The Art of Graphics Programming

Technical Context

From main() to Processing

Basic Architectural Patterns in Graphics Software

Patrick Hebron
NYU ITP

A pure C, C++ or Java program starts with something like this:

```c
int main(int argc, const char * argv[]) {
    for(int i = 0; i < 10; i++) {
        int numsq = i * i;
        printf("%i squared is %i.\n", i, numsq);
    }
    return 0;
}
```

The program enters the main function, executes each command in order and then exits.

This structure is linear and straightforward.
The above program will run to completion in a fraction of a second.

However, we don't want our graphics program to exit in a fraction of a second.

So, we need to expand upon this architecture to make room
for the temporal nature of graphics software.

Processing, oF and Cinder use a “setup-update-draw” design pattern:

```
int counter;

void setup() {
    counter = 0;
}

void draw() {
    int numsq = counter * counter;
    printf("%i squared is %i.\n", counter, numsq);
    counter++;
}
```

This pattern divides software tasks into two basic categories:

- Steps performed once when the application is launched: setup()
- Steps performed repeatedly throughout application runtime: update() and draw()

(Processing consolidates update() and draw() into a single function)

This design pattern is just one of many possible,
but is well-suited to graphics software.

But, main() is still under the hood:

```
int main(int argc, const char * argv[]) {
    // The while-loop below will run forever.
    // We'll eventually need to tie this variable to a keyboard "esc" event.
    bool keepRunning = true;

    setup();
    while( keepRunning == true ) {
        // Processing consolidates update() into draw()
        //update();
        draw();
    }
    // In C/C++, a third category - steps you perform once at the end - can be
    // helpful for proper memory management:
    //shutdown();

    return 0;
}
```

As shown above, the setup-update-draw pattern is created
by adding a single while() loop to main().

The program will continue to call draw() until the value of keepRunning is set to false.
We could also add a delay timer to the loop to ensure that frame drawing is evenly spaced.

We can now pass successive frames to OpenGL.