gl-react

fitting OpenGL shaders into React paradigm

Gaëtan Renaudeau – **@greweb**

Gaëtan Renaudeau

@greweb

Frontend Lead Engineer at Ledger

github.com / gre / **gl-react**

Gardener during summer 🌱 *father of 2* 👨‍👩‍👦‍👦

- The « Functional rendering » paradigm of shaders
- gl-transitions OSS initiative
- How & why gl-react embraces React
- Ray-marching distance functions

	« Functional rendering »	« Procedural rendering »
2D		
3D		

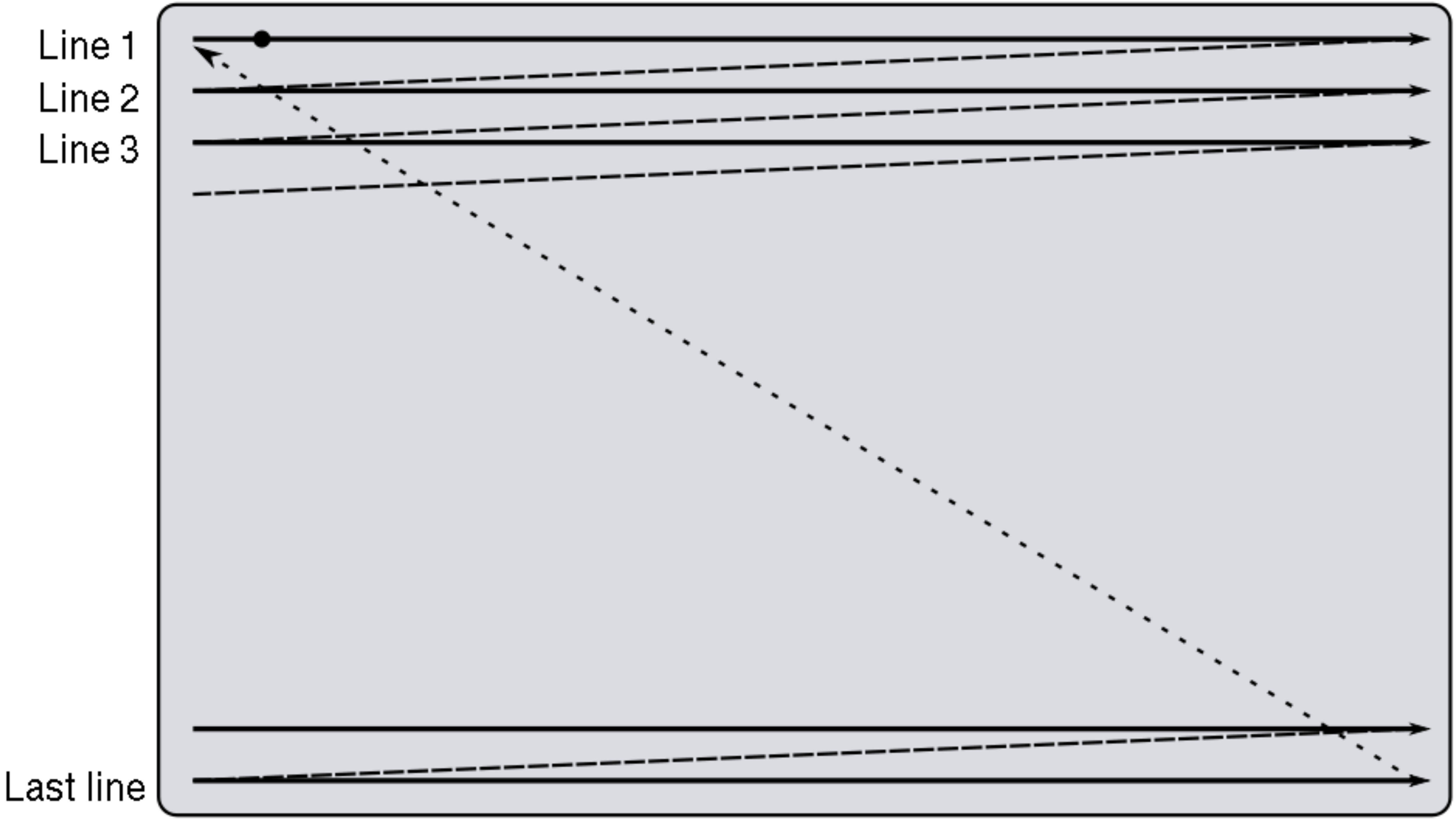
« Functional Rendering »

As opposed to « Procedural Rendering »

ATARI 2600

How was rendering during that time?

Analog TV: scan lines



Rendering was about

- Figuring out color to output for current beam position
- sleep(time) to jump to a specific beam position!
- Repeat each frame rate

What rendering has become

imperative

descriptive

```
ctx = canvas.getContext("2d")
ctx.fillStyle = "blue"
ctx.fillRect(0, 0, 200, 200)
ctx.fillStyle = "yellow"
ctx.beginPath()
ctx.arc(99, 99, 99, 0, 2*Math.PI)
ctx.fill()
ctx.fillStyle = "blue"
ctx.font = "24px sans-serif"
ctx.textAlign = "center"
ctx.fillText("Hello World", 99, 99)
```

```
<div style="background: blue; width: 200px; h
  <div style="background: yellow; color: blue
  100px; text-align: center; font: 24px sans-se
    Hello World
  </div>
</div>
```

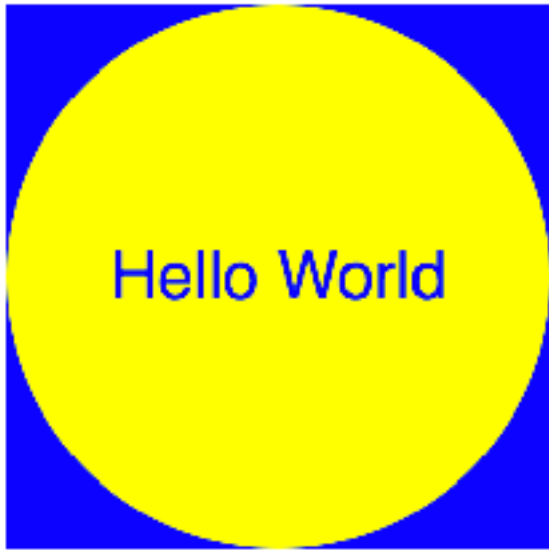
```
<svg width=200 height=200>
  <rect x=0 y=0 width=200 height=200 fill=blue />
  <circle cx=100 cy=100 r=100 fill=yellow />
  <text x=100 y=100 fill=blue text-anchor=middle
    Hello World
  </text>
</svg>
```

React is here

HTML+CSS

SVG

Canvas 2D



https://jsfiddle.net/d0crLpLv

- CPU🚀 allowed us better rendering abstractions
- Graphics framework developed to cover 99% use cases
- high-level API, simpler to use and more productive
- Finite set of drawing primitives / shapes (rect, circle, text)

« Procedural rendering »

« Functional rendering »

`(x,y) ⇒ (r,g,b,a)`

`Position ⇒ Color`

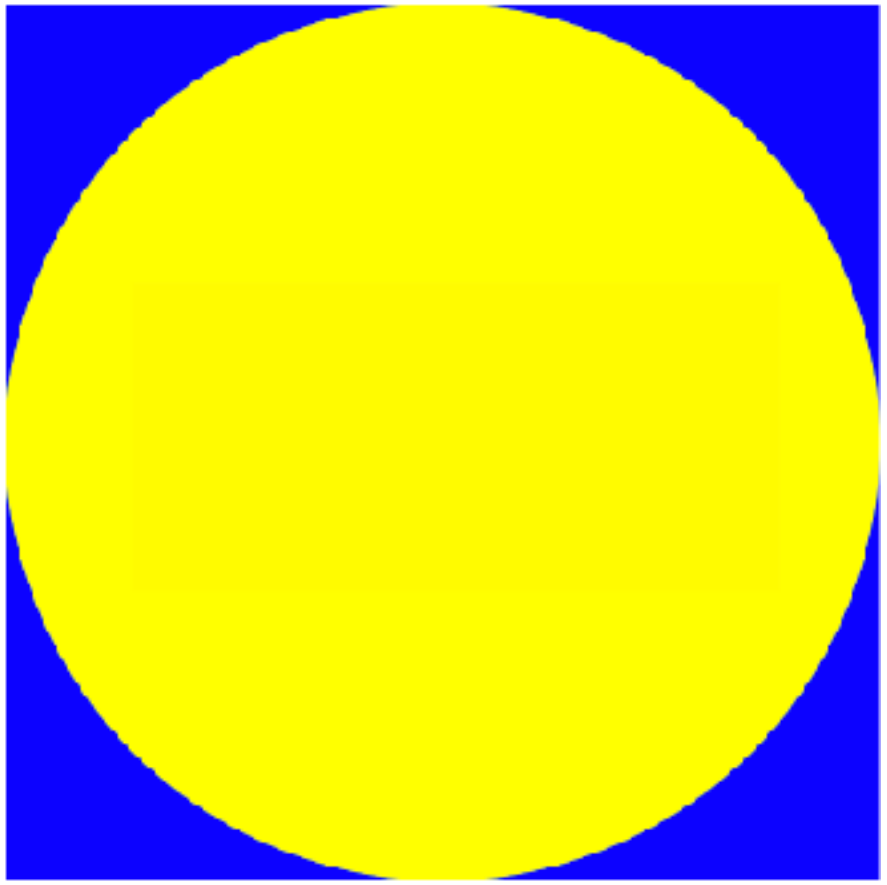
« Functional rendering »

- Just a pure function (Position => Color)
- Powerful & not restricted to primitives
- math & functional
- low level (at pixel level)

Circle

Functional

```
(x,y) ⇒
 (x-0.5)*(x-0.5)
+(y-0.5)*(y-0.5) < 0.25
? yellow
: blue
```



Procedural

```
ctx.beginPath();
ctx.arc(99, 99, 99, 0, 2*Math.PI);
ctx.fill();
```

```
<circle fill=yellow
  cx=100 cy=100 r=100 />
```

Image

Functional

```
p ⇒ texture(img, p)
```

(image stretched to full viewport)

Procedural

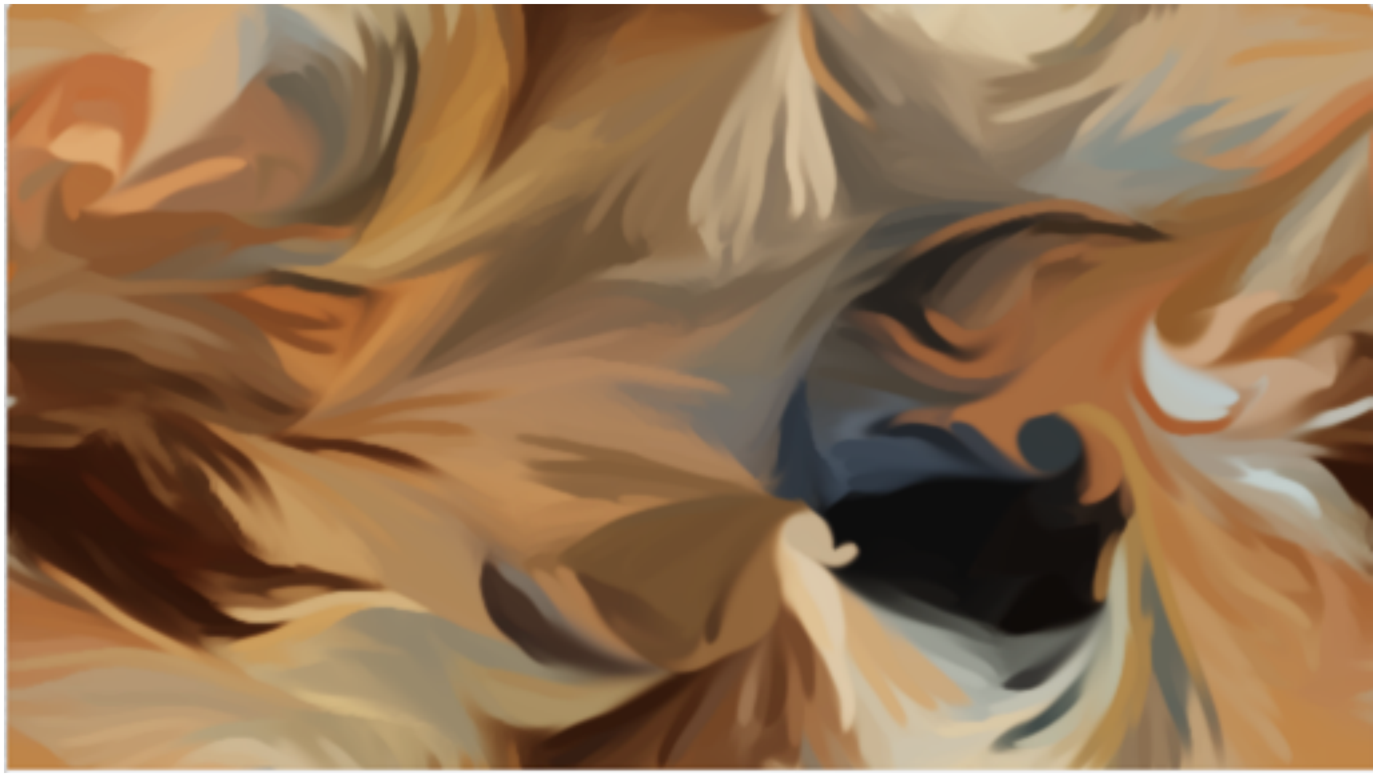
```
ctx.drawImage(img, 200, 200);
```

```
<img src=..
  width=..
  height=.. />
```

more advanced shapes

Functional

```
p ⇒ <insert math here!>
```



https://www.shadertoy.com/view/ldcSDB

Procedural

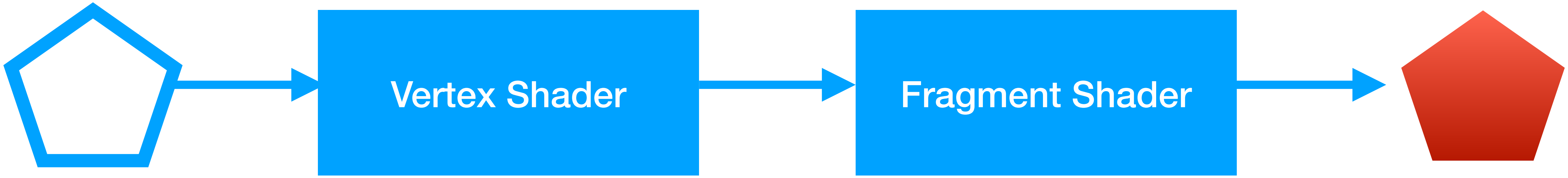
- bezier curves
- gradients
- filters, masks,…

In OpenGL…

GLSL fragment shaders

are in this **Point->Color** paradigm

OpenGL pipeline



Vertex Shader

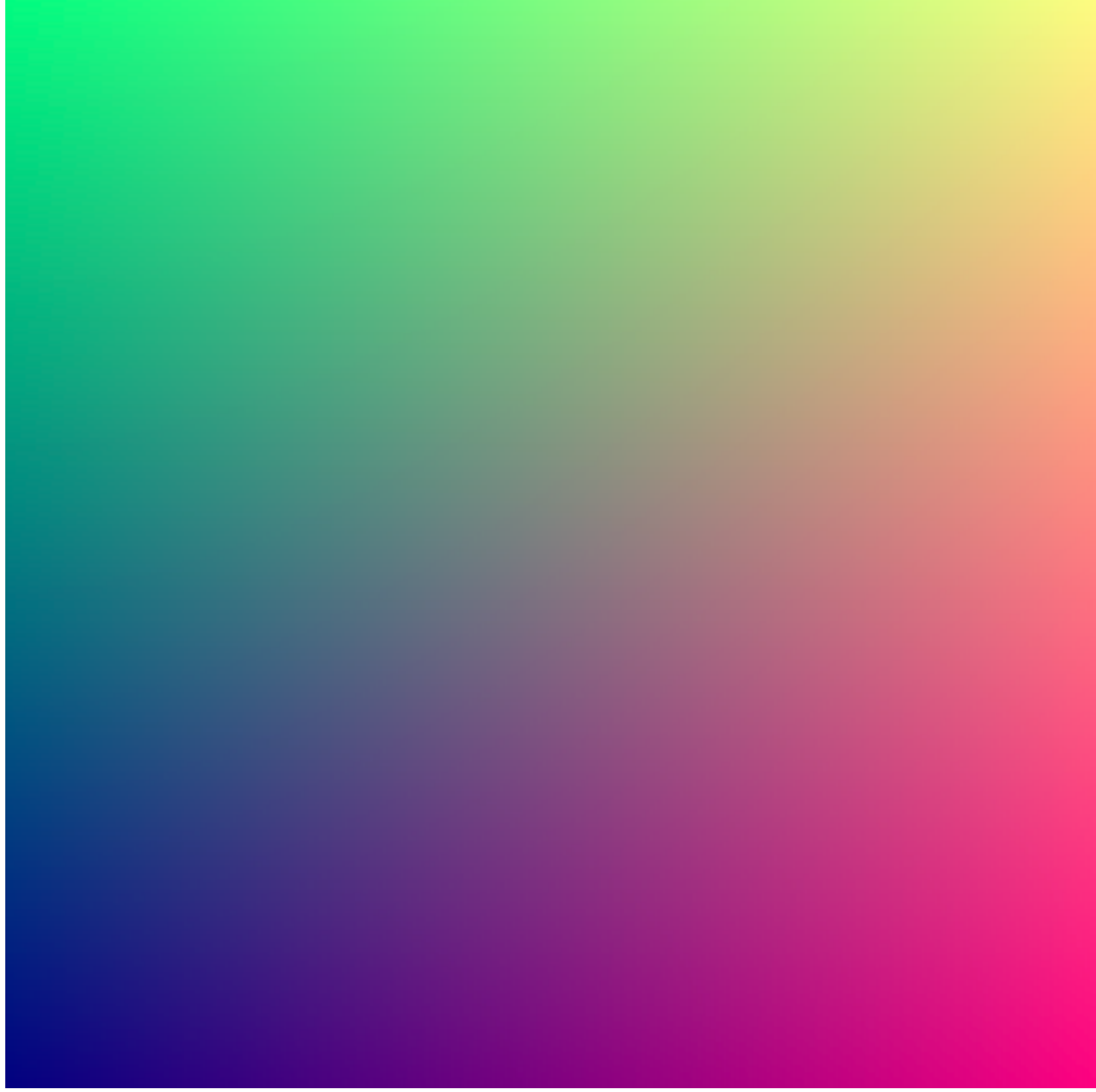
Fragment Shader

```
Source code…
Source code…
Source code…
Source code…
```

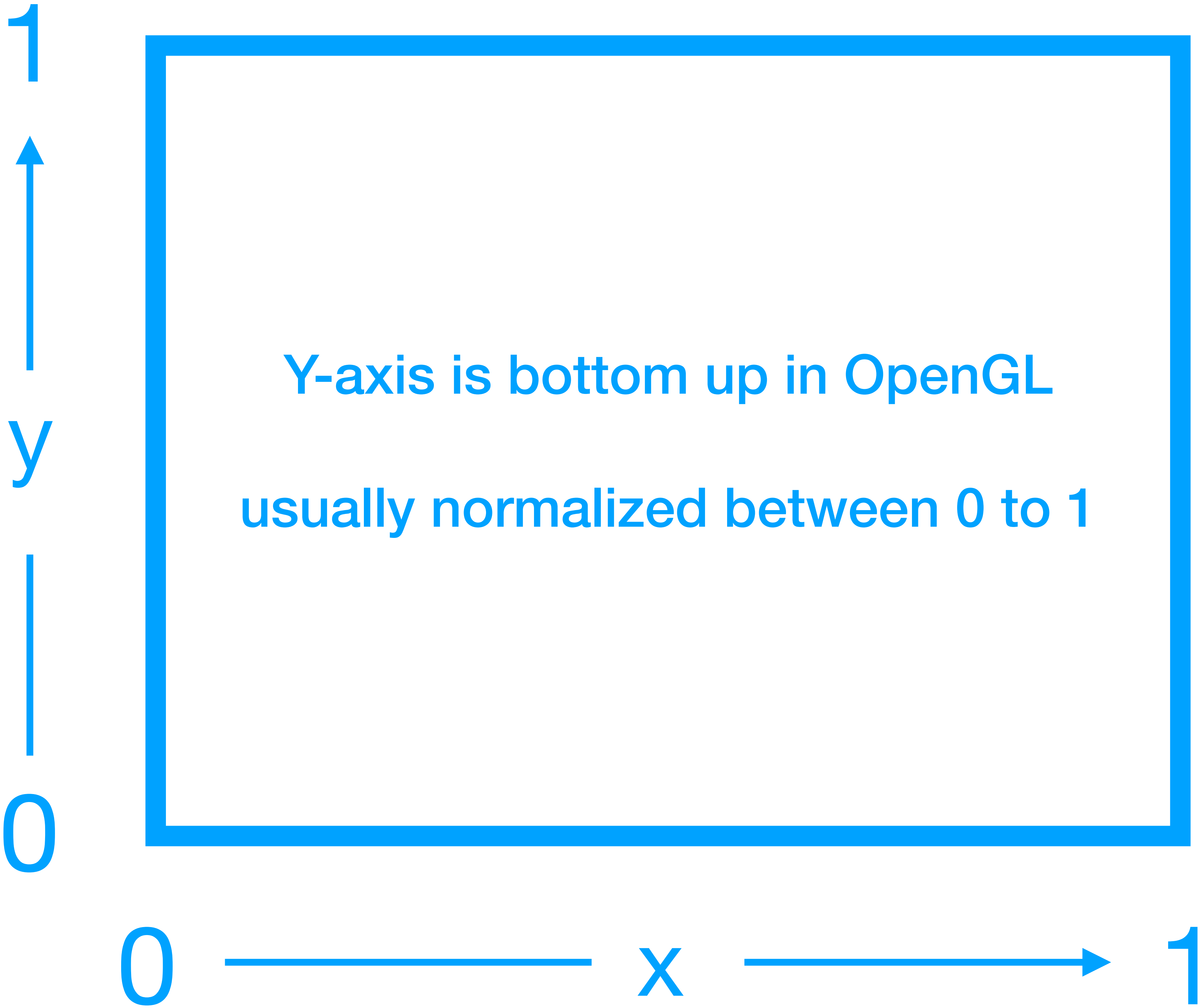
```
Source code…
Source code…
Source code…
Source code…
```

(simplification of the actual pipeline)

Hello GL



```
varying vec2 uv;
void main() {
  gl_FragColor = vec4(uv.x, uv.y, 0.5, 1.0);
}
```



Y-axis is bottom up in OpenGL

usually normalized between 0 to 1

Resources on shaders

- thebookofshaders.com (*Patricio Gonzalez Vivo*)
- shadertoy.com

	« Functional rendering »	« Procedural rendering »
2D	**GLSL**, Atari! *point => color*	Graphic libs / Canvas / SVG / …
3D		

gl-transitions.com

- Open Source initiative to establish an **universal collection of transitions** that various softwares can use (including Movie Editors).
- Uses GLSL, most appropriate language to implement transitions at pixel level.
- GLSL is highly performant & universal (OpenGL / WebGL)

Libraries

- **gl-transition**: render a frame with a WebGL Context
- **regl-transition**: render a GL Transition with a regl context
- **react-gl-transition**: gl-react React component
- **gl-transitions**: all the transitions created

More on **https://github.com/gre/gl-transition-libs**

react-gl-transition

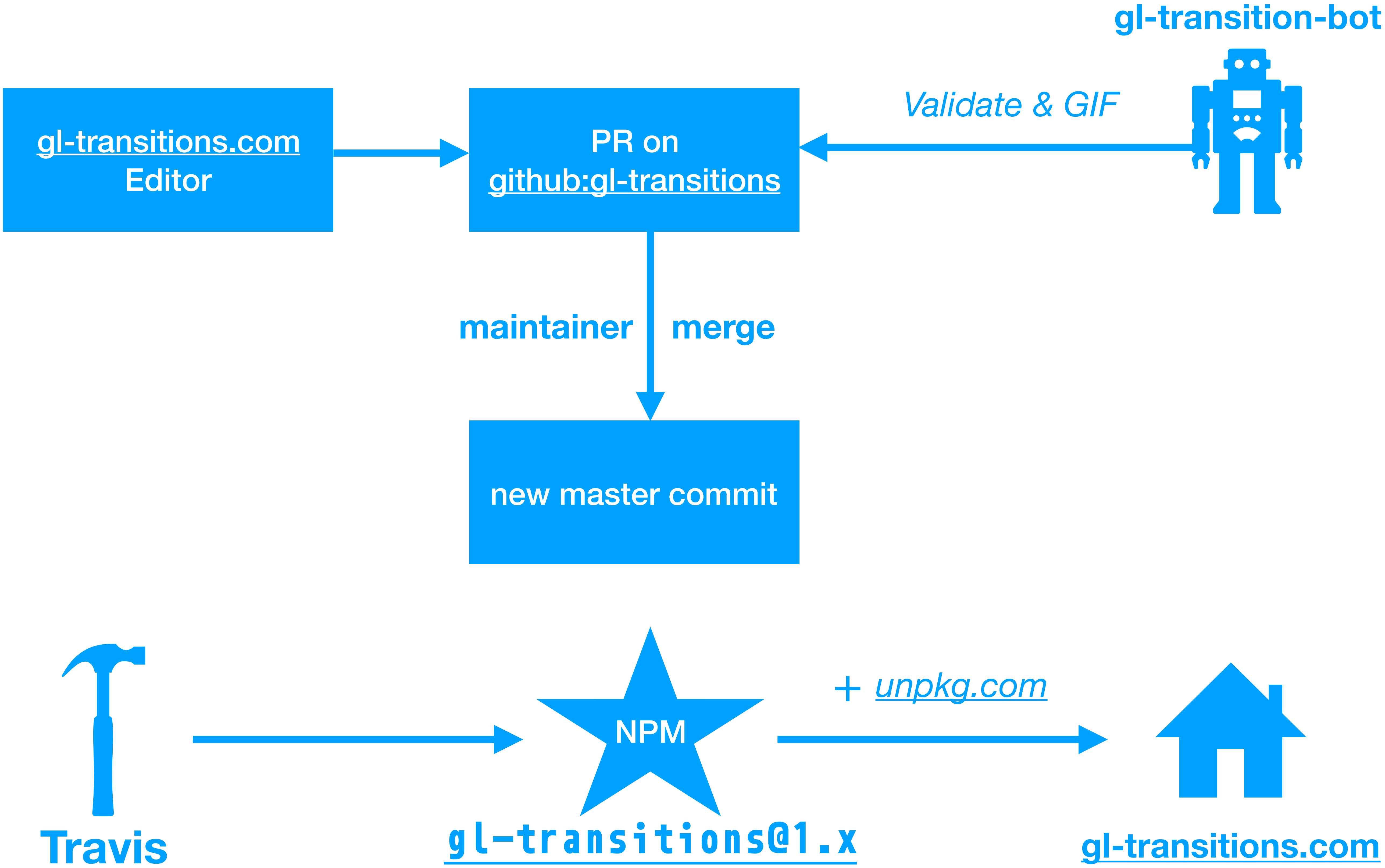
```
<GLTransition
  progress={progress}
  from={from}
  to={to}
  transition={transition}
/>
```

Example **https://gl-react-cookbook.surge.sh/transitions**

Live Coding

a new transition

```
uniform float count; // = 10
uniform float smoothness; // = 1.6
vec4 transition (vec2 p) {
  float pr = smoothstep(-smoothness, 0.0, p.x - progress * (1.0 + smoothness));
  float s = step(pr, fract(count * p.x));
  return mix(getFromColor(p), getToColor(p), s);
}
```



gl-transition-bot

gl-transitions.com
Editor

PR on
github:gl-transitions

Validate & GIF

maintainer merge

new master commit

NPM

+ *unpkg.com*

Travis

gl-transitions@1.x

gl-transitions.com

gl-react

Libraries

- **gl-react**, the core & universal library
- **gl-react-dom**, WebGL implementation
- **gl-react-native** backed by **react-native-webgl**
- **gl-react-expo** backed by **Expo**’s GLView
- **gl-react-headless** backed by **gl** (for node)

gl-react cookbook

gl-react-cookbook.surge.sh

gl-react v3 embraces React

- Using the context to make the Node knows its parent and rebuild the scene **graph** structure.

- Using React update lifecycle to re-draw the graph with a dirty flag system to only render part that changes.
  a Node holds a framebuffer. (see GameOfLife rot)

- `<Bus>` solution as a way to share computation and express a graph in React component tree. (blurmapdyn)

React suspense paradigm

```
const movieDetailsFetcher = createFetcher(
  fetchMovieDetails
);

function MovieDetails(props) {
  const movie = movieDetailsFetcher.read(props.id);
  return (
    <div className="MovieDetails">
      <h1>{movie.title}</h1>
    </div>
  );
}
```

Talk: Dan Abramov – Suspense!

https://www.youtube.com/watch?v=6g3g0Q_XVb4

React suspense paradigm

- gl-react manage loading WebGL textures
- currently uses https://github.com/gre/webgltexture-loader
- plan to migrate to suspense! Remove boilerplate from user land if React can provide a uniform way to handle asynchronous things to load.

Ray-marching Distance Functions

classical 3D graph scene code

```
function init() {
    camera = new THREE.PerspectiveCamera(70, w/h, 0.01, 10 );
    camera.position.z = 1;
    scene = new THREE.Scene();
    geometry = new THREE.BoxGeometry( 0.2, 0.2, 0.2 );
    material = new THREE.MeshNormalMaterial();
    mesh = new THREE.Mesh( geometry, material );
    scene.add( mesh );
    renderer = new THREE.WebGLRenderer( { antialias: true } );
    renderer.setSize( window.innerWidth, window.innerHeight );
    document.body.appendChild( renderer.domElement );
}

function animate() {
    requestAnimationFrame( animate );
    mesh.rotation.x += 0.01;
    mesh.rotation.y += 0.02;
    renderer.render( scene, camera );
}
```

(x,y,z) ⇒ dist

Point3D ⇒ float

dist: estimated distance of the closest 3D object

Live Coding

https://codesandbox.io/s/j4n8zv8r55

https://github.com/gre/memocart **lowrezjam 2017**

MEMO CART
stuck in a runaway mine cart. all that can save you is your memory

- https://youtu.be/WkRFp5gRfSo 5 minutes play
- https://youtu.be/Xg9nnW-NiFk graphics overview
- https://youtu.be/hqjSnd16uyU development timelapse (97 hours!)
- https://youtu.be/w3c41RaggFc technical code overview

Ray-marching DF resources

- http://jamie-wong.com/2016/07/15/ray-marching-signed-distance-functions/
- http://mercury.sexy/hg_sdf/
- iquilezles.org/www/articles/raymarchingdf/raymarchingdf.htm
- iquilezles.org/www/articles/distfunctions/distfunctions.htm
- https://www.shadertoy.com/view/Xds3zN

} by **IQ**

Íñigo Quílez (IQ), god of ray-marching DF

	« Functional rendering »	**« Procedural rendering »**
2D	**GLSL** / Atari ***gl-react*** `point ⇒ color`	Graphic libs / Canvas / SVG / *Pixi.js* / …
3D	**Ray-marching Distance Estimation functions**	Scene graph / *Three.js* / …

Questions?

github.com/**gre**

@greweb