Cross-Browser-Tracking-Summary-Part-3

一叶飘零 / 2018-12-30 09:29:00 /

前言

本篇文章主要讲部分技术的代码实现与测试。

渲染部分实现

多数代码均利用webgl进行渲染，更重要的是模型和参数的选取，已经在前一篇文章简单介绍。对渲染代码感兴趣的可以自行学习图形学。这里我们主要介绍数据的收集和利

数据收集

这里对图像的渲染结果基本利用下述方法进行结果获取

```
this.getData = function(gl, id) {
   if (!this.finalized) {
     throw "Still generating ID's";
     return -1;
   }
   var WebGL = true;
   var pixels = new Uint8Array(256 * 256 * 4);
   gl.readPixels(0, 0, 256, 256, gl.RGBA, gl.UNSIGNED_BYTE, pixels);
   var ven, ren;
   var debugInfo = gl.getExtension('WEBGL_debug_renderer_info');
   if (debugInfo) {
     ven = gl.getParameter(debugInfo.UNMASKED_VENDOR_WEBGL);
     ren = gl.getParameter(debugInfo.UNMASKED_RENDERER_WEBGL);
   } else {
     console.log("debugInfo is not accessable");
     ven = 'No debug Info';
     ren = 'No debug Info';
   }
   var hash = pixels.hashCode();
   this.toServer(WebGL, ven, ren, hash, id, pixels);
   if (sumRGB(pixels) > 1.0) {
     return hashRGB(pixels);
   } else {
     return 0;
   }
 };
```

关键点

```
gl.readPixels(0, 0, 256, 256, gl.RGBA, gl.UNSIGNED_BYTE, pixels);
```

这里7个参数分别为

```
void gl.readPixels(x, y, width, height, format, type, pixels);
```

x

A `GLint` specifying the first horizontal pixel that is read from the lower left corner of a rectangular block of pixels.

y

A `GLint` specifying the first vertical pixel that is read from the lower left corner of a rectangular block of pixels.

width

A `GLsizei` specifying the width of the rectangle.

height

A `GLsizei` specifying the height of the rectangle.

由于图片均为256*256的，所以前4个参数为0,0,256,256
第5个参数

format

A `GLenum` specifying the format of the pixel data. Possible values:

- `gl.ALPHA`: Discards the red, green and blue components and reads the alpha component.
- `gl.RGB`: Discards the alpha components and reads the red, green and blue components.
- `gl.RGBA`: Red, green, blue and alpha components are read from the color buffer.

第6，7个参数

type

A `GLenum` specifying the data type of the pixel data. Possible values:

- `gl.UNSIGNED_BYTE`
- `gl.UNSIGNED_SHORT_5_6_5`
- `gl.UNSIGNED_SHORT_4_4_4_4`
- `gl.UNSIGNED_SHORT_5_5_5_1`
- `gl.FLOAT`

pixels

An `ArrayBufferView` object to read data into. The array type must match the type of the `type` parameter.

- `Uint8Array` for `gl.UNSIGNED_BYTE`.
- `Uint16Array` for `gl.UNSIGNED_SHORT_5_6_5`, `gl.UNSIGNED_SHORT_4_4_4_4`, or `gl.UNSIGNED_SHORT_5_5_5_1`.
- `Float32Array` for `gl.FLOAT`.

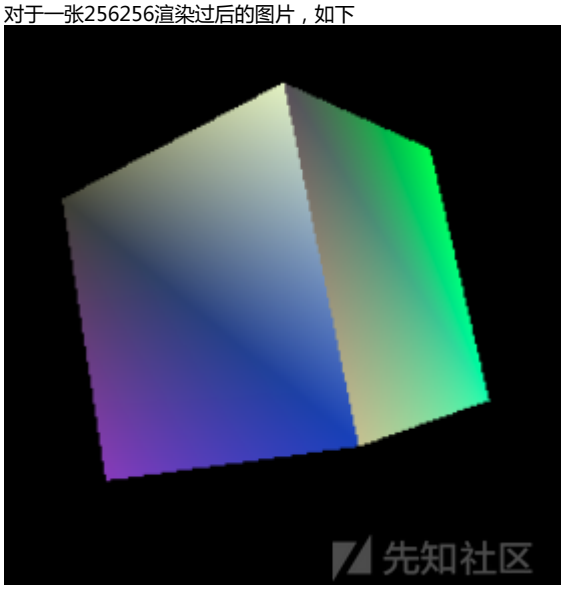
那么我们代码中的使用

```
var pixels = new Uint8Array(1 * 1 * 4);
gl.readPixels(0, 0, 1, 1, gl.RGBA, gl.UNSIGNED_BYTE, pixels);
```

对于一个像素点返回为

```
[0, 0, 0, 255]
```

对应的分别为Red, green, blue and alpha
对于一张256256渲染过后的图片，如下



可以得到如下数组

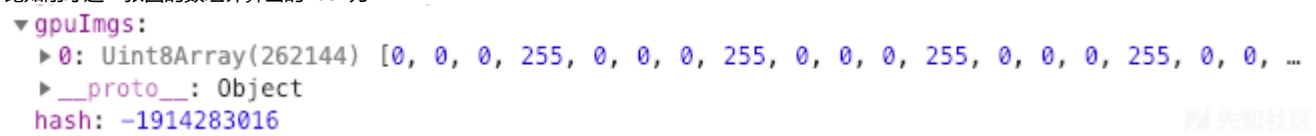
toServer.js:194

```
Uint8Array(262144) [0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255,
 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0
▼, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0,
255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255,
0, 0, 0, 255, …]
  ▶ [0 … 9999]
  ▶ [10000 … 19999]
  ▶ [20000 … 29999]
  ▶ [30000 … 39999]
  ▶ [40000 … 49999]
  ▶ [50000 … 59999]
  ▶ [60000 … 69999]
  ▶ [70000 … 79999]
  ▶ [80000 … 89999]
  ▶ [90000 … 99999]
  ▶ [100000 … 109999]
  ▶ [110000 … 119999]
  ▶ [120000 … 129999]
  ▶ [130000 … 139999]
  ▶ [140000 … 149999]
  ▶ [150000 … 159999]
  ▶ [160000 … 169999]
  ▶ [170000 … 179999]
  ▶ [180000 … 189999]
  ▶ [190000 … 199999]
  ▶ [200000 … 209999]
  ▶ [210000 … 219999]
  ▶ [220000 … 229999]
  ▶ [230000 … 239999]
  ▶ [240000 … 249999]
  ▶ [250000 … 259999]
  ▶ [260000 … 262143]
```


作者生成了一个256256*4的数组存放Red, green, blue and alpha
然后经过hash后返回给Server

```
var hash = pixels.hashCode();
```

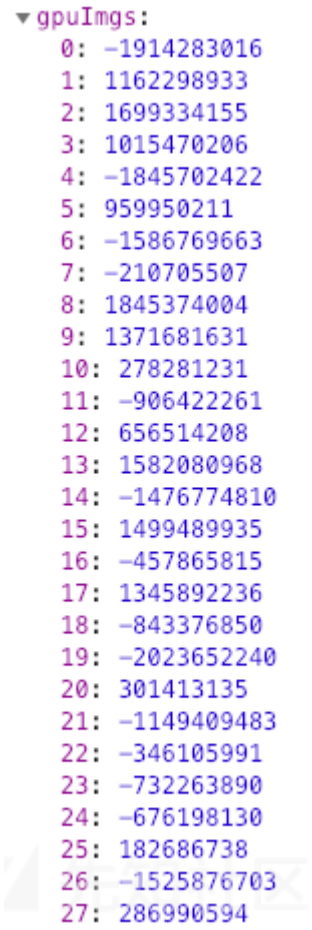
比如刚才这一张图的数组计算出的hash为



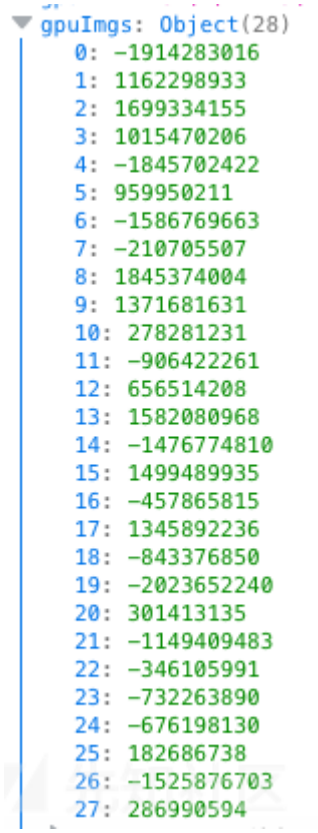
功能测试

我打开了所有的渲染任务，进行测试:

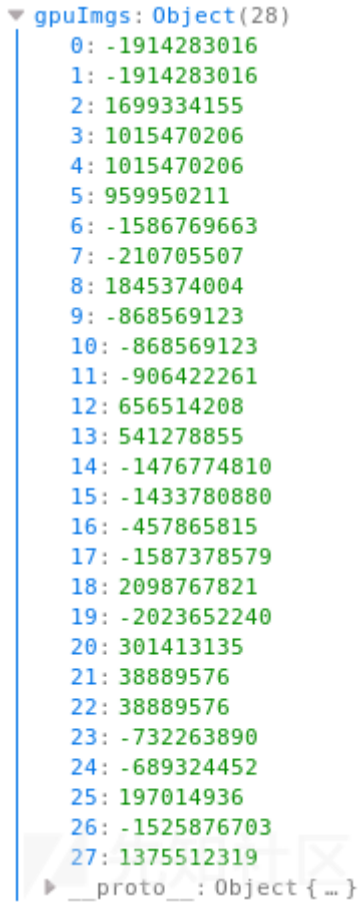
Chrome浏览器



Firefox浏览器



然后我打开了虚拟机，用Firefox浏览器访问



换了一台windows电脑，用Chrome浏览器访问

```
gpu: "ANGLE (Intel(R) HD Graphics F
▼gpuImgs:
   0: -1914283016
   1: 858292596
   2: 1699334155
   3: 1015470206
   4: -1957635466
   5: 959950211
   6: -1586769663
   7: -210705507
   8: 1845374004
   9: 1464747134
   10: 548961603
   11: 1102448745
   12: -1732715138
   13: 1660192166
   14: -1476774810
   15: 1499489935
   16: -1050801720
   17: 2016939339
   18: 1800772270
   19: -1193671538
   20: -1487175827
   21: 118102169
   22: -1830159834
   23: -732263890
   24: -2010764159
   25: 1293501200
   26: -1525876703
   27: 479678180
```

用Firefox浏览器访问

```
gpu: "ANGLE (Intel(R) HD Graphics Family Direct3D1
▼gpuImgs: Object(28)
   0: -1914283016
   1: 1162298933
   2: 1699334155
   3: 1015470206
   4: -1892436752
   5: 959950211
   6: -1586769663
   7: -210705507
   8: 1845374004
   9: 1464747134
   10: 371346734
   11: 1102448745
   12: -1732715138
   13: 1660192166
   14: -1476774810
   15: 1499489935
   16: -1050801720
   17: 2016939339
   18: 1800772270
   19: -1193671538
   20: -1487175827
   21: -118653215
   22: -1830159834
   23: -732263890
   24: -2010764159
   25: 1293501200
   26: -1525876703
   27: 479678180
▶ <prototype>: Object { … }
```

不难发现，不同设备，相同浏览器渲染任务返回的hash值出现不同，但相同设备，不同浏览器的渲染任务返回hash大部分一致。这一点也充分说明了GPU之间存在差异性，

LanguageDetector实现

对于Writing
Script的检测实现，作者使用了CoffeeScript编写，然后编译成JavaScript引入，之所以使用CoffeeScript编写再编译，因为可以减少很多代码工作量，并且语言更加简洁易
整体代码工作量不大，进行了如下几步：

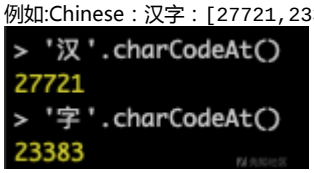
1.基础定义
2.长宽识别
3.长宽校验
4.统计结果

基础定义

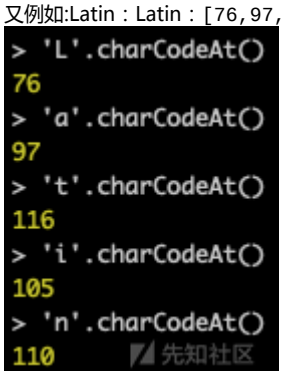
```
safeParseJSON = (s) ->
 try
   JSON.parse s
 catch
   false

class LanguageDetector
 constructor: ->
   @names = safeParseJSON '[
   "Latin",
   "Chinese",
   "Arabic",
   "Devanagari",
   "Cyrillic",
   "Bengali/Assamese",
   "Kana",
   "Gurmukhi",
   "Javanese",
   "Hangul",
   "Telugu",
   "Tamil",
   "Malayalam",
   "Burmese",
   "Thai",
   "Sundanese",
   "Kannada",
   "Gujarati",
   "Lao",
   "Odia",
   "Ge-ez",
   "Sinhala",
   "Armenian",
   "Khmer",
   "Greek",
   "Lontara",
   "Hebrew",
   "Tibetan",
   "Georgian",
   "Modern Yi",
   "Mongolian",
   "Tifinagh",
   "Syriac",
   "Thaana",
   "Inuktitut",
   "Cherokee"
   ]'

   @codes = safeParseJSON "[[76,97,116,105,110],
   [27721,23383],
   [1575,1604,1593,1585,1576,1610,1577],
   [2342,2375,2357,2344,2366,2327,2352,2368],
   [1050,1080,1088,1080,1083,1080,1094,1072],
   [2476,2494,2434,2482,2494,32,47,32,2437,2488,2478,2496,2479,2492,2494],
   [20206,21517],
   [2583,2625,2608,2606,2625,2582,2624],
   [43415,43438],
   [54620,44544],
   [3108,3142,3122,3137,3095,3137],
   [2980,2990,3007,2996,3021],
   [3374,3378,3375,3390,3379,3330],
   [4121,4156,4116,4154,4121,4140],
   [3652,3607,3618],
```

```
   [7070,7077,7060,7082,7059],
   [3221,3240,3277,3240,3233],
   [2711,2753,2716,2736,2750,2724,2752],
   [3749,3762,3751],
   [2825,2852,2893,2837,2867],
   [4877,4821,4829],
   [3523,3538,3458,3524,3517],
   [1344,1377,1397,1400,1409],
   [6017,6098,6040,6082,6042],
   [917,955,955,951,957,953,954,972],
   [6674,6682,6664,6673],
   [1488,1500,1508,1489,1497,1514],
   [3926,3964,3921,3851],
   [4325,4304,4320,4311,4323,4314,4312],
   [41352,41760],
   [6190,6179,6185,6189,6179,6191],
   [11612,11593,11580,11593,11599,11568,11606],
   [1808,1834,1825,1821,1808],
   [1931,1960,1928,1964,1920,1960],
   [5123,5316,5251,5198,5200,5222],
   [5091,5043,5033],
   [55295, 7077]]" #may need a new code for 7077

   @fontSize = 9
   @fontFace = "Verdana"
   @extraHeigth = 15
   @results = []
```

作者选择了36种语言，然后选择了相应的语言输出相应的语言名，
例如:Chinese：汉字：[27721,23383]



又例如:Latin：Latin：[76,97,116,105,110]



长宽识别

如何检测字体的长宽？这里作者没有直接去对字体长宽进行测量，还是选择了测量div的长宽

```
@test_div = document.createElement "div"
document.body.appendChild @test_div
@test_div.id = "WritingTest"
for code in @codes
     @height = []
     @width = []
     #generate div
     @div = document.createElement "div"
     @test_div.appendChild @div
     round += 1
     @div.id = round
     @div.style.display = "inline-block"
```

这样一来对div的长宽测量就变得容易了许多

```
for c in code
       @div.innerHTML = "<font face = '#{@fontFace}' size = " + @fontSize + ">&#" + c + "</font>"
       @height.push document.getElementById(round).clientHeight
```

```
      @width.push document.getElementById(round).clientWidth
```

然后测量每个字体div的长度和宽度，放入数组height[]和width[]，例如

然后对应合并

```
for c in code
      @div.innerHTML += "<font face = '#{@fontFace}' size = " + @fontSize + ">&#" + c + "</font>"
 @test_div.innerHTML += @height + ";" + @width + "<br>"
 @heights.push @height
 @widths.push @width
```

Latin58,58,58,58,58;27,29,19,13,30

汉字67,67;48,48

العربية68,68,68,68,68,68,68;14,23,28,20,31,31,17

देवनागरी72,72,72,72,72,72,72;25,32,26,25,45,27,19,45

Кирилица58,58,58,58,58,58,58;33,31,30,31,30,31,29

বাংলা / অসমীয়া72,72,72,72,72,0,58,0,72,72,72,72,72,72,72;26,45,53,35,45,0,22,0,43,32,28,45,27,32,45

仮名67,67;48,48

ਗੁਰਮੁਖੀ72,72,72,72,72,72,72;35,28,28,29,28,29,38

□□60,60;35,35

长宽校验

```
@tw = @widths.pop()
@sw1 = @tw[0]
@sw2 = @tw[1]
@sh = @heights.pop()[0]

for height in @heights
 @passed = 0
 for h in height
   if h != @sh
     @support.push true
     @passed = 1
     break
 if @passed == 0
   @support.push false

@writing_scripts_index = 0
for width in @widths
 for w in width
   if @support[@writing_scripts_index] == false
     if w != @sw1 && w != @sw2
       @support[@writing_scripts_index] = true
 @writing_scripts_index += 1
```

这里我们发现所有的校验都是和@sh = @heights.pop()[0]进行比较
那么我们需要知道这个数组的最后一个值是什么，我们看到最开始@codes的定义
不难发现@codes的数组长度是37，而@names的数组长度是36，这样做的原因就是作者故意在最后一组放置了无法被任何浏览器渲染的字体。这样即可让所有字体的长宽和
注：这里不直接使用方块的原因是有的浏览器可能渲染失败了未必出现方块，可能是其他形状，这样就增大了准确性和稳定性

统计结果

```
@res = []
@writing_scripts_index = 0
for s in @support
 @test_div.innerHTML += "#{@names[@writing_scripts_index]}: #{s} <br>"
 if s == true
   @res.push @names[@writing_scripts_index]
 @writing_scripts_index += 1
@test_div.remove()
```

最后将可渲染字符打印出来

功能测试

将代码理解完成后，自己实践了一下，不难发现不同浏览器之间的差异：

使用chrome浏览器，得到如下结果


```
Latin,Chinese,Arabic,Devanagari,Cyrillic,Bengali/Assamese,Kana,Gurmukhi,Hangul,Telugu,Tamil,Malayalam,Burmese,Thai,Kannada,Guj
```

使用safari浏览器，得到如下结果


```
Latin,Chinese,Arabic,Devanagari,Cyrillic,Bengali/Assamese,Kana,Gurmukhi,Javanese,Hangul,Telugu,Tamil,Malayalam,Burmese,Thai,Su
```

后记

对作者的demo进行分析十分有趣，不仅可以学到知识，还能引发一些自己的思考~

上一篇：带外通道（OOB）Bind XML... 下一篇：gogs/gitea CVE-20...

1. 0 条回复
   - 动动手指，沙发就是你的了！