scope: *design;* contribution: *tools & techniques* **(p)**, *applications* **(s)**

Applying constraints to enforce users' intentions in free-hand 2-D sketches

by D. L. Jenkins and R. R. Martin

Our system, Easel, is a smart CAD package which attempts to analyse and tidy up sketches as a designer draws them. In this paper, we present an overview of the system, details of its user interface and the curve-fitting approach. The main part of the paper is concerned with automatic methods to infer and enforce geometric relations intended by the user. We also discuss our conclusions as regards performance.

1 Easel

1.1 Overview

Many early CAD packages used textual input. Users had to learn a large set of commands and then be taught how to draw by more experienced designers. This approach is awkward for two main reasons. First, there is not an intuitive enough relationship between the drawing and the set of instructions that produce it. Secondly, a complex drawing may need a bewilderingly large amount of lines of input to fully describe it.

As technology improved, CAD packages became more interactive and user-friendly. Designers could now draw directly on displays using light pens, graphics tablets or mice, and see the result of their actions almost instantaneously on the screen. In addition, more powerful user interfaces provided menus from which the designer could select predefined shapes such as straight lines, circular arcs, and so on, or predefined actions such as resize, cut etc.

However, this approach lacks generality. Essentially, it is geared to the later stage of design in which the designer knows exactly what is needed, and it is simply a matter of entering the design into the system. In addition, it is more suited to machine parts which, generally speaking, are composed of regular geometric shapes, such as straight lines and circular arcs, rather than free-form curves. We aim to assist the designer during the early stages of a design (when, traditionally, ideas are sketched on the back of an envelope) by attempting to capture the designer's intentions and to replace this rough sketch with a more geometrically exact version as sketching occurs. The results can be further manipulated or even passed directly to another CAD package. In particular, the designer is allowed to sketch in a natural manner, not having to think in terms of straight lines, circular arcs of Bézier curves. Easel can work out how to describe the sketch in terms of these basic primitives.

1.2 Implementation details

Easel is implemented on a Sun-IPC workstation, using SunOS 4.1 and OpenWindows 2.0. Easel is written entirely in the C programming language.

Easel is composed of two functionally independent blocks. The first, the editor, allows the user to enter and manipulate strokes of the sketch. The second, the analyser, takes what has been drawn and attempts to tidy it up as the user sketches. Currently, this architecture is implemented using a single process, with the editor being periodically interrupted to allow analysis. Ideally, the

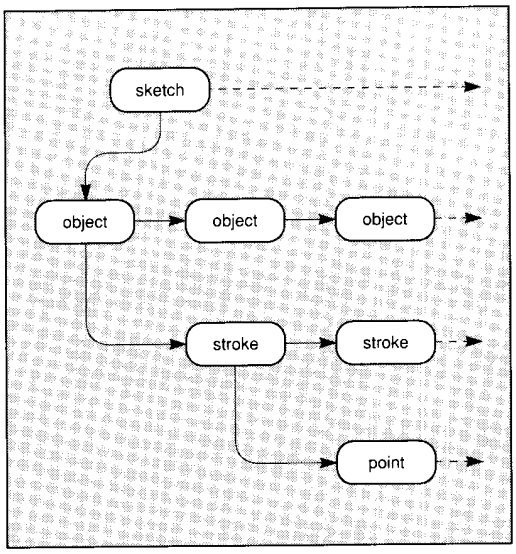

Fig. 1 Data structures associated with each sketch

editor and the analyser should be made separate processes. The editor would then be given a higher priority than the analyser, allowing it to be more responsive to the user during sketching. The analyser would then inspect the sketch during periods of user inactivity, for example, when the user is thinking about what to draw next. However, problems inherent in various methods of communication between processes in UNIX make this approach less attractive than it seems. A more detailed examination of this topic can be found in Reference 1.

1.3 Objectives

Easel has the following objectives.

- Easel must behave in a natural and intuitive manner, which mimics as closely as possible sketching with pencil and paper.
- Analysis must not interrupt the user for too long or too frequently.
- When the user is idle, CPU use must be maximised to produce the results of analysis quickly as possible.

1.4 Comparison with other work

The basic curve-fitting routines used here are widely known. More interesting are the higher order principles used to constrain the geometry of the sketch.

The first work in this area was by Sutherland [2], who essentially laid the foundation work for graphical interfaces. The interface in this case involved a light pen, a set of buttons and a computer screen with which the user could edit the sketch and add new constraints with the buttons.

A different approach was followed by Light [3]. Essentially, his approach used the different constraints found in the sketch and the co-ordinates of each primitive to form a set of simultaneous non-linear equations. A variation of the Newton-Raphson method is then used to find a solution for these equations. Once a solution is found, the position of the primitives are changed so that all the constraints are met.

Another system allowing the input of sketches was proposed by Suzuki *et al.* [4]. Here the designer can enter

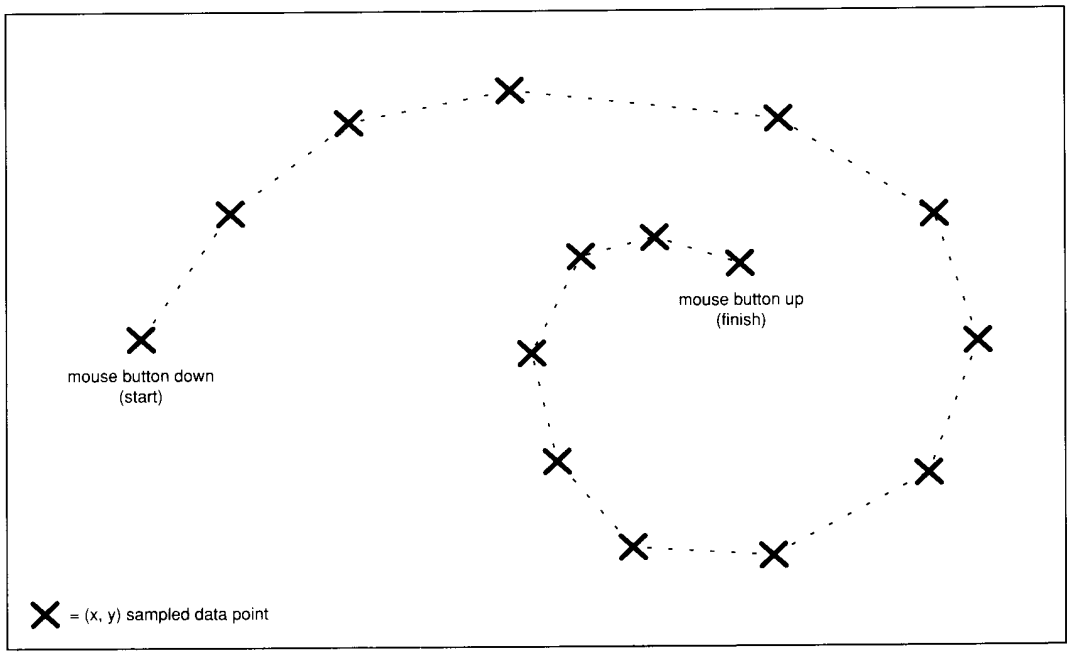

Fig. 2 Composition of a single stoke

a crude sketch, plus some dimensions, and the sytem is capable of tidying up the sketch further. Constraints are stored in a database where they are represented as facts.

Kasturi *et al.* [5] describe a system capable of generating a description of a drawing in terms of various graphics primitives, their interconnections and their spatial relationships. The system can also separate text from graphics and cope with CAD notation such as dashed lines and hatching. Higher order shapes such as triangles, pentagons and hexagons are also recognised.

Borning's ThingLab [9] is a simulation system written in Smalltalk-72, the domain of which is a microcosm of simple physics experiments. ThingLab itself knows nothing of the theory of these experiments, but instead provides an easy-to-use environment for the user to construct objects which contain such knowledge. With the electronic circuit model, for example, one object may be a resistor and a typical message may be 'show', which will cause the resistor to be displayed. Constraints are manually added, which then restrict these objects. ThingLab attempts to satisfy constraints in one pass using relaxation and the method of assumed states.

Pavilidis *et al.* [10] describe a method that infers constraints from a given rough drawing and then modifies the drawing to satisfy the constraints if possible. To achieve this, images are restricted so that they can be defined in terms of points (vertices of polygons, centres of circles etc.). These constraints are then expressed in terms of linear equations. However, to avoid collapsing entities in a picture, Pavilidis *et al.* also allow negative constraints, which are expressed as inequalities. When attempting to satisfy constraints, more desirable constraints are satisfied first by assigning a penalty to each constraint. The solution of the constraint equations is not always guaranteed, but it usually finds a result and does so quickly.

Generally speaking, the main difference between these systems and ours is that the former rely heavily on user interaction to produce the constraints either by stating relations explicitly, or by adding dimensions. The main objective of Easel is to behave in as nearly an automatic manner as possible. In particular, Easel itself attempts to infer the constraints that the user intends.

2 Sketch structure

2.1 *Sketches, objects, strokes and points*

To the application, each sketch is made up of a number of linked objects. Each object consists of a number of linked strokes, each of which contain the raw data points sketched by the user (Fig. 1).

As far as the user is concerned, each sketch is created from a number of free-hand lines or strokes. Each stroke contains a series of orginal (x, y) points sampled along a path drawn by the user. The user can construct a stroke by first pushing the mouse button down, dragging the mouse to form the desired shape on the screen and then releasing the mouse button to terminate the stroke (Fig. 2).

Although we appreciate that user input would be considerably easier and more accurate using a tablet rather than a mouse, our main interest lies in the processing of the input.

2.2 *Analyser overview*

As the user draws, a temporary stroke is created, holding all the points making up the free-hand path. As soon as the mouse button is released, this stroke progresses through the analyser in the manner shown in Fig. 3.

Each new stroke entered is first passed to the preprocessor, which initially reduces the number of points in the stroke by filtering out ones which, if removed, have little effect on the shape of the stroke. Next, the kinks, i.e. small loops and unwanted cusps in the stroke, are

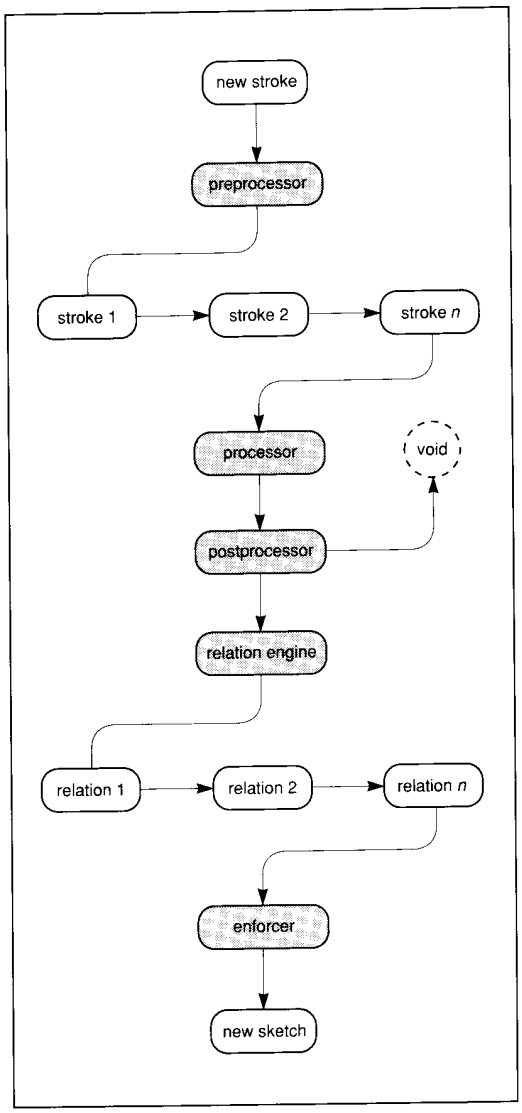

Fig. 3 Analyser overview

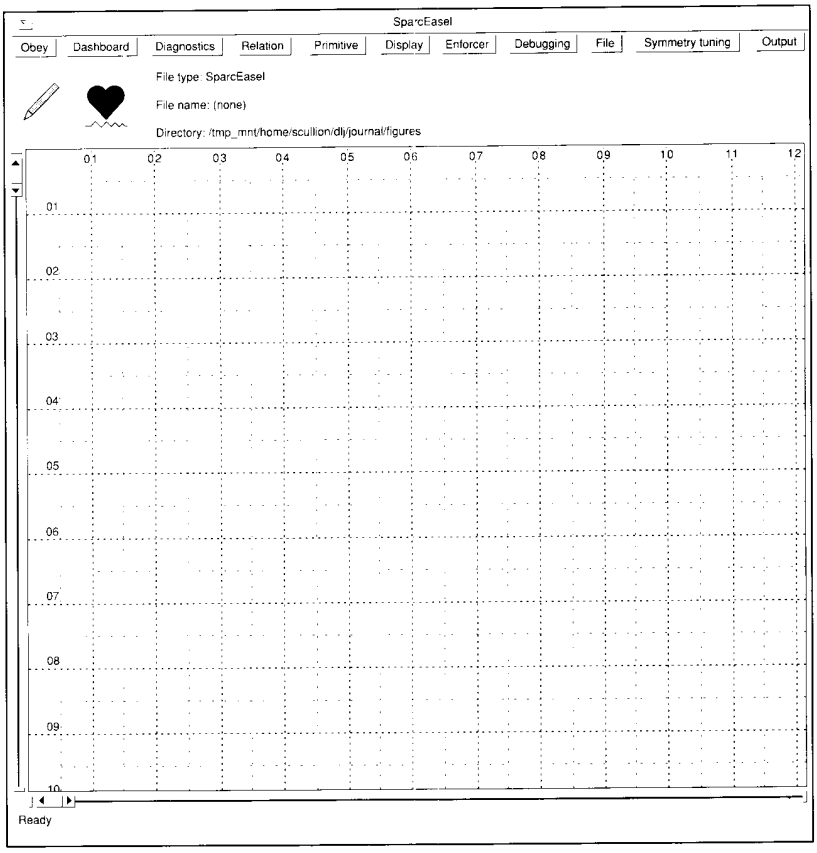

Fig. 4 Easel

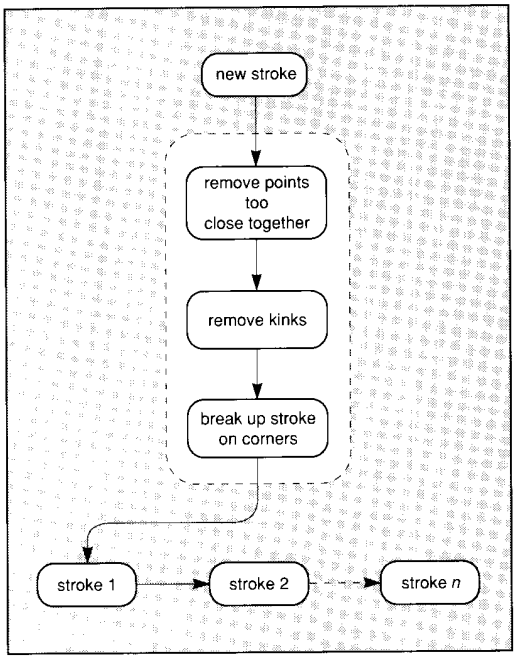

Fig. 5 Preprocessor overview

removed. Finally, corners are detected, and the stroke is split at these points to form one or more new strokes if necessary. These are then passed one by one into the processor, which attempts to recognise them as either a straight line or a circular arc. If it fails to do so, the stroke is fitted to a composite cubic Bézier curve. The postprocessor then discards any erroneous strokes (currently, it only discards straight lines which are shorter than a predefined length).

These processed strokes are then passed to the relation engine, which compares each of the strokes with every other stroke in the current sketch to infer whether any geometric relations (for example, connectivity or parallelism) exist between them. The relation engine then generates a set of relations (possibly empty). Finally, these are passed to the enforcer, which attempts to enforce each of the relations to produce an updated sketch.

3 The user interface

Fig. 4 shows Easel's user interface. Along the top is a line of buttons, each of which has an associated menu. Some of these menus are provided for debugging Easel. Immediately below these are two icons. The pencil icon indicates that sketching is allowed and is replaced by a

stop sign when drawing is prevented. Next to this is a heart icon, which beats in synchronisation with Easel's internal clock. This is used to confirm that Easel is still 'alive'. The main area of the window is taken up by a canvas on which the actual sketching takes place. On this canvas is displayed a grid with units displayed in inches.

3.1 Editing a sketch

In addition to the automatic analysis described in Section 2.2, Easel allows the user to edit the sketch in a number of ways. Essentially, Easel provides these facilities for two reasons. First, the user may wish to alter the sketch simply because of a mistake, or because a new idea requires the sketch to be changed. Secondly, Easel can also make faulty assumptions about the designer's intentions. The editor therefore provides a way for the user to correct an error made by Easel. Each facility is provided in an easy-to-use manner, with the user selecting an operation from a menu and then using the mouse to perform the operation on objects in the sketch. Some of the facilities are listed below and are comparable to those found in current commercial drawing packages.

☐ Entities may be moved, resized and rotated.
☐ Whole entities, or parts of individual entities, can be cut, pasted or copied to the clipboard.
☐ Entities may be cloned or destroyed.

3.2 The object-oriented model

Easel is fully object-oriented. Thus, the user can group strokes together to form compound objects, which are then treated as single entities. Objects can be created in two ways. First, the user can manually group strokes together as objects. Secondly, Easel can do this automatically. At the moment, the automatic creation of objects is limited to the case when a new stroke is broken up by the preprocessor and the component strokes are grouped together as a single new object. However, a future goal is that Easel will also do this when certain relationships exist between two strokes (of which connectivity is an obvious example).

4 The preprocessing step

Many of the ideas in this Section are based on work by Schneider [6]. Generally speaking, the preprocessing stage has two main roles (Fig. 5).

- It filters strokes entered by the user to make them more 'presentable' to the processor (see Sections 4.1 and 4.2).
- It breaks up strokes at corners and automatically groups them together to form a single object.

4.1 Removing co-incident points

Points that are too close together (typically less than three pixels) or are co-incident are removed from the original

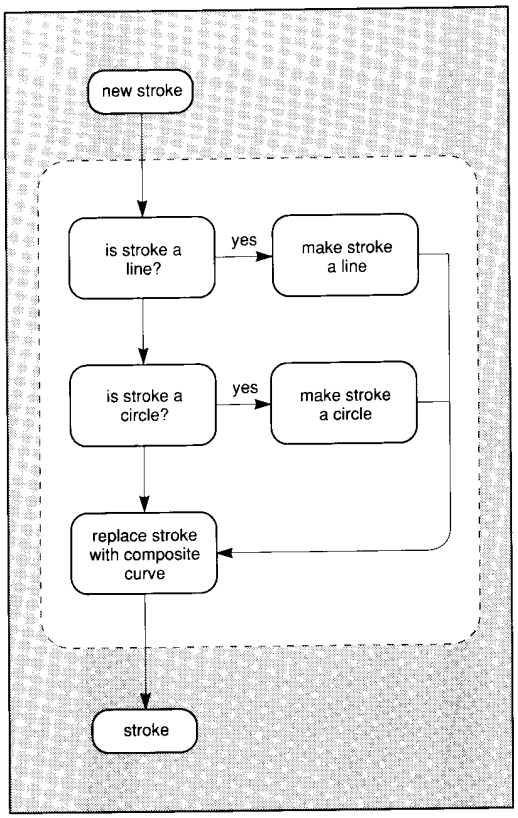

Fig. 6 Processor overview

stroke. Since the performance of many of the processor's routines is dependent on the number of points in a stroke, reducing this number improves overall throughput. However, the time saved by this method is moderate and can possibly be non-existent. The greatest savings are achieved when the user is drawing slowly or drawing with care. Smaller savings are made when the converse is true.

4.2 Removing kinks

When the user sketches, the final stroke may include unwanted loops or cusps. These are caused by

☐ hand shake on the part of the user (either isolated slips or more generally as a result of a lower level of skill).
☐ the mouse failing to accurately reflect the intentions of the user (either as a result of a low resolution or a slow sampling rate).

Either way, these kinks need to be removed for two reasons.

- Spurious data in the form of rogue points adversely affect the processor's primitive fitting routines.
- Kinks fool the corner detector, giving rise to a potentially large number of bogus corners.

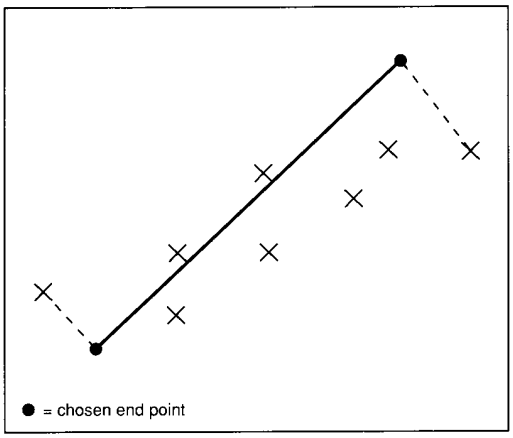

Fig. 7 Choosing straight line end points

To smooth an input stroke, successive triplets of points are taken from the original data and the middle point is moved so that it becomes the weighted average of the three. This process is repeated for each of the points in the stroke, with the exception of the first and last points of the stroke. These remain in their original positions because designers make more effort to get these end points in the right place than the many points in between. Additional smoothing may be achieved by repeating this procedure an additional number of times (the exact number depends on the user and the mouse; typically it is found useful to repeat it twice).

4.3 Detecting corners

Corners are detected whenever the angle between three consecutive points is greater than a chosen value (we typically use a value of around 50°). On finding a corner, the stroke is broken at the middle point, resulting in two new strokes. This process continues until the stroke has been completely examined. The original stroke is then deleted and the new strokes are grouped together to form a single object.

5 The processing step

The processor's main job is to take strokes passed to it from the preprocessor and to fit one of the predefined primitives to the point data received. The simplest primitive types are considered first, so that initially the processor tests if a straight line is a good fit to the stroke data. If this fails, a circular arc is next considered, and if this too fails, a composite Bézier curve is used to represent a general freehand curve (Fig. 6).

All the algorithms in the Sections below use variants of the least-squares method.

5.1 Straight lines

The least-squares fitting of straight lines is well known and covered in Reference 7. Essentially, a straight line is fitted to the data points making up the original stroke. The end points of this straight line are chosen by dropping a perpendicular from the end points of the original stroke to the fitted straight line (Fig. 7).

5.2 Circular arcs

The least-squares fitting of circular arcs is performed in a similar manner, with the circular arc end points chosen by intersecting the circumference with a straight line

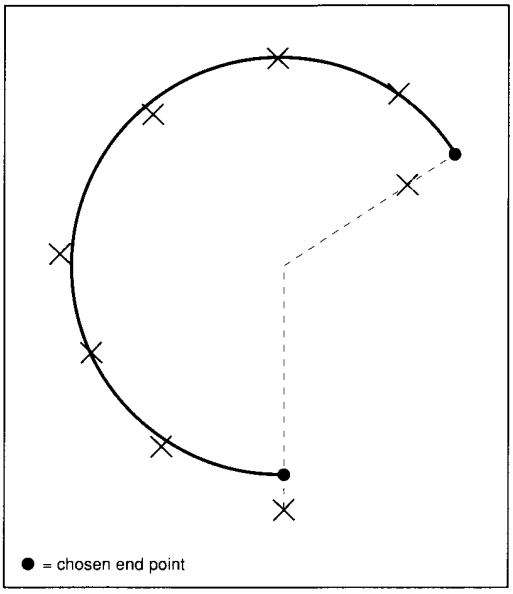

Fig. 8 Choosing circular arc end points

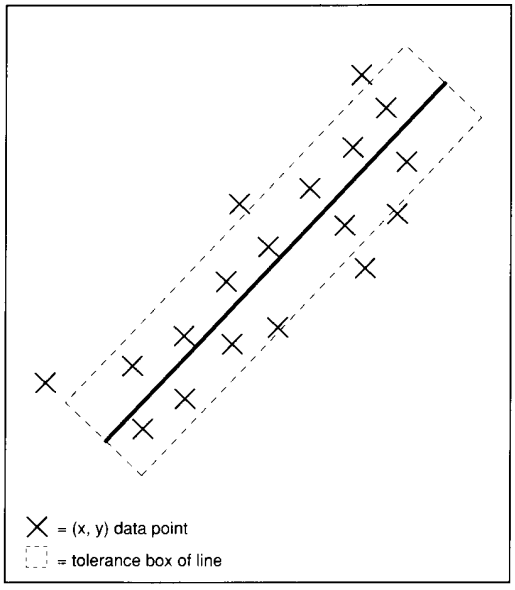

Fig. 9 Straight line tuning

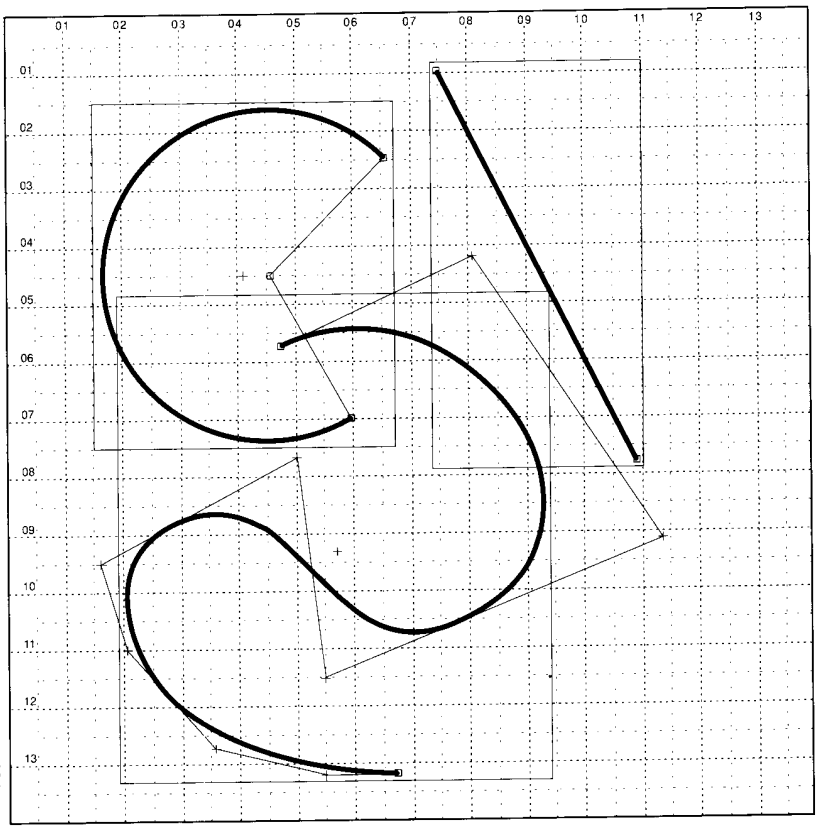

Fig. 10 Stroke enclosing boxes and centres of interest (after tidying up)

drawn from the centre of the circular arc to the end points of the original stroke (Fig. 8).

5.3 *Composite Bézier curves*

If the processor fails to fit either a straight line or a circular arc to the stroke, the stroke is replaced by a composite cubic Bézier curve. Easel uses the recursive algorithm described in References 6 and 8. Briefly, a section of the stroke is taken (initially the whole stroke) and a cubic Bézier curve is fitted to the data. If the fit is within a predefined tolerance, the stroke is replaced by this curve. Otherwise, the stroke is broken into two pieces at the point of maximum error and the fitting process is applied to both sections. This continues until all the Bézier sections fit within tolerance. Joins between curve sections are constrained during the fitting process to ensure C^1 continuity.

5.4 *System tuning*

As mentioned above, Easel uses tolerances extensively when deciding whether a stroke is a straight line or a circular arc, and when fitting a composite Bézier curve. Inside the processor, our tolerance method takes the form of a pair of tuning values for each primitive known by the system. The effect of these two values can be seen by looking at the case of the straight line as an example (Fig. 9). The first of the tuning values for a straight line determines the width of a box centred on the fitted line. The number of points that lie inside this box are then counted (13 in the example) and compared to the total number of points in the stroke (18 in the example) to give a percentage 'hit rate' ($\frac{13}{18} \approx 72\%$ in the example shown). If this percentage is greater than or equal to the second straight line tuning value, the system accepts these data as representing a straight line.

Easel also has a number of other tuning values associated with the preprocessor and the editor. Some of these have been previously discussed and include

- **gravity field strength:** how close an entity has to be to the mouse pointer when attempting to select it.
- **jitter error:** how close points can be before they are removed by the preprocessor.
- **noise reduction:** the number of iterations of the smoothing routine in the preprocessor.
- **corner angle:** the maximum change in direction before a corner is detected.
- **minimum straight line length:** lines shorter than this are discarded.

No two users are the same in their drawing ability and competence. Accordingly, Easel has to be 'taught' how to cope with users of different abilities. This could be done

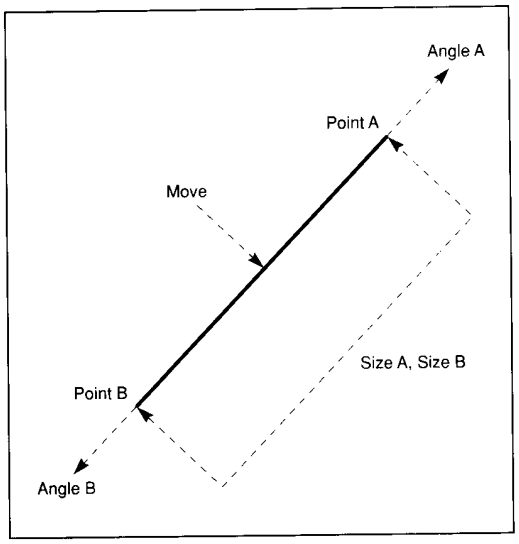

Fig. 11 Straight line locks

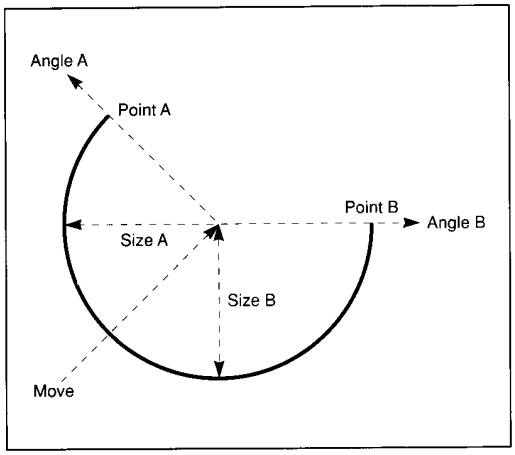

Fig. 12 Circular arc locks

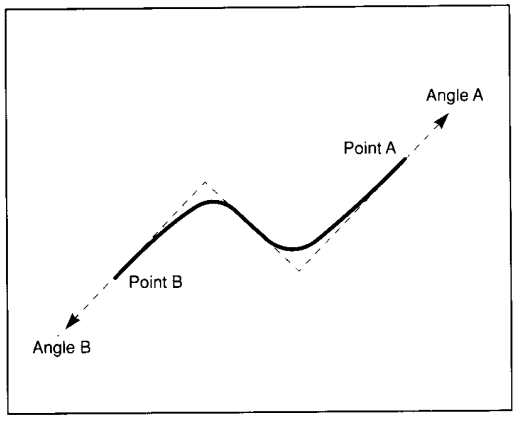

Fig. 13 Bézier locks

by manually choosing the tuning values described earlier, so that when a user draws a circular arc, the user gets a circular arc on the screen as expected. However, this method is tedious to say the least, requiring much trial and error, and also means that the user has to have a greater knowledge of the inner workings of the system than necessary. A better method therefore is a system (which we have yet to implement) that, when encountering a new user, would automatically go into a teaching session consisting of the following lessons:

- The user would be required to draw ten straight lines and then ten circular arcs, for example. Since the system knows that these are definitely meant to be straight lines or circular arcs, it can thus work out what the tuning values for straight lines and circular arc should be for this user.
- This idea could be extended to the other tuning values to build up a profile for each user on the system. This could be achieved by asking the user to reproduce various drawings shown, containing elements which depend on each of the tuning parameters. Another method may be to determine these values when the user is freely sketching, as this represents more realistic input.
- It is inevitable that Easel will make mistakes and incorrect deductions. The user must be able to override these mistakes. The system should also use information about such mistakes to update its tuning profile for the user.

6 General relations

After each new stroke is entered, it is compared with every other stroke in the sketch to determine the implied existence of geometric relations such as connectivity, parallelism, and so on. As in the case of fitting primitives, each relation has its own tuning value (or in a few cases two), which Easel uses to judge if a particular geometric rule has been fulfilled. For example, in the case of connectivity, Easel looks at the end points of a pair of primitives and works out whether they are less than a certain distance apart. If they are, it is supposed that these two primitives should be connected.

6.1 Unitary relations

Unitary relations are properties of a single stroke on its own. Easel supports three unitary relations, *snap angle*, *closed* and *anchor to*.

The snap angle relation applies only to lines and circular arcs. For lines, Easel examines the slope of the straight line to see if it is close to one of a set of predefined angles (every 15° from 0° at present). If the error is within tolerance, the direction of the straight line is changed so that it becomes exactly that angle. In the case of circular arcs, the angles formed from the centre of the circular arc to the two end points of the arc are examined to determine if these angles match any of the predefined set of angles. If the error is within tolerance again, the angle subtended by the arc is changed.

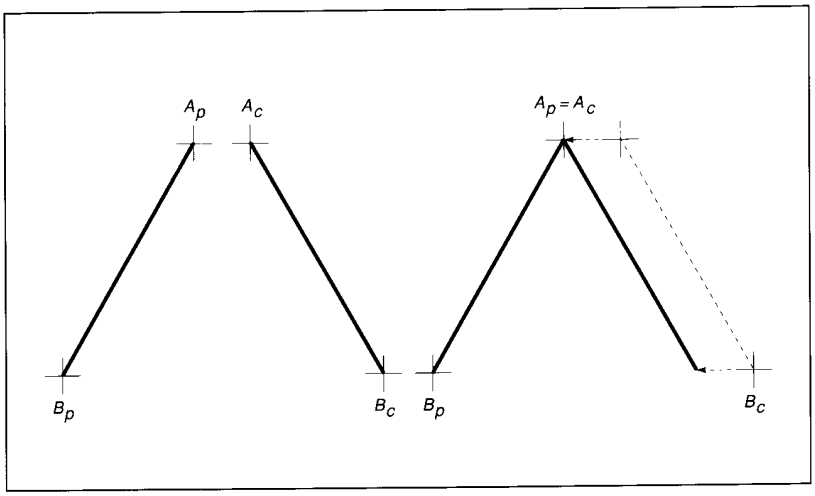

Fig. 14 Scenario one, before and after

The closed relation applies only to circular arcs and composite Bézier curves. In this case, the two end points of a single curve are examined and the distance between them computed. If this error is within tolerance, the end points are moved so that they become co-incident. In the case of Bézier curves, the closed curve is not, however, made tangent continuous.

The anchor to relation applies to all types of strokes. When designers sketch, they often take advantage of a grid. Easel has a grid modelled on a piece of graph paper. The grid size (the distance between horizontal and vertical lines) can be selected to be $\frac{1}{4}''$, $\frac{1}{3}''$, $\frac{1}{2}''$ or 1". In each case, the end points of the stroke are examined to determine if they lie close enough to a grid intersection to be made co-incident with it. Moreover, in the special case of a circular arc, its centre is inspected to determine if it lies close to a grid intersection and its radius is examined to determine if it is close to a multiple of the grid size.

Table 1 Scenario one

	parent locks						
state	Point A	Point B	Move	Size A	Size B	Angle A	Angle B
before	⊗	⊗	⊗	⊗	⊗	⊗	⊗
after	●	⊗	⊗	⊗	⊗	⊗	⊗
	child locks						
state	Point A	Point B	Move	Size A	Size B	Angle A	Angle B
before	○	○	○	⊗	⊗	⊗	⊗
after	●	○	○	⊗	⊗	⊗	⊗

● = set, ○ = unset, ⊗ = don't care

6.2 *Pairwise relations*

Pairwise relations are geometric properties shared between two strokes. Easel currently supports the following pairwise relations:

☐ **Connected:** the end points of two strokes are examined to determine whether these strokes should be connected. Applies to all stroke type combinations.
☐ **Tangent to:** two strokes are compared to determine if they are tangential to one another. Applies to straight line-circular arc, straight line-composite curve, circular arc-circular arc, circular arc-composite curve and composite curve-composite curve stroke type combinations.
☐ **Touching:** the end points of one stroke are examined to determine if either falls on the path of the other stroke. Applies to all stroke type combinations.
☐ **Equal length:** in the case of a straight line-straight line combination, the lengths of the lines are examined to determine if they are equal. In the case of a circular arc-circular arc combination, the radii are compared.
☐ **Concentric:** the centres of two circular arcs are compared to determine if they are sufficiently close to be made co-incident. Applies only to circular arc-circular arc combinations at present.
☐ **Parallel:** the slopes of two lines are examined to determine if they are the same. Applies only to straight line-straight line combinations at present.
☐ **Perpendicular:** the slopes of two lines are examined to determine if they are perpendicular. Applies only to straight line-straight line combination at present.
☐ **Common point centre:** the end points of a stroke are compared with the centre of a circular arc to determine if they should be made co-incident. Applies to straight line-circular arc, circular arc-circular arc and composite curve-circular arc combinations.
☐ **At an interesting fraction:** the end points of a stroke are compared with their point of contact with a straight line to determine if it represents a predefined fraction. Currently, these are the points generated by $\frac{1}{10}$ steps along a line, although we recognise that fractions of $\frac{1}{3}$ and $\frac{1}{4}$ are also useful. Applies to straight line-straight line,

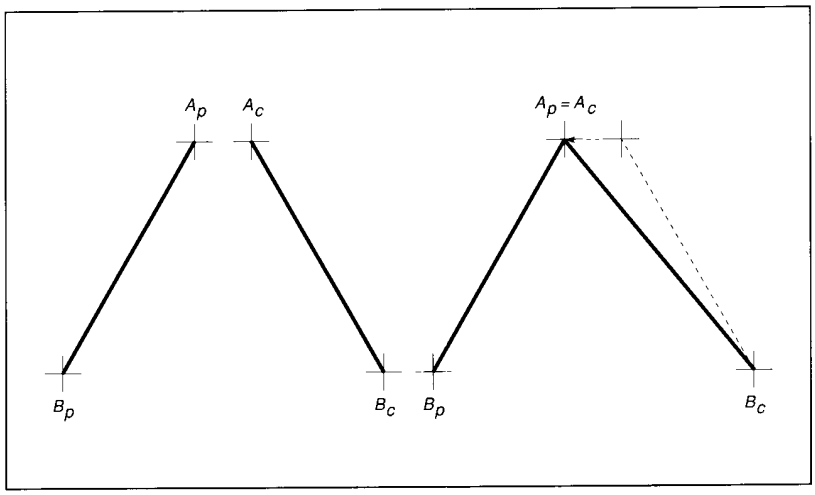

Fig. 15 Scenario two, before and after

Table 2 Scenario two

	parent locks						
state	Point A	Point B	Move	Size A	Size B	Angle A	Angle B
before	⊗	⊗	⊗	⊗	⊗	⊗	⊗
after	●	⊗	⊗	⊗	⊗	⊗	⊗
	child locks						
state	Point A	Point B	Move	Size A	Size B	Angle A	Angle B
before	○	●	○	○	○	○	○
after	●	●	○	○	○	○	○

● = set, ○ = unset, ⊗ = don't care

straight line-circular arc and straight line-composite curve combinations.

□ **Common axis:** vertical and horizontal lines are examined to see if they pass through the centre of a circular arc. Applies only to straight line-circular arc combinations.

The combination restrictions applied to these pairwise relations are imposed for two reasons. First, some relations applied to certain stroke combinations simply do not make sense; lines cannot be concentric, for example. Secondly, we recognise that, in some cases, relations could be made more useful by extending the number of permissible combinations. For example, lines could be allowed to be perpendicular to Bézier curves. One reason for omitting these combinations is development time, although another consideration is that checking for many unlikely combinations slows down the sketch analysis considerably for little expected further benefit to the user. We also recognise that some of the relations can be extended further; for example, lines of any slope could be allowed in the common axis relation.

6.3 Reducing the number of relations generated

With even a moderately simple sketch, the number of relations generated can be large. Each new relation needs to be enforced and this takes time. A number of methods have been explored to try to reduce the number of relations created, and hence the work load of the enforcer.

- *Time stamping:* each stroke created is given a time of birth. When the relation engine compares two strokes looking for possible relations, the times of birth of the two strokes are examined. If the time between each birth time is greater than a set period, this stroke pair is prevented from entering the relation engine.
- *Sequence stamping:* each stroke is given a unique positive number when it is created. Strokes created after a given stroke will have a unique positive number greater than that of previous strokes, but not necessarily in sequence (this is because Easel creates temporary strokes when breaking up curves at corners, discarding the original stroke). If the difference between the two strokes' sequence numbers is outside a predefined range, this stroke pair is prevented from entering the relation engine.
- *Boxing:* a number of relations rely on proximity, such as connected to, tangent to, touching, concentric and common point centre. When each stroke is created, a bounding box is computed which entirely encloses it (Fig. 10). If the two boxes enclosing the strokes fail to overlap, these relations are not considered for this stroke pair.

Both time and sequence stamping try to exploit a habit of designers when drawing, which is that related details of a sketch tend to be drawn in roughly the same time frame. A typical example is that an architect working on the

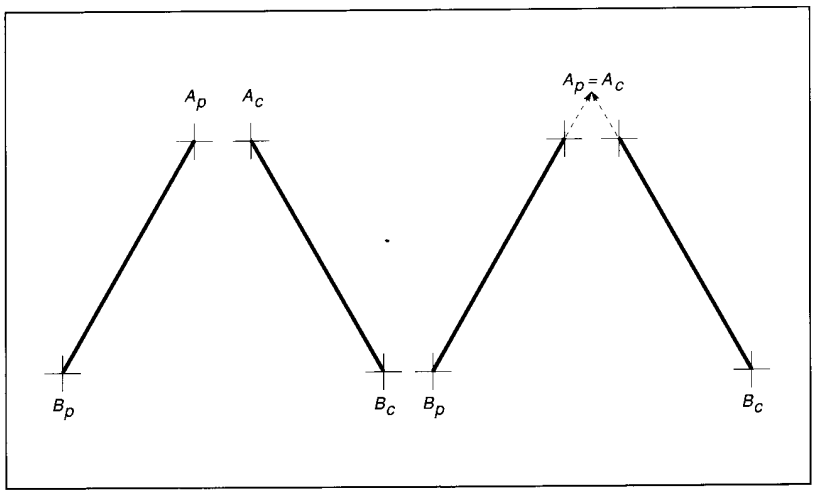

Fig. 16 Scenario three, before and after

detail of a chimney will be not concerned at the same time with details of the windows since these are separate entities. Time stamping can reduce the number of relations generated quite significantly, but suffers from one major limitation; eventually, the user will change attention to another part of the sketch. Relations between parts of the sketch that should be generated will not be whenever the time gap is too long. For example, the chimney should fit onto the roof in a certain way, but this will not be recognised if the chimney is created some time after the roof.

Sequence stamping looks only at the order in which relations are generated. This also reduces the number of relations generated by exploiting the fact that parts of sketches consist of strokes drawn one after another in order, but again suffers from the problem of the user moving attention to another part of the sketch as described in time stamping.

Essentially, the main difference between time and sequence stamping is that, in time stamping, the user is effectively given a time period in which to work on a section of a sketch, whereas in sequence stamping, the user is limited to a number of strokes. The two are quite clearly related; a complex section of a drawing involving a large number of strokes will take a comparatively long time to draw.

Another method might be to adopt active regions. In this idea, the user would be able to specify a region within which drawing would be limited. Relations would only be generated for strokes inside that region, and strokes outside would be ignored. Multiple regions could be allowed, with the user able to switch between them to determine which is active. This idea reduces the automatic ideal of Easel, but would certainly reduce the number of relations produced. It also raises the issues of what to do with overlapping regions, the shape of the regions and allowing the merging or splitting of regions.

Boxing reduces the number or relations produced and can also speed up the relation engine by eliminating comparisons which never yield new relations. Two types of boxing have been examined; global and discriminatory. Discriminatory boxing applies only to those relations that depend on close proximity to be true. The connected and tangent to relations are two obvious examples. This works very well for these relations, eliminating a series of unnecessary intensive computations. Global boxing for all types of relations reduces comparisons further, but can prevent the generation of needed relations. For example, two lines that are a comparatively long distance apart will not produce any relations although they may be parallel to each other.

Table 3 Scenario three

	parent locks						
state	Point A	Point B	Move	Size A	Size B	Angle A	Angle B
before	○	⊗	⊗	○	○	⊗	⊗
after	●	⊗	⊗	○	○	⊗	⊗
	child locks						
state	Point A	Point B	Move	Size A	Size B	Angle A	Angle B
before	○	⊗	⊗	○	○	⊗	⊗
after	●	⊗	⊗	○	○	⊗	⊗

● = set, ○ = unset, ⊗ = don't care

7 Enforcing relations

The enforcer takes the list of relations produced by the relation engine and progresses through them attempting to enforce, ideally, all the relations in the sketch.

7.1 Stroke locks

When the enforcer enforces a particular relation, it is important that the result of this action does not break any relation enforced previously. To prevent this from hap-

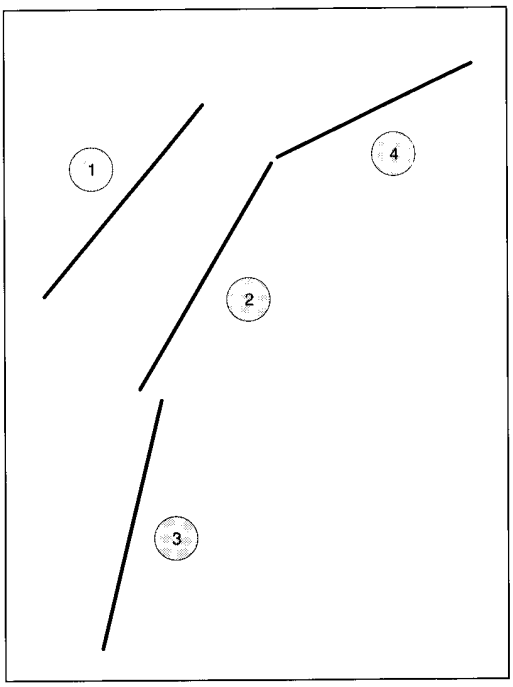

Fig. 17 Initial sketch

pening, Easel uses locks to fix certain stroke attributes, which are not allowed to be altered by subsequent enforcer processing. Easel maintains a fixed set of stroke locks, with each type of stroke using a subset of these. The subset is dependent on the type of stroke and is defined below for straight lines, circular arcs and composite Bézier curves.

7.1.1 Straight lines (see Fig. 11)

☐ **PointA, PointB:** these two locks refer to the position of the two end points of the straight line. For example, if PointA lock is set, the position of point A is not allowed to change.
☐ **Move:** set if another stroke touches this stroke.
☐ **SizeA, SizeB:** both of these locks refer to the length of the straight line. If either of these locks is set, the length of the straight line is not allowed to change.
☐ **AngleA, AngleB:** both of these locks refer to the slope of the straight line. If either of these locks is set, the slope of the straight line is not allowed to change.

The SizeA and SizeB locks refer to the same attribute (a straight line can only have one length). However, both are maintained to support the object-oriented view that the enforcer has of each stroke. This makes it easier to write the enforcer routines. A similar argument applies to the AngleA and AngleB locks.

7.1.2 Circular arcs (see Fig. 12)

• **PointA, PointB:** these two locks refer to the position of the two end points of the stroke. For example, if PointA lock is set, the position of point A is not allowed to change.
• **Move:** set if another stroke touches this stroke.
• **SizeA, SizeB:** both of these locks refer to the radius of the circular arc. If either of these locks is set, the radius of the circular arc is not allowed to change.
• **AngleA, AngleB:** these locks refer to the arc angles of the circular arc.

Two size locks, SizeA and SizeB, are supported to cope with the inclusion of ellipses at a later date.

7.1.3 Bézier curves (see Fig. 13)

☐ **PointA, PointB:** these two locks refer to the position of the two end points of the stroke. For example, if PointA lock is set, the position of point A is not allowed to change.
☐ **Move:** set if another stroke touches this stroke.
☐ **SizeA, SizeB:** do not apply to Bézier curves.
☐ **AngleA, AngleB:** these locks refer to the tangent direction of the curve at the stroke's end points.

7.2 General enforcer strategy

When the enforcer is requested to enforce a pairwise relation, it attempts first to alter only one of the strokes (the child) while leaving the other (the parent) alone. If the enforcer still cannot enforce the relation, only then is it allowed to alter the parent stroke. The state of a stroke's locks is called a stroke lock scenario, and for each possible stroke lock scenario, the enforcer needs to be told how to enforce a particular relation. These scenarios are ordered with the least restricting solution attempted first, i.e. the one which affects the least

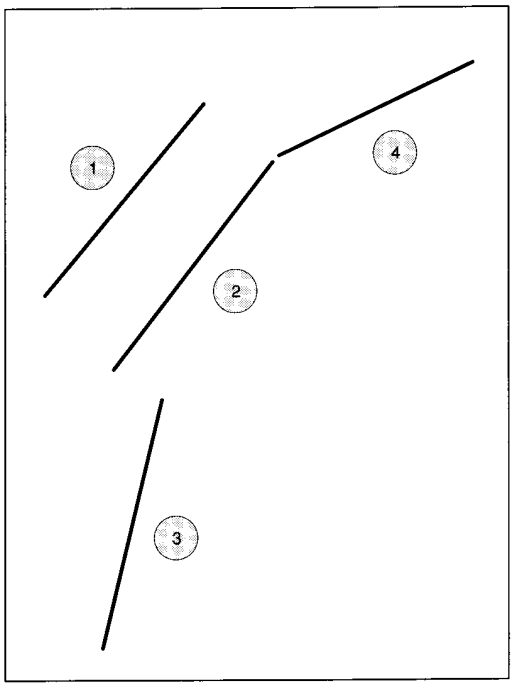

Fig. 18 After enforcing parallel relation only

number of stroke locks. This is to give the enforcer more flexibility when it comes to enforce other relations that this stroke may be part of.

As an example, we illustrate how the enforcer enforces a connected relation between two lines.

7.2.1 Scenario one: points A_c and B_c of the child straight line are moved such that A_c is now coincident with A_p. The PointA lock of the child is set, but the PointB lock is not affected since it is possible to move point B_c and still maintain this relation (Fig. 14 and Table 1).

7.2.2 Scenario two: the free point A of the child is moved to coincide with A_p, and the fixed point B_c remains in the same position. The PointA lock of the child is then set (Fig. 15 and Table 2).

7.2.3 Scenario three: the enforcer can no longer enforce a relation by altering the child alone, it now has to alter the parent as well. It does this by finding the intersection of the two lines and then moving point A of each straight line so that they become co-incident with it. The AngleA and AngleB locks of both lines are then locked (Fig. 16 and Table 3).

7.2.4 Other scenarios: currently, the enforcer has no more methods to directly enforce this relation, and so it fails, leaving the relation unenforced. However, if the enforcer gets to this stage, it will swap the roles of the parent and child in an effort to enforce a relation.

7.3 Enforcer methods

As well as a single relation, the enforcer also has to be told in what order the relations found in a sketch are to be enforced. This order is critical and can have a dramatic effect on the performance P, defined as

$$P = \frac{\text{number of enforced relations}}{\text{total number of relations}}$$

expressed as a percentage. A number of different algorithms have been tried.

7.3.1 First come: in this method, the enforcer simply attempts to enforce each relation in turn in the order that they are created by the relation engine. This method does not have a high performance, but is fairly fast since only one pass of the relations in the sketch is performed.

7.3.2 Rigid order: in this method, a list of relation types is maintained in a set order. The enforcer looks at this list and goes through the entire sketch enforcing all relations of a particular type in turn. This allows each relation to be given an effective priority, which reflects the importance of each relation type in a typical sketch. For example, the fixed list may contain the relation types in this order indicating that parallelism has a higher priority than connectivity:

parallel → connected to → equal length → · · ·

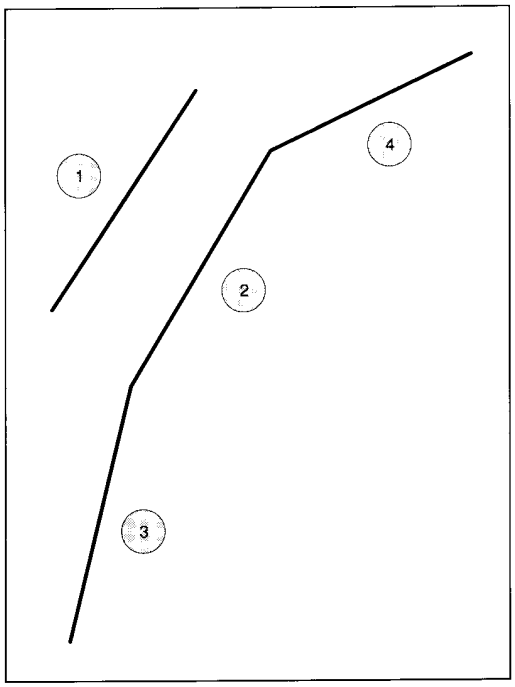

Fig. 19 After enforcing connected then parallel relations

This means that the enforcer would enforce all parallel relations in the sketch first, then all connected to relations, then all equal length relations, and so on. This method is easy to implement and fairly quick, but performance is poor, especially if there are a large number of different relation types. Low performance is a result of the rigid nature of this method and indeed of the first come algorithm. To maximise performance, a responsive method is required, capable of changing the order in which relations are enforced.

To see why, take the following part of a sketch as an example. In Fig. 17, there are lines numbered 1 to 4. Suppose, for example, that lines 3 and 4 are fixed (all their stroke locks are set). Assume also that the relation engine has decided that lines 1 and 2 should be parallel and that straight line 2 should be connected to lines 3 and 4.

Suppose all parallel relations are enforced first. This leads to the sketch in Fig. 18, in which lines 1 and 2 have been made parallel. This now means that the connected relations that exist between lines 2 and 4 and lines 2 and 3 cannot be enforced without breaking the parallel relation already enforced.

However, if the connected relation had been enforced first, straight line 1 could subsequently be moved later to enforce the parallel relation (Fig. 19).

The varying nature of sketches makes it difficult to choose a fixed order of enforcing relations, which makes this method unsuitable for practical sketches.

7.3.3 Certainty and order: the two previous methods make no attempt to intelligently decide which relation

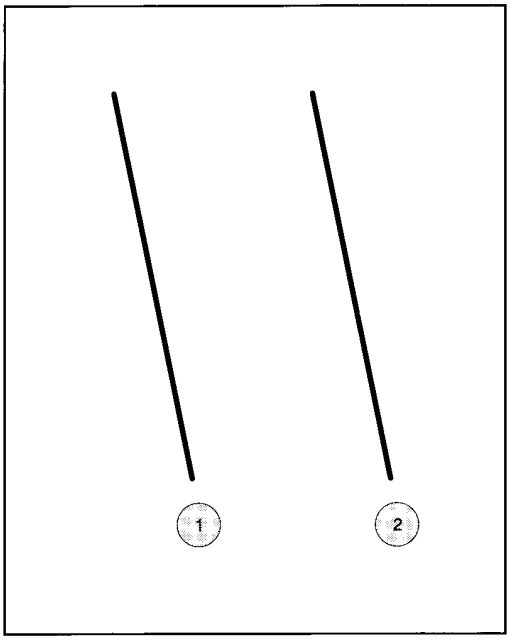

Fig. 20 Already enforced (before)

should be enforced next. However, some reasoning may be applied to the enforcer if the certainty and order of each relation are taken into account. When the relation engine generates a particular relation, it computes a probability (certainty) which indicates how 'strong' a relation is. The certainty of a relation reflects the confidence the relation engine has of the user's intentions when drawing the sketch. For example, with connectivity, the relation engine decides how far two end points are apart and compares this with the tuning error in the following way:

$$\text{certainty} = 1.0 - \frac{\text{distance between end points}}{\text{tuning error}}$$

End points that are very close together would be given a certainty close to 1.0, whereas end points farther away receive a certainty closer to 0.0. The enforcer then arranges the relations to be enforced in existence so that the highest certainty relations are attempted first. This method has a high performance and, on average, we are able to satisfy about 75% of all relations in a sketch. However, the performance may be lower. Sketches are not exact, and so a user may mislead this method into assigning an incorrectly high certainty by accident. Mouse slips are one obvious source of problems.

Certainty, however, takes no account of the fact that some relations have a naturally higher priority than others. Users, for example, are much better at connecting strokes together than they are at drawing lines which are tangents to curves. This implies that some sort of priority needs to be applied to the certainty as a weighting factor. The enforcer computes the order of a relation in the following way:

order = certainty of relation × priority of relation

As before, the enforcer then ranks the relations so that the highest order relations are attempted first. This method has a better performance than the certainty only method, but again suffers from two problems. First, input mouse slips are again a problem, but are reduced to a certain extent by the weighting effect of a relation's priority. This is because those relations that a user is good at generating, such as connected to, anchor to etc., are given a high priority, whereas other relations with a higher probability of mistake are given a lower weighting. Secondly, chooosing a priority for some relations can be difficult, and often involves trial and error. The value of a particular relation's priority is dependent on the users (how good they are at generating this relation) and the nature of the sketch (how often this relation is used).

7.3.4 Proximity: as mentioned before, users tend to work on a single area of a sketch at one time. Thus, it seems reaonable to assume that the proximity between strokes may give a clue as to the order in which relations are to be enforced. In this method, a box is drawn around a stroke completely enclosing it. The centre of this box is computed and said to be the stroke's 'centre of interest'. Each relation (except a unitary relation) has two strokes associated with it. The distance between the strokes' respective centres of interest is computed and is said to be the relation's proximity. The enforcer then arranges the existing relations so that the highest proximity relations are attempted first. This method is slower than the previous three methods, and it has a lower performance than the certainty method. The performance is very dependent on the types of existing relations. It works well for relations that depend on close proximity to be true (concentric, common axis and touching are obvious examples), but gives a false order when used with relations that may have large distances between

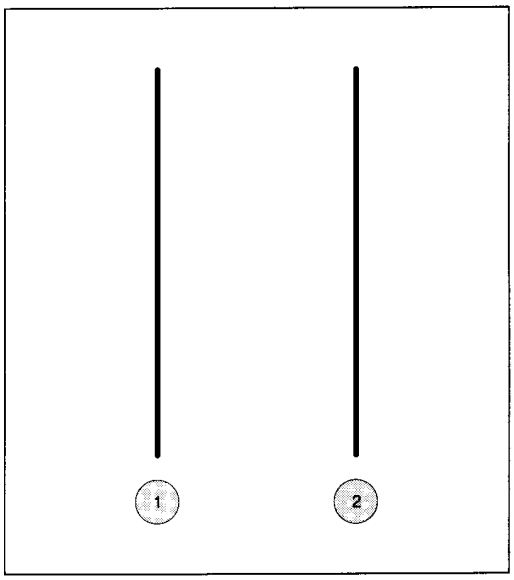

Fig. 21 Already enforced (after)

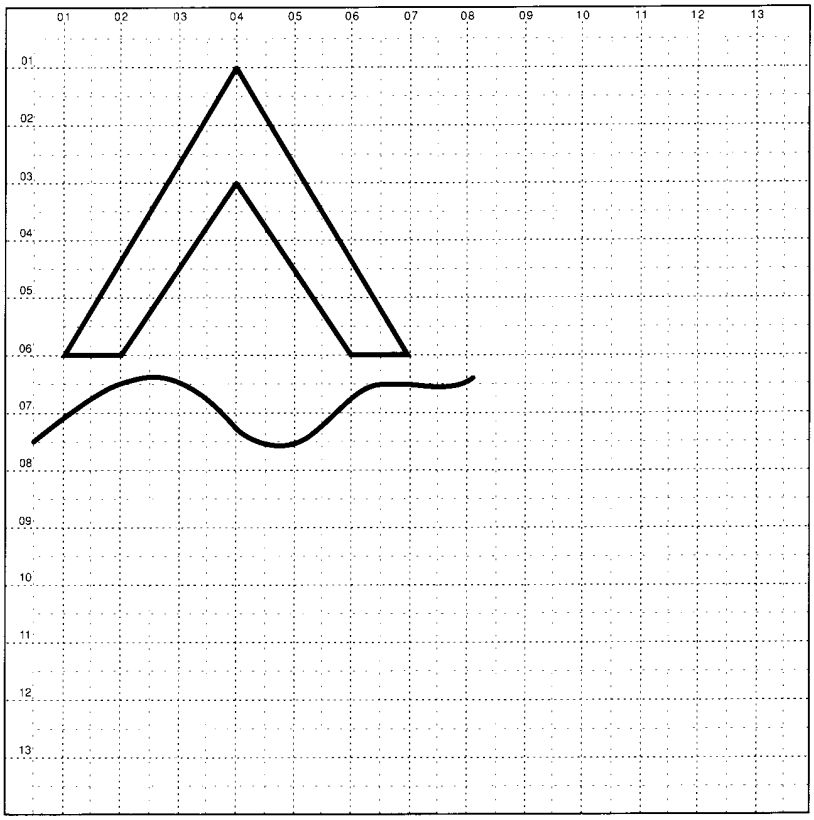

Fig. 22 Example sketch (after tidying up)

Table 4 Typical Easel diagnostic

memory used: 10736 byte(s)

sketch job times, seconds		entity totals		performance	
preprocess	0.000	objects	0	number of relations	37
process	1.780	strokes	7	enforced	3
generate unitary	0.030	points	244	indirectly	27
generate pairwise	2.030	relations	37	not enforced	0
enforce	1.210			unenforceable	7
				performance	81%
total	5.050				

them (such as parallel or equal length). It is also completely unsuitable for unitary relations such as anchor to or snap angle, since these relations only depend on a single stroke. This means that a separate ordering method is required to cope with these relations (currently, all unitary relations are enforced before all pairwise relations on a first-come-first-served basis).

7.3.5 Exhaustive search: the performance of the enforcer hinges on the order in which relations are enforced. Since the previous methods fail to yield a guaranteed 100% performance, an additional possibility may be to generate every different relation order and then pick the order with the highest performance. In this mode, the enforcer generates $N!$ different orders for the N relations in the sketch. The enforcer does not examine each of these possibilities for two reasons; first, it may come across a relation order which yields a sufficiently high performance (ideally 100% but, in practice, less than this); secondly, it can exceed a predefined limit on the number of different relation orders to try before giving up.

Ideally, of course, the permutator would be allowed to exhaust all the possible relation orders before giving up

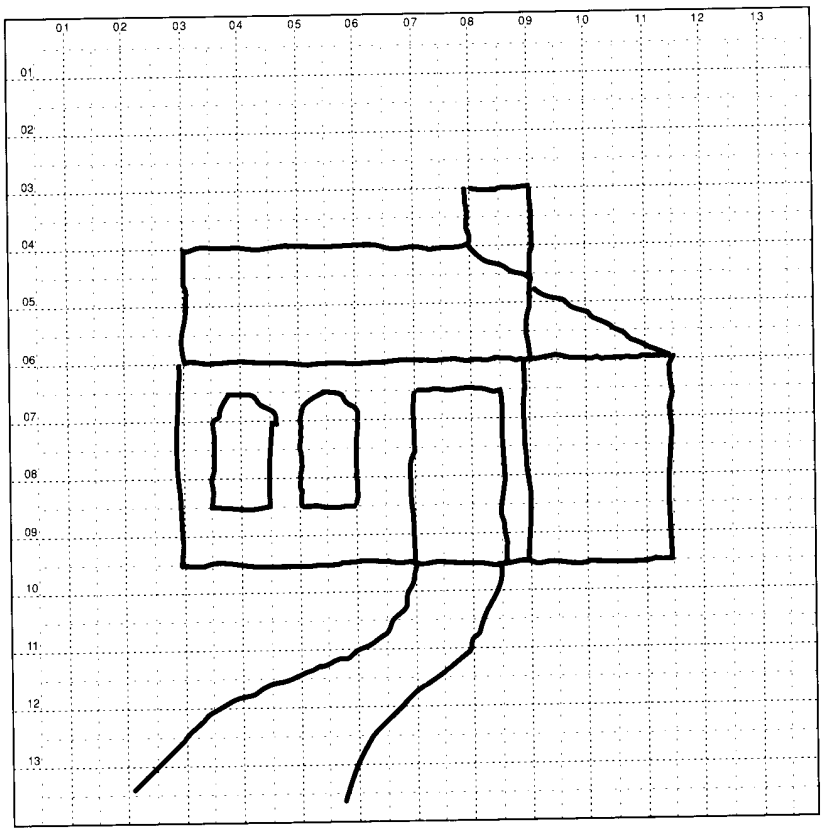

Fig. 23 Sketched house (before tidying up)

but the sheer number of possibilities makes this impractical even on a machine such as a Sun-IPC. However, consideration of a few hundred permutations can lead to an improvement in enforcer performance, albeit with a severe time penalty.

Realistically, this is not seen as a practical enforcer method, but could be used in conjunction with a more powerful system to analyse patterns in successful relation orders in a number of varied sketches, or as a last resort method when the performance of the previous methods is too low.

7.3.6 Already enforced short-cut: as stated in Section 7.1, the enforcer fails to enforce a relation when it has no tactic for enforcing a particular lock scenario for the strokes involved. However, it is possible that enforcing one relation indirectly enforces another (Fig. 20).

In this example, suppose the following relations are required to be enforced:

straight line 1 snapped to 90°
straight line 2 snapped to 90°
straight line 1 parallel to straight line 2

Assume that the first two relations are enforced first. This results in the AngleA and AngleB locks of both lines being set. This prevents the enforcer from enforcing the third relation, since it cannot alter the direction of either of the two lines, but, as can be seen from Fig. 21, this relation has been indirectly enforced by the previous two.

Performing an already enforced check can speed up the enforcer, since it does not need to attempt to enforce a relation which has been indirectly enforced. In addition, it increases the perceived performance of the enforcer as a greater number of relations are recognised as enforced.

There is a time penalty. This is because the enforcer has to examine each relation to see if it has been indirectly enforced. However, the code to perform this is significanly less intensive than the routines used to enforce a particular relation.

This method has a tuning value, which indicates how well a relation has been indirectly enforced. This value is non-zero because of limitations in mathematical precision of the enforcer routines and can be made quite large if a low resolution output device is used. The larger this tuning value, the greater the number of relations that the enforcer regards as being already enforced.

7.3.7 Resetting stroke locks: as explained above, the enforcer uses stroke locks to prevent it breaking a previously enforced relation. As the enforcer makes additonal passes through the relation list (after each new relation is added), more and more strokes become locked and the sketch becomes more rigid, reducing the opportunity of the enforcer to enforce relations. One way to give the enforcer a greater chance of enforcing relations is

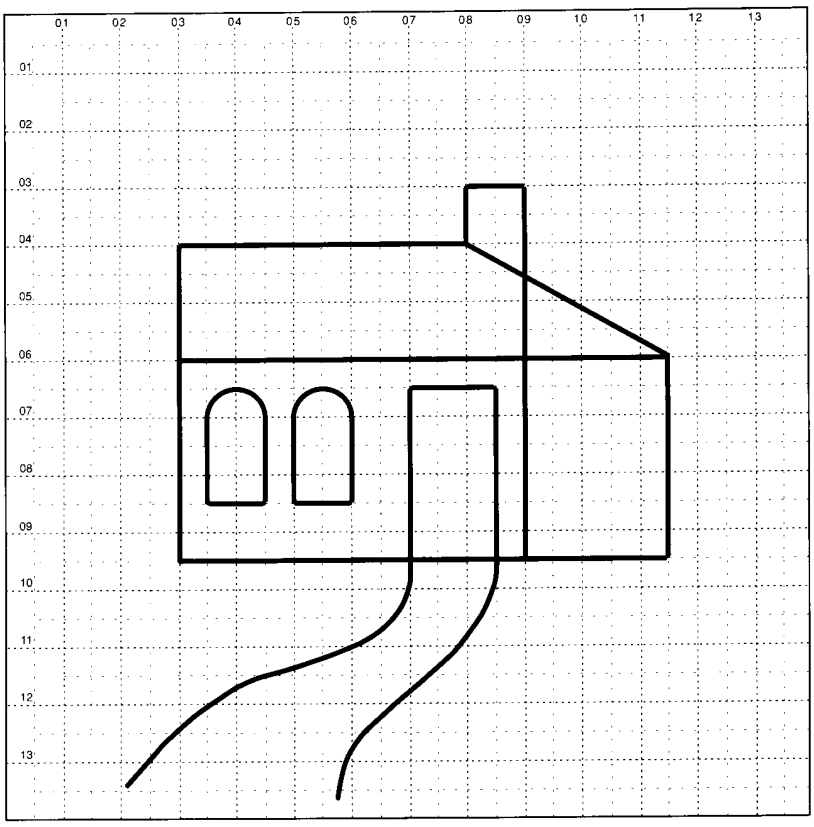

Fig. 24 Sketched house (after tidying up)

to give it a free hand each time it passes through the relation list. To do this, every stroke in the stroke list has its locks unset before each pass of the enforcer, irrespective of whether its associated relations have been enforced. This procedure is obviously done when the permutation method is used, but is optional on all the other enforcer methods. Resetting the stroke locks increases the performance of the enforcer (since it now has a greater choice of enforcer strategy), but takes longer for a pass (since all the relations in the sketch need to be enforced as opposed to only those extra relations created when a new stroke is entered).

7.4 Performance in practice

In this Section, we present three examples of sketches tidied up by Easel. Easel, as we have pointed out earlier, attempts to tidy up the sketch as the user draws, and so the times given for generating relations and then enforcing them are those as if Easel had been presented with a new sketch and asked to tidy up the entire sketch in one go. If a new stroke is added to a sketch, the time taken by Easel to generate all the new relations associated with this stroke is less than the time given. In addition, the enforcer times are very much quicker in practice, since it only has to attempt to enforce those new relations generated by this new stroke.

A simple example is given in Fig. 22. For this sketch, Easel provides the diagnostics dump shown in Table 4. A slightly more complex example is a simple sketch of a house (Fig. 23). It has 27 strokes and 655 relations. Easel takes around 20 seconds to generate the complete relation list, and a further 6 seconds to enforce 98% of them (Fig. 24).

A more adventuous example is shown in Fig. 25. This shows a tidied up sketch of the IEE lion, containing 41 strokes and 173 relations. Easel generates the relations in 43 seconds and manages to enforce 86% of these in another 1.5 seconds. This is longer than the house example earlier although there are less relations. This difference is mainly due to the number of Bézier curves present. Generally speaking, Easel takes much longer to process this type of primitive.

Fig. 26 shows the same sketch but with the grid removed and 'construction lines' added (control polygons and so on). Each □ indicates the end point of a primitive. Correctly connected primitives have these boxes overlapping. This Figure also illustrates how unenforced relations manifest themselves on screen. Fig. 27 shows the detail of the lion's left front paw. Easel has failed to connect the leading slanting line of the paw to the leading slanting line of the leg (the □s do not overlap).

Another example is shown in Fig. 28, in which the raised right leg of the lion has not been properly drawn (the end point of the Bézier curve should be touching the leg, instead it has been anchored to a grid intersection).

The user has a number of options when it comes to correcting this kind of mistake. The first and most

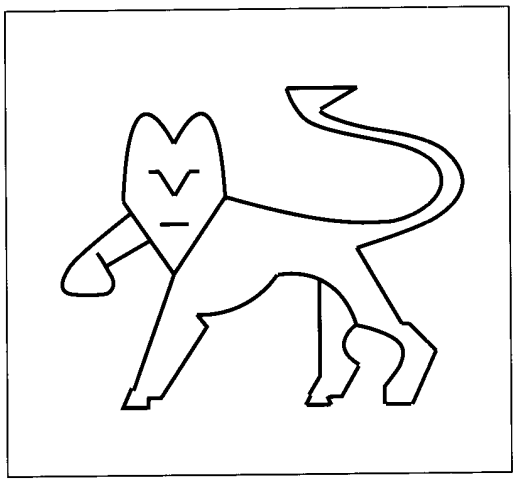

Fig. 25 IEE lion (after tidying up)

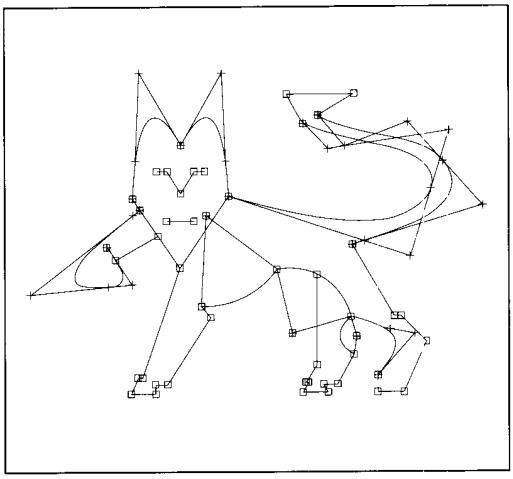

Fig. 26 IEE lion, with control polygons and marked end points (after tidying up)

obvious method is to edit or delete the stroke manually. In most cases, this works. However, in some examples, Easel can prove frustrating as it stubbornly refuses to accept the changes (this is because the new stroke is insufficiently different from the original and the same relation list is generated). This suggests that the user needs to be able to turn off the relation enforcer if desired. Another less obvious method is to leave the sketch alone and try each of the various enforcer algorithms until one gives the desired sketch. This again is not ideal and conflicts with the automatic principle of Easel.

8 Conclusions

Generally speaking, the preprocessing, processing, and postprocessing modules of Easel work well and are comparatively fast. Extending the processor to include new primitives, ellipses for example, should be straightforward. Detecting intended relations in a particular sketch also does not present any real problems, either in terms of speed or developing algorithms. Of course, in complex sketches, the number of relations grows quadratically with the number of strokes, limiting the complexity of sketches that can be drawn in practice. The real dificulty is in the enforcing of these relations. The performance of the enforcer is not 100%, but could be improved in a number of ways.

First, the enforcer could be provided with more ways to enforce a particuar relation. As the enforcer attempts to enforce a relation, its success is limited by the number of stroke lock scenarios to which it must react. This number is large (for seven different locks it is $2^7 = 128$). However, the number of alternatives can be reduced by noting, for example, that with a straight line-straight line combination SizeA and SizeB work as a single lock. The same is true for the AngleA and AngleB pair. Nevertheless, to code each of these is impractical, but implementing further cases would lead to an increase in enforcer performance.

Secondly, an improved method is required for deciding in which order relations should be enforced. This could be approached in two ways. First, an enforcer strategy may exist that guarantees almost 100% success for all sketches, although this seems unlikely. This would involve the computationally intensive process of analysing patterns in each possible enforcement order for many sketches. Secondly, a 'super enforcer' could be developed, incorporating some of the best qualities of each of the enforcer methods outlined here and other new methods, for example, ordering a particular relation with respect to its 'popularity' in a sketch (i.e. the ratio of the number of occurrences of a particular relation in a sketch to the total number of relations in a sketch). This may result in an improvement in enforcer performance, but is unlikely to lead to the elusive 100% success rate.

However, does the enforcer need to be 100% successful? It may be that Easel's interpretation of the user's sketch would be regarded as acceptable by the user even if its actual performance was only 75%, for example (this is particularly true in graphic art). Basically, we can consider Easel to be a success if the time taken to enter a drawing is reduced, allowing for the time a user takes to make edits to enforce missed relations or to correct

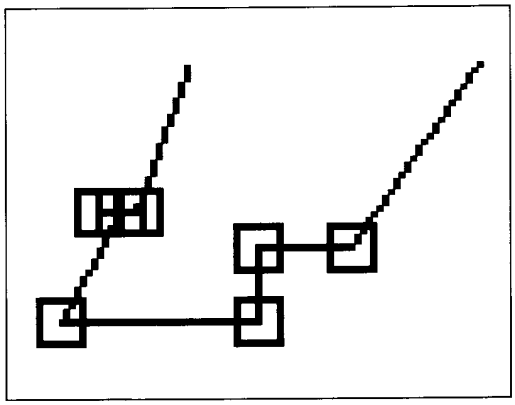

Fig. 27 Easel fails

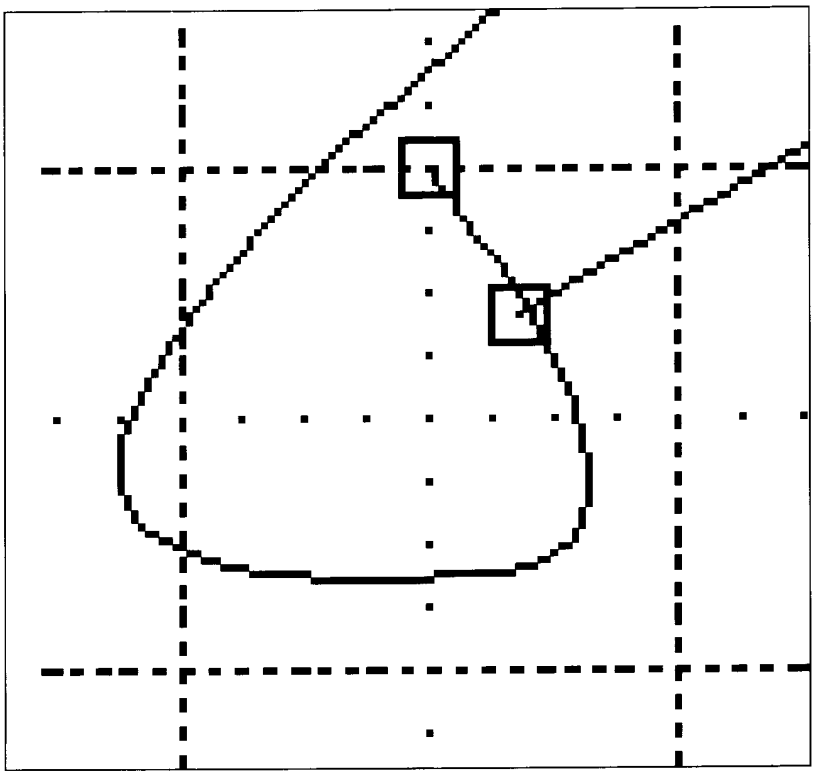

Fig. 28 Easel fails again

errors. A secondary, but major, point of importance is whether the users feel more at ease with Easel compared to a traditional draughting system, or whether they feel frustrated by its mistakes.

Currently, Easel's performance is simply not good enough for practical sketches consisting of perhaps hundred of elements. With the lion example (Fig. 25), adding an additional stroke results in a delay of around 40 seconds (while analysis and tidying up takes place) before control is returned to the user and sketching can continue. Analysis times are difficult to link to one quantity. The number of strokes and relations are obviously important, but so is the composition of the sketch. This can be seen by comparing the analysis times of the house (Fig. 24) and the lion (Fig. 25). The lion takes longer because it contains Bézier curves. Nevertheless, this is clearly unacceptable. However, optimising our algorithms for speed and exploiting future hardware improvements will ease this problem.

In conclusion, the results appear promising. A high enforcer success rate is achieved, which yields sketches that, on average, are good interpretations of the user's intention. However, in some cases, the enforcer can get it badly wrong.

9 References

[1] JENKINS, D. L., and MARTIN, R. R.: 'Weaknesses in shared memory and software interrupts in UNIX'. Proc. European Unix Users Group, Munich, Germany, Spring 1990, pp. 163–176

[2] SUTHERLAND, I. E.: 'Sketchpad, a man-machine graphical communication system'. PhD Thesis, Massachusetts Institute of Technology, 1963

[3] LIGHT, R. A.: 'Symbolic dimensioning in computer aided design'. MSc Thesis, Massachusetts Institute of Technology, 1979

[4] SUZUKI, H., ANDO, H., and KIMURA, F.: 'Geometric constraints and reasoning for geometrical CAD systems', *Comput. Graph.*, 1990, **14**, (2), pp. 211–224

[5] KASTURI, R., BOW, S. T., EL-MASRI, W., SHAH, J., GATTIKER, J. R., and MOKATE, U. N.: 'A system for the interpretaton of line drawings', *IEEE Trans.*, 1990, **PAMI/12**, (10), pp. 978–992

[6] SCHNEIDER, P. J.: 'Phoenix: an interactive design system based on the automatic fitting of hand-sketched curves'. MSc Thesis, University of Washington, 1988

[7] O'NEIL, P. V.: 'Advanced engineering mathematics' (Wadsworth Publishing, 1983) pp. 1091–1093

[8] PLASS, M., and STONE, M.: 'Curve-fitting with piecewise parametric cubics', *Comput. Graph.*, 1983, **17**, (3), pp. 229–239

[9] BORNING, A.: 'ThingLab — an object-oriented system for building simulations using constraints'. Proc. 5th Int. Conf. on Artificial Intelligence, August 1977, pp. 497–498

[10] PAVILIDIS, T., and VAN WYK, C. A.: 'An automatic beautifier for drawings and illustrations', *Comput. Graph.*, 1985, **19**, (3), pp. 225–234

The paper was first received on 1 August 1991 and in revised form on 14 January 1992.

The authors are with the Department of Computing Mathematics, PO Box 916, University of Wales College of Cardiff, Cardiff CF2 4YN.