IJAC 3-1 body 7/4/05 3:55 pm Page 75

Structural Sketcher: Representing and applying well-structured graphic representations in early design

Slava Pranovich, Henri Achten, Bauke de Vries and Jack van Wijk

Structural Sketcher: Representing and applying well-structured graphic representations in early design

Slava Pranovich, Henri Achten, Bauke de Vries and Jack van Wijk

Computational drawing support has the potential to improve design support in the early phase. Much work in this area is devoted to input of design information, manipulation, and presentation. Based on a review of current work, we note that among other things, digital drawing tools should be close to the conventions and techniques already used by architects. This is, in principle, possible by processing strokes in a more or less traditional sketch approach, or by offering specialised commands that provide a direct implementation of such conventions. The latter approach is covered by *Structural Sketcher*. A subset of drawing conventions developed earlier, called graphic units, is adopted within the system. In order to contribute to design support, the application of such graphic units should be fast and intuitive, and the definition of internal relationships should be quick and straightforward. For intuitive manipulation, Structural Sketcher incorporates the "paper and scissors" metaphor, and introduces a novel UI-concept called the KITE. To achieve an easy and fast maintenance of relationships, a graph based on anchor-points is built-up on the fly. Performance of the system has been tested on a quantitative and qualitative basis. The system shows the benefit that graphic units can bring to drawing support, and how these can be implemented. To conclude, limitations and further work are discussed.

1. Introduction

The phase of early design is the part between client interview and the creation of a concept design. In this phase, the architect develops a tentative solution for the design task by creating, reviewing, rejecting or changing many (sub) solutions. It is essential in this phase that the creation of (sub) solutions is fast and effortless so it does not impede the flow of ideas and observations. Currently, pen and paper still seem to be the best counterparts to enable designers to meet these requirements. The level of detail and sense of 'work in progress' afforded by this medium are well suited to the early design phase; it can be adjusted to the need of the architect (by drawing more or less accurately); and architects have an almost life-long experience in drawing.

When looking from the point of view of computational design support, pen and paper has the drawback that it does not provide a fluent entry means for a digital design support system. Much of the knowledge represented in design drawings such as their internal structuring, organisation of the building layout, and principles of the design is implicit and therefore not stored in standard drawings, nor in their recorded format as images. As a consequence, this knowledge is not available to computational design support. One avenue for improving early design support therefore lies in the combination of the speed of production as offered by pen and paper with a computational representation that can maintain such relationships. Architects then can utilise their considerable drawing skills while the implicit knowledge embedded in the drawing is retained in the transfer to digital format.

A principal problem that arises is how to make the drawing accessible to the computer. There are two major directions to take here: either automated recognition of the drawing that an architect makes, or to offer specialised tools for the production of drawings that are already structured in some way [1:pp.10]. Sketch and drawing recognition still are mostly in development. Only under very restricting conditions can sketch recognition techniques determine what is being drawn. In the study presented here we focus on the specialised tools approach. An important starting point is the observation that architects use drawing conventions such as grid, zone, axial system, contours, and so forth. Such elements form groups of graphic primitives that are meaningful to the architect. They not only describe the design in terms of appearance (walls, spaces, furniture, etc.) but also structure of the design (organisation, grid, layout, circulation, etc.). In our work, we use such meaningful groups of well-structured graphic representations for the implementation of a design supporting sketch tool for early design, called Structural Sketcher.

2. Drawing support tools

Many current generic CAAD systems are ill suited for early design. They offer highly specialised objects such as walls, windows, doors, and so forth that either imply a great deal of assumptions the designer is unwilling to commit to, or that require high overhead to be managed. In the area of 2D and 3D drawing support there is much research that aims to address this question, by prototype systems such as Cocktail Napkin, Pegasus, DDDoolz, GIDA, and SketchUp [2-6]. They all try to improve the interaction with a design system such that it becomes more easy, intuitive, and natural for architects. We distinguish three major areas in current research in terms of improving support for computer drawing:

- Input of design information to the system. Recent examples of simplifying input of design information are seen in sketching tools like SATIN, SILK, and SmartSketch [7-9]. They typically utilise recognisers, interpreters, and multi-interpreters and then apply beautification techniques. Other techniques include 3D surface construction on the basis of 2D strokes (Teddy [10]), next stroke prediction (Pegasus [3]), and sketching at different refinement levels (DENIM [11]).
- Manipulation of design objects and design information. Many techniques aim to simplify aspects such as positioning an object or set of objects; changing their shapes, sizes, or to align and mirror objects. Most of the systems forget about these relationships after the positioning operation is complete [12]. The positioning and orientation of objects is often achieved through "virtual gravity" [13]. Anchor points as in SketchUp [6] give a visual feedback of reaching some geometrical restraints. Constraint techniques are acknowledged early on as powerful addition to interaction techniques. SketchPad [14] was the first drawing system that used explicit constraints. Many other applications followed: ThingLab, Coral, Constraint-Based Diagrammer, Garnet, Inventor, Briar's, WHIZZ'ED, and SCAFFOLD [15-22]. The success of constraint-based approaches to drawing is limited because it imposes additional overhead for creating and managing constraints, solving them, and presenting them to user.
- Presentation of design information. Architects often trace with a tracing paper and a pencil over computer output to improve the presentation of design information, because they feel that the computer output appears stale compared to the "more alive" presentation of hand-drawn graphics. Strothotte et al. [23] have presented a system that can render architectural drawings in a sketchy look. The SKETCH drawing system uses non-photorealistic rendering techniques to make their drawings look ambiguous [24]. Plimmer and Apperley argue that a sketch environment should delay visual feedback until the user requests it [25]. Other design systems offer advanced assessment of design consequences based on early design input [26, 27].

In general the systems mentioned above function on graphic primitives or classes of shapes, most of which have no bearing on a specific domain. This means that they do not have content or meaning beyond their immediate formal appearance. Meaning or more complex shapes with meaning are constructed by the user, and usually not captured in the representation of the system. Manipulation takes place on the level of primitives or shapes, but again without consideration whether specific groupings of them have significance within the domain. As a consequence, the representation often highlights the single shapes (and their manipulation possibilities) but does not reveal on a higher level what specific groupings may mean (grid, zone, axes, circulation, etc.). Such meaningful groupings are conceived as a whole by the architect and therefore should be separate entities for creation in a design-drawing tool.

Furthermore, between several such groupings there are also meaningful relations; for example a grid is used to coordinate elements such as spaces, walls, and furniture; axes structure symmetric arrangements; and circulation schemes organise the building layout and adjacent spaces. It is desirable that these meaningful relations are kept when changes occur. For example, when a grid is moved, the contours that conform to the grid should also move; when the axial system is rearranged, the corresponding shapes are also rearranged; and so on. Many of such relations that are meaningful to an architect are not supported in CAAD packages or in all cases not persistently encoded in the design drawing. Therefore, in this study we also look at structuring and managing the relations between meaningful groupings.

3. Graphic units

Achten [28] identified twenty-four meaningful sets of graphic primitives such as grid, zone, axial system, and so on. These are termed graphic units. A graphic unit is a set of graphic entities that are arranged in specific ways and that are distinguished from other graphic units by the meaning they have for architects. Graphic entities are primitives such as lines, circles, arcs and points. The arrangements between graphic entities are for example closed polygonal shapes for contours or two sets of parallel lines placed orthogonal for rectangular grids. The distinction between graphic units can be illustrated for example by zone and contour. Both are generally depicted as closed shapes, but the zone indicates an area with particular purpose, and a contour indicates a space boundary.

Significant features of graphic units for this research are:

- Graphic units that describe the design (simple contour, contour, specified form, elaborated structural contour, complementary contours, function symbols, element vocabulary, structural element vocabulary, combinatorial element vocabulary, functional space, partitioning system, and circulation) are drawn in elementary shapes and simple lines. The focus lies on shape without going into details how the shape, shape boundary, or lines are elaborated.

- Graphic units that organise the design (measurement device, zone, schematic subdivision, modular field, grid, refinement grid, tartan grid, structural tartan grid, schematic axial system, axial system, proportion system, and circulation system) are visual aids to the architect to provide structure in the design. These elements typically occur in the design phase. In the final design, where the focus is on the realisation of the elements that have to be built, they are often omitted.

Graphic units are more complex than graphic primitives but less complex than many advanced drawing objects in CAAD systems. In terms of meaning, they are well understood and widely shared in the architectural design domain. In particular for early design, we propose that they constitute the right level of detail for drawing support. In our research, they form the basis for specialised drawing tools. Table 1 shows the list of all graphic units and their description. Graphic units that are listed in italic typeface have been implemented in the Structural Sketcher system. The term in brackets indicates the computer graphics type that has been used to implement the graphic unit. This will be discussed in the next section.

Graphic unit (implemented)	Description
Simple contour (contour)	Regular shape showing an outline.
Contour (contour)	Any irregular shape showing an outline.
Specified form (contour)	Contour with specified dimensions.
Elaborated structural contour (contour)	Outline with structural detail.
Complementary contours (contour)	Composition of outlines.
Measurement device	Measure for establishing dimensions.
Function symbols	Textual indication of function.
Functional space	Outline combined with function indicator.
Zone (zone)	Area with specific use or function.
Schematic subdivision	Principle subdivision not applied to concrete shape.
Partitioning system	General subdivision of a large shape.
Modular field	Irregular subdivision of area along coordinating lines.
Grid (grid)	Modularly repeating coordinating lines.
Refinement grid (grid)	Smaller grid coordinated in a grid.
Tartan grid (grid)	Double grid based on two alternating modules.
Structural tartan grid (grid)	Double grid with one band for structural elements.
Schematic axial system	Organisation by means of axes.
Axial system (axial system)	Organisation by means of axes applied to building design.
Element vocabulary (image)	Shapes depicting (interior) elements.
Structural element vocabulary (image)	Shapes depicting structural elements.
Combinatorial element vocabulary (image)	Location principles between elements.
Proportion system	System for deriving proportions.
Circulation system	Principle layout of circulation not applied to shape.
Circulation	Layout of circulation applied to shape.

◀ **Table 1. List of graphic units, their description, and implemented list.**

4. Structural sketcher

Structural Sketcher is a prototype design aid system for the early phase of architectural design which utilizes graphic units. The overall system, underlying representation, interface, and implementation are described in the following sections.

4.1. Overview

Structural Sketcher works by offering specialised commands for creating each kind of graphic unit. Thus there are commands for grids, contours, zones, axes, and so forth. The aim of Structural Sketcher is to provide the designer with the following:

- Easy creation of graphic units in a drawing.
- Intuitive manipulation of graphic units.
- Capture of relationships between graphic units.
- Automatic maintenance of relationships between graphic units.

Research goals of the work are to:

- Provide a computational representation of graphic units and their underlying relational structure.
- Develop and test a natural interface for working with graphic units.
- Show the productivity of graphic units in early design.

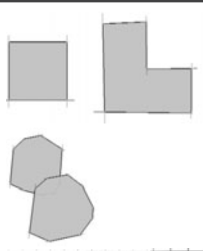
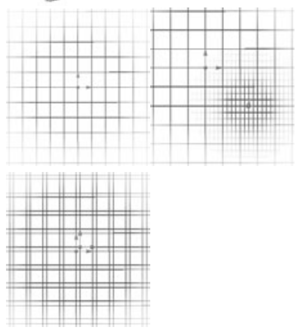
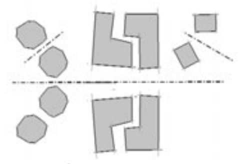
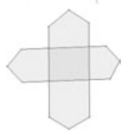
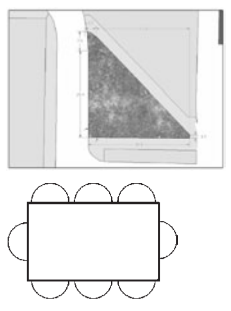
Making and processing strokes lies closest to traditional sketching. Hardware technology moves ever closer to real-time natural drawing input [29], however there are yet no stroke processing mechanisms that deliver reliable and real-time interpretation in terms of graphic units without considerable limitations what range of shapes can be interpreted. In this research we therefore limit ourselves to specialised commands for graphic units. This is done for three reasons: (i) To provide proof of concept that working with graphic units is a worthwhile approach (if not, then further work is pointless); (ii) if the architect has the intent to produce a grid, zone, or contour, then offering that particular command is most efficient; and (iii) the underlying relational structure and manipulation of graphic units is not trivial and has to be solved.

Obviously, this limitation automatically excludes all other kinds of drawings that are not graphic units, and it also ignores much of the individual expression that is captured in strokes. For the purposes of the current work, this limitation is not problematic. The question is further addressed in the conclusion section.

4.2. Graphic unit objects

From all 24 graphic units, 14 graphic units have been implemented in the system. This has been achieved through five separate classes of objects (Table 2).

◀ **Table 2. Implemented graphic unit objects.**

Description	Examples of instances
Five graphic units that deal in some way with spatial shapes are considered as contours and defined as sets of vertices (N_p, p_1 .. p_n, c) with N_p the number of vertices, p_1 .. p_n vertices of a contour, and c a Boolean value for an open or closed contour. Contours can overlap. The object *contour* defines the graphic units Simple contour, Contour, Specified form, Elaborated structural contour, and Complementary contours. The examples show various contours that can be created with *contour*.	
Four graphic units that deal with grids are considered as grids and defined as sets (N_{GC}, p_0 .. p_{NGC}) with N_{GC} is the number of grid components, p_0 defines the origin of all grid components, and p_1 .. p_{NGC} define separate grid components. A grid component is a set of parallel lines. Grids can overlap. The object *grid* defines the graphic units Grid, Refinement grid, Tartan grid, and Structural tartan grid. The grid is defined in such a way that it also accommodates non-rectangular grids, which is particularly useful when manipulating grids. The examples show grid, refinement grid, and tartan grid (clockwise, starting top left).	
The graphic unit Axial system is considered as an object axial system, defined as (p_1, p_2, M) with p_1 and p_2 defining the axis, and M is {(g_i, g_j), ..} with g_i and g_j identifying the mirrored objects. Objects can correspond to multiple axes. The example shows an assembly using various axial systems.	
The graphic unit Zone is considered as an object zone, and defined as (N_p, p_1 .. p_n) with N_p the number of vertices that define the area of the zone, and p_1 .. p_n the vertices of the zone. Zones can overlap. The example shows two overlapping zones.	
Three graphic units that deal with normative layouts are defined as *images* (*bm*, p_1, p_2, p_3, p_4) with bm a bitmap image, and p_1, p_2, p_3, p_4 the vertices of the parallelogram in which the bitmap fits. The object *image* defines graphic units Element vocabulary, Structural element vocabulary, and Combinatorial element vocabulary. Because there is no additional structuring within the bitmap, this object basically functions as a short-cut to create the arrangements that are encoded in these graphic units (see definitions Table 1). The example shows an *image* used as design background, an a table configuration.	

Each object representing a graphic unit has its own characteristics with respect to other objects. The vertices of a contour have gravity and can respond to gravity of other objects. In other words, when a vertex of a contour comes within range of the vertex of a contour or gravity point of another object, it will jump to that point. A grid's intersection points also have gravity so that objects can be coordinated. Furthermore, a grid is only highlighted around the objects that relate to this particular grid. Complex grids such as tartan grids are simply defined by two superimposed grids. An axial system will create mirror-symmetric objects when an object is being made that relates to that axial system. The end-points of an axial system can respond to gravity. Changes in position and orientation of either the axial system or the related objects affect the related objects to keep the design intention of the axes consistent. The zone keeps all objects that are located within its area together over all transformations (move, rotate, change vertex points) since these belong to the same group. The image typically is used for overdrawing (site maps) or as an element within a contour or zone. Its boundary vertices can respond to gravity.

As can be seen from the summary above, the possible interactions between graphic units increase rapidly as the number of implemented graphic units increases as well as the number of instances per graphic unit. Good management of these relations for designers has to take into account two things: (a) Not to overload the designer with overhead; and (b) Predictable, extendible, and flexible definition and maintenance of relationships. The first aspect requires that the handling of objects and relation feels natural. This is further addressed in section 4.3. The second aspect requires an appropriate and efficient structure. This is further discussed in section 4.4.

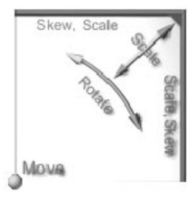
4.3. User Interface: paper & scissor, pins, and KITE

To the user, the means of handling the objects in Structural Sketcher should be clear from the outset. Furthermore, manipulation should involve a minimum amount of actions. Also, the concrete response of the objects should be intuitive.

As a general UI strategy, we adopt the "paper and scissors" metaphor. We view objects as superimposed on top of each similar to a stack of papers. If paper from the stack is moved (in our case similar to manipulating a grid, zone, contour, etc.), then everything that lies above that paper moves along; everything that is connected to the paper moves along, and everything that lies below that paper remains unaffected. This proves to be an intuitively very appealing metaphor for a user to quickly organise and grasp organisations of objects. Since we are not dealing with real paper (objects), visual cues about this stacking are provided by means of shading.

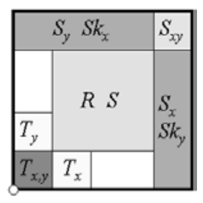
Carrying the "paper and scissors" metaphor further, we introduce the concept of pins by which two or more objects can be put together. Similar to a real pin, these objects undergo translations as a group, whereas relative to each other they can still be rotated. For the architect who needs to put constraints on object-level, there are pins in several (combinations of) colours that block specific transformations (rotation, translation, scaling, skewing, and so forth).

Finally, we have developed a new interface technique called the KITE (acronym of the modestly termed Kool Integrated Transformation Editor). The KITE combines two classic manipulation paradigms; the box-based approach (featured in for example MacDraw and Powerpoint) and the frame-based approach (e.g. as in Maya or 3DStudio Max). The KITE appears as an additional object (in the shape of a rectangle) transparently superimposed on the object that has to be transformed (Figure 1). It integrates one-axis translation (T_x, T_y), free translation ($T_{x,y}$), non-uniform one axis scaling (S_x, S_y), non-uniform scaling ($S_{x,y}$), uniform scaling (S), rotation (R), and skewing (Sk_x, Sk_y) in a single metaphor. By affecting the KITE, the user manipulates the selected object(s). In this way, the KITE applies the box-frame paradigm. The point in the bottom-left corner defines the relative position where the transformations are applied to the object. In this way the frame paradigm is applied [30].



◀ Figure 1. The KITE as it appears to a user.

The transformations are derived through a map of 7 interaction zones underlying the transparent skin of the KITE (Figure 2). The axis scaling and the skewing interaction zones, the uniform scaling zone and the rotation interaction zone overlap. We use the initial movement of the mouse at the start of a drag action to infer the manipulation intention. Scaling is activated for example, when the user drags the mouse orthogonal to the scaling/skewing interaction zone, whereas skewing is activated when this occurs along this zone. These inferences are close to user expectations of behaviour of the KITE, and they allow us to greatly reduce the required amount of UI actions. Both aspects are verified by a user panel and quantitative comparison with other software (see section 5).

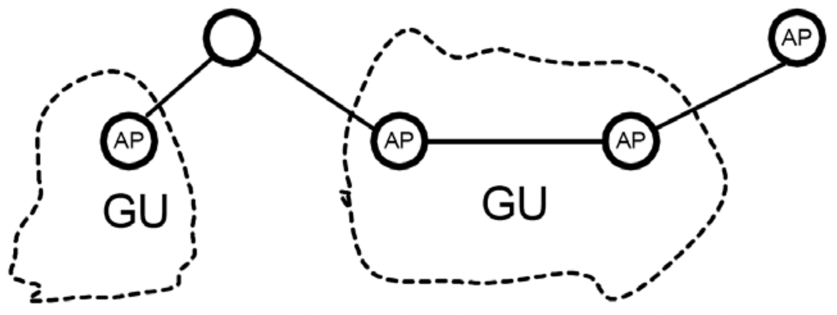


◀ Figure 2. The interaction zones of the KITE.

4.4. Processing: anchor points, spanning trees, and propagation

As the number of instantiated objects and their interrelations increases, the management of relations and maintaining these relations has to be robust, predictable, easy and fast. The UI principles outlined above address the aspect of clarity and intuitiveness, in this section the geometry engine is outlined. An in depth description can be found elsewhere [31].

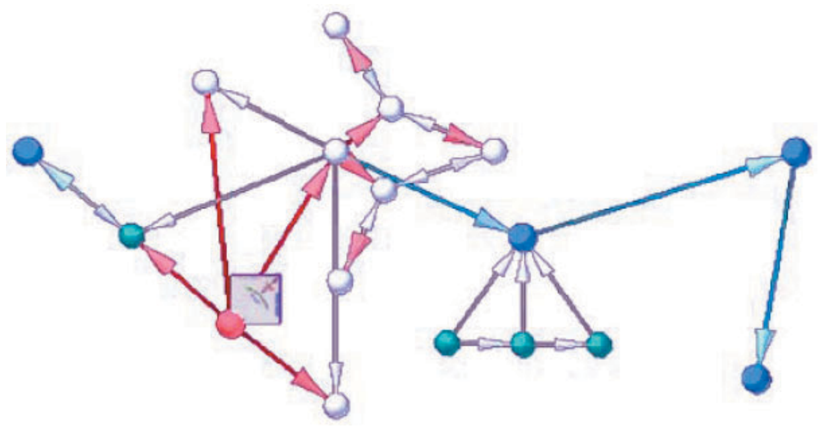
The graphic units that the geometry engine works with, described in section 4.2, are defined by sets of points. For the representation and processing of relations, we introduce another type of object: the anchor point. Each graphic unit has one or more associated anchor points, and an anchor point can also lie outside a graphic unit (Figure 3). Relations between graphic units are defined as directed edges from one anchor point to another and derived from the order of the objects in the user interface. Together, these edges define a graph of anchor points (APG – Anchor Point Graph).



► Figure 3. Relations, anchor points, and graphic units.

The geometry engine uses the APG as a transformable skeleton for the propagation of manipulations between graphic units. There are no restrictions to the graph: It does not have to be connected, multiple independent subgraphs can be used, and the graphs can contain cycles. Transformations are initiated through the KITE interface and always work on an object that has an anchor point in the APG. Next, these transformations are propagated along the edges. Each edge and anchor point can optionally block certain types of transformations, for instance, if all graphic units downstream must maintain their orientation, rotations can be blocked. Ambiguities in the form of multiple paths can exist from the anchor point to which the transformation was applied to an anchor point downstream. To prevent these, we derive a spanning tree [32, 33]. A spanning tree of a graph is a tree that connects all vertices. This ensures that every connected graphic unit is affected only. Each anchor point serves as a local origin. From the anchor point the transformation is applied to the points of the associated graphic unit. Furthermore, dependent on the type of graphic unit further actions are taken, such as sending mirroring transformations for axial systems and snapping for grids.

Figure 4 shows the graph structure underlying a number of graphic units. This view can also be presented to the user of Structural Sketcher. To aid the user in understanding, we change the thickness of lines for outlines of activated graphic units and use colours to highlight a spanning tree when a geometrical transformation is propagated in the graph of graphic units. Red is used for the visualization of relations and graphic units that will be transformed; blue is used for relations and graphic units that are in the tree, but blocked for this transformation. The remaining elements are coloured white.



◀ Figure 4. Propagation of a transformation.

In the first version of Structural Sketcher, users could model objects only directly in terms of graphic units, anchor points, and relations. This low level approach offered a great flexibility, but was also found to be too complex. We therefore added the concepts described in the previous section, i.e., the paper and scissors metaphor, which are internally translated into anchor points and relations between them.

5. User study

Throughout the development, parts of Structural Sketcher are tested with small user panels to assess for example the functionality of APG, KITE, and interaction techniques. The final version of Structural Sketcher is tested in two ways: (i) Making a design by a group of test subjects; and (ii) Comparative study of user actions.

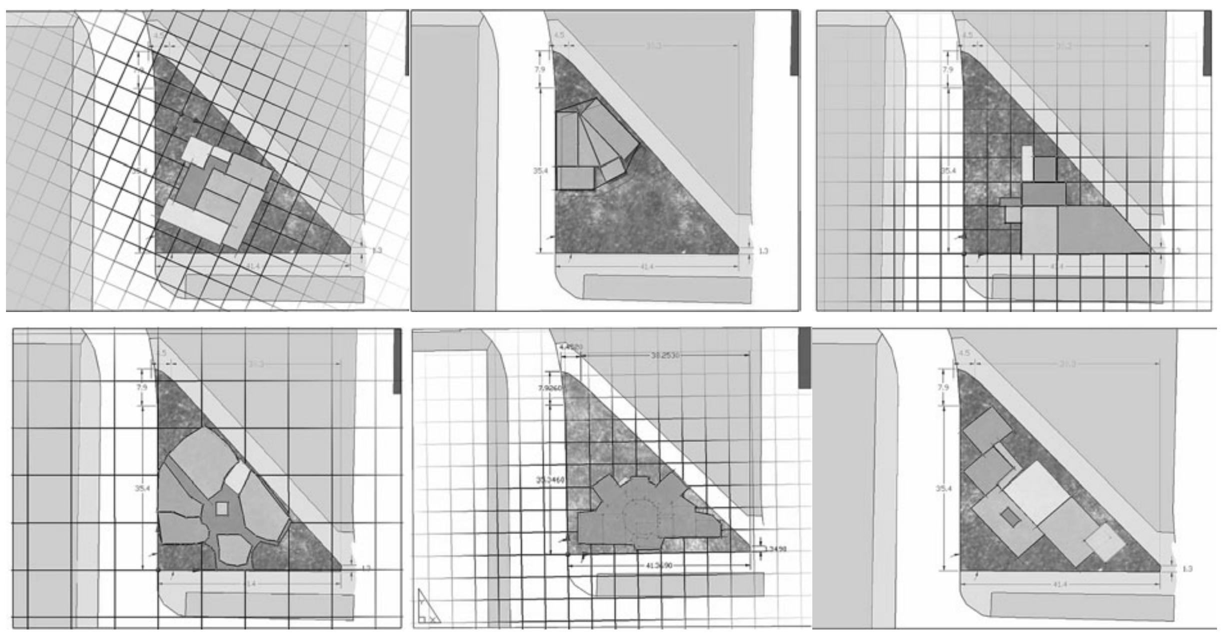
5.1. Design experiment

In the design experiment ten subjects from the Department of Architecture perform the following experiment:

1. Introduction: Overview of the purpose of the experiment and the system (10 minutes).
2. Tutorial: Four tasks of creating and manipulating configurations that the subject has to follow step-by-step using Structural Sketcher (20 minutes).
3. Design assignment: Make a concept design for a doctor's practice (30 minutes).
4. Questionnaire: Review of Structural Sketcher and comparative

questions to other software in terms of perceived suitability for early design, learnability, ease of manipulation, appearance, and level of subjective satisfaction (20 minutes).

Details on the experimental setup can be found in [33:pp.82-85]. We find that the tutorial forms a good starting point to become acquainted with Structural Sketcher. During the design task no problems in terms of UI occurred. The results of the design task (Figure 5) indicates that a rather wide scope of designs is possible.



▲ Figure 5. Some results from the user experiment.

In the questionnaire, we ask the subjects to compare Structural Sketcher with other software as well as traditional pen & paper. A list of existing packages is provided, but subjects are free to add any other software if they are using that software in their design process. The following questions are posed:

1. What other systems are you using in early design? Other systems mentioned by at least seven subjects were: Pen & paper, AutoCAD, PhotoShop, 3DStudio, and ArchiCAD. Powerpoint is mentioned by three subjects, and eight other programs are mentioned by one or two people.
2. Are the systems suitable for early design? Pen & paper ranks first, followed by Structural Sketcher. Corel Draw follows (mentioned by two people), and at some distance 3DStudio Viz.
3. Rate for the amount of time needed to learn to use them. Pen & paper ranks first, after which comes Structural Sketcher, followed

closely by Paint.

4. Rank ease of manipulation of design objects. Structural Sketcher ranks first, followed by AutoCAD and 3DStudio. Pen & paper ranks worst in this case.
5. Is the "look and feel" of the systems appropriate for the early phase of design? Pen & paper ranks first, followed by Structural Sketcher. 3DStudio Viz and Corel Draw follow at some distance.
6. Rank the level of subjective satisfaction. Pen & paper ranks first, second is Structural Sketcher, followed closely by AutoCAD and 3DStudio Viz.

For the statistical analysis of the group Pen & paper, Structural Sketcher, AutoCAD, PhotoShop, 3DStudio, and ArchiCAD we use $\alpha = 0.05$ and for Powerpoint $\alpha = 0.15$ given the small number of subjects who mention it. We also calculate the confidence interval for the group with $\alpha = 0.05$ [33:pp.85-88]. The results are analysed by means of the Wilcoxon signed rank test [35] because it is specially designed for the analysis of nonparametric data. The results of this analysis are presented in Table 3 and show the significance of evaluated aspects for Structural Sketcher versus all other evaluated systems. Significance (P) denotes the probability of the null hypothesis: there is no difference between design systems with respect to evaluated usability aspects.

Structural Sketcher performs better than:	Suitability	Learnability	Manipulability	Look and feel	Satisfaction
Pen and paper	0.005	0.180 n.s.	0.065 n.s.	0.005	0.102 n.s.
AutoCAD	0.016	0.005	0.087 n.s.	0.029	0.429 n.s.
Photoshop	0.011	0.014	0.017	0.038	0.030
3DStudio	0.016	0.07	0.039	0.065 n.s.	0.914 n.s.
ArchiCAD	0.017	0.017	0.042	0.034	0.111 n.s.
Powerpoint	0.109	0.157 n.s.	0.317 n.s.	0.109	0.102

◀ Table 3. Results of statistical analysis. Significant results are shown bold.

It appears that Structural Sketcher scores rather well in terms of ease of learning, ease of manipulation, efficiency, and subjective satisfaction. Points of improvement concern appearance of the system, and suitability for the early phase of design. In the current version, shapes are defined as polygons rather than the freehand shapes of a hand-drawn sketch. Substituting this definition for a stroke-based approach may also further increase the perceived suitability for early design.

5.2. Counting clicks

Apart from the subjective rating by means of the questionnaire, we also compare the number of user actions required to obtain particular results. The comparison is made between Structural Sketcher, AutoCAD, 3DStudio Viz, and Powerpoint, on the basis of three tasks from the Tutorial session.

The first task concerns manipulation of single objects (rotation, scaling, and translation of rectangles); the second task concerns embedded shapes and changing size, shape and location of the container shape; and the third task concerns manipulation of objects that have specific relations between each other. Every mouse action and keyboard interaction counts as a single action. The tasks are performed by expert users of the software who are asked to minimize the amount of actions (no time limit is imposed, and they can search for the shortest chain of actions).

The comparison test shows that Structural Sketcher substantially reduces the number of required user actions to successfully manipulate objects. Over the three tasks, compared to AutoCAD, Structural Sketcher requires respectively 65%, 59%, and 55% fewer actions, compared to 3DStudio Viz this is respectively 40%, 43%, and 54% fewer actions, and compared to Powerpoint this is respectively 13%, 16%, and 54% fewer actions. This indicates that working with Structural Sketcher is faster than with the compared systems.

6. Conclusion

Structural Sketcher is a sketching support system for early design. The implementation of graphic units in the system shows that graphic units and their interrelationships can be represented and maintained in an easy and intuitive fashion. The results produced with Structural Sketcher in the design experiment indicate that a wide range of formal solutions are possible with the system. User experiences place Structural Sketcher between traditional pen & paper and more advanced CAAD software. The possibility for a sketch-like representation is clearly appreciated by the users. Quantitative analysis shows that the new UI-paradigm of the KITE substantially reduces the amount of user actions.

Structural Sketcher is deliberately limited to the support of graphic units. This has the drawback that sketching activities not directly aimed at design representation such as doodling, drawing things not related to the task, improving or refining sketches and so forth are not supported by the system. Although it is not clear how these activities contribute to design thinking, we can assume that there is a factor of plain fun undertaking such activities. Apart from a full implementation of graphic unit-based sketch tools therefore, we feel there is also a need to allow 'doodling', which is not necessarily directly purposeful.

Acknowledgements

Structural Sketcher is the result of the PhD Project by Slava Pranovich, at the Faculty of Mathematics & Computer Science, under supervision of Prof. J. van Wijk. The work was performed in a collaborative research effort with the User Centred Engineering Group of the Faculty of Industrial Design and Design Systems of the Faculty of Architecture, Building and Planning.

References

1. Achten, H.H., *On the Computation of Well-Structured Graphic Representations in Architectural Design*, Eindhoven, Eindhoven University of Technology, 2004.
2. Gross, M., *The Electronic Cocktail Napkin – A Computational Environment for Working With Design Diagrams, Design Studies*, 1996, 17(1), 53-69.
3. Igarashi, T., Matsuoka, S., Kawachiya, S. and Tanaka, H., Pegasus: A Drawing System for Rapid Geometric Design, in: *Proceedings of CHI'98*, April 1998, Los Angeles, ACM Press, 24 – 25.
4. Vries, B. de and Achten, H.H., DDDoolz: Designing With Modular Masses, *Design Studies*, 2002, 23(6), 515-531.
5. Do, E.Y.-L. Functional and Formal Reasoning in Architectural Sketches, in: American Association for Artificial Intelligence, *AAAI Spring Symposium SSS02*, 2002, 37-44.
6. Sketchup: http://www.sketchup.com/ [10-7-2004].
7. Hong, J.I. and Landay, J.A., *SATIN: A Toolkit for Informal Ink-based Applications*, UIST-2000, San Diego, CA, 2000, 63-72.
8. Landay J.A. and Myers B.A., Interactive Sketching for the Early Stages of User Interface Design, in: *Proceedings of ACM CHI '95 Conference on Human Factors in Computing Systems*, 1995, 43-50.
9. Harrod, G., SmartSketch LE – Review, http://www.cadinfo.net/reviews/SmartSketch.htm [10-7-2004].
10. Igarashi, T., Matsuoka, S. and Tanaka, H., Teddy: A Sketching Interface for 3D Freeform Design, in: *ACM SIGGRAPH'99*, Los Angeles, 1999, 409-416.
11. Lin, J., Newman, M., Hong, J. and Landay, J., DENIM: Finding a Tighter Fit Between Tools and Practice for Web Site Design, in: *Proceedings CHI'2000, Conference on Human Factors in Computing Systems*, The Hague 2000, 510-517.
12. Gleicher, M. and Witkin, A. *Drawing With Constraints, The Visual Computer,* 1994, 11(1), 39-51.
13. Bier, E.A. and Stone, M.C., Snap-Dragging, *ACM SIGGRAPH Computer Graphics*, 1986, 20(4), 233-240.
14. Sutherland, I., *Sketchpad: A Man-Machine Graphical Communication System*. Reprint of PhD-thesis 1963, Cambridge: Massachusetts Institute of Technology, Technical Report 574, Cambridge: University of Cambridge, 2003.
15. Maloney J.M., Borning A. and Freeman-Benson, B.N., Constraint Technology for User-Interface Construction in ThingLab II, in: *Proceedings of OOPSLA'89*, 1989, 381-388.
16. Szekely, P.A. and Myers, B.A., A User Interface Toolkit Based on Graphical Objects and Constraints, in: *Proceedings of OOPSLA'88*, 1988, 36-45.
17. Ervin, S.M., Designing With Diagrams: a Role for Computing in Design Education and Exploration, in: McCullough, M., Mitchell, W.J. and Purcell, P. (eds.), *The Electronic Design Studio*, The MIT Press, Cambridge, Massachusetts, 1990, 107-122.
18. Myers, B.A. et al., Garnet: Comprehensive Support for Graphical, Highly-Interactive User Interfaces, *IEEE Computer*, 1990 23(11), 71-85.
19. Strauss P.S. and Carey R., An Object-Oriented 3D Graphics Toolkit, in: *Proceedings of SIGGRAPH '92*, 1992, 341-349.
20. Gleicher, M., A Graphics Toolkit Based on Differential Constraints. *ACM: Symposium on User Interface Software and Technology*, 1993, 109-120.
21. Esteban O., Chatty S. and Palanque P., (1995). Whizz'Ed: a Visual Environment for Building Highly Interactive Software, in: *Proceedings of Interact'95*, 1995, 121-126.

22. Kelleners, R.H.M.C., Constraints in Object-Oriented Graphics, PhD Thesis, Eindhoven University of Technology, 1999.

23. Strothotte, T., Preim, B., Raab, A., Schumann, J. and Forsey, D.R., How to Render Frames and Influence People, in: *Computer Graphics Forum*, 1994 (13)3, Proceedings of EuroGraphics 1994, 455-466.

24. Zeleznik, R.C., Herndon, K.P. and Hughes, J.F., SKETCH: An Interface for Sketching 3D Scenes, in: *SIGGRAPH '96*, 1996, 163-170.

25. Plimmer, B. and Apperley, M., Computer-Aided Sketching to Capture Preliminary Design, in: *Third Australasian User Interface Conference (AUIC2002)*, 2002, 9-12.

26. Leclercq, P., Programming and Assisted Sketching – Graphic and Parametric Integration in Architectural Design, in: Vries, B. de, van Leeuwen, J.P. and Achten, H.H., eds., *CAAD Futures 2001 – Proceedings of the 9th International Conference on Computer Aided Architectural Design Futures*, Kluwer Academic Publishers, Dordrecht, 2001, 15-31.

27. Geebelen, B., Daylight Availability Prediction in the Early Stages of the Building Design Process, PhD Thesis, Katholieke Universiteit Leuven, 2003.

28. Achten, H.H., Generic Representations: an Approach for Modelling Procedural and Declarative Knowledge of Building Types in Architectural Design, Ph.D. Thesis, Eindhoven University of Technology, 1997.

29. Cheng, N.Y.-W. Stroke Sequence in Digital Sketching, in: *Architecture in the Network Society: Proceedings of the 22nd International Conference on Education and Research in Computer Aided Architectural Design in Europe*, Copenhagen, Royal Danish Academy of Fine Arts, School of Architecture, 2004, 387-393.

30. Pranovich, S. and Wijk, J.J. van and Overveld, K. van, The KITE Geometry Manipulator, in: *Extended Abstracts CHI2002*, Vol. 4, Issue 1, ACM SIGCHI, 2002, 764-765.

31. Pranovich, S., Structural Sketcher: a Tool for Supporting Architects in Early Design, PhD Thesis, Technische Universiteit Eindhoven, 2004.

32. Bellman, R.E., On a Routing Problem, *Quarterly of Applied Mathematics*, 1957, 16(1), 87-90.

33. Moore, E.M., The Shortest Path Through a Maze, in: *International Symposium on the Theory of Switching*, 1959, 285-292.

34. Hower, W. and Graf, W.H.: *A Bibliographical Survey of Constraint-Based Approaches to CAD, Graphics, Layout, Visualization, and Related Topics, Knowledge-Based Systems*, 1996, 9(7), 449-464.

35. Levin, R.I., *Statistics for Management*, Prentice Hall, Englewood Cliffs, 1994.

Slava Pranovich[1], Henri Achten[2], Bauke de Vries[2], Jack van Wijk[1],
Technische Universiteit Eindhoven,
[1]Department of Mathematics & Computer Science,
[2]Department of Architecture, Building and Planning,
Den Dolech 2, Postbus 513, 5600 MB Eindhoven, The Netherlands

H.H.Achten@tue.nl