1.为什么需要压缩?

> 移动端的优化需要。
>
> 一张 512 x 512 的ARGB8888的未经过压缩的图片 的大小为 4 x 512 x 512 = 1048576B = 1024kb = 1M。这样的大小对于一些比较大型的游戏来说，是非常恐怖的，渲染3D场景往往需要大量的贴图，这对带宽和内存的消耗的都非常的巨大。

2.为什么能压缩?

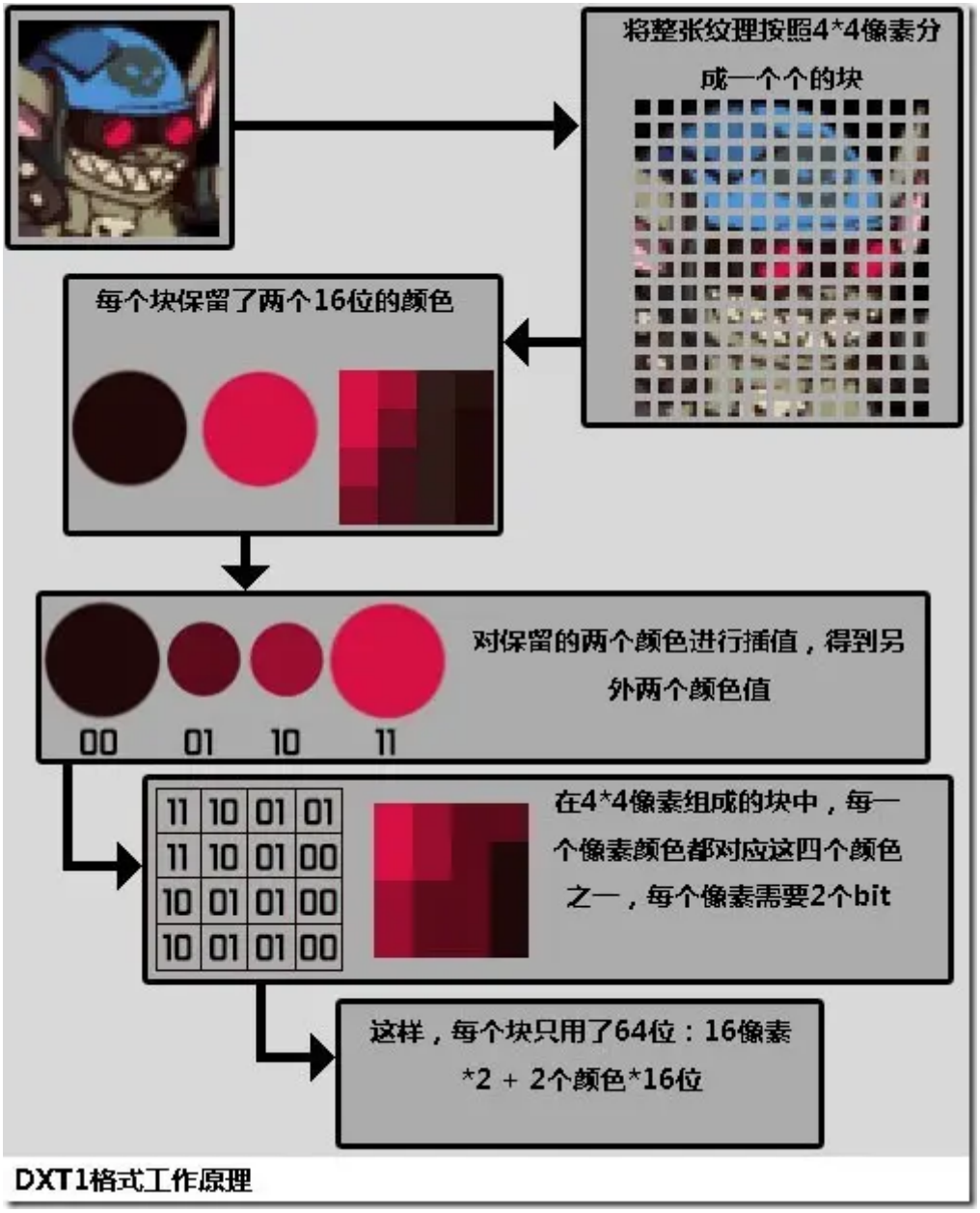
> 1.因为人眼对于颜色的感知并没有那么强。但是就看压缩到什么程度，这是一个经验主义的结果。
> 2.折中，类似也有很多所谓优化都是在几个选项中间取得一个平衡。 时间换空间，体验换性能。 （Mipmap, Lod, 单声道）

> 压缩又分为有损压缩和无损压缩。
> 无损压缩：JPEG，PNG, JPG, BMP , ...
> 有损压缩： ETC, DXT , PVRTC, ASTC （一般采用的都是向量量化的技术）
>
> > 💡向量量化(Vector quantization, VQ)**是一种量化技术，将一组大量的点(向量)分成具有近似相同数量的最接近它们的点的组。每个组用它的质心点表示，因此存在数据误差，适用于有损压缩。放到纹理压缩中来理解，就是例如将4x4块像素的颜色以2个基色来表示。

> 为什么要选择有损压缩?
> 因为GPU渲染需要高效的随机访问一个像素。所以在压缩的时候慢一点不要紧，但是解码必须要快。

3.常见几种纹理压缩格式的压缩原理（表层）

（1）DXT压缩原理


将整张纹理按照4*4像素分成一个个的块
每个块保留了两个16位的颜色
00
01
10
11
对保留的两个颜色进行插值，得到另外两个颜色值
11 10 01 01
11 10 01 00
10 01 01 00
10 01 01 00
在4*4像素组成的块中，每一个像素颜色都对应这四个颜色之一，每个像素需要2个bit
这样，每个块只用了64位：16像素*2 + 2个颜色*16位

DXT1格式工作原理

selector value	pixel color
0	color0
1	color1
2	(2 * color0 + color1) / 3
3	(color0 + 2 * color1) / 3

DXT encoding can therefore be visually represented in the following way:

color0 (RGB565) 16 bits/block | color1 (RGB565) 16 bits/block | selectors 2 bits/pixel | decoded DXT 4 bits/pixel

💡baseColor是怎么算的？为什么是4 x 4? 猜测4x4 可能是经验主义.

💡这里有两个问题

2.basecolor只有16位，表示颜色一般是R5G6B5，最多只能 RGB(31,63,31), 那么假如渲染一张白色的图该怎么办？假如每个像素都是（255,255,255）

（2）ETC2压缩原理

ETC1 采取4x2 block , 4位表索引

```
flip                                          1    bit
basecolor  RGB444*2 || RGB555+RGB333          24   bits
 diff (RGB444 OR  RGB555)                      1    bit
 modifier table index     3*2=                 6    bits
 selctors 16*2                                 32   bits
```

a	*e*	*i*	*m*
b	*f*	*j*	*n*
c	*g*	*k*	*o*
d	*h*	*l*	*p*

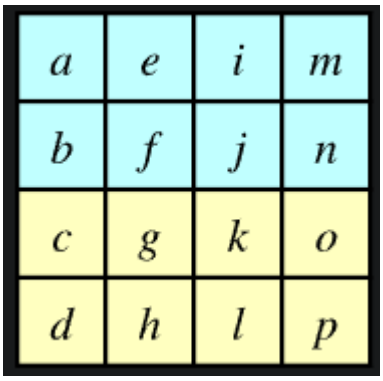


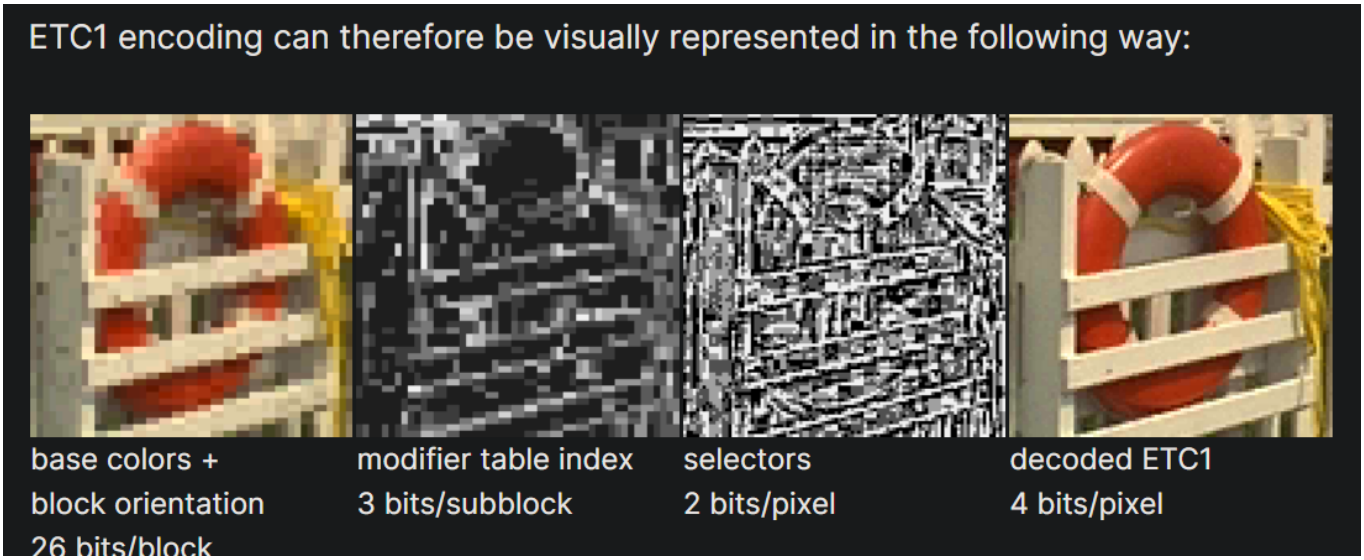
Table codeword	modifier0	modifier1	modifier2	modifier3
	Modifier table			
0	-8	-2	2	8
1	-17	-5	5	17
2	-29	-9	9	29
3	-42	-13	13	42
4	-60	-18	18	60
5	-80	-24	24	80
6	-106	-33	33	106
7	-183	-47	47	183

这样一个子块由1个基本颜色值和4个修饰值可以确定出4种新的颜色值：

```
color0 = base_color + RGB(modifier0, modifier0, modifier0)

color1 = base_color + RGB(modifier1, modifier1, modifier1)

color2 = base_color + RGB(modifier2, modifier2, modifier2)

color3 = base_color + RGB(modifier3, modifier3, modifier3)
```

最终的颜色就是从这4个颜色值中选出。其原理就是另外32位数据中包含16个2位选择器数据，每个像素的颜色都根据2位选择器的值从这4种中选出。

ETC1编码方式直观图如下：



（3）PVRTC压缩原理

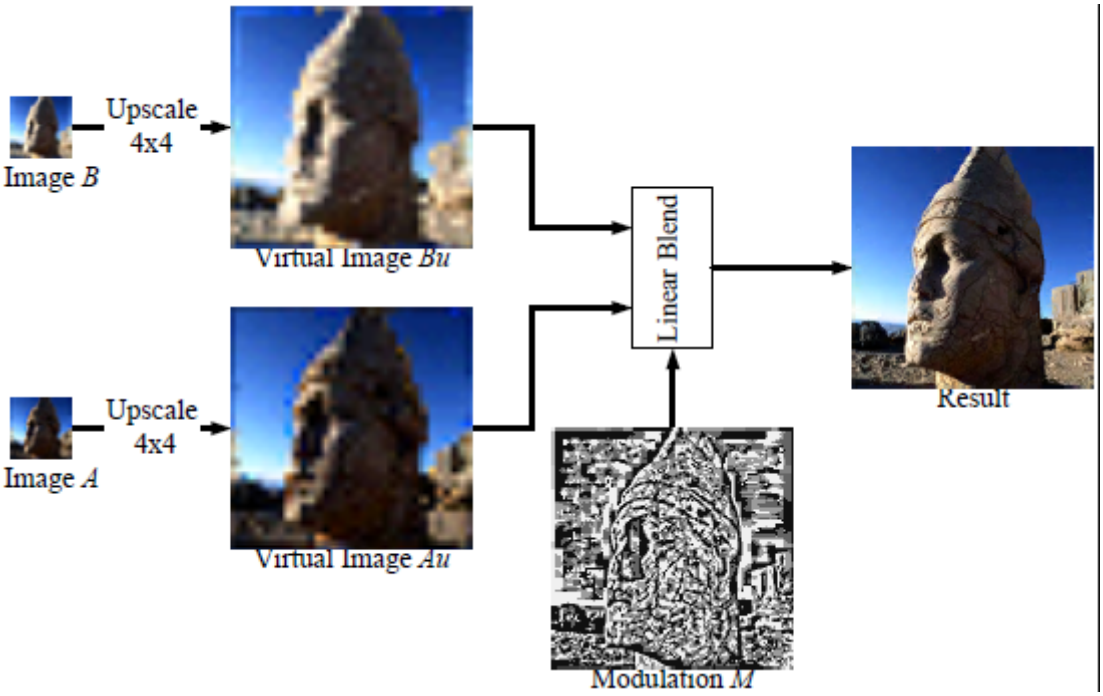
PVRTC (PowerVR Texture Compression)由Imagination公司专为PowerVR显卡核心设计，由于专利原因一般它只被用于苹果的设备，仅Iphone、Ipad和部分PowerVR的安卓机支持。这可能是这几种压缩格式中最不公开的技术。

PVRTC的不是基于block的方式生成的，但是却也可以理解为以block方式组织的。

PVRTC不同于DXT和ETC这类基于块的算法，而将整张纹理分为了高频信号和低频信号，低频信号由两张低分辨率的图像A和B表示，这两张图在两个维度上都缩小了4倍（有点类似于第二层的Mipmap），高频信号则是全分辨率但低精度的调制图像M，M记录了每个像素混合的权重。要解码时，A和B图像经过双线性插值（bilinearly）宽高放大4倍，然后与M图上的权重进行混合。

💡对图像而言： 低频分量(低频信号)：代表着图像中亮度或者灰度值变化缓慢的区域，也就是图像中大片平坦的区域，描述了图像的主要部分，是对整幅图像强度的综合度量。 高频分量(高频信号)：对应着图像变化剧烈的部分，也就是图像的边缘（轮廓）或者噪声以及细节部分。

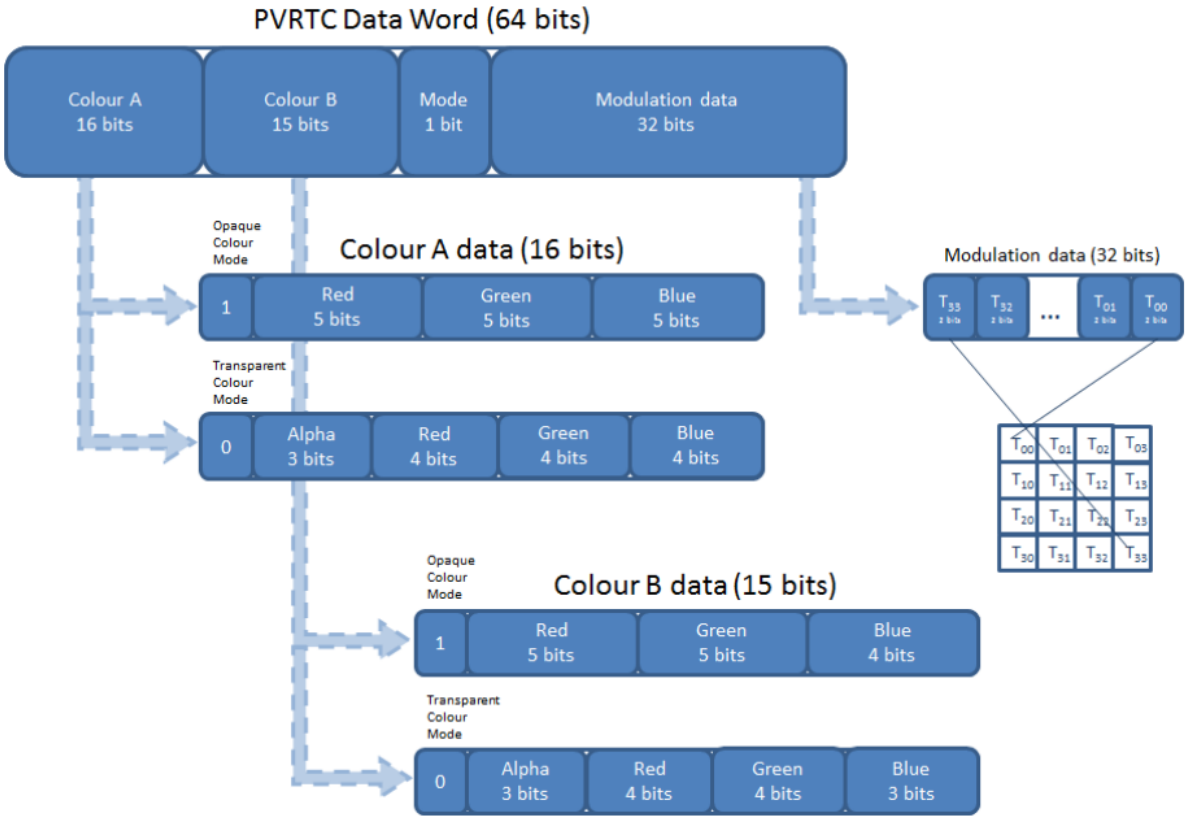
还原的过程如下图：



压缩后的一个4x4的block中的bits的组成内容为：

- 1bit：对应的融合方式，透明或不透明；
- 32bits：对应于16个pixel，每个pixel有2bits的调制因子；
- 31bits：对应两个缩略图中的该块映射到的两个像素上的颜色，若是透明模式，则两个颜色为RGBA44431，RGBA34431；若是不透明模式，则两个颜

色为RGBA5551，RGBA4551;



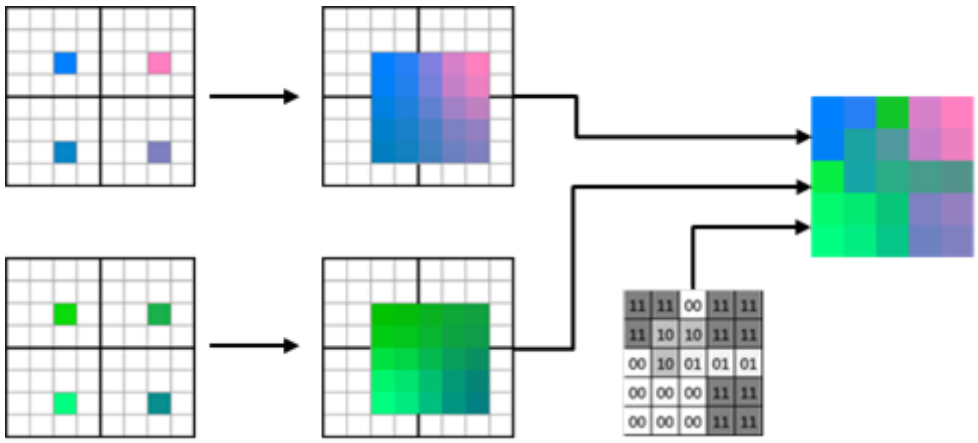
两种模式下的调制因子分别为：

----------	----------	----------
	不透明	透明
00	0/8	0/8
01	3/8	4/8
10	5/8	--
11	8/8	8/8

透明情况下的10调制因子会强制设置当前pixel的alpha为0；
通过调节上述两张缩略图的大小，可以相应的改变对应的压缩比，比如由（w/4，h/4）修改为（w/8，h/4），而其它的映射方式不变，即可将压缩比增大一倍。

混合的时候使用双线性过滤来对A和B进行扩大（这就要求必须要使用相邻的4x4的像素块，这应该就是为什么要求NOT的原因）,然后根据M中的调制因子的值进

行混合得到最终的混合结果。



（4）ASTC压缩原理

ASTC（Adaptive Scalable Texture Compression），由ARM和AMD联合开发，2012年发布，是较新的一种压缩格式，唯一一个不受专利权影响的压缩格式。

ASTC在压缩率、图像质量、种类上都表现不俗，也正在逐步代替前三种，最大的缺点可能就是兼容性还不够完善和解码时间较长，但以现在移动端的发展趋势来看，GPU计算能力越来越难成为瓶颈，因此非常有希望在以后能成为统一的压缩格式。

Block Size	Bits Per Pixel
4x4	8.00
5x4	6.40
5x5	5.12
6x5	4.27
6x6	3.56
8x5	3.20
8x6	2.67
10x5	2.56
10x6	2.13
8x8	2.00
10x8	1.60
10x10	1.28
12x10	1.07
12x12	0.89

4.怎么选择?

1.）相对来说ASTC的效率和质量是比较高的

仅能保证图片为2的幂，减少GPU的计算量。
Unity2022 默认选择的就是ASTC。

ASTC压缩的算法比较智能，它会为变化更大的通道RGB或者A分配更高的权重，而且对于单色图，RGB通道内容一样时，使用较低的像素占用就可以达到很好的效果。对于单色图完全没有必要使用R8压缩格式，而是应该将RGB通道填充一样的信息，选择ASTC较低的像素占比，如ASTC8x8。

2.）无绝对。假如有一张图本身就是PVRTC压缩之后的一张图，或者说非常接近，再使用PVRTC来进行压缩，效果肯定更好。