FabCode: Visual Programming Environment for Digital Fabrication

Harshit Agrawal
Indian Institute of Technology Guwahati
Department of Design
harshit.rnnh@gmail.com

Rishika Jain
Indian Institute of Technology Guwahati
Department of Design
rishika.j92@gmail.com

Prabhat Humar
Indian Institute of Technology Guwahati
Department of Design
prabhat1992@gmail.com

Pradeep Yammiyavar
Indian Institute of Technology Guwahati
Department of Design
pradeep@iitg.ernet.in

ABSTRACT
In this paper, we introduce FabCode, a visual programming environment using which one can create designs that can be manufactured using digital fabrication techniques like 3D printing and laser cutting. This project is primarily about making accessible and enhancing the kinds of 'thinking' that the computational medium is capable of supporting and spreading. FabCode is situated in the context of design and engineering of objects, and is based on the premise that programming 3D models for personal fabrication would enable practice of computational thinking for the same. Children will learn as they work on personally meaningful projects—building, describing, printing and playing with things, and debugging and discussing their processes and outcomes. It will be a child-centered, constructionist tool for FabLabs.

Categories and Subject Descriptors
D.3.3 [**Programming Languages**]: Language Constructs and Features

Keywords
Constructionism, Digital Fabrication, Computational Thinking, 3D Modeling, Visual Programming, Blockly

1. INTRUCTION
Digital fabrication is inherently an iterative process based on the established iterative cycle of creative learning [1]. Digital fabrication and "making" have been argued to be a new and major chapter in this process of bringing powerful ideas, literacies, and expressive tools to children. What Logo did for geometry and programming – bringing complex mathematics within the reach of schoolchildren – fabrication labs can do for design and engineering [2]. Computational design and digital fabrication offer many compelling opportunities for personal creative expression through programming [6].

IDC'14, June 17–20, 2014, Aarhus, Denmark.

http://dx.doi.org/10.1145/2593968.2610490

FabCode could serve as a 'Mathland' [3] inspired by LOGO, because through programming designs of objects, one can let math do the making. All objects can then possibly serve as objects-to-think-with [4] while playing with them, and in the process of making them by considering aspects such as their mechanical functions, needs fulfilled by them, and attributes.

Computational Thinking involves defining, understanding, and solving problems, reasoning at multiple levels of abstraction, understanding and applying automation, and analyzing the appropriateness of the abstractions made [5]. Computational thinking shares with engineering thinking in the general ways in which we might approach designing and evaluating a large, complex system that operates within the constraints of the real world. Computational design offers a number of benefits that can extend traditional design techniques. These benefits include: precision and automation, generativity and randomness, parameterization [6].

Papert has explained his pedagogical philosophy thus: "constructionism boils down to demanding that everything be understood by being constructed" [7]. With objects, the starting points are concrete and grounded. FabCode can support thinking in 'mind sized bites' [3], where programmers start with one specific case, entirely understood, develop intuition and then gradually generalize, level by level, in a way that they still fully understand the program at each level of abstraction. Deconstruction of objects and associated notions may lead to breaking of black-boxes, followed by investigation and reconstruction or re-invention of tangible objects through new methods and techniques. In this manner, our platform could be a realization of the computational medium acting as a material-to-think- with.

Also, Piaget and other developmental psychologists have emphasized the importance of using physical objects for children's cognitive development, which is a major part of the iterative cycle of digital fabrication [8, 9]. The benefits of digital fabrication and FabLabs for children have been well established through research projects like FabLab@School [10].

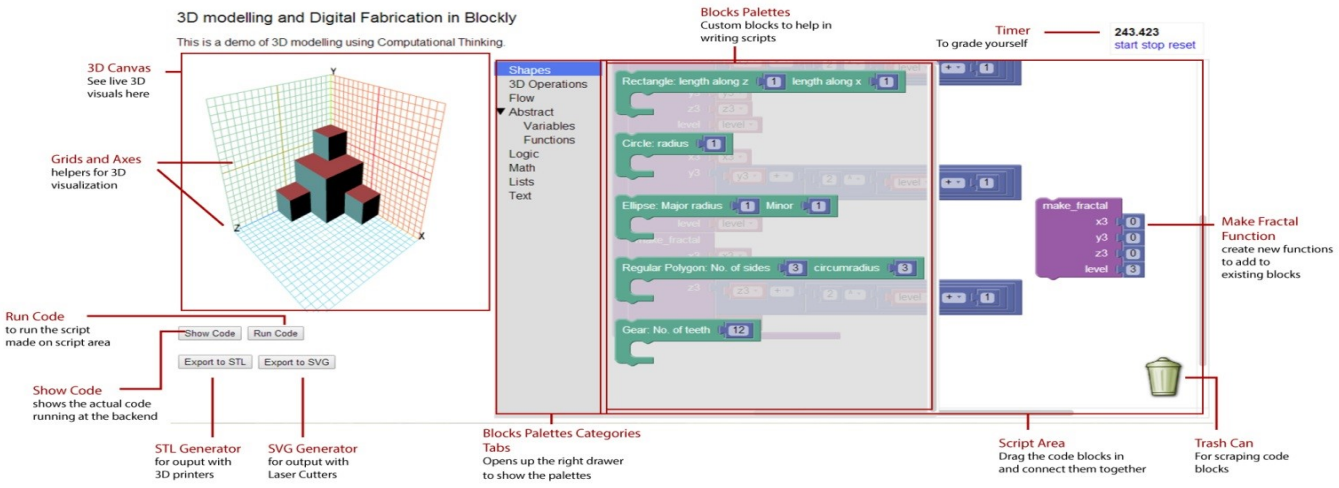

Fig 1. FabCode interface with labeling of different parts. A fractal is constructed here combining the blocks of FabCode

Programming tools such as Scratch [11] and NetLogo [12] have been instrumental and fundamental in encouraging children to think and create new things computationally. Scratch, linked learning of programming concepts and computational thinking skills to a personally meaningful output that kids could create in the form of animations, games and other kind of media. Correspondingly, FabCode aims to bring forth the computer as a new material for learning [13] and making by situating digital

literacy in the context of creation of physical objects through 3D digital design and fabrication.

2. Existing Software

We studied existing software in two domains:

- Modeling software
- Software that combine computational design and digital fabrication

The software currently available for creating models for digital fabrication can be roughly put in two categories: tools for beginners with a low threshold and tools for professional users with a high ceiling [14]. Tools for amateur users include Google SketchUp [15], Tinkercad[16], and Autodesk 123D Design [17]. 3D objects are created by combining and modifying basic shapes, such as cubes, spheres, cylinders, and planes. The functionality to modify the shapes by altering the underlying geometrical figures is limited, thereby limiting understanding of the link between 2D shapes and 3D ones. Sketch It, Make It is a 2D CAD tool that allows users to constrain their designs through gestures made using a digital drawing tablet [18]. These software have their limitations in bringing clarity to how 3D shapes are constructed from 2 D shapes. They are also limited in the detail of design that can be modeled. The tools of these software do not have an inherent aim of linking computational thinking and programing constructs with designing of models.

On the other hand, there are 3D modeling tools for professional 3D modeling like SolidWorks [19], Rhinoceros [20] or Autodesk Inventor [21]. They allow for developing sophisticated 3D objects, but seem to be less suitable for beginners. There are also professional tools that are explicitly developed for computational design like Grasshopper, a third-party add-on for the Rhinoceros 3D modeling tool [22]. In the context of digital fabrication, one of the most important elements of these tools is their ability to import and export a wide variety of file formats, thus facilitating the transitions between a digital design and the required file type for a specific fabrication tool. Despite their power, and due to their high cost and complex feature set, these professional tools are extremely difficult for amateurs to access and use. Successfully using them in a FabLab context requires prior knowledge.

Platforms with the aim of making computational designing (programming driven) for digital fabrication to be accessible to novice users have been built. FlatCAD seeks to connect programming and digital fabrication and allows users to build customized construction kits with a laser cutter by programming in FlatLang, a novice-oriented programming language modeled on Logo [23]. Codeable Objects allows for novice programers to create design for fabrication using the laser cutter and the vinyl cutter [6]. Both these support 2D designs and only. They could be put together for constructing 3D structures, but these platforms do not provide for modeling shapes in 3D directly, thereby having limited capability. The existing environments, especially the ones aimed at beginners, do not have an inherent intention of encouraging users to think and work computationally to create the designs that they want to. There has been little concrete attempt at creating a scalable platform that links the two skills- that of programing with design and engineering.

3. FABCODE

FabCode is an online platform where users can use visual programming language to create models for digital fabrication. FabCode derives its design principles from the guiding principles for designing construction kits as laid out by Resnick et al [24]. These include: learning through designing, having low floor and wide walls and supporting many paths for designing the same object. FabCode, being an online environment will also make it simple for users to share their designs that are personally meaningful to them, a component of effective learning as proposed in the theory of Constructionism.

3.1 Features of current interface

3.1.1 Single-Window User Interface

A single window multi pane design ensures that key components are always visible, and interface navigation is eased. To invite coding, the command palette is always visible, wherein the

commands are divided into categories, and their ordering and color coding helps keep the different commands visible, comprehensible and accessible [25]. The render area in FabCode can be rotated, zoomed in and out, panned across for viewing different views. A timer is also provided to time how long it takes you to create something if you want to see it [Fig 1].

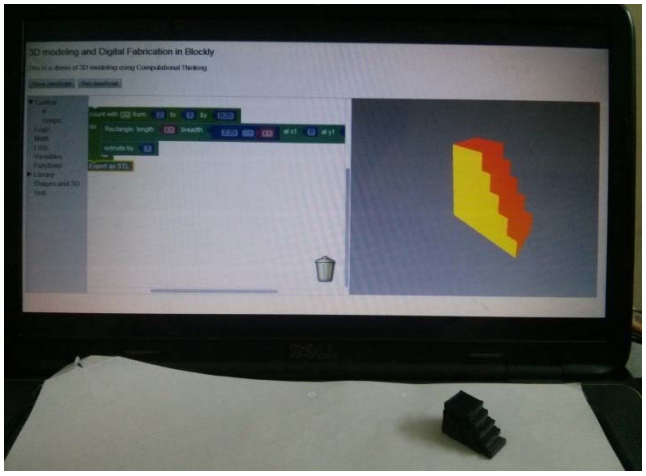

Figure 2. A looping program written to generate the model of a staircase in an earlier version of FabCode. The 3D printed model as created from the STL file that FabCode produced is shown, thereby showing the cycle of writing a program for modeling, seeing the virtual model, then 3D printing it for iteration and studying its tangible features

3.1.2 Block-based programming

FabCode's interface was built using the Blockly API [26]. It is a graphical programming editor which removes the hassle of mastering syntax of programming languages before one can code. To this visual programming interface, we add our own blocks for the purpose of creating shapes and performing operations to generate 3D geometries. These can be easily understood by the user. Each block has a tooltip message attached to it to explain to novice users the functionality of each block, if need be. These blocks include 2 categories- shapes which comprise of rectangle, circle, ellipse, n-sided regular polygon and gear and 3D Operations which consists of bulge (extrude), move to, rotate around axis and intersection, subtraction, joining of solids. Contrary to most novice CAD environments, we decided not to provide ready-made 3D models to users. Therefore, we skipped giving blocks of cuboid, sphere, cylinder, pyramid etc., instead encouraging the user to think of how these can be made from the blocks provided. Users can make static or functional objects using blocks like gear block in combination with other parts. The interface also has a 'show code' option where a person would be able to see the actual underlying JavaScript code running [Fig 1].

3.1.3 Tinkerability

FabCode, like Scratch, is tinkerable because it lets users experiment with commands and code snippets the way one might tinker with mechanical or electronic components.

3.1.4 Low threshold, High ceiling and wide walls

Sets of tools and materials like clay, paints and canvas, etc. have their respective barriers and accessibilities. In computational design, many barriers do not exist, and new powerful tools are available. We make these tools available through programming. Block-based programming, pre-defined starting tool set and pre-platform activities will help provide structures-to-think-with and lower the threshold to the generation of complex objects. Logo was designed for children. Our platform will potentially grow with children into adulthood. They will add tools to toolsets by making those tools as functions first. They can then move to more complex designs by black-boxing self-designed functions which can create models through passing of parameters. Sharing of user-made functions as tools will enable more complex creations through collaboration. FabCode also has the characteristics of wide walls- supporting different ways of expression or many paths towards the same object.

3.2 Development

To have it be capable of supporting extensive 3D modeling, FabCode's functioning is written using three.js, a lightweight cross-browser JavaScript library/API used to create and display 3D computer graphics on a Web browser [27]. Blockly has various visual programming 'blocks' built into it that implement programming constructs like variable defining, loops, math operations, conditional statements etc. We have built various new blocks that help model 2D and 3D designs and convert these designs to file formats that are supported by digital fabrication machines, STL for 3D printing and SVG for laser cutters [Fig 1].

3.3 FabCode Workflow

To explain the entire workflow and to bring clarity to how we are linking computational thinking and modeling of objects, let us take an example of modeling a spiral staircase. To be able to model a spiral staircase in FabCode, one has to understand the design of a staircase by applying computational thinking skills of decomposition, pattern recognition, pattern generalization, abstraction and algorithm design. A user would have to understand the pattern of how each step is placed in relation to the previous step such that they form a spiral. They would then have to device an efficient algorithm to represent this pattern using the tools of programing that FabCode provides. For example, one would write a loop program for doing this. Using programing as a tool to design can result in users making more abstract designs by designing algorithms such that they could create a staircase with different base polygons- like a hexagon staircase. FabCode provides for creation of new functions that become a part of the blocks palette. Thus, one could create a generic function using which one could create a spiral staircase with different sided polygons as base. All this would require a coming together of different ideas and concepts! [Fig 2].

4. FUTURE WORK

4.1 Evaluation

We plan to evaluate FabCode in terms of the usability of the functions provided and the effectiveness of FabCode in reaching its intended goal of encouraging computational thinking of objects.

For the former, the usability and effectiveness of FabCode will be evaluated with users and the results will be used to give direction to the design of FabCode. Right now, we have made blocks based on the existing modeling software and its nomenclature, but we need to validate their usability in our context. For the latter, we will record the designs that users create along with the 'code' they write to generate it. Analyzing this, we aim to see if our direction has been effective in getting users to think computationally and in an efficient manner. Considering most of the designs that users create would have been possible to model in a variety of ways, we

will aim to see if the user has used ‘brute' ways of modeling rather than thinking of an efficient algorithm.

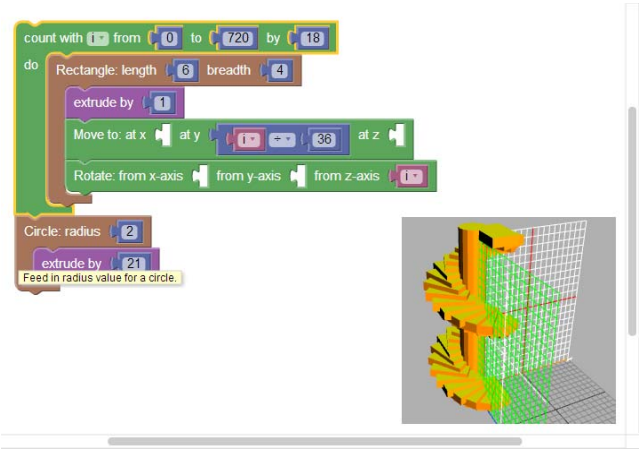

Figure 3. Creating a spiral staircase model (code on left and rendered model in sub- image)

4.2 Activity Design

We plan to design activities that fit well into curricula of STEAM disciplines. We see FabCode’s and the activities’ active deployment in maker spaces and places like FabLab@School. FabLab@School already aims to integrate its activities in the curriculum and we will iterate upon our activities so that they can be easily integrated.

5. CONCLUSION

FabCode is an attempt at making a platform wherein programing for physical objects acts as a tool to encourage computational thinking and build engineering and design skills. We want to situate FabCode such that it facilitate imagination and investigation of ideas, not acting just as just a validator of imagined ideas but as a material-to-think-with.

6. REFERENCES

[1] Resnick, Mitchel. "Sowing the Seeds for a More Creative Society." Learning & Leading with Technology 35.4 (2008): 18-22.

[2] Blikstein, Paulo. "Digital Fabrication and ‘Making’in Education: The Democratization of Invention." FabLabs: Of Machines, Makers and Inventors(2013): 1-21.

[3] Eisenberg, M. (2003). Mindstuff: Educational Technology Beyond the Computer. In Convergence, Summer 2003.

[4] Papert, S. (1980). Mindstorms : children, computers, and powerful ideas. NewYork: Basic Books.

[5] http://www.hindawi.com/journals/tswj/2014/428080/

[6] Jacobs, Jennifer, and Leah Buechley. "Codeable objects: computational design and digital fabrication for novice programmers." Proceedings of the 2013 ACM annual conference on Human factors in computing systems. ACM, 2013.

[7] Papert, S. 1980. Constructionism vs. Instructionism. Proceedings from Japanese Educators Conference. http://www.papert.org/articles/const_inst/

[8] Piaget, J. (1962). Play, dreams, and imitation in childhood. New York: Norton.

[9] Ginsburg, Herbert P., and Sylvia Opper. *Piaget's theory of intellectual development* . Prentice-Hall, Inc, 1988.

[10] Blikstein, Paulo, and Dennis Krannich. "The makers' movement and FabLabs in education: experiences, technologies, and research." Proceedings of the 12th International Conference on Interaction Design and Children. ACM, 2013.

[11] Resnick, Mitchel, et al. "Scratch: programming for all." Communications of the ACM 52.11 (2009): 60-67.

[12] Tisue, Seth, and Uri Wilensky. "Netlogo: A simple environment for modeling complexity." International Conference on Complex Systems. 2004.

[13] Resnick, M. 2006. Computer as Paintbrush: Technology, Play, and the Creative Society. In D. Singer, Golikoff, R., and Hirsh-Pasek, K. (Eds.), Play = Learning: How Play Motivates and Enhances Children’s Cognitive and Social-Emotional Growth. Oxford University Press.

[14] Zeising, Anja, Eva-Sophie Katterfeldt, and Heidi Schelhowe. "Considering Constructionism for Digital Fabrication Software Design."

[15] http://www.sketchup.com/

[16] https://tinkercad.com/

[17] http://www.123dapp.com/design

[18] Johnson, Gabe, et al. "Sketch it, make it: sketching precise drawings for laser cutting." CHI'12 Extended Abstracts on Human Factors in Computing Systems. ACM, 2012.

[19] http://www.solidworks.com/

[20] http://www.rhino3d.com/

[21] http://www.autodesk.com/products/autodesk-inventor-family/overview

[22] http://www.grasshopper3d.com/

[23] Johnson, Gabe. "FlatCAD and FlatLang: Kits by code." VL/HCC. 2008.

[24] Resnick, Mitchel, and Brian Silverman. "Some reflections on designing construction kits for kids." Proceedings of the 2005 conference on Interaction design and children. ACM, 2005.

[25] Maloney, J., Resnick, M., Rusk, N., Silverman, B., and Eastmond, E. 2010. The scratch program- ming language and environment. ACM Trans. Comput. Educ. 10, 4, Article 16 (November 2010), 15 pages. DOI = 10.1145/1868358.1868363. http://doi.acm.org/10.1145/1868358.1868363.

[26] https://code.google.com/p/blockly/

[27] http://threejs.org/