Limited Embodied Programming

Teaching programming languages to architects

Julien Nembrini¹, Guillaume Labelle², Jeffrey Huang³
¹,²,³ Media & Design Lab, EPFL, Lausanne, Switzerland
¹,²,³ http://mxd.epfl.ch, http://ANAR.ch
¹ julien.nembrini@epfl.ch, ² guillaume.labelle@epfl.ch, ³ jeffrey.huang@epfl.ch

Abstract. The paper presents a teaching experiment using the LOGO language to introduce geometric programming to architectural students with no a priori coding knowledge. Based on extreme simplicity as well as instant visual feedback, the language allows to introduce core programming concepts with little technicality. The extension of the language to a 3D space triggers designers interest and creativity while the introduction of a simple robotic drawing machine confront them with a simple fabrication context. These elements concur to develop a critical approach of the use of digital tools in the architectural design process, with the underlying aim to raise the students awareness on the implication of tooling on their design practice.

Keywords. Geometric Programming; CNC; Digital Tools Teaching; Low-Tech.

Introduction

With the proliferation of digital media in the architectural production, designers are engaged with challenging integration of computational techniques and software tools to perform a wide range of their professional tasks. According to research studies, design tools have an impact on the design process (Fussler 2008). Indeed (Kalay 2006) warns that, despite the fact that information technology can support “uniqueness and unpredictability of creative design processes”, the use of specific digital tools in design must be carefully balanced according to their affordance, i.e. “the potential of the technology to enable the assertive will of its user”. Early enthusiasm for CAD software led to a fascination for complex geometries now easily generated without clear understanding of the underlying concepts, comforting the assumption that what could be drawn could also be built. This assumption may partially explain the current fascination for CNC fabrication techniques, seen as the perfect solution to overcome the physical and technical limitations introduced by increased levels of complexity in digital models (Cardoso 2009).

This paper presents a recent teaching experiment engaging architectural students with an involving hands-on self-made digital process from computational form generation to CNC fabrication. Through a rapid introduction to the basics of programming, the aim of this workshop is to propose a critical framework to overcome superficial usage of software. From previous experience in teaching full featured object-oriented programming for geometry, and introducing a limited imperative approach, we suggest that a restricted programming environment becomes an opportunity to reduce programming complexity for the benefit of core concepts understanding, without impairing creative expression.
We also show that the use of low precision CNC machines introduces the notion of limitations of fabrication processes, emphasizing the development of critical approaches towards computational techniques applied to architecture.

Teaching Computational Tools in Architecture

With the growing interest for programming interfaces (API) dedicated to geometric 3D modeling, geometry programming has democratized and is now an expected feature of modern CAD softwares [RhinoScript, FormZ, AutoLISP, OpenGL, Processing.org]. While the potentialities of such profusion of APIs has sparked increasing interest among students as well as teachers, we are now facing the challenging problem of determining what are appropriate forms of teaching programming to architecture students, which especially lack prior background in this topic.

Specifically, the early stages of such learning process can be frustrating and an approach based on pre-baked geometric examples can easily lead to over-directed design, channeling students production around variants of provided examples. In the proposed paper, we outline strategies through which architectural students are guided to overcome such difficult early stages and how those represent a different attitude toward CAAD software culture.

Universality of Programming Teaching

The field of CAAD is highly dominated by competing commercial software, whose programming API are usually tightly integrated to an existing geometric kernel. Fierce competition between industrial CAD standards leads to proprietary aspects, resulting in tendency of studio teaching, as well as CAAD research, to carry a branded meaning. We postulate here that this observation goes against a universal point of view needed for teaching.

Constraints as Creativity

To overcome what is perceived as a limitation, we propose an approach based on a long-standing language with a fairly limited set of functions: LOGO, for Logic Oriented Graphic Oriented, introduced in 1967 to support constructivist theories of learning (Resnick 1999). Beyond LOGO’s original aim to teach children logic and algorithmic, we emphasize here on the relevance of particularities of the language to teach programming for architectural design.

Imperative Programming

At first sight, LOGO has an imperative programming style, where programming is defined as a sequence of commands and where the output results from a sequence of subsequent actions. Imperative programming may be seen as a good introduction to automation as it emphasizes on basics of sequence handling, which is a crucial aspect in learning programming. On the other hand, LOGO subroutines (procedural programming) and loops are elementary enough to deal with universal mechanical aspects of programming.

Intrinsic Extrinsic geometry

In terms of geometry, one important aspect of LOGO is the use of a local frame of reference to control the drawing tip – called a Turtle. This comes in accordance with a subjective perception of the physical world and helps in the representation of agent-based systems. If the original LOGO version is only 2-dimensional with a single turtle, the programming framework used, Processing.org [1] and a library ANAR+ [2] provides students with 3D world and multiple turtles to support their ability as architectural designers to think about space.

Embodied approach

Following subsequent developments of LOGO into physical turtles, we propose to the students to further develop their project using a DIY handcrafted physical realization of a LOGO turtle, made of LEGO
Figure 1
‘Fabrication’ of a drawing. Students chose to ink the robot’s wheels to record the drawing construction process. [Students: Grangirard & Mermod]

Figure 3:
first examples presented by students using LOGO
parts, step motors and an Arduino board [3] for control. This robot’s accuracy is very low, also because of low-cost components.

Together with the robot, a suite of simple codes are provided to export LOGO instruction sequences into an ad-hoc robot commands instruction set. This provides a direct and intrinsically simple insight in the translation process of architectural drawings into CNC commands, commonly used by students but usually abstracted as file format compatibilities. It results in confronting students with material complexities of CAD/CAM operation.

**Geometric Framework ANAR+**

The teaching experiment presented here takes advantage of a research project aiming to propose an open source engine for parametric architectural design, known as the ANAR+[2] library. It takes the form of an extension to the IDE project Processing.org [1] which has proven very successful in introducing programming to designers. If the library itself has many geometric capabilities already reported elsewhere (Labelle 2009), we will concentrate here on the specific implementation of LOGO that is included in it. This implementation does not claim to replicate the whole LOGO language but instead focuses on a reduced set of instructions which we believe are crucial to introduce core computation concepts to designers. This nature as an extension of the JAVA programming language enables the consideration of more and more programming concepts such as variables, randomness, etc. as the need becomes evident from the students’ requests.

The LOGO language originally introduced to facilitate human-computer interaction, an artifact named the Turtle, which the programmer controlled by typing in some instructions. This turtle had the ability to start drawing along its path if asked to do so. The aim of the authors of this computer language was to demonstrate the potential of the computer to help children understand logic and geometric elaborate concepts through direct involvement and experimentation. Ease of control as well as instant graphical feedback were specifically designed assets to engage children and raise their attention. In later developments, a robotic turtle was introduced to enable manipulation. Children could move the robot and the corresponding sequence of instruction would be recorded.

The implementation of LOGO as part of the ANAR+ library introduces the ability to simultaneously control multiple “turtles” in a 3-dimensional environment but concentrates on a reduced set of instructions originally available. It mainly consists in the following:

- **FORWARD N** or **BACKWARD N** to move the turtle N units back and forth
- **RIGHT N** or **LEFT N** to change the orientation of the turtle by an angle of N degrees on the current plane
- **PENDOWN** or **PENUP** to start or stop drawing
- **REPEAT N […]** to repeat N times a given set of instructions
- **UP N** or **DOWN N** to change by an angle of N degrees the orientation of the current turning plane along the plane defined by the turtle direction and the turning plane’s normal
- **HOME** to go back to the original position.

These instructions are usually shortened in two letters (for instance FORWARD becoming FD). They can be combined in an instruction sequence which produces a graphical output when the code is run, as depicted in Figure 2. Also, the implementation does not allow procedural programming in LOGO itself, though one of the asset of the original language, but would instead require from students to write JAVA functions that output strings of LOGO sequences. For instance, randomness, not being part of the original language, can easily be introduced using the JAVA random function and some string concatenation.

When using the CNC robot, only a subset of these instructions is available, namely the first four delineated above. Indeed robot movement is constrained to the plane and it has no means to go back
to an origin, lacking global positioning information.

```java
Turtle t = new Turtle();

void setup(){
    size(800,400,OPENGL);
    Anar.init(this);
    Anar.drawAxis(true);
    t.LOGO("repeat 10 [fd 100 rt 130 fd 100 rt 100 fd 45 lt 90 fd 100 up 30 fd 100]");
}

void draw(){
    background(200);
    t.draw();
}
```

Figure 2: Initial code example and corresponding output

**Teaching Approach**

The teaching experiment reported here was part of a larger computer-based architectural studio teaching. It consisted in a workshop running over three weeks, divided in three blocks of two full consecutive days. The themes of the blocks were the following:

- Learning programming basics with LOGO which consisted in introducing concepts such as flow control, loops or memory states using LOGO turtles.
- Structuring code and Agent Based simulations introducing notions of code reuse such as functions, as well as presenting the potential of having turtles influence each other.
- Material production with the help of a DIY handcrafted CNC machine which encouraged students to explore and question the provided production means to produce tangible outputs.

Within this structure, students were asked to (re)create their own specific digital tools with limited expectations, in order to let them explore the potential of a programming approach to architectural design.

In the teaching provided, the emphasis was put on letting students formulate questions and difficulties before providing programming solutions or examples. A policy of direct code exchange with the use of Subversion versioning system, enabled instant propagation of individual solutions to the rest of the students, be it provided by the tutors or invented by some student. This care in avoiding pre-baked solutions triggered unexpected results as presented in the following section.

Firstly, students were asked to explore algorithmically defined form as such to produce a set of examples which they found interesting, such as depicted in Figure 3. They were then asked to use multiple turtles and introduce some interaction between them, leading to further complexity (Figure 4). Along with the graphical output, they were asked to state which intention or process led to producing or selecting the presented work. Interestingly, some students chose to refine the work that others left as unfit to further development.

Finally, students were asked to simplify one of their examples to fit with the constraints of the robotic turtle. Doing so, they were encouraged to explore and divert the proposed means of production, which consisted in a simple robotic vehicle on wheels with the ability to lower and raise a pencil. This last step engaged the students to confront the difficulty of adapting their design to the limited capabilities of the robot. The availability and inexpensiveness of the robot allowed for experimentation beyond the originally intended robot tracing behaviour, triggering their creativity in defining production processes prevented in commonly expensive CNC environments.
Figure 4: complex patterns produced by multi-agent logo code [Student: Potterat]

Figure 5: Uncalibrated execution of a drawing. The execution shows a mechanical aesthetic. [Student: Zonderland]
CNC turtle student projects

Here are presented selected student projects most revealing the variety of approaches and outcomes of the last step of the workshop. Together they present the potential of a new approach in teaching numerical tools usage in the production of architecture.

Note that, although presented with the possibility, none of the presented examples chose to modify the code steering the robot, nor would their approach act significantly on the instructions given to the robot, which is the preferred way of action in common CNC operation. Instead they chose to induce an outcome by altering the means of action of the robot itself.

 Revealing the construction process
A group of students chose to ink the robot’s wheels to let emerge the robot path carrying on a simple set of instructions (Figure 1, Grandgirard & Mermod). Despite the simplicity of the drawn figure, the robot path adds a complexity layer to the outcome, revealing the process producing a given figure.

 Exploration of robot accuracy
A student wanted to draw a schematic perspective view which needed the drawing of several parallels, thus implying quite accurate control of the robot position after many rotations. This process initially failed (Figure 5, Zonderland). However by repeatedly running the robot and consistently measuring its movement, the student was able to demonstrate a flaw in the calibration of the control algorithm. According to the student itself, his understanding of CNC accuracy and production limitations greatly improved through this process.

Magnetic perturbation
Another student wanted to make the robot draw on iron filing by fitting magnets on it and let its movement perturb an initially uniform field. For robot motors’ health, the robot was fitted on a glass sheet above the filing and made to draw on glass to make the relation between filing patterns and programmed path more evident. The process resulted in distinctive pattern and grain as can be seen in Figure 6 (Hefti). Such an outcome could evidently not be attained without direct and full access to the numerical facility for non-conventional use.

Discussion
Integrated into a stereotypical cycle of digital design (from form-finding to CNC), we outline the standing relevance of the framework chosen as an introduction to geometrical programming for various
aspects:
- Tight relationship between physical world and computational approaches
- Basic representation of mechanical sequences of commands
- Intrinsic coordinate system, best describing the internal logic of a form
- Simplification of transposition from LOGO to an elementary CNC machine

This relevance enabled the students to reach beyond technical difficulties almost immediately after being introduced in the matter. With no a priori knowledge in programming they were able to tackle agent-based simulations and transpose the acquired programming skills on a physical device with great success. They were thus put in a position to formulate meaningful questioning of the use of such digital tools in the design process as well as producing interesting outcomes.

Acknowledgements

This workshop was an integral part of Prof. Jeffrey Huang’s Design Studio Organicité, held during the winter semester 2009 at the School of Architecture, Ecole Polytechnique Fédérale of Lausanne, Switzerland. In addition to the authors, the studio was taught by Nathaniel Zuelzke, Mark Meagher and Trevor Patt.

References
