Graduate Computer Graphics, Fall 2019 Onboarding

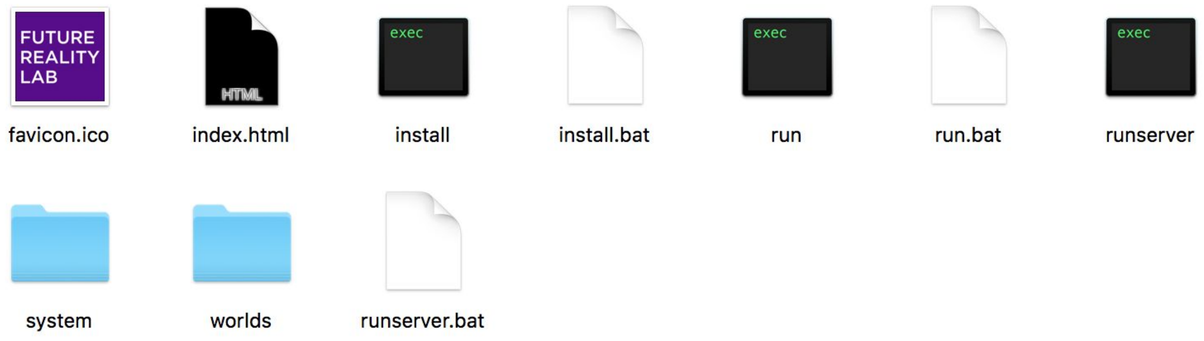
We are providing a library as well as a custom/simple live editor (based on open-source libraries and our own code) for your graphics programs so you can quickly see the results of your creations. We're calling this framework the Metaroom, which will contain your explorations in computer graphics. We'll update the system as necessary. This is version 0.999999.
Here are the instructions for using the system for local development.

Getting Started (First Time):

First: Install the required software (You only need to do this once on your machine):

1. Latest LTS release for NodeJS here: https://nodejs.org/en/
2. Google Chrome/Chromium (for the best experience, and for consistency)

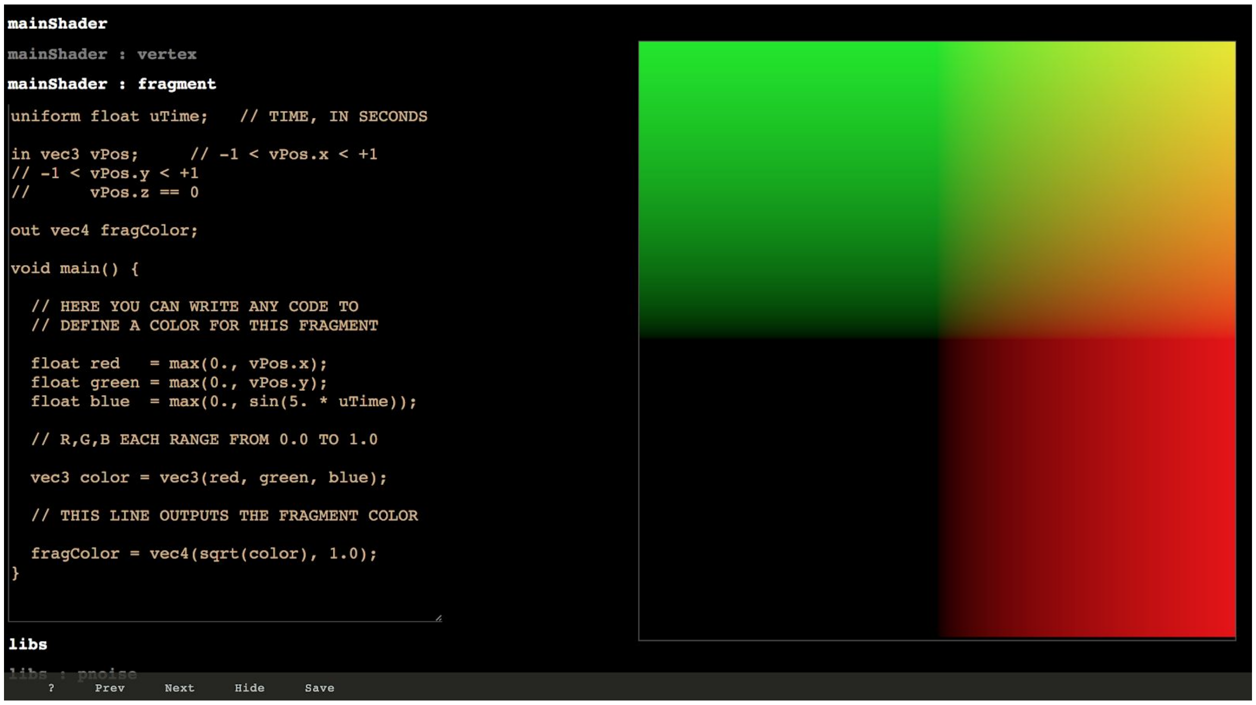
Second: In your preferred terminal, go to the project directory, which should contain a few run scripts and two folders: "worlds" and "system."



Windows users: execute *install.bat*, followed by *runserver.bat*

Mac users: execute *install*, followed by *run* (Linux should be the same since these are bash scripts)
(Sidenote: You can open these scripts in a text editor if you're curious. They just do a few installation commands for node and start the local server in the correct directory.)
If you have an installation error after running the install script, try running with *sudo ./install* on Mac/Linux

If successful, the terminal should read "MetaRoom server listening on port 3000". (This server is necessary to support some of the editor features, as well as using textures, which you'll learn about later.)



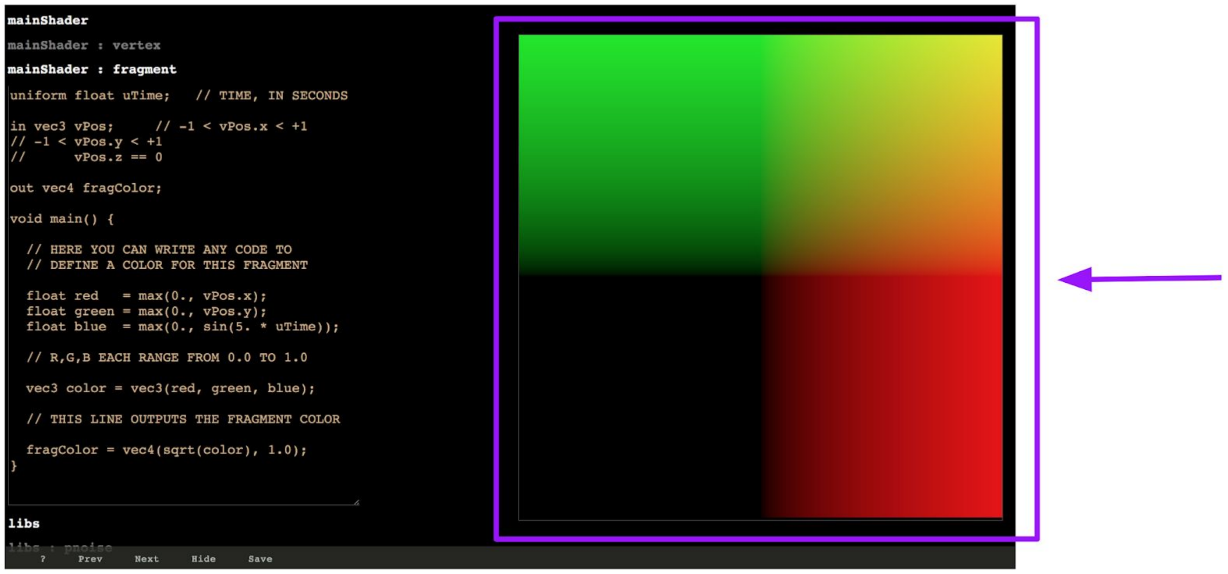
```
uniform float uTime;   // TIME, IN SECONDS

in vec3 vPos;     // -1 < vPos.x < +1
// -1 < vPos.y < +1
//      vPos.z == 0

out vec4 fragColor;

void main() {

  // HERE YOU CAN WRITE ANY CODE TO
  // DEFINE A COLOR FOR THIS FRAGMENT

  float red   = max(0., vPos.x);
  float green = max(0., vPos.y);
  float blue  = max(0., sin(5. * uTime));

  // R,G,B EACH RANGE FROM 0.0 TO 1.0

  vec3 color = vec3(red, green, blue);

  // THIS LINE OUTPUTS THE FRAGMENT COLOR

  fragColor = vec4(sqrt(color), 1.0);
}
```

Last: Open Google Chrome/Chromium to http://localhost:3000
You should see something like this:

You're done! Keep in mind, we plan to give you a new version of the library for you to download most weeks so we can continue to improve it and make sure no one is left behind due to any one assignment. **To run the system** after this first-time installation, just start the run script again and go to the browser page. **Instructions for posting your assignments will come in a separate document**.

Using the System:

Canvas:



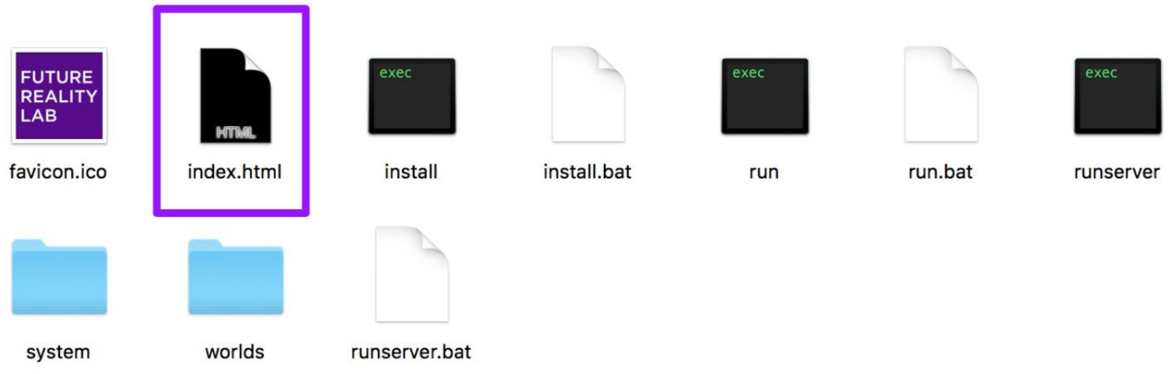
```
mainShader
mainShader : vertex
mainShader : fragment
uniform float uTime;   // TIME, IN SECONDS

in vec3 vPos;     // -1 < vPos.x < +1
// -1 < vPos.y < +1
//      vPos.z == 0

out vec4 fragColor;

void main() {

  // HERE YOU CAN WRITE ANY CODE TO
  // DEFINE A COLOR FOR THIS FRAGMENT

  float red   = max(0., vPos.x);
  float green = max(0., vPos.y);
  float blue  = max(0., sin(5. * uTime));

  // R,G,B EACH RANGE FROM 0.0 TO 1.0

  vec3 color = vec3(red, green, blue);

  // THIS LINE OUTPUTS THE FRAGMENT COLOR

  fragColor = vec4(sqrt(color), 1.0);
}

libs
libs : pnoise
```

The canvas is where your graphical output is displayed. It follows your window scroll so you can see it at all times, even if you have a lot of code.

Press and hold the ` key to snap the canvas to your mouse cursor position. Now you can offset the canvas somewhere else out of the way.

Tap the Alt key to toggle the canvas size between full and half-size.

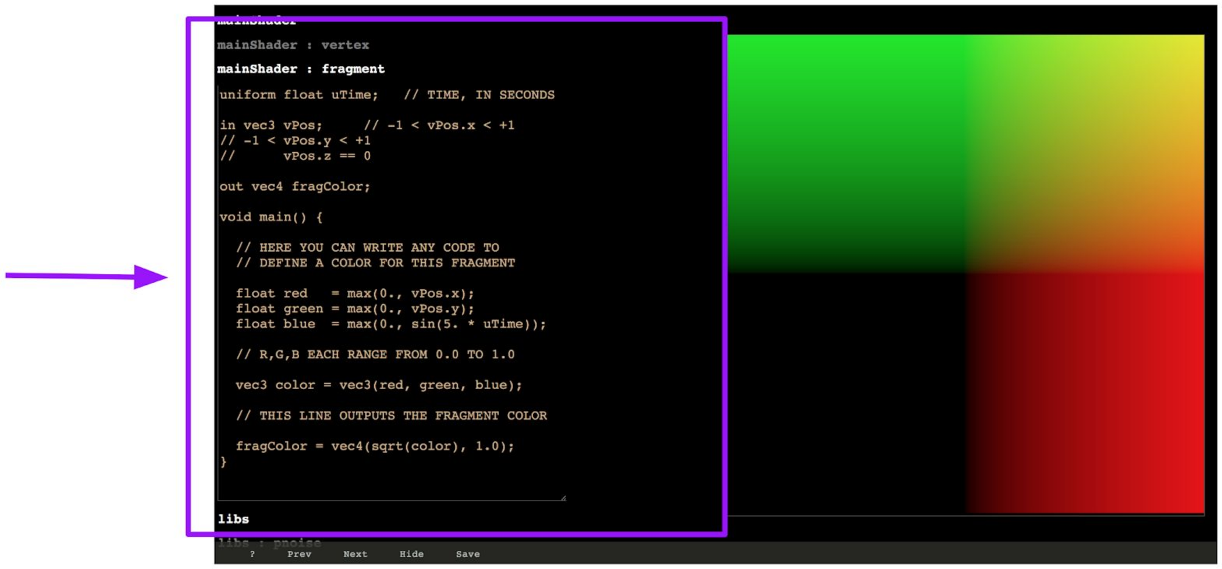
By default it's a 1:1 square. If you want to change the resolution of the canvas, open index.html in a text editor:



and change:
  <div id='resolution' value=**'720,720'**></div>
to a desired resolution, for example 1280 by 720 would be:

<div id='resolution' value=**'1280,720'**></div>
then reload the page

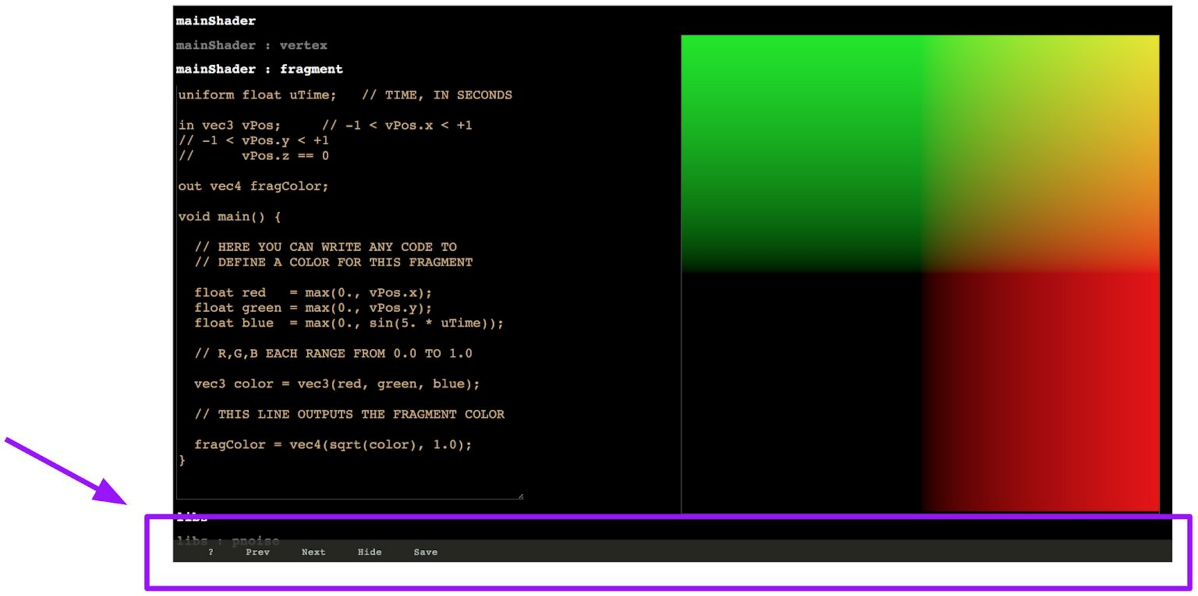
Shader Text Area:



```
mainShader
mainShader : vertex
mainShader : fragment
uniform float uTime;   // TIME, IN SECONDS

in vec3 vPos;      // -1 < vPos.x < +1
// -1 < vPos.y < +1
//       vPos.z == 0

out vec4 fragColor;

void main() {

  // HERE YOU CAN WRITE ANY CODE TO
  // DEFINE A COLOR FOR THIS FRAGMENT

  float red   = max(0., vPos.x);
  float green = max(0., vPos.y);
  float blue  = max(0., sin(5. * uTime));

  // R,G,B EACH RANGE FROM 0.0 TO 1.0

  vec3 color = vec3(red, green, blue);

  // THIS LINE OUTPUTS THE FRAGMENT COLOR

  fragColor = vec4(sqrt(color), 1.0);
}
libs
libs : pnoise
```

At the beginning of the semester we will focus purely on shaders. To support fast iteration, we've built a simple in-browser editor for live coding of the shaders.
As you will learn, shader programs are composed of multiple stages. WebGL supports vertex and fragment shader stages, which are shown in the editor.
For now we have one default shader for you to edit ("mainShader"). Click on the headers to show/hide the shader sections. **You probably will only want to edit the fragment section** at the moment, but feel free to play with the vertex shader as well to see what happens!
The programs recompile as you type. If there are errors, you will know.

Menu Bar:



```
uniform float uTime;   // TIME, IN SECONDS

in vec3 vPos;     // -1 < vPos.x < +1
// -1 < vPos.y < +1
//      vPos.z == 0

out vec4 fragColor;

void main() {

  // HERE YOU CAN WRITE ANY CODE TO
  // DEFINE A COLOR FOR THIS FRAGMENT

  float red   = max(0., vPos.x);
  float green = max(0., vPos.y);
  float blue  = max(0., sin(5. * uTime));

  // R,G,B EACH RANGE FROM 0.0 TO 1.0

  vec3 color = vec3(red, green, blue);

  // THIS LINE OUTPUTS THE FRAGMENT COLOR

  fragColor = vec4(sqrt(color), 1.0);
}
```

Save:
Save all of your changes locally to disk. This is why you need the server running. Saving is for local development only and won't work when you host your assignments. **Note:** The system will prevent you from saving if there are errors in your program, so don't worry about accidentally overwriting your work with buggy code.
Note: Command+S / Control+S saves as well.

External Editing:
You may choose to use an external editor and hide the in-browser code with the hide button. Just save your shader file externally and the web editor will update the shaders automatically. (The server also handles this.) **Note: in version 0.999999** we don't protect you from saving buggy code within an external editor.

Completely new to shaders? Want to learn more?

https://webgl2fundamentals.org/

https://www.khronos.org/opengles/sdk/docs/reference_cards/OpenGL-ES-2_0-Reference-card.pdf

https://thebookofshaders.com/

Sincerely,

the Future Reality Lab Team