### *Human GPU #0013 – Fragment shader*

hello! let's talk about the elephant in the room....Introducing you the *Fragment Shader*!

Until now, you only did half of the job, completing just a part of the real WebGL pipeline.

Drawing the triangles, in the way we did, is generally called Shape or Primitive Assembly. We worked in a theorethical vectorial space, without taking into account the actual screen (or better saying, the frame buffer) resolution.

After drawing our triangles, we need to "project" our shapes to a pixel grid. This phase is called rasterization.

The frame buffer where we are drawing has in fact a well defined resolution.

We are going to work with a whopping 60x60px resolution! For a total of 3600 pixels. (I'm trying to help you out, human...).

You'll see the frame buffer has now a new grid. Each little square in this new grid will rapresent a pixel.

Similarly to our vertex shader, the fragment shader is written in `GLSL` and it has only a job: define and set the global variable `gl_FragColor`.

The `gl_FragColor` will set the color of the pixel in RGBA.

Let's give it a try: draw and *color* 2 triangles!

### *Buffers*

```
{
  "data": [ -0.5, 0.5, 0.0, 0.5, 0.5, 0.0,
            0.5, -0.5, 0.0, -0.5, -0.5, 0.0 ],
  "_index": [ 3, 0, 1, 1, 2, 3 ]
}
```

### *Attributes*

```
{ "aPosition": { "buffer": "data", "size": 3 } }
```

### *Vertex shader*

```
attribute vec3 aPosition;

void main() {
  gl_Position = vec4(aPosition, 1.0);
}
```

### *Fragment shader*

```
precision highp float;

void main() {
  gl_FragColor = vec4(0.0, 0.0, 0.0, 1.0);
}
```