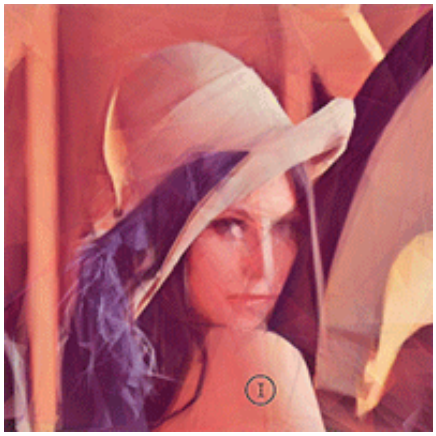
# GLSL-Projectron

This is a WebGL/GPGPU demo I made to teach myself shader programming. It generates random 3D polygons which resemble a given target image when projected. Basically it's the concept from this blog post, but done in 3D and on the GPU.

After many generations, you get a chaotic bunch of polygons that align into an image, but only from just the right angle:



## Live demos:

- Create a projection (uncheck "Paused" to begin)
- View one I made earlier, or this other one
- Or the obligatory Mona Lisa

I also put up a blog post here explaining the algorithm, and how I made it run fast on the GPU.

# Installation & Usage

```
git clone [this repo]
cd glsl-projectron
npm install
npm start
```

That launches a local copy of the "Create" demo linked above. Alternately, doing `npm run viewer`launches a local copy of the viewer demo. Use `npm run bundle` to browserify the examples.

To use the core module directly:

```
var proj = require('path/to/glsl-projectron/')
proj.init( glReference, imageReference )
//..
proj.runGeneration()        // many times..
proj.paint()                // once per frame..
```

The sample "create" and "view" clients in the `examples` folder implement this.

# API

- **init( glRef, imageRef, size )**

  Takes arguments:

  - gl object of the canvas you'll use for output
  - [optional] source image (an HTML Image object)
  - [optional] resolution of the framebuffer used internally for comparison (default 256)

  If no image is supplied, the library will initialize but assume you're going to use it to display data that was already created. (So calls to e.g. `runGeneration()` will fail.)

- **runGeneration()**

  Runs one generation of the genetic(?*) algorithm - mutating the internal data and keeping the result if it scores higher than the previous version.

- **paint( xRot, yRot )**

  Paints the current best data to the gl context. Optionally takes x- and y-rotation [Euler] angles.

- **paintReference()**

  Paints the reference framebuffer to the gl context (this may differ from the reference image in resolution)

- **paintScratchBuffer()**

  Paints the internal comparison framebuffer

- **exportData()**

Exports the current polygon data, in an extremely dumb ad-hoc string format.

- **importData( str )**

  Imports data in the same ad-hoc format as above.

- **score**

  (read only) Returns the score (fitness) of the current data. Ranges from minus a few hundred up to 100 for a perfect match.

- **numPolys**

  (read only) Number of polygons (triangles) currently used

- **minAlpha** / **maxAlpha**

  (defaults 0.05, 0.5) Min/max values the algorithm will use when randomizing any vertex's alpha value. I found that setting these moderately low (e.g. .1 to .5) made images get more interesting more quickly than just using the range 0..1.

- **compareonGPU**

  (default true) Flag for whether to use the GPU when comparing rendered images to the target image. Turning this off will slow things down a lot.

- **fewerPolysTolerance**

  (default 0) A tolerance within which the algorithm will accept generations with a lower score as long as they have fewer polygons than the previous.

- **useFlatPolys**

  (default false) When set, all new polygons will share a common z value (i.e. they will be parallel to the camera plane). I thought this might make for interesting visuals, but didn't get anywhere with it.

# Known issues:

- Displaying data fails spectacularly in Firefox Windows/Android. (Mac is fine.) No idea why - either alpha in WebGL is broken or my code is. Might be due to this ownerless FF bug.
- Doesn't detect most error cases (just whether WebGL is supported)

- Library treats input images as if they were square. To use for other aspects, just run it normally and change the aspect of the canvas you use to display the results. Example files could be updated to do this..

**Note:**

It's not at all clear to me that the algorithm here (and in the blog I got the idea from) is truly a "genetic" algorithm, but I'm deferring to the source material on terminology. There's no formal genome, I just have a data structure that I perturb once per generation and compare to its parent.