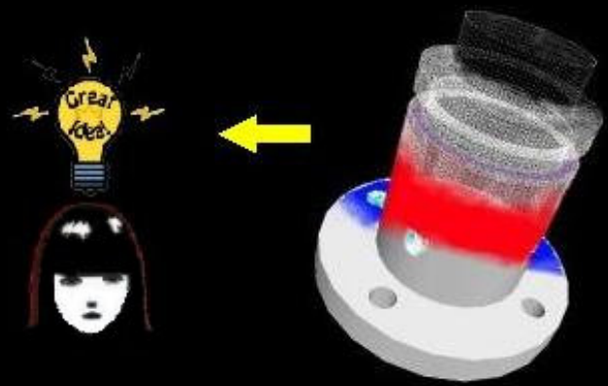
School of Computer Science
Cardiff University

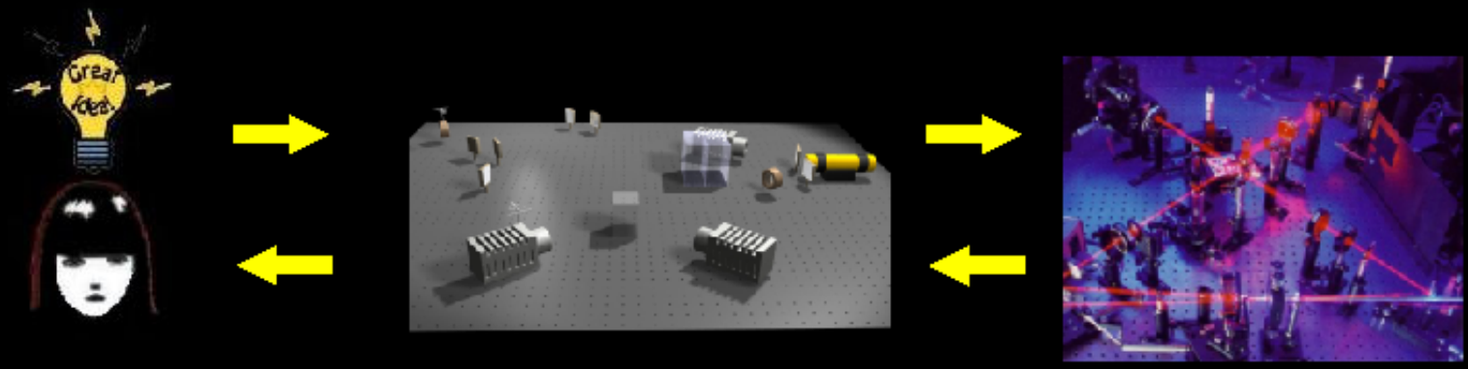
Reverse Engineering
From Artifacts to Concepts

F. C. Langbein



Reverse Engineering

- Engineering converts a *concept into an artifact*
  - From an idea to a physical object



- Reverse engineering converts an *artifact into a concept*
  - Reverse process from evidence about physical object back to idea
- Develop tools to (semi-)automate this process for *geometric models*, software, . . .

Science?

- Science's goal is to *explain the physical world* and *provide models* of the underlying principles
  - Link theories and experiments via *verifiable* hypotheses
  - See philosophy of science...
  - Never only a one-way path from experiment to theory
- Reverse engineering: from experiments to theory
  - But mainly about *artifacts* (things made by humans)
  - Different direction to standard engineering may provide new insights into engineering

Why Reverse Engineer?

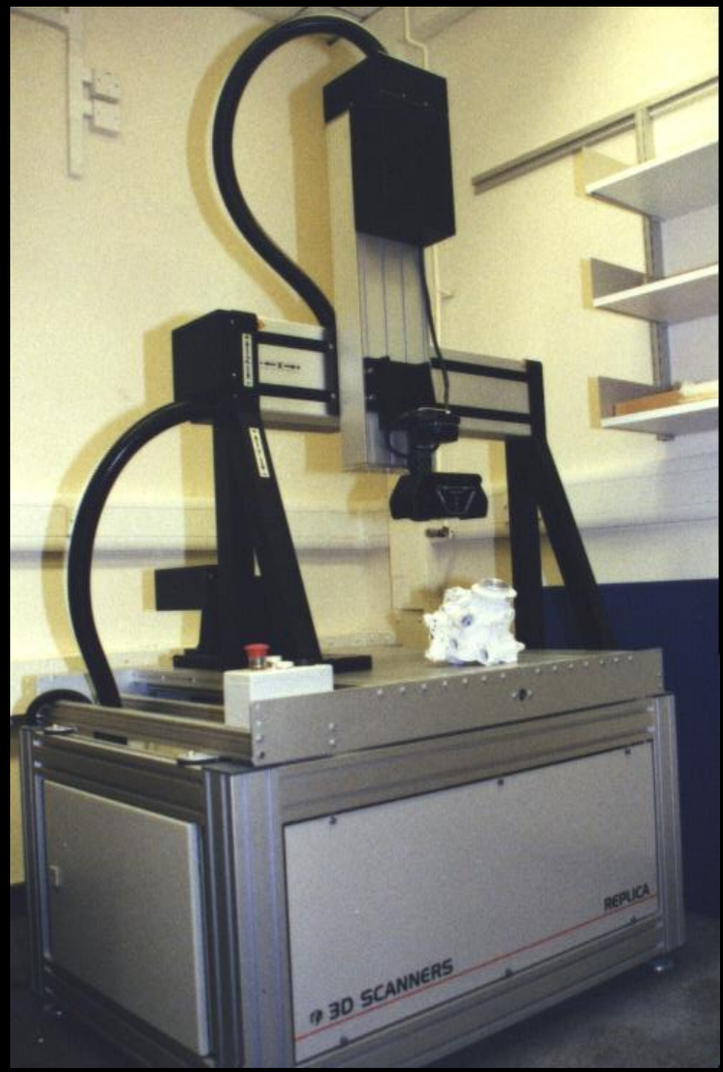
- Because we can: *power and control*
  - Believe that the black box is a miracle or
  - Figure out how it works
- *Learning*
  - How is a particular feature implemented?
  - Understand ideas of others, then apply them yourself
- *Recovering* lost information
  - Often original concepts are not available anymore
- Learn something about humans?
- Is reverse engineering always possible? What can be done?

Reverse Engineering Shapes

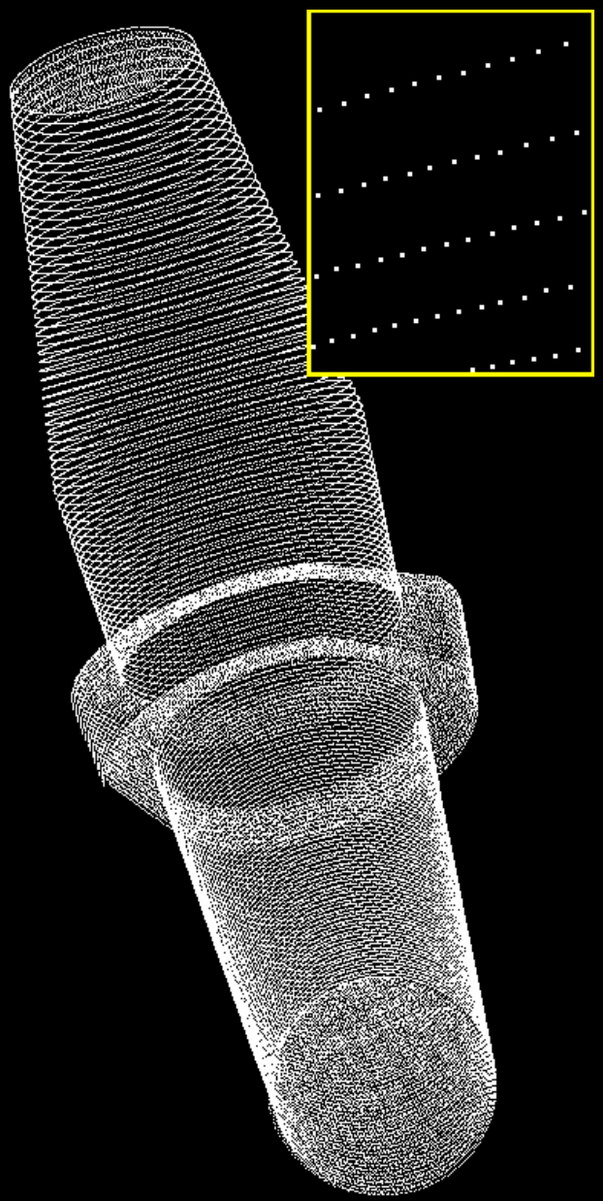
- Extract sufficient shape information from physical object for particular purpose (recover shape)
- For *reproduction* applications:
  - Exact information about shape of physical object is sufficient for one-to-one copy
- For *quality control* applications:
  - Exact shape information has to be compared with an original model
- For *redesign* applications:
  - Reconstructed model should exhibit exactly the same intended geometric concepts as the original model

Reverse Engineering Process

- *Data Capture*

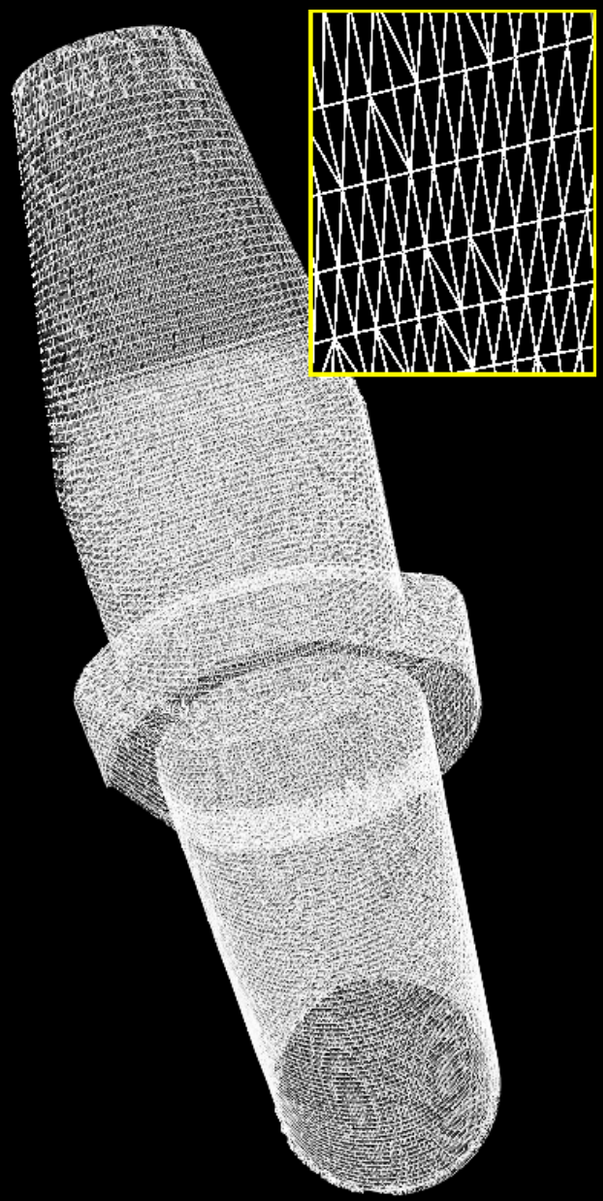


Reverse Engineering Process



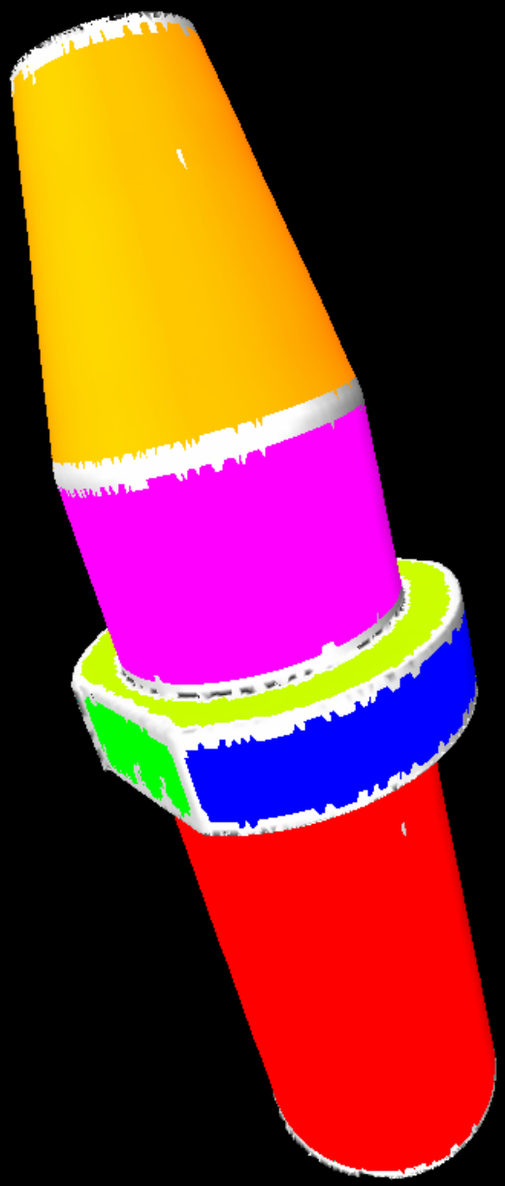
- *Data Capture*
  - Obtain multiple views from a 3D laser scanner
  - Register views to a single 3D point set

Reverse Engineering Process



- *Data Capture*
- *Triangulation*
  - Create a triangular mesh for the point set

Reverse Engineering Process



- *Data Capture*
- *Triangulation*
- *Segmentation & Surface Fitting*
  - Split the point set into subsets representing natural surfaces
  - Find the surface type and fit a surface of this type for each subset

Reverse Engineering Process

- *Data Capture*
- *Triangulation*
- *Segmentation & Surface Fitting*
- *CAD Model Creation*
  - Create an initial CAD model by stitching surfaces
  - Only “surface model”, no design intent

Design Intent

- Design intent is a *detailed representation of the concept*
- Explicit representation of design intent required for high-level CAD applications
  - Description of intended properties of the object's shape (*geometric regularities*)
  - Different *abstraction levels*
    (e.g. "a cube", "6 parallel/orthogonal planes",
    "$n_1{}^t x - d_1 = 0$, $n_2{}^t x - d_2 = 0$, ...")
  - Represent *functional properties*
    (a plug, a hammer, a petrol engine, ...)

Design Intent in CAD Applications

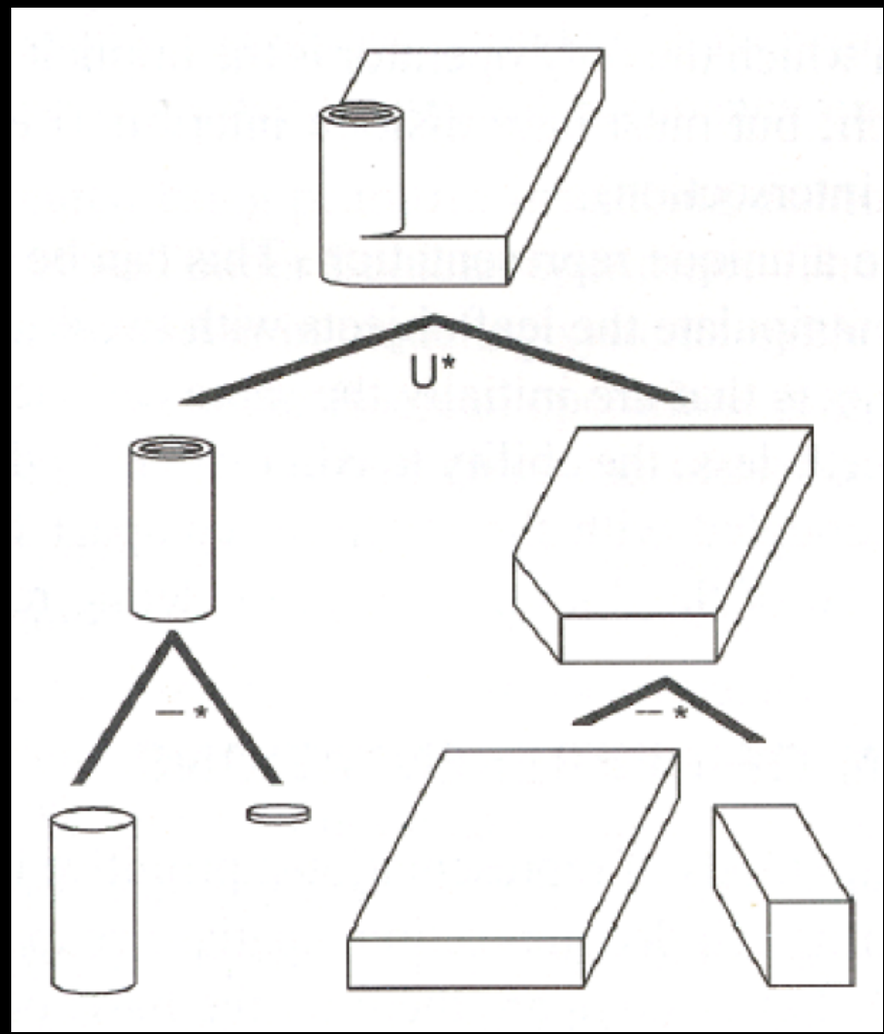
- High-level representation of design intent in CAD applications
  - Allow *modifications* and adjustments without destroying important properties unintentionally
  - Improve *robustness* of modelling operations
  - Enable *data exchange* between different applications without creating broken models or loosing important properties (Healing)
  - *Shape search* to retrieve suitable CAD model from database or find similar models
  - *Analyse* the model's properties

Approaches towards Design Intent

- Standard CAD model data structures do not explicitly represent design intent
  - *Constructive Solid Geometry* (CSG):
    union, intersection, etc. of primitive shapes
  - *Boundary representation*:
    faces, edges and vertices with geometry and topology (boundary relations)
- Extensions of data structures for design intent:
  - *Feature-based modelling*
  - *History-based modelling*
  - *Constraint-based modelling*

Feature-based Modelling

- Describe model by machining, design, . . . *features* (holes, slots, pockets, . . .)
  - Common method for creating models
  - Hard to detect features (many alternative interpretations possible)
  - Features add semantics to CSG-type data structures

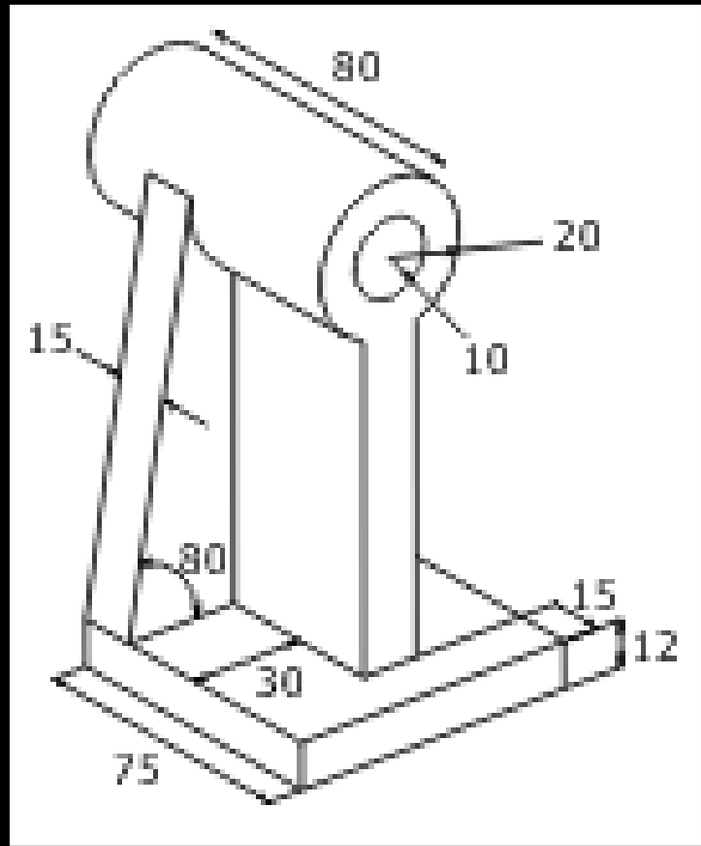


History-based Modelling

- Idea: store the complete *history* of the model building operations
  - Edit object by changing the history and “replaying” it (or relevant part of it)
  - Edit operations are simpler and more robust
- But complete history often contains *irrelevant information*
- *Infinitely many* histories!
- Operations used to make object may contain *hints for design intent*

Constraint-based Modelling

- Specify desired relations between geometric objects by *geometric constraints* (equations)
  - One huge polynomial equation system describes the whole object
- Design intent specified exactly, but
  - Hard to find a solution
  - Under- and over-constrained cases are hard to determine by the user
  - Constraints only describe low-level relations



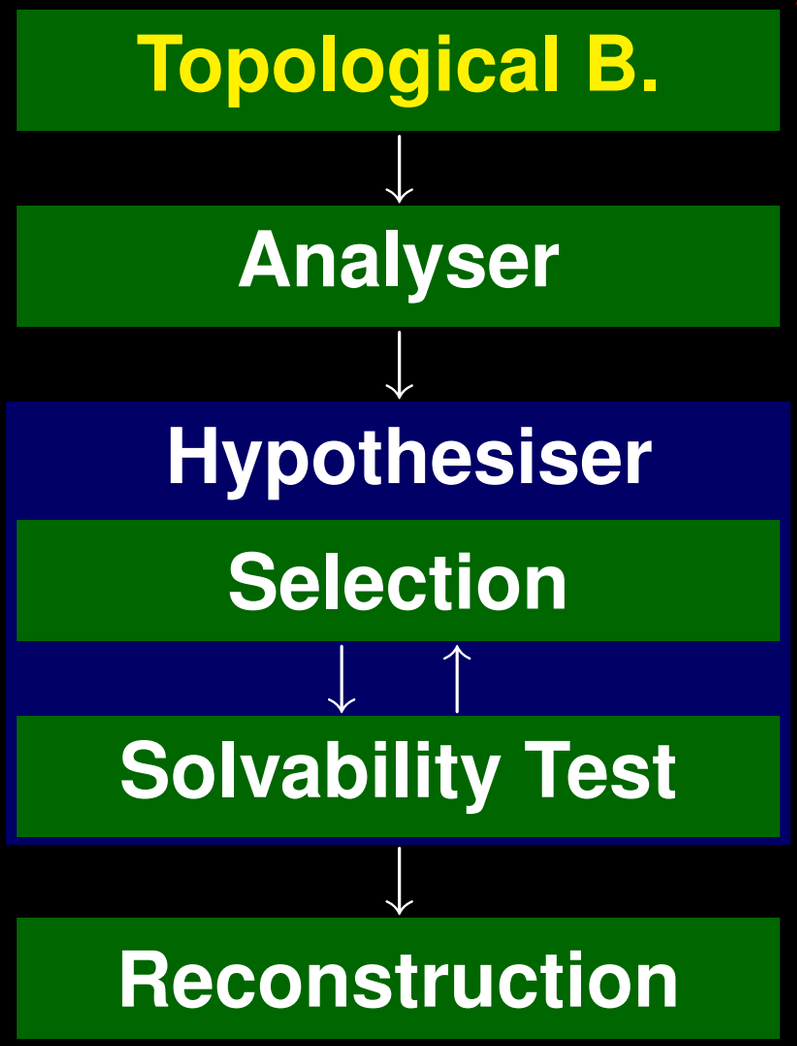
Forward and Reverse Problem

- Need an appropriate representation of high-level design intent
- *Forward* problem:
  - Record design intent during model creation
- *Reverse* problem:
  - Determine the design intent of a given model

Beautification

- *Problem:* Reverse engineered models suffer from inaccuracies caused by
  - sensing errors (data capture)
  - approximation and numerical errors (reconstruction)
  - possible wear of the object
  - manufacturing method used to make the object
- *Goal:* Reconstruct an ideal model of a physical object with intended geometric regularities
  - Design intent has to be considered at some stage
- *Beautification* aims to improve the reconstructed model in a post-processing step

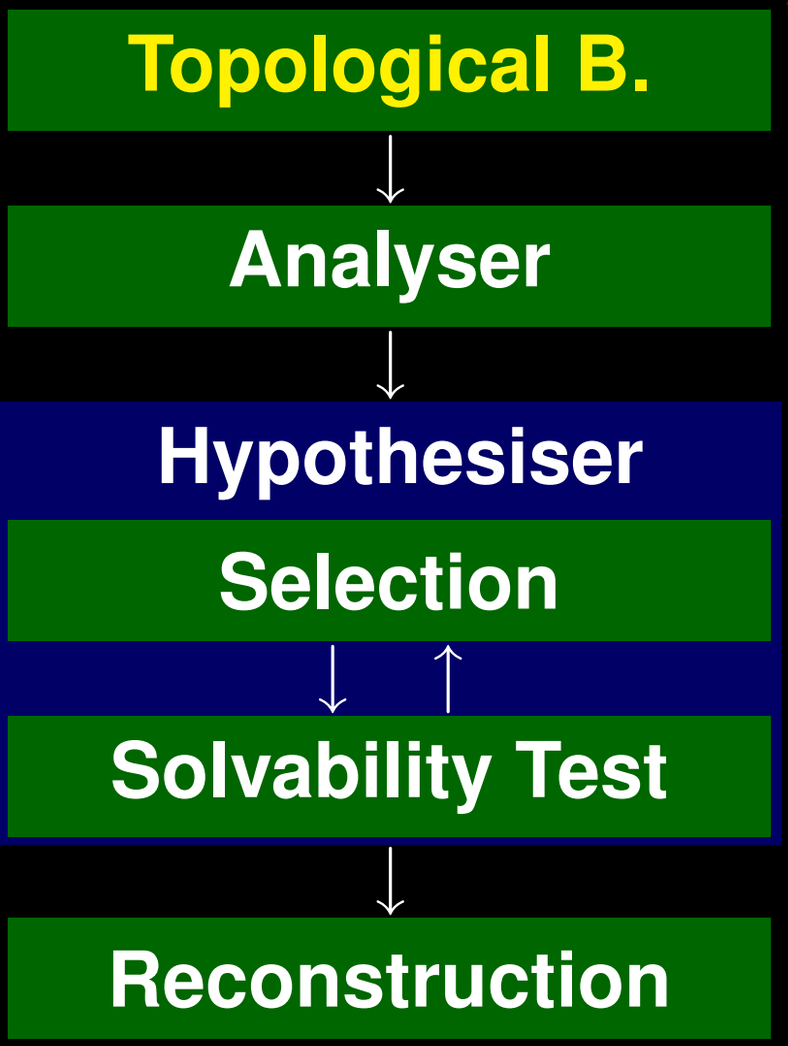
Beautification Process

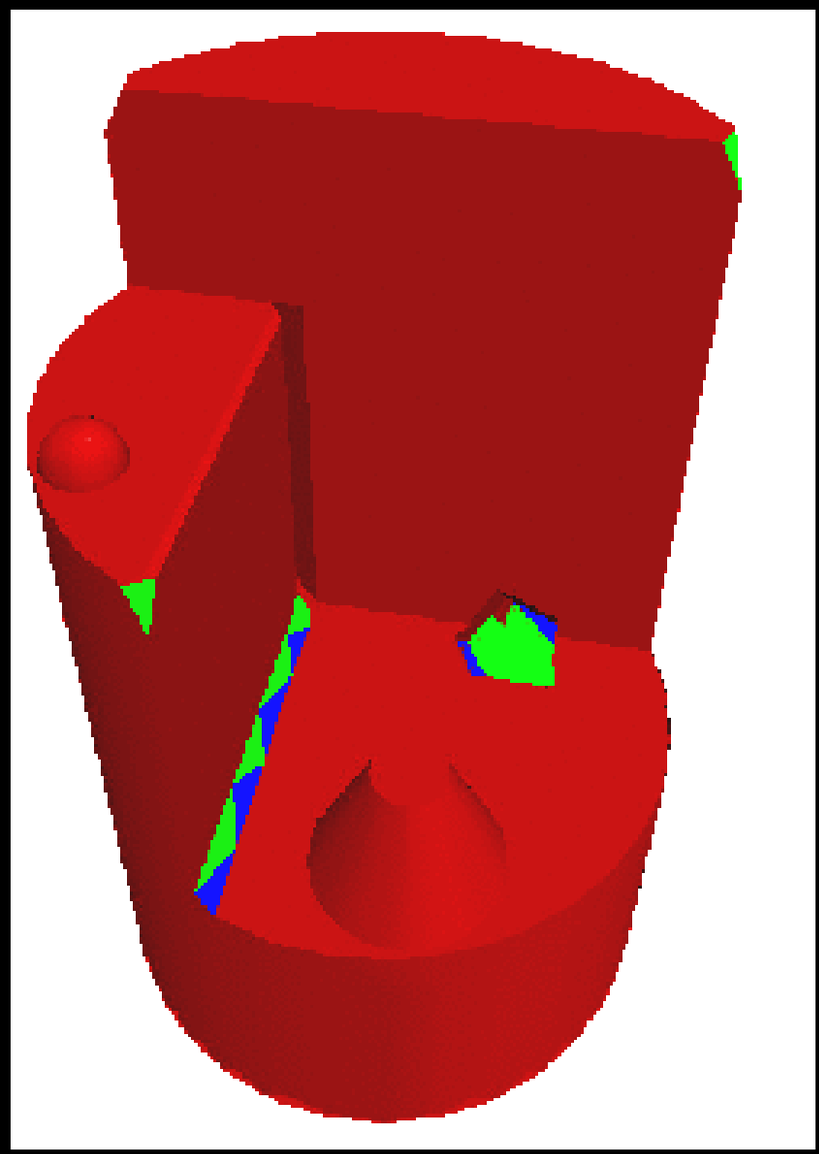


- *Local Topological Beautification*
  - Detect top. defects (gaps, pinched faces, small faces, sliver faces, short edges,. . .)
  - Repair defects by replacing faces with edges, edges with vertices, extending faces, . . .

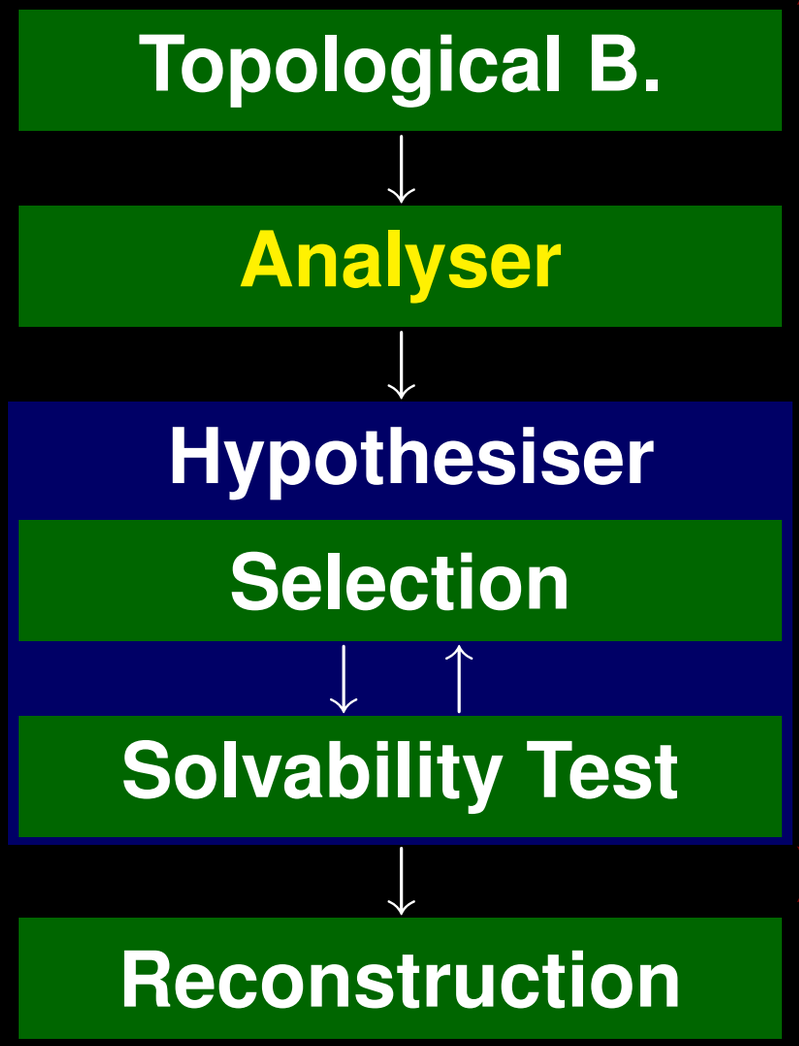
Beautification Process

- *Local Topological Beautification*



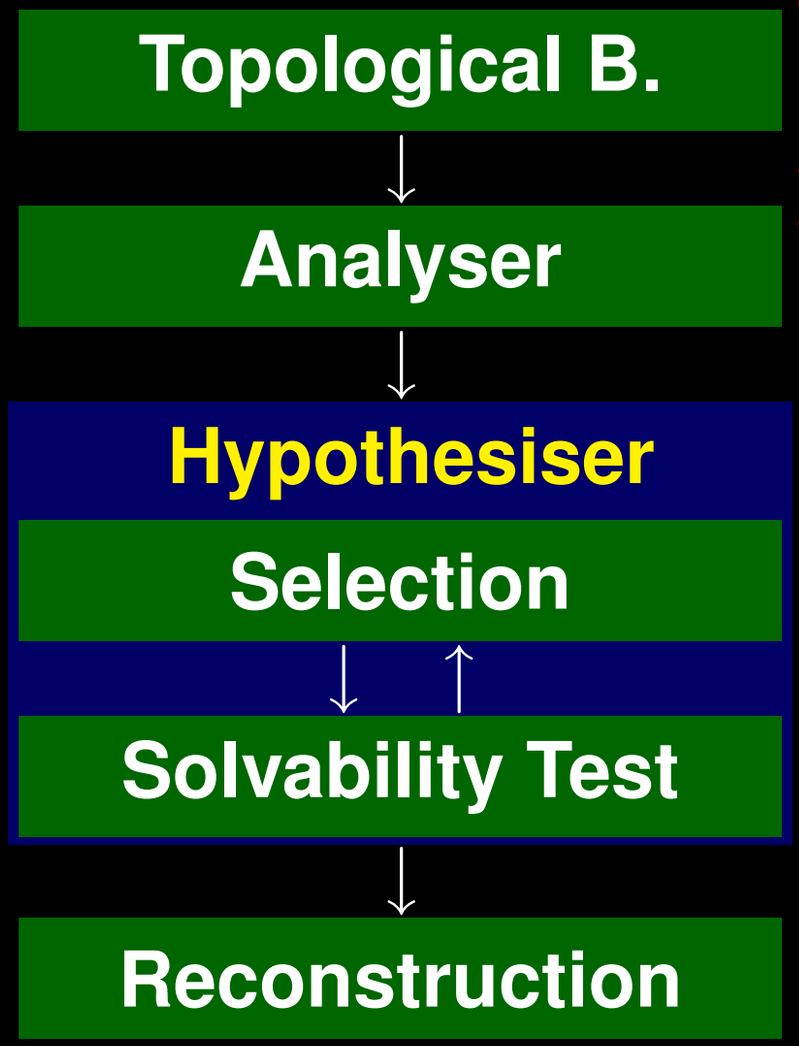


Beautification Process



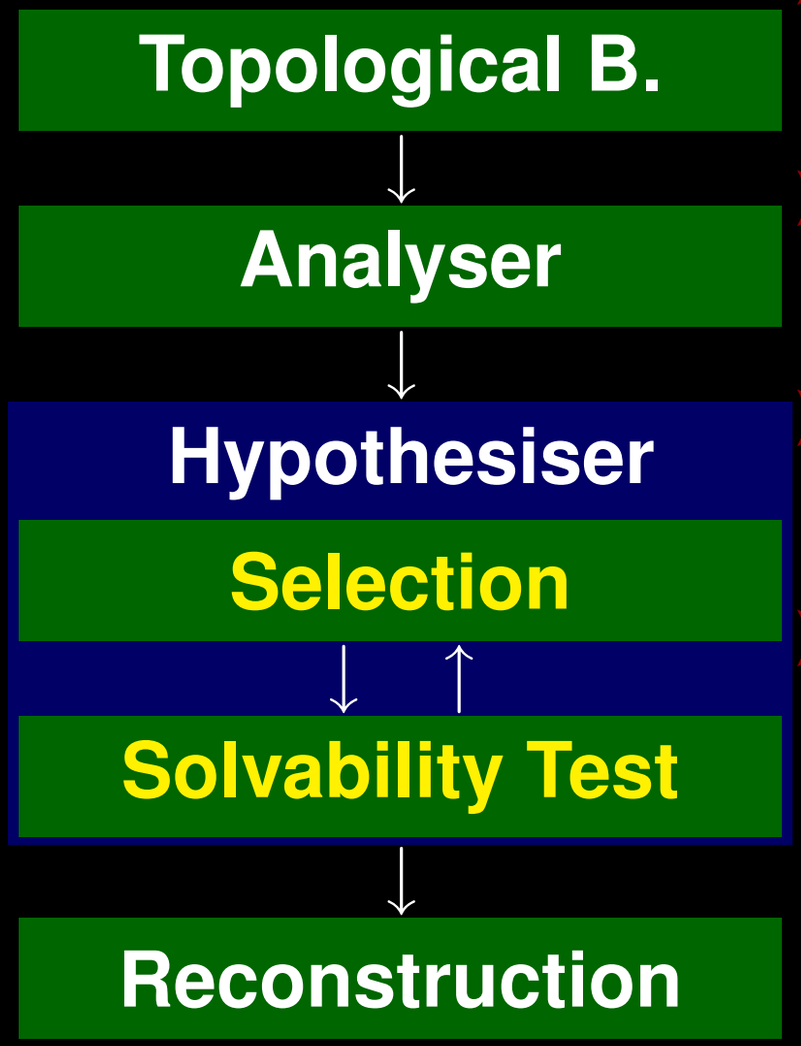
- Detect *approximate geometric regularities*
  - Approximate symmetric arrangement of faces, vertices, directions, etc.
  - Large number of potential regularities
  - Regularities may or may not be intended
- *Exact conditions* for approximate regularities are used rather than arbitrary tolerances

Beautification Process



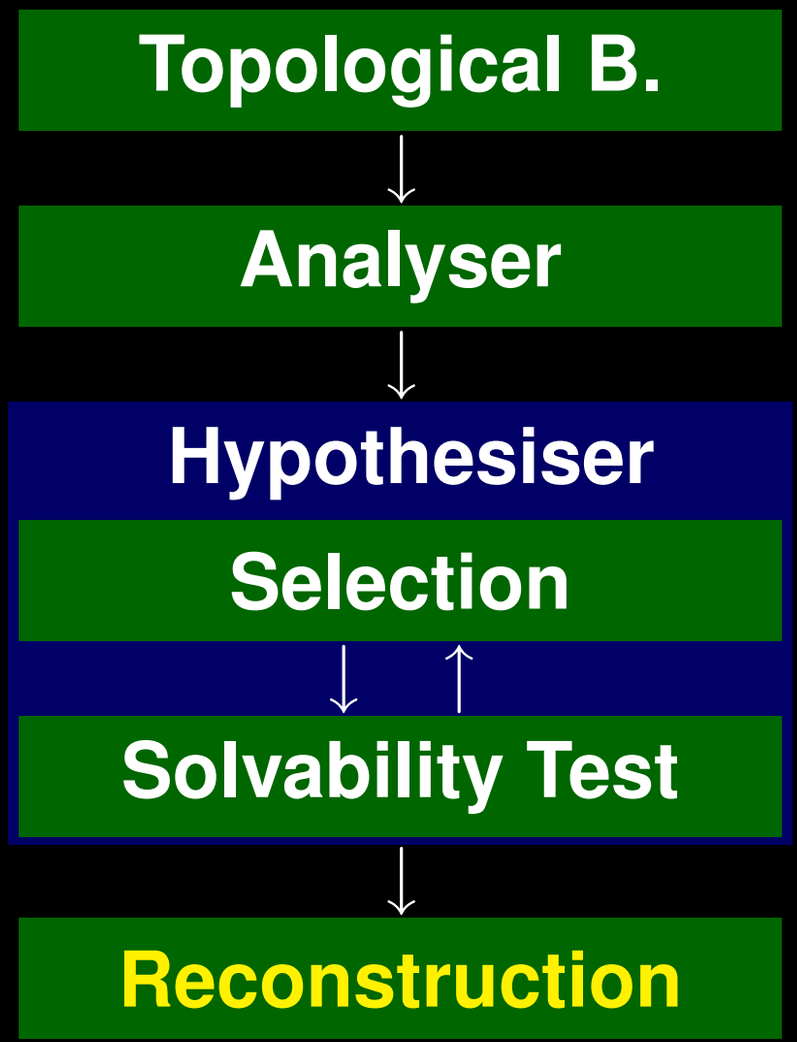
- Detected regularities are unlikely to be mutually consistent
- Have to *select* regularities consistent with respect to
  - *design intent*
  - *simultaneous realisability* (solvability)

Beautification Process



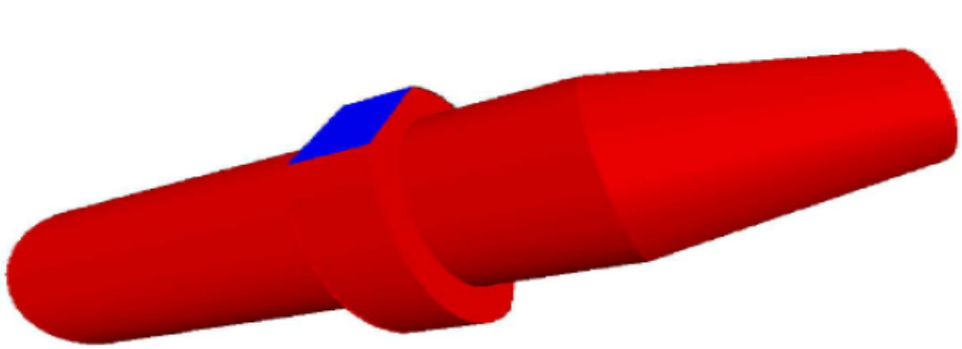
- Use geometric constraints to describe regularities
- Add regularities in order of a priority to a constraint system
- Only accept regularity if constraint system remains solvable
- Priority is based on
  - *how common* the regularity is
  - *"desirability"* of regularity
  - *error* of regularity in original model

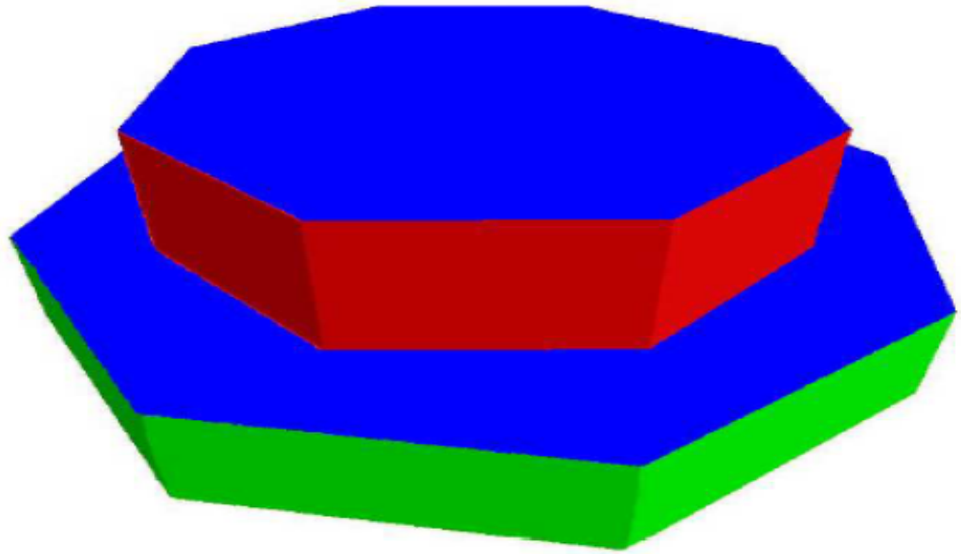
Beautification Process

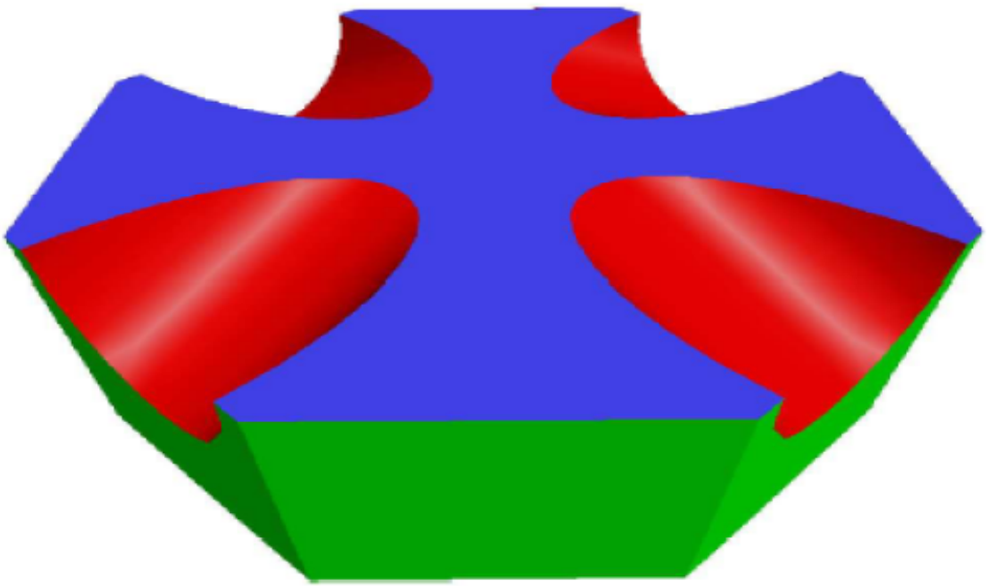


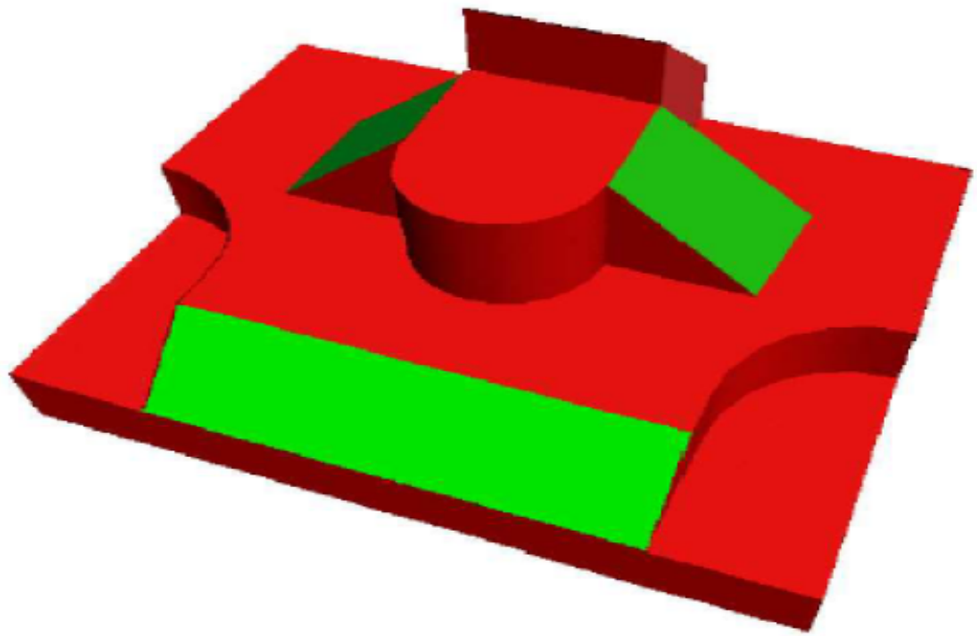
- Compute solution of constraint system
  - Numerical optimiser
  - Decomposition/recombination solver
- *Rebuild model* from solution and topology of original model
- Align model with coordinate axes, fix potential topological defects, ...

Beautification Examples









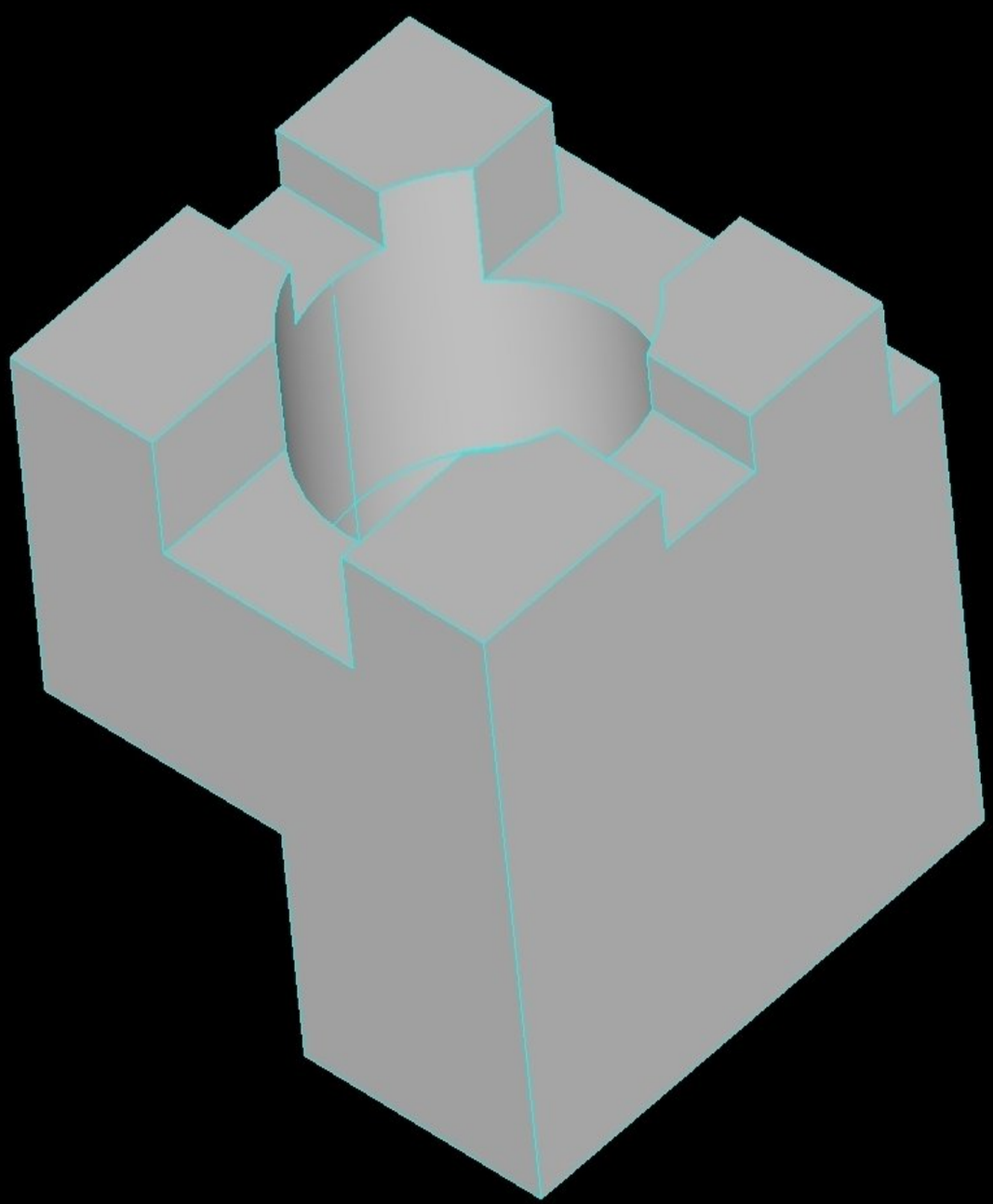
Problems of This Approach

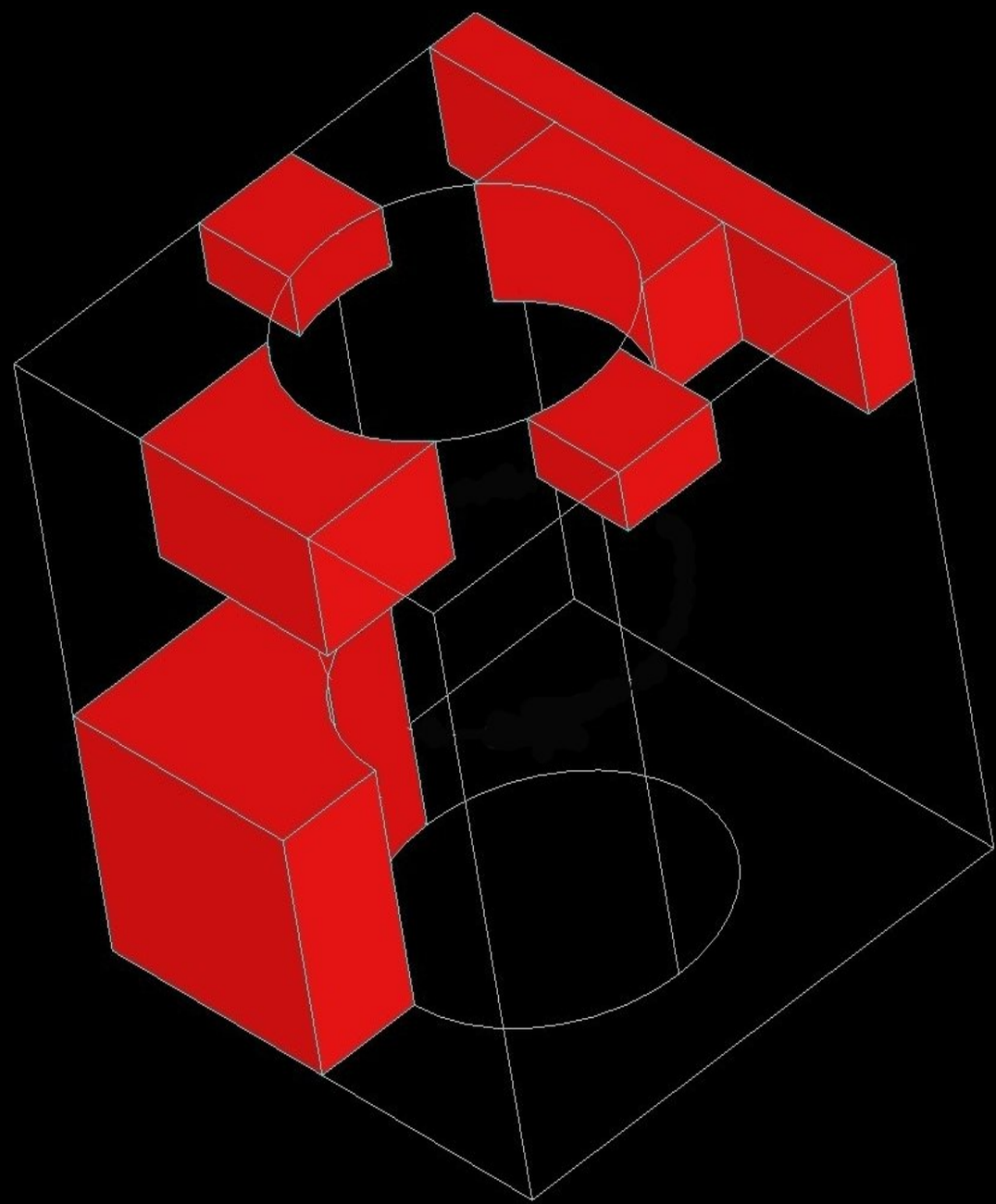
- Can improve simple to medium complexity models
  - Independent, major regularities relating to most of the faces (global symmetries, orthogonal systems)
  - Desirable regularities with high accuracy
- Problems in selecting regularities:
  - Individual regularities rather than combinations
  - Many dependent, ambiguous regularities for complex models
- For complex models selected regularities are consistent w/r to solvability, but not w/r to design intent

Hierarchical Decomposition

- Regularity detection for complex models
  - Often complex models can be partitioned into interesting sub-parts (feature-based modelling)
- *Hierarchical beautification*
  - *Partition* model hierarchically into suitable sub-parts
    - E.g. determine symmetry breaks and take model apart such that sub-parts are more symmetric
  - Beautify sub-parts separately and *re-combine* sub-parts
    - Requires suitable relations between sub-parts (rotations, translations, etc. to specify relative positions and symmetries)

Hierarchical Decomposition





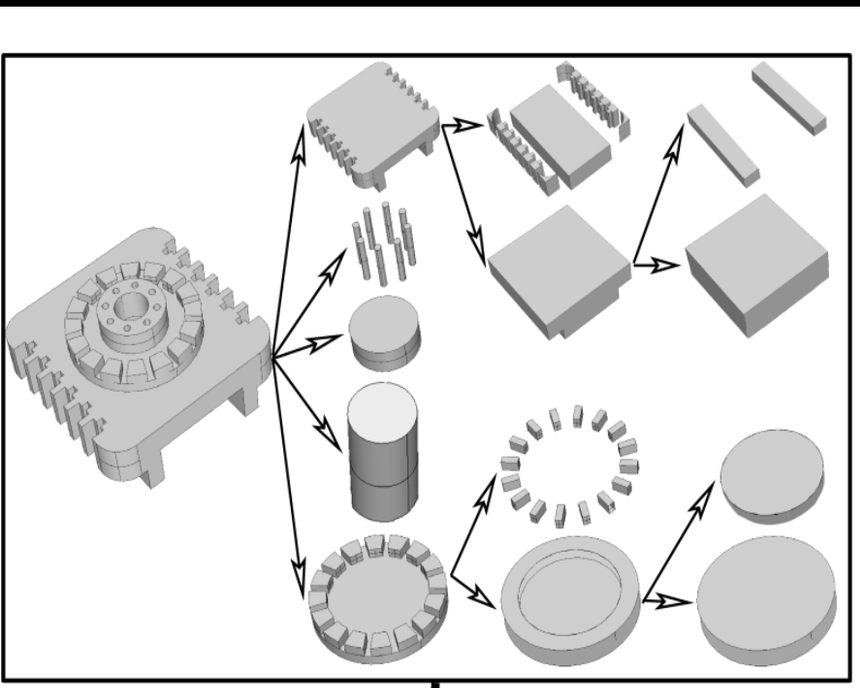
Symmetries

- What is a shape regularity of an object?
  - Use *symmetries* of various *shape properties*
- Symmetries as *invariants* [Felix Klein's Erlanger programme]
  - A symmetry of an object is an operation which does not change it (or certain aspects of it)
    - E.g. rotate a square about its centre by $90°$, ...
- Symmetries as *generators* [M. Leyton, generative geometry]
  - A symmetry of an object is an operation which generates the whole object from sub-parts
    - E.g. create a square by rotating an edge, ...
- Must detect *approximate* symmetries!

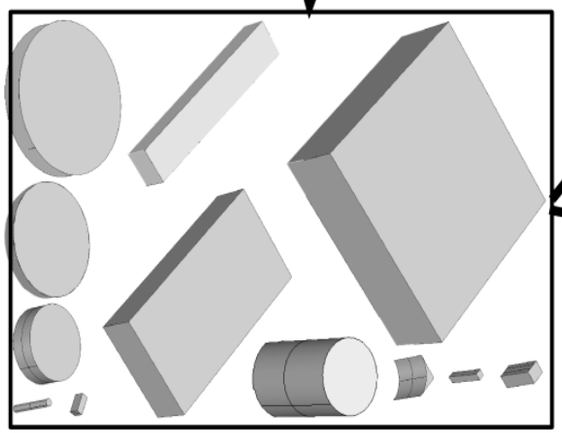
Regularities in Hier. Decomp.

- *Decompose* objects into small sub-parts
- Detect regularities in sequence
  - *Congruent* sub-parts
  - *Symmetries* in each set of congruent sub-parts
  - *Symmetric arrangements* of congruent sub-parts
- Merge compatible symmetries
- Use *approximate symmetries*
  - Careful consideration of different tolerance levels
  - Exact conditions on when a approximate symmetry is present *unambiguously*

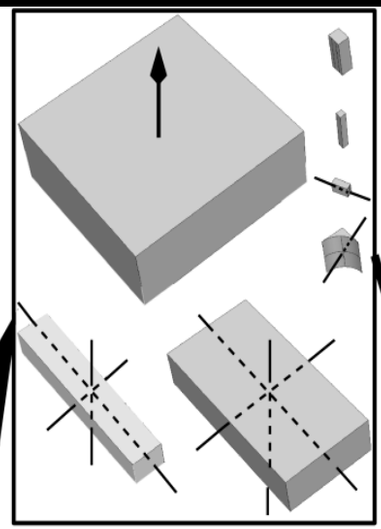
Regularities in Hier. Decomp.



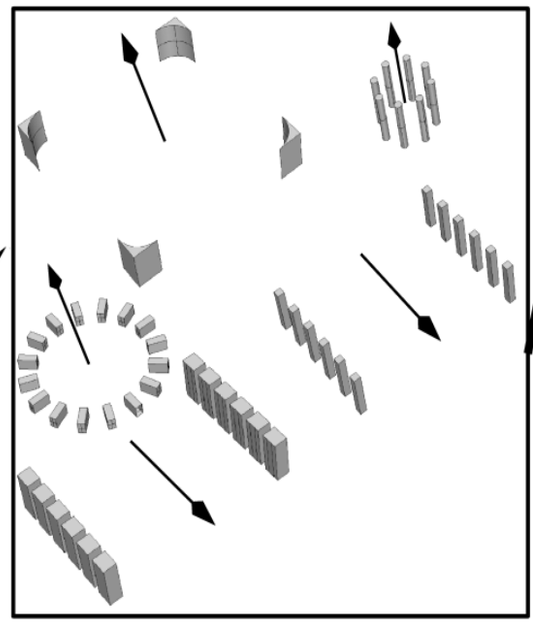
(a) Regularity feature tree



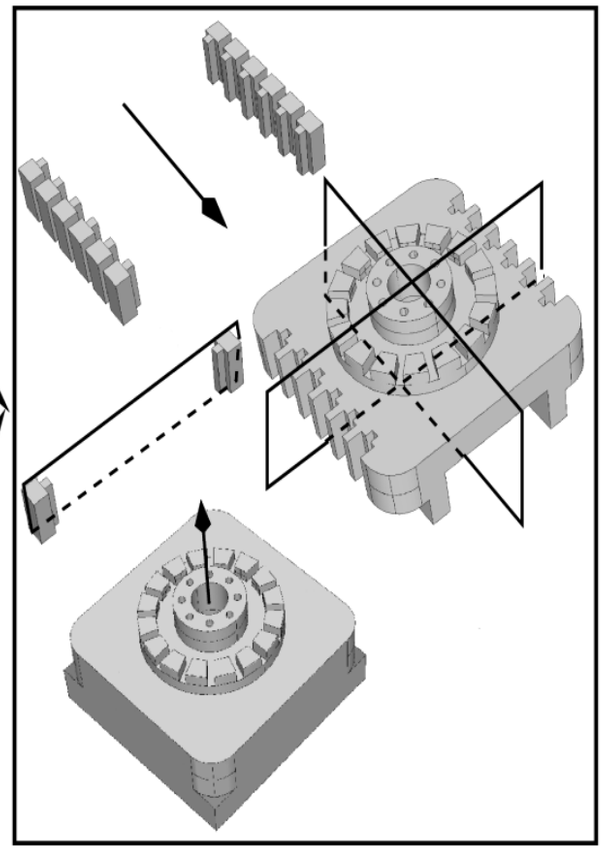
(b) Example part for each set of congruent leaf-parts



(c) Symmetries within each congruent leaf part



(d) Symmetric arrangements of the leaf-parts



(e) Examples of symmetries merged into compatible symmetries induced by a single isometry

Hacking?

- *Yes:* legal situation not always clear
  - Reverse engineering may circumvent copyright mechanisms
    (Digital Millennium Copyright Act, etc. . .)
  - Patents limit use of reverse engineering results
  - There are some legal exceptions referring specifically to reverse engineering

Hacking?

- *But:* perfectly legal applications exist
  - Often companies have to recover concepts developed by them
  - Engineering parts: plans got lost, never existed in digital form, need for replacement parts, . . .
  - Software: cleanup legacy systems, determine how to inter-operate with existing software, . . .

Conclusion

- Reverse engineering
  - *Interesting applications* directly relevant to industry problems
  - Involves many *core issues*
  - Reverse perspective provides *new insights* in process
- Design intent essential for working on *high abstraction levels*
- Symmetry is a very useful concept, but must use *approximate symmetries*
- Talk to me if you are interested in student projects on reverse engineering, CAD, geometry, graphics, simulations, physics...

Some Resources

- http://www.langbein.org/research/did, http://www.langbein.org/research/BoRG
- http://www.ipab.inf.ed.ac.uk/mvu/
- http://www.sztaki.hu/gml/
- T. Várady, R. R. Martin, J. Cox. Reverse Engineering of Geometric Models—an Introduction, Computer-Aided Design, 29(4):255–268, 1997.
- Some CAD applications with design intent features:
  - UGS PLM and Solid Edge: http://www.ugs.com/
  - Solid Works: http://www.solidworks.com/