Carpentry Compiler

CHENMING WU, Tsinghua University and University of Washington
HAISEN ZHAO, University of Washington
CHANDRAKANA NANDI, University of Washington
JEFFREY I. LIPTON, University of Washington
ZACHARY TATLOCK, University of Washington
ADRIANA SCHULZ, University of Washington

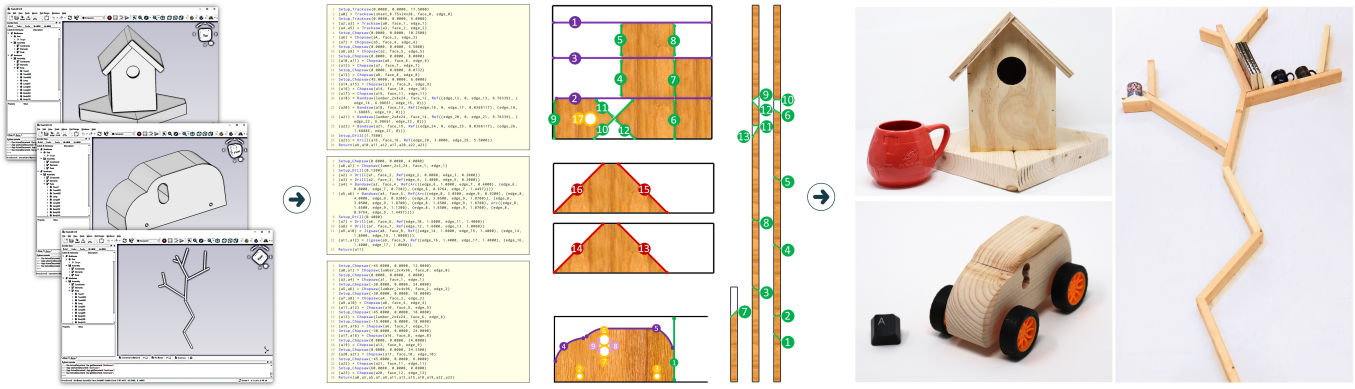


Fig. 1. Our carpentry compiler converts high-level geometric designs made by users to low-level fabrication instructions that can be directly followed to manufacture parts. The compiler performs multi-objective optimization on the low-level instructions to generate Pareto-optimal candidates.

Traditional manufacturing workflows strongly decouple design and fabrication phases. As a result, fabrication-related objectives such as manufacturing time and precision are difficult to optimize in the design space, and vice versa. This paper presents HL-HELM, a high-level, domain-specific language for expressing abstract, parametric fabrication plans; it also introduces LL-HELM, a low-level language for expressing concrete fabrication plans that take into account the physical constraints of available manufacturing processes. We present a new compiler that supports the real-time, unoptimized translation of high-level, geometric fabrication operations into concrete, tool-specific fabrication instructions; this gives users immediate feedback on the physical feasibility of plans as they design them. HELM offers novel optimizations to improve accuracy and reduce fabrication time as well as material costs. Finally, optimized low-level plans can be interpreted as step-by-step instructions for users to actually fabricate a physical product. We provide a variety of example fabrication plans in the carpentry domain that are designed using our high-level language, show how the compiler translates and optimizes these plans to generate concrete low-level instructions, and present the final physical products fabricated in wood.

CCS Concepts: • **Computing methodologies** → **Shape modeling**; *Graphics systems and interfaces*;

Additional Key Words and Phrases: Design for manufacturing, CAD, programming languages, hardware abstractions

ACM Reference Format:
Chenming Wu, Haisen Zhao, Chandrakana Nandi, Jeffrey I. Lipton, Zachary Tatlock, and Adriana Schulz. 2019. Carpentry Compiler. *ACM Trans. Graph.* 38, 6, Article 195 (November 2019), 14 pages. https://doi.org/10.1145/3355089.3356518

1 INTRODUCTION

Next-generation manufacturing techniques are revolutionizing the on-demand fabrication of complex custom products. This has spurred important research advances to enable *fabrication-oriented design optimization*. In many applications, however, *fabrication is design-dependent*, defined by a sequence of operations on multiple processes, where the order of operations and choice of hardware can only be optimized for a specific design. This scenario raises unexplored challenges since product design and fabrication instructions must be coupled, even as they are separately optimized.

The key insight of this work is that both designs and fabrication plans are programs. One of the most influential developments in computer architecture was the introduction of instruction set architectures (ISAs) [Patterson and Sequin 1981] which define abstract models of computers and serve as an interface between their software and hardware. This abstraction layer enabled the independent

Authors' addresses: Chenming Wu, wcm15@mails.tsinghua.edu.cn, Tsinghua University and University of Washington; Haisen Zhao, haisen@cs.washington.edu, University of Washington; Chandrakana Nandi, cnandi@cs.washington.edu, University of Washington; Jeffrey I. Lipton, jilipton@uw.edu, University of Washington; Zachary Tatlock, ztatlock@cs.washington.edu, University of Washington; Adriana Schulz, adriana@cs.washington.edu, University of Washington.

0730-0301/2019/11-ART195 $15.00
https://doi.org/10.1145/3355089.3356518

development of both hardware and software. Our work examines whether similar hardware and software decoupling can be achieved for manufacturing.

This work addresses the above question in the context of carpentry design and manufacturing for several reasons. First is application scope. Carpentered items comprise the structures we live in and the furniture they contain, and they are commonly personalized to fit spaces and functions. Second, carpentry provides an appropriate level of complexity for initiating research in this area. It combines multiple processes and assembly constraints within the confines of a bounded problem. Finally, recent advances in mobile robotics allow automated carpentry to occur outside factory floors, making the end-to-end design and fabrication of personalized carpentry not only possible but an exciting and open area of research [Lipton et al. 2018].

In this work, we introduce Hardware Extensible Languages for Manufacturing (HELM), a system that allows us to represent carpentry designs and fabrication instructions. We take inspiration from traditional compilers to propose two layers of abstraction, one high-level and process-agnostic (HL-HELM) and the other low-level and process-specific (LL-HELM). HL-HELM is a *design language* related to traditional parametric feature-based CAD languages but focused on subtractive operations that can be mapped to physical woodworking processes. LL-HELM is a *fabrication language*. Programs in LL-HELM can be directly followed to manufacture a part, where each operation in a program is drawn from an extensible set of fabrication instructions. We also propose a new compiler that verifies HL-HELM code and optimizes fabrication instructions (LL-HELM code). Because the target language, LL-HELM, is process-specific, we design an architecture that is *extensible* to new hardware.

In addition to the abstraction and compilation system, a key technical contribution of our paper is a novel optimization algorithm for manufacturing enabled by our proposed pipeline. Cut planning directly affects the precision of manufactured parts, material wastage, and production time. Optimizing multi-process cuts is challenging because it involves a long sequence of interdependent steps with multiple conflicting objectives, and if not done properly, it can cause significant labor overhead. By representing the fabrication process as a program, we can draw on ideas from compiler optimization to find an efficient sequence of operations that meet user specifications. We adapt search-based superoptimization techniques based on *e-graphs* [Joshi et al. 2002; Tate et al. 2009].[1] E-graphs compactly represent (exponentially) large equivalence classes of terms, support extensibility via simple syntactic rewrites, and enable cooperation of various solvers through a common representation. However, applying e-graphs in the context of fabrication requires addressing several technical challenges: fabrication operations are generally not *linear*,[2] some operations do not map to standard algebraic operations, some equivalences are difficult to express as syntactic rewrites, and objectives are multiple and conflicting. Our work overcomes these challenges by developing new geometric solvers that communicate with e-graphs, and by extending e-graph extraction to produce a set of Pareto-optimal candidate fabrication plans.

We illustrate the advantages of our pipeline by demonstrating the expressiveness of our design tool and show how our compiler can automatically verify manufacturability. We demonstrate how the resulting fabrication instructions can be optimized to meet different user-specified objectives, such as accuracy, fabrication time, and material cost.

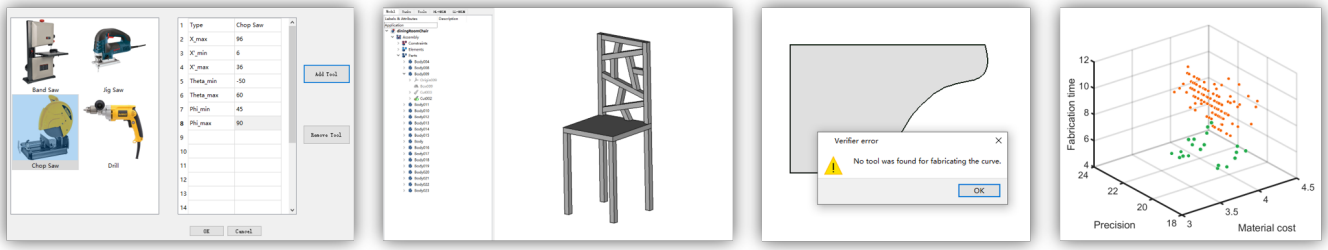
2 RELATED WORK

Fabrication-Oriented Design. Design for fabrication is gaining attention in the computer graphics community [Bickel et al. 2018]. Many newly proposed systems guide designers in searching the space of possible designs to both meet user specifications and ensure manufacturability. For example, several works optimize for design appearance [Dong et al. 2010; Lan et al. 2013], deformation behavior [Bickel et al. 2010; Ma et al. 2017], spinnability [Bächer et al. 2017], or buoyancy [Wang and Whiting 2016] while ensuring fabricability with an additive process. Other works focus on specific processes, such as interlocking quadrilateral elements [Skouras et al. 2015], plush toys [Mori and Igarashi 2007], LEGO [Luo et al. 2015], or zippables [Schüller et al. 2018]. These works all assume that a point in the design space completely determines the fabrication method. In contrast, our approach decouples fabrication from the design specification. In our approach, a design is created and optimized in HL-HELM, while the fabrication process is expressed and optimized in LL-HELM. We developed a new compiler that converts designs to fabrication instructions and verifies that a design is manufacturable with the available processes. It optimizes instructions for multiple objectives like precision, time, and material cost and can thus generate different fabrication plans depending on which objective is being optimized for. Thus, in our system, a single design can generate multiple diverse fabrication plans that can be optimized to meet differing requirements of the manufacturing facility.

Computer-Aided Manufacturing (CAM). Decades of CAM research focused on developing optimal fabrication plans for single specific fabrication processes, such as 5-axis milling [Zhao et al. 2018], sheet-metal stretching [Konaković et al. 2016], and 3D printing [Alexa et al. 2017; Dai et al. 2018]. An important effort to create a multi-process representation was STEP-NC [NC 2019], which abstracts away from machine-specific G-code operations to make tool-type-specific machining operations. These operations are interpretable or compilable on different hardware, allowing for inter-machine operations and closed-loop control at the tool-path level [Brecher et al. 2006; Xu and Newman 2006]. Extensions to the STEP-NC framework have permitted its expansion from multi-axis milling to other metal-working processes, such as Electrical Discharge Machining (EDM) [Sokolov et al. 2006], sheet metal forming [Xie and Xu 2006], and 3D printing [Um et al. 2017]. However, manual operations are still needed to convert a CAD file to a STEP-NC fabrication plan. More importantly, this task requires expert knowledge to select the fabrication process and verify that geometry is properly mapped to tooling operations. In contrast, our system is designed on top of process-level abstractions; thus, it is compatible with many different processes. Its optimization framework chooses the process for each

[1] A clarifying example for the use of e-graphs is arithmetic expression simplification [Panchekha et al. 2015]. Equivalences like commutativity and associativity are encoded as rewrite rules; a search engine then explores the space of all possible rewrites in order to minimize the expression's cost.

[2] Here "linear" is meant in the type-theoretic sense [Wadler 1990].



(a) Tool selection (b) Modeling UI (c) Verifier (d) Pareto front exploration

Fig. 2. Design process: First, users import a library of materials and tools so that the compiler can map design features to fabrication operations. Second, they create a design in an intuitive interface. Third, at each step of the design process, our verifier checks the manufacturability of the design; for example, in (c), the maximal radius of curvature is too big for the part to be fabricable using any of the available processes in the library. Fourth, the compiler generates a set of fabrication plans with different trade-offs. The instructions generated without optimization are shown in orange, and the outputs of our system (i.e., optimized instructions) are shown in green.

part automatically, with no human intervention. There are a few industrial CAM tools [DDX 2019; Solutions 2019; Čeli APS 2019] that can be used for carpentry. To the best of our knowledge, none of these tools can do cut planning on multiple machines, and none of them uses similar language abstractions.

Programming Languages for Geometric Modeling and Fabrication. Geometric modeling has a long-standing history of using domain-specific languages (DSLs) to describe a sequence of operations that construct geometry. These include early constructive solid geometry (CSG) approaches [Laidlaw et al. 1986], modern CAD scripting languages [FeatureScript 2019], and many procedural modeling systems [Müller et al. 2006; Prusinkiewicz et al. 1996; Schwarz and Müller 2015]. These languages represent design as a process and can be used for optimization as well as inverse design [Du et al. 2016; Nandi et al. 2018]. DSLs have also been used for describing fabrication for a single process, such as multi-material 3D printing [Vidimče et al. 2013] and knitting [McCann et al. 2016]. Our work draws on these ideas to define DSLs for *both* design *and* multi-process fabrication. The languages are developed to allow a compiler to efficiently validate a design and optimize the fabrication process.

Optimizing Compilers. Traditional programming language compilers typically have optimizations for minimizing execution time, memory, and power consumption using techniques like constant folding, loop unrolling, common sub-expression elimination, and dead store elimination [Aho et al. 1986] that are applied sequentially. With emerging architectures, developers can often identify architecture-specific-local *peephole* optimizations on top of traditional compiler optimizations. Other approaches [Bansal and Aiken 2006; Massalin 1987; Phothilimthana et al. 2016] use search algorithms to find optimal programs. E-graphs [Nelson 1980] offer a scalable approach for finding optimized programs that rely on equivalences between programs. Joshi et al.'s Denali [2002] for optimizing assembly programs and Tate et al.'s Equality Saturation [2009] for optimizing complex Java programs with loops and exceptions are two examples of optimizers that use e-graphs. This paper demonstrates a new application of e-graphs, i.e., optimizing a set of carpentry instructions. We developed algorithms to populate and modify the e-graph that rely on geometry when syntactic rewrites are not sufficiently expressive. Further, we developed new methods for e-graph extraction for multi-objective optimization.

Design and Fabrication for Carpentry. Our work is also related to computational design approaches for carpentry and furniture. Fu et al. [2015] suggest using an interlocking structure and Song et al. [2017] extend these ideas to designs that can be reconfigured. Umetani et al. [2012] propose an interactive exploration tool for furniture design, where structural stability is evaluated at interactive rates. Lau et al. [2011] address the problem of converting a manually designed 3D model into parts and connectors, while Li et al. [2015] target the foldability problem. These works propose fabrication-oriented design optimization, but they assume that a design uniquely determines a fabrication process. Our work builds upon those ideas, defining languages where both design and fabrication can be optimized. In terms of fabrication optimization, packing problems are well studied for material saving. More recently, Koo et al. [2017] investigated this problem and proposed a guided tool for furniture design. The novelty in our optimizer is that it considers the full fabrication processes, which involves not only material usage but also type and order of operations. This is enabled by treating fabrication as a program and defining a multi-objective optimization solution on top of a data structure (e-graphs) that can represent all equivalent programs.

3 OVERVIEW

We now consider the typical process of designing and fabricating a simple wooden part. First, designers consider available materials and fabrication processes and use this information to guide the first draft of their design. The design, typically modeled in a *parametric* CAD system, is then used to iteratively explore possible variations. The designer uses feedback provided by the CAD tool as well as potential simulation plug-ins to iterate on the design. Once satisfied with the resulting configuration, a specific way to fabricate the part must be identified. For example, the fabricator chooses the stock to use for each part, the cutting tools to maximize precision, the order of cuts to minimize the number of setups, or when and how

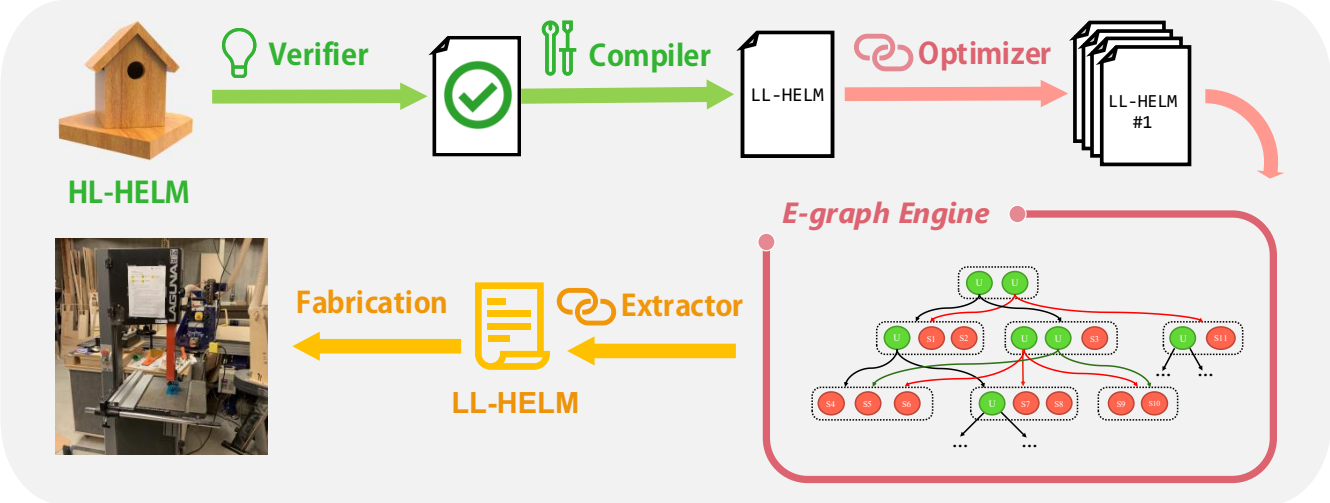


Fig. 3. System pipeline. The input to our system is a HL-HELM program designed by a user in our IDE. The verifier first checks if the design is manufacturable. The compiler converts the verified HL-HELM program to a LL-HELM program. Then the various optimizers populate an e-graph by finding various equivalent optimal programs. Finally, the extractor performs a multi-objective optimization to find the most optimal programs from the e-graph.

to stack parts to minimize the number of cuts. In a workshop, the designer and fabricator may be the same person; in a corporate setting, they may be different teams in different companies or in different countries.

The above description of the typical pipeline has two important yet conflicting takeaways in the design space and the fabrication space. First, decoupling design and fabrication could advance computational tools that assist each process. On the other hand, it is essential to take fabrication into account during design since it defines the space of what can be physically realized. Mapping free-form designs to a fabrication plan will likely lead to approximations that affect performance or impose unjustifiably high fabrication costs.

3.1 Design Philosophy

Our proposed architecture accounts for both seemingly conflicting ideas noted above. We aim to ensure that *design is driven by available fabrication options*. On the other hand, we seek to provide abstractions, similar to ISAs, that decouple design and fabrication. This lets us propose advanced algorithms to optimize designs that are fabrication independent and to optimize fabrication that is hardware dependent. We now discuss how we take these considerations into account in our proposed system, and conclude with a list of features we incorporate into the proposed pipeline. The remainder of the paper details how we implement these features.

Fabrication-Oriented Design. HL-HELM is inspired by feature-based CAD languages, which define a sequence of geometric operations that construct the shape bottom-up. We leverage this modeling technique because it defines geometry as programs and has been proven effective. The key difference is that we define *fabrication-aware* features. Because carpentry is a subtractive process, we define features that perform subtractive operations on stock instead of performing standard CSG operations. This ensures that the resulting programs can be effectively verified and compiled to LL-HELM while preserving the process-independent constructive geometry modeling paradigm that designers are accustomed to using.

Fabrication-Independent Design Optimization. While it is important to ensure that design is driven by manufacturing realities, we also seek abstractions that allow design optimization without the need to compute low-level fabrication details. Two methods can be used to search the design space: interactive exploration and automatic optimization.

For *interactive exploration*, it is important to efficiently verify that a program in HL-HELM is feasible. One option would be to define a dense language, which ensures that all programs in the language are valid, i.e., map to a valid fabrication plan. However, languages constrained to be dense are typically less expressive and intuitive. We prioritize the latter attributes and develop a verifier that can validate a HL-HELM program in real-time. Additionally, we implement an IDE (Integrated Development Environment) for HL-HELM inspired by modern CAD systems that lets users intuitively explore the design space while ensuring the validity of every design. Using this IDE, user interactions with a 3D model are automatically mapped to HL-HELM code, and constraints on valid programs are translated to constraints on user interactions, guiding users in defining valid programs.

In addition to the interactive exploration of the feasible design space provided by the verifier/IDE, our design system uses parametrization as a basis for *automated optimization*. As in standard CAD tools, our feature-based system is *parametric* from construction, defining a search space for physically-based optimization, as is done in the previous work [Schulz et al. 2017].

n::= *Int* | *Float* pt$_2$::= *n n* pt$_3$::= *n n n* catalog_id::= *UID string*

```
geom::= Point pt2
  | Line pt2 pt2
  | Circle pt2 n
  | Spline pt2*
  | ...

constraint::= Parallel geom geom
  | Concentric geom geom
  | ...

query::= Query_Face_By_Closest_Point n n n
  | Query_Vertex_By_Closest_Point
  | ...

sketch::= Make_Sketch query geom* constraint*

design_op::= Make_Stock n n n
  | Make_Cut id sketch
  | Make_Hole id sketch
  | ...

hlhelm::= (Assign id* design_op)*;
```

```
face::= uid

edge::= uid

setup_op::= Setup_Chopsaw angle angle offset
  | Setup_Drill diameter
  | ...

ref_pt::= edge offset edge offset

fab_op::= Lumber catalog_id
  | Sheet catalog_id
  | Stack id id
  | Unstack id
  | Chopsaw id face edge
  | Bandsaw id face ref_pt*
  | Jigsaw id face ref_pt*
  | Drill id face ref_pt*
  | ...

llhelm::= (setup_op | Assign id* fab_op)* Return id*
```

Fig. 4. Syntax of HL-HLEM (left) and LL-HELM (right).

Hardware-Dependent Fabrication Optimization. The system's backend must define a complete list of instructions that can be directly used for fabrication. Therefore, it must define CAM operations for specific types of tools, each with pre-defined capabilities and requirements on the workpieces they can accommodate. It is essential to provide an *extensible* architecture because it would be impossible to define a language that explicitly represents all existing (and emerging) fabrication processes. Extensibility is achieved by establishing a list of verifier rules for each LL-HELM operation and a surjective mapping from HL-HELM to LL-HELM operations. This lets us establish clear guidelines for incorporating new processes into the language.

Finally, our system must automatically generate fabrication plans for a given design. We therefore design a compiler that can generate LL-HELM code from HL-HELM. The compiler has an optimizer that can handle multiple and conflicting costs, for example, fabrication time, material cost, and accuracy.

In summary, we propose an architecture with the following properties:

- HL-HELM represents subtractive feature-based modeling.
- HL-HELM validity is supported by a verifier and IDE.
- HL-HELM is parametric.
- LL-HELM and the verifier reference the available hardware.
- The full stack is easily extensible to new hardware.
- The compiler performs multi-objective optimization.

3.2 Design Processes

We complete our overview by describing the system from its users' point of view. Before starting their design, users should import libraries of materials and tools so the compiler has feasible instructions for mapping the designs to specific hardware (Figure 2(a)). Users then create designs with an intuitive interface that adheres to the same process as standard parametric feature-based CAD systems; in fact, our tool is built as a plug-in for FreeCAD [2019]. The key difference is that the allowed features map to subtractive operations that correspond to carpentry operations: get stock, make poly-cuts, and make holes (Figure 2(b) and supplemental video). The manufacturability of the designs is checked by the verifier, and users are notified if any features are invalid. As in standard CAD tools, parametric modeling lets users iterate on their designs while satisfying constraints (Figure 2(c)). Once designs are finalized, an optimizing compiler generates a set of LL-HELM programs with different trade-offs from which users can choose (Figure 2(d)).

4 LANGUAGE AND COMPILER

This section describes the design language, compiler, and fabrication instruction language that were designed based on the considerations and requirements discussed in Section 3. We highlight key language features and include details in the supplemental material.

High-level HELM. Figure 4 (left) shows the grammar for HL-HELM programs. These programs consist of a sequence of assignments that bind *design_op*s to identifiers. *design_op*s are high-level fabrication operations which depend on a set of parameters. The proposed language is inspired by standard feature-based CAD scripting languages [FeatureScript 2019], where features map to fabrication operations (e.g., get stock and make cut) as opposed to purely geometric operations (e.g., extrude and loft). As in CAD languages, 2D sketches are used to specify the path of operations and are defined by

a set of 2D parametric primitives and constraints. Computationally, a HL-HELM program can be evaluated with an interpreter that runs each assignment in sequence. To run an assignment, the interpreter evaluates the operation to a B-rep (Boundary Representation) using a geometric kernel (OpenCASCADE [SAS 2019]) in the context of bindings resulting from previous assignments, referenced using identifiers.

We also draw ideas from CAD referencing schemes [Baba-Ali et al. 2009; Bidarra et al. 2005], using *queries* to reference part of the geometry (e.g., an edge or face) on top of which operations can be defined. This approach allows consistent referencing of parts of the model that, coupled with a direct specification of constraints, allow models to be consistently regenerated after parameter updates. As in modern parametric modeling systems, it lets us define and constrain the ways a model can vary, defining a parameter space that can be used for design optimization [Schulz et al. 2017]. Note that programmers do not need to manually write out complex query parameters since an IDE automatically creates queries when users select a part—i.e., click on a part with the mouse.

Low-level HELM. Figure 4 (right) shows the syntax of LL-HELM. A LL-HELM program is a sequence of either *setup_op*s or assignments that bind *fab_op*s to identifiers. *fab_op*s are fabrication operations that explicitly reference available hardware and material. These operations include taking a piece of lumber from a material catalog, performing cuts with different tools, and stacking, i.e., placing parts together to allow operations to be applied simultaneously to improve fabrication efficiency. Some fabrication operations require a setup that configures the tool to perform the task, e.g., setting the angles of a chopsaw. A LL-HELM program is concluded by a *Return* statement which returns the resulting parts obtained from fabrication operations.

Unlike HL-HELM, LL-HELM lacks the concept of queries because it is intentionally non-parametric: the compiler finds the optimal fabrication plan for concrete design, as specified by an instance of the parameters. References must be defined to allow the accurate positioning of parts with respect to the cut blade. In LL-HELM, reference points for the cut operation are defined by the intersection of two lines, where each line is specified by an offset from an edge on the part.

Process Characterization. To generate LL-HELM code, the compiler must understand the capabilities and constraints on each fabrication process, e.g., the maximum depth of a stock that can be set up on a chopsaw. It must also be able to measure the performance of each process in order to optimize fabrication time and accuracy. As part of our architecture, HELM retains for each process the set of constraints and performance measurements in the form of *process characterization*. The process characterization enables the compiler to measure feasibility, fabrication time, and accuracy for a given fab_op in LL-HELM. For example, it uses process characterization to determine that the accuracy of a chopsaw is higher than a bandsaw. The entire process characterization is included in the supplemental material.

Compiler. The compiler from HL-HELM to LL-HELM maps abstract, high-level fabrication operations to concrete, process-specific operations. The first step of this process is to ensure that a valid mapping exists since it is possible to generate HL-HELM programs that do not correspond to feasible instructions. We defined languages so that every assignment in HL-HELM can be mapped to one or more sequences of assignments and setups in LL-HELM. For example, Make_Cut maps to Setup_Chopsaw followed by Chopsaw, and also to Bandsaw, while Make_Hole maps to Setup_Drill followed by Drill. Our verifier sequentially attempts to map each assignment of a HL-HELM program to the possible LL-HELM programs it can be mapped to. It is essentially a simulator to evaluate context using 1) the same geometry kernel used in the front-end, and 2) the process characterization to measure feasibility. If this process can be executed to completion, the HL-HELM program is valid. This can be done interactively and used to provide design feedback in the IDE. If the available hardware changes, this process can automatically verify feasibility and map HL-HELM code to the newly available resources.

Once we have a valid LL-HELM program, the compiler considers different ways it can be re-written to optimize the fabrication process. This multi-stage process is discussed in Section 5.

Extensibility. The surjective mapping from HL-HELM to LL-HELM along with the process characterization allows this architecture to be easily extensible to new fabrication processes. Adding a new process involves three steps: 1) adding a new LL-HELM operation and (possibly) setup for the new process, 2) defining the process characterization, 3) defining a mapping from HL-HELM to this process. The third step can be done either by assigning a mapping from an existing design operation or defining a new one. We demonstrate this using the following two examples. Consider adding a tablesaw process. This requires defining the operation and its corresponding setup in LL-HELM as follows:

Tablesaw id face edge, *Setup_Tablesaw angle offset*,

adding the process characterization for this tool to define the validity constraints and cost functions, and extending the *Make_Cut* operation in HL-HELM to also map to the tablesaw process. Consider another example where a drill press with an arbitrary hole depth is added to the language: *DrillPress id face ref_pt depth*. In this case, the change in HL-HELM involves adding a new operation to allow partial holes: *Make_Partial_Hole id sketch n*, where *n* is the depth.

5 FABRICATION OPTIMIZATION

This section details how our compiler optimizes low-level fabrication plans to provide diverse Pareto-optimal candidates trading off between material cost, fabrication time, and precision.

5.1 E-graphs Background

E-graphs propagate equality information through a common graph representation that stores multiple equivalent versions of the original program. E-graphs are composed of *equivalence classes (e-classes)* each of which contains a set of *equivalent nodes (e-nodes)* that represent some operation applied to argument *e-classes*. This is encoded as edges from e-nodes to e-classes. An advantage of e-graph-based optimizers over sequential rewrite engines is that they avoid the

phase ordering problem. Phase ordering occurs when optimizations are applied destructively in sequential order, thereby causing the quality of the resulting code to depend on the order of application of the optimizations [Bansal and Aiken 2006; Whitfield and Soffa 1990, 1997]. E-graph-based optimizers avoid this problem by retaining previous versions of expressions even after transformations are applied. These semantically equivalent expressions are stored in the same e-class. An e-graph is populated by repeatedly applying optimizations. Finally, a cost function is used to extract optimized expressions from each e-class, which are then composed to return the best program from the e-graph.

Using e-graphs in fabrication requires addressing three technical challenges. First, e-graphs were originally developed for automated theorem proving in structural logics [Nelson 1980], where there are no *linearity* constraints on variable reuses. However, our e-graph engine needs to account for linearity in fabrication. For example, after a piece of lumber L is cut into two pieces, the fabrication plan should no longer refer to L since it no longer exists. Second, the fabrication domain requires new *conditional* rewrites, for example, encoding conditions under which cuts can be stacked. Third, since fabrication includes many different and often conflicting objectives, our system needs to generate several candidates (i.e., Pareto-front candidates) based on user-defined multi-objective cost functions, rather than extracting a single solution from the e-graph bottom-up.

In HELM, we focus on *stock* and *union* e-nodes. Stock e-nodes represent a series of subtractive operations all applied on a single piece of stock, capturing both part layout and per-cut fabrication process selection. Union e-nodes point to a set of child e-classes, each of which contains stock e-nodes or (recursively) more union e-nodes. Each union e-node thus represents all the fabrication plans that can be built by selecting representatives from its children. This partitioning into stock and union e-nodes enables encoding plans that reuse *scrap*, or "offcuts". Each e-node in the root e-class represents a set of fully concrete fabrication plans, corresponding either to a particular concrete set of layout and process choices (in the case of a stock e-node) or all the recursive combinations of plans from child e-classes (in the case of a union e-node). In our e-graph, equivalence is defined as producing identical output, and so all programs that generate the same result will be represented in the same e-class.

Before describing our method for populating an e-graph, we first illustrate the e-graph for a simple design that outputs three 20″ long 2x4 parts, when two types of stock are available in the library: one is 24″ long and the other is 96″ long (Figure 7). A sequence of cuts performed on a single stock is represented as a stock e-node, e.g., cutting parts 1 and 2 on a 24″ stock. Since the union of cutting part 2 on a 24″ stock and part 3 on a 24″ stock is equivalent to cutting these two parts on a 96" inch stock, these candidates are represented in the same e-class. Even in this simple example, many different programs can be extracted from the e-graph as shown in the figure: all parts on a 96″ stock, each part on a 24″ stock, or one part (part 1) on a 24″ stock and the other two on a 96″ stock – and within each of these layout strategies many different fabrication process selections are possible.

Below we describe how we use geometric solvers to construct a set of e-classes and e-nodes in the e-graph, supporting both linearity constraints and conditional rewrites (Sec. 5.2) and detail a new method of e-graph extraction which supports the multi-objective optimization requirements for fabrication (Sec. 5.3).

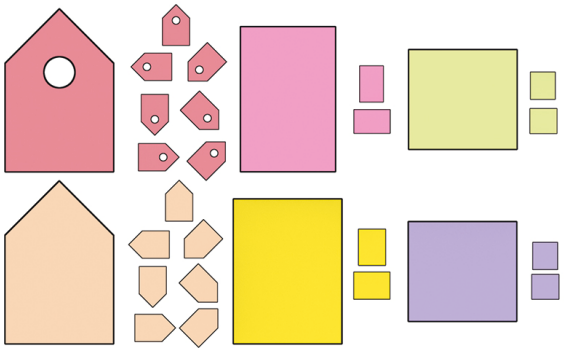


Fig. 5. 2D shapes for birdhouse and their different orientations.

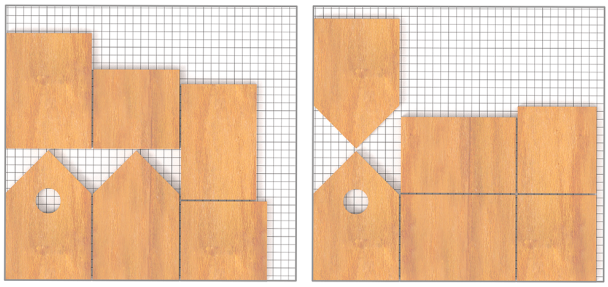


Fig. 6. Comparison between traditional packing result (left) for minimizing bounding volume and our packing result (right) for maximizing the number of shared edges.

5.2 E-Graph Construction

In general, defining a fabrication plan for making parts in a carpentry project involves the following two major steps: 1) laying out parts on stock lumber and 2) choosing appropriate cutting tools and the order to apply them. Constructing an e-graph that covers many of the possible manufacturing plans is challenging because there are many different ways to assign material, order cuts, and combine multiple tools, *each* of which results in a combinatorial explosion. Combining all of them makes the space of programs even larger. To make the space of programs tractable, we define several pruning strategies that eliminate programs that correspond to unrealistic scenarios and keep only those that are feasible in practical carpentry.

Packing Pieces onto Stock. At the first stage of our optimization pipeline, the parts designed by users are assigned to stock lumbers, where orientation and degree-of-freedom are also decided. We provide a common library of stock lumber that can be readily purchased at home improvement stores; the library can easily be extended with other customized stock. Our system takes as input the bounding boxes of the parts and compares their dimensions with the sizes of the available stock to evaluate the feasibility of an assignment. In our prototype implementation, we use a library that consists of

commonly used materials, and our packing algorithm generates candidates for all stock lumber pieces on which the parts fit. Since many cuts are straight and most parts are polygons, it is possible to minimize the number of cuts by aligning multiple parts so that a single cut can be applied to more than one part (example shown in Figure 6). As a result, unlike conventional packing problems which primarily minimize the bounding volume [Burke et al. 2006; Hopper and Turton 2001], ours also minimizes the number of cuts.

Unfortunately, packing problems have been shown to be NP-hard, and it is infeasible to explore the space of all possible packing strategies due to combinatorial explosion. In our work, we observe that parts designed in carpentry are usually not arbitrary so we target the packing problem by proposing a simple-yet-efficient algorithm in the cases of 1-DOF (1D packing) and 2-DOFs (2D packing).

Given a set of shapes, e.g., the shapes for the birdhouse shown in Figure 5, the goal is to pack them on to a sheet for cutting to maximize the number of aligned edges. To start packing, the algorithm randomly picks an oriented shape and places it on the initial rectangular sheet. This changes the shape of the remaining sheet, as shown in Figure 6. To pack the next shape in the remaining sheet, the algorithm picks two edges of the sheet, two edges of the shape, and solves a linear set of constraints to check if the pairs of edges can be aligned. If they cannot be aligned, it continues to pick a different pair of edges from the sheet and the shape. If there is no solution for any pair of edges, the algorithm randomly picks one edge from the sheet and the shape and aligns them to minimize its volume, as is done in standard cutting and packing algorithms [Burke et al. 2006; Hopper and Turton 2001]. Our packing algorithm takes into account the dimension of the "kerf", i.e., the parts are separated from each other by the width of the saw blade.

This process is repeated for all the shapes in the design to obtain a candidate packing on a stock, and further, our packing algorithm is repeated for all stock pieces in the library. Our method organizes all of the packed results as stock e-nodes, and constructs e-graphs simultaneously. Since our tool generates many packing strategies for every design, we use a heuristic to prune some of the results. Packing solutions with more aligned edges are better since they require fewer cuts. Hence, we sort all solutions by the number of aligned edges and keep the top n results.

Defining Cuts on Stock. Once we have arranged a set of parts on a piece of stock, we must select the fabrication process for each cut and the order of cuts. Moreover, process-specific setups and references need to be identified under tool constraints and workpiece constraints. For instance, the process characterization of our chopsaw specifies that the maximum thickness of a workpiece is $4''$. These constraints must be considered when selecting tools for each operation. A cut may not be mappable to a fabrication instruction due to violation of constraints. On the other hand, some cuts can be mapped to multiple feasible processes. Further, a cut can be either across the whole workpiece or only to a certain position. We call the latter *partial* cuts. Our system automatically takes all these cases into account to generate a large family of equivalent fabrication plans, which essentially creates more e-nodes for the e-classes that pointed to from stock e-nodes, and populates the e-graphs. We propose two heuristic pruning strategies.

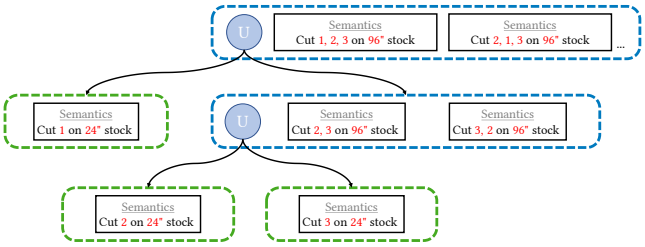


Fig. 7. An example of an e-graph for a simple design that outputs three 20" long two-by-four parts when the stock library has 96" and 24" stock lumber.

(1) Some measurements are easier to accurately take than others. For example, in the imperial system, distances are more precise and easier to measure if they are integer factors of $[1'', 1/2'', 1/16'']$ as those corresponding to the demarcations of common measuring tapes, and angles when they are integer factors of $[45°, 15°, 1°]$. If our system finds multiple setups for the same process, it keeps only those that lead to the best measurements and discards others; if any setup involves one of the above measurements then all others are discarded; otherwise, the ones with values closest to some entry in the above lists are kept.
(2) Our compiler prefers complete cuts over partial cuts when possible because partial cuts are difficult to perform (refer to process characterizations in supplementary material), and they tend to be imprecise. For any instruction, if both partial and full cuts are possible, our compiler prunes away the partial cut candidates, and keeps only the full cut candidates. However, if no full-cut solutions can be found, the compiler will use a partial cut solution.

5.3 E-Graph Extraction

Even with the pruning strategies described in the previous section, the e-graph can have up to $O(N \times K)$ e-nodes, where N is the number of e-classes and K is the number of sub-programs in each e-class. The total number of programs that can be generated by combinations of all these sub-programs grows exponentially (i.e., $O(2^{N \times K})$), so it is important to have efficient ways of exploring this space for extraction.

Objectives. Our system produces a set of optimized fabrication plans with respect to the following three objectives.

- *Cost f_c:* Every assignment statement that uses stock, i.e., lumber or sheet is assigned a cost depending on the type of the stock (plywood, two-by-four, two-by-three, etc.).
- *Precision f_p:* The process characterization of a tool provides a precision value. For example, when making straight cuts, a chopsaw is more precise than a bandsaw, which is more precise than a jigsaw. We define f_p to be the product of a tool's precision and the error introduced while making a cut. A standard tape measure or ruler provides divisions of an inch in increments of one-sixteenth. Therefore, in our implementation, a measurement has zero error if it is a multiple of

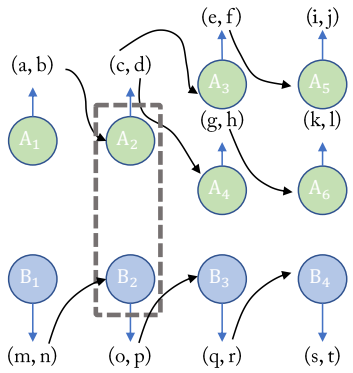


Fig. 8. Vertex-collapsing algorithm to measure time. Each sub-program is a dependency graph and can be represented by a directed graph. Assume two non-adjacent vertices A_2 and B_2 have the same setups, they can be stacked and cut at the same time if there is no cycle after collapsing.

one-sixteenth (or better, a whole number). For other measurements, the error is the absolute value of its difference with the closet marking on the tape measure. Due to the modular design of our tool, it is also possible to plug in alternate implementations for error.

- *Time f_t:* Different fabrication processes require different manufacturing time. For example, it is easier to perform a chopsaw cut than a tracksaw cut in practice. Moreover, a program that requires users to change the setup for every cut is worse than a program that reorders and makes multiple cuts with the same setups. Our time metric f_t considers the minimization of configuration switching on a single tool.

We detail the computation of all the metrics in the supplemental material. In summary, given a set of sub-programs constituting a complete fabrication plan, our system needs to compute all objectives efficiently. f_c is computed by summing up the costs of a program's stock input. f_p is defined as the average value of the sum of all errors (the deviation from the lowest scale) scaled by the precision weight of the tool being used. Both of these objectives are *modular*, i.e., the best representative node for an e-class can be computed directly from its children, and thus are straightforward to compute. However, to compute f_t we must measure the benefits of sharing setups across different operations within a program. This requires analyzing and optimizing fully concrete fabrication plans, since considering operations in isolation does not capture the global sharing benefits of a particular ordering of operations. Greedy algorithms are also insufficient for extracting efficient schedules from our e-graph due to linearity: not every pair of statements can be re-grouped. Thus to measure f_t in HELM, we perform another optimization step that schedules cuts efficiently using a vertex-collapsing-based optimization.

Graph algorithm for measuring time. Time can be minimized in two ways: (1) By setup elimination: when two instructions in a program use the same setup, they can be reordered so that a single setup can be used for both instructions, as long as linearity is not violated. (2) By stacking parts together: when two instructions use the same setup *and* also use two separate pieces of lumber that do not depend on each other, the pieces can be stacked together. We developed a new graph algorithm to minimize manufacturing time using the above techniques.

For a set of sub-programs, our algorithm builds a dependency graph $\mathcal{G}$ by looking at the input arguments and returned values of each statement. $\mathcal{G}$ is a directed graph and may have multiple connected components, which correspond to different programs. $\mathcal{G}$ corresponds to a valid program only if it has no cycles because a cycle indicates a violation of linearity. If two nodes (a, b) have the same setups, our algorithm *collapses* them and checks that no cycle is introduced in the graph (Figure 8). Interestingly, there are two cases that may arise after nodes are collapsed: if (a, b) are adjacent (i.e., there is a directed edge either from a to b or from b to a), collapsing will result in setup elimination; if (a, b) are not adjacent, these two statements can be performed at the same time (for example, stacking them on a chopsaw or parallelizing cross multiple workers). Our optimizer uses this graph algorithm to find a program that minimizes time by minimizing the number of setups while respecting *linearity*. This also reorders the sequence of instructions in a program to put two instructions which partially share setups close to each other.

Multi-Objective Optimization. For many simplification tasks [Panchekha et al. 2015], greedy approaches can be used to extract a program from an e-graph where it is traversed bottom-up and the best e-node from each e-class is chosen. This approach does not work for optimizing manufacturing time since it is not an additive metric. Previous work on e-graphs have used constraint solvers [Joshi et al. 2002] to extract programs non-greedily, but those approaches are expensive when extracting multiple programs. Further, the three objectives we defined for e-graph extraction may be conflicting. For example, using more stock would allow simultaneous scheduling of cuts that use the same setups but can increase the cost of lumber. In our prototype, we address these problems by using a genetic algorithm for multi-objective optimization.

In multi-objective optimization, genetic algorithms are one of the most common approaches, having been successfully adopted in many fields [Zhang and Xing 2017]. We use the NSGA-II [Deb et al. 2002] method in our implementation. The NSGA-II algorithm can improve the fitness of a set of candidate solutions to a Pareto front bounded by a multi-objective function. It is an evolutionary process that has selection, mutation, and crossover. The population is classified into a hierarchy of subgroups by diversity metrics for selection. Our system encodes each individual as a tree $\mathcal{T}_i$ which is a subset of the e-graph. A full program can be recovered by traversing $\mathcal{T}_i$ from top to bottom. Since we adopt the tree representation, it is difficult to directly use off-the-shelf methods of crossover and mutation. To solve this problem, we define new mutation and crossover operations based on equivalence relations encoded in the e-graph.

In mutation, our algorithm traverses all e-classes in $\mathcal{T}_i$ and mutates their e-nodes if $rng < p_m$, where rng is the random number generator that uniformly produces a probability in $[0, 1]$ and p_m is the probability of mutation. Our algorithm can also mutate an e-node to represent it by its argument e-classes. It therefore also

randomly expands the e-classes recursively since the leaf nodes of $\mathcal{T}_i$ must only be sub-programs.

In a crossover, the algorithm first randomly selects a pair of individuals $(\mathcal{T}_i, \mathcal{T}_j)$ where $\mathcal{T}_i$ and $\mathcal{T}_j$ have edges to the same e-classes. It switches each pair of same e-classes by exchanging their e-nodes if $rng < p_c$ where p_c is the probability of crossover.

6 RESULTS AND DISCUSSION

In this section, we evaluate our system against the following criteria: we discuss the expressiveness of HL-HELM, we evaluate the quality of our compiler-generated fabrication instructions, we demonstrate how our tool can be used for end-to-end optimizations, and we show how designs can be physically realized by users following LL-HELM fabrication plans. We developed our system in C++ and tested it on a PC with Intel E5 2620 and 64GB RAM.

6.1 Expressiveness of HL-Helm

Figure 9 demonstrates HELM's expressiveness by showing examples of a wide range of valid designs made by three experienced woodworkers (with more than three years of experience) using HL-HELM who were trained to use our IDE. The three carpentry experts generated the models in Figure 9 by using an iterative process, with their time split between conceptual exploration and design. These same experts created the physical models shown in the teaser and filled out a survey relating their experience with the tool and comparing it to conventional CAD systems.

Based on feedback from the woodworkers, we conclude that while it is easier to produce arbitrary models in standard CAD systems, for carpentry items HELM was faster and more intuitive. This is because HELM allows the designer to keep the fabrication process in mind during the design process. We report comments from the experts in the supplemental material.

6.2 Optimized Fabrication Instructions

We tested our optimizing compiler on all of the designs shown in Figure 9, apart from 9.E because it is too simple. Our results show that the compiler successfully optimizes all the designs sketched by the experts. To evaluate the quality of the optimized results, we asked four woodworking experts to come up with fabrication instructions by hand and then computed the cost of their designs. Comparative results are shown in Figure 11, and additional results are reported in the supplemental material, along with implementation details and optimization parameters.

In eight out of the nine experiments, the system found solutions that Pareto-dominate the expert fabrication plan. This result validates that the proposed approach is not only a method that can help users with little expertise to find efficient fabrication plans but can also discover solutions that behave better than the ones designed by experienced woodworkers. We believe this is due to the high-dimensionality of the search space and the need to simultaneously consider multiple conflicting objectives, which makes manual exploration challenging. The added benefit of our approach, which is also shown in Figure 11, is that it returns not one but multiple solutions with different trade-offs, allowing engineers to pick the one that is more suitable for a specific application.

There was one model in which our system did not find a solution that Pareto-dominates the expert. It is interesting to note that the expert solution also does not dominate the solutions of the system, but instead indicates a different trade-off that was not found by our method. This result indicates that while our method can find good solutions that outperform or match the experts, there are no guarantees that the solutions found are truly Pareto-optimal or that the full Pareto-front is found. Cut planning is a combinatorial problem and, while the use of e-graphs and our pruning strategy make the problem tractable, it is possible that (1) optimal designs are pruned and (2) the genetic algorithm does not discover the full front. To further evaluate our method, we vary the amount of pruning and plot graphs that show how the performance and computation time varies. The comparative results, reported in the supplemental material, show that time increases linearly, but the performance quickly increases and then tapers off, which validates the pruning method.

To further illustrate the different solutions and trade-offs, we show fabrication plans in Figure 12. In the bookcase, example 9.C, the expert grouped similar cuts on individual pieces of stock and cut them in order from left to right leading to high accuracy at the expense of material cost and time. Solution (A) Pareto-dominates the expert. In this example, HELM was able to significantly reduce the amount of material by using an optimal packing strategy while slightly reducing fabrication time and maintaining the same accuracy. In solution (B), HELM was able to significantly reduce fabrication time at the expense of higher precision error and material cost.

For the flower pot, example 9.I, the differences between the optimized results generated by HELM compared to the expert boils down to sheet packing. HELM made the same tool selection and material choices as the expert. HELM, however, was able to reorder and rearrange cuts to improve fabrication time and accuracy. On the other hand, the expert optimized the utility of unused stock, which was not accounted for in our cost functions.

6.3 Design Optimization

Figure 10 shows how the parametric nature of HL-HELM is useful for design optimization driven by high-level, fabrication-independent performance metrics. In this example, four design parameters affect the geometry of the bookcase expressed in HL-HELM and the performance metric we want to optimize is stability. The figure illustrates different configurations that can be achieved with different parameters and can be measured by the distance of the projection of the center of mass to the convex hull of the contact points. This allows us to optimize designs, which are independent of fabrication, while ensuring the designs before optimization and after optimization are manufacturable using carpentry processes.

6.4 Physical Realization from LL-Helm

To evaluate the practicality of our optimizing compiler and languages, we provided the three experts with the LL-HELM code for three models (bookcase, birdhouse, and toy car) which are shown in Figure 1.We created a user interface (UI) to link the names of variables in LL-HELM programs with correct geometric details, and enable users to interactively visualize them in 3D space. They

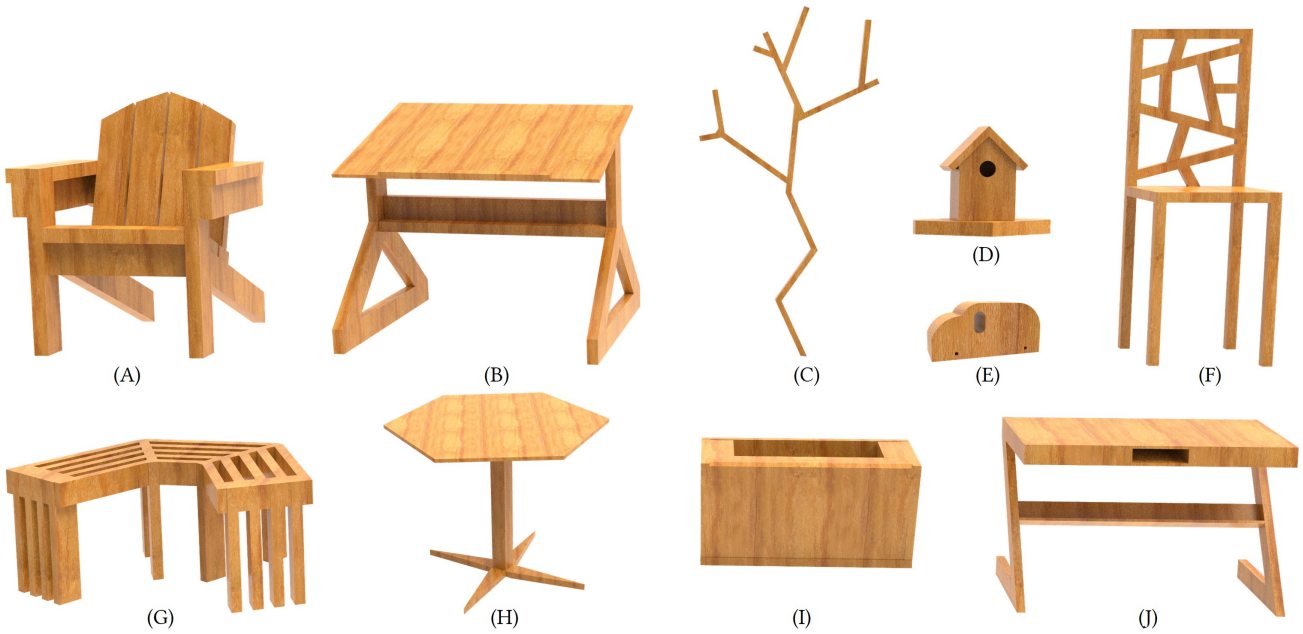


Fig. 9. A gallery of carpentry designs modeled in our proposed system. The design time for each model is as follows. (A) Adirondack chair: 3:30 hr; (B) Drafting table: 2:16 hr; (C) Book case: 1:00 hr; D) Bird house: 1:38 hr; E) Toy car: 0:45 hr; F) Dining room chair: 1:20 hr; (G) Bench: 2:02 hr; (H) Coffee table: 0:56 hr; (I) Flower pot: 2:00 hr; (J) Z-table: 1:34 hr.

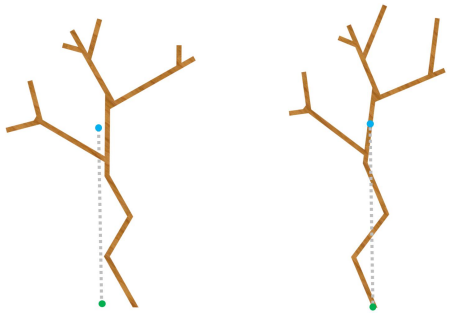


Fig. 10. Example of design optimization that is enabled by the parametric nature of HL-HELM. The different shapes and corresponding design parameters and performance value (stability) are shown and the optimal one is highlighted (right).

successfully manufactured all of these designs by following the LL-HELM code step-by-step. We also have a video to show the fabrication processes.

6.5 Limitations and Future Work

Developing programming languages techniques for carpentry is a new direction and our work demonstrates the feasibility of this research avenue. However, there are still some limitations that require further investigation.

First, our prototype does not support shapes involving free-form geometry. Even though these designs can be manufactured using subtractive techniques, additive techniques are usually preferred. Since we currently do not support additive methods, such designs would require special treatment. Second, our compiler optimizations are not complete because we do not capture all possible equivalences. As a consequence, the compiler cannot perform optimizations that involve inserting additional cuts, or other temporary operations which may sometimes be useful. Third, our compiler first populates the e-graph with valid programs and then prunes it using heuristics to make the search more tractable. While efficient, this may not always return the optimal fabrication plan. However, as our results show, the instructions generated automatically by ours can already match or improve upon plans manually developed by human experts. Fourth, our compiler currently uses a fixed-sized kerf for cuts which may make the dimensions of cuts inaccurate. Further, our work uses three simple metrics that were developed with the help of expert carpenters to evaluate fabrication plans. While these metrics can effectively demonstrate the capabilities of our multi-objective optimization pipeline, it would be interesting to investigate richer cost models, for example, to take into account stackability, correlated errors, and grain-orientation into the precision metric. Finally, while our compiler can optimize for precision, fabrication uncertainty can still affect the outcome. Figure 1 (toy car) shows an example of fabrication error due to which the shape of the car's window is different from the original design. Accounting for fabrication error during design is a hard problem even for single-process manufacturing because it depends on available processes. Typically this is handled by having designers predetermine error tolerances which are later verified. Decoupling design from fabrication can let us minimize

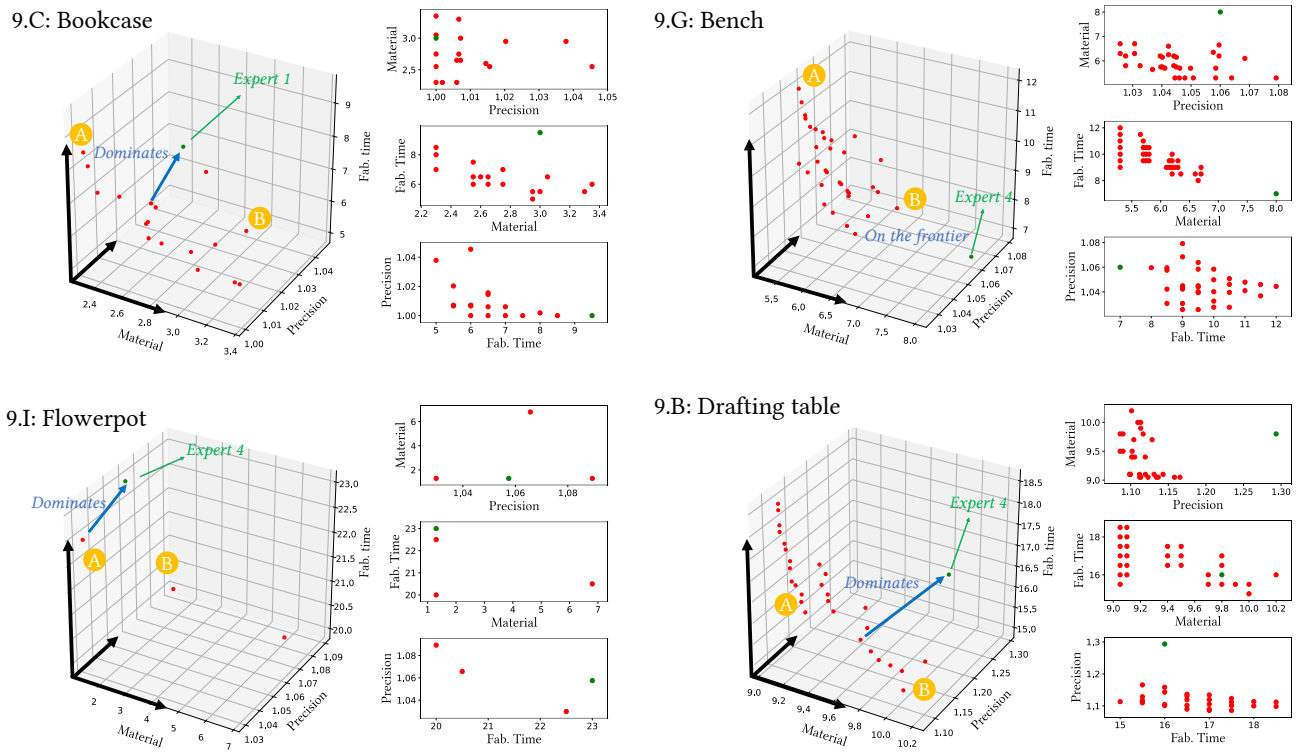


Fig. 11. Results of the Pareto-fronts discovered by our system (red) as compared to fabrication instructions hand-written by experts (green). For each example, we highlight a point in our discovered front that Pareto-dominates the expert fabrication plan. In addition to the 3D plots, we show 2D projections on the three main axis for better visualization of the different trade-offs.

error, but it still does not let us take the error into account at the design stage, for example, while performing finite element analysis.

7 CONCLUSION

This paper presents HELM, a system for making high-level, abstract designs and automatically translating them to low-level, optimized fabrication plans. Our key insight is that *fabrication plans are programs*. Based on this insight we developed new domain-specific languages for high-level (HL-HELM) designs and low-level (LL-HELM) plans, applied and extended compiler techniques to support multi-objective optimization, and demonstrated how these components simultaneously enable fabrication-aware design and optimization while shielding designers from fabrication details. Our compiler from HL-HELM to LL-HELM automatically verifies manufacturability and provides novel optimizations to improve precision, and reduce material cost and manufacturing time. In order to efficiently represent all programs obtained by various optimizations that correspond to a particular fabrication plan, we leverage an e-graph data structure from traditional programming languages and compilers. We demonstrate how to extract Pareto-optimal programs from the e-graph by performing multi-objective optimization.

Our approach opens many exciting avenues for future work. Our HELM prototype provides a solid foundation for exploring interactions between subtractive processes, e.g., carpentry or machining, and additive processes, e.g., 3D printing or welding. Such interactions will enable even more flexibility in generating and optimizing low-level manufacturing plans and further empower designers to take full advantage of the ever-increasing diversity of available fabrication processes. It would also be interesting to exploit HL-HELM to create designs with for-loops, which can be directly unrolled in a pre-processing step, and investigate solutions for supporting recursions. Combining subtractive and additive processes will also enable *error recovery* when a user makes a mistake: for example, if a cut is made too short, a low-level "program patch" could be generated automatically using program synthesis techniques [Gulwani et al. 2017] to build the botched part back up and enable resuming execution of the original plan rather than starting over from scratch. Cross-process fabrication plans could also be automatically scheduled for tighter integration between available processes, e.g., using a robotic arm to embed magnets in a part as it is 3D printed or using available processes to construct jigs that make otherwise-infeasible operations possible. Looking further ahead, as more robotic fabrication processes become available, exploring the potential to automatically schedule and optimize human-robot interaction in the fabrication

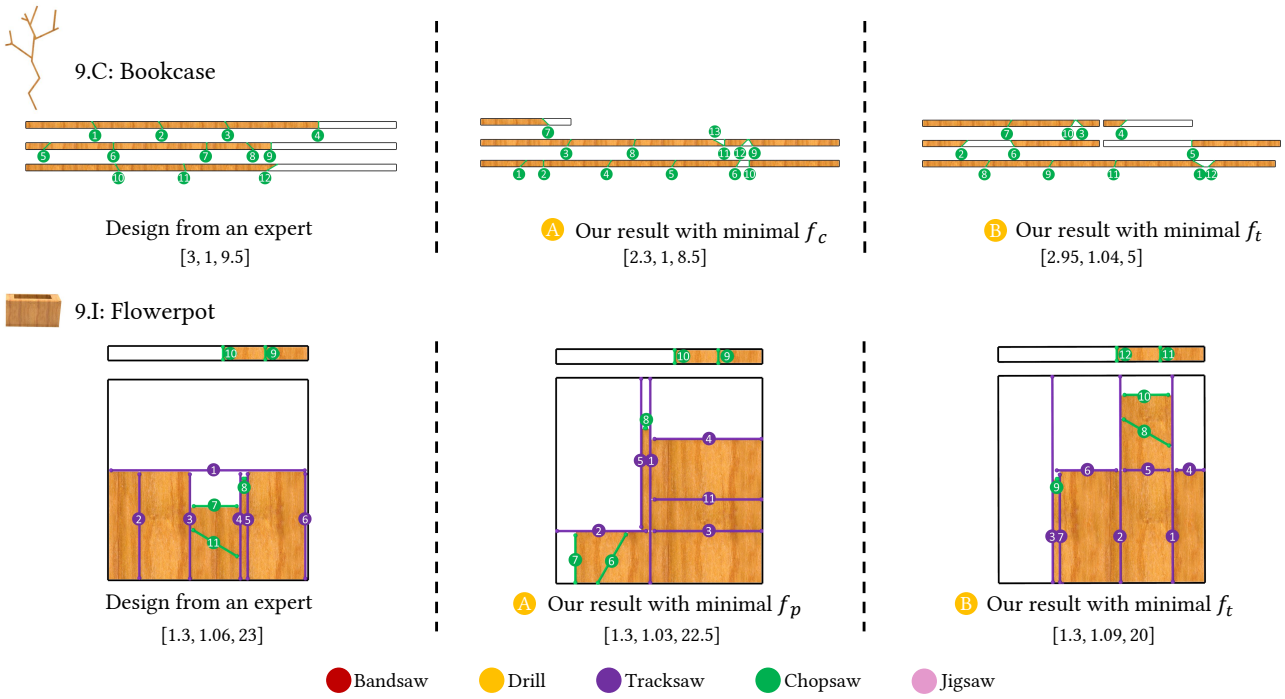


Fig. 12. The visualization results of our auto-generated LL-HELM programs and the fabrication plans hand-written by experts. Colors identify the process and numbers the order of cuts. The costs are shown in the order: f_c, f_p, f_t in square brackets below each figure.

setting will become essential in providing quality, efficiency, and safety in workshops of the future.

As manufacturing processes become increasingly sophisticated, and demand for customization increases, designers, fabricators, and even end-users will need more frameworks like HELM to support an increasingly automated and flexible idea-to-product pipeline.

ACKNOWLEDGMENTS

The authors would like to thank anonymous reviewers for their helpful feedback; Dan Grossman and Benjamin T. Jones for insightful discussions; Colin Topper, Victor Wu, Mary Mattsen, Guoxin Fang, Liang He for helping the experiments; Max Willsey and Yuxuan Mei for feedback on the draft. The work is supported by the National Science Foundation by Grant No.: 1813166 and 1644558, and an Adobe Research Fellowship. Chenming Wu is partially supported by Tsinghua Scholarship for Overseas Graduate Studies.

REFERENCES

Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. 1986. *Compilers: Principles, Techniques, and Tools.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Marc Alexa, Kristian Hildebrand, and Sylvain Lefebvre. 2017. Optimal Discrete Slicing. *ACM Trans. Graph.* 36, 1, Article 64b (Jan. 2017). https://doi.org/10.1145/2999536

Mehdi Baba-Ali, David Marcheix, and Xavier Skapin. 2009. A method to improve matching process by shape characteristics in parametric systems. *Computer-Aided Design and Applications* 6, 3 (2009), 341–350.

Moritz Bächer, Bernd Bickel, Emily Whiting, and Olga Sorkine-Hornung. 2017. Spin-it: Optimizing Moment of Inertia for Spinnable Objects. *Commun. ACM* 60, 8 (July 2017), 92–99. https://doi.org/10.1145/3068766

Sorav Bansal and Alex Aiken. 2006. Automatic Generation of Peephole Superoptimizers. In *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS XII).* ACM, New York, NY, USA, 394–403. https://doi.org/10.1145/1168857.1168906

Bernd Bickel, Moritz Bächer, Miguel A. Otaduy, Hyunho Richard Lee, Hanspeter Pfister, Markus Gross, and Wojciech Matusik. 2010. Design and Fabrication of Materials with Desired Deformation Behavior. *ACM Trans. Graph.* 29, 4, Article 63 (July 2010), 10 pages. https://doi.org/10.1145/1778765.1778800

Bernd Bickel, Paolo Cignoni, Luigi Malomo, and Nico Pietroni. 2018. State of the Art on Stylized Fabrication. *Computer Graphics Forum* 37, 6 (2018), 325–342. https://doi.org/10.1111/cgf.13327

Rafael Bidarra, Paulos J Nyirenda, and Willem F Bronsvoort. 2005. A feature-based solution to the persistent naming problem. *Computer-Aided Design and Applications* 2, 1-4 (2005), 517–526.

Christian Brecher, Mirco Vitr, and Jochen Wolf. 2006. Closed-loop CAPP/CAM/CNC process chain based on STEP and STEP-NC inspection tasks. *International Journal of Computer Integrated Manufacturing* 19, 6 (2006), 570–580.

Edmund Burke, Robert Hellier, Graham Kendall, and Glenn Whitwell. 2006. A New Bottom-Left-Fill Heuristic Algorithm for the Two-Dimensional Irregular Packing Problem. *Operations Research* 54, 3 (2006), 587–601. https://doi.org/10.1287/opre.1060.0293 arXiv:https://doi.org/10.1287/opre.1060.0293

Chengkai Dai, Charlie C. L. Wang, Chenming Wu, Sylvain Lefebvre, Guoxin Fang, and Yong-Jin Liu. 2018. Support-free Volume Printing by Multi-axis Motion. *ACM Trans. Graph.* 37, 4, Article 134 (July 2018), 14 pages. https://doi.org/10.1145/3197517.3201342

DDX. 2019. EasyWOOD, CAD/CAM software for 5 axis woodworking, nesting true shape | DDX. http://www.ddxgroup.com/en/software/easywood. (2019).

K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation* 6, 2 (April 2002), 182–197. https://doi.org/10.1109/4235.996017

Yue Dong, Jiaping Wang, Fabio Pellacini, Xin Tong, and Baining Guo. 2010. Fabricating Spatially-varying Subsurface Scattering. *ACM Trans. Graph.* 29, 4, Article 62 (July 2010), 10 pages. https://doi.org/10.1145/1778765.1778799

Tao Du, Adriana Schulz, Bo Zhu, Bernd Bickel, and Wojciech Matusik. 2016. Computational Multicopter Design. *ACM Transactions on Graphics* 35, 6 (Nov. 2016), 227:1–227:10.

FeatureScript. 2019. Weclome to FeatureScript. (2019). https://cad.onshape.com/FsDoc/.

Chi-Wing Fu, Peng Song, Xiaoqi Yan, Lee Wei Yang, Pradeep Kumar Jayaraman, and Daniel Cohen-Or. 2015. Computational Interlocking Furniture Assembly. *ACM*

Trans. Graph. 34, 4, Article 91 (July 2015), 11 pages. https://doi.org/10.1145/2766892
Sumit Gulwani, Oleksandr Polozov, and Rishabh Singh. 2017. Program Synthesis. *Foundations and Trends in Programming Languages* 4, 1-2 (2017), 1–119. https://doi.org/10.1561/2500000010
E Hopper and B.C.H Turton. 2001. An empirical investigation of meta-heuristic and heuristic algorithms for a 2D packing problem. *European Journal of Operational Research* 128, 1 (2001), 34 – 57. https://doi.org/10.1016/S0377-2217(99)00357-4
Rajeev Joshi, Greg Nelson, and Keith Randall. 2002. Denali: A Goal-directed Superoptimizer. *SIGPLAN Not.* 37, 5 (May 2002), 304–314. https://doi.org/10.1145/543552.512566
Mina Konaković, Keenan Crane, Bailin Deng, Sofien Bouaziz, Daniel Piker, and Mark Pauly. 2016. Beyond Developable: Computational Design and Fabrication with Auxetic Materials. *ACM Trans. Graph.* 35, 4, Article 89 (July 2016), 11 pages. https://doi.org/10.1145/2897824.2925944
Bongjin Koo, Jean Hergel, Sylvain Lefebvre, and Niloy J. Mitra. 2017. Towards Zero-Waste Furniture Design. *IEEE Transactions on Visualization and Computer Graphics* 23, 12 (Dec 2017), 2627–2640. https://doi.org/10.1109/TVCG.2016.2633519
David H. Laidlaw, W. Benjamin Trumbore, and John F. Hughes. 1986. Constructive Solid Geometry for Polyhedral Objects. In *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '86)*. ACM, New York, NY, USA, 161–170. https://doi.org/10.1145/15922.15904
Yanxiang Lan, Yue Dong, Fabio Pellacini, and Xin Tong. 2013. Bi-scale Appearance Fabrication. *ACM Trans. Graph.* 32, 4, Article 145 (July 2013), 12 pages. https://doi.org/10.1145/2461912.2461989
Manfred Lau, Akira Ohgawara, Jun Mitani, and Takeo Igarashi. 2011. Converting 3D Furniture Models to Fabricatable Parts and Connectors. In *ACM SIGGRAPH 2011 Papers (SIGGRAPH '11)*. ACM, New York, NY, USA, Article 85, 6 pages. https://doi.org/10.1145/1964921.1964980
Honghua Li, Ruizhen Hu, Ibraheem Alhashim, and Hao Zhang. 2015. Foldabilizing Furniture. *ACM Trans. Graph.* 34, 4, Article 90 (July 2015), 12 pages. https://doi.org/10.1145/2766912
J. I. Lipton, A. Schulz, A. Spielberg, L. H. Trueba, W. Matusik, and D. Rus. 2018. Robot Assisted Carpentry for Mass Customization. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, Brisbane, QLD, Australia, 1–8. https://doi.org/10.1109/ICRA.2018.8460736
Sheng-Jie Luo, Yonghao Yue, Chun-Kai Huang, Yu-Huan Chung, Sei Imai, Tomoyuki Nishita, and Bing-Yu Chen. 2015. Legolization: Optimizing LEGO Designs. *ACM Trans. Graph.* 34, 6, Article 222 (Oct. 2015), 12 pages. https://doi.org/10.1145/2816795.2818091
Li-Ke Ma, Yizhonc Zhang, Yang Liu, Kun Zhou, and Xin Tong. 2017. Computational design and fabrication of soft pneumatic objects with desired deformations. *ACM Transactions on Graphics (TOG)* 36, 6 (2017), 239.
Henry Massalin. 1987. Superoptimizer: A Look at the Smallest Program. In *Proceedings of the Second International Conference on Architectual Support for Programming Languages and Operating Systems (ASPLOS II)*. IEEE Computer Society Press, Los Alamitos, CA, USA, 122–126. https://doi.org/10.1145/36206.36194
James McCann, Lea Albaugh, Vidya Narayanan, April Grow, Wojciech Matusik, Jennifer Mankoff, and Jessica Hodgins. 2016. A Compiler for 3D Machine Knitting. *ACM Trans. Graph.* 35, 4, Article 49 (July 2016), 11 pages. https://doi.org/10.1145/2897824.2925940
Yuki Mori and Takeo Igarashi. 2007. Plushie: An Interactive Design System for Plush Toys. *ACM Trans. Graph.* 26, 3, Article 45 (July 2007). https://doi.org/10.1145/1276377.1276433
Pascal Müller, Peter Wonka, Simon Haegler, Andreas Ulmer, and Luc Van Gool. 2006. Procedural Modeling of Buildings. *ACM Trans. Graph.* 25, 3 (July 2006), 614–623. https://doi.org/10.1145/1141911.1141931
Chandrakana Nandi, James R. Wilcox, Pavel Panchekha, Taylor Blau, Dan Grossman, and Zachary Tatlock. 2018. Functional Programming for Compiling and Decompiling Computer-aided Design. *Proc. ACM Program. Lang.* 2, ICFP, Article 99 (July 2018), 31 pages. https://doi.org/10.1145/3236794
Step NC. 2019. Step-nc. (2019). http://www.step-nc.org/index.htm.
Charles Gregory Nelson. 1980. *Techniques for Program Verification.* Ph.D. Dissertation. Stanford University, Stanford, CA, USA. AAI8011683.
Pavel Panchekha, Alex Sanchez-Stern, James R. Wilcox, and Zachary Tatlock. 2015. Automatically Improving Accuracy for Floating Point Expressions. *SIGPLAN Not.* 50, 6 (June 2015), 1–11. https://doi.org/10.1145/2813885.2737959
David A. Patterson and Carlo H. Sequin. 1981. RISC I: A Reduced Instruction Set VLSI Computer. In *Proceedings of the 8th Annual Symposium on Computer Architecture (ISCA '81)*. IEEE Computer Society Press, Los Alamitos, CA, USA, 443–457. http://dl.acm.org/citation.cfm?id=800052.801895
Phitchaya Mangpo Phothilimthana, Aditya Thakur, Rastislav Bodik, and Dinakar Dhurjati. 2016. Scaling Up Superoptimization. *SIGPLAN Not.* 51, 4 (March 2016), 297–310. https://doi.org/10.1145/2954679.2872387
Przemyslaw Prusinkiewicz, Mark Hammel, Jim Hanan, and Radomir Mech. 1996. L-systems: from the theory to visual models of plants. In *Proceedings of the 2nd CSIRO Symposium on Computational Challenges in Life Sciences*, Vol. 3. Citeseer, 1–32.
Open CASCADE SAS. 2019. OPEN CASCADE. (2019). https://www.opencascade.org.
Christian Schüller, Roi Poranne, and Olga Sorkine-Hornung. 2018. Shape Representation by Zippables. *ACM Trans. Graph.* 37, 4, Article 78 (July 2018), 13 pages. https://doi.org/10.1145/3197517.3201347
Adriana Schulz, Ariel Shamir, Ilya Baran, David I. W. Levin, Pitchaya Sitthi-Amorn, and Wojciech Matusik. 2017. Retrieval on Parametric Shape Collections. *ACM Transactions on Graphics* 36, 1 (Jan. 2017), 11:1–11:14.
Michael Schwarz and Pascal Müller. 2015. Advanced Procedural Modeling of Architecture. *ACM Trans. Graph.* 34, 4, Article 107 (July 2015), 12 pages. https://doi.org/10.1145/2766956
Mélina Skouras, Stelian Coros, Eitan Grinspun, and Bernhard Thomaszewski. 2015. Interactive Surface Design with Interlocking Elements. *ACM Trans. Graph.* 34, 6, Article 224 (Oct. 2015), 7 pages. https://doi.org/10.1145/2816795.2818128
Alexei Sokolov, J Richard, VK Nguyen, Ian Stroud, W Maeder, and P Xirouchakis. 2006. Algorithms and an extended STEP-NC-compliant data model for wire electro discharge machining based on 3D representations. *International Journal of Computer Integrated Manufacturing* 19, 6 (2006), 603–613.
RSA Solutions. 2019. woodCAD|CAM. https://www.rsasolutions.com/products/woodcadcam/. (2019).
Peng Song, Chi-Wing Fu, Yueming Jin, Hongfei Xu, Ligang Liu, Pheng-Ann Heng, and Daniel Cohen-Or. 2017. Reconfigurable Interlocking Furniture. *ACM Trans. Graph.* 36, 6, Article 174 (Nov. 2017), 14 pages. https://doi.org/10.1145/3130800.3130803
Ross Tate, Michael Stepp, Zachary Tatlock, and Sorin Lerner. 2009. Equality Saturation: A New Approach to Optimization. In *Proceedings of the 36th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '09)*. ACM, New York, NY, USA, 264–276. https://doi.org/10.1145/1480881.1480915
The FreeCAD Team. 2019. FreeCAD Your own 3D parametric modeler. (2019). https://www.freecadweb.org/.
Jumyung Um, Matthieu Rauch, Jean-Yves Hascoët, and Ian Stroud. 2017. STEP-NC compliant process planning of additive manufacturing: remanufacturing. *The International Journal of Advanced Manufacturing Technology* 88, 5-8 (2017), 1215–1230.
Nobuyuki Umetani, Takeo Igarashi, and Niloy J. Mitra. 2012. Guided Exploration of Physically Valid Shapes for Furniture Design. *ACM Trans. Graph.* 31, 4, Article 86 (July 2012), 11 pages. https://doi.org/10.1145/2185520.2185582
Čeli APS. 2019. Woodwork for Inventor - Furniture design software. https://www.woodworkforinventor.com. (2019).
Kiril Vidimče, Szu-Po Wang, Jonathan Ragan-Kelley, and Wojciech Matusik. 2013. OpenFab: A Programmable Pipeline for Multi-material Fabrication. *ACM Trans. Graph.* 32, 4, Article 136 (July 2013), 12 pages. https://doi.org/10.1145/2461912.2461993
Philip Wadler. 1990. Linear Types Can Change the World!. In *PROGRAMMING CONCEPTS AND METHODS*. North.
L. Wang and E. Whiting. 2016. Buoyancy Optimization for Computational Fabrication. *Computer Graphics Forum* 35, 2 (2016), 49–58. https://doi.org/10.1111/cgf.12810
D. Whitfield and M. L. Soffa. 1990. An Approach to Ordering Optimizing Transformations. *SIGPLAN Not.* 25, 3 (Feb. 1990), 137–146. https://doi.org/10.1145/99164.99179
Deborah L. Whitfield and Mary Lou Soffa. 1997. An Approach for Exploring Code Improving Transformations. *ACM Trans. Program. Lang. Syst.* 19, 6 (Nov. 1997), 1053–1084. https://doi.org/10.1145/267959.267960
SQ Xie and Xun Xu. 2006. A STEP-compliant process planning system for sheet metal parts. *International Journal of Computer Integrated Manufacturing* 19, 6 (2006), 627–638.
Xun W Xu and Stephen T Newman. 2006. Making CNC machine tools more open, interoperable and intelligent - a review of the technologies. *Computers in Industry* 57, 2 (2006), 141–152. https://doi.org/10.1016/j.compind.2005.06.002
J. Zhang and L. Xing. 2017. A Survey of Multiobjective Evolutionary Algorithms. In *2017 IEEE International Conference on Computational Science and Engineering (CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC)*, Vol. 1. 93–100. https://doi.org/10.1109/CSE-EUC.2017.27
Haisen Zhao, Hao Zhang, Shiqing Xin, Yuanmin Deng, Changhe Tu, Wenping Wang, Daniel Cohen-Or, and Baoquan Chen. 2018. DSCarver: Decompose-and-spiral-carve for Subtractive Manufacturing. *ACM Trans. Graph.* 37, 4, Article 137 (July 2018), 14 pages. https://doi.org/10.1145/3197517.3201338