TABLE OF CONTENTS

Chapter 1

Introduction

Computers continue to revolutionize the way that we do and teach science.

1.1 Computation is important

Computation is important as both a research tool and a pedagogical tool.

1.1.1 As a research tool

Computation is increasingly being referred to as the third leg of modern physics, along with experiment and theory. Its utility in solving complicated problems (e.g., predicting chaotic or non-linear motion) that cannot currently be solved analytically makes it indispensable in many field, from physics to chemistry to biology to engineering.

1.1.2 As a pedagogical tool

Sherwood developed and maintained VPython until it turned in Glowscript. VPython was expressly designed to engage students with computational modeling and the three-dimensional visualization that are characteristic to it.

Chabay et al. have studied computation involving VPython in introductory physics classrooms from a number of different lenses. These lenses range from investigating dif-

ficulties that students run into while computationally modeling, to investigating how to be incorporate computation into the classroom, to investigating how students are thinking computationally.

Although much work has been done around computation at the introductory level, there are still many unanswered or only partially answered questions.

1.2 Computational thinking

Computational thinking is a term that has become increasingly popular since its introduction in the early 1980s. This term, although frequently used today, is difficult to concisely explain. Even within the fields of education and computer science, many different viewpoints exist on the topic, and the corresponding definitions are just as varied [9]. However, many of these definitions share one fundamental characteristic: solving complex problems through abstraction and analytic thinking with the aid of computer algorithms.

Seymour Papert first introduced computational thinking in terms of students actively constructing knowledge (constructionism) through the production of an artifact (i.e., a computer program). However, Papert does not initially attempt to define computational thinking. Rather, he comments that attempts to integrate computational thinking into everyday life have failed because of the insufficient definition of computational thinking. He optimistically claims that more attempts to define computational thinking will be made and eventually the "pieces will come together [13]." Papert would later go on to say that computational thinking involves "forging new ideas" that are both "accessible and powerful [14]."

Building on Papert, Jeanette Wing defines computational thinking in terms of taking advantage of the processing power of modern computers with the addition of human cre-

ativity. This echoes the core sentiments expressed by Papert: Using human creativity to forge new ideas that are computationally powerful. Wing is careful to remind readers that computational thinking is a fundamental skill for everyone, not just computer scientists [17].

Further elaboration by Alfred Aho points out that the process of finding the right tool for the right job is a clear indicator of computational thinking. Mathematical abstraction (modeling) is at the heart of computational thinking, and be able to choose between competing abstractions (models) is of critical importance [2]. Aho points out that although there are many useful definitions of computational thinking within computer science, new domains of investigation (e.g., introductory physics) require definitions of their own.

Most recently, the Next Generation Science Standards (NGSS) laid out a framework for identifying computational thinking in K-12 settings. As early as the fifth grade students are expected to be able to think computationally. They describe computational thinking, at this level, in terms of analyzing data and comparing approaches. By the time students reach middle school, computational thinking advances to analyzing large data sets and generating explanations. Finally, in high school, computational thinking expands to constructing computational models and using them to answer questions [12]. Clearly, computational thinking is a complicated concept which requires substantial explanation.

Experts in the field still have a ways to go when it comes to clearly defining computational thinking within physics education. However defined, though, this type of abstract and algorithmic thinking is pervasive – it extends beyond computer science into fields from geology to astronomy, and even beyond STEM [4]. It is becoming increasingly clear that "computational thinking is a fundamental skill for everyone, not just computer scientists [18]."

One such domain of investigation that could benefit from a clear definition of compu-

tational thinking is that of introductory physics. Many realistic physics problems require thinking abstractly and the computational power of computers (i.e., non-linear forces in Newton's second law). This combination of abstract and algorithmic thinking (computational thinking) is the "heart" of the "computational physics approach" to solving problems [1]. This approach relies heavily on computational thinking as it relates to a particular tool and on both technical and physics computational skills. The AAPT defines 3 technical skills and 7 physics skills. **DC: I think that you can expand on this in Ch 2. Include the table and discuss it.**

Many of the physics skills defined by the AAPT involve modeling and the modeling process. Computational modeling is a powerful problem solving tool. Research shows that computational modeling can be successfully incorporated into the (middle-divison) physics classroom [5]. There are a number of common mistakes students make when solving a Newtonian gravitational problem. We should encourage students to synthesize both analytic and computational skills [6]. **DC: I think you can expand on the prior work that studies computation in physics courses in Ch 2. There's also a number of references that you can use to describe the integration of computation into physics courses. They should appear here.**

Other research on (high school) computational modeling has focused on success. Students' ability to adapt to novel problems seems to rest on the ability to synthesize physics and computational skills [3]. The ability of students to understand the iterative process, that is characteristic of computation, plays a crucial role in the ability to construct novel computational force models.

Therefore, it is important to encourage computational thinking in introductory physics courses. **DC: I'm not sure how this follows. I want a little more of making an**

argument. You are describing what folks have done or said, but I want you to have a point from each of these that supports this statement.

1.3 Computational physics practices

The efficient construction of domain specific knowledge by students has always been one of the ultimate goals of physics education research. The fundamental concepts, ideas, and theories of introductory physics are the foundation for all inquiry in not just the field of physics, but in many related disciplines (e.g., chemistry and engineering). However, there is more to being a productive member of the scientific community than just amassing a collection of facts – it is important that this knowledge be applied in some practical way (e.g., utilizing Newton's second law to numerically predict the trajectory of a rocket). For this reason, the NGSS has broadly defined scientific practices as a combination of both knowledge and skill "to emphasize that engaging in scientific investigation requires not only skill but also knowledge that is specific to each practice [15]." **DC: This is a good paragraph**

Given the recent interest in scientific practices, and computational thinking more specifically, a taxonomy of the computational practices indicative of computational thinking has been defined [16]. This taxonomy, comprised of twenty-two individual yet inter-related practices, fitting into four different categories, is meant to help guide instructors and researchers as they attempt to teach and better understand computational thinking in science classrooms. Each practice, according to the taxonomy, is defined broadly so as to be applicable to a wide range of science classrooms.

However, the broad definitions that make the taxonomy widely applicable also leave it relatively vague and difficult to apply to any particular situation. Reducing the vagueness

and difficulty of applying this taxonomy to a specific domain of inquiry (i.e., introductory physics) is a challenging but important task. Having a taxonomy that is both precise and easy to apply will provide a solid foundation for instructors to generate/validate computational problems and for researchers to analyze the learning process. Accordingly, it is important that we identify, through direct observation, the set of computational practices that are common to computational introductory physics. This involves not only identifying the practices, but also the underlying knowledge and skills. **DC: I think this is where you want to say, and this is what this thesis is about: In Ch 2 we blah blah, and in 3 we blah blah blah, etc.**

Chapter 2

Background

2.1 Historical and Recent comp. research

2.2 Methods for my research

2.3 Framework

Chapter 3

Context

3.1 Learning Goals

P^3 uses a number of learning goals to direct pedagogy. These goals have been generated from two broad categories of focus: the practices we expect students to engage in and the content we hold them responsible for. Within each of these two broad categories the focus is further broken down into two sub-categories: the fundamental physics principles and the computational implementation of those principles.

3.1.1 Practices

A selection of the scientific practices which we have developed learning goals around are enumerated in no particular order below:

P1. developing and using models,

P2. planning and carrying out investigations,

P3. analyzing and interpreting data,

P4. constructing explanations,

P5. and engaging in argument from evidence.

These scientific practices were used to inform not only the shared learning goals but also the materials (i.e., projects, exams, homework) we use to achieve those goals.

One of the scientific practices used heavily on both analytic and computation days is that of (P1) developing and using models. Whether those models be mathematical or computational, we expect students to not only work together in groups to develop the model, but also to utilize that model in further investigations. As an example, this scientific practice was used to inform the learning goals of:

> Collect, analyze, and evaluate data to explain the motion of objects and the responsible interactions

and

> Evaluate the applicability/limitations of models and the validity of predictions for different types of motion.

This type of scientific practice (P1) and the associated learning goals were further used to generate the type of in-class project explicated in Sec. 3.2.

3.1.2 Content

Within the content of P^3 are many ideas that we expect students to grapple with both individually and within their group. A selection of the previously mentioned sub-categories that we focus on are enumerated below. The first focus is on the fundamental physics principles:

F1. macroscopic phenomena are the result of atomic interactions,

F2. forces external to a system can change the system's momentum,

F3. and work done on or by a system and heat exchanged with the system's surroundings can change the system's energy.

The second focus is on the computational implementation of those principles, which takes full advantage of the power of computation:

C1. computation can be used to predict the otherwise intractable dynamics of real-world phenomena,

C2. and computation can be used to generate dynamic and graphic representations of real-world phenomena.

Each of these sub-categories of focus were similarly used to inform the shared learning goals and materials used in P^3.

3.1.2.1 Physics

One such learning goal, which focuses on the fact that (F2) forces external to a system can change the system's momentum, is presented to the students as follows:

> Apply the momentum principle ($\Delta\vec{p} = \vec{F}_{\text{net}}\Delta t$; $d\vec{p} = \vec{F}_{\text{net}}dt$) analytically to predict the motion or determine the properties of motion/net force acting on a single-particle system where the net force is a constant vector (e.g., due to the near Earth gravitational force).

This goal focuses on the application of a fundamental physics principle (*the Momentum Principle*, $\vec{p}_f = \vec{p}_i + \vec{F}_{\text{net}}dt$) in order for students to better understand the dynamics of motion. This type of reasoning is used extensively in parts A and B of the type of in-class project described in Sec. 3.2.

3.1.2.2 Computation

The computational implementation of the above learning goal which focuses on how (C1) computation can be used to predict the otherwise intractable dynamics of a real-world system is presented to the students in the following way:

> Apply the momentum principle ($\Delta\vec{p} = \vec{F}_{\text{net}}\Delta t$; $d\vec{p} = \vec{F}_{\text{net}}dt$) iteratively/computationally to predict the motion or determine the properties of motion/net force acting on a single-particle system where the net force is not constant (e.g., due to spring-like restoring forces or dissipative drag forces).

This goal also focuses on the application of the Momentum Principle, yet takes full advantage of the computational power afforded to the students. This affordance shows up in the incorporation of a velocity dependent force as described in part C of the in-class project described in Sec. 3.2.

3.2 Example Project

In order to best illustrate the scientific practices students engage with in P^3, we present a typical problem requiring both analytical and computational techniques over the course of a week (two in-class meetings). During the second week of class, students are learning how to predict the motion of point particle systems using Newton's Second Law (*the Momentum Principle*, $\vec{p}_f = \vec{p}_i + \vec{F}_{\text{net}}dt$) in a project called Escape from Ice Station McMurdo.

In the first part of this three-part problem (Fig. 3.1), students work with position vs time data to model analytically the motion of two hovercrafts as they race across an ice field – determining the net force from this data and using appropriate models (constant velocity vs constant force) to predict when the hovercrafts will be at the same location. In the second

Project 2: Part A

You are a member of a scientific research team at McMurdo ice station which is funded by the Carver Media Group in Antarctica.

Two members of your research team have recently returned from investigating an incident at a Norwegian research facility. They brought with them a burnt humanoid body with two faces. Since the disturbing discovery several inhabitants of the ice station have disappeared. Frightened, a member of your team decided to flee the station on a fan powered hovercraft but you receive a distress call not long after their escape that their steering and acceleration controls have been jammed and they need your help.

Time	Team's Position	Your Position
0 s	2536.40 m	10.47 m
10 s	3072.80 m	41.88 m
20 s	3609.20 m	94.22 m

You decide to attempt a rescue in another hovercraft. You must decide how many members of your team help in the rescue operation. The hovercrafts do not have a velocity or acceleration gauge but they do have GPS locators and you possess your trusty stop watch. The GPS locator tells you the exact position of both your craft and other team members craft relative to the ice station. You are following their path. You collect the following data for the first 20 seconds of your journey.

You need to tell the runaway researcher the exact time from your starting time to jump onto your hovercraft as you may only have one shot at this rescue.

Figure 3.1: The first part of a project where students are asked to model the motion of an object given position vs time data.

Project 2: Part B

Just as you are about to radio the time to jump to the runaway researcher, you realize the steering and acceleration controls have become frozen on your hovercraft and so it continues to accelerate and you cannot change direction. 200m ahead of the point at which you were going to tell the researcher to jump is an ice ravine. At the bottom of the ice ravine, 400m below, is an unfrozen salt water pool surrounded by stalagmites. From the ravine's edge to the pool is 490m and the pool stretches for 900m. You are moving too quickly to survive jumping off the hovercraft, but might survive the fall into the pool by staying on the hovercraft; it has seat belts. You now have a choice to make, to stay on your hovercraft or jump to the runaway researcher's hovercraft. One or both may make it to the pool. Your choice may be the difference between life and death.

Figure 3.2: The second part of a project where students are asked to model the motion of an object under free-fall conditions.

part of Escape from McMurdo (Fig. 3.2), students find that the controls of both hovercrafts are frozen and they are heading towards a cliff. They must determine which hovercraft to board in order to survive the fall (there's a salty unfrozen pool at a specific distance from the bottom of the cliff). The generality and ubiquity of the Momentum Principle is highlighted in the third part of Escape from McMurdo (Fig. 3.3) where the students computationally model the motion of the hovercrafts including air drag.

As written above, the Momentum Principle can be used iteratively (through Euler-Cromer integration) to predict the motion of most systems accurately.[**?**] It turns out both hovercrafts will land safely when modeled without air drag, but students are told that one hovercraft crashes (it lands short). They must file an accident report, which includes a simulation of the accident, to explain how the accident occurred.

Given the complexity of the problems in P^3, a number of different learning scaffolds have been put into place. These scaffolds are meant to support the students in not only the

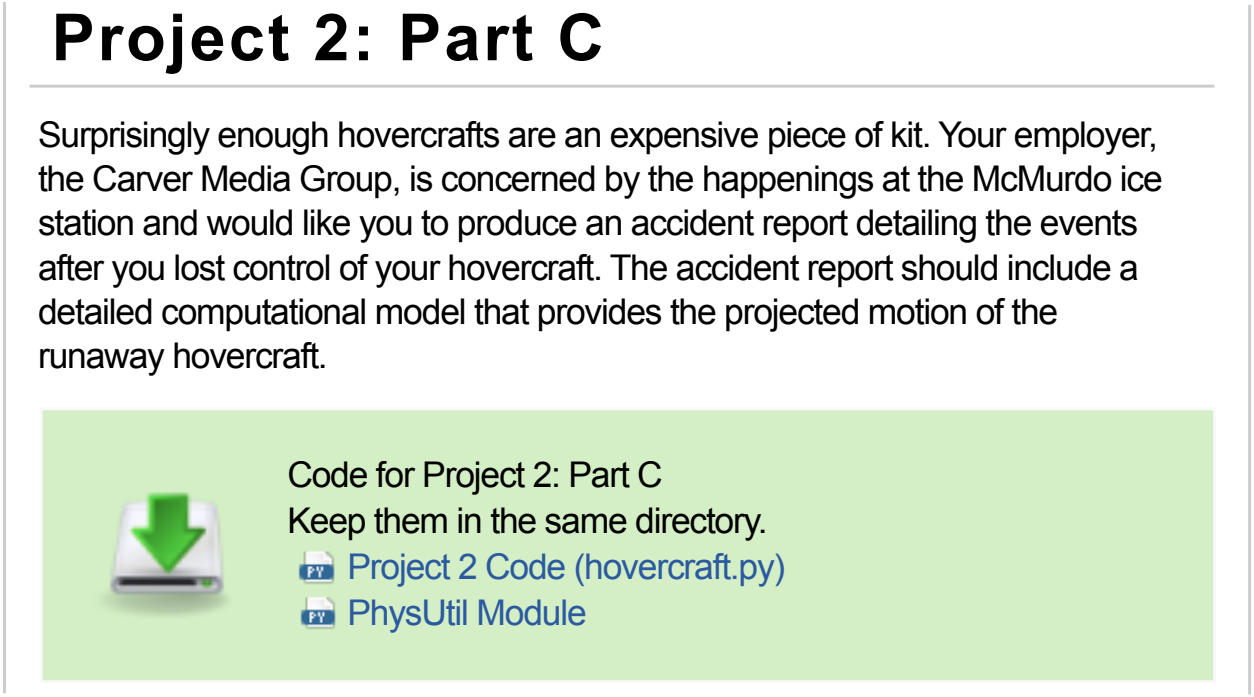


Figure 3.3: The third part of a project where students are asked to model the motion of an object computationally.

physics concepts they must use, but also in the scientific practices that they must engage in. Below, we expound on four such learning scaffolds.

3.3 Learning Scaffolds

3.3.1 Modeling Scaffolds

3.3.1.1 The Four Quadrants

To scaffold the modeling process (Fig. **??**), we have introduced a conceptual tool – the Four Quadrants. The Four Quadrants provide a designated location for students to record, reference, and update collectively agreed upon information during the modeling process. In a sense, the Four Quadrants form the basis of the model, and we provide a designated white board to each group for just this purpose. Using the Four Quadrants,

- students identify the **Facts**, which are presented in the problem statement;
- they determine what is **Lacking** from the information they have or can obtain easily;
- they discuss and negotiate the **Assumptions & Approximations** they are making; and
- they provide **Representations** of the problem, which may include diagrams, graphs, or equations.

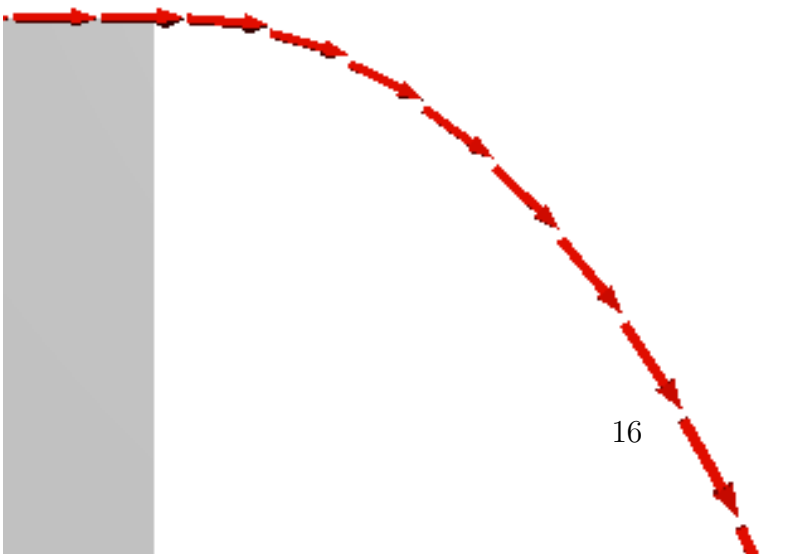
The Four Quadrants are displayed so that not only each member of a group can easily see and modify the agreed upon information, but also so that a group's tutor can easily "ping" the group to see where they are in the solution process. That is, without needing to disrupt the solution process, a tutor can get a well-rounded idea of what the group is thinking and where the group is heading. This is particularly useful as many tutors are working with multiple groups at any given time.

3.3.1.2 Minimal Working Programs

Many problems in P^3 require the computational modeling of motion and the generation of dynamic plots. To keep up with the increasing complexity of the problems, students must learn to write VPython code. However, less than 10 % of students taking P^3 have any significant computational experience. We have extended the work of Weatherford[**?**] and Lunk,[**?**] who introduced the concept of "Minimal Working Programs" (MWPs) to scaffold student sense-making about computing, and to develop a unique, group-oriented, inquiry-based instructional model for computation. This use of MWPs not only encourages sense-making, but also helps to assuage the anxiety many students have about engaging in computation for the first time. In this way, we further scaffold the (computational) modeling process.

In P^3, students are given no explicit instruction on VPython. Rather, on computational

```
#Objects
hovercraft = sphere(pos=vector(-200,400,0), radius=1)

#Parameters and Initial Conditions
g = vector(0,-9.81,0)

hovercraftm = 1500
hovercraftv = vector(53.64,0,0)
hovercraftp = runawaycraftm*runawaycraftv

#Time and time step
t = 0
tf = 10
dt = 0.01

#MotionMap/Graph
hovercraftMotionMap = MotionMap(hovercraft, tf, 5)

#Calculation Loop
while hovercraft.pos.x < 0:
    Fgrav = hovercraftm*g
    Fground = -Fgrav
    Fnet = Fgrav + Fground

    hovercraftp = hovercraftp + Fnet*dt
    hovercraft.pos = hovercraft.pos + (hovercraftp/hovercraftm)*dt

    hovercraftMotionMap.update(t, hovercraftp/hovercraftm)

    t = t + dt
```



days, students are provided with a MWP that already predicts the motion of some aspect of the problem (e.g., the motion of a hovercraft up to a certain point on a cliff, as seen in Fig. 3.4). This experience is similar to receiving "user-developed" code from a colleague and extending it to a new situation – a common practice in science and engineering labs. By engaging in discussion and negotiation – two essential science and engineering experiences – students develop an understanding of what the program is doing and how it is doing it. They then use that understanding to modify/write additional program statements to model the situation in question. As a result of this instruction, students having no prior experience with computational modeling are able to write essential elements of the VPython code needed to model novel situations.

We further scaffold this process by providing students with comments – the gray "hash-tagged" (#) statements in Fig. 3.4 that are used consistently and repeatedly in each computational modeling problem. This common thread throughout MWPs helps to orient the student across many instances of computational motion problems. Typically, the number of additional program statements students will write is less than 10; they are focused on the core aspects needed to model the system (i.e., motion prediction and visualization).

For example, in the code appearing in Fig. 3.4, students would write several additional lines of code to model the falling hovercraft. This includes writing a second `while` loop that stops once the vertical position of the hovercraft coincides with the water, representing force calculations for the air drag and gravitational force in VPython code, and implementing the motion prediction algorithm that performs the appropriate Euler step.

3.3.2 Content Scaffolds

3.3.2.1 Conceptual Homework

Much of the student experience in P^3 is working through complex problems that require discussion and negotiation among group members to develop a complete solution. To support student success when solving these problems in class, we "prime the pump" with pre-class readings/video lectures and homework. Each week, students read online lecture notes, watch short video lectures, and solve conceptual and reasoning-focused homework on-line, which are meant to scaffold students' conceptual understanding that they will bring to bear in class that week.

Often, solutions to computational problems can take many syntactical forms, while the underlying computational algorithm is identical. That is, VPython has a number of functions to simplify the typing/reasoning that any group of students must undertake. For example, as shown in Fig. 3.5, when taking the magnitude of a vector `object.pos`, two possible methods accomplish the same goal: the "manual" way of explicitly squaring, summing, and square rooting the vector componentits `sqrt(object.pos.x**2+obj.pos.y**2+obj.pos.z**2)` and the VPython short-cut way of `mag(obj.pos)`. Given the diverse computational background of students in P^3, we highlight these differences/similarities in the pre-homework to scaffold the discussion and solution process.

After each pair of weekly class meetings, students complete additional conceptual and reasoning-focused homework as well as more typical back-of-the-book style problems. These are designed to provide a wrap-up of the week's material and to encourage the students to take full advantage of the opportunity to engage in group discussion and sense making in class.

9. Calculating a unit vector in VPython

Students in your class are continuing to model the motion of Triton (one of Neptune's 13 moons) around Neptune, but now using VPython. The code your class has received contains the following snippet of VPython code.

```
Neptune = sphere(pos=vector(100,200,300), radius=1)
Trion = sphere(pos=vector(10,20,30), radius=2)
```

(a) From this snippet, which of the following lines of code might your group write to describe the separation vector pointing from Neptun to Triton?

- ○ `rvec = Triton.pos - Neptune.pos`
- ○ `rvec = Neptune.pos - Triton.pos`

(b) Several groups have written different lines of code to calculate the magnitude of the separation vector; some are correct and some are not. From your understanding of the line(s) of code below, which of them correctly represent the magnitude of the separation vector?

- ☐
```
rmag = mag(Neptune.pos) - mag(Triton.pos)
```
- ☐
```
rmag = mag(Triton.pos - Neptune.pos)
```
- ☐
```
rmag = sqrt((Triton.pos.x - Neptune.pos.x)**2
           +(Triton.pos.y - Neptune.pos.y)**2
           +(Triton.pos.z - Neptune.pos.z)**2)
```
- ☐
```
rmag = sqrt((Neptune.pos.x - Triton.pos.x)**2
           +(Neptune.pos.y - Triton.pos.y)**2
           +(Neptune.pos.z - Triton.pos.z)**2)
```
- ☐
```
rmag = mag(Neptune.pos - Triton.pos)
```
- ☐
```
rmag = mag(Triton.pos) - mag(Neptune.pos)
```

Figure 3.5: Computational pre-homework problem focusing on the different ways to construct a unit vector in VPython.

3.3.2.2 Tutor Questions

With such open ended projects as shown in Sec. 3.2, there are many possible solution paths that lead to the same answer, and the course staff works to encourage any solution path that utilizes the concepts and ideas being focused on for that week. In order to ensure a group's understanding of the content being focused on, we have introduced a compilation of questions which are meant to confront common misconceptions and to test for basic understanding, as exemplified in Fig. 3.6.

The generation of these tutor questions is a continuous and iterative process. Initially, questions are generated by consulting with the literature and by "testing" the project. As the projects are delivered to students in-class, new solution paths inevitably arise with unique lines of reasoning. These new lines of reasoning generate new tutor questions, which are added to the compilation. We aim to compile a robust set of tutor questions spanning many different solution paths.

The nature of the pre-class homework mentioned previously is largely conceptual, and often requires an answer of the free-response type. Given that these homeworks are delivered using an electronic web-based system, answers may be easily collected and analyzed by course staff to highlight any particularly difficult topics. These topics may then be used by the tutors as "just-in-time" style talking points during class. In this way, the tutor is not only informed of which topics to check for understanding on, but also which students may need a little extra attention.

Tutor Questions

- **Question:** What assumptions did you make about the motion of the hovercrafts?
- **Expected Answer:** That the runaway craft has a constant velocity, and the rescue craft starts from rest with a constant acceleration.

- **Question:** Are these velocities and accelerations calculated from the numbers given exact?
- **Expected Answer:** No, these are only average numbers, not instantaneous. In order to get more "exact" numbers, we would need more data.

- **Question:** Is the predicted position of the rescue craft a good one?
- **Expected Answer:** Not really, basing the trajectory off the first 20 seconds of data is probably not the best – but it is all we have to work with.

- **Question:** Can you draw a plot of position vs. time for both crafts? What are the important features of this graph?
- **Expected Answer:** The point where the two curves cross is when we should jump. One should be linear, the other quadratic.

- **Question:** Can you draw a plot of velocity vs. time for both crafts? What are the important features of this graph?
- **Expected Answer:** The acceleration is the slope of each curve (constant in both cases).

Figure 3.6: A selection of the tutor questions that are asked during the hovercraft problem.

Chapter 4

Motivation for identifying broader practices

4.1 Debugging

4.1.1 Introduction

Since the development of reasonably affordable and fast computers, educators have argued for their inclusion in the classroom as both a learning aid and a tool.[?, ?] With the recent development of high-level programming languages capable of quickly rendering three-dimensional real-world simulations, the argument for the inclusion of computers in the classroom has not only persisted, but has grown.[8] Well into the 21st century, with its prevalence in modern physics research, we see computation being increasingly referred to as the "third pillar" of physics – along with the more canonical pillars of theoretical and experimental physics.[7]

One physics curriculum that includes a computational element is Matter & Interactions (M&I). The M&I curriculum differs from a traditional one not only through the inclusion of computation (VPython), but also through its emphasis on fundamental physics principles and the addition of a microscopic view of matter.[?, ?] Recent work[10, ?] involving students' use of VPython with the M&I curriculum has begun to analyze the patterns seen in implementing and assessing the use of computation in introductory physics courses.

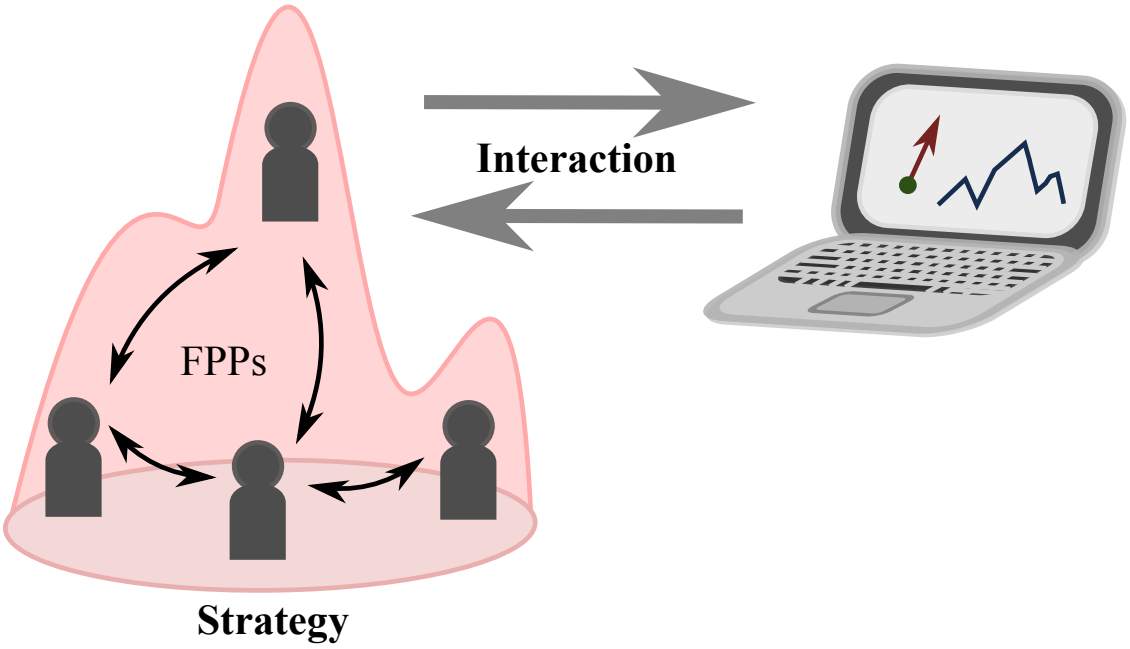
However, numerous questions remain unanswered regarding the processes observed while groups of students work together to model real world phenomena computationally. We extend our research to a novel implementation of M&I with an emphasis on computation in a group setting, called Projects and Practices in Physics (P^3), where students negotiate meaning in small groups, develop a shared vision for their group's approach, and employ science practices to navigate complex physics problems successfully. Borrowing from the literature of computer science education research,[11] we use the notion of computational debugging in a physics context to help uncover the salient practices unique to computational physics problems.

In this paper, we present a case study of a group of students immersed in this P^3 environment solving a computational problem. This problem requires the translation of a number of fundamental physics principles into computer code. Our analysis consists of qualitative observations in an attempt to describe, rather than generalize, the computational interactions, debugging strategies, and learning opportunities unique to this novel environment.

4.1.2 Assumptions

In an increasingly technological world, we have at our disposal computers well suited for the most procedural and tedious, yet indispensable of STEM tasks. Accordingly, in P^3, we treat the computer as an indispensable *modern tool* with which students must familiarize themselves. In spite of the fact that modern computers *take* meaningful direction quite well, they do not yet posses the faculty to *generate* meaningful direction on their own. This necessitates a human factor, in which groups of students must leverage their understanding of fundamental physical principles to model real world phenomena computationally (i.e., generate meaningful direction to a computer).

Figure 4.1: A group of students interacting with VPython where social exchanges focus on FPPs and desirable strategies are being exhibited.



We focus our research on the interactions between group and computer to begin to understand the ways in which computation can influence learning. Particularly, we are interested in the interactions occurring simultaneously with social exchanges of fundamental physics principles (FPPs) specific to the present task (e.g., discussing $d\mathbf{r} = \mathbf{v}\,dt$ on a motion task) and the display of desirable strategies (e.g., divide-and-conquer), as illustrated in Fig. 4.1. These group-computer interactions vary in form, from the more interactional process of actively sifting through lines of code, to the less interactional process of observing a three-dimensional visual display – in this work, the more the student interacts with lines of code, the more interactional.

One previously defined computational interaction that reinforces desirable strategies,[**?**] borrowing from computer science education research, is the process of debugging. Computer science defines debugging as a process that comes after testing *syntactically* correct code where programmers "find out exactly where the error is and how to fix it."[11] Given the

generic nature of the application of computation in computer science environments (e.g., data sorting, poker statistics, or "Hello, World!" tasks), we expect to see unique strategies specific to a computational *physics* environment. Thus, we extend this notion of computer science debugging into a physics context to help uncover the strategies employed while groups of students debug *fundamentally* correct code that produces unexpected physical results.

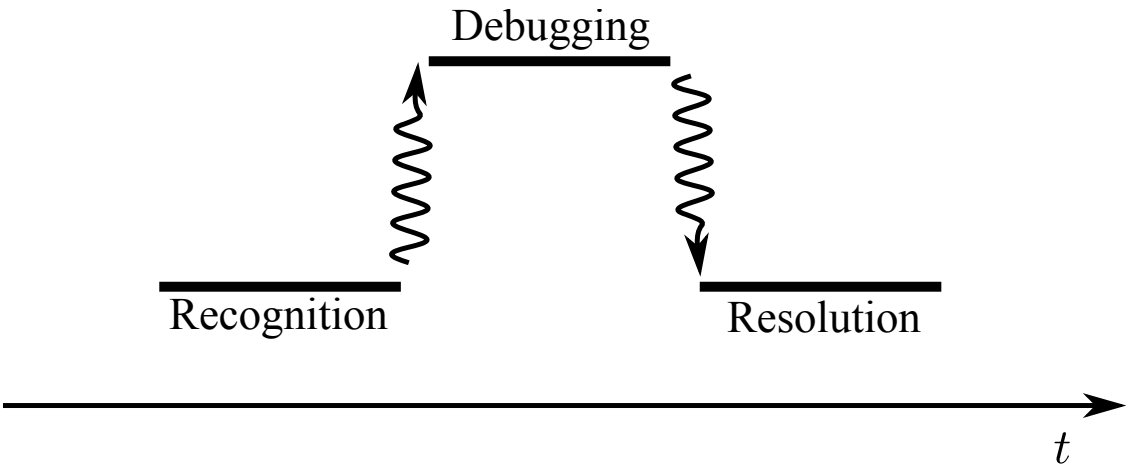
4.1.3 Data

In Fall 2014, P^3 was run at Michigan State University in the Physics Department. It was this first semester where we collected *in situ* data using three sets of video camera, microphone, and laptop with screencasting software to document three different groups each week. From the subset of this data containing computational problems, we *purposefully sampled* a particularly interesting group in terms of their computational interactions, as identified by their instructor. That is, we chose our case study not based on generalizability, but rather on the group's receptive and engaging nature with the project as an *extreme case*.[**?**]

The project that the selected group worked on for this study consists of creating a computational model to simulate the geosynchronous orbit of a satellite around Earth. In order to generate a simulation that produced the desired output, the group had to incorporate a position dependent Newtonian gravitational force and the update of momentum, using realistic numerical values. The appropriate numerical values are Googleable, though instructors encouraged groups to solve for them analytically.

This study focuses on one group in the fourth week of class (the fourth computational problem seen) consisting of four individuals: Students A, B, C, and D. The group had primary interaction with one assigned instructor. Broadly, we see a 50/50 split on gender, with one ESL international student. Student A had the most programming experience out

Figure 4.2: The debugging process necessarily corresponds to a phase beset on either side by the phases of recognition and resolution. Note the absence of a vertical scale, as the vertical separation merely acts to distinguish phases.



of the group. It is through the audiovisual and screencast documentation of this group's interaction with each other and with the technology available that we began our analysis.

4.1.4 Analysis

To focus in on the group's successful physics debugging occurring over the 2 h class period, we needed to identify phases in time when the group had recognized and resolved a physics bug. These two phases in time, *bug recognition* and *bug resolution* are the necessary limits on either side of the process of *physics debugging*, as represented in Fig. 4.2. We identified these two bounding phases at around 60(5) min into the problem, and further examined the process of debugging in-between. That is, we focused on the crucial moments surrounding the final modifications that took the code from producing unexpected output to expected output.

4.1.4.1 Bug recognition

At around 55 min into the problem, following an intervention from their instructor, the group began to indicate that they were at an impasse:

```
SB:  We're stuck.
SD:  Yeah...
```

The simulation clearly displayed the trajectory of the satellite falling into the Earth – not the geosynchronous orbit they expected. This impasse was matched with an indication that they believed the FPPs necessary to model this real world phenomenon were incorporated successfully into the code:

```
SB:  And it's gonna be something really
     dumb too.
SA:  That's the thing like, I don't think
     it's a problem with our understanding
     of physics, it's a problem with our
     understanding of Python.
```

Instead of attributing the unexpected output with a mistake in their understanding or encoding of FPPs, they instead seemed to place blame on the computational aspect of the task.

During this initial phase, we see a clear indication that the group has recognized a bug – there is an unidentified error in the code, which must be found and fixed:

```
SA:  I don't know what needs to change
     here...
```

SD: I mean, that means we could have
like anything wrong really.

Although they have identified the existence of the bug, they still are not sure how to fix it – this necessitates the process of debugging.

4.1.4.2 Physics debugging

Within the previously identified phase of bug recognition, the group developed a clear and primary task: figure out exactly how to remove the bug. Eventually, following a little off-topic discussion, the group accepted that in order to produce a simulation that generates the correct output, they must once again delve into the code to check every line:

SA: ...I'm just trying to break it down
as much as possible so that we can
find any mistakes.

In this way, the group not only determined the correctness of lines of code that have been added/modified, but also examined the relationships *between* those lines.

For example, the group began by confirming the correctness of the form of one such line of code:

SA: Final momentum equals initial momentum
plus net forces times delta t. True?
SC: Yeah...
SB: Yes.
SA: O.K. That's exactly what we have
here. So this is not the problem.

```
     This is right.
SD:  Yeah.
```

That is, Student A *(i)* read aloud and wrote down the line of code $\vec{p}_f = \vec{p}_i + \vec{F}_{\text{net}}\Delta t$ while the entire group confirmed on its correct form. This written line was then boxed, and was shortly followed up *(ii)* with a similar confirmation of the line $\vec{r}_f = \vec{r}_i + \vec{v}\Delta t$ that immediately prompted *(iii)* the confirmation of $\vec{v} = \vec{p}/m$. Thus, not only do we see the group determining the correctness of added/modified lines of code as in *(i)*–*(iii)*, we further see confirmation with the links *between* those lines, as in the velocity from *(ii)* being updated through the momentum from *(i)* given their relationship from *(iii)*.

The group ran through these types of confirmations with FFPs rapidly over the span of a few minutes. Once the group had confirmed all the added/modified lines of code to their satisfaction, the discussion quieted down. The FPPs were winnowed from the discussion, and after a little more off-topic discussion we find them seeking help from the instructor:

```
SD:  Maybe we should just stare at him
     until he comes help us...
```

Suddenly, a haphazard change to the code:

```
SA:  You know what, I'm gonna try
     something.
```

where Student A changed the order of magnitude of the initial momentum a few times. This modification eventually resulted in a simulation that produced the correct output.

4.1.4.3 Bug resolution

At 65 min into the problem, Student A changed the order of magnitude of the momentum one final time, which produced something *closer* to the output that they expected:

```
SA:  Oh wait... Oh god...

SD:  Is it working?
```

The satellite now elliptically orbited the Earth. This marks the end of the debugging phase and the beginning of the resolution phase – the bug had successfully been found and remedied. Given that the only line of code modified to produce this change was the initial momentum, they began to rethink the problem:

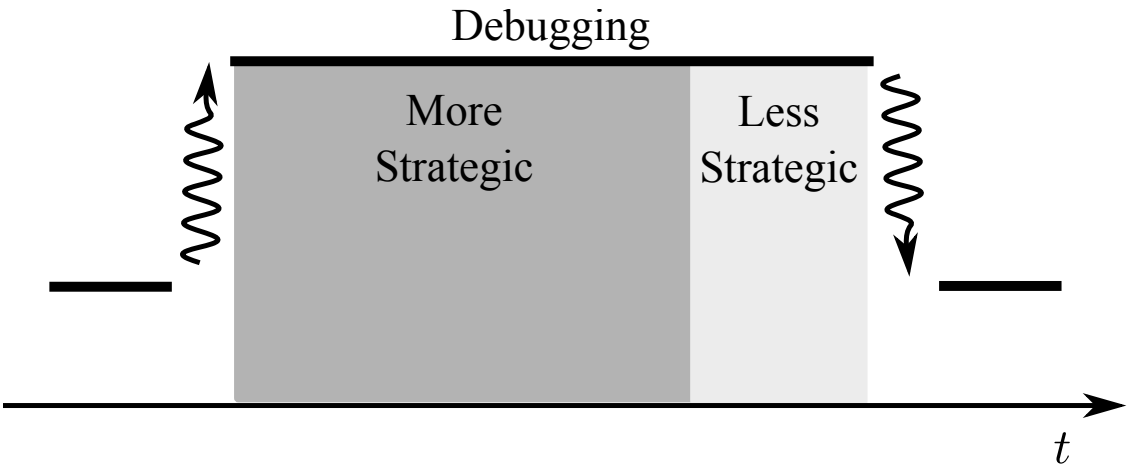
```
SD:  I think that is the issue is that
     we don't have the initial momentum...
SA:  ...momentum correct?
```

That is to say, the group pursued the issue of determining the correct initial momentum with the added insight gained through debugging fundamentally correct VPython code.

4.1.5 Discussion

To summarize, in analyzing this particular group, we first identified the two phases in time when the group had recognized and resolved (see Fig. 4.2) a physics bug. We then necessarily identified the phase in-between as the process of physics debugging in P^3, where the fundamentally correct code was taken from producing unexpected output to producing expected output. Given our assumption that the process of computer science debugging encourages desirable strategies, we then began to analyze this process of physics debugging further for strategies unique to P^3.

Figure 4.3: Physics debugging phase consisting of more and less strategic strategies specific to our case study. Note the absence of a vertical scale, as the vertical separation merely acts to distinguish phases.



Given the actions exhibited during the debugging phase, we can separate them into two distinct parts: a more strategic part and a less strategic part, as shown in Fig. 4.3. The group initially gave indication that they were working in a considerate, thorough, and consistent manner, which we classify as more strategic. This is contrasted by the later indications of more haphazard actions, which we classify as less strategic. These are the two physics debugging strategies that, together, led to the resolution of the bug in this context.

The more strategic strategy was exhibited through the confirmation of individual FPPs as well as their relation to others. Not only did the group confirm through discussion, they simultaneously wrote, boxed, and referenced equations in the code – this helped to reduce the number of FPPs they needed to cognitively juggle at any given time.[**?**] This confirmation of FPPs through discussion presented a great learning opportunity for the entire group, where creative and conceptual differences could be jointly ironed-out. Accordingly, we tentatively refer to this strategy as **self-consistency**.

Although the resolution of the bug might not be tied directly to this self-consistency, that does not negate the learning opportunities afforded to the group along the way. Specif-

ically, we saw the group double-checking every fundamental idea used and, possibly more importantly, the links between those ideas. Being physically self-consistent in this manner is a desired strategy in P^3.

The less strategic strategy was exhibited during the haphazard changes to the initial momentum. These changes to the code that eventually resolved the bug, though one of the benefits of computation (i.e., the immediacy of feedback coupled with the undo function), could have been more thoughtful. A deeper understanding of the physics or computation could have tipped the group off to the fact that the initial momentum was too small. Again, this does not negate the learning opportunities afforded to the group through this less strategic strategy, which resembles that of "productive messing about."[**?**] Accordingly, we tentatively refer to this strategy as **play**.

Both of these strategies identified here, self-consistency and play, hold implications for the learning opportunities afforded to groups. More research is needed to dissect these learning opportunities and to deepen our understanding of the strategies themselves.

4.1.6 Conclusion

This case study has described two strategies (one more and one less strategic) employed by a group of students in a physics course where students develop computational models using VPython while negotiating meaning of fundamental physics principles. These strategies arose through the group's process of debugging a fundamentally correct program that modeled a geosynchronous orbit. The additional data we have collected around students' use of computation is rich, and further research is needed to advance the depth and breadth of our understanding of the myriad of ways in which students might debug computational models

in physics courses.

4.2 Phenomenography

4.3 Task Analysis

4.3.1 Introduction

Given the ever increasing complexity of modern scientific problems, computation is essential practice that physics and engineering majors must learn [**?**, **?**]. However, integrating computation into physics courses does not come without its challenges for both instructors and students [**?**]. Currently, researchers understand little about the challenges that students face when connecting computational practices with physical concepts. Thus, it is important that we not only identify these challenges, but also investigate the way in which students overcome them.

The study presented in this paper takes place in a practice focused group learning environment at Michigan State University called P-cubed [**?**]. In this environment students solve analytical and computational mechanics problems (using VPython) in small groups. For computational problems, we make use of minimally working programs – incomplete but functional programs that encourage students to focus on the fundamental physics of the phenomena, rather than getting bogged down in code syntax and program structure [**?**].

As part of a larger project investigating the challenges that students experience with computation in introductory physics classrooms, we conducted a task analysis on a computational problem that we have observed students struggling to complete over the last several semesters. This gravitational orbit problem is delivered in the third week of class

and requires groups to construct a position dependent Newtonian gravitational force that is fully generalizable (i.e., capable of handling elliptical orbits). We used task analysis to identify specific sub-tasks that students must complete in order to solve the problem. We then identified clips from in-class video where students are engaging in those tasks in order to describe the different lines of reasoning that students are using while solving the problem. Due to the limit of space, our analysis presented here is restricted to the construction of the direction of the gravitational force.

4.3.2 Methods

Our primary source of data comes from in-class video of students developing a VPython program to model a gravitational orbit. In order to focus our analysis on specific challenges students demonstrate while working this problem, we developed a framework for analyzing this problem by performing an expert task analysis on the problem. A task analysis consists of breaking a complex "task" (e.g., modeling a gravitational system) down into related "sub-tasks" (e.g., determine the direction of the force). We aim to identify somewhat more manageable steps that must be taken in order to complete the overall task [**?**].

The task analysis of this problem was initially constructed by a single content expert. After the first iteration it was presented to additional experts. Through this discussion, it became clear that the construction of the position dependent Newtonian gravitational force in code is a multi-step procedure involving a number of different sub-tasks. The task analysis was iteratively refined through this process until all experts agreed that the sub-tasks were sufficiently described to be useful in video analysis. At the same time, targeted pre-class homework problems were developed to help scaffold students overcoming what we perceived to be challenges based on the task analysis. In doing so, we attempted to place

(most) students in the Zone of Proximal Development (ZPD) [**?**]. For example, the pre-class homework problems shown in Fig. 4.4 were developed to facilitate student understanding of the unit vector of a separation vector between two objects prior to working the computational problem.

Table 4.1: Some of the necessary steps that must be taken when constructing a Newtonian gravitational force in code. Each step is associated with the construction/modification of a line of code.

Step (Sub-Task)	**Associated Code**
Construct separation vector between interacting objects	`sep = obj2.pos - obj1.pos`
Construct the unit vector	`usep = sep/mag(sep)`
Construct the net force vector	`Fnet = -G*m1*m2*usep /mag(sep)**2`
Integrate the net force over time into momentum	`obj.p = obj.p + Fnet*dt`

With an agreed upon task analysis, we further focused our video analysis on the sub-tasks that were closely related to the construction of the position dependent Newtonian gravitational force (see Table. 4.1). In order to identify when groups were actually engaging in these sub-tasks, we analyzed groups on two levels: their *speech*, including any and all words or utterances, and the associated *visuals*, including any drawings, writings, lines of code, gestures, or simulation visualizations. Theses two levels were then used to better understand how students overcome the challenges associated with the problem.

4.3.3 Analysis

There were a number of challenging tasks undertaken by students that were identified in the video. For the sake of clarity within the allotted space, we will focus on one sub-task: the construction of the *direction* of the Newtonian gravitational force in code. This was one of more challenging sub-tasks with which groups struggled. It requires the group, depending

on how they break the problem down, to:

1. define the separation vector between the satellite and the Earth,
2. construct its unit vector, and
3. use that unit vector to give a direction to the force.

It is important that all of these definitions be placed in the calculation loop of the MWP given that they must change over (simulated) time.

Below, we present three episodes of three different groups engaging with the problem described in Sec. 4.3.1. We have paid particular attention to the lines of reasoning that the students are taking while articulating their solutions.

4.3.3.1 Episode: Group A

Group A is comprised of Cody, Chuck, Shelley, and Joe. Group A has an equal gender distribution and Chuck has the most programming experience (and does most of the coding). This episode begins at around half way into the class after a tutor interaction where the group is dissuaded from using a centripetal form (i.e., mv^2/R) of the net force. Rather, they decide to use a Newtonian form (i.e., $F_{\mathrm{G}} = Gm_1m_2/R^2$) so that it will be able to handle elliptical orbits.

Shelley: *But ummm wait, hold on, remember this? The uniform circular is equal to the gravity is equal to the net? So we could just do what you did, except instead of using the uniform circular motion equation we use that gravity equation [points to equation].*

Joe: *Yeah...*

Chuck: *Okay, yeah, that sounds good.*

However, after coding the magnitude of the Newtonian gravitational force (i.e., `Fnet =`

`G*mEarth*msat/R**2`) the group is still unsure of how to add in the direction.

Chuck: *How do we, okay, how do we define a direction?*

Cody: *I don't know...*

Chuck: *Isn't the direction like, okay, so here I'm gonna give like four points on a circle [drawing on whiteboard] so this is the center, and this is a b c and d. Isn't it always just the position vector of a, so ummm what is it, like satellite dot position minus position dot Earth, and then you can divide that by magnitude?*

At this point, the group has worked out a plan to construct a unit vector that can be used to add onto the magnitude of the Newtonian gravitational force to give it the proper direction. They successfully follow through with this plan after comparing the unit vector at multiple points along a unit circle.

Chuck: *Okay, so we're going e minus a, so it's just negative, it's always negative e, right? What if we pretend it was here, what is it doing, now would it be like, x is point five... I think, okay so, arbitrary like it's not on the circle, so it would be like point five in the x direction, point five in the y direction, and zero in the z direction. So you're still taking that minus that and the direction would be negative points five negative point five, so I think that would still apply, if we are taking c minus a, and then divide by magnitude, would that make sense?*

Cody: *So how does that apply to this? How can we apply that concept?*

They then construct this in the code taking advantage of the built in `mag()` function to calculate a magnitude of a vector.

```
R = 123
[calculation loop]
```

```
dir = sat.pos/mag(sat.pos)
Fnet = -G*m1*m2*dir/R**2
```

Although the magnitude of the net force is not generalizable enough to handle elliptical orbits at this point (the distance `R` is defined outside of the calculation loop), they have successfully added in the direction of the Newtonian gravitational force. It is important to note that Chuck uses a procedure that is very similar to what is required on one of the pre-class homework problems shown in Fig. 4.4. In both cases, comparing multiple points along a unit circle can help to develop a sense for the direction of a separation vector and its use in giving a direction to a force.

4.3.3.2 Episode: Group B

Group B is comprised of Kayla, Jon, Angela, and Alan. Group B has an equal gender distribution and Alan has the most programming experience. This episode begins at about a quarter of the way into the class with the group agreeing to use a centripetal form of the gravitational force that they just so happen to have a vector equation for (i.e., $\vec{F}_{\text{net}} = mv^2\langle \sin\theta, \cos\theta - 1\rangle / R\theta$).

Kayla: *So, I have the uniform circular motion up... [scanning notebook] Ummm, magnitude of net force... uniform... Ummm...*

Alan: *We should calculate the force, the "centripetal force", and then use that number?*

Kayla: *Yeah...*

Alan: *Sounds good. What are the equations for that?*

Kayla: *Oh wait, hey, look at that, okay, these ones have, they are vector components ummm*

[writing on whiteboard].

The group is unsure of the validity of the equation that used the satellite's polar angle (θ), and Alan begins to consider how the velocity will have a different direction at various points along a unit circle.

Alan: *So like right here... like we would have a total velocity like going in the x direction or from here in the x direction and then right here going down so now it's all in the y direction there's no x direction, it's like oscillated between the maximums so we could get a rotation... but how do we represent that as a force means it's obvious that they want us to do that so do we...*

Immediately, Kayla proposes the equation from her notebook again. The group is convinced that this equation will lead to the expected oscillation of the direction of the velocity.

Kayla: *Sine and cosine?*

Alan: *Sine and cosine?*

Kayla: *If we do sine and cosine, if we do both of them one in the x, one in the y, like this is saying [points to equation], then even if one goes to zero like you were saying then the other one is gonna be close to one and so...*

Alan: *Do we have our angles to use?*

Angela: *Ohhh...*

Alan: *And we have access to angles... because we have access to the time of day.*

Angela: *That would make sense cause then if you're at some weird angle because like...*

Kayla: *Yeah...*

Angela: *Right here one of them is gonna go to one, the other one is gonna go to zero like...*

Bruce: *Oh my god that's so great, that's perfect, you're totally right.*

Indeed, they have all the necessary components needed to quickly construct this calculation in their code (i.e., the angle that the Earth is making).

Although this group is making progress toward a net force that changes with position, they still have several steps to complete this angular calculation. However, before they can construct any significant piece of code, the group is dissuaded by a learning assistant from using the polar angle of the satellite and are encouraged to use a unit vector in terms of a separation vector vector and its magnitude. It is important to note that, like Group A, Group B was also comparing multiple points along a unit circle in order to develop a sense for the way in which a physical quantity (i.e., the velocity of the satellite) needs to change direction.

4.3.3.3 Episode: Group C

Group C is comprised of Tori, Natalia, Connor, and Alex. Group C has an equal gender distribution, with one ESL student, and Connor has the most programming experience. The episode begins at around three-quarters of the way into the problem. We can see Connor comparing the "radius" of the satellite's orbit at two different locations, one point on the perimeter of a unit circle and one point beyond it.

Connor: *Well, if we start like that [drawing on whiteboard], but then we move over say here where it isn't equal radius, then our y value changes right?*

Tori: *Yes.*

This immediately turns into Tori reasoning out the way to calculate a separation vector

between two objects.

Tori: *So ummm, I thought, so the Earth is at zero zero zero, so in all of our homeworks, what we did was, the Earth is zero zero zero and then we have a position of the satellite at ummm...*

Connor: *4210...*

Tori: *Yeah, like that zero, zero. So when you subtract this, it is like, you know what I mean? We need one going from here to there, which would mean that we need to take this one minus this one so this one, minus that, which is just that, because you're just taking this minus zero.*

Again, we see Connor comparing a physical quantity at two different locations, and Tori explicitly references the pre-class homework.

Connor: *But then say we are up here [points to whiteboard]...*

Tori: *Yeah, and I think because in all the homeworks we have done this one was for like, oh, this is from okay a planet to a star, but they never talked about that moving in circular orbit...*

This discussion eventually leads to the construction of the net force in code that uses the separation vector between the Earth and the satellite.

```
R = 123
[calculation loop]
    Fnet = -const_G*m1*m2/R**2 * sat.pos
```

Although the magnitude of the gravitational force is still incorrect at this point, they have successfully incorporated one of the important components for the direction of the net force in code (i.e., the separation vector `-sat.pos`). It is important to note that Tori explicitly

reference the pre-class homework problem shown in Fig. 4.4 when reasoning out the direction of the gravitational force. Also, Connor can be seen running through a very similar process to that of Groups A and B (i.e., comparing physical quantities at different locations).

4.3.4 Discussion & Conclusions

We have shown the lines of reasoning that three different groups have taken while working to overcome the challenges associated with constructing a position dependent Newtonian gravitational force in VPython code. Due to the brevity of this paper, we focused on the sub-task of constructing the *direction* of the Newtonian gravitational force. In our analysis, we saw three separate cases of groups comparing the change in a physical quantity (e.g., the gravitational force or the velocity) at multiple points along and near the perimeter of a unit circle. In each case, while groups made these comparisons, we also saw them beginning to construct the direction of the Newtonian gravitational force: Groups A and C added in a direction based on a separation vector, and Group B started to add a direction based on a polar angle.

These episodes suggest that the pre-class homeworks shown in Fig. 4.4 are providing appropriate scaffolding for students [**?**]. Group A explicitly compares the direction of the net force at four different points on the unit circle. Group B compares the direction of the satellite's momentum at two different points on the unit circle. Group C compares the distance of the satellite from the origin at two different points on and near the unit circle. It appears that asking students to solve related yet sufficiently different problems can familiarize students with the important concepts so that they can be extended into novel in-class problems.

In an effort to broaden this investigation, we have begun conducting one-on-one post-

class interviews to investigate the qualitatively different ways in which students experience introductory physics with computation. These interviews show promising descriptions of the reasoning taking place around computational physics tasks. For example,

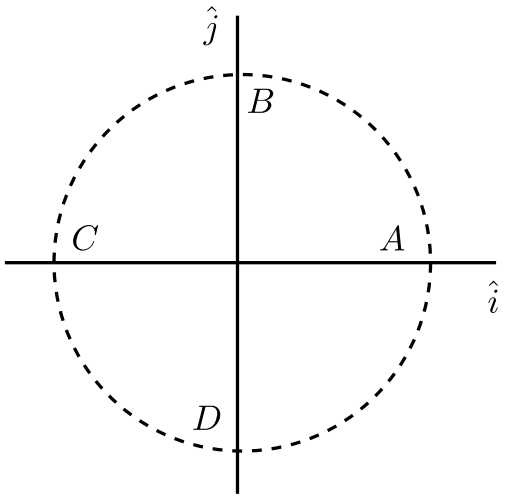
Ski: *This [gravitational] problem especially showed me why it's important to update the force, the direction of the force. Going into this, I never thought of force, I didn't even know force was a vector. I thought it was a magnitude with a direction... which is a vector. I don't know, I guess that in my head force was positive or negative in direction. This problem definitely helped me to learn that we needed to use the r hat vector... For me, this entire problem was about velocity and the force pointing downwards, but downward changes.*

In the future, we aim to synthesize this and other data to form a deeper and broader understanding of how computation can be used in physics courses to help students develop a deeper understanding of physics content and practice.

A stationary star is located at $\langle 1, 3, 0 \rangle \times 10^{14}\,\mathrm{m}$ and a planet moving with a velocity of $\langle 2, -1, 0 \rangle \times 10^{3}\,\mathrm{m/s}$ is located at a position $\langle -4, 1, 0 \rangle \times 10^{14}\,\mathrm{m}$. What is the vector pointing from the initial location of the star to the planet?

$\vec{r} = \langle \quad , \quad , \quad \rangle$

The Moon orbits the Earth in a roughly circular orbit. To calculate the force the Earth exerts on the Moon, you need to know the direction of the separation unit vector ($\hat{r}$) and the gravitational force unit vector ($\hat{F}$). For locations A-D, find $\hat{r}$ and $\hat{F}$.



At A:

$\hat{r} = \langle \quad , \quad , \quad \rangle$

$\hat{F} = \langle \quad , \quad , \quad \rangle$

At C:

$\hat{r} = \langle \quad , \quad , \quad \rangle$

$\hat{F} = \langle \quad , \quad , \quad \rangle$

Figure 4.4: Pre-class homework problems delivered in the second week of class that are specifically designed to strengthen the connection between a separation vector, its unit vector, and the direction of a force.

Chapter 5

Identifying computational practices

Our analysis of the in-class group computational problem solving in P^3 can be broken down into a number of different resolutions/scales. One resolution/scale that we can focus on is the individual group and the different practices that they are engaging in. Each practice identified within a particular group, as broadly defined in the Weintrop Framework, manifests in multiple possible variations. The amount of variation observed usually depends not only on the practice (i.e., some practices are generally more diverse/varied than others) but also on the particular group (i.e., some groups engage in more diverse/varied practices).

5.1 Specific methods

5.1.1 Transcription

We transcribe the entire problem solving process that has been captured on video, paying specific attention not only to what the group is saying, but also to what the group is doing (e.g., typing lines of code, gesturing, etc.). Any lines of code that are typed can be verified by checking against the screencast of the group's laptop. Any written equations or scratch work can be checked against the overhead camera. These different sets of data are then used in conjunction while searching for practices.

It is very rare that we see groups explicitly defining their actions in terms of the language

of the framework. Accordingly, we need to infer what the groups are thinking from the data.

5.2 Findings

Recall that, according to the framework, each category of practice can be broken down into a number of individual practices. Each individual practice, further, can be identified in terms of a number of fundamental characteristics. Depending on the particular situation, some categories show up in the data more often than others. For example, we expect to see fewer systems thinking practices and more computational modeling practices. Further, within any category, some individual practices show up in the data more than others. For example, within the data practices, we expect to see fewer instances of collecting data and more instances of creating data.

Most importantly, within any individual practice, there is a broad set of defining characteristics. Ideally, each characteristic would be present in some form when its corresponding practice has been identified. However, each characteristic can be organized in terms of being either necessary or sufficient for the cause. That is, we need not necessarily observe every characteristic in order to identify a practice according to the framework.

5.2.1 Assessing computational models

The most important computational modeling practice indicative of computational thinking is that of assessing computational models. This practice shows up 100 % of the time. Assessing computational models seems to be crucial to the process of designing and constructing computational models. Given that computational models take a lot of work and iteration to get to an acceptable level, it is no surprise that assessing shows up so frequently.

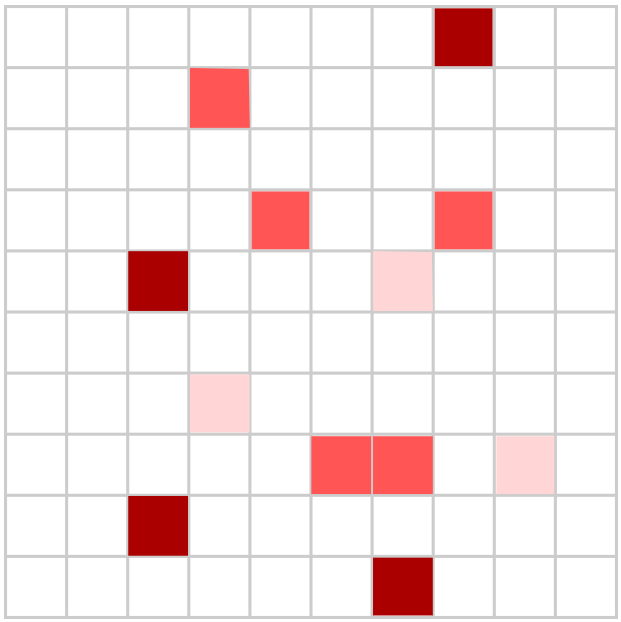


Figure 5.1: A "density plot" for all computational practices and all groups.

There are three characteristics that are necessary to be observed in an excerpt to warrant classification of "assessing computational models." The first characteristic that needs to be observed is a "model." There are several models that show up in the data. The second characteristic that needs to be observed is a "phenomenon." For the most part, the phenomena are related to the simulation of the trajectory of the geostationary satellite and its various physical interpretations (e.g., a central attractive force or a circular orbit). The third characteristic that needs to be observed is a "comparion" between the model and phenomenon. The comparisons that we see are ultimately varied, given that they depend on not only the model but also the phenomenon.

a	b	c
c	d	100 %

Figure 5.2: A table showing rough statistics for assessing computational models.

5.2.1.1 Models

The two big models that we have observed in the data are a position dependent Newtonian gravitational force in terms of a separation vector and a centripetal force that depends on the polar angle of the satellite. Each model shows up 100 % and 100 % of the time, respectively.

5.2.1.1.1 Group A (gold star)

5.2.1.1.2 Group B (near miss)

5.2.1.2 Phenomena

The phenomena that we have observed in the data are almost always centered around the simulation of the trajectory of the satellite. The physical interpretations of the phenomena are, not surprisingly, closely related to the topics covered in the couse notes and the topics questioned in the weekly homework. These interpretations range from the crude (e.g., a circular orbit) to the sophisticated (e.g. a centripetal force and its properties). This variety of phenomena show up 100 % of the time.

5.2.1.2.1 Group C

5.2.1.2.2 Group D

5.2.1.3 Comparisons

The comparisons that we have observed in the data are quite varied. Given that the comparison is between the models and phenomena, we expect this characteristic to be the most varied.

5.2.1.3.1 Group E

5.2.1.3.2 Group F

Chapter 6

Discussion

Chapter 7

Concluding Remarks

APPENDICES

Appendix A

Additional excerpts

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] Undergraduate Curriculum Task Force AAPT. Recommendations for computational physics in the undergraduate physics curriculum. Technical report, AAPT, 2016.

[2] Alfred Aho. Computation and computational thinking. *The Computer Journal*, 2012.

[3] John Aiken, Marcos Caballero, Scott Douglas, John Burk, Erin Scanlon, Brian Thoms, and Michael Schatz. Understanding student computational thinking with computational modeling. In *PERC Proceedings*, 2012.

[4] Alan Bundy. Computational thinking is pervasive. *Journal of Scientific and Practical Computing*, 2007.

[5] Marcos Caballero, Matthew Kohlmyer, and Michael Schatz. Fostering computational thinking in introductory mechanics. In *PERC Proceedings*, 2011.

[6] Marcos Caballero and Steven Pollock. A model for incorporating computation without changing the course: An example from middle-division classical mechanics. *Americal Journal of Physics*, 2014.

[7] Ruth Chabay and Bruce Sherwood. Computational physics in the introductory calculus-based course. *American Journal of Physics*, 2008.

[8] Norman Chonacky and David Winch. Integrating computation into the undergraduate curriculum: A vision and guidelines for future developments. *American Journal of Physics*, 2008.

[9] Shuchi Grover and Roy Pea. Computational thinking in k-12: A review of the state of the field. *Educational Resarcher*, 2013.

[10] Matthew Kohlmyer. *Student performance in computer modeling and problem solving in a modern introductory physics course*. PhD thesis, Carnegie Mellon University, 2005.

[11] Renee McCauley, Sue Fitzgerald, Gary Lewandowski, Laurie Murphy, Beth Simon, Lunda Thomas, and Carol Zander. Debugging: a review of the literature from ane ducational perspective. *Computer Science Education*, 2008.

[12] Committee on a Conceptual Framework for New K-12 Science Education Standards. *A Framework for K-12 Science Education: Practices, Crosscutting Concepts, and Core Ideas*. The National Academies Press, 2012.

[13] Seymour Papert. *Mindstorms: Children, computers, and powerful ideas*. Basic Books, 1981.

[14] Seymour Papert. An exploration in the space of mathematics educations. *Technology, Knowledge, and Learning*, 1996.

[15] NGSS Lead States. *Next generation science standards: for states, by states*. The national Academies Press, 2013.

[16] David Weintrop, Elham Behesthi, Michael Horn, Kai Orton, Kemi Jona, Laura Trouille, and Uri Wilensky. Defining computational thinking for mathematics and science classrooms. *Journal of Science Education Technology*, 2015.

[17] Jeanette Wing. Computational thinking and thinking about computing. *The Royal Society*, 2008.

[18] Jeannette Wing. Computational thinking. *Communications of the ACM*, 2006.