Coding Landscape: Teaching Computer Programming to Landscape Architects

Caroline Westort[1]

[1]Iowa State University, Iowa/USA · cwestort@iastate.edu

Abstract: How to best teach coding to landscape architects? Domain-specific approaches to teaching computer programming are surprisingly rare. Most computer programming curricula teach skills in a generic way, to be broadly relevant to many people. A rapidly increasing number and ways of teaching how to code to a range of skill levels is now available online, usually for free (see Appendix, RICHTEL 2015, GASCA 2014, FRAMPTON 2015, SIMS et al. 2011). Yet in landscape architecture coding is often regarded as too difficult, too resource-intensive, insufficiently relevant to practice, or otherwise peripheral to the core mission of the profession to teach (WESTORT et al. 2013) . As a result, fundamentals of coding logic remain largely un-taught in accredited core curricula in the U.S. This paper has three objectives:

1. Offer a landscape architecture-specific approach to teaching introductory computer programming that combines a) *landscape parametrics* with b) concepts of computer programming *logic* and c) basic *computer graphics.*
2. Present a *sequence of exercises* intended to impart fundamental skills and peak student interest.
3. Showcase student project results that use the approach.

A sequence of short programming exercises asks students to define the geometry of elements from the landscape palette – vegetation, landform, water, weather, lighting – and then to modify them using increasingly more advanced and complex coding principles in a modular fashion. The following criteria for successful landscape design software is offered to students as a guide to structuring their software:

- Graphically display landscape geometry, such that it is
- Interactively editable/modifiable/deformable and
- Analysable with accuracy and some precision
- Quickly, while being
- Easy to learn

Keywords: Landscape parametrics, computer programming, coding, design computing, landscape palette, teaching, Geodesign

1 Introduction

Teaching computer programming to landscape architects is not new. Required in the core curriculum at Harvard University's Graduate School of Design from 1991-1993, MITCHELL et. al. 1991 offered computer programming using a "top-down" approach with Pascal. It remains the only case of a required course for coding in a landscape architecture curriculum this author is aware of. Computing in undergraduate landscape architecture programs usually consists of instruction in off-the shelf software, often following a 2D, 3D, 4D model, or a generative, analytical, visualization, narrative structure (CANTRELL 2013). Moreover, the nature of coding and software tool-making has evolved significantly in recent decades. Off the shelf software is used and sometimes customized with code with languages like *Python* for *Arc Info*, *AutoLISP* for *AutoCAD*, often pushed to its algorithmic or performance limits with the large datasets and scales typically called for in a landscape architecture design project.

"High level" coding is now possible for the previously uninitiated via use of application programming interfaces (APIs), scripts, macros, and other "click and drag" integrated programming environments (IDEs). Also known as visual content creation tools, *Scratch*, *Processing*, *DWNLD* and *Telerik* are among the most popular with large user bases, and many landscape designers and geospatial thinkers are active using them.

This author's assertion that it is important for landscape architects to know the fundamentals of computer programming stems from two convictions:

1) Current software available for landscape design rarely fulfills the criteria for effective landscape design software, so more landscape architects need to be involved in developing software tools.

2) Digital representation of our landscape palette needs more algorithmic attention. Curved surfaces, fuzzy edges, large scales characterize our palette, and remain limited, unwieldy and largely divorced from analytical functionality in design software used in design studio.

Landscape architecture may be regarded as an Information Technology (IT) profession, and in the U.S. has ambition to be formally recognized as a Science Technology Engineering and Mathematics (STEM) discipline (ASLA 2013, LARCH 2013). Several notable contributors to landscape architecture, IT and Geographic Information Systems (GIS) in particular have made their contributions in part from knowledge of computer programming, e. g. Dana Tomlin, Jack Dangermond (CRISMAN 2006, TOMLIN 1994, 2013, 1990) exemplifying how the discipline benefits from coding literacy.

Two recent phenomena also highlight the opportunity for heightened understanding of computer programming among landscape architects:

(1) Google Earth was launched in 2008 which has quickly raised geospatial literacy and expectations among an increasingly tech-savvy global public. Together with Google Maps an opportunity for landscape architects to customize the application of geospatial concepts to a broader potential customer base is apparent, and coding knowledge could leverage this opportunity.

(2) The Hour of Code phenomenon, where 46 million-and-counting people started to learn how to code online at codeacademy.org the first year, and millions more in subsequent years. Hour of Code is broad acknowledgement that coding, and the associated computational thinking it calls for are essential skills for the 21st Century (RICHTEL 2014, FOLEY 2013, MCDONALD 2013, PARTOVI et al. 2014).

The two phenomena considered together represent powerful converging forces for how control over one's digital environments and tools is becoming a universal human priority. Coders with design training or experience are recognized across industries as uniquely valuable in sectors where creative thinking coupled with application knowledge is a particularly potent combination (WIDDICOMBE 2014, WATKINS, 2014, TO 2014).

2 Approach

The approach to teaching introductory computer programming to landscape architects offered here is based on combining three topics:

1. Introductory coding logic – Elementary concepts of computer programming that are present in the majority of computer programming languages. The list is derived from a survey of content offered in online coding courses listed in the Appendix, and the syllabus of an introduction to computer science course offered at the Massachusetts Institute of Technology (GRIMSON).
2. Basics of computer graphics – Euclidean geometry, interactivity, color. These concepts represent a subset of concepts introduced in
3. Landscape parametrics – Algorithmic description of the geometry of the landscape palette consisting of vegetation, terrain, water, weather and lighting effects.

	Landscape Parametrics		**Computer Graphics**			**Generic Coding Logic**
LANDSCAPE PALETTE						
ELEMENT	DYNAMIC	DIMENSION	GEOMETRY	COLOR	INTERACTION	CONCEPT
terrain		2D	point	RGB	mouse	variables
vegetation			line	grayscale	position	scope
water			polygon		down	types
weather			ellipse		up	conditionals
lighting			rectangle		click	loops
			triangle		character entry	recursion
						functions
		2.5D - 3D	grid/field/mesh			arrays
			sphere			classes
			column			objects
			cube			
	growth	4D				
	gravity					
	wind					
	waves/ripples					
	rain/snowfall					
	etc.					

Fig. 1: Matrix Summary list of the approach topic elements, landscape parametrics, computer graphics and coding logic

The proposed approach combines "generic" computer programming logic, shown in blue in Figure 1, with two essential core areas of landscape computing: Landscape parametrics, shown in green, computer graphics, yellow. Landscape Parametrics is the piece that customizes the approach to our domain (Figure 2). The combination of these three topics with the criteria for effective landscape design software structure the individual coding exercises.

2.1 Criteria for Effective Landscape Design Software

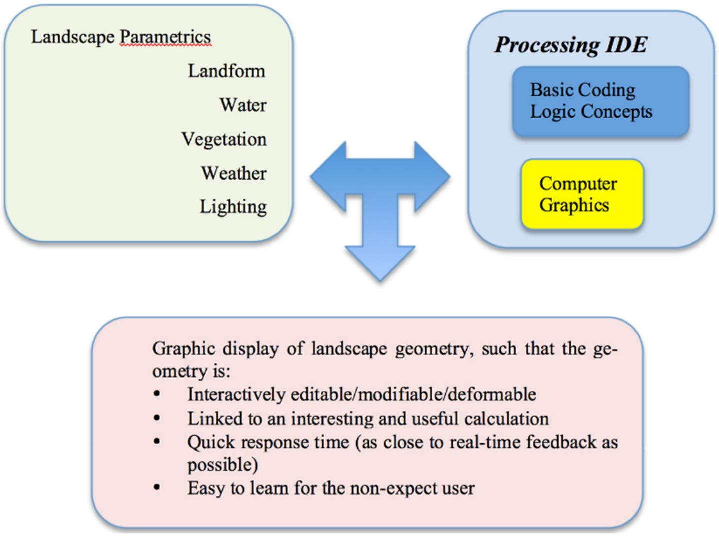
For coding to feel relevant and useful to landscape architects, it should fulfill the following criteria for successful design software:

1. Display a 3D representation
2. Edit/Modify/Deform/Change the representation
3. Analyze the representation quantitatively
4. Do so quickly, in close to real time
5. Such that it is easy to learn (for the non 'expert user')

For more discussion on these criteria see WESTORT 2015 and ERVIN et al. 1995.

These criteria contribute to the selection of the coding tools and environment. The open source programming language and IDE *Processing* was selected for this course. One of the stated aims of Processing is to act as a tool to get non-programmers started with programming, through the instant gratification of visual feedback. Its focus on the electronic arts, new media art, and visual design communities with the purpose of teaching the fundamentals of computer programming in a visual context with visual feedback. The "sketch" as a version of "drafts", or easily changed versions of alternatives, makes it consistent with the above criteria for landscape design software, and highly suited to serve as the platform for the class. The language builds on the Java language, but uses a simplified syntax and graphics programming model (WIKIPEDIA PROGRAMMING LANGUAGE).

Figure 2 diagrams how Processing provides the tools for basic coding and computer graphics content, and the exercises and domain knowledge are engaged in the Landscape Parametric module. Processing plus landscape parametrics are combined to fulfil the criteria for effective landscape design software in the assignment sequence.



Graphic display of landscape geometry, such that the geometry is:

- Interactively editable/modifiable/deformable
- Linked to an interesting and useful calculation
- Quick response time (as close to real-time feedback as possible)
- Easy to learn for the non-expect user

Fig. 2: Summary of approach topic elements that combines Landscape Parametrics, Computer Graphics and Coding Logic

2.2 Exercises

A sequence of short programming exercises asks students to define the geometry of elements from the landscape palette – vegetation, landform, water, weather, lighting (ERVIN, 2001) – and then to modify these geometries in a modular fashion using increasingly more advanced and complex coding principles.

Three primary priorities structure the sequence of exercises:

1. Each exercise shows graphics on the screen.
2. Initial coding exercise represent the geometric parameters of an initial subset of parameterized landscape "primitive" forms:
   a. tree
   b. water droplet
   c. landform field/grid

Figure 3 shows the graphic output of the first 3 coding exercises.

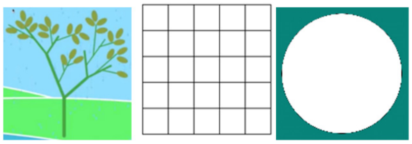


Fig. 3: Graphics from first 3 Coding Assignments, representing landscape primitives: tree, landform, water droplet

3. Each landscape primitive form is coded in at least 2 different ways, with the graphics on the screen remaining constant. This is to show how the digital design medium involves at least 2 representations: external and internal. External representations are the display – or what is visible on the screen – and internal representations are the algorithms and data structures written in code or on disk that describe the graphics that are visible.

To illustrate how this works, we follow the exercise sequence for landform:

First, students are asked to program a 2-dimensional horizontal line to represent a simple section profile. Students are shown how to manipulate the position on the screen of its end points, the line's color, stroke thickness using the pre-programmed Processing function *line*(). Students are then asked to draw multiple horizontal lines, offset from one another as parallel lines. Students are asked to do the same for vertical lines, then to super-impose these lines onto the horizontal lines. Next students draw a box around all lines, to resemble a landform 'field' shown above.

Students are then asked to code the same grid using a looping structure. The graphic output is the same yet the algorithm is more complicated than the previous 'hard-coded' line by line solution. Students are asked to re-write their grid code such that there is a *drawGrid*() function that contains all grid-related calls.

Finally students are asked to re-draw their grid from vertex to vertex. Again the graphics looks identical to prior grid outputs to the screen. So far the grids have all been 2D. At this stage students are asked to add a z-coordinate to each vertex, making them three-dimensional points. Interactive computer graphics techniques are then introduced from the Processing IDE's library of pre-programmed functions for displaying the grid in axonometric view and add graphical user input functionality. Students are then asked to interactively manipulate single vertices, see Figure 4.

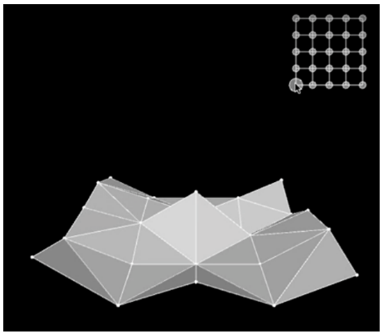


Fig. 4:
Student coding assignment showcasing fulfilment of criteria for effective landscape architecture design software through editable 3D vertex coordinates that are interactively editable

Final Project – Synthesis

The second half of the semester asks students to develop their own user-defined final project. The project objective is to implement a software programming project that synthesizes prior programming concepts and features GRAPHIC, INTERACIVE, and ANALYSIS elements:

GRAPHIC: What do you need to see? Defines and displays the geometry of at least 1 of the following landscape elements, landform, water, vegetation, weather effects, including lighting, and

INTERACTIVE: How do you want the user to interact with the landscape element? Allow direct interactivity of a set of geometric operations upon your selection.

ANALYSIS: What useful/interesting/novel calculation can be accurately performed on your landscape element?

Each student video records a demonstration of results publicly, online and for juried review. Example graphics from these projects are shown below:

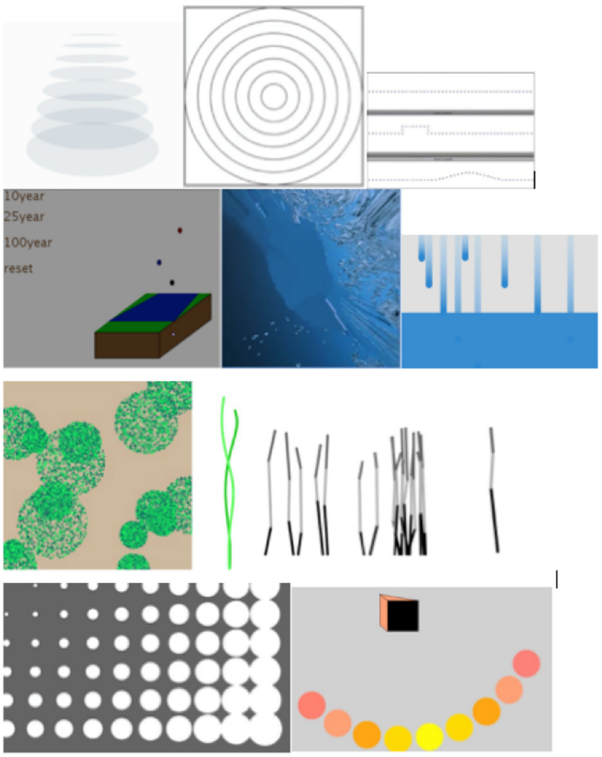


Fig. 5: Example graphics representing individual landscape palette elements from student generated code, final project assignment. Top row to bottom row: Landform, water, vegetation, lighting.

3 Discussion

Feedback the author has received on this approach follow one of two general themes:

1) *The landscape palette definition is too narrow or confining.*
The geometry assigned in the assignments is too simplistic.
The idea that discrete Euclidean geometric forms, e. g. circles, squares, cones, etc. don't and can't capture the richness and complexity of our landscape environment was of concern to the author. Initial offerings of the course has brought encouraging results. Students seem able to quickly and easily scale-up the simpler building blocks to realize their much more complicated intentions. It appears the simple act of "calling a circle a drop of water and coloring it blue", is enough to capture and sustain the interest and attention of students long enough to impart coding skills *and* confidence. That said, the exercise sequence could certainly be refined, and the topic elements listed in the Summary Matrix 'bundled' more comprehensively and elegantly in the exercise sequence.

2) *Landscape architects don't need to know how to code.*
Landscape architecture is not and never will be a STEM discipline.
*Coding will be obsolete in the future anyway (*for a take on this see NICHOLS 2015). This theme is broader and requires a more lengthy discussion than the space available here. In short, however, this author believes an approach to teaching coding from a domain-specific vantage point could catalyze both the acceptance of computer programming as relevant, useful and powerful to our discipline, and the willingness to learn how to do it.

4 Conclusion and Outlook

Offered in this paper is an approach and example results for teaching computer programming in a domain-specific way. The objective is to make the topic relevant and interesting to students of landscape architecture, in order to inspire them to pursue information technology beyond end-use of software. The approach uses landscape parametrics as the basis for teaching this content in a sequence of exercises that follows a progression of complexity for individual landscape palette elements, landform, vegetation, water, weather, lighting, and systems. These encoded parametric descriptions can be both applied to a wide range of project types, and scaled up to increasingly more complex functionality.

The idea that landscape architects could as part of their design process invent "internal" representations of their design intentions in the form of code that can be manipulated, analysed and virtually experienced on a computer screen or a head-mounted display is an exciting prospect that belongs squarely in our discipline. There is a technological imperative to equip emerging generations of landscape architects with the skills to participate in these kinds of opportunities and others yet to be imagined. An accessible, domain-specific approach that folds in the landscape palette with a targeted set of coding principles and computer graphics functionality strikes this author as a worthwhile, interesting, and hopefully useful.

Appendix

ARDUINO. http://www.arduino.cc/
CODE ACADEMY “Hour of Code”, Ed. Code Academy. https://www.codecademy.com/
CODE SCHOOL. https://http://www.codeschool.com/
CODE. http://code.org/
CODESPELLS. http://codespells.org/
CODINGBAT. http://codingbat.com/
COURSERA. https://http://www.coursera.org/
CYCLING ’74 MAX. https://cycling74.com/
EDX. https://http://www.edx.org/
HTML DOG. http://htmldog.com/
KHAN ACADEMY. https://http://www.khanacademy.org/
KUATO STUDIOS. http://www.kuatostudios.com/games/hakitzu-elite/
LCODETHW. http://learncodethehardway.org/
LYNDA “lynda.com”. https://www.lynda.com/
MADE WITH CODE. https://http://www.madewithcode.com/
OF. http://openframeworks.cc/
RUBYMONK. https://rubymonk.com/
SCRATCH. http://scratch.mit.edu/
TREEHOUSE. http://teamtreehouse.com/
TRY PYTHON. http://www.trypython.org/
UDACITY. https://http://www.udacity.com/

References

A.S.L.A. COUNCIL OF EDUCATORS IN LANDSCAPE ARCHITECTURE (2013), Council on Education, ASLA Committee on Education STEM Initiative, Stephanie Rolley, Chair, July 2, 2013.

BROWN LEE, J. (2015), http://www.fastcodesign.com/3043624/25-ideas-shaping-the-future-of-design “The Rise Of The Self-Educated Designer”.

CANTRELL B. (2013), Course description: http://lab.visual-logic.com/academia/la-7103-media-iii/.

CHRISMAN, N. (2006), Charting the unknown: How computer mapping at Harvard became GIS. Esri Press.

ERVIN, S. M. & WESTORT, C. Y. (1995), Procedural Terrain; A Virtual Bulldozer. In: Proceedings CAAD-Futures, International Conference on Computer Aided Architectural Design, Singapore, 1995.

ERVIN, S. M. (2001), Landscape Modeling: Digital Techniques for Landscape Visualization. ERVIN, S. M. & HASBROUCK, H. (Eds.). McGraw-Hill, New York,

FRAMPTON, J. (2015), 12 Sites That Will Teach You Coding for Free, Sept 8, 2015. http://www.entrepreneur.com/article/250323 (Sept 14, 2015).

GASCA, P. (2014), Teach Yourself Coding on Your Own Time with These Resources. Enterpreneur Magazine, August 18, 2014. http://www.entrepreneur.com/article/236511 (October 13, 2015).

GRIMSON, E. & GUTTAG, J. (2012), MIT 6.00 MIT OPEN COURSEWARE, MIT. http://www.youtube.com/watch?v=k6U-i4gXkLM (Introduction to Computer Science & Programming" Lecture 1: What is Computation?).

HADI PARTOVI, J. M. & HURST, V. (2014), Is Coding the Language of the Digital Age? In: A. M. ALEXA LIM (Ed.), Science Friday.

LARCH (2013), Discussion about STEM initiative of ASLA, CELA. Ed. Landscape Architecture Electronic Forum LARCH-L@LISTSERV.SYR.EDU, May 7, 2013.

MITCHELL, W. J. & TAN, M. (1991), "Top Down Knowledge-Based Design" in Digital Design Media; A Handbook for Architecture & Design Professionals. MITCHELL, W. J. & MCCULLOUGH, M. (Eds.). Van Nostrand Reinhold, New York.

NICHOLS, S. (2015), Coding Academies Are Nonsense; CRUNCH Network. http://techcrunch.com/2015/10/23/coding-academies-are-nonsense/.

RICHTEL, M. (2014), Reading, Writing, Arithmetic, and Lately, Coding. http://www.nytimes.com/2014/05/11/us/reading-writing-arithmetic-and-lately-coding.html? r=0

SIMS, Z. & BUBINKSI, C. (2011), "Codecademy". http://www. codeacademy.com.

TO, M. (2014), Designers Code Differently. https://medium.com/learning-xcode-as-a-designer/designers-code-differently-e163a354d6cc.

TOMLIN, C. D. (1990), GIS and Cartographic Modelling, 1st Ed. Prentice Hall College Div.

TOMLIN, C. D. (1994), Map algebra: one perspective. Landscape and Urban Planning, 30, 3-12.

VAN DAM, A. & FOLEY, J. D. (1982), Fundamentals of Interactive Computer Graphics. Addison-Wesley.

WATKINS, C. (2014), Why Designers Really Should Learn to Code. http://blog.capwatkins.com/why-designers-really-should-learn-to-code.

WESTORT, C. Y., ERVIN, S. M., PETSCHEK, P., CANTRELL, B., HOINKES, R. & DANAHY, J. (2013), Compiling in the Core; Computer Programming in the Landscape Architecture Curriculum; Apps, Scripting, Macros & Interaction. In: CELA Annual Meeting: Space. Time/Place. Duration. The University of Texas at Austin, Austin, Texas USA, 2013, p. 361.

WESTORT, C. Y. (2015), Designing DTMs for Automated Machine Guidance (AMG); A Small Scale Case Study Look at Software Requirements (ICCEASI 2015). Proceedings of the 3rd International Conference on Civil Engineering, Architecture, and Sustainable Infrastructure, July 1-3, 2015, Hong Kong University of Science & Technology Hong Kong.

WIDDICOMBE, L. (2014), The Programmer's Price; Want to hire a coding superstar? Call the agent. The New Yorker, 54-64.

WIKIPEDIA, "Google Earth".

WIKIPEDIA "Programming language". https://en.wikipedia.org/wiki/Processing_(programming_language).