Embodied Representations in Computing Education: How Gesture, Embodied Language, and Tool Use Support Teaching Recursion

Amber Solomon, asolomon30@gatech.edu, Georgia Institute of Technology
Miyeon Bae, mbae7@gatech.edu, Georgia Institute of Technology
Betsy DiSalvo, bdisalvo@cc.gatech.edu, Georgia Institute of Technology
Mark Guzdial, mjguz@umich.edu, University of Michigan

Abstract: Computing education research has yet to think about how instructors use embodied actions and ideas when teaching the skills involved in "doing and learning" computing. In this paper, we describe two case studies of computing instructors using embodied representations - in the form of gestures, embodied language, and tool use - to teach recursion. We used grounded theory to analyze a set of naturalistic video recordings of undergraduate computing professors teaching recursion to their class. We contribute a conceptual framework of the kinds of embodied representations teachers use in computing classrooms as the first step towards understanding how embodiment support student learning.

Introduction

Understanding how teachers teach is essential to understand how students learn. Peter Burton highlighted this when he suggested computing education research (CER) should be critical of the modalities between "what actually gets taught; what we think is getting taught; what we feel we'd like to teach; what would actually make a difference" (Burton, 1998, p.54). Recently, other computing education researchers have expanded this view, contending that CER needs to move beyond individualistic theories of cognition to explain teaching and learning (Deitrick et al., 2015). Otherwise, we miss important details that describe how teachers orchestrate available resources to scaffold students' understanding and what "knowledge students utilize to make sense of the problem-solving activities in computer programming" (Deitrick et al., 2015, p.51).

In this paper, we critically reflected on the current practice of instruction, using embodiment as a lens to analyze teaching practices. To understand how the teaching of computing is embodied is to know how embodied representations support the reasoning involved when thinking computationally and expressing computational ideas. Evidence tells us that embodied representations (i.e., gesture, embodied language like perspective-taking and conceptual metaphors, and tool use) are pervasive in computing classrooms, which suggests that embodiment may be central to both the learning and teaching of computing (Cunningham et al., 2017 & Solomon et al., 2018). However, in CER, we have little discussion about how instructors use their bodies when teaching the skills involved in "doing and learning" computing. Learning and the practice of computing are not "purely intellectual activities, isolated from social, cultural, and contextual factors" (Deitrick et al., 2015, p.55), but are dependent on our bodily-grounded experiences and opportunities.

We investigated the following research question: What embodied representations do teachers use while teaching in computing classrooms? Answering that question is the first step towards understanding how embodied representations can support students' reasoning and learning in computing classrooms. To answer the question, we present two case studies where we studied the teaching of recursion in computing classrooms to understand how teaching was embodied during instruction. We selected recursion because the difficulties with learning the topic are well documented in computing education literature, and it is a topic that is generally agreed upon that every computing student should know (Chao et al., 2018). Drawing from contemporary theories of embodied cognition (e.g., Ackerman, 2012; Kirsh, 2013; Stevens, 2012), we used grounded theory to analyze a set of naturalistic video recordings of undergraduate computing professors teaching recursion to their class. We paid particular attention to how the professor used embodied representations - i.e., gesture, embodied language, and tool use - to support the learning of recursion. We contribute a conceptual framework of the types of embodied representations teachers use in computing education, which we elicited from the two case studies. The paper concludes by suggesting how these uses of embodiment in teaching recursion may impact student learning.

Setting the context of recursion

Our discussion of recursion applies to programming languages under the imperative paradigm, which includes C++ and Scratch. While imperative languages might be considered the most *authentic* in professional practice, it does limit the applicability of our findings. To start, we must understand what invoking a function means; in summary, it is to tell the computer to execute a set of actions. Recursive functions are functions, or sub-routines, defined in terms of themselves, that is a function that invokes itself. Recursive functions have two parts: (1) the

base case, which is the terminating case of the recursive invocation, and (2) the recursive case, which is the recursive invocation in which the function uses itself to solve part of the original problem (Chao et al., 2018).

Students notoriously have difficulty forming viable mental models of recursive functions, and without a viable mental model, students cannot accurately reason about recursive functions. A mental model of recursion includes "the beliefs/understanding/views of how recursion works" (McCauley et al., 2015, p.45). Learning to write and trace recursive functions is challenging for three main reasons (Chao et al., 2018). First, people do not naturally think recursively, and there are no 'obviously recursive' real-world experiences or analogies that students can draw upon to understand recursion. Second, students have trouble understanding recursive execution because the code and computer hide the execution of a recursive function, and it is a difficult process to describe (Chao et al., 2018). To write and trace a recursive function, students must understand the following concepts: function invocations, function returns, parameter name reuse, and a function invoking itself. The third reason is that most students have not "previously encountered neither the concept nor its associated vocabulary" (McCauley et al., 2015, p.54). Computing education, programming languages, and software development all rely on metaphors. The terms 'recursion' and 'function' are conceptual metaphors used to name abstractions (Colburn, 2008). Computing teachers have the daunting responsibility of scaffolding the student's development of a viable mental model of recursion. Researchers have described several strategies teachers use to do this but have not used embodiment as a lens. In the following sections, we present an analysis of the ways computing teachers use embodiment, by analyzing the types of embodied representations they used.

Theoretical framework

In general, embodiment implies that the mind is not the sole source of knowledge, but we make meaning about the world from our body-based, lived experiences (Stevens, 2012). Using embodiment in the analysis of teaching practices, therefore, shifts the unit of analysis from the individual's mind, to an activity that is culturally and historically situated. Teachers are using embodiment to express computation, even if not intentionally (e.g., through gestures). Teachers cannot help but use embodiment because they are themselves embodied, and students are watching them. Students are sensing the motion and gesture, even if teachers are not intentionally using gesture and other embodiment. Consider Seymour Papert, who argued that body-syntonicity, or using the knowledge and sense of one's body, contributed to learning Logo because it helped students make the abstract concrete (Papert, 1980). Moreover, interventions like the MoveLab and CS Unplugged exploit the relationship between body movements and computational thinking to motivate and engage underrepresented students to learn computational thinking (Aranda & Ferguson, 2018; DesPortes et al., 2016).

However, CER has yet to view classroom instruction of computing as an embodied activity. As a consequence, computing education theory is missing a significant part of how educators support student reasoning and learning (even if the educators are unaware of it consciously). We need exploratory, ground-up research to develop theory to think about the relationship between teaching and embodiment, and its impacts on learning. We contribute a conceptual framework to understand the embodiment in computing instruction. We focus on three embodied representations: gesture, embodied language, and tool use. These representations were the most salient in computing classrooms and have practical implications for pedagogy.

Gesture

It is well-documented that gestures, or spontaneous hand movements produced when talking, are evidence of embodiment and impact what and how students learn. Previous research suggests gesture promotes learning and reasoning by encouraging students to link abstract concepts and by "acting out" an abstract concept, which gives students a concrete representation with which to engage (Goodwin, 2013). Taxonomies of gesture classify gestures based on specific functions or models of gesture production. A deictic gesture is when "a speaker points to actual objects that are either present, non-present, or metaphorical in nature" (Roth et al., 2002, p.287). Iconic gestures are when "they bear a perceptual relation with concrete entities and events" (Roth et al., 2002, p.287). Lastly, metaphoric gestures are "similar to iconic gestures in that they have a narrative character, but the images produced relate to abstractions" (Roth et al., 2002, p.287).

In their exploratory study observing gesture production in introductory computing courses, Solomon et al. attempt to fit the gestures seen in computing classrooms into an existing gesture taxonomy (2018). They suggest that computing and its lack of reconciliation with defining abstract, make differentiating between the types of gesture difficult. These are consistent findings with Roth et al. (2002). Deictic (pointing) gestures are easy to determine, but it is difficult to differentiate between iconic and metaphoric gestures. Therefore, in our study, we identify gestures as either deictic or metaphoric (including iconic).

Embodied language: Conceptual metaphor and perspective-taking

We discuss two ways that language is a product of embodiment: conceptual metaphor and perspective-taking. Lakoff and Núñez state that language, specifically conceptual metaphor, is a product of embodied thought: "much of what is 'abstract'… concerns coordination of meanings and sense-making based on… forms of metaphorical thought. Abstract reasoning and cognition are thus genuine, embodied processes" (Lakoff & Núñez, 1997, p.30). Metaphors make "apprehending" abstract concepts "accessible" to students through comparison with familiar, bodily experienced concrete concepts. We focus on one type of conceptual metaphor, ontological metaphors. Ontological metaphors use concrete objects to represent an abstract idea (e.g., personification).

The second way is perspective-taking. Ackermann considers perspective-taking to be a body-based activity, necessary for acquiring a deeper understanding (2012). Ackerman uses the language of "diving in," or situating oneself to become part of the phenomena, and "stepping out" of the phenomena, which helps someone reflect on their experiences, forming more abstract insights. In their seminal paper, Ochs et al. investigated how physicists' reason about graphical data representations (1996). They describe a physicist using first-person statements as if they are the "experiencer": "When I approach a phase transition…" (Ochs et al., 1996, p.341). They also describe physicists using "physics-centered grammar" that turns "physics" into agents: "Then it crosses to domain state" (Ochs et al., 1996, p.337). Ochs et al. write that "scientists express their subjective involvement ... by taking the perspective of (and empathizing with) some object being analyzed and by involving themselves in graphic (re)enactment of physical events" (1996, p.357).

Tool use

David Kirsh argues that tool use and embodiment are entangled: "…interacting with tools changes the way we think and perceive – tools, when manipulated, are soon absorbed into the body schema, and this absorption leads to fundamental changes in the way we perceive and conceive of our environments" (Kirsh, 2013, p.1). Thus, our use of tools, like programming languages and metaphors, becomes part of how we think about ourselves and how we relate to the world (i.e., the body schema). In our analysis, we focus on sketches of code traces, since it is a commonly used tool in computing classrooms. In their analysis of the kinds of sketches of a code trace that students draw, Cunningham et al. defined a sketch in computing as the following: "a programmer's written visualizations of program state or any other computing process" (Cunningham et al., 2017, p.164). Sketches, and by extension, sketching, is typically considered a mechanism to manage cognitive load. However, when viewed as a body-based activity, sketching is an effective activity for reasoning because it acts as "an external mediating structure" (Kirsh, 2013, p.5). The act of creating the externalization primes someone's thinking, and the "aspects" a person chooses to sketch highlights their subject of reasoning (Kirsh, 2013, p.5).

Method

Data sources

The video data that was analyzed in this study was collected as part of an exploratory study about how spatial representations (i.e., gesture, spatial language, and visualizations) appear in computing teaching in classrooms. In total, we gathered 227 minutes of video data from four undergraduate professors. We emailed professors at universities that are allowed to video record their classrooms, scoured MOOCs, and online video databases to create our video corpus. We received or looked at 33 videos. Because the study was based upon who responded and who was able to send us video recordings, it potentially provides a skewed view of what happens in computing classrooms. This is a limitation for two reasons. First, all professors were middle-aged white men at universities in the United States of America. Second, professors in the videos are known as excellent computer science teachers. However, this sample is representative of teachers and teaching practices that are rewarded by institutions – these professors were all tenured at prestigious computer science departments – and modeled by others. Each course was taught in a different imperative language. All the courses were "conventionally structured" (McCauley et al., 2015), with Socratic-style lectures in an auditorium classroom and practical laboratory work in another session. The videos show only the professor, but some student voices are intelligible.

For this paper, we review two of the instructional moments related to recursion from two instructors. These two were selected because they were the most salient examples of embodied representations. The two cases offer us concrete examples in which to talk about a conceptual framework that can be used to think about embodiment in computing education. Our goal is not to make any confirmatory or nonconfirmatory claims about intentionality or if the students understood the embodied representations but to document the use of and to provide an initial conceptual framework of the embodied representations used in computing teaching.

Data analysis

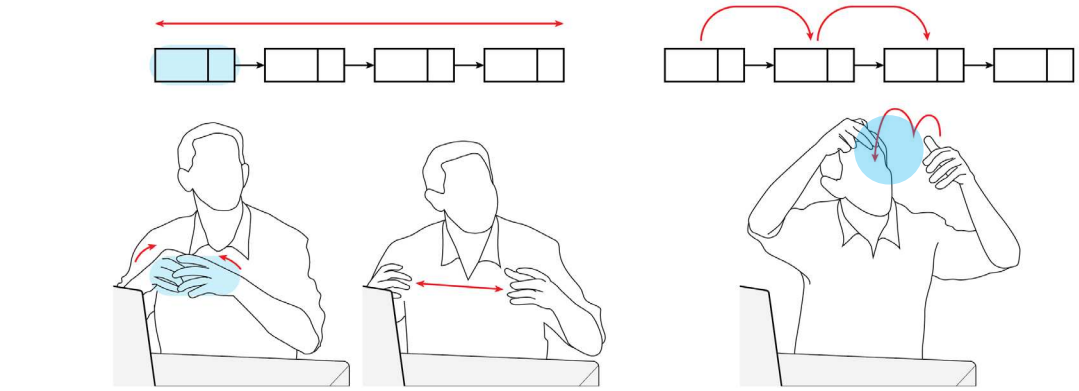
The video data were analyzed using grounded theory (Charmaz & Belgrave, 2007). For each recording, we first reviewed and transcribed the entire recording to get familiar with the content of the video, and then created a timestamped content log for each recording. A micro ethnographic approach to data reduction was used to 'tag' moments that we believed indicated the teacher's use of embodied representational content. We identified those moments from our content logs. We then created multimodal transcripts of the moments produced in style inspired by Goodwin (2013). These tagged moments were first analyzed inductively to form our theoretical understanding of the ways teaching was embodied during instruction. Afterward, we reanalyzed the codes and classified them deductively as metaphoric or deictic gestures, embodied language, or tool use. Two researchers played back the videos within video analysis software repeatedly with and without sound to pay attention to when and what type of gesture was produced. Deictic was any pointing gesture, and metaphoric was any gesture that was not a deictic or a communicative gesture, like a thumbs up or okay sign. We studied the transcripts to identify utterances containing ontological metaphors and perspective-taking, i.e. when the teacher used first or third person. Tool use was anytime a professor created a sketch of a code trace.

Case studies

In this section, we present two case studies in the style of vignettes to illustrate how gesture, embodied language, and tools were used by two computing instructors.

Case study 1: Gesture

This series of excerpts highlight the ways the professor used gestures and other embodied representations, while collaboratively writing a recursive program. The class is a more advanced introductory course and taught in Scratch. There was a table in the classroom that the teacher sat at with their computer connected to a projector. During the 23-minute video excerpt analyzed for this case study, the professor is reviewing recursion for the class's upcoming midterm. The professor has the class participate in a "group programming session," where the professor, in real-time, writes the code for a game on the computer, which is then projected for students to see. The professor asks the students to help him figure out the code. While describing the rules of the game, the professor projects a diagram of a tree (see Figure 2.A) for students to see and uses a hand gesture that resembles the tree diagram. A tree is a type of data structure, and it is a standard convention to state that a tree has "children" nodes. The professor tells the students that they have to write three functions: (1) a function that generates a new position, (2) a function that returns a value from a list, and (3) a recursive function that iterates through the list. Then, the professor takes down the diagram of the tree, opens the Scratch programming environment, and begins writing code.



(A) Talk: So, this is going to return*
Action: *brings hands together*

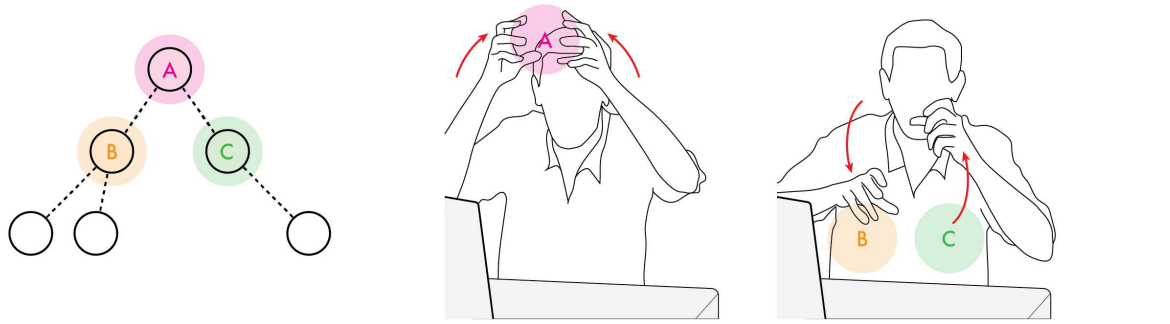
(B) Talk: a list* of all the arrows.
Action: *Spreads hands apart*

(C) Talk: All* the moves you've got
Action: *Uses left hand to point to different positions in space*

Figure 1. The professor uses metaphoric and deictic gesture to describe a list. At the top of this figure is a typical conceptualization of a list. Talk marked with an asterisk (*) co-occurred with the gestures shown in the image.

The professor quickly writes the first function since it only had one line of code. To write the second function, the professor asks the students how "they might get some value from the list." Figure 1 is the transcript of a clip of this interaction, which shows the instructor using a metaphoric gesture that represents a list (a type of data structure, see Figure 1.A and Figure 1.B) and a series of deictic gestures to provide a visual of each element in the list as if he is iterating through the list (see Figure 1.C). The professor "points" to different locations in

space as if he were pointing to different chunks of data in the list. The intention of that gesture seems to be to help students reason about the logic for the program. The metaphoric and deictic gestures seem to provide the instructor's mental model of a list, which could help students conceptually understand the functionality of a list. Consequently, the gestures appear to provide a concrete example that a student could use to reason about how the function might work. Frequently, conceptual models of lists show a linear array of "chained" boxes (see Figure 1), and his movements seemed to mimic this conceptualization.



(A) Image of the tree diagram the professor presented to the class.

(B) Talk: Remember, I have 5*.
Action: Forms hand into *a circle (A)*

(C) Talk: I want to get 6* (B) and 7* (C).
Action: *Drops hand (B), then drops hand (C)*

Figure 2. The professor uses metaphoric gesture while describing the tree and children nodes. Talk marked with an asterisk (*) co-occurred with the gestures shown in the image.

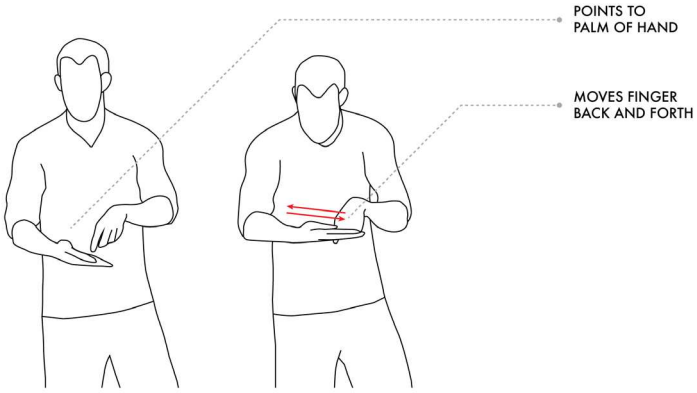
After writing the code for the second function, the professor describes the logic for the recursive function. First, the professor describes the base case. While repeating the series of deictic gestures of iterating through a list, he asked the students, "When are you done? If not at a leaf node, end of list, then these things happen." These gestures could connect "being done," the base case, with the list reaching a specific state, which provides a visual representation that could help students understand when the recursive invocation ends. Next, the professor describes the recursive case, saying, "they need to understand what children the tree has." Figure 2 is the shortened transcript of this interaction. The professor used the same metaphoric gesture that resembled the tree diagram – without projecting the tree diagram – by raising his arms and placing his hands in a circle as if showing a specific node's placement in the tree. He then moves his left hand, followed by his right hand, as if he were traversing a tree to get to another node (see Figure2.B and Figure2.C). Repeating the series of gestures and gesturing "traversal" appears to help students recall the knowledge they need to solve the problem and could help with learning recursion by making visible the recursive function execution.

Case study 2: Embodied language and tool use

This sequence of excerpts highlights an instructor's use of embodied language and tools when defining recursion and tracing a recursive function. The class is an advanced introductory course and taught in C++. The teacher had a podium with a computer and stylus, and a connected projector. During this 17-minute video clip, the professor is introducing recursion to the students. First, the professor describes recursion with an analogy of looking up a set of words in a dictionary in combination with a visual-gestural narrative that acts out the analogy, seen in Figure 3. The professor points to the palm of his hand as if he was reading from the dictionary (see Figure 3.A), then uses a sweeping gesture to indicate looking up the definition of other words that were part of the definition of the first word (see Figure 3.B). Then, the professor describes recursive solutions with the ontological metaphors "powerful" and "elegant:"

> *And, one of the reasons we're going to spend so much time on it is that it's extremely powerful° for solving certain kinds of problems. There are certain examples and, we will do some of them together, where this technique will allow you to write these very short, elegant° solutions.*
> °Indicates an ontological metaphor.

The contrasting metaphors invoke feelings of strength, influence, and daintiness. Most recursive solutions are considered "good code," since they have considerably fewer lines of code than solutions that use for-loops yet are functionally the same. The embodied representations and analogy appear to invite the students to use their lived experiences to understand, functionally, how recursion operates, and when recursion is appropriate to use in a code solution.

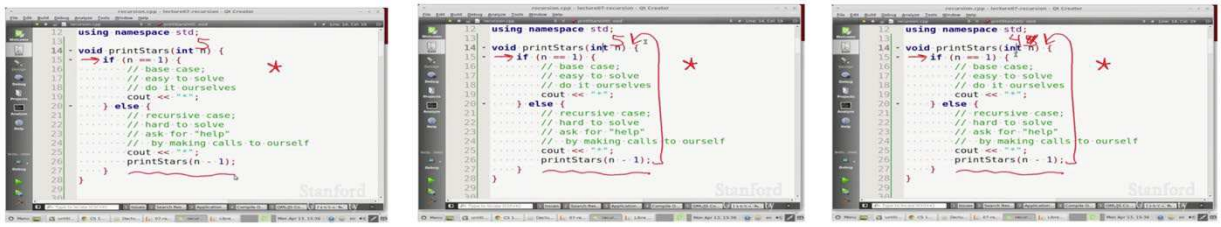


(A) Talk: The entry* for the word.
Action: *Points to palm of hand*

(B) Talk: And you read* all of, you know, you read the definition to yourself.
Action: *Moves finger back and forth*

Figure 3. The professor uses gesture to 'act out' looking up words in a dictionary. Talk marked with an asterisk (*) co-occurred with the gestures shown in the image.

Next, the professor has the class participate in a group programming activity to rewrite a function to use a recursive solution instead of a looping solution. The function, **printStars**, produces 'n' number of stars. After writing the solution, the instructor sketches a trace of the recursive execution. The sketch of the code trace allows someone to "see" the process of a recursive execution. Figure 4 is a segment from the interaction. The professor writes a "5" above the statement **(int n)** to show the students that the parameter '**n**' has a starting value of 5. Then, he draws an arrow next to the statement **if (n == 1)** to show the students that the program will begin by evaluating that condition. After determining the condition is false, he draws the output – a star. He then underlines the recursive invocation, emphasizing the function invoking itself (Figure 4.A). Notice the professor uses embodied language: "Then **it** says "... "now **I** have to **call**." The statement, "then it says," suggests the professor is using perspective-taking, by *stepping out* and giving the recursive invocation "agency." Then, the professor switches perspective, "now I have to call," as if he is *diving into the code,* experiencing the invocation. The professor also uses a common type of ontological metaphor, personification, when he states that "he*"* has to "call" the recursive invocation, similar to calling someone over the phone. Next, the professor draws an arrow that points back to the function while saying, "so it makes it call this function again" (see Figure 4.B). Again, the professor uses personification and perspective-taking, but this time to "show" a function invoking itself. Then, the professor crosses out the '5' and draws a '4' (see Figure 4.C). Cunningham et al. refer to this sketching technique as the crossout method. They suggest that it "demonstrates that previous values are no longer accessible by a strike-through" (Cunningham et al., 2017, p167), which is an excellent way to depict to students a function reusing a parameter: a concept students typically have trouble understanding when learning recursion.



(A) Talk: Then it± says, "now I¢ have to call° printStars with n minus 1." n minus 1. 5 minus 1. That's 4.
Action: *Underlines printStars(n-1)*

(B) Talk: So that makes it± call° this function again.
Action: *Draws arrow pointing to the function header*

(C) Talk: But, with an n of 4, right?
Action: *Crosses out the 5 and draws a 4*

Figure 4. The professor sketching a code trace while using embodied language. The professor's talk indicating embodied language are noted: *stepping out±, diving in¢, ontological metaphor°.*

Discussion and conclusion

We described two case studies to show how computing instructors use embodied representations - in the form of gestures, embodied language, and tool use - to teach recursion. Table 1 is a summary of the embodied representations found. We contribute a conceptual framework of the kinds of embodied representations teachers

use in computing classrooms as the first step towards understanding how embodiment supports student learning. Next, we hypothesize two questions that are important for future research to consider: (1) Which of these embodied representations are intentional, and which are not? (2) What is the effect on student learning?

Table 1: A summary of the embodied representations teachers used

Embodied Representation	**Example**
Gesture	
Deictic	Case Study 1: Professor "pointing", suggesting iteration
Metaphoric	Case Study 1: Professor gestures a "tree"
Embodied Language	
Ontological Metaphor	Case Study 2: When the professor said, "that makes it call"
Perspective-Taking	Case Study 2: When the professor said, "Then it says"
Tool Use	Case Study 2: Professor sketches a code trace

The goal of a computing instructor is to help students form viable mental models of how the computer works. However, to form viable mental models, learners must understand abstract concepts that cannot be perceived directly through the senses. Herb Simon argued that computing is a science of the artificial, something "designed" and not inherited from nature (1996). For example, the discipline relies on metaphors based on lived experiences, e.g., naming procedures or subroutines a function, to name the abstractions (Colburn, 2008). Computing education researcher Ben Shapiro argues that because computing is a science of the artificial, we cannot understand learning about computing without interrogating the practices and sociocultural contexts that help students with understanding and reasoning (Ko, 2019). Our findings indicate why it is important to use embodiment as a lens to understand learning in computing education: the embodied representations are likely some of the few resources students can use to understand the abstract. We know that students attend to "shallow features." Movement, gesture, language choice, and tool use are features we might expect a novice to attend to because they do not know enough about what is not important to attend to.

If we assume that students are attending to these embodied representations, then they likely affect student learning. We discuss two ways the embodied representations possibly supported student learning. In case study 2, we identified instances of embodied representations in the professor's language and tool use when he sketched a code trace. Research about program comprehension indicates that it is hard for students to form mental models of code execution because they do not understand statement sequencing (McCauley et al., 2015). Therefore, one interpretation is that the embodied representations supported students' formation of viable mental models by "concretizing" statement sequencing and letting students "symbolically live the experiences" of function invocation. Sketches, then, could have helped concretize statement sequencing in two ways. First, the teacher's sketch of a code trace showed his mental model of recursive invocation by drawing the critical "aspects" that students need to know to understand statement sequencing (Kirsh, 2013). Second, the act of sketching could prime students to make predictions and think about what happens next. Research in science education shows that students often find it helpful to identify with individual elements in a model, and then view phenomena from the perspective of this element (Ackermann, 2012). Using embodied language likely "immersed" students into the function invocation - as if they were physically in it - helping them envision statement sequencing.

In Case Study 1, we described an instance of the professor using both deictic and metaphoric gestures to describe the functionality of a list and to 'act out' iterating through a list. The professor likely used embodiment to make salient the critical aspects of a list, i.e., iteration, which provided a utility for and the knowledge needed for students to reason about the logic. Moreover, metaphoric gestures typically depict concepts that are challenging to describe in words and are "shaped" by particular objects. Scopelitis et al. argue that, “the gesturing hands can be employed as tools to build a representational object that both the speaker and the hearer act upon in order to achieve a shared understanding of a complex concept” (2010, p.1125). Therefore, when the teacher used metaphoric gestures, they were likely embodying and sharing their mental model of a list, which gave students something concrete with which to reason.

Teachers are using embodiment to express computation, whether they are conscious of doing that or not. There are things professors are doing in the classroom that are calculated and purely pedagogical content knowledge, and other things that are less conscious and about personal communication styles. For example, the dictionary analogy and accompanying gesture described in Case Study 2 may be a strategy that has been refined over time. It seemed that the analogy was carefully crafted. The ontological metaphor - "this calls the function" – could be intentional because "calling" is a standard convention to describe function invocation (Colburn, 2008),

but could not be intentional for communicating meaning. In other words, the terminology is likely so automatized for the teacher, that they are not intentional for conveying embodied meaning. The gesture of a list described in Case Study 1 may not be intentional but produced spontaneously to help students' reason. Asking questions about intentionality is essential for criticality and reflectivity of exactly what teachers are communicating by using these embodied representations. Research that has studied the effects of scripted gesture on mathematical learning has generally shown positive learning outcomes. If computing instructors were more intentional in their use of embodied representations, then we may see learning gains.

The analysis presented in this paper is the first step towards understanding how embodiment might affect understanding and opportunities for learning about computing. It is premature to offer suggestions for teachers to effectively use embodied approaches. However, our study points toward a need for a more in-depth investigation of the ways teachers and students use embodied representations and its effects on student learning.

References

Ackermann, E. (2012). Perspective-taking and object construction: Two keys to learning. In *Constructionism in practice* (pp. 39-50). Routledge.

Aranda, G., & Ferguson, J. P. (2018). Unplugged Programming: The future of teaching computational thinking?. *Pedagogika*, *68*(3), 279-292.

Burton, P. (1998). Kinds of language, kinds of learning. *ACM SIGPLAN Notices*, *33*(4), 53-61.

Chao, J., Feldon, D. F., & Cohoon, J. P. (2018). Dynamic Mental Model Construction: A Knowledge in Pieces-Based Explanation for Computing Students' Erratic Performance on Recursion. *Journal of the Learning Sciences*, *27*(3), 431-473.

Charmaz, K., & Belgrave, L. L. (2007). Grounded theory. *The Blackwell encyclopedia of sociology*.

Colburn, T. R., & Shute, G. M. (2008). Metaphor in computer science. *Journal of Applied Logic*, *6*(4), 526-533.

Cunningham, K., Blanchard, S., Ericson, B., & Guzdial, M. (2017, August). Using tracing and sketching to solve programming problems: Replicating and extending an analysis of what students draw. In *Proceedings of the 2017 ACM Conference on International Computing Education Research* (pp. 164-172). ACM.

Deitrick, E., Shapiro, R. B., Ahrens, M. P., Fiebrink, R., Lehrman, P. D., & Farooq, S. (2015, July). Using distributed cognition theory to analyze collaborative computer science learning. In *Proceedings of the eleventh annual International Conference on International Computing Education Research* (pp. 51-60). ACM.

DesPortes, K., Spells, M., & DiSalvo, B. (2016, February). The MoveLab: Developing congruence between students' self-concepts and computing. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education* (pp. 267-272). ACM.

Goodwin, C. (2013). The co-operative, transformative organization of human action and knowledge. *Journal of pragmatics*, *46*(1), 8-23.

Kirsh, D. (2013). Embodied cognition and the magical future of interaction design. *ACM Transactions on Computer-Human Interaction (TOCHI)*, *20*(1), 3.

Ko, A. J. (2019, July 13). Dagstuhl trip report: learning and teaching programming language semantics. Retrieved from https://medium.com/bits-and-behavior/dagstuhl-trip-report-learning-and-teaching-programming-language-semantics-b8d8d9007380.

Lakoff, G., & Johnson, M. (2008). *Metaphors we live by*. University of Chicago press.

McCauley, R., Grissom, S., Fitzgerald, S., & Murphy, L. (2015). Teaching and learning recursive programming: a review of the research literature. *Computer Science Education*, *25*(1), 37-66.

Ochs, E., Gonzales, P., & Jacoby, S. (1996). " When I come down I'm in the domain state": grammar and graphic representation in the interpretive activity of physicists. *Studies in Interactional Sociolinguistics*, *13*, 328-369.

Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas*. Basic Books, Inc..

Roth, W. M. (2001). Gestures: Their role in teaching and learning. *Review of educational research*, *71*(3), 365-392.

Scopelitis, S., Mehus, S., & Stevens, R. (2010). Made by hand: gestural practices for the building of complex concepts in face-to-face, one-on-one learning arrangements.

Simon, H. A. (1996). *The sciences of the artificial*. MIT press.

Solomon, A., Guzdial, M., DiSalvo, B., & Shapiro, B. R. (2018, August). Applying a Gesture Taxonomy to Introductory Computing Concepts. In *Proceedings of the 2018 ACM Conference on International Computing Education Research* (pp. 250-257). ACM.

Stevens, R. (2012). The missing bodies of mathematical thinking and learning have been found. *Journal of the Learning Sciences*, *21*(2), 337-346.