WebGL纹理详解——压缩纹理的使用

为什么要用压缩纹理

下面这张图是一辆陆虎越野车模型所用的纹理，原始分辨率为1024 x 1024。

浏览器从服务端加载这样一张图片时，其格式通常为JPEG，文件尺寸只有166KB，但是当WebGL处理一张纹理时就需要按照位图处理（这里所说的位图是指没有使用任何压缩算法的原始图片数据），如果图像中每个像素需要RGB三个通道，每个通道需要8位空间，那么整张图片就需要使用1024 x 1024 x 8 x 3 位的信息，也就是3M，这3M的信息都需要加载到GPU缓存当中，这和图片文件采用什么样的压缩格式没有任何关系。当调用gl.texImage2D方法时，浏览器内部就会将图片文件进行解压，转换成位图格式。如果图片包含透明信息，那么RGBA格式那么还要额外增加内存使用。

在简单了解WebGL处理纹理的过程后，会知道使用JPEG或者PNG文件作为纹理时会有这样的问题：

1. 需要有图片解压过程，比较耗时。
2. 因为纹理数据较大，所以传输纹理数据耗时较多。
3. 纹理数据占用内存较多。通常是浏览器和GPU各自保存一份位图数据。

压缩纹理的出现就是来解决这些问题，经过某种算法压缩之后的纹理可直接被GPU使用，只有当shader进行纹理查询（texture lookup）才会进行解压操作，找到对应位置的像素颜色。GPU通常会对解压过程进行优化从而提升性能。

浏览器的图片缓存策略

为了更好的了解WebGL处理纹理时如何分配和使用内存，请看下面的实验。

首先，页面加载前文提到的陆虎车的纹理，打开Chrome的任务管理器（Task Manager），把 Image Cache 和 GPU Memory 这两项勾选，查看图片缓存和GPU内存用量。你可以打开这个demo自行观察。

这时 Image Cache 显示：4262K，GPU Memory 显示：17.7M。

现在，我们注释掉加载纹理的语句，这时候图片不会加载，纹理也不会被绘制出来。

此时 Image Cache 显示：0K，GPU Memory 显示：13.7M。

如果加载图片，但是不创建纹理会如何?

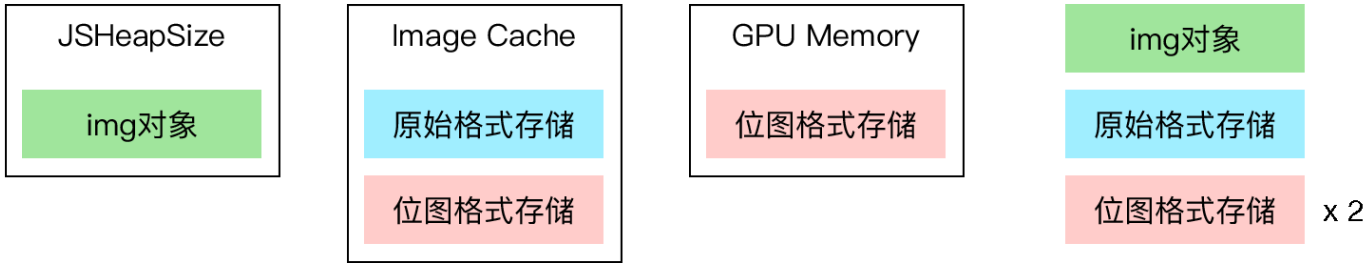
结果是 Image Cache显示：166K，GPU Memory显示：13.7M。

通过以上观察可以推出以下结论。

- 当仅下载图片时 Image Cache 以图片原始格式进行缓存，当创建纹理时，浏览器会解压成为位图格式，并将位图数据进行缓存。
- GPU Memory 以位图格式保存纹理数据。
- 通过Chrome的Profile工具可以看到在JavaScript层面，其内存消耗只包含HTMLImageElement对象所占用的内存大小。

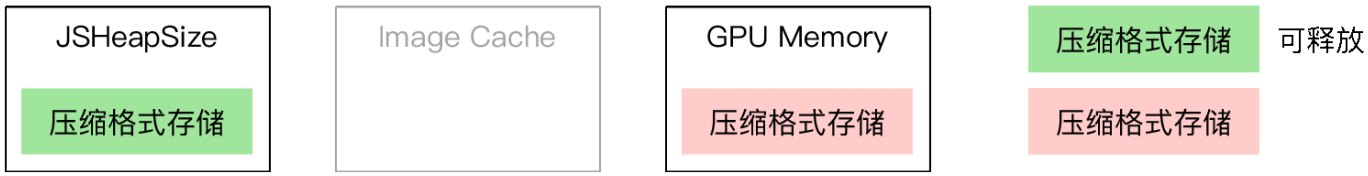
以上结论可以用下面这张图来概括：

通过img对象创建纹理的内存使用



当使用压缩纹理时，内存使用状况变为：

压缩纹理的内存使用



压缩纹理的种类

不同的GPU厂商会有不同的纹理压缩格式，具体如下：

- S3TC/DXTn/BCn：桌面计算机常见的压缩格式。名字虽然有不同叫法，但都是指同一种压缩方式。通常以DDS文件格式保存。

- PVRTC/PVRTC2：iOS设备上使用的压缩纹式。
- ETC/ETC2：随着OpenGL ES 2.0 出现。
- ASTC：2012年出现的一种新压缩格式。
- ATC：Adreno GPU支持的一种压缩格式，Android手机上常用。

压缩纹理支持情况

WebGL1.0里，压缩纹理是通过扩展支持的，因此要看当前浏览器支持哪些扩展。具体判断方法如下：

```
1
2
3
4
5
6
7
8
var availableExtensions = gl.getSupportedExtensions();
for (var i = 0; i < availableExtensions.length; i++) {
    if (availableExtensions[i].indexOf('texture') >= 0
        && availableExtensions[i].indexOf('compressed') >= 0) {
        // show in console
        console.log(availableExtensions[i]);
    }
}
```

在我的Macbook Pro Chrome上，上面代码片段会输出如下结果：

WEBGL_compressed_texture_s3tc

也就是说，Macbook Pro Chrome支持S3TC格式的压缩纹理。

下面表格列举了一些设备和浏览器对压缩纹理的支持情况：

压缩纹理工具

下面的工具可以将JPEG、PNG等常见的图片转换为压缩纹理。此外还有很多在线转换工具，这里就不一一列出了。

压缩纹理实战

下面我们来通过实例来看如何实现压缩纹理。由于兼容性，我们首先准备不同格式的压缩纹理（可以用上面提到的工具提前对纹理进行压缩）。

完整代码请见这里。

下面来分析一下核心代码，初始化时首先判断当前浏览器支持什么格式的压缩纹理，判断方法如下：

```
01
02
03
04
05
06
07
08
09
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
var compressedTextureType = ['s3tc', 'etc1', 'pvrtc'];
var currentCompressedTextureType = null;
function getSupportedCompressedTextureType() {
    var availableExtensions = gl.getSupportedExtensions();
    for (var i = 0; i < availableExtensions.length; i++) {
        for (var j = 0; j < compressedTextureType.length; j++) {
            if (availableExtensions[i].indexOf(compressedTextureType[j]) > 0) {
                var extension = gl.getExtension(availableExtensions[i]);
                // 下面这句话必须在getExtension之后调用
                var formats = gl.getParameter(gl.COMPRESSED_TEXTURE_FORMATS);
                console.log(formats);
                for (var key in extension) {
                    console.log(key, extension[key], '0x' + extension[key].toString(16));
                }
                return {
                    type: compressedTextureType[j],
                    extension: extension,
                    formats: formats
                };
            }
```

```
        }
    }
}
```

我的Chrome浏览器运行后会打印如下信息：

formats是一个数组，可以看到里面有四个数值：[33776, 33777, 33778, 33779]

遍历extension会打印如下信息：
COMPRESSED_RGB_S3TC_DXT1_EXT 33776 0x83f0
COMPRESSED_RGBA_S3TC_DXT1_EXT 33777 0x83f1
COMPRESSED_RGBA_S3TC_DXT3_EXT 33778 0x83f2
COMPRESSED_RGBA_S3TC_DXT5_EXT 33779 0x83f3

这些信息就是浏览器支持压缩纹理的具体格式，可以看到formats和extension中的信息是一样的。只不过一个是数组，一个是对象。可以根据这些信息提前准备好压缩格式。

在创建纹理的代码中，使用 gl.compressedTexImage2D 替换原有的 gl.texImage2D 方法：

1

```
gl.compressedTexImage2D(gl.TEXTURE_2D, 0, type, width, height, 0, source);
```

这里 type 是前面通过extension获取到的那些常量，需要根据当前纹理的具体格式选择。source就是压缩纹理的数据，这里需要注意的是不同压缩格式获取数据的方法都不一样，通常压缩纹理开头部分存放一些描述信息，之后才是真正的纹理数据。

在Chrome中使用压缩纹理后，打开Task Manager后可以看到 Image Cache 变为 0K，GPU Memory 也比之前有所下降。

压缩纹理的优势是节省内存开销，但是纹理文件的尺寸却比原来的JPG或者PNG要大，因此会影响加载的时间。那么项目当中是否要用压缩纹理就需要权衡了。如果你的3D场景同时使用的纹理较多或者其他因素导致内存消耗较大，同时需要长时间运行，那么你可以考虑使用压缩纹理，在场景初始化时需要预先加载必要的纹理，同时通过进度条指示状态，这种做法类似大型3D游戏的初始化。要注意控制纹理的分辨率，过大的纹理压缩之后仍然会上兆，这非常影响加载速度。