Converting 3D Furniture Models to Fabricatable Parts and Connectors

Manfred Lau[1*] Akira Ohgawara[1,3] Jun Mitani[1,2] Takeo Igarashi[1,3]
[1]JST ERATO Igarashi Design Interface Project, Tokyo Japan
[2]University of Tsukuba [3]The University of Tokyo

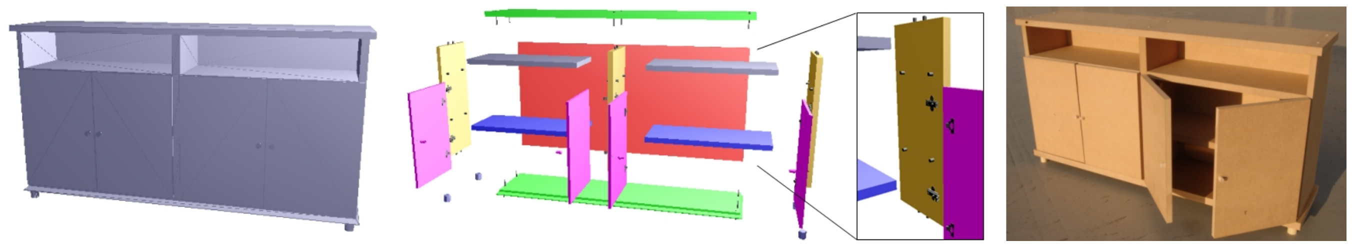

Figure 1: *Left: Arbitrary 3D model of IKEA ALVE cabinet downloaded from Google 3D Warehouse. Middle: Fabricatable parts and connectors generated by our algorithm. Right: We built a real cabinet based on the structure and dimensions of the generated parts/connectors.*

Abstract

Although there is an abundance of 3D models available, most of them exist only in virtual simulation and are not immediately usable as physical objects in the real world. We solve the problem of taking as input a 3D model of a man-made object, and automatically generating the parts and connectors needed to build the corresponding physical object. We focus on furniture models, and we define formal grammars for IKEA cabinets and tables. We perform lexical analysis to identify the primitive parts of the 3D model. Structural analysis then gives structural information to these parts, and generates the connectors (i.e. nails, screws) needed to attach the parts together. We demonstrate our approach with arbitrary 3D models of cabinets and tables available online.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Modeling Packages;

Keywords: 3D modeling, procedural modeling, fabrication, grammar, assembly instructions, exploded view illustrations

1 Introduction

The use of 3D models for non-professionals has become widespread in recent years, as users can easily download them [Shilane et al. 2004] from the internet or create their own 3D models [Igarashi et al. 1999]. For example, you can find far more varieties of virtual furniture models on the internet than real ones in your nearby furniture store. Our goal is to enable individual users to "print" their favorite 3D model to obtain real furniture to leverage these resources. This is partly inspired by recent interests in personal fabrication [Gross 2007; Landay 2009], in which individual users design and build personalized products instead of just buying mass-produced ones. However, printing using standard materials such as wooden plates is difficult because virtual 3D models do not have the structure necessary for physical construction. We solve the problem of taking as input a 3D model of a man-made object, and automatically generating the parts and connectors needed to build the corresponding physical object. We focus on furniture models that end-users have the ability to build with standard wooden materials. Methods for fabricating real objects from virtual models exist for other specific object types [Mori and Igarashi 2007; Saul et al. 2011].

Our work is inspired by Agrawala et al.'s work [2003] for creating step-by-step assembly instructions from 3D models and Li et al.'s work [2008] for visualizing explosion diagrams of existing 3D models of parts. These previous methods assume the existence of parts and connectors as input, and they do not begin from generic 3D models. Our work bridges the gap between generic 3D models and these visualization methods.

Our method uses a formal grammar defined for each type of object for structural analysis. We developed one grammar for cabinets from 42 types of real IKEA cabinets/bookcases. While there are minor differences in the structure and connection types among these cabinets, the underlying framework for their construction is similar and our grammar captures this framework. We also developed one grammar for tables from 11 types of IKEA tables. Each grammar describes a set of directed graphs. Each directed graph represents an object, each node of the graph represents a part, and each edge of the graph represents a connection. Each part and connection type also includes information, which we call *expert rules*, used when generating the parts and connectors. For example, we have a rule to specify the number and positions of nails to use for connecting two part types. We use examples of real IKEA furniture to derive these rules, and hence we call them *IKEA-expert rules*.

We first perform lexical analysis to identify separate tokens (primitive shapes) of the 3D model. This process gives us a primitive graph, which consists of primitive shapes and their contact relationships. We then use the grammar to apply structural analysis to the primitive graph and derive a fabricatable graph. This process gives detailed specification of the primitives and how to connect them. Structural analysis implicitly performs structure completion of missing parts and produces a sequence of assembly instructions for building the actual object.

*e-mail: manfred.lau@gmail.com

We demonstrate our approach with models of IKEA-style cabinets and tables downloaded from Google 3D Warehouse and other websites (Princeton Shape Benchmark [Shilane et al. 2004]). All our test models have arbitrary topology and structure, and none of them has the geometry or information for the connectors needed for fabricating the real object. Our contributions are: (1) We solve the problem of taking as input an arbitrary 3D furniture model, and generating as output fabricatable parts and connectors; and (2) We solve this problem by defining a single formal grammar for many varieties of each type of object.

1.1 Related Work

Grammar-based Modeling. There exists previous research that use grammar-based methods for understanding and modeling new architecture [Stiny 1980], cities [Parish and Müller 2001], buildings [Müller et al. 2006], details of facades [Wonka et al. 2003], and plants [Prusinkiewicz and Lindenmayer 1991]. Previous methods have also explored the procedural modeling of larger shapes from a smaller input example [Merrell and Manocha 2008] and the inverse procedural modeling of 3D shapes [Bokeloh et al. 2010]. The novelty of our work is in the application of a grammar for automatically parsing furniture models into separate buildable parts and connectors.

Mesh Segmentation. Segmenting 3D meshes is a well studied problem. The publicly available 3D mesh segmentation benchmark [Chen et al. 2009] compares between segmentation methods that use K-means [Shlafman et al. 2002], random walks [Lai et al. 2009], primitive fitting [Attene et al. 2006], randomized cuts [Golovinskiy and Funkhouser 2008], core extraction [Katz et al. 2005], and the shape diameter function [Shapira et al. 2008]. The novelty of our approach is that we parse, label, and re-mesh the original mesh into fabricatable parts. Previous segmentation methods do not consider segmentation into fabricatable parts, and this is important if the actual physical parts were to be built into a stable and usable product.

Mesh Analysis. There has been much recent work on analyzing 3D meshes for various purposes. These include computing the upright orientation of objects [Fu et al. 2008], manipulating joints connecting different components of meshes [Xu et al. 2009], re-meshing a triangle model to a quad-dominant mesh [Lai et al. 2010], and computing the part correspondence between meshes for the purpose of style-content separation [Xu et al. 2010]. Oh et al.'s work [2006] is closely related, as they create physical wood pieces with joints for fitting them together. Whiting et al.'s work [2009] also uses procedural modeling, and focuses on creating buildings that are structurally feasible. Our work focuses on finding the appropriate connectors such as nails or screws for the purpose of actual construction.

Personal Fabrication. We believe that individual users will play a role in designing and creating their own products in the future [Gross 2007; Landay 2009]. Non-professional users can already build their own customized plush toys [Mori and Igarashi 2007] and chairs [Saul et al. 2011]. We contribute to this area by solving the overall problem of converting a 3D mesh to separate fabricatable parts and connectors.

2 Formal Grammar

We first define a formal grammar for describing each type of object, give a concrete example for the 2D case of cabinets, and then describe extensions to the general 3D case. Our grammar is inspired by those from programming languages [Manning and Schutze 1999].

2.1 Grammar

We define a grammar for each type of object (i.e. cabinets or tables). We assume that the classification of object type of our input mesh is known, and use the corresponding grammar for each input mesh. Defining a grammar is useful as we use it as a recipe to autonomously perform mesh analysis algorithms to solve our overall problem. A grammar is defined as (N, Σ, P, S) where N is the set of non-terminal symbols, $\Sigma = \{\Sigma_{nodes}, \Sigma_{edges}\}$ is the set of terminal symbols for nodes and edges, P is the set of production rules, and $S \in N$ is the start symbol. Our language specifies a directed graph. We could represent the graph as a string of the language, but we choose to draw the graph itself for better intuition. Each node of the graph represents a separate part of the object, and each edge represents a connection between two parts that are in contact.

Each type of terminal symbol $t \in \Sigma_{nodes}$ has a pre-defined primitive shape $Shape_t$ and a set of example dimensions $ExampleDim_t$. The set $ExampleDim_t$ comes from measurements of real IKEA cabinets. A specific instantiation of a part has dimensions dim_t. Each type of connection $a \rightarrow b$ (where $a, b \in \Sigma_{nodes}$) has a set of example connectors connecting parts a and b: $(Dim_a, Dim_b, Type_c, Number_c, Transformations_c)$ where $Type_c$ is the type of connector (i.e. type of screw), $Number_c$ is the number of connectors, and $Transformations_c$ contains the position and orientation of each connector relative to parts a and b. This set also comes from real examples of furniture. A specific instantiation of a connection has a specific $(type_c, number_c, transformations_c)$, and a relative change in position and orientation $delta_c$ (between parts a and b).

2.2 Concrete Example for 2D Cabinets

We define $N = \{S, B, X, Y\}$ and $\Sigma = \{hb, ht, v, ha, leg, wheel\}$, where S is the start symbol, B represents the cabinet's base, X represents a compartment separated by vertical walls, Y represents a compartment separated by horizontal walls, hb represents horizontal-bottom (the horizontal and bottom piece of the cabinet), ht represents horizontal-top, v represents vertical, and ha represents horizontal-adjustable. Figure 2 shows the production rules P. Inspired by terminology from programming languages, we define hb, ht and ha as subtypes of h. We specify certain relationships in the rules for the purpose of structure completion. For example, in rules 8-9, the two $wheel/leg$ parts are symmetric with respect to hb. A directed edge specifies that the two corresponding parts are in contact. An undirected edge means that the two parts are not in contact; the edge exists to provide spatial intuition. Figure 3(a,b) shows an example cabinet. In the graph, the directed edges specify the positioning of the parts. This is best shown in Figure 3(c,d).

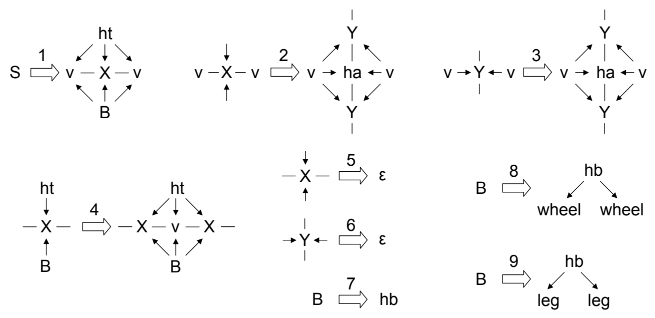

Figure 2: *Production rules for 2D cabinets. ε is the empty graph.*

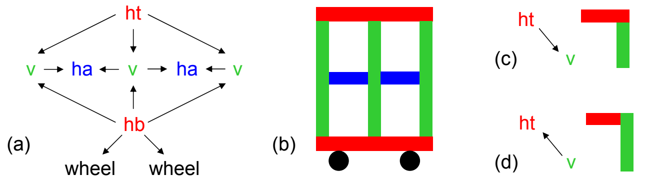

Figure 3: *(a) A graph from our example grammar, and (b) the corresponding cabinet. Each node contains the dimensions of each part (dim_t). Each edge contains $delta_c$ and ($type_c, number_c, transformations_c$). (c,d) The direction of the edges specifies the positioning of the parts. The positionings are important for ensuring that the parts are structurally buildable.*

Although our production rules are manually defined, it is quite intuitive to define them. In general, we typically have one rule per terminal symbol for each family of models. For example, in Figure 2, rule 2 is for adding an ha to the overall structure, rule 3 is for adding more ha's, rule 4 is for adding v, and rules 8/9 are for adding $wheels/legs$. While we need to define new rules to handle new object types, this does not affect the generality of our approach to a larger set of rules. This is similar to grammar rules for natural or programming languages where we have to define new rules to describe new grammar-types.

2.3 Extensions to General 3D Case

We show the grammars for 3D cabinets and tables in the supplementary material. We extend the 2D cabinet grammar to 3D by including these additional parts: back, front door, front drawer, and front handle. We also define extra production rules for 3D cabinets to demonstrate the extension of the grammar to handle another type of cabinet models. For 3D table models, our grammar describes tables with a tabletop and four legs or two supporting sides.

3 Conversion to Parts and Connectors

Our framework is inspired by programming language analysis. We first perform lexical analysis to identify separate tokens (i.e. primitives shapes) from the 3D model. Structural analysis then uses the grammar to autonomously generate fabricatable parts and connectors. As a consequence of this process, we can perform structure completion, and generate a sequence of assembly instructions. We use an example 2D cabinet to illustrate our algorithms throughout this section.

3.1 Lexical Analysis

This process takes a 3D model as input, and identifies its separate primitive parts (Figure 4). We first convert the input model into voxel representation. The motivation for the voxel representation is that parsing the voxels into parts and then converting the parts back to a mesh representation allows the original model to be segmented and re-meshed effectively. We parse the voxel representation by scanning it with a set of primitive shapes [Attene et al. 2006]. The shapes that we parse include horizontal pieces, vertical (side) pieces, vertical (front-facing) pieces, wheels, legs, and front handles. We assume that the orientations of the model are given as is common in [Agrawala et al. 2003; Li et al. 2008], or can be found automatically with [Fu et al. 2008]. To parse for horizontal (or other axis-aligned) pieces, we scan the voxel for the default horizontal minimum shape (a cuboid), and "expand" this cuboid until we have reached the default maximum shape or until the percentage of empty voxels to cuboid volume has reached the default limit. This limit and the default minimum/maximum shapes are found with $ExampleDim_t$. We repeat scanning for the default minimum shape until we cannot find one. To parse for wheels/legs, we use the same procedure except we search for a spherical/cylindrical shape, and the "expanding" process is successful if the expanded shape encloses voxels and the immediate outward parts are empty voxels. We search for the larger axis-aligned pieces first such that this step can be fast. The remaining voxels are assumed to be front handles. We represent each parsed shape by a node in our graph. Lexical analysis may fail if individual front pieces are too close together that the voxel representation do not consider them as separate pieces. The axis alignment is not a requirement, as long as we have the appropriate shape detector to identify each shape.

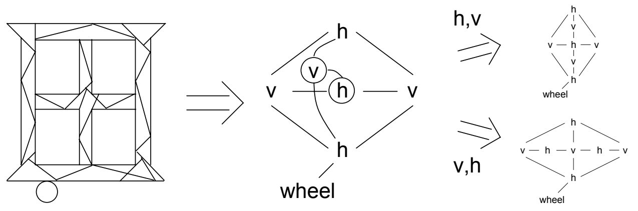

Figure 4: *Lexical Analysis: The input can be an irregularly-shaped mesh (left). We convert the mesh to a voxel representation and fit primitive shapes to it. The resulting shape of each part may not necessarily have that exact primitive shape (i.e. top and bottom horizontal pieces). The circled parts v and h (middle) completely cross through each other, and we parse them again in either order to create two possible primitive graphs (right).*

We perform contact detection between each pair of nodes of the graph to test if the corresponding two primitives are in contact. We check if the two primitives have common voxels or if the neighboring voxels of one primitive includes the other. If two primitives are in contact, we add an edge to the graph to connect those two nodes. We now have our *primitive graph*. At this stage, we allow multiple shapes to overlap at corners and T-junctions. However, if any two parts completely cross through or intersect through each other, we parse them again in either order to create two possible graphs (Figure 4).

3.2 Structural Analysis

We apply structural analysis to all primitive graphs. The correctly parsed primitive graph survives, and is augmented to a *fabricatable graph* (Figure 5).

3.2.1 Fabricatable Parts and Connectors Generation

For this part of the algorithm, we slightly alter and augment the production rules in two ways. First, we replace every occurence of each subtype in the rules with the corresponding higher type. In our 2D example, we replace each hb, ht, ha with h. Second, for each "empty" rule (i.e. a rule with the right side being empty), we iterate through each non-empty rule and generate additional rules by replacing subgraphs on the right side of the non-empty rule that match the left side of the empty rule with the empty graph. All combinations of such empty rule replacement are taken (i.e. there can be multiple replacements for each non-empty rule), and a new additional rule is generated for each case. We then delete the empty rules to form the set of augmented rules.

With these augmented rules, we take each primitive graph and perform a backward tracing procedure to it. The backward tracing tries to find a subgraph in the current graph that matches the right side of an augmented production rule and applies that rule in reverse. We use a simple tree-search enumeration procedure for subgraph matching. Since the edges in the primitive graphs do not contain directions, the matching ignores the direction of the edges. The matching also considers the part relationships defined in the rules, and considers the matching to be successful if a missing part can be generated from the others. For example, rule 8 of our 2D grammar specifically includes a condition that when performing subgraph matching, one of the wheels can be missing. We allow for backtracking in the tracing process if no rules can be applied. This can potentially lead to a combinatorial explosion, but this is not an issue as both the subgraphs and number of rules are small. We can use more sophisticated matching algorithms if the rules are more complex. If no production rules can be applied in reverse and we have not arrived at S, that primitive graph is not chosen. The primitive graph that can be traced back to S survives. The *most important aspect of this backward tracing process* is that by re-applying the production rules in the forward direction starting from S, we generate the fabricatable graph. Hence we have re-labeled the parts and found the direction of the edges. In our example, the backward tracing procedure matches this backward order of rules: {8,6,6,2,6,6,2,4,1}. Applying these rules in the forward order starting from S gives us the graph in Figure 5 right. The backward tracing process can fail if the object's structure is different from those in the grammar or the 3D model is missing too many parts. In either case, no primitive graph survives.

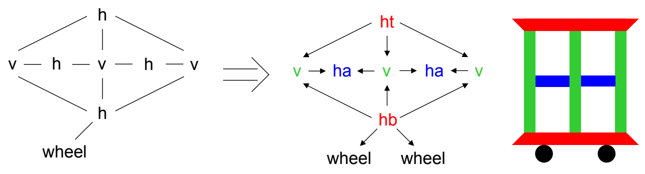

Figure 5: *Structural Analysis: The primitive graph (left) that matches the formal grammar is augmented to a fabricatable graph (right) that represents separated parts and connectors (not shown). This procedure re-labels, re-positions, and re-meshes the parts. We also perform structure completion to generate the other wheel.*

With the fabricatable graph, we re-position the parsed parts such that they can be physically attached together. We iterate through each directed edge and re-position the parts as shown in Figure 3(c,d). Each resulting part has dimensions dim_t. For 3D meshes that only contain the surface of the shape, the thicknesses of each part can be small. For these cases, the system automatically assigns default thicknesses derived from $ExampleDim_t$. For example, given the length and width of a cuboid part, we use a k-nearest-neighbor regression technique and $ExampleDim_t$ to find its depth.

We perform re-meshing by converting each part's voxels into a 3D mesh with the marching cubes [Lorensen and Cline 1987] algorithm. In our example in Figure 5, the top and bottom parts are not exact rectangular shapes, and we re-mesh the parsed pixels. In these cases, the bounding box of the re-meshed shape has dimensions dim_t. If the physical object were to be built, the user may find it convenient to approximate shapes as cuboids and not perform re-meshing. This was the case for the horizontal-bottom piece of the real cabinet in Figure 1.

For 3D cabinets, front doors and drawers are both originally parsed as vertical front-facing parts. We differentiate between them with an IKEA-expert rule. This rule uses the dimensions of the part and corresponding front handle, and example data $ExampleDim_t$ (which includes the dimensions of front doors and drawers from real IKEA furniture). Details of when to apply this rule is provided in the supplementary material. For front doors, we identify their axes of rotation for visualization purposes. We assume there are two possibilities for the axis (i.e. opening from left or right side), and we identify it with an IKEA-expert rule. For front drawers, the input 3D models that we have tested contain only the surface of the overall cabinet and hence the front piece of the drawer. We generate four additional rectangular pieces with another IKEA-expert rule to form a complete drawer.

We generate connectors for each directed edge of the fabricatable graph. We use the example set of connectors $(Dim_a, Dim_b, Type_c, Number_c, Transformations_c)$ for each edge $a \rightarrow b$. We have the values of dim_a and dim_b from the parts, and we use them to generate the values of $(type_c, number_c, transformations_c)$ with a k-nearest-neighbor regression technique.

3.2.2 Structure Completion

A consequence of the pattern matching method is structure completion of the original 3D model. In our example in Figure 5, we generate the geometry and position of the other wheel based on rule 8 and the geometry of the existing wheel. Structure completion is not the main goal of this paper, and the process only works for those cases that are defined in the production rules of the grammar.

3.2.3 Assembly Instructions Generation

Our main purpose is to generate parts and connectors given a 3D mesh. Our output can be used with Agrawala et al.'s work [2003] to generate assembly instructions. However, an interesting consequence of our overall approach is that the sequence of production rules from S to the fabricatable graph corresponds to a possible set of instructions for assembling the object. To generate a set of instructions, we take the sequence of production rules in the forward direction, combine consecutive steps of the same production rule into one step, and generate a graph for each step starting from S. We then keep only the non-terminal symbols and associated edges in each graph. The resulting sequence of graphs may have consecutive graphs that are the same, in which case we keep only one of each in our sequence. For each graph in the remaining sequence, we display one image of the virtual parts and connectors corresponding to the graph. Figure 6 shows one example. In some cases, there can be more than one possibility of steps (all of which can make intuitive sense) depending on the order of production rules the backward tracing process iterates through. Regardless of the possibility that is chosen, the final image (i.e. the separated parts and connectors) is the same.

4 Results

We tested our method with IKEA-style and other arbitrary 3D cabinet/bookcase models (Figures 1 and 7). These are downloaded from Google 3D Warehouse, the Princeton Shape Benchmark [Shilane et al. 2004] and other websites. For a set of 13 cabinets/bookcases from the benchmark, our original grammar works successfully with 10 of them. The others (one of which is in Figure 7 bottom right) did not work as their structures are somewhat different. We therefore defined additional production rules to represent these models and incorporated the rules into the original 3D cabinet grammar (see supplementary material). The new grammar then works for all 13 examples. The runtime for each example is at most one second. Conversion to voxel representation occupies most of the runtime. We use a simple method to explode the parts for better visualization. The output of our method can be used as input to Li et al.'s method [2008] for creating better exploded view diagrams.

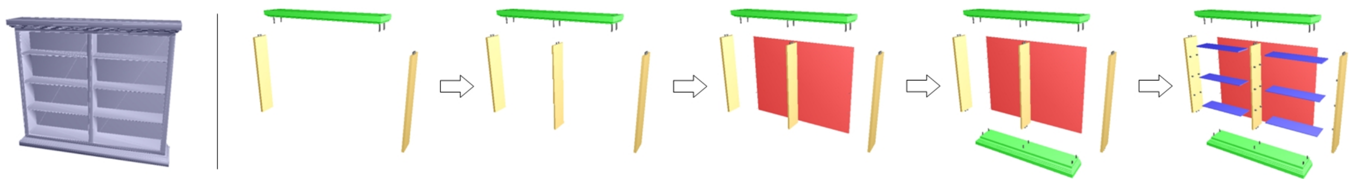

Figure 6: *Model m844 (left) from the Princeton Shape Benchmark, and assembly instructions generated from our grammar. The focus of our work is to generate fabricatable parts and connectors, but our approach can also generate assembly instructions.*

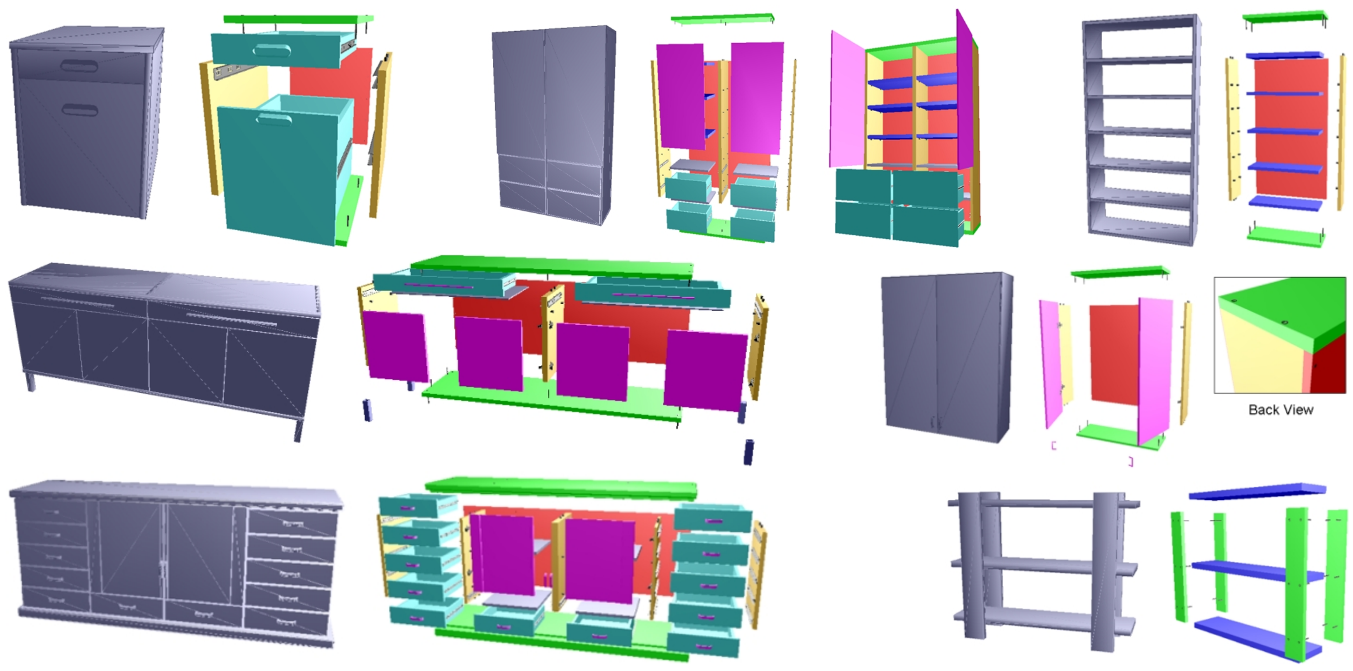

Figure 7: *Input 3D models and fabricatable parts/connectors generated by our algorithm for arbitrary cabinet/bookcase models from Google 3D Warehouse and the Princeton Shape Benchmark. From top left: GALANT (IKEA), BESTA, m864 (benchmark), BJURSTA, m957, model from other website, m860. Top right example demonstrates structure completion of the back piece. Bottom right example did not work with our original cabinet grammar, but it worked after we added additional production rules.*

We also defined a grammar for 3D tables (see supplementary material) and demonstrated it with various 3D table models (Figure 8). Our framework works well with most of the models from the shape benchmark. For the models that did not work, the lexical or structural analysis fails to parse the 3D shape because the structure of these models are quite different from those represented by the grammar. Defining additional production rules to represent such models is typically a good starting point for handling such cases.

5 Discussion

We have developed a framework for generating fabricatable parts and connectors from a 3D furniture model. To make our framework more general in the future, we can take an arbitrary mesh of *any object*, classify the object into a specific object type and then use the grammar for that type to perform the mesh conversion. The main limitation of our system is that the grammar and expert rules are defined manually. For future work, it would be useful to automatically learn the grammar given examples of a type of object.

We currently do not make verifications of object dimensions. One possibility is that vertical pieces of the cabinets should have the same height. Another possibility is that there is currently no limit for the number of ha's in the graph, but there is a limit based on the cabinet's height and the height of each ha. More general shape verifications can be performed with the output of our system. Exploring shape completion from noisy scanned data [Nan et al. 2010] together with our grammar-based mesh algorithms can also be an interesting direction for future work.

References

AGRAWALA, M., PHAN, D., HEISER, J., HAYMAKER, J., KLINGNER, J., HANRAHAN, P., AND TVERSKY, B. 2003. Designing effective step-by-step assembly instructions. *ACM Transactions on Graphics 22*, 3, 828–837.

ATTENE, M., FALCIDIENO, B., AND SPAGNUOLO, M. 2006. Hierarchical mesh segmentation based on fitting primitives. *Visual Computer 22* (March), 181–193.

BOKELOH, M., WAND, M., AND SEIDEL, H.-P. 2010. A connection between partial symmetry and inverse procedural modeling. *ACM Transactions on Graphics 29*, 4, 104.

CHEN, X., GOLOVINSKIY, A., AND FUNKHOUSER, T. 2009. A benchmark for 3D mesh segmentation. *ACM Transactions on Graphics 28*, 3, 73.

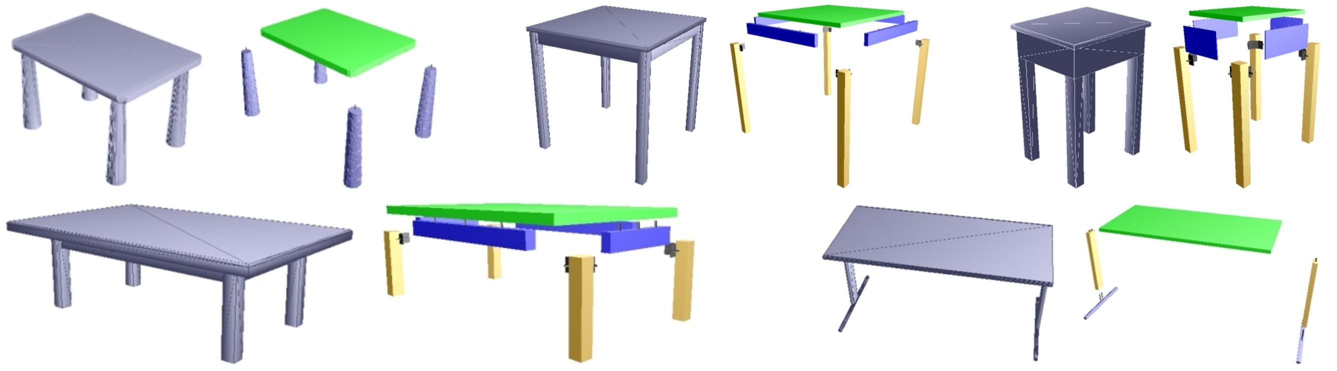

Figure 8: *Input 3D models and fabricatable parts/connectors generated by our algorithm for two IKEA table models (MAMMUT, NORDEN) from Google 3D Warehouse and three arbitrary table models (m871, m895, m907) from the Princeton Shape Benchmark. The legs of the MAMMUT table were parsed as cuboids and re-meshed to conic shapes.*

FU, H., COHEN-OR, D., DROR, G., AND SHEFFER, A. 2008. Upright orientation of man-made objects. *ACM Transactions on Graphics 27*, 3, 42.

GOLOVINSKIY, A., AND FUNKHOUSER, T. 2008. Randomized cuts for 3D mesh analysis. *ACM Transactions on Graphics 27*, 5, 145.

GROSS, M. 2007. Now more than ever: computational thinking and a science of design. *Japan Society for the Science of Design 16*, 2, 50–54.

IGARASHI, T., MATSUOKA, S., AND TANAKA, H. 1999. Teddy: a sketching interface for 3d freeform design. *ACM SIGGRAPH*, 409–416.

KATZ, S., LEIFMAN, G., AND TAL, A. 2005. Mesh segmentation using feature point and core extraction. *The Visual Computer (Pacific Graphics) 21*, 649–658.

LAI, Y.-K., HU, S.-M., MARTIN, R. R., AND ROSIN, P. L. 2009. Rapid and effective segmentation of 3d models using random walks. *Computer Aided Geometric Design 26* (Aug.), 665–679.

LAI, Y.-K., KOBBELT, L., AND HU, S.-M. 2010. Feature aligned quad dominant remeshing using iterative local updates. *Computer Aided Design 42* (Feb.), 109–117.

LANDAY, J. 2009. Design tools for the rest of us. *Communications of the ACM 52*, 12, 80.

LI, W., AGRAWALA, M., CURLESS, B., AND SALESIN, D. 2008. Automated generation of interactive 3d exploded view diagrams. *ACM Transactions on Graphics 27*, 3, 101.

LORENSEN, W. E., AND CLINE, H. E. 1987. Marching cubes: A high resolution 3d surface construction algorithm. *ACM SIGGRAPH*, 163–169.

MANNING, C., AND SCHUTZE, H. 1999. *Foundations of Statistical Natural Language Processing*. MIT Press.

MERRELL, P., AND MANOCHA, D. 2008. Continuous model synthesis. *ACM Transactions on Graphics 27*, 5, 158.

MORI, Y., AND IGARASHI, T. 2007. Plushie: an interactive design system for plush toys. *ACM Transactions on Graphics 26*, 3, 45.

MÜLLER, P., WONKA, P., HAEGLER, S., ULMER, A., AND VAN GOOL, L. 2006. Procedural modeling of buildings. *ACM Transactions on Graphics 25*, 3, 614–623.

NAN, L., SHARF, A., ZHANG, H., COHEN-OR, D., AND CHEN, B. 2010. Smartboxes for interactive urban reconstruction. *ACM Transactions on Graphics 29*, 4, 93.

OH, Y., JOHNSON, G., GROSS, M. D., AND DO, E. Y.-L. 2006. The designosaur and the furniture factory: simple software for fast fabrication. *International Conference on Design Computing and Cognition*, 123–140.

PARISH, Y. I. H., AND MÜLLER, P. 2001. Procedural modeling of cities. *ACM SIGGRAPH*, 301–308.

PRUSINKIEWICZ, P., AND LINDENMAYER, A. 1991. *The Algorithmic Beauty of Plants*. Springer Verlag.

SAUL, G., LAU, M., MITANI, J., AND IGARASHI, T. 2011. SketchChair: An all-in-one chair design system for end-users. *International Conference on Tangible, Embedded and Embodied Interaction (TEI)*, 73–80.

SHAPIRA, L., SHAMIR, A., AND COHEN-OR, D. 2008. Consistent mesh partitioning and skeletonisation using the shape diameter function. *Visual Computer 24* (March), 249–259.

SHILANE, P., MIN, P., KAZHDAN, M., AND FUNKHOUSER, T. 2004. The Princeton Shape Benchmark. *Shape Modeling International*, 167–178.

SHLAFMAN, S., TAL, A., AND KATZ, S. 2002. Metamorphosis of polyhedral surfaces using decomposition. *Computer Graphics Forum*, 219–228.

STINY, G. 1980. Introduction to shape and shape grammars. *Environment and Planning B 7*, 343–361.

WHITING, E., OCHSENDORF, J., AND DURAND, F. 2009. Procedural modeling of structurally-sound masonry buildings. *ACM Transactions on Graphics 28*, 5, 112.

WONKA, P., WIMMER, M., SILLION, F., AND RIBARSKY, W. 2003. Instant architecture. *ACM Transactions on Graphics 22*, 3, 669–677.

XU, W., WANG, J., YIN, K., ZHOU, K., VAN DE PANNE, M., CHEN, F., AND GUO, B. 2009. Joint-aware manipulation of deformable models. *ACM Transactions on Graphics 28*, 3, 35.

XU, K., LI, H., ZHANG, H., COHEN-OR, D., XIONG, Y., AND CHENG, Z. 2010. Style-content separation by anisotropic part scales. *ACM Transactions on Graphics 29*, 5, 184.