Facilitating the Parametric Definition of Geometric Properties in Programming-Based CAD

J. Felipe Gonzalez
Carleton University
Univ. Lille, CNRS, Inria, Centrale Lille, UMR 9189 CRIStAL
Lille, France
johannavila@cmail.carleton.ca

Audrey Girouard
Carleton University
Ottawa, ON, Canada
audrey.girouard@carleton.ca

Thomas Pietrzak
Univ. Lille, CNRS, Inria, Centrale Lille, UMR 9189 CRIStAL
Lille, France
thomas.pietrzak@univ-lille.fr

Géry Casiez*
Univ. Lille, CNRS, Inria, Centrale Lille, UMR 9189 CRIStAL
Lille, France
gery.casiez@univ-lille.fr

ABSTRACT

Parametric Computer-aided design (CAD) enables the creation of reusable models by integrating variables into geometric properties, facilitating customization without a complete redesign. However, creating parametric designs in programming-based CAD presents significant challenges. Users define models in a code editor using a programming language, with the application generating a visual representation in a viewport. This process involves complex programming and arithmetic expressions to describe geometric properties, linking various object properties to create parametric designs. Unfortunately, these applications lack assistance, making the process unnecessarily demanding. We propose a solution that allows users to retrieve parametric expressions from the visual representation for reuse in the code, streamlining the design process. We demonstrated this concept through a proof-of-concept implemented in the programming-based CAD application, OpenSCAD, and conducted an experiment with 11 users. Our findings suggest that this solution could significantly reduce design errors, improve interactivity and engagement in the design process, and lower the entry barrier for newcomers by reducing the mathematical skills typically required in programming-based CAD applications.

CCS CONCEPTS

• **Human-centered computing → Interaction techniques**.

KEYWORDS

3D programming-based CAD, OpenSCAD, parametric design

*Also with Institut Universitaire de France.

1 INTRODUCTION

Parametric *Computer-Aided Design* (CAD) uses parameters to define object geometry, allowing quick modifications and reusability [3]. This flexibility supports practices like digital personal fabrication [2], enabling users to create and share customizable models [46]. For instance, web applications such as Customizer [32] from Thingiverse [33] and MakeWithTech [43] enable users to upload models with exposed parameters, allowing others to generate various versions of the base models. Most CAD applications use direct manipulation, allowing users to edit models through visual elements like drag-and-drop, menus, and buttons [44]. They incorporate parametric features via *constraints*, which are rules applied to control dimensions, positions, or relationships within the model using modifiable parameters [7, 52]. For example, FreeCAD [47] lets users set line lengths as constraints linked to spreadsheet cells, updating the sketch when these values are changed.

Applications like OpenSCAD [27] and JSCAD [36] follow a *Programming-based* approach [11]. In Programming-based CAD, models are defined textually using programming languages, with the application rendering a visual representation in a viewport after compilation. Parametric designs are created by defining geometric properties through variables and arithmetic expressions, keeping relationships consistent regardless of parameter values. For example, a variable height can set both the height of one cube and the position of another on top of it, ensuring both adjust correctly when the variable's value changes. Creating parametric designs in programming-based CAD applications is challenging due to the complexity of deriving expressions for geometric properties, requiring math skills and spatial reasoning [11]. Even experts face difficulties aligning parameters with spatial axes, formulating correct expressions, and navigating nested transformations [11].

The geometric properties of different model parts are interrelated. Consider the OpenSCAD model 6402905 of a parametric *"Pot lid holder"* from Thingiverse[1], shown in Figure 1. The position of the highlighted part must be defined relative to the two *rounded spikes*. The user must determine the spikes' positions through spatial transformation definitions to align them with the center of the

---

[1] https://www.thingiverse.com/thing:6402905 accessed on 01/09/2024

highlighted part, as shown in Listing 1. This expression ensures that changes in the parameters controlling the spikes' positions automatically adjust the highlighted part's placement. In the scenario where the highlighted part is about to be placed, the application already has the parametric definitions for the spikes' sizes and positions. Allowing users to select components directly within the view could simplify the process, as visual identification is easier than code navigation [10, 50]. For example, clicking on a spike in Figure 1 could reveal its parametric definition, enabling users to adjust it to place the highlighted part accurately. This approach reduces the need for manual code analysis, speeding up the process.

```
65  translate([–width/2,num*(length+spike_thickness)+
        spike_thickness–thickness,0])
66  cube([width,thickness,thickness]);
```

Listing 1: Example of parametric definition of a translate in OpenSCAD
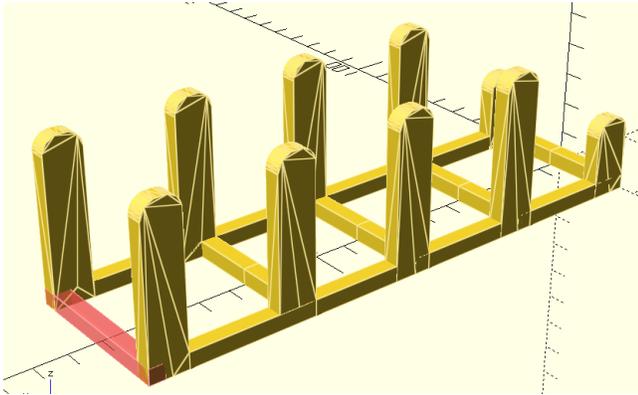

Figure 1: Pot lid holder model from Thingiverse. ID 6402905

We aim to improve the parametric design capabilities within programming-based CAD applications by introducing *bidirectional programming interactions*. Bidirectional programming allows users to directly leverage the information from the view in the code in programming-based applications. By analyzing 30 OpenSCAD models from Thingiverse, we found that geometric properties are mainly linear combinations of variables. Building on the work of Gonzalez *et al.* [10], which enabled selecting elements directly in the view, we extended OpenSCAD to allow users to extract parametric definitions from the view, simplifying model creation. An evaluation with 11 OpenSCAD users demonstrated the effectiveness of the solution, showing that it reduces design errors, improves interactivity, and lowers entry barriers by reducing the mathematical skill needed in programming-based CAD. Our contributions involve a formative study to understand better how geometric properties are defined in programming-based CAD applications, a design goal to facilitate the process of parametric design in these applications, a proof of concept in OpenSCAD, and a validation of the proposed solution.

Our implementation is available at http://ns.inria.fr/loki/bp.

2 RELATED WORK

First, we define the interaction paradigms of this work to frame our findings. Then, we explain different approaches for parametric design in CAD applications.

2.1 Interaction paradigms

Programming-based CAD refers to applications where users describe models entirely through coded instructions, with the system rendering the result in a viewport [11]. In this paradigm, the code serves as the complete model description, and any model edits are reflected in the code. Text-based applications such as JSCad [36], BRL-CAD [42], and OpenSCAD [27] fall into this category. In addition, CAD applications that use visual programming, such as BlockSCAD [22] or Grasshopper [8], are also considered programming-based CAD.

McGuffin and Fuhrman [35] introduce the concept of *Bidirectional Programming* applications, where users can modify the output using both direct manipulation and instructions. In such applications, changes made to the output through direct manipulation trigger updates in the code to maintain coherence, as seen in scalable vector graphics (SVG) environments like Sketch-N-Sketch [13] or Twoville [24]. Bidirectional programming CAD applications [4, 10, 25, 26] adhere to the programming-based CAD paradigm by keeping the code as the full model description, while also extending the interaction capabilities to include direct manipulation in the viewport, as seen in applications such as Antimony [25].

CadQuery [48] is a programming-based framework for modeling design, using Boundary Representation (BREP) [15] to specify geometric information of objects' faces, vertices, or edges. Users can modify a part by selecting it through a query and applying editing commands. However, deriving queries in complex models can be challenging and require users to connect the code with the view mentally. Mathur *et al.* [34] developed features allowing users to extract queries through mouse clicks directly from the view where selecting parts is notably easier [17]. We draw on this concept to facilitate the retrieval of information from the view.

Gonzalez *et al.* [10] present a modified version of OpenSCAD with editing and navigation features and direct manipulation interactions. The application allows users to edit the model directly in the view, applying spatial transformation through drag-and-drop interactions. Although features facilitate the editing of the model, modifications only support spatial transformation with raw numbers without using arithmetic expressions and do not facilitate the design of parametric models. Navigation features allow users to connect the code and the view with visual cues.

We leverage the concept of bidirectional programming, specifically in Mathur *et al.* [34] and Gonzalez *et al.*'s work [10], as the foundation for developing our solution.

2.2 Parametric design in direct manipulation CAD applications

Parametric CAD applications implementing a direct manipulation approach fix geometric properties through *constraints* [7, 14]. A constraint is a rule applied to geometric elements within a model to control dimensions, positions, or relationships between components. There are two types of constraints: Geometric and Dimensional [30]. Geometric constraints define the relationship between

two or more elements in the scene. For example, to force two lines always to keep the same length. Dimensional constraints fix the values of geometric properties of elements such as positions, sizes, or angles. Table 1 describes common constraints in CAD applications such as FreeCAD [40, 47], Fusion360 [20, 21], or AutoCAD [18, 19].

Constraints help define and enforce specific geometric relationships between different design parts. In parametric design, these constraints are often expressed using variables, which users can adjust to create various versions of the design [23]. For instance, consider a box design where a variable defines the width. By exposing this variable as a model parameter, users can easily modify its value as needed. Subsequently, the application will regenerate the box with the updated width value.   Even for experts, creating constraints can be challenging. Solutions like CODA [49] assist by suggesting applicable constraints based on elements in the view. We drew inspiration from this concept, recognizing that leveraging existing application information can enhance our solution's definition of new geometric properties.

Table 1: Common Constraints in Direct Manipulation CAD

Geometric
Coincident: Forces two points or objects to share the same location.
Collinear: Requires two elements to lie on the same straight line.
Concentric: Enforces a common center point for two circles.
Parallel: Aligns two lines or edges to be parallel.
Perpendicular: Forces two lines or edges to meet at a right angle.
Dimensional
Distance: Specifies the distance between two points or objects.
Angle: Defines the angle between two lines or edges.
Radius/diameter: Sets the radius/diameter of a circle or arc.
Length: Determines the length of a line or the size of an object.
Width/height: Specifies the width or height of an object.

It is noteworthy that in direct manipulation CAD applications users express design intents through tools that are described in a more explicit language compared to programming-based CAD. As seen in Table 1, constraints include high-level definitions such as making two lines collinear. This forces the application to interpret them and propose a solution. In other words, the user expresses **WHAT** they want, and the system seeks a solution to provide it. This differs from programming-based CAD applications where users need to describe **HOW** the models are built.

2.3 Parametric design in programming-based CAD applications

When designing in programming-based CAD applications, users define all the geometric properties of the model, except when the application provides default values to information not provided in the code. For example, creating a cube without specifying its size parameter results in a default cube of size 1×1×1 being generated. Users describe **HOW** geometric properties are formed through programming and mathematical expressions. Understanding how users in programming-based environments define geometric properties in parametric designs is crucial to facilitating the design process.

However, beyond code comments, there is often no clear indication of the users' intentions behind these definitions. For example, in the provided example in Listing 1, the rationale behind specific choices for location and size definitions is not immediately apparent. Furthermore, there is a lack of research investigating design patterns in defining geometric properties in programming-based CAD. Chytas and Tsilingiris [5] study how 13 to 17-year-old students create programming-based models. Later, Chytas *et al.* [6] studied several OpenSCAD models from websites to identify programming patterns and design preferences. Their research provides statistics on various code statements (*e.g.* frequency of loops, conditionals, or spatial transformation usage) but does not delve into how geometric properties are interrelated with other objects. Previous research [11] with OpenSCAD users has shown that users often struggle to formulate mathematical expressions for parametric models, a process scarcely supported by existing tools. Furthermore, users indicated that the definition of objects' positions and sizes is frequently relative to the positions and sizes of other objects, highlighting a complex interdependence in design decisions.

We draw on these works to address the identified challenges.

3 METHOD

We aim to facilitate the parametric design in programming-based CAD applications. First, we conducted a formative study analyzing 30 OpenSCAD models from Thingiverse to identify how the geometric properties are defined. Then, based on our findings, we define the design goals for a bidirectional application that facilitates the definition of geometric properties in parametric models. Later, we reused and modified the source code of Gonzalez *et al.* [10] to allow users to retrieve parametric definitions of objects directly from the view to be reused in the code. Finally, we tested our modified version with OpenSCAD users and analyzed their user experience.

3.1 Formative study

Programming-based CAD applications allow users to define geometric properties with programming expressions. For example, the size of a cube can be defined with a raw number, a variable, an arithmetic expression, or more complex programming structures such as conditionals. Based on Gonzalez *et al.*'s survey about the challenges of OpenSCAD users [11], we hypothesize that users usually define the positions and sizes of elements as a linear combination of the positions and sizes of other elements. For example, a common operation is placing a box on top of another box. In such a case, the position of the second box is defined in terms of the size and position of the first cube, as depicted in Listing 2.

```
4  // First cube
5  cube(size = size_cube_a, center = true);
6  // Second cube
7  translate([0,0,size_cube_a/2 + size_cube_b/2])
8  cube(size = size_cube_b, center = true);
```

Listing 2: Parametric model of a cube on top of another cube.

With more complex models, the definition of geometric properties considers the position and orientation of multiple parts. As a result, we hypothesize that, often, they are described as linear

combinations defined as `translate([tx, ty, tz])` where $t_i = \sum \alpha_j x_j + c$, with α_j, c constants, and x_j a variable.

To assess this assumption, we have analyzed 30 models from Thingiverse. We filtered the customizable models with the option "*Popular Last 7 Days*" and downloaded the first ten models. We repeated the same process with the filters "*Popular Last 30 Days*" and "*Popular This Year*". Duplicated models were discarded and replaced with the next ones on the list. Figure 11 in the appendix A contains all the models used in the formative study.

We modified OpenSCAD to analyze the definition of geometric properties. The application parses the code into an *Abstract Syntax tree* (AST) [1]. Then, it scans the AST to identify code statements responsible for generating primitive geometries (*e.g.* spheres or cylinders) and executing spatial transformations (*i.e.* translations, rotation, scale). The application identifies the nature of parameter definitions, either *C1* for non-default raw numerical values, *C2* for a single variable, *C3* for a linear combination of variables, *C4* for a non-linear combination, or *C5* for a structure involving more complex programming constructs, as delineated in Table 2. For example, a cube defined as `cube(size = [5, size_y, size_z+3])` would be classified under C1, C2, and C3, whereas a spatial transformation like `translate([0,0, size_x*i])` would be allocated solely to the C4 category. C1 and C2 are included in the C3 definition, but we keep the difference seeking a detailed analysis.

Table 2: Categories of expressions in OpenSCAD models.

ID	Classification	Description
C1	Raw number	Non-default numeric. e.g., `4.0`
C2	One variable	A variable call. e.g., `var1`
C3	Linear combination	Linear combination of variables $\sum \alpha_i \cdot x_i + c$. e.g., `3 + 2*var1 - var2`
C4	Polynomial expression	Non-linear polynomial expressions $\sum \alpha_i \cdot x_i \cdot y_i$. e.g., `3 + 2*var1*var2`
C5	Other	Other programming structures such as conditionals. *e.g.*, `(var1>3)?1:2`

The results of the analysis are detailed in Table 3. It is important to note that the `scale` statement is barely used in the models. Its participation in the total of expressions analyzed is only 1%. Furthermore, most expressions within the `rotate` statements are raw numbers, with 140 out of 286 `rotate` statements. Indeed, when validating the results, we confirmed that in most cases, rotations are performed at standard angles such as 45 or 90 degrees. Finally, we confirmed our hypothesis, verifying that most of the positions (through `translate` statements) and sizes (through primitive definitions) are defined as raw numbers (C1), one variable call (C2), or a linear combination of existing variables (C3). These categories represent 71% of the total parameters analyzed in primitives definition (44%) and spatial transformations (27%).

3.2 Design goals

Users define geometric properties by using the relationships between objects, as outlined in previous research [11] and corroborated by the formative study. Concerning sizes and positions, statements frequently define these properties as linear combinations of

Table 3: Formative study results. Total and percentual occurrences per category used to define parameters in primitives, translation, rotation, and scale statements

	Primitive	Translate	Rotate	Scale	Total
C1	196 (11%)	130 (7%)	140 (8%)	8 (0.4%)	474 (25%)
C2	294 (16%)	126 (7%)	35 (2%)	2 (0.1%)	457 (25%)
C3	312 (17%)	234 (13%)	29 (2%)	5 (0.3%)	580 (31%)
C4	0 (0%)	48 (3%)	31 (2%)	4 (0.2%)	83 (4%)
C5	26 (1%)	191 (10%)	51 (3%)	0 (0%)	268(14%)
Total	828 (45%)	729 (39%)	286 (15%)	19 (1%)	1862 (100%)

variables, corresponding to linear relationships between elements. The positioning and sizing of a new model element often depend on the location and dimensions of another object. Moreover, identifying a position is straightforward in the visual representation [10, 34]. A valuable tool would enable the extraction of model information to define the geometric properties of other elements within the code. This information must be readily accessible, allowing users to understand its spatial implications directly in the visual representation, where identification is the simplest. In essence, the design goal is to *facilitate the extraction of geometric information from objects' parametric definitions in the view for code reuse.*

4 BIDIRECTIONNAL PROGRAMMING TO DEFINE GEOMETRIC PROPERTIES

We have implemented a proof-of-concept that introduces features that enhance the creation of parametric models using bidirectional programming. We have re-used the modified version of OpenSCAD from Gonzalez *et al.* [10]. Specifically, we reused the implemented feature to select an element in the view by clicking on it. We modified this version based on Mathur *et al.* [34] work to fulfill the design goal by allowing users to retrieve information from the view.

OpenSCAD parses code into an AST and later into the Abstract CSG Tree, evaluating all programming structures and variables and replacing them with the raw values, retaining only numeric information after evaluation. No information about the parametric definition of objects is stored at this stage. We modified the source code of OpenSCAD to ensure that CSG tree nodes store the parametric definition of the geometric properties used to define them. Primitives store the definition of the size, whereas spatial transformation stores the parametric definition of the transformation.

When users position an object relative to another, they are often concerned with specific locations around it. For instance, when placing cube *A* on top of cube *B*, the `translate` statement needs to consider the top of cube *B* and the bottom of cube *A*. Unfortunately, most programming-based CAD applications work with a CSG representation where definitions are abstract, and there is no information on vertices, faces, or edges [15, 39]. We redefined CSG node definitions in OpenSCAD to include *handles* that the user can use to retrieve the parametric definition of the position of an object. Handles were added, creating a grid of 3×3×3 points distributed symmetrically around the object's center in 3D primitives. For 2D primitives, the application created a 3×3 grid. Details about the distribution of handles are shown in Table 4 in Appendix A.

Non-primitive nodes include a single handle at the node's position, covering boolean operations, spatial transformations, or programming structures. Given a determined handle of a selected node, the application can define the position of the handle in terms of the variables used in the code. The application iterates on the CSG tree to locate the selected node. Then, the selected node provides the definition of the position of the handle relative to the node's center. Later, the application iterates on `translate` nodes in the branch of the selected node, adding their definitions to the position of the handle. Figure 2 describes how OpenSCAD derives the parametric position of the handle placed in the middle of the bottom face of the cube (axis z) created by the code in Listing 3.

```
7  translate([tx,ty,tz])
8  cube([size_x,size_y,size_z]);
```

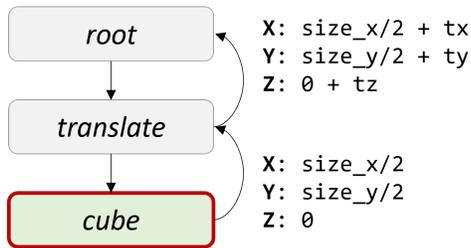**Listing 3: Handles example**

Figure 2: Representation of how OpenSCAD computes the position of the handle placed at the center of the bottom face of a cube.

OpenSCAD gathers information from the CSG nodes without filtering out trivial values, such as translating 0 units in one direction, leading to less readable expressions and requiring an expression simplification. To streamline the development, we implemented a Python server that exposes a service to simplify arithmetic expressions using the *simpy* library [45]. OpenSCAD sends the raw expression to the server, which answers with a simplified expression. If there is a communication error with the server, OpenSCAD uses the non-simplified expression instead. We developed two features to facilitate information retrieval from the model's view. These features, built on the modified version of Gonzalez *et al.* [10], enable users to *select* objects within the model and use the capabilities of *position* and *delta vector*.

4.1 Position

The position feature allows users to determine the location of a handle in a selected object relative to the origin (*i.e.* , CSG root position `[0,0,0]`) of the view. Users activate this feature by selecting the *Position* button in the menu bar. Then, users can select an object [10] and the application displays the handles, marking the object's center with an always-visible purple handle. The rest of the handles behave like any other geometry and can be hidden behind other geometries. This feature aims to provide information to the user without automatically editing the code, ensuring user control over the definition of the model [34]. Users can right-click on any

handle to turn it green, indicating that the application has copied the parametric position to the clipboard so the user can use it in the code to define a new element property.

Consider the case where a user is designing a cup. The user has placed a cylindrical base and a cylindrical stem on top, as depicted in Listing 4. Using the position feature, the user could place a cylinder for the cup on top of the stem (Figure 3a).

```
1   thickness = 6;
2   r_base = 24;
3   r_stem1 = 6;
4   r_stem2 = 3;
5   h_stem = 30;
6   r_top = 18;
7   h_top = 33;
8   // Cup
9
10
11  // Stem
12  translate([0,0,thickness]){
13  cylinder(r1=r_stem1,r2=r_stem2,h=h_stem);
14  }
15  //Base
16  cylinder(r=r_base,h=thickness);
```

Listing 4: Example before using position feature

The user could first select the stem's cylinder by right-clicking on it. The user could select the cylinder in the menu displaying the CSG nodes involved in that part, as depicted in Figure 3b. OpenSCAD would display the handles, and the user could select the one in the middle of the top where the cup cylinder will be placed. The handle would turn green so that the user would have in the clipboard the definition of the position of that point in terms of the variables used in the model (Figure 3c). The user can then create a translation, paste the definition stored in the clipboard as shown in Listing 5, and add a cylinder to create the cup (Figure 3d).

```
7   h_top = 33;
8   // Cup
9   translate( [0,0,h_stem+thickness] )
10  cylinder(r = r_top, h = h_top);
```

Listing 5: Example after using position feature

4.2 Delta Vector

The delta vector feature calculates the vector between two handles. This arithmetic expression allows aligning a handle of one object with a handle of another object, establishing a *coincidence* or a *snapping* effect. The process is similar to the position feature but in a two-step process. Users enable this feature by pressing the *delta vector* button in the menu bar and then selecting the object they want to move. The application marks the center of the object with always visible purple handles. The rest of the handles behaves like any other geometry and can be hidden behind other geometries. The user selects a destination object, and the application also shows its handles, with centers in purple and others in blue. In this setup, red points indicate origin points, and blue points mark destination points. After right-clicking on a red handle, which turns it white, the user selects the destination point by right-clicking on it. This action
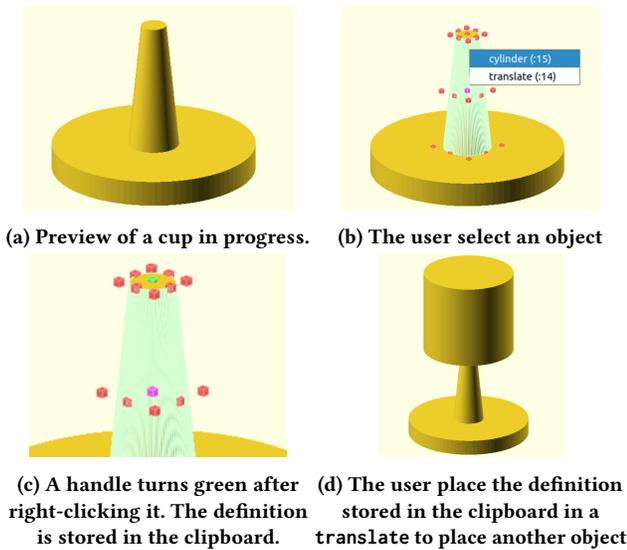
(a) Preview of a cup in progress.

(b) The user select an object

(c) A handle turns green after right-clicking it. The definition is stored in the clipboard.

(d) The user place the definition stored in the clipboard in a `translate` to place another object

Figure 3: Position feature allows users to retrieve the location of an object's handle relative to the origin.

turns both handles green to inform the user that the system has placed the parametric definition in the clipboard. The delta vector feature calculates the difference between the destination and origin handles, allowing users to determine the necessary transformation to align the origin and the destination points.

Revisiting the example of a cup, consider the addition of decorative spheres around its upper part. Using the delta vector feature allows for precise placement. The user selects the sphere and the cylinder at the top of the piece, prompting the handles to appear on both. By right-clicking on corresponding handles intended to align, as shown in Figure 4a, the application generates the exact transformation [r_top − r_sphere,0, thickness + h_stem + h_top] and stores it in the clipboard. Inserting it into a `translate` statement accurately positions the sphere, as seen in Figure 4b. A loop can be later added to place additional decorations symmetrically.
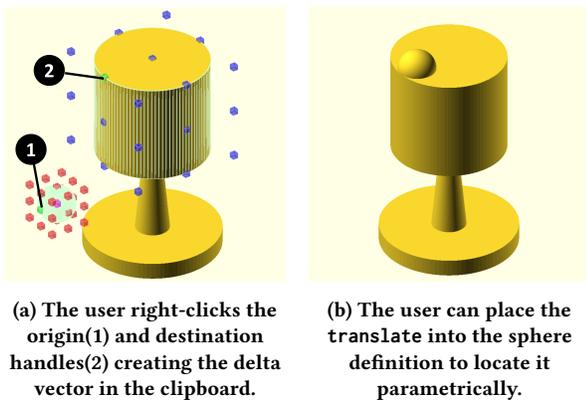

(a) The user right-clicks the origin(1) and destination handles(2) creating the delta vector in the clipboard.

(b) The user can place the `translate` into the sphere definition to locate it parametrically.

Figure 4: Delta vector allows users to place one object's handle relative to another object's handle.

5 USER STUDY

We conducted an experiment with eleven OpenSCAD users to evaluate the effectiveness of bidirectional programming in simplifying the parametric design process in programming-based CAD.

The experiment consisted of three parts. Firstly, we collected demographic information from participants and asked about their experience with other CAD applications and general programming languages. In the second part, participants performed a task to create a parametric design using the original OpenSCAD version. Upon completion, we discussed the challenges encountered and their overall experience. We then introduced and demonstrated the features implemented in OpenSCAD. Participants practiced briefly with the enhanced OpenSCAD version before creating a second parametric model, utilizing the new features where applicable. Given the participants' expertise in OpenSCAD, we deemed any learning effect negligible and thus did not counterbalance the use of the two OpenSCAD versions. For consistency, participants utilized the original version first as a control step, followed by the modified version. The third part involved participants sharing their experiences using the new features and discussing the potential impact of such solutions in programming-based CAD applications.

Each experiment session lasted approximately 90 minutes. We took detailed notes on participants' responses and their design thinking processes. Additionally, we recorded the screen during the design tasks to assess performance.

5.1 Recruitment and Participants

We recruited participants from social media OpenSCAD channels on Reddit (r/openscad) and Facebook (OpenSCADAcademy) to conduct the experiment using video conferencing. The only requirement for participation was proficiency in creating parametric designs with OpenSCAD. Before the sessions, participants were instructed to install the AnyDesk remote desktop application [9]. They accessed a Linux machine we prepared using AnyDesk to perform parametric design tasks during the experiments.

All participants self-identified as male and varied in age between 20 and 69 years old (average: 44.5, standard deviation: 14.2). All participants, except P3, had four or more years of 3D modeling experience (average: 8.9y, standard deviation: 5.8). Except for P3, all participants self-rated with four or more in at least one programming language. Finally, participants self-rated their skill level with OpenSCAD as follows: Two participants with 2, four participants with 3, four participants with 4, and one participant with 5.

5.2 Design tasks

Participants performed two parametric design tasks, first using the original and later our modified version of OpenSCAD. We proposed two models: model A, a chalice-like model (Figure 5a), and model B, a box (Figure 5b), aiming to make them comparably challenging in terms of the number of required primitives, spatial transformations, and boolean operations. However, we designed them with distinct structures to avoid redundancy in the design experience. For model A, participants were required to expose parameters for the size of the cutouts in the base, the sizes of the holes in the cup, the height and radius of the cup, and the length of the stem. In Model B, participants were required to expose parameters for the length of

the legs, the size of the windows, and the box's height, width, and depth. We converted the models into STL files and uploaded them to the STL online viewer, 3DViewer [29], generating shareable links that allowed participants to view the models in 3D on their computers. To mitigate order bias, we counterbalanced the sequence: half of the participants worked on Model A first, followed by Model B, while the others started with Model B, and then proceeded to Model A.
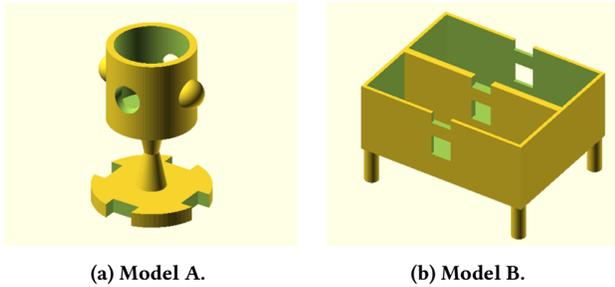

(a) Model A. **(b) Model B.**

Figure 5: Models used in the experiment. Participants replicated the models, exposing parameters as required

In the first task, participants used the original OpenSCAD version. This exercise had two goals: first, to establish a baseline for comparing the performance of the design process and user experience with the modified OpenSCAD version; second, to refresh participants' understanding of parametric design, facilitating a subsequent discussion about its challenges. After completing the first design, we asked the participants about their user experience, task difficulty, and specific challenges in the execution of parametric designs with OpenSCAD. We then introduced the new OpenSCAD features using elements from their initial designs. This was followed by tasks like determining the parametric position of a cube's corner or positioning the bottom of a sphere on top of a cube to familiarize participants with the new features. After about 10 minutes of practice and answering questions, participants embarked on the second design task, encouraged to utilize the new features where feasible. The users then discussed their experience and the potential of such solutions for programming-based CAD applications.

5.3 Data collection

We recorded the OpenSCAD window activity while participants worked on both design tasks. Recording using the original version of OpenSCAD were compared to the recordings of the modified version of OpenSCAD to evaluate the potential and challenges of our solution. In addition, upon completing each model, participants were asked to rank the task's difficulty. At the end of the second design exercise, they provided comparative evaluations of both versions of OpenSCAD. Participants answered Likert scale questions focused on the functionality and usability of the new features and engaged in discussions about their perceptions of these solutions.

5.4 Data analysis

We evaluated participants' feedback on task difficulty, feature functionality, and usability. We summarize their responses. Furthermore,

we recorded participants' `translate` statements and their verification attempts, comparing these results with those from designs in the original OpenSCAD version.

5.4.1 Perception on implemented features. Participants shared their experiences with the difficulty of creating the models and the functionality and usability of the implemented features. All participants rated the difficulty of both models between *Neutral* (option 3), *Easy* (option 4), and *Very Easy* (option 5), as shown in Figure 6.
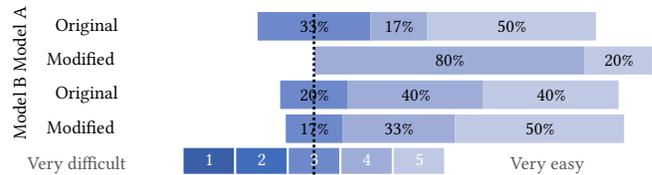

Figure 6: Perceived difficulty of models in the original and modified version of OpenSCAD using the scale: 1 (Very Difficult), 2 (Difficult), 3 (Neutral), 4 (Easy), and 5 (Very Easy).

After completing the design exercise using the implemented features, we asked participants if the modified version of OpenSCAD made the design task easier or more difficult (Figure 7). All participants answered above *About the same* (option 3), with seven participants with *Somewhat Easier* (option 4) and four participants with *Much easier* (option 5). Then, we asked a similar question but individually targeted both features. Not all participants used both features in the design exercise according to personal preference, so answers in Figure 7 report the percentage of the total of participants who used the feature and answered the question: seven participants for the position feature and ten participants for the delta vector feature. For the position feature, one participant (14.3%) answered about the same, and six participants (85.7%) answered somewhat easier. Regarding the delta vector feature, four participants (40%) answered somewhat easier, and six participants (60%) answered much easier.
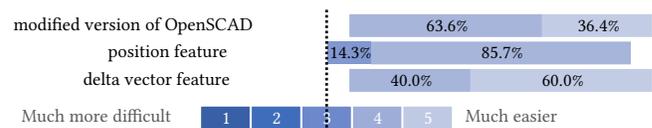

Figure 7: Participants answered if the modified version of OpenSCAD and each feature made the design easier or more difficult. 1-Much more difficult, 2–Somewhat more difficult, 3–About the same, 4-Somewhat easier, 5–Much easier

We also asked how difficult it was to use each feature regarding usability, as depicted in Figure 8. Similar to the previous question, the answers report the total number of participants who used and answered the question about the feature. In all cases, all participants answered between *Neutral* (option 3), *Easy* (option 4), and *Very easy* (option 5). For the position feature, six participants (58.7%) answered easy, and one participant (14.3%) answered very easy. Regarding the delta vector feature, four participants (44.4%) answered

neutral, and five participants (55.6%) answered easy. Three participants indicated that selecting control points with a right-click is inconvenient. P4 commented "*My initial inclination is to left-click those handles; it's a little bit extra to remember to right-click the handle*". Furthermore, the control points did not scale when zoomed in, which was also reported as problematic. For the delta vector, half of the participants answered Neutral (option 3) and the other half Easy (option 4). Participants found the process with two objects difficult to remember and listed some problems. For example, participants missed visual cues that guided them through the different steps. P5 commented "*There was no clear prompt indicating what was copied to the clipboard; it's unclear if it's correct. Upon pasting, the directionality, whether red goes into blue or vice versa, is confusing.*" Further, P3 mentioned that using color to indicate the process can be difficult for some people "*I'm not exactly colorblind, but it's hard for me to see colors. So it's nice for people like us if you have an indication that is not entirely dependent on colors.*" However, they found it easy overall, as commented P1 "*It is not obvious but easy*".
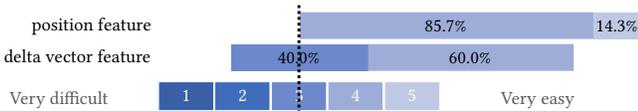

Figure 8: Participants answered how difficult was to use the implemented features with the scale 1 – Very difficult, 2 – Difficult, 3 – Neutral, 4 – Easy, 5 – Very easy

Later, we asked participants if they thought that these features would help them in the design process in their normal modeling process. All participants answered **Yes**. Participants found several advantages. P1 commented that it could help to avoid errors when designing parametrically: "*A few days ago, I positioned objects by adding variables I believed would bring them to the correct position. It appeared to be correct in the preview …However, when one value changed, the alignment was disrupted. One part was not where it was supposed to be. It was only correctly positioned when the variables coincidentally lined up.*". P2 found that this is more interactive than other alternatives that try to include "anchors" selectable from the code. "*Tools like Cascade are being used by designers who are trying to add anchors to objects, making them selectable in the code. What you're doing is making a user interface more interactive, combining the interactivity of Fusion 360 with the capabilities of OpenSCAD, and putting together the advantages of both worlds. So I think this is a better solution to the problem.*" For instance, P4 and P9 found that such features could facilitate the transition of people with little experience into programming-based CAD applications. P9 commented "*I think it would be incredibly valuable in helping users transition from normal CAD to scripted CAD, especially for those who don't have a rigorous background in computer science or math*" respectively. Finally, P2, P5, P6, P7, P8, and P10 mentioned that this would facilitate the deriving of mathematical expressions, making the design faster. P2 said "*When designing objects that need to be combined to form one design, I often do calculations to position things. This would mean fewer calculations for me to do*".

5.4.2 Comparing original and modified version of OpenSCAD. Our analysis focused on participant approaches to defining `translate`

statements in both the original and the modified versions of Open-SCAD. When participants defined a translation, they continued to render the result to verify the correctness of the code. Each rendering *attempt* was logged and categorized based on the outcome: *Success* for correct placements, *Wrong location* for incorrect placements, *Uncertain/false positive* for when participants were unsure or incorrectly deemed the placement, and *Syntax errors* for errors in the programming syntax.

Participants generated a total of 94 `translate` statements using the original OpenSCAD version (52 in Model A and 42 in Model B) and 85 with the modified version (42 in Model A and 43 in Model B). The original version had 155 rendering attempts (averaging 1.64 attempts per translation), while the modified version had 117 attempts (averaging 1.37 attempts per translation), possibly implying that with our modified version, participants required fewer attempts to reach a successfully `translate` definition. The distribution of these attempts across different categories reveals significant insights. As the number of spatial transformations differs between programming styles, we focused on investigating how difficult it is to correctly define the `translate` statements defined by the users in terms of the number of attempts per statement. As illustrated in Figure 9, the modified version demonstrated a higher success rate and fewer instances of incorrect placements, uncertain/false positives, and syntax errors compared to the original version.
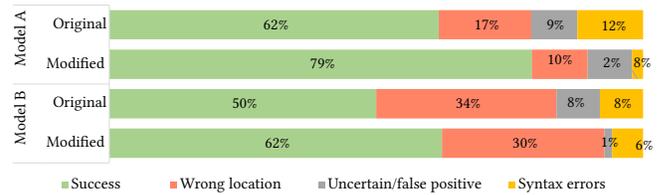

Figure 9: Attempts to verify a `translate` statement upon rendering was logged and classified to compare versions.

During the second design using the modified version of Open-SCAD, not all translations were defined using the developed features. Participants often opted to calculate expressions manually. Figure 10 depicts the different attempts using the modified version of OpenSCAD using the implemented features or describing the `translate` manually. The participants actively tried to use the features. Interestingly, Model B presented more errors using the features. We perceived that Model B geometry presented more cases where the control points were hidden by geometries and cases where participants could not find a control point in the location they needed.

6 DISCUSSION

We aim to facilitate the parametric designs in programming-based CAD applications, particularly on the difficulty of defining objects' parametric geometric properties based on others' properties [11]. We evaluated the potential and challenges of our proposed solution.

Participants unanimously agreed that our solution would ease the design process, helping avoid errors, simplifying mathematical definitions, enhancing interactivity, and lowering the skill barrier for newcomers in programming-based CAD. We identified two
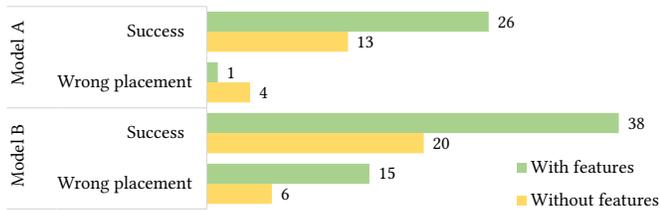
Figure 10: Outcomes of attempts to verify `translate` using the modified version of OpenSCAD

user scenarios: newcomers and experts. Prior studies indicate newcomers often avoid design tasks due to high-skill requirements [16], worsened in programming-based CAD by the semantic gap between natural and programming languages [28, 31, 41]. Effective design requires visual inspection to determine the next element's parametric position. Our solution reduces this "gulf" between evaluation and execution [51] by allowing users to extract positions directly from the view, bypassing the need to interpret code.

Experts also face challenges with mathematical definitions [11]. They follow mechanical workflows, yet hesitate over parameter accuracy in complex designs, resorting to trial and error. Our solution speeds up this process by providing accurate positional descriptions, but challenges experienced users accustomed to coding workflows. Users often find solutions restricted to the features of the programming language. For instance, JSCAD offers functions to extract geometry boundary box information (*i.e.*, minimum and maximum values in all dimensions) for later use in code [12], but this still confines users to the code editor. In contrast, our approach focuses on direct view interaction, allowing easy identification of positions without deriving them in the code, proven efficient in other applications [34].

The comparison between the original version of OpenSCAD and our approach revealed that using the developed features, participants would require fewer attempts to reach the aimed geometric properties, resulting in a proportional lower rate of errors and higher rates of successful attempts. Despite these benefits, experienced users showed resistance to changing established workflows. P6 noted, "*It seems easy to use, but changing the mindset to use the view for design is challenging.*" Similar opinions appeared in OpenSCAD forums [37, 38], where discussions about integrating such solutions are limited to programming language features. P11 "*This is a philosophical difference among OpenSCAD users regarding textual programming and visual editing. Some users prefer doing everything in code.*" While resistance may occur, our study suggests the benefits can facilitate adaptation.

Our implementation has usability challenges, as noted by users. Some control points were inaccessible due to overlaps with other volumes, and using right-click for selection was not intuitive. These feedback points are crucial considerations for future refinements of our solution. Another issue involves removed geometries in `difference` statements. Participants wanted to use the features to place subtracted elements, but these were not reachable from the view. Some used background modifiers to make these geometries visible and selectable. Typically, participants placed the cursor in the statement they were modifying. Applications could make

visible the element creating the code statement where the cursor is placed, using visual cues [10]. This provides an explicit visual representation of the part being worked on, helpful for subtracted geometries. A final challenge is understanding nested transformations. Participants sometimes wanted to use the position feature to replace a `translate` statement, but these were often inside other `translate` statements. Since the position feature gives the position relative to the CSG root, the retrieved definition is not useful inside another transformation. The application could consider the cursor's position to incorporate previous transformations when placing new definitions.

7 LIMITATIONS

Our study intended to compare the performance of the original and the modified version of OpenSCAD. However, the 15-minute practice session seemed to be insufficient for users to get used to the logic of the new features. We concluded that a longer use time would be necessary to evaluate this factor and focused on the user experience, which we considered more important. Moreover, our solution only considers a limited set of cases. Specifically, it does not include cases with spatial transformations other than `translate`. Some of our recommendations are related to newcomers, although none of the participants were newcomers. Further exploration with beginner users must be carried out to confirm our suggestions.

8 CONCLUSION

We hypothesized a general structure for creating geometric properties in parametric designs within programming-based CAD applications. We conducted a formative study to test our hypothesis, analyzing the code of thirty OpenSCAD models sourced from Thingiverse, which validated our initial assumptions. Subsequently, we proposed a design goal centered on a bidirectional programming approach to streamline the creation of parametric models in programming-based CAD applications. We modified the source code of OpenSCAD to achieve this goal, implementing features that align with our design objectives. To validate our solution, we conducted an experimental study involving eleven OpenSCAD users who created parametric designs using both the original and our modified versions of OpenSCAD. They evaluated their experience and discussed the challenges and potential of such solutions.

Our findings indicate that allowing users to retrieve information directly from the view using direct manipulation interactions to reuse in the code holds significant promise. This approach could notably reduce design errors, enhance the interactivity and appeal of the design process, and facilitate the entry for newcomers by reducing the mathematical skills requirements typically associated with programming-based CAD applications.

ACKNOWLEDGMENTS

REFERENCES

[1] Alfred Aho, Monica Lam, Ravi Sethi, and Jeffrey Ullman. 2006. *Compilers - Principles, Techniques, and Tools* (2 ed.). Pearson/Addison Wesley, Boston. https://dl.acm.org/doi/10.5555/1177220

[2] Alexander Berman, Francis Quek, Robert Woodward, Osazuwa Okundaye, and Jeeeun Kim. 2020. "Anyone Can Print": Supporting Collaborations with 3D Printing Services to Empower Broader Participation in Personal Fabrication. In *Proceedings of the 11th Nordic Conference on Human-Computer Interaction: Shaping Experiences, Shaping Society (NordiCHI '20)*. Association for Computing Machinery, New York, NY, USA, 1–13. https://doi.org/10.1145/3419249.3420068

[3] Jorge D. Camba, Manuel Contero, and Pedro Company. 2016. Parametric CAD modeling: An analysis of strategies for design reusability. *Computer-Aided Design* 74 (May 2016), 18–31. https://doi.org/10.1016/j.cad.2016.01.003

[4] D. Cascaval, M. Shalah, P. Quinn, R. Bodik, M. Agrawala, and A. Schulz. 2022. Differentiable 3D CAD Programs for Bidirectional Editing. *Computer Graphics Forum* 41, 2 (2022), 309–323. https://doi.org/10.1111/cgf.14476

[5] Christos Chytas, Ira Diethelm, and Alexandros Tsilingiris. 2018. Learning programming through design: An analysis of parametric design projects in digital fabrication labs and an online makerspace. In *2018 IEEE Global Engineering Education Conference (EDUCON)*. 1978–1987. https://doi.org/10.1109/EDUCON.2018.8363478 ISSN: 2165-9567.

[6] C. Chytas, A. Tsilingiris, and I. Diethelm. 2019. Exploring Computational Thinking Skills in 3D Printing: A Data Analysis of an Online Makerspace. In *2019 IEEE Global Engineering Education Conference (EDUCON)*. 1173–1179. https://doi.org/10.1109/EDUCON.2019.8725202 ISSN: 2165-9567.

[7] Steven Anson Coons. 1963. An outline of the requirements for a computer-aided design system. In *Proceedings of the May 21-23, 1963, spring joint computer conference (AFIPS '63 (Spring))*. Association for Computing Machinery, New York, NY, USA, 299–304. https://doi.org/10.1145/1461551.1461588

[8] Scott Davidson. 2023. Grasshopper. https://www.grasshopper3d.com/

[9] AnyDesk Software GmbH. 2024. Anydesk -The smart choice for remote access. https://anydesk.com/en

[10] J Felipe Gonzalez, Danny Kieken, Thomas Pietrzak, Audrey Girouard, and Géry Casiez. 2023. Introducing Bidirectional Programming in Constructive Solid Geometry-Based CAD. In *Proceedings of the 2023 ACM Symposium on Spatial User Interaction (SUI '23)*. Association for Computing Machinery, Sydney, Australia, 1–12. https://doi.org/10.1145/3607822.3614521 https://hal.science/INRIA2/hal-04194045v1.

[11] J. Felipe Gonzalez, Thomas Pietrzak, Audrey Girouard, and Géry Casiez. 2024. Understanding the Challenges of OpenSCAD Users for 3D Printing. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI 2024)*. Association for Computing Machinery, Honolulu, Hawaii, USA. https://doi.org/10.1145/3613904.3642566 https://hal.science/INRIA2/hal-04475132v1.

[12] JSCAD User Group. 2024. Design guide measurements in JSCAD. https://openjscad.xyz/dokuwiki/doku.php?id=en:design_guide_measurements

[13] Brian Hempel, Justin Lubin, and Ravi Chugh. 2019. Sketch-n-Sketch: Output-Directed Programming for SVG. In *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology (UIST '19)*. Association for Computing Machinery, New York, NY, USA, 281–292. https://doi.org/10.1145/3332165.3347925

[14] R. C. Hillyard and I. C. Braid. 1978. Analysis of dimensions and tolerances in computer-aided mechanical design. *Computer-Aided Design* 10, 3 (May 1978), 161–166. https://doi.org/10.1016/0010-4485(78)90140-9

[15] Christoph M. Hoffmann. 1989. *Geometric and solid modeling: an introduction*. Morgan Kaufmann, San Mateo, Calif.

[16] Nathaniel Hudson, Celena Alcock, and Parmit K. Chilana. 2016. Understanding Newcomers to 3D Printing: Motivations, Workflows, and Barriers of Casual Makers. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems (CHI '16)*. Association for Computing Machinery, New York, NY, USA, 384–396. https://doi.org/10.1145/2858036.2858266

[17] Edwin Hutchins, James Hollan, and Donald Norman. 1985. Direct Manipulation Interfaces. *Human-computer Interaction* 1 (Dec. 1985), 311–338. https://doi.org/10.1207/s15327051hci0104_2

[18] Autodesk Inc. 2024. AutoCAD for Mac & Windows | 2D/3D CAD Software | Autodesk. https://www.autodesk.ca/en/products/autocad/overview

[19] Autodesk Inc. 2024. AutoCAD LT 2024 Help | About Parametric Drawing and Constraints | Autodesk. https://help.autodesk.com/view/ACDLT/2024/ENU/?guid=GUID-899E008D-B422-4DF2-AC8D-1A4F5701ED4E

[20] Autodesk Inc. 2024. Fusion 360 | 3D CAD, CAM, CAE, & PCB Cloud-Based Software | Autodesk. https://www.autodesk.com/products/fusion-360/overview

[21] Autodesk Inc. 2024. Fusion Help | Constraints in sketches | Autodesk. https://help.autodesk.com/view/fusion360/ENU/?guid=SKT-CONSTRAINTS

[22] BlocksCAD Inc. 2023. BlocksCAD. https://www.blockscad3d.com/

[23] R. Joan-Arinyo and A. Soto-Riera. 1999. Combining constructive and equational geometric constraint-solving techniques. *ACM Transactions on Graphics* 18, 1 (1999), 35–55. https://doi.org/10.1145/300776.300780

[24] Chris Johnson. 2023. Computational Making with Twoville. *Journal of Computing Sciences in Colleges* 38, 8 (2023), 39–53.

[25] Matthew Keeter. 2024. Antimony. https://www.mattkeeter.com/projects/antimony/3/

[26] Matt Keeter. 2024. libfive. https://libfive.com/

[27] Marius Kintel. 2024. OpenSCAD. http://openscad.org

[28] Amy J. Ko, Brad A. Myers, and Htet Htet Aung. 2004. Six Learning Barriers in End-User Programming Systems. In *2004 IEEE Symposium on Visual Languages - Human Centric Computing*. Rome, Italy, 199–206. https://doi.org/10.1109/VLHCC.2004.47

[29] Viktor Kovacs. 2024. Online 3D Viewer. https://3dviewer.net

[30] V. C. Lin, D. C. Gossard, and R. A. Light. 1981. Variational geometry in computer-aided design. *ACM SIGGRAPH Computer Graphics* 15, 3 (1981), 171–177. https://doi.org/10.1145/965161.806803

[31] L. Ma, J. Ferguson, M. Roper, and M. Wood. 2011. Investigating and improving the models of programming concepts held by novice programmers. *Computer Science Education* 21, 1 (March 2011), 57–80. https://doi.org/10.1080/08993408.2011.554722 Publisher: Routledge.

[32] LLC MakerBot Industries. 2024. Customizer by MakerBot on Thingiverse - Thingiverse. https://www.thingiverse.com/app:22

[33] LLC MakerBot Industries. 2024. Thingiverse - Digital Designs for Physical Objects. https://www.thingiverse.com/

[34] Aman Mathur, Marcus Pirron, and Damien Zufferey. 2020. Interactive Programming for Parametric CAD. *Computer Graphics Forum* 39, 6 (2020), 408–425. https://doi.org/10.1111/cgf.14046

[35] Michael J. McGuffin and Christopher P. Fuhrman. 2020. Categories and Completeness of Visual Programming and Direct Manipulation. In *Proceedings of the International Conference on Advanced Visual Interfaces (AVI '20)*. Association for Computing Machinery, New York, NY, USA, 1–8. https://doi.org/10.1145/3399715.3399821

[36] Rene K Mueller, Z3 Development, Mark Moissette, and et al. JSCAD developers. 2024. JSCAD - JavaScript CAD. https://openjscad.xyz

[37] Filip Mulier. 2013. Functions that can query a given 3D shape and provide size or location info · Issue #301 · openscad/openscad. https://github.com/openscad/openscad/issues/301

[38] Laird Popkin. 2014. request: object as variable w/ introspection · Issue #954 · openscad/openscad. https://github.com/openscad/openscad/issues/954

[39] Aristides G. Requicha. 1980. Representations for Rigid Solids: Theory, Methods, and Systems. *Comput. Surveys* 12, 4 (1980), 437–464. https://doi.org/10.1145/356827.356833

[40] Jürgen Riegel, Werner Mayer, and et al. FreeCAD Contributors. 2024. Sketcher Micro Tutorial - Constraint Practices - FreeCAD Documentation. https://wiki.freecad.org/Sketcher_Micro_Tutorial_-_Constraint_Practices

[41] Majid Rouhani, Miriam Lillebo, Veronica Farshchian, and Monica Divitini. 2022. Learning to Program: an In-service Teachers' Perspective. In *2022 IEEE Global Engineering Education Conference (EDUCON)*. 123–132. https://doi.org/10.1109/EDUCON52537.2022.9766781 ISSN: 2165-9567.

[42] Himanshu Sekhar Nayak, Sadeep Weerasinghe, Daniel Rossberg, Vidit Jain, and Christopher Sean Morrison. 2024. BRL-CAD: Open Source Solid Modeling. https://brlcad.org/

[43] Irv Shapiro. 2024. MakeWithTech Blog. https://www.makewithtech.com/

[44] Ben Shneiderman. 1983. Direct Manipulation: A Step Beyond Programming Languages. *Computer* 16, 8 (1983), 57–69. https://doi.org/10.1109/MC.1983.1654471

[45] Team SimPy. 2024. Overview — SimPy 4.1.1 documentation. https://simpy.readthedocs.io/en/latest/index.html

[46] Evgeny Stemasov, Tobias Wagner, Jan Gugenheimer, and Enrico Rukzio. 2020. Mix&Match: Towards Omitting Modelling Through In-situ Remixing of Model Repository Artifacts in Mixed Reality. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems (CHI '20)*. Association for Computing Machinery, New York, NY, USA, 1–12. https://doi.org/10.1145/3313831.3376839

[47] Abdullah Tahiriyo, Alexander Golubev (Fat-Zer), Bernd Hahnbach, and et al. FreeCAD Contributors. 2024. FreeCAD: Your own 3D parametric modeler. https://www.freecadweb.org/

[48] Adam Urbanczyk, Jeremy Wright, Marcus Boyd, and et al. CadQuery developers. 2024. CadQuery. https://github.com/CadQuery/cadquery

[49] Tom Veuskens, Florian Heller, and Raf Ramakers. 2021. CODA: A Design Assistant to Facilitate Specifying Constraints and Parametric Behavior in CAD Models. (2021), 10 pages, 877. https://doi.org/10.20380/GI2021.11

[50] Bret Victor. 2013. Bret Victor - The Future of Programming. https://vimeo.com/71278954

[51] Catherine G. Wolf and James R. Rhyne. 1987. A Taxonomic Approach to Understanding Direct Manipulation. *Proceedings of the Human Factors Society Annual Meeting* 31, 5 (Sept. 1987), 576–580. https://doi.org/10.1177/154193128703100522 Publisher: SAGE Publications.

[52] Qiang Zou, Zhihong Tang, Hsi-Yung Feng, Shuming Gao, Chenchu Zhou, and Yusheng Liu. 2022. A review on geometric constraint solving. https://doi.org/10.48550/arXiv.2202.13795

A   APPENDIX

Table 4: Description of Handles for Primitive Shapes.
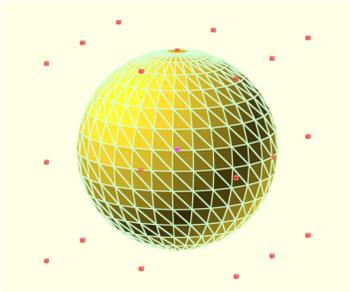
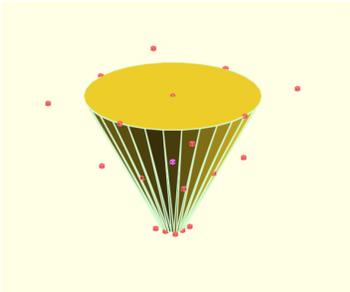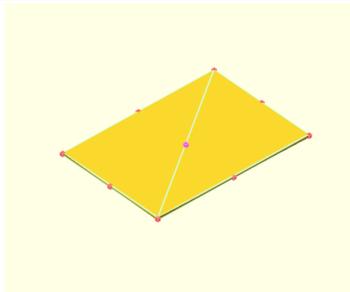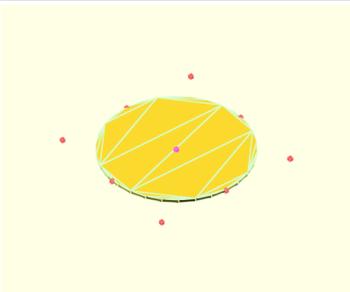|	
Cube (27 points): Cube nodes create a grid of 3x3x3 points, including 1 point in the center, 1 at each corner (8 points), 1 in the center of each face (6 points), and 1 in the middle of each edge (12 points).	
Sphere (27 points): Spheres create a boundary cube using the diameter as the size for height, width and depth. The node places handles on the boundary cube following the same distribution as the cube nodes.	
Cylinder (27 points): Cylinder nodes form a boundary cuboid, with its bottom and top square faces sized by the cylinder parameters d1 and d2, respectively. The height of the cuboid is determined by h. Handles are positioned on the cuboid's boundary, following the cube nodes' distribution pattern.	
Square (9 points): Square nodes create a grid of 3x3 points, including 1 point in the center, 1 at each corner (4 points), and 1 at the middle of each edge (4 points).	
Circle (9 points): Circle nodes create a boundary square using the diameter as the size of the height and width. 1 point in the center and 1 at each extreme along each axis (4 points).	
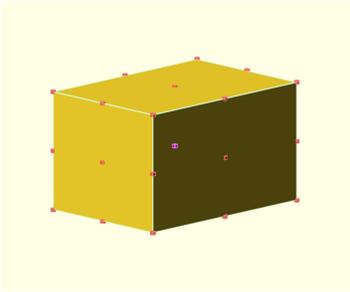
(i) Multi-Color Pencil Cup Remix, Thing:6291495

(ii) Master Lock M1 Key, Thing:6289846

(iii) Gridfinity Kcup Holder, Thing:6284181

(iv) Kumihimo Disk, Thing:6290477

(v) Spare Peg For Ikea Hammer, Thing:6287601

(vi) Customized Kettle Whistle, Thing:6291759

(vii) Hook On A Plate, Thing:6287141

(viii) Turret Cap Generator, Thing:6292455

(ix) Cable Hook, Thing:6288817

(x) Door Hook 1 Angle, Thing:6287795

(xi) Parametric Porch Hook, Thing:6286541

(xii) Filament Spool Holder Frame, Thing:6255969

(xiii) Battery End Caps, Thing:6248029

(xiv) Small Customizable Box, Thing:6266913

(xv) Infinity Cube Customizer, Thing:6249758

(xvi) Customizable Key Tag, Thing:6249968

(xvii) Halloween Spider, Thing:6253273

(xviii) Filter Basket For Fluval Evo, Thing:6267303

(xix) Halloween Candle, Thing:6267335

(xx) Modular Drawer Divider, Thing:6250410

(xxi) Twist-Lock Hose Flange, Thing:5988719

(xxii) Knurled Screw Lid Container, Thing:6095952

(xxiii) 3D Snowflake Ornament, Thing:5673707

(xxiv) Shotgun Shell Case, Thing:6153068

(xxv) Customizable Jar/Bottle, Thing:6211287

(xxvi) Parametrizable Rugged Box, Thing:5983067

(xxvii) Customizable Cable Tie, Thing:5789087

(xxviii) Phone Mount, Thing:5816088

(xxix) Folding Utility Knife, Thing:6117454

(xxx) Halloween Jack-O'-Lantern, Thing:6221369

Figure 11: OpenSCAD models taken from Thingiverse for the formative study.

Supplemental Material for "Facilitating the Parametric Definition of Geometric Properties in Programming-Based CAD"

J. Felipe Gonzalez
Carleton University
Univ. Lille, CNRS, Inria, Centrale Lille, UMR 9189 CRIStAL
Lille, France
johannavila@cmail.carleton.ca

Thomas Pietrzak
Univ. Lille, CNRS, Inria, Centrale Lille, UMR 9189 CRIStAL
Lille, France
thomas.pietrzak@univ-lille.fr

Audrey Girouard
Carleton University
Ottawa, ON, Canada
audrey.girouard@carleton.ca

Géry Casiez*
Univ. Lille, CNRS, Inria, Centrale Lille, UMR 9189 CRIStAL
Lille, France
gery.casiez@univ-lille.fr

We expand on the details of our user study. We first present comprehensive demographic information of the participants. Then, we present the extended observations of the experiment.

1 DEMOGRAPHIC INFORMATION

We report the demographics of the participants and their previous experience with CAD applications in Table 1. All participants self-identified as male and varied in age: one was between 20 and 29, three were between 30 and 39, four were between 40 and 49, one was between 50 and 59, and two were between 60 and 69 (average: 44.5, standard deviation: 14.2). All participants, except P3, had four or more years of 3D modeling experience (average: 8.9y, standard deviation: 5.8). Except for P3, all participants self-rated with four or more in at least one programming language. All participants except P4 and P7 had experience with other CAD applications, but only P3, P6, P8, P9, and P10 self-rated with 3 or more at least one of them. Finally, participants self-rated their skill level with OpenSCAD as follows: Two participants with 2, four participants with 3, four participants with 4, and one participant with 5.

2 AUTHORING STRATEGIES IN PARAMETRIC DESIGN IN PROGRAMMING-BASED CAD

We analyzed the design process of participants creating parametric designs in the original version of OpenSCAD. We detail our findings on authoring strategies related to common behaviors in the design process, common errors, and strategies.

Table 1: Demographics and self-rated skill level in CAD applications and programming languages. Participants self-rated their skill level on the scale: 1 (Novice), 2 (Advanced Beginner), 3 (Competent), 4 (Proficient), 5 (Expert). The level reported in the category *Other Applications* and *Programming languages* is the highest rank expressed by the participant.

Participant	Age Range	3D Modeling Experience (y)	OpenSCAD	Other applications	Programming languages
P1	40-49	8	3	FreeCAD (1)	Python,PHP (5)
P2	60-69	9	5	Rhino (2)	C++ (5)
P3	60-69	2	3	Fusion360 (3)	Python(1)
P4	50-59	10	3		Python, C, JS (5)
P5	30-39	18	4	Fusion360 (1)	Javascript (4)
P6	30-39	20	2	Rhino (4)	C++ (3)
P7	40-49	12	4		Python, C, C#, JS (5)
P8	30-39	5	4	Fusion360 (4)	Python, C++, JS (4)
P9	20-29	4	3	FreeCAD (4)	Python (4)
P10	40-49	5	4	TinkerCAD (4)	Python, C++ (3)
P11	40-49	5	2	AudtoCAD (3)	C++ (5)

2.1 Parametric design

The approaches of the participants to creating parametric designs in OpenSCAD were analyzed through screen recordings, revealing a significant anticipatory mental process. Initially, all participants focused on setting up parameters, with some attempting to anticipate all necessary parameters for the entire model, while others concentrated on parameters for specific sections, revisiting the creation process as needed. Regardless of the approach, additional parameters were often introduced as the design progressed.

All participants mentioned they would try to define the geometric properties parametrically when possible. Indeed, none of the geometric properties were defined with raw numbers, and participants frequently asked about the relationship of an object with others to better generalize the definition of the geometric properties. For instance, common questions were related to the position of the spheres in model A in relation to the cup height or the position of the windows in the box of model B.

---

*Also with Institut Universitaire de France.

All participants needed to play with the parameter values at some point because the initial ones did not allow them to have a good preview with correct proportions.

2.2 Design style

Participants split the design into sub-parts and designed them separately, although the way to split the design varied. For example, in the case of model B, some participants divided the design in two: the box and the legs. Nine participants opted to design the box as a cube with subtracted parts. However, P2 and P10 created the box as a union of different walls.

Three participants followed the strategy of creating subparts at the origin and then removing them from the scene by commenting on the code statement to continue with the next subpart. When all subparts were completed, the participants started by placing the bottom part and used `translate` statements to place each part on top of the other. The rest of the participants created the designs cumulatively without removing elements.

2.3 Defining positions

In creating parametric designs using OpenSCAD, participants exhibited a consistent common methodology and encountered specific challenges. Initially, they aimed to mentally locate the center of the object's coordinate system, accounting for any preceding spatial transformations and the object's center. Then, axis by axis, participants started to find the required translation based on the existing variables.

For example, consider the base of model A. Initially, some participants constructed a cylinder to serve as the base. Then, this cylinder was enclosed within a `difference` block to incorporate cube-shaped cutouts along its edges. Initially, a cube geometry not centered (*i.e.*, with the parameter `center` set to `false`)—is created. Subsequently, a `translate` transformation is applied to position this cutout, typically along the positive 'x' axis, following the syntax's axis order. The process involves initially positioning the cube's moving center at the origin by adding the cylinder's radius (or half its diameter) to align it with the cylinder's edge. Then, participants subtract half of the cube's x-dimension to embed the cube halfway into the cylinder. A similar approach is taken for the 'y' axis, while for 'z,' the cylinder's height is subtracted.

This process involves identifying the object's center being manipulated, adjusting its position relative to other components in the design by considering their dimensions, and iteratively applying addition or subtraction as needed. Common errors encountered in this process include:

(1) Neglecting the specific center that the `translate` operation targets. Spatial transformations might use a center that varies based on the geometric center of the object. For example, `cube(center = true)` centers the cube's geometry at the origin, while `cube()` or `cube(center = false)` places the cube's corner at the origin. This requires considering an offset for translations and possibly adjusting the rotation axis when rotating for centered versus non-centered objects.

(2) Misinterpreting the multipliers needed for positioning. For instance, participants occasionally miscalculate the offset using a quarter instead of half of the object's dimension, leading to trial and error to get the desired visual outcome.

(3) Misinterpreting the coordinate system's orientation and mistakenly applying the wrong sign to variables. Participants accurately identified the necessary variables and their multiplier factors but occasionally hesitated on the sign, adding when they should subtract, indicating a disconnection between spatial conceptualization and code expression.

(4) Confusing variables, particularly in complex expressions involving multiple variables, lead to the inclusion of incorrect variables in computations.

(5) Difficulties in mathematically deriving complex expressions when dealing with subtracted elements not visually represented in the model. To counteract this, some participants temporarily removed elements from `difference` blocks for verification or used modifiers for visual guidance, reintegrating elements when satisfied.

Strategies to address these errors typically involved trial and error with variable factors, signs, and meticulous code examination to ensure accurate expression definition. Positioning subtracted elements posed a significant challenge due to the lack of visual representation. To address this issue, participants rendered parts outside the `difference` or `intersection` statements. They also used the OpenSCAD modifiers, special characters for debugging, to make visible subtracted elements.

2.4 Ensuring overlapping

Another common strategy involved creating a variable with a minimal value to ensure necessary overlap. OpenSCAD provides a preview mode with a fast rendering but slightly less accurate. Participants frequently utilized preview mode in OpenSCAD due to its speed advantage. However, this mode demonstrates limitations in CSG expression. Given the abstract nature of CSG definitions, transitioning to geometric representation can exhibit unintended behaviors in preview mode. Specifically, when elements theoretically align perfectly, their visual representation might not clearly depict this coincidence. For instance, in scenarios where two cubes should intersect on a face, the application may fail to execute the intended subtraction if they are precisely coincident. Participants introduced a "delta" variable to bypass this issue, slightly enlarging the elements to ensure overlapping. This adjustment ensures that the geometric calculations accurately reflect the desired behavior, addressing the preview mode's imperfections.