Interactive Beautification: A Technique for Rapid Geometric Design

†*Takeo Igarashi*, †*Satoshi Matsuoka*, ‡*Sachiko Kawachiya*, †*Hidehiko Tanaka*
†Dept. of Information Engineering, ‡Dept. of Information Science
The University of Tokyo
7-3-1 Hongo, Bunkyoku, Tokyo, Japan
+81-3-3812-2111 ext. 7413
{takeo, tanaka}@mtl.t.u-tokyo.ac.jp, {matsu, sachiko}@is.s.u-tokyo.ac.jp

ABSTRACT

We propose *interactive beautification*, a technique for rapid geometric design, and introduce the technique and its algorithm with a prototype system Pegasus. The motivation is to solve the problems with current drawing systems: too many complex commands and unintuitive procedures to satisfy geometric constraints. Interactive beautification system receives the user's *freestroke* and beautifies it by considering *geometric constraints* among segments. A single stroke is beautified one after another, preventing accumulation of recognition errors or catastrophic deformation. Supported geometric constraints includes perpendicularity, congruence, symmetry, etc., which were not seen in existing freestroke recognition systems. In addition, the system generates *multiple candidates* as a result of beautification to solve the problem of ambiguity. Using the technique, the user can draw precise diagrams rapidly satisfying geometric relations without using any editing commands.

Interactive beautification is achieved by three sequential processes; 1) inferring underlining geometric constraints based on the spatial relationships among the input stroke and the existing segments, 2) generating multiple candidates combining inferred constraints appropriately, and 3) evaluating the candidates to find the most plausible candidate and to remove the inappropriate candidates. An user study was performed using the prototype system, a commercial CAD, and an OO-based drawing system. The result showed that the users can draw required diagrams more *rapidly* and more *precisely* using the prototype system.

KEYWORDS: Drawing programs, sketching, pen-based computing, constraints, beautification.

INTRODUCTION

Commercial Object-Oriented(OO) drawing editors such as MacDraw and CAD systems have various editing commands and special interaction modes. An user can construct a diagram with geometric constraints by combining these commands appropriately. For example, symmetry can be achieved by the combination of duplication, flipping, and location adjustment, while perpendicularity can be achieved by duplication and 90 degree rotation. In addition, CAD systems often have special interaction modes such as a mode for drawing perpendicular lines. However, invoking these commands or switching to the special editing modes requires additional overhead, and selection of appropriate commands or interaction modes is difficult, especially for novice users[12].

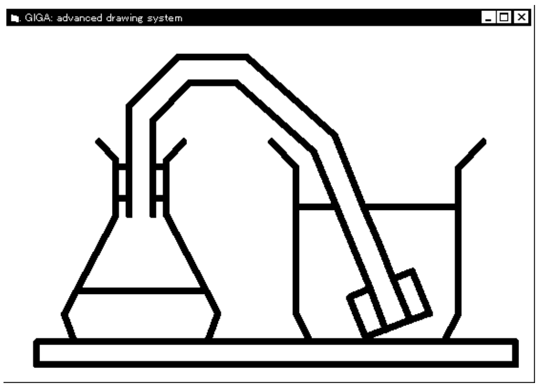

Figure 1: A diagram drawn on the prototype system Pegasus: this diagram is drawn *without any editing commands* such as rotation, copy, or gridding.

To solve these problems, we propose a new interaction technique for drawing, *interactive beautification*. Interactive beautification is a technique for rapid construction of geometric diagrams (an example is shown in Figure 1) without using any editing commands or special interaction modes. Interactive beautification can be seen as an extension of free stroke vectorization [7] and diagram beautification [18]. It receives a user's free stroke and beautifies the stroke considering various geometric constraints among segments. The intuitiveness of the technique allows novice users to draw such precise diagrams rapidly without any training.

Interactive beautification is characterized by the following three features; 1) stroke by stroke beautification, 2) automatic inference and satisfaction of higher level geometric constraints, and 3) generation and selection of multiple candidates as a result of beautification. These three features work together to achieve rapid and intuitive drawing, avoiding the problem of ambiguity.

Interactive beautification is currently implemented on a prototype system Pegasus (an acronym for “Perceptually Enhanced Geometric Assistance Satisfies US!”), and user evaluations using it showed promising results. This paper introduces interactive beautification, and describes the implementation of prototype the system Pegasus in detail.

The remainder of this paper is organized as follows: the next section describes related work in diagram drawing on computers. Then, we describe the technique as seen by the user using several examples. We describe the algorithm of the technique in detail, and introduce the prototype system Pegasus. An user study performed to confirm the effectiveness of the technique is described. Finally, we consider the limitation of our current implementation and conclude the paper.

RELATED WORK

Much work has been done to facilitate the diagram drawing on computers for these 35 years. We will overview several important techniques that have been developed, which affected the design of interactive beautifiacation.

At a glance, the system may seem similar to existing sketch-based interfaces including commercial products such as Apple’s Newton, GO’s Penpoint, and freestroke drawing mode in typical drawing editors (SmartSketch, Corel Draw, etc.). These systems convert freestrokes into vector segments, and satisfy primitive geometric constraints such as connection. The difference is that interactive beautification considers complex, global constraints such as parallelism, symmetry, or congruence, which enhances the range of geometric models. In addition, the generation and selection of multiple candidates is unseen in the existing systems.

Gesture based systems [1][23][19][16] also employ freestroke input, but they convert input strokes into independent primitives, while interactive beautification converts them into simple line segments satisfying geometric relations. Gross pointed out the importance of context in solving the problem of ambiguity[9], which has influenced our idea.

Beautification systems [18][22][14] are basically batch-based, which can lead to unwanted results because of ambiguity in users input. Interactive beautification prevents such results by interactively presenting multiple candidates and requesting user’s confirmation.

While interactive beautification systems control the placement of two vertices (start and end) simultaneously, many existing drawing systems assist the placement of a vertex by controlling the movement of the mouse cursor. Grid restricts cursor placement to some specific geometry and gravity function snaps the cursor to some meaningful places [3]. For example, the Adobe Intellidraw editor[17] automatically aligns the cursor to existing edges. In comparison, the advantages of interactive beautification are as follows: 1) Freestroke drawing is more intuitive and less cumbersome than careful manipulation of the cursor, especially for pen-based interface. 2) The system can attain more information from freestroke trace than cursor placement. For example, equality of interval between parallel lines cannot be detected from the placement of a single vertex.

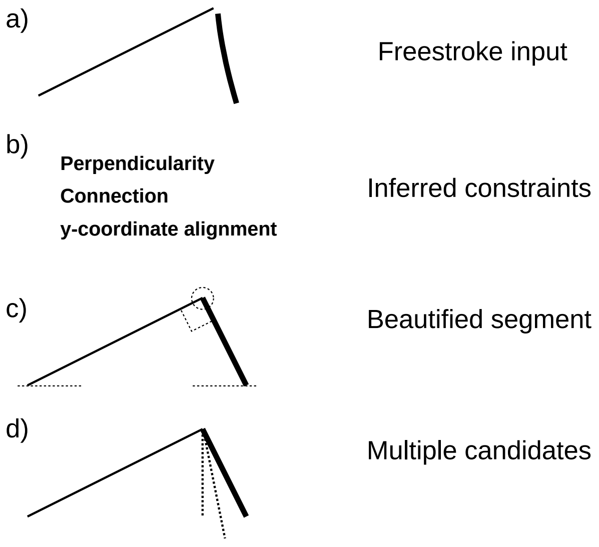

Figure 2: Basic operation of interactive beautification

Bier’s Snap Dragging[2], an extension of gravity-active grids, has the same motivation as ours; to make construction of geometric design easier. However, interactive beautification requires much simpler and fewer operations to construct precise diagrams. Moran et al.’s work [20] shares our aims, but does not support the construction of precise diagrams.

Constraint based systems [10][5][21][6] facilitate the construction of complex diagrams with many constraints, but require considerable amount of effort to specify the constraints. Interactive beautification aims at an opposite goal: to reduce the effort by focusing on relatively simple diagrams.

INTERACTIVE BEAUTIFICATION

Basically, interactive beautification is a freestroke vectorization system; it receives a freestroke and converts it into a vector segment, inferring and satisfying geometric constraints.

First, the user draws an approximate shape of his desired segment with a freestroke using a pen or a mouse (Figure 2a). Then, the system infers geometric constraints the input stroke should satisfy by checking the geometric relationship among the input stroke and existing segments(Figure 2b). Finally, the system calculates

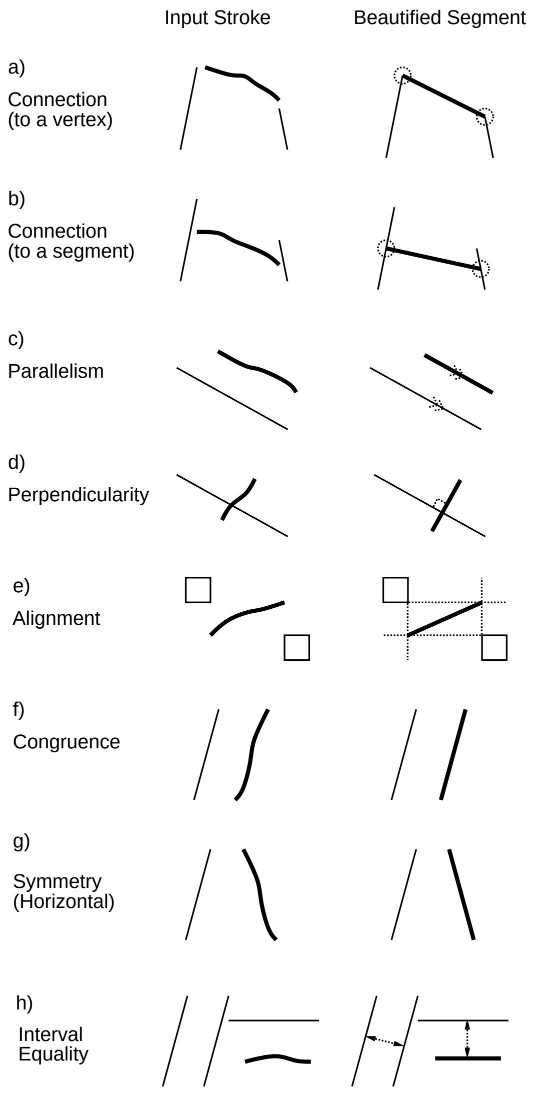

Figure 3: Supported geometric relations

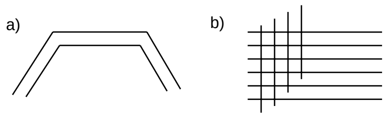

Figure 4: Example use of interval equality among segments

the placement of the beautified segment by solving the simultaneous equations of inferred constraints, and display the result to the user(Figure 2c). In addition, the system generates multiple candidates to deal with the ambiguity of the freestroke (Figure 2d).

The characteristics of interactive beautification are 1) stroke by stroke beautification, satisfying higher level constraints such as congruence, perpendicularity, or symmetry, and 2) generation and selection of multiple candidates. We describe the detail of the interaction in the following subsections.

Stroke by Stroke Beautification Satisfying Geometric Constraints

This subsection describes how diagrams are constructed using stroke by stroke free stroke beautification, satisfying various geometric constraints. To make it simple, we assume that the system generates only one candidate as a result of beautification in this subsection. Next subsection describes the generation of multiple candidates in detail.

Figure 3 shows some examples of supported constraints, input strokes, and beautified segments. Figures 3a,b describe the connection constraint. If the user draws a free stroke whose start or end point is located near a vertex of an existing segment, the system automatically detects the adjacency and connects the point to the vertex or the body of a segment.

Figures 3c,d illustrate parallelism and perpendicularity constraints. The system compares the slope of the input stroke and those of existing segments, and if it finds an existing segment with approximately the same slope, it makes the slope of the beautified segment identical to the detected slope. Similarly, if the system finds an existing segment approximately perpendicular to the input stroke, it converts the stroke into a precisely perpendicular segment.

Figure 3e shows vertical and horizontal alignment constraints. When a free stroke is drawn, the system individually checks the x and y coordinates of the vertices of the input stroke, and makes the coordinates precisely identical to the existing ones if they are near.

Figures 3f,g show congruence and symmetry constraints. When a new input stroke is drawn, the system searches for a segment almost congruent to the stroke among the existing segments. If such a segment is found, the system makes the input stroke exactly congruent to the segment (Figure 3f). Similarly, the system searches for a segment that is similar to the vertically or horizontally flipped input stroke. If such a segment is found, the system makes the input stroke exactly congruent to the flipped one (Figure 3g).

Figure 3h describes interval equality. This relation is detected by comparing the interval between the input stroke and an existing line segment parallel to the stroke, and intervals between existing parallel segments. This mechanism can be used to draw a pipe with a constant width or to draw cross stripes or grids (Figure 5). Construction of these diagrams is particularly difficult with menu-based systems, where the user must copy, rotate, and move the segment.

In actual drawing, the geometric constraints described above are combined and work together to produce a precise diagram. In Figure 3a, relations such as connection, perpendicularity, and y-coordinate alignment are simultaneously satisfied. In Figure 3b, interval equality, y-coordinate alignment and flipped congruence (symme-

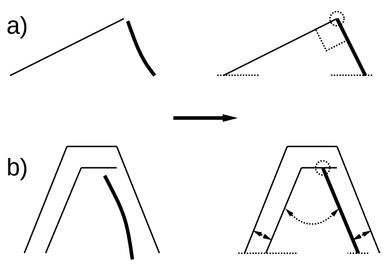

Figure 5: Construction of a diagram with many constraints

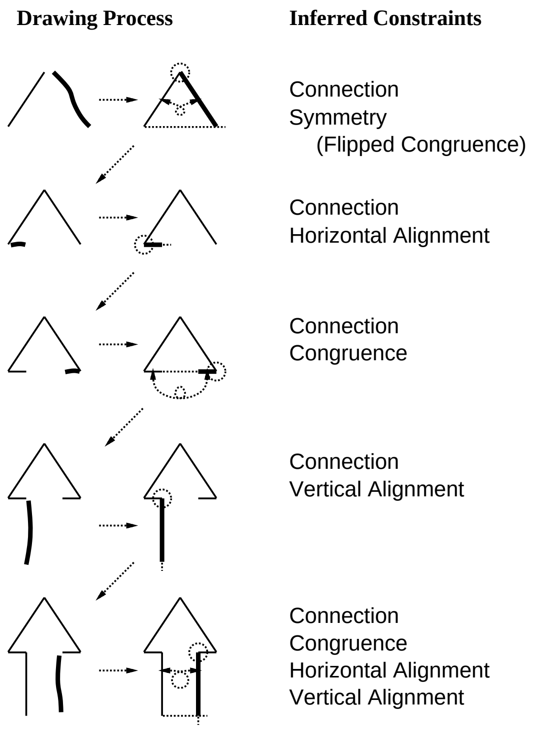

Figure 6: Construction of a symmetric diagram

try) work together to generate the arch (The unnecessary line fragments can be removed easily by 'erasing' interaction, which is explained later).

Figure 6 illustrates how a symmetric diagram is constructed using interactive beautification. For each input stroke, the system infers appropriate constraints and returns a beautified segment. Notice that, except for the slope sides which constitute the arrowhead, the symmetry for the rest of the arrow shape is achieved solely by locally defined relationships (alignment, congruence and connection constraints) without resorting to some special constraints to achieve global symmetry.

Generation and Selection of Multiple Candidates

The inherent difficulty with any freestroke recognition systems is that a freestroke is ambiguous in nature. The user draws an input stroke with an intended image in mind, and the system must infer the intended image

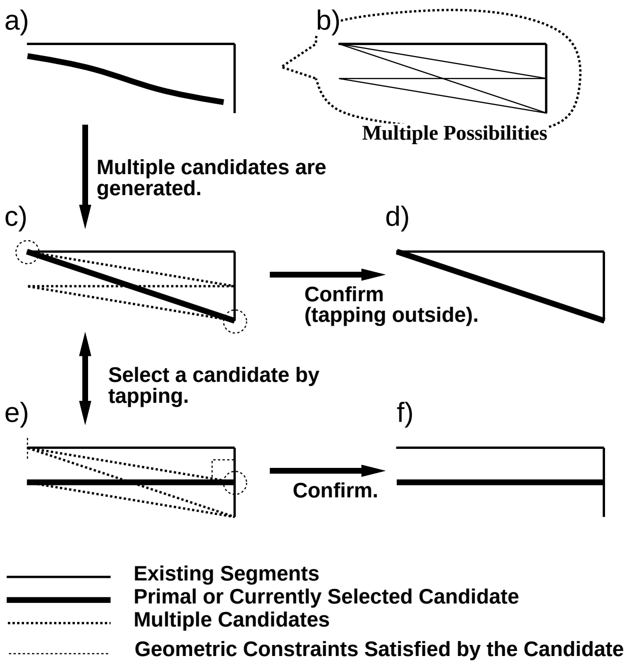
a)
b)
Multiple Possibilities
Multiple candidates are generated.
c)
d)
Confirm (tapping outside).
Select a candidate by tapping.
e)
f)
Confirm.
Existing Segments
Primal or Currently Selected Candidate
Multiple Candidates
Geometric Constraints Satisfied by the Candidate

Figure 7: Interaction with multiple candidates: the user can select a candidate by tapping on it, and satisfied constraints are visually indicated.

based on the shape of the freestroke. However, it is not an easy problem to reconstruct the intended image from the ambiguous input stroke. For example, when the system observes an input stroke shown in Figure 7a, it is difficult to guess which segment in Figure 7b is the one the user intended. Existing systems do not consider these multiple possibilities, and just return a single segment as a result. If the user is not satisfied with the result, he must draw the stroke again, but the revised stroke may also fail.

To solve the problem, interactive beautification infers all possible candidates and allows the user to select one among them (Figure 7c). If the user is not satisfied with the primary candidate, he can select other candidates by tapping on them directly (Figure 7e). During the selection, the system visually indicates what kinds of constraints are satisfied by the currently selected candidate. Visualized constraints ensure that the desired constraints are precisely satisfied. In addition, they assist the selection of a candidate in a cluttered region, where it is difficult to find the desired one. The selection completes when the user taps on outside the candidates or draws the next stroke (Figure 7d,f).

Generation of multiple candidates, together with visualization of the satisfied constraints, greatly reduces the failure in recognition, and makes it possible to construct complex diagrams such as Figure 1 using freestroke only. Additional overhead caused by candidate selection is minimum because the user can directly go to the next stroke without any operation when the primary candidate is satisfactory.

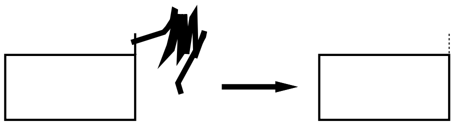

Figure 8: Erasing gesture and trimming operation

Auxiliary Interfaces
In addition to free stroke drawing and selection by tapping, the current system supports a floating menu and an erasing gesture. The floating menu is a button on the screen, and the user can place the button anywhere by dragging it. Menu commands appear when the user taps on the button, similar to a pie menu[11]. Currently, 'clear screen' and 'undo' commands are implemented in the menu.

The erasing gesture is scribbling. If the system detects the gesture, it deletes the nearest line segment to the start point of the scribbling gesture. As the system partitions the line segments at every cross point and contact point beforehand, the user can easily *trim* the unnecessary fragments (Figure 8). Trimming is a frequently used operation on any drawing system, and this easily accessible trimming operation greatly contributes to the efficient construction of complex geometric diagrams.

ALGORITHM

This section describes the algorithm of interactive beautification in detail. From a programmer's point of view, the interactive beautification system works as follows (Figure 9); 1)When the user finishes drawing and lifts the pen from the tablet, the system first checks whether the stroke is an erasing gesture or not. 2)If the input stroke is not an erasing gesture, the beautification routine is called. It receives the stroke and the scene description as input and returns multiple candidates as output. Then, the generated candidates are indicated to the user, allowing him to select one. 3)*The settlement routine* is called when the user finishes selection, that is, starts to draw the next stroke or taps on outside the candidates. The settlement routine adds the selected candidate to the scene description and discards all other candidates. 4)If an erasing gesture is recognized, the erasing routine detects the segment to be erased and removes the segment from the scene. The settlement routine is called after the erasing routine to refresh the scene description. Settlement routine also performs some preliminary calculations to accelerate the beautification process (sorting the vertex coordinates, for example).

We now describe the algorithm of beautification routine in detail. The beautification routine consists of three separate modules (Figure 10). First, a constraint inference module infers the underlining constraints the input stroke should satisfy. Next, a constraint solver generates multiple candidates based on the set of inferred constraints. Finally, an evaluation module evaluates the certainty of generated candidates and selects a primary candidate. The separation of the constraint inference

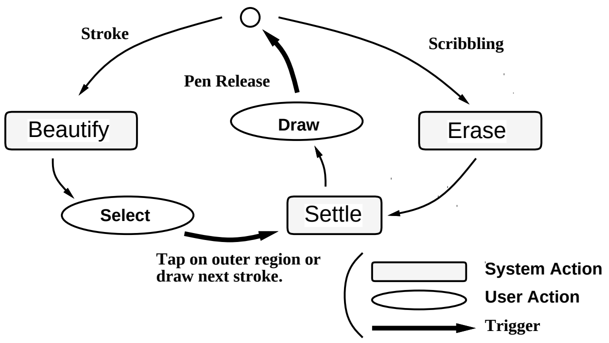

Figure 9: Operational model of interactive beautification

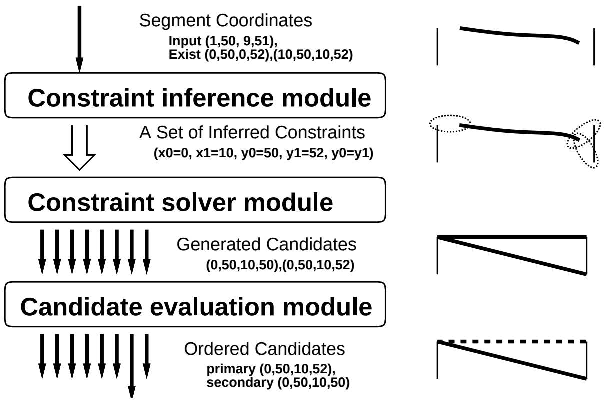

Figure 10: Sturcture of the beautification routine

and the constraint solving remarkably improves the efficiency of multiple candidates generation, because the system performs the most time-consuming task of checking all combinations of segments only once, instead of performing the task for each candidates.

The evaluation process must follow to the solver because it is necessary to consider the resulting coordinates as well as the satisfied constraints to calculate the certainty of a candidate. That is, the candidate located close to the input stroke should be evaluated highly, but the location is unkown until the constraint is solved.

Constraints are represented as numerical equalities binding four variables (coordinates of the new segment). The constraint inference module communicates the inferred geometric relations in a form of numerical equalities, and the constraint solver solves the simultaneous equations. Figure 11 shows the currently supported geometric relations and the corresponding numerical equalities.

Constraint Inference module
First, the system searches the table of parameters of all the existing segments, in order to find values that are 'adjacent' to those of the input stroke and generates constraints that would constraint the parameters of the input stroke as variables. To be specific, the system examines and compares the 5 parameters of the input stroke (x, y coordinates of start/end vertex, and

Geometric Relations	Corresponding Equalities
Connection (start point on a vertex)	**x0 = const y0 = const**
Connection (end point on a vertex)	**x1 = const y1 = const**
Connection (start point on a line)	**y0 = const * x0 + const**
Connection (end point on a line)	**y1 = const * x1 + const**
Alignment (start -x)	**x0 = const**
Alignment (start -y)	**y0 = const**
Alignment (end -x)	**x1 = const**
Alignment (end -y)	**y1 = const**
Vertical line	**x0 = x1**
Horizontal line	**y0 = y1**
Congruence (Symmetry)	**x1 - x0 = cosnt y1 - y0 = const**
Parallelism (Perpendicularity)	**y1 - y0 = const * (x1 - x0)**
Interval equality	**y0 = const * x0 + const y1 = const * x1 + const**

Figure 11: Relation between geometric relations and equalities

the slope of the stroke). As a result, constraints to represent geometric relations such as x and y coordinate alignment, parallelism, and perpendicularity, are generated. As the parameters of all segments in the scene are sorted in the settlement routine, the computational complexity of this routine is $O(\log n)$ while n is the number of existing segments. Perpendicularity is achieved by storing 90 degrees rotation of the existing slopes.

Next, all the segments in the scene are examined to find various geometric relations between the existing segments and the input stroke, such as congruence, connection and symmetry. In addition, to find the equality of intervals among segments, this routine calculates the interval between the input stroke and each approximately parallel segment in the scene, and searches for the stored interval that are adjacent. The computational complexity of this routine is $O(n \log n)$.

This two-phased constraint inference process generates a set of constraints to be satisfied. To reduce unnecessary overhead in constraint solving, the system checks the duplication whenever a new one is created during the constraint inference.

Constraint Solver
Subsequently to the constraint inference, the system calculates the coordinates of the beautified segment based on the inferred constraints. As the inferred constraints are usually over-constrained (they can not be under-constrained because all variables are automatically bounded to the original coordinates of the input stroke), the system searches for all the possible combi-

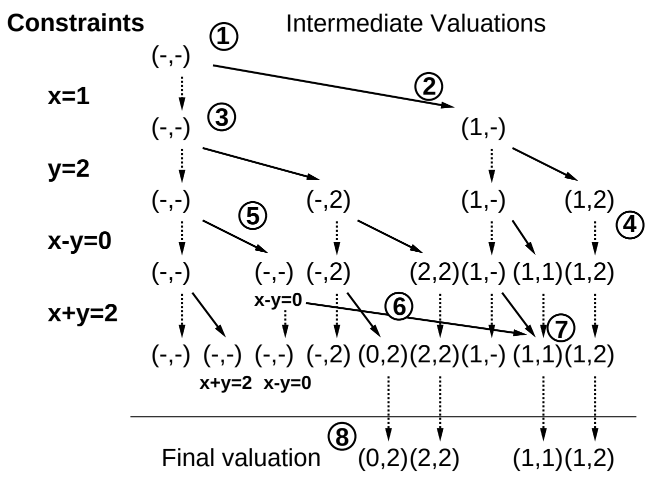

Figure 12: Algorithm for constraint solving

nations of inferred constraints to generate multiple candidates.

The constraint solver is a modification of the equality solver of CLP($\Re$)[13] with an extension to generate multiple candidates from over-constrained equalities. Similar to the equality solver of CLP($\Re$), the initial state consists of an empty valuation, and the system tries to apply the constraint one by one to the intermediate valuation. The difference is that the system maintains a set of valuations instead of a single valuation, and the new valuation is *added* to the valuation set without *discarding* the previous valuation when a constraint is successfully applied.

Figure 12 shows how the solver works using a simplified example with two variables and four constraints. First, the solver creates an empty valuation (1), and then, applies the first constraint `(x=1)` to the valuation. Naturally, the constraint is successfully applied and a new valuation is created `(1,-)`(2). Note that the initial valuation `(-,-)` is preserved instead of being replaced by the new valuation (3). When the solver tries to apply the constraint `(x-y=0)` to the valuation `(1,2)`, the application fails and no new valuation is created (4). On the other hand, the constraint can be successfully applied to the empty valuation `(-,-)`, creating a new valuation with a suspended (delayed) constraint (5). The suspended constraints are solved when enough variables are ground or enough equalities are given(6). Identical valuations are detected and unified by the solver to prevent redundant calculations (7). Finally, the system returns the fully grounded valuations as multiple candidates (8).

To improve efficiency, intermediate valuations are stored in a tree structure whose root node is the initial empty valuation. This representation is natural because every valuation is created as a child of another valuation with additional grounded variables or additional suspended constraints. If a constraint fails to be applied to a valu-

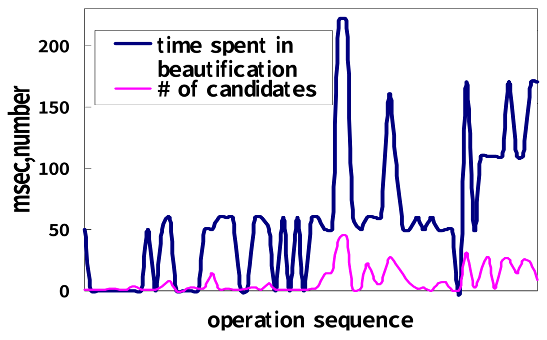

Figure 13: Time spent in the beautification routine and the number of generated candidates during the construction of Figure 1.

ation, it means that the constraint cannot be applied to all its descendants, and the system can avoid wasteful calculations.

The basic method to solve simultaneous equations is Gaussian elimination, because current implementation supports only liner equations. Other algorithms, such as Newton's method[8][10] would be required to support non-liner constraints, such as line length equality or tangency of curved segments. Pair equalities for such constraints as connection to a vertex, congruence, and interval equality (see Figure 11) are bound by `and` condition; both equalities fail if one of them is not satisfied.

In summary, our constraint solver is a multi-way numerical equality solver with an extension to generate multiple solutions efficiently from over-constrained constraints. The complexity of computation is $O(2^n)$, but is substantially reduced by proning wasteful calculations using a tree structure and unifying identical intermediate valuations, and has not caused problems in interaction so far in our prototype system.

PROTOTYPE SYSTEM PEGASUS

The prototype system, Pegasus, is being developed with Microsoft Visual Basic and Visual C++ on Windows 95. The user interface part of the code that manages the input operations and visual feedbacks is written in Visual Basic for ease of implementation and frequent revision. The beautification routine is written in Visual C++ to accelerate the most time consuming process.

Pegasus can work on any PC where Windows 3.1/95 is in operation. However, as Pegasus is basically designed for pen based input, it is developed and tested mainly on portable pen computers (Mitsubishi AMiTY SP) and a pen-based electronic blackboard system (Xerox Liveboard). As pen based freestroke input and mouse based freestroke input have considerably different characteristics, the preprocessor of the recognition algorithm needs to be *tuned* differently depending on the input device (pen or mouse).

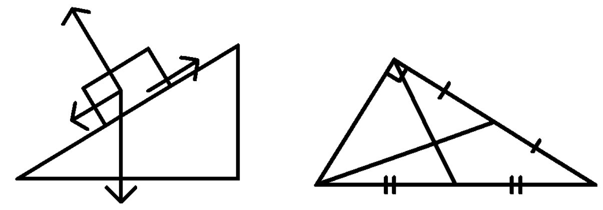

Figure 14: Diagrams for physics and mathematics

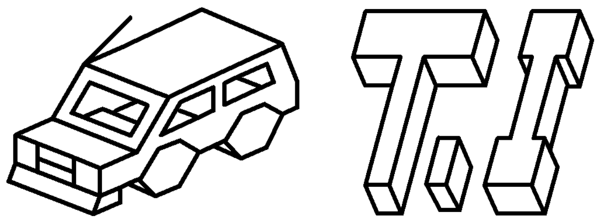

Figure 15: Three-dimensional illustrations

Figure 13 briefly illustrates the processing performance of the current beautification routine. This data was recorded during the construction process of Figure 1 on a PC/AT machine (Pentium 75MHz). Recorded time is not accurate because of the coarse sampling rate of the system call, but approximately 80% of the beautifications finished within 100 msec, sufficient for interactive drawing. The number of generated candidates are usually small, where 62% of beautifications generated less than 5 candidates. However, in some cases (17%) the system generated more than 20 candidates, which made candidate selection difficult.

We show some of the pictures that have been produced on Pegasus. Figure 14 illustrates the usage of the technique in classrooms. Menu-based operations have deterred the use of precise diagrams on electronic whiteboards during oral communications. but the simplicity of interactive beautification may encourage the use of more precise diagrams. Figure 15 shows 3D illustrations. The construction of these diagrams is achieved using parallelism and congruence among segments. It is notable that these diagrams are easily constructed using rather simple constraints, instead of some special techniques for 3D models. Figure 16(left) shows an example

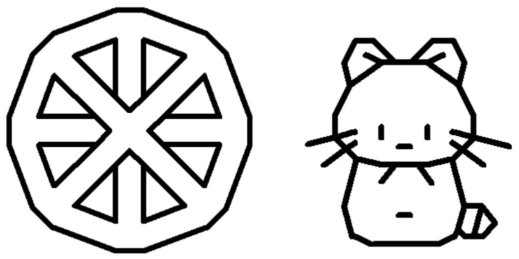

Figure 16: Geometric illustrations

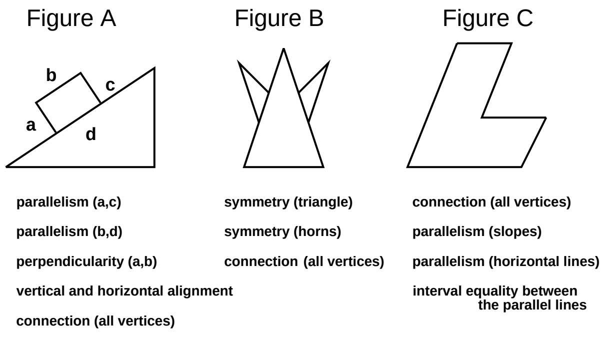
Figure A
Figure B
Figure C
a
b
c
d
parallelism (a,c)
parallelism (b,d)
perpendicularity (a,b)
vertical and horizontal alignment
connection (all vertices)
symmetry (triangle)
symmetry (horns)
connection (all vertices)
connection (all vertices)
parallelism (slopes)
parallelism (horizontal lines)
interval equality between the parallel lines

Figure 17: The diagrams used in the experiment, and required geometric relations

of a geometric design. The widths of the ring and spokes are all identical, which may be difficult for conventional editors. Figure 16(right) gives an example of symmetric illustration. As horizontal symmetry is achieved without any additional operation, a designer can concentrate on *design* itself, instead of struggling with complex operations.

EXPERIMENT

This section describes an experiment performed to evaluate the interactive beautification using the prototype system compared to existing drawing systems in some diagram drawing tasks. We were particularly interested in whether or not interactive beautification would improve the task performance time (*rapidness*) and the completeness of the geometric constraint satisfaction in the diagrams (*precision*). Similar experiment is presented in [12], but this experiment is focused on *evaluation* of the technique, while previous paper intended to clarify the problems of existing drawing editors.

Method

Systems The experiment was conducted on a Mitsubishi pen computer AMiTY SP (i486DX4 75MHz, Win95). Along with our prototype system Pegasus, we used a CAD system (Auto Sketch by AutoDesk Inc.) and an OO-based drawing system (Smart Sketch by Future Software Inc.) The CAD system is used as a representative for precise geometric design systems, and the OO-based editor is selected as a representative for easy-to-use rapid drawing editors.

Task Subjects were required to draw three diagrams shown in Figure 17 using the editors. They were instructed to 1) draw as rapidly as possible, satisfying the required geometric relations as much as possible, 2) to quit drawing when drawing time exceeds the limit of 5 minutes, and 4) give the completion of drawing priority over the complete constraint satisfaction, if it appears to be too difficult.

Subjects 18 student volunteers served as subjects in the experiment. They vary in their proficiency in using computers and each software. 8 subjects were accustomed to typical window-based GUI, but other subjects had little experience with computers.

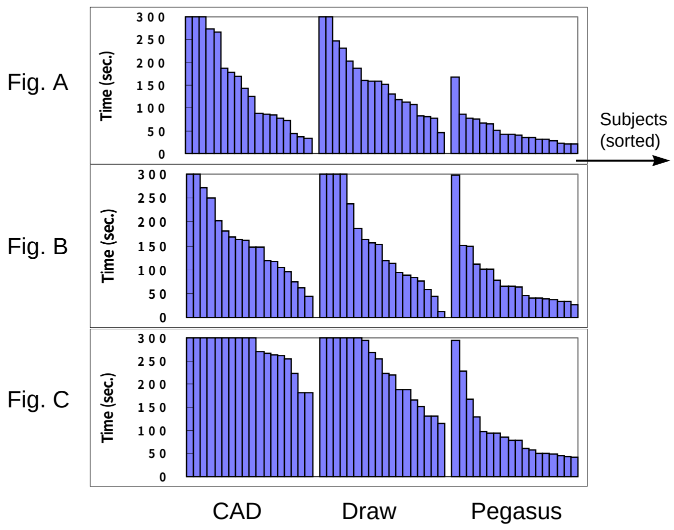

Figure 18: Drawing time required for each task: Each column corresponds to a drawing session of a subject. The order of subjects is sorted by the time required.

Procedure To avoid the effect of learning, the order of editor usage was changed for each subject in a balanced way. The experiment consisted of 18 (subjects) $\times$ 3 (systems) $\times$ 3 (diagrams) = 162 diagram drawing sessions in total. Each session lasted less than 5 minutes and they were video-recorded and examined later.

Prior to performing the experiment with each system, each subject was given a brief explanation of each system and a practice trial. This tutorial session lasted 5 - 10 minutes varying among systems and subjects. CAD system generally required more tutorial time than others.

Result and discussion

Rapidness Figure 18 shows the time required for each subject to complete each task. Each column corresponds to a drawing session of a subject. The order of subjects is sorted by the drawing time. As the drawing time was limited to 300sec., drawing sessions which exceeded the limit are indicated as 300sec. The time required with the prototype system was clearly shorter than with other systems, and all sessions finished within the limit, while many sessions exceeded the limit with the CAD system and the OO-based drawing editor.

Figure 19 shows how many sessions are finished within the limit. Many subjects failed to finish drawing tasks within the limit using the CAD system and the OO-based editor, while all subjects finished drawing using our prototype. Whether the required constraints are precisely satisfied or not is not considered in this graph.

It is impossible to calculate the exact mean drawing time and the mean variance because the recorded drawing time was limited to 300sec., but Figure 20 gives an approximation of the mean drawing time. Drawing time

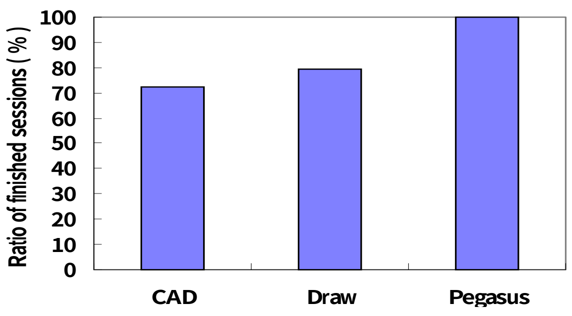

Figure 19: The ratio of finished sessions: this figure shows in how many sessions subjects finished drawing within 300sec. among each $3 \times 18 = 54$ sessions.

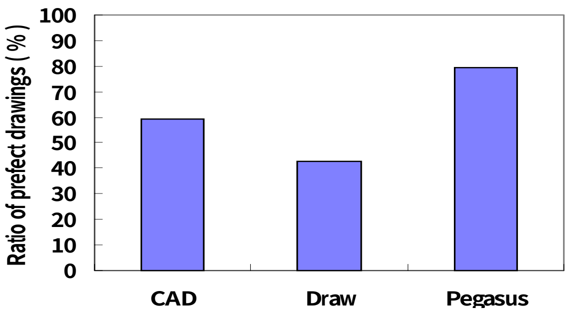

Figure 21: The ratio of diagrams where required constraints are perfectly satisfied: this graph shows in how many sessions subjects successfully satisfied all the required geometric constraints among each $3 \times 18 = 54$ sessions.

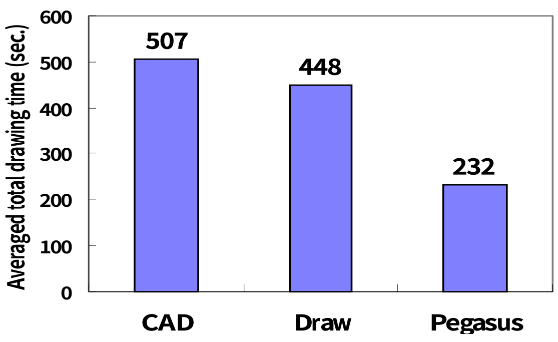

Figure 20: Estimation for time required for a subject to draw the three diagrams: the prototype system exhibits considerable advantage.

is averaged for each diagram-editor combination over those sessions that finished within the limit, and the averaged time for each editor is summed to estimate "total drawing time for a subject to draw three diagrams on each editor." According to the calculations, subjects were able to draw the three diagrams at least 48 % faster than the OO-based editor and 54 % faster than the CAD system. As the averages do not include sessions exceeding 300sec., actual differences are greater.

Precision Even if task performance time might be improved, the benefit could be nullified if the precision of the resulting diagrams is considerably lost. Figure 20 shows how many sessions finished satisfying *all* the required geometric relations shown in Figure 17. The sessions where the subjects finished drawing within 300sec. but failed to satisfy the required geometric relations completely are not counted. It is interesting to see that the OO-based system is superior to the CAD system in time performance, but the opposite holds true concerning the precision, which is in accordance with the natural expectation. Our prototype system showed better performance in both criteria than either systems.

We must note, however, that this experiment is still a preliminary evaluation. Many important aspects of diagram drawing are not accounted for, such as line pattern variation, scaling, rotation, etc. Curves, circles, and text did not appear in the diagrams. Also various kinds of diagrams must be considered, such as node-link diagrams, informal illustrations, complex mechanical diagrams, etc. In spite of these limitations, this preliminary experiment clearly shows a promising potential of interactive beautification system, particularly its significant advantage in rapid and precise construction of simple geometric diagrams. Time performance and constraint satisfaction rate were considerably improved, even though interactive beautification is rather new for the subjects compared with other systems.

LIMITATIONS AND FUTURE WORK

Unsolved problem with interactive beautification is that it is difficult to select the intended candidate among many overlapping candidates. This problem becomes serious when one draws complex diagrams. Possible solutions are to reduce the number of generated candidates and to improve the user interface for candidate selection.

The number of candidates can be reduced by restricting the number of inferred constraints in the constraint inference module and the number of valuations in the constraint solving module, and removing the unwanted candidates in the evaluation module. Various heuristics and user adaptation may be required to find intended constraints and candidates.

Improvement of user interface is also required. One solution is to magnify the cluttered region to help the user to distinguish the desired one from others. Another technique is to let the user specify the reference segment and display those candidates that satisfy constraints related to the specified reference segment.

We plan to implement curves, texts, and line pattern variations to see whether interactive beautification can work as an established interaction technique. Implementation of arcs and curves give rise to various difficulties, but is strongly desirable because satisfaction of curve-related constraints is especially difficult with conventional menu based editors.

We would like to perform more user studies to answer various questions: what kinds of constraints are required for rapid geometric design, how fast user can master the effective use of the technique, and to what extent the generation of multiple candidates facilitates the interaction, etc.

Integration of interactive beautification into 3D scene construction systems such as [24] is also being considered. The most challenging issue may be how to *display* half-constructed 3D models and multiple candidates without confusing the user.

SUMMARY

We have proposed *interactive beautification*, a technique for rapid geometric design. The beautification system receives a freestroke and converts it into a precise segment. The technique is characterized by stroke-by-stroke beautification, recognition of global geometric constraints, and generation and selection of multiple candidates, which make the technique suitable for *precise* geometric design preserving considerable *dexterity*. Our prototype system, *Pegasus*, is implemented on pen computers, and user evaluations showed promising results. The beautification process consists of three independent modules, constraint inference, constraint solving, and candidate evaluation, which achieves efficient generation of multiple candidates.

This technique can be used for geometric modeling on traditional CAD systems, but more informal pen-based rapid drawing of simple diagrams seems to be the most promising target. To be specific, interactive beautification appears to be an ideal technique for note-taking on pen-based PDA systems and graphical explanation on electronic whiteboards during meeting or in classrooms. Finally, this technique can be used for creative design process[15], which has been done with traditional pen and paper rather than on computers because of complex operations.

REFERENCES

1. Apte,A., Vo,V., Kimura,T.D., "Recognizing Multistroke Geometric Shapes: An Experimental Evaluation," *Proc. of UIST'93*, pp. 121-128, 1993.
2. Bier,E.A., Stone,M.C., "Snap Dragging", *Proc. of SIGGRAPH '86*, pp. 233-240, 1986.
3. Bier,E.A., "Snap Dragging: Interactive Geometric Design in Two and Three Dimensions", Ph.D thesis, U.C. Berkley EECS Department, April, 1988.
4. Bolz, D., "Some Aspects of the User Interface of a Knowledge Based Beautifier for Drawings", *Proc. of 1993 Int'l Workshop on Intelligent User Interfaces*, ACM Press, New York, 1993.
5. Borning,A., "The Programming Language Aspects of ThingLab, A constraint-Oriented Simulation Laboratory", *ACM Trans. on Program. Lang. Syst.*, Vol.3, No.4, pp.353-387. 1981.
6. Bouma,W., Fudos,I., Hoffman.D., Cai,J., Paige,R., "Geometric constraint solver", *Computer Aided Design*, Vol.27, No.6, pp. 487-501, 1995.
7. Chen,C.L.P., Xie,S., "Freehand drawing system using a fuzzy logic concept", *Computer Aided Design*, Vol.28, No.2, pp.77-89, 1996.
8. Conte,S.D., Boor,d.C., "Elementary Numerical Analysis", McGraw-Hill, 1972.
9. Gross,M.D., Do,E.Y., "Ambiguous Intentions: A Paper-like Interface for Creative Design", *Proc. of UIST'96*, pp. 183-192, 1996.
10. Heydon,A., Nelson,G., "The Juno-2 Constraint-Based Drawing Editor", SRC Research Report 131a, System Research Center, Digital Equipment Corporation, Palo Alto, California, USA, December, 1994.
11. Hopkins,D., "The design and implemetation of pie menus", *Dr.Dobb's Journal* 1, Vol.6, No.12, pp.16-26, 1991.
12. Igarashi,T., Kawachiya,S., Matsuoka,S., Tanaka,H., "In Search for an Ideal Computer-Assisted Drawing System" *Proc. of INTERACT'97*, 1997, (in press).
13. Jaffar,J., Michaylov,S., Stuckey,P.J., Yap,R.H.C., "The CLP($\Re$) Language and System", *ACM Trans. on Program. Lang. Syst.*, Vol.14, No.3, pp. 339-395, 1992.
14. Kurlander,D., Feiner,S., "Interactive Constraint-Based Serach and Replace", *Proc. of CHI'92*, pp.609-618, 1992.
15. Lakin,F., Wambaugh,J., Leifer,S., Cannon,D., Steward,C., "The electronic notebook: performing medium and processing medium", *Visual Computer*, Vol.5, pp.214-226, 1989.
16. Landay,J.A., Myers,B.A., "Interactive Sketching for Early Stages of User Interface Design", *Proc. of CHI'95* , pp. 43-50, 1995
17. Myers,B.A., Wolf,R., Potosnak,K., Graham,C., "Huristics in Real User Interfaces", INTERCHI'93 Panel, *Proc. of InterCHI'93*, pp.304-307, 1993.
18. Pavlidis,T., VanWyk,C.J., "An Automatic Beautifier for Drawings and Illustrations", *Proc. of SIGGRAPH '85* , pp. 225-234, 1985.
19. Rubine,D., "Combining Gestures and Direct Manipulation", *Proc. of CHI'92*, pp.659-660, 1992.
20. Saund,E., Moran,T.P., "A Perceptually Supported Sketch Editor", *Proc. of UIST'94*, pp. 175-184, 1994.
21. Sutherland,I.E., "Sketchpad: A Man-Machine Graphical Communication System", *Proc. of Spring Jint Computer Conf.*, No.23, pp.329-346, 1963.
22. Weitzman,L., "Designer: A Knowledge-Based Graphic Design Assistant", ICS Report 8609, University of California, San Diego, 1986.
23. Zao,R., "Incremental Recognition in Gesture-Based and Syntax-Directed Diagram Editors", *Proc. of InterCHI'93*, pp. 95-100, 1993.
24. Zeleznik,R.C., Herndon,K.P., Hughes,J.F., "SKETCH: An Interface for Sketching 3D Scenes", *Proc. of SIGGRAPH '96* , pp. 163-170, 1996.