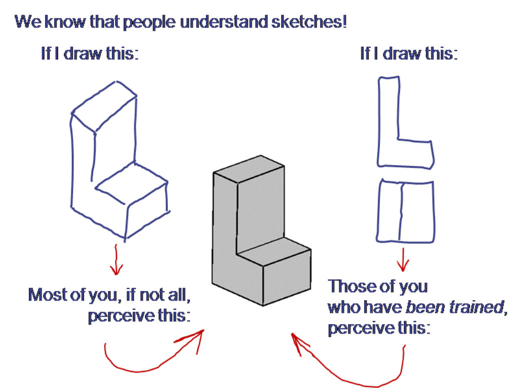
Sketch Input of Engineering Solid Models: Tutorial Notes

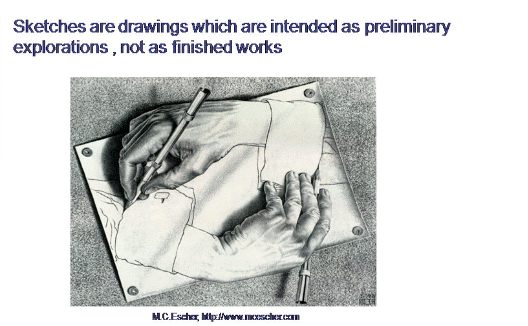
P. Company and P.A.C.Varley
Department of Mechanical Engineering and Construction, Universitat Jaume I,
Castellón de la Plana, Castellón, Spain
{pcompany, varley}@emc.uji.es

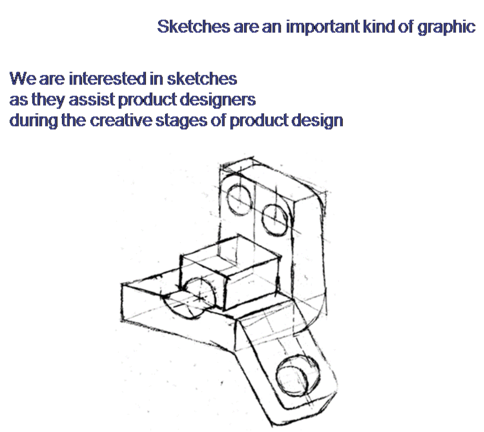
Abstract
In this tutorial, we describe the state of the art of sketch input of engineering solid models. Firstly, we show that sketching has historically been an important aspect of engineering culture. We then discuss and classify various current approaches to computer interpretation of sketches. We present our selection of the most important algorithms used for interpreting sketches of engineering objects. Finally, we discuss some of the most interesting open problems.

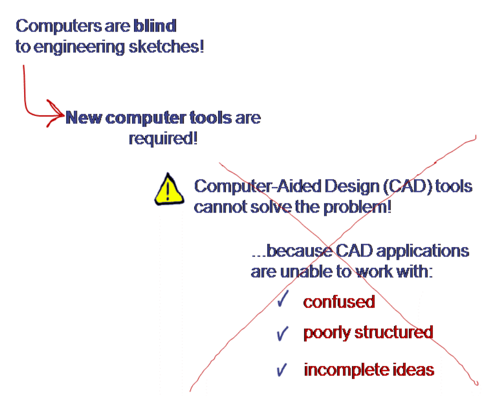
1. Introduction

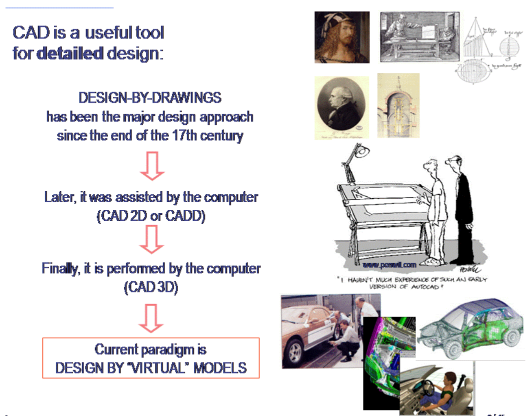
Slides 2 to 15 are a summary of our paper *The Importance of Sketching in Engineering Culture* [VC08].

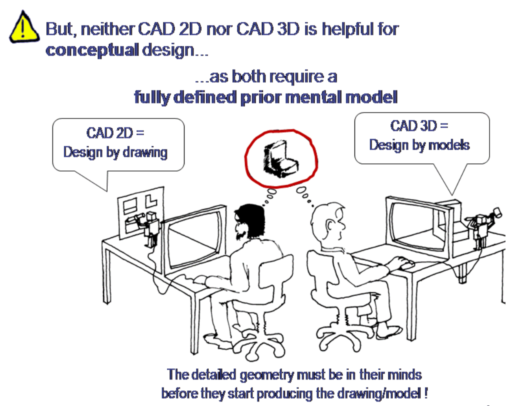


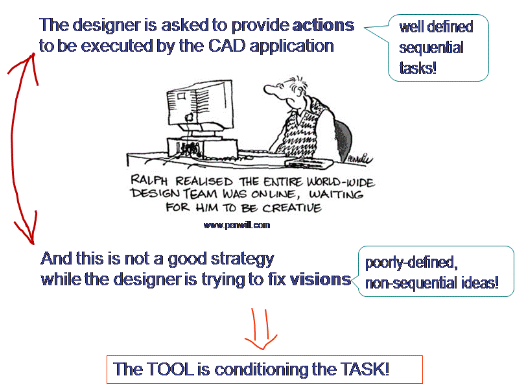


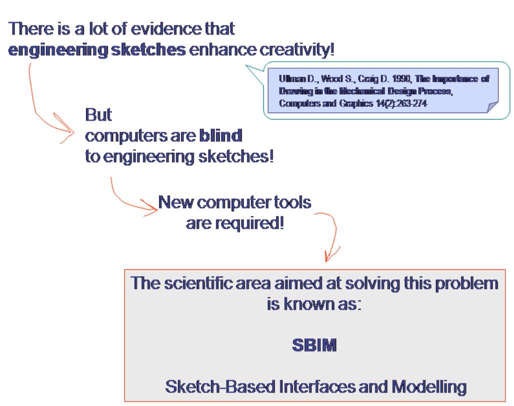


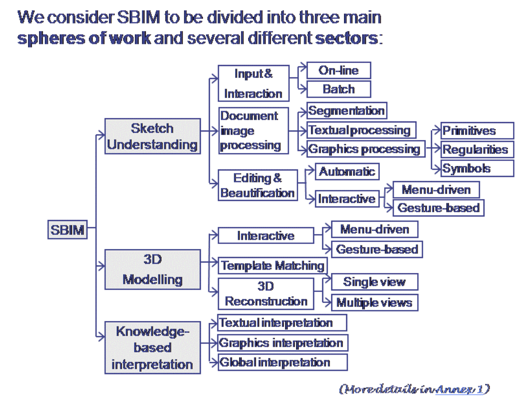


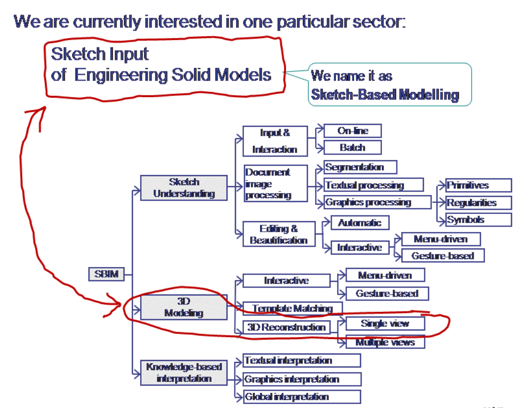


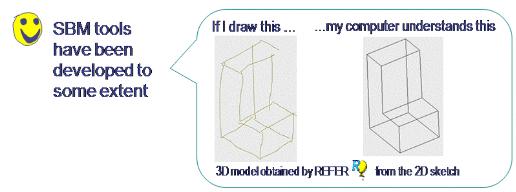


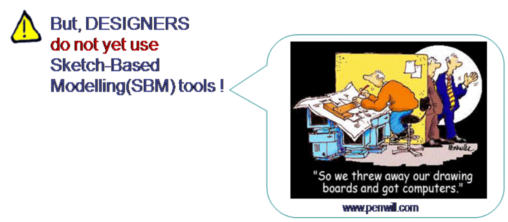












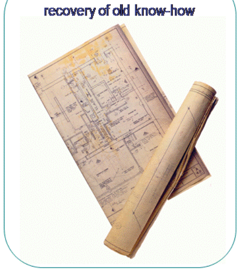
Slides 16 to 22 present the historical background to computer interpretation of engineering drawings. This information has been collated from a number of sources including [Com04], [Com07] and [CCV09].

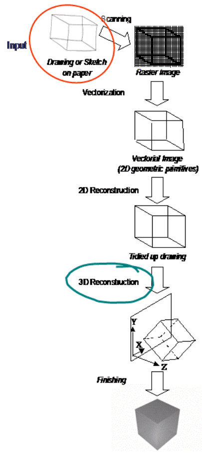
What we now know as
Sketch-Based Modelling...

...comes from what was
formerly known as
Geometrical Reconstruction

The former goal of geometrical reconstruction was extracting information from old engineering blueprints

In other words, "archaeological" recovery of old know-how



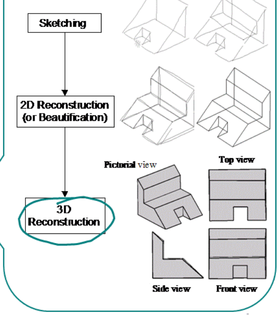


The main **goal** of the **reconstruction community** changed in the 1990s

Nowadays, most of the systems are oriented toward **conceptual design**

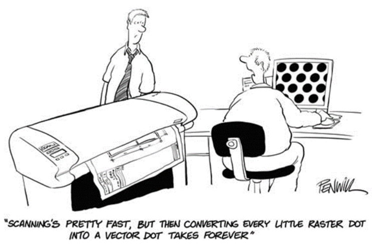
via **sketch-based modelling**

using sketches generated by the user as input data



But the task proved difficult…

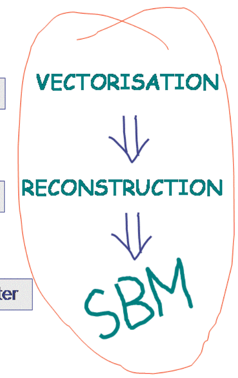
…because the vectorisation stage is complex…



www.penwill.com

The goal has changed over time:

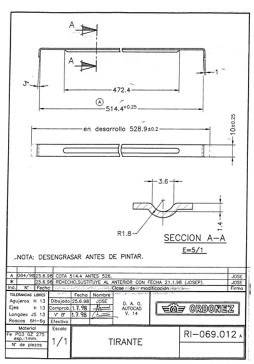
2D + paper	⇨ 2D + computer
2D + paper	⇨ 3D + computer
Conceptual design	⇨ 3D + computer



…and because engineering drawings convey:

- ✗ 3D information represented through *complex* views
  - main orthographic views, particular views, cuts, etc.
- ✗ annotations
  - dimensions, tolerances, etc.

Dori D.; Tombre K. (1995) From engineering drawings to 3D CAD models: are we ready now? Computer-Aided Design 27, pp. 243-254



Slides 23 to 43 present a taxonomy of existing sketch-based modelling tools, based on a similar taxonomy presented in 2004 [CPC04].

The short term problem was solved trough brute force:

⚠ Although this goal is still alive in architecture:

Xuetao Y., Wonka, P., Razdan, A. (2009) Generating 3D Building Models from Architectural Drawings: A Survey. IEEE Computer Graphics and Applications, 29 (1), 20-30

The current situation in **producing solid models from sketches** may be summarised as follows:

There is **no general approach** which solves all the SBM problems (More details in Annex 2)

→ Some **critical features** produce different bottlenecks

→ **States of the art are different** for every critical feature

We propose a taxonomy of **critical features** !

The features we consider critical are:

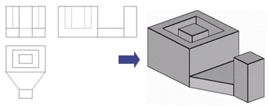
(More details in Annex 3)

Company P., Piquer A., Contero M. and Naya F. (2005) A Survey on Geometrical Reconstruction as a Core Technology to Sketch-Based Modeling. Computers & Graphics. Vol. 29, No 6. pp. 892-904.

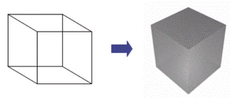
1. Number of views
2. Types of surface
3. Variety of inputs
4. Design intent

Two kinds of **VIEW** are distinguished for reconstruction approaches:

√ multiple orthographic views



√ single pictorial view



(More details in Annex 4)

More active in the beginning, less active now

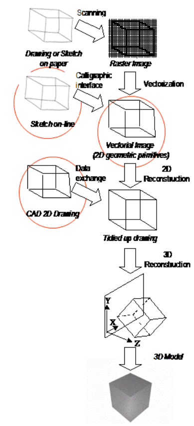
Year	Authors
1973	Idesawa
1976	Lafue
1981	Wesley & Markowsky
1982	Haralick & Queeney
1983	Sakurai
	Aldefeld
1984	Preiss
	Aldefeld & Richter
1986	Gu et al
1988	Chen & Perng
1989	Gujar & Nagendra
1992	Chen et al
1993	Meeran & Pratt
1994	Yan et al
1995	Ah-Soon & Tombre
	Lysak et al
1996	You & Yang
1997	Masuda & Numao
	Shum et al
1998	Kuo
	Shin & Shin
	Tanaka et al
1999	Suh et al
	Sastry et al
2001	Liu et al.
	Shum et al.
2002	Geng et al.
2003	Soni & Gurumoorthy
2004	Zhang et al.
2005	Lee & Han

INPUT comprises:

1 perfect line drawings

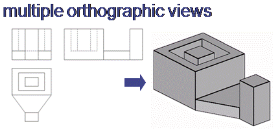
2 line drawings containing some "geometrical" mistakes

3 freehand sketches

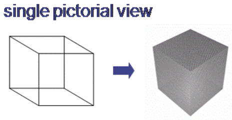


Two kinds of **VIEW** are distinguished for reconstruction approaches:

√ multiple orthographic views



√ single pictorial view



(More details in Annex 4)

More active nowadays

Year	Authors
1963	Roberts
1968	Guzman
1971	Huffman
	Clowes
1973	Mackworth
1975	Waltz
1978	Sugihara
1980	Kanade
1982	Sugihara
1986	Sugihara
1987	Malik
	Wei
1989	Wang and Grinstein
1990	Lamb and Bandopahay
1991	Marill
1992	Wang
	Leclerc and Fischler
1993	Wang and Grinstein
	Marti et al.
1994	Branco et al
	Shimshoni and Ponce
1995	Grimstead and Martin
1996	Grimstead and Martin
	Lipson and Shpitalni
	Parodi
	Brown and Wang
1999	Company et al
2000	Varley and Martin
2001	Varley and Martin
2002	Ros and Thomas
2003	Oh and Kim
	Varley et al
2004	Kang et al.
	Company et al

All three input types have been studied, but…

…perfect line-drawings were the most frequent in the beginning …

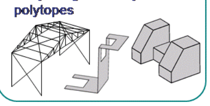
…now (in single view approaches) we are evolving towards hand-drawn line-drawings

Year	Authors	Input: Perfect line-drawing	Input: Imperfect line-drawing	Input: Sketch
1963	Roberts		*	
1968	Guzman	*		
1971	Huffman	*		
	Clowes	*		
1973	Mackworth	*		
1975	Waltz			
1978	Sugihara	*		
1980	Kanade	*		
1982	Sugihara		*	
1986	Sugihara		*	
1987	Malik	*		
	Wei		*	
1989	Wang and Grinstein	*		
1990	Lamb and Bandopahay			*
1991	Marill	*		
1992	Wang		*	
	Leclerc and Fischler	*		
1993	Wang and Grinstein	*		
	Marti et al.		*	
1994	Branco et al			*
	Shimshoni and Ponce		*	
1995	Grimstead and Martin			*
1996	Grimstead and Martin			*
	Lipson and Shpitalni			*
	Parodi	*		
	Brown and Wang	*		
1999	Company et al		*	
2000	Varley and Martin			*
2001	Varley and Martin			*
2002	Ros and Thomas		*	
2003	Oh and Kim			*
	Varley et al	*		
2004	Kang et al.			*
	Company et al			*

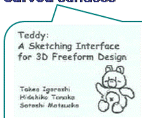
Our classification distinguishes two kind of **SURFACE**:

√ algorithms which only accept flat surfaces

They are generically known as polytopes



√ algorithms which accept curved surfaces



(More details in Annex 5)

http://www-ui.is.s.u-tokyo.ac.jp/~takeo/teddy/teddy/teddy.html

Rivers, A., Durand, F., Igarashi, T. (2010) 3D modeling with silhouettes. ACM Transactions on Graphics 29 (4), art. no. 109

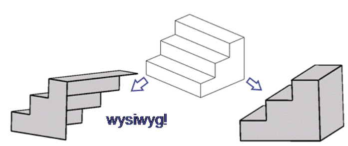
Roth-Koch S. and Westkaemper E. (2010) The implementation of a sketch-based virtual product development. Prod. Eng. Res. Devel. 4:175–183

Use of **HIDDEN LINES** in the input drawing results in two different inputs:

wireframes (transparent models) ⇔ **natural (opaque models)**

methods where the input includes all lines in the drawings

methods which reconstruct from an input which only contains the visible edges



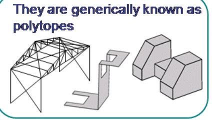
All lines must be drawn in the input, but generally there is no need to distinguish between visible and hidden lines

The system generally infers the rear part of the model after reconstructing the front part

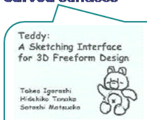
Our classification distinguishes two kind of **SURFACE**:

√ algorithms which only accept flat surfaces

They are generically known as polytopes



√ algorithms which accept curved surfaces



(More details in Annex 4)

http://www-ui.is.s.u-tokyo.ac.jp/~takeo/teddy/teddy/teddy.html

Both have been studied, but planar surfaces are more developed

Year	Authors	Surface: Planar	Surface: Curve
1973	Idesawa	*	
1976	Lafue	*	
1981	Wesley & Markowsky	*	
1982	Haralick & Queeney	*	
1983	Sakurai	*	*
	Aldefeld	*	*
1984	Preiss	*	*
	Aldefeld & Richter	*	*
1986	Gu et al	*	*
1988	Chen & Perng	*	
1989	Gujar & Nagendra	*	
1992	Chen et al	*	
1993	Meeran & Pratt	*	*
1994	Yan et al	*	
1995	Ah-Soon & Tombre	*	*
	Lysak et al	*	*
1996	You & Yang	*	*
1997	Masuda & Numao	*	*
	Shum et al	*	*
1998	Kuo	*	*
	Shin & Shin	*	*
	Tanaka et al	*	*
1999	Suh et al	*	
	Sastry et al	*	
2001	Liu et al.	*	*
	Shum et al.	*	*
2002	Geng et al.	*	*
2003	Soni & Gurumoorthy	*	*
2004	Zhang et al.	*	*
2005	Lee & Han	*	*

Natural drawings have been less studied than wireframes

The need to infer the rear of the object makes the reconstruction process more difficult

Year	Authors	2D hidden lines: Yes	2D hidden lines: No
1963	Roberts		*
1968	Guzman		*
1971	Huffman		*
	Clowes		*
1973	Mackworth		*
1975	Waltz		
1978	Sugihara	*	
1980	Kanade		*
1982	Sugihara		*
1986	Sugihara		*
1987	Malik		*
	Wei		*
1989	Wang and Grinstein		*
1990	Lamb and Bandopahay		*
1991	Marill	*	
1992	Wang		*
	Leclerc and Fischler	*	
1993	Wang and Grinstein		*
	Marti et al.	*	
1994	Branco et al		*
	Shimshoni and Ponce		*
1995	Grimstead and Martin		*
1996	Grimstead and Martin		*
	Lipson and Shpitalni	*	
	Parodi		*
	Brown and Wang		*
1999	Company et al	*	
2000	Varley and Martin		*
2001	Varley and Martin		*
2002	Ros and Thomas		*
2003	Oh and Kim	*	
	Varley et al		*
2004	Kang et al.	*	
	Company et al	*	

Design Intent and CAD have been linked for many time

However, the definition of Design Intent is ambiguous

Back in 1989 Design Intent was associated with design constraints and the methods of manipulating design constraints during product design activities

Kimura F. And Suzuki H. (1989) A CAD System for Efficient Product Design Based on Design Intent. CIRP Annals - Manufacturing Technology, 38 (1), 149-152.

...it still continues to be for many people !

When CAD people use the word "design", they usually mean "model"

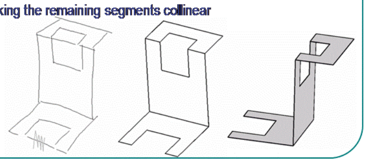
http://www.dezignstuff.com/blog/?p=3612

Modelling is just representing the design in some way

Design intent equates to the phrase Design for Change
This implies that you are modelling a concept that can be flexible through changes

Something has been done in the SBM sector to cope with design intent understood as design-for-change

Sketching one single line and then removing the central segment implicitly conveys the design intent of making the remaining segments collinear



However, no practical approaches have yet considered the explicit capture of complex design intent from the input sketches!

We understand design intent as a mix of:

- √ Geometry — ...as far as it is linked to the shape
- √ Psychology — ...as far as it is not always explicit in the sketches
- √ Engineering — ...as far as it is linked to the function

We understand design intent as a mix of:

- √ Geometry
- √ Psychology
- √ Engineering

When geometry dominates, design intent is mainly conveyed through geometrical features

which have already been studied as "**regularities**"

Lipson H, Shpitalni M. (1996) Optimization-based reconstruction of a 3D object from a single freehand line drawing. Computer-Aided Design, 28(8) 651-663

Yuan S., Tsui L.Y., Jie S. (2008). Regularity selection for effective 3D objects reconstruction from a single line drawing. Pattern Recognition Letters 29 (10), 1486-1495

Li M, Langbein F.C. and Martin R.R. (2010) Detecting design intent in approximate CAD models using symmetry. Computer-Aided Design 42 (3) 183-201

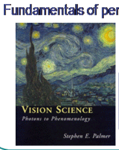
We understand design intent as a mix of:

- √ Geometry
- √ Psychology
- √ Engineering

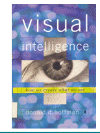
Psychology: Information not explicitly included is perceived through "**perceptual cues**"

sometimes clues

Fundamentals of perceptual cues have been studied:



Palmer SE. Vision Science. Photons to Phenomenology. Cambridge, MA: The MIT Press, 1999



Hoffmann D. Visual Intelligence. How we create what we see. New York: WW Norton & Company, 2000

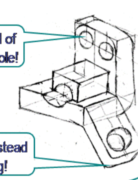
We understand design intent as a mix of:

- √ Geometry
- √ Psychology
- √ Engineering

When function dominates, design intent is mainly conveyed through "**engineering features**"

Drill, instead of cylindrical hole!

Round, instead of blending!



Consequently, we can define **Design Intent** as:

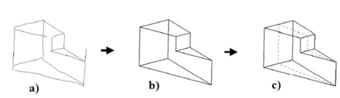
The set of intentions in sketches conveyed though **cues**, which, when perceived, reveal **regularities** or **features** of the object

⚠ Just a few of them have already been studied

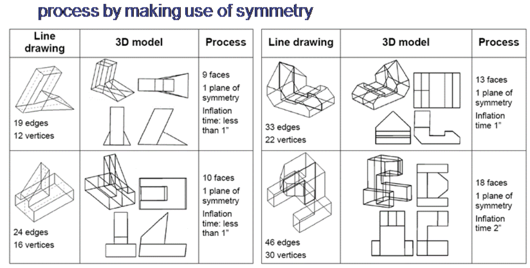
- √ Edge parallelism
- √ Face planarity
- √ ...

Example:

Early detection of symmetry in a 2D line-drawing



and improvement of the reconstruction process by making use of symmetry



⚠ WIMP user interfaces
are not appropriate for
conceptual design stages

But SBM tools
are not yet used

Summary

→ SBM tools look suitable, but need improvement

→ Roughly speaking, there are two categories of problem:
- ✓ Problems where a reasonably good solution exists (...although some improvements are still required)
- ✗ Open problems

→ Our taxonomy helps in finding critical features which must be studied further:
1. Number of views
2. Types of surface
3. Variety of inputs
4. Design intent

2. Wireframe Drawings

We shall describe algorithms representative of the current state of the art in these stages:

1. 2D sketching — Learn more on **segmentation:** Xiong, Y. LaViola J. (2010) **A ShortStraw-Based Algorithm for Corner Finding in Sketch-Based Interfaces**, *Computers and Graphics*, 34(5):513-527 *(See a demo in Demo 1)*
2. 2D beautification or tidying up
3. Extraction of geometrical and perceptual information — We shall describe our algorithm for **finding faces**
4. Inflating a rough 3D model — We shall describe our algorithm for **inflating quasi-normalons**; We shall describe our algorithm for **optimisation-based inflation**
5. 3D model refinement — We shall describe our algorithm for **finding rounds and fillets**

Slides 5 to 22 are a summary of our paper *A New Algorithm for Finding Faces in Wireframes* [VC10], explaining both why a new algorithm was needed and how the new algorithm works.

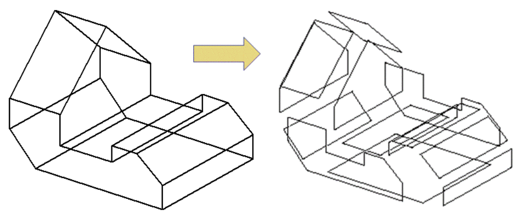
Perception is the stage where information required to produce a 3D model out of the 2D input is sought

Relevant information may be:
- explicit — e.g. edges connected to the same vertices. Not so difficult!
- implicit — e.g. faces of a polyhedral shape. Not so easy! Some heuristics are required to extract this information!

Let us study the finding faces problem!

Problem:

Given a wireframe line drawing of a polyhedral object, determine a list of loops of edges which correspond to faces of the object



If we have accurate 3D coordinates for the vertices, it is not so difficult

G. Markowsky and M.A. Wesley, 1980. **Fleshing Out Wire Frames**, IBM Journal of Research and Development, 24(5) 582–597.]

Without accurate 3D coordinates, it is not so easy

Shpitalni and Lipson determine all possible sets of loops of edges, and then use heuristics to pick the best one

⚠ This works, more or less, but is very slow (of the order of *days*)

M. Shpitalni and H. Lipson, 1996. **Identification of Faces in a 2D Line Drawing Projection of a Wireframe Object**, IEEE Transactions on Pattern Analysis and Machine Intelligence 18(10), 1000–1012.

⇔

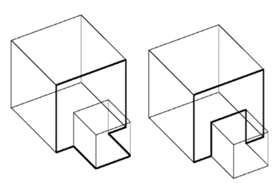
Liu and Tang use a genetic algorithm

⚠ This also works, more or less, but it can never be fully reliable (it is driven by random numbers)

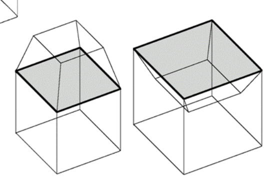
It also needs a lot of tuning

J. Liu and X. Tang, 2005. **Evolutionary Search for Faces from Line Drawings**, IEEE Transactions on Pattern Analysis and Machine Intelligence 27(6), 861–872.

So what are the difficulties?

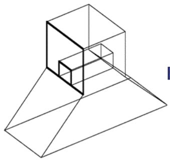


Loop Ambiguity: Wrong and Right Choices

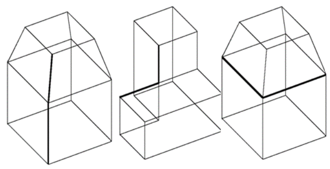


Internal Faces

More difficulties

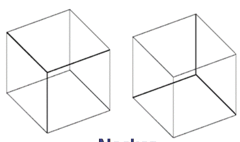


Multiple circuits in a planar edge subgraph

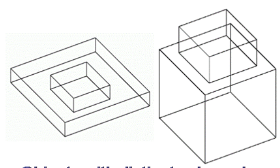


Edge pairs not in true faces

More difficulties



Necker reversal

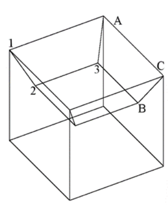


Objects with distinct subgraphs

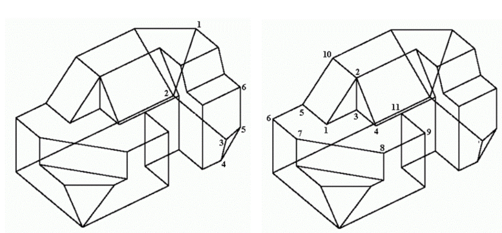
It is a graph theory problem

↳ Why not use Dijkstra's Algorithm (or something like it) to pick off loops one by one?

↳ This often works, but sometimes leads to problems



More problems with Dijkstra's Algorithm approaches



So what is going wrong?

The fundamental problem with Dijkstra's Algorithm approaches is that they assume a fixed cost for traversing any edge, irrespective of the route taken to reach the edge

We do not want this
we want the cost of traversing an edge to be a function of how well it fits in with any particular loop, taking into account the route taken to reach the edge

What we want is a graph algorithm which allows for the cost of traversing an edge to be context-dependent

We could not find one in the literature, so we came up with our own

Our new algorithm:

Data Structures:

Strings are *concatenated* sequences of half-edges

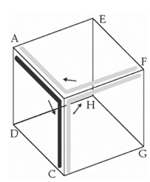
The shortest possible *strings* are single half-edges

Operations:

Two *strings* can be *concatenated* if the final vertex of the first *string* is also the start vertex of the second *string*, except that:

1. two *strings* cannot be *concatenated* if any other vertex appears in both *strings*
2. two *strings* cannot be *concatenated* if the new triple of three consecutive vertices appears in reverse order in any existing face or already-concatenated *string*

Data Structures and Operations (examples)



Starting with only the half-edges, we can concatenate AH and HC to give AHC

Once we have AHC, we cannot concatenate CH and HA

In fact, the only possible concatenation of CH is with HF, to give CHF

Similarly, the only possible concatenation of HA is with FH, to give FHA

Data Structures (continued):

Cyclisation is a double-*concatenation* where the final vertex of each *string* is the same as the start vertex of the other *string*

Cyclisation produces faces

Operations (continued):

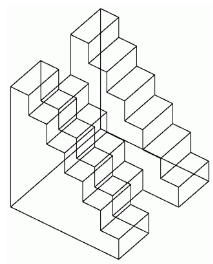
the same rules apply to *cyclisation* as to *concatenation*

since the purpose of the algorithm is to produce faces, *cyclisation* takes priority over other operations

Finding small (often quadrilateral) faces is easy ⇔ Finding large (often irregularly-shaped) faces is not so easy – there is more opportunity to go wrong

So we want to find small faces first, leaving the larger faces until the end

But how do we know which face loops are going to be the small ones?



The procedure may be summarised as follows:

1 Start with several seeds

2 Add to each seed simultaneously (or in turn) until one of them (the smallest) turns out to be a face loop

Don't worry if one of them takes a wrong turn somewhere (one of the others will generally finish first)

3 When we have a face loop, discard the rest and start again

4 We implement this by maintaining two lists of strings:

- √ The master list records only things which must be true
- √ The working list is used to explore hypotheses

When the hypotheses produce a face loop, add this to the master list and throw away the rest of the current working list

(More details in ...) Varley P.A.C. and Company P. (2010) A new algorithm for finding faces in wireframes. Computer-Aided Design 42 (4), 279-309

The resulting algorithm is short and easy to implement:

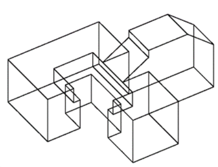
(Top level of algorithm)

- Create initial master string list, two entries per edge
- Assign priorities to all strings in the list
- Choose a trihedral vertex, and concatenate two strings at this vertex
- While there are strings remaining in the master list
  - Examine the master list for the presence of forced concatenations
  - If there are forced concatenations, perform them
  - Otherwise, examine the master list for the presence of voluntary mergers
    - If there are voluntary mergers, perform them
    - Otherwise, examine hypotheses (see next column)

(Subroutine: Examine Hypotheses)

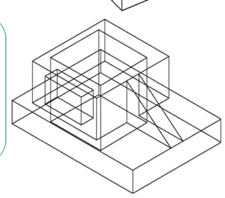
- Take a working copy of the master string list
- Repeat
  - Take the highest-priority string *S* in the working string list
  - Find the string *T* which has the best *mating value* with *S*
  - Concatenate *S* and *T*, and reduce the priority of the resulting string
  - Repeat
    - Examine the working string list for the presence a forced concatenation
    - If there is a forced concatenation, perform it
      - If the forced concatenation created a new face, update the master list accordingly and exit this subroutine
    - If there is no forced concatenation
      - If there is a voluntary merger available, create the face, update the master list accordingly and exit this subroutine
      - Otherwise exit this inner loop

When tested on 84 drawings, the new algorithm got them all right except for these two:



A previous approach using Dijkstra's Algorithm failed altogether on 19 drawings and got the wrong answer on another 2

The other previous state-of-the-art approach, a genetic algorithm by Liu and Tang, has not been tested on complex drawings such as the two for which our new algorithm fails – every drawing which the genetic algorithm processes correctly is also processed correctly by the new algorithm

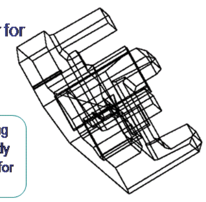


How fast is it?

√ The algorithm is polynomial - counting loops would suggest a worst-case performance of $O(n^5)$

In practice, processing a sequence of similar drawings, the time complexity is around $e^{2.7}$ (where e is the number of edges)

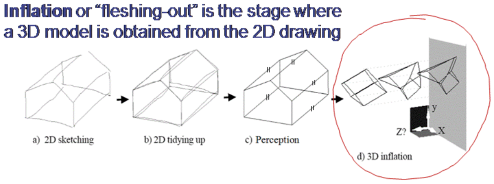
√ Dijkstra's Algorithm is noticeably faster for all drawings because of its low time constant, but the difference is greater for smaller drawings - our new algorithm actually has a *better* practical time complexity



When applied to our most complex drawing (251 edges), the new algorithm took slightly more than one second – it is fast enough for use in interactive systems

Slides 23 to 27 summarise the current state of the art of inflation. They bring together ideas found in a number of different papers, especially [Per68], [LS96], [CCC04] and [MVS05].

Inflation or "fleshing-out" is the stage where a 3D model is obtained from the 2D drawing



Two different strategies coexist:

1 Direct inflation — Without intermediate solutions

2 Iterative inflation — When tentative solutions are tested en route to the final solution

Two kinds of direct inflation approaches can be considered:

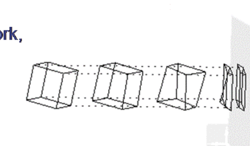
1 When enough information is available

i.e. Line drawings of **semi-normalons** allow direct inflation

2 When geometrical information is incomplete and perceptual information is ambiguous

i.e. **Linear programming**

When direct inflation does not work, **iterative approaches** are used

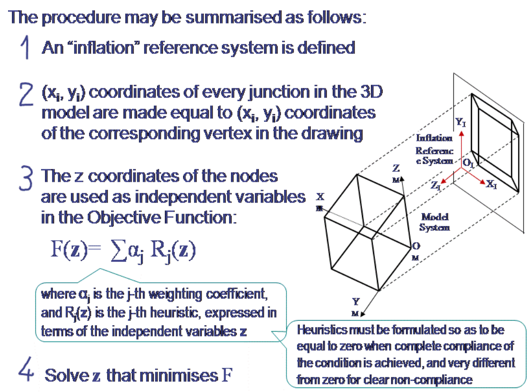


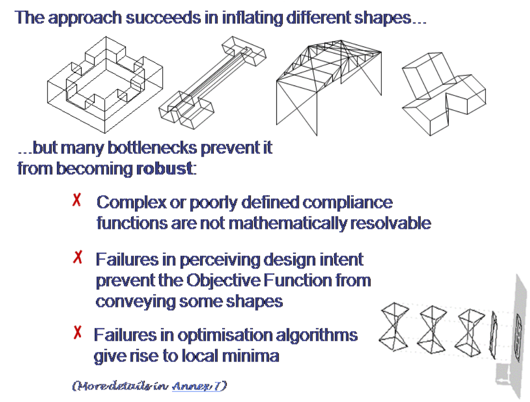
The most frequent strategy is:

1 The multiple heuristics are formulated as **compliance functions**

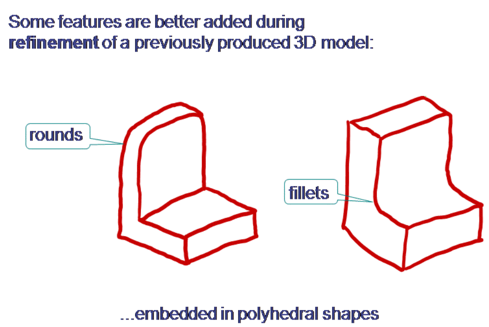
2 The compliance functions are combined to produce a single **objective function**

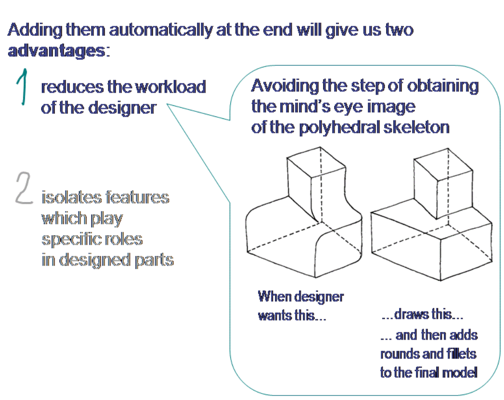
3 The solution which minimises/maximises the objective function is sought by way of **mathematical optimisation** strategies

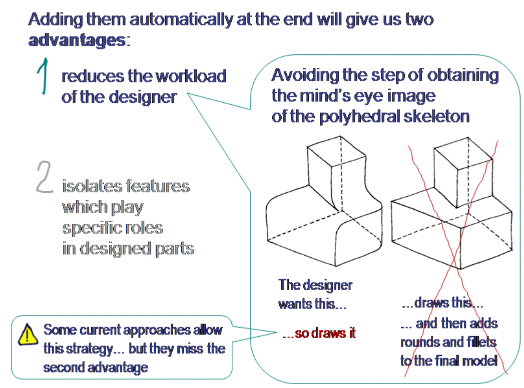


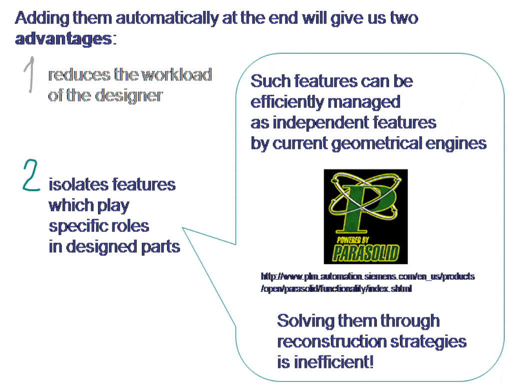


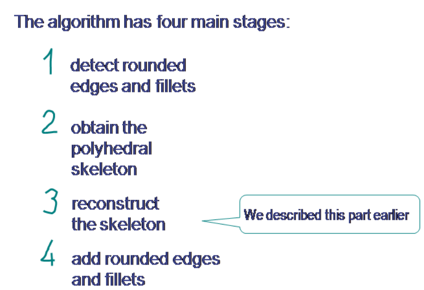
Slides 28 to 41 discuss methods for identifying and processing rounds and fillets. The theoretical basis of this work has already been presented [CV10], and a longer paper is planned which will discuss implementation and results.

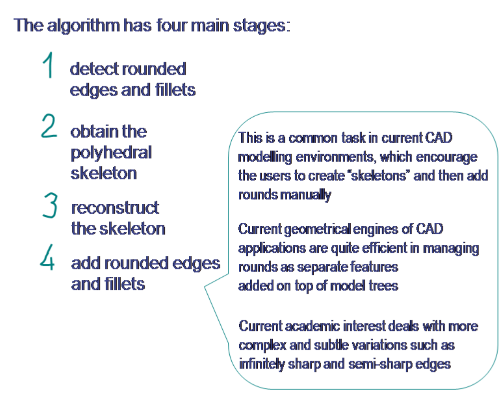


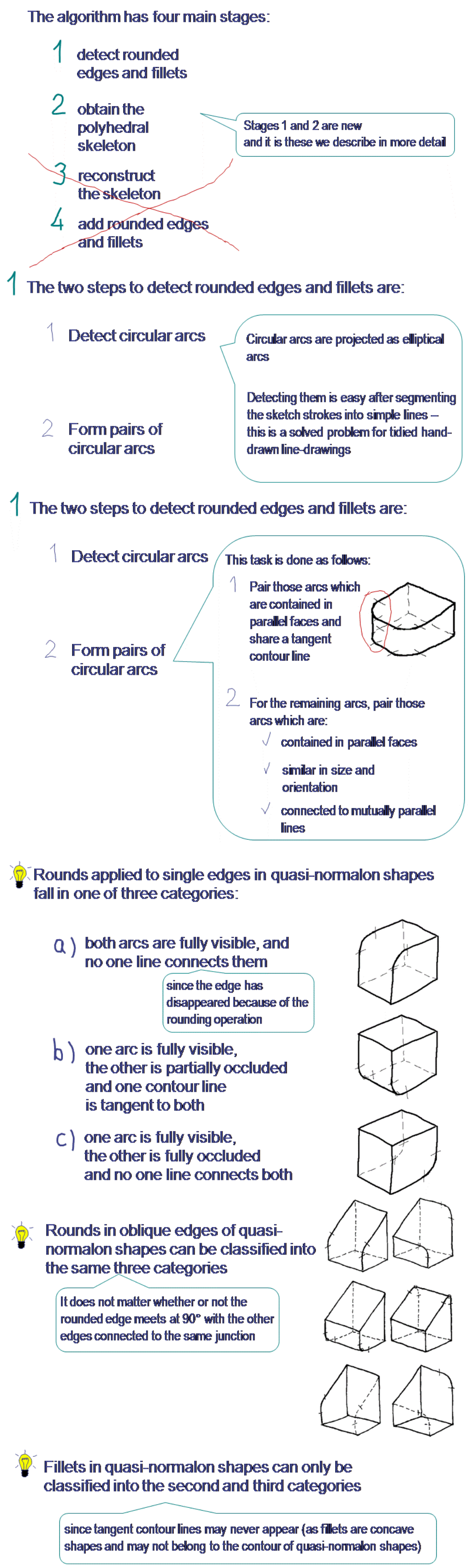


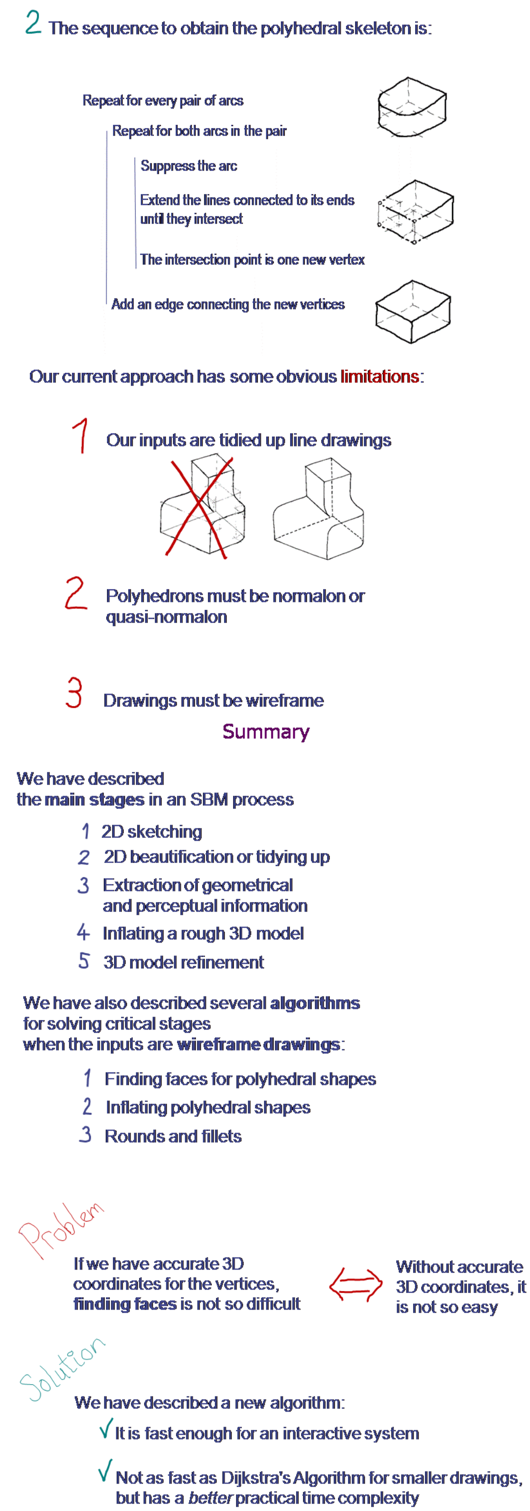












3. Natural Line Drawings

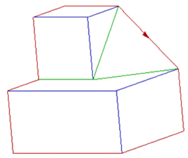
Slides 3 to 15 discuss line labelling. Much of the information to be presented is also to be found in [VMS05].

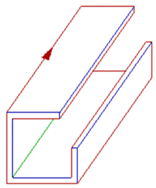
Line labelling labels each line in a drawing as:

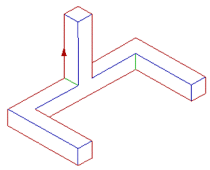
convex

concave

occluding

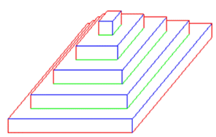






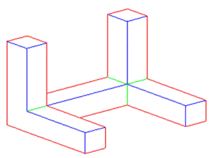
The original purpose of line labelling was as a method of identifying and rejecting impossible drawings

But line labelling also has many other uses ...



1 Line labels indicate which edges bound the visible faces or partial faces of the object and which merely occlude them

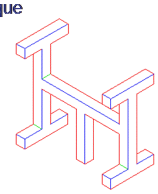
2 The underlying vertex types implied by the junction labels limit the possible hidden topologies



3 The junction labels constrain the geometry of any edges to be extended or added

4 Labelling is also a useful input to **inflation**

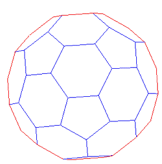
Clowes-Huffman line labelling (catalogue labelling) is a well-established technique



√ It is very effective for drawings of objects containing only *trihedral vertices*

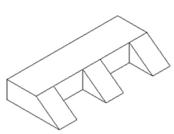
√ There are only 18 possible ways of labelling trihedral junctions

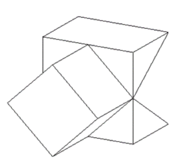
√ Often, there is only one consistent labelling for the whole object



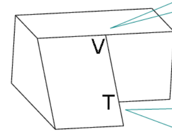
Clowes-Huffman line labelling is less effective (when it works at all) for drawings of objects containing higher-order vertices:

- There are over 100 possible ways of labelling 4-hedral junctions
  - Drawings of tetrahedral objects usually have many possible labellings
  - Catalogue labelling is slow and unreliable
- There are thousands of possible ways of labelling higher-order junctions (5-6-7-8-hedral)
  - Even determining the catalogues is not practical
- Clowes-Huffman labelling can also lead to labellings which have no geometric interpretation





The most important function of line labelling is to distinguish occluding from non-occluding T-junctions

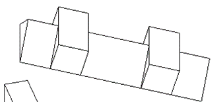


There is a real vertex at *V*. The vertex is at least 4-hedral, so one more edge must be added

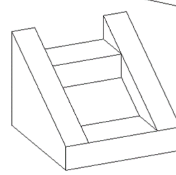
There is no vertex at *T* – it is just the point at which one edge becomes occluded by another. There is a vertex somewhere further along the line, but we do not know anything else about it

These differences will become important when we try to construct the complete object

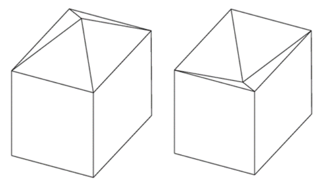
Some specific problems:



1 A junction label which normally indicates an occluding *T*-junction here represents an extended-*K*-junction



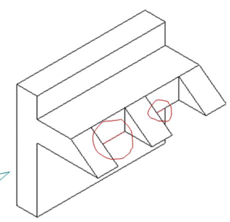
2 Traditional algorithms methods do not use **geometry** at all, so cannot distinguish these two



3 Geometry affects Labelling:

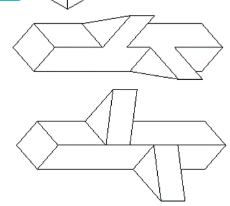
A line which separates two regions corresponding to parallel faces must occlude one or the other - it cannot be convex or concave

there are two such lines in this drawing



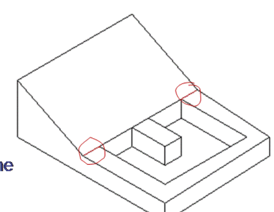
4 Symmetry constrains labelling:

The central line corresponds to an edge with an axis of symmetry through its mid-point, so for reasons of symmetry as well as geometry it cannot be occluding



5 Non-Local Constraints:

When two or more edges lie between the same two faces



If the edges are collinear, the labels must be the same

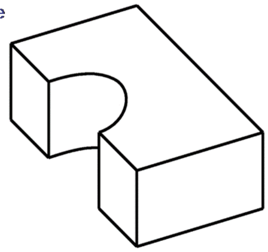
If they are non-collinear, the labels must be different, and at least one must be occluding

Here, the two edges are collinear (and both concave)

6 Curved Objects?

In principle, drawings of curved objects can also be labelled, but there are problems

The label of one of the lines in the drawing changes from one end to the other!



State of the Art:

- ✓ Traditional line labelling algorithms solve local discrete constraint satisfaction problems
- ✓ 1-node constraints: each junction must have a valid label
- ✓ 2-node constraints: each line must have the same label at both ends
- ✗ Traditional algorithms cannot handle non-local constraints
- ✗ **Traditional algorithms ignore geometry**
- ✓ For trihedral drawings, there is often only one solution, so ignoring geometry does no harm
- ✗ When there are many solutions, ignoring geometry causes problems

Why not determine line labels geometrically?

If we can inflate the drawing to 2½D first, all we have to do is measure the resulting geometry to determine which lines are convex, concave and occluding – we do not need catalogues or constraint satisfaction algorithms

However, line labelling is a useful *input* to inflation

→ How reliable would inflation be without line labels?

Answer: even without line labels, inflation is usually reliable for drawings which meet all of the following criteria:

- Most corners are cubic corners
- The drawing is not in cabinet projection or similar
- The centre of the drawing is nearer than the edge to the viewer

Line labelling helps inflation, inflation helps line labelling

→ This suggests an alternating process, which inflates, determines line labels, re-inflates, re-labels, etc, until it converges

This represents the current state of the art, but although it is reasonably reliable it is still not perfect

It is also comparatively slow

Using a combination of the geometric insights provided by line labelling and those provided by the compliance functions discussed next seems the best way to determine frontal geometry

But there is still research to be done to determine the best combinations

Summary

- ✗ Line labelling is not going to go away: it is a very well-understood Valued Discrete Constraint Satisfaction Problem, and will continue to be investigated as a test of VDCSP algorithms
- ✗ At present, there is no reliable catalogue labelling algorithm for 4-hedral objects, and even the catalogues themselves for 5-hedral objects and beyond are too large for determining them to make sense
- ✓ Even if it is not possible to label a drawing completely, partial labelling remains useful
- ✓ Even more importantly, the geometric insights from line labelling remain true even if the algorithms used to implement it are limited

Slides 16 to 28 discuss inflation to 2½D. This is similar to, but not identical to, the inflation problem for wireframes, as additional compliance functions are available (such as that described in [VMS04]).

Inflation of natural line drawings to 2½D is easier than inflation of wireframes:

- ✓ We still use compliance functions
- ✓ Sometimes we use the same compliance functions, but they give us more information
- ✓ If we can label the drawing, this gives us other compliance functions to choose from

Objectives

1. The depth ordering of adjacent pairs of visible vertices must be correct
2. Depth ordering must not be sensitive to inaccuracies in the drawing
3. Depth information must be calculated in a fraction of a second for drawings of typical engineering components
4. Depth information should be based on as little prior processing of the drawing as possible

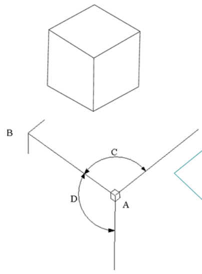
   Depth information will be used to test hypotheses, so it should not presuppose these hypotheses if this can be avoided

5. Depth information should be as good an interpretation of the drawing as is possible while achieving the other objectives
6. The results of inflation do not have to be perfect

   We can add a beautification stage after completing the object topology

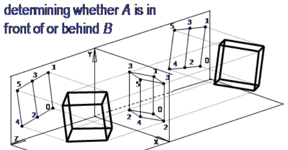
   This will give us another chance to improve the geometry later

Cubic Corners



$$Z_B - Z_A = \pm AB\sqrt{(\tan C \tan D - 1)}$$

Note that there must be a separate mechanism for determining whether *A* is in front of or behind *B*



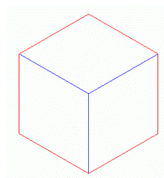
- If we have labelled the drawing, this often tells us whether *A* is in front of or behind *B* (e.g. the all-convex Y-junction, the central vertex is in front of the others)
- Even without labelling, we often have clues (e.g. boundary vertices are often behind internal vertices)

Junction Label Pairs:

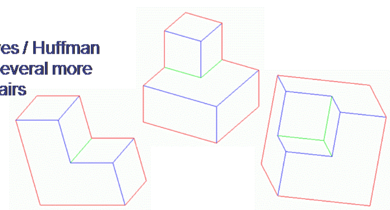
Consider **pairs** of connected junctions in the drawing

We can deduce, just from the line labels, which is the nearer

(and roughly by how much)



The other Clowes / Huffman solids give us several more junction label pairs



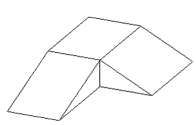
The extended trihedral solids give still more pairs

However, adding in the 4-hedral junctions (91 of them!) is impractical

No 2-label combination involving a 4-hedral junction is common enough to justify hard-coding it in an algorithm

Adding in the 5-hedral junctions and beyond is not even worth thinking about

Perpendicularity: Introduction

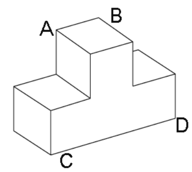


Assumptions to do with perpendicularity are very important:

✓ perpendicularity is the most common regularity in engineering objects

✓ perpendicularity is an important part of the human perception process

Line Parallelism



$n z_A - n z_B = m z_C - m z_D$

Where m is the 2D length of line *AB* and n is the 2D length of line *CD*

✓ Easily arranged into linear or explicit equations

✓ Not inherently inflationary: the trivial solution $z=0$ satisfies the equations

Face Planarity

✓ Can be arranged into linear equations if we include face normals as well as vertex z-coordinates as the unknowns

✓ Quadrilateral faces can always be arranged into linear or explicit equations

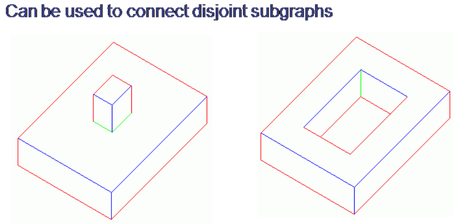
✗ Larger (pentagonal and beyond) faces cannot be arranged into linear equations if the only unknowns are the vertex z-coordinates

N.B. making groups of four vertices coplanar does not necessarily ensure that the entire face is planar

Face Planarity (continued)

✗ Not inherently inflationary: the trivial solution z=0 satisfies the equations

✓ Can be used to connect disjoint subgraphs



Once we have chosen our compliance functions, how do we apply them?

✓ Linear system approaches are quickest and best

We shall describe one such linear system approach

(More alternatives in Annex 1)

✗ Iterative approaches have also been tried

✗ they are slow

✗ there are no compensating advantages

The simplest and most effective approach is to use a linear system where the only unknowns are vertex *z*-coordinates

The question is then, what to include and what to leave out?

- √ Junction label pairs are good, but require a correct line labelling (Junction label pairs on their own do not work for non-graph-connected drawings)
- √ Cubic corners are nearly as good as JLP in most cases, but they do not distinguish +*z* from –*z* (N.B. for cabinet projections cubic corners is dreadful)
- √ Line parallelism is essential for good results (for large drawings generating parallelism equations for each pair of parallel lines is too much; it is better to generate one equation for each line, making it parallel to the ideal line of the bundle)
- ✗ Face planarity is not generally recommended (it does not help much, and makes badly-drawn drawings worse; it is the best way to join unconnected graph segments, e.g. hole loops)

Inflation: Summary

- √ Inflation using linear system of *z*-coordinates and a careful choice of compliance functions achieves its objectives
- √ This is the least problematic area of sketch interpretation (Particularly if we have reliable line labels to work with)

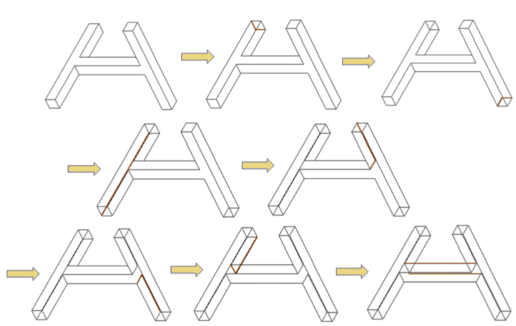
Slides 29 to 45 discuss the creation of hidden topology, a requirement which is unique to the interpretation of natural line drawings. There are two promising methods. The first, based on completing the object wireframe, is described in Chapter 10 of [Var03] and has not been published separately.

What does the back of the object look like?

Two promising approaches, both iterative

- Recreate a complete wireframe (We only need vertices and edges since we already know how to find faces)
- Model the object as unions and intersections of extrusions

Recreate a complete wireframe, one or two edges at a time



This is a Greedy Approach

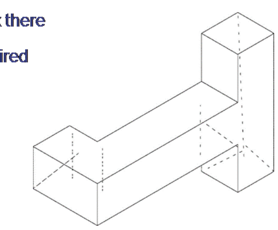
Sometimes it works, sometimes it does not

Main problems:

- ✗ Expanding it to a depth-first tree search does not help much
- ✗ When it goes wrong, the result is usually the wrong object, not an invalid object
- ✗ There is no trigger to invoke backtracking

Basic idea:

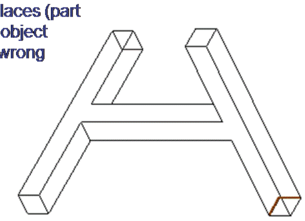
1. Extend lines in all major directions from all incomplete vertices
2. Note where the lines cross
3. Pick the best (using heuristics and probability theory)
4. Place a new vertex there
5. Add edges as required



Refinement:

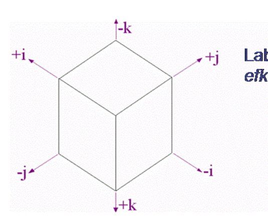
Any hypothesis which places a vertex outside the object silhouette is (probably) wrong

Any hypothesis which places (part of) an edge outside the object silhouette is (probably) wrong



Refinement: Neighbourhood matching

We can divide the space around each vertex into eight subspaces, using the three orthogonal places as half-space dividers



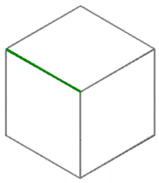
Label the eight subspaces *efg*, *efk*, *ejg*, *ejk*, *ifg*, *ifk*, *ijg* and *ijk*

efg is nearest the viewer

ijk is furthest from the viewer

Using the labelling, we can make deductions that some subspaces must be full and some subspaces must be empty

Given: the green line is *i-aligned* and *convex*

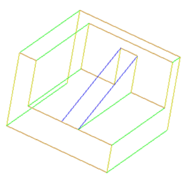


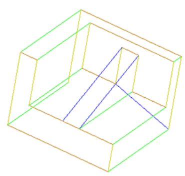
We can say that:
. One of the subspaces at the near vertex must be *full*
. Three of the subspaces at the near vertex must be *empty*
. One of the subspaces at the far vertex must be *full*
. Three of the subspaces at the far vertex must be *empty*

Note: at every visible vertex, the zone *efg* cannot be *full*
. this is the subspace which includes the line-of-sight

Subspaces belonging to two vertices (behind one and in front of the other) cannot be both full and empty

Any added edge which would create a subspace mismatch must be wrong



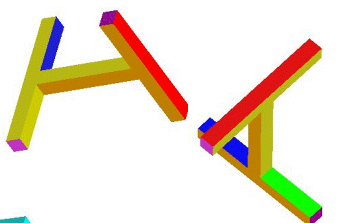


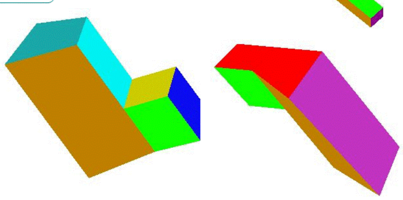
Results:

Mixed

Very dependent on the first few

if these are right, the rest is usually right too

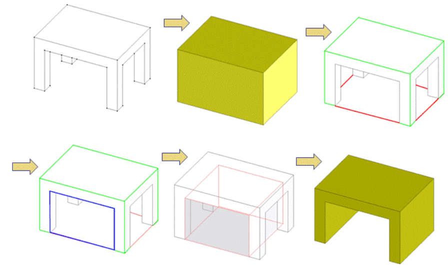




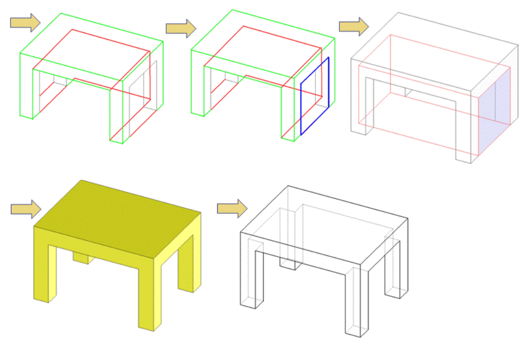
Unions and intersections of extrusions

Most of this comes from: Y.S. Suh (2007) ***Reconstructing 3D Feature-Based CAD Models By Recognizing Extrusions From A Single-View Drawing***, Proc. IDETC/CIE 2007)

Identify a *profile face* and extrude it



Continue identifying further *profile faces* until the entire object is modelled



A profile face is one which, when extruded along a major axis, explains some of the unidentified lines in the drawing

Some candidate profile faces are better than others and should be processed first:

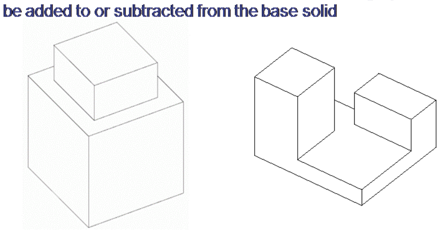
1. 2D area of the profile face: The larger the better
2. Number of profile face edges: The more the better (exception: triangles are better than quadrilaterals)
3. Number of extrusion lines (all in the same direction) leading away from the profile face: The more the better
4. 2D length of the extrusion lines: The longer the better
5. Number of points on profile face whose 3D positions are known: The more the better

Subgraphs

Sometimes, if we break a sketch at T-junctions, we get two or more disjoint subgraphs

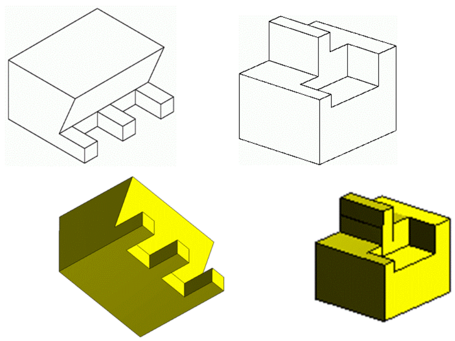
Each subgraph leads to a solid object

The subgraph which has the largest bounding box is treated as the base solid, and other solids from other subgraphs will be added to or subtracted from the base solid

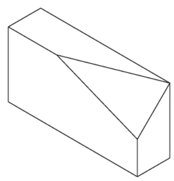


The second (and at the time of writing the more promising), based on reconstructing the polyhedron as the union and intersection of extrusions, is as described in [Suh07], with some minor improvements of our own. The figures in slides 38, 39 and 42 are taken from [Suh07].

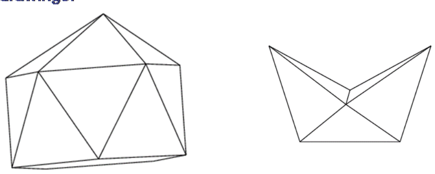
√ Results are generally good:



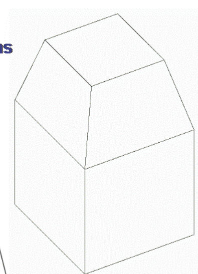
✗ But the method is limited to those objects which can be modelled as unions or intersections of axis-aligned extrusions



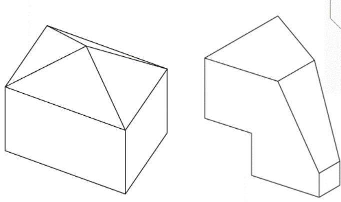
It cannot process these drawings:



There are also a few objects which can be modelled as **unions or intersections** of axis-aligned extrusions but for which the algorithm does not work:



One or more necessary profile faces is not part of the object



If an object can be modelled as unions and intersections of extrusions, this method is usually more reliable

The alternative is creating a wireframe by projecting edges and locating their intersections

√ Is more flexible

It can, in principle, model any polyhedron

✗ But the greater flexibility gives it more opportunity to go wrong

4. Open Problems

Slides 3 to 16 discuss how engineers actually use pencil and paper in practice. Do current sketching tools offer all the modalities which users of pencil-and-paper expect? What is missing? This is a summary of [CV09].

⚠ User studies assert that current SBM tools

✗ Are still less usable than paper and pencil sketches

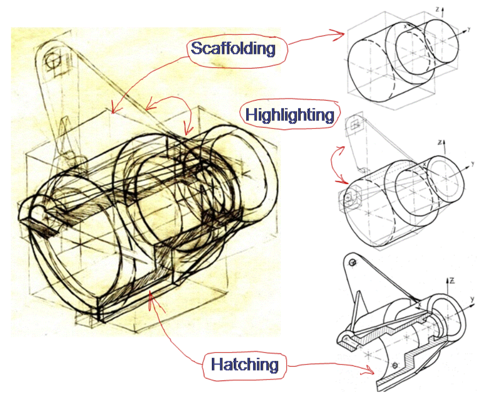
✗ Do not possess significantly improved functionality

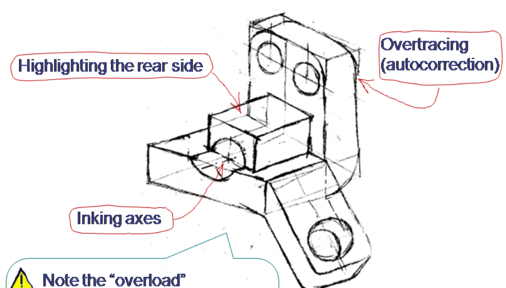
The "hardware" of paper and pencil sketching is **simple** ...

... but its operation is sophisticated ...

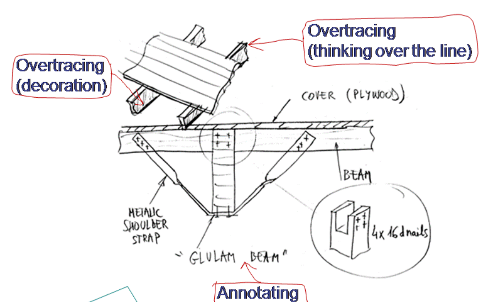
... as paper and pencil sketching is **multimodal**!

Let us see some examples!



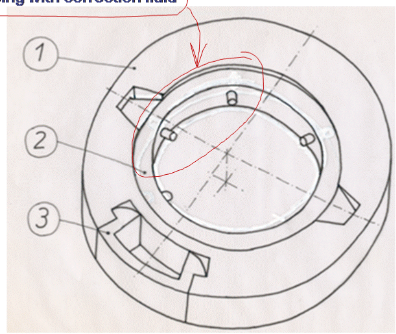


⚠ Note the "overload" of thin lines for:
- ✓ scaffolding
- ✓ highlighting the rear side
- ✓ inking axes

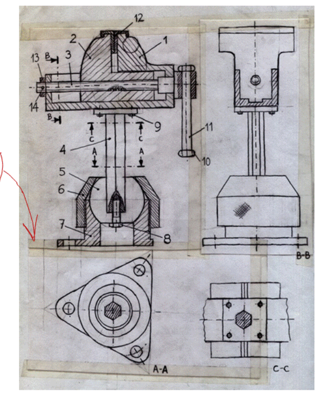


⚠ Also note the informal mixing of views:
- ✓ orthographic view
- ✓ detailed view
- ✓ pictorial view!

Erasing with correction fluid

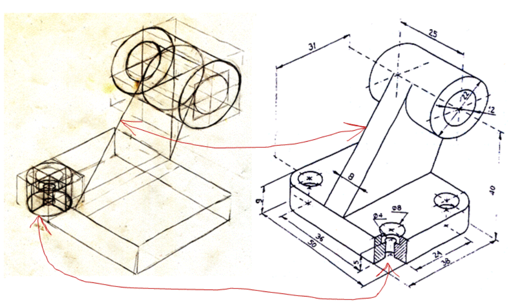


Hard cut and paste



Symmetry
to complete the drawing

The copies were made though transparent paper superimposed and displaced over the original drawing



We conclude that:

√ The set of **operational modes** is complex:

- Inking
  - Scaffolding
  - Highlighting
  - Highlighting the rear side
  - Inking axes
  - Hatching
  - Overtracing
    - Autocorrection
    - Thinking
    - Decoration
  - Annotating
- Editing
  - Erasing
  - Copy and paste
  - Copy with symmetry

√ The **switching strategy** is non-intrusive !

We may reduce the gap between actual paper and pencil and SBM tools:

√ Replicating as many modes as possible

√ Adding extra features

Wrong paradigms guide human-to-computer interaction

command-driven and menu-driven

No interface paradigm which would suit experienced design engineers

Identifying such a paradigm is a difficult task in itself

We may reduce the gap between actual paper and pencil and SBM tools:

√ Replicating as many modes as possible

√ Adding extra features

There is a broad agreement on some of the main advantages of computer-based systems:

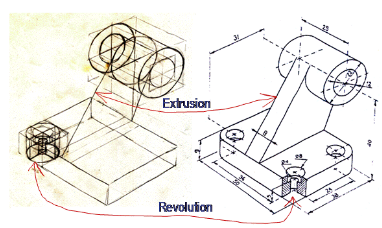
√ Work is easier to edit

√ Work is easier to file

√ Work is easier to interface to other applications

Besides, ...

... adding some current CAD operations could also help in reducing editing tasks



Final goal:

SBM tools should be as easy to use as actual paper and pencil!

Sculptris by "Dr Petter" is a sucessful example http://www.drpetter.se/project_sculpt.html

To this end:

1 Hardware advances are required

For example, tablets have been reported to be less comfortable to use than pencil and paper because of the small gap (both in time and distance) between the cursor and the pencil tip

2 Software improvements are also required

Use and maintenance of computers still requires technical knowledge which designers, quite rightly, do not see as part of their job

3 We still require a full taxonomy of operating modes

Including their mutual relationships and descriptions of cues used to discriminate between them

The list we have described is illustrative, but far from exhaustive

4 Two questions must be posed

- How many functions can be provided without buttons and menus?
- How many functions does a design engineer require?

If the second answer is larger than the first, we need a new paradigm

1 Full modeless applications are not the final goal

If we wish to replicate real paper-and-pencil scenarios in virtual environments, we must be aware that actual paper-and-pencil scenarios include a rich variety of different modes

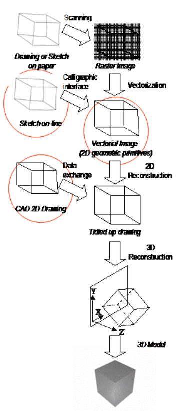
2 Replicating paper-and-pencil scenarios in virtual environments is still unfeasible

Although the goals of SBM have supposedly already been accomplished, practical implementations have unfriendly interfaces

Three types of input have been considered so far for SBM:

1 perfect line drawings
2 line drawings containing some "minor" mistakes
3 freehand sketches

But this simple classification may be refined!



Slides 18 to 33 discuss the problem of interpreting annotated sketches. Which strokes are annotation rather than object? What does the annotation mean? How should it be applied? This section collates information from several published papers ([CAN08], [CCV09]).

We can consider three purposes for sketching:

- *thinking*
- *talking*
- *prescribing*

Combined with two levels of geometrical information:

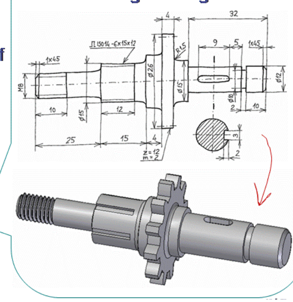
- *Line drawings*
- *Sketches*

And two levels of non-geometrical information

- *With annotations*
- *Without annotations*

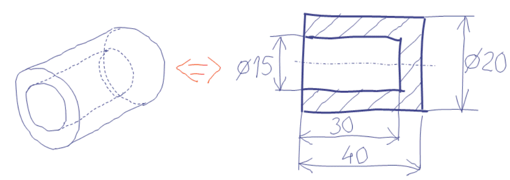
(More details in Annex 8)

An open problem derived from this classification is **interpreting annotated engineering sketches**



The generic term "annotations" includes:

- Many standardised conventions (e.g. dimensions)
- Cut views with hatchings
- A large etcetera of icons and symbols



The proposed approach for the open problem of **producing 3D models from annotated engineering drawings** is:

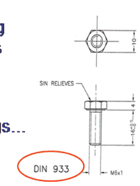
- Capture and record the data
- Separate annotation data from drawing data
- Interpret drawing
- Interpret annotations
- Apply the annotations to the drawing

To learn more:

LaViola, J. (2007) Advances in Mathematical Sketching: Moving Toward the Paradigm's Full Potential, IEEE Computer Graphics and Applications, 27(1):38-48

The goal of interpreting engineering symbols is not trivial, since...

...behind apparently quite simple drawings...

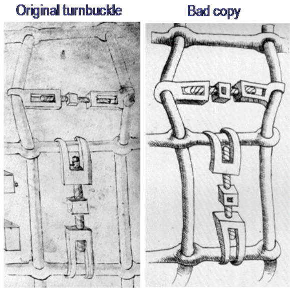


...there are hundreds of standards...

...defining the exact meaning of many symbols and conventions

It is obvious that communication of relevant information depends on the **meaning of symbols**:

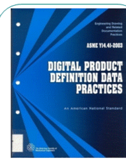
Original turnbuckle | Bad copy



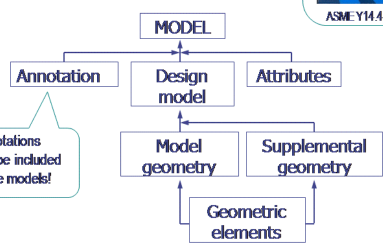
Ferguson E.S. ***Engineering and the Mind's Eye***, MIT Press (1992)

Misunderstanding of symbols causes technical information loss!

The problem becomes still more challenging if we realise that **new standards** already allow annotations in 3D models:

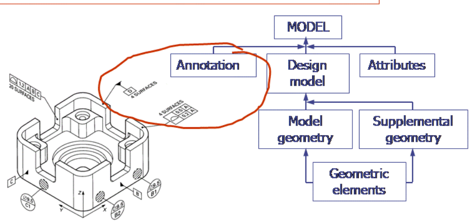


ASME Y14.41-2003



Annotations can be included in the models!

Today, computers are blind to these annotations !



The annotations are just "labels" added to the model

✓ Which the user can read and modify,

✗ but the geometrical engine does not use them, neither to construct, nor to edit or validate the model.

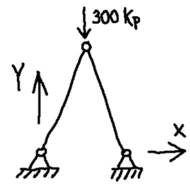
One interesting related open problem is **interpreting sketched data input for Computer-Aided Engineering applications**

It is an open problem since data is input through two alternative **WIMP user interfaces**: (window/icon/menu/pointing device)

Stand-alone **CAE pre-processors** which define both geometric data and attributes ⇔ Combination of **CAD applications plus downstream CAE pre-processors**

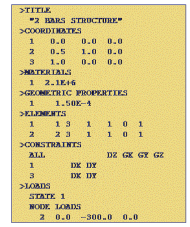
CAD exports the geometry and CAE adds attributes

Input



The input are those sketches which designers typically draw aside to fix their ideas before interacting with CAE pre-processors

Output



```
>TITLE
  "2 BARS STRUCTURE"
>COORDINATES
  1   0.0   0.0  0.0
  2   0.5   1.0  0.0
  3   1.0   0.0  0.0
>MATERIALS
  1  2.1E+6
>GEOMETRIC PROPERTIES
  1    1.50E-4
>ELEMENTS
  1    1 3   1   1  0  1
  2    2 3   1   1  0  1
>CONSTRAINTS
  ALL              DZ GX GY GZ
  1       DX DY
  3       DX DY
>LOADS
  STATE 1
  NODE LOADS
    2  0.0  -300.0  0.0
```

The output is a file which meets the specifications of the desired analysis code

Two reasonable **assumptions** are:

1. The input sketches are drawn directly onto a computer screen acting as "virtual paper and pencil" — Not side-drawn on an actual paper sheet!

2. The user is still in the process of conceptual design and is not yet ready to progress to a detailed design stage

Hence, the **goals** are:

1. supply the user with a computer interface similar to classical paper-and-pencil

2. minimise the amount of information provided by the user ...

... and give the user more freedom in inputting and editing it

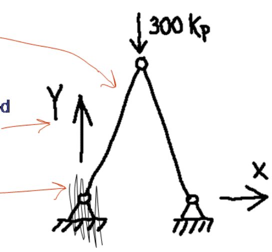
Our application, Pre/Adef, distinguishes:

Company P., Aleixos N., Naya F., Varley P.A.C., Contero M. and Fernandez-Pacheco D.G. (2008) **A New Sketch-Based Computer Aided Engineering Pre-Processor.** Proc. Sixth Int. Conf. on Engineering Computational Technology. Civil comp Ltd. Paper-149.

1. Geometric entities

2. Symbols associated with annotations

3. Gestures associated with editing tasks (i.e. "sketched commands")



Finally, having interpreted each group in isolation, we must **combine** them into a whole

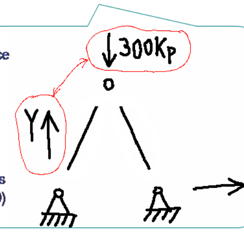
√ Connect the bar elements to the appropriate nodes

√ Apply the loads to the right nodes or elements

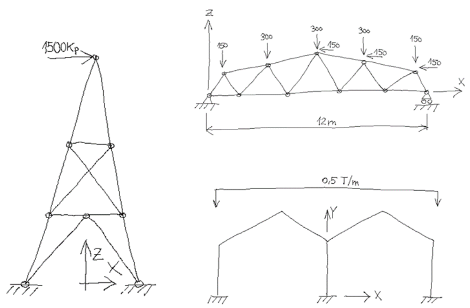
The arrow of the 300 kp force is parallel (more or less) to the arrow of the "Y" axis

They have opposite senses

The vector force should thus be interpreted as (0, -300, 0)



We tested our approach by sketching a set of examples



ADVANTAGES	UNSOLVED PROBLEMS
Valid output files are obtained √	✗ Modes required (but only to change the mode when moving to a very different task)
Training not required √	
The user is not urged by the system to define exact dimensions √	✗ Users do not always feel comfortable with an on-line parser!

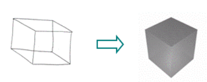
Our interface is similar to but not yet as good-for-thinking as actual paper-and-pencil

But produces output files !

(More details in Annex 9)

Slides 34 to 40 consider the possibility of sketching assemblies of parts. This is unpublished material which was briefly presented at [Com07b].

Currently, we are limited to reconstruct **isolated parts**



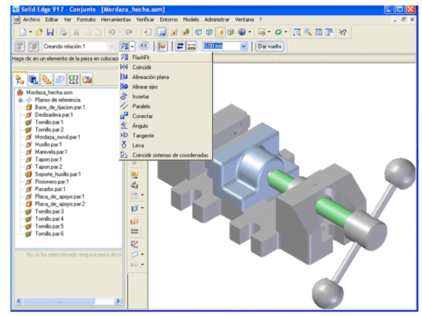
But we want to be able to **create assemblies from sketches !**

Our vision is
to define and implement a set of symbols
that can help a SBM system
to assemble 3D models obtained from 2D sketches

The basic guidelines of our approach should be:

- √ The symbols must be sketched themselves, as part of a "natural" design process
- √ The meaning of the symbols must be "robust"
  - In the sense of being understood without mistakes by the geometrical engine in charge of assembling the parts
- √ The symbols should overtake the faults of current sets of CAD operations

What is wrong with current CAD applications?



SolidEdge: originally developed and release by Intergraph in 1996
using the ACIS geometric modeling kernel
it later changed to using the Parasolid kernel

Components can be positioned within the product assembly using:

- √ absolute coordinate placement methods
- √ **mating conditions**

Mating conditions are definitions of the relative position of components with respect to one another

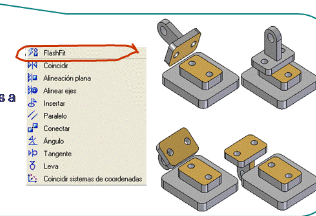
For example alignment of axis of two holes or distance of two faces from one another

The final positions of all components based on these relationships is calculated using a geometry constraint engine built into the CAD or visualisation package

Some **tools for mating conditions** assist the user to get an intuitive and friendly set of constraints:

1. As users place parts in an assembly, assembly relationships position new parts relative to parts already in the assembly.
2. There are several relationship types for positioning parts relative to each other.

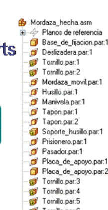
Starting with v8 (2000), Solid Edge also has a FlashFit option which can reduce steps required to position parts.



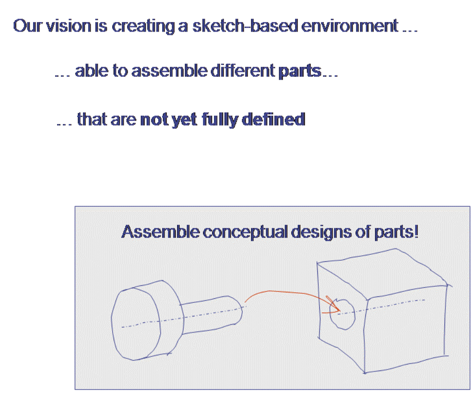
However, we find one main drawback:

Only **complete and consistent parts** can be assembled

CAD assembly sub-systems require standard CAD parts input



Detailed design of parts is an assembly **pre-requisite!**



References

[Com04] Company P., *Computer-Aided Ideation through sketch-based modeling*. Conference in seminar “The second Las Palmas seminar on computational geometry and graphics in engineering”. Universidad de Las Palmas de Gran Canaria, November 4-5th 2004

[CCC04] Company P., Contero M., Conesa J. and Piquer A. *An optimisation-based reconstruction engine for 3D modelling by sketching*. Computers & Graphics. 28 (6) 955-979, 2004.

[CPC04] P. Company, A. Piquer and M. Contero, *On the evolution of geometrical reconstruction as a core technology to sketch-based modeling*. Eurographics Symposium Proceedings. Sketch-Based Interfaces and Modeling (SBM04). (ISBN 3-905673-16-9). 97–106. 2004.

[CPC05] Company P., Piquer A., Contero M. and Naya F. (2005) A Survey on Geometrical Reconstruction as a Core Technology to Sketch-Based Modeling. Computers & Graphics. Vol. 29, No 6. pp. 892-904

[Com07] Company P., *Computer-aided Ideation through Sketch-based interfaces and modelling*. Conference in the “9th Summer School on Engineering Design Research” Design Society. Universitat Jaume I, Castellón, May 18th 2007.

[Com07b] Company P., *Tools for easing the Human-Computer Interaction during Virtual Assembly Process, by way of Sketch-Based Interfaces*. VI Edition of Italian-Spanish seminar. Napoli, June 4th, 2007.

[CAN08] Company P., Aleixos N., Naya F., Varley P.A.C., Contero M. and Fernandez-Pacheco D.G. *A New Sketch-Based Computer Aided Engineering Pre-Processor*. Proceedings of the Sixth International Conference on Engineering Computational Technology. Civil comp Ltd. Paper-149. 2008

[CCV09] Company P., Contero M, Varley P.A.C. Aleixos N. and Naya F. *Computer-Aided Sketching as a Tool to Promote Innovation in the New Product Development Process*. Computers in Industry. 60 (8), pp. 592–603. 2009

[CV09] P. Company and P.A.C. Varley, *Operating Modes in Actual versus Virtual Paper-and-pencil Design Scenarios*, Intelligent User Interfaces (IUI) Workshop on Sketch Recognition, Sanibel Island, Florida, 8th February, 2009..

[CV10] P. Company and P.A.C. Varley, *A Method for Reconstructing Sketched Polyhedral Shapes with Rounds and Fillets*, 10th International Symposium on Smart Graphics, Banff, Canada, 24th-26th June 2010.

[DT95] Dori D.; Tombre K. (1995) *From engineering drawings to 3D CAD models: are we ready now?* Computer-Aided Design 27, pp. 243-254

[Fer92] Ferguson E.S. *Engineering and the Mind's Eye*, MIT Press (1992)

[HPE09] T. Hammond, B. Paulson, B. Eoff, *Tutorial on Sketch Recognition*, in Ed. K. Museth and D. Weiskopf, Proc. EUROGRAPHICS 2009.

[Hof00] Hoffmann D. *Visual Intelligence. How we create what we see*. New York: WW Norton & Company, 2000

[KS89] Kimura F. And Suzuki H. (1989) *A CAD System for Efficient Product Design Based on Design Intent*. CIRP Annals - Manufacturing Technology , 38 (1), 149-152.

[Lav07] LaViola, J. (2007) *Advances in Mathematical Sketching: Moving Toward the Paradigm's Full Potential*, IEEE Computer Graphics and Applications, 27(1):38-48

[LLM10] Li M, Langbein F.C. and Martin R.R.(2010) *Detecting design intent in* *approximate CAD models using symmetry*. Computer-Aided Design 42 (3) 183-201

[LS96] H. Lipson and M. Shpitalni, *Optimization-based Reconstruction of a 3D Object from a Single Freehand Line Drawing*, Computer-Aided Design **28** (8), 651-663, 1996.

[LT05] J. Liu and X. Tang, 2005. *Evolutionary Search for Faces from Line Drawings*, IEEE Transactions on Pattern Analysis and Machine Intelligence 27(6), 861–872.

[MW80] G. Markowsky and M.A. Wesley, 1980. *Fleshing Out Wire Frames*, IBM Journal of Research and Development, 24(5) 582–597.]

[MVS05] Ralph Martin, Peter Varley and Hiromasa Suzuki, *Perpendicularity as a Key to Interpreting Line Drawings of Engineering Objects*, Proc. Digital Engineering Workshop: 5th Japan-Korea CAD/CAM Workshop, 115-120, 2005.

[Pal99] Palmer SE. Vision Science. *Photons to Phenomenology*. Cambridge, MA: The MIT Press, 1999

[Per68] D. N. Perkins, *Cubic Corners*, Quarterly Progress Report 89, 207-214, MIT Research Laboratory of Electronics, 1968.

[RDI10] Rivers, A., Durand, F., Igarashi, T. (2010) *3D modeling with silhouettes*. ACM Transactions on Graphics 29 (4), art. no. 109

[RW10] Roth-Koch S. and Westkaemper E. (2010) *The implementation of a sketch-based virtual product development*. Prod. Eng. Res. Devel. 4:175–183

[SL96] M. Shpitalni and H. Lipson, 1996. *Identification of Faces in a 2D Line Drawing Projection of a Wireframe Object*, IEEE Transactions on Pattern Analysis and Machine Intelligence **18**(10), 1000–1012.

[Suh07] Y.S. Suh, *Reconstructing 3D Feature-ased CAD Models By Recognizing Extrusions From A Single-View Drawing*, Proc. IDETC/CIE 2007

[UWC90] Ullman D., Wood S., Craig D. 1990, *The Importance of Drawing in the Mechanical Design Process*, Computers and Graphics **14**(2):263-274

[Var03] P. A. C. Varley, A*utomatic Creation of Boundary-Representation Models from Single Line Drawings*, PhD Thesis, Cardiff University, 2003.

[VMS04] P. A. C. Varley, R. R. Martin and H. Suzuki, *Making the Most of Using Depth Reasoning to Label Line Drawings of Engineering Objects*, Proc. SM'04, Genoa, 2004.

[VMS05] P. A. C. Varley, R. R. Martin and H. Suzuki, *Frontal Geometry from Sketches of Engineering Objects: Is Line Labelling Necessary?*, Computer Aided Design **37** (12), 1285-1307, 2005.

[VC08] P. Varley and P. Company, *Automated Sketching and Engineering Culture*, in ed. B. Plimmer and T. Hammond, Proc. VL/HCC Workshop on Sketch tools for Diagramming, Herrsching am Ammersee, 15 Sep 2008, 83-92.

[VC10] P.A.C. Varley and P. Company, *A New Algorithm for Finding Faces in Wireframes*, Computer-Aided Design **42**(4), 279-309, April 2010.

[XL10] Xiong, Y. LaViola J. (2010) *A ShortStraw-Based Algorithm for Corner Finding in Sketch-Based Interfaces*, Computers and Graphics, 34(5):513–527.

[XWR09] Xuetao Y., Wonka, P., Razdan, A. (2009) *Generating 3D Building Models from Architectural Drawings: A Survey*. IEEE Computer Graphics and Applications, **29** (1), 20-30

[YTJ08] Yuan S., Tsui L.Y., Jie S. (2008). *Regularity selection for effective 3D objects reconstruction from a single line drawing*. Pattern Recognition Letters 29 (10), 1486-1495