A computational introduction to STEM studies

Eric Freudenthal[1], Alexandria N. Ogrey[1], Mary K. Roy[1], and Alan Siegel[2]

[1]Computer Science Department
University of Texas at El Paso
El Paso, Texas
{efreudenthal, mkroy}@utep.edu; anogrey@miners.utep.edu

[2]Computer Science Department
Courant Institute of Mathematical Sciences
New York, NY
siegel@cs.nyu.edu

Abstract— **We report on the content and early evaluation of a new introductory programming course "Media Propelled Computational Thinking," (abbreviated MPCT and pronounced iMPaCT). MPCT is integrated into a freshman-level program designed for under-prepared students with interests in a STEM discipline. It is intended to reduce attrition rates by fostering student intuition in, appreciation of, and confidence about basic pre-calculus concepts.**

The MPCT curriculum is problem-driven, with analytical challenge exercises that are intended to motivate inquiry and to illustrate the reasoning used in the STEM professions Preliminary evaluation results are encouraging – students from a wide range of academic majors found MPCT engaging, and report that the course conveyed insight, and decreased anxiety about foundational mathematical concepts.

Keywords-component; introductory computing curriculum, CS0, MPCT, entering students program, CCLI, CPATH.

I. Introduction

We report on the content and early evaluation of an introductory programming course titled "Media-Propelled Computational Thinking" (MPCT) [5] that is intended to retain students in STEM disciplines by strengthening their understandings of pre-calculus concepts.

At UTEP, MPCT has evolved into variants taught with the introductions to several of the STEM subjects. The original version was integrated into some of the Entering Students Programs (ESP) [4] attended by entering freshmen at the University of Texas at El Paso, a primarily Hispanic-Serving Institution (HSI) serving an economically disadvantaged bi-national urban area on the US-Mexico border. The ESP course is designed to help students develop the skills necessary for academic success, and to assist students in career selection. Consequently, half of the ESP course addresses issues such as study, note-taking, presentation, writing skills, and career guidance. The other half features a technical content module such as MPCT. Its purpose is to present an engaging broad-brush perspective about learning technical content.

Introductory courses for technical disciplines such as computer science can only provide a limited set of activities. One of the main challenges we faced in the design of MPCT was how to use our very limited classroom time in a way that is not only engaging, but that first and foremost improves student readiness for calculus and the challenges of mathematical reasoning. At UTEP, students often have to spend four semesters in remedial and background classes before enrolling in a first course in a STEM major. Consequently, we needed a half-course that could enhance student interest in CS, while simultaneously providing enough mathematical intuition to improve student retention in the high-attrition STEM majors. Trial courses based on outside programs identified some engaging curricula -- especially for students interested in art or in the use of applications to solve specific types of problems -- but none of the programs seemed to have enough focus on pre-calculus mathematics concepts backed by hands-on programming. So we designed our own course, which turned out to require far more work than we had anticipated.

Section II presents the main pedagogical concerns that guided the overall design of MPCT.

The need for an easy-to-use programming environment was also of great concern, since students need to focus most of their time on analytical tasks and algorithm design, and not program syntax. Likewise, we realized that typical programming projects in MPCT had to be very short. Typical assignments are solved with four to ten lines of Python code that includes numerical iteration controlled by for-range statements. To provide students with visceral understandings of program behavior, the output is generally graphical, and involves the direct manipulation of pixels within an RGB image plotted in Cartesian coordinates. Section III presents some of the constructs used to minimize the syntactic difficulties.

The progression of concepts and projects is presented in Section IV. Programming fundamentals are introduced and exercised within lab assignments during the first couple of weeks of class in the context of graphical manipulation such as such as the use of nested for-range statements to enumerate the coordinates of pixels within geometric objects. Subsequent labs first examine the plotting of simple mathematical functions, which are later extended to explore the simulation and mathematical simulation of familiar physical phenomena such as ballistics and resonance.

Section V mentions the adaptation of this education program for other courses at UTEP, and Section VI presents ongoing efforts to evaluate MPCT. A preliminary study indicates increased confidence and interest in pursuing

STEM studies despite MPCT's lack of socially motivated context. Our research plans include a longitudinal study of academic progress that examines the success of students who pursue STEM studies after attending MPCT.

II. The Pedagogical Approach

MPCT's design is motivated, in part, by observations that deficiencies in, and anxieties about mathematics cause many students to avoid computer science and other STEM disciplines, and are likewise responsible for much of the attrition from these programs. Thus, the key question in designing MPCT is how to foster better mathematics intuition, and how to instill these understandings in an engaging (and hopefully enduring) way. Of course, the results to date, though encouraging, are only preliminary, and much more needs to be done to see if the basic approach can be extended to encompass a more substantial curriculum. Nevertheless, it is not too soon to communicate some preliminary conclusions -- especially when learned from mistakes and occasional failures.

In retrospect, this project has benefitted from a small number of critical design principles. Some of the most important are listed below.

1) Minimize syntactic overload.

Students need to spend most of their time mastering concepts and developing intuition. The syntax of object oriented programming and excessive type declarations can be very time-consuming for beginners.

2) Use a sequence of exercises that present a coherent progression of worthwhile projects.

Since MPCT is intended to use computation to strengthen mathematical reasoning, we designed exercises (coupled with interesting applications) that illustrate increasingly advanced principles in pre-calculus and calculus. Section IV covers the sequence in detail. We also selected problems that are in some sense representative of the issues faced by practicing STEM professionals.

3) Hands-on computing and a see-for-yourself approach to learning is unlikely to succeed unless each inquiry-based exercise is designed with the utmost of care, and tested to ensure that the outcomes match the intentions.

The first version of MPCT used an early exercise that asked students to the graph lines y=ax+b and experiment with the parameter a to see what happens. To our surprise, the overall lesson failed to instill a sound understanding of slope. A revised exercise sequenced through smaller, more carefully structured steps: Fill row 13 of an array; Plot the line y=13, Now treat y as a running sum that begins at 13, and grows by the constant 0.1 from x-value to x-value. Now change the growth constant from 0.1 to 0.5. Here the primitive was slope, and students quickly grasped the idea that straight lines and constant slope (growth) were equivalent. More importantly, they displayed a solid understanding of slopes, and the progression to parabolas based on constant incrementation of the growth rate (slope) was much easier to teach.

III. Minimizing Empahsis on Syntax

Programming techniques in early courses should be chosen to minimize cognitive load while maximizing pedagogical value. The focusing of MPCT to introductory computation included a significant reevaluation of the programming interfaces used to support coursework. The original programming interface used the rich object oriented (OO) Java AWT toolbox exposed by the programming framework of [1]. With this approach, even the design of extremely simple algorithms requires fairly complex access code before anything can be programmed. Consequently, the *conceptual* content embedded within our introductory programming lessons were often overwhelmed by the mainly *clerical* task of managing the access and manipulation abstractions for pixels in Java.

This motivated our development of an alternative shallow Raster class described in Figure 1. All pixel accesses explicitly specify row-column addresses. Thus, a pixel location is just a (row ,column) pair of integers represented using a native Python tuple (vector) type, and a pixel color is just a red-green-blue tuple. Algorithms that visit all pixels in a rectangular region can be programmed by a pair of nested loops. As reported in [5], students have little trouble understanding nested iteration in this context.

Constructors
- `Raster((numCols, numRows))`
- `Raster((filename))`

Accessors
Origin is in lower-left corner
Parameters & return values are tuples
- `r, g, b = getRGB((col, row))`
- `setRGB((col, row), (r, g, b))`

Controls
- `setAutoRepaint(state)`
  redraw on every access (default true)
- `setIgnoreOutOfRange(state)`
  silently ignore out-of-range accesses (default true)
- `width, height = getWidthHeight()`
  returns raster geometry

Actions
- `write(filename)`
  save to file
- `repaint()`
  redraw now

Figure 1. Public interface to Raster class

The course begins with by presenting simple code that draws dots and lines. Students adapt them to draw rectangles, triangles, trapezoids and parallelograms by the end of the second class. Figure 2 presents a program sometimes examined within the first two weeks of class that dramatically modifies a familiar cartoon image. The example was selected to illustrate low-overhead programming – *the code does not even require a function definition.*

In order to facilitate projects that plot mathematical functions and leverage students' incoming knowledge, `Raster`'s origin is located in the lower-left corner, and thus column-row addressing directly mimics x-y coordinates within the first quadrant of a Cartesian plane.

```
url = "http:….jpg"
p = Raster(url)
green = (0, 255, 0)
cols,rows =  p.getWidthHeight()
 for col in range (cols):
  for row in range (rows):
     r,g,b = p.getRGB((x,y))
     if r<40 and g<40 and b<40:
        p.setRGB((col,row), green)
```

Figure 2. Program to recolor dark pixels

In short, the simplifications in the necessary code allow the class time to focus on the logic and the algorithms rather than object-oriented hierarchies that provide abstractions ill suited to the problem being addressed. Later projects require the plotting of functions with negative range that are (deliberately) inconvenient to represent with Raster's origin, which is located in the image's lower-left corner. MPCT pragmatically utilizes this inconvenience to motivate the introduction of a `PosNegGraph` class that extends `Raster`. Both `Raster` and `PosNegGraph` are referenced by examples in this paper.

The next section summarizes the curriculum, including example problems in the newly developed modules on mechanical resonance and coupling, and planned extensions of MPCT's pedagogical approach to other courses. Finally, we report on our in-progress course evaluation and adaptations to the evaluation plan in response to MPCT's evolved focus.

IV. Topic Sequence

Figure 1 presents a flowchart of course modules and prerequisite dependencies. A strong–inter-module dependency is represented by a solid line; and a weaker dependency (indicated by a dotted line) indicate an alternate path that permits a course to move more quickly at the cost of removing a particularly engaging presentation of a key concept.

Initial programming exercises are sufficiently short to be conveniently typed directed to the interpreter. The first exercises contain no explicit arithmetic operations, which are gradually introduced to implement increasingly sophisticated computation. The difficulty of reliably typing more advanced programs motivates their storage within files.

The overall program is designed to foster experimentation and to encourage students to review fundamental concepts in algebra and geometry in a way where motivated understanding leads to successful program-generated displays.

For example, A.2 provides a preferred first exposure to nested looping that provides opportunity for students to play with both column- and row- major filling of simple polygons in a dramatic and motivated context. In A.3, students apply their newly gained knowledge of nested iteration to the filling of a rectangular region in A.3, which uses nested looping over both columns and rows to visit every pixel within a rectangular region.

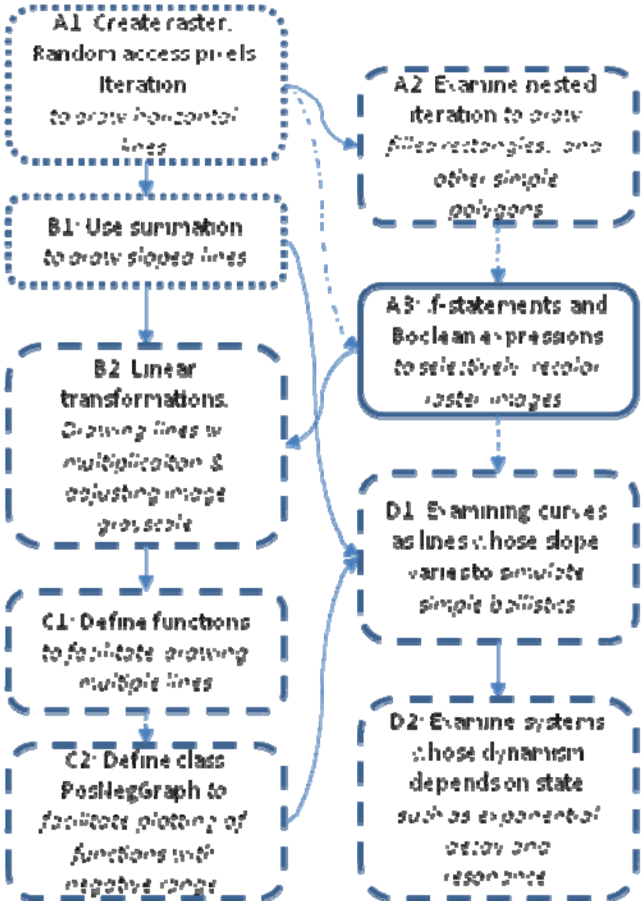

A1 Create raster. Random access pixels Iteration to draw horizontal lines
A2 Examine nested iteration to draw filled rectangles, and other simple polygons
B1 Use summation to draw sloped lines
A3 If-statements and Boolean expressions to selectively recolor raster images
B2 Linear transformations. Drawing lines w multiplication & adjusting image grayscale
D1 Examining curves as lines whose slope varies to simulate simple ballistics
C1 Define functions to facilitate drawing multiple lines
C2 Define class PosNegGraph to facilitate plotting of functions with negative range
D2 Examine systems whose dynamism depends on state such as exponential decay and resonance

Figure 3. Linkages among course modules

A. *Programming Basics*

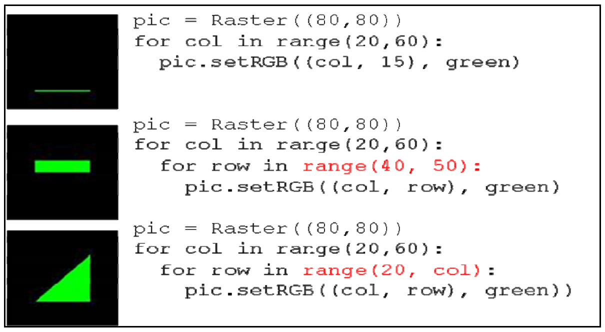
In a manner analogous to an immersive language course, students begin to 'converse' in Jython using only three statements to draw a multitude of geometric objects:

- `Raster(size)` – Used to "construct" a computer image:
- `setRGB(pos, color)` – Used to draw a dot:
- `for loops` - Used for iteration.

As described below, these commands allow students to:

- Draw lines as a sequence of dots stacked in a row.
- Draw boxes as a sequence of lines stacked in a column
- Draw triangles, parallelograms, and trapezoids by deriving the inner loop's range from the outer loop's iteration variable.

Module A.1: Create raster, random access pixels, iteration to draw horizontal lines: This introductory lesson exploits Python's interactive mode to permit students to issue single commands to create and manipulate a raster image (represented as a Raster object) and "draw" upon through the column-row selection of individual pixels that are conveniently represented using Python's native tuple (vector) type. The clerical challenge of reliably typing sequences of commands first motivates the storage of a sequence of commands within a file in a manner can be easily repaired or reused using copy/paste. Finally, iteration is presented as a convenient and intuitive solution to the challenge of repeating a single command multiple times while varying a single parameter in order to draw horizontal or vertical lines as is illustrated in the top row of Figure 4.



```
pic = Raster((80,80))
for col in range(20,60):
  pic.setRGB((col, 15), green)

pic = Raster((80,80))
for col in range(20,60):
  for row in range(40, 50):
    pic.setRGB((col, row), green)

pic = Raster((80,80))
for col in range(20,60):
  for row in range(20, col):
    pic.setRGB((col, row), green))
```

Figure 4. Exploration of iteration

Module A.2: Examine nested iteration to draw filled rectangles, and other simple polygons: As illustrated in the lower two rows of Figure 4, A.2 exploits students' familiarity with direct pixel manipulation and iteration presented in A.1 to provide students with an engaging opportunity to playfully tinker with tiny programs that generate filled geometric forms. A.2's engaging sequence of exercises first introduce students to the nesting of loops as an intuitive and simplifyin approach to the clerical challenge of enumerating the location of every pixel within rectangular region and then provide an opportunity to develop intuition about inter-loop interactions.

For example, the first project traverses every pixel within a rectangular region in column-major order, and its immediate successor challenge students to fill the same region in row-major order. Conceptual errors result in easily understood graphical results that are amenable to student analysis and repair. A.2's scope can be modulated by available time and desired learning outcome. Optional lessons examine interactions between inner and outer iteration bounds in order to draw non-rectilinear forms such as triangles, parallelograms, and trapezoids and manipulate iteration stride to enable the drawing of striped patterns. Should the learning priorities include only a superficial understanding of iteration, A.2 can be omitted entirely and the topic can instead be briefly presented in the context of A.3.

Module A.3: If-statements, relational, and boolean operators to selectively recolor raster images: As illustrated above in Figure 2, this lesson uses nested iteration and exploits conditional expressions including relational, Boolean, and grouping operators to conditionally change the color of pixels within a rectangular image. Earlier modules used intuitively named pre-defined constants (e.g. "green") to represent colors. A.3 exposes the graphical system's underlying representation of color as red-green-blue tuples. A raster image is initialized from a jpeg explicity obtained from "the web" using HTTP.

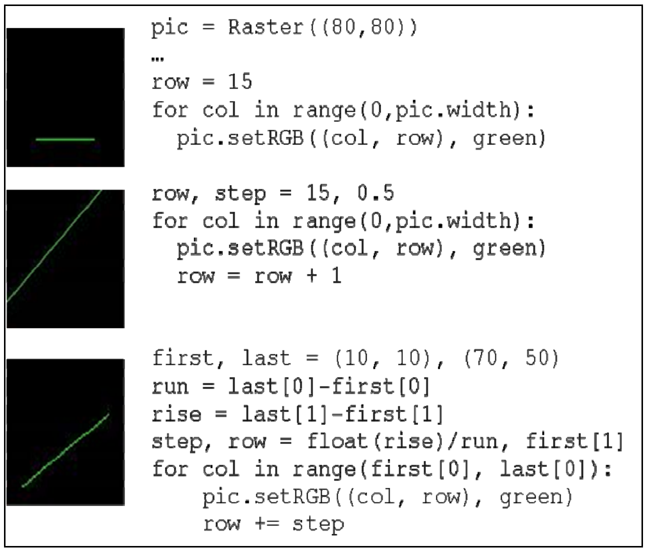
Each pixel location within the rectangular (raster) image is systematically visited using nested iteration. Each pixel's red-green-blue vector is obtained and unpacked into its components using a single getRGB method. An if-statement that contains relational and Boolean operators controls whether the pixel's color-vector is overwritten. Projects challenge students to both experiment with the creation of conditional expressions that select alternate colors and the specification of vectors representing desired alternate colors. Finally, students are exposed to the complexity of automating the detection of forms easily recognized by the human vision system. Optional projects examine simple image processing techniques such as blurring that provide insight into convolutional operations.

B. *Linear systems*

Module B.1: Use of summation to draw sloped lines and the computation of slope: This module examines the nature of linear systems. As illustrated by the top two images and programs from Figure 5, initial projects extend the concept of iteration introduced to draw horizontal lines in A.1 to instead draw sloped lines using summation. A row variable is initialized at column zero. A constant "step size" value is repeatedly summed into the row variable increasing its value at a linear rate that is graphically depicted. Students characterize the effect of various step sizes and initial row values. Later exercises lead students to derive the meaning of their generalization as slope and y-intercept.

As illustrated by the image and program at the bottom of Figure 5, students are subsequently challenged to draw lines that connect designated points (say, to draw a geometric shape). To do this, students must derive the step size from the desired change in row and column using division. We observe that most attending the class can parrot "y=mx+b" as an equation for a line, but nonetheless initially have trouble computing step size; even math-phobic students enrolled in non-STEM programs and are visibly delighted when they derive an approach to determining step size (slope) using division.

Module B.2: Linear transformations: Drawing lines with multiplication and adjusting image grayscale. This module challenges students to generalize their understanding of line generation to the linear projection from an input variable to a dependent output variable. Early projects examine the concept and utility of computing a y-intercept in order to easily compute the row corresponding to any column.



```
pic = Raster((80,80))
…
row = 15
for col in range(0,pic.width):
  pic.setRGB((col, row), green)

row, step = 15, 0.5
for col in range(0,pic.width):
  pic.setRGB((col, row), green)
  row = row + 1

first, last = (10, 10), (70, 50)
run = last[0]-first[0]
rise = last[1]-first[1]
step, row = float(rise)/run, first[1]
for col in range(first[0], last[0]):
    pic.setRGB((col, row), green)
    row += step
```

Figure 5. Exploration of line drawing

A particularly motivational project from this lesson examines the grayscale correction of pictures with limited dynamic range. Following a discovery-learning model, students are initially challenged to “improve” the picture’s contrast without guidance Typically students will suggest appliying either linear scaling or constant offsets.

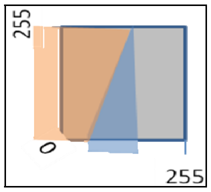
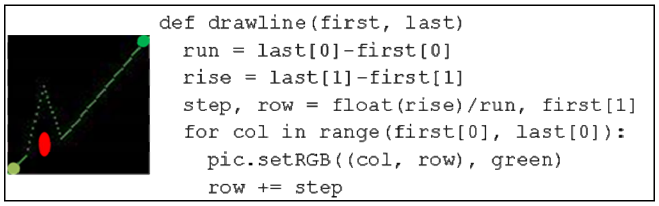


Figure 6. Expansion of image grayscale

Figure 6, illustrates a composite approach that applies a linear transform that projects the input image’s limited grayscale to a full 0…255 range. The challenge of correctly displaying a negative image permits students to explore the computation of inverse relationships.

C. *Managing complexity using language features*

Module C.1: Defining functions to facilitate drawing multiple lines: Functions are introduced as an approach to reduce complexity through the creation of reusable code fragments. As illustrated in Figure 7, students are challenged to write programs that draw a path using line segments that connected designated start and end points and avoid colored obstacles. Rather than focusing on the path problem, this module provides opportunities to first practice the application of computing slope. Observe that, the body of drawline appearing in Figure 7 is identical to the program at the bottom of Figure 5. In this manner, reduction of the program complexity is achieved by drawing multiple lines with multiple calls to a single function rather than the insertion of repeated code fragments.



```
def drawline(first, last)
  run = last[0]-first[0]
  rise = last[1]-first[1]
  step, row = float(rise)/run, first[1]
  for col in range(first[0], last[0]):
    pic.setRGB((col, row), green)
    row += step
```

Figure 7. Maze and function from C.1

Module C.2: A user-defined class that facilitates plotting of functions with negative range: The linear systems examined previously only have non-negative range. In this module, students are encouraged to examine negative range by translating the image through the adding of a constant to the row value. The repetitive task of translating the image to examine the negative range motivates the examination of a user-defined class that centers and draws the X-axis. As illustrated by Figure 8, the easily understood PosNegGraph class conveniently translates the origin to the middle of the left edge.

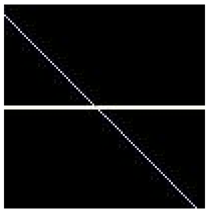


Figure 8. Illustration of PosNegGraph plot

D. *Non-linear systems*

To ensure that students are not intimidated by unfamiliar mathematical abstractions, we prefer to first introduce evolving processes using the simplest-possible generators, and then to guide students into discovering algebraic simplifications they are fully prepared to understand.

The topics sequence through increasingly complex computational themes where all of the physical modeling is based on rates of change or summations (as opposed to integration).

Lab exercises examining non-linear systems extend the summation techniques previously examined to simulate the dynamics of familiar physical phenomena. Exercises mimic the familiar phenomenon of ball bounce and spring resonance, which are frequently poorly understood, even by students who have completed a semester of college physics [6].

Finite differences are computed at each time step and are summed into system state. Longitudinal studies are being conducted to determine if cohorts of students who are first exposed to these concepts through simulation have improved academic success in subsequent courses that examine the same phenomena using calculus.

For problems in mathematics and physics, we endeavor to minimize the level of outside knowledge required, and prefer show how processes evolve at an intuitive level and where incremental changes in the process state can be explained in physical and computational terms -- much like [7].

Math-centric programming projects generally begin with an exploration of the effects of rates-of-change. Afterwards, students are guided to reduce the already familiar summation problems (implicitly a recursive formulation) to closed form.

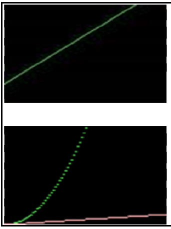
D.1: Examining curves as lines whose slope varies to simulate simple ballistics: Boolean expressions and conditional statements (taught in A.3) should be taught if the simulation of a ball bounce is included in the lesson. The instructor leads a discussion about the similarities and differences between lines and curves and elicits an observation that a curve's slope must vary. Students add the rate variable, and begin to experiment with drawing curves.

The main study concerns curves whose slope changes linearly as shown in the parabola in the bottom Figure 9. Once sufficient time has been spent experimenting with drawing curves, students begin to apply what they have learned to simulating familiar phenomena such as ballistics and ball bounce.

In ballistic problems, objects are accelerated only by gravity. Their trajectory is parabolic. The slope of their trajectory with respect to time corresponds to velocity. The slope of their velocity with respect to time corresponds to acceleration. The mapping of trajectories to parabolas is straightforward: slope corresponds to velocity, and the slope's constant rate of change maps to acceleration. Students first simulate a single "toss," and then are challenged to simulate a bounce as an inelastic collision with the ground, where at each bounce, velocity is reduced by 20%. This leads to an exponential decay in the maximum height achieved after each bounce. The `parabola` program's overlaid plot of position and velocity illustrates both continuous and discontinuous evolution of physical parameters.

As illustrated in Figure 10, students are able to easily examine the effects of varying initial conditions including examples where the slope and rate have different and same initial signs. Finally, the relationship between parabolas and quadratic functions are made concrete through geometric proofs such as depicted in Figure 11.

D.2: Examination of systems whose dynamism depends on state such as exponential decay and resonance. It is interesting that upperclassmen who have attended multiple courses in Calculus are enthralled by the progressive computation of exponential decay illustrated in Figure 12 that could easily be understood by a freshman if presented in an accessible context. We observe that the spring resonator of Figure 13 is accessible to students attending a first course in mechanics. The electrical resonator of Figure 13 is incorporated in an introductory programming course offered to students in electrical engineering in the Spring of 2010 that incorporates the pedagogy of MPCT.



```
row, step = 15, 0.5
for col in range(0,pic.width):
  pic.setRGB((col, row), green)
  row = row + 1

row = step = 0
rate = 0.1
for col in range(0,pic.width):
  pic.setRGB((col, row), green)
  pic.setRGB((col, step), pink)
  row += step
  step += rate
```

Figure 9. Progression from lines to curves

```
def bounce(height=50, speed=0, accel = -0.1):
  pic = PosNegGraph((100, 75))
  for time in range(0,pic.width):
    pic.setRGB((time, height), green)
    pic.setRGB((time, speed), pink)
    height += speed; speed += accel
    if height < 0 and speed < 0:
      speed = -0.8 * speed
```

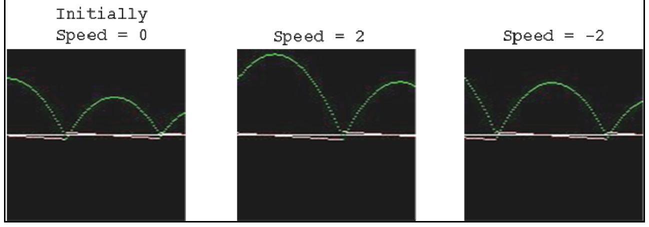
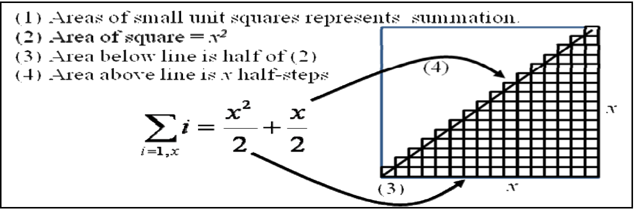
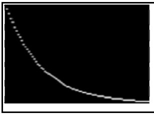


Figure 10. Ballistic simulations


(1) Areas of small unit squares represents summation.
(2) Area of square = x^2
(3) Area below line is half of (2)
(4) Area above line is x half-steps
(4)
(3)
x
x

$$\sum_{i=1,x} i = \frac{x^2}{2} + \frac{x}{2}$$

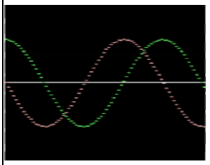
Figure 11.
Graphical depiction of linear sums in closed form as a quadratic function



```
i = Raster((80, 80))
row = 80
for col in range(0, 80):
  i.setRGB((col, row), white)
  row = row * 0.95
```

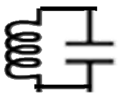
Figure 12. Exponential decay

MPCT completes with an example of coupled resonance illustrating the principle underlying an opera singers' wine-glass shattering trick and the catastrophic failure of the Tacoma Narrows bridge (see Figure 3) in 1940 and the crashes of several Lockheed Electras around 1960.



```
i = PosNegGraph()
pos = 45; speed = 0
springConst = 0.01
deltaT= 0.1
mass = 0.01
for x in range(0, i.width):
  i.setRGB((x, pos), green)
  i.setRGB((x, speed), pink)
  force = -pos * springConst
  accel = force / mass
  speed += accel * deltaT
  pos += speed * deltaT
```

Figure 13. Simple mechanical resonator



```
img = PosNegGraph((300, 100))
V, I = 0, 50.0e-3    # initial conditions
C, L = 1e-9, 1e-3    # farad, henry
for x in range(0, img.width):
  img.setRGB((x, 1000*I), white) # fs=100ma
  img.setRGB((x, V), green)
  dv = (I/C) * dt
  di = -(V/L) * dt
  V = V + dv
  I = I + di
```

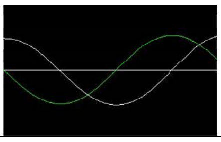


Figure 14. Electrical resonator

MPCT completes with an example of coupled resonance illustrating the principle underlying an opera singers' wine-glass shattering trick and the catastrophic failure of the Tacoma Narrows bridge (see **Photo image courtesy of Ed Elliott www.camerashoptacoma.com. All rights reserved. [8]**

Figure 15) in 1940 and the crashes of several Lockheed Electras around 1960.

Figure 16 illustrates the result of an advanced project that examines coupling when both the oscillator and resonator are tuned to the same frequency. Like the catastrophic failures enumerated above, this resonator quickly accumulates energy from the oscillator.

Photo image courtesy of Ed Elliott www.camerashoptacoma.com. All rights reserved. [8]

Figure 15. The Tacoma Narrows Bridge

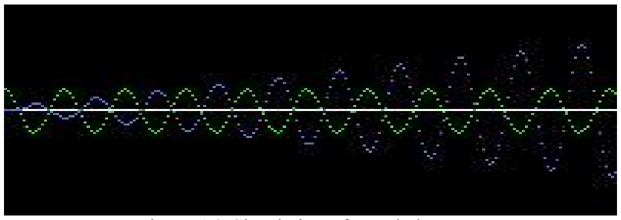


Figure 16. Simulation of coupled resonance

V. Extensions to Other Courses

A new introductory course titled "Computation for Science and Engineering" in computational science intended for upper-division students of STEM disciplines that traditionally do not include computation such as biology, geology, finance, and math is planned for Spring 2010. This course will introduce programming using examples from MPCT, and then will proceed to implement simulations modeling more dynamic systems in which a sudden action could change the "normal" or expected behavior such as production-consumer markets, investment valuation, predator-prey models, and biological processes.

Two other pilots are beginning in Spring 2010.: A section of an introductory programming course offered to students enrolled in Electrical Engineering will be taught using MPCT's pedagogical approach and will include projects that simulate dynamism in electrical systems. A section of a statistics course attended by students of psychology will also include elements from MPCT with the expectation that the process of constructing simulators of stochastic systems will assist in students' understanding of coupled and independent random processes.

We are also adapting this approach of motivating math from concrete problems to the teaching of algorithms. There, the objective is to use specific problems as a vehicle for teaching algorithms as general methods that can be adapted to solve related problems of interest. We find that by teaching algorithmic schemas to solve "purified" problems, students can follow the reasoning far more easily and can sometimes develop the computational idea themselves. As in MPCT, the layered elaborations are introduced step-by-step to solve increasingly complex problems as described in [9].

VI. Evaluation

More than half of the students entering UTEP intending to study CS are not "calculus ready," as required for CS1. First-attempt pass rates for CS-1 range from 50 to 70%, which surely contributes both to student time-to-graduation and attrition. Thus the need for intervention is clear. During the development of MPCT, several variants of Media Programming were offered at UTEP. In all versions, almost all students demonstrated proficiency at basic programming concepts and passed.

UTEP is a member of the Computing Alliance of Hispanic Institutions (CAHSI), which is evaluating various CS-zero courses offered at multiple institutions serving predominantly Hispanic populations of students. These evaluations include intermittent classroom observations and both pre- and post-course surveys examining preparation, social context, student engagement with the course, interest in further study of computation, and success in subsequent coursework.

In their 2008 report, the CAHSI evaluator measured high levels of both interest and confidence [10] among entering students prior to attending the course, though less than 20% had previously programmed. Post-course surveys of students attending the precursor courses indicate that 25% of the students were not motivated to continue studies related to computer science. This is likely a positive result if the course helped students correct unrealistic expectations about computer science early in their academic careers.

Students who attended a variety of CS-0s (including Media Programming at UTEP and Alice at other institutions) had similar passing rates that were similar to the general population. The mathematically-oriented revisions to MPCT have required several semesters of development and have only recently been implemented – hence no longitudinal data is available yet.

Entering STEM students attended MPCT during the Fall semester of 2008, and non-STEM students attended distinct sections during both the Fall and Spring semesters. The mathematics included in the section was first offered Spring 2009, and therefore was only attended by non-STEM students. A post-course survey of the Spring 2009 non-STEM cohort examined changes in (1) perceptions of knowledge and understandings of key concepts, (2) perceptions of mathematics' relevance to 'real life' scenarios, and (3) attitudes toward learning math concepts in the context of programming.

As described in [11], the preliminary findings from the non-STEM section are encouraging. Although the sample size was too small to draw reliable conclusions, they reflect the instructors' observations of student motivation and engagement. Survey results indicate shifts from low level to higher levels of understanding math concepts introduced in MPCT, positive attitude toward MPCT structure and its intended objectives, and highly favorable perceptions of MPCT's relevance to real-life applications. We expect statistically reliable results from the Fall 2009 cohort, which is much larger, containing more than sixty pre-STEM and twenty non-STEM students.

During the Spring 2009 term, the evaluation was broadened to include instruments that examine changes in interest, self-efficacy and competence related to mathematics. Several open-ended essay questions were included in order to guide the selection of relevant questions for the intended Fall evaluation.

Pre-survey results indicate that in addition to high confidence in programming their programming skill, students have high confidence in their math skills that are inconsistent with their performance in class.

Our longitudinal evaluation will be broadened to also compare the academic success of students in subsequent math courses with the academic success of students who do not attend MPCT.

VII. Conclusion

Continuing evaluation of introductory programming offerings at UTEP targeting pre-STEM students have motivated evolutions in curriculum, course objectives, and evaluation strategies. Interestingly, the resulting course, MPCT, which engages students in a "computational reasoning," integrates both programming and mathematics, is engaging to both pre-STEM and non-STEM students with weak math skills. Results from early evaluation efforts are encouraging and have lead to refinements in evaluation strategy that mirror the course's evolution.

VIII. Acknowledgement

This report is based on work supported by the National Science Foundation through grants CNS-0540592, IIS-0829683, and DUE-0717877 and gifts from Texas Instruments Inc. and Calculex, Inc.. Any opinions, findings, and conclusions or recommendations expressed in the paper are those of the authors and do not necessarily reflect the views of the NSF or other funders.

IX. References

[1] Guzdial, *Computing and Programming with Python, a Multimedia* Approach, Prentice Hall, 2006.

[2] Guzdial, *Design Process for a Non-Majors Computing* Course, Proc.36th ACM Technical Symposium on Computer Science Education (SIGCSE), ACM, 2005.

[3] Guzdial, *Narrating Data Structures: The Role of Context in CS2*, The Journal of Educational Resources in Computing (JERIC), ACM, 2008.

[4] Eric Freudenthal, Mary K. Roy and Ann Q. Gates, *Work in Progress – The Synergistic Integration of an Entering Students Program with an Engaging Introductory Course in Programming*, Proc, Frontiers in Education, Fall, 2009..

[5] Eric Freudenthal, Mary K. Roy, Alexandria Ogrey, Tanja Magoc, and Alan Siegel, *A Computational Introduction to Computer Science*, Proc. Annual Symposium of the Special Interest Group on Computer Science Education (ACM SIGCSE), 2010.

[6] Hestenes, Wells, and Swackhamer, *Force Concept Inventory*, The Physics Teacher, Vol. 30, March 1992, pp 141-158.

[7] Kalman, *Elementary Mathematical Models*, Mathematical Association of America (Press), 1997.

[8] Ed Elliott (1940) The Camera Shop.

[9] Siegel and Freudenthal, *Experiments in teaching an engaging and* demystifying introduction to algorithms: Installment 1: Huffman Codes, UTEP Computer Science Technical Report UTEP-CS-09-12, April 2009.

[10] Thiry, Barker, and Hug, CAHSI Evaluation Progress Report, The Computing Alliance for Hispanic Serving Institutions, 2009, http://cahsi.cs.utep.edu/Portals/0/2008InterimEvaluationReport.pdf

[11] Suskavcevic, Kosheleva, Gates, and Freudenthal, Preliminary Assessment of Attitudes towards Mathematics for a Non-STEM Section of Computational Computer Science Zero, UTEP CS Technical Report UTEP-CS-09-13, May 2009