Virtual Machine Bindings

Overview

Virtual machines (VMs) are a common feature of modern computing that allow engineers develop systems that can run on a variety of architectures. As VMs proliferate, so do the support systems for working with them. One critical area is facilitating communication between the ***host*** machine and the ***guest*** VM. In this activity, students will develop routines in C++ targeting the WebAssembly (Wasm) VM together with an HTML document to load and display images.

VM Bindings

In heterogenous systems, ***bindings*** provide interoperability for otherwise incompatible languages. This mapping allows translation of meaning from one environment into another. Common binding categories include…

1) Data (a C++ **`int`** into a JavaScript **`number`**)
2) Procedure calls (executing a **C function** in **Python script**)
3) General Associations (Creating a **socket** that represents an **IP address and port**)

Project Structure

This project involves three major components:

1) Build the HTML file (with JS "glue") to invoke the C/C++ functions
2) Write C++ function(s) to display image from header in a Canvas element
3) Write C++ function(s) to fetch an image from a URL, then display in a Canvas element

Specification

Students will write a WebAssembly module in C++ which will load and display images. One image will be loaded from a header file, while another will be fetched via HTTP and displayed.

Image from Header File ***Image from URL***

An HTML file named **`cpp_loader.html`** should be constructed to demonstrate functionality. It should use **`gator.rgba`** as the image resource. Only the necessary "glue" elements should be written in JavaScript in this activity within the HTML file. Additionally, all JavaScript must be written inline within script tags (i.e., no external JavaScript sources may be included!) The purpose of this assignment is to learn language bindings and use them – not to write an image parser in JavaScript or use JavaScript libraries. **NOTE**: The Wasm VM is a ***little-endian*** architecture (like x86) – **which is not network byte order**!

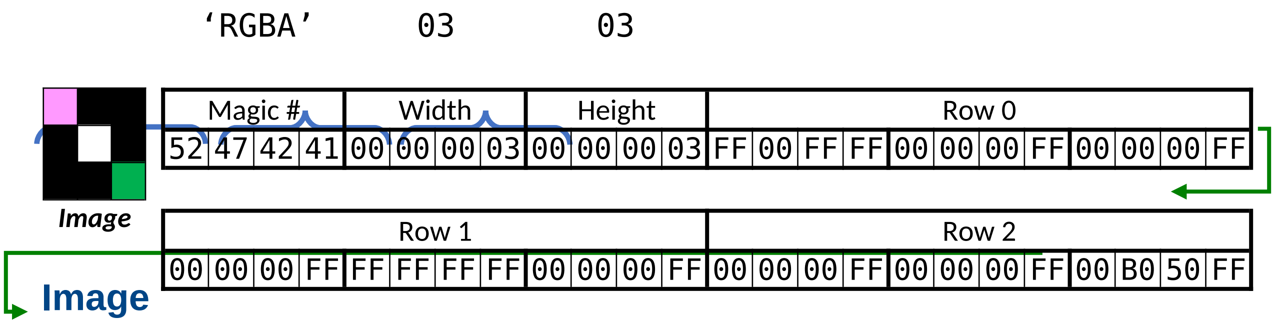
Header File

The module should use the **`gator.h`** header file (included) as image data. It contains three global variables that define the image, where "0" represents **black** and "1" is **transparent**:

`gatorWidth` – the width, in pixels, of the image
`gatorHeight` – the height, in pixels, of the image
`gatorImage` – the array of pixel values; length is **`gatorWidth * gatorHeight`**

Data File Format

This activity involves reading the RGBA true-color (**.rgba**) file format. The first four bytes are the ASCII characters 'RGBA', which is the file type's ***magic number***. This is followed by the width and height (4 bytes each, ***big-endian***). Thereafter, each four bytes is one pixel, row-after-row, with each band having a value of 0-255. Here's an example from a 3x3 image:



'RGBA'
03
03
Image
Magic #
Width
Height
Row 0
52 47 42 41
00 00 00 03
00 00 00 03
FF 00 FF FF 00 00 00 FF 00 00 00 FF
Row 1
Row 2
00 00 00 FF FF FF FF FF 00 00 00 FF
00 00 00 FF 00 00 00 FF 00 B0 50 FF

Image Functions

The module should provide functionality to load an image and display it within a Canvas HTML element. The module should export the following functions, <u>at a minimum</u>:

```
void displayHeaderImage(std::string id)
```

Displays the image from the header file in the `Canvas` object with the specified **`id`**.

```
void initiateImageLoad(std::string url, std::string id)
```

Starts load of RGBA file from **`url`**; sets callback to display in `Canvas` object with specified **`id`**.

<u>The following functions are also recommended, but not required:</u>

```
void copyToCanvas(int width, int height, uint8_t *pixels, const char
*id)
```

Displays image (**`width`**, **`height`**, and **`pixels`**), to `Canvas` object with specified **`id`**.

`void` **`finalizeImageLoad`**`(semscripten_fetch_t *result)`
Receives the fetch **`result`**, processes errors, and displays the image in the `Canvas` object.

Submission

Your submission will be composed of the following elements:

- Screencast of your application running as an unlisted Internet resource (e.g., YouTube)
- Report (**WASMReport.pdf**) on Canvas, *including the link to your screencast*
- Source and build files as a gzipped tar file (**`cpp_loader.tgz`**) archive on Canvas

Screencast

Your video demonstration will be submitted on Canvas as a list in your report, must be no more than 2 minutes, and should include the following, at a minimum:

- Display the file directory/directories for the project, including the Makefile(s)
- Show and explain the contents of all Makefiles in your build
- Build the module to completion to show the build works
- Run a simple HTTP server and demonstrate the functionality of the module

Report

Your report will explain the process you used to develop the application, including what tools you used. It will describe how testing was performed and any known bugs. Please be sure to include all template elements. The report should be 500 words or fewer, cover all relevant aspects of the project, and be organized and formatted professionally – ***this is not a memo!***

Compressed Archive (cpp_loader.tar.gz)

The archive should have this structure:

```
cpp_loader.tgz
 └→ cpp_loader.tar
     └→ cpp_loader (directory)
         ├→ Makefile
         ├→ cpp_loader.html
         ├→ gator.rgba
         └→ src (directory)
             ├→         gator.h
             └→ (other files)
```

These commands are used to build and test:

```
$ tar zxvf cpp_loader.tgz
$ cd cpp_loader
$ make
$ python3 -m http.server
```

Additional Resources

In addition to the documents provided, students may also find these resources helpful.

Useful DOM References
https://developer.mozilla.org/en-US/docs/Web/API/HTMLCanvasElement
https://developer.mozilla.org/en-US/docs/Web/API/CanvasRenderingContext2D
https://developer.mozilla.org/en-US/docs/Web/API/ImageData

C/C++ APIs
https://emscripten.org/docs/api_reference/index.html
https://emscripten.org/docs/api_reference/fetch.html
https://emscripten.org/docs/api_reference/val.h.html
https://emscripten.org/docs/api_reference/html5.h
https://emscripten.org/docs/porting/connecting_cpp_and_javascript/embind.html
https://github.com/emscripten-core/emscripten/blob/main/system/include/emscripten/fetch.h

General Resources
https://webassembly.org/
https://doc.rust-lang.org/book/
https://emscripten.org/
https://rustwasm.github.io/docs/book/introduction.html
https://developer.mozilla.org/en-US/ (for HTML and JavaScript items)