Web Image Analysis

Introduction:

Organization: Stony Brook University Biomedical Informatics

Name: Gunjan Shrivastava

This Project is done as a part of GSoC 2016 under the supervision of Prof. Jonas Almeida. The initial proposal can be found here https://summerofcode.withgoogle.com/projects/#6491824516497408

Contact Info: jonas.almeida@stonybrook.edu , gunjan.sh123@gmail.com

I would like to thank GSoC community for giving me this opportunity to work on this project. I would also like to thank my mentor and my organization who provided me a lot of support and helped me to stay positive. Lastly, I would like to thank the Emscripten group who took their time to answer my questions and reply and helped me to solve many bugs. I have learned many things from this project and hope to continue my contributions.

Important links:

GitHub: https://github.com/gunjan-sh/webimg
Git Commits(master branch): https://github.com/gunjan-sh/webimg/commits/master
Git Commits (gh branch): https://github.com/gunjan-sh/webimg/commits/gh-pages
Served at https://gunjan-sh.github.io/webimg/
Log Book:
https://docs.google.com/document/d/1mKnoXTemK6b6tyTjrQNtfzUNC6V0xdjMi9zJ5tAtCOU/edit
GSoC Proposal: https://summerofcode.withgoogle.com/projects/#6491824516497408
Emscripten-- https://kripken.github.io/emscripten-site/docs/getting_started/Tutorial.html
Emscripten Google Group: https://groups.google.com/forum/#!forum/emscripten-discuss
Webassembly Module - https://github.com/WebAssembly/design/blob/master/Modules.md

Objective:

The main task of this project is to develop a web API for Image analysis for algorithms like Image segmentation, Edge detection etc. to make these transformation easier. With the development of new web platform “Webassemly” which will serve as a new language as well as a compile target to transform c/c++ to low level binary format, this task could be made much easier. We tried to experiment how it can be done and evaluate the performance of the three web languages namely, JS, Asm.js and Webassembly. The final product of this experiment would be a web page with three paths to analyze the image (Via. js, Asm.js and webassembly) and give you the fastest of them.

What is Webassembly?

Webassembly is a joint initiation by Mozilla, Microsoft, Google and Apple to set a new standard for the Web. Its goal is to be load-time efficient binary compiler target to achieve near-native computational performance. WebAssembly code defines an AST (Abstract Syntax Tree) represented in a binary format. It is a low-level asm.js subset of Java Script. It's still in its initial phase of development so while using it we may face a lot many errors even when the C/C++ codes are running correctly.

Browser Support for Webassembly:

Firefox nightly and Canary (Chrome) are the only browsers which support webassembly currently. If you recently downloaded any of these, you need to go to their config setting and enable webassembly support. For enabling webassembly in Firefox nightly you can do the following:

- Open a new tab in firefox nightly and open 'about:config' , then set javascript.options.wasm to true.

How to get started!

The first step is to get acquainted with Emscripten, which is the tool for converting C/C++ code to asm.js. An extension to it is Binaryen which compiles C/C++ to Wasm.js. Below is the stepwise description of how to install it and the problems I faced. I worked in Linux environment so all the commands and description refers to it.

Pre-download installs:

```
#Update the package lists
sudo apt-get update
# Install *gcc* (and related dependencies)
sudo apt-get install build-essential
# Install cmake
sudo apt-get install cmake
# Install Python
sudo apt-get install python2.7
# Install node.js
sudo apt-get install nodejs
# Install Java (optional, only needed for Closure Compiler minification)
sudo apt-get install default-jre
```

Download:

Emscripten

You can directly download portable Emscripten SDK for Linux from https://kripken.github.io/emscriptensite/docs/getting_started/downloads.html. After extracting the file go inside the directory and run the following commands:

```
cd Downloads/emsdk_portable
./emsdk update
# Download and install the latest SDK tools.
./emsdk install latest
# Set up the compiler configuration to point to the "latest" SDK.
./emsdk activate latest
# Linux/Mac OS X only: Set the current Emscripten path
source ./emsdk_env.sh
```

Note: these commands will install and activate the latest master branch. You need to run the last four commands every time you start to use emscripten for the first session.

A lot of problem which I faced were because I used master branch and not the incoming branch. The incoming branch of Emscripten has the latest developments after which they get merged to the master branch. So to work with the latest issues so that we don't face errors, you must install the 'sdk-incoming-64bit' branch of Emscripten. That can be done in the following way.

```
./emsdk update
./emsdk install sdk-incoming-64bit
./emsdk activate sdk-incoming-64bit
source ./emsdk_env.sh
```

OpenCv Installation:

Followed: https://github.com/kakukogou/opencv/tree/opencvjs

Binaryen:

Binaryen is a compiler and toolchain infrastructure library for WebAssembly, written in C++. It can be used with Emscripten's incoming branch and making the flag equal to 1 like follows:

./emcc sobel.c -o sobel.html -s BINARYEN=1

Installation: https://github.com/kripken/emscripten/wiki/WebAssembly

Running the first program:

In the incoming directory run the following command to verify the install.

```
./emcc –v
```

As an introductory tutorial Emscripten official site has given Hello World tutorial.

```
./emcc tests/hello_world.c -o hello.js
```

This would generate a asm.js file. If you want to generate a html file which can run on your browser:

```
./emcc tests/hello_world.c –o hello.html
```

This would give us an html file as well as a .js file.

```
firefox hello.html
```

this will open the html file in the browser.

This is a simple test program which will show you a .js file being generated from a C/C++ code, which is pretty amazing. But I took this to next step by trying to convert an Edge detection code written in C/C++ to .js.

Tasks Completed:

The first task was building a web page using web picker API. I attended a workshop on Java script conducted by my mentor which helped me a lot in developing this.

I wrote the Sobel Edge Detection algorithm in 4 languages C, C++, java script and MATLAB and compiled the C and C++ code to Asm.js and wasm.js. I also compiled them to the html format.

Task Done:

1. JS code for image processing algorithm (Edge detection using Sobel Filter).
2. Matlab code for Edge Detection
3. C++ code for Edge detection using OpenCv.
4. Matlab code for Segmentation.
5. Setup Emscripten and tested for simple C codes.
6. Setup Emscripten for OpenCv platform.
7. Create a code in asm.js (using Emscripten)
8. Writing Hello World! in WebAssembly.
9. C code for Edge detection without openCV
10. Compiled Asm.js from C code
11. Compiled Wasm.js, wasm.html from C code

Working on:

1. Compare the runtime performance
2. Create a JS application with 3 buttons, each to use the different implementation in JS, Asm.js, WebAssembly.

Problems faced:

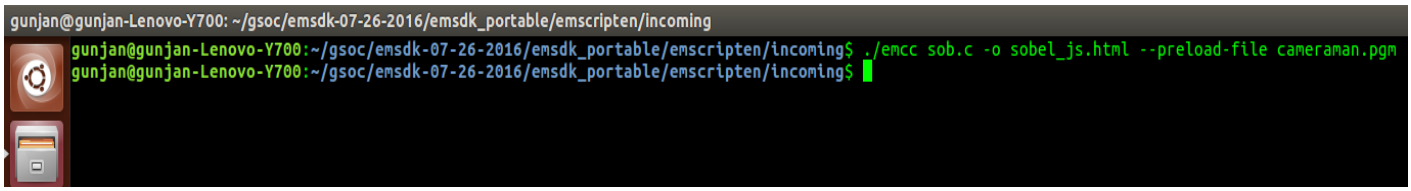
1. Installing Opencv: OpenCv is a very useful library when we talk about image processing. It simplifies a major coding part mainly in reading the image, loading the image saving it, etc. When I installed OpenCv I ran into the following error which took a lot of my time to solve but I couldn't. Finally I posted my query to the Emscripten group and in reply I was told to use the 'incoming branch' and not the master branch. After installing the incoming branch I got an error saying it can't find the header files. This issue is still unresolved and I am working on it.
   https://groups.google.com/d/msg/emscripten-discuss/uaRH2R62WOk/hujMVl-WAQAJ
2. I tried to run the segmentation demos which is available here so that I can see if it could generate asm.js code via Emscripten. But I faced the following error, which is still unresolved. https://groups.google.com/d/msg/emscripten-discuss/uaRH2R62WOk/BaHVxOkfBwAJ

3. As I was getting a lot of errors using OpenCv, I decided to write new C code for Sobel edge detection without using any library. The program asks for the input file name. The code gets compiled correctly by running the following command:
   ./emcc sobel.c -o sobel.js
4.

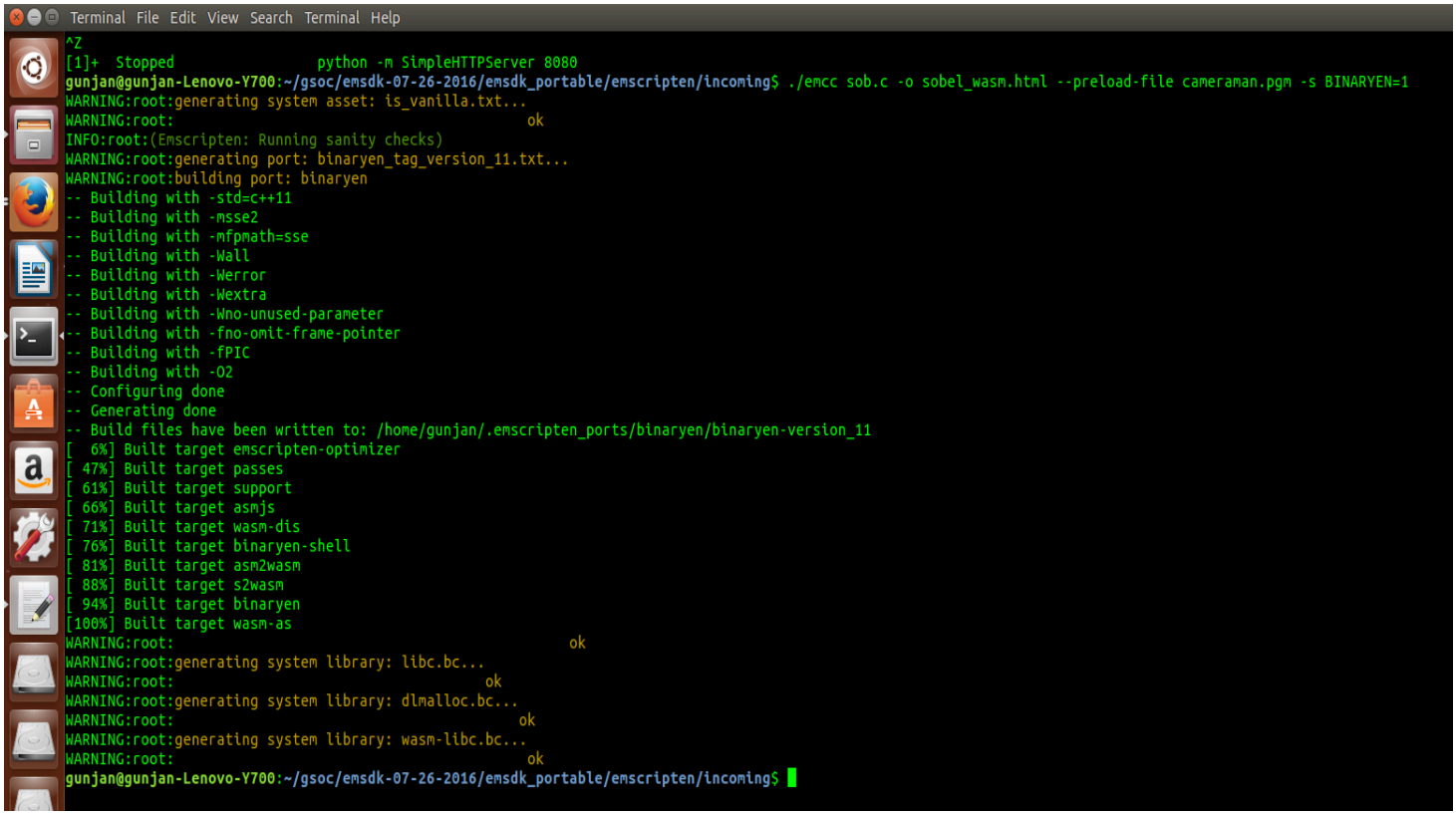
   This generates the asm.js file which when executed using 'node' like:
   node sobel.js
   It is not able to read the input when the input file name is entered. I posted the error to Emscripten group (https://groups.google.com/forum/#!topic/emscripten-discuss/LuMkNzLuUB8) and in reply I came to now that Emscripten uses a virtual file system (so that it can work on the web). So you must either preload or embed the files you want to access,
   http://kripken.github.io/emscripten-site/docs/porting/files/packaging_files.html
   or use NODEFS which in node.js specifically lets you directly access files,
   http://kripken.github.io/emscripten-site/docs/api_reference/Filesystem-API.html#filesystem-api-nodefs

   A tutorial preloading the data file can be found here:
   http://kripken.github.io/emscripten-site/docs/getting_started/Tutorial.html#tutorial-files
   Unfortunately, when I tried to preload the image, it still could not find the image.

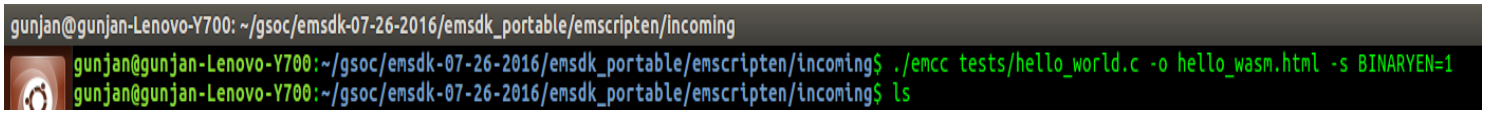
Snapshots



Generating sobel.html with file preloading. This will give asm.js and html file



Generating sobel_wasm.html with file preloading, using Binaryen



Generating Hello_world in wasm.htm with Binaryen

Conclusion

The experiment was only half successful as it was unable to produce asm.js and wasm.js code which generate correct result. The reason being Emscripten and Binaryen are still in there developing phase and Webassembly has still a long way to go to be used as a working language. Through this experiment I learned that Webassembly can be a very powerful tool as it breaks the boundaries between the native languages and Web languages.

Future work

The last step to complete this work would be to resolve the issues, most important being reading the input image. After this is resolved, the performance evaluation between the three routes can be easily done.

One can take the work one step further by building the algorithm for colored images without using any library and convert it into asm.js and wasm.js for measuring performances.

References:

1. https://medium.com/javascript-scene/what-is-webassembly-the-dawn-of-a-new-era-61256ec5a8f6#.3yxphom2e
2. https://hacks.mozilla.org/2015/12/compiling-to-webassembly-its-happening/
3. https://github.com/kripken/emscripten/wiki/WebAssembly
4. https://kripken.github.io/talks/wasm.html#/
5. https://wasm.news/
6. https://github.com/mayflower/webassembly
7. https://github.com/WebAssembly/binaryen#building
8. http://cultureofdevelopment.com/blog/build-your-first-thing-with-web-assembly/
9. https://www.sitepoint.com/understanding-asm-js/
10. Webassembly Module- https://github.com/WebAssembly/design/blob/master/Modules.md

Good Reads

https://hacks.mozilla.org/2016/03/a-webassembly-milestone/

https://www.infoq.com/presentations/webassembly