TEACHING INTRODUCTORY COMPUTER GRAPHICS WITH THE PROCESSING LANGUAGE

Dino Schweitzer, Jeff Boleng, Paul Graham
United States Air Force Academy, CO 80840
dino.schweitzer@usa.edu

ABSTRACT
Different approaches to teaching introductory computer graphics have been documented in the literature over the years. Different languages, environments, API’s, and graphical standards have been proposed. Some unusual approaches have included using games and gaming engines, animation and simulations, and even spreadsheets. A common characterization of the different teaching methods is a bottom-up approach (basic algorithms and primitives first) versus a top-down approach (functional modules of an application). Many texts use OpenGL as the API of choice for teaching graphical functions and capabilities in a top-down means. One problem with this approach is the student becomes API-centric, and sees graphics concepts through the eyes of the system being used. Criticism of the bottom up approach is that students need to learn the math and low-level algorithms before they can begin creating useful and interesting applications. At the United States Air Force Academy, we have taught the introductory computer graphics course using both approaches. In the most recent implementation, we chose a more bottom up approach to emphasize fundamental algorithms and concepts in an API-agnostic fashion. To allow students to quickly advance to interesting applications, we chose the open-source Processing language and environment. Processing (www.processing.org) was developed at MIT as a way to teach the fundamentals of computer programming in a visual context. It is a Java-based language without requiring knowledge or experience with object-oriented concepts. It is well suited for people with minimal programming experience, and as a rapid prototyping tool. The ease of quickly creating interactive graphical applications makes it a good choice for learning fundamental graphic algorithms and applying them to interesting applications. This paper will describe our approach of using Processing to teach graphics, include sample programming projects, and report on student attitudes and performance.

INTRODUCTION
Computer Graphics is fun to teach. Students find the topics interesting; there are many occasions to tell anecdotes about animations, movies, and games; class projects provide opportunities to express creativity; and students are self-motivated to create their own computer generated images and animations. Our experience is that students are willing to spend many more hours working on their computer graphic projects than other project-based CS courses. In addition, the material is inherently visual which makes for more lively classroom presentations.

As with many CS courses, different educators have proposed different approaches to teaching computer graphics: different languages, different systems, an API approach, a tool-based approach, a web-based approach, game-based approach, a survey-based approach, and even teaching it with spreadsheets [1-6]. One general categorization is a “top down” versus a “bottom up” approach [1]. A top down approach is defined as an API-driven approach in which the emphasis is on students learning and using an existing API, such as OpenGL, to quickly write high-level programs and generate interesting images and graphic applications. The advantages of this approach are that students are motivated to develop graphically interesting programs and learn an API common in the field. The disadvantage is that students can become “API-centric” and not understand the underlying concepts.

The bottom up approach focuses on algorithms and the underpinning concepts of the field. In the extreme case, students write all of their computer graphic programs from scratch, down to setting individual pixels in line drawing algorithms. The advantages of the bottom up approach are that students have a better understanding of underlying concepts and can apply these to follow-on courses. However, in an undergraduate program in which students take a single computer graphics course, if the majority of
time is spent focusing on low-level programming problems, students do not get the opportunity to
develop more motivational projects such as sophisticated interactive games.

OUR COURSE
At USAFA, we have taught the course using both approaches, partially depending on the biases of the instructor for the specific offering. Students take the course in their Junior or Senior year with the only pre-requisite being CS2. This past spring semester, the course was offered using a hybrid approach – algorithms and underlying concepts were taught but students did not implement low-level algorithms such as fill, line drawing, and rasterization. They did not use a high-level API, but rather used the fundamental drawing capabilities built into a language. They used an existing modeling tool (Blender) for a modeling exercise, and they wrote programs to do realistic shading, rendering, and stereoscopic displays. The complete list of lesson topics is shown in Table 1.

<table>
<thead>
<tr>
<th>Intro to Computer Graphics</th>
<th>Visual Perception</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graphic Primitives</td>
<td>Basic Algorithms</td>
<td>User Interaction</td>
</tr>
<tr>
<td>Math Review</td>
<td>Transformations</td>
<td>Transformations II</td>
</tr>
<tr>
<td>Viewing Transformation</td>
<td>3D Viewing</td>
<td>Modeling</td>
</tr>
<tr>
<td>Procedural Modeling</td>
<td>Modeling Tool</td>
<td>Hidden Surfaces</td>
</tr>
<tr>
<td>Shading</td>
<td>Advanced Shading</td>
<td>Ray Tracing Review</td>
</tr>
<tr>
<td>Increased Realism</td>
<td>Intro to Animation</td>
<td>Animation II</td>
</tr>
<tr>
<td>Virtual Reality</td>
<td>Image Formats</td>
<td>Image Processing</td>
</tr>
<tr>
<td>Image Compositing</td>
<td>Visualization</td>
<td>Non-photorealistic Rendering</td>
</tr>
<tr>
<td>OpenGL</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1. List of topics taught in Introductory Computer Graphics class

For the final project in the course, students worked in teams to create an immersive display cave for an Unmanned Aerial Vehicle (UAV) training application. The project consisted of hardware, software, modeling, and interfacing with existing systems.

An early decision in preparing the course was the choice of textbook. In the previous offering of the course, a top down approach based on OpenGL, students used the API reference books as their only text. For the current hybrid offering, the instructor felt it important for students to have a written reference of the concepts and algorithms being taught. After reviewing available texts, it was decided that none of the current texts in the market exactly hit the right level. They were either too API-centric or too detailed for an undergraduate computer graphics overview course. Thus, the instructor wrote a set of readings for the students, one per lesson that covered the material at the desired level.

Another interesting aspect of the course was an emphasis on hands-on instruction. Every lesson included an interactive tool for learning a central concept in the day’s lecture, such as a specific algorithm, setting parameters in shading models, and morphing shapes and faces. 12 minutes of every class was dedicated to students using the tool, experimenting with its capabilities, and answering an associated “challenge question” on the concept. Figure 1 shows an example of one of the tools.
LANGUAGE CHOICE

With the decision not to use an existing API, such as OpenGL, there was a lot of flexibility in what language to teach the course in. While most of the students were Computer Science majors, there were some students with minimal programming experience. The goals were to choose a language that had sufficient graphics functionality available, was easy to learn (for non-programmers) and use, and supported simple interaction and animation. The Processing language and IDE were chosen to meet these objectives.

Processing (www.processing.org) is a Java-based open source programming language and environment specifically designed for creating images, animations, and interactions. It was created by Ben Fry and Casey Reas at MIT as a tool for artists and designers to quickly generate visual applications with a minimal of programming experience. While it is built on Java, it does not require knowledge of the object-oriented paradigm, and greatly simplifies programming requirements such as classes, paths, and environment variables. It also has a fairly sophisticated 2D and 3D library of graphic routines available for drawing shapes and manipulating images. The example in Figure 2 shows a simple Processing program and resulting screen to draw shapes.
To provide animation and interaction, programs are written in *interactive mode* which uses the pre-defined functions *setup* and *draw*. Setup is called once when the program begins to perform any initialization. The draw routine is continually called so that any changes will appear animated. Figure 3 is an example of a simple bouncing ball:

```
int y = 100; //global variable for ball's y position
int dy = 5;  //global variable for ball's y velocity
void setup(){
    size(200,200);  //open a window
}

void draw(){
    background(0);  //clear screen to black
    ellipse(100,y,50,50); //draw circle at current y position
    y = y + dy; //update y position to new position
    if((y < 50) || (y > 150)) //if ball at top or bottom of
        dy = -dy;     // reverse direction
}
```

Figure 3. Simple bouncing ball in Processing.

For user interactions, call-back routines are available for events such as *mousePressed*, *mouseDragged*, and *keyPressed*. A well-documented reference to the available functions in Processing is provided under the help menu in the Program Development Environment (PDE). When desired, a program can be easily exported as a standalone Java application or as a Java applet for embedding in web pages. The language includes a number of libraries for different capabilities, and, if necessary, straight Java code can be embedded in the program.

**USING PROCESSING IN THE COURSE**

Students are introduced to Processing at the start of the course, and examples are provided throughout. The customized readings for the course contain all of the examples in Processing, and have sections dedicated to how to accomplish certain activities, such as user interaction and graphic primitives. In addition, all of the daily interactive tools described above are written in Processing, and students have access to the documented source code to see how certain interactions are accomplished. They are encouraged to copy useful sections of code, such as a button press routine or slider bar, into their own programs.

The course required students to write three separate programming exercises in addition to the modeling exercise using Blender and the final integration project. The three projects and associated course concepts being reinforced are listed along with sample outputs in Figure 4:

<table>
<thead>
<tr>
<th>Project</th>
<th>Course Concepts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic Paint Program</td>
<td>Graphic primitives, simple drawing, basic user interaction</td>
</tr>
<tr>
<td>3D Anaglyphs</td>
<td>3D transformations, binocular stereopsis, perspective, advanced user interaction</td>
</tr>
<tr>
<td>Ray Tracing</td>
<td>Advanced rendering, simple models, hidden surfaces, shading calculations, shadows</td>
</tr>
</tbody>
</table>
RESULTS
The class size was relatively small with only nine students, so it is difficult to make definitive statements as to the effectiveness of the approach with such a sample size. Most students successfully completed the required exercises. The average scores on the first two were 95% and 85% respectively. The third exercise, ray tracing, was more challenging and students only averaged 75%. Students who did not successfully complete it agreed the problems were with their logic, and were not related to the language.

On the course critique, students were asked to rate the use of Processing as the required programming language. On a scale of 1-5 (5 being the highest), the average student rating was 4. Comments on the language included:
- Fun PEX’s (programming exercises)
- Processing did not have much overhead
- Need something with a better debugger

The last comment is based on the fact that the Processing environment does not have an integrated debugger, and students are forced to use more traditional techniques such as `println`’s. While understanding a sophisticated IDE and debugger is important for future computer scientists developing complex systems, learning a new system can be cumbersome and take time away from the underlying algorithms. Our students are exposed to such IDE’s in other classes.

In summary, the use of Processing for the introductory Computer Graphics course was deemed a success. The students were able to complete relatively complex programs that reinforced graphic concepts and not get bogged down with a complicated language or environment to learn. The aspects of the language that support user interaction and animation were beneficial in reinforcing certain concepts in the course. By providing several examples through the readings and daily interactive tools, students had no difficulty understanding the programming aspects of the exercises (although several struggled with the complexity of the ray tracing exercise). While the higher level functions of a sophisticated API are not available, the use of a language such as Processing worked well with this hybrid approach.

REFERENCES


