line and the big circle are intersected and the resulting intersection points are the centers of two small circles. When the radius of the big circle is made smaller, at some point there is no intersection any more (Figure 19 right). This needs to be intercepted.

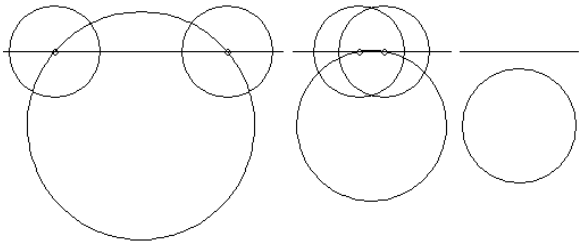

Figure 19: Occuring problems when ignoring the third rule.

2.3.2 Multiple Primitive Selection in Command-Lines

In general, Easydrawer supports users when they enter their command-lines by coloring already selected primitives in green and selectable primitives in red. Additionally all green primitives can not be selected twice. See Figure 20 for an example of this feature.

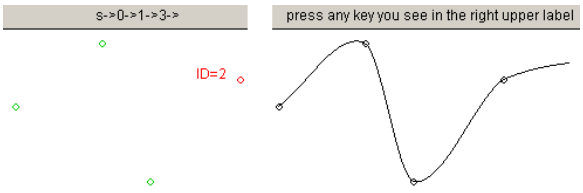

Figure 20: Helper colors during command-line creation.

However sometimes geometers need to select an object twice in order to get the construction done. A common example is constructing the circle, given by its center point and a tangent. See Figure 21 for an explanation of this issue.

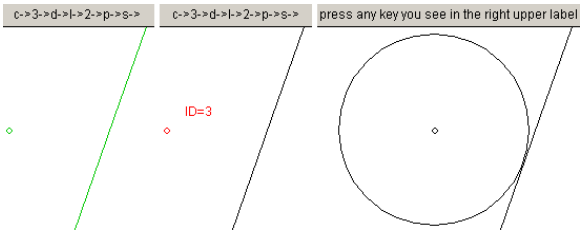

Figure 21: Situation where primitive twice selection is neccessary.

As the command-line-part reveils, the circle creation has been started with `c`, the center point has been selected with `3`, the distance interpretation between a line and a point has been initiated with `d` and `l`, the line has been selected as its primitive-id is `2` and point selection has been started with `p` and `s`. Now the situation conforms to the leftmost graphic in Figure 21. To be able to finish the command-line, users need to select the circle's center point again. Easydrawer allows multiple selection of already selected primitives when it is explicitely requested by the user. This special request can be formalized by pressing any key which is not allowed to be entered at the current command-line position. So in this example the `space`-key has been pressed and the screen immediately switches to the middle graphic. Now the point can be selected again and the circle can be created as it is shown in the rightmost graphic in Figure 21.

2.4 User-Test

The target group for Easydrawer are students, who could use it at school as well at home. Therefore a user-test has been arranged with a group of eight 7th grade AHS-students. Before the test took place the students' teacher had been interviewed about the class' know-how on different tasks.

All of the students attended a geometrical drawing class and were familiar with the topic. Regarding the specified tasks they had to fulfill, they were only used to draw on paper and had never used a computer program. In a more general way the students also used computer aided design (CAD) tools in the 3D sector. The difference between these tools and Easydrawer is the different scope the term "construction" is interpreted. Easydrawer tries to stick to traditional technical drawing as it is needed for constructing on a sheet of paper. The tools the students worked with concentrate more on the design part and allow users to create different three dimensional geometric objects with one click and furthermore to move, rotate and intersect these objects. So for the user-test it could be taken as given that the students are able to do the tasks from the "know-how" part as well as from the "introduced to work with PCs" part.

The first step was the design of a questionnaire, which contained 16 questions on the usability of the program and the students' attitude to geometrical drawing on paper versus the use of a computer. This questionnaire comes along with a handout, which explains the main functions of Easydrawer and introduces four examples which are to be solved by the probants during the user-test. You can see the handout in Figure 24 and the questionnaire in Figure 25 in the Appendix for detailed information.

1. The first example is the easiest, because it was designed to help the probands getting familiar with the program. It contains a step-by-step explanation which tells how to construct a triangle.
2. The second example deals on constructing the in-circle of a triangle and is more complex. There is still supportive information because new functionality is introduced.
3. In the third example the test persons had to construct an ellipse. This example introduced Easydrawer's "mirror-point" function.
4. The last task is the most difficult one and there is no additional information explicitly dealing with Easydrawer. This task's description exactly looks like the specification of an ordinary test example for a paper-construction in a 7th grade.

Before the students were asked to solve the examples a preliminary talk had been held, introducing the program and its possibilities. This talk took 15 minutes and contained information on Easydrawer's objective, how it is to be used and showed a few short examples. Without this information the test would not have been as successful as it turned out to be. Subsequent to this talk the double-sided handout with a description of Easydrawer's user-interface and the four examples was handed over to the participants. Ten minutes before the test time was up, they were asked to fill out the questionnaire.

2.4.1 Questionnaire

The questionnaire which finalized the user-test contained the following questions and provided the answers listed in Table 3:

1. Are you male [ ] or female [ ]?
2. Do you know a better name for the program?
3. When you need to draw something on paper, would you prefer using Easydrawer instead?
4. Can you imagine to use this program in your own projects (outside of school)?
5. Is learning to work with Easydrawer easy?
6. Is the way of userinput to create primitives intuitive/logical to you? If no: what would you change? If yes: What do you like most?
7. Do you have troubles with the intensive keyboard-use this program forces when manipulating primitives?
8. Can you imagine to draw faster/more efficient and more exact with the help of Easydrawer than on a sheet of paper?
9. Could find out in the test time whether this tool could be a learning help for you?
10. Do you miss any functionality you have, when drawing per hand?
11. Do you miss a delete-functionality (since there is an undo-functionality)?
12. Do you think there are different approaches when constructing with Easydrawer or on a sheet of paper. What are the differences (please be exact)?
13. In general, what do you like more - constructing per PC or per hand and why?
14. Would you miss drawing with pen, ruler and a pair of compasses on a sheet of paper?
15. Should computer software be used instead in future?
16. Other comments?

2.4.2 Interpretation of the Questionnaire

All in all, Easydrawer scored off well in the user-test. None of the probands had difficulties to get familiar with the program and to solve the given examples. The minority of the students needed the first example for orientation and getting familiar with the user interface. When they started with the second example they were already able to work faster.

All of the probands described the user input as simple and easy to learn. That militates in favor of the use of Easydrawer in class. After a short period of familiarisation most of the students were prepared for working with the program without hours of practising and introductory lessons. Even probands who described themselves as not so good at the use of computers had no serious difficulties, which proves Easydrawer's usability.

When it comes to the construction of primitives the positive opinions prevail. The students liked the keyboard-explanations on the right side and the show mode. The predominant use of the keyboard is looked upon favorably because of the easy handling and high construction speed specially after some training with the software. There was also a point of critique: some of the probands were missing a possibility to delete primitives (in addition to the undo-functionality).

All of the probands think, that they would draw more efficiently and faster using the PC. Half of them can imagine using the program for their own projects and not only in an academic context, considering the fact that they are students this is a respectable result. The

Table 3: User-Test Answers

Q.	Yes	No	Other/Additional
1			• 1 female • 7 male
2	1	7	• "Helpdrawer"
3	5	3	
4	3	4	• "Maybe" (1x)
5	8		
6	6	2	• "use of keyboard is very good → high construction speed" • "keyboard infos on the right side are very good" • "easy to learn" • "logical but not intuitive" • "use of uppercase letters is disturbing"
7	1	7	
8	8		
9	3	5	• "Time was too short." (1x)
10	3	4	• "Delete functionality" • No comment (1x)
11	4	4	
12	2	4	• "when drawing by hand a line can be constructed without having points" • "when drawing by hand, half of a line can be simply measured" (not true for idealized ruler and compass constructions introduced by Euclid - see Section 2.1 (page 2) for details on this topic) • No comment (2x)
13	1	5	• "by hand, I am not well versed in using PCs." (1x) • "by hand, it was always this way." (2x) • "by hand, because it is easier for me." (2x) • "Equals" (1x) • "Differs" (1x)
14	4	2	• No comment (2x)
15	3	4	• "Why not" (1x)
16			• "Sometimes errors occur, the user produced himself, but did not notice." • "Scale is not visible to user." (included as a reaction)

minority of the probands said that they would prefer the use of the program over drawing on paper.

This result is quite interesting. Combining the fact that all of them think using Easydrawer is faster than painting by hand and the small number of them preferring the program over drawing on paper, leads to the following conclusion. The students like drawing per hand even though using Easydrawer is faster and more exact. This is also reinforced by the result of question 13 as interpreted below. This result is exactly what Easydrawer tries to obtain. The software should be used for learning and afterwards users should be able to draw on both, a sheet of paper or on computers, using the know-how which has been previously acquired. This can not be granted by other software.

When it comes to question 13 (dealing with the future use of paper or computer), some students could not imagine to completely replace drawing on paper with using a computer program. Considering the fact that the probands are used to draw on paper to solve the given tasks, this result is not very surprising. Nevertheless there were three students who declared that computer software should be used instead of paper and pencil in future. The results concerning usablity and efficiency show that the attitude towards drawing with pencil and paper would change in favor of Easydrawer if it would be used at school regularly. Taking a look on the facts and trying to foresee the future: Three students out of eight are highly interested in Easydrawer after one hour of working with the program. What happens after one week or one semester of working with Easydrawer? The number of students favouring Easydrawer will definitely increase when they realize the software's full potential and make use of it.

2.4.3 Opinion of the Teacher

In the run-up to the user-test the students' teacher had been interviewed after a presentation of the program. He was also asked to do the tasks on the handout sheet. Altogether the teacher was pleasantly surprised: "At the beginnig the program seemed to be difficult to use, but I soon got familiar with the functionality since the keys always stay the same." Finally he stated that he could imagine using Easydrawer in his lessons.

2.4.4 Evaluation Process

After the user-test the program was revised and the requested delete-function was implemented. Another addition, finally added to the program, upon the recommendation of the teacher was a scale. During the talk with the teacher he mentioned, that Easydrawer could also be used in lower grades, because of the easy handling. Therefore a second user-test with lower grade students could be performed next year.

3 Results and Conclusion

Easydrawer turned out to be a highly flexible tool which allows technical drawers to create constructions as if they were using a sheet of paper and a pen. This is why Easydrawer could be used in studying of technical drawing by both teachers and students. Teachers could do their constructions with Easydrawer to assist students in understanding and learning. Therefore students could use Easydrawers built-in previous- or next-screen functionality which allows to visualize what happened before or after a specified construction step. Students on the other hand could create their work with Easydrawer and teachers would be able to see exactly what they did by looking at the construction command-lines.

Easydrawer's operating range reaches from small constructions (halfing an angle or creating a triangle and its special points) to middle-size technical drawings (architecture overviews with shadow casts in gnomonic projection or visualizing objects in 1-2-axis). It is not intended to be used for large projects. Therefore it has little support for abstraction layers, changeability of primitives after creation and multiple primitives management to mention just a few reasons. Furthermore, it would be a good idea to use specialized software for appropriate construction problems. For example, when creating objects in 3D space users are encouraged to use software which is specialized to assist users working in 3D space like AutoCAD or others. As such needs end up with complex software not able to support users given know-how from drawing on a sheet of paper, Easydrawer distances from supporting large-scale technical drawings.

The best way to demonstrate Easydrawer's performance is by showing two technical drawings as they would be in real life. They make use of nearly every feature the software supports.

The first example, displayed in Figure 26 in the Appendix, shows the last task from the user-test but has a slightly different specification considering the point's coordinates. This is a 1-2-axis (horizontal and elevation projection) construction of a triangle's in-circle.

1. The triangle's edges are given by A(-137/33/79), B(21/248/288) and C(222/97/174).
2. During the construction process the triangle is rotated along an affinity axis (blue line named "a").
3. The rotated triangle and its in-circle are marked to be invisible in order to be highlighted in "invisible dotted" visibility mode (C) but not to disturb the image in "visible fat" mode (D).
4. The triangle and its in-circle are regarded as the result of this construction and are therefore drawn fat. Any other line and point is marked to be a help-line and therefore only shown in "everything visible" mode (E).

When taking a more precice look at Figure 26 part E, the massive number of help-lines shows the urgent need for the help-line-feature. Part F shows how the drawing looks like when all the unnecessary help-lines are not shown. With this little number of lines on the screen constructing is much easier and needs less concentration.

As Figure 27 in the Appendix informs, the second example shows the front of a house, drawn in central projection. The house is given by its horizontal projection (see **A'**) and its front elevation (see **A"**). The central projection ($\mathbf{A}^c$) is achieved by the following construction steps (attention: the image groups named A and B are specially arranged after construction has been finished to save space, whereas image C remains unchanged):

1. Construction of the images of the vanishing points in horizontal projection (Yuc' and Xuc') and the points O and H.
2. By placing H, we get the horizon which is a horizontal line running in $\mathbf{A}^c$.
3. Since the vanishing points Xuc and Yuc lie on the horizon, we can draw them as well, using normals to the horizon connected with their horizontal projection images Xuc' and Yuc'.
4. The line running through Xuc' and Yuc' is the picture plane, called π'. H is where the view axis intersects π.
5. To get started with central projection these steps need to be performed:
   (a) Construct an intersection of π' and any line in horizontal view, which touches the ground (do not take a roof-edge).
   (b) Create a vertical line running through the intersection.
   (c) Intersect the vertical line with the lengthened ground line from front elevation (this is why the first point taken needs to touch the ground).
   (d) Now connect this point with any of the two vanishing points Xuc or Yuc or both of them, depending on which axis the line, the point lies on, is parallel to. This is the first vanishing line.
6. To bring general points, given by the horizontal projection, to the central projection, the following procedure is used:
   (a) Create line between O and the specified point given in the horizontal projection (**A'**).
   (b) Intersect this line with π'.
   (c) Construct a vertical line running through the intersection.
   (d) Intersect this vertical line with the corresponding vanishing line (which must already exist).
7. All points intersecting π' in horizontal projection, come with a speciality. Any height running through these points is unabbreviated. So for all of these points, the corresponding heights can be connected from front elevation with horizontal lines.

Part B in Figure 27 shows the result of this construction without help-lines. The front elevation is not shown twice because it has no hidden lines. Part C finally describes a more technical view of the drawing in the "ignore help-lines" mode. The massive number of vanishing lines running through the vanishing points Xuc and Yuc shows that this construction nearly reaches the maximum of Easydrawer's field of application. The ever-increasing problem is the correct line selection and the ability to see whether a line really runs through a specific point or if it is neccessary to create a new one. Even at this level, no performance problems can be measured[7].

3.1 Comparison with other Software

When comparing Easydrawer with other tools, the following aspects are taken into account:

- Functionality (additional/less),
- Learnability, and
- Input-methods.

Easydrawer has a very specialized field of application. Therefore finding comparable software is difficult. For the comparison two programs were picked because of their similarity.

3.1.1 Geogebra

Geogebra[8] comes with a huge amount of functionality. In general most construction steps are initiated using the graphical menu at the top of the window. The program provides good graphical response when it comes to selecting existing objects. This form of user input on one hand invites and assists users in taking their first steps with the tool. On the other hand, it is hard to come into a fast working flow because it is neccessary to switch between different submenus of this graphical menu quite often. Figure 22 shows a screenshot of Geogebra. Probably due to this fact Geogebra also supports a command-line input feature, allowing users to enter primitives in parametric form following mathematical conventions. For example, "`g:y=k*x+d`" or various functions like "`cos(x)`" can be used and combined to create lines and other primitives. This feature is very flexible on one hand. On the other hand it is hard to learn this "language" because users do not intuitively know which input will produce which output and it is hard to predict which text input is allowed and which not.

[7] construction done on laptop with 2.20GHz Dual Core CPU and 2GB RAM.

[8] `http://www.geogebra.org/cms/`

In combination the primitive input is easy to understand at first but users can not get faster when they learned the usage of the software. Easydrawer on the other hand fulfills this task but is harder to start out with because it requires more energy to overcome one's inhibition threshold caused by the intense use of the keyboard. Additionally Easydrawer's keyboard input system is very organized, informs users what can be entered as next step and performs semantic checks during input.

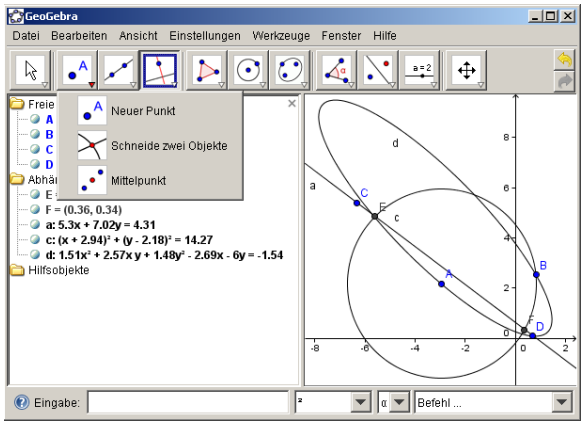

Figure 22: Geogebra's user interface.

Geogebra has more features than Easydrawer. Some of them are creating conic sections as ordinary primitives (so they are really exact and not approximated using splines), using mathematical language to describe a function's graph or the ability to create user-primitives. Concerning user friendly interpolation or approximation of curved segments running through or being controlled by given points, Geogebra offers no user-friendly way as other softwares do by implementing splines.

As conclusion can be stated: Geogebra is exactly the right software for people who want to construct technical drawings once a week. When it comes to speeding up the construction process Geogebra has some (but very limited) potential by using its command-line. Easydrawer can be used by nearly the same clientele. People should take a look when they want to raise their productivity, as long as Easydrawer's functionality is enough for their purposes.

3.1.2 WinCAG

Easydrawer is furthermore compared with WinCAG[9]. Figure 23 shows its user interface. WinCAG also offers a very large amount of functionality but is less organized than Geogebra. For every constructing step the huge menu needs to be searched for the proper entry. Many entries have no fittingly name and it can not be foreseen what they will do. The input system is mostly managed by mouse. WinCAG always uses left mouse clicks for the "`yes`" in answers whereas "`no`" is expressed by a right mouse click. This is a feature which is quite handy to speed-up construction process. When an operation has been selected, WinCAG automatically restarts this operation once the previous one has been finished. The bad thing is: When users try to click on the menu to select another operation, the mouse click is incorrectly interpreted as belonging to the already selected operation and the click most often creates a point at the upper edge of the screen. To do it right, the `ESC`-key should be used to stop the operation mode. The combination of these parameters describing the input system results in a user interface which is hard to use and hard to learn. Easydrawer is intended and implemented always keeping in mind to create an easy-to-use user interface and therefore has no difficulties to outnumber WinCAG when comparing usability.

To cover WinCAG's keyboard support: The tool supports mnemonics for some of the menu items. However, they are not obviously taken and therefore it is very hard to reach a suitable working flow.

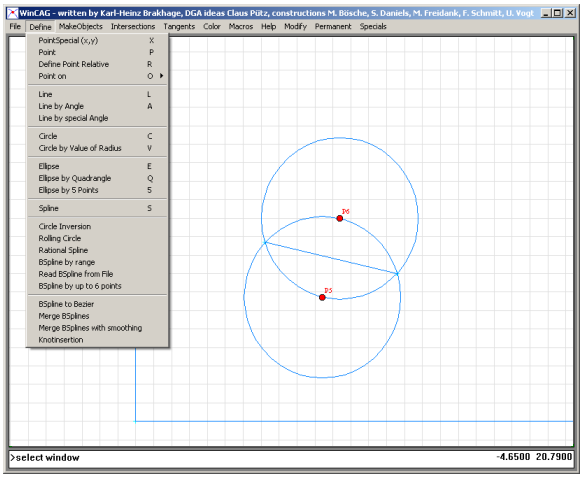

Figure 23: WinCAG's user interface.

WinCAG implements a very powerful and advanced technology to watch and create diashows. This is not possible with Easydrawer.

The software treats ellipses as standalone-primitives and needs no splines to approximate them. It covers many ways to create and handle curves. Some of them are Spline, B-Spline or Bezier-Curve. The only problem with the provided user-friendly ways to create curves is that all of them approximate points and none of them interpolates them. This means the points do not neccessarily lie on the surface of the curve. Finally to help not overloading the screen, WinCAG only shows names of primitives when the mouse is moved over the specific object.

WinCAG is mainly intended as a demonstration tool which allows teachers to produce a construction including a diashow and later exhibit this show to students. Users who want to speed up their creation process should use Easydrawer, when it provides the functionality needed. Generally when the construction is to be emphasized Easydrawer is the better choice, whereas WinCAG should be used when the presentation of technical drawing is more important.

3.1.3 Problems and Future Work

Even though this work mostly recommends Easydrawer, it nevertheless is never-failing. Splines can only have one y-value for each x-coordinate, for example. This limits their use when it comes to painting closed rounded surfaces because one spline needs to be splitted-up into two separate splines.

Another restriction is that similar objects lying on top of each other can not be distinguished. This goes hand-in-hand with the problem that primitives can sometimes not get their visibility marked, because points do not really lie on existing lines and circles.

In the future Easydrawer should provide a makro functionality to automatically repeat certain recorded construction steps. This

[9] `http://www.igpm.rwth-aachen.de/brakhage/Strobl/index.html`

would bring the software into another grade of complexity because users would be able to create their own "primitives". So, for example, the construction of an ellipse could be logged and afterwards replayed on other input primitives to create another similar ellipse. This feature could reduce the time required for performing repeated tasks.

References

BÉZIER, P. 1966. Définition numérique des courbes et surfaces i. *Automatisme XI*, 625–632.

DE BOOR, C. 1962. Bicubic spline interpolation. *J. Math. and Phys. 41(3)*, 212–218.

HEARN, D., AND BAKER, M. P. 2004, 1994, 1986. *Computer Graphics with OpenGL Updated Edition*, third, international ed. Pearson Prentice Hall.

HOUSE, P. D. H. Spline curves. [Online; `http://www.cs.clemson.edu/~dhouse/courses/405/notes/splines.pdf`; accessed 25-November-2010].

TWIGG, C., 2003. Catmull-rom splines. [Online; `http://graphics.cs.cmu.edu/nsp/course/15-462/Fall04/assts/catmullRom.pdf`; accessed 24-November-2010].

WIKIPEDIA, 2006. File:basic-construction-demo.png — wikipedia, the free encyclopedia. [Online; `http://en.wikipedia.org/wiki/File:Basic-construction-demo.png`; accessed 10-November-2010].

WIKIPEDIA, 2010. Compass and straightedge constructions — wikipedia, the free encyclopedia. [Online; `http://en.wikipedia.org/w/index.php?title=Compass_and_straightedge_constructions&oldid=394802031`; accessed 10-November-2010].

A Appendix

Easydrawer Usergruppentest am 23.06.2010

BRG 7 Kandlgasse, Klasse: 7AB

Handout

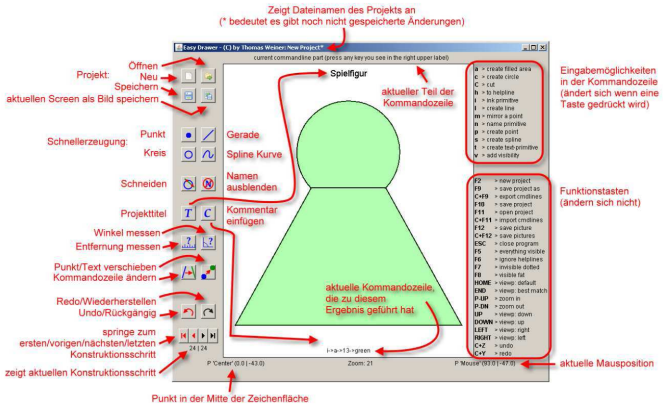

Abb. 1: Easydrawer User-Interface Beschreibung

Tastatureingabemöglichkeiten kennen lernen

1. Beispiel: Dreieck (leichtes Einführungsbeispiel mit Erklärungen)
   Gegeben sind die Eckpunkte eines Dreiecks: **A**(-100|0), **B**(0|-88,4) und **C**(100,5|0).
   Benennen Sie diese Punkte entsprechend der Angabe (n→p→[Punkt wählen]→[Text eingeben][Enter zum Abschluss der Texteingabe]).
   Erstellen Sie Linien, die durch diese Eckpunkte gehen (l→ [Punkt wählen]→[Punkt wählen]).
   Erzeugen Sie eine gefärbte Darstellung dieses Dreiecks (a→ l→[Linie wählen]→l→[Linie wählen]→ l→[Linie wählen][Enter zum Abschluss]).
   Färben Sie die Dreiecksfläche grün (i→a→[Fläche wählen]→[Farbnamen oder Hex-Wert der Farbe tippen z.B.: green oder #00ff00][Enter zum Abschluss]).
   Stellen Sie die Sichtbarkeit der Linien auf **visible** ein (v→l→[Linie wählen]→[Startpunkt wählen]→V→[Endpunkt wählen]→N[Enter zum Abschluss]). Beachten Sie hier, dass die beiden Punkte exakt auf der Linie liegen müssen, die Sie angegeben haben. Wechseln Sie mit **F7** oder **F8** zum Sichtbarkeitsmodus „invisible parts dotted“ oder „invisible parts ignored“ um den Effekt zu sehen. Mit **F5** bzw. **F6** kommen Sie in den Sichtbarkeitsmodus „show all“ oder „helplines ignored“ zurück. Wiederholen Sie die Schritte für die restlichen Linien und für die grüne Fläche (v→a→ L[mit „L“ wird in vielen Situationen im Programm das zuletzt erzeugte Objekt des entsprechenden Typs automatisch ausgewählt]→V).
   Geben Sie dem Projekt den Titel „Grünes Dreieck“.
   Wechseln Sie zum Sichtbarkeitsmodus „invisible parts ignored“ (**F8**), zoomen Sie soweit wie möglich heran, ohne mit einem Teil der Schrift das Dreieck zu verdecken und zentrieren Sie das Dreieck. Probieren Sie hier evtl. den Effekt der „Ende“- und „Pos1“-Taste (das Bild kann mit den Cursortasten verschoben werden).
   Speichern Sie das Projekt in Ihrem Verzeichnis im Unterordner „easydrawer“ unter dem Namen „bsp1.drw“. Das Programm speichert auch die aktuellen Zoom- und Centerwerte und den aktiven Sichtbarkeitsmodus ab.
   Speichern Sie den aktuellen Screen als Bild „bsp1.png“ im gleichen Verzeichnis ab.

2. Beispiel: Innkreismittelpunkt eines Dreiecks konstruieren.
   Erzeugen Sie ein Dreieck aus 3 beliebigen Eckpunkten und definieren Sie, wie im vorigen Beispiel, die Sichtbarkeit für die Linien. (Diesmal keine Fläche)
   Konstruieren Sie nun den Innkreismittelpunkt dieses Dreiecks und benennen Sie ihn mit „I“.
   TIPP: Machen Sie alle Primitive, die als Hilfslinien/-punkte anzusehen sind, sofort wenn Sie sie nicht mehr brauchen zu Hilfsprimitives und arbeiten Sie generell im **F6**-Modus. Dadurch bleibt die Zeichenfläche möglichst übersichtlich.
   Konstruieren Sie den Innkreis. Hier gibt es einen Spezialfall zu beachten: Angenommen der Mittelpunkt eines Kreises und eine seiner Tangenten sind bekannt, dann können Sie mit c→[Mittelpunkt wählen]→d→l→[Tangente wählen]→[Leertaste, um die erneute Auswahl des Mittelpunkts zu ermöglichen][Mittelpunkt wählen] den Kreis konstruieren.
   Anschließend definieren Sie für Innkreismittelpunkt und Innkreis den „**visible**“ Sichtbarkeitsmodus.
   Definieren Sie weiters Sichtbarkeit „**hidden**“ für die 3 Eckpunkte des Dreiecks und wechseln Sie anschließend in den **F7** Modus. (Sie sehen, dass die Punkte hier nun sichtbar sind)
   Optional: Erweitern Sie die Zeichnung um die anderen 3 ausgezeichneten Punkte im Dreieck (Höhenschnittpunkt, Schwerpunkt und Umkreismittelpunkt) und zeichnen Sie die Eulersche Gerade ein. Geben Sie den Punkten die Namen „H“, „S“ und „U“.
   Klicken Sie jetzt auf den Button  und wählen Sie einen Punkt des Dreiecks aus. Bewegen Sie die Maus etwas. Wenn Sie fertig sind, klicken Sie erneut auf die Zeichenfläche.
   Geben Sie dem Projekt den Titel „Besondere Punkte im Dreieck“.
   Speichern Sie das Projekt in Ihrem Verzeichnis im Unterordner „easydrawer“ unter dem Namen „bsp2.drw“. Speichern Sie den aktuellen Screen als Bild „bsp2.png“ ab.

3. Beispiel: Konstruktion einer Ellipse
   Konstruieren Sie eine Ellipse in 1. Hauptlage mit **a=150** und **b=80**.
   Benutzen Sie die Scheitelkrümmungskreise, um möglichst wenige Ellipsenpunkte konstruieren zu müssen.
   Konstruieren Sie die Brennpunkte und benennen Sie sie F1 und F2 um die Ellipsenpunkte konstruieren zu können. Spiegeln Sie die auf einer Seite gewonnenen Ellipsenpunkte mit (m→[Punkt wählen]→Y) auf die andere Seite.
   Benutzen Sie Catmul-Rom-Splines um die konstruierten Ellipsenpunkte zwischen den Scheitelkrümmungskreisen zu verbinden.
   Fertigen Sie die Ellipse so aus, dass im **F8** Sichtbarkeitsmodus nur die Ellipse zu sehen ist. Achten Sie darauf, dass die Ellipse möglichst kontinuierlich verläuft.
   Speichern Sie das Projekt in Ihrem Verzeichnis im Unterordner „easydrawer“ unter dem Namen „bsp3.drw“. Speichern Sie den aktuellen Screen als Bild „bsp3.png“ ab.

Komplexeres Beispiel – 1-2-Achse (als „12achse.drw“ speichern)

Geg.: Dreieck mit **A** (260|-200|100), **B** (20|-40|280) und **C** (180|230|10).
Ges.: Innkreis dieses Dreiecks in Grund- und Aufriss (Resultat: Dreieck und Innkreis).
Tipps: In **F6** konstruieren und Objekte zu Hilfslinien machen, sobald sie nicht mehr benötigt werden, Punkte benennen, Dreiecksseitenlinien so früh, wie möglich **visibility** geben, Affinitätsachse wegen Übersichtlichkeit färben, Primitives in Null-Lage **invisible** machen.

Vielen Dank fürs Mitmachen!
Thomas Weiner

Figure 24: User-test Handout.

Fragebogen – Usergruppentest am 23.06.2010

- Sie sind männlich [ ] / weiblich [ ] ?
- Fällt Ihnen ein passenderer Name für das Programm ein?
- Wenn etwas am Papier zu zeichnen waere, wuerden Sie statt dessen dieses Programm verwenden?
- Können Sie sich vorstellen dieses Programm in eigenen Projekten (außerhalb der Schule) zu verwenden?
- Ist die Benutzereingabe einfach/schnell zu lernen?
- Ist die Benutzereingabe um die Primitives zu erzeugen intuitiv/logisch? Wenn nein, was würden Sie ändern? Wenn ja, was hat Ihnen besonders gefallen?
- Stört Sie die verstärkte Verwendung der Tastatur zur Eingabe von Primitives?
- Können Sie (sich vorstellen) durch Verwendung dieses Programms schneller/effizienter und exakter zu zeichnen, als per Hand?
- Haben Sie in der Testzeit feststellen können, ob dieses Tool für Sie eine Lernhilfe sein kann?
- Fehlt Funktionalität, die bei Konstruktionen mit der Hand manchmal benötigt wird?
- Ist Ihnen eine Möglichkeit zum Löschen von Primitives abgegangen (vorallem weil es eine undo-Funktionalität gibt)?
- Sind beim Konstruieren mit diesem Programm bzw. am Papier unterschiedliche Herangehensweisen nötig? Wo liegen die Unterschiede (bitte genau erklären)?
- Zeichnen Sie generell lieber am PC oder per Hand und warum?
- Wuerde Ihnen das Zeichnen mit Bleistift, Lineal und Zirkel fehlen?
  Sollte in Zukunft immer der PC verwendet werden?
- Sonstige Bemerkungen?

Figure 25: Questionnaire.

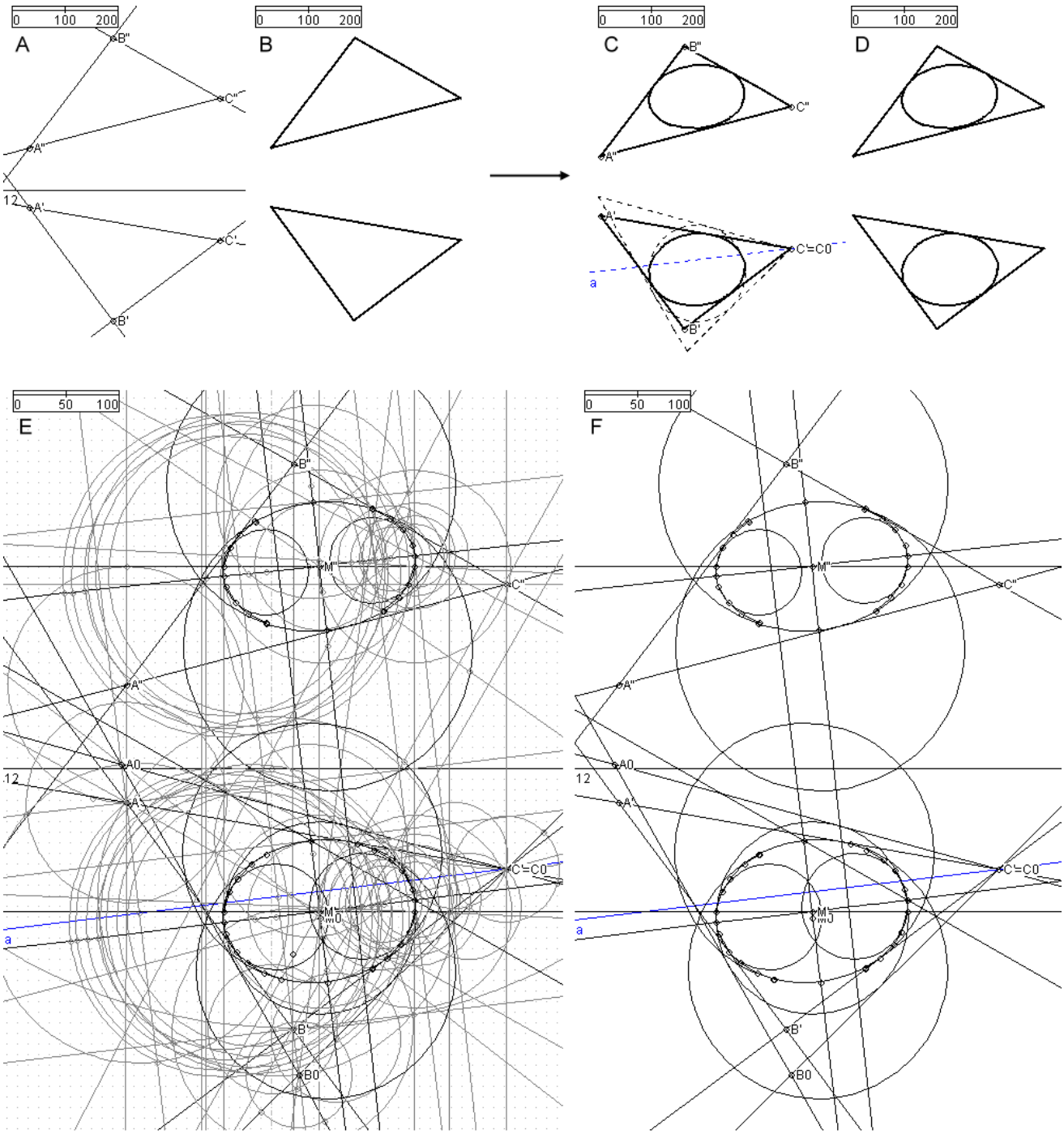

Figure 26: Construction of a triangle's in-circle.

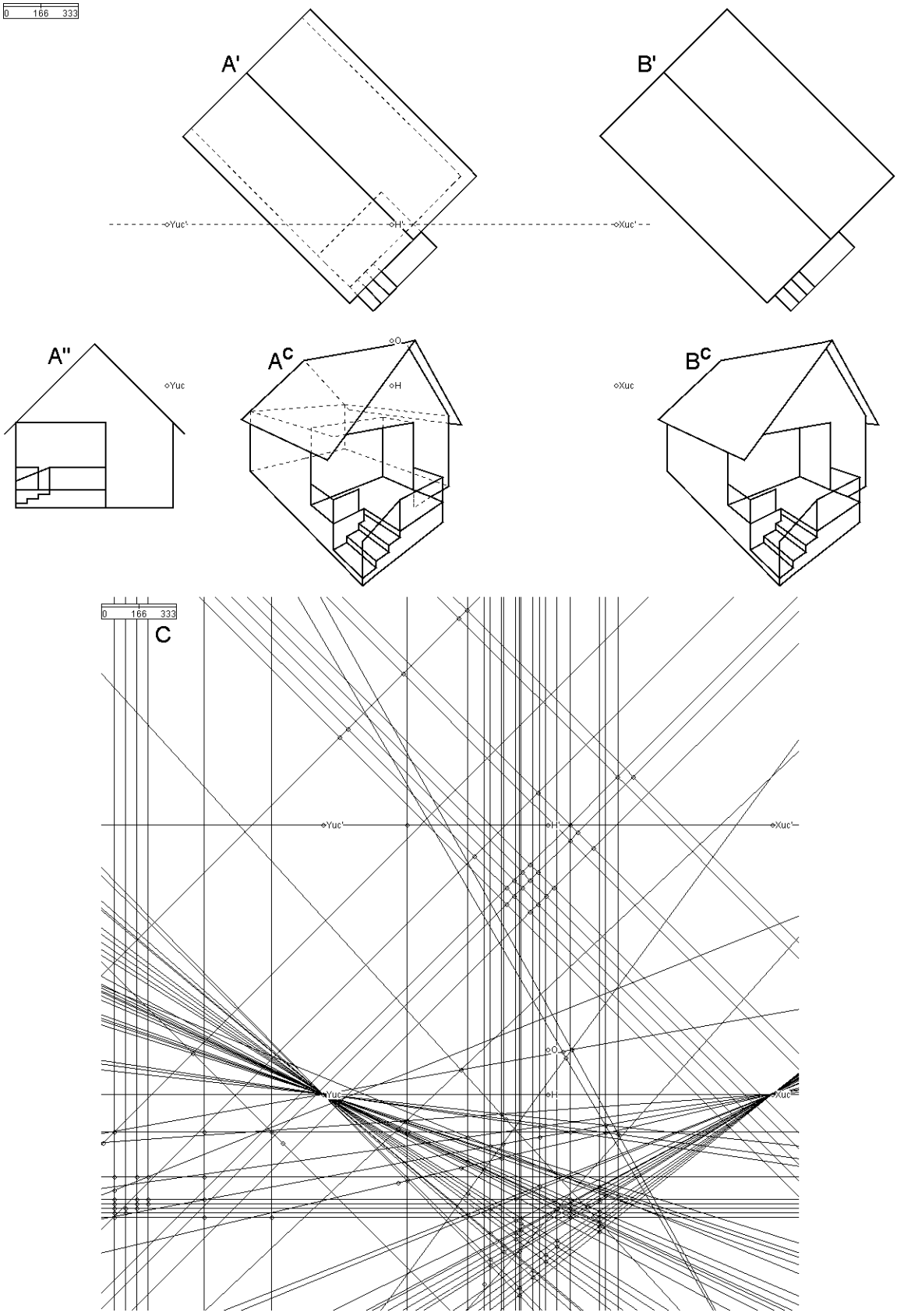

Figure 27: Construction of a house in central projection.