Reactive Game Engine Programming for STEM Outreach

Alan Cleary  
Montana State University  
Computer Science  
PO Box 173880  
Bozeman, MT 59717  
alan.cleary@cs.montana.edu

Lucas Vandenberghe  
Western State Colorado University  
600 South Adams St  
Gunnison, CO 81231  
lucas.vandenbergh@western.edu

John Peterson  
Western State Colorado University  
600 South Adams St  
Gunnison, CO 81231  
jpeterson@western.edu

ABSTRACT
In this paper we present our experiences using a novel programming style, reactive programming, to deliver a summer camp for students in grades 8 through 12. This software uses a declarative programming approach to allow students without a background in computing to explore a wide variety of subject material within a 3D virtual environment, including computer science, mathematics, physics, and art. This work is based on PyFRP, a reactive programming library written in Python. We describe our camp experience and provide examples of how this style of programming supports a wide variety of educational activities.

Categories and Subject Descriptors
K.3.2 [Computers and Education]: Computer and Information Science Education—computer science education, curriculum

General Terms
Design, Documentation, Human Factors

Keywords
STEM, K-12 Instruction, Functional Reactive Programming, Virtual Worlds, Python, Game Development

1. INTRODUCTION
Summer camps provide an excellent opportunity to introduce computing to new audiences. Since 2006, Western State Colorado University (WSCU) has hosted a computer science summer camp for students in grades 8 through 12. Each session lasts one week, during which time the students build 3D virtual worlds as a means of interdisciplinary learning. The worlds are created using the Panda3D game engine and a Functional Reactive Programming (FRP) Python library. The goal of this camp is not to teach solely computing but rather to use computing as a vehicle for the exploration of a wide range of topics. Campers use our library to create a variety of compact interactive animations and games without requiring in-depth knowledge of computing. We will demonstrate how the addition of FRP to the Python environment significantly increases the ease of programming in an interactive 3D environment without overly constraining the set of programs supported by our system. In particular, we are able to use continuous mathematical operators such as integral and derivative to simplify movement within the 3D environment.

Our system currently uses Panda3D[10], a cross-platform open source game engine with extensive Python support. However, our reactive programming library, PyFRP, is independent of the details of this engine and is currently being ported to other similar systems. We believe that Python provides an excellent introduction to computing and many students come to our camp wanting to learn a “real” programming language. In fact, some parents make a point to mention they want their student to learn Python. Our camp also uses the Blender[7] modeling system to create new 3D models beyond those that are packaged with our library. Blender is a cross-platform open source project as well.

Some of the topics covered include 3D geometry, coordinate systems, vectors, trigonometry, calculus, projectile motion, object collision, control systems, functions, loops, reactive programming, and 3D modeling. We strive to bring mathematics to the foreground in hopes that the students will develop a deeper understanding of the content. The use of FRP allows the mathematics to closely integrate with the graphical environment.

Students work on a variety of projects during the course of the camp. We try to provide them with an open, creative, and collaborative environment with all the resources they need to succeed and have fun.

Our paper is structured as follows: In Section 2 we give a brief description of FRP and our implementation, in Section 3 we discuss the goals of the camp, in Section 4 we outline the design of the camp, in Section 5 we provide examples to demonstrate our programming style and the curriculum of the camp, in Section 6 we reflect on the experience of the camp from both our perspective and that of the students, in Section 7 we survey related work, and in Section 8 we explore future directions of PyFRP and the camp.

2. REACTIVE GAME ENGINE PROGRAMMING
FRP is a general framework for programming hybrid systems in a high-level, declarative manner. It integrates the
idea of time flow into the purely functional programming style. By handling time flow in a uniform and pervasive manner an application gains clarity and reliability.

FRP is composed of two key ideas: behaviors and events. Behaviors are time-varying reactive values, though some are constant, such as numbers and colors. Events are time-ordered sequences of discrete-time event occurrences. Basic events include mouse button and keyboard presses. There is a rich set of operators the user can use to compose new behaviors and events from existing ones. In essence, a FRP program is just a set of mutually-recursive behaviors and events.

A major design goal for FRP is to free the programmer from presentation details by providing the ability to think in terms of modeling. FRP achieves this by making a clear distinction between the what of a presentation and the how. It attempts to automate the how by providing the user high-level abstractions, which prevent them from having to explicitly manage common implementation chores that have nothing to do with the content of an animation.

We found that due to its dynamic typing, semantic overloading, and non-strict semantics, Python lends itself to the data-types and functions of FRP. An added benefit of this approach is that students are learning a full featured language, regardless of the fact that they have no background in computing. While Python is not a purely functional language like Haskell, we use only the functional core of Python.

Our library augments the original FRP library with a notion of objects. A reactive object is one in which the attributes of the object are described using reactive equations. For example, the following program serves as a kind of “hello world” in our system:

```python
panda(hpr = hpr(time,0,0)))
```

The `panda` function creates a new model, a panda, whose hpr (heading, pitch, and roll) are defined by the given reactive equation. Here, the heading is the elapsed time, causing the panda to rotate. Every 3D model has a set of reactive attributes, including position, orientation (hpr), size, color, and texture.

A more complex example of the reactive style is as follows:

```python
panda(position = integral(hold(p3(0,0,0)),
key("upArrow",p3(0,0,1)) +
key("downArrow",p3(0,0,-1)))))
```

This creates a panda which moves up and down using the arrow keys. This is an example of a velocity controller: the `hold` function (a primitive in FRP) is used to remember the current velocity. The `key` functions create events which carry new velocities to the `hold` function; the `+` operator merges the event streams. Finally, the `integral` function converts the velocity to a position.

This style of programming avoids the complexity of event handlers and explicit time flow. The primitives in FRP allow the programmer to work with continuous equations in a very natural manner. As our work is concerned with presenting PyFRP from a user’s perspective, we refer the reader to [6] for an exploration of the formal semantics of FRP and to [10] for a useful collection of data types and functions, as well as FRP implementation details. The examples provided in Section 5 will demonstrate our programming style and the curriculum of our camp.

3. CAMP GOALS

The camp is built around STEM (Science, Technology, Engineering, and Mathematics) topics. Specifically, we focus on teaching fundamental mathematics and programming constructs that are important for game programming and animations, as well as the basic physics that allows games to mimic or bend the rules of reality. We also focus on the artistic side of programming, showing the students how to create animations and pieces of art programmatically and with Blender. In essence, we are trying to put computer science into context by showing students how it is connected to other disciplines.

We want the students to confront the mathematics required for game programming and animations, rather than hiding it behind a GUI. This is why we have taken a purely text-based approach. By leveraging FRP, it allows students to more easily work at the level of equations and models.

We also want to provide the students with quality mentoring. Mathematics and programming does not come naturally to all students. By providing the students with mentors, we are accommodating a variety of learning styles and speeds.

4. CAMP DESIGN

The camp is designed to accommodate a wide variety of abilities, from complete novices to camp regulars. This is done by having the students learn by experimentation and giving them creative freedom in their work.

Throughout the day the students are given lectures on various topics, each supplemented by a concise code example. The students are walked through the code and periodically given time to experiment with the parameter values. We believe this allows them to develop their own intuitive understanding of the lecture material while becoming more comfortable with the code. In this way, the students are gradually introduced to more complex topics and features of the library. During this time, advanced students are encouraged to extend the code to do something more interesting within the realm of the lecture topic.

Students are also given individual projects that require them to draw on material from the lectures. Some scaffold code is provided as a starting point for each project. The projects are designed to make the students solve a particular problem, such as describing the behavior of an object mathematically or using logic to constrain where an object can move on the screen. Beyond this, the projects are open-ended. This allows the students to put their own creative spin on the projects and pursue what interests them. We find that some students focus on the physics of the projects while others prefer to make their own 3D models for everything, and some just want to make games that the other students can compete for high scores on.

Mentors (student assistants) are an essential part of our camp. We use one mentor for every three participants, giving us a lot of opportunity for one on one interaction with the campers. As we have a wide range of age and ability in our camp, having mentors allows each camper to work at the level of their abilities. It also allows more advanced students to work with features of the game engine that are not
covered in lecture, enabling them to pursue projects that are much more open ended. Because we are working directly in the Python environment, these mentors are necessary to assist the students with debugging and errors as well as offer further insight and encouragement with the material.

The mentors receive a couple of days of training prior to the camp to ensure that they are familiar with the curriculum and the software. Many of them are WSCU mathematics and computer science undergraduates who helped write the FRP library. On multiple occasions mentors had attended the camp themselves prior to enrolling at WSCU. One such mentor is an author of this paper.

On some days professors from other departments give guest lectures in topics such as physics, mathematics, or programming. These lectures bring an essential variety to the camp, helping to engage campers in multiple ways. We have been able to build software for these lectures on the fly without having to teach our guests how to program in our system.

While learning programming, mathematics, and physics, the students also get to experience great outdoor activities such as rock climbing, mountain biking, kayaking, and rafting. Apparently students get restless if they have to sit in front of a computer for hours on end. These activities allow them to get a break from learning, enjoy the outdoors, and let off some steam.

At the end of the camp the students showcase their work to their parents. This gives the parents an opportunity to evaluate for themselves what their students have learned and discuss their students experiences with the mentors.

5. EXAMPLES

The following examples demonstrate our programming style and the curriculum of our camp. Specifically, they demonstrate that FRP’s declarative nature makes creating interactions and animations quick and painless and that FRP can be used to teach a variety of subjects.

5.1 Computer Science

Example Dodge:

```python
# Spawning dodgeballs
def generate(m,v):
    for i in range(amount.value):
        h = randomRange(-pi/2,pi/2)
        p = -acos(0.5)/h
        s = sphere(size = 0.0005*time+0.05, color = color(0.7,0.02,0.02))
        launch(s, neck, hprToP3(hpr(h,p,0)))
        hit(b,s,death)

# Getting user input
v = hold(p3(0,-.5,0),
    key("leftArrow",p3(-1.5,0,0))
    + key("rightArrow",p3(1.5,0,0))
    + happen(getX(b.position)<-2,p3(0.25,0,0))
    + happen(getX(b.position)>2,p3(-0.25,0,0)))

# Get input to move player model
b.position = p3(0,0,-2.5)+integral(v)

# Making it run
react(clock(.1),lambda m,v:amount.add(1))
react(clock(0.5),generate)
```

The dodge game example teaches the students how to create interactions between game objects as well as getting user input. The dodge game works well for this because the two main components to the game are moving the player model around the screen and checking if they hit any of the dodgeballs.

The collision detection is handled on line 8 by creating a hit reaction between the player character and each dodgeball as they are created. User input is done on line 11 which maps the input directly to the position of the player model. This line also handles keeping the player in bounds. Note the use of an anonymous function on line 21.

5.2 Mathematics

Example Tornado:

```python
# Aeroplane path
def path(t):
    return p3c(t/10,t-time*time,(t-25)/10)

# Make an aeroplane
def plane(m,v):
    boeing707(position = path(localTime*2.7), color = black, size = .4)

# Making it run
react(clock(.1),plane)
```

The tornado example shows how the students can manipulate the objects on screen using time and mathematical functions to create interesting patterns and shapes. The models spawned in the example get their position adjusted each time-step based on the time of their spawn relative to global time. The image below shows the product of the code in its entirety.

5.3 Physics

Example Driving:

```python
# Setting up the camera
flatRod(camera, car, distance = 2)

# Steering
def driving(model, p0, h0, speed0):
    model.angle = integral(delta + decay)
    forward = (speed0 * 4) + engineForce *
                integral(dspeed)
    # Force on vehicle
    forward = (speed0 * 4) + engineForce *
              integral(dspeed)

# Vehicle velocity
```

The example driving code uses the FRP library to create a simple driving simulation. The steering is done by incrementing the car's angle based on a decay value and the speed is adjusted by a force calculation. The force calculation takes into account the speed and engine force and is integrated over time to give the vehicle a realistic appearance.
The driving game teaches students about basic physics principles such as friction, velocity, and acceleration, all of which are important in game development. It also strongly reinforces the declarative nature of FRP by allowing us to ignore loops, which can be confusing to novice programmers. The driving function is a perfect example of this. Students can declare how the car will move in a way that makes sense to them without having to fiddle with loops.

Notice the use of the integral function in this example. On line 17 it is used to determine the car’s heading from its speed and angle, and on line 20 it is used to determine the car’s position from its velocity.

5.4 Art

Example Volcano

```python
# Make the scene
ambientLight(color = color(.5,.5,.5))
directionalLight(color = orange, hpr = hpr(0,3,0))
pandaModel("volcanoModel", texture = "volcanoModelTexture.png", size = .9, position = p3(0,0,-2.8))
rectangle(p3(-25,-25,d), p3(-25,25,d), p3(25,-25,d), texture = "grass")

# How to erupt
def erupt(m,v):
    h = randomRange(0,2*pi)
p = randomRange(degrees(35),degrees(50))
s = sphere(color = orange, size = randomRange(.07,.3))
launch(s, neck, hprToP3(hpr(h,p,0))*3)
when(s, getZ(s.position) <= -2, flower)

# Create flowers at landing point
def flower(m,v):
    exit(m)
pandaModel("flower#3", texture = "flowerPurple.png", position = now(m.position),
size = randomRange(.3,.9))

# Rotate camera around center
camera.position = p3c(25,time/4, 1)
camera.hpr = hpr(h + pi/2,0,0)
```

The camp teaches the students to create their own models using Blender. We facilitate this by making it very easy to import your own models and textures into the software. Lines 4 and 17 in this example demonstrate how this is done. The first defines a volcano, and the second defines a flower, complete with both a model file and a texture image. The image below was produced by this example.

6. EXPERIENCES

The camp started in 2006 and has seen growing attendance since. Typically one third the students are girls and a quarter of the students have attended the camp in the past. Occasionally a veteran student will make the transition to mentor.

As our intention is to help students explore a variety of subject material while fostering creativity, we do not utilize testing as a measure of the camp’s effectiveness. Instead, we rely on feedback from the students and that of their parents. For example, when a student was asked why they preferred our camp over another they attended that same summer, they responded, “It’s just funner...I can do a lot more here”.

Recently, a parent stated that, “While the camp is only a week long, the quality of knowledge gained has been substantial...Having the opportunity to study under and work with college professors and students has been monumental for [my students]”.

We rely on the feedback of the mentors as well, since they work closely with the students throughout the duration of the camp. One mentor that was studying computer science at WSCU said, “The students had a lot of fun, and it was a good experience for them to use a full programming language”. A mentor that had just graduated with a degree in computer science said, “I’ve had a lot of experience teaching college students, but not that age group...It solidified the fact that teaching has always been an option for me, even grade school”. Obviously the mentors benefit from the camp as well.

7. RELATED WORK

There are many programming languages that specifically target novice programmers. For example, Alice was designed to teach event driven object oriented programming to children by creating interactive stories. And Scratch is a blocks-based graphical programming language used to create animated stories and games. These projects hide the underlying mathematics from the users and do not provide the full set of features that a more advanced programming language offers. In contrast, our method emphasizes mathematics within a full featured language.

A system intended for both novices and experts is Game Maker. It accommodates the creation of cross-platform video games using drag and drop and/or a scripting lan-
guage known as Game Maker Language, which can be used to develop more advanced games. Game Maker was designed to allow novice programmers to be able to make computer games without much programming knowledge via a GUI. Though more feature rich than Alice and Scratch, Game Maker is still domain specific and relies heavily on a GUI to accommodate novice users. Conversely, our method is purely text based.

Unreal[8] is a game engine that features an advanced suite of tools for creating 3D models, textures, and other game assets. It supports C++ for UI elements and classes, and a graphic scripting language called Blueprints. It is intended for beginner and advanced programmers. A project that is more similar to ours is PyGame[14]. PyGame is a set of modules for Python specifically designed for writing games. It is a full featured game engine designed for users that have more programming experience than our target demographic. The PyFRP software simplifies the game engine by using continuous time where Unreal and PyGame use explicit time and event handlers to run the game, something that may be unintuitive for novices.

Lastly there is Elm[4]. Elm is a standalone FRP language designed to make interactive web programming easier. Elm, like Game Maker, is more robust than Alice and Scratch but lacks the features a “real” language provides.

There is a variety of math and computer science summer camps for K-12 students, such as those offered by Digital Media Academy[1], iD Tech[12], Classroom Antics[2], and ProjectFun[5]. These camps provide a serious programming experience but lack the interdisciplinary aspect of our camp. While we agree that immersing students in an instructed environment is more engaging and productive than simply giving them the material, we believe the high-level, declarative syntax of FRP allows students to learn a wider variety of material in a shorter period of time.

A drawback of our method is that it was designed for and tested in a classroom environment. Though our code is publicly available online for anyone to download, it is intended to be used in an instructed environment.

8. FUTURE WORK

As mentioned in Section 1 we are currently porting our library to systems similar to Panda3D, namely, Unity-3D[13]. As development and support of Panda3D seems to be in decline, Unity-3D is an appealing alternative. It offers much of the same functionality as Panda3D and more. As mentioned in Section 1 some advanced students and mentors work directly with the game engine. By moving to Unity-3D, they will be learning to use a tool that has a strong presence in industry.

The modularity of our library allows it to interact with systems other than game engines. For example, we have coupled it with micro-controllers and various IO devices. This was done in a collaborative effort between the computer science and art departments at WSCU for a course on interactive art, an emerging genre. This work is ongoing.

Our system currently targets the PC, Mac, and Linux operating systems. We would like to further extend it to support Windows Mobile, iOS, and Android as well.

As touched on in Section 1 our method was designed for a classroom environment. To make it accessible outside the classroom, we would like to create videos to supplement the existing documentation and tutorials.

9. CONCLUSIONS

In this paper we discussed our use of FRP to deliver a summer camp for students in grades 8 through 12. Our method was used to cover a variety of topics, many of which relied heavily on math. We demonstrated how FRP in the Python environment significantly increases the ease of programming interactive 3D environments, abstracting away the non-trivial boiler plate code without overly constraining the capabilities of the system. In particular, we showed the ease of use of continuous mathematical operators in our system. We also emphasized the use of mentors in our camp, which allows us to accommodate a variety of learning styles, speeds, and interests. Based on our experiences and the feedback we have received, we believe our method is a fun and effective means of interdisciplinary study.

The source code, documentation, and camp material are available at our website, www.reactive-engine.org. This work is supported by NSF Grant 1302327.

10. REFERENCES