TEACHING COMPUTER SCIENCE WITH GEOMETRIC MODELING

António M. Leitão, INESC-ID, Instituto Superior Técnico, Universidade de Lisboa, Portugal
antonio.menezes.leitao@ist.utl.pt

ABSTRACT

Computer Science is becoming a mandatory subject in many high schools and universities. In order to convince students of the actual usefulness of the subject, it is important to teach it in the context of their main interests. In this paper we describe a Computer Science course tailored for students of Architecture. The course was designed to address several architectural needs that were previously identified as good examples for the application of Computer Science and it places a strong emphasis in the use of geometric concepts, particularly, three-dimensional geometric modeling. During the course, students solve a set of architectonic problems using algorithms, which they implement in Racket, a pedagogic programming language. This approach allows students to learn and explore Computer Science as an important tool for architecture and design.

INTRODUCTION

As it happened with other disciplines, such as Mathematics and Physics, Computer Science is becoming a mandatory subject in many high schools and universities. Unfortunately, making a subject mandatory does not necessarily mean that students will enjoy it. In order to convince students of the actual usefulness of the subject, it is important to teach it in the context of their main interests.

In this paper we describe a Computer Science course tailored for students of Architecture which we have been developing in the last seven years. Before designing the course, we interviewed several architects and students of architecture and we studied their activities. This allowed us to identify a set of tasks where we could demonstrate the usefulness of Computer Science for Architecture.

Using the identified tasks as input, we developed the course in such way that several of those tasks ended up becoming the starting point for the explanation of the subject matter.

Given the architectural context, it should not be surprising that the course places a strong emphasis in the use of geometric concepts, particularly, three-dimensional geometric modeling, as motivating topics for the exploration of Computer Science. For example, coordinates and coordinate systems are used to explain data abstraction. The Doric order motivates

functional abstraction. Stairs, step pyramids, corbeled arches, and fractals are good examples for the explanation of recursion. City modeling allow us to explore state-based computation and randomness. Trusses and space-frames are good applications of data-structures. Several different architectural examples, including the columns of the Sagrada Familia and the shells of the Sydney Opera House, are used to explain constructive-solid geometry and shape transformation. Finally, parametric curves and surfaces provide good use-cases for higher-order functions. The course covers a significant fraction of both a Computer Science course and a Geometric Modeling course, and provides students with the necessary programming skills for solving a large range of design problems.

COMPUTER SCIENCE COURSES

Nowadays, there are several architecture courses that include computer science techniques. The Massachusetts Institute of Technology, for example, offers several different courses in this area, including Introduction to Computation in Architectural Design and Design Scripting. The first one teaches building information modeling, generative methods, prototyping, shape calculation and simulation, focusing on the use of Revit. In the final part of the course, students learn shape grammars in the context of AutoCAD. The second course teaches basic programming concepts and representation of formal design knowledge, including parameterized objects, procedural representation of form, typology and architectural grammar, and related topics. This course is divided into two parts, one where students learn RhinoScript in order to work with Rhinoceros 3D, and the other where they learn Processing.

In most cases, these courses take advantage of the programming languages that are available in most CAD tools, namely RhinoScript and Grasshopper for Rhinoceros 3D, VBA and AutoLISP for AutoCAD, MEL for Maya, GDL for ArchiCAD, etc. This has the obvious advantage of providing a working environment that is directly attached to a tool that, supposedly, they know how to use. On the other hand, this is also the cause of two important problems that affect all these courses: (1) students learn how to program for a specific CAD tool but, in general, it is very difficult for them to program for a different CAD tool, and (2), the programming languages that are available in those CAD tools are, in general, not well suited for the pedagogic teaching of computer science.

In order to avoid the first problem, some schools teach more than one programming language or CAD tool. The University of

Campinas, for example, is currently experimenting teaching both VBA and Grasshopper [1]. The University of East London teaches NetLogo 3D, VBA, RhinoScript, AutoLISP, and MEL. Unfortunately, this requires additional teaching time or, alternatively, less teaching material related to computer science. This leads us to consider a different approach: teach just one language, but make it independent of any particular CAD tool by focusing on the generic geometric modeling operations that are available in all CAD tools.

In order to solve the second problem, we must select a programming language with good pedagogic qualities. According to several studies [2, 3, 4], the Scheme family of programming languages is considered one of the best options for introductory Computer Science courses. Racket is a recent and prominent member of that family that places a strong emphasis on the pedagogic aspects of the learning process. For these reasons, we selected Racket as the programming language to use during the course.

The structure of the course was inspired by the famous textbook "Structure and Interpretation of Computer Programs", by Sussman and Abelson [5], but the content is almost exclusively devoted to the solution of architectural problems. In the next sections we explain the main topics of the course.

FUNDAMENTAL CONCEPTS

We begin by teaching the use of algorithms for the rigorous description of processes and the use of programming languages as the medium for such descriptions. We then explain the syntax and semantics of programming languages and, particularly, of Racket.

Throughout the course, we place a strong emphasis on presenting the programming language as a syntactical variation of mathematics. To this end, we focus on the functional programming paradigms, and we minimize the use of concepts that do not have a simple equivalent in mathematics. This approach takes advantage of the fact that students already have several years of experience in the use of mathematics, thus simplifying the learning process.

DATA ABSTRACTION

After the introduction of the fundamental concepts, we move on to data structures and data abstraction. This is an important topic that paves the way for an explanation of coordinates and coordinate systems. Students learn how to describe locations in space and how to operate them in geometric terms, using rectangular, polar, cylindrical and spherical coordinate systems. It is also in this part of the course

that we explain basic modeling operations, first in two dimensions (lines, circles, rectangles, etc.) and then in three dimensions (spheres, boxes, cylinders, etc.). Instead of discussing the specific operations of a particular cad tool, such as AutoCAD, we work with abstract versions of these operations that are applicable to most CAD tools. This allows us to gain independence from any particular CAD tool and simplifies the transition to other CAD tools.

FUNCTIONAL ABSTRACTION

This topic explains parameterized functions, using the Doric order as a motivating example: students learn how to parameterize shapes and how to establish dependencies between parameters so that a particular architectural canon is achieved.

In this part of the course, the only control structures that we teach are function calls, conditional expressions, and recursion. Note that we postpone teaching while-, repeat-, and for-loops, because they break the mathematical properties of the programs, thus making it more difficult for students to informally prove the correctness of their programs. In fact, recursion is strictly more powerful and, in many cases, easier to use that any specific looping construct, so that is what our students learn.

STATE AND RANDOMNESS

Immediately after recursion, we teach computational state, in the sense of named values that affect a computation but that are also affected by that computation. This requires assignment but we restrict its use to functions that really benefit from it, such as random number generators.

We also teach students to hide the assignment operations behind abstraction barriers so that they can forget that they are being used. As usual, students experiment these concepts in the context of some architectural problem. In this case, we ask them to model a simplified city, exemplified in Figure 1.

Figure 1 –Recursive and randomized generation of cities.

In this modeling exercise, students combine several levels of recursion with the use of randomness, so that no two buildings are exactly identical.

Besides learning how to use randomness, our students also learn how to *control* randomness. This is also visible in Figure 1: only a predefined fraction of the

buildings are cylindrical towers, and the height of each building, in spite of being a random value, is capped by a Gaussian distribution.

RECURSIVE DATA STRUCTURES

After acquiring some practice in the use of recursion, students learn recursive data structures, such as lists. We place a strong emphasis in the use of lists for separating the generation of geometrical coordinates from its use for some particular purpose. As examples, students are asked to implement sinusoidal curves and space frames, such as the one presented in Figure 2, where both concepts are used.

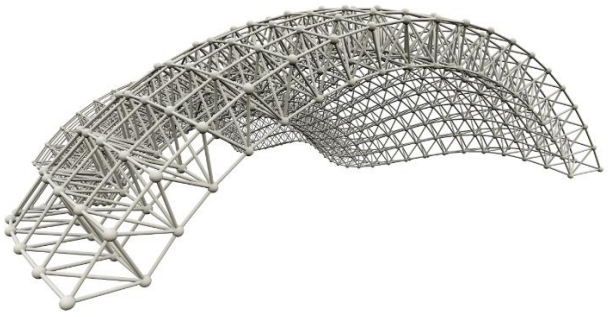

Figure 2 –A space frame with a circular arc whose radius follows a sinusoidal function.

CONSTRUCTIVE SOLID GEOMETRY

The next topic in the course is Constructive Solid Geometry (CSG). Instead of explaining the specific operations provided by the CAD tool being used, we concentrate our efforts in describing a solid as an (infinite) set of points in space, so that modeling operations can be explained in terms of set operations such as union, intersection and subtraction.

In order to allow a mathematical treatment of the subject matter, we also introduce the concepts of empty set and universal set, as identity elements of the set operations. We also demonstrate that without these special elements, that do not have correspondence in any CAD tool, it becomes more difficult to define CSG operations over sequences of shapes.

This approach makes it clear to the students that algorithmic descriptions become easier to develop when we follow a mathematically correct approach instead of just using what is provided by the scripting languages of the CAD tools being used. Figure 3 shows an example of the use of the CSG operations for modeling a shelter.

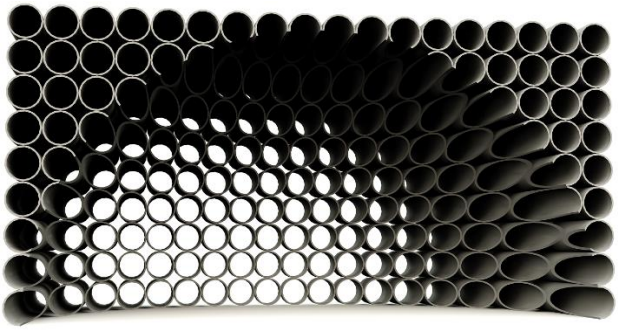

Figure 3 –A shelter made using cylinders, a sphere, and CSG operations.

Besides the basic CSG operations, we also introduce shape forming operations such as revolving, extrusion, sweeping, and lofting, and geometric transformations,

such as, translation, rotation, scaling, and mirroring, as always, by teaching functional abstractions of the actual operations provided by the CAD tool and by providing actual architectural examples, such as the columns idealized by Gaudi for the Sagrada Familia cathedral or the shells conceived by Jørn Utzon for the Sydney Opera House.

HIGHER-ORDER FUNCTIONS

At this point of the course, students have already acquired a strong set of modeling approaches and they have been practicing them in the laboratories. It is then time to teach more advanced programming techniques, particularly, those that depend on the use of higher-order functions. These are functions that accept other functions as arguments and/or return other functions as results.

To illustrate this powerful idea we design a building modeled using a higher-order function that accepts as argument the function that defines the shape of the balcony.

Students also learn higher-order operations over collections, such as mappings, filterings, and reductions, a skill that becomes very useful to understand other languages, such as MEL or Grasshopper, that provide operators applicable both to scalars and collections.

As another application of higher-order functions, we explain the automatic generation of three-dimensional models from site data, a task that, previously, students had to painfully do by hand. One example of the outcome of this process is presented in Figure 4.

Figure 4 –Automatic generation of three-dimensional models from site data.

PARAMETRIC CURVES AND SURFACES

The final topics of the course are parametric curves and surfaces, which become trivial applications of higher-order functions: a function describing a mathematical curve or surface is repeatedly applied over a range of coordinates.

Particular emphasis is placed in teaching some useful mathematical curves and surfaces used in architecture that are not available in most CAD tools, such as the catenary, or the hyperbolic paraboloid. We also explain how to map functions over these curves and surfaces to produce more sophisticated shapes, such as the ones exemplified in Figure 5.

Figure 5 –Shapes that result from mapping simpler shapes over parametric surfaces.

EVALUATION

As was previously described, our approach for teaching Computer Science to students of Architecture is strongly based in the use of computer science concepts for the exploration of architectural examples. The fact that a large number of those examples have been provided by the students themselves is a testimony of the perceived usefulness of the course for their profession.

The students are evaluated using a written exam and a project that is implemented by groups of two students. The project is, usually, the implementation of a parametric model of some well-known building and is proposed by the students. These projects include works of Santiago Calatrava (Turning Torso, Ysios Winery), Richard Rogers (T4 Terminal/Barajas), Norman Foster (Millau Viaduct, Millenium Tower), etc. Figure 6 shows one example of a student's project.

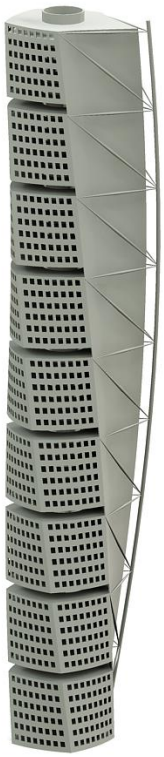

Figure 6 –Calatrava's Turning Torso, generated by the program of one of the students.

Every year, the course is evaluated by the students themselves, which provides us with valuable feedback. These evaluations allow us to conclude that (1) the students recognize the transformative potential of the subject matter for the Architecture profession, (2) the course is perceived as a moderately difficult one, mainly due to the limited amount of time available (just one semester), and (3), the course demands a considerable amount of work, particularly, during the execution of the project.

We are currently trying to convince the school that it would be in the best interest of our students to extend the course duration to two semesters and we believe that this extension will be implemented in the near future.

CONCLUSIONS

In the last years we have been teaching (and evolving) the course described in the previous sections. The course structure was initially influenced by Sussman [5] but we gave it an almost extreme bias towards applications in architecture. This is also the approach taken by Woodbury [6] but whereas he prefers to embrace the modeling techniques promoted by the CAD tool being used (Generative Components), we opt for a more formal treatment of programming, with smaller emphasis on the CAD tool capabilities.

With time, the course evolved to become independent of specific CAD features, liberating students from the typical addiction to a particular CAD tool. We believe we have succeeded in this goal: some of our students are currently developing programs that work with both AutoCAD and Rhino [7].

The programming language used during the course is Racket but we are also looking for other languages with similar pedagogic qualities. Currently, the Python programming language seems to be a strong candidate and we have plans to provide the exact same course but using Python instead. Due to the support for Python that is provided by several CAD tools, this change will allow our students to more easily move between those tools.

ACKNOWLEDGMENTS

We thank our students by the invaluable feedback they have given us since 2007 and, specially, Paulo Fontainha for letting us present his work. The work reported in this article was supported by national funds through FCT under contract Pest-OE/EEI/LA0021/2013 and by the Rosetta project under contract PTDC/ATP-AQI/5224/2012.

REFERENCES

[1] G. Celani, C. Vaz: *Cad Scripting and Visual Programming Languages for Implementing Computational Design Concepts: A Comparison From a Pedagogical Point of View*, in *International Journal of Architectural Computing*, 1(10), 122-137, 2012.

[2] N. Chen: *High School Computing: The inside Story*, in *The Computing Teacher*, 19(8), 51-52, 1992.

[3] A. Berman: *Does Scheme enhance an introductory programming course? Some preliminary empirical results*, *ACM SIGPLAN Notices*, 29(2), 44-48, 1994.
[4] M. Felleisen, C. Findler, M. Flatt, S. Krishnamurthi: *The Structure and Interpretation of the Computer Science Curriculum*, in *Functional and Declarative Programming in Education*, 21-26, 2002.
[5] H. Abelson, G. Sussman: *Structure and interpretation of computer programs*, MIT Press, 1985.
[6] R. Woodbury: *Elements of parametric design*. Routledge, 2010.
[7] J. Lopes, A. M. Leitão: *Portable Generative Design for CAD Applications*, in *Integration through Computation*: Proceedings of the 31st Annual Conference of the Association for Computer Aided Design in Architecture (ACADIA), 196-203, 2011.