

FACILITATING PROGRAMMING-BASED 3D COMPUTER-AIDED DESIGN USING BIDIRECTIONAL PROGRAMMING

FACILITER LA CONCEPTION ASSISTÉE PAR ORDINATEUR 3D BASÉE SUR LA PROGRAMMATION EN UTILISANT LA PROGRAMMATION BIDIRECTIONNELLE

Doctoral dissertation in Information Technology, with a
specialty in Human-Computer Interaction. Defended
on April 11th 2024 by

*Thèse de doctorat en informatique, spécialisée en Interaction
Humain-Machine. Soutenue le 11 Avril 2024 par*

JOHANN FELIPE GONZALEZ AVILA

To obtain the titles of **Docteur** from the Université de
Lille and **Doctor of Philosophy** from Carleton
University.

*Pour obtenir le titre de **Docteur** de l'Université de Lille et le
titre de **Doctor of Philosophy** de l'Université Carleton.*

Jury committee

Jury de soutenance

President <i>Président</i>	Anastasia Bezerianos	Professeure à Université Paris-Saclay
External examiners <i>Rapporteurs</i>	Michael J. McGuffin Kris Luyten	Professeur à École de Technologie Supérieure (Montréal) Professor at Hasselt University
Internal examiners <i>Examineurs</i>	Hamza Bashady Steven Pong	Assistant Professor at Carleton University (Ottawa) Assistant Professor at Carleton University (Ottawa)
Supervisors <i>Directeurs</i>	Audrey Girouard Géry Casiez	Professor at Carleton University (Ottawa) Professeur à Université de Lille
Supervisor <i>Encadrant</i>	Thomas Pietrzak	Maître de conférences à Université de Lille



Johann Felipe Gonzalez Avila: Facilitating Programming-Based 3D Computer-Aided Design Using Bidirectional Programming , Doctoral dissertation, Doctor of Philosophy in Information Technologies, © April 2024

Johann Felipe Gonzalez Avila: Faciliter la Conception Assistée par Ordinateur 3D Basée sur la Programmation en Utilisant la Programmation Bidirectionnelle , Thèse de doctorat, Docteur en philosophie en technologies de l'information, © Avril 2024

ABSTRACT

3D Computer-Aided Design (CAD) applications allow users to create visual representations of models, helping create, edit, test, and analyze designs. Most offer a Graphical User Interface (GUI) with direct manipulation providing easy-to-use interactions, while a less popular category adopts a programming-based approach requiring users to describe models using specific programming languages. Programming-based CAD applications provide multiple benefits to 3D design, but their use remains limited, potentially due to higher entry barriers and extensive programming requirements. Regrettably, a profound lack of understanding of the challenges faced by users of programming-based CAD applications prevents a clear comprehension of the issues of these applications. Furthermore, research addressing CAD challenges has predominantly focused on applications that provide direct manipulation interactions.

This doctoral thesis aims to improve the usability of programming-based CAD applications, focusing on their role in Personal Digital Fabrication with 3D printers. Our research seeks to understand and address programming-based CAD users' challenges during the design process. In our first study, we interviewed twenty OpenSCAD users, a leading programming-based CAD application in the 3D printing community. Data analysis via a Reflexive Thematic Analysis (RTA) led to the development of a comprehensive codebook categorizing three main themes: user profiles, 3D design challenges, and 3D printing challenges.

Our second study addressed the identified design challenges in linking 3D views with code and difficulties in performing spatial transformations on the model. We proposed to address these difficulties by introducing the concept of bidirectional programming in programming-based CAD, allowing users to interact with both the code and the view. We modified the source code of OpenSCAD to implement this approach, developing bidirectional navigation features and allowing users to edit the model by interacting with the view while the application updates the code coherently.

The third study addressed the keystone challenge of defining geometric properties in parametric designs. After analyzing thirty OpenSCAD models, we developed bidirectional programming features in OpenSCAD to facilitate the definition of parametric properties, directly extracting information from the view to use in the code. Experimentation with OpenSCAD users showed our solution may streamline design, reduce errors, and lower entry barriers for newcomers.

RÉSUMÉ

Les applications de Conception Assistée par Ordinateur (CAO) 3D permettent aux utilisateurs de créer des représentations visuelles de modèles, aidant à créer, éditer, tester et analyser des conceptions. La plupart offrent une Interface Graphique Utilisateur (GUI) avec manipulation directe, fournissant des interactions faciles à utiliser, tandis qu'une catégorie moins populaire adopte une approche basée sur la programmation, nécessitant des utilisateurs de décrire les modèles en utilisant des langages de programmation spécifiques. Les applications de CAO basées sur la programmation offrent de multiples avantages pour la conception 3D, mais leur utilisation reste limitée, potentiellement en raison de barrières d'entrée plus élevées et d'exigences de programmation étendues. Malheureusement, un manque de compréhension profond des défis rencontrés par les utilisateurs des applications de CAO basées sur la programmation empêche une compréhension claire des problèmes de ces applications. De plus, la recherche traitant des défis de la CAO s'est principalement concentrée sur les applications offrant des interactions de manipulation directe.

Cette thèse doctorale vise à améliorer l'utilisabilité des applications de CAO basées sur la programmation, en se concentrant sur leur rôle dans la Fabrication Numérique Personnelle avec des imprimantes 3D. Notre recherche cherche à comprendre et à relever les défis des utilisateurs de CAO basée sur la programmation pendant le processus de conception. Dans notre première étude, nous avons interviewé vingt utilisateurs d'OpenSCAD, une application de CAO basée sur la programmation leader dans la communauté de l'impression 3D. L'analyse des données via une Analyse Thématique Réflexive RTA a conduit au développement d'un codebook complet catégorisant trois thèmes principaux : les profils d'utilisateurs, les défis de conception 3D et les défis d'impression 3D.

Notre deuxième étude a abordé les défis de conception identifiés dans la liaison des vues 3D avec le code et les difficultés à effectuer des transformations spatiales sur le modèle. Nous avons proposé de relever ces difficultés en introduisant le concept de programmation bidirectionnelle dans la CAO basée sur la programmation, permettant aux utilisateurs d'interagir à la fois avec le code et la vue. Nous avons modifié le code source d'OpenSCAD pour implémenter cette approche, développant des fonctionnalités de navigation bidirectionnelle et permettant aux utilisateurs de modifier le modèle en interagissant avec la vue tandis que l'application met à jour le code de manière cohérente.

La troisième étude a abordé le défi clé de la définition des propriétés géométriques dans les conceptions paramétriques. Après avoir analysé 30 modèles OpenSCAD, nous avons développé des fonctionnalités de programmation bidirectionnelle dans Open-

SCAD pour faciliter la définition des propriétés paramétriques, extrayant directement les informations de la vue pour les utiliser dans le code. L'expérimentation avec les utilisateurs d'OpenSCAD a montré que notre solution pourrait rationaliser la conception, réduire les erreurs et abaisser les barrières à l'entrée pour les nouveaux utilisateurs.

PUBLICATIONS

- [1] Gonzalez, J.F., Kieken, D., Pietrzak, T., Girouard, A. and Casiez, G. 2023. Introducing Bidirectional Programming in Constructive Solid Geometry-Based CAD. Proceedings of the 2023 ACM Symposium on Spatial User Interaction (Sydney, Australia, Oct. 2023), 1–12.
- [2] Gonzalez, J.F., Pietrzak, T., Girouard, A. and Casiez, G. 2024. Understanding the Challenges of OpenSCAD Users for 3D Printing. Manuscript accepted for publication in proceedings of the 2024 CHI Conference on Human Factors in Computing Systems (Hawaii, USA, May. 2024).

*If we knew what it was we were doing,
it would not be called research, would it?*

— Albert Einstein

ACKNOWLEDGEMENTS

Agradezco de manera profunda a mis padres, Luis y Leonor. Su dedicación incansable ha sido el motor que ha impulsado nuestras vidas, brindándonos todo sin reservas, tanto con su palabra cálida y alentadora como con su firme y educador llamado de atención. Mi dedicación es solo una muestra de lo que ustedes sembraron en mí. A Jenny, has sido esa compañía incondicional siempre recordándome que "sí se puede" y "no estoy solo" Tu consejo y tus palabras fueron el apoyo constante que siempre me impulsó hacia adelante. Este triunfo también es tuyo. Gracias a toda mi familia, a mis hermanos Diego y Julián, quienes han cultivado en mí un sentido crítico y la capacidad de defender mis convicciones. A mi hermanita Valeria, eres la personificación de la inocencia pura que ha cultivado en mí un deseo de empatía único. Agradezco también el apoyo de mi tía Nohora y mi prima Tatiana. Este logro no solo es el resultado de mi esfuerzo, sino también de ustedes, quienes me han brindado su ánimo incondicional.

To my supervisors, I extend my heartfelt gratitude. Audrey, your support and the opportunity you provided for me to embark on this incredible journey have been invaluable. I am deeply thankful for your trust and advice. Géry and Thomas, your mentorship and guidance at every step of my Ph.D. journey have been indispensable. I am grateful for all I have learned from you over these years. Thank you for your patience and unwavering support. Of course, my acknowledgments must be lengthy, but you know that being short on words is not exactly my style.

I also want to express my appreciation for the laboratories at CIL in Carleton. Although the pandemic limited our time together, I cherished every moment I shared with my colleagues. Thanks to Leo for your invaluable advice and feedback; your openness and willingness to help have always been palpable.

To the LOKI team at INRIA, my time in France was memorable (and yes, I acknowledge I could have spoken French more frequently). I thank Danny, Sylvain, Rahul, Bruno, Mathieu, Damien, Aurélien, Travis, Grégoire, Axel, Alice, and Suliac for their camaraderie.

Special thanks to my colleagues Eva, Raphaël, and Constant; I am grateful to have met you and to have you as my colleagues in both work and drinks. I extend my gratitude and best wishes to you all.

Finally, I thank everyone who supported me on this extensive journey. Your support has been a cornerstone of this success. Thank you.

CONTENTS

1	INTRODUCTION	1
1.1	Research Outline	6
1.2	Research contributions	8
1.3	Document structure	8
2	STATE OF THE ART	11
2.1	Computer-Aided Design (CAD)	11
2.1.1	Interactive Approach	12
2.1.2	Construction Mode	26
2.1.3	Data representation	28
2.1.4	OpenSCAD	30
2.2	Digital Personal Fabrication and 3D printers	31
2.2.1	Designing from scratch	32
2.2.2	Sharing and re-using models.	33
3	UNDERSTANDING PROGRAMMING-BASED CAD USERS	35
3.1	Method	36
3.1.1	Recruitment and Participants	37
3.1.2	Data Analysis	39
3.2	Themes	39
3.2.1	Programming-based CAD users profile	40
3.2.2	Design	41
3.2.3	Fabrication	55
3.3	Discussion	56
3.3.1	Programming-based CAD users in 3D printing	57
3.3.2	Programming-based CAD design challenges	58
3.3.3	3D printing challenges.	61
3.4	Limitations	62
3.5	Conclusion	62
4	NAVIGATION AND SPATIAL EDITING CHALLENGES	65
4.1	Specific Related Work	66
4.1.1	Bidirectionnal Navigation	67
4.1.2	Navigation in OpenSCAD	67
4.2	Design	71
4.2.1	Initial exploration	71
4.2.2	Design goals	72
4.3	Bidirectional programming for programming-based CAD	73
4.3.1	Reverse Search Navigation	74
4.3.2	Forward Search Navigation	76

4.3.3	Transformations with direct manipulation	77
4.3.4	Informal validation by example	80
4.4	Discussion	82
4.5	Conclusion	85
5	PARAMETRIC DEFINITION OF GEOMETRIC PROPERTIES	87
5.1	Specific Related Work	91
5.1.1	Parametric design in direct manipulation	91
5.1.2	Parametric design in programming-based	93
5.2	Method	94
5.2.1	Formative study	94
5.2.2	Design goals	96
5.3	Bidirectionnal programming to define geometric properties	97
5.3.1	Absolute location	100
5.3.2	Relative location	101
5.4	User study	103
5.4.1	Recruitment and Participants	103
5.4.2	Design tasks	104
5.4.3	Data collection	106
5.4.4	Data analysis	106
5.5	Discussion	113
5.6	Limitations	115
5.7	Conclusion	115
6	CONCLUSION	117
6.1	Research contribution summary	117
6.2	Future work	119
6.2.1	Other Explorations	119
6.2.2	Research gaps	123
I	APPENDIX	127
A	APPENDIX A	129
B	APPENDIX B	135
	BIBLIOGRAPHY	137

LIST OF FIGURES

Figure 1	CAD application examples	2
Figure 2	Digital Personal Fabrication process	4
Figure 3	Rankings of CAD applications by popularity	5
Figure 4	Summary of thesis structure	7
Figure 5	CAD application examples	11
Figure 6	First CAD application: Sketchpad	13
Figure 7	Direct manipulation example	14
Figure 8	Programming-based CAD application OpenSCAD	18
Figure 9	Programming-based CAD application examples	19
Figure 10	Visual programming-based CAD application examples	20
Figure 11	Programming By Example example in CAD applications	22
Figure 12	SVG - Bidirectional Programming example	25
Figure 13	Bidirectionnal Visual Programming example in CAD applications	26
Figure 14	Parametric CAD examples	27
Figure 15	CSG tree representation	29
Figure 16	BREP and CSG comparisson	30
Figure 17	Introducing bidirectionnal programming in programming-based CAD applications	66
Figure 18	Approaches of bidirectionnal navigation in programming-based CAD	68
Figure 19	Navigation feature in OpenSCAD	69
Figure 20	CAD example from thingiverse : Battery box	74
Figure 21	Reverse search features part 1	75
Figure 22	Reverse search features part 2	77
Figure 23	Simple editing features	79
Figure 24	Informal validation example part 1	80
Figure 25	Informal validation example part 2	81
Figure 26	Informal validation example part 3	81
Figure 27	Informal validation example part 4	82
Figure 28	Informal validation example part 5	82
Figure 29	Thingiverse parametric model example	88
Figure 30	Thingiverse model example	89
Figure 31	Example of control points	99
Figure 32	Example preview of a cup in progress.	100
Figure 33	Absolute location feature example	101
Figure 34	Relative location feature example	102

Figure 35	Proposed models for the experiment	105
Figure 36	Participants' answers on experiment models' difficulty	109
Figure 37	Participants answers of the contribution of the modified version of OpenSCAD in the design task	110
Figure 38	Participants answers on the difficulties of using the implemented features	111
Figure 39	General comparison between the original and modified versions of OpenSCAD	112
Figure 40	Results of using features in OpenSCAD	113
Figure 41	OpenSCAD models taken from Thingiverse for the formative study	136

LIST OF TABLES

Table 1	Understanding users' challenges - Participants demographic information	38
Table 2	Understanding programing-based challenges theme - User profile	40
Table 3	Understanding programing-based challenges theme - Design .	42
Table 4	Understanding programing-based challenges theme - Printing .	55
Table 5	Modifiers in OpenSCAD	70
Table 6	Common Geometric and Dimensional Constraints in CAD . . .	92
Table 7	Categories used to classify expressions of OpenSCAD models .	96
Table 8	Formative study results	96
Table 9	Description of Control Points for Primitive Shapes	98
Table 10	Parametric Definition of Geometric Properties - Participants demographic information	104

LISTINGS

Listing 1	CadQuery Example	24
Listing 2	OpenSCAD Example	69
Listing 3	Example of parametric definition of a translate in OpenSCAD .	89
Listing 4	OpenSCAD example. Parametric model of a cube on top of another cube	94
Listing 5	Control points example	99
Listing 6	Example before using absolute location feature	100
Listing 7	Example after using absolute location feature	101

ACRONYMS

CAD	Computer-Aided Design
CAO	Conception Assistée par Ordinateur
CSG	Constructive Solid Geometry
BREP	Boundary Representation
COP	Content-Oriented Programming
COVP	Content-Oriented Visual Programming
PBE	Programming By Example
GUI	Graphical User Interface
RTA	Reflexive Thematic Analysis
STL	STereoLithography
SVG	Scalable Vector Graphics
DOFs	Degrees-of-freedom
API	Application Programming Interface
CAutoD	Computer-Automated Design
CNC	Computer-Numeric Controlled
CRT	Cathode-Ray Tube
AST	Abstract Syntax Tree
TNP	Topological Naming Problem
DIY	Do-It-Yourself
PCB	printed circuit board

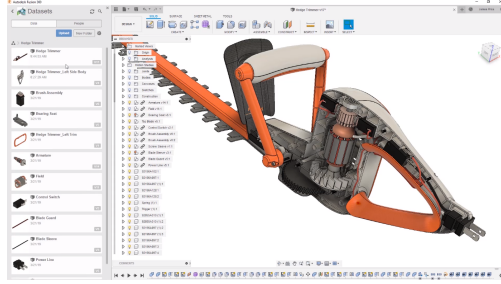
INTRODUCTION

Computer-Aided Design (**CAD**) refers to an iterative problem-solving approach that involves graphic design in a collaborative interaction between humans and machines. This approach enables users to digitally and visually *model* problems, providing a means to understand and sometimes verify results through digital simulations. **CAD** applications are designed to assist users in articulating and refining detailed problem statements, thereby facilitating the overall problem-solving process [174]. **CAD** applications find extensive use across various industries [92], including architecture and construction [16, 80], automotive [5, 195], aerospace [49, 177], electronics [21, 207], mechanical engineering [48, 152], product design and manufacturing [161, 198], gaming and entertainment [17, 140], medical devices [42, 93], fashion and apparel [28, 73], interior design [76, 216], or oil and gas [81, 96]. The market for these applications is valued in billions of dollars [202], as they accelerate product design, improve quality, facilitate drafting, and enhance documentation [55].

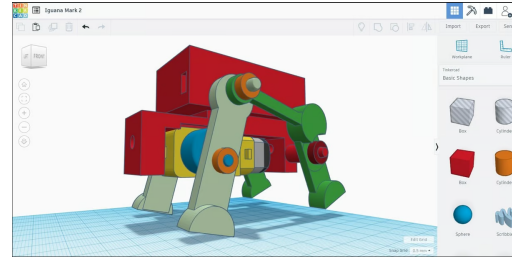
CAD applications provide users with a language for modeling problems as sets of objects with defined relationships. The machine interprets these models to solve problems based on user specifications. The process typically follows an “outside-in” dynamic [174], where users start with broad ideas, and the modeling process reveals nuanced details, leading to a clearer understanding and articulation of the problem, allowing users to refine the model. For instance, OrCAD [207] is an application tailored for circuit simulation and printed circuit board (**PCB**) design. Users can assemble representations of electronic components using graphical icons within a viewer and establish connections to create electronic schematics. The application provides frameworks for electronic components, allowing users to connect them and simulate behaviors to verify the output. When users find errors in the simulations, they modify the model to fix the problems and re-run the simulations. Upon completion of the modeling process, users can export the corresponding **PCB** files for fabrication.

Given the highly interactive nature of **CAD** applications, the communication means and the interactive paradigms that these applications offer become crucial. The initial **CAD** applications [56, 206] allowed the creation of 2D drawings, akin to blueprints, with graphical input instead of the classical text-based input for computers of that era. Most **CAD** applications have since followed a similar interactive paradigm. The **Direct manipulation** [192] approach enables users to interact directly with the graphical representation of the output, modifying it through simple metaphors like drag-and-drop interactions, menus, and buttons. This approach provides immediate feedback, incremental and reversible operations [191, 192], resulting in a rapid learning curve [189].

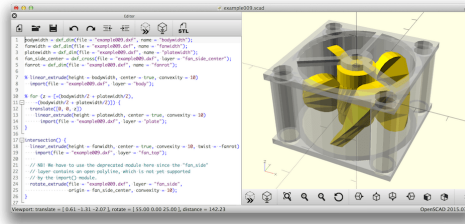
However, the direct manipulation paradigm has known limitations, such as difficulties in performing repetitive actions, manipulating objects in high-density part models, or ambiguity in resolving user intent [65, 117, 138]. For example, Autodesk Fusion360 [93] (Figure 1a) is a **CAD** application providing direct manipulation interactions, enabling users to create 3D models, simulate designs, and generate toolpaths for machining processes, including computer-aided engineering (CAE) and computer-aided manufacturing (CAM) features. Autodesk Tinkercad [15] (Figure 1b) is an easy-to-use, web-based 3D **CAD** application designed for beginners, educators, and hobbyists who want to create 3D models without the complexity of traditional **CAD** applications.



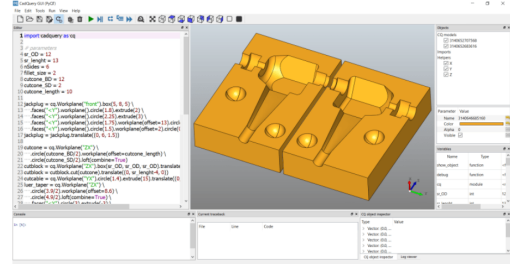
(a) Fusion360 is a **CAD** application that implements direct manipulation used for engineering and manufacturing projects.



(b) Tinkercad, an easy-to-use, web-based **CAD** application that implements direct manipulation.



(c) OpenSCAD is a programming-based **CAD** application that provides a functional language to describe 3D models.



(d) CadQuery is a python library of programming-based modeling.

Figure 1: Fusion360 and Tinkercad provides direct manipulation interactions while OpenSCAD and CadQuery are programming-based technologies ¹.

A less popular category of **CAD** applications uses a **Programming-based** approach. This method allows users to create models by writing code in a text editor using a specific programming language. Programming brings valuable advantages to 3D design, enabling easy automation of repetitive actions using programmatic structures like conditionals and loops. Additionally, programming allows for creating complex structured geometries, such as fractals or trees, using techniques like recursion. Furthermore, programming provides a more suitable environment for using mathematical formulas and abstractions. Finally, current version control platforms enable better

sharing of coded models [229]. OpenSCAD [160] is a free and open-source software application for creating solid 3D CAD, providing a functional programming language to describe geometries that render in a view, as depicted in Figure 1c. CadQuery is a Python library for parametric 3D modeling that allows the creation complex geometries through scripts (Figure 1d).

Despite the potential advantages, the adoption of programming-based CAD applications remains limited (Figure 3a), possibly attributed to the associated learning curve in programming that requires extended training periods and repetitive practices [111]. However, the specific challenges faced by users utilizing CAD applications based on programming are not well-defined. While previous research has identified programming barriers in general environments [111], little attention has been given to the domain of 3D modeling using programming-based CAD applications [105, 229].

Within various sectors and industries leveraging CAD, programming-based applications have demonstrated significant engagement in the practice referred to as **Digital Personal Fabrication**. This process involves creating objects through digital tools like laser cutters, Computer-Numeric Controlled (CNC) mills, or 3D printers [13, 149]. Among these, 3D printers are gaining popularity due to their affordability and ease of use, allowing makers of all skill levels to create precise geometric objects. These tools create geometric objects by successive addition of material in layers [91]. Compared to other fabrication options, 3D printers produce less waste and are more accessible to a wider audience, making them an increasingly popular choice [149]. 3D printing contributes to modeling and prototyping in industries such as automotive, aerospace, and medical [98], forcing a redesign of the production chain in the industry [86, 139] in a billionaire market [214]. In research, 3D printing has also contributed to different fields such as sports [209], accessibility and prostheses [30, 31, 162], furniture fabrication [113, 121], health [142], or robotics [172].

Throughout the digital personal fabrication process, the *maker* undergoes iterative stages of *Ideation*, *Design*, *Print*, and *Validation* [88, 217], as depicted in Figure 2. Design, a pivotal stage, requires the user to obtain a digital object model stored in a format readable by the printer, such as the STereoLithography (STL) file [43]. Makers can create models from scratch using a CAD application or download designs from repositories like Thingiverse [213] or Printables [235]. Downloaded models may require further modifications to get a customized version, which makers can achieve by uploading the model in a CAD application and editing it. Either for creating or modifying, the maker must go through a CAD modeling application to produce a model.

In the context of 3D printing, the modeling process goes beyond mere utilization of a CAD application. It involves diverse tasks, such as seeking inspiration, sketching,

¹ Images taken from Fusion360: <https://help.autodesk.com/view/fusion360/ENU/?guid=GUID-27Do89Co-5FC5-4AD4-841F-6E983AC99DCF>, TinkerCAD: <https://www.instructables.com/Tinkercad-Robotics-for-School-Create-TWO-Walking-M/>, OpenSCAD: <https://openscad.org/>, CadQuery: <https://snapcraft.io/install/cadquery-editor/debian>. Accessed on 12/01/2024

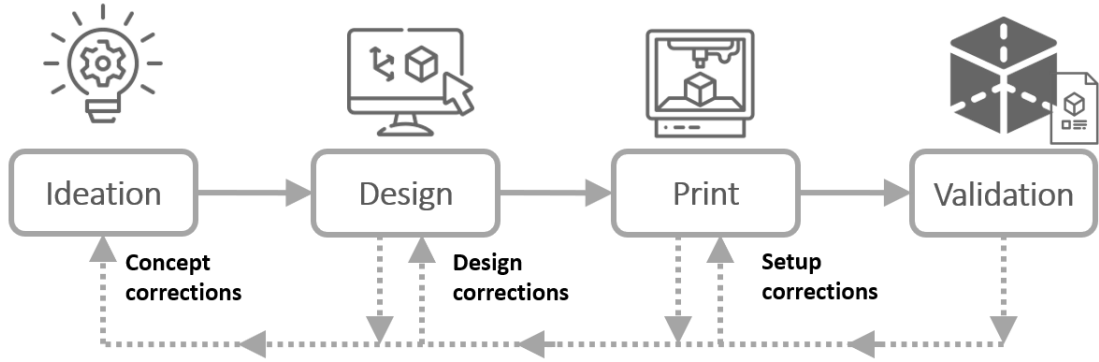


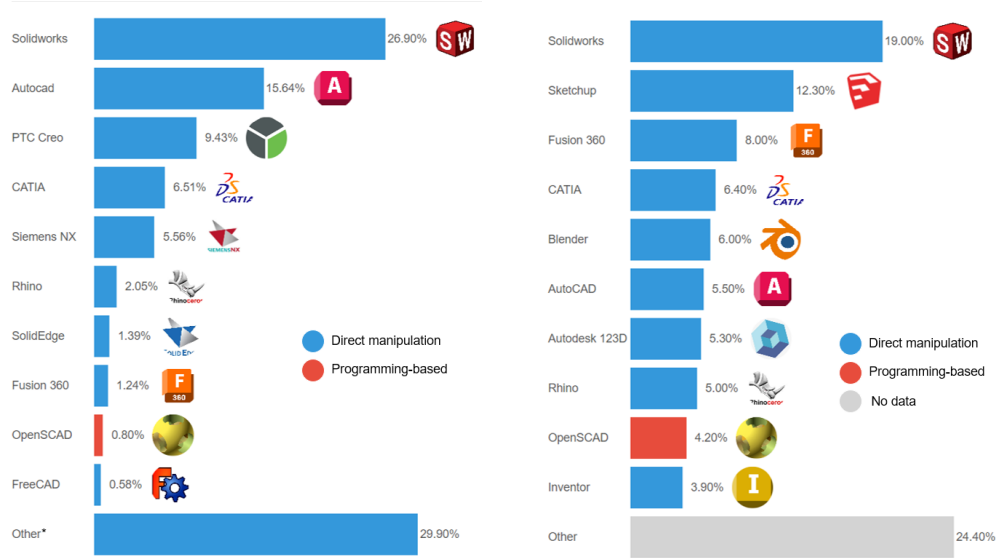
Figure 2: A simplified version based on [88] of a common workflow of the Digital Personal Fabrication process with 3D printing.

measuring physical objects, transferring data to design tools, incorporating physical objects into designs, interpreting pre-existing models, considering printer capabilities, and modifying models based on visual inspection of printed objects [13, 88]. The choice of CAD technology significantly impacts the user experience and the complexity of these tasks. For example, as discussed later, creating a cube with rounded corners may be effortlessly accomplished in some applications while posing a considerable challenge in others.

Figure 3b presents the ranking of CAD applications for 3D printing by the number of users in the 3D Hubs community [148], a global network of manufacturing partners. Although the data is from 2016, it offers a valuable snapshot of the trending CAD applications due to the limited information available on the Internet. Similar to the rankings of CAD applications in the general landscape (Figure 3a), the majority of CAD applications in the 3D printing community, such as AutoCAD [16], CATIA [49], or Fusion360 [93], predominantly adopt the direct manipulation paradigm [148, 192, 237]. Conversely, less popular CAD applications such as OpenSCAD [160], IceSL [127], or JSCad [159] choose a programming-based approach.

Despite the lower prevalence of programming-based CAD applications in the community, the impact of such applications is substantial. Various web-storing platforms empower makers to upload models for sharing. While most sites only allow downloading the STL model files, specific web applications like Customizer [212] from Thingiverse, MakeWithTech [187], or 3dCustomizer [1], permit users to upload OpenSCAD models exposing modifiable parameters through widgets. This enables other users to recreate customized versions of the models simply by modifying the parameter values, all within a web browser. For instance, on Thingiverse, users employed these programming-based models with the Customizer tool to recreate customized versions and re-upload them to the website. Approximately 74,949 customized models were uploaded, representing 42% of all public things, from only 1,692 programming-based parametric models, which represented less than 1% of the models on the website [158].

In essence, less than 1% of the available models were uploaded in the programming-based format, enabling other users to create about 40 times more models for the community. No other web application provides the same features for storing parametric models created with direct manipulation CAD applications.



(a) Ranking of the most popular CAD applications based on the Reddit's users [237]. *Other* category includes the applications: Autodesk Inventor, Civil 3D, Microstation, Revit, Sketchup, Alias, Vectorworks, Draftsight, Spaceclaim, Tekla, Geomagic Design, ArtCam, Inroads, ArchiCAD, and BricsCAD

(b) Ranking of the most popular CAD applications in the 3D printing community according to the website 3D hubs [148, 164]

Figure 3: Rankings of CAD applications by popularity. Values represent the percentage of users compared to the total. Applications with blue bars implement mainly direct manipulation interactions. Applications with red bars implement the programming-based paradigm. There is no information about the interactive paradigm in bars with gray bars.

This doctoral thesis focuses on improving the user experience of programming-based CAD applications. Due to the broad spectrum that CAD may cover, we have focused on studying these applications in the field of digital personal fabrication with 3D printers, where programming-based CAD applications have a significant influence. Given the lack of research on understanding the challenges of applications implementing this interactive paradigm, the objective of the first study of this thesis is to establish the challenges that programming-based CAD applications empirically face in the 3D printing community. Later, our aim is to address some of the identified challenges by introducing the concept of *Bidirectional Programming* [19, 72, 143]. Bidirectional pro-

programming refers to systems that enable interaction with program output to update input after defining a backward transformation, always maintaining coherence between both [58]. Some GUI builders use bidirectional programming, and some research has explored it to create 2D vector graphics [78, 103]. We have addressed some of the identified challenges of the programming-based paradigm by modifying the source code of the most popular programming-based CAD application in the 3D printing community, OpenSCAD (Figure 3), in two additional studies.

Specifically, we aim to answer the research question,

"How can interaction techniques be used to facilitate design in programming-based CAD applications for 3D printing?"

To answer this question, this doctoral thesis aims to answer the following sub-questions.

- RQ1 *What are the motivations and challenges that users face when using programming-based CAD applications when designing models for 3D printing in personal fabrication?*
- RQ2 *How can bidirectional programming be used to enhance navigation and editing in programming-based CAD applications?*
- RQ3 *How can bidirectional programming facilitate the definition of geometric properties of models in programming-based CAD applications?*

1.1 RESEARCH OUTLINE

The research comprises three studies, as depicted in Figure 4. In the first study, our aim is to address the knowledge gap regarding user experience in programming-based CAD applications for 3D printing. We conducted a comprehensive study that involved interviewing twenty OpenSCAD users. The interview included user experience questions and a hands-on exercise replicating common tasks users perform while designing in programming-based CAD applications. The goal was to understand users' perspectives on the challenges of such applications, with the hands-on exercise revealing challenges faced during design. We took notes during the interviews, documenting participants' answers and behaviors in the hands-on exercise. We then performed a Reflexive Thematic Analysis (RTA), producing a codebook with three main themes: *Programming-based CAD users' profile*, *Design*, and *Printing*. Detailed results of this analysis are presented in Chapter 3, including a discussion on how these findings can motivate further exploration in programming-based CAD applications.

The second study of our research work aims to tackle some of the identified challenges in the user experience of programming-based CAD applications. Specifically, our goal was to address user challenges related to difficulties in navigating and relating the code with the visual representation of the models and difficulties in performing spatial

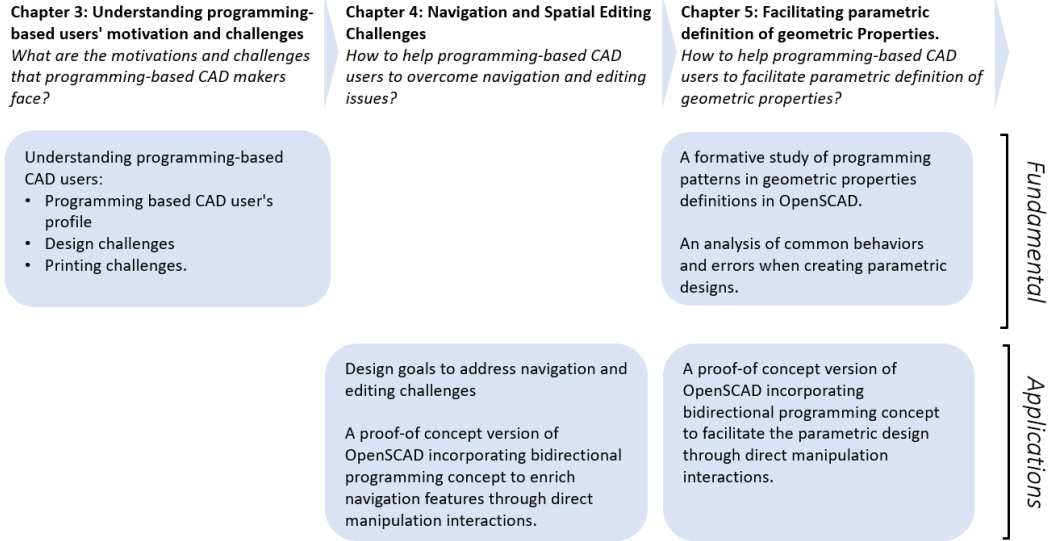


Figure 4: Overview of chapters summarising their research contributions.

edits. We based our work on previous research to apply the concept of Bidirectional Programming in programming-based CAD applications. We proposed design goals to address the challenges of navigation and model editing. Then we implemented the design goals by modifying the source code of the programming-based CAD application, OpenSCAD. We present and explain the features depicting how the new features addressed the identified challenges in the Chapter 4.

In the third study, our goal is to address the difficulties that programming-based CAD users face in defining geometric properties through arithmetic expressions when creating parametric designs found in the first study. We conducted a formative study to better understand how the definition of geometric properties is created in OpenSCAD models uploaded to the website Thingiverse. Based on these findings, we modified the source code of OpenSCAD to implement features that enable users to retrieve the parametric definition of the model's parts directly from the view to be re-used in the code. Then, we performed a user study, recruiting ten OpenSCAD users. In the experiment, participants answered questions related to their experience with parametric design. Later, they created one model in the normal version of OpenSCAD and one model using the implemented features. We tracked performance measurements in both cases to compare them. Moreover, participants scored the difficulty of the modeling task in both cases and discussed the potential or difficulties of the implemented features. We present the results of this study and discuss its implications in Chapter 5.

1.2 RESEARCH CONTRIBUTIONS

The contributions of this thesis includes:

- A comprehensive understanding of the motivations and challenges of programming-based CAD applications users organized in three categories: Programming-based CAD applications users' profile, challenges when modeling, challenges when printing.
- A modified version of OpenSCAD available at <http://ns.inria.fr/loki/bp>. This version incorporates bidirectional programming features, specifically tailored to enrich navigation capabilities and simplify editing through direct manipulation interactions.
- A formative study describing programming patterns in OpenSCAD models in terms of creation of primitives and spatial transformations. The formative study describes the structure that geometric properties definitions follow.
- A modified versions of OpenSCAD including a bidirectional programming features to retrieve parametric definition of geometric properties of models' parts directly from the view to be re-used in the code aiming to facilitate the parametric design.
- A user study validating the potential of using bidirectionnal programming features to facilitate parametric design in programming-based CAD applications.
- An analysis on common behaviors and errors that OpenSCAD users follow when designing to better understand the design process.

1.3 DOCUMENT STRUCTURE

Chapter 2 reviews relevant literature to this research, detailing CAD technologies and their interaction paradigms. It explores key concepts such as construction modes [184] and data representation [84], crucial for understanding CAD user experiences. Additionally, it outlines the process of Digital Personal Fabrication with 3D printers and examines significant studies in CAD design.

Chapters 3, 4, and 5 detail the empirical studies conducted.

Chapter 3 reports on interviews with users of programming-based CAD applications to identify their challenges. This investigation uncovers specific challenges within these applications, guiding the focus of subsequent studies.

Chapter 4 addresses challenges in model navigation and spatial transformations in programming-based CAD applications identified in the previous chapter. This chapter proposes a design solution and illustrates its application through integrating bidirectional programming in OpenSCAD.

Chapter 5 investigates the challenge of defining geometric properties in parametric designs within programming-based CAD models. Following insights from Chapter 3 and a formative study, this chapter introduces a design solution implemented in OpenSCAD. An evaluation with OpenSCAD users demonstrates the utility of this approach.

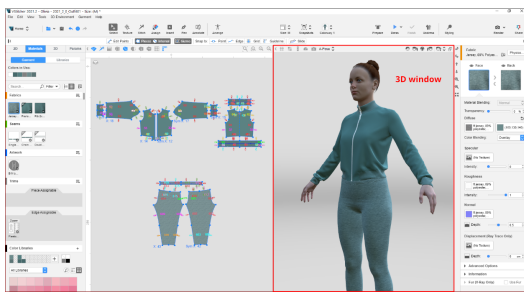
Chapter 6 summarizes the research findings and discusses their implications. It also outlines potential directions for future research, building on the contributions of this thesis.

STATE OF THE ART

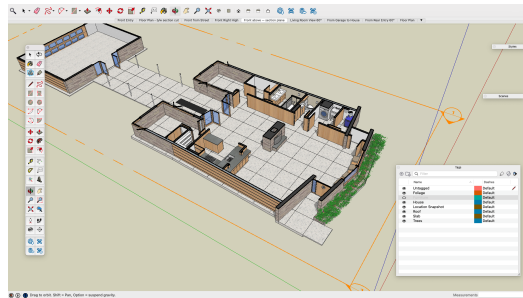
To better grasp the scope of the thesis, it is essential first to establish some fundamental terms. This section will begin by outlining the specifics of CAD technologies and the existing interaction paradigms including bidirectional programming. It will then delve into an explanation of the background of digital personal fabrication practice using 3D printers, as well as the current challenges faced by its users.

2.1 COMPUTER-AIDED DESIGN (CAD)

Computer-Aided Design (CAD) refers to computer applications that assist humans in the practice of creating, modifying, testing, and analyzing designs. This broad definition covers several different CAD applications used in a wide range of industries [5, 76, 80, 92, 96, 195, 216]. Across diverse technologies, the fundamental essence of CAD remains consistent. CAD applications enables users to articulate a model, generate a visual representation for verification, and sometimes conduct tests. For instance, VStitcher [28], a fashion CAD application, allows designers to craft clothing patterns and designs. It offers a visual preview of body avatars for iterative adjustments (Figure 5a). Similarly, SketchUp [216], widely used in architecture and interior design, facilitates layout and model creation, providing a visual representation for design validation (Figure 5b).



(a) VStitcher is a CAD application specialized in fashion design.



(b) SketchUp is a CAD application specialized in architecture and interior design.

Figure 5: CAD applications allows users to create a visual representation of models for visual validation. ¹

¹ Images taken from <https://help.browzwear.com/hc/en-us/articles/4921641889945-VStitcher-3D-Window-Browzwear> and <https://blog.sketchup.com/article/creating-plan-your-sketchup-model-layout>

In both VStitcher and SketchUp and most CAD applications, the design process involves iterative interactions between the user and the application. Users edit the model, visually verify results, and iterate until a satisfactory result is achieved. The effective communication channels provided by CAD applications are pivotal in delivering a positive user experience.

While various challenges have been reported in 3D modeling applications, such as modeling, dialog boxes, terminology, and others [124], this dissertation will focus on a key aspect of CAD applications workflows and capabilities: interaction paradigms. Our interest lies in exploring the potential of the programming-based paradigm in CAD applications. To facilitate a more meaningful comparison between different applications, we will concentrate on those used in the field of personal fabrication with 3D printers, where programming-based CAD applications exert significant influence [148, 237].

We will delve into different interaction paradigms within CAD applications, comparing their challenges and potential with the programming-based CAD paradigm investigated in this thesis. Subsequently, we will describe the categories within two aspects in CAD design that are closely related to the capabilities and challenges of interaction paradigms: the construction mode [184] and data representation [84].

2.1.1 Interactive Approach

D.T. Ross et al. [174] introduced the term CAD in the 1960s as a revolutionary approach to problem-solving. It emphasized a more collaborative human-machine process, challenging the predominant problem-solving paradigms of the time: Computer-Automated Design (CAutoD) [104] and automatic programming [18]. The former relied on elaborate automatic procedures, restricting user involvement to setting input parameters for desired outputs. The latter provided tools for programmers to create routines and programs but lacked user support in problem-solving. In other words, “CAutoD has the computer do too much, and the human do too little, whereas automatic programming has the human do too much and the computer do too little” [174]. Ross proposed CAD as an alternative that allowed intense human-machine interaction, enabling users to express design intents that the machine would interpret and support. Consequently, the communication and interaction channels provided by CAD applications as user input mechanisms gained significant importance.

Early CAD applications like SketchPad (1963) [206] are believed to be among the first to implement a Graphical User Interface (GUI). These applications facilitated the creation of 2D drawings akin to blueprints by directly drawing on a Cathode-Ray Tube (CRT) display using a light pen [46], possibly introducing the concept of *direct manipulation* interactions. For instance, Sketchpad (Figure 6) allowed users to select a geometric shape, like a line, and draw a path with a pen, transforming the user’s trace into a straight line. The user could then use this line as the circle’s radius, illus-

trating direct interaction with the design, which differs from the traditional text-entry command mechanism at the time, although the use of scripts was also available.



Figure 6: Sketchpad is one of the first CAD applications using a GUI to interact with the users with direct manipulation interactions.

Other applications like DAC-1 (1965) [115] or UNISURF (1968) [34] employed a command-driven interface, requiring users to use code commands for data input—likely due to the high cost of technologies such as light pens for implementing direct manipulation interactions [46]. However, as computer technology advanced, direct manipulation [192] became the dominant interaction paradigm in CAD applications. This approach mirrored designers’ drawing practices, creating a different communication logic with machines compared to text entry.

Currently, main interactive paradigms can be broadly categorized into direct manipulation and programming-based. As explained earlier, the former allows users to interact directly with the design, while the latter provides a programming language for users to programmatically describe a model, with the systems compiling and rendering the result. These paradigms are not mutually exclusive; several applications make efforts to combine them, creating new interaction paradigms. For example, Blender [62] enables users to create geometries through direct manipulation while integrating a text editor for executing scripts for specific actions.

McGuffin *et al.* [143] describe a taxonomy of systems using one, the other, or both paradigms to create an output. We based on this taxonomy applied to the CAD field to clarify the different interactive paradigms present in CAD applications. We will begin by defining the two main interaction paradigms in CAD applications: direct manipulation and programming-based. Subsequently, we will describe different approaches

combining direct manipulation with programming-based CAD, following McGuffin *et al.*'s taxonomy. Finally, we will describe the concept of bidirectional programming.

2.1.1.1 Direct Manipulation

Direct manipulation [191] interactions in CAD applications empower users to modify the visual representation of models through intuitive interactions, providing a seamless point-and-click modification experience with instantaneous feedback [89]. These interactions often align with the Skeuomorphism principle, where the visual representation and interactions mirror physical world metaphors to enhance user comprehension [200]. For example, in Tinkercad [15], to modify the size or position of a cube, the user selects the object and moves the pointer to the desired position while the program constantly updates the object's property (Figure 7). This way, to modify the digital dimension, the user needs to physically modify the controller's position, which is their hand or finger. The direct manipulation approach emphasizes the direct and easy-to-understand manipulation of on-screen objects, enabling users to move, resize, and delete objects through direct physical actions [201].

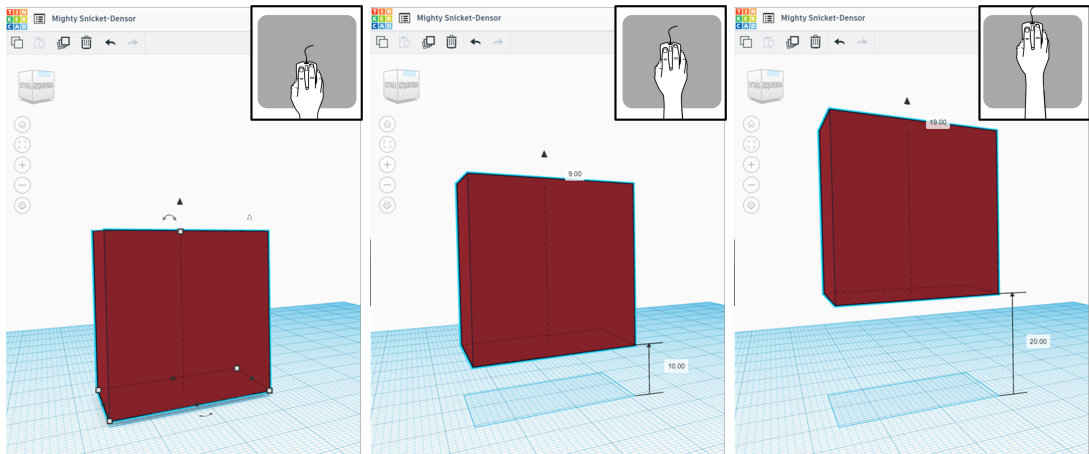


Figure 7: Direct manipulation aims to provide interactions in which their execution implies functionality. In CAD applications such as Tinkercad, for instance, to change the position of a cube, the user can select the top face and pull by moving the pointer up while the application continuously updates the object's position.

Compared to programming-based approaches, direct manipulation interfaces help reduce the gap, referred to as "gulf", between execution and evaluation by providing immediate feedback on the user's tasks [89, 226]. For instance, when scaling an object using a drag-and-drop interaction, the system provides continuous real-time updates on the object's size throughout the entire process. This immediate feedback allows users to assess the consequences of their actions swiftly [89]. In contrast, programming-based CAD applications lack this immediacy as users must modify the code, followed

by the system's compilation and rendering of the geometry. This distinction is noteworthy because immediate feedback simplifies the validation of actions' outcomes [191]. Therefore, direct manipulation interfaces enhance the user's comprehension of digital interactions by minimizing the gap between intention and action, ultimately promoting increased user engagement [89].

Direct manipulation interaction is based on the following principles [189, 191, 192] :

1. **Permanent representation of the objects of interest.** There is a visual representation of the object, including (and especially) when these are being edited. It achieves a highly appreciated usability principle, continuing visibility of the system status [153].
2. **The use of physical actions instead of complex syntax.** Interactions are executed through more friendly input mechanisms than a command line, such as pointer movements, pressing buttons, or clicks.
3. **Fast, incremental, and reversible operations with an immediate and visible impact on the object of interest.** As a consequence of the continued visibility of the system status, users can validate each action performed to continue or reverse it if the result is not desired.
4. **Progressive learning** The use of direct manipulation has been reported to be easier to learn than a programming-based approach [153, 228].

Applying the principles of direct manipulation helps to achieve usability goals such as constant visibility of the system's status, user control and freedom, recognition rather than recall, and, to some extent, flexibility and efficiency of use [153]. Consequently, the direct manipulation paradigm helps users to be more engaged [89, 236] and reduces the cognitive resources required to understand the user interfaces [193]. Thus, users can obtain decent results with little effort [3, 4] and the learning curve is normally shorter than programming-based paradigms [228]. Most CAD applications, such as Autocad [16], Tinkercad [7], or FreeCAD [210] implement direct manipulation interactions.

Nevertheless, the direct manipulation paradigm presents important and well-known challenges.

SELECTION The keystone operation in direct manipulation CAD applications is selecting elements in the visual representation [203]. Frequently, users need to point to individual parts by hovering a pointer over them and clicking to select them. The time required for this action has been effectively modeled by Fitts' Law [133], which states that the selection time increases with the distance to the target and decreases with the target's size. Hence, selecting small, distant objects takes notably longer. Following the same logic, the difficulty of selecting objects is similarly impacted by the distance to the object and its size [157]. While

initially conceptualized for one-dimensional tasks, extended studies in both 2D [134] and 3D environments [75] have shown that these difficulties persist across dimensions. Consequently, selecting and manipulating objects in models featuring small parts can be particularly challenging. The complexity further escalates in scenes densely populated with elements [54], where overlapping parts can hinder user's ability to efficiently locate and select the desired parts due to limited viewport capabilities and suboptimal zooming or filtering functionalities [89, 117].

PRECISION The precision capability in CAD applications is closely linked to the challenges in object selection [157]. Users often face difficulties in achieving precise pointing and meticulous control over design elements, particularly in complex or finely detailed designs. This is especially evident when users attempt to position the cursor accurately for small-scale components or tasks requiring high precision levels. To address these limitations, some CAD applications incorporate features that allow numerical input for defining object locations, thereby enhancing precision [89, 134].

SPATIAL TRANSFORMATIONS Most 3D CAD applications require users to manipulate 3D objects on a screen with 2D input devices such as a mouse pointer. This causes the problem of how to map a fundamentally 2D input device with two Degrees-of-freedom (DOFs) into different operations with more DOFs, such as moving and rotating an element in 3D [100]. Various solutions have been proposed, including touch-based manipulation [180] and pseudo-physical interaction techniques [32]. Nevertheless, the majority of CAD applications utilize graphic "*manipulators*" [100], which are visual representations that aid in object manipulation, such as arrow-shaped widgets that restrict movement to a specific direction.

NAVIGATION AND SPATIAL THINKING Direct manipulation CAD applications require intensive user interaction with the viewport for model navigation. Typically, 3D views in most CAD applications encompass eight DOFs: three for positional placement (*i.e.* translation), three for angular placement (*i.e.* , rotation), and two for zooming in and out (*i.e.* , depth translation). Jankowski and Hachet [100] highlight three primary challenges in navigating 3D spaces with 2D input devices. Firstly, the complexity of simultaneously managing multiple navigation parameters to achieve a specific point, angle, and distance. Further, standard actions like orbiting and panning are not consistently executed across different CAD applications. Secondly, the uniformity of interaction mechanisms across diverse tasks with varying requirements, such as navigating large-scale environments versus conducting precise inspections of 3D objects. Lastly, applications' lack of wayfinding features may disorient users in complex 3D scenes. Navigation in programming-based CAD applications is primarily utilized for verifying models in the viewport, as opposed to direct manipulation interfaces where navigation

is essential for both verification and model modification. Moreover, research indicates that comprehending 3D spaces on 2D screens can be challenging. Understanding the perspective is often difficult [88], and executing spatial transformations, such as translations and rotations, demands an in-depth understanding of the 3D coordinate system. The projection of 3D objects onto 2D screens relies on cerebral interpretation, which can lead to misconceptions in the visual representation [221].

REPETITIVE TASKS Automation and repetitive tasks can also be challenging indirect manipulation. Typically, performing repetitive tasks often results in tedious manual tasks (*e.g.* copy-paste an object) that, depending on the complexity of the desired output, can lead to tiresome and error-prone work. For example, if the user would like to create multiple copies of an object, then adjust their position one by one, and finally change one of their properties by editing each of them independently [89, 117].

AMBIGUITY Moreover, direct manipulation may introduce ambiguity. Gesture-based interactions, common in direct manipulation, may have inherent ambiguity. Similar gestures could be interpreted differently based on context, leading to confusion. CAD applications require resolution heuristics from the system to interpret the user's intention when acting. As a result, a similar action can have different results, making it difficult to create robust parameterized models with such applications [138].

In summary, the direct manipulation paradigm significantly enhances user interaction by employing intuitive metaphors and providing immediate feedback, effectively narrowing the gap between execution and evaluation. This approach offers notable benefits compared to programming-based CAD applications, where interactions are confined to a text editor interface.

2.1.1.2 *Programming-based*

Programming-based or programmatic interfaces [138] allow the creation of 3D models using a textual description in the form of source code. Programming-based CAD applications provide a programming language so the users can define objects, their geometric properties, and their relationships through scripts. The user creates the script and later triggers a compile action; the application will transform the coded description into a visual representation so the user can verify the result as depicted in Figure 8. In programming-based CAD applications, the code fully describes the model, and every modification is performed in the code so the system re-compiles the scripts and re-render the visual representation of the model. Some CAD applications based on direct manipulation interactions allow the execution of scripts as seen in FreeCAD

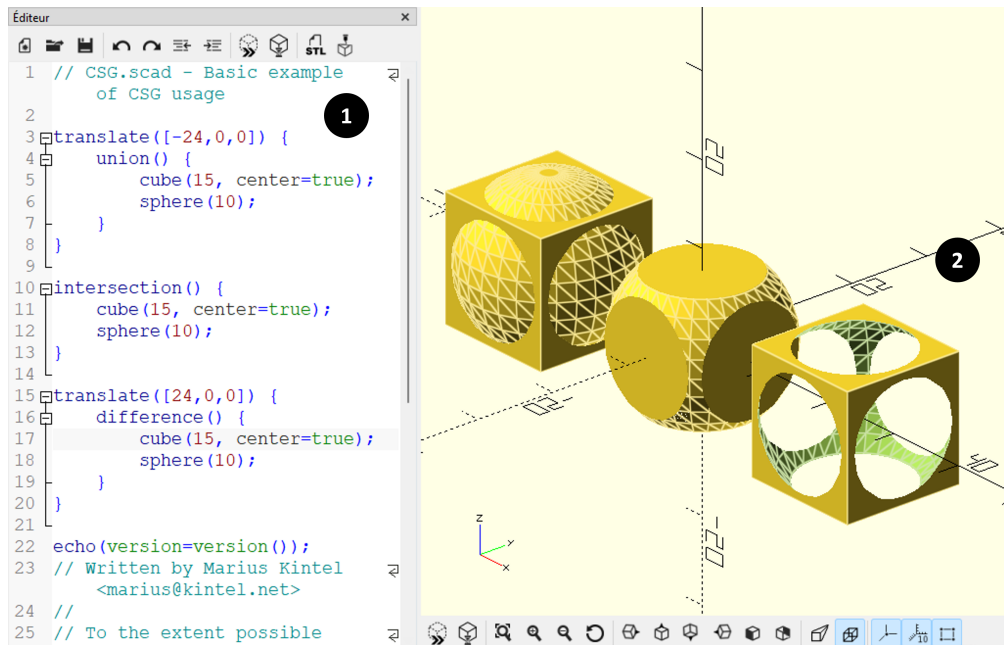


Figure 8: OpenSCAD is a [CAD](#) application implementing the programming-based paradigm. Normally, programming-based applications provide a text editor (1) to describe a model with a programming language. The application compiles and transforms a visual representation in a viewer (2).

[210] with Python scripts [63]. However, these solutions do not fall under the definition of programming-based [CAD](#) applications because the code does not fully describe the model but executes specific actions. Programming-based [CAD](#) applications can re-create the models with only the source code. Commonly, programming-based applications have a text editor in which users create the code, a compile button, and a view where the applications display the visual representation of the model, which usually offers few or no interaction capabilities except to change the position and angle of the view to inspect the model (Figure 9). Some examples of programming-based [CAD](#) applications are OpenSCAD [160], Cadquery [35], JSCAD [159], BRL-CAD [38], ImplicitCAD [131], and IceSL [127].

Programming-based description can also include *visual programming-based* paradigm [143], where applications allow users to describe models by manipulating graphical elements rather than writing lines of code. These graphical elements replace text-based programming constructs, such as functions, loops, and conditional statements, and are often presented as icons, blocks, or nodes [173]. This visual representation of code elements simplifies the understanding of programming concepts and maps conventional programming concepts into visual metaphors, significantly impacting the learning

2 Model taken from <https://www.thingiverse.com/thing:6415987> accessed the 26/01/2024

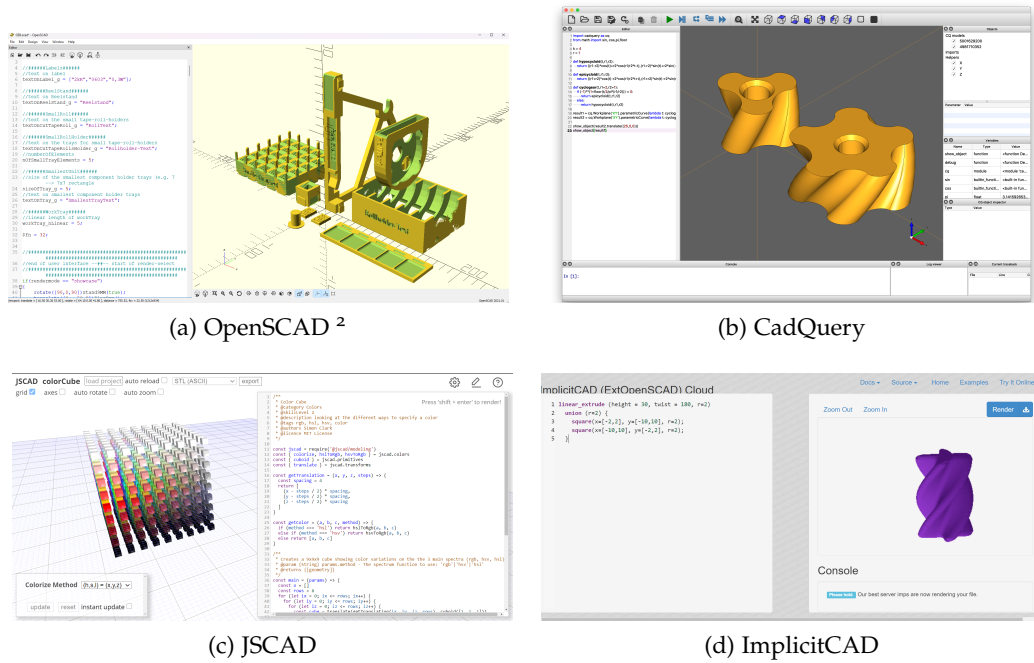


Figure 9: Programming-based CAD applications usually provides a text editor where users create a script describing the model and the application renders the result in a view.

of computer programming over text programming on beginners [155, 176]. However, some evidence shows that visual programming-based paradigms can be less productive than text-based CAD applications, especially for large-scale, complex design tasks [128]. BlocksCAD [94] and Grasshopper [50] are examples of visual programming-based CAD applications as depicted in Figure 10.

Programming brings highly appreciated advantages to 3D design [41, 229]. Repetitive and automatic actions are easily scripted [89]. For instance, making copies of an object and placing them in a specific pattern is significantly easier with programming compared to direct manipulation. Moreover, precision is an important advantage on programming-based CAD applications. Geometric properties can be defined with exact dimensions or expressed through mathematical expressions that assure exactitude in the designs. Some complex geometries that can be significantly challenging to express with direct manipulation, such as fractals, can be generated using programming strategies such as recursion. Versioning and collaborative works are also remarkable advantages of designing with programming-based CAD compared to direct manipulation. Furthermore, programming-based provides the user with complete design control with a non-ambiguous programming language, providing a consistent and robust

³ Model taken from <https://www.blocksad3d.com/community/projects/596293> accessed the 26/01/2024

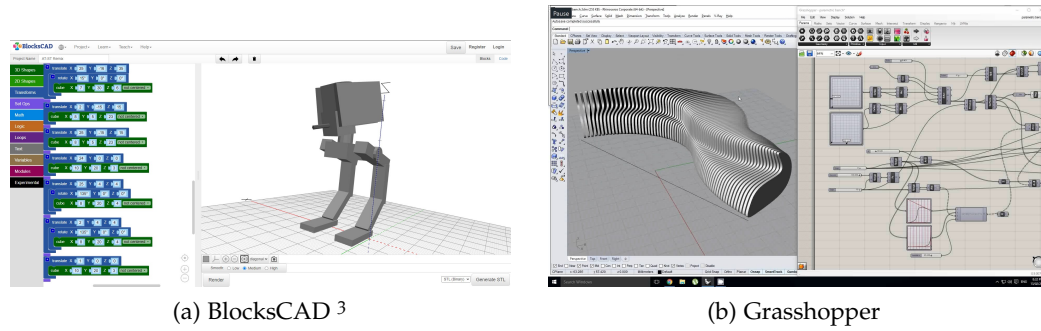


Figure 10: Visual programming-based CAD applications allows users to describe models programmatically through a graphical interface rather than traditional textual coding. Instead of writing code in a programming language, users visually construct and connect nodes or blocks representing various operations and functions.

mechanism to design. Direct manipulation requires the application to interpret user design intent and transform it into specific actions in the model. Different applications can interpret design intent differently on the same action. Thus, it is possible to find different results on the same action across different applications [138].

Despite the benefits of the programming-based paradigm, very few CAD applications implement it compared to other approaches, as illustrated in Figure 3 in Chapter 1. One of the reasons for this is possibly the difficulties that learning to program presents, which can be an entry barrier for users [111]. Programming is difficult to learn because of the distance semantics between the natural and programming languages. Previous research [111, 132, 175] has extensively studied different difficulties that users must overcome to program successfully. Novice programmers often encounter difficulties in understanding fundamental programming concepts, including the notion of machines and the syntax of languages to communicate with them [132]. Additionally, acquiring core programming concepts such as iteration, specific language constructs, and program design has been identified as a significant challenge for learners [175]. Moreover, transforming intent into a syntax using the available features that programming languages offer has also been reported as difficult [111, 227].

Despite the known problems of general programming, there is no clear understanding of the challenges that programming-based CAD users face. Programming-based CAD users may face (or not) some of the problems mentioned previously, but there is no empirical evidence or research effort to understand the specific problems presented by these applications. For example, some research reveals the importance of *spatial thinking* ability in successfully solving problems with multiple spatial parameters, particularly in engineering problem-solving [114]. Further, previous studies also look for new strategies to facilitate spatial skill learning among engineering students in CAD education [14]. Developing CAD applications for 3D modeling has further empha-

sized the relevance of spatial visualization skills in modern design practices within engineering fields [95]. Nevertheless, all these studies are focused mostly on direct manipulation interactions where the user has constant visual feedback of the object they are modifying, and often, the application provides visual cues to facilitate the task. When moving an object in Tinkercad, for instance, the user can select an arrow pointing in a specific direction, and while moving the object, the applications display a ruler to inform the distance that the object has moved as depicted in Figure 7. The user experience is completely different when the user intends to translate an object in a programming-based CAD application such as OpenSCAD. The user would need to use a `translate` statement, which receives three values corresponding to the distance as parameters to move along each of the three spatial axes. First, the user needs to understand the code and locate where the intended object is created to place a spatial translation statement. Then, the user needs to mentally match the view dimensions with the parameters in the code statement to move the object in the target direction. While in direct manipulation, the application provides visual cues to facilitate the process; often, in programming-based CAD applications, the user is mentally loaded to solve the problem [165].

In summary, programming introduces significant advantages to the field of CAD, yet users' utilization of programming-based CAD applications remains limited. Notably, prior research has not extensively explored the challenges these users face, which could facilitate the use of programming-based CAD applications. This thesis contributes by offering a thorough understanding of both the motivations and challenges encountered by users of programming-based CAD applications detailed in Chapter 3.

2.1.1.3 *Using programming in direct manipulation applications*

Several applications that implement direct manipulation interactions allow certain integrations with code. McGuffin and Fuhrman's taxonomy describes some approaches where applications allow certain collaboration between direct manipulation and code editing [143] to take advantage of both representations.

The Content-Oriented Programming (COP) approach allows users to modify the output with both direct manipulation interaction and executing scripts. Scripts can also be described through visual programming in the Content-Oriented Visual Programming (COVP) approach. An illustration of the COP approach is found in web programming, where an HTML document can be generated using a direct manipulation tool, followed by the composition of JavaScript instructions to modify the HTML document. Further, Arawjo *et al.* [11] designed an extension to Jupyter notebooks [47] that allow users to create pen-based circuit representation through drawings and combining them with Python scripts.

In CAD applications, several applications based on direct manipulation interactions also provide a COP approach. For instance, Fusion360 [93] allows users to create models by direct manipulation and to execute Python scripts to perform specific actions. Thus,

users can use programming to automate edits, such as making multiple copies of an object.

Another approach in McGuffin and Fuhrman's taxonomy is Programming By Example (PBE). Applications that implement PBE allow users to perform direct manipulation interactions while the system generates the instructions in code that would perform the same modification. Then, users can learn the scripted instructions to execute them later. FreeCAD [210] is a CAD application implementing PBE. Users can open an output console where the application will generate code statements corresponding to the direct manipulation actions performed in the view. Users can learn from these examples and execute similar code statements later in a code editor, as depicted in Figure 11. This approach contributes to the learning process. However, it is limited to a few actions because complex abstract elements such as loops or conditionals are not supported.

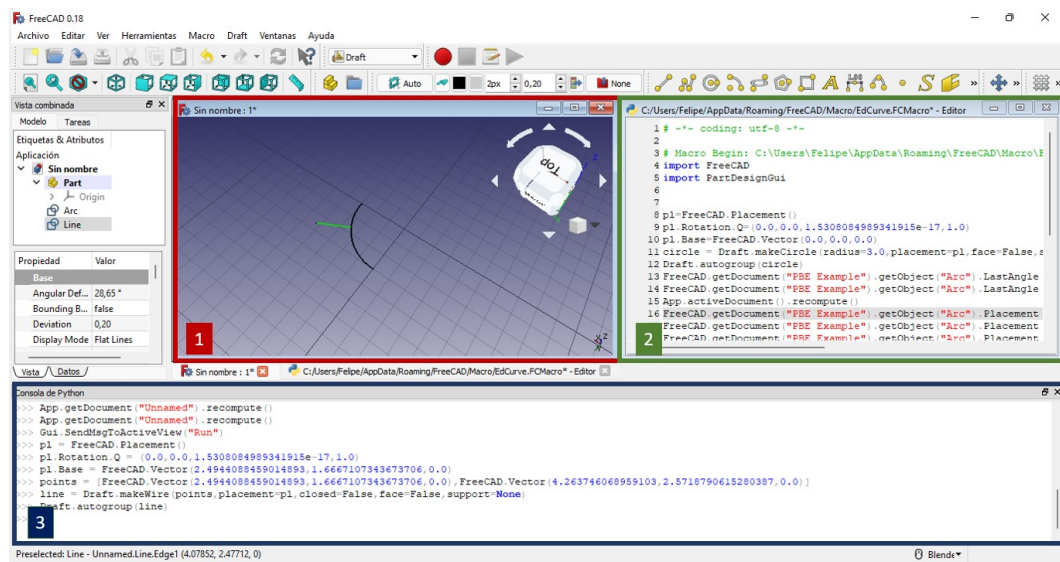


Figure 11: FreeCAD is CAD applications implementing the PBE approach. Users can perform direct manipulation interactions in the view (1). The system generates the corresponding in an output console (3) that users can execute later in the dedicated text editor for scripts (2).

These approaches present a fundamental difference compared to programming-based applications. The code is used to perform specific actions, but it does not comprehensively describe the 3D model visually represented. In programming-based CAD, a modification requires the user to go through the code, edit it coherently, and re-execute all the scripts to re-generate the model; in the other paradigms, the code executes specific actions to modify the model's current state.

2.1.1.4 Bidirectional Programming

McGuffin and Fuhrman define bidirectional programming in the context of programming interfaces that comprise both a code editor and visual content related to the set of instructions [143]. The instructions and the visual content define two different representations of the same entity. It is possible to edit the model by editing the code or directly manipulating the visual content. Furthermore, any update to either representation updates the other, maintaining synchronization between the two.

The concept of bidirectional programming is related to *bidirectional transformations* in databases, concerning the problem of *view updating* [19, 72]. In this context, a view created by a query on a data source must often be edited. Hence, the data source must also be updated so that both ends remain coherent. Similar problems and approaches have been explored in different disciplines, such as Model-Driven Software Development [52], Graphical User Interfaces [118], Relational Databases [19], or Structure Editors [87]. Despite the differences between different disciplines, there is some common ground in the definition of bidirectional transformations [45]. There are two representations of the information: the source and the view. It comprises a pair of unidirectional transformations: one that transforms the information from the source to the view, also called a forward transformation, and the other that interprets changes in the view to update the source, also called a backward transformation.

A few examples in diverse disciplines implement the concept of bidirectional programming. A common example is some GUI builders with which users can create an interface by dragging and dropping widgets. The corresponding instructions are automatically generated and can be edited to update the interface in the GUI builder window. In a seminal work, Victor demonstrated how to generate instructions by drawing graphical elements with direct manipulation [220]. The generated instructions can then be edited to update the graphical view. i-LaTeX [70] allows users to edit the content of a document from the view by adding a transitional view. For instance, users can click on the generated PDF's tables, mathematic definitions, or figures. The application will display an intermediate view of the element, combining the L^AT_EX code and the output. The user can modify the content assisted by visual cues while the application modifies the L^AT_EX code and the output. Similarly, Mage implements direct manipulation interactions in the output view of Python notebooks, allowing edits that the system reflects coherently in the code editor [107].

Some approaches provide a visual representation of code to facilitate understanding and allow users to manipulate both textual and visual code. Hempel and Chug [77] developed Maniposynth, which creates a visual representation of the code. This allows users to employ direct manipulation interaction while maintaining coherence between the code and the visual representation. Similarly, 3code [83] enables users to create mathematical procedures in code, displaying the same procedures with blocks in a visual programming paradigm. Users can edit either representation, and changes in one

representation are automatically updated in the other. These approaches exemplify the advantages of using code to execute complex instructions while allowing interaction in the view, where some tasks can be easier to perform.

Bidirectional programming is further exemplified by Sketch-N-Sketch [40, 78], a content creation tool for Scalable Vector Graphics (SVG) images (Figure 12). The system presents a programming interface with a 2D view that can be edited through direct manipulation while the system synchronizes both. Sketch-N-Sketch allows users to create basic shapes, such as rectangles or circles, directly in the 2D view. Furthermore, the program places control points around the shape to control characteristics (*i.e.*, position, size, color) by clicking on them. When a shape is created, the system inserts the code statement that creates the shape, including the arguments related to its characteristics. By performing direct manipulation, the user can update these arguments in the code from the view. Moreover, the user can link different control points to create constraints between the figures, such as keeping the size of two shapes equal. As a result, a variable is created in the instructions that are used by the different shapes and manipulated by the control points in the view. When the user edits a characteristic controlled by multiple variable constraints, Sketch-N-Sketch uses resolution heuristics to define the best way to update the code, and the changes are propagated to other shapes using the same variables. Twoville [103] adopts a similar approach; it is an SVG application where users can define scenes programmatically, adding control points that are editable from the view with drag-and-drop interactions while the applications update the code coherently. These examples solve specific limitations for SVG. However, 3D CAD design follows different requirements, with different data structures and different challenges.

In the context of 3D CAD software, CadQuery is a programming-based Python module for building parametric 3D CAD models [35] based on Boundary Representation (BREP). CadQuery allows users to navigate and filter geometries from the model in the code by using queries to modify them. For instance, a user can select specific faces of a cube to apply a fillet (rounding) to the edges with the code described in the listing 1.

```

1 # Create a solid cube
2 solid_cube = cq.Workplane("XY").box(10, 10, 10)
3 # Apply a fillet to the edges of the solid cube
4 filleted_cube = solid_cube.edges("|Z").fillet(2)

```

Listing 1: CadQuery Example

Users need to define correctly the query to select the elements they want to modify. The application does not assist in constructing the query, which can be challenging. Mathur *et al.* [138] facilitates the creation of query for CadQuery in FreeCAD [210].

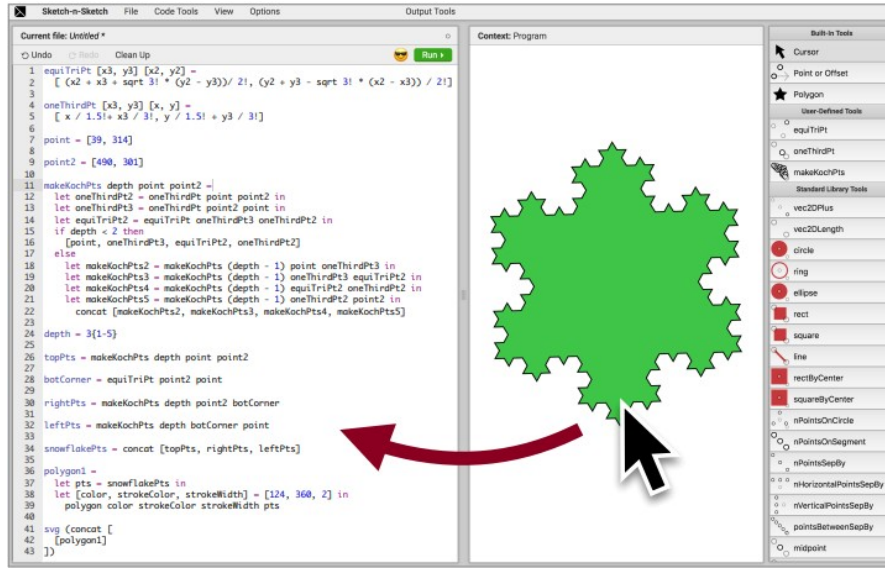


Figure 12: **SVG** - Bidirectional Programming example. Sketch-N-Sketch [78] creates and edits the output through direct manipulation and scripting. Every change in the script modifies the output and vice versa.

The users can select the element of interest directly on the view by clicking on it while the application synthesizes the query statement and places it into the clipboard so the user can re-use it to modify the part.

Blender is a 3D modeling application providing tools to create geometries with a **BREP** representation. Users can manipulate models by modifying faces, vertices, or points. Cascaval *et al.* [37] develop an addon for Blender [62] where users can create a design based on a coded description in Python. The add-on enables users to modify the position of elements in the view with direct manipulation interactions while the system synthesizes the corresponding changes in the code to keep coherence.

Antimony [106] is a bidirectionnal visual programming **CAD** application. Users can describe the model programmatically as in visual programming **CAD**. Antimony provides blocks that represent primitives and operations, and the user can connect inputs and outputs with direct manipulation interactions. Once the visual representation is rendered, users are provided with control points representing the blocks' inputs. Users can move these points with drag-and-drop interaction while the system updates the block definitions coherently.

Previous work on bidirectionnal programming for **CAD** applications [37, 106, 138] addresses specific problems in each domain. We draw inspiration from this work to address other challenges present in programming-based **CAD**.

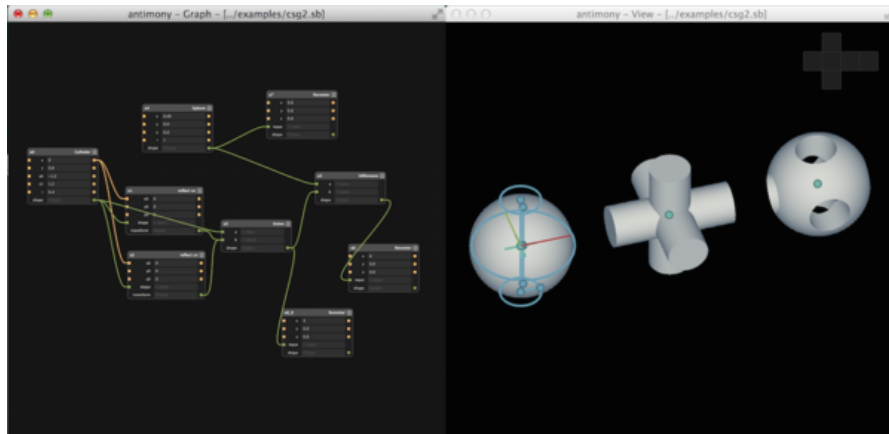


Figure 13: Antimony is a bidirectionnal visual programming. Users can describe models by visual programming and edit the model in the view while the system updates variable values coherently

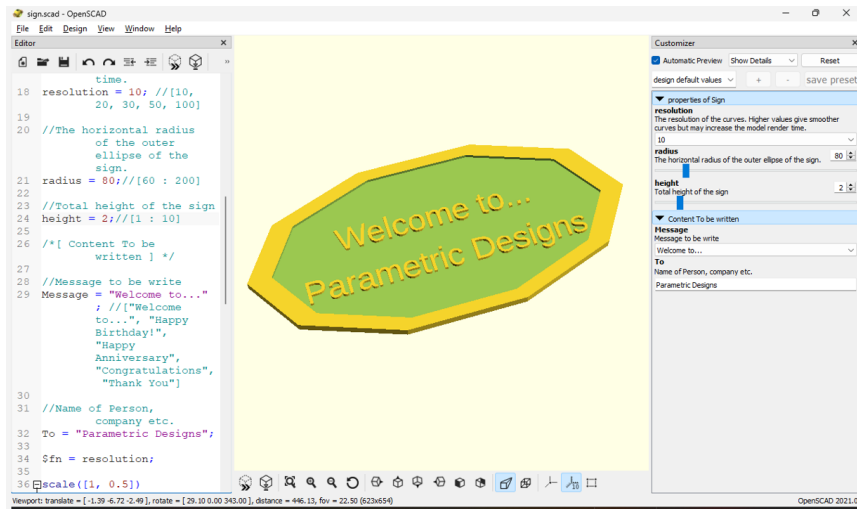
2.1.2 Construction Mode

Design applications employ different construction modes based on two key characteristics: whether the model is parametric or non-parametric and whether the model maintains a history of design steps (history-based) or not (non-history-based). While there are theoretically four possible combinations of these characteristics, in practice, two main types of CAD applications are prevalent [233]: history-based parametric modeling, commonly known as parametric modeling, and non-history-based non-parametric modeling, typically referred to as direct modeling.

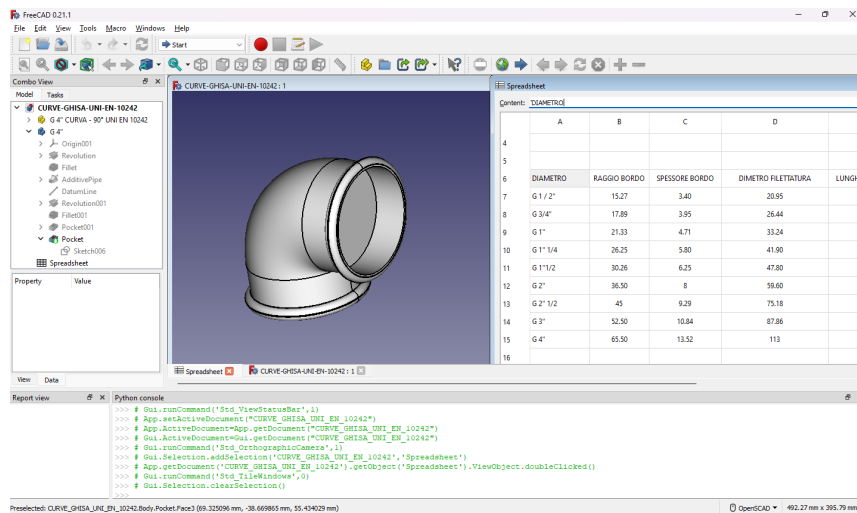
2.1.2.1 Parametric Modeling

Parametric modeling, referred to as feature-based or history-based modeling [8, 109], relies on parameters to define and control a model's geometric features and relationships. The design is created by specifying parameters and rules governing their interactions, facilitating easy modifications by adjusting parameter values. Programming-based CAD applications are naturally parametric due to their coding nature involving a history of construction embedded in the code with a re-executable set of steps defined by parameters. For instance, in OpenSCAD, users define parameters at the outset of the code editor. The application places a control element in a dedicated panel, allowing users to re-generate models by modifying parameters without directly editing the code (Figure 14a). Direct manipulation applications embracing a parametric approach enable users to define constraints and operations stored as features, visible to the user in a *history tree*. Users can revisit steps or features, modify them, and re-execute the entire history tree to re-generate the model. Moreover, modifiable parameters control these

features, allowing users to verify and edit the parameter values and re-execute the history tree based on the new values to obtain solid model variants without modifying the base model [233]. For instance, in FreeCAD [210], geometric properties are controlled by creating sketches that define geometric properties referencing cells within a spreadsheet. Users can alter the values of the spreadsheet cells, providing a means of recreating different versions of the model (Figure 14b).



(a) OpenSCAD provides a Customizer panel (on the right) to modify the parameters defined in the code.



(b) In FreeCAD, users can define geometric properties referencing variable cells of a spreadsheet (on the right) and re-execute the features in the history tree (on the left).

Figure 14: Parametric CAD applications examples in programming-based and direct manipulation paradigms.

Direct manipulation [CAD](#) applications employing parametric modeling present a challenge known as *Topological Naming Problem (TNP)* [64, 101]. During the process, users can refer to specific parts in the model to apply operations or constraints to them. These parts are created by the history tree and the parameter values defined before compiling the model. The application assigns a name to these parts, which serves as a reference for the user to perform operations on them. For instance, a user can refer to an edge, named "*edge_1*", to perform an operation such as smoothing corners created as a result of an intersection between two parts. The problem arises when the user makes changes in the history tree, such as changing operations or parameter values, resulting in changes to the names of different parts. In our previous example, certain changes may cause the application to name the target edge "*edge_2*", or even render the target part non-existent if the parts no longer intersect. However, the operation of smoothing corners on "*edge_1*" still exists in the history tree, creating inconsistency and yielding undesired results or even compilation problems.

2.1.2.2 *Direct modeling*

Direct modeling allows users to edit the model without preserving the history of these edits. Unlike parametric modeling, which captures a re-executable sequence of steps, direct modeling only retains the model's current state with no history tree. In other words, users only have access to the last status of the model, and the system does not store the executed steps in the past. Users can make changes without revisiting previous steps, akin to sculpting clay. This approach offers flexibility by eliminating concerns about features and their interdependencies that impact when making changes [29]. Tinkercad [7] serves as an example of a direct modeling application.

2.1.3 *Data representation*

There are several data representation types for 3D volumes, but two are mainly used in [CAD](#) applications: Constructive Solid Geometry ([CSG](#)) and [BREP](#) [84].

2.1.3.1 *Constructive Solid Geometry (CSG)*

Constructive Solid Geometry ([CSG](#)) serves as a robust geometric modeling framework extensively utilized in [CAD](#) and computer graphics. The modeling process in [CSG](#) initiates by incorporating primitives such as spheres, cylinders, cones, or cubes. These primitives undergo spatial transformations, including translation, rotation, or scaling, enabling modifications to their geometric properties. Additionally, primitives can be combined using fundamental boolean operations such as union, difference, and intersection [169]. The resulting geometries can undergo further combinations using the same operations. A binary tree representation is commonly employed for [CSG](#) mod-

els, where leaves signify primitives, and intermediate nodes denote transformations or operations, as illustrated in Figure 15.

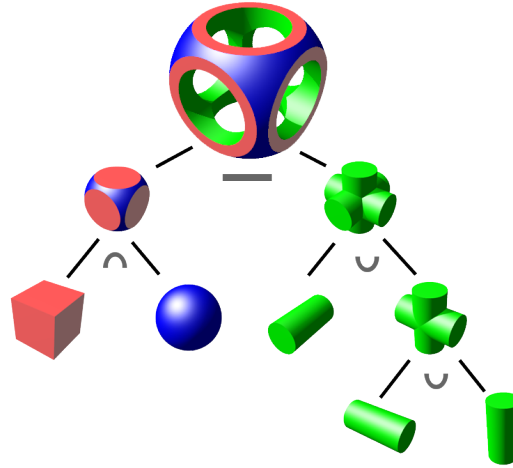


Figure 15: **CSG** models can be represented in a binary tree where the leaves are primitives and intermediate nodes are transformation and operations ⁴.

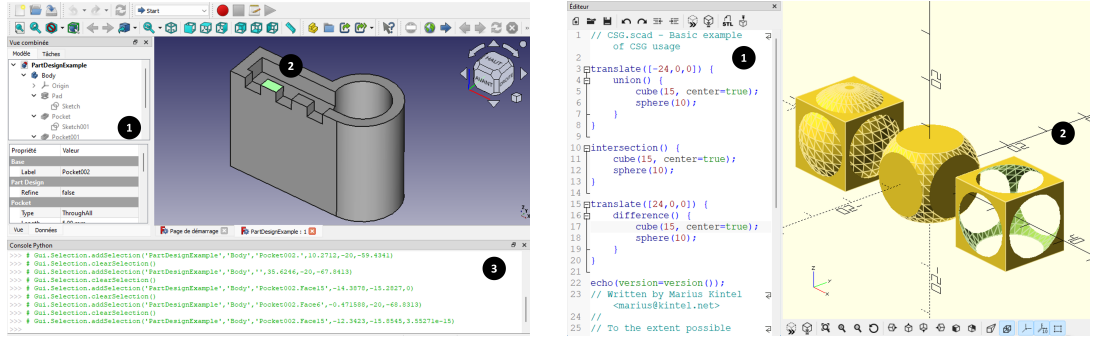
CSG allows the creation of very complex geometries, providing conciseness and mathematical accuracy. Both the surface and the interior of an object are implicitly defined. In other words, there is no description of specific geometric properties such as points, edges, or positions but abstract descriptions of primitives and operations. **CSG** objects are always *watertight* and manifold [205] if the primitives are [84]. Therefore, **CSG** provides closed and well-formed geometries that are printable, making **CSG** attractive for graphics [10] and 3D printing [68].

However, complex **CSG** models may incur longer rendering times, impacting performance in **CAD** applications. Furthermore, the inherent lack of surface detail in **CSG** may add limitations to creating specific shapes, such as curves. Figure 16b depicts how the **CSG** representation is implemented in OpenSCAD.

2.1.3.2 Boundary Representation (**BREP**)

Boundary Representation (**BREP**) is widely used to represent 3D geometric models in **CAD** applications. In **BREP**, a solid object is defined by describing its boundary or surfaces. It represents the shape of an object by explicitly capturing its boundaries, faces, edges, and vertices. **BREP** can be rendered efficiently on a graphic display system, enabling easy differentiation between vertices, edges, and faces. This offers more flexibility than **CSG** and allows for valuable operations in 3D printing, such as cham-

⁴ Image taken from https://en.wikipedia.org/wiki/Constructive_solid_geometry



- (a) FreeCAD is a parametric direct manipulation software using B-rep. (1) Users can check the history tree to revisit their steps. (2) They can also interact in the view, individualizing vertices, edges, or faces. (3) FreeCAD provides a Python console in which users can execute specific actions through code.
- (b) OpenSCAD is parametric programming-based CAD application using CSG. (1) Users can describe the models through primitives (e.g. sphere), transformations (e.g. translate), and boolean operations (e.g. union) in a text editor. (2) The system compiles and renders the result in a 3D viewer.

Figure 16: Comparisson between a parametric BREP-based and a parametric CSG-based CAD application

fering, blending, or drafting [68, 84]. FreeCAD [210] uses BREP as shown in Figure 16a.

2.1.4 OpenSCAD

The main focus of this dissertation is to understand the challenges of programming-based CAD users and facilitate the design process in these applications. We have decided to focus the efforts only on OpenSCAD users to conduct the investigation for two main reasons. The first one is that although most programming-based CAD applications share similar characteristics such as a similar layout, a similar pipeline, the use of CSG representation, or limited interactive features, there are differences in aspects such as programming language syntax, programming features, or performance that can make experience comparison difficult if using multiple CAD applications users. The second and most important is that OpenSCAD is the most popular programming-based CAD application with no real competitor [148, 237].

OpenSCAD is an open-source parametric CSG programming-based CAD application. Users can describe 3D models in its *functional declarative* programming language using the text editor, and the system compiles and renders the scripts in a viewer. Although OpenSCAD has applications in various domains [196], it mainly provides 3D printing-oriented features. For instance, OpenSCAD preferences menu offers features such as connecting with OctoPrint [90], a web interface to control consumer 3D printers or export models into the standard STL format for 3D printing [43]. OpenSCAD aims to

give full control over the design by being purely programming-based. It is the most popular of the programming-based CAD applications, which are mainly parametric CSG applications such as IceSL [127], JSCad [159], BRL-CAD [38], ImplicitCAD [131], or RapCAD [20].

OpenSCAD is not an interactive modeler and does not focus on the artistic aspects of 3D modeling but on the CAD aspects [160]. In addition to CSG modeling techniques, it allows the extrusion of 2D outlines. It also provides a *preview* mode that generates approximations for rapid visualization and a *render* mode that generates exact geometries using longer rendering times. In preview mode, OpenSCAD allows the user to right-click on the models in the view to display a menu of the CSG elements that create the clicked part. The user can click on a menu item while the system places the text cursor in the line of code that creates the CSG element. Moreover, OpenSCAD language includes *modifiers*⁵ for debugging. OpenSCAD modifiers are special characters placed before code statements to alter the visual representation of the model. These modifiers are beneficial for debugging by highlighting specific objects or modifying their rendering state, thus facilitating a clearer understanding of the code's impact on the model's appearance.

2.2 DIGITAL PERSONAL FABRICATION AND 3D PRINTERS

Digital Personal Fabrication is positioned within the Do-It-Yourself (DIY) movement and is characterized as “any creation, modification, or repair of objects without the aid of paid professionals” [116]. Gershenfeld [67] briefly defines it as the integration of all necessary steps, including logic, design, sensing, or actuation, to produce objects using digital tools like 3D printers, laser cutters, or CNC mills [13, 149].

3D printers construct geometric objects through the successive addition of material in layers [91], employing various techniques and materials for this purpose [186]. Compared to other fabrication options like laser cutters or CNC machines, 3D printers require minimal logistics and produce less waste and dust, making them the preferred choice for makers [149]. This research exclusively focuses on personal digital fabrication with 3D printers due to their potential affordability, ease of use, and installation [88, 149], heavily influenced by the use of programming-based CAD applications [148, 237].

Figure 2 in Chapter 1 illustrates a simplified adaptation of the fabrication process with 3D printers as described by Hudson *et al.* [88]. The process initiates in the **ideation** stage with a need or an idea, where motivations range from self-expression to creativity, skill development, cost-effective production, or personalization of items [88, 116, 190]. Moving to the **design** stage, users acquire a digital design that aligns with their idea and is in a format readable by the printer, such as STL format [43]. Design can be performed from scratch using a CAD application. Alternatively, users often explore

⁵ https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/Modifier_Characters Accessed: 10/01/2024

model repositories like Thingiverse [213] or MyMiniFactory [151] to download pre-existing models. Subsequently, users may either directly print the model or customize it to their specifications before printing in a CAD application. Then, the **print** stage involves correctly configuring the printer and fabricating the object with the design files. The **validation** stage follows, where users assess the result, iterating through previous stages if modifications are necessary until achieving a satisfactory outcome.

This research specifically delves into the challenges users face in the **design** stage, particularly when employing programming-based CAD applications. The literature review explores the difficulties encountered by users in the practice of digital personal fabrication with 3D printing, focusing on the design stage.

2.2.1 *Designing from scratch*

Makers can create digital designs from scratch using a CAD application. Designing a model for 3D printing encompasses more than just interacting with a computer. In instances where the object to be fabricated is intended for repairing [88] or enhancing other objects (referred to as *Augmented Fabrication*), makers often need to take physical measurements of objects and translate this information into the digital design [13]. When problems are identified during the physical inspection of printed objects, makers must revisit the design stage to address the edits necessary for resolving these issues.

Previous research has delved into the challenges and solutions associated with designing for personal fabrication, considering both direct manipulation and programming-based CAD applications.

2.2.1.1 *Modeling with direct manipulation CAD*

In 3D printing CAD, Hudson *et al.* [88] studied novices interacting with TinkerCAD and reported that manipulating elements in a 3D space through a 2D screen can confuse and lead to errors that are known problems in other digital applications [59]. Similar difficulties related to spatial thinking skills have also been reported in children using TinkerCAD [22]. Specifically, problems related to understanding the 3D perspective, understanding rotation in a 3D space, using the correct primitive shapes, and grouping primitive shapes. Programming-based CAD applications can present similar problems by having a 3D viewer rendering the output.

An everyday use of 3D printing is to repair [88] or to augment objects [13]. In such cases, measurements of physical objects, their transfer, and their meaningful use in the design are necessary. Mahapatra *et al.* [136] study how self-identified novice users capture physical measurements, transfer them to digital design, and verify their accuracy. The study carried out on TinkerCAD reported several obstacles when novices create a digital replica of an object, classified into three groups according to the moment they occur: 1) *Physical* when capturing the data, 2) *Digital* when using the data in the digi-

tal design, and 3) *Transition* when transferring (from physical to digital) or evaluating (from digital to physical) the data. Although physical obstacles are independent of the modeling CAD, digital and transition may differ when using a programming-based CAD applications. Probably some challenges may persist (e.g. "3D camera causes confusion"), others may occur differently (e.g. "Relative placement problems"), and others may not make sense in programming-based CAD (e.g. "Miscalculating by hand"). We take inspiration from this work to explore what problems programming-based CAD users face when measuring, transferring, and using data in the design process.

2.2.1.2 Modeling with programming-based CAD

Significant efforts have been made to understand the difficulties in learning programming languages [111, 171]. Expectedly, programming-based CAD users may present similar challenges, such as difficulties in structuring and breaking down the problem into smaller problems, difficulty finding the features the programs offer, or documentation problems.

Yeh and Kim [229] report problems with programming-based CAD and direct manipulation software offering scripting features for 3D design. These problems include difficulties reading code, re-using code, aligning objects, selecting parts, refactoring code, and 3D printing. Unfortunately, these undetailed findings were obtained from undocumented feedback from novice students with OpenSCAD and online forums such as StackOverflow. We aim to cover a broader scope of the 3D design, 3D printing, and re-using pre-existing models experience with OpenSCAD.

2.2.2 Sharing and re-using models.

Sharing is a keystone in the 3D printing community [116]. Multiple model-storing websites allow authors to upload models to share with other users, such as Thingiverse [213], MyMiniFactory [151] or Printables [235]. Some of them allow authors to upload coded parametric models that expose widgets so other users can modify parameter values for the system to create a customized version of the models [1, 187, 212]. Thingiverse, with its Customizer application [212] is an example of these solutions that store OpenSCAD models. Oehlberg *et al.* [158] reported how, after a year of the release of Customizer, about 40% of the Thingiverse models were created from parametric models. These findings depict how some users re-use models to remix them by changing parameters.

Alcock *et al.* [6] highlight several challenges faced by makers using applications such as Thingiverse, including difficulties in understanding model functionality, limited customization capabilities, issues with recognizing printability limitations of designs, and obstacles in understanding the creation process. Notably, this research does not concentrate on the experiences of programming-based model authors or examine how

programming-based users might reuse these models. We aim to understand the role of model-storing websites in programming-based CAD design.

In summary, programming-based CAD applications are crucial in the 3D printing community. They offer distinct advantages over direct manipulation applications, fostering a unique design thinking process that could attract a diverse user base. A significant benefit of programming-based CAD applications lies in their ability to support web applications, enabling users to upload and customize parametric models easily. Despite these strengths, the utilization of programming-based CAD applications remains limited, with only modest efforts directed toward understanding and resolving the challenges faced by their users. This thesis aims to thoroughly understand the user experience with programming-based CAD applications and propose practical solutions to some of the identified challenges.

UNDERSTANDING PROGRAMMING-BASED CAD APPLICATIONS USERS' MOTIVATION AND CHALLENGES

Note: The content of this chapter is the basis of a manuscript that has been accepted for publication in the proceedings of the 2024 CHI Conference on Human Factors in Computing Systems, to be held in Hawaii, USA, in May 2024.

Previous research on user behavior in CAD applications for 3D printing has predominantly focused on investigating improvement opportunities for CAD almost exclusively in direct manipulation tools [110, 112, 136, 217]. However, the findings from these studies are often not directly applicable to programming-based CAD. For instance, Mahapatra *et al.* [136] investigated the barriers involved in measuring objects, transferring these measurements to CAD applications, and digitally manipulating the data. They identified difficulties classified as “Digital” that are specific to direct manipulation tools, such as “3D camera causes manipulation errors”.

Furthermore, the literature related to programming-based CAD applications is limited. Previous research has explored the coding patterns on coded designs stored in online makerspaces [41] and the potential in programming-based CAD for students to learn programming through 3D design [105]. However, these studies address specific task difficulties within a limited scope of the design experience for 3D printing.

A better understanding of the user experience with these applications, their advantages, and the problems users face when 3D printing is still missing. In this study, we aim to investigate the following research questions: What are the motivations and challenges that users face when using programming-based CAD applications when designing models for 3D printing in personal fabrication? We seek to understand the current limitations of these applications and compare them with the challenges found previously with direct manipulation applications. There is an interesting opportunity for HCI to investigate the current challenges programming-based CAD applications users face to facilitate the design process and possibly ease the entry barriers for newcomers.

This project investigates how users of programming-based CAD applications experience the design and fabrication process. We conducted semi-structured interviews with twenty users of the most popular programming-based CAD application in 3D printing for personal fabrication, OpenSCAD [135, 148, 154, 237]. We asked participants about their design experiences with programming-based and direct manipulation CAD applications and their comparisons in the design and printing process. Additionally, we draw on previous work to explore their motivations and contrast barriers

found in 3D printing with direct manipulation tools. Specifically, we included questions related to problems measuring physical objects to create digital designs [136] and limitations working with pre-existing models from websites [6]. We also included a short hands-on exercise to observe design workflows and difficulties [229]. Based on the findings from the interviews, we provide a comprehensive analysis of the preferences of programming-based CAD users, current challenges in the design and printing process, and desired features expressed by the participants to improve these tools.

Our findings suggest that OpenSCAD users, often with a long programming experience and a programming-oriented mindset, face significant difficulties measuring and designing organic and curve shapes, mentally connecting the code with the view, performing spatial transformation due to the required mathematical skills, and addressing uncertainty when 3D printing.

3.1 METHOD

We conducted twenty semi-structured interviews to understand the motivations and challenges of OpenSCAD users empirically. The interview was divided into three parts. First, we asked participants for demographic information. Also, we asked them to self-rate their skill level on a scale from one (novice) to five (expert) on direct manipulation CAD applications, programming-based CAD applications, and general programming languages outside CAD. Similarly, we asked participants to self-rate their skill level in OpenSCAD on the same scale. The responses are reported in Table 1.

In the second part, we asked participants open questions about their experience in 3D printing and 3D modeling. Specifically, we were interested in understanding the motivations of makers in using OpenSCAD for 3D design, the challenges and limitations they face using OpenSCAD for 3D printing, their perception of direct manipulation programs compared to OpenSCAD, and ideas to improve OpenSCAD that might apply to other programming-based CAD applications. Furthermore, we draw on previous work on understanding the complexity and challenges of 3D modeling in direct manipulation applications to contrast these findings with the experience of OpenSCAD users. Concretely, we have included questions related to difficulties measuring physical objects and transferring data to digital designs [136], and sharing and re-using models in model-storing websites [6, 229]. The full questionnaire is attached in the Appendix A.

Finally, we wanted to understand the limitations of more specific actions when designing in OpenSCAD. We decided it would be easier to study participants' behavior in a real scenario while we observe them instead of only asking them to describe how they use the software, which could lead to easily missing specific actions or strategies they use. Thus, we asked the participants to perform a short hands-on exercise in the third part to observe their behavior while performing tasks. Based on the findings of previous work, we report problems in programming-based CAD related to selecting

specific parts to apply operations, including challenges in reading, navigating, and refactoring code [229]. If possible, we asked the participants to bring one of their own OpenSCAD models to the interview. P2 did not provide a model, so he used the example `candleStand.scad` provided by OpenSCAD.

The participants explained the motivation behind the model and went through their code, discussing difficulties and how they modeled their object. We asked them to perform search tasks replicating the need to select a part to modify it or apply an operation. We pointed at specific parts in the 3D view and asked the participant to locate the lines of code that created them. We asked participants to think aloud while we carefully observed the process, recurrent behaviors, and strategies. We paid special attention to the software features they used, the typical patterns they followed to perform the tasks, and the errors they made. Last, we discussed ideas they could have to improve their experiences in OpenSCAD.

The interviews lasted approximately 60 minutes on average. We took notes of their answers and the observed behaviors during the hands-on exercise. The experiment protocol was examined and approved by the ethics board in our laboratory.

3.1.1 Recruitment and Participants

We relied on the common use of 3D modeling in research and the active sharing nature of programming and maker communities on social media. We recruited participants from research laboratories and OpenSCAD channels on Reddit ([r/openscad](#)) and Facebook (OpenSCADAcademy) to conduct the semi-structured interview using video conferencing or in person. The only requirement was having enough experience with OpenSCAD to read and write code, but we also expressed that having 3D printing experience would be an asset.

We report our participant demographics and experience in Table 1. All the participants self-identified as male and varied in age: one was between 20 and 29, four were between 30 and 39, seven were between 40 and 49, four were between 50 and 59, and four were between 60 and 69 (average: 48.0, standard deviation: 11.7). All participants, except P8, had three or more years of 3D printing experience (average: 7.9y, standard deviation: 3.8). Except for P13 and P18, all participants self-rated with four or more in at least two programming languages. Moreover, all participants, except P20 mentioned having experience with direct manipulation CAD applications. Only five participants self-rated their direct manipulation CAD application skills with four or more. Regarding experience with other programming-based CAD applications or applications that allow scripting, only six participants expressed having any, and only P2 and P12 self-rated their skill level above 3 in one of those applications. Finally, participants self-rated their skill level with OpenSCAD as follows: One participant with 1, one participant with 2, six participants with 3, eight participants with 4, and four participants with 5.

Table 1: Demographics and self-rated skill level in CAD programs 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 *Others* is the highest rank expressed by the participant among the options.

*Direct manipulation CAD others: LibreCAD, Sketch Up, AutoCAD, Curve3D, On-Shape, Catia, SolidWorks.

**Programming Language Others: Prolog, MaxMSP, PureData, Ruby, GoLink, MatLab, Cobol, Pearl, Pascal, Groovy, TypeScript.

Participant	Age Range	3D printing experience (y)	OpenSCAD	Direct Manipulation CAD						Programming based CAD						Programming languages							
				Blender	FreeCAD	Fusion 360	TinkerCAD	Rhinoceros	Others*	IceSL	Python API	JsCAD	CadQuery	BRLCAD	BlocksCAD	C	C++	C#	Java	JavaScript	Python	PHP	Others**
P1	50 - 59	10	3	1	1				1							4			4	4	4	4	4
P2	40 - 49	6	1	3						5	2						5			3	4		3
P3	20 - 29	7	4		2	4			1								4				4		5
P4	30 - 39	5	3		3	3										4	4	4	4		4		4
P5	40 - 49	6	4						4							4		4	4				3
P6	40 - 49	15	4				2											4	2		4	3	
P7	30 - 39	8	4			3						3	3			4			4		4		
P8	40 - 49	1.5	3				2									4	4			5			5
P9	30 - 39	14	3	4	3			4	3								4			5	3		3
P10	60 - 69	8	5				2	1									5				5		5
P11	40 - 49	7	4		1							1								5	5	5	
P12	50 - 59	13	4		1		1		1			4		4	4								4
P13	60 - 69	5	3		1	2							1			1					1		
P14	60 - 69	4	5	1	3		5											2	4			4	3
P15	40 - 49	8	3	1			1		1											4	4		4
P16	50 - 59	6	4						1							4	4			4	4		
P17	30 - 39	15	5	4		4	4									2	2			4	4		
P18	60 - 69	8	2		1	1	1																
P19	40 - 49	3	4			1	1												5		5		3
P20	50 - 59	9	5								1							4			4		5

3.1.2 Data Analysis

We followed a Reflexive Thematic Analysis (RTA) [25, 33] approach in an iterative coding process. Our study aims to understand the user experience when using OpenSCAD, and part of the data collected included behavioral observations from the hands-on experience. Thus, we opted for a data analysis approach suitable for these studies [26, 33] that allows flexible participation of the researcher's interpretations rather than other qualitative analysis approaches such as code reliability or ground theory [24, 197].

We imported the interview data into the MaxQDA data analysis software [69]. One of the researchers performed an inductive analysis to develop a set of codes by coding the first ten interviews. Then, the coder started grouping codes by recognizing recurring patterns and identifying codes describing a central concept to create subthemes and themes [27]. To achieve a richer interpretation of the coding process [26], a second researcher performed a deductive thematic analysis on a randomly selected interview. The second coder used the codes created by the first coder in this interview and could create new codes when necessary. Then, both coders discussed the disagreements and refined the codes by removing, merging, changing, or adding new codes. After reorganizing codes, subthemes, and themes, the first ten interviews were re-coded with the resulting set of codes. In the second iteration, the first coder continued the inductive analysis with the next five interviews, followed by the deductive coding from the second coder, discussions on the codes, refinement of the codebook, and re-coding of the interviews. A third iteration was performed to complete the coding of the total of interviews.

Although RTA does not seek reliability coding [25], we were interested in tracking the level of agreement between coders. We calculated Cohen's kappa index in every iteration to verify inter-coder reliability [141]. At the end of the coding of all interviews, we achieved a *substantial* [119] agreement: iteration 1 ($\kappa = 0.543$), iteration 2 ($\kappa = 0.592$), iteration 3 ($\kappa = 0.617$). Most of the codes were created in the first fourteen interviews, achieving a potential code saturation. However, it was not until the seventeenth interview that codes, themes, and subthemes found in the codebook did not have substantial changes, and their meaning was well established, achieving meaning saturation [79].

3.2 THEMES

We created 266 individual codes to code a total of 783 segments of our notes. We grouped codes into subthemes and then into three main themes. *Programming-based CAD user profile* theme groups 22 codes (59 segments coded) and 9 subthemes as depicted in Table 2. Table 3 depicts the theme *Design* that covers 193 codes (632 seg-

ments coded) and 80 subthemes. Finally, *Printing* theme includes 51 codes (92 segments coded) and 20 subthemes (Table 4).

3.2.1 Programming-based CAD users profile

We start by discussing the design experience of the participants in OpenSCAD. Later, we discuss why participants use a programming-based CAD and their opinions on direct manipulation programs.

Table 2: Structure of theme *Programming-based CAD user profile*. Color intensity is proportional to the number of interviews coded with codes of the theme and subthemes.

Theme	Subtheme		
Programming-based CAD users (n = 18)	Experience with OpenSCAD (n = 13)	Advantages (n = 10)	OpenSCAD is popular and open source (n = 4)
			Programming features make design easier (n = 3)
			Easy to describe complex shapes (n = 2)
			Parameters allow a flexible definition of objects (n = 3)
	Disadvantages (n = 7)		
	Programmer mindset (n = 13)		
Direct manipulation mindset does not work for me (n = 6)			

3.2.1.1 Experience with OpenSCAD

"I do not know any other realistic competitor against OpenSCAD, I do not hint there is anything"

Participants (n = 10) mentioned the advantages of using OpenSCAD. The first is related to the parametric capability of programming-based CAD. P17, for instance, discussed his work in a laboratory making prototypes *"it was useful to have this programmatic base to create arbitrary variations of similar things"*. Participants found it helpful to define complex geometries through mathematical definitions instead of storing high volumes of data when having geometric information, as happens in direct modeling. P2 worked with robotics and talked about his needs *"I want compact shape descriptions that generate highly complex geometries (...) we don't want to store all the geometry with triangles"*. Programming-based CAD also helps to generalize models better. For instance, P2 mentioned that resizing a robotic articulated arm involves more than just geometric scaling. The operation may require adding another articulated section, and describing this behavior in direct manipulation is difficult. Moreover, programming features such as abstraction allow participants to re-use work. Further, participants found it convenient that OpenSCAD is open-source, runs on all major operating systems, and is the most popular programming-based application, so it has community support. (P6 *"I do not know any other realistic competitor against OpenSCAD, I do not hint there is anything"*). P12 mentioned *"It's also the most popular program. If you find a problem, someone else already had it and you can just copy the code or see a different approach"*.

However, some participants (n = 7) identified liabilities of using OpenSCAD. Despite the support community, they feel that the development of the application is slow.

P4 mentioned trying to contribute to the project on GitHub and realizing that multiple pull requests have been on hold for a long time without being integrated into the application. Moreover, they found that the available features of the application are too basic and the rendering time of complex models inconveniently long.

3.2.1.2 Programmer mindset

All participants except P18 (Table 1) had a programming background and found the 3D modeling programming-based paradigm convenient. P19 expressed *"I discovered OpenSCAD. I was like, 'Oh, hey, this is just models and software' (...) It was beneficial to be able to develop parametric models in code, which was part of my skill set."* Programmatic interfaces are a good entry door to 3D printing for this population. For instance, P20 mentioned *"I was learning the modeling (...) When I saw the OpenSCAD code and I looked at and then I said 'This is programming, this is source code, this is how I do things' (...) it was a continuation of what I already did rather than learning something completely new."*

"Oh, hey, this is just models and software"

Participants ($n = 13$) also mentioned that OpenSCAD also fits their mathematically oriented mindset. Interestingly, P12 and P15 expressed that despite having a programmer mindset, one of their problems was their lack of math skills. P12 said *"I'd like to say I'm an expert in OpenSCAD (...) The one thing that bugs me down is the math. I always get stuck on that"*.

3.2.1.3 Experience with direct manipulation programs

Six participants commented that the direct manipulation paradigm did not work with their mindset. P11 and P15 said respectively *"I'm not a visual guy, not really an artist. I can imagine what I want to do and write it down without a preview"* and *"(direct manipulation) seems to be complicated for me for some reason"*.

"I'm not a visual guy, not really an artist"

Interestingly, some participants ($n = 4$) thought that they would inevitably need to learn direct manipulation applications due to the perceived limitations of programming-based ones. However, some of them have succeeded without learning a new application. P5 commented *"I've always said to myself, I'll learn AutoCAD when I need it, and so far I haven't needed it."*

3.2.2 Design

We discussed several aspects of the design process and how it relates to the other stages in the fabrication process.

Participants discussed the difficulties they usually face when performing this task in OpenSCAD and compared it with direct manipulation applications.

Table 3: Structure of theme *Design*. Color intensity is proportional to the number of interviews coded with codes of the theme and subthemes.

Theme	Subtheme						
Design (n = 20)	Working with existing objects (n = 13)	Physical Measuring (n = 12)	Linear measurements (n = 11) Curves, round, and organic shapes (n = 11) I use non orthodox methods (n = 7) I prefer not to deal with it (n = 4) Challenges (n = 6)				
		Making things fit (n = 5)					
		Spatial actions (n = 15)	In direct manipulation programs (n = 2) In programming-based CAD (n = 15)				
	Manipulating the model (n = 16)	Parametric modeling (n = 11)	In programming-based CAD (n = 9) In direct manipulation programs Advantages (n = 3) Disadvantages (n = 3)				
		Achieving digital precision (n = 9)	In direct manipulation (n = 2) In programming based CAD (n = 9) OpenSCAD is convenient (n = 3) Verifying (n = 8)				
	Creating organic and curved shapes (n = 10)	In programming-baed CAD (n = 9) In direct manipulation programs (n = 2)					
	Screen problems (n = 2)						
	CSG (n = 7)	Not possible to individualize points or faces (n = 3) Subtracted parts (n = 6)					
	Versioning and collaborative work (n = 3)						
	Repetitive actions (n = 4)						
	Programming-based specifics (n = 20)	Working with several files/parts (n = 9)					
		Dealing with code (n = 18)	Code practices (n = 14) Challenges (n = 7) Code editor (n = 6)				
			Code-view navigation, selecting parts and understanding (n = 20)	Reading the code to understand (n = 18) Guessing from the view (n = 1) OpenSCAD search (n = 4) Text Search feature (n = 5) Errors (n = 4) Other challenges (n = 8) Using modifiers (n = 15) Changing colors (n = 3) Removing objects temporarily (n = 10) Trial and error (n = 9)	Relate objects with something I know (n = 4) Read and analyze the code (n = 13) I remember what I coded (n = 12)		
				Challenges (n = 7)			
		Other challenges (n = 3)					
		Improvement opportunities (n = 17)		Spatial information extraction (n = 7) Interactive editing (n = 8) Interactive selection and navigation (n = 11)			
				Reusing models or Design from scratch (n = 20)	My pipeline (n = 10)		
					Design from scratch (n = 12)	I enjoy doing it, wanna do it better (n = 5) My needs are too specific for pre-existing models (n = 6) It is easier, faster, or better quality if I do it (n = 4)	
		Reusing models (n = 20)				Pre-existing models do not fulfill my needs (n = 16) Model storing websites and its offer of models (n = 7) Limitations with parametric models (n = 4) Search engines and availability (n = 6) Licenses restrictions (n = 2) Pre-existing models are time savers (n = 7) Pre-existing models are for inspiration (n = 8) Re-using meshes (n = 14) It is difficult to edit meshes (n = 4) STL files are usually broken non manifold (n = 11) Code models are more flexible to re-use (n = 3) Libraries (n = 2) Re-using code (n = 10) Available code quality, understanding other people's code (n = 5) I adapt pre-existing code to my models (n = 4)	
						Sharing models (n = 8)	

3.2.2.1 Working with existing objects

Often, makers fabricate objects that will interact with other objects, such as in the case of repairing or augmenting an object. Participants shared their experiences in such cases.

LINEAR MEASURING The preferred measurement tool for linear measurements is the digital caliper. However, two participants stated that the task of measuring increases uncertainty and leads to more iterations. P8 mentioned *"If I was better at taking measurements, I would go through fewer iterations, and my first print would probably be closer to what I want"* and P20 *"I would say the difficulty of measuring on my part. I am not good at working with my hands"*. Moreover, calipers have size limitations, according to two participants. P14 commented *"calipers don't go big enough to measure a lot of this stuff"* and P17 *"There are cases where the thing that you're trying to measure is too big for calipers"*.

MEASURING ORGANIC SHAPES AND CURVES. Measuring organic or curved shapes is complex ($n = 11$). For instance, P12 and P15 commented on their work repairing pieces that *"the rounded organic hardware, if you need to make it fit, it can be a pain."* and *"it's rectangular and then there's a curve, and that part I had to print like 15 times to get that right."*. To deal with it, seven participants reported using creative solutions. For example, P12, P14, and P20 have used cameras or scanners to get the outline of a shape and use it later in the design by transforming it into an SVG or by measuring the outline and approximating the curves. P14 commented *"I photocopied the object, I put it down on a photocopier, so I could get a picture of the rim of the profile, from which I could make measurements of it. Then I had to run an optimization program in a spreadsheet to figure out how everything fits together, how all the curves match, and what angles join each other..."*. P14 also reported using photogrammetry with no good results. *"I tried photogrammetry to make a 3D model of this; it just doesn't work if you have shiny or transparent surfaces. I got a nice model but also sort of a cloud of nondescript points because of all the reflections on the surface"*. Other participants have tried to measure some points to interpolate by guessing in a trial-and-error strategy. P4 mentioned *"I guess there I would do some measurements, and then there will be some guesswork and trial and error"*. P12 mentioned using a contour gauge to approximate curves and P14 said that at some point, he would hold the physical object in front of the screen to validate the result *"I would hold up the vacuum cleaner in front of the screen to see how well it matched."*

"... the rounded organic hardware, if you need to make it fit, it can be a pain"

Some participants ($n = 4$) prefer to avoid organic shapes and curves. P20 commented that *"I try to avoid them (curves and organic shapes) in my designs. If I'm just talking about rounding corners, I pick up a set of radius measurement tools so I can get the correct size of rounded edges. Beyond that, it's trial and error."*

"I'd like to have a model of that object in my CAD software so that I can build around it or in it"

USING DIGITAL REPLICAS. It is easy to miss context elements when fabricating objects interacting with other things ($n = 5$). Makers cannot imagine every aspect of the physical objects in the 3D view to see if the design will satisfy their needs. P9 explained *"this piece is a cover for an emergency button that you need to screw in manually. I did not think about it in my first iteration, so I could not access the screw hole and had to repeat it, adding a small hole"*. Some participants create a digital representation of the physical object to have a reference to work with, making them more confident about the design decisions. For instance, P11 used an STL model in a project for his phone: *"I've used a model of my phone for that. I just use an STL of the phone and design around it"*. P14 mentioned the convenience of having digital replicas for his projects *"if I'm making something to fit another object that already exists that (...) I'd like to have a model of that object in my CAD software so that I can build around it or in it"*

3.2.2.2 Spatial transformations

The keystone modeling action is manipulating objects' position, orientation, and size. Participants discussed the difficulties they usually face when performing this task in OpenSCAD and compared it with direct manipulation applications.

Some participants ($n = 2$) acknowledge how easy this task is in direct manipulation applications. P2 and P19 commented *"You have immediate feedback of the shape with direct manipulation. It's extremely easy when you want to move things in the 3D space."* and *"With Fusion360 or TinkerCAD (it is handy) to make a cylinder of the right dimension in the right place, basically drag and drop and get it where it belongs"*. On the contrary, the same task in programming-based CAD is reported to be very difficult ($n = 15$). Trying to place an element in the right place can be challenging, as commented by P7 *"Most times, my difficulty is not drawing but where it should be drawing. Like I aim to draw a sphere here and OpenSCAD put it over there, and I am like 'why?'"*.

It seems to be challenging to relate the transformation parameters of a code statement with the spatial coordinates in the view without visual cues. For instance, if a translation is applied with 10 units in the second parameter on a sphere, it is not easy to predict the direction the sphere will move in the view. The camera view can be in a position and orientation that may make it hard to understand where the axes are located and there is not enough visual help for the user. OpenSCAD view has a widget representing the canonical x , y , and z axes directions. However, users often need to locate coordinate systems different from canonical ones. The position and orientation of objects are typically the result of multiple nested spatial transformations. Each transformation has a scope where the relative coordinates system does not match with the canonical one, so the widgets are useless. P7 and P15 said *"Let's say you have a geometry and you need to have features that have a certain offset, and you create the translate for these offsets but these translations then build above each other and then you are like wait, what one I am changing now?"* and *"if you are creating some volume (...) it is difficult to predict, with all the operation, where they land and what precise coordinates would be"*

Participants also found dealing with translations and rotations of the same object challenging. These spatial transformations are not commutative. In other words, applying a translate after a rotate would not give the same result if the commands are executed in the opposite order. P19 commented *“you need to think about how you want to translate and rotate it before you can even get it to where you want. (Otherwise) You might find that you get your translation operations out of order, and all of a sudden, you’re in the wrong place. One of the most challenging parts of working with OpenSCAD is understanding translations and rotations”*. To deal with this, participants use a trial-and-error technique. P6 mentioned *“If you would ask me right now, to rotate this in a certain direction, I would not, without testing, be able to tell you if it is -90, 0, 90, or what combination of the three parameters I need to get it in the direction I want”*. P9 commented on having a specific order for transformations: *“I rotate first and translate later. It is easier because when I translate before rotation, sometimes the rotation center is not the same as the object’s center.”* The participants proposed another strategy implementing position and orientation checkpoints. They correctly generated the transformation required for placing objects where they belong. The elements were then designed in the canonical coordinate system, and after completion, the participants applied the previously calculated transformation. P17 commented *“I remember doing like a checkpoint where I start. I make sure that everything starts from this origin point. Then, when you add in multiple modules, make sure that they’re centered around so you don’t have to keep track of all the different systems.”*. Similarly, P16 commented that he created modules solely to place elements in a location and orientation of interest. Some participants mentioned avoiding having several layers of transformations because it becomes unmanageable. P3 commented about a model they shared with us *“my approach is to avoid it (multiple nested transformations). This volume is at the base of most parts in this model, and it has this particular translation that is my X-Y plane for all of time. And I don’t ever want to change it, so everything can be referenced with respect to this plane.”*

In addition, it was deemed challenging to calculate the appropriate parameter values for the spatial transformations. As objects’ position and orientation are built upon multiple transformations and involve the sizes of other objects, the coordinates system changes constantly, and participants must mathematically derive parameters’ values of spatial transformations. Depending on the previous spatial transformations, the relative coordinate system varies. P14 commented *“... I have to painfully mathematically calculate where that plane is in space, its slope, and where its normal vector is, and then get the surface on there...”*. Interestingly, some of the self-considered knowledgeable participants found themselves lacking math skills. For instance, P12 said *“I’d like to say I’m an expert because I can pretty much try everything in the language. The one thing that bugs me down is the math. I always get stuck on that.”*.

“... I have to painfully mathematically calculate where that plane is in space, its slope ...”

3.2.2.3 Parametric modeling

The creation and use of parametric models were reported as valuable for the participants ($n = 11$). For instance, P14 stated that direct modeling is not enough for serious developments *“TinkerCAD is easy and fun, but not useful for parametric modeling or anything serious.”*. Further, they value the possibility of revisiting their steps. P7 said *“because everything is written down ... if I pick up something one year later, I’ll remember exactly what I did.”*. Moreover, they made clear differences between creating a parametric model in direct manipulation software and OpenSCAD.

Six participants shared their thoughts about creating parametric models through constraints in direct manipulation applications such as FreeCAD and Fusion360. The ability to select objects of interest directly from the view is perceived as practical, as mentioned by P14 *“you can grab a vertex and snap it to some other location and not have to worry about numbers and measurements so much, but you can make things fit just by moving things around and snapping things to grid points or to other vertices, I find that’s very nice.”*. However, the constraints management in such programs was perceived as difficult. Participants found tracking constraints challenging because they are represented as a list in a panel different from the view as P9 said *“It is not evident to keep track of the constraints, especially in FreeCAD where you have many panels, well as many other CAD programs I guess.”*. In consequence, it is easy to get into a model with several constraints that are hard to edit or that end up being overconstrained. P9 commented *“I try to avoid using constraints normally because when I’ve tried, I’ve ended up stuck, I have too many constraints and keep receiving the “Overconstrained” error message, and it is not easy to fix it for me. I do not know how to know which constraint is the problem”*.

Moreover, P13 mentioned not trusting parametric modeling in direct manipulation CAD applications due to the Topological Naming Problem (TNP) (section 2.1.2.1): *“...from a practical standpoint, you really can’t go back and change anything. You can change parameters, but you can’t change anything in connection. If you change a parameter that causes a connection to break, you lose it. And I’m constantly going back and changing things. To me, that’s the power of CAD. That’s actually why I use OpenSCAD, because you can go back and rearrange your whole model, change your whole concept, and it doesn’t break.”*

Some participants ($n = 9$) commented on the way of creating parametric models by defining object properties through parameters and variables in OpenSCAD. All agreed on the importance of generalizing model behaviors through the use of variables and avoiding the definition of object properties with hard-coded numbers. P6 said *“...everything is based on making the outer frame as a combination of parameters, so the parameters can change and then the model still works”*. However, participants mentioned that defining everything in terms of variables can be exhausting, so sometimes, they use variables only when they anticipate that a certain property will need to change. Moreover, working out the mathematical expressions for creating parametric models is perceived as very challenging. P6 expressed *“I sometimes have a harder time doing the*

“It is not evident to keep track of the constraints, especially in FreeCAD where you have many panels ...”

math (to define object's properties), using all the combinations of variables where they should be".

3.2.2.4 *Achieving digital precision*

One primary need of some participants ($n = 9$) when printing is to achieve precision. They perceive that OpenSCAD allows them to achieve accuracy in easier ways by explicitly expressing the sizes and dimensions of every placed part. P1 commented *"from Blender I could not make things precise (...) I use CAD programs for electronic devices, from there I get measurements. In OpenSCAD, it is very easy to be precise with this. I only need to make a box of this size or that size ..."*. P12 acknowledges that some direct manipulation programs allow one to express precise measures but it is not as clear and flexible.

Paradoxically, while users feel a sense of precision by being able to describe position and sizes explicitly, eight participants complained about the lack of means to check dimensions in the OpenSCAD view. P1 and P19 mentioned *"the problem is that I can not verify from the view the dimensions I am creating from the code."* and *"I had a really tough time taking measurements and showing measurements visually as part of the model."* As mentioned, spatial transformation involves several nested operations and variables, so verification is important. P14 said *"I wrote all these formulas, but did the resulting piece have the correct size and location?"*. To deal with this, participants use echo operations to print the expression results in a console (P4 *"I could put some echos to verify those formulas, but It would be much better measure on the screen."*) or visually inspect parts on the view by emulating a ruler (P11 *"...I put a cube of the right size next to it and just visually inspect if they are the same height"*)

"I wrote all these formulas, but did the resulting piece have the correct size and location?"

3.2.2.5 *Designing curves and organic shapes*

Nine participants said that, in general, they feel that OpenSCAD is not a friendly application for creating nonstructured curves and organic shapes that are difficult to define using mathematical expressions. For instance, creating smooth corners was reported as *difficult* and *painfull*. Participants commented: P4 *"(in OpenSCAD) making rounded edges is a pain"*; P13 *"It is just easier for the printer to 3D print something that's rounded (...) But designing it, that is really super hard to do in OpenSCAD"*. P16 *"That (rounding edges) is the most difficult part of OpenSCAD"*. Even using prebuilt OpenSCAD functions for this purpose is hard, such as Minkowski function. P16 and P1 mentioned *"the most difficult in OpenSCAD is figuring out how to do rounded edges and fillets and chamfers without using Minkowski because it is too time-consuming to render"* and *"...making smooth corners is possible, I use the Minkowski tool but the dimensions changes, it is inconvenient."*. However, three participants said that when the shape or curve can be defined mathematically, OpenSCAD is convenient for designing it. P10 discussed a project where he defined a Bézier curve that changed parametrically. Although the mathematical expression was

hard to create, it would work better to have a parametric design because the curve can be expressed in code.

On the other hand, this task seems significantly easier in B-rep direct manipulation programs. P6 mentioned *“in Fusion360 if I want to have a squared box and I want to have a nice rounded corner, I just can click on both faces and I have a nice rounded edge”*. This difference in difficulty between programming-based and direct manipulation applications in this seemingly simple task creates frustration for the participants. P4 expressed *“What makes it hard is that you can not point to a specific edge or corner and make it round. I don’t like that”*

3.2.2.6 Dealing with CSG

Some participants ($n = 7$) mentioned limitations inherent in the CSG representation. First, four participants said it is hard to verify the result when using the difference and intersect operations that remove volume. The removed parts are not visible, and verifying the correctness of the operations seems problematic. P4 commented *“When you do a subtraction, it is hard to figure out if you are doing it right, if the subtracted piece is in the right spot, and if it has the right shape”*. One mechanism to deal with this difficulty is using OpenSCAD modifiers (see section 2.1.4). However, modifiers require a trial-and-error approach, which is still time-consuming. P16 expressed *“(I use modifiers) with the invisible things, the things I’m subtracting. I’d said it’s an easy way to figure it out, but it always takes some time”*. Second, three participants expressed frustration that they were unable to individualize points or faces from the volumes as it is possible with B-rep. For instance, P4 mentioned it was hard not to be able to point a corner and round it from the view *“What makes (OpenSCAD) hard is that you cannot point to a specific edge or corner and make it a round, for instance, this is hard”*. A similar problem occurs when, for instance, the model requires two cubes to touch in one face. By placing one box at the end of the other, there is no guarantee that the objects are overlapping. CSG does not represent faces or vertices information but abstract definitions. Hence, users can only define two boxes with coincident faces by correctly defining their positions and sizes. Thus, it seems to be common practice to add small offsets to ensure overlapping as mentioned P13 *“You can’t have coincident faces; they have to go past each other. Every time you put one thing on top of another, you have to add those overlaps”*.

3.2.2.7 Versioning and collaborative work

Few participants ($n = 3$) highly valued the flexibility of code to use repositories such as GitHub for versioning and collaborative work. Although it is possible with direct manipulation programs such as FreeCAD, it is not that convenient, as expressed by P3 *“I especially like how well OpenSCAD checks in the GitHub repository ... this is a very collaborative project, and you can check in Fusion360 in GitHub but it just does not work. The*

“...you cannot point to a specific edge or corner and make it a round, for instance, this is hard”

repository becomes huge very fast. But OpenSCAD is text-based, making it very well suited to the collaborative environments we are already using for the source code."

3.2.2.8 Specifics of programming-based CAD

We discussed with the participants the advantages and limitations of programming-based CAD applications in general and specifically in OpenSCAD.

WORKING WITH SEVERAL FILES/PARTS It is normal to break down the model into parts when having complex models. In addition to facilitating the design process, it is an alternative when the part has to be printed in several parts. This scenario presented some difficulties for the participants ($n = 9$). OpenSCAD does not have any option to define parts in the design, so the participants can export parts individually. One solution mentioned by the participants was to create the design and later apply a difference and intersection with a cube. First, the participant would apply the difference and remove half of the model to export it to an STL file for printing. Later, the participant would use the same cube and apply an intersection, removing the second half to export and print. Once both parts are printed, they will be assembled. The result would not make parts easy to connect. Therefore, other participants used some libraries to split the model into parts to better assemble them.

In other cases, applying difference and intersection is not enough when the model is complex. Thus, participants create a file for each part. This allows them to isolate each part and focus their efforts without being distracted by the other parts. Nevertheless, the process of validating all the parts together is very difficult. For instance, P5 shared with us a model of a GoPro camera gimbal with many parts that form an articulated arm. He imported all the parts into the same project when he wanted to verify how it would look together. However, when modeling individually, all pieces are placed in the center. Thus, when importing all parts together, it is not easy to place them in the correct position and orientation only to verify the result.

DEALING WITH CODE Eighteen participants discussed issues and experiences managing code in OpenSCAD. In general, participants try to keep good coding practices to make their models easy to understand. They mentioned the importance of commenting on code, avoiding very long modules, organizing the code, and using expressive and telling names for variables and modules. However, they acknowledge that using expressive names for all variables is impossible. Moreover, using the expressiveness of the language is not always useful. For instance, P15 explained his model, which was extensively documented in his mother tongue, Slovakian.

Some participants ($n = 7$) discussed the challenges they face when dealing with code. According to them, problems include difficulties in easily finding parts in the code, keeping track of variables on complex models, and refactoring code. Moreover, we observed another challenge in the hands-on exercise. When participants were look-

ing for a code statement of a particular part, they would analyze and perform tests on the code that was not being evaluated. For example, elements created inside a conditional structure that was not evaluated. OpenSCAD does not warn users about this situation, and they realize this after spending some time analyzing the code.

Some participants ($n = 6$) stressed that OpenSCAD code editor is basic and lacks more advanced code editor features, as commented by P6 *“OpenSCAD really lacks richness in helpers how to write code, there is no autocompletion and that kind of stuff”*.

CODE-VIEW NAVIGATION In programming-based CAD, the description of the model (*i.e.* code editor) is separated from the model visualization (*i.e.* viewer). Thus, users must constantly switch between the code where they edit the model and the view where they validate the result of the modifications. As a result, navigating the model and making edits can be difficult [229]. Some participants opted to use Visual Studio Code (VS Code) [145] instead of using OpenSCAD code editor. They stated that VS Code allows for the installation of OpenSCAD plugins that streamline the coding process. The participants used VS Code to modify their code files and accessed a separate window within OpenSCAD to review the output. We observed the behavior of the participants in the hands-on exercise and discussed with them the challenges related to these tasks.

We identified a three-step search pattern when looking for code statements that create specific parts in the view. First, they would try to identify the block of code where the target part could be defined. Then, they would study the code to confirm the selected code statement logically. Finally, they would seek a visual confirmation in the view.

Participants had five strategies for trying to locate the code statement based on the view: rely on their memory, link the part to a variable and search the variable, guess how the part should be created and look for the pattern, follow the comments, and using OpenSCAD search feature (see section 2.1.4). As participants worked with their own models, some participants ($n = 12$) tried to remember how the model was built and relate it to their normal way of structuring the code. For instance, when P7 and P9 tried to find the code statement of a part, they said *“I know how the hex array (main frame of the model) is organized in the first play, so I would go here (scrolls the code until finding the hex_array module), ok here it is”* and *“well I know I started by creating a big cube for this”*. The second strategy was to link the target part with the variables and use a text search feature, as P17 commented *“I would assume that it’s related to this variable”* before searching for the occurrences of the variable. This strategy did not work well when the model repeatedly used the variable the participant picked to locate the target. The large number of occurrences made it difficult to study all of them to decide which code statement was correct. Moreover, OpenSCAD code editor features are basic and do not provide visual cues to help developers understand the code (*e.g.* highlight calls in the scroll bar of a selected variable or jump to the definition of a selected module

by clicking on the module call). Participants who use VS Code could easily follow the places in the code where a variable was used. In some occasions, participants using VS Code made mistakes by editing the code of a file different from the file OpenSCAD was rendering. They edited the code and did not see any change in the view until they realized the problem after some time. The third strategy was to try to think about how the selected part should have been created in the code and look for that pattern in the text editor. For example, the target element for P6 and P2 was a hole, so they started looking for a difference code statement. In the fourth strategy, when the model was well documented, participants used comments to understand the structure of the code and find the correct statement. Finally, participants could locate the code statement with the OpenSCAD search feature. Only three participants knew about these features and mentioned not using them normally. When seeking the code statement, participants always read the code to understand it and confirm that it was the target statement.

After the participants thought they had located the target code statement, they normally sought visual confirmation. The strategies used to confirm were removing the code statements and verifying missing objects, changing the parameter values of the code statement and verifying changes in the object's properties, using a color operations to highlight the object, and using OpenSCAD modifiers as a debugging task. We could identify some challenges when performing these strategies. Removing code statements can break the syntax of the code. For example, when a `translate` statement is written without opening and closing brackets, the transformation is applied only to the next code statement. The system will report an error if the code statement is temporarily removed and the next statement is not an object (e.g. variable definition). Also, using `color` operation does not work if the object is already inside a `color` scope. The system will override the statements, prioritizing the statement placed higher in the tree of operations. Participants frequently use modifiers for visual inspection but they often forget the correct syntax and characters to use. Moreover, modifiers do not work when they are used on 2D elements or code statements that are not being used (e.g. not evaluated conditional). In any case, all participants used a trial-and-error technique and required reading and understanding of the code.

Some participants ($n = 8$) mentioned some challenges they identified after the hands-on exercise. They acknowledge that reading and understanding the code is difficult. It becomes more challenging in complex models with long scripts, highly decoupled with several module calls, and with several parameters. They also mentioned that finding code based on the view is a repetitive, hard, and mentally demanding task. P13 commented “if I could point at something in OpenSCAD and say and tell me where this is, that would be a huge help. Especially if you have complex geometry, it's super hard to figure out just where you are.”.

“...if I could point at something in OpenSCAD and say and tell me where this is, that would be a huge help”

OTHER PROGRAMMING-BASED CAD CHALLENGES Eight participants mentioned some difficulties they identified when designing with OpenSCAD. We observed that participants are often surprised by the changes performed on the models because it is not possible to confidently anticipate the result of the edit. programming-based applications do not provide a transition between two states of the code. Consequently, understanding the impact of the changes made to the code is not always easy or obvious, as commented by P2 *“One of the difficulties is that you don’t have immediate feedback when you change something as a result of the screen; there are always delays.”*. Other challenges were related to difficulties in managing text elements in the view, and confusion with OpenSCAD units.

3.2.2.9 How to improve OpenSCAD

Some participants ($n = 14$) shared ideas about features that they considered would help their modeling experience. One key idea was the capability to easily identify parts of the code based on the view and vice versa. Participants thought that there could be features to help with this task. In particular, they would like to have a visual cue that helps them locate the code based on interactions with the view. P13 commented *“If I could point at something in OpenSCAD and tell me where this is (in the code editor), that would be a huge help. Especially if you have complex geometry, it’s just super hard to figure out just where you are.”*. Participants also commented on how to facilitate the task of applying spatial transformations. First, they mentioned the need to be able to measure distances in the view. They also commented that it would be very convenient to be able to extract spatial coordinates of objects directly from the view. P14 commented that *“There are cases where I want to know all those coordinates, but OpenSCAD doesn’t give that to you. I would like it to tell me what the bounding boxes of my model are. It doesn’t have to show me on the screen, but at least when I render it in the output area, it would tell me the bounding box and the center of mass”*. Moreover, seven participants indicated that they would like the system to assist them in retrieving the spatial coordinates in terms of the existing variables for parametric models by interacting with the view. P3 said *“(I would like) if I could click on this point (in the view) and OpenSCAD would automatically create an expression using the variables to describe that point (...) and then with this other point, create the constraints in terms of my variables (...) not hardcoded numbers because it is not useful anymore that way”*. Further, in addition to working out the expressions, they would like the system to allow the creation of constraints directly from the view. P5 commented *“So being able to simply point at an object to say, I want to attach this face of this item to that point there and for it to be able to calculate that distance would be such a timesaver”* and P17 *“we need some extra assistance to make it useful like guides where you can see like this is 15 degrees, this is 45 degrees or perhaps even like smart snapping where it’s like snap this part to some other geometry. I can think of times when that would have been useful instead of putting in multiple numbers, many times with trial and error to get it to the right spot”*.

“(I would like) if I could click on this point (in the view) and OpenSCAD would automatically create an expression using the variables to describe that point”

Finally, participants mentioned the need to facilitate some recurrent tasks for 3D printing. Specifically, they mentioned that they would like to have more elaborate libraries to perform usual actions, such as creating chamfers.

3.2.2.10 *Designing from scratch or re-using models*

We discussed re-using models from websites such as Thingiverse, Printables, or Github with the participants. Half of the participants ($n = 10$) expressed that it is normal to check pre-existing models from websites before starting to design one from scratch. However, the reasons for that differ according to several aspects. For instance, P8 said that he usually starts by searching for models in Google when it is something complicated. This is probably because P8 has a short experience with OpenSCAD and 3D printing. Nine participants mentioned that if they needed to print something very popular, they would definitely go to check other people's solutions first. P20 said *"if it's something that's really popular that I absolutely know that there would be models for (...) I'll just go get one rather than design yet another one"*.

Nevertheless, opinions on preferences between re-using models (and what using them for) or designing from scratch are divided.

DESIGNING FROM SCRATCH Six participants prefer to design from scratch because they have specific needs and look for very customized models. P4 commented *"I can design to fit my own setup ...for instance, my screwdriver holder, all the holes are a different size to fit my particular set of screwdrivers so that they fit snug..."*. Other participants ($n = 5$) expressed their satisfaction in challenging themselves to build models of good quality on their own, as mentioned by P11 *"I just like the process of designing and printing things and be ready to say, I've done this by my own."*. Furthermore, four participants said that starting from scratch is easier, faster, and has better quality results.

RE-USING PRE-EXISTING MODELS We discussed with participants their thoughts about using pre-existing models. We asked them if they use them, their motivations, limitations, and the model format they prefer to look for.

Most of the participants ($n = 16$) reported that pre-existing models do not meet their needs. Even when several models exist, they usually do not exactly fit participants' needs. And even if the model is parametric, available parameters rarely cover the modification participants want. Moreover, some models are too complex, with several parameters adding significant complexity to the printing process, so they are discouraged from using them. In addition, participants perceive several difficulties with the available model-storing websites. To start, six participants commented that searching on these websites is challenging. The naming system is deficient, so it is difficult to find useful models. Moreover, dealing with licenses can be problematic. Some participants use 3D printing for commercial applications, so using public models is not ideal for them. Participants also commented on the *Customizer* application

from Thingiverse. Although they find it useful, they also mentioned that it is very limited because authors can only upload models with one file, without the possibility of making references to other files or libraries.

However, the participants also acknowledge the potential of such websites. Six participants mentioned that pre-existing models can be time savers. They said that for printing popular objects, and for people with little experience in design, it is a good alternative. Some participants ($n = 8$) used pre-existing models as inspiration. These websites present alternatives for participant's projects in progress where they can collect ideas. Some of them mentioned being surprised to see things they never thought possible to do with 3D printing.

Although participants did not use pre-existing models to print them directly, some have incorporated or edited pre-existing designs for their projects. Fourteen participants commented on difficulties editing STL files in direct manipulation and programming-based applications. All of them agreed that finding manifold and printable geometries is rare. In most cases, they needed to fix broken geometries before being able to use them, which was reported to be "*difficult*" and "*painful*". Participants preferred to have a pre-existing model *with code* instead. The offer of models with source code is more limited and quality varies significantly. Often, available models are not coded following good programming practices, such as adding documentation, so reading and understanding the code is very hard. Some participants mentioned the importance of good-quality code if it is meant to be shared as mentioned P5 "*One thing is designing for yourself, and other designing for sharing*" This is not always the case and depends on the author.

*"One thing is
designing for
yourself, and other
designing for
sharing"*

SHARING MODELS Some participants ($n = 8$) had profiles on websites such as Thingiverse and Printables, where they shared their designs. They manifested that they enjoyed sharing their models, contributing, and seeing other people using and commenting on their models. Three participants said that although they liked sharing, they did not want to spend more time adjusting their models. Unfortunately, the models need to be adjusted to share parametric models in applications like Thingiverse Customizer. For instance, Customizer only accepts one-file models. P5 commented that they would like to share more of his models, but they often re-uses their own libraries that cannot be uploaded to the website. P6, on the other hand, said that they would like to customize the parameters to provide a better experience using sliders instead of input boxes, but would not spend time learning how to do it. Also, some participants mentioned that websites are not controlled, and they had seen users taking models from authors and selling them on other websites.

Table 4: Structure of theme *Printing*. Color intensity is proportional to the number of interviews coded with codes of the theme and subthemes.

Theme	Subtheme	
Printing (n = 19)	Motivations for 3D printing (n = 16)	It started with a family member or a friend (n = 3)
		What do I like about it? (n = 10)
		I like creating customized things (n = 7)
		I feel intellectually satisfied with achieving things (n = 3)
		I like tech (n = 2)
		Hobby (n = 3)
	What do I use it for? (n = 8)	I like to repair things (n = 3)
		I use it for work (n = 3)
		Prototyping (n = 3)
	Design for printing (n = 6)	
	Iterating the design after printing (n = 2)	
	Printing challenges (n = 15)	Clearances introduce uncertainty, making objects fit (n = 6)
		Material properties (n = 3)
		Printer limitations (n = 9)
		Limitations to iterate (n = 1)
		Other challenges (n = 8)
	Testing (n = 4)	

3.2.3 Fabrication

Some of the questions in the interview were intended to identify challenges in the fabrication process. We discuss our findings related to motivations and challenges.

3.2.3.1 Motivations for 3D printing

Participants use 3D printing for prototyping, for work, for repairing other objects, and as a hobby. They discussed what they liked about 3D printing. Five participants said that they were interested in technology and applications so it was almost natural for them to try 3D printing. Other participants mentioned that they have an *“intellectual satisfaction”* when they fabricate complex customizable things successfully. In addition, seven participants mentioned enjoying the fabrication process and especially having the physical result.

3.2.3.2 Design for printing

Three participants mentioned that designing and designing for printing are two different ideas. As explained by P11 *“Surely you can design everything in CAD, but having prints with many overhangs or supports is impossible. You need to design your models with printing in mind”*. They also commented that it is easy to find very complex models that, in practice, are not printable. When rendering a model in OpenSCAD, users can use a rapid preview or a more realistic rendering option. P3 commented that these differences add uncertainty to the design process because the way both rendering options process the information can lead the designer to errors.

“...you need to design your models with printing in mind”

3.2.3.3 Printing challenges

Participants ($n = 15$) discussed different problems encountered with 3D printing. Very often, they expressed that tolerances and clearances are factors that introduce high uncertainty on the success of the printed object. 3D printing does not precisely reproduce the dimensions and sizes in the digital design. P10 commented “ *I usually have to print multiple times to get the clearances correct, especially if there are moving parts. It usually takes several times to get the tolerances right*”.

Another challenge is the difficulty of taking into account the material property in the design process. Different materials have different behaviors that are difficult to include in the design because OpenSCAD does not support this information. For instance, how to design taking into account weak points in the design. P14 commented that “*most difficult in designing for 3D printing is making sure you’re accounting for the anisotropic strength properties of the material ...anything you print is weaker along layer lines, for instance*”.

Nine participants considered that knowing the printer they use is a key factor in minimizing uncertainty. They mentioned that hardware in 3D printing is sensitive to failure and requires good skills to set up for success, as commented by P19 “ *if it’s your own printer, you have a better sense of what to expect and you have agency over the state of your machine*”.

Other limitations were discussed. When the model is large, there is less flexibility to iterate due to the amount of material required. Moreover, participants mentioned that sometimes, putting models in a correct orientation for better printing is not trivial. Furthermore, some participants ($n = 4$) mentioned that printing with supports can be tedious, so they prefer to avoid it when possible.

3.2.3.4 Testing

Testing was important to avoid false printings and waste of material. Participants ($n = 4$) acknowledge that their only validation strategy was test prints. P13 and P16 said they would print layers to verify clearances before printing the entire piece. Although test prints are time-consuming, they were unable to find another testing mechanism to anticipate what the result of printing would look like.

3.3 DISCUSSION

We answer our research question **RQ1**: What are the motivations and challenges that users face when using programming-based CAD applications when designing models for 3D printing in personal fabrication? We discuss the motivations of programming-based CAD users and their challenges and limitations.

3.3.1 Programming-based CAD users in 3D printing

3.3.1.1 Users motivations

Findings in programming-based CAD user preferences (section 3.2.1) allowed us to identify two types of motivations when users 3D print. The utilitarian and the enthusiastic. Users with utilitarian motivation use 3D printing to solve problems pragmatically. They are more flexible in design decisions when there are limitations or a lack of motivation. For instance, participants expressed not liking re-using pre-existing models, but due to the time limitations, for example, P19 found pre-existing models a very convenient solution. He expressed *“As a father with a full-time job, it’s difficult to sit down and develop a model (...) So oftentimes, I’ll look for existing models, and if they work, I go with them because it’s often easier to have something that works”*. Similar situations occurred when printing common objects that participants felt they could find on the Internet.

In contrast, users with enthusiastic motivation invest time and energy in achieving neat and highly customized objects. Participants used very creative methods and could iterate several times to achieve a satisfactory result. P4 shared how a small model for a screwdriver hole changed over 5 iterations to achieve a satisfactory result. P14 had experimented with different creative solutions to capture the outline of curved objects he wanted to repair. When having an enthusiastic motivation, users are more likely to design from scratch rather than use pre-existing STL models with questionable quality and printability, or coded models that would require time and effort to understand. Moreover, some participants were moved by the feeling of pride of *“intellectual satisfaction”* when achieving a result. As commented by P16 *“I like OpenSCAD because it is very functional, not procedural. That’s an intellectual like ...functional languages (like OpenSCAD) are intellectually satisfying for me.”*

3.3.1.2 Perception on direct manipulation CAD applications

Programming-based CAD applications users often have a programming or engineering background, but some of them expressed having issues with the mathematical requirements that parametric modeling needs. Most participants opted to use OpenSCAD because they could reuse their previous experience with other programming languages. Also, most of them mentioned not liking direct manipulation CAD applications in their first interactions with them, and only a few participants had experience with them and used them for their workflow. We identified some common difficulties that users face with these applications.

Direct modeling applications (section 2.1.2), such as TinkerCAD, are perceived as too basic for doing *“serious”* 3D modeling. Parametric applications such as FreeCAD and Fusion360 presented similar challenges. First, having the different parts that describe the model, such as the tree history and the constraints panel, disconnected and

distributed in the interface is confusing. Participants mentioned how difficult it was to visualize all the constraints and relate them to the model. Moreover, although direct manipulation applications allow the use of mathematical expressions for doing parametric modeling, the definition of expressions is also difficult to manage and understand because the properties of the objects are defined in multiple places in the application. For instance, in FreeCAD, users must first create a spreadsheet containing the parameter's definition in cells. Then in a different window in the view, create constraints using the cells' definition to relate geometric properties. These constraints are also stored as a list in a different panel. So users have three different views for creating a parametric model, while in OpenSCAD, everything is defined in one place, the code editor. Moreover, creating coherent constraints can be challenging. P9 mentioned he was normally concerned to have an overconstrained warning when using FreeCAD, for instance. Another problem is related to the [TNP](#) (section [2.1.2](#)). Participants mentioned that introducing changes in the tree history in applications such as FreeCAD easily resulted in breaking the model.

3.3.1.3 *Fabrication as a hobby*

We identified that the fabrication process involved stages that users can or cannot enjoy. P7 mentioned that “(3D printing) It's actually three hobbies in one, which is why a lot of people find it very overwhelming (...) There's the modeling, setting and improving the 3D printer, and the actual making of the 3D prints”. Indeed, we could observe different enjoyment from the participants at the different stages of the process. For instance, P14 described a project that took him weeks to create a highly customizable model for pliers. On the other hand, P16 explained a workflow he developed to create objects with multiple colors on a single-color printer by playing with the printer settings and the design. Moreover, P8 said that, although he enjoyed the design process, what he liked was the result. Consequently, he rushed other parts despite the errors he could make, such as taking measurements with care. “I think like to go to the good part, I want to get to the fun, having the thing in my hand.” Previous studies [88] reported that novices were easily frustrated designing objects for 3D printing, probably because their motivations were to get the objects, not to design them.

3.3.2 *Programming-based [CAD](#) design challenges*

The interviews reveal challenges that programming-based [CAD](#) users face (section [3.2.2](#)), representing a research opportunity for the HCI community.

Programming-based [CAD](#) inputs are coded instructions that the machine uses to create a visual representation where the user can verify the result and perform edits in the code if required. Therefore, the user verifies in one space and edits in another in an intensive and iterative exercise. Consequently, users are forced to understand the

connections between the code and the rendered model, but current tools barely try to help users in this task.

NAVIGABILITY Our hands-on exercise revealed that users use several strategies to identify code statements responsible for creating specific parts based on visual inspection with considerable difficulty. Most of the participants went through the code to understand it, and in every case, they needed to make edits to seek visual confirmation. Programming-based CAD applications could facilitate this task by using the implicit connection between the code and the view. OpenSCAD provides a backward search (section 2.1.4) to place a cursor on a code statement based on a selected part in the view. However, it does not provide visual cues to confirm that the selected part is aimed, and users need to modify the code to seek visual confirmation. IceSL [127] provides better visual cues to highlight code statements responsible for creating parts in the view but does not discriminate between statements involved, so it is impossible to navigate the CSG structure. In Chapter 4, we propose enhanced navigation features that facilitate users to explore the model and understand the connection between the different parts in the visual representation with their corresponding code statements.

UNDERSTANDING SPATIAL COORDINATE SYSTEMS IN THE CODE Understanding 3D spaces on a 2D screen is a difficult task reported in direct manipulation [88]. Our interview revealed that this problem can be more difficult in programming-based CAD where the editing space is disconnected from the view, and no visual help to relate them. In other words, users have to mentally imagine the behavior in the view that a spatial transformation will have when stated in code. Moreover, CSG structures create nested scopes where the coordinate system is relative to the aggregated effect of spatial transformations performed previously. Participants mentioned the need for trial and error strategies to apply a spatial transformation successfully. The task becomes more complex when using rotations, which are generally more difficult to understand. Some participants tried to avoid nested transformations due to difficulty understanding them. Programming-based CAD applications could support the understanding of spatial properties when coding. For instance, when modeling, CAD applications could add widgets in the view to clarify the relative position and orientation of a specific object's center in each axis. Moreover, these widgets could have different colors to distinguish the translation axis, which the code editor could use in the space of the transformation parameters to make this correspondence explicit. Moreover, by clicking on the widgets, the application could open a text dialog to edit the corresponding parameter directly in the view, as is possible in some direct manipulation applications [93].

DEFINING GEOMETRIC PROPERTIES BASED ON PRE-EXISTING INFORMATION In CAD design, the geometric properties of an object are closely related to those of other

objects. For example, users may want to place a cube on top of another. Direct manipulation applications facilitate user interaction through visual aids such as rulers and volume highlighting during overlap, the use of snap effects that guide positioning during drag-and-drop actions, or the use of constraints [63, 93]. These applications facilitate the re-use of additional information from the model within the model. However, programming-based CAD makes this task more challenging. Applications such as OpenSCAD or JSCAD often limit interactions within the view, preventing users from selecting specific parts or re-using information, for example, the position of an object. Certain applications such as RapCAD [20] permit such operations within the code. For instance, users can use functions to extract an object's position as a parameter, store it in a variable, and later use this information to define another element. While this method enables the selection of objects to extract and re-use information in the code, it maintains a separation between the code and the visual representation, necessitating users to conceptualize the behavior mentally. Applications could use visual elements as rulers that allow for measurement or even retrieve raw positions, orientations, or sizes from objects directly from the view when clicking on parts of the model's visual representation. Implementing bidirectional programming [143] behaviors could support this task as has been done in SVG [78] and CAD previous work [106].

Two cases emerge in this context. First, applications could allow for retrieving raw positions, orientations, or sizes from objects directly from the view. For instance, users could select a cube and obtain the center's position. Applications might allow for selecting specific parts, enabling the extraction of fundamental information such as center position and size. This capability could extend to derived positions, such as determining the position of the corners of a cube, calculated using the cube's center position and size. Second, the application could articulate this information based on the variables used in the code to define it. Participants expressed difficulties working out mathematical expressions to define an object's position parametrically. The application could report how a geometric property was defined regarding variables, empowering users to automatically retrieve this information and create other objects based on it. Additionally, the application could allow the user to express the intent of certain actions directly on the view and assist in coherently implementing the code. For example, users could select an object, choose an option "next to another object," select a second object, and the system would create a `translate` statement in the code using variables to place the first object next to the second. This output-oriented behavior or bidirectional programming [143] has already been implemented in SVG [78] with different logics and even in programming-based CAD applications [106], but not to define geometric properties parametrically. In Chapter 5, we introduced features based on bidirectional programming that enable users to define objects' positions parametrically, relative to the positions of other objects, through interaction with the view.

In general, programming-based CAD could benefit from allowing a more enriched interaction in the derived information rendered in the view while coherently connecting

it with the code and supporting the edit based on this information. Moving towards bidirectional programming [78] or live programming [219] paradigms could bridge the gap existing between both spaces, code and view, which is one significant difficulty in programming [156].

3.3.3 3D printing challenges.

The interviews revealed problems in the process related to the disconnection between the CAD model and the target environment as described in section 3.2.3.

FIT BETWEEN THE DESIGN OBJECT AND OTHER PHYSICAL OBJECTS Users cannot consider contextual limitations, resulting in longer processes and creativity limitations [194]. P9 explained that the lack of context resulted in more iterations fabricating a case for an emergency button. Bridging the digital design and the physical environment can facilitate the design process. One approach is incorporating digital references of the physical environment into the digital design. Some participants upload STL replicas of objects into OpenSCAD and FreeCAD to have a reference of the object and design around it. Websites such as Thingiverse [213] or MyMiniFactory [151] offer models, mostly in STL format, that users can use as references when designing, although participants and previous work have reported some problems with the search engines and meshes quality [39, 129]. Another possibility is to bring the model into the environment. DesignAR [168], for instance, allows designers to work in augmented reality environments and place models in physical environments using direct manipulation. Programming-based CAD applications could explore having virtual or augmented reality previews for a realistic preview of objects. Moreover, these environments could also facilitate code interaction and understanding, as previously explored in other fields [85, 108, 182, 183]. Furthermore, expanding interaction in programming-based CAD can leverage bidirectional programming [78, 143].

INCLUDE PHYSICAL MEASUREMENTS Capturing data from physical objects has been also reported as challenging [110, 136, 167]. Participants reported difficulties in measuring curved and organic shapes using programming-based CAD. Research could explore sensing devices to capture and transfer information to CAD applications. Some participants use photogrammetry and scanners to capture the outline of curves and reproduce them digitally, but these solutions were found to be time-consuming, complex, and imprecise. Additionally, retrieved information as a point cloud is difficult to parameterize. An alternative solution is to use Bezier curves, where a few control points mathematically define a contour. Solutions like ShArc [185] can capture data from these control points with less effort, creating a suitable solution to replicate organic shapes parametrically. The use of augmented reality could also help to capture control points to create Bezier curves.

OTHER CHALLENGES IN 3D PRINTING Users go through different disconnected stages when 3D printing. CAD applications often limit functionalities to the design, ignoring limitations concerning the printing process. For example, participants used to apply spatial transformations to their models to locate them in an optimal position and orientation for 3D printing. However, if any edit was required, they removed these transformations to see the model in a more familiar position and orientation. Similarly, when participants needed to split a model into parts. They first finish the model and then apply a difference and intersection with conditionals to save two different STL files.

CAD applications could facilitate the design by including information related to the printing process. For example, the same model would not print the same on different printers or with different materials. One of the main factors of uncertainty expressed by the participants was the tolerances and clearances. After printing and detecting problems, they would have to go to the CAD model, adjust it, export it, and print it again. CAD software could inform users of possible problems related to the design complexity (*e.g.* need for supports), printer tolerances, printer capabilities (*e.g.* possible angles of printing), or materials properties when printing.

We identified other problems related to misuse of applications (*e.g.* using Blender [62] as an STL viewer) and licensing in model-storing websites (*e.g.* people using models from one website to make a profit in another website without authorization).

3.4 LIMITATIONS

The hands-on exercise was a short observation task rather than a controlled user study. Findings related to it may be limited, missing other challenges that users face in the design process. Moreover, having experience in direct manipulation programs was not an exclusion criterion. Consequently, some of the participants had no previous experience in such software, and their answers related to these applications were not based on a reasonable experience and understanding of the direct manipulation paradigm. An exploration with direct manipulation CAD applications users can provide a different perspective on the perceived challenges of these solutions. Finally, our interview only included OpenSCAD users. Although most programming-based CAD applications also use a CSG representation, each tool provides different features, and not all of our findings may be generalizable. Further, a few programming-based CAD applications that use B-rep representation, such as CadQuery, may provide a different user experience and challenges than the ones reported in our findings.

3.5 CONCLUSION

We interviewed twenty users of the most popular programming-based CAD application, OpenSCAD, to investigate their motivations and challenges in the design of 3D

objects and the 3D printing process. During these interviews, we included hands-on experience to observe behaviors and difficulties when navigating the model. With the information collected, we performed a reflexive thematic analysis in an iterative process, developing main themes related to the user's profile, design experience, and printing experience. Our findings reveal that users are motivated to use programming-based CAD applications thanks to their parametric capability, the possibility of using mathematical expressions, and the precision for 3D printing. Moreover, it reveals several challenges in connecting the code with the view, understanding and performing spatial transformations, measuring and designing organic and curve shapes, validating dimensions in the view, and re-using pre-existing models. Programming-based CAD could facilitate some of these tasks by enabling the information that the system stores and effectively communicating it to the user. Last, our findings also reveal difficulties in the 3D printing process, such as handling uncertainty introduced by printers and material properties, identifying code locations to perform correction based on physical inspection, and validation before printing.

Chapter 4 and Chapter 5 address specific design challenges inspired by the findings of this study.

NAVIGATION AND SPATIAL EDITING CHALLENGES

Note: The content of this chapter is the basis of a manuscript that has been published and presented in the Proceedings of 2023 ACM Symposium on Spatial User Interaction (SUI'23), held in Sydney, Australia, in Oct. 2023 [71].

A notable difficulty identified in Chapter 3 involves navigating and correlating the code with the 3D view, such as identifying the code statement responsible for a specific part of the model. Users often use various strategies to locate these code statements corresponding to particular model parts without the application support. Additionally, the study highlighted the reciprocal challenge where users struggle to determine the specific impact of a code statement on the model's visual representation. This bidirectional difficulty in linking code and visual elements significantly complicates the design process.

Another significant challenge reported is executing spatial transformations. Users must not only comprehend the existing code to identify the correct insertion point for new spatial transformation statements but also mentally visualize the effects of prior transformations on the coordinate system. This often leads to a trial-and-error approach, where users experiment with random values, seeking visual cues to guide the correct application of transformations. These findings underscore the complexity of the design process in programming-based CAD applications, where users frequently rely on repetitive and time-consuming methods to bridge the gap between code and visual output.

One alternative to bridge the “gulf” [89, 226] between the code and view spaces is the bidirectional programming approach. As elaborated in Chapter 2, bidirectional programming facilitates user interaction with both the code and its corresponding output. This method ensures coherence between the two, enabling changes in one to be reflected in the other. In essence, bidirectional programming allows for an information flow from the view back to the code, diverging from the typical unidirectional flow characteristic of programming-based CAD applications. Drawing on prior work in this domain, we propose alternative solutions based on the bidirectional programming approach to address the specific navigation and editing challenges faced by programming-based CAD users.

We introduce our modified version of OpenSCAD (Figure 17), featuring a navigation system that bridges code and the 3D model to make explicit their relationships. This system also supports object manipulation in the 3D view for tasks that are intu-

itively simpler than code modification. For instance, users can relocate an object component by selecting it in the 3D view using the mouse and relocating it through direct manipulation, prompting an automatic code update or the addition of a transformation to accommodate the move. Our contributions include (1) the conceptualization of bidirectional navigation and editing functionalities for CSG programming-based CAD applications, and (2) a proof-of-concept through a modified version of OpenSCAD.

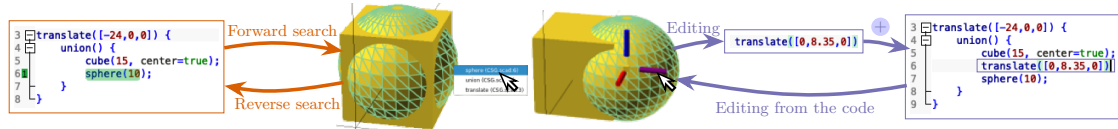


Figure 17: Bidirectional Programming features implemented in OpenSCAD. The system allows to navigate the code through direct manipulation in the view (reverse search) and vice versa (forward search). Also, the program enables modification of the 3D model from the view while the system updates the code coherently.

Based on the insights from the study of Chapter 3 and an explorative study, we propose design goals for a bidirectional programming approach: (1) facilitating the understanding of models by creating a navigation system that leverages the inherent connection between the view and code, and (2) easing the execution of spatial transformations through direct manipulation interactions within a programming-based environment. Implementing these design goals, we present our modified version of OpenSCAD (Figure 17). We implemented a navigation system between the code and the 3D view, allowing a better understanding of the relationships between the code and the different parts of a 3D model. Our modifications also integrate code editing by interacting with the objects in the 3D view for operations that appear more straightforward than modifying the code. For example, the user can translate an element of an object simply by selecting it in the 3D view with their computer mouse and adjusting its position with direct manipulation. As a result, the values in the code that define its position would be updated, or a transformation would be added to support the translation. Our contributions are (1) the design of bidirectional navigation and editing features for CSG programming-based CAD application; (2) a proof-of-concept in a modified version of OpenSCAD.

Our implementation is available as Supplementary Materials and on the website <http://ns.inria.fr/loki/bp>.

4.1 SPECIFIC RELATED WORK

The bidirectional programming approach facilitates a two-way flow of information between code and view. Beyond enabling model editing as detailed in Section 2.1.1.4, this approach also offers opportunities in navigability.

We will first discuss relevant studies on bidirectional navigation. Subsequently, we explore the existing capabilities within OpenSCAD for navigating between the code and the view.

4.1.1 *Bidirectional Navigation*

Programming-based CAD inputs consist of coded instructions that the machine uses to create a visual representation where the user can verify the result and edit the code if necessary. Therefore, the user validates in one domain (the visual representation) and modifies in another (the code) in an intensive and iterative exercise. For every modification, the user needs to locate where in the code modifications need to be placed before doing the edit. Consequently, establishing a link between specific elements of the visual representation and the corresponding code statements that generate them is essential.

Some applications, outside the scope of CAD, allow the user to perform bidirectional navigation providing mechanisms that make explicit the connection between both the code and the output. For example, SyncTex allows synchronizing a L^AT_EX source document with its corresponding PDF [123]. It is then possible to click on a sentence in the PDF viewer to jump to the corresponding line in the source document or click on a line in the source document to display the corresponding paragraph in the PDF viewer. Similarly, modern web browsers have an inspector that allows users to navigate between elements in the web view and HTML source code. To avoid switching between a code editor and the corresponding visual rendering, Gliimpse introduced a way to create smooth in-place transitions between markup code and its visual rendering [53].

In programming-based, some CAD applications allow a certain level of navigation. IceSL [127] is a programming-based CAD application implementing a CSG representation. IceSL highlights the corresponding instructions when an object is clicked, as shown in Figure 18. However, it is impossible to differentiate between the different CSG nodes that contribute to creating the clicked part.

In general, navigation capabilities between the code and the view in programming-based CAD applications are limited and nearly nonexistent.

4.1.2 *Navigation in OpenSCAD*

Drawing on insights from Chapter 3, users typically need to undertake two actions to connect the view with the code. Often, after visually inspecting a model in the view, users must identify the code statements responsible for generating a specific part of the model, *i.e.* navigating from the view to the code. Additionally, users seek to determine the impact of particular code statements on the model within the view,

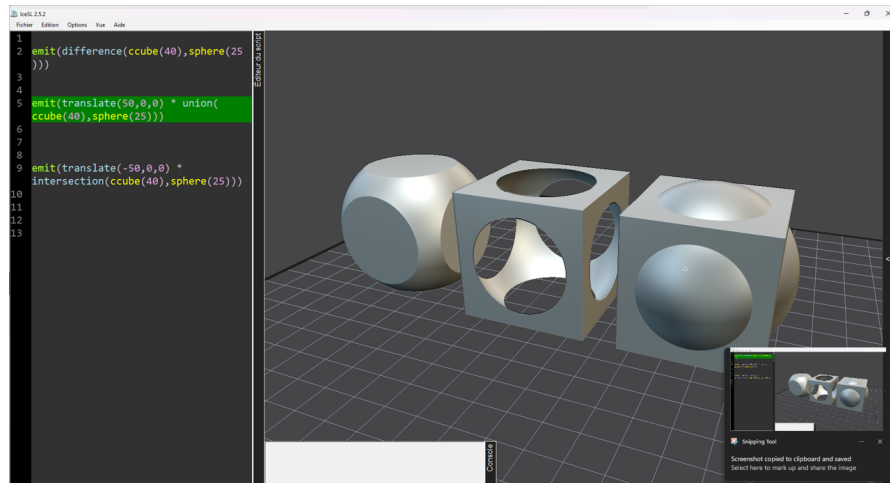


Figure 18: IceSL allows users to click on elements in the view while the application highlights the corresponding code.

i.e. navigating from the code to the view. OpenSCAD offers limited functionalities to somewhat facilitate these tasks.

4.1.2.1 *Navigating from the view to the code*

To help understand the relationship between the code and the 3D model, the last version of OpenSCAD introduced the ability to jump to the source code from the 3D preview. Users can click on the model, and the application will display a menu showing the different code statements that contribute to creating the clicked part. The text editor will place the cursor on the corresponding code statement by clicking on the menu items. Although this feature can help to locate where in the code a part is created, it is very limited in creating an explicit connection between the code and the view. Consider part of the example `CSG.scad` provided by OpenSCAD in listing 2. To start, it is impossible to select the removed volumes, *i.e.* the subtracted sphere in the difference statement, because it does not have representation in the view (Figure 19). Moreover, when clicking on one of the other statements, it is impossible to confirm what part of the view corresponds to the statement nor what the statement's scope is in the code editor. Additionally, it is not possible to determine what part of the visual representation corresponds to the selected statement.

4.1.2.2 *Navigating from the code to the view*

Users often need to identify the specific impact of a code statement on the view. Despite strategies like inserting a color statement to differentiate parts of the model or

```

1 translate([24,0,0]) {
2     difference() {
3         cube(15, center=true);
4         sphere(10);
5     }
6 }

```

Listing 2: OpenSCAD Example

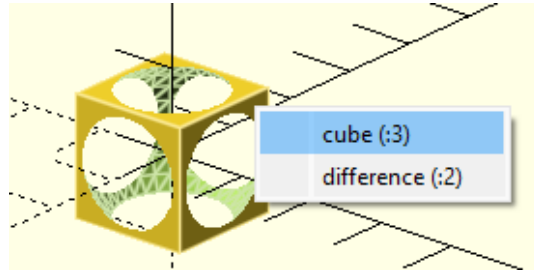


Figure 19: OpenSCAD displays a menu containing all the statements contributing to creating the right-clicked part.

adjusting parameters to observe visual changes highlighted in Chapter 3, users can utilize OpenSCAD *modifiers*¹.

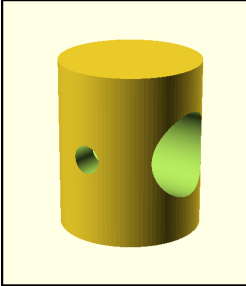
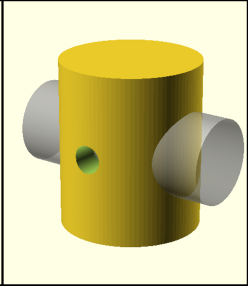
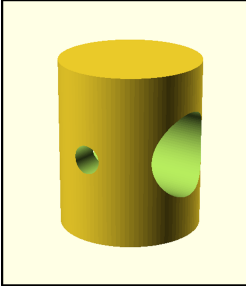
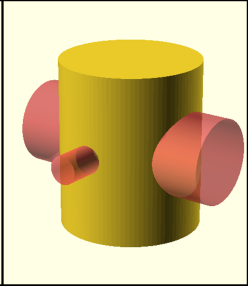
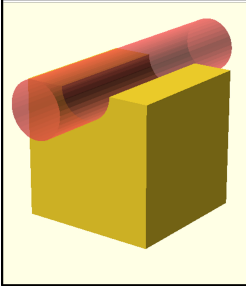
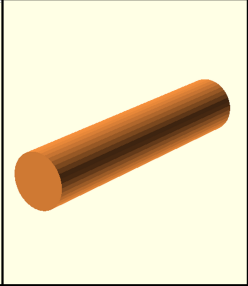
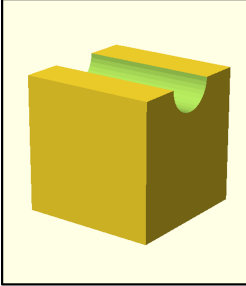
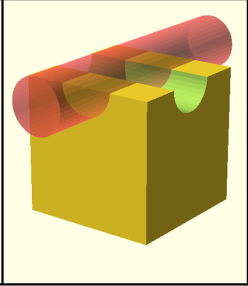
Modifiers are special characters that precede code statements, altering the model’s visual representation to simplify code understanding. Specifically, modifiers influence the way that OpenSCAD processes the CSG tree as illustrated in Table 5. Modifiers provide the following features:

- **Background:** By placing the character ‘%’, the application will ignore the subtree created in that code statement for the normal rendering process and draw it in transparent gray (all transformations are still applied to the nodes in this tree).
- **Debug :** By placing the character ‘#’, the application will use the subtree created in that code statement as usual in the rendering process but also draw it unmodified in transparent pink.
- **Root:** By placing the character ‘!’, the application will ignore the rest of the design and use the subtree marked as the design root.
- **Background:** By placing the character ‘*’, the application will simply ignore this entire subtree.

Modifiers facilitate altering the visual representation of the model, aiding in comprehending the impact of specific code statements on the view. Regardless, their utilization requires code modification. Additionally, users frequently forget the exact effect or the specific character, leading to a practice of trial and error, as detailed in Chapter 3. Our approach is inspired by the functionality offered by OpenSCAD modifiers.

¹ https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/Modifier_Characters Accessed: 10/01/2024

Table 5: Code example and result in preview of applying modifiers in OpenSCAD

	Code Example	Preview	
Background (%)	<pre> difference() { cylinder(h = 12, r=5, center = true); rotate([90,0,0]) cylinder (h = 15, r=1, center = true); %rotate([0,90,0]) cylinder (h = 15, r=3, center = true); } </pre>		
Debug (#)	<pre> difference() { cylinder(h = 12, r=5, center = true); #rotate([90,0,0]) cylinder (h = 15, r=1, center = true); #rotate([0,90,0]) cylinder (h = 15, r=3, center = true);} </pre>		
Root (!)	<pre> difference() { cube(10, center = true); translate([0, 0, 5]) { !rotate([90, 0, 0]) { #cylinder(r = 2, h = 20, center = true);} } } </pre>		
Disable (*)	<pre> difference() { cube(10, center = true); translate([0, 0, 5]) { rotate([0, 90, 0]) { cylinder(r = 2, h = 20, center = true);} *rotate([90, 0, 0]) { #cylinder(r = 2, h = 20, center = true);}}} </pre>		

4.2 DESIGN

Building upon our previous findings, we initiated an in-depth investigation to identify the fundamental causes of the challenges in navigation and editing within programming-based CAD applications. Our first step involved an exploratory analysis of OpenSCAD models from the Thingiverse website. This analysis aimed to understand better user needs and the root problems users face. Subsequently, we formulated specific design goals for a bidirectional programming application tailored to these challenges. Informed by these objectives, we developed and implemented a solution within the programming-based CAD application OpenSCAD. This implementation is a practical approach to addressing the previously identified navigation and editing difficulties.

4.2.1 Initial exploration

The study detailed in Chapter 3 highlighted challenges users encounter when attempting to connect the code with the visual representation of models. We analyzed the code of twelve Thingiverse models to investigate these challenges further. We download the models after filtering Thingiverse models, selecting those labeled as "Customizable" (*i.e.* including an OpenSCAD file) and "Popular all time.". The length of the models' code ranged from 63 to 392 lines of code (average: 189.6, standard deviation: 110.9). Our analysis involved editing these models to fulfill specific requests made by users on Thingiverse or to implement improvements we considered as beneficial. For example, for the model ID:421886², titled "*Customizable drawer box with hex pattern sides*", one user requested the addition of holes for magnets on each surface to allow multiple prints to adhere to each other, along with customization options for magnet dimensions. Through this process, we focused on identifying areas within the code that were difficult to navigate, repetitive, or particularly time-consuming to edit. This led to recognizing two prevalent issues: navigating the model based on its visual output and modifying it.

The first challenge relies on the constant need to mentally associate specific code statements with their corresponding elements in the visual representation and vice versa. Upon visually inspecting the model, users must identify the relevant modification code statement. However, due to the disconnection between the code and its visual output, the users are often required to confirm the accuracy of the targeted code statement. One method to address this is using OpenSCAD's search functionality, which allows users to locate code segments based on visual observations. Nevertheless, this approach often lacks contextual information and feedback, making it unclear if the selected part is indeed the one requiring modifications. The context menu accessed via right-click (Section 4.1.2) provides several options, yet it may not always be apparent which code statement is the target one. Additionally, code within difference and

² <https://www.thingiverse.com/thing:421886>

intersect blocks, which are visually subtracted, are inaccessible through this feature. Another strategy involves analyzing the code to associate it with visual components and variables, as discussed in Chapter 3. Both approaches are time-consuming and require verification.

Verification relies on visual feedback, aligning with findings from Chapter 3. Each confirmation method is essentially a trial-and-error process that involves modifying the code to employ modifiers, inserting a color statement, adjusting parameter values, or removing code sections. Modifiers require memorization of their characters and functions. Color statements fail with subtracted elements and may conflict with other color statements. Altering parameter values can become complex, especially when parameters influence other objects of the model. Removing code sections might result in misunderstandings if the corresponding contribution is hidden in the view.

The second challenge involves understanding the code's logic to execute modifications. Spatial transformations, such as translate, rotate, and scale, require an understanding of the view's coordinate system in relation to the code's parameters. Determining the impact of modifying a spatial transformation parameter on the visual output is not straightforward. This complexity escalates when additional transformations are applied, complicating the mental visualization of their effects on the model's orientation and scope center. The immediate recourse to trial and error can be both challenging and time-consuming.

4.2.2 *Design goals*

We establish two primary design goals for our approach: 1) improving the navigability of the system and 2) facilitating spatial editing.

4.2.2.1 *Improving the navigability of the system*

Typically, the user codes and the system compiles and renders. A direct relationship exists between code statements and the different parts of the model. After visually inspecting the output, the user returns to the input to modify it. However, to do this, the system's assistance in locating the precise place in the input through existing relationships to modify the output is practically nonexistent. Thus, as noted in the participants' interviews and our exploratory exercise, the user needs to make this trip back from the output to the input on their own.

The system must provide interactive ways to inform users about the links between code statements and the view to facilitate navigation. Using identifiers with visual cues, such as OpenSCAD modifiers, with effective search mechanisms can significantly facilitate the design process. For instance, the user could click on a pixel in the view, and the system would show the different code statements that create it. Moreover, the user could select a code statement while the system would color the corresponding

subpart in the view and highlight the code statements. This type of navigation should also be available for objects in the design that do not have a visual representation (*i.e.* elements removed from the model in intersect and difference statements). Also, the system could provide a mechanism to visually isolate the contribution of a specific set of code statements in the view.

4.2.2.2 *Spatial editing*

The users find difficulties in performing spatial transformations in programming-based CAD applications due to the lack of visual assistance. Our previous study also revealed that this task is considerably easier to perform in direct manipulation programs.

The system must provide direct manipulation actions to perform spatial transformations while keeping the code coherent. For example, the system could select a subpart in the model. The system would add visual cues to inform the current position and orientation of the subpart. The user would then perform edits through drag-and-drop mechanisms while the system adds the necessary changes to the code.

4.3 BIDIRECTIONAL PROGRAMMING FOR PROGRAMMING-BASED CAD

We created a proof-of-concept of bidirectional programming for CSG programming-based CAD application by modifying the code of OpenSCAD. Before explaining the features we added to OpenSCAD, we describe its overall architecture.

First, OpenSCAD parses the code to create an Abstract Syntax Tree (AST) [2], which is a structured interpretation of the OpenSCAD language. Then, it processes the AST by identifying the instantiating statements and evaluating the expressions (*e.g.* variables, loops, functions) to create an *Abstract CSG Tree* [59]. Each node in this tree is identified with an id number and represents an element that contributes to the creation of the model and is a module instance. The tree leaves are always primitives (*e.g.* spheres or cylinders). Intermediate nodes can be boolean operations (*e.g.* union), transformations (*e.g.* translate), or groups such as control structures (*e.g.* conditionals or loops). Each node in this tree represents an element that contributes to the model's creation and is a module instance. Subsequently, OpenSCAD uses the CSG to compute a mesh hierarchy that contains the 3D points, normal vectors, and colors of all nodes in the CSG and stores it in a *Geometric Tree*. Finally, OpenSCAD uses this tree to render the objects in the 3D view.

In *preview* mode (2.1.4), the rendered objects retain the ID of the CSG leaf node. When the user right-clicks on an object, the application retrieves the ID of the CSG leaf node. Subsequently, it iterates through the CSG tree to locate the branch containing the selected node. The application then generates a menu listing the branch's nodes and presents it to the user, allowing them to select a specific node.

We present the new features of OpenSCAD in three categories: (1) reverse search navigation, (2) forward search navigation, and (3) transformations with direct manipulation. We illustrate these features with the 3D model depicted in Figure 20.

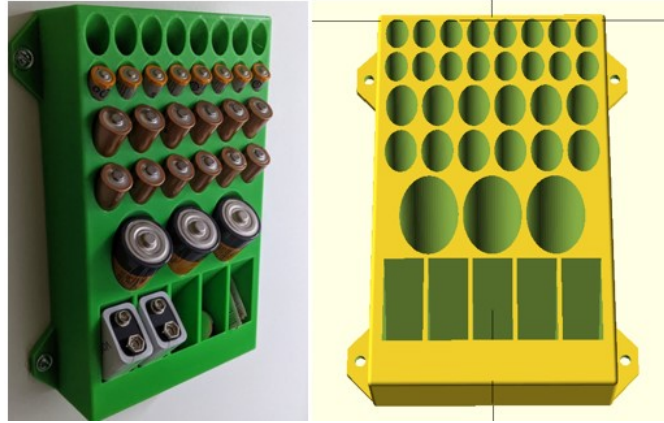


Figure 20: A battery box model from Thingiverse ³Left: After 3D printing. Right: 3D view in OpenSCAD

4.3.1 Reverse Search Navigation

Reverse search allows users to explore the code by interacting with the 3D model and select elements. For instance, the user wants to locate the line of code that creates a specific element of the model (*e.g.* the holes for the batteries in the battery box). Using reverse search, the user can hover or select an element in the 3D view and get visual feedback related to that element both in the 3D view and in the code editor (Figure 21).

We modified the source code of OpenSCAD. Special flags were added to the `CSG` nodes, allowing the application to designate them as *target* or *impacted*. When selecting a part, the application marks the chosen node as a target. Subsequently, it utilizes the `CSG` node to access the corresponding `AST` node and traverses the `CSG` tree to identify other nodes generated by the same `AST` node. Nodes within the `CSG` tree, distinct from the selected one but originating from the same `AST` node, are labeled as impacted. For example, primitives generated inside a loop or through multiple calls to a user-defined module result in different `CSG` nodes created from the same `AST` node. In essence, the *target* element denotes the node within the branch of the `CSG` node selected by the user in the view. The *impacted* elements represent other `CSG` nodes created from the same code statement or `AST` node as the selected part.

³ <https://www.thingiverse.com/thing:5485266>

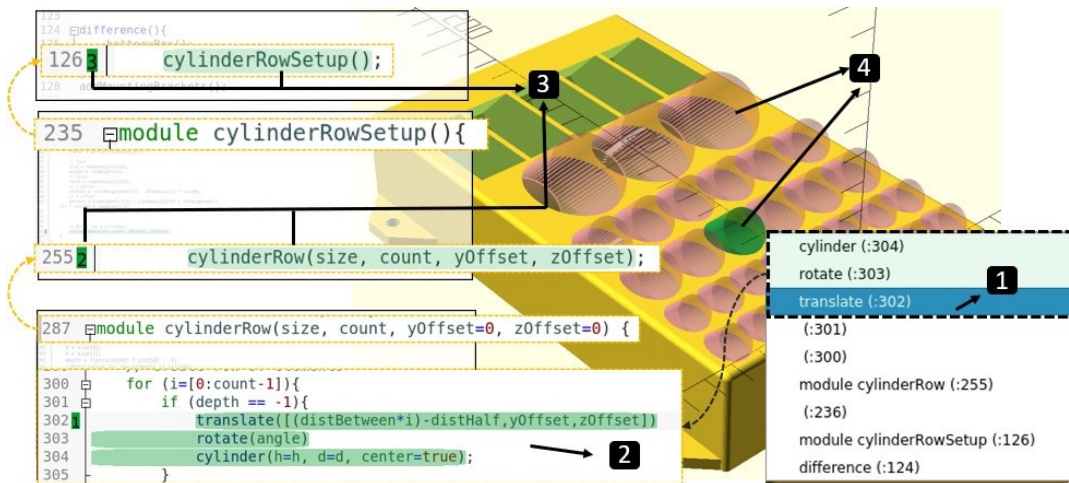


Figure 21: Reverse navigation features. **F1:** (1) The user hovers items in the contextual menu to select an element. **F2:** (2) The code of the selected element is highlighted in green. (3) Instantiating statements are also highlighted in green and marked in the margin with the call order. **F3:** (4) The 3D view shows ghosts of removed elements from differences, highlights the selected element in green and impacted elements in pink

F1. BROWSE THE CSG NODES OF AN ELEMENT In the original OpenSCAD, a popup menu appears when the user right-clicks an element in the 3D view. The items of this menu represent all the nodes in the **CSG** tree from the clicked element up to the root and the line number of the associated instruction in the code.

When the user clicks and *selects* one of these menu items, the menu disappears, and the cursor of the code editor moves to the corresponding code statement. If the user wants to locate the code of the other nodes, they need to click again on the same element in the view and *select* a different item, which breaks the navigation flow. Also, OpenSCAD does not differentiate the elements in the 3D view, and the user may be unable to make this distinction. Hence, navigation with this selection process also exposes the user to accidentally clicking on a different element between trials, causing an error by exploring an element different from the one initially *selected*. We improved this feature by *selecting* elements by hovering the pointer over the menu items. Therefore, the user can browse the different pieces of code that created this element without closing the menu and the node references. It helps them identify the instructions they are searching for and navigate through the code, mainly when the instructions are scattered across different parts of an extended code.

F2. HIGHLIGHT THE SELECTED ELEMENT IN THE CODE EDITOR When selecting an item in the 3D view, the code displays visual cues to inform the user about the relationship between the selected element and the code.

The application identifies the branch containing the target part. The code editor adds a number in the margin of the branch's nodes, indicating the instruction's call order in the call stack. It also adds a green highlight with decreasing intensity. Further, the system highlights the code of the impacted nodes in pink. It indicates these elements will change if the user edits the selected element.

F3. HIGHLIGHT THE SELECTED ELEMENT IN THE 3D VIEW We implemented visual feedback in the 3D view, following the logic on the code editor, to make the connection between the code and the 3D model evident. First, the system colors the edges of the selected item in green to mark it as selected. Moreover, it colors the edges of the elements corresponding to the impacted nodes in pink. It explicitly shows the parts that would change if the user edits this code.

Intersection and difference operations subtract the volume of elements (2.1.3.1). Elements that produce these operations are not all clickable in the 3D view. To address this limitation, when the selected element is one of these operations, we draw the elements used in its creation as ghosts. Now, the user can see and select these elements. Ghosts are also classified as targeted or impacted and are duly colored in semi-transparent green or pink.

To implement it, the renderer interprets the selected or impacted flags from the CSG nodes to add semi-transparent volumes in the view when the selected element is an intersection or a difference operation. It adds either a semi-transparent green or pink color consistently with the code color scheme.

4.3.2 *Forward Search Navigation*

Forward search is the opposite of reverse search. It allows users to explore the 3D view by interacting with the code. For example, the user would like to understand which elements of the 3D view are created by a specific expression in the code. Using forward search, they select this expression in the code editor, and the system highlights the impacted elements on the 3D view. These features are illustrated in Figure 22.

F4. FORWARD SEARCH IN EXPRESSIONS CORRESPONDING TO AN ELEMENT Instantiating statements contribute to creating at least one node in the CSG tree. When the user selects two or more characters from such an expression and presses the **F1** key, the system highlights the resulting elements in the code editor and the 3D view.

The application iterates on the CSG tree, looking for all the nodes created by the selected statement. If there is only one, the application marks the node as targeted; if there is more than one, it marks all nodes as impacted. Then, the code and the model apply visual feedback as explained in F2 and F3.

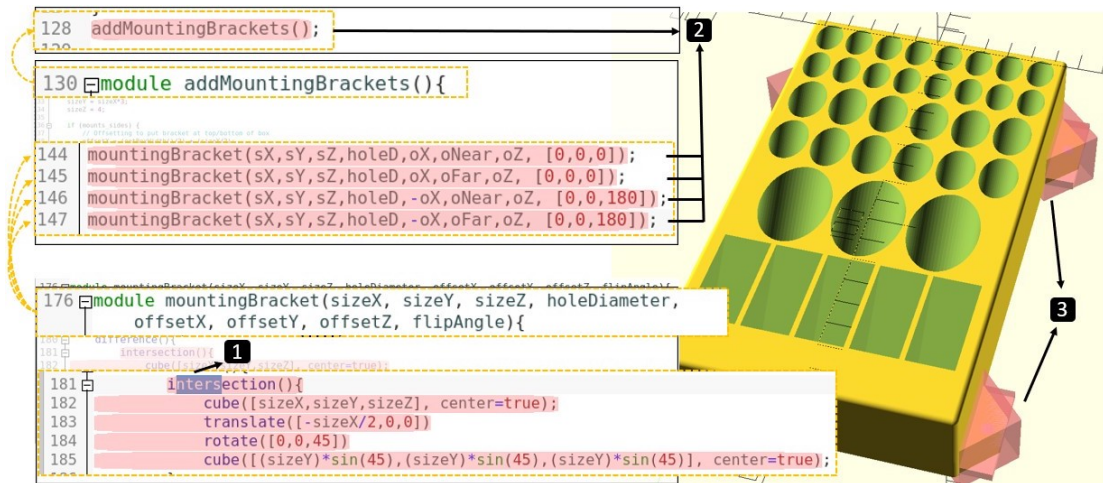


Figure 22: Forward navigation features. **F4**: After selecting a portion of an instantiating statement, (1) all the instance creations of the selected code are highlighted in pink, and (2) all the resulting elements are highlighted in pink in the 3D view (3).

F5. FORWARD SEARCH IN VARIABLES Variables are used in arithmetic expressions in the instruction parameters. Therefore, modifying these variables affects the elements defined by these instructions. When the user selects two or more characters from a variable and presses the **F1** key, the system identifies all affected nodes and highlights in pink all the expressions in the code editor affected by this variable and the corresponding elements in the 3D view.

4.3.3 Transformations with direct manipulation

CAD software typically allows users to perform transformation operations, such as translations, rotations, and scaling through direct manipulation action on the elements on the view. It is convenient because users can immediately validate the result and quickly set the value according to this validation. The same task through the code requires multiple trials and errors. We implemented similar features in OpenSCAD. For example, when users want to translate an element, they select it in the 3D view and click a *translation* button in the toolbar. A translation gizmo appears in the relative position and orientation of the selected object (*i.e.* applying previous translation and rotation from the root to the selected object), and they can drag-and-drop one of the three axes to translate the element accordingly. The element moves continuously and the system modifies the code simultaneously. As moving the pointer produces large changes, the user can use the mouse wheel to make small changes of 0.1 units to achieve precise edits.

To achieve this, we have modified the source code of OpenSCAD, enabling direct interaction with the view by adding buttons to activate the three basic spatial transformations: translate, rotate, and scale. When the user interacts with the gizmo for each transformation, the application calculates the corresponding value for editing based on the user's pointer position and the viewport camera's position and orientation. We have ensured that the application does not generate unnecessary code. Therefore, when modifying a node, the application searches for `CSG` nodes within the tree that can be modified to achieve the desired result. If it finds one, it modifies the parameters coherently. Otherwise, it creates and injects the code, as explained in the following sections.

Modifying elements in the view can result in various possible modifications in the code. This presents scenarios where the application needs to interpret the user's intent and decide [78]. Particularly, when modifying an element in the view where the application has identified impacted objects, one solution could be to apply the change to all elements. Alternatively, the application could make the necessary changes in the code to affect only the target `CSG` node. We followed Sketch-N-Sketch rationale for these cases [78]. We assume that significant modifications must be initiated by the user. Therefore, when performing a modification, the application warns the user that other parts may be impacted by that modification (*i.e.* through visual cues on *impacted* `CSG` nodes) and modifies the object, allowing changes to propagate to other elements. This implies that the application only modifies the code by changing spatial transformation parameters or injecting spatial transformation code statements. It will not add more complex programming structures such as conditionals.

This feature is illustrated in Figure 23.

F6. TRANSLATION FROM THE 3D VIEW When the user translates an element in the 3D view as described before, the system adds a `translate` element in the `CSG` tree and the code. It adjusts the `x`, `y`, or `z` parameter depending on the gizmo axis the user is dragging. The system does not add another `translate` element if an existing one only affects the translated element.

F7. ROTATION FROM THE 3D VIEW The rotation of elements is similar to the translation described above. The system places a rotation gizmo at the relative position and orientation of the object, which is the rotation center. Then, the user adjusts the rotation axes through drag-and-drop. Similarly to translations, the system only adds a `rotate` element if necessary; otherwise, it modifies an existing one.

F8. SCALING FROM THE 3D VIEW The user can resize an element directly from the view. We added two options in the menu for this purpose: *Scale* and *Scale primitive*. The user can perform the *Scale* option with any selected part. If it is the only child of a `scale` element, the system updates the parameters of this `scale` element. Otherwise,

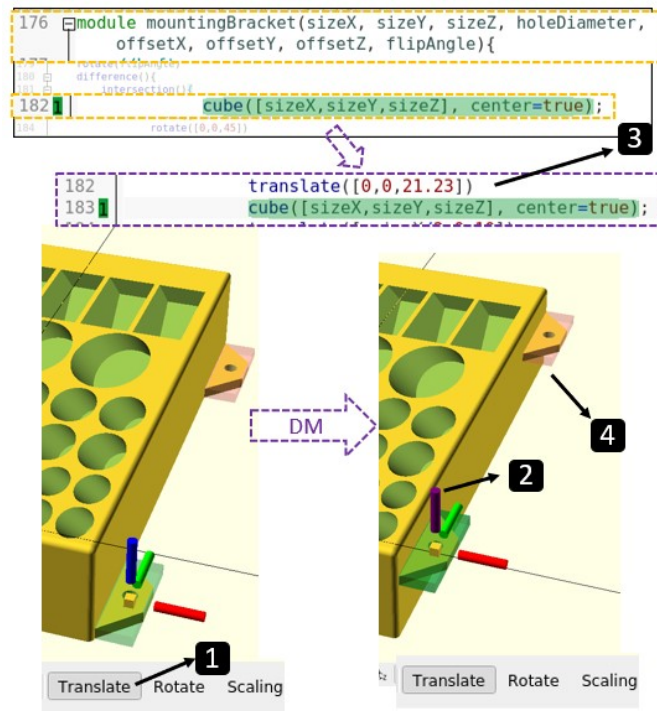


Figure 23: Simple 3D view editing. [F6](#): (1) After selecting an element, the user enters editing mode by clicking on the translate button and a gizmo appears. (2) The user clicks and holds the z-axis and moves the pointer to the desired position. (3) The system infers the code changes by adding a translate statement. (4) All elements impacted are also updated in the view.

the system adds a new scale element. Likewise, the user can perform the *Scale primitive* option if the selected part is a primitive. The system will update the instantiating parameters.

4.3.4 Informal validation by example

We aim to show how our system addresses the design goals. Specifically, we demonstrate it allows the user to (1) Navigate interactively between the code and the 3D model making explicit the relationship between them, including removed elements from difference and intersection operations. (2) Isolate the contribution of specific code statements. (3) Perform spatial edits on the model without the need to fully understand the code. We explored 11 models on Thingiverse under the “Popular Last 30 Days” and “Customizable” filters. These models have on average 195 lines of code (sd 113). We performed modifications requested by Thingiverse users in the comments section of the models. We describe below the case of a buckle box⁴ (Figure 24), which has two parts linked by a hinge.

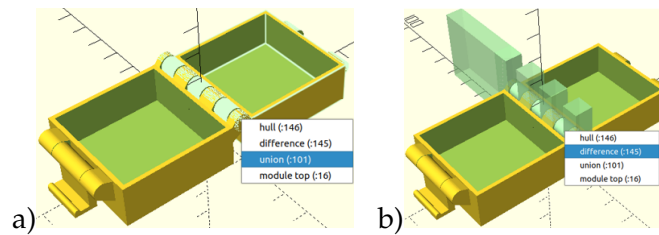


Figure 24: 3D View when highlighting a) the union item; b) the difference item.

First, we explored how the hinge structure is built (1). With one click on the hinges, the system displayed the menu of involved elements. We hovered the pointer on the elements of this menu to explore the structure of this element. The module top defines a union between a part of the box and a part of the hinge (Figure 24). The part of the hinge part is created by a difference statement between a hull and a set of cubes. When hovering over the difference statement we can observe the parts used to remove some volume of the hull, represented by transparent green cubes.

By right-clicking on one of these cubes, we repeated the navigation exercise. When selecting the subsequent `translate` statement (Figure 25), we quickly see that all cubes are created by the same statement when the system colors pink some of the elements in the view. We confirmed this in the highlighted code which shows the statement inside a loop structure. With two clicks, we could picture the code structure of a module of 84 lines of code of the model.

Then we looked at the code to understand its logic (2). For example, we look around further in the code and find a comment indicating the start of the description of a “new

⁴ <https://www.thingiverse.com/thing:82620>

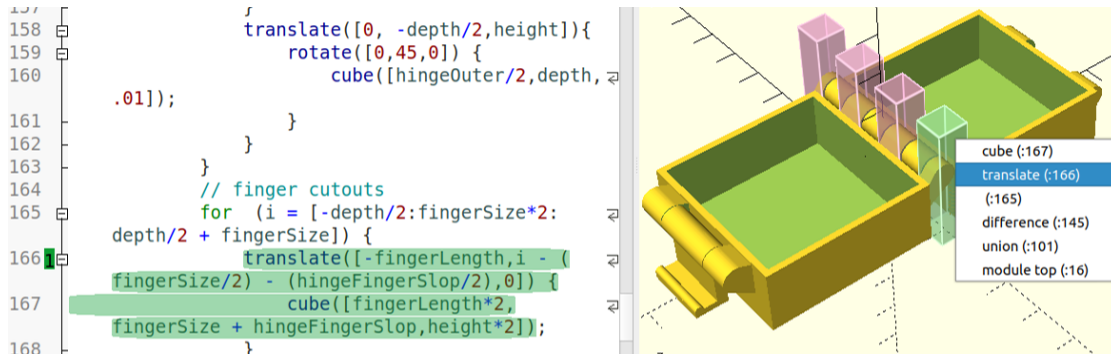


Figure 25: After right-clicking on one of the transparency, code editor and 3D view highlighting the translate item.

latch". By placing the cursor on the first code statement and pressing **F1**, we could see the complete scope of the code by the highlighted text and isolate its contribution on the view with the highlighted elements and added transparencies (Figure 26). Then we clicked on the transparencies on the view to observe each part individually in the code.

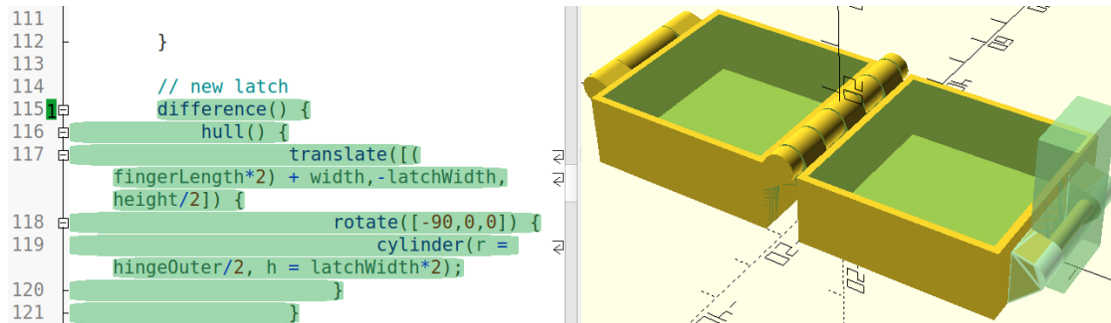


Figure 26: By placing the cursor on a code statement and performing a forward search, the system highlights the code and the 3D models consistently.

Last, we checked that we could perform spatial edits in the model through the 3D view (3). For example, we aimed to perform an operation that some of the participants mentioned. Once they finish the model, they often reorganize it to print it in an efficient way. We further explored and realized that the model defines 3 parts. We then selected each of these parts and performed spatial operations directly on the view to place the different parts to print them. We started by translating and rotating only with direct manipulation actions the view (Figure 27).

We moved the parts in a satisfactory position and orientation while the system updated the code accordingly (Figure 28). We referred to the code to fix imprecise values in the spatial operations. We could easily reorganize the elements from the

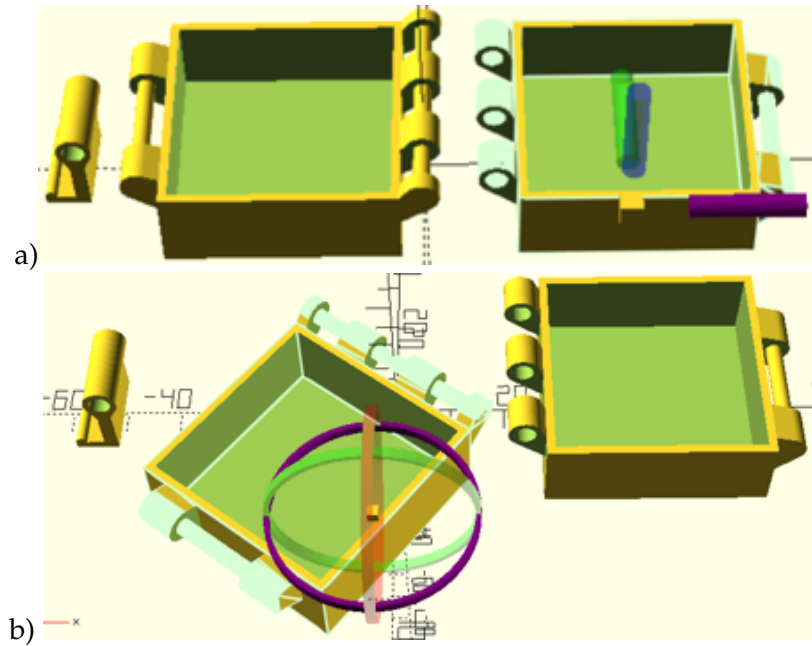


Figure 27: Editing the model with a direct manipulation a) translate and b) rotate transformations on the view.

view without needing to calculate the exact angles or on what axis the edits had to be done.

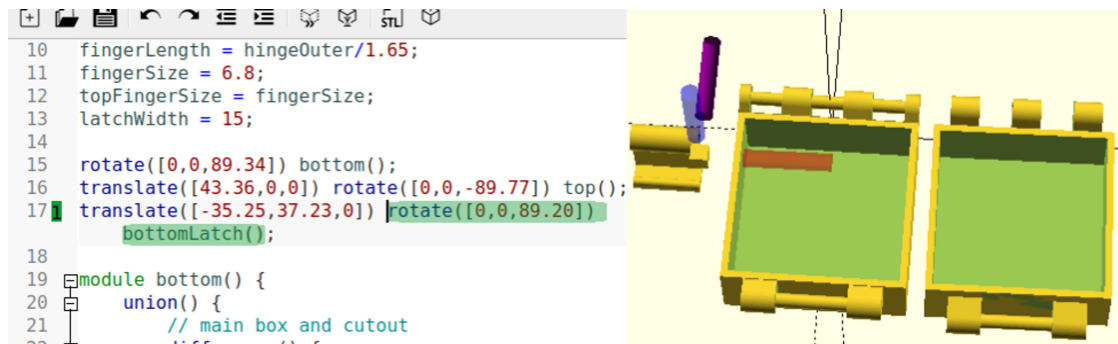


Figure 28: The code updates accordingly to the spatial transformation performed on the view

4.4 DISCUSSION

We discuss the results of our work giving an answer to the research question **RQ2**: How can bidirectional programming be used to enhance navigation and editing in programming-based CAD applications?. Specifically, we discuss the findings of our

formative study, the potential of bidirectional programming to solve problems that programming-based CAD users face, and the challenges remaining that we will address in future work.

THE IMPORTANCE OF PROGRAMMATIC INTERFACES. Previous works [78, 138, 229] highlight the importance of programming in design due to its flexibility, precision, and potential in complex tasks. In addition to confirming these technical aspects with our work, we also found that programming-based CAD facilitates access to 3D design for a more programmatic-oriented population that cannot or does not want to use direct manipulation programs. The interviews show that this public includes not only programmers but also engineers of different fields with math-oriented academic backgrounds. These results suggest that there is an opportunity for HCI research to understand better in-depth and assist this group of designers in the whole process of personal fabrication.

SUPPORTING THE PROCESS OF EXPLORING A 3D MODEL. The code helps to describe the model, whereas the view creates a visual representation that helps validate and identify errors. If users need to edit the model based on a visual inspection of the view, they must return to the code, make sense of it, locate a specific statement, and edit it coherently. Users need to follow a similar process if they want to reuse and adapt models to new projects. Making this link mentally can be challenging not only because there is no trivial transformation from the view to the code (e.g. one code statement can create multiple parts in the view when it is inside a loop) but also because current tools do not facilitate this task. Indeed, our formative study revealed that users do not know an easy way to do it. Participants showed different strategies to achieve it that required a significant effort to understand the code followed by a visual confirmation (e.g. removing statements or using modifiers). None of them could quickly locate a code statement based on the view without studying the code.

Our solution comprehensively addresses this limitation in programming-based CAD applications for the first time. It allows users to visually and quickly understand the structure of the code-based models, solving specific problems found in the formative study: difficulty in finding a code statement that creates a part in the view, isolating the contribution of a code statement in the view, understanding the code structure when a code statement creates multiple parts (e.g. a module called multiple times inside a loop), and lacking visual representation of removed objects in intersect and difference operations.

With the *reverse* and *forward search*, users can navigate between the code and the view back and forth. They can identify a part in the view by clicking on it and locating the corresponding code statements involved in the text editor without studying the code while the system coherently highlights the part in the view and the corresponding code statements. This is also helpful when exploring models from other authors, as

expressed in the interviews by P6 in Chapter 3: “In other people’s code, even trying to figure out what I need to isolate to see where it fits can take a lot of time”. The system creates a visual representation of removed objects so that navigation features can be used on them. These features improve the system of placing debug modifiers on the code, a solution that programming-based 3D CAD modelers repetitively use when designing.

Our navigation system is novel compared to existing alternatives. IceSL [127] highlights code when hovering the pointer over the model in the view, but it does not acknowledge that the contribution of multiple statements creates the selected part and the user cannot navigate or differentiate them in the view or the code. Sketch-N-Sketch [78] presents a type of *reverse search*, mainly based on the recognition of the influence of variables on the selected object rather than the instantiating statements. Moreover, 2D SVG data structure differs from CSG, which builds objects by operating on them, creating a tree data structure that can benefit more from our navigation system. Finally, none of the previous work presents a *forward search* feature.

We have identified non-solved challenges in our modified version of OpenSCAD. We noticed that different nodes of the code statement produce the same visual feedback when selecting elements in the view with our navigation features. For instance, when there is a sphere inside a translation inside a rotation statement, selecting any of these three nodes will color the sphere identically. The users do not have a way to see on the 3D view the difference between code statements in these scenarios. In our future work, we will investigate visual feedback for spatial transformation statements.

SPATIAL TRANSFORMATIONS. Spatial understanding and transformation are difficult tasks for 3D CAD modelers in general [138]. Our study showed that it can be even more challenging for programming-based CAD users because they have to transform a visual location and orientation into written operations with no visual representation of the relative coordinate system of the parts on the view. Moreover, they do not perceive immediate and incremental feedback when modifying. Thus, performing spatial transformation is a task that extensively requires a trial-and-error strategy. Our approach solves this problem by placing interactive widgets on the axis of the coordinate system of the model parts. Users can understand the relative position and orientation of the parts and edit them directly in the view.

However, spatial transformations in bidirectional programming have limitations. Users typically use variables and arithmetic expressions in transformations to create constraints between parts of their models. Therefore, there are several ways to modify an existing transformation: either changing the value of one of its variables or even changing the arithmetic expression. Sketch-N-Sketch [78] addresses this problem for SVG models by deciding on the best solution based on heuristics. However, our interviews show that programming-based CAD users would like to control the model they design and make precise and deterministic modifications. Non-code-based parametric programs, such as FreeCAD, use a *constraint solver* to compute solutions that fulfill all

the constraints and choose one of them when there are several possibilities. Therefore, instead of making decisions on behalf of the user, we would like to give them control over which variables they would like to change or not.

4.5 CONCLUSION

We presented an adaptation of the concept of *Bidirectional Programming* to programming-based [CSG CAD](#) applications. Users can browse and edit 3D models from both their programmatic description and the 3D view with direct manipulation. Based on our previous study and exploratory analysis, we found that code analysis in 3D modeling requires either a fine knowledge of the code and the model, or trial-and-error procedures to navigate the code and perform edits. Moreover, we noticed that participants struggled to perform spatial transformations, in particular when they are combined. We explained to the participants the concept of bidirectional programming and they expressed interest in it and mentioned situations in which they would find it useful. Then, we proposed design goals based on this study and described the features of a proof-of-concept implementation based on OpenSCAD that implements them. We describe an informal validation with a detailed walkthrough that illustrates how the new features help with the current difficulties we observed among the participants of our initial study. Finally, we discuss the strengths and limitations of our work, as well as future work.

The next Chapter addresses another key design challenge identified in Chapter [3](#) related to the difficulties of defining geometric properties for creating parametric designs in programming-based [CAD](#) applications.

CAD parametric design is a modeling approach that includes parameters in the design process to define and manipulate the shape and geometry of objects within the design. Parametric designs allow for rapid alteration of existing models by simply editing the values of specific parameters, allowing reusability and flexibility of the design [36]. This customization is particularly valuable in the democratization of manufacturing in practices such as personal fabrication, allowing consumers to create personalized and customized artifacts for their individual needs and contexts [204]. Moreover, designers can share their models so that other users can create new versions of a model without the need to re-design the model. In web applications such as Customizer [212] from Thingiverse, MakeWithTech [187], or 3dCustomizer [1], users can upload OpenSCAD parametric models exposing parameters so that other users can create different versions of the base models. For example, model ID 6402905¹ allows users to re-generate different versions of a "Pot lid holder" as depicted in Figure 29

To create parametric designs, the author needs to carefully define and relate the different parts of the models using variables instead of using fixed numeric values. Selected variables, designated as parameters, are made accessible for user modification, allowing users to modify their value to re-generate the model and preserve the design's structure. However, the process of creating parametric designs in programming-based CAD applications can be challenging, as revealed in Chapter 3. Programming-based CAD applications barely assist this task, significantly differing from the dynamic in direct manipulation CAD applications.

In direct manipulation CAD applications, users create models defining constraints [44] attached to variables that can be later modified. A constraint is a rule or condition applied to geometric elements within a model to control dimensions, positions, or relationships between different components. For example, the user can create a box, applying a constraint to define the box's width with a variable `box_width`. The variable can be exposed as a parameter so the user can edit its value at any time. Then, the application would re-create the model, replacing the variable's new value in the model and propagating its effects. CAD applications provide a set of possible constraints that the user applies to the model to describe it.

On the other hand, in programming-based CAD applications, users are tasked with defining geometric properties using variables normally deriving arithmetic expressions that encapsulate the desired behavior. We will refer to these expressions as *parametric definitions*. For example, to ensure a cube *B* is placed on top of another cube

¹ <https://www.thingiverse.com/thing:6402905> accessed on 01/09/2024

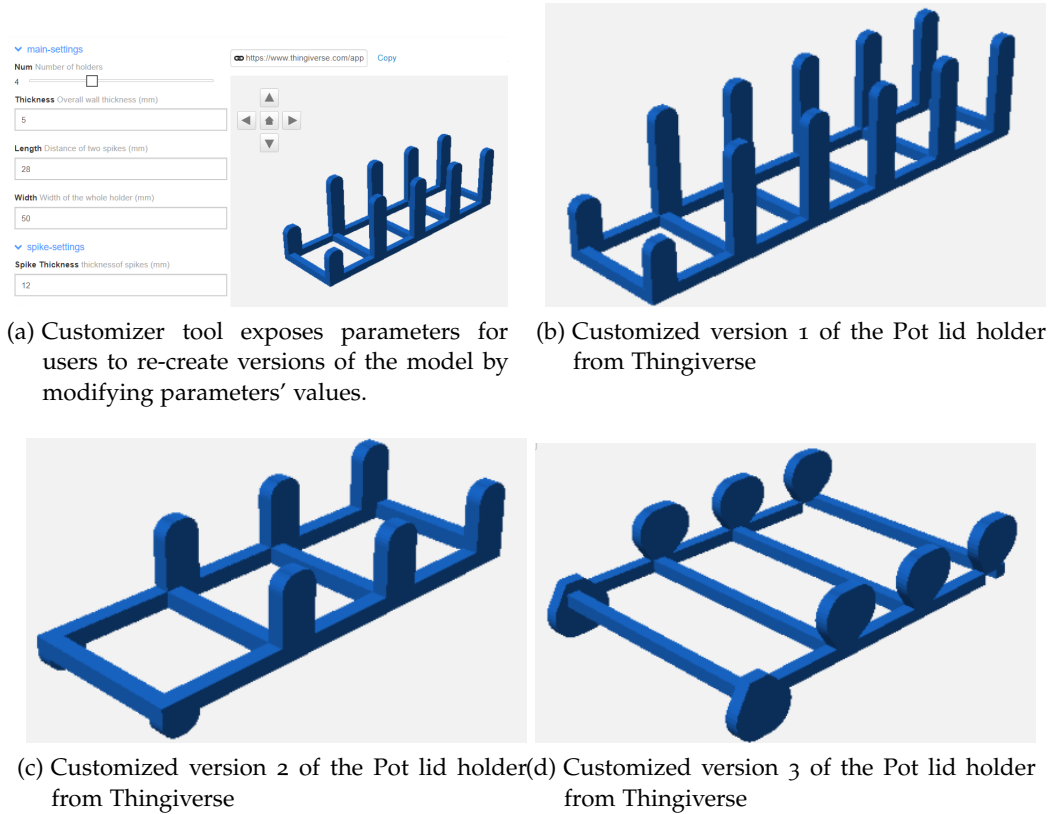


Figure 29: Model of a Pot lid holder from the thing number 6402905 on Thingiverse. By modifying the parameters the Customizer application re-generates customized versions of the model

A, the position on the z-axis of B could be defined in terms of a variable that also defines the height of A. However, accurately formulating mathematical expressions for geometric properties in programming-based CAD applications poses a significant challenge for users. It demands both mathematical proficiency to articulate geometric properties with variables and spatial reasoning skills to establish a connection between the variables in the code editor and their impact on the model in the visual representation. Furthermore, some expressions can be intricate to derive, especially when multiple parts and parameters are involved. Remembering the Pot lid holder from Thingiverse (Figure 29), the position of the highlighted element in red in Figure 30 is encoded as in Listing 3.

```

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

```

Listing 3: Example of parametric definition of a translate in OpenSCAD

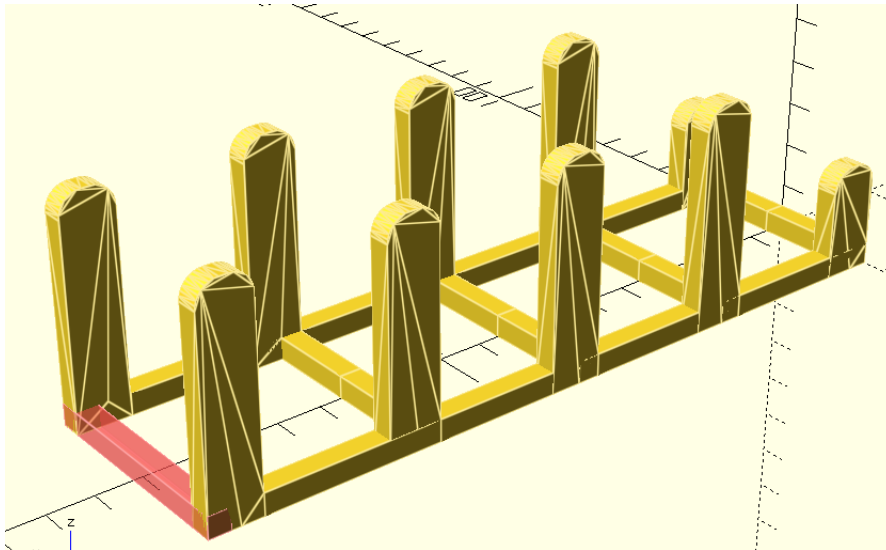


Figure 30: Model of a Pot lid holder from the thing number 6402905 on Thingiverse.

Overlooking the meaning of the variables and the lack of programming context, it is noteworthy that crafting such expressions can be somewhat challenging, even for an arguably simple model like this one, which consists of 68 lines of code and 8 parameters.

Interestingly, programming-based CAD applications possess all the information and means to facilitate the formulation of these expressions. Consider a scenario where a user designs the object depicted in Figure 30. The highlighted part of the object is missing, and the user is about to create and position it. The location of this highlighted

part is related to the position of the two vertical *rounded spikes*—as labeled in the original code—located on the left side of the object. Consequently, the user must deduce the position of the spikes, considering all spatial transformations, and mentally correlate this with the center of the highlighted part to formulate the mathematical expression outlined in Listing 3. The expression must ensure that any change in the parameters controlling the positions of the spikes also correctly adjusts the placement of the highlighted part.

It is important to note that the geometric definitions of all parts, including sizes and positions, are already defined in the code. Therefore, the application possesses all the necessary data to calculate the positions of the parts based on their parametric definition. The application could allow users to specify a part of interest and the application to calculate and provide the parametric definition of the part's geometric properties. Given that pinpointing parts of interest is more straightforward in the visual representation of the model than in the textual code, the application could facilitate users in selecting these components directly within the view. For instance, in the previous example of the model depicted in Figure 30, the user could select one of the spikes directly on the view while the application provides the parametric definition of the spike position. The user can adjust this position, including the information about the highlighted part dimensions, to create a `translate` statement and place the part correctly. Thus, the user would not need to study the code and derive the expression, speeding up the process.

We aim to enhance the parametric design capabilities within programming-based CAD applications by introducing bidirectional programming interactions. This enables users to leverage information from the view directly in the code. We started by analyzing thirty OpenSCAD models from Thingiverse to understand how typical geometric definitions for elements are constructed. Subsequently, we modified the source code of OpenSCAD, incorporating features that empower users to extract parametric definitions of geometric properties from objects in the view following the concept of bidirectional programming. This simplifies the creation of parametric models within the code. In the final phase, we tested the effectiveness of the implemented solution with ten OpenSCAD users to assess its potential. The findings reveal that geometric properties are expressed mainly as linear combinations of variables. Furthermore, users highly appreciate the ability to extract information from the view, mirroring certain constraint interactions found in direct manipulation CAD applications to reduce design errors, enhance the interactivity and appeal of the design process, and facilitate entry for newcomers by reducing the mathematical skill requirements in programming-based CAD.

5.1 SPECIFIC RELATED WORK

Parametric design needs users' foresight about which aspects of a model might require future modifications. Consider, for example, the design of a lamp structure. A user may want to be able to create multiple lamps of varying heights. This requires a careful correlation between this requirement and the geometric properties of the parts of the model, achieved through the use of variables. In the lamp example, the overall height could be a function of the heights of its constituent parts: the base, the stem, and the lampshade. The user might opt to define fixed proportions for the heights of these components, such as 20% for the base, 60% for the stem, and 20% for the shade needing only one exposed parameter. Alternatively, the user might choose to make each part's height adjustable independently, making the lamp height the total sum of all parts, thereby requiring the exposure of multiple parameters. In either scenario, effectively linking these parameters to the respective sizes and positions of the parts is crucial for successfully implementing a parametric model.

Direct manipulation and programming-based CAD applications present a different workflow and tools to achieve parametric models.

5.1.1 Parametric design in direct manipulation

When designing in CAD applications with direct manipulation, users start by creating general sketches defined to fix the geometrical properties and dimensions later [82]. In parametric CAD applications implementing a direct manipulation approach, users fix geometric properties through *constraints* [44]. A constraint is a rule or condition applied to geometric elements within a model to control dimensions, positions, or relationships between different components. There are two types of constraints: Geometric Constraints and Dimensional Constraints [130]. Geometric constraints define the relationship between two or more elements in the scene. For example, users can apply a constraint to ensure that two lines always keep the same length. Dimensional constraints fix the values of geometric properties of elements such as positions, sizes, or angles. Table 6 describes some of the most common constraints in CAD applications such as FreeCAD², Fusion360³, or AutoCAD⁴.

Constraints are crucial in maintaining the design's intent and ensuring that the model structure holds as intended when parameters change. They help define and enforce specific geometric relationships between different design parts. When designing parametrically, these constraints can be defined using variables that users can

² <https://wiki.freecad.org/Constraint> accessed on 20/01/2024

³ <https://help.autodesk.com/view/fusion360/ENU/?guid=SKT-CONSTRAINTS> accessed on 20/01/2024

⁴ <https://help.autodesk.com/view/ACDLT/2024/ENU/?guid=GUID-899E008D-B422-4DF2-AC8D-1A4F5701ED4E> accessed on 20/01/2024

Table 6: Common Geometric and Dimensional Constraints in CAD

Geometric Constraints	Dimensional Constraints
Coincident: Forces two points or objects to share the same location.	Distance: Specifies the distance between two points or objects.
Collinear: Requires two elements to lie on the same straight line.	Angle: Defines the angle between two lines or edges.
Concentric: Enforces a common center point for two circles or arcs.	Radius/diameter: Sets the radius/diameter of a circle or arc.
Parallel: Aligns two lines or edges to be parallel.	Length: Determines the length of a line or the size of an object.
Perpendicular: Forces two lines or edges to meet at a right angle.	Width/height: Specifies the width or height of an object.
Tangent: Ensures that a curve or circle is tangent to another curve or circle at a specified point.	Depth: Sets the depth or thickness of an object.
Symmetry: Requires elements to be symmetric with respect to a specified axis or plane.	Chamfer: Creates a beveled edge or corner.
Perpendicular Bisector: Defines a line that is perpendicular to and passes through the midpoint of another line.	
Midpoint: Forces two points to share the same midpoint.	

redefine to create different design versions [102]. For instance, a box design can have a constraint for its width, where a variable defines the value. This variable can be exposed as a parameter of the model, so if users change its value, the application will re-generate a new box with the new width specified by the user. Figure 14b in Chapter 2 depicts a typical case in FreeCAD of a parametric design. Geometric properties are linked to the value of cells in a spreadsheet. Users can modify cells' values to re-generate a new version of the model.

When users define constraints in a CAD application, the application internally expresses them as algebraic equations, with variables representing the coordinates of the geometric entities involved [130]. Consequently, the application formulates an equation system, which it solves using a *geometric constraint system* [234]. The outcome possibilities of this system depend on the consistency of the defined equations. In cases where all geometric properties are uniquely and coherently defined with con-

straints, the system is termed "*fully constrained*". This implies a well-defined equation system with a unique solution. In contrast, when not all geometric properties are constrained, but those constrained are defined coherently, the system is considered consistent and under-constrained. In such instances, multiple solutions exist, and the application must decide which solution to apply. For example, a user creates two lines pointing in different directions and imposes a constraint to make them parallel. The application then faces several options to resolve this constraint. It can maintain one line in its original position while adjusting the other or vice versa. Alternatively, the application may relocate both lines to achieve parallelism. Geometric properties can also be defined with conflicting constraints, making the system inconsistent. An illustrative case is applying a parallel constraint followed by an orthogonal constraint to two lines [234]. Even for experts, creating constraints can be challenging. Solutions like CODA [218] 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.

It is important to note 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 applications. As seen in Table 6, 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.

5.1.2 Parametric design in programming-based

When designing in programming-based CAD applications, users need to define all the geometric properties of the elements of the model. The only exception would be when the application provides default values to information not provided in the code by the user. For example, when a cube is created in OpenSCAD without defining the parameter size, the application will create a cube of 1x1x1. Under the definition of the constrained system, we could arguably say that programming-based models are always consistent and fully constrained. However, the definition of constraints would not be the same as in direct manipulation applications. Dimensional constraints (Table 6) equivalents would define sizes, positions, and orientations in the code. For instance, the "distance" constraint could be expressed as applying a `translate` transformation. On the other hand, geometric constraints such as keeping two lines parallel must be described by the user with arithmetic expressions.

Users articulate geometric properties in programming-based CAD applications through programming and mathematical expressions. To facilitate the parametric design process, it is crucial to understand how users in programming-based CAD environments define geometric properties in parametric designs. As noted, in programming-based

CAD applications, users specify *HOW* elements are sized and positioned using mathematical expressions. However, beyond code comments, there is often no clear indication of the users' intentions behind these definitions. Consider, for instance, the example presented in Listing 3 from the introduction of this chapter. The rationale behind the author's specific choices for the definition of location and size is not immediately apparent. Furthermore, there is a lack of research investigating design patterns in defining geometric properties in programming-based CAD applications. Chyteas *et al.* [41] conducted a study on 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. The study detailed in Chapter 3 unveils that users often encounter difficulties in formulating mathematical expressions for parametric models, a process scarcely supported by existing tools. Furthermore, participants 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.

5.2 METHOD

This study aims to facilitate the creation of parametric models in programming-based CAD applications. Firstly, we conducted a formative study analyzing thirty OpenSCAD models from Thingiverse to identify how the geometric properties are defined. Then, based on the findings, we proposed a design goal for a bidirectional application that facilitates the definition of geometric properties in parametric models. To achieve this goal, we have modified the source code of OpenSCAD to allow users to retrieve parametric definitions of objects directly from the view to be re-used in the code. Finally, we have tested our modified version with OpenSCAD users and discussed their user experience.

5.2.1 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 the findings of the project described in Chapter 3, we hypothesize that most of the time, users define positions and sizes of elements as a linear combination of the positions and sizes of other elements. For example, a very common operation is to place 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 4:

```

1 // Size of the cubes
2 size_cube_1 = 10;
3 size_cube_2 = 20;
4 // First cube
5 cube(size = size_cube_1, center = true);
6 // Second cube
7 translate([0,0,size_cube_1/2 + size_cube_2/2])
8     cube(size = size_cube_2, center = true);

```

Listing 4: OpenSCAD example. Parametric model of a cube on top of another cube

When the model becomes more complex, the definition of geometric properties considers the position and orientation of multiple parts. As a result, these definitions can be defined as linear combinations of the defined variables of the type `translate([tx, ty, tz])` where $t_i = \sum \alpha_i \cdot x_i$, α_i a constant, and x_i a variable.

The focus is on identifying the nature of parameter definitions, categorizing them as either non-zero raw numerical values (C1), a single variable (C2), a linear combination of variables (C3), a non-linear combination (C4), or a structure involving more complex programming constructs (C5), as delineated in Table 7. 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.

To assess this hypothesis, we have analyzed thirty models from Thingiverse. Accessing the website on 02/11/2023, we filtered customizable models with the option “Popular Last 7 Days” and downloaded the first ten models. We have repeated the same process with the filters “Popular Last 30 Days” and “Popular This Year”. Duplicated models were discarded and replaced with the next in the list. Figure 41 in the appendix B contains all the models used in the formative study.

We modified OpenSCAD to analyze the definition of geometric properties. Upon loading a model, the modified application parses its code into an AST. It then traverses the AST to identify code statements responsible for generating geometric primitives (e.g. spheres, cubes, or cylinders) and executing spatial transformations (i.e. translations, rotation, scale). The application would identify the nature of parameter definitions, categorizing them as either non-zero raw numerical values (C1), a single variable (C2), a linear combination of variables (C3), a non-linear combination (C4), or a structure involving more complex programming constructs (C5), as delineated in Table 7. 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.

The analysis results are depicted in Table 8. 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%. Moreover, most expressions in the rotate statements (48.95%)

Table 7: Categories used to classify expressions of OpenSCAD models.

ID	Classification	Description
C1	Raw number	A double ignoring zero values. e.g., 4.0
C2	One variable	A variable call. e.g., var1
C3	Linear combination	Linear combination of variables $\sum \alpha_i \cdot x_i$. 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

are raw numbers. 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, one variable call, or a linear combination of existing variables.

Table 8: 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%)

5.2.2 Design goals

Users define geometric properties by leveraging the relationships between objects, as outlined in Chapter 3 and corroborated by the formative study. Concerning sizes and positions, many instances define these properties as linear combinations of variables, reflecting linear relationships between elements. The positioning of a new model element often depends on another object's location and dimensions. Moreover, identifying a desired position is straightforward in the visual representation. Therefore,

a valuable tool would enable the extraction of model information for defining other elements' geometric properties within the code. This information must be readily accessible, allowing users to understand its spatial implications directly in the visual representation, where identification is simplest. In essence, the design goal is to *facilitate the extraction of geometric information from objects' parametric definitions in the view for code re-use*.

5.3 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 in the OpenSCAD application to fulfill the design goal.

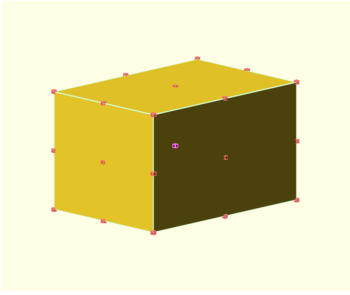
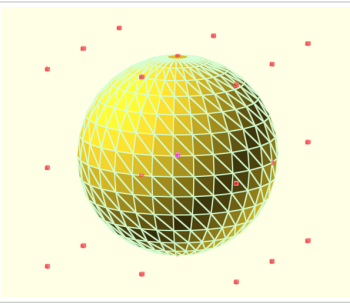
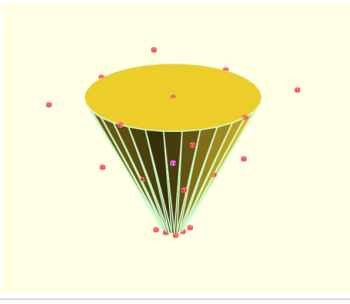
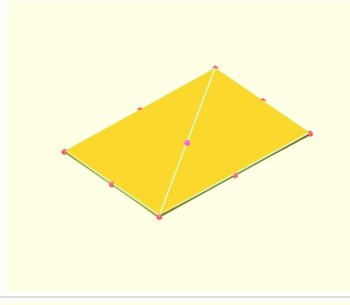
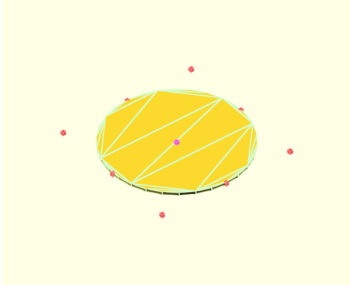
Typically, OpenSCAD parses code into an [AST](#) and later into the Abstract [CSG](#) Tree, evaluating all programming structures and variables, retaining only numeric information after evaluation. No information about the parametric definition of objects is stored at this stage. We modified the source code of OpenSCAD to ensure that [CSG](#) tree nodes store the parametric definition of the geometric properties used to define them. Primitives store the size's definition, while 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 lacking information on vertices, faces, or edges, as explained in Section 2.1.3. We redefined node definitions in OpenSCAD to include "*control points*" that the user can use to retrieve the parametric definition of an object's position. Control points were added to each primitive as described in Table 9.

Non-primitive nodes include a single control point in the node's position, covering boolean operations, spatial transformations, or programming structures. We updated OpenSCAD source code so that, given a determined control point of a selected node, the application can define the control point's position 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 control point's position 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 control point's position. Figure 31 describes how OpenSCAD derives the parametric position of the control point placed in the middle of the bottom face of the cube created by the code in Listing 5.

This proof-of-concept currently supports cases where only translations apply as spatial transformations to objects. Control points are displayed correctly in models with other transformations, like rotations or scaling, but the system does not calculate the control points' position. The system defines the position of the control point by aggre-

Table 9: Description of Control Points for Primitive Shapes

<p>Cube (27 points): Cube nodes create a grid of $3 \times 3 \times 3$ 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).</p>	
<p>Sphere (27 points): Spheres create a boundary cube using the diameter as the size for the height, width, and depth. The node places control points on the boundary cube following the same distribution as the cube nodes.</p>	
<p>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 cuboid's height is determined by h. Control points are positioned on the cuboid's boundary, following the cube nodes' distribution pattern.</p>	
<p>Square (9 points): Square nodes create a grid of 3×3 points, including 1 point in the center, 1 at each corner (4 points), and 1 in the middle of each edge (4 points).</p>	
<p>Circle (9 points): Circle nodes create a boundary square using the diameter as the size for the height and width. 1 point in the center and 1 at each extreme along each axis (4 points).</p>	

```

1 tx = 10;
2 ty = 15;
3 tz = 0;
4 size_x = 15;
5 size_y = 15;
6 size_z = 5;
7 translate([tx,ty,tz])
8   cube([size_x,size_y,size_z]);

```

Listing 5: Control points example

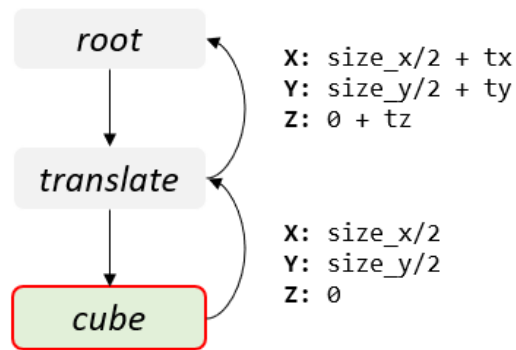


Figure 31: Representation of how OpenSCAD computes the position of the control point placed in the center of the bottom face of a cube.

gating all `translate` definitions from the root to the selected object and incorporating the local position of the control point relative to the center of the selected object. This process gathers information without filtering out trivial values, such as translating 0 units in one direction, possibly leading to less easily readable final expressions and requiring an expression simplification. To streamline the development, we implemented a Python server that exposes a service for simplifying arithmetic expressions using the “*simpy*” library⁵. The OpenSCAD application sends the raw expression to the server, returning a simplified expression.

Additionally, OpenSCAD considers parameters within primitives that affect the objects’ center positions, thereby influencing the calculated positions for control points. For instance, the control point located at the top center of a cylinder defined by `cylinder(h = height, r1 = rad1, r2 = rad2);` would yield a relative position to the cylinder’s center as `[0,0,height]`. However, if the cylinder is defined with `cylinder(h = height, r1 = rad1, r2 = rad2, center = true);`, the returned position adjusts to `[0,0,height/2]`, reflecting the parameter’s impact on the positioning of control points.

We developed two features to facilitate information retrieval directly from the model’s view. These features, built upon the developments outlined in Section 4.3.1, enable users to *select* objects within the model and utilize the capabilities of *Absolute location* and *Relative location*

⁵ <https://simpy.readthedocs.io/en/latest/index.html> Accessed on 01/09/2024

5.3.1 Absolute location

The absolute location feature enables users to determine the position of a control point in a selected object relative to the origin (*i.e.*, CSG root's position $[0, 0, 0]$) of the view. Users activate this feature by selecting the "Absolute location" button in the menu bar and then choosing an object, following the process detailed in section 4.3.1. When a user selects an object, the application displays the control points. The application marks the object's center with always-visible purple control points. The rest of the control points behave like any other geometry and can be hidden behind other geometries. Based on previous work [138], this feature aims to provide information to the user without automatically editing the code, ensuring user control over the model definition. Users can right-click on any control point to turn it green, indicating that the system has copied the parametric position to the clipboard so the user can place it in the code to define new element translations.

Consider the case where a user is designing a board game piece. The user has placed a cylindrical base and a cylindrical stem on top, as depicted in Listing 6. Using the absolute location feature, the user could then place a cylinder for the cup on top of the stem (Figure 32).

```

6  r_top = 18;
7  h_top = 33;
8  // Cup
9
10
11 // Stem
12 translate([o,o,thickness]){
13     cylinder(r1=r_stem1,r2=r_stem2,h=
14             h_stem);
15 }
16 // Base
17 cylinder(r=r_base,h=thickness);

```

Listing 6: Example before using absolute location feature

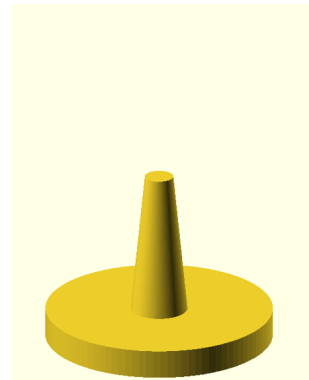


Figure 32: Example preview of a cup in progress.

The user could first select the stem's cylinder by right-clicking on it. The user could select the cylinder after the menu with the CSG nodes involved in that part, as depicted in Figure 33a. OpenSCAD would display the control points, and the user could select the one in the middle of the top where the cup's cylinder will be placed. The control point would turn green so the user would have in the clipboard the definition of that point's position in terms of the variables used in the model (Figure 33b). Then, the

user can create a translation, paste the definition stored in the clipboard as shown in Listing 7, and add a cylinder to create the cup (Figure 33c).

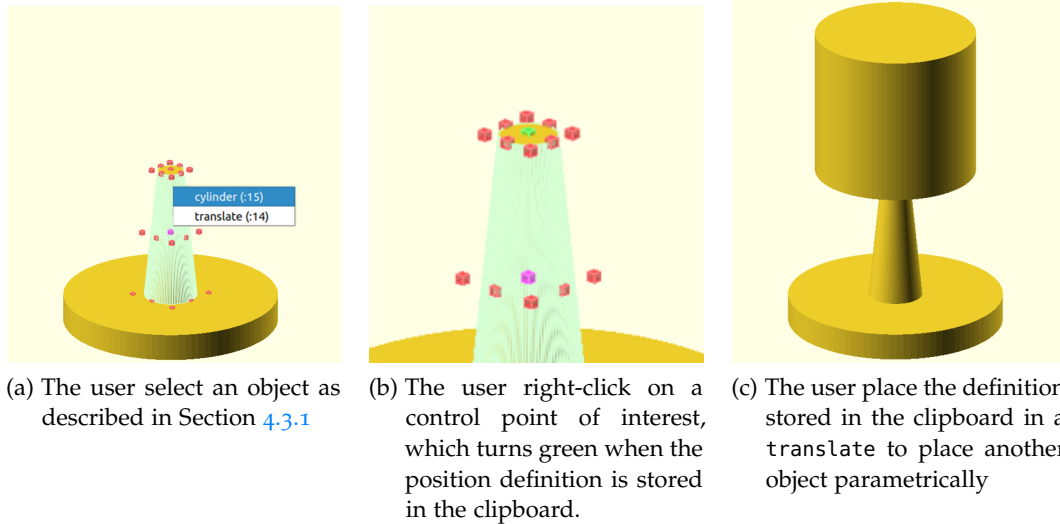


Figure 33: Absolute location feature allows users to retrieve the location of an object's control point relative to the origin.

```

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

```

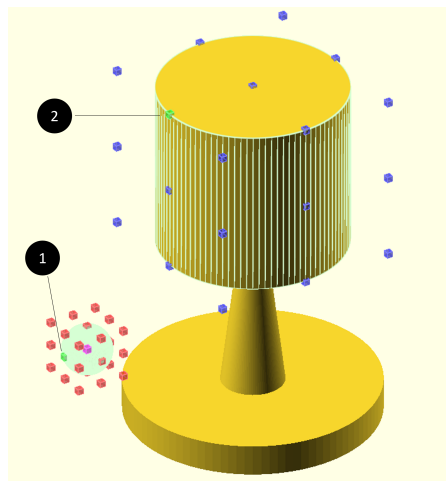
Listing 7: Example after using absolute location feature

5.3.2 Relative location

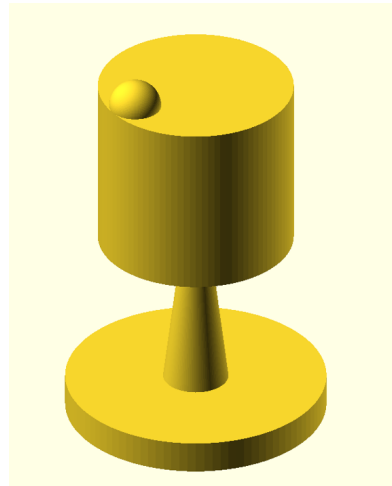
The relative location feature calculates the arithmetic expression required to align one object's control point with another object's control point, effectively establishing a "co-incidence" constraint or a "snapping" effect. The process is similar to the absolute location feature but in a two-step process. Users enable this feature by pressing the "Relative location" button in the menu bar and then selecting the object they want to move, as described in section 4.3.1. The application marks the object's center with always-visible purple control points. The rest of the control points behave like any other geometry and can be hidden behind other geometries. The user then selects a destination object, and the application also shows its control points, 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 a red control point, which turns it white, the user selects the destination point by right-clicking it. This action turns both control points green, signaling that the system has placed the parametric definition in the clipboard. The relative location feature essentially calculates the difference between the destination and origin control points, allowing users to determine the necessary transformation to align the origin point with the destination point.

Revisiting the example of a board game piece, consider the addition of decorative spheres around its upper part. Using the relative location feature enables precise placement. The user selects the sphere and then the cylinder at the top of the piece, prompting control points to appear on both. By right-clicking on corresponding control points intended to align, as shown in Figure 34a, the application generates the exact transformation $[r_{\text{top}} - r_{\text{sphere}}, 0, \text{thickness} + h_{\text{stem}} + h_{\text{top}}]$ and stores it in the clipboard. Inserting this into a `translate` statement accurately positions the sphere, as seen in Figure 34b. This process can be repeated or used in a loop to place additional decorations symmetrically.



(a) After selecting the object to move and the destination object, the user right-clicks the control point to move (1) and then the destination control point (2). The system stores the necessary `translate` definition in the clipboard.



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

Figure 34: Relative location allows users to place one object's control point relative to another object's control point.

5.4 USER STUDY

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

The experiment consisted of three parts. Firstly, we collected demographic information from participants and asked about their experience with other CAD applications. Participants self-rated their skill levels on a scale from one (novice) to five (expert) for each application. We also collected information about their experience with general-purpose programming languages and asked them to rate their OpenSCAD skills on the same scale. Additionally, we discussed their understanding of parametric design and interest in creating parametric models. In the second part, participants performed a task to create a parametric design using the original OpenSCAD version. They verbalized their thought process throughout the task. 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. 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.

5.4.1 Recruitment and Participants

The recruitment process mirrored that of the study described in Chapter 3. We contacted participants from the study described in Chapter 3 and recruited in OpenSCAD communities on Reddit (r/openscad) and Facebook (OpenSCADAcademy) to conduct the experiment using Zoom video conferencing. The only criterion for participation was the ability to create parametric designs with OpenSCAD.

Participants used a configured version of OpenSCAD on a computer we set up. Before the session, we asked them to install the remote desktop application Anydesk⁶. During the experiments, they accessed a Linux machine we prepared with Anydesk to execute their parametric design tasks.

We report participants' demographics and previous experience with CAD applications in Table 10. All participants self-identified as male and varied in age: one was between 20 and 29, three were between 30 and 39, three were between 40 and 49, one was between 50 and 59, and two were between 60 and 69 (average: 44.6, standard deviation: 15.0). All participants, except P3, had four or more years of 3D modeling

⁶ <https://anydesk.com/>

experience (average: 9.3y, standard deviation: 5.9). 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 application, 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: One participant with 2, four participants with 3, four participants with 4, and one participant with 5.

Table 10: 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 *Others* is the highest rank expressed by the participant among the options.

Others*: Onshape, MeshMixer, Inventor, CATIA

Others**: Matlab, PHP, Bash, IBM Assembly, ARM Assembly

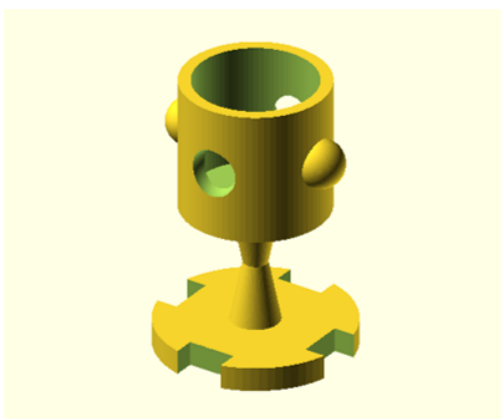
				Other CAD applications							Programming languages							
Participant	Age Range	3D modeling experience (y)	OpenSCAD	FreeCAD	Rhino	TinkerCAD	Fusion360	SketchUP	AutoCAD	Solidworks	Others*	Python	C	C++	C#	Javascript	Java	Others**
P1	40-49	8	3	1								5						5
P2	60-69	9	5		2						2	4		5				5
P3	60-69	2	3	2			3					1						
P4	50-59	10	3									5	5			5		
P5	30-39	18	4				1				1	3				4	2	2
P6	30-39	20	2	3	4					1	2	2		3				2
P7	40-49	12	4									5	5		5		5	
P8	30-39	5	4	1			4					4		4		4		
P9	20-29	4	3	4			3			2	3	3		3			2	4
P10	40-49	5	4			4		2	1							5		3

5.4.2 Design tasks

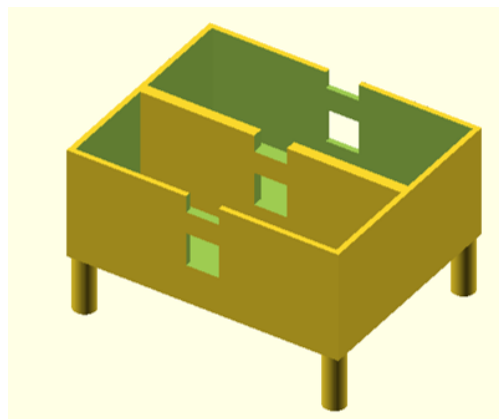
Participants performed two parametric design tasks first using the original version of OpenSCAD and later our modified versions of OpenSCAD.

We proposed two models: the model A, a chalise-like model (Figure 35a), and model B, a box (Figure 35b), 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 make a parametric design exposing parameters at least 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 create a parametric design that exposed parameters for the length of the legs, the size of the windows, and the height, width, and depth of the box. We converted the models into STL files and uploaded them to the STL online viewer, 3DViewer⁷, generating shareable links that allowed participants to view the models in 3D on their computers. As mentioned in the first model participants used the original version of OpenSCAD and in the second model they used the modified version of OpenSCAD. To mitigate order bias, we counterbalanced the sequence: half of the participants worked on Model A first, followed by Model B, while the remaining participants started with Model B, and then proceeded to Model A.



(a) Model A. Participants were required to replicate it parametrically, exposing parameters for the size of 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.



(b) Model B. Participants were required to replicate it parametrically, exposing parameters for the length of the legs, the size of windows, and the dimensions (*i.e.* height, width, and depth) of the box.

Figure 35: Models used in the experiment. Participants replicated the models, exposing parameters as required

In the first task, participants used the original OpenSCAD version, articulating their thought process. This exercise had three goals: first, to establish a baseline for comparing the performance of the design process and user experience with the later exercise using the modified OpenSCAD version; second, to refresh participants' understanding of parametric design in OpenSCAD, facilitating a subsequent discussion about its challenges; and third, to allow observation and analysis of workflows, practices, and difficulties in creating parametric designs. After completing the first design, we asked the participants about their user experience, task difficulty, and specific challenges in executing parametric designs with OpenSCAD. We then introduced the new Open-

⁷ <https://3dviewer.net/>

SCAD 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 resolving doubts, 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.

5.4.3 *Data collection*

During the experiment, various forms of data were collected. We recorded the OpenSCAD window activity while participants worked on both design tasks. Recordings of participants creating the first model with the original OpenSCAD version were analyzed to find common patterns and behaviors in designing. Moreover, these recordings were compared to the recording of participants creating the design with the modified version of OpenSCAD to evaluate the potential and challenges of our solution.

Moreover, 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.

5.4.4 *Data analysis*

We created a log of participants' events, creating a parametric design in the original version of OpenSCAD. Events included detailed actions such as creating parameters, first rendering, creating spatial transformations, and errors performing design tasks. We tracked all the `translate` statements declaration and all attempts to verify their correctness. We compared these logs and identified common behaviors and patterns.

Then, we analyzed the participant's answers regarding the difficulty of the tasks, functionality, and usability of the features. We summarize their answers and present them.

Finally, we created a log of participants' `translate` statements declaration and all attempts to verify their correctness. We compared these results with those from the designs on the original version of OpenSCAD.

5.4.4.1 *Parametric design in programming-based CAD*

We identified common behaviors when participants designed with the original version of OpenSCAD.

PARAMETRIC DESIGN The participants' approaches 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 forecast 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. 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.

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 into 2: the box and the legs. Eight 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 showed the pattern of creating subparts in 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 creating `translate` statements to place each part on top of the other. The rest of the participants created the designs cumulatively without removing elements.

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. 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. This cylinder was then 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 center of the object 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 achieve 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 erred on the sign, adding when they should subtract, indicative of a disconnect between spatial conceptualization and code expression.
4. Confusing variables, particularly in complex expressions involving multiple variables, lead to the inclusion of incorrect variables in calculations.
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 upon satisfaction with the placement.

Strategies to address these errors typically involved trial and error with variable factors and signs, along with meticulous code examination to ensure accurate expression formation.

ENSURING OVERLAPPING Another common strategy involved creating a variable with a minimal value to ensure necessary overlap. 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. To bypass this issue, participants introduced

a "delta" variable, slightly enlarging the element meant for subtraction. This adjustment ensures that the geometric calculations accurately reflect the desired subtraction, addressing the preview mode's imperfections.

5.4.4.2 Perception on implemented features

Participants shared their experiences on the difficulty of creating the models and the functionality and usability of the implemented features.

Most participants rated both models as relatively easy to create, as shown in Figure 36. P1, P2, and P4 indicated that they perceived Model B as slightly more challenging than Model A. However, all participants rated the difficulty of both models between *Neutral* (option 3), *Easy* (option 4), and *Very Easy* (option 5). Specifically, for Model A using the original version, two participants considered it neutral, one easy, and two very easy. In the design of Model A with the modified version of OpenSCAD, 4 participants considered easy, and one very easy. For Model B using the original version of OpenSCAD, one found it neutral, two easy, and two very easy. Participants ranked equally for Model B using the modified version of OpenSCAD.

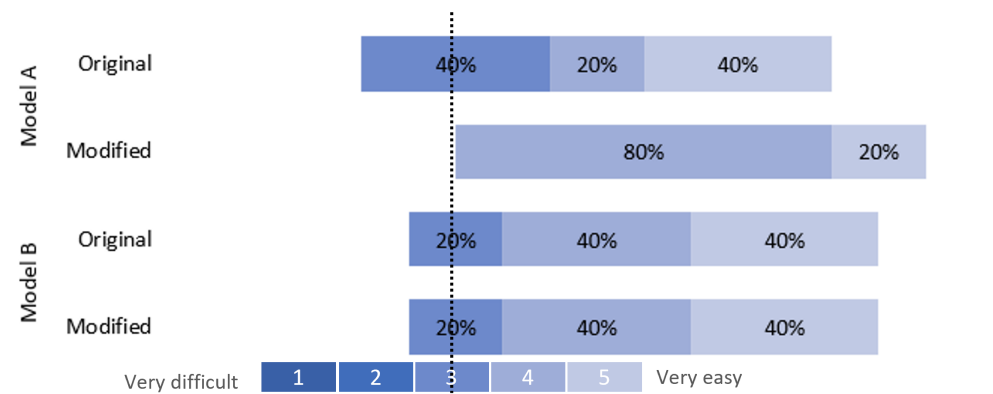


Figure 36: Participants ranked the difficulty of Models A and B in the original and modified version of OpenSCAD using the scale: 1 (Very Difficult), 2 (Difficult), 3 (Neutral), 4 (Easy), and 5 (Very Easy).

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 37). All participants answered above *About the same* (option 3), with seven participants with *Somewhat Easier* (option 4) and three participants with *Much easier* (option 5). Then, we asked a similar question but individually targeted both features, absolute location and relative location. Not all participants used both features in the design exercise according to personal preference, so answers in Figure 37 report the percentage of the total of participants who used the feature and answered the question: seven participants for the absolute location feature and nine participants for

the relative location feature. For the question, “Did the absolute location make the design easier or more difficult?” one participant (14.3%) answered about the same, and six participants (85.7%) answered somewhat easier. Regarding the relative location feature, four participants (44.4%) answered somewhat easier, and five participants (55.6%) answered much easier. Models A and B were perceived as similarly difficult, although some participants mentioned that model B was slightly more difficult.

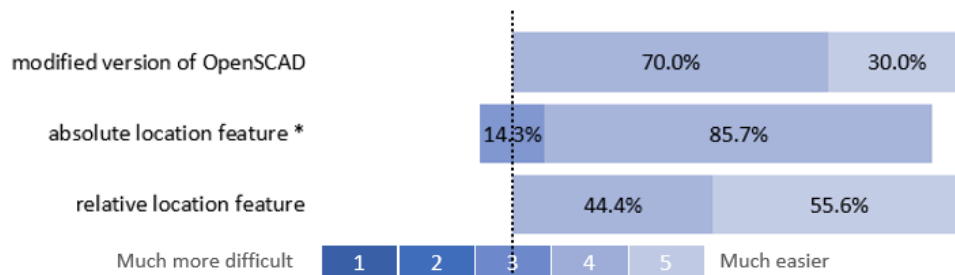


Figure 37: Participants answered if the modified version of OpenSCAD and the individual features made the design task easier or more difficult with the scale 1 – Much more difficult, 2 – Somewhat more difficult, 3 – About the same, 4 – Somewhat easier, 5 – Much easier

We also asked how difficult it was to use each of the implemented features in terms of usability, as depicted in Figure 38. Similarly to the previous question, the answers report on the total 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 absolute location feature, six participants (58.7%) answered easy and one participant (14.3%) answered very easy. Regarding the relative location 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 and P5 commented “*My initial inclination is to left click those handles; it’s a little bit extra to remember to right-click the handle*” and “*For me, selecting with the right click is a bit unintuitive*”. Furthermore, the control points did not scale when zoomed in, which was also reported as problematic. P9 commented “*It was easy except for the control points size; it would be nice to have scaling on those.*” For the relative location, 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. A potential improvement from the rendering side could be to draw an animated arrow to indicate the directionality of the relative location, providing better guidance on its use.*” Further, P3 mentioned that using color as indication of 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".

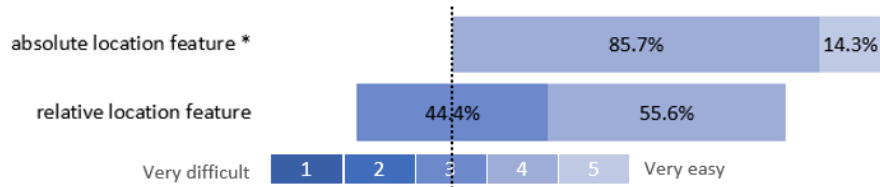


Figure 38: Participants answered how difficult was to use the implemented features with the scale 1 – Very difficult, 2 – Difficult, 3 – Neutral, 4 – Easy, 5 – Very easy

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 because the two values were similar. 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 a more interactive alternative than other alternatives that try to include "anchors" selectable from the code. "Tools like Cascade and others 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. They commented "It would definitely ease the transition into (programming-based) parametric design for those people who are used to using traditional CAD (direct manipulation)." and "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".

5.4.4.3 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 OpenSCAD. 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 outcome: Success for

correct placements, *Wrong location* for incorrect placements, *Uncertain/false positive* for when participants were unsure or incorrectly deemed the placement, and *Syntac 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 39a, 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. Similar behavior is depicted in Figure 39b when the analysis is done per model. Notably, Model B exhibited a lower success rate and an increase in wrong location attempts in both OpenSCAD versions, aligning with participant feedback that Model B was slightly more challenging than Model A.

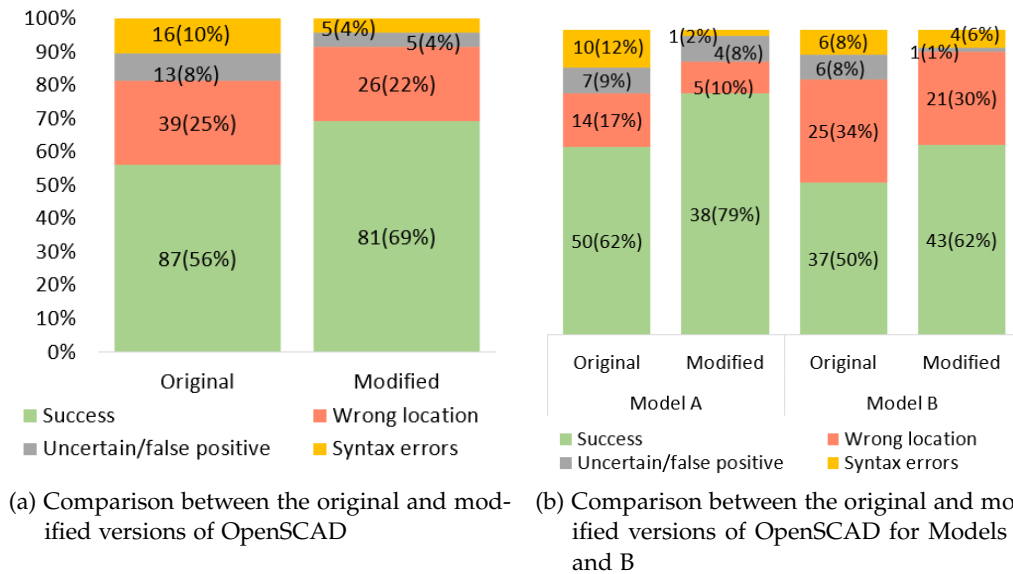


Figure 39: Each attempt to verify a translate statement upon rendering was logged and classified to compare both versions.

During the second design using the modified version of OpenSCAD, not all translations were defined using the developed features. Participants often opted to calculate expressions manually. Figure 40a 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, the Model B presented a higher number of 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. Moreover, Figure 40b shows the trials of using the implemented features. In most of cases, users could use the features although in some cases they made mistakes or required assistance with specific questions.

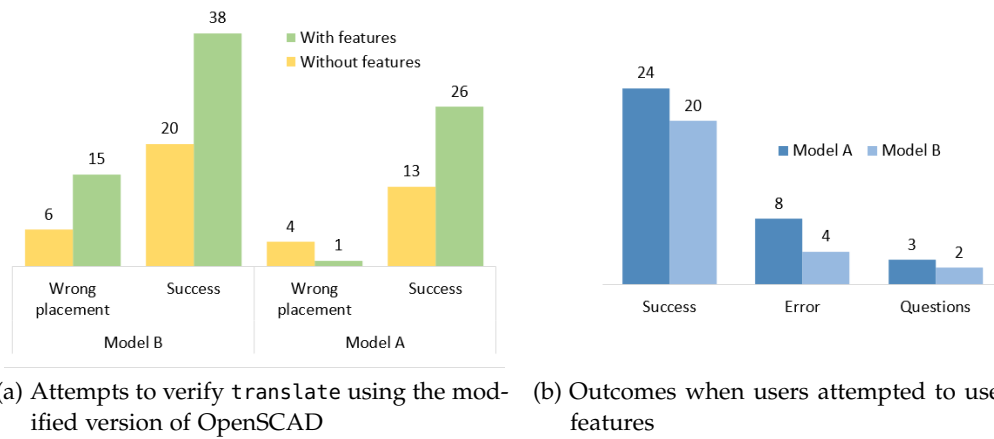


Figure 40: Results of feature usage in OpenSCAD and possible outcomes.

5.5 DISCUSSION

We are interested in facilitating the parametric designs in programming-based CAD applications, particularly addressing the challenge identified in Chapter 3: the difficulty of defining parametric geometric properties of certain objects based on the properties of others.

We have evaluated the potential and challenges of our proposed solution, answering the research question **RQ3**: How can bidirectional programming facilitate the definition of geometric properties of models in programming-based CAD applications?.

Participants unanimously agreed that our solution would significantly ease the design process. They highlighted that such alternatives could help avoid errors, simplify the mathematical definition of properties, make the design process more interactive and appealing, and lower the mathematical skill barrier for newcomers in programming-based CAD applications.

We foresee two distinct user scenarios for these alternatives: one involving newcomers and the other experts. Prior studies have indicated that newcomers often avoid design tasks due to perceived high-skill requirements [88]. This issue is exacerbated in

programming-based CAD, where the semantic gap between natural and programming languages is more pronounced [111, 132, 175]. To design effectively, users typically need to visually inspect the current state of the model to determine the next element's parametric position. Our solution minimizes this “gulf” between evaluation and execution [226] by allowing users to extract positions directly from the view, bypassing the need to interpret and manipulate code to derive parametric expressions.

This solution is equally beneficial for experts. The study described in Chapter 3 revealed that even self-identified expert OpenSCAD users face challenges with mathematical definitions. Experienced participants often exhibited mechanical workflows when defining parameters, yet hesitated over the accuracy of their parametric definitions in complex designs, resorting to trial and error. Our solution can speed up this process by automatically providing accurate positional descriptions. However, it also challenges experienced users accustomed to established programming workflows. Users seem to be accustomed to reaching solutions restricted to the features of the programming language. P3 noted that some applications, like JSCAD, offer functions to extract geometry boundary box information (*i.e.* minimum and maximum values in all dimensions) for later use in code⁸, but this still limits users to the code editor and mental reconstruction of code results. Our approach, by contrast, focuses on direct understanding and interaction in the view, easily identifying a required position in the view but having to derive it in the code. Similar challenges are faced by other programming-based CAD tools like CadQuery. Mathur *et al.* [138] propose a solution for creating queries that also incorporate interaction in the view, albeit for a different purpose, but with the same underlying principle of extracting information from the view for code reuse.

The comparison between the original version of OpenSCAD and our approach revealed that using the developed features participants would require less attempts to reach the aimed geometric properties, resulting in a proportional lower rate of errors and higher rates of successful attempts. Model B presented a higher rate of errors when using the developed features.

Despite these advantages, we observed experienced users' resistance to deviating from their established workflows. P6 expressed, “*It seems easy to use, but changing the mindset to use the view for design is challenging without practice.*” Similar views were echoed in OpenSCAD web forums⁹, where discussions about integrating such solutions are limited to programming language features or in a Application Programming Interface (API).

While there may be initial resistance among experienced users, our user study suggests that the perceived benefits can facilitate adaptation.

⁸ https://openjscad.xyz/dokuwiki/doku.php?id=en:design_guide_measurements accessed on 28/01/2024

⁹ <https://github.com/openscad/openscad/issues/954> accessed on 28/01/2024
<https://github.com/openscad/openscad/issues/301> accessed on 28/01/2024

However, our implementation does present some usability challenges, as users noted. Certain control points were inaccessible due to overlaps with other volumes, and using right-click for selection was not always intuitive. These feedback points are crucial considerations for future refinements of our solution.

Another identified problem is related to the removed geometries in the difference statements. Participants wanted to use the features to place subtracted elements but these elements were not reachable from the view. Some participants used background modifiers to make these geometries visible and selectable so they could use the features. Normally, participants placed the cursor in the statement they were modified. Thus, the application could make always visible the element that create the code statement where the cursor is placed so the user has an explicit visual representation of the part they are working on and could help in cases where geometries are subtracted. Other challenge is related to the difficulty of understanding nested transformations. Sometimes, participants wanted to use the absolute location feature to replace a `translate` statement definition. However, often, these statements were placed inside other `translate` statements. As the absolute location feature gives the position relative to the `CSG` root, the definition retrieved is not useful to use inside another transformation. The application could take into account the position of the cursor to locate where the new definition will be placed and incorporate previous transformations.

5.6 LIMITATIONS

Our study initially 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.

5.7 CONCLUSION

Based on the insights of the study presented in Chapter 3, we hypothesized a general structure for the creation of geometric properties in parametric designs within programming-based `CAD` applications. To test our hypothesis, we conducted a formative study, 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. To achieve this goal, we modified the source code of OpenSCAD, implementing features that align with our design objectives. To validate our solution, we conducted an experimental study involving ten OpenSCAD users. These participants engaged in creating parametric designs using both the original and our modified versions of OpenSCAD. They evaluated their experience and engaged in discussions about the challenges and potential of such solutions.

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

CONCLUSION

This thesis is dedicated to improving the user experience in programming-based CAD applications. We have conducted three distinct studies, each contributing to discovering previously unidentified challenges faced by users of these applications. To address some of these challenges, we have implemented bidirectional programming solutions within the programming-based CAD application, OpenSCAD. These solutions aim to alleviate user difficulties, thus improving the overall usability of programming-based CAD applications.

6.1 RESEARCH CONTRIBUTION SUMMARY

This research aimed to identify and address the challenges faced by users of programming-based CAD applications to improve their usability. Initially, we encountered a substantial gap in understanding the user experience within these applications. We have focused our investigation on 3D design in the field of personal digital fabrication with 3D printers, where programming-based CAD applications have a significant influence. In our first study, we conducted a comprehensive study that included interviews with active users of popular programming-based CAD applications like OpenSCAD. The data collected were analyzed using a Reflexive Thematic Analysis (RTA), revealing three main themes in 3D printing design practice: the profile of programming-based CAD applications users, challenges in 3D modeling and difficulties encountered in 3D printing. These insights, detailed in Chapter 3, provide an in-depth understanding of the motivations and behaviors of makers in programming-based CAD, along with the challenges observed in a hands-on exercise and reported by users answering the research question **RQ1**. Our findings extend to programming-based CAD applications beyond digital personal fabrication and to broader challenges of 3D printing within the digital personal fabrication community. Therefore, this study has laid the groundwork for future research to address specific problems and facilitate the adoption of programming-based CAD applications.

After our initial study, we addressed two specific design challenges: the difficulty in linking the 3D view with the corresponding code and the complexities involved in executing spatial transformations. An exploratory exercise was conducted to delve deeper into these challenges, which led to the development of specific design goals for a system designed to mitigate these issues. Embracing the bidirectional programming approach, we modified the source code of OpenSCAD to implement these goals. We specifically introduced two features to enhance navigation between the view and the

code: reverse search and forward search. These features, relying on direct manipulation interactions, enable users to select parts of the model directly on the view and receive visual cues that make the code-view connection explicit. This innovation simplifies the location of specific code statements based on visual inspections and isolates the impact of a code statement in the view. Furthermore, the application supports spatial transformations such as translate, rotate, and scale through drag-and-drop interactions, with automatic code updates. This approach allows users to easily comprehend the relative coordinate systems of specific parts and make view-based edits, where design intent is more intuitively established. This study focused on responding to the research question **RQ2**. The results, including informal validations of these features, are discussed in Chapter 4, evaluating the potential, limitations, and future directions of this approach.

In Chapter 5, we tackle a key challenge identified in our initial study: the precise definition of geometric properties in parametric models. Recognizing users' difficulty in formulating mathematical expressions for parametric designs, we hypothesized that geometric properties in programming-based CAD applications are commonly formulated as linear combinations of existing variables. This hypothesis was substantiated through a formative study analyzing thirty models from Thingiverse. Based on these insights, we proposed and implemented a design solution in OpenSCAD, enabling users to directly extract geometric information from the view for use in the code through bidirectional programming interactions. An experimental study with ten OpenSCAD users evaluated this solution, focusing on its influence on the creation of parametric designs and its potential to streamline the design process. Our findings reveal unanimous agreement among participants on the usefulness of this feature, indicating its potential to simplify the access of newcomers to programming-based CAD applications and aid in the accurate definition of geometric properties giving an answer to the research question **RQ3**.

In conclusion, this doctoral thesis presents a thorough approach to answer our research question:

"How can interaction techniques be used to facilitate design in programming-based CAD applications for 3D printing?"

We start by presenting an understanding of the programming-based CAD user community that uncovers numerous challenges these users encounter during the design process, establishing a foundation for future investigations. Furthermore, this research addresses three significant identified challenges through the application of bidirectional programming in two distinct studies. The proposed solutions, implemented in OpenSCAD, are designed to be compatible and operational together, as a proof-of-concept of the potential of these solutions in enhancing programming-based CAD usability and efficiency.

6.2 FUTURE WORK

This section delves into the potential facets of work that can be a continuation of our research. Initially, we will discuss two projects that were conceived and started during our research but were not completed. Although not fully realized, these projects offer valuable insight and learning experiences that have contributed to the overall research journey.

Subsequently, we will explore potential research opportunities that could further extend and build on our current investigation. These opportunities represent prospective avenues for continued exploration and development in the field, potentially opening new doors for advancements and discoveries based on the groundwork laid in this thesis.

6.2.1 *Other Explorations*

In exploring potential contributions to address the challenges faced by users of programming-based CAD applications in 3D printing, as identified in Chapter 3, our focus initially narrowed to two additional issues. Due to time constraints, our efforts were mainly directed toward the studies presented in Chapters 4 and 5. However, early discussions and initial work on some of these challenges are worth sharing.

The first challenge involves the disconnection between the visual representation of objects in the design environment and their actual physical context. The second challenge concerns the difficulties that users experience in modifying models post-printing.

6.2.1.1 *Bidirectional Programming on Augmented Reality*

In digital personal fabrication with 3D printers, objects are often intended to be integrated into a physical environment, either to create new objects [67], repair them [88], or augment existing ones [13]. Designing these objects typically requires considering constraints from the physical environment. However, most available design applications are limited to a 2D screen interface, which presents an isolated, empty canvas. Users encounter problems handling 3D spaces in 2D viewports [100] and are forced to design without considering the context of the intended physical environment. This limitation not only restricts creativity [194] but also complicates the verification of the accuracy of the model [110]. Additionally, by separating the creative and validation stages [57], the likelihood of needing multiple printing trials increases, which is time consuming [149].

Various approaches have attempted to bridge the gap between the digital and physical worlds. Techniques such as scanning physical objects or capturing 2D images for 3D object design [60, 61, 122, 166, 224] and using physical 3D props such as blocks

[9, 126] or clay [188] have been explored. While these methods allow some interaction with physical objects, the design process remains isolated from the target environment due to the 2D screen limitation. To provide a more immersive design experience, extensive research has been done on virtual [66, 97, 137, 144, 146, 178, 215, 230] and augmented reality solutions [12, 23, 51, 99, 120, 168, 208, 225]. These 3D environments enhance spatial perception, potentially mitigating depth perception issues [97] and aiding in understanding the real scale of objects [120]. Furthermore, 3D interactions can offer more intuitive workflows for 3D environments compared to 2D screens [147], which benefits general 3D modeling tasks [181]. Augmented reality, in particular, improves user experience and performance by combining design and validation stages in a pre-printing phase, reducing design time and fabrication iterations [57].

Nevertheless, similar to the broader trend in CAD applications, these solutions have predominantly focused on design assistance in direct manipulation applications. This is likely due to the differing interactive nature of programming-based and direct manipulation approaches. In essence, both virtual and augmented reality allow for direct manipulation interactions. Thus, it is arguably evident that the transition between a 2D screen and these environments included applications that followed the same interaction approach: direct manipulation.

Introducing bidirectional programming into programming-based CAD applications offers an opportunity to leverage the interactive space provided by augmented reality, effectively addressing the issue of design isolation from the intended physical environment of the object. These immersive environments could also enhance code interaction and comprehension, a concept previously explored in various fields [85, 108, 182, 183]. Furthermore, augmented environments could be enriched by the solutions proposed in Chapters 4 and 5.

Future studies could explore the potential of augmenting programming-based CAD applications with bidirectional programming interactions in an augmented reality setting based on previous work.

CAPTURING SHAPES FROM PHYSICAL OBJECTS Various projects have attempted to incorporate the physical environment into the digital realm. KidCAD [61] enables children to imprint toy shapes onto a malleable gel input device for subsequent design use. CopyCAD [60] captures 2D shapes of arbitrary objects using a camera/projector system for integration into new models. Lau *et al.* [122] extract contours from photographs for use in designing new objects. RetroFab [166] scans device interfaces to redesign adapted layouts incorporating actuators. These solutions underscore the importance of integrating geometries from the physical environment into digital design. However, they typically only incorporate a fraction of the environment into the digital realm and lack mechanisms for exporting digital models back into the physical world for validation.

MODELING THROUGH PROPS Other approaches emphasize design using physical objects instead of solely capturing geometries. Sheng *et al.* [188] investigate sculpting via physical proxies, where the system tracks the user's actions in sculpting clay and translates them into digital designs. Anderson *et al.* [9] advocate for designing with interconnected blocks to represent geometries later digitized. StrutModeling [126] introduces a kit for low-fidelity modeling using magnetic connection pieces. While these methods facilitate quick validation and interactive design, they are not designed to transform designs into coded formats and may struggle with modifications to pre-existing models.

MIXED REALITY The capacity of virtual and augmented reality to display geometries in 3D space makes them appealing for 3D design. Research in mixed reality design is extensive, covering both virtual and augmented reality. Trika *et al.* [215] integrate CAD with virtual reality to improve model understanding, particularly in complex junctions. Gao *et al.* [66] combine virtual reality with voice recognition to improve design precision. Mine *et al.* [146] address virtual reality's precision challenges by incorporating 2D touch surfaces. Lift-Off [97] and Martin [137] explore enhancing virtual reality CAD modeling. However, these virtual reality solutions still separate the physical environment from the design process.

Augmented reality appears to be more suitable for integrating the physical environment. Arisandi *et al.* [12] and Mockup Builder [51] combine augmented reality with physical props and multi-touch surfaces for 3D modeling. Lau *et al.*'s Situated Modeling [120], MixFab [225], and DesignAR [168] further this integration with augmented reality and multi-touch screens for design.

Although building on these solutions to explore different interaction techniques, current research does not facilitate programming-based design in 3D spaces.

6.2.1.2 Development

To effectively integrate OpenSCAD into an augmented reality environment and leverage the solutions implemented in Chapters 4 and 5, it was imperative that the application supports preview features similar to those offered by OpenSCAD. A significant challenge in this integration is the lack of existing solutions capable of creating and manipulating CSG geometries in runtime within an augmented reality context in applications such as Unity [211]. Addressing this gap, we initiated the development of a library specifically designed to dynamically generate and manipulate CSG structures and geometries for augmented reality applications.

This work was carried out by Danny Kieken, a member of the Loki team and co-author of one of our studies [71] in the course of this investigation. The framework developed due to this collaboration is accessible at <https://github.com/LokiResearch/LibCSG-Runtime>, offering a novel library for creating new meshes from CSG operations in Unity during the runtime.

6.2.1.3 Facilitating post-printing modifications in programming-based CAD applications

Personal digital fabrication inherently involves an iterative process. Despite thorough validations during the design stage, users frequently need to modify their designs post-fabrication [57].

A common approach to ascertain the required changes in the digital model is to first experiment with physical modifications on the printed object. While this process ensures the model meets user expectations, physical edits do not automatically translate to the digital model [223]. Subsequently, users are tasked with interpreting these physical modifications and translating them into corresponding code adjustments. This process necessitates a second round of analysis, where edits, already made and validated visually, must be re-implemented in the code.

There are some efforts to allow users to integrate modifications performed in the physical object into the digital design. Some studies have explored initiating the design process with a physical prototype [125, 179], later digitized using image capture tools like scanners. Others have proposed systems for annotating physical prototypes, where annotations are recognized and transferred to the digital model, aiding in editing [199]. Additionally, the concept of *interactive fabrication* has been proposed, where a dynamic collaboration between the user and the digital tool allows for an incremental design and fabrication process [74, 150, 163, 170, 231, 232]. Notably, Weichel *et al.* [223] explore *bidirectional fabrication*, wherein the user and machine collaborate to perform edits and update the digital model simultaneously.

However, these approaches primarily cater to direct manipulation applications and do not address scenarios where the digital design is code-based. Future work could investigate assistive systems that integrate physical edits on printed objects into digital designs created by programming-based CAD applications, thus facilitating the fabrication process.

The creation of prototypes as a preliminary step in design, serving as a pre-fabrication validation, has been explored in several research studies. Leen *et al.* [126] introduce a toolkit comprising interconnected pieces with a digital counterpart. Users can assemble and disassemble these pieces to form a low-fidelity structure, which is then immediately reconstructed on the screen. Clay modeling has also been used for the creation of prototypes, either integrated into the digital design process through finger tracking [188] or by photographing the finished model [9]. These methods enhance the design process with interactivity and provide a clearer understanding of the final object's appearance in the physical world. However, they primarily function as an initial design step and are not suited for iterative design processes. In essence, these technologies are beneficial for crafting an initial object prototype and digitizing it, but subsequent edits made to the printed design do not directly influence the digital model.

HotFlex [74] offers a post-printing customization option. It embeds a system that can alter shapes using bendable materials, allowing users to adapt the final object after

printing, based on anticipated design changes. While this method provides some flexibility for post-print modifications, it requires precise anticipation of potential changes and is limited by the material's properties.

The concept of interactive fabrication, breaking away from traditional 3D printing workflows, has been explored in various innovative solutions. FreeD [231, 232], a hand-held digital milling tool, assists users during fabrication, alerting them to potential errors. Rivers *et al.* [170] propose a system that guides users in sculpting 3D objects over different materials by projecting color indications. FormFab [150] enables users to reshape thermoplastic materials through hand gestures, tracked by a robotic heat gun. Similarly, Roma [163] combines an augmented reality headset with a robotic printer for an incremental fabrication process. These interactive methods promote creativity by allowing design alterations during the fabrication stage. Weichel *et al.* [223] further this concept with bidirectional fabrication, where both the user and machine collaborate on prototype edits, with the system updating the digital model in response to physical modifications. However, most of these solutions do not maintain the changes in the final digital design and are not readily applicable to designs based on programming-based CAD applications.

Song *et al.* [199] propose a system that enables users to annotate physical prototypes created from a digital design. The annotations made on the prototype's surface are then transferred to the digital CAD model, assisting in editing the digital design. This method opens up possibilities for integrating physical modifications into programming-based CAD applications, enhancing the workflow for 3D printed models.

This literature review provides a foundation for future research. It highlights existing developments and underscores significant research gaps, thereby guiding future studies in facilitating the inclusion of digital modifications based on physical changes of post-printed objects.

6.2.2 Research gaps

We discuss several research possibilities that our investigation has uncovered.

6.2.2.1 Understanding specific challenges

The study outlined in Chapter 3 uncovers a range of previously unidentified challenges encountered by users in 3D printing with programming-based CAD models. While providing valuable insights, the study did not probe deeply into the nuances of these challenges.

A notable issue identified involves difficulties in measuring linear dimensions, which becomes particularly pronounced with curved and organic shapes. Confirming this, prior research, such as by Hudson *et al.* [88], acknowledges measurement as an error-prone aspect in 3D printing workflows. The measurement process extends beyond

mere tool usage and data capture; it encompasses accurate tool operation, interpretation of measurements, and their meaningful integration into the design.

Mahapatra *et al.* [136] explore this process extensively in the context of TinkerCAD, a CAD application following a non-history-based direct manipulation paradigm. Their study outlines a series of challenges in transferring information from the physical to the digital realm, utilizing this data in digital designs, and digitally validating results against physical references. It is plausible that these challenges manifest distinctively across different CAD technologies. To illustrate, consider the task of designing a light switch plate, as investigated by Mahapatra *et al.*. In a CAD application like TinkerCAD, a user measures a light switch to determine the hole's dimensions in the middle of the plate, creating a cube as a subtractive volume in the plate using drag-and-drop interactions. Conversely, in OpenSCAD, the user must account for the positions of other objects and potential nested spatial transformations to position the hole accurately. Moreover, the challenges in a programming-based CAD application based on BREP such as CadQuery might present differently. In essence, the difficulties identified in studies like Mahapatra *et al.*'s could vary significantly when applied to programming-based CAD applications. A better understanding in this process could lead to specialized solutions to facilitate data measurement and integration into models, such as SPATA toolkit [222] intended for direct manipulation CAD applications.

The formative study presented in Chapter 5 delves into the methodology of creating geometric properties in OpenSCAD models with a formative study. While a coded model represents a sequence of steps in constructing a geometry, interpreting the author's intention solely from the code is difficult. Aside from explicit comments, the underlying purpose behind each code statement is not obvious. Future research could focus on a more in-depth exploration of design patterns used by programming-based CAD users, aiming to uncover specific challenges they encounter during the design process. Notably, in the experiment conducted for the study in Chapter 5, some participants expressed a desire for constraint definition options in OpenSCAD, similar to those available in FreeCAD or Fusion360. It remains uncertain whether users would actively utilize such geometric definitions if they were readily accessible or if current limitations or complexities in coding constraints hinder their implementation. A more comprehensive understanding of these aspects could facilitate the development of solutions, like those proposed in this dissertation, to address these challenges more effectively.

This dissertation has primarily focused on improving 3D design in programming-based CAD applications, uncovering various challenges and addressing some of them. However, the research involved only experienced users. Challenges for novices might differ. Future research should investigate novices' experiences with programming-based CAD applications, aiming to understand and alleviate entry barriers, thus increasing access to these applications.

In summary, future research should delve deeper into the nuanced challenges encountered by programming-based CAD applications users. This includes examining the complexities in measuring physical objects and the subsequent data transfer to digital designs. In addition, exploring the general programming patterns and the specific difficulties novice users face in this domain deserves attention. Such investigations could significantly improve our understanding and contribute to facilitating and expanding the use of programming-based CAD applications.

6.2.2.2 *Design challenges*

Editing geometric properties is a crucial task in CAD applications, and our research, as outlined in Chapters 4 and 5, has identified key issues and opportunities for advancement.

Editing geometric properties is arguably the most important task when designing in CAD applications. Our investigation revealed problems addressed by the studies described in Chapters 4 and 5, paving the way for further explorations.

The editing functionality proposed in Section 4.3.3 enables users to directly apply spatial transformations to elements in the view while the system correspondingly updates the code. However, this feature is currently restricted to transformations defined by raw numerical values. When an element is defined using a variable-based expression, the system's response is to merely adjust the value by adding or subtracting a fixed number to achieve the desired result. Two specific user scenarios would greatly benefit from enhanced functionality that includes coherent variable editing, similar to how Sketch-N-Sketch operates with SVG outputs.

The first scenario involves users who initially set variables with arbitrary values, planning to refine these once a more developed model and visual representation are available. Here, users use a trial-and-error method, tweaking one parameter at a time to arrive at satisfactory values. This process can be challenging, particularly when it is difficult to predict how changes in a single parameter might affect the entire code. The second scenario involves debugging the model errors by altering parameter values to detect errors visually. However, this approach often leads to confusing results, making it difficult for users to pinpoint the issue.

A solution that not only allows users to directly edit elements in the view, as proposed in Chapter 4.2, but also enables the system to coherently modify existing variables would be highly beneficial. Such an approach would not alter the code's structure but would adjust the existing variables to reflect the updated model. Users would not have to scrutinize parameters to determine necessary changes tediously; instead, they could directly interact with the view, making the process more efficient. Additionally, this immediate feedback could improve code comprehension, resonating with the principles of direct manipulation [193] and live coding [219].

Nonetheless, this approach poses certain challenges. Modifications in the view could lead to multiple potential solutions, especially if multiple variables define the ge-

ometric property being manipulated. Future research might explore strategies like application-specific heuristics for decision-making, as seen in Sketch-N-Sketch [78], or innovative interactions that present users with different options, allowing them to make informed choices.

The features to define geometric properties, as introduced in Chapter 5, could be further enhanced with direct manipulation interactions. Although these features were well received, they also highlighted certain challenges. Users could easily determine positions in the view and obtain parametric data, requiring them to incorporate this information into the code manually. The separation between the code and the view requires users to mentally interpret the impact of code changes, which can be challenging and demanding.

Future research could investigate ways to simplify the entire process of positioning an object in relation to a parametric position in the view using direct manipulation. An idea could be to allow users to move objects in space, utilizing the features proposed in Chapter 4, while the application employs a 'snap-to-grid' effect at control points, based on the parametric logic detailed in Chapter 5. Such a solution would modify the code coherently.

Another limitation noted in Chapter 5 is that control points are fixed, limiting users to predetermined positions. For example, a participant noted the feature's usefulness for centering a part on a cube's edge but questioned its utility for placing a part at a specific, nonstandard position, like a quarter way along the edge. Future studies could explore ways to generalize information extraction from the view, accommodating more diverse and precise positioning needs. The formative study indicated that geometric properties are typically defined as linear combinations of existing variables, with objects often positioned as simple proportions of these variables. For instance, it is common to see definitions like `translate[2/3*size_cube,0,0]` rather than `translate[1.8596*size_cube,0,0]`. A potential development could enable users to select any part of a model and receive a close approximation of the clicked position in any part of the model based on existing variables. For example, clicking on the edge of a cube defined as `cube([size_x,size_y,size_z])` could prompt the application to return a position like `[(9/10)*size_x,size_y,size_z]`, simultaneously displaying this position visually. This approach would enhance the user experience, allowing for a generalized way to extract parametric information from the view.

Part I

APPENDIX

APPENDIX A

Base questionnaire used in the semi-structured interviews.

1. Are you at least an Advanced Beginner with OpenSCAD? (Are you capable of creating designs and understanding the code of a model?)
2. What is your gender?
3. How old are you?
4. What is your academic background?
5. What is your current job?
6. Do you have experience with 3D printing?
7. If yes, tell me about your experience with 3D printing. When did you start? What were your motivations?
8. How often do you 3D print?
 - Daily
 - Weekly
 - Monthly
 - Every two months
 - Every semester
 - Less often

Here, we define the terms of direct manipulation and programming-based paradigms that we use in the rest of the interview.

9. What **direct manipulation CAD** applications have you used before, and what is your experience in each? Name of the application and skill level
 - 1 - Novice
 - 2 - Advanced Beginner
 - 3 - Competent
 - 4 - Proficient

- 5 - Expert
10. Other than CAD, what other **programming languages**, in general, have you used, and what is your skill level in each one? Programming language name and skill level
 - 1 - Novice
 - 2 - Advanced Beginner
 - 3 - Competent
 - 4 – Proficient
 - 5 - Expert
 11. What is your skill level in **OpenSCAD** ?
 - 1 - Novice
 - 2 - Advanced Beginner
 - 3 - Competent
 - 4 – Proficient
 - 5 - Expert
 12. What motivated you to learn/use OpenSCAD specifically? How did you start? Did you try other applications? Why OpenSCAD and no others?
 13. Let's talk about the last three objects you 3D printed. Describe the object and motivation.
 14. Would you say that, in general, you 3D print for the motivations mentioned before, or are there other main reasons you print for?
 15. How did you get the design for those objects? Design them from scratch, Pre-existing models
 16. How do you normally get your models? Design them from scratch, Pre-existing models
 17. What CAD applications did you use to design/edit your last three objects and why?
 18. What were the major difficulties you found in the process of fabricating these objects (Including all the processes, ideation, design, configuration, printing, iteration, etc)? What is the most time-consuming part? What brings more uncertainty? (What makes you iterate more?)

19. In general, what are the most difficult parts of the fabrication process (including all the processes, ideation, design, configuration, printing, iteration, etc)? What is the most time-consuming part? What brings more uncertainty? (What makes you iterate more?)
20. What factors bring more uncertainty or usually make you iterate more times?
21. Specifically in the model design part, what is the most difficult and time-consuming part? Is it something related to the software? Is it different when you use direct manipulation than programming-based?
22. If different than the previous answer, Specifically in OpenSCAD, what is the most difficult and time-consuming part?
23. Do you need to measure physical sizes to transfer them into the digital design? How do you do it? What tools and strategies do you use? How do you verify the correctness of the measurements?
24. Tell me about measurement difficulties, Linear measurements, Curved and organic shape measurements
25. Generally, when you 3D print, how often do you design the models you print from scratch? Why?
 - (0%) Never
 - (1% - 20%) Rarely
 - (20% - 40%) Often
 - (40% - 60%) Sometimes
 - (60% - 80%) Frequently
 - (80% - 99%) Very frequently
 - (100%) Always
26. Generally, when you 3D print, how often do you use a pre-existing model for the models you print (previous projects, friend's model, website model)? Why?
 - (0%) Never
 - (1% - 20%) Rarely
 - (20% - 40%) Often
 - (40% - 60%) Sometimes
 - (60% - 80%) Frequently
 - (80% - 99%) Very frequently

- (100%) Always
27. What motivates you to design from scratch or to use a pre-existing model?
28. Do you know model-storing websites such as Thingiverse? What others?
29. If yes, what do you think about them? Do you use them? Do you find them useful?
30. If you know Thingiverse, have you used the Customizer tool? Talk to me about your experience with this tool.
31. When you use pre-existing models, in what format do you get them? (stl, obj, code...)
32. When you re-use a **non-coded** pre-existing model, how often do you need to edit it? What types of modifications do you make?
- (0%) Never
 - (1% - 20%) Rarely
 - (20% - 40%) Often
 - (40% - 60%) Sometimes
 - (60% - 80%) Frequently
 - (80% - 99%) Very frequently
 - (100%) Always
33. When you re-use a **non-coded** pre-existing model, how difficult is it to edit it? Explain what applications you use and why the level of difficulty you selected.
- 1- Very easy
 - 2- Easy
 - 3- Neutral
 - 4- Difficult
 - 5- Very difficult
34. When you re-use a **coded** pre-existing model, how often do you need to edit it? What types of modifications do you make?
- (0%) Never
 - (1% - 20%) Rarely
 - (20% - 40%) Often
 - (40% - 60%) Sometimes

- (60% - 80%) Frequently
 - (80% - 99%) Very frequently
 - (100%) Always
35. When you re-use a **coded** pre-existing model, how difficult is it to edit it? Explain what applications you use and why the level of difficulty you selected.
- 1- Very easy
 - 2- Easy
 - 3- Neutral
 - 4- Difficult
 - 5- Very difficult
36. Did you bring some of your previous OpenSCAD projects? Talk to me about one of them, How was the process, how many iterations did you need, and what was the most difficult part of the process?
37. **(Hands-on exercise)** I will ask you to localize in the code the specific statements that create a part that I will point out in the view. Share aloud the thinking process you follow to find it. Is this a task you normally do when designing: looking for a specific part in the code based on the view? What is the hardest part of doing it? What strategies do you use normally?
38. How difficult was the task?
39. In OpenSCAD (and programming-based), how easily can you link the output in the view to the code?
40. What would you say is the best of OpenSCAD and the worst? What would you say is the best of programming-based CAD and the worst?
41. What are the advantages and disadvantages of direct manipulation and programming-based applications like OpenSCAD? When do you prefer to use one or the other?

APPENDIX B



- | | | | |
|---|---|---|---|
|  |  |  |  |
| (i) Multi-Color Pencil Cup,
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) 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 Holder,
Thing:6255969 |
|  |  |  |  |
| (xiii) Battery End Caps,
Thing:6248029 | (xiv) Small Box,
Thing:6266913 | (xv) Infinity Cube,
Thing:6249758 | (xvi) Key Tag,
Thing:6249968 |



Figure 41: OpenSCAD models taken from Thingiverse for the formative study.

BIBLIOGRAPHY

- [1] Customizer.net 3D. *3dCustomizer.net: Design custom 3D Models for 3D printing*. 2024. URL: <https://www.3dcustomizer.net> (visited on 01/16/2024).
- [2] Alfred Aho, Monica Lam, Ravi Sethi, and Jeffrey Ullman. *Compilers - Principles, Techniques, and Tools*. 2nd ed. Boston: Pearson/Addison Wesley, 2006. ISBN: 0-321-48681-1 978-0-321-48681-3. URL: <https://dl.acm.org/doi/10.5555/1177220>.
- [3] Robert Aish. "DesignScript: Origins, Explanation, Illustration." In: *Computational Design Modelling*. Ed. by Christoph Gengnagel, Axel Kilian, Norbert Palz, and Fabian Scheurer. Berlin, Heidelberg: Springer, 2012, pp. 1–8. ISBN: 978-3-642-23435-4. DOI: [10.1007/978-3-642-23435-4_1](https://doi.org/10.1007/978-3-642-23435-4_1).
- [4] Robert Aish. "DesignScript: Scalable Tools for Design Computation." In: *Stouffs, Rudi and Sariyildiz, Sevil (eds.), Computation and Performance – Proceedings of the 31st eCAADe Conference – Volume 2, Faculty of Architecture, Delft University of Technology, Delft, The Netherlands, 18-20 September 2013, pp. 87-95*. Delft, The Netherlands: CUMINCAD, 2013, pp. 87–95. URL: http://papers.cumincad.org/cgi-bin/works/BrowseTreefield=seriesorder=AZ/Show?ecaade2013_297 (visited on 06/30/2021).
- [5] Faraz Akbar. "Comparative study of hatchback vehicles through modelling and computational tools." In: *Mehran University Research Journal of Engineering and Technology* 41.3 (July 2022). Number: 3, pp. 149–160. ISSN: 2413-7219. DOI: [10.22581/muet1982.2203.15](https://doi.org/10.22581/muet1982.2203.15). URL: <https://publications.muet.edu.pk/index.php/muetrj/article/view/2526> (visited on 12/28/2023).
- [6] Celena Alcock, Nathaniel Hudson, and Parmit K. Chilana. "Barriers to Using, Customizing, and Printing 3D Designs on Thingiverse." In: *Proceedings of the 19th International Conference on Supporting Group Work. GROUP '16*. New York, NY, USA: Association for Computing Machinery, Nov. 2016, pp. 195–199. ISBN: 978-1-4503-4276-6. DOI: [10.1145/2957276.2957301](https://doi.org/10.1145/2957276.2957301). URL: <http://doi.org/10.1145/2957276.2957301> (visited on 02/05/2021).
- [7] Amabilis. *Amabilis Software*. 2022. URL: <http://amabilis.com/> (visited on 06/30/2021).
- [8] D. C. Anderson and T. C. Chang. "Geometric reasoning in feature-based design and process planning." In: *Computers & Graphics* 14.2 (Jan. 1990), pp. 225–235. ISSN: 0097-8493. DOI: [10.1016/0097-8493\(90\)90034-U](https://doi.org/10.1016/0097-8493(90)90034-U). URL: <https://www.sciencedirect.com/science/article/pii/009784939090034U> (visited on 08/01/2023).

- [9] David Anderson, James L. Frankel, Joe Marks, Aseem Agarwala, Paul Beardsley, Jessica Hodgins, Darren Leigh, Kathy Ryall, Eddie Sullivan, and Jonathan S. Yedidia. "Tangible interaction + graphical interpretation: a new approach to 3D modeling." In: *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*. SIGGRAPH '00. USA: ACM Press/Addison-Wesley Publishing Co., July 2000, pp. 393–402. ISBN: 978-1-58113-208-3. DOI: [10.1145/344779.344960](https://doi.org/10.1145/344779.344960). (Visited on 10/20/2022).
- [10] Marwan Ansari, ed. *Game development tools*. Boca Raton, FL: CRC Press, 2011. ISBN: 978-1-56881-432-2.
- [11] Ian Arawjo, Anthony DeArmas, Michael Roberts, Shrutarshi Basu, and Tapan Parikh. "Notational Programming for Notebook Environments: A Case Study with Quantum Circuits." In: *Proceedings of the 35th Annual ACM Symposium on User Interface Software and Technology*. UIST '22. New York, NY, USA: Association for Computing Machinery, Oct. 2022, pp. 1–20. ISBN: 978-1-4503-9320-1. DOI: [10.1145/3526113.3545619](https://doi.org/10.1145/3526113.3545619). (Visited on 05/06/2024).
- [12] Ryan Arisandi, Yusuke Takami, Mai Otsuki, Asako Kimura, Fumihisa Shibata, and Hideyuki Tamura. "Enjoying virtual handcrafting with ToolDevice." In: *Adjunct proceedings of the 25th annual ACM symposium on User interface software and technology*. UIST Adjunct Proceedings '12. New York, NY, USA: Association for Computing Machinery, Oct. 2012, pp. 17–18. ISBN: 978-1-4503-1582-1. DOI: [10.1145/2380296.2380306](https://doi.org/10.1145/2380296.2380306). (Visited on 10/20/2022).
- [13] Daniel Ashbrook, Shitao Stan Guo, and Alan Lambie. "Towards Augmented Fabrication: Combining Fabricated and Existing Objects." In: *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems*. CHI EA '16. New York, NY, USA: Association for Computing Machinery, May 2016, pp. 1510–1518. ISBN: 978-1-4503-4082-3. DOI: [10.1145/2851581.2892509](https://doi.org/10.1145/2851581.2892509). (Visited on 03/11/2021).
- [14] Nilüfer Atman Uslu, Hatice Yildiz Durak, and Gökçe Mehmet Ay. "Comparing reflective and supportive scaffolding in 3D computer-aided design course: Engineering students' metacognitive strategies, spatial ability self-efficacy, and spatial anxiety." In: *Computer Applications in Engineering Education* 30.5 (2022), pp. 1454–1469. ISSN: 1099-0542. DOI: [10.1002/cae.22531](https://doi.org/10.1002/cae.22531). URL: <https://onlinelibrary.wiley.com/doi/abs/10.1002/cae.22531> (visited on 01/07/2024).
- [15] Autodesk. *Tinkercad | Create 3D digital designs with online CAD*. 2020. URL: <https://www.tinkercad.com/> (visited on 10/10/2021).
- [16] Autodesk. *AutoCAD for Mac & Windows | 2D/3D CAD Software | Autodesk*. 2022. URL: <https://www.autodesk.ca/en/products/autocad/overview> (visited on 06/30/2021).
- [17] Autodesk. *Logiciel Maya | Connaître les prix et acheter le logiciel Maya 2024 officiel*. 2023. URL: <https://www.autodesk.fr/products/maya/overview> (visited on 12/28/2023).

- [18] R. Balzer. "A 15 Year Perspective on Automatic Programming." In: *IEEE Transactions on Software Engineering* SE-11.11 (Nov. 1985), pp. 1257–1268. ISSN: 1939-3520. DOI: [10.1109/TSE.1985.231877](https://doi.org/10.1109/TSE.1985.231877). URL: <https://ieeexplore.ieee.org/document/1701945> (visited on 11/14/2023).
- [19] F. Bancilhon and N. Spyratos. "Update semantics of relational views." In: *ACM Transactions on Database Systems* 6.4 (1981), pp. 557–575. ISSN: 0362-5915. DOI: [10.1145/319628.319634](https://doi.org/10.1145/319628.319634). URL: <https://doi.org/10.1145/319628.319634> (visited on 07/13/2022).
- [20] Giles Bathgate. *RapCAD*. 2023. URL: <https://gilesbathgate.com/category/rapcad/> (visited on 09/09/2023).
- [21] Mikhail E. Belkin, Tatiana Bakhvalova, Vladislav Golovin, Yuriy Tyschuk, and Alexander S. Sigov. "Modeling and Simulation in Microwave-Photonics Applications." In: *Modeling and Simulation in Engineering - Selected Problems*. IntechOpen, Apr. 2020. ISBN: 978-1-83968-250-6. DOI: [10.5772/intechopen.91940](https://doi.org/10.5772/intechopen.91940). URL: <https://www.intechopen.com/chapters/71601> (visited on 12/28/2023).
- [22] Srinjita Bhaduri, Quentin L Biddy, Jeffrey Bush, Abhijit Suresh, and Tamara Sumner. "3DnST: A Framework Towards Understanding Children's Interaction with Tinkercad and Enhancing Spatial Thinking Skills." In: *Proceedings of the 20th Annual ACM Interaction Design and Children Conference*. IDC '21. New York, NY, USA: Association for Computing Machinery, June 2021, pp. 257–267. ISBN: 978-1-4503-8452-0. DOI: [10.1145/3459990.3460717](https://doi.org/10.1145/3459990.3460717). URL: <https://dl.acm.org/doi/10.1145/3459990.3460717> (visited on 07/24/2023).
- [23] Natalia Bogdan, Tovi Grossman, and George Fitzmaurice. "HybridSpace: Integrating 3D freehand input and stereo viewing into traditional desktop applications." In: *2014 IEEE Symposium on 3D User Interfaces (3DUI)*. Mar. 2014, pp. 51–58. DOI: [10.1109/3DUI.2014.6798842](https://doi.org/10.1109/3DUI.2014.6798842).
- [24] Richard E. Boyatzis. *Transforming qualitative information: Thematic analysis and code development*. Transforming qualitative information: Thematic analysis and code development. Pages: xvi, 184. Thousand Oaks, CA, US: Sage Publications, Inc, 1998. ISBN: 978-0-7619-0960-6 978-0-7619-0961-3.
- [25] Virginia Braun and Victoria Clarke. *Successful qualitative research: a practical guide for beginners*. OCLC: ocn811733656. Los Angeles: SAGE, 2013. ISBN: 978-1-84787-581-5 978-1-84787-582-2.
- [26] Virginia Braun and Victoria Clarke. "Can I use TA? Should I use TA? Should I not use TA? Comparing reflexive thematic analysis and other pattern-based qualitative analytic approaches." In: *Counselling and Psychotherapy Research* 21.1 (2021), pp. 37–47. ISSN: 1746-1405. DOI: [10.1002/capr.12360](https://doi.org/10.1002/capr.12360). URL: <https://onlinelibrary.wiley.com/doi/abs/10.1002/capr.12360> (visited on 09/10/2023).

- [27] Virginia Braun, Victoria Clarke, and V Hayfield. *Answers to frequently asked questions about thematic analysis April 2019.pdf*. 2019. URL: <https://cdn.auckland.ac.nz/assets/psych/about/our-research/documents/Answers%20to%20frequently%20asked%20questions%20about%20thematic%20analysis%20April%202019.pdf> (visited on 09/10/2023).
- [28] Ltd Browzwear Solutions Pte. *VStitcher: 3D Fabric Design Software & Photorealistic 3D Rendering*. 2023. URL: <https://browzwear.com/products/v-stitcher> (visited on 12/28/2023).
- [29] Mark Brunelli. *Parametric vs. Direct Modeling: Which Side Are You On?* Dec. 2022. URL: <https://www.ptc.com/en/blogs/cad/parametric-vs-direct-modeling-which-side-are-you-on?> (visited on 08/02/2023).
- [30] Erin Buehler, Amy Hurst, and Megan Hofmann. "Coming to grips: 3D printing for accessibility." In: *Proceedings of the 16th international ACM SIGACCESS conference on Computers & accessibility*. ASSETS '14. New York, NY, USA: Association for Computing Machinery, Oct. 2014, pp. 291–292. ISBN: 978-1-4503-2720-6. DOI: [10.1145/2661334.2661345](https://doi.org/10.1145/2661334.2661345). URL: <https://doi.org/10.1145/2661334.2661345> (visited on 11/12/2021).
- [31] Erin Buehler, Shaun K. Kane, and Amy Hurst. "ABC and 3D: opportunities and obstacles to 3D printing in special education environments." In: *Proceedings of the 16th international ACM SIGACCESS conference on Computers & accessibility*. ASSETS '14. New York, NY, USA: Association for Computing Machinery, Oct. 2014, pp. 107–114. ISBN: 978-1-4503-2720-6. DOI: [10.1145/2661334.2661365](https://doi.org/10.1145/2661334.2661365). URL: <https://doi.org/10.1145/2661334.2661365> (visited on 11/12/2021).
- [32] Richard W. Bukowski and Carlo H. Séquin. "Object associations: a simple and practical approach to virtual 3D manipulation." In: *Proceedings of the 1995 symposium on Interactive 3D graphics*. I3D '95. New York, NY, USA: Association for Computing Machinery, 1995, 131–ff. ISBN: 978-0-89791-736-0. DOI: [10.1145/199404.199427](https://dl.acm.org/doi/10.1145/199404.199427). URL: <https://dl.acm.org/doi/10.1145/199404.199427> (visited on 01/18/2024).
- [33] David Byrne. "A worked example of Braun and Clarke's approach to reflexive thematic analysis." In: *Quality & Quantity* 56.3 (June 2022), pp. 1391–1412. ISSN: 1573-7845. DOI: [10.1007/s11135-021-01182-y](https://doi.org/10.1007/s11135-021-01182-y). URL: <https://doi.org/10.1007/s11135-021-01182-y> (visited on 02/27/2023).
- [34] Pierre E. Bézier. "UNISURF, from styling to tool-shop." In: *Computers in Industry* 4.2 (June 1983), pp. 115–126. ISSN: 0166-3615. DOI: [10.1016/0166-3615\(83\)90017-9](https://www.sciencedirect.com/science/article/pii/0166361583900179). URL: <https://www.sciencedirect.com/science/article/pii/0166361583900179> (visited on 12/30/2023).
- [35] CadQuery. *CadQuery*. original-date: 2018-10-28T17:57:18Z. Sept. 2023. URL: <https://github.com/CadQuery/cadquery> (visited on 09/06/2023).

- [36] Jorge D. Camba, Manuel Contero, and Pedro Company. "Parametric CAD modeling: An analysis of strategies for design reusability." In: *Computer-Aided Design* 74 (May 2016), pp. 18–31. ISSN: 0010-4485. DOI: [10.1016/j.cad.2016.01.003](https://doi.org/10.1016/j.cad.2016.01.003). URL: <https://www.sciencedirect.com/science/article/pii/S0010448516000051> (visited on 01/09/2024).
- [37] D. Cascaval, M. Shalah, P. Quinn, R. Bodik, M. Agrawala, and A. Schulz. "Differentiable 3D CAD Programs for Bidirectional Editing." In: *Computer Graphics Forum* 41.2 (2022), pp. 309–323. ISSN: 1467-8659. DOI: [10.1111/cgf.14476](https://doi.org/10.1111/cgf.14476). URL: <https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.14476> (visited on 06/30/2023).
- [38] DEVCOM Analysis Center. *BRL-CAD: Open Source Solid Modeling*. 2023. URL: <https://brlcad.org/> (visited on 09/12/2023).
- [39] Erik Champion and Hafizur Rahaman. "Survey of 3D digital heritage repositories and platforms." In: *Virtual Archaeology Review* 11.23 (July 2020). Number: 23, pp. 1–15. ISSN: 1989-9947. DOI: [10.4995/var.2020.13226](https://doi.org/10.4995/var.2020.13226). URL: <https://polipapers.upv.es/index.php/var/article/view/13226> (visited on 12/06/2023).
- [40] Ravi Chugh, Brian Hempel, Mitchell Spradlin, and Jacob Albers. "Programmatic and direct manipulation, together at last." In: *ACM SIGPLAN Notices* 51.6 (June 2016), pp. 341–354. ISSN: 0362-1340. DOI: [10.1145/2980983.2908103](https://doi.org/10.1145/2980983.2908103). URL: <https://doi.org/10.1145/2980983.2908103> (visited on 11/04/2022).
- [41] C. Chytas, A. Tsilingiris, and I. Diethelm. "Exploring Computational Thinking Skills in 3D Printing: A Data Analysis of an Online Makerspace." In: *2019 IEEE Global Engineering Education Conference (EDUCON)*. ISSN: 2165-9567. Apr. 2019, pp. 1173–1179. DOI: [10.1109/EDUCON.2019.8725202](https://doi.org/10.1109/EDUCON.2019.8725202).
- [42] Leonardo Ciocca, Massimiliano Fantini, Francesca De Crescenzo, Franco Persiani, and Roberto Scotti. "Computer-aided design and manufacturing construction of a surgical template for craniofacial implant positioning to support a definitive nasal prosthesis." In: *Clinical Oral Implants Research* 22.8 (2011), pp. 850–856. ISSN: 1600-0501. DOI: [10.1111/j.1600-0501.2010.02066.x](https://doi.org/10.1111/j.1600-0501.2010.02066.x). URL: <https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1600-0501.2010.02066.x> (visited on 12/28/2023).
- [43] Library of Congress. *STL (STereoLithography) File Format Family*. web page. Sept. 2019. URL: <https://www.loc.gov/preservation/digital/formats/fdd/fdd000504.shtml> (visited on 09/08/2022).
- [44] Steven Anson Coons. "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). New York, NY, USA: Association for Computing Machinery, May 1963, pp. 299–304. ISBN: 978-1-4503-7880-2. DOI: [10.1145/1461551.1461588](https://doi.org/10.1145/1461551.1461588). URL: <https://dl.acm.org/doi/10.1145/1461551.1461588> (visited on 09/21/2023).

- [45] Krzysztof Czarnecki, J. Nathan Foster, Zhenjiang Hu, Ralf Lämmel, Andy Schürr, and James F. Terwilliger. “Bidirectional Transformations: A Cross-Discipline Perspective.” In: *Theory and Practice of Model Transformations*. Ed. by Richard F. Paige. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2009, pp. 260–283. ISBN: 978-3-642-02408-5. DOI: [10.1007/978-3-642-02408-5_19](https://doi.org/10.1007/978-3-642-02408-5_19).
- [46] Georgi Dalakov. *Ben Gurley (PDP-1) – Computer Timeline*. 2023. URL: <http://www.computer-timeline.com/timeline/ben-gurley/> (visited on 11/27/2023).
- [47] Afshin Darian, Brian Granger, Jason Grout, Fernando Pérez, Ana Ruvalcaba, and Zach Sailer. *Project Jupyter*. May 2024. URL: <https://jupyter.org> (visited on 05/06/2024).
- [48] SolidWorks Corporation Dassault Systèmes. *3D CAD Design Software | SOLIDWORKS*. 2023. URL: <https://www.solidworks.com/home-page-2021> (visited on 09/21/2023).
- [49] Systèmes Dassault. *CATIA*. Dec. 2023. URL: <https://www.3ds.com/products/catia> (visited on 12/28/2023).
- [50] Scott Davidson. *Grasshopper*. 2023. URL: <https://www.grasshopper3d.com/> (visited on 11/22/2023).
- [51] Bruno R. De Araùjo, Géry Casiez, and Joaquim A. Jorge. “Mockup builder: direct 3D modeling on and above the surface in a continuous interaction space.” In: *Proceedings of Graphics Interface 2012*. GI ’12. CAN: Canadian Information Processing Society, May 2012, pp. 173–180. ISBN: 978-1-4503-1420-6. (Visited on 10/26/2020).
- [52] Zinovy Diskin. “Model Synchronization: Mappings, Tiles, and Categories.” In: *Generative and Transformational Techniques in Software Engineering III: International Summer School, GTTSE 2009, Braga, Portugal, July 6-11, 2009. Revised Papers*. Ed. by João M. Fernandes, Ralf Lämmel, Joost Visser, and João Saraiva. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2011, pp. 92–165. ISBN: 978-3-642-18023-1. DOI: [10.1007/978-3-642-18023-1_3](https://doi.org/10.1007/978-3-642-18023-1_3). URL: https://doi.org/10.1007/978-3-642-18023-1_3 (visited on 07/13/2022).
- [53] Pierre Dragicevic, Stéphane Huot, and Fanny Chevalier. “Gliimpse: Animating from markup code to rendered documents and vice versa.” In: *Proceedings of the 24th annual ACM symposium on User interface software and technology*. UIST ’11. New York, NY, USA: Association for Computing Machinery, Oct. 2011, pp. 257–262. ISBN: 978-1-4503-0716-1. DOI: [10.1145/2047196.2047229](https://doi.org/10.1145/2047196.2047229). URL: <https://doi.org/10.1145/2047196.2047229> (visited on 09/09/2022).
- [54] Niklas Elmqvist and Jean-Daniel Fekete. “Semantic Pointing for Object Picking in Complex 3D Environments.” In: *ACM*, May 2008, p. 243. URL: <https://inria.hal.science/hal-00851711> (visited on 01/18/2024).

- [55] Components Essentra. *Is CAD still relevant in the manufacturing industry?* July 2022. URL: <https://www.essentracomponents.com/en-gb/news/trends/industry-40/is-cad-still-relevant-in-the-manufacturing-industry> (visited on 11/21/2023).
- [56] Kim Eui Song. *IdeaSpace / Blog*. June 2020. URL: <https://www.spacesmith.com/ideaspace/the-history-of-cad> (visited on 11/14/2023).
- [57] M. Fiorentino, R. de Amicis, G. Monno, and A. Stork. "Spacedesign: a mixed reality workspace for aesthetic industrial design." In: *Proceedings. International Symposium on Mixed and Augmented Reality*. Oct. 2002, pp. 86–318. DOI: [10.1109/ISMAR.2002.1115077](https://doi.org/10.1109/ISMAR.2002.1115077).
- [58] Sebastian Fischer, ZhenJiang Hu, and Hugo Pacheco. "The essence of bidirectional programming." In: *Science China Information Sciences* 58.5 (May 2015), pp. 1–21. ISSN: 1869-1919. DOI: [10.1007/s11432-015-5316-8](https://doi.org/10.1007/s11432-015-5316-8). URL: <https://doi.org/10.1007/s11432-015-5316-8> (visited on 07/13/2022).
- [59] James D. Foley, Foley Dan Van, Andries Van Dam, Steven K. Feiner, and John F. Hughes. *Computer Graphics: Principles and Practice*. Addison-Wesley Professional, 1996. ISBN: 978-0-201-84840-3.
- [60] Sean Follmer, David Carr, Emily Lovell, and Hiroshi Ishii. "CopyCAD: remixing physical objects with copy and paste from the real world." In: *Adjunct proceedings of the 23rd annual ACM symposium on User interface software and technology*. UIST '10. New York, NY, USA: Association for Computing Machinery, Oct. 2010, pp. 381–382. ISBN: 978-1-4503-0462-7. DOI: [10.1145/1866218.1866230](https://doi.org/10.1145/1866218.1866230). URL: <https://doi.org/10.1145/1866218.1866230> (visited on 10/20/2022).
- [61] Sean Follmer and Hiroshi Ishii. "KidCAD: digitally remixing toys through tangible tools." In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '12. New York, NY, USA: Association for Computing Machinery, May 2012, pp. 2401–2410. ISBN: 978-1-4503-1015-4. DOI: [10.1145/2207676.2208403](https://doi.org/10.1145/2207676.2208403). URL: <https://doi.org/10.1145/2207676.2208403> (visited on 10/20/2022).
- [62] Blender Foundation. *blender.org - Home of the Blender project - Free and Open 3D Creation Software*. 2023. URL: <https://www.blender.org/> (visited on 11/18/2023).
- [63] FreeCAD. *Python scripting tutorial - FreeCAD Documentation*. 2023. URL: https://wiki.freecad.org/Python_scripting_tutorial/en (visited on 09/12/2023).
- [64] FreeCAD. *Topological naming problem - FreeCAD Documentation*. 2023. URL: https://wiki.freecad.org/Topological_naming_problem (visited on 08/01/2023).
- [65] David M Frohlich. "The history and future of direct manipulation." In: *Behaviour & Information Technology* 12.6 (Nov. 1993). Publisher: Taylor & Francis, pp. 315–329. ISSN: 0144-929X. DOI: [10.1080/01449299308924396](https://doi.org/10.1080/01449299308924396). URL: <https://doi.org/10.1080/01449299308924396> (visited on 07/05/2021).

- [66] Shuming Gao, Huagen Wan, and Qunsheng Peng. "An approach to solid modeling in a semi-immersive virtual environment." In: *Computers & Graphics* 24.2 (Apr. 2000), pp. 191–202. ISSN: 0097-8493. DOI: [10.1016 / S0097 - 8493\(99 \) 00154 - 5](https://doi.org/10.1016/S0097-8493(99)00154-5). URL: <https://www.sciencedirect.com/science/article/pii/S0097849399001545> (visited on 10/20/2022).
- [67] Neil Gershenfeld. *Fab: The Coming Revolution on Your Desktop—from Personal Computers to Personal Fabrication*. USA: Basic Books, Inc., 2007. ISBN: 978-0-465-02746-0.
- [68] Ian Gibson, David Rosen, and Brent Stucker. *Additive manufacturing technologies: 3D printing, rapid prototyping and direct digital manufacturing*. Second Edition. New York Heidelberg Dodrecht London: Springer, 2015. ISBN: 978-1-4939-2112-6 978-1-4939-4455-2.
- [69] VERBI GmbH. *Software de Análisis de Datos Cualitativos - MAXQDA*. 2023. URL: <https://www.maxqda.com> (visited on 09/10/2023).
- [70] Camille Gobert and Michel Beaudouin-Lafon. "i-LaTeX : Manipulating Transitional Representations between LaTeX Code and Generated Documents." In: *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*. CHI '22. New York, NY, USA: Association for Computing Machinery, 2022, pp. 1–16. ISBN: 978-1-4503-9157-3. DOI: [10.1145/3491102.3517494](https://doi.org/10.1145/3491102.3517494). URL: <https://doi.org/10.1145/3491102.3517494> (visited on 09/09/2022).
- [71] J Felipe Gonzalez, Danny Kieken, Thomas Pietrzak, Audrey Girouard, and Géry Casiez. "Introducing Bidirectional Programming in Constructive Solid Geometry-Based CAD." In: *Proceedings of the 2023 ACM Symposium on Spatial User Interaction*. SUI '23. Sydney, Australia: Association for Computing Machinery, Oct. 2023, pp. 1–12. ISBN: 9798400702815. DOI: [10.1145 / 3607822.3614521](https://doi.org/10.1145/3607822.3614521). URL: <https://dl.acm.org/doi/10.1145/3607822.3614521> (visited on 11/05/2023).
- [72] Georg Gottlob, Paolo Paolini, and Roberto Zicari. "Properties and update semantics of consistent views." In: *ACM Transactions on Database Systems* 13.4 (Oct. 1988), pp. 486–524. ISSN: 0362-5915. DOI: [10.1145 / 49346.50068](https://doi.org/10.1145/49346.50068). URL: <https://doi.org/10.1145/49346.50068> (visited on 07/13/2022).
- [73] Katherine Celia Greder, Jie Pei, and Jooyoung Shin. "Design in 3D: a computational fashion design protocol." In: *International Journal of Clothing Science and Technology* 32.4 (Jan. 2020). Publisher: Emerald Publishing Limited, pp. 537–549. ISSN: 0955-6222. DOI: [10.1108/IJCST-07-2019-0110](https://doi.org/10.1108/IJCST-07-2019-0110). URL: <https://doi.org/10.1108/IJCST-07-2019-0110> (visited on 12/28/2023).
- [74] Daniel Groeger, Elena Chong Loo, and Jürgen Steimle. "HotFlex: Post-print Customization of 3D Prints Using Embedded State Change." In: *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. CHI '16. New York, NY, USA: Association for Computing Machinery, May 2016, pp. 420–432.

- ISBN: 978-1-4503-3362-7. DOI: [10.1145/2858036.2858191](https://doi.org/10.1145/2858036.2858191). URL: <https://doi.org/10.1145/2858036.2858191> (visited on 10/25/2022).
- [75] Tovi Grossman and Ravin Balakrishnan. "Pointing at trivariate targets in 3D environments." In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '04. New York, NY, USA: Association for Computing Machinery, 2004, pp. 447–454. ISBN: 978-1-58113-702-6. DOI: [10.1145/985692.985749](https://doi.org/10.1145/985692.985749). URL: <https://dl.acm.org/doi/10.1145/985692.985749> (visited on 01/03/2024).
- [76] Özge Gül. "A Study on Instructional Methods Used in CAD Courses in Interior Architecture Education." In: *Procedia - Social and Behavioral Sciences*. International Conference on New Horizons in Education, INTE 2014, 25-27 June 2014, Paris, France 174 (Feb. 2015), pp. 1758–1763. ISSN: 1877-0428. DOI: [10.1016/j.sbspro.2015.01.834](https://doi.org/10.1016/j.sbspro.2015.01.834). URL: <https://www.sciencedirect.com/science/article/pii/S1877042815008861> (visited on 12/28/2023).
- [77] Brian Hempel and Ravi Chugh. "Maniposynth: Bimodal Tangible Functional Programming." In: (2022), 29 pages, 3971254 bytes. ISSN: 1868-8969. DOI: [10.4230/LIPIcs.ECOOP.2022.16](https://doi.org/10.4230/LIPIcs.ECOOP.2022.16). URL: <http://arxiv.org/abs/2206.14992> (visited on 05/06/2024).
- [78] Brian Hempel, Justin Lubin, and Ravi Chugh. "Sketch-n-Sketch: Output-Directed Programming for SVG." In: *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology*. UIST '19. New York, NY, USA: Association for Computing Machinery, Oct. 2019, pp. 281–292. ISBN: 978-1-4503-6816-2. DOI: [10.1145/3332165.3347925](https://doi.org/10.1145/3332165.3347925). URL: <https://doi.org/10.1145/3332165.3347925> (visited on 05/17/2021).
- [79] Monique M. Hennink, Bonnie N. Kaiser, and Vincent C. Marconi. "Code Saturation Versus Meaning Saturation: How Many Interviews Are Enough?" In: *Qualitative Health Research* 27.4 (Mar. 2017). Publisher: SAGE Publications Inc, pp. 591–608. ISSN: 1049-7323. DOI: [10.1177/1049732316665344](https://doi.org/10.1177/1049732316665344). URL: <https://doi.org/10.1177/1049732316665344> (visited on 12/04/2023).
- [80] Achten Henri H. "New Design Methods for Computer Aided Architectural Design Methodology Teaching." In: *International Journal of Architectural Computing* 1.1 (Jan. 2003). Publisher: SAGE Publications, pp. 72–91. ISSN: 1478-0771. DOI: [10.1260/147807703322467441](https://doi.org/10.1260/147807703322467441). URL: <https://doi.org/10.1260/147807703322467441> (visited on 12/28/2023).
- [81] AB Hexagon. *Intergraph Smart 3D*. 2023. URL: <https://hexagon.com/products/intergraph-smart-3d> (visited on 12/28/2023).
- [82] R. C. Hillyard and I. C. Braid. "Analysis of dimensions and tolerances in computer-aided mechanical design." In: *Computer-Aided Design* 10.3 (May 1978), pp. 161–166. ISSN: 0010-4485. DOI: [10.1016/0010-4485\(78\)90140-9](https://doi.org/10.1016/0010-4485(78)90140-9). URL: <https://www.sciencedirect.com/science/article/pii/0010448578901409> (visited on 09/21/2023).
- [83] Nicholas Hoff. *3code*. 2024. URL: <https://3code.io/> (visited on 05/06/2024).

- [84] Christoph M. Hoffmann. *Geometric and solid modeling: an introduction*. The Morgan Kaufmann series in computer graphics and geometric modeling. San Mateo, Calif: Morgan Kaufmann, 1989. ISBN: 978-1-55860-067-6.
- [85] Akihiro Hori, Masumi Kawakami, and Makoto Ichii. "CodeHouse: VR Code Visualization Tool." In: *2019 Working Conference on Software Visualization (VIS-SOFT)*. Sept. 2019, pp. 83–87. DOI: [10.1109/VIS-SOFT.2019.00018](https://doi.org/10.1109/VIS-SOFT.2019.00018). URL: <https://ieeexplore.ieee.org/document/8900968> (visited on 12/05/2023).
- [86] Diogo Horst, Charles Duvoisin, and Rogerio Vieira. "Additive Manufacturing at Industry 4.0: a Review." In: *International Journal of Engineering and Technical Research* 8 (Sept. 2018), pp. 3–8.
- [87] Zhenjiang Hu, Shin-Cheng Mu, and Masato Takeichi. "A programmable editor for developing structured documents based on bidirectional transformations." In: *Proceedings of the 2004 ACM SIGPLAN symposium on Partial evaluation and semantics-based program manipulation*. PEPM '04. New York, NY, USA: Association for Computing Machinery, 2004, pp. 178–189. ISBN: 978-1-58113-835-1. DOI: [10.1145 / 1014007.1014025](https://doi.org/10.1145/1014007.1014025). URL: <https://doi.org/10.1145/1014007.1014025> (visited on 07/13/2022).
- [88] Nathaniel Hudson, Celena Alcock, and Parmit K. Chilana. "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. New York, NY, USA: Association for Computing Machinery, May 2016, pp. 384–396. ISBN: 978-1-4503-3362-7. DOI: [10.1145/2858036.2858266](https://doi.org/10.1145/2858036.2858266). URL: <https://doi.org/10.1145/2858036.2858266> (visited on 11/02/2021).
- [89] Edwin Hutchins, James Hollan, and Donald Norman. "Direct Manipulation Interfaces." In: *Human-computer Interaction* 1 (Dec. 1985), pp. 311–338. DOI: [10.1207/s15327051hcio104_2](https://doi.org/10.1207/s15327051hcio104_2).
- [90] Gina Häußge. *OctoPrint.org*. 2023. URL: <https://octoprint.org/> (visited on 11/23/2023).
- [91] ISO. *ISO/ASTM 52900:2015(en), Additive manufacturing — General principles — Terminology*. 2022. URL: <https://www.iso.org/obp/ui/#iso:std:iso-astm:52900:ed-1:v1:en> (visited on 01/03/2022).
- [92] Peter P. Ikubanni, Adekunle A. Adeleke, Olayinka O. Agboola, Chiebuka T. Christopher, Boluwatife S. Ademola, Joseph Okonkwo, Olanrewaju S. Adesina, Peter O. Omoniyi, and Esther T. Akinlabi. "Present and Future Impacts of Computer-Aided Design/ Computer-Aided Manufacturing (CAD/CAM)." In: *Journal Européen des Systèmes Automatisés* 55.3 (June 2022), pp. 349–357. ISSN: 12696935, 21167087. DOI: [10.18280/jesa.550307](https://doi.org/10.18280/jesa.550307). URL: <https://www.iieta.org/journals/jesa/paper/10.18280/jesa.550307> (visited on 12/28/2023).

- [93] Autodesk Inc. *Fusion 360 | 3D CAD, CAM, CAE, & PCB Cloud-Based Software | Autodesk*. 2023. URL: <https://www.autodesk.com/products/fusion-360/overview> (visited on 08/18/2023).
- [94] BlocksCAD Inc. *BlocksCAD*. 2023. URL: <https://www.blockscad3d.com/> (visited on 07/05/2022).
- [95] Sanchit Ingale, Anirudh Srinivasan, and Diana Bairaktarova. "CAD Platform Independent Software for Automatic Grading of Technical Drawings." In: American Society of Mechanical Engineers Digital Collection, Nov. 2017. DOI: [10.1115/DETC2017-67612](https://doi.org/10.1115/DETC2017-67612). URL: <https://dx.doi.org/10.1115/DETC2017-67612> (visited on 01/07/2024).
- [96] Vitalii Ivanov, Ivan Pavlenko, Oleksandr Liaposhchenko, Oleksandr Gusak, and Vita Pavlenko. "Computer Aided Positioning of Elements of the System "Fixture – Workpiece"." In: Dec. 2018. ISBN: 978-1-63190-167-6. URL: <https://eudl.eu/doi/10.4108/eai.6-11-2018.2279706> (visited on 12/28/2023).
- [97] B. Jackson and D. F. Keefe. "Lift-Off: Using Reference Imagery and Freehand Sketching to Create 3D Models in VR." In: *IEEE Transactions on Visualization and Computer Graphics* 22.4 (Apr. 2016), pp. 1442–1451. ISSN: 1941-0506. DOI: [10.1109/TVCG.2016.2518099](https://doi.org/10.1109/TVCG.2016.2518099).
- [98] Anketa Jandyal, Ikshita Chaturvedi, Ishika Wazir, Ankush Raina, and Mir Irfan Ul Haq. "3D printing – A review of processes, materials and applications in industry 4.0." In: *Sustainable Operations and Computers* 3 (Jan. 2022), pp. 33–42. ISSN: 2666-4127. DOI: [10.1016/j.susoc.2021.09.004](https://doi.org/10.1016/j.susoc.2021.09.004). URL: <https://www.sciencedirect.com/science/article/pii/S2666412721000441> (visited on 09/20/2023).
- [99] Sung-A Jang, Graham Wakefield, and Sung-Hee Lee. "Incorporating Kinesthetic Creativity and Gestural Play into Immersive Modeling." In: *Proceedings of the 4th International Conference on Movement Computing*. MOCO '17. New York, NY, USA: Association for Computing Machinery, June 2017, pp. 1–8. ISBN: 978-1-4503-5209-3. DOI: [10.1145/3077981.3078045](https://doi.org/10.1145/3077981.3078045). URL: <http://doi.org/10.1145/3077981.3078045> (visited on 10/26/2020).
- [100] J. Jankowski and M. Hachet. "Advances in Interaction with 3D Environments." In: *Computer Graphics Forum* 34.1 (2015), pp. 152–190. ISSN: 1467-8659. DOI: [10.1111/cgf.12466](https://doi.org/10.1111/cgf.12466). URL: <https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.12466> (visited on 01/18/2024).
- [101] Shuxu Jing and Qi Yuan. "Consistent naming for sweeping features in replicated collaborative modeling system." In: *2009 IEEE 10th International Conference on Computer-Aided Industrial Design & Conceptual Design*. Nov. 2009, pp. 899–904. DOI: [10.1109/CAIDCD.2009.5375053](https://doi.org/10.1109/CAIDCD.2009.5375053). URL: <https://ieeexplore.ieee.org/document/5375053> (visited on 01/18/2024).

- [102] R. Joan-Arinyo and A. Soto-Riera. "Combining constructive and equational geometric constraint-solving techniques." In: *ACM Transactions on Graphics* 18.1 (1999), pp. 35–55. ISSN: 0730-0301. DOI: [10.1145/300776.300780](https://doi.org/10.1145/300776.300780). URL: <https://dl.acm.org/doi/10.1145/300776.300780> (visited on 09/21/2023).
- [103] Chris Johnson. "Computational Making with Twoville." In: *Journal of Computing Sciences in Colleges* 38.8 (2023), pp. 39–53. ISSN: 1937-4771.
- [104] L. A. Kamentsky and C. N. Liu. "Computer-Automated Design of Multifont Print Recognition Logic." In: *IBM Journal of Research and Development* 7.1 (Jan. 1963), pp. 2–13. ISSN: 0018-8646. DOI: [10.1147/rd.71.0002](https://doi.org/10.1147/rd.71.0002). URL: <https://ieeexplore.ieee.org/document/5392331> (visited on 11/14/2023).
- [105] Petra Kastl, Oliver Krisch, and Ralf Romeike. "3D Printing as Medium for Motivation and Creativity in Computer Science Lessons." In: *Informatics in Schools: Focus on Learning Programming*. Ed. by Valentina Dagienė and Arto Hellas. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2017, pp. 27–36. ISBN: 978-3-319-71483-7. DOI: [10.1007/978-3-319-71483-7_3](https://doi.org/10.1007/978-3-319-71483-7_3).
- [106] Matthew Keeter. *Antimony*. 2023. URL: <https://www.mattkeeter.com/projects/antimony/3/> (visited on 11/18/2023).
- [107] Mary Beth Kery, Donghao Ren, Fred Hohman, Dominik Moritz, Kanit Wongsuphasawat, and Kayur Patel. "mage: Fluid Moves Between Code and Graphical Work in Computational Notebooks." In: *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology*. UIST '20. New York, NY, USA: Association for Computing Machinery, Oct. 2020, pp. 140–151. ISBN: 978-1-4503-7514-6. DOI: [10.1145/3379337.3415842](https://doi.org/10.1145/3379337.3415842). URL: <https://doi.org/10.1145/3379337.3415842> (visited on 09/09/2022).
- [108] Pooya Khaloo, Mehran Maghoumi, Eugene Taranta, David Bettner, and Joseph Laviola. "Code Park: A New 3D Code Visualization Tool." In: *2017 IEEE Working Conference on Software Visualization (VISOFT)*. Sept. 2017, pp. 43–53. DOI: [10.1109/VISOFT.2017.10](https://doi.org/10.1109/VISOFT.2017.10). URL: <https://ieeexplore.ieee.org/document/8091185> (visited on 12/05/2023).
- [109] Hyun Suk Kim, Heedong Ko, and Kunwoo Lee. "Incremental feature-based modeling." In: *Proceedings on the second ACM symposium on Solid modeling and applications*. SMA '93. New York, NY, USA: Association for Computing Machinery, June 1993, pp. 469–470. ISBN: 978-0-89791-584-7. DOI: [10.1145/164360.164525](https://doi.org/10.1145/164360.164525). URL: <https://dl.acm.org/doi/10.1145/164360.164525> (visited on 08/01/2023).
- [110] Jeeun Kim, Anhong Guo, Tom Yeh, Scott E. Hudson, and Jennifer Mankoff. "Understanding Uncertainty in Measurement and Accommodating its Impact in 3D Modeling and Printing." In: *Proceedings of the 2017 Conference on Designing Interactive Systems*. DIS '17. New York, NY, USA: Association for Computing

- ing Machinery, June 2017, pp. 1067–1078. ISBN: 978-1-4503-4922-2. DOI: [10.1145/3064663.3064690](https://doi.org/10.1145/3064663.3064690). URL: <https://doi.org/10.1145/3064663.3064690> (visited on 11/17/2021).
- [111] Amy J. Ko, Brad A. Myers, and Htet Htet Aung. “Six Learning Barriers in End-User Programming Systems.” In: *2004 IEEE Symposium on Visual Languages - Human Centric Computing*. Rome, Italy, Sept. 2004, pp. 199–206. DOI: [10.1109/VLHCC.2004.47](https://doi.org/10.1109/VLHCC.2004.47).
- [112] Siniša Kolarić, Halil Erhan, Robert Woodbury, and Bernhard E. Riecke. “Comprehending parametric CAD models: an evaluation of two graphical user interfaces.” In: *Nordic Conference on Human-Computer Interaction* (Oct. 2010). MAG ID: 2002429313 S2ID: ead43517e8402c84be0e4aec844a3316cefd12f2, pp. 707–710. DOI: [10.1145/1868914.1869010](https://doi.org/10.1145/1868914.1869010).
- [113] Robert Kovacs. “Human-Scale Personal Fabrication.” In: *The Adjunct Publication of the 34th Annual ACM Symposium on User Interface Software and Technology*. UIST ’21. New York, NY, USA: Association for Computing Machinery, Oct. 2021, pp. 162–165. ISBN: 978-1-4503-8655-5. DOI: [10.1145/3474349.3477588](https://doi.org/10.1145/3474349.3477588). URL: <http://doi.org/10.1145/3474349.3477588> (visited on 10/18/2021).
- [114] Maria Kozhevnikov, Michael A. Motes, and Mary Hegarty. “Spatial Visualization in Physics Problem Solving.” In: *Cognitive Science* 31.4 (2007), pp. 549–579. ISSN: 1551-6709. DOI: [10.1080/15326900701399897](https://doi.org/10.1080/15326900701399897). URL: <https://onlinelibrary.wiley.com/doi/abs/10.1080/15326900701399897> (visited on 01/07/2024).
- [115] Fred N. Krull. “The Origin of Computer Graphics within General Motors.” In: *IEEE Annals of the History of Computing* 16.03 (July 1994). Publisher: IEEE Computer Society, pp. 40–56. ISSN: 1058-6180. DOI: [10.1109/MAHC.1994.298419](https://doi.org/10.1109/MAHC.1994.298419). URL: <https://www.computer.org/csdl/magazine/an/1994/03/man1994030040/13rRUwhpBS7> (visited on 12/15/2023).
- [116] Stacey Kuznetsov and Eric Paulos. “Rise of the expert amateur: DIY projects, communities, and cultures.” In: *Proceedings of the 6th Nordic Conference on Human-Computer Interaction: Extending Boundaries*. NordiCHI ’10. New York, NY, USA: Association for Computing Machinery, Oct. 2010, pp. 295–304. ISBN: 978-1-60558-934-3. DOI: [10.1145/1868914.1868950](https://doi.org/10.1145/1868914.1868950). URL: <https://doi.org/10.1145/1868914.1868950> (visited on 11/03/2021).
- [117] Bum chul Kwon, Waqas Javed, Niklas Elmqvist, and Ji Soo Yi. “Direct manipulation through surrogate objects.” In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI ’11. New York, NY, USA: Association for Computing Machinery, May 2011, pp. 627–636. ISBN: 978-1-4503-0228-9. DOI: [10.1145/1978942.1979033](https://doi.org/10.1145/1978942.1979033). URL: <https://doi.org/10.1145/1978942.1979033> (visited on 06/28/2021).

- [118] Meertens Lamberts. "Designing Constraint Maintainers for User Interaction (PS). A methodology for designing constraint maintainers." URL: <https://www.kestrel.edu/people/meertens/> (visited on 07/13/2022).
- [119] J. Richard Landis and Gary G. Koch. "The Measurement of Observer Agreement for Categorical Data." In: *Biometrics* 33.1 (1977). Publisher: [Wiley, International Biometric Society], pp. 159–174. ISSN: 0006-341X. DOI: [10.2307/2529310](https://doi.org/10.2307/2529310). URL: <https://www.jstor.org/stable/2529310> (visited on 07/31/2023).
- [120] Manfred Lau, Masaki Hirose, Akira Ohgawara, Jun Mitani, and Takeo Igarashi. "Situated modeling: a shape-stamping interface with tangible primitives." In: *Proceedings of the Sixth International Conference on Tangible, Embedded and Embodied Interaction*. TEI '12. New York, NY, USA: Association for Computing Machinery, Feb. 2012, pp. 275–282. ISBN: 978-1-4503-1174-8. DOI: [10.1145/2148131.2148190](https://doi.org/10.1145/2148131.2148190). URL: <https://doi.org/10.1145/2148131.2148190> (visited on 10/20/2022).
- [121] Manfred Lau, Akira Ohgawara, Jun Mitani, and Takeo Igarashi. "Converting 3D furniture models to fabricatable parts and connectors." In: *ACM Transactions on Graphics* 30.4 (July 2011), 85:1–85:6. ISSN: 0730-0301. DOI: [10.1145/2010324.1964980](https://doi.org/10.1145/2010324.1964980). URL: <http://doi.org/10.1145/2010324.1964980> (visited on 10/18/2021).
- [122] Manfred Lau, Greg Saul, Jun Mitani, and Takeo Igarashi. "Modeling-in-context: user design of complementary objects with a single photo." In: *Proceedings of the Seventh Sketch-Based Interfaces and Modeling Symposium*. SBIM '10. Goslar, DEU: Eurographics Association, June 2010, pp. 17–24. ISBN: 978-3-905674-25-5. (Visited on 10/20/2022).
- [123] Jerome Laurens. "Direct and reverse synchronization with SyncTeX." In: *Proceedings of the 2008 Annual Meeting*. Dijon: TUGboat, 2008, pp. 365–371. URL: <http://itexmac.sourceforge.net/SyncTeX.html>.
- [124] Ghang Lee, Charles M. Eastman, Tarang Taunk, and Chun-Heng Ho. "Usability principles and best practices for the user interface design of complex 3D architectural design and engineering tools." In: *International Journal of Human-Computer Studies* 68.1-2 (Jan. 2010), pp. 90–104. ISSN: 1071-5819. DOI: [10.1016/j.ijhcs.2009.10.001](https://doi.org/10.1016/j.ijhcs.2009.10.001). URL: <https://doi.org/10.1016/j.ijhcs.2009.10.001> (visited on 11/17/2021).
- [125] Johnny C. Lee, Daniel Avrahami, Scott E. Hudson, Jodi Forlizzi, Paul H. Dietz, and Darren Leigh. "The calder toolkit: wired and wireless components for rapidly prototyping interactive devices." In: *Proceedings of the 5th conference on Designing interactive systems: processes, practices, methods, and techniques*. DIS '04. New York, NY, USA: Association for Computing Machinery, 2004, pp. 167–175. ISBN: 978-1-58113-787-3. DOI: [10.1145/1013115.1013139](https://doi.org/10.1145/1013115.1013139). URL: <https://doi.org/10.1145/1013115.1013139> (visited on 10/25/2022).

- [126] Danny Leen, Raf Ramakers, and Kris Luyten. "StrutModeling: A Low-Fidelity Construction Kit to Iteratively Model, Test, and Adapt 3D Objects." In: *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*. UIST '17. New York, NY, USA: Association for Computing Machinery, Oct. 2017, pp. 471–479. ISBN: 978-1-4503-4981-9. DOI: [10.1145/3126594.3126643](https://doi.org/10.1145/3126594.3126643). URL: <https://doi.org/10.1145/3126594.3126643> (visited on 10/25/2022).
- [127] Sylvain Lefebvre, Salim Perchy, Cédric Zanni, and Pierre Bedell. *IceSL*. 2022. URL: <https://icesl.loria.fr/> (visited on 07/26/2022).
- [128] António Leitão, Luís Santos, and José Lopes. "Programming Languages for Generative Design: A Comparative Study." In: *International Journal of Architectural Computing* 10.1 (Mar. 2012). Publisher: SAGE Publications, pp. 139–162. ISSN: 1478-0771. DOI: [10.1260/1478-0771.10.1.139](https://doi.org/10.1260/1478-0771.10.1.139). URL: <https://doi.org/10.1260/1478-0771.10.1.139> (visited on 01/18/2024).
- [129] Chen Liang, Anhong Guo, and Jeeun Kim. "CustomizAR: Facilitating Interactive Exploration and Measurement of Adaptive 3D Designs." In: *Designing Interactive Systems Conference*. DIS '22. New York, NY, USA: Association for Computing Machinery, 2022, pp. 898–912. ISBN: 978-1-4503-9358-4. DOI: [10.1145/3532106.3533561](https://doi.org/10.1145/3532106.3533561). URL: <https://doi.org/10.1145/3532106.3533561> (visited on 01/25/2023).
- [130] V. C. Lin, D. C. Gossard, and R. A. Light. "Variational geometry in computer-aided design." In: *ACM SIGGRAPH Computer Graphics* 15.3 (1981), pp. 171–177. ISSN: 0097-8930. DOI: [10.1145/965161.806803](https://doi.org/10.1145/965161.806803). URL: <https://dl.acm.org/doi/10.1145/965161.806803> (visited on 09/21/2023).
- [131] Julia Longtin. *ImplicitCad.org*. 2023. URL: <https://www.implicitcad.org/> (visited on 07/05/2021).
- [132] L. Ma, J. Ferguson, M. Roper, and M. Wood. "Investigating and improving the models of programming concepts held by novice programmers." In: *Computer Science Education* 21.1 (Mar. 2011). Publisher: Routledge, pp. 57–80. ISSN: 0899-3408. DOI: [10.1080/08993408.2011.554722](https://doi.org/10.1080/08993408.2011.554722). URL: <https://doi.org/10.1080/08993408.2011.554722> (visited on 01/07/2024).
- [133] I. Scott MacKenzie. "Fitts' Law as a Research and Design Tool in Human-Computer Interaction." In: *Human-Computer Interaction* 7.1 (Mar. 1992). Publisher: Taylor & Francis _eprint: https://doi.org/10.1207/s15327051hcio701_3, pp. 91–139. ISSN: 0737-0024. DOI: [10.1207/s15327051hcio701_3](https://doi.org/10.1207/s15327051hcio701_3). URL: https://doi.org/10.1207/s15327051hcio701_3 (visited on 01/04/2024).
- [134] I. Scott MacKenzie and William Buxton. "Extending Fitts' law to two-dimensional tasks." In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '92. New York, NY, USA: Association for Computing Machinery, June 1992, pp. 219–226. ISBN: 978-0-89791-513-7. DOI: [10.1145/142750.142794](https://doi.org/10.1145/142750.142794). URL: <https://dl.acm.org/doi/10.1145/142750.142794> (visited on 01/04/2024).

- [135] Felipe Machado, Norberto Malpica, and Susana Borromeo. "Parametric CAD modeling for open source scientific hardware: Comparing OpenSCAD and FreeCAD Python scripts." In: *PLOS ONE* 14.12 (May 2019). Publisher: Public Library of Science, e0225795. ISSN: 1932-6203. DOI: [10.1371/journal.pone.0225795](https://doi.org/10.1371/journal.pone.0225795). URL: <https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0225795> (visited on 08/31/2023).
- [136] Chandan Mahapatra, Jonas Kjeldmand Jensen, Michael McQuaid, and Daniel Ashbrook. "Barriers to End-User Designers of Augmented Fabrication." In: *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. Glasgow Scotland Uk: ACM, May 2019, pp. 1–15. ISBN: 978-1-4503-5970-2. DOI: [10.1145/3290605.3300613](https://doi.org/10.1145/3290605.3300613). URL: <https://dl.acm.org/doi/10.1145/3290605.3300613> (visited on 02/05/2021).
- [137] Pierre Martin, Stéphane Masfrand, Yujiro Okuya, and Patrick Bourdot. "A VR-CAD Data Model for Immersive Design." In: *Augmented Reality, Virtual Reality, and Computer Graphics*. Ed. by Lucio Tommaso De Paolis, Patrick Bourdot, and Antonio Mongelli. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2017, pp. 222–241. ISBN: 978-3-319-60922-5. DOI: [10.1007/978-3-319-60922-5_17](https://doi.org/10.1007/978-3-319-60922-5_17).
- [138] Aman Mathur, Marcus Pirron, and Damien Zufferey. "Interactive Programming for Parametric CAD." In: *Computer Graphics Forum* 39.6 (2020), pp. 408–425. ISSN: 1467-8659. DOI: [10.1111/cgf.14046](https://doi.org/10.1111/cgf.14046). URL: <https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.14046> (visited on 09/27/2022).
- [139] Maria Mavri. "Redesigning a Production Chain Based on 3D Printing Technology." In: *Knowledge and Process Management* 22.3 (2015), pp. 141–147. ISSN: 1099-1441. DOI: [10.1002/kpm.1466](https://doi.org/10.1002/kpm.1466). URL: <https://onlinelibrary.wiley.com/doi/abs/10.1002/kpm.1466> (visited on 09/20/2023).
- [140] Kieran May. "UnityRev - Bridging the gap between BIM Authoring platforms and Game Engines by creating a Real-Time Bi-directional Exchange of BIM data." In: *Jeroen van Ameijde, Nicole Gardner, Kyung Hoon Hyun, Dan Luo, Urvi Sheth (eds.), POST-CARBON - Proceedings of the 27th CAADRIA Conference, Sydney, 9-15 April 2022, pp. 527-536. CUMINCAD, 2022. URL: https://papers.cumincad.org/cgi-bin/works/paper/caadria2022_424* (visited on 12/28/2023).
- [141] Nora McDonald, Sarita Schoenebeck, and Andrea Forte. "Reliability and Interrater Reliability in Qualitative Research: Norms and Guidelines for CSCW and HCI Practice." In: *Proceedings of the ACM on Human-Computer Interaction* 3.CSCW (Nov. 2019), 72:1–72:23. DOI: [10.1145/3359174](https://doi.org/10.1145/3359174). URL: <https://dl.acm.org/doi/10.1145/3359174> (visited on 07/31/2023).

- [142] Samantha McDonald, Niara Comrie, Erin Buehler, Nicholas Carter, Braxton Dubin, Karen Gordes, Sandy McCombe-Waller, and Amy Hurst. "Uncovering Challenges and Opportunities for 3D Printing Assistive Technology with Physical Therapists." In: *Proceedings of the 18th International ACM SIGACCESS Conference on Computers and Accessibility*. ASSETS '16. New York, NY, USA: Association for Computing Machinery, Oct. 2016, pp. 131–139. ISBN: 978-1-4503-4124-0. DOI: [10.1145/2982142.2982162](https://doi.org/10.1145/2982142.2982162). URL: <https://doi.org/10.1145/2982142.2982162> (visited on 11/12/2021).
- [143] Michael J. McGuffin and Christopher P. Fuhrman. "Categories and Completeness of Visual Programming and Direct Manipulation." In: *Proceedings of the International Conference on Advanced Visual Interfaces*. AVI '20. New York, NY, USA: Association for Computing Machinery, Sept. 2020, pp. 1–8. ISBN: 978-1-4503-7535-1. DOI: [10.1145/3399715.3399821](https://doi.org/10.1145/3399715.3399821). URL: <https://doi.org/10.1145/3399715.3399821> (visited on 08/09/2021).
- [144] Daniel Mendes, Daniel Medeiros, Maurício Sousa, Ricardo Ferreira, Alberto Raposo, Alfredo Ferreira, and Joaquim Jorge. "Mid-air modeling with Boolean operations in VR." In: *2017 IEEE Symposium on 3D User Interfaces (3DUI)*. Mar. 2017, pp. 154–157. DOI: [10.1109/3DUI.2017.7893332](https://doi.org/10.1109/3DUI.2017.7893332).
- [145] Microsoft. *Visual Studio Code - Code Editing. Redefined.* 2023. URL: <https://code.visualstudio.com/> (visited on 03/20/2023).
- [146] Mark Mine, Arun Yoganandan, and Dane Coffey. "Making VR work: building a real-world immersive modeling application in the virtual world." In: *Proceedings of the 2nd ACM symposium on Spatial user interaction*. SUI '14. New York, NY, USA: Association for Computing Machinery, Oct. 2014, pp. 80–89. ISBN: 978-1-4503-2820-3. DOI: [10.1145/2659766.2659780](https://doi.org/10.1145/2659766.2659780). URL: <https://doi.org/10.1145/2659766.2659780> (visited on 10/20/2022).
- [147] Mark Mine, Arun Yoganandan, and Dane Coffey. "Principles, interactions and devices for real-world immersive modeling." In: *Computers & Graphics* 48 (May 2015), pp. 84–98. ISSN: 0097-8493. DOI: [10.1016/j.cag.2015.02.004](https://doi.org/10.1016/j.cag.2015.02.004). URL: <https://www.sciencedirect.com/science/article/pii/S0097849315000126> (visited on 10/24/2022).
- [148] Michael Molitch Hou. *3D Hubs' April Trend Report is Here! - 3D Printing Industry*. Jan. 2016. URL: <https://3dprintingindustry.com/news/3d-hubs-april-trend-report-is-here-75901/> (visited on 01/10/2024).
- [149] Catarina Mota. "The rise of personal fabrication." In: *Proceedings of the 8th ACM conference on Creativity and cognition*. C&C '11. New York, NY, USA: Association for Computing Machinery, Nov. 2011, pp. 279–288. ISBN: 978-1-4503-0820-5. DOI: [10.1145/2069618.2069665](https://doi.org/10.1145/2069618.2069665). URL: <http://doi.org/10.1145/2069618.2069665> (visited on 10/18/2021).

- [150] Stefanie Mueller, Anna Seufert, Huaishu Peng, Robert Kovacs, Kevin Reuss, François Guimbretière, and Patrick Baudisch. “FormFab: Continuous Interactive Fabrication.” In: *Proceedings of the Thirteenth International Conference on Tangible, Embedded, and Embodied Interaction*. TEI '19. New York, NY, USA: Association for Computing Machinery, Mar. 2019, pp. 315–323. ISBN: 978-1-4503-6196-5. DOI: [10.1145/3294109.3295620](https://doi.org/10.1145/3294109.3295620). URL: <https://doi.org/10.1145/3294109.3295620> (visited on 10/25/2022).
- [151] MyMiniFactory. *MyMiniFactory | Discover STL files for 3D printing ideas and high-quality 3D printer models*. 2022. URL: <https://www.myminifactory.com/> (visited on 02/12/2021).
- [152] M.W. Naing, C.K. Chua, K.F. Leong, and Y. Wang. “Fabrication of customised scaffolds using computer-aided design and rapid prototyping techniques.” In: *Rapid Prototyping Journal* 11.4 (Jan. 2005). Publisher: Emerald Group Publishing Limited, pp. 249–259. ISSN: 1355-2546. DOI: [10.1108/13552540510612938](https://doi.org/10.1108/13552540510612938). URL: <https://doi.org/10.1108/13552540510612938> (visited on 12/28/2023).
- [153] Jakob Nielsen and Rolf Molich. “Heuristic evaluation of user interfaces.” In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '90. New York, NY, USA: Association for Computing Machinery, Mar. 1990, pp. 249–256. ISBN: 978-0-201-50932-8. DOI: [10.1145/97243.97281](https://doi.org/10.1145/97243.97281). URL: <https://doi.org/10.1145/97243.97281> (visited on 01/05/2022).
- [154] Yuenyong Nilsiam and Joshua M. Pearce. “Free and Open Source 3-D Model Customizer for Websites to Democratize Design with OpenSCAD.” In: *Designs* 1.1 (Sept. 2017). Number: 1 Publisher: Multidisciplinary Digital Publishing Institute, p. 5. ISSN: 2411-9660. DOI: [10.3390/designs1010005](https://doi.org/10.3390/designs1010005). URL: <https://www.mdpi.com/2411-9660/1/1/5> (visited on 09/09/2023).
- [155] Mark Noone and Aidan Mooney. “Visual and textual programming languages: a systematic review of the literature.” In: *Journal of Computers in Education* 5.2 (June 2018), pp. 149–174. ISSN: 2197-9995. DOI: [10.1007/s40692-018-0101-5](https://doi.org/10.1007/s40692-018-0101-5). URL: <https://doi.org/10.1007/s40692-018-0101-5> (visited on 01/07/2024).
- [156] Donald Norman. “Cognitive Engineering.” In: *User Centered System Design: New Perspectives on Human-Computer Interaction*. Journal Abbreviation: User Centered System Design: New Perspectives on Human-Computer Interaction. Jan. 1986, pp. 31–61. ISBN: 978-0-367-80732-0. DOI: [10.1201/b15703-3](https://doi.org/10.1201/b15703-3).
- [157] Kent L. Norman and Jurek Kirakowski, eds. *The Wiley Handbook of Human Computer Interaction*. 1st ed. Wiley, Feb. 2018. ISBN: 978-1-118-97613-5 978-1-118-97600-5. DOI: [10.1002/9781118976005](https://doi.org/10.1002/9781118976005). URL: <https://onlinelibrary.wiley.com/doi/book/10.1002/9781118976005> (visited on 01/04/2024).

- [158] Lora Oehlberg, Wesley Willett, and Wendy E. Mackay. "Patterns of Physical Design Remixing in Online Maker Communities." In: *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*. CHI '15. New York, NY, USA: Association for Computing Machinery, Apr. 2015, pp. 639–648. ISBN: 978-1-4503-3145-6. DOI: [10.1145/2702123.2702175](https://doi.org/10.1145/2702123.2702175). URL: <https://doi.org/10.1145/2702123.2702175> (visited on 11/17/2021).
- [159] OpenJSCAD.org. *JSCAD - JavaScript CAD*. 2023. URL: <https://openjscad.xyz/> (visited on 09/06/2023).
- [160] OpenSCAD. *OpenSCAD*. 2020. URL: <http://openscad.org> (visited on 06/30/2021).
- [161] CREO PTC. *Creo CAD Software: Enable the Latest in Design | PTC*. 2023. URL: <https://www.ptc.com/en/products/creo> (visited on 12/28/2023).
- [162] Jeremiah Parry-Hill, Patrick C. Shih, Jennifer Mankoff, and Daniel Ashbrook. "Understanding Volunteer AT Fabricators: Opportunities and Challenges in DIY-AT for Others in e-NABLE." In: *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. CHI '17. New York, NY, USA: Association for Computing Machinery, May 2017, pp. 6184–6194. ISBN: 978-1-4503-4655-9. DOI: [10.1145/3025453.3026045](https://doi.org/10.1145/3025453.3026045). URL: <https://doi.org/10.1145/3025453.3026045> (visited on 11/12/2021).
- [163] Huaishu Peng, Jimmy Briggs, Cheng-Yao Wang, Kevin Guo, Joseph Kider, Stefanie Mueller, Patrick Baudisch, and François Guimbretière. "RoMA: Interactive Fabrication with Augmented Reality and a Robotic 3D Printer." In: *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. CHI '18. New York, NY, USA: Association for Computing Machinery, 2018, pp. 1–12. ISBN: 978-1-4503-5620-6. DOI: [10.1145/3173574.3174153](https://doi.org/10.1145/3173574.3174153). URL: <https://doi.org/10.1145/3173574.3174153> (visited on 10/25/2022).
- [164] Network Protolabs. *3D printing trend report*. 2023. URL: <https://www.hubs.com/get/trends/> (visited on 01/16/2024).
- [165] Yizhou Qian and James Lehman. "Students' Misconceptions and Other Difficulties in Introductory Programming: A Literature Review." In: *ACM Transactions on Computing Education* 18.1 (Oct. 2017), 1:1–1:24. DOI: [10.1145/3077618](https://doi.org/10.1145/3077618). URL: <https://doi.org/10.1145/3077618> (visited on 07/27/2022).
- [166] Raf Ramakers, Fraser Anderson, Tovi Grossman, and George Fitzmaurice. "Retro-Fab: A Design Tool for Retrofitting Physical Interfaces using Actuators, Sensors and 3D Printing." In: *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. CHI '16. New York, NY, USA: Association for Computing Machinery, May 2016, pp. 409–419. ISBN: 978-1-4503-3362-7. DOI: [10.1145/2858036.2858485](https://doi.org/10.1145/2858036.2858485). URL: <https://doi.org/10.1145/2858036.2858485> (visited on 10/25/2022).

- [167] Raf Ramakers, Danny Leen, Jeeun Kim, Kris Luyten, Steven Houben, and Tom Veuskens. "Measurement Patterns: User-Oriented Strategies for Dealing with Measurements and Dimensions in Making Processes." In: *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. CHI '23. New York, NY, USA: Association for Computing Machinery, 2023, pp. 1–17. ISBN: 978-1-4503-9421-5. DOI: [10.1145/3544548.3581157](https://doi.org/10.1145/3544548.3581157). URL: <https://dl.acm.org/doi/10.1145/3544548.3581157> (visited on 12/05/2023).
- [168] Patrick Reipschläger and Raimund Dachsel. "DesignAR: Immersive 3D-Modeling Combining Augmented Reality with Interactive Displays." In: *Proceedings of the 2019 ACM International Conference on Interactive Surfaces and Spaces*. ISS '19. New York, NY, USA: Association for Computing Machinery, Nov. 2019, pp. 29–41. ISBN: 978-1-4503-6891-9. DOI: [10.1145/3343055.3359718](https://doi.org/10.1145/3343055.3359718). URL: <http://doi.org/10.1145/3343055.3359718> (visited on 10/26/2020).
- [169] Aristides G. Requicha. "Representations for Rigid Solids: Theory, Methods, and Systems." In: *ACM Computing Surveys* 12.4 (1980), pp. 437–464. ISSN: 0360-0300. DOI: [10.1145/356827.356833](https://doi.org/10.1145/356827.356833). URL: <https://dl.acm.org/doi/10.1145/356827.356833> (visited on 01/10/2024).
- [170] Alec Rivers, Andrew Adams, and Frédo Durand. "Sculpting by numbers." In: *ACM Transactions on Graphics* 31.6 (Nov. 2012), 157:1–157:7. ISSN: 0730-0301. DOI: [10.1145/2366145.2366176](https://doi.org/10.1145/2366145.2366176). URL: <https://doi.org/10.1145/2366145.2366176> (visited on 11/02/2022).
- [171] Anthony Robins, Janet Rountree, and Nathan Rountree. "Learning and Teaching Programming: A Review and Discussion." In: *Computer Science Education* 13.2 (June 2003). Publisher: Routledge, pp. 137–172. ISSN: 0899-3408. DOI: [10.1076/csed.13.2.137.14200](https://doi.org/10.1076/csed.13.2.137.14200). URL: <https://doi.org/10.1076/csed.13.2.137.14200> (visited on 09/01/2023).
- [172] Bruna Goveia da Rocha, Johannes M. L. van der Kolk, and Kristina Andersen. "Exquisite Fabrication: Exploring Turn-taking between Designers and Digital Fabrication Machines." In: *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. 434. New York, NY, USA: Association for Computing Machinery, May 2021, pp. 1–9. ISBN: 978-1-4503-8096-6. URL: <http://doi.org/10.1145/3411764.3445236> (visited on 10/18/2021).
- [173] José María Rodríguez Corral, Iván Ruíz-Rube, Antón Civit Balcells, José Miguel Mota-Macías, Arturo Morgado-Estévez, and Juan Manuel Dodero. "A Study on the Suitability of Visual Languages for Non-Expert Robot Programmers." In: *IEEE Access* 7 (2019), pp. 17535–17550. ISSN: 2169-3536. DOI: [10.1109/ACCESS.2019.2895913](https://doi.org/10.1109/ACCESS.2019.2895913). URL: <https://ieeexplore.ieee.org/document/8629035> (visited on 01/07/2024).

- [174] D.T. Ross and Massachusetts Institute of Technology. Electronic Systems Laboratory. *Computer-aided design: a statement of objectives*. Technical memorandum 8436. M.I.T. Electronic Systems Laboratory, 1960. URL: <https://books.google.ca/books?id=4IquHAAACAAJ>.
- [175] Majid Rouhani, Miriam Lillebo, Veronica Farshchian, and Monica Divitini. "Learning to Program: an In-service Teachers' Perspective." In: *2022 IEEE Global Engineering Education Conference (EDUCON)*. ISSN: 2165-9567. Mar. 2022, pp. 123–132. DOI: [10.1109/EDUCON52537.2022.9766781](https://doi.org/10.1109/EDUCON52537.2022.9766781). URL: <https://ieeexplore.ieee.org/document/9766781> (visited on 01/07/2024).
- [176] Daisuke Saito, Hironori Washizaki, and Yoshiaki Fukazawa. "Comparison of Text-Based and Visual-Based Programming Input Methods for First-Time Learners." In: *Journal of Information Technology Education: Research* 16 (June 2017), pp. 209–226. URL: <https://www.informingscience.org/Publications/3775> (visited on 01/07/2024).
- [177] Jose Luis Saorín, Vicente Lopez-Chao, Jorge de la Torre-Cantero, and Manuel Drago Díaz-Alemán. "Computer Aided Design to Produce High-Detail Models through Low Cost Digital Fabrication for the Conservation of Aerospace Heritage." In: *Applied Sciences* 9.11 (Jan. 2019). Number: 11 Publisher: Multidisciplinary Digital Publishing Institute, p. 2338. ISSN: 2076-3417. DOI: [10.3390/app9112338](https://doi.org/10.3390/app9112338). URL: <https://www.mdpi.com/2076-3417/9/11/2338> (visited on 12/28/2023).
- [178] Naoki Sasaki, Hsiang-Ting Chen, Daisuke Sakamoto, and Takeo Igarashi. "Face-tons: face primitives with adaptive bounds for building 3D architectural models in virtual environment." In: *Proceedings of the 19th ACM Symposium on Virtual Reality Software and Technology*. VRST '13. New York, NY, USA: Association for Computing Machinery, Oct. 2013, pp. 77–82. ISBN: 978-1-4503-2379-6. DOI: [10.1145/2503713.2503718](https://doi.org/10.1145/2503713.2503718). URL: <https://doi.org/10.1145/2503713.2503718> (visited on 10/20/2022).
- [179] Valkyrie Savage, Sean Follmer, Jingyi Li, and Björn Hartmann. "Makers' Marks: Physical Markup for Designing and Fabricating Functional Objects." In: *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*. UIST '15. New York, NY, USA: Association for Computing Machinery, Nov. 2015, pp. 103–108. ISBN: 978-1-4503-3779-3. DOI: [10.1145/2807442.2807508](https://doi.org/10.1145/2807442.2807508). URL: <https://doi.org/10.1145/2807442.2807508> (visited on 10/25/2022).
- [180] Ryan Schmidt, Karan Singh, and Ravin Balakrishnan. "Sketching and Composing Widgets for 3D Manipulation." In: *Computer Graphics Forum* 27.2 (2008), pp. 301–310. ISSN: 1467-8659. DOI: [10.1111/j.1467-8659.2008.01127.x](https://doi.org/10.1111/j.1467-8659.2008.01127.x). URL: <https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8659.2008.01127.x> (visited on 01/18/2024).

- [181] Udo Schultheis, Jason Jerald, Fernando Toledo, Arun Yoganandan, and Paul Mlyniec. "Comparison of a two-handed interface to a wand interface and a mouse interface for fundamental 3D tasks." In: *2012 IEEE Symposium on 3D User Interfaces (3DUI)*. Mar. 2012, pp. 117–124. DOI: [10.1109/3DUI.2012.6184195](https://doi.org/10.1109/3DUI.2012.6184195).
- [182] Markus Schütz and Michael Wimmer. "Live Coding of a VR Render Engine in VR." In: *2019 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*. ISSN: 2642-5254. Mar. 2019, pp. 1150–1151. DOI: [10.1109/VR.2019.8797760](https://doi.org/10.1109/VR.2019.8797760). URL: <https://ieeexplore.ieee.org/document/8797760> (visited on 12/05/2023).
- [183] Víctor Stefano Segura Castillo, Leonel Merino, Geoffrey Hecht, and Alexandre Bergel. "VR-Based User Interactions to Exploit Infinite Space in Programming Activities." In: *2021 40th International Conference of the Chilean Computer Science Society (SCCC)*. ISSN: 2691-0632. Nov. 2021, pp. 1–5. DOI: [10.1109/SCCC54552.2021.9650396](https://doi.org/10.1109/SCCC54552.2021.9650396). URL: <https://ieeexplore.ieee.org/document/9650396> (visited on 12/05/2023).
- [184] Jami J. Shah. "Designing with Parametric CAD: Classification and comparison of construction techniques." In: *Geometric Modelling: Theoretical and Computational Basis towards Advanced CAD Applications. IFIP TC5/WG5.2 Sixth International Workshop on Geometric Modelling December 7–9, 1998, Tokyo, Japan*. Ed. by Fumihiko Kimura. IFIP — The International Federation for Information Processing. Boston, MA: Springer US, 2001, pp. 53–68. ISBN: 978-0-387-35490-3. DOI: [10.1007/978-0-387-35490-3_4](https://doi.org/10.1007/978-0-387-35490-3_4). URL: https://doi.org/10.1007/978-0-387-35490-3_4 (visited on 01/10/2024).
- [185] Fereshteh Shahmiri and Paul H. Dietz. "ShArc: A Geometric Technique for Multi-Bend / Shape Sensing." In: *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. CHI '20. New York, NY, USA: Association for Computing Machinery, Apr. 2020, pp. 1–12. ISBN: 978-1-4503-6708-0. DOI: [10.1145/3313831.3376269](https://doi.org/10.1145/3313831.3376269). URL: <http://doi.org/10.1145/3313831.3376269> (visited on 10/01/2020).
- [186] N. Shahrubudin, T. C. Lee, and R. Ramlan. "An Overview on 3D Printing Technology: Technological, Materials, and Applications." In: *Procedia Manufacturing*. The 2nd International Conference on Sustainable Materials Processing and Manufacturing, SMPM 2019, 8-10 March 2019, Sun City, South Africa 35 (Jan. 2019), pp. 1286–1296. ISSN: 2351-9789. DOI: [10.1016/j.promfg.2019.06.089](https://doi.org/10.1016/j.promfg.2019.06.089). URL: <https://www.sciencedirect.com/science/article/pii/S2351978919308169> (visited on 01/03/2022).
- [187] Irv Shapiro. *MakeWithTech Blog*. 2023. URL: <https://www.makewithtech.com/> (visited on 08/30/2023).
- [188] Jia Sheng, Ravin Balakrishnan, and Karan Singh. "An interface for virtual 3D sculpting via physical proxy." In: *Proceedings of the 4th international conference on Computer graphics and interactive techniques in Australasia and Southeast Asia*. GRAPHITE '06. New York, NY, USA: Association for Computing Machinery,

- Nov. 2006, pp. 213–220. ISBN: 978-1-59593-564-9. DOI: [10.1145/1174429.1174467](https://doi.org/10.1145/1174429.1174467). URL: <https://doi.org/10.1145/1174429.1174467> (visited on 10/20/2022).
- [189] Samyukta Sherugar and Raluca Budiu. *Direct Manipulation: Definition*. 2016. URL: <https://www.nngroup.com/articles/direct-manipulation/> (visited on 01/05/2022).
- [190] Rita Shewbridge, Amy Hurst, and Shaun K. Kane. “Everyday making: identifying future uses for 3D printing in the home.” In: *Proceedings of the 2014 conference on Designing interactive systems*. DIS ’14. New York, NY, USA: Association for Computing Machinery, June 2014, pp. 815–824. ISBN: 978-1-4503-2902-6. DOI: [10.1145/2598510.2598544](https://doi.org/10.1145/2598510.2598544). URL: <https://doi.org/10.1145/2598510.2598544> (visited on 11/04/2021).
- [191] Ben Shneiderman. “The future of interactive systems and the emergence of direct manipulation.” In: *Behaviour & Information Technology* 1.3 (July 1982). Publisher: Taylor & Francis _eprint: <https://doi.org/10.1080/01449298208914450>, pp. 237–256. ISSN: 0144-929X. DOI: [10.1080/01449298208914450](https://doi.org/10.1080/01449298208914450). URL: <https://doi.org/10.1080/01449298208914450> (visited on 08/09/2021).
- [192] Ben Shneiderman. “Direct Manipulation: A Step Beyond Programming Languages.” In: *Computer* 16.8 (1983), pp. 57–69. ISSN: 1558-0814. DOI: [10.1109/MC.1983.1654471](https://doi.org/10.1109/MC.1983.1654471).
- [193] Ben Shneiderman. “Direct manipulation for comprehensible, predictable and controllable user interfaces.” In: *Proceedings of the 2nd international conference on Intelligent user interfaces*. IUI ’97. New York, NY, USA: Association for Computing Machinery, Jan. 1997, pp. 33–39. ISBN: 978-0-89791-839-8. DOI: [10.1145/238218.238281](https://doi.org/10.1145/238218.238281). URL: <https://doi.org/10.1145/238218.238281> (visited on 08/11/2021).
- [194] Ben Shneiderman et al. “Creativity Support Tools: Report From a U.S. National Science Foundation Sponsored Workshop.” In: *International Journal of Human–Computer Interaction* 20.2 (May 2006). Publisher: Taylor & Francis, pp. 61–77. ISSN: 1044-7318. DOI: [10.1207/s15327590ijhc2002_1](https://doi.org/10.1207/s15327590ijhc2002_1). URL: https://doi.org/10.1207/s15327590ijhc2002_1 (visited on 10/24/2022).
- [195] Digital Industries Software Siemens. *NX software* | *Siemens Software*. 2023. URL: <https://plm.sw.siemens.com/en-US/nx/> (visited on 12/28/2023).
- [196] Brendan M. Sleight. *laser-cut.scad*. original-date: 2015-08-26T20:13:41Z. Nov. 2023. URL: <https://github.com/bmsleight/laser-cut> (visited on 11/23/2023).
- [197] Joanna Smith and Jill Firth. “Qualitative data analysis: the framework approach.” In: *Nurse Researcher* 18.2 (Jan. 2011), pp. 52–62. ISSN: 1351-5578, 2047-8992. DOI: [10.7748/nr2011.01.18.2.52.c8284](https://doi.org/10.7748/nr2011.01.18.2.52.c8284). URL: <http://rcnpublishing.com/doi/abs/10.7748/nr2011.01.18.2.52.c8284> (visited on 09/10/2023).

- [198] Eun-Sung Song, Young-Jun Lim, and Bongju Kim. "A User-Specific Approach for Comfortable Application of Advanced 3D CAD/CAM Technique in Dental Environments Using the Harmonic Series Noise Model." In: *Applied Sciences* 9.20 (Jan. 2019). Number: 20 Publisher: Multidisciplinary Digital Publishing Institute, p. 4307. ISSN: 2076-3417. DOI: [10.3390/app9204307](https://doi.org/10.3390/app9204307). URL: <https://www.mdpi.com/2076-3417/9/20/4307> (visited on 12/28/2023).
- [199] Hyunyoung Song, François Guimbretière, and Hod Lipson. "The ModelCraft framework: Capturing freehand annotations and edits to facilitate the 3D model design process using a digital pen." In: *ACM Transactions on Computer-Human Interaction* 16.3 (Sept. 2009), 14:1–14:33. ISSN: 1073-0516. DOI: [10.1145 / 1592440.1592443](https://doi.org/10.1145/1592440.1592443). URL: <https://doi.org/10.1145/1592440.1592443> (visited on 10/25/2022).
- [200] Konstantinos Spiliotopoulos, Maria Rigou, and Spiros Sirmakessis. "A Comparative Study of Skeuomorphic and Flat Design from a UX Perspective." In: *Multimodal Technologies and Interaction* 2.2 (June 2018), p. 31. ISSN: 2414-4088. DOI: [10.3390/mti2020031](https://doi.org/10.3390/mti2020031). URL: <http://www.mdpi.com/2414-4088/2/2/31> (visited on 01/02/2024).
- [201] David Squires. "The use of direct manipulation in educational software design." In: *World Conference on Computers in Education VI: WCCE '95 Liberating the Learner, Proceedings of the sixth IFIP World Conference on Computers in Education, 1995*. Ed. by J. David Tinsley and Tom J. van Weert. IFIP — The International Federation for Information Processing. Boston, MA: Springer US, 1995, pp. 273–282. ISBN: 978-0-387-34844-5. DOI: [10.1007/978-0-387-34844-5_28](https://doi.org/10.1007/978-0-387-34844-5_28). URL: https://doi.org/10.1007/978-0-387-34844-5_28 (visited on 11/30/2023).
- [202] Statista. *Global computer aided design (CAD) market 2028*. 2023. URL: <https://www.statista.com/statistics/789999/worldwide-computer-aided-design-market/> (visited on 11/21/2023).
- [203] A. Steed. "Towards a General Model for Selection in Virtual Environments." In: *3D User Interfaces (3DUI'06)*. Mar. 2006, pp. 103–110. DOI: [10.1109/VR.2006.134](https://doi.org/10.1109/VR.2006.134). URL: <https://ieeexplore.ieee.org/abstract/document/1647515> (visited on 01/18/2024).
- [204] Evgeny Stemasov, Tobias Wagner, Jan Gugenheimer, and Enrico Rukzio. "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. New York, NY, USA: Association for Computing Machinery, 2020, pp. 1–12. ISBN: 978-1-4503-6708-0. DOI: [10.1145/3313831.3376839](https://doi.org/10.1145/3313831.3376839). URL: <https://dl.acm.org/doi/10.1145/3313831.3376839> (visited on 07/24/2023).
- [205] David Stutz. *A Formal Definition of Watertight Meshes* • David Stutz. Jan. 2018. URL: <https://davidstutz.de/a-formal-definition-of-watertight-meshes/> (visited on 09/12/2023).

- [206] Ivan E. Sutherland. "Sketch pad a man-machine graphical communication system." In: *Proceedings of the SHARE design automation workshop*. DAC '64. New York, NY, USA: Association for Computing Machinery, 1964, pp. 6.329–6.346. ISBN: 978-1-4503-7932-8. DOI: [10.1145/800265.810742](https://doi.org/10.1145/800265.810742). URL: <https://dl.acm.org/doi/10.1145/800265.810742> (visited on 09/21/2023).
- [207] Cadence Design Systems. *OrCAD | PCB Design Software and Schematic Editor*. 2023. URL: <https://www.orcad.com/> (visited on 11/16/2023).
- [208] Jeff K. T. Tang, Tin-Yung Au Duong, Yui-Wang Ng, and Hoi-Kit Luk. "Learning to create 3D models via an augmented reality smartphone interface." In: *2015 IEEE International Conference on Teaching, Assessment, and Learning for Engineering (TALE)*. Dec. 2015, pp. 236–241. DOI: [10.1109/TALE.2015.7386050](https://doi.org/10.1109/TALE.2015.7386050).
- [209] Nagendra Gautam Tanikella, Benjamin Savonen, John Gershenson, and Joshua Pearce. "Viability of distributed manufacturing of bicycle components with 3-D printing: CEN standardized polylactic acid pedal testing." In: *Journal of Humanitarian Engineering* (Jan. 2016). URL: https://digitalcommons.mtu.edu/materials_fp/71.
- [210] The FreeCAD Team. *FreeCAD: Your own 3D parametric modeler*. 2022. URL: <https://www.freecadweb.org/> (visited on 06/30/2021).
- [211] Unity Technologies. *Unity*. 2023. URL: <https://unity.com/> (visited on 08/16/2023).
- [212] Thingiverse.com. *Customizer by MakerBot on Thingiverse - Thingiverse*. 2022. URL: <https://www.thingiverse.com/app:22> (visited on 09/08/2022).
- [213] Thingiverse.com. *Thingiverse - Digital Designs for Physical Objects*. 2022. URL: <https://www.thingiverse.com/> (visited on 02/12/2021).
- [214] Mary Kathryn Thompson et al. "Design for Additive Manufacturing: Trends, opportunities, considerations, and constraints." In: *CIRP Annals* 65.2 (Jan. 2016), pp. 737–760. ISSN: 0007-8506. DOI: [10.1016/j.cirp.2016.05.004](https://doi.org/10.1016/j.cirp.2016.05.004). URL: <https://www.sciencedirect.com/science/article/pii/S0007850616301913> (visited on 09/20/2023).
- [215] S. N. Trika, P Banerjee, and R. L. Kashyap. "Virtual reality interfaces for feature-based computer-aided design systems." In: *Computer-Aided Design*. Virtual Reality 29.8 (Aug. 1997), pp. 565–574. ISSN: 0010-4485. DOI: [10.1016/S0010-4485\(96\)00092-9](https://doi.org/10.1016/S0010-4485(96)00092-9). URL: <https://www.sciencedirect.com/science/article/pii/S0010448596000929> (visited on 10/20/2022).
- [216] Inc Trimble. *3D Design Software | 3D Modeling on the Web | SketchUp*. 2023. URL: <https://www.sketchup.com/> (visited on 12/28/2023).
- [217] Hannah Twigg-Smith, Jasper Tran O'Leary, and Nadya Peek. "Tools, Tricks, and Hacks: Exploring Novel Digital Fabrication Workflows on #PlotterTwitter." In: *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. CHI '21. New York, NY, USA: Association for Computing Machinery, May 2021,

- pp. 1–15. ISBN: 978-1-4503-8096-6. DOI: [10.1145/3411764.3445653](https://doi.org/10.1145/3411764.3445653). URL: <https://dl.acm.org/doi/10.1145/3411764.3445653> (visited on 07/24/2023).
- [218] Tom Veuskens, Florian Heller, and Raf Ramakers. “CODA: A Design Assistant to Facilitate Specifying Constraints and Parametric Behavior in CAD Models.” In: (2021), 10 pages, 877. ISSN: 0713-5424. DOI: [10.20380/GI2021.11](https://doi.org/10.20380/GI2021.11). URL: <https://graphicsinterface.org/proceedings/gi2021/gi2021-11/> (visited on 04/03/2024).
- [219] Bret Victor. *Bret Victor - The Future of Programming*. July 2013. URL: <https://vimeo.com/71278954> (visited on 09/09/2022).
- [220] Bret Victor. *Drawing Dynamic Visualizations*. May 2013. URL: <https://vimeo.com/66085662> (visited on 09/14/2022).
- [221] L.R. Wanger, J.A. Ferwerda, and D.P. Greenberg. “Perceiving spatial relationships in computer-generated images.” In: *IEEE Computer Graphics and Applications* 12.3 (May 1992), pp. 44–58. ISSN: 1558-1756. DOI: [10.1109/38.135913](https://doi.org/10.1109/38.135913). URL: <https://ieeexplore.ieee.org/document/135913> (visited on 01/18/2024).
- [222] Christian Weichel, Jason Alexander, Abhijit Karnik, and Hans Gellersen. “SPATA: Spatio-Tangible Tools for Fabrication-Aware Design.” In: *Proceedings of the Ninth International Conference on Tangible, Embedded, and Embodied Interaction*. TEI ’15. New York, NY, USA: Association for Computing Machinery, Jan. 2015, pp. 189–196. ISBN: 978-1-4503-3305-4. DOI: [10.1145/2677199.2680576](https://doi.org/10.1145/2677199.2680576). URL: <https://doi.org/10.1145/2677199.2680576> (visited on 02/12/2021).
- [223] Christian Weichel, John Hardy, Jason Alexander, and Hans Gellersen. “ReForm: Integrating Physical and Digital Design through Bidirectional Fabrication.” In: *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*. UIST ’15. New York, NY, USA: Association for Computing Machinery, Nov. 2015, pp. 93–102. ISBN: 978-1-4503-3779-3. DOI: [10.1145/2807442.2807451](https://doi.org/10.1145/2807442.2807451). URL: <https://doi.org/10.1145/2807442.2807451> (visited on 10/25/2022).
- [224] Christian Weichel, Manfred Lau, and Hans Gellersen. “Enclosed: a component-centric interface for designing prototype enclosures.” In: *Proceedings of the 7th International Conference on Tangible, Embedded and Embodied Interaction*. TEI ’13. New York, NY, USA: Association for Computing Machinery, Feb. 2013, pp. 215–218. ISBN: 978-1-4503-1898-3. DOI: [10.1145/2460625.2460659](https://doi.org/10.1145/2460625.2460659). URL: <https://doi.org/10.1145/2460625.2460659> (visited on 10/20/2022).
- [225] Christian Weichel, Manfred Lau, David Kim, Nicolas Villar, and Hans W. Gellersen. “MixFab: a mixed-reality environment for personal fabrication.” In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI ’14. New York, NY, USA: Association for Computing Machinery, 2014, pp. 3855–3864. ISBN: 978-1-4503-2473-1. DOI: [10.1145/2556288.2557090](https://doi.org/10.1145/2556288.2557090). URL: <https://doi.org/10.1145/2556288.2557090> (visited on 10/20/2022).

- [226] Catherine G. Wolf and James R. Rhyne. "A Taxonomic Approach to Understanding Direct Manipulation." In: *Proceedings of the Human Factors Society Annual Meeting* 31.5 (Sept. 1987). Publisher: SAGE Publications, pp. 576–580. ISSN: 0163-5182. DOI: [10.1177/154193128703100522](https://doi.org/10.1177/154193128703100522). URL: <https://doi.org/10.1177/154193128703100522> (visited on 11/30/2023).
- [227] Belle Selene Xia. "A Pedagogical Review of Programming Education Research: What Have We Learned." In: *International Journal of Online Pedagogy and Course Design (IJOPCD)* 7.1 (2017). ISBN: 9782017010104 Publisher: IGI Global, pp. 33–42. ISSN: 2155-6873. DOI: [10.4018/IJOPCD.2017010103](https://www.igi-global.com/gateway/article/www.igi-global.com/gateway/article/164972). URL: <https://www.igi-global.com/gateway/article/www.igi-global.com/gateway/article/164972> (visited on 01/07/2024).
- [228] Eldad Yechiam, Ido Erev, and Avi Parush. "Easy First Steps and Their Implication to the Use of a Mouse-Based and a Script-Based Strategy." In: *Journal of Experimental Psychology: Applied* 10.2 (2004). Place: US Publisher: American Psychological Association, pp. 89–96. ISSN: 1939-2192. DOI: [10.1037/1076-898X.10.2.89](https://doi.org/10.1037/1076-898X.10.2.89).
- [229] Tom Yeh and Jeeun Kim. "CraftML: 3D Modeling is Web Programming." In: *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. Montreal QC Canada: ACM, Apr. 2018, pp. 1–12. ISBN: 978-1-4503-5620-6. DOI: [10.1145/3173574.3174101](https://dl.acm.org/doi/10.1145/3173574.3174101). URL: <https://dl.acm.org/doi/10.1145/3173574.3174101> (visited on 06/22/2022).
- [230] Xiaoqiang Zhu, Lei Song, Lihua You, Mengyao Zhu, Xiangyang Wang, and Xiaogang Jin. "Brush2Model: Convolution surface-based brushes for 3D modelling in head-mounted display-based virtual environments." In: *Computer Animation and Virtual Worlds* 28.3-4 (2017), e1764. ISSN: 1546-427X. DOI: [10.1002/cav.1764](https://onlinelibrary.wiley.com/doi/abs/10.1002/cav.1764). URL: <https://onlinelibrary.wiley.com/doi/abs/10.1002/cav.1764> (visited on 10/20/2022).
- [231] Amit Zoran and Joseph A. Paradiso. "FreeD: a freehand digital sculpting tool." In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '13. New York, NY, USA: Association for Computing Machinery, 2013, pp. 2613–2616. ISBN: 978-1-4503-1899-0. DOI: [10.1145/2470654.2481361](https://doi.org/10.1145/2470654.2481361). URL: <https://doi.org/10.1145/2470654.2481361> (visited on 11/02/2022).
- [232] Amit Zoran, Roy Shilkrot, and Joseph Paradiso. "Human-computer interaction for hybrid carving." In: *Proceedings of the 26th annual ACM symposium on User interface software and technology*. UIST '13. New York, NY, USA: Association for Computing Machinery, Oct. 2013, pp. 433–440. ISBN: 978-1-4503-2268-3. DOI: [10.1145/2501988.2502023](https://doi.org/10.1145/2501988.2502023). URL: <https://doi.org/10.1145/2501988.2502023> (visited on 11/02/2022).
- [233] Qiang Zou. *Parametric/direct CAD integration*. Mar. 2022. DOI: [10.48550/arXiv.2203.02252](https://arxiv.org/abs/2203.02252). URL: [http://arxiv.org/abs/2203.02252](https://arxiv.org/abs/2203.02252) (visited on 08/01/2023).

- [234] Qiang Zou, Zhihong Tang, Hsi-Yung Feng, Shuming Gao, Chenchu Zhou, and Yusheng Liu. *A review on geometric constraint solving*. Aug. 2022. DOI: [10.48550/arXiv.2202.13795](https://doi.org/10.48550/arXiv.2202.13795). URL: <http://arxiv.org/abs/2202.13795> (visited on 09/20/2023).
- [235] Prusa Research a.s. *Base de datos de modelos 3D*. 2023. URL: <https://www.printables.com> (visited on 09/10/2023).
- [236] D. Åkesson and Caitlin Mueller. "Using 3D direct manipulation for real-time structural design exploration." In: *Computer-Aided Design and Applications* 15.1 (Jan. 2018). Publisher: Taylor & Francis, pp. 1–10. ISSN: null. DOI: [10.1080/16864360.2017.1355087](https://doi.org/10.1080/16864360.2017.1355087). URL: <https://doi.org/10.1080/16864360.2017.1355087> (visited on 07/05/2021).
- [237] Paweł Ślusarczyk. *Ranking of CAD systems in the world according to Reddit*. May 2016. URL: <https://3dprintingcenter.net/ranking-of-cad-systems-in-the-world-according-to-reddit/> (visited on 12/04/2023).