Using Conceptual Blending to model how we interpret computational models

Brandon R. Lunk
Department of Physics, Texas State University, 749 N. Comanche St., San Marcos, TX 78666

With the growing integration of computational modeling in introductory physics curricula, educators face an increasing need to understand how students read and compose programming code so as to better support those students' learning. In this paper, I will discuss Conceptual Blending as a framework for modeling how we read physical, mathematical, and logical meaning into the structural and grammatical features of programming code; modeling how features of the programming representation can affect student reasoning, both productively and counter-productively; and informing instructional interventions. After a discussion of the framework, I will present a case study to help illustrate how conceptual blending can help interpret student difficulties.

2019 PERC Proceedings edited by Cao, Wolf, and Bennett; Peer-reviewed, doi.org/10.1119/perc.2019.pr.Lunk
Published by the American Association of Physics Teachers under a Creative Commons Attribution 4.0 license.
Further distribution must maintain attribution to the article's authors, cover page, and DOI.

I. INTRODUCTION

As the inclusion of computational modeling in the introductory curriculum continues to gain traction within the broader physics community [1–5], educators face an increasing need to understand how students read and compose programming code so as to better support these students' learning. While there exist broad discussions of computational thinking [e.g. 6] and catalogues of student errors [7], a detailed picture of student reasoning ought to contextualize these within the process of mapping physical models and mathematical relationships into programming code [8] and integrating these representations into a coherent understanding [9, 10].

The framework of *Conceptual Blending* [9–11] can provide a means for describing how we read physical, mathematical, and logical meaning into the structural and grammatical features of programming code; how we connect features of the code to the results of running the code; and how features of the code can affect student reasoning, both productively and counter-productively. From here, the framework can also help prescribe instructional interventions. Following a discussion of the theory and how it can relate to programming representations, I will present a case study to illustrate how Conceptual Blending can help us interpret student difficulties.

II. CONCEPTUAL BLENDING THEORY

According to Fauconnier and Turner [9], people make meaning by selectively projecting elements from multiple conceptual schema into common—or blended—mental spaces. The result of forming these blends is to generate new connections between elements of the input spaces and to compress/expand disparate or abstract relationships (e.g. time & space, causality, role) from the input spaces into something that can be easily imagined within the human experience.

Consider, for example, a computer desktop GUI (Graphical User Interface): by blending the space of computer commands and information processing with the familiar spaces of office work and looking-through-windows, the desktop establishes a blended environment in which a user can navigate complex computational systems with minimal effort. Within this blended mental space, operations from the computational space, such as "save" and "print," are combined with operations from the office space, such as "trash" and "file," to produce a streamlined workflow [9].

A "mental space" is characterized by a set of cognitive resources that can be activated together under an organizing frame that specifies the relationships and roles among the elements of that space [10]. Blending can involve inputting elements from one mental space into the frame of a second (called "single-scope blending"); it can also involve combining the input frames themselves ("double-scope blending"). The result of this is to create emergent structure that is not present in the input spaces.

The blending process itself is tacit and largely subconscious however Fauconnier and Turner outline multiple optimization principles for constructing blends [9]. Of particular relevance to this discussion: a blend ought to productively facilitate a particular mode of reasoning; it ought to maintain an internal consistency (principle of "integration"); relationships and roles in the blended space ought to reflect those of the input spaces (principle of "topology"); and the blend ought to be reversible—that is, one should be able to reconstruct the original input spaces.

While these principles should all be satisfied, some may need to be relaxed in order to optimize the overall blend or to accommodate disagreements—or clashes—between elements of the input spaces. For example, in order for desktop GUIs to maintain a comfortable user experience, the "trashcan" icon doubles as an ejection mechanism; this violates "topology" because office trashcans are for discarding items, not disconnecting them; it also violates "integration" by presenting a hardware manipulation in the guise of a computer memory manipulation [9]. While clashes can be the source of emergent structure—especially in "double-scope" blends like the desktop GUI—failure to recognize that an optimization principle has been relaxed in order to accommodate a clash can lead to errors in reasoning, such as assuming of a blended space too much of an input's topology.

A. Representational Systems

Representational systems are tools by which a community can encode and communicate disciplinary knowledge and which can facilitate the often complex manipulations of the problem-solving process [10, 11]. Indeed, one aspect of expertise is a familiarity with common representational systems [11]. Within *Conceptual Blending* theory, representational systems like the GUI form "material anchors" that can both stabilize a particular blend and facilitate manipulations within that blend [9, 11]. Computational code is an especially important system to consider within the framework of conceptual blending as it can provide a wholly alternative means of representing physical systems, in parallel to algebraic representations [12].

When representing a physical process in programming code, the physicist must attend to three principle bodies of knowledge: "physics," including fundamental principles and relationships, physical models, and disciplinary norms; "mathematics," including mathematical grammar and syntax and rules for algebraic manipulations; and "programming" including data structures, the logic of information flow, and the grammatical and syntactic rules of code. In the language of *Conceptual blending*, these are understood to be input mental spaces. Learning to blend physical knowledge with mathematics is an important component of expertise [10], but when generating a computational model the physicist must negotiate all three mental spaces and blend them in order to generate algorithms that obey computational logic while maintaining a clear connection to physical principles and a consistency with their typical algebraic representation (see Fig. 1).

One aspect of this process is detailed in the theory of *Forms and Devices* [12]. Symbolic forms are cognitive resources through which people blend intuitive understanding of the

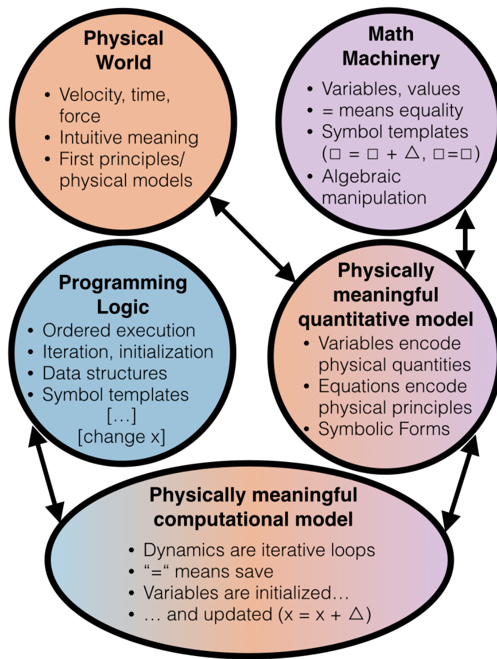

FIG. 1: Physical intuition must be blended with mathematical structures to create meaningful quantitative models [10]. This must be further blended with programming logic in order to design or understand a computational model. Both stages are double-scope blends.

physical world with the grammatical symbol templates that make up both algebraic and programming representational systems. For example, the algebraic *form* "base-plus-change" blends the abstract symbol template

$$\square = \square + \Delta$$

with intuitive notions of change, growth, combination, and identity; The programming *form* "variation" blends the symbol template

[. . .]
[change x],

which abstracts the structure of a program as a two-stage setup-then-iterate process, with the idea that dynamics are tied to this structure and that values to be varied over the execution of the program can be initialized in the setup but must be updated in an iterative loop [16]. Complete fluency requires the physicist to blend algebraic and programming symbolic forms together with more formalized physical, mathematical, and computational reasoning [10, 12].

The variable update provides a concrete example: Within the logical flow of the program, variables take the form of numbers that have specific values within the computer's memory and which may be changed or updated multiple times over the course of a program's execution. The author must deliberately plan the definition and initialization of these variables and prescribe rules for updating them within the structure of the program. Variables that represent dynamic physics quantities such as "v" are commonly updated using the form,

```
v = v + (Fnet/m)*deltat.        (1)
```

This line of VPython code is written to deliberately evoke its algebraic counterpart,

$$\mathbf{v}(t+\Delta t) = \mathbf{v}(t) + (\mathbf{F}_{\text{net}}/m)\Delta t, \tag{2}$$

so as to maintain a clear connection both to the physics of Newton's 2nd law and to the algebraic grammar of the "base-plus-change" *form*. However, this line of code also inherits from the "programming" input space a non-algebraic use of the "=" symbol and a deliberate placement within the iterative portion of the program structure according to the "variation" *form*. A fluent reading requires one to recognize and unpack all of these features.

In addition to considering the coding representation, the physicist must also anticipate the output produced by running the program, be it a number, a graph, or an animation. In order to produce a meaningful output, the physicist must form a blend between the program code and the features of the output representation (e.g. time and length scales). For our present purposes, I will focus on the blending required to read and write a coding representation of a physical model.

B. Failure modes

In using *Conceptual Blending* to model how students read and manipulate representational systems in physics, Gire and Price [11] predict three ways that representational systems might prompt student errors despite otherwise robust reasoning: 1) A feature of the representation supports multiple conflicting meanings in the input spaces—think of how length in a force diagram can indicate both force magnitude and a location in space; 2) a literal feature of the representation is used to represent a quantity of a different type—using arrow length to represent force magnitude, for example; and 3) students are unfamiliar with disciplinary conventions regarding how information is encoded in a representation. The first two of these are examples in which Fauconnier and Turner's optimization principles have been relaxed in order to accommodate a clash between input spaces. While an assortment of student errors in generating computational models have been catalogued [7], those errors associated with the program structure and variable updates—and not syntax or physics—point to possible student misinterpretations of emergent features in the computational modeling blend.

Consider, again, the variable update: the "=" symbol, while suggestive of equality, functions more like a "save" command; calculate the value `v + (Fnet/m)*deltat` and then save it under the name "`v`." In this context, "=" supports conflicting interpretations in the input spaces of mathematics and programming logic. Additionally, "`v`" is reused in anticipation of the next iteration, but the presence of identical variables on both sides of an equation violates algebraic norms, requiring a relaxation of algebraic topology. By contrast, the "+" symbol does support a purely algebraic interpretation (in this context), however placing it alongside the non-algebraic "=" requires a relaxation of the "integration" principle [17].

More subtle is the role of initialization. In the algebra input space, variables represent a spectrum of possible values and

the use of initial conditions or other special cases is often designated with specialized symbols (e.g. v_0). Computationally, variables represent specific values in the computer memory and must be explicitly set to some initial value prior to their use and without the need for special symbols. The distinction between a variable and an initial or boundary parameter is made by its location and use within the code and not by its association with a specific value; Without knowing to attend to the global structure or the logic of the program, this relaxation of algebraic topology may not be evident to the student.

In both cases, the structural features of the code present clashes between the input spaces of mathematics and programming logic while optimizing the need for computational flow and strong connections to the physics input space. According to Gire and Price [11], we ought to anticipate student difficulties in interpreting these representational features. Indeed, Caballero et al. [7] and Lunk [13] have both observed students composing (physically reasonable) momentum updates suggestive of such mis-construals (e.g. `deltap = Fnet*deltat` and `pfinal = p + Fnet*deltat`). To further support Gire and Price's prediction, I turn to an interview with a student interpreting completed code.

III. A CASE STUDY: RENÉE INTERPRETS A COMPUTATIONAL MODEL

Renée (a pseudonym) was a biology major recruited out of her general-level physics 1 course at NCSU during the F'14 semester to participate in a pilot study designed to explore life-science majors' attitudes towards computation [14]. As part of the study, participants were asked to read through a VPython model of a 2-body gravitational interaction (Fig. 2) and speak aloud their thoughts and interpretations as they read the code and predicted the output. I viewed footage of Renée and the other participants with the TXState PER group and we discussed the data until we formed a consensus interpretation.

While the other participants of this study each spent roughly 5 min. scanning through the program and reading some minimal physics content from the code—largely connecting variable names to physics quantities—Renée spent nearly 25 min. parsing the code line-by-line, correctly interpreting many of the syntactic features of the VPython code and often circling back when a line of code shed meaning on a previous line. Renée had previously revealed that she had had experience coding HTML and JavaScript which helps to contextualize her willingness to parse through the code in as much detail as she did. Despite this, Renée encountered difficulty in making sense of the three variable updates present in the code:

```
vcraft=vcraft+(Fnet/Mcraft)*deltat
```

Renée: This is an equation it's using? I think, there's one unknown in this equation; it's gonna be F net. So you probably use it to find out what F net is? Um, V craft I know is a vector and that's a vector we already know so we can cross out V craft and the second V craft as known variables.

```
from __future__ import division
from visual import *
scene.width = 800
scene.height = 800

G = 6.7e-11
mEarth = 6e24
mcraft = 15e3
deltat = 60

Earth = sphere(pos=vector(0,0,0), radius=6.4e6, color=color.cyan)
craft = sphere(pos=vector(-6.4e7,0,0), radius=1e6, color=color.yellow)
trail = curve(color=craft.color)

vcraft = vector(0,2e3,0)
pcraft = mcraft*vcraft
t=0

while t<10*365*24*60*60:
    rate(100)

    r = Earth.pos - craft.pos
    rhat = r/mag(r)
    Fnet = G*mEarth*mcraft*rhat/mag(r)**2

    vcraft = vcraft + (Fnet/mcraft)*deltat

    craft.pos = craft.pos + (vcraft)*deltat

    ## This adds the new position of the spacecraft to the
    ## trail--like leaving breadcrumbs
    trail.append(pos=craft.pos)

    t = t + deltat
```

FIG. 2: The VPython model of a gravitational orbit participants were asked to read.

```
craft.pos=craft.pos+vcraft*deltat
```

Renée: So, this would say, the position of the craft equals the position of the craft plus V craft times delta T. Uh, V craft is a known vector. The position of craft is a known vector as well. And delta T is a known number. So, I'm not sure why this equation exists. [laugh]. Maybe its:::, no::. Craft equals craft, err, position of craft equals position of craft plus V craft delta T. So that would mean that V craft delta T would equal zero. So, either V craft equals zero or delta T equals zero.

```
t = t + deltat
```

Renée: Time equals time plus delta time. Delta time is zero? Hmm. What was T? T equals zero. Zero equals zero plus delta time. Delta time is zero? Is- no, delta time is sixty. How is this making sense? [laugh]

In each of these encounters, Renée is consistent in applying the machinery of algebraic manipulation to the VPython code. I want to point to two aspects of this: First, Renée begins by explicitly describing the velocity update as an equation that is to be used to solve for an unknown which she identifies as `Fnet`. Renée then describes a manipulation of the code, verbally canceling `vcraft` out of both sides. As she encounters each new update, Renée tries to apply the same strategy: identify what is known and mentally manipulate the equation to solve for any unknown quantity, including canceling like terms from both sides.

Second is Renée's repeated references to variables as "knowns" and "unknowns." In particular, statements like "V craft I know" and "T equals zero," in the context of manipulating equations, suggest that Renée is interpreting the variable initializations as if they are defining constants that can be plugged into equations.

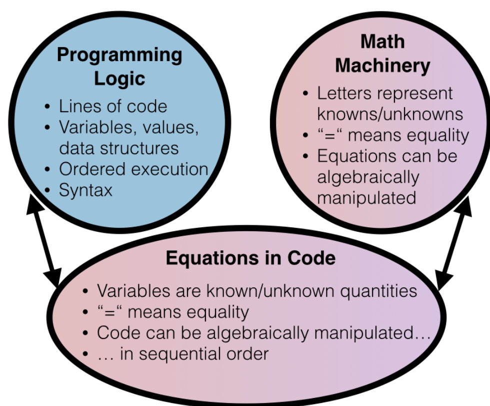

FIG. 3: A single-scope blend in which features of the coding representation are interpreted through the organizing frame of math machinery.

From the standpoint of *Conceptual Blending*, Renée appears to be assuming of the programming code a fundamentally algebraic structure, projecting the topology of math machinery onto the code and "solving" it as one might an equation. While some of this topology is accurate ("+" does indeed mean "+" in this context), much of the machinery that Renée uses is inappropriate to apply to a computational model. Renée also does not appear to be making sense of the *form* of the programming code nor does she attempt to integrate elements from the input spaces of physics intuition or programming logic. This all suggests that Renée is reconstructing the blend as "single-scope," framing the task as being principally algebraic in nature (see Fig. 3)—a stance she may have carried over from her physics class [10]. Renée does recognize that something is amiss, wondering why the position update exists when she isn't able to identify an unknown and asking how the clock update makes sense when `deltat` seems to be two-valued, but these objections are raised through the lens of an algebraic, "single-scope" blend.

Elby [15] suggests the alternate interpretation that Renée is simply activating the "what-you-see-is-what-you-get" resource with regard to the equal sign. While this may indeed be the cuing mechanism, it does not diminish the argument that conceptual blending can inform our understanding of where specific failure modes like Renée's might emerge, what kinds of knowledge students like Renée might apply to a task and, more generally, how experts and novices go about composing, manipulating, and debugging representational systems [11].

An important aspect of *Conceptual Blending* is that student difficulties do not necessarily imply a lack of requisite knowledge but rather a failure to integrate some aspect of the input space. Indeed, during Renée's debriefing, the interviewer discussed the update commands with her and afterward, without prompting, Renée reinterpreted the role of the clock update:

> **Renée**: So, for the first run through, T is going to equal zero and then for the rest of it T is going to be T plus delta T.

This suggests that Renée was capable of engaging in the iterative reasoning required to interpret a programming update statement, albeit with some scaffolding.

IV. DISCUSSION

Conceptual Blending emphasizes the mechanisms by which people might blend mathematics, physics, and programming logic into a coherent computational model and allows us to appreciate the detailed considerations, both deliberate and not, that novices and experts make during computational modeling activities. Additionally, Conceptual Blending allows us to view many student errors in terms of "understandable" failure modes [10, 11] and offers progress towards a predictive framework for anticipating where and why these might arise. Understanding these failure modes can inform the development of instructional interventions, both in-the-moment and pre-designed.

The case study of Renée helps to illustrate a class of student difficulties predicted by Gire and Price [11] whereby students misinterpret features of a representational system that support multiple possible interpretations within the input spaces. Here, Renée assumes of the computational variable updates an algebraic structure and subsequently dedicates time and effort to projecting an unproductive algebraic topology onto the code. Although Renée's background distinguishes her from the population of physics students who might be the immediate benefactors of computational instruction, I argue that many of those students will have a sufficient familiarity with algebra and lack of familiarity with programming code that we might expect them to similarly encounter difficulties with those aspects of the code that support multiple interpretations or rely on specific disciplinary conventions. Future work will involve applying this framework to students in classroom contexts.

In considering the instructional implications, I return to the central principle of *Conceptual Blending*: relate abstract ideas to familiar human experiences [9]. Indeed, it is not hard to interpret Renée's difficulties or other observed errors [7, 13] as being broadly attributable to a familiarity and comfort with algebraic machinery or algebraic representations of physics. In helping students to reflect on the individual input spaces and those clashes that might prompt errors, instructors ought to deliberately scaffold the inclusion of additional input spaces with which we might expect students to have had experience. For example, students' extensive experience saving computer documents could be leveraged to help them understand the variable update; and the use of spreadsheets can help orient students to data structures and to the processes of iteration and initialization.

ACKNOWLEDGEMENTS

I would like to thank the Texas State PER group and especially Hunter Close for providing productive discussions on my analysis and framework, and the NCSU and Elon Physics departments for facilitating the research from which I drew the case study.

[1] Ruth W. Chabay and Bruce A. Sherwood "Computational Physics in the Introductory Calculus-Based Course," *Am. J. Phys.* **76**, 4 pp. 307-313 (2008)
[2] Norman Chonacky and David Winch "Integrating Computation into the Undergraduate Curriculum: A Vision and Guidelines for Future Developments," *Am. J. Phys* **76**, 4 pp. 327-333 (2008)
[3] Marcos D. Caballero and Laura Merner "Prevalence and Nature of Computational Instruction in Undergraduate Physics Programs Across the United States," *Phys. Rev. PER* **14**, 020129 (2018)
[4] JTUPP (2016) *Phys21: Preparing Physics Students for 21st Century Careers* Rep. Joint Task Force on Undergraduate Physics Programs, Oct, 2016 http://www.compadre.org/JTUPP/report.cfm
[5] Partnership for the Integration of Computation in Undergraduate Physics, http://www.gopicup.org
[6] Jeannette W. Wing "Computational Thinking," *Communications of the ACM* **49**, 3 (2006)
[7] Marcos D. Caballero, Matthew A. Kohlmyer, and Michael F. Schatz "Implementing and Assessing Computational Modeling in Introductory Mechanics," *Phys. Rev. ST: PER* **8**, 020106 (2012)
[8] Ruven Brooks "Toward a Theory of the Comprehension of Computer Programs," *Int. J. of Man-Machine Studies*, **18**, 6 pp. 543-554 (1983)
[9] Giles Fauconnier and Mark Turner, *The Way We Think: Conceptual Blending and the Mind' Hidden Complexities* (Perseus Book Group, New York, 2002)
[10] Thomas Bing and E.F. 'Joe' Redish "The Cognitive Blending of Mathematics and Physics Knowledge," in *PERC AIP Conference Proceedings*, Eds. L. McCullough, L. Hsu, and P. Heron (2006)
[11] Elizabeth Gire and Edward Price, "Arrows as Anchors: An Analysis of the Material Features of Electric Field Vector Arrows" *Phys. Rev. PER* **10**, 020112 (2014)
[12] Bruce L. Sherin "The Symbolic Basis of Physical Intuition: A Study of Two Symbol Systems in Physics Instruction," Ph.D. Diss. UC Berkely (1996)
[13] Brandon R. Lunk "A Framework for Understanding Physics Students' Computational Modeling Practices," Ph.D. Diss. North Carolina State University (2012)
[14] Brandon R. Lunk and Robert Beichner "Life Science Students' Attitudes Towards Computational Modeling in Physics," in *PERC AIP Conference Proceedings*, Eds. D. Jones, L. Ding, and A. Traxler (2016)
[15] Andy Elby "What Students' Learning of Representations Tells Us About Constructivism," *J. of Math. Behavior* **19**, 4 pp. 481-502 (2000)
[16] Sherin drew the representation of "variation" from the Boxer programming language in which the variable update less clearly resembles an equation [12].
[17] Although the variable update is part of a numerical integration, Fauconnier and Turner's "Principle of Integration" holds a different meaning entirely, as described in Sec. II [9].