Work done so far:

- Sorted out OpenMP on OSX, and png parsing with C++, which was a lot harder than anticipated.
- Written sequential convolution algorithm using an edge detection mask
- Parallelized convolution with OpenMP and ISPC
- Written sequential ray tracing algorithm for single or multiple light sources
- Parallelized sequential ray tracing across multiple light sources with OpenMP
- Ran (but not analyzed) sequential and parallel implementations to see speedup

Here is a recap of our schedule:

11/16: Our sequential convolution algorithm would be ready, and we would be part way through parallelizing this. We would begin writing a basic ray tracing algorithm.

11/23: Parallelized convolution would be finished on at least one framework, our basic ray tracing algorithm would be ready, and we would be starting our parallel approaches for the ray tracing algorithm.

11/30: The parallelized ray tracing algorithm would be ready on at least one framework. We would begin comparing results and analyzing data.

12/7: At this stage, we should have finished comparing results, hopefully for all the frameworks planned, and begin trying to achieve stretch goals or any unreached goals

12/14: Achieve stretch goals

12/17: Finalize Writeup Report

12/18: Finish up final touches on presentation

According to our schedule, we are on schedule. We have parallelized both convolution and ray tracing on at least one framework and will finish the other frameworks and analyze results soon.

Updated schedule:
12/3:
> Emily: Create speedup graphs of sequential and OpenMP convolution, work on CUDA implementation of convolution and/or fix compiler bug
> Rahul: implement pixel-based parallelization, create speedup graphs of sequential and openMP ray tracing

12/7:
> Emily: Finish CUDA implementation of convolution, create speedup graphs of CUDA and ISPC implementation of convolution.
> Rahul: Finish CUDA implementation of ray-tracing, create speedup graphs of CUDA and ISPC implementation of convolution.

12/10:
> Emily: Start working on live demo: Live demo will consist of taking screencap, running convolution+raytracing pipeline on it, and displaying the modified frame in real time. By this point I want to figure out how to do the screencap + display
> Rahul: Generate sufficient test cases for the project, may be able to show these static visuals as well for demo.

12/14:
> Emily: Get something working for a demo and also compare results of parallelization.
> Rahul: Put together presentation notes and slides, give time to fix any other bugs or make improvements to our builds for the demo

12/17: Finalize report
12/18: Present

We believe we can produce our deliverables, which, to reiterate, are:

- ~~Implement sequential image convolution and ray tracing alg~~
- Parallelize convolution and ray tracing alg with CUDA, IS~~PC~~, ~~OpenMP~~
- Compare performance across methods, parallelizing image convolution and ray tracing separately will allow us to observe the individual performance gains in each)
- Achieve speed fast enough to support high fps gaming
- Present speedup graphs vs. implementations and processor counts

Although we have only crossed off 1.5 things from this list, most of the list is analytical and can be done quickly after we have our base implementation. Image convolution is very fast and can definitely support high fps gaming. Raytracing seems to be the costly step as thousands of rays can get propagated across the image at once. At lower ray counts and moderate 300x300 pixel images, the sequential implementation may take a few milliseconds to render a lightsource, meaning a successful parallel implementation that parallelizes across rays should be able to support 60FPS. For larger images, higher ray counts are needed as propagations may travel further, meaning accuracy in the ray count is needed at such depths. On a larger 1200x650 pixel image one test yielded the sequential version taking close to 0.8 seconds to trace rays from one light source. We may need to tweak some parameters and/or ray propagation functionality to get this base time down, as parallelizing across multiple light sources yielded roughly 3.5x speedup with naive testing.

Concerns:

1. A bug with compiler or library or operating system makes OpenMP not runnable on one computer. This bug makes it so that when compiling using llvm/clang or with the commands from the makefile, an error called "more atoms allocated than expected" occurs. An online search yields a problem with OSX's XCode version, which seems irrelevant since I am not using xcode, and because both partners' xcode versions are the same. The sequential implementation of convolution works fine using g++, but the raytracing sequential implementation yields more compilation errors that will have to be resolved. Again, this is all on one machine. We may work around this by doing most of the testing on the other machine.
2. ISPC doesn't take in png::image types or 2D variable-length arrays. We were unable to find a sufficient way to pass in the image to ISPC without copying the entire image to a new 1D array and then copying the results back when done. This will likely cause significant overhead, and we aren't sure whether to include this in speedup calculations.
3. Raytracing with multiple light sources is memory intensive, and running a significantly sized image with 2 or more light sources causes memory overflow. This is a problem because currently parallelization of raytracing is parallelizing across light sources, which means that for significantly sized images, we will be unable to parallelize at all. We are resolving this by parallelizing across pixels in addition to light sources.
4. Stretch goals still unclear. We'd like to create a live demo where frames of a screen or application are updated with light in real time. One idea is to show a side by side of frames rolling without the ray tracing and frames rolling with a single source of light at a fixed position on the screen throughout the scroll.
5. Continuing previous point: In addition, we would like to compare our implementation against some other methods of producing the same result and see if ours is faster. The problem with this is to locate something to compare against – maybe an implementation in Halide? And also how we could meaningfully measure the time difference.